

**UNIVERSIDAD DE SANTIAGO DE CHILE**  
**FACULTAD DE INGENIERÍA**  
**Departamento de Informática**



**Desarrollo de un SDK para dar soporte a OpenGlove en dispositivos móviles**

**Israel Gedeón Elías Matínez Montenegro**

Profesor guía: Profesor Roberto González-Ibañez

Trabajo de titulación presentado  
en conformidad a los requisitos  
para obtener el título de Ingeniero  
Civil en Informática

Santiago – Chile

2018

© Israel Gedeón Elías Matínez Montenegro y Roberto González-Ibañez, 2018



• Algunos derechos reservados. Esta obra está bajo una Licencia Creative Commons Atribución-Chile 3.0. Sus condiciones de uso pueden ser revisadas en:  
<http://creativecommons.org/licenses/by/3.0/cl/>.

## RESUMEN

OpenGlove es un dispositivo diseñado en el Departamento de Ingeniería Informática por el grupo de investigación InTeracTion perteneciente a la Universidad de Santiago de Chile. El guante permite la retroalimentación vibrotáctil o *haptic feedback* cuando se interactúa con objetos en ambientes de realidad virtual (VR), aumentada (AR) o mixta (MR). Esta retroalimentación es generada a través de la vibración de motores distribuidos en distintas partes del guante, distribución que depende de los requerimientos del usuario o la aplicación. El guante también posee sensores de flexibilidad para la captura del movimiento de los dedos y una unidad de medición inercial o IMU(Imperial Measurement Unit) para capturar la orientación de la mano.

Si bien es posible tener una inmersión en estos ambientes, esta es parcial, dejando de lado el sentido del tacto. El estado actual de OpenGlove no permite el uso del mismo en comunidades de desarrollo de VR, AR y MR en entornos móviles como Android e iOS. Esto limita la portabilidad y desacople del sistema operativo Windows. Por ello lo que se propone es dar soporte a OpenGlove en dispositivos móviles enriqueciendo la interacción en los entornos ya mencionados. Esto se realizó mediante el desarrollo de un SDK para dispositivos móviles, el cual permite la conexión por Bluetooth de varias instancias de OpenGlove, como también la configuración de los mismos mediante la carga de perfiles utilizando una aplicación de configuración. Esta aplicación expone las funcionalidades por medio de un servidor WebSocket, el cual es utilizado mediante las APIs en Java y C#. Se realizaron evaluaciones de rendimiento, obtención de resultados y pruebas de concepto utilizando el sistema propuesto.

**Palabras Claves:** Haptic Feedback; Virtual Reality (VR); Augmented Reality (AR); Mixed Reality (MR); OpenGlove; SDK; API

*Dedicado a...*

## **AGRADECIMIENTOS**

Agradezco a

# TABLA DE CONTENIDO

<b>1</b>	<b>Introducción</b>	<b>1</b>
1.1	Antecedentes y motivación . . . . .	1
1.2	Descripción del problema . . . . .	2
1.3	Solución Propuesta . . . . .	3
1.3.1	Características de la solución . . . . .	3
1.3.2	Propósitos de la solución . . . . .	4
1.4	Objetivos y alcance del proyecto . . . . .	4
1.4.1	Objetivo general . . . . .	4
1.4.2	Objetivos específicos . . . . .	4
1.5	Metodologías y herramientas utilizadas . . . . .	5
1.5.1	Metodología a usar . . . . .	5
1.5.2	Herramientas de Software . . . . .	6
1.5.3	Herramientas de Hardware . . . . .	7
1.6	Organización del documento . . . . .	8
<b>2</b>	<b>Marco teórico</b>	<b>9</b>
2.1	Marco conceptual . . . . .	9
2.1.1	API . . . . .	9
2.1.2	SDK . . . . .	9
2.1.3	Tipos de aplicaciones móviles . . . . .	10
2.1.3.1	Nativas . . . . .	10
2.1.3.2	Web . . . . .	11
2.1.3.3	Híbridas . . . . .	11
2.1.4	WebSocket . . . . .	12
2.2	Estado del arte . . . . .	13
2.2.1	OpenGlove . . . . .	14
2.2.2	AvatarVR . . . . .	15
2.2.3	Dexmo . . . . .	16
2.2.4	Manus VR . . . . .	16
2.2.5	HaptX . . . . .	17
2.3	Resumen . . . . .	18
<b>3</b>	<b>Análisis</b>	<b>19</b>
3.1	Levantamiento de requisitos de software . . . . .	19
3.1.1	Antecedentes . . . . .	19
3.1.2	Requisitos . . . . .	19
3.2	Prototipos . . . . .	25
3.2.1	Primer prototipo: Activación de motores . . . . .	25
3.2.2	Segundo prototipo: Obtención de datos desde los flexores . . . . .	27
3.2.3	Tercer prototipo: Activación de motores y obtención de datos desde los flexores . . . . .	28
3.2.4	Cuarto prototipo: Navegación aplicación, administración dispositivos Bluetooth, servidor WebSocket y configuración de la placa . . . . .	30
3.2.5	Quinto prototipo: Mapeo de actuadores y prueba de actuadores . . . . .	32
3.2.6	Sexto prototipo: Mapeo de flexores, prueba de flexores y la configuración del IMU. . . . .	34
3.2.7	APIs . . . . .	36
3.2.7.1	Cuarto prototipo . . . . .	37
3.2.7.2	Quinto prototipo . . . . .	37
3.2.7.3	Sexto prototipo . . . . .	38

3.3 Resumen . . . . .	38
<b>4 Diseño e implementación</b>	<b>40</b>
4.1 Arquitectura general . . . . .	40
4.2 Estructura . . . . .	42
4.2.1 Software de control Arduino . . . . .	42
4.2.2 API C# de bajo nivel . . . . .	44
4.2.3 APIs de alto nivel . . . . .	47
4.2.4 Aplicación de configuración . . . . .	48
4.3 Comportamiento . . . . .	51
4.3.1 Métodos aplicados sobre cliente WebSocket . . . . .	52
4.3.2 Métodos aplicados sobre servidor WebSocket . . . . .	52
4.3.3 Métodos aplicados sobre instancias de OpenGloveDevice . . . . .	53
4.3.4 Métodos aplicados sobre el software de control Arduino . . . . .	55
4.3.5 Escuchando mensajes provenientes de Arduino . . . . .	56
4.3.6 Escuchando el estado de conexión del dispositivo Bluetooth . . . . .	58
4.3.7 Escuchando el estado de conexión del cliente WebSocket . . . . .	59
4.4 Aspectos de implementación . . . . .	60
4.4.1 Desarrollo multiplataforma en Xamarin.Forms . . . . .	60
4.4.2 API C# bajo nivel . . . . .	61
4.4.3 Servidor WebSocket . . . . .	61
4.4.4 Software de configuración . . . . .	61
4.4.5 APIs . . . . .	61
4.4.5.1 Java . . . . .	61
4.4.5.2 C# . . . . .	61
4.5 Resumen . . . . .	61
<b>5 Evaluación técnica y pruebas de concepto</b>	<b>63</b>
5.1 Evaluación aplicaciones nativa y multiplataforma . . . . .	63
5.1.1 Prototipo 3 : Droid - Galaxy . . . . .	64
5.1.1.1 Motores . . . . .	64
5.1.1.2 Flexores . . . . .	67
5.1.2 Prototipo 4: Xamarin - Galaxy . . . . .	69
5.1.2.1 Motores . . . . .	69
5.1.2.2 Flexores . . . . .	72
5.1.3 Prototipo 3 : Droid - Nexus . . . . .	74
5.1.3.1 Motores . . . . .	74
5.1.3.2 Flexores . . . . .	77
5.1.4 Prototipo 4: Xamarin - Nexus . . . . .	79
5.1.4.1 Motores . . . . .	79
5.1.4.2 Flexores . . . . .	82
5.2 Evaluación tiempo de activación usando APIs . . . . .	84
5.2.1 API C# . . . . .	86
5.2.2 API Java . . . . .	89
5.3 Evaluación tiempo de lectura de datos usando APIs . . . . .	89
5.3.1 API C# . . . . .	90
5.3.2 API Java . . . . .	93
5.4 Pruebas de concepto . . . . .	93
5.5 Resumen . . . . .	93

<b>6 Conclusiones</b>	<b>94</b>
6.1 Objetivos . . . . .	94
6.1.1 Objetivos específicos . . . . .	94
6.1.1.1 Desarrollar la aplicación de configuración de OpenGlove en Android. Permitiendo la creación, visualización, actualización y eliminación de los perfiles de configuración: . . . . .	94
6.1.1.2 Desarrollar APIs en Java y C# que permitan la administración de dispositivos Bluetooth en segundo plano en Android permitiendo la conexión, desconexión, activación y listado de guantes OpenGlove. También permitirá la activación y control de actuadores, flexores e IMU . . . . .	95
6.1.1.3 Realizar evaluaciones de rendimiento para las APIs Java y C# : . . . . .	95
6.1.1.4 Demostrar el uso del SDK en un ambiente de VR, AR o MR, utilizando las APIs desarrolladas: . . . . .	95
6.1.2 Objetivo general . . . . .	96
6.2 Resultados obtenidos . . . . .	96
6.2.1 Desarrollo de software . . . . .	96
6.2.2 Resultados de las pruebas . . . . .	96
6.3 Alcances y limitaciones . . . . .	97
6.4 Trabajo futuro . . . . .	98
6.5 Observaciones finales . . . . .	99
<b>Glosario</b>	<b>100</b>
<b>Referencias bibliográficas</b>	<b>102</b>
<b>Anexos</b>	<b>104</b>
<b>A Low Level Communication Protocol</b>	<b>104</b>
<b>B High Level Communication Protocol</b>	<b>105</b>
<b>C High Level OpenGlove APIs Reference</b>	<b>106</b>
C.1 C# API . . . . .	106
C.2 Java API . . . . .	106
<b>D Proof of concepts</b>	<b>107</b>

# ÍNDICE DE TABLAS

Tabla 2.1 Comparación SDK de distintos Guantes . . . . .	14
Tabla 3.1 Requisitos funcionales de software . . . . .	20
Tabla 3.2 Requisitos no funcionales de software . . . . .	25
Tabla 3.3 Primer prototipo: activación de motores . . . . .	26
Tabla 3.4 Segundo prototipo: obtención de datos desde flexores . . . . .	27
Tabla 3.5 Tercer prototipo: activación de motores y obtención de datos desde flexores . . . . .	29
Tabla 3.6 Cuarto prototipo . . . . .	30
Tabla 3.7 Quinto prototipo . . . . .	33
Tabla 3.8 Sexto prototipo . . . . .	35
Tabla 3.9 Matriz de prototipos de software vs requisitos funcionales . . . . .	39
Tabla 3.10 Matriz de prototipos de software vs requisitos no funcionales . . . . .	39
Tabla 4.1 Descripción de la estructura software de control Arduino . . . . .	44
Tabla 4.2 Descripción del diagrama de clases API C# de bajo nivel . . . . .	46
Tabla 4.3 Descripción del diagrama de clases APIs de alto nivel . . . . .	48
Tabla 4.4 Descripción del diagrama de clases aplicación de configuración . . . . .	49
Tabla 5.1 Resumen de los resultados de los tiempos de activación de motor Droid-Galaxy . . . . .	64
Tabla 5.2 Resumen de los resultados de los tiempos de lectura de flexores Droid-Galaxy . . . . .	68
Tabla 5.3 Resumen de los resultados de los tiempos de activación de motores Xamarin-Galaxy . . . . .	69
Tabla 5.4 Resumen de los resultados de los tiempos de lectura de flexores Xamarin-Galaxy . . . . .	72
Tabla 5.5 Resumen de los resultados de los tiempos de activación de motores Droid-Nexus . . . . .	74
Tabla 5.6 Resumen de los resultados de los tiempos de lectura de flexores Droid-Nexus . . . . .	77
Tabla 5.7 Resumen de los resultados de los tiempos de activación de motores Xamarin-Nexus . . . . .	79
Tabla 5.8 Resumen de los resultados de los tiempos de lectura de flexores Xamarin-Nexus . . . . .	82
Tabla 5.9 Resumen resultados de los tiempos de activación usando API C# . . . . .	86
Tabla 5.10 Resumen resultados de pruebas de lectura flexores e IMU usando API C# . . . . .	90

# ÍNDICE DE ILUSTRACIONES

Figura 1.1	Arquitectura OpenGlove . . . . .	2
Figura 2.1	Representación visual de WebSocket . . . . .	13
Figura 2.2	Guantes OpenGlove . . . . .	15
Figura 2.3	Guantes AvatarVR y TrackBand . . . . .	15
Figura 2.4	Guantes Dexmo . . . . .	16
Figura 2.5	Guantes Manus VR . . . . .	17
Figura 2.6	Guantes Haptx . . . . .	18
Figura 3.1	Primer prototipo: activación de motores . . . . .	26
Figura 3.2	Segundo prototipo: obtención de datos desde flexores . . . . .	28
Figura 3.3	Tercer prototipo: activación de motores y obtención de datos desde flexores . . . . .	29
Figura 3.4	Cuarto prototipo: app navigation, devices, server and board configuration . . . . .	31
Figura 3.5	Quinto prototipo: actuator mapping and actuator test . . . . .	33
Figura 3.6	Sexto prototipo: flexor mapping, flexor test and IMU configuration . . . . .	35
Figura 4.1	Diagrama conceptual general de OpenGlove . . . . .	40
Figura 4.2	Detalle Arquitectura de Openglove . . . . .	41
Figura 4.3	Estructura del software de control Arduino . . . . .	43
Figura 4.4	Diagrama de clases API C# de bajo nivel . . . . .	45
Figura 4.5	Diagrama de clases API C# de alto nivel . . . . .	47
Figura 4.6	Diagrama de clases API Java de alto nivel . . . . .	47
Figura 4.7	Diagrama de clases aplicación de configuración . . . . .	49
Figura 4.8	Diagrama de actividades de los métodos aplicados sobre el cliente WebSocket . . . . .	52
Figura 4.9	Diagrama de actividad de los métodos aplicados sobre el servidor WebSocket . . . . .	53
Figura 4.10	Diagrama de actividades de las acciones OpenGlove sobre instancias de OpenGloveDevice en servidor . . . . .	54
Figura 4.11	Diagrama de actividades de los métodos aplicados sobre el software de control Arduino . . . . .	56
Figura 4.12	Diagrama de actividades cuando se escuchan mensajes provenientes de Arduino . . . . .	57
Figura 4.13	Diagrama de actividades cuando se escucha el estado de la conexión del dispositivo Bluetooth . . . . .	59
Figura 4.14	Diagrama de actividades cuando se escucha la conexión del dispositivo Bluetooth . . . . .	60
Figura 5.1	Histogramas de los tiempos de activación de motores Droid-Galaxy . . . . .	65
Figura 5.2	Gráfico QQ de los tiempos de activación de motores motores Droid-Galaxy . . . . .	66
Figura 5.3	Gráficos de cajas de los tiempos de activación de motores de motores Droid-Galaxy . . . . .	67
Figura 5.4	Histogramas de los tiempos de lectura de flexores Droid-Galaxy . . . . .	68
Figura 5.5	Gráfico QQ de los tiempos de lectura de flexores Droid-Galaxy . . . . .	68
Figura 5.6	Gráficos de cajas de los tiempos de lectura de flexores Droid-Galaxy . . . . .	69
Figura 5.7	Histogramas de los tiempos de activación de motores Xamarin-Galaxy . . . . .	70
Figura 5.8	Gráfico QQ de los tiempos de activación de motores Xamarin-Galaxy . . . . .	71
Figura 5.9	Gráficos de cajas de los tiempos de activación de motores Xamarin-Galaxy . . . . .	72
Figura 5.10	Histogramas de los tiempos de lectura de flexores Xamarin-Galaxy . . . . .	73
Figura 5.11	Gráfico QQ de los tiempos de lectura de flexores Xamarin-Galaxy . . . . .	73
Figura 5.12	Gráficos de cajas de los tiempos de lectura de flexores Xamarin-Galaxy . . . . .	74
Figura 5.13	Histogramas de los tiempos de activación de motores Droid-Nexus . . . . .	75

Figura 5.14	Gráfico QQ de los tiempos de activación de motores Droid-Nexus . . . . .	76
Figura 5.15	Gráficos de cajas los tiempos de activación de motores Droid-Nexus . . .	77
Figura 5.16	Histogramas de los tiempos de lectura de flexores Droid-Nexus . . . . .	78
Figura 5.17	Gráfico QQ de los tiempos de lectura de flexores Droid-Nexus . . . . .	78
Figura 5.18	Gráficos de cajas de los tiempos de lectura de flexores Droid-Nexus . . .	79
Figura 5.19	Histogramas de los tiempos de activación de motores Xamarin-Nexus . .	80
Figura 5.20	Gráfico QQ de los tiempos de activación de motores Xamarin-Nexus . . .	81
Figura 5.21	Gráficos de cajas de los tiempos de activación de motores Xamarin-Nexus .	82
Figura 5.22	Histogramas de los resultados de los tiempos de lectura de flexores Xamarin-Nexus . . . . .	83
Figura 5.23	Gráfico QQ de los resultados de los tiempos de lectura de flexores Xamarin- Nexus . . . . .	83
Figura 5.24	Gráficos de cajas de los resultados de los tiempos de lectura de flexores Xamarin-Nexus . . . . .	84
Figura 5.25	Aplicación utilizada para realizar las pruebas de la API C# . . . . .	85
Figura 5.26	Aplicación utilizada para realizar las pruebas de la API Java . . . . .	85
Figura 5.27	Histogramas de los tiempos de activación de motores usando API C# . . .	87
Figura 5.28	Gráfico QQ delos tiempos de activación de motores usando API C# . . . .	88
Figura 5.29	Gráficos de cajas de los tiempos de activación de motores usando API C# .	89
Figura 5.30	Histogramas de Flexores e IMU usando API C# . . . . .	91
Figura 5.31	Gráfico QQ de Flexores e IMU usando API C# . . . . .	92
Figura 5.32	Gráficos de cajas de Flexores e IMU usando API C# . . . . .	93

## ÍNDICE DE CÓDIGO

Código 4.1	Métodos aplicados sobre el cliente WebSocket . . . . .	52
Código 4.2	Métodos aplicados sobre el servidor WebSocket . . . . .	53
Código 4.3	Métodos aplicados sobre instancias de OpenGloveDevice . . . . .	54
Código 4.4	Métodos aplicados sobre el software de control Arduino . . . . .	55
Código 4.5	Escuchando mensajes provenientes de Arduino . . . . .	57
Código 4.6	Escuchando el estado de conexión del dispositivo Bluetooth . . . . .	58
Código 4.7	Escuchando el estado de conexión del cliente WebSocket . . . . .	60
Código 4.8	Implementación ICommunication en diferentes plataformas . . . . .	62

# CAPÍTULO 1. INTRODUCCIÓN

## 1.1 ANTECEDENTES Y MOTIVACIÓN

El tamaño del mercado de la realidad virtual (VR)<sup>1</sup> y realidad aumentada (AR)<sup>2</sup> registra 6.1 billones de dólares para el año 2016 y se estima que para el 2017 ascienda a 11.4 billones (Statista, 2016). En este contexto también aparece la denominada realidad mixta (MR)<sup>3</sup> la cual es un punto intermedio entre VR y AR. Actualmente es bastante popular el uso de VR, AR y MR en dispositivos móviles, pero estos carecen del soporte de otros sentidos como lo es el tacto lo cual genera una disruptión cuando se interactúa con los objetos virtuales. En este contexto se puede presentar una brecha cuando se desea incluir retroalimentación vibrotáctil o el denominado *haptic feedback*<sup>4</sup> a aplicaciones en los entornos ya mencionados. Adicionalmente se incluye la captura de movimiento, para su representación en los ambientes ya mencionados.

En primer lugar, el tamaño del mercado entre el 2017 y 2020 para VR y AR espera más de un 1200% de aumento en sumas de billones de dólares. Esto es una interesante oportunidad de negocio, como también el aprovechar las distintas comunidades de desarrollo de VR, AR y MR para masificar el uso de OpenGlove.

En segundo lugar, la disruptión que se genera cuando se interactúa con objetos virtuales y no se obtiene una respuesta similar a la experiencia real, crea un punto de quiebre entre lo real y lo virtual. Esto implica una inmersión parcial en los ambientes de VR, AR y MR.

En tercer lugar se tiene el alto costo asociados a la adquisición de los guantes. Es posible constatar precios de guantes que van desde los 300€ hasta los 1300€ y licencias desde los 2500€ hasta los 13300€. Esto se traduce en costos más altos de desarrollo como también para los usuarios finales. Además genera un mercado con un alcance más limitado.

En base a los argumentos desarrollados, se evidencia una brecha en la inclusión de Haptic Feedback y la captura de movimiento de las manos en proyectos de VR, AR y MR en dispositivos móviles, la cual actualmente no es cubierta de manera global.

El estado actual de OpenGlove no permite el uso del mismo en comunidades de desarrollo de VR, AR y MR en entornos móviles como Android e iOS. Esto limita la portabilidad y desacople del sistema operativo Windows, reduciendo el alcance que puede tener en las ya mencionadas comunidades de desarrollo.

<sup>1</sup> **Virtual Reality (VR):** “La realidad virtual (VR) proporciona un entorno 3D generado por computadora que rodea al usuario y responde a las acciones de esa persona de forma natural ...” Gartner (2017d).

<sup>2</sup> **Augmented Reality (AR):** “es el uso de información en tiempo real en forma de texto, gráficos, audio y otras mejoras virtuales integradas con objetos del mundo real ... ” (Garther, 2017a)

<sup>3</sup> **Mixed reality (MR):** “es el resultado de mezclar el mundo físico con el mundo digital, incluyendo la interacción con objetos virtuales representados en el real ... ” (Microsoft, 2017).

<sup>4</sup> **Haptic Feedback:** Haptics es una tecnología táctil o de retroalimentación de fuerza que aprovecha el sentido del tacto de una persona al aplicar vibraciones y / o movimiento a la punta del dedo del usuario ... ” (Gartner, 2017b)

## 1.2 DESCRIPCIÓN DEL PROBLEMA

Actualmente el proyecto OpenGlove posee la arquitectura que se aprecia en la Figura 4.1. El guante posee motores distribuidos en distintos lugares de la mano, los cuales pueden ser activados y desactivados según se requiera mediante APIs (Monsalve, 2015), también se tienen sensores de flexibilidad y la captura de movimientos de la mano (Cerda, 2017) . Existe un servicio que utiliza SOAP y REST para exponer los servicios mediante Bluetooth en Windows. Luego las APIs de alto nivel (Meneses, 2016) para lenguajes de programación como C#, C++, Java, y JavaScript, permiten la abstracción de la complejidad en el uso de instancias y configuración de OpenGlove. El estado actual de OpenGlove no permite el uso del mismo en comunidades de desarrollo de realidad virtual móvil en Android VR<sup>5</sup>, ni iOS <sup>6</sup> sin depender del sistema que permite la configuración y la ejecución del servicio en Windows. Lo cual dificulta la integración de OpenGlove para soluciones en dispositivos móviles y la independencia de servicios alojados en otro sistema operativo, para poder desarrollar aplicaciones de VR, AR o MR para dispositivos móviles.

El problema se puede resumir en la siguiente pregunta ¿Cómo facilitar a la comunidad de desarrolladores de VR/AR/MR a integrar OpenGlove en entornos móviles y realizar una fácil configuración para los dispositivos?.

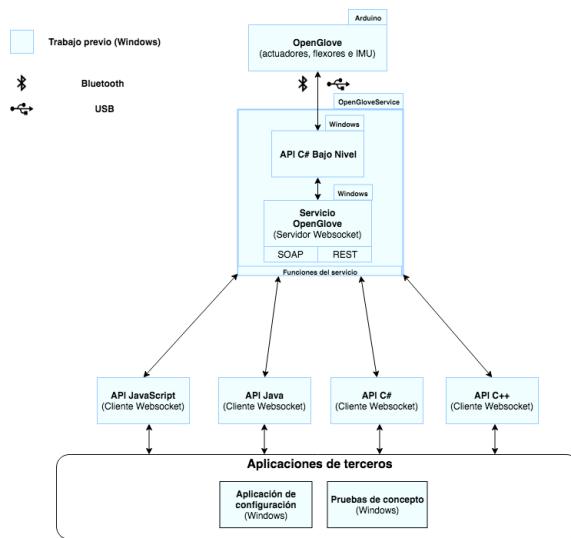


Figura 1.1: Arquitectura OpenGlove  
Fuente: Elaboración propia (2018)

<sup>5</sup>Comunidad Android: <https://developers.google.com/vr/android/>

<sup>6</sup>Comunidad iOS: <https://developers.google.com/vr/ios/>

## 1.3 SOLUCIÓN PROPUESTA

A continuación se describen las características y el propósito de la solución propuesto para el problema planteado.

### 1.3.1 Características de la solución

Para resolver el problema, se propuso una solución para Android, la cual corresponde a un SDK<sup>7</sup> que incluye las herramientas necesarias para el desarrollo de aplicaciones de terceros relacionadas a VR/AR/MR. El SDK incluye la documentación, APIs y software de configuración. Las APIs permiten activar y desactivar los distintos motores y sensores distribuidos en el guante como también el agregar, quitar y listar dispositivos Bluetooth. El software de configuración permite levantar un servicio Bluetooth en segundo plano, que permita agregar, activar, desactivar y listar dispositivos conectados. También se pueden guardar los perfiles de configuración de los guantes en el dispositivo. Dicha aplicación de configuración soporta la conexión de múltiples dispositivos, para lograr utilizar uno o más OpenGlove desde dispositivos móviles.

Cabe destacar que el SDK no implica un costo para los desarrolladores, ni para los usuarios del guante.

Las tecnologías que permiten implementar las características anteriormente mencionadas, son las siguientes:

- Hilos en segundo plano en Android<sup>8</sup>, que permiten la ejecución de tareas en paralelo, lo que permite administrar múltiples conexiones Bluetooth para la escritura y lectura de datos en dispositivos conectados.
- Websocket<sup>9</sup>, tecnología que proporciona un canal de comunicación bidireccional y full-duplex (información enviada simultáneamente) sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor.
- Bluetooth<sup>10</sup>, tecnología de red inalámbrica que permite la transmisión de datos entre dispositivos.

---

<sup>7</sup>SDK: conjunto de utilidades de desarrollo para escribir aplicaciones de software, generalmente asociadas a entornos específicos (por ejemplo, el SDK de Windows) (Gartner, 2017c).

<sup>8</sup>Multiple Threads Runnable Code: <https://developer.android.com/training/multiple-threads/define-runnable>

<sup>9</sup>Websocket: <http://websocket.org/aboutwebsocket.html>

<sup>10</sup>Bluetooth Android: <https://developer.android.com/guide/topics/connectivity/bluetooth.html>

- Xamarin.Forms<sup>11</sup> tecnología que permite el desarrollo de aplicaciones multiplataforma para iOS, Android, Windows y Tizen en una versión estable. En este proyecto, permite compartir la interfaz de usuario en ambos sistemas operativos móviles de manera nativa y realizar las implementaciones específicas para cada uno.

### **1.3.2 Propósitos de la solución**

El propósito del proyecto es poner al alcance de distintas comunidades las herramientas para el desarrollo y uso de dispositivos de respuesta vibrotáctil y seguimiento de las manos en ambientes de realidad virtual, mixta y aumentada, junto con promover el uso de OpenGlove en la comunidad Android, los entusiastas del DIY (Do It Yourself) o “házlo tú mismo” y fabricantes que quieran hacer uso de estas tecnologías para sus proyectos.

## **1.4 OBJETIVOS Y ALCANCE DEL PROYECTO**

### **1.4.1 Objetivo general**

El objetivo general del proyecto es desarrollar un SDK que permita dar soporte a OpenGlove en dispositivos móviles.

### **1.4.2 Objetivos específicos**

En esta sección se listan los resultados parciales definidos para lograr el objetivo general del proyecto de titulación.

1. Desarrollar la aplicación de configuración de OpenGlove en Android. Permitiendo la creación, visualización, actualización y eliminación de los perfiles de configuración.
2. Desarrollar APIs en Java y C# que permitan la administración de dispositivos Bluetooth en segundo plano en Android permitiendo la conexión, desconexión, activación y listado de

---

<sup>11</sup>Xamarin.Forms: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/>

guantes OpenGlove. También permitirá la activación y control de actuadores <sup>12</sup>, flexores <sup>13</sup> e IMU (*Inertial Measurement Unit*).

3. Realizar evaluaciones de rendimiento para las APIs Java y C#.
4. Demostrar el uso del SDK en un ambiente de VR, AR o MR, utilizando las APIs desarrolladas.

## 1.5 METODOLOGÍAS Y HERRAMIENTAS UTILIZADAS

En esta sección se presenta la metodología utilizada a lo largo del proyecto, las herramientas y el ambiente de desarrollo del SDK que permitirá el soporte de OpenGlove en dispositivos móviles.

### 1.5.1 Metodología a usar

Para alcanzar los objetivos planteados en este proyecto, también es necesario elegir una metodología adecuada para el proyecto. En base a esto y a la incertidumbre que presenta el proyecto, es necesario que se elija una metodología que reduzca los riesgos asociados, permitiendo entregas de software funcional de manera rápida y que puedan ser validados por el cliente (profesor guía). En este contexto y dado que el proyecto de título debe ser realizado dentro del semestre considerando un trabajo de 612 horas cronológicas y basado en la experiencia y el desarrollo de proyectos de similares características, la metodología planteada a continuación es la adecuada para el presente proyecto de ingeniería propuesto.

La metodología corresponde a la utilizada por el grupo de investigación y desarrollo InTeracTion. Puede separarse en dos etapas principales. La primera etapa consiste en el desarrollo de prototipos funcionales, con la finalidad de obtener retroalimentación del cliente y así cubrir los requerimientos del cliente de manera iterativa. Esta etapa tiene como referencia la metodología RAD (Rapid Application Development) (Martin, 1991), la cual está orientada a grupos pequeños, enfocada en el producto y no en la documentación generada, necesitando entregas rápidas para obtener retroalimentación del producto desecharable desarrollado. El prototipo final aprobado será la base con lo que se desarrollará la siguiente etapa. La segunda etapa consiste en el desarrollo de las funcionalidades requeridas para el producto, utilizando como base el prototipo funcional de

---

<sup>12</sup>Definición actuadores: Un actuador es un dispositivo inherentemente mecánico cuya función es proporcionar fuerza para mover o “actuar” otro dispositivo mecánico. Dependiendo de el origen de la fuerza el actuador se denomina “neumático”, “hidráulico” o “eléctrico” (Aie, 2017)

<sup>13</sup> “... es un sensor de flexión que produce una resistencia variable en función del grado al que este doblada”.(Rambal, 2018)

la primera etapa. El proyecto se desarrolla de manera iterativa e incremental, para su gestión se establece como mínimo una reunión semanal en la cual se tratan temas como el estado de avance, los compromisos asumidos para la semana, los compromisos siguientes y los problemas ocurridos. En estas reuniones es posible tomar decisiones sobre los problemas ocurridos y cambiar de estrategia según sea necesario, esto permite reducir los riesgos ocasionados por la incertidumbre del proyecto.

En resumen la metodología a utilizar contempla las siguiente características:

- Primera etapa de desarrollo mediante prototipos basada en RAD.
- Segunda etapa de desarrollo iterativo del prototipo maduro y aceptado.
- Como mínimo reuniones una vez a la semana en todas las etapas.

Para la gestión del proyecto se utilizan los siguientes recursos, que permiten tener un seguimiento del trabajo a realizar :

- Kanban físico y compromisos semanales.
- Kanban simple de tres columnas o estados por hacer (TO DO), haciendo (DOING) y hecho (DONE).

Para que el proyecto llegue a buen término, se debe elegir la metodología adecuada al proyecto dado el contexto del mismo. Dada las características y la interacción que se desea establecer con el profesor guía como cliente, se adoptó la metodología utilizada por InTeracTion. Además, esta metodología presenta ventajas por sobre las tradicionales respecto de la adaptabilidad, colaboración con el cliente y entregas funcionales iterativas que pueden ser evaluadas cada cierto tiempo, mostrando los avances y favoreciendo el producto funcionando por sobre la documentación exhaustiva <sup>14</sup>.

### **1.5.2 Herramientas de Software**

En esta sección se listan las herramientas de Software que se utilizaron para el desarrollo del proyecto de título. La aplicación fue desarrollada utilizando el IDE Visual Studio Community 2018 para Mac <sup>15</sup> en conjunto con el ecosistema de plataformas para compilar <sup>16</sup> aplicaciones móviles Xamarin.Forms, el cual utiliza el lenguaje de programación C# para el desarrollo de aplicaciones

---

<sup>14</sup>Manifiesto ágil: <http://agilemanifesto.org/>

<sup>15</sup>Xamarin más VS Community para Mac 2018: <https://store.xamarin.com/>

<sup>16</sup> En resumidas palabras un compilador es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación. Este proceso de traducción se conoce como compilación. <http://www.ictea.com/cs/knowledgebase.php?action=displayarticle&id=8817>

nativas multiplataforma. Adicionalmente se utilizó el IDE Android Studio<sup>17</sup>, el cual corresponde al IDE oficial para el sistema operativo Android, el cual integra distintas herramientas de desarrollo, como el SDK de Android, el editor de texto, el soporte para control de versiones, compilador, etc. A continuación se listan otras herramientas de software para el apoyo para el desarrollo del proyecto.

1. Github<sup>18</sup> para el control de versiones.
2. Texmaker<sup>19</sup>, para la escritura de la memoria.
3. RStudio<sup>20</sup>: para el análisis de datos de la evaluación del proyecto.
4. Google Drive<sup>21</sup>, para generar y compartir documentos y archivos de manera colaborativa.
5. Microsoft Project Professional y Gantter, para el desarrollo de la carta gantt del proyecto de título.

### 1.5.3 Herramientas de Hardware

El ambiente de desarrollo en el cual se desarrolló SDK fue un Macbook Pro con las siguientes características:

1. Sistema operativo SO macOS Sierra versión 10.13.3.
2. Procesador Intel Core i5, 3,1 GHz .
3. Memoria RAM 8GB 2133 MHz LPDDR3.
4. 512 GB disco duro SSD.
5. Gráficos Intel Iris Plus Graphics 650 1536 MB.

Las pruebas de la aplicación se realizarán utilizando Android Virtual Device (AVD) mediante Android Studio y un smartphone *Samsung Galaxy S5 mini*<sup>22</sup> con las siguientes características:

1. Sistema Operativo Android Marshmallow 6.0.1.
2. CPU: 1.4Ghz Quad-Core ARM Cortex-A7.
3. GPU: ARM Mali-400 MP4 450Mhz.

<sup>17</sup>Android Studio: <https://developer.android.com/studio/index.html?hl=es-419>

<sup>18</sup>Github: <https://github.com/>

<sup>19</sup>Texmaker : <http://www.xm1math.net/texmaker/>

<sup>20</sup>Rstudio: <https://www.rstudio.com/>

<sup>21</sup>Google Drive: [https://www.google.com/intl/es\\_ALL/drive/](https://www.google.com/intl/es_ALL/drive/)

<sup>22</sup>Smartphone: <http://www.movilcelular.es/samsung-galaxy-s5-mini-duos-sm-g800h/caracteristicas/1659>

4. Memoria RAM 1,5GB LPDDR2.
5. Almacenamiento interno 16GB (12GB accesible al usuario).
6. Bluetooth Versión 4.0 con A2DP.
7. Sensores: Acelerómetro, Proximidad, Brújula , Luz ambiental, Giroscopio, Biométrico (huellas digitales).

El prototipo de dispositivo OpenGlove disponible en InTeracTion fue utilizado para realizar las pruebas y el desarrollo del SDK. Se usaron el IMU, motores y sensores de flexibilidad disponibles en el prototipo.

1. Sensores de flexibilidad de 2,2", SparkFun.
2. Sensor de rastreo IMU, SparkFun.
3. Actuadores (En específico motores de vibración).
4. Bluetooth mate silver, Sparkfun.

## 1.6 ORGANIZACIÓN DEL DOCUMENTO

El presente documento posee seis capítulos que abarcan la totalidad de este proyecto de desarrollo en sus distintas fases. El Capítulo 1, Introducción, incluye los antecedentes y motivación del proyecto, presentando el problema y solución propuesta junto con los objetivos necesarios para concretarla. El Capítulo 2, Marco teórico, introduce los conceptos relevantes que permitirán una mejor compresión del problema junto a su solución. También se incluye el estado del arte, el cual detalla las soluciones alternativas actuales del problema planteado, detallando y comparándolas entre ellas. En el Capítulo 3, Análisis, se realiza un análisis sobre el desarrollo del SDK analizando los componentes de hardware y software disponibles, generando prototipos hasta que se genere el producto esperado. Luego en el Capítulo 4, Diseño e implementación, se diseña la solución a nivel de mockups o maquetas, también a nivel arquitectural, comportamiento y de protocolos de comunicación. Posteriormente se implementan para lograr la solución diseñada. El Capítulo 5, Evaluación técnica, se realiza una evaluación del software desarrollado para establecer cuáles son los tiempos de respuesta (latencia) presentes y la comparativa con la versión de escritorio. Además se considera otro factor importante respecto a la solución , esto es, la eficiencia energética de la misma. Por último, en el Capítulo 6, Conclusiones, se detallan los objetivos cumplidos, también los resultados obtenidos con la solución, los alcances y limitaciones de la misma, posibles mejoras para trabajos futuros y observaciones finales pertinentes.

## CAPÍTULO 2. MARCO TEÓRICO

Este capítulo permite establecer las bases teóricas necesarias para una mejor comprensión del presente documento. Primero se presenta el Marco conceptual, en el cual se establecen los conceptos relevantes del proyecto, luego en el Estado del arte se realiza una revisión del mismo.

### 2.1 MARCO CONCEPTUAL

#### 2.1.1 API

*Application Program Interface* por sus siglas en inglés, es código que actúa como interfaz para la programación de aplicaciones, permitiendo por ejemplo, que dos aplicaciones se comuniquen entre si, como el acceder a funcionalidades sin la necesidad de conocer la complejidad del código implementado (Rouse, 2017). Por ejemplo, para el sistema operativo<sup>1</sup> Android, todo su conjunto de funciones está disponible mediante APIs escritas en el lenguaje Java. Estas APIs son los cimientos necesarios para crear aplicaciones de Android simplificando la reutilización de componentes del sistema, servicios centrales y modulares.

#### 2.1.2 SDK

Es un conjunto de utilidades de desarrollo para escribir aplicaciones de software, generalmente asociadas a entornos específicos (Gartner, 2017c). Ejemplos de SDKs incluyen a Windows 7 SDK, the Mac OS X SDK, iPhone SDK y Android SDK.

Los SDK suelen incluir un entorno de desarrollo integrado (IDE), que sirve como interfaz de programación central. El IDE puede incluir una ventana de programación para escribir el código fuente, un depurador para corregir errores del programa y un editor visual, que permite a los desarrolladores crear y editar la interfaz gráfica de usuario (GUI) del programa. Los IDE también incluyen un compilador, que se usa para crear aplicaciones a partir de archivos de código fuente (Techterms, 2010).

---

<sup>1</sup>En resumidas palabras, un sistema operativo (OS por sus siglas en inglés) es un software que, después de ser cargado en la computadora por un programa de arranque inicial, administra los recursos de una computadora, controlando el flujo de información hacia y desde un procesador principal, administrando recursos como, la memoria, redes, archivos, etc. (Gartner, 2018)

La mayoría de los SDK contienen código de ejemplo, que proporciona a los desarrolladores ejemplos de programas y bibliotecas. Estas muestras ayudan a los desarrolladores a aprender cómo crear programas básicos con el SDK, lo que les permite eventualmente crear aplicaciones más complejas. Los SDK también ofrecen documentación técnica, que puede incluir tutoriales y preguntas frecuentes. Algunos SDK también pueden incluir gráficos de muestra, como botones e iconos, que se pueden incorporar a las aplicaciones (Techterms, 2010). En resumen, un SDK usualmente incluye lo siguiente:

- APIs
- IDE
- Software de configuración
- Códigos de ejemplo (programas, pruebas de concepto y bibliotecas)
- Documentación (técnica, tutoriales, preguntas frecuentes)
- Materiales gráficos de muestra (iconos, botones, etc.)

### **2.1.3 Tipos de aplicaciones móviles**

Actualmente los sistemas operativos que predominan el mercado de dispositivos móviles son dos, iOS con un 14.4% y Android con un 84.8% (Statista, 2017). Cuando se habla de desarrollar aplicaciones móviles se tienen como objetivo los dispositivos que tienen iOS o Android como sistema operativo. A continuación se detallan los tres principales tipos de aplicaciones móviles: Nativas, Híbridas y Web.

#### *2.1.3.1 Nativas*

Apple y Google ofrecen a los desarrolladores de aplicaciones sus propias herramientas de desarrollo, elementos de interfaz y SDK estandarizado; Xcode y Android Studio. Esto permite a cualquier desarrollador profesional desarrollar una aplicación nativa con relativa facilidad.

Las ventajas que ofrecen las aplicaciones móviles nativas, son mayor rapidez que las aplicaciones Híbridas y Web, también una experiencia con elementos de interfaz nativos. Con aplicaciones nativas, es posible acceder a la cámara, el micrófono, almacenamiento interno, Bluetooth, la

brújula, el acelerómetro y deslizar gestos fácilmente. Todavía es posible usar las alternativas, pero es más directo de manera nativa (MobiLoud, 2018).

Las desventajas de las aplicaciones nativas es que se requiere más de un código base, dado que las aplicaciones iOS no se ejecutarán en Android y viceversa, por lo que se tiene que trabajar con diferentes códigos para cada plataforma que se elija construir. Esto se traduce a mayores costos de mantenimiento, puesto que se necesitan desarrolladores con conocimientos en ambas plataformas o bien equipos diferenciados, teniendo ambos que desarrollar lo mismo en ambas aplicaciones nativas (MobiLoud, 2018).

Actualmente existen alternativas a las herramientas para desarrollo nativo ofrecidas por iOS y Android. Estas son, Xamarin, React Native y Titanium, las cuales permiten desarrollar aplicaciones nativas multiplataforma (cross-platform) para iOS y Android. Gracias a estas alternativas se permite compartir código para ambas plataformas.

#### *2.1.3.2 Web*

La mayoría están desarrolladas en JavaScript, CSS y HTML5. No poseen SDK para trabajar con iOS y Android. En resumidas cuentas son aplicaciones web que son adaptadas para parecer aplicaciones nativas, pero sin acceder completamente a las funcionalidades del dispositivo y sin tener un comportamiento igual al nativo.

Hasta hace poco, las aplicaciones web carecían de la funcionalidad de las aplicaciones nativas, como la capacidad de enviar notificaciones automáticas, trabajar sin conexión y cargar en la pantalla de inicio.

Sin embargo, han habido algunas mejoras en los navegadores y aplicaciones web que ofrecen estas características. Las aplicaciones que aprovechan estas características se denominan aplicaciones web progresivas (MobiLoud, 2018). Esta alternativa permite compartir el 100% del código para ambas plataformas, ya que son aplicaciones web accedidas por Internet desarrolladas con herramientas web.

#### *2.1.3.3 Híbridas*

Se instala como una aplicación nativa, pero en realidad es una aplicación web en el interior. Las aplicaciones híbridas, como las aplicaciones web, se crean con Javascript, HTML y CSS y se ejecutan en algo llamado Webview, un navegador simplificado dentro de su aplicación. Las

ventajas que presentan estas aplicaciones es que son menos costosas ya que comparten el mismo código base. Al igual que con las aplicaciones nativas, conserva la misma capacidad para acceder a las funciones del dispositivo. Esto es gracias a soluciones como PhoneGap que actúan como un puente entre el SDK nativo y la vista web en la que se ejecuta la aplicación.

La principal desventaja de este tipo de aplicación móvil, es su rendimiento, además de que presentan diferencias en el comportamiento respecto de las aplicaciones nativas. Algunas de las alternativas existentes para desarrollar aplicaciones híbridas son: PhoneGap/Cordova y Canvas. (MobiLoud, 2018)

#### **2.1.4 WebSocket**

WebSocket es un protocolo que provee un canal de comunicación bidireccional y simultáneo (full-duplex) entre un cliente y un servidor, utilizando un único socket TCP<sup>2</sup> (Transmission Control Protocol). Este protocolo inicia un HandShake, el cual consiste en un request HTTP<sup>3</sup> para establecer la conexión entre el cliente y servidor (Websocket.org, 2018a). Esta petición establece el upgrade del protocolo HTTP al de WebSocket. Por tanto el protocolo HTTP solo se utiliza para establecer una conexión persistente mediante un socket TCP, el cual permite el intercambio de mensajes full-duplex mientras el socket sea cerrado por alguna de las partes. El servidor puede mantener múltiples clientes conectados, intercambiando mensajes de manera simultánea, permitiendo el desarrollo de aplicaciones que requieren interacción en tiempo real, como por ejemplo, los juegos en línea o gráficas en tiempo real para el mercado de las divisas (Currency Market) (Websocket.org, 2018b). La Figura 2.1 muestra la representación visual del protocolo WebSocket anteriormente explicado.

---

<sup>2</sup>TCP: es un protocolo que cubre la capa de transporte del modelo de red de siete capas OSI (Open System Interconnection) <https://www.gartner.com/it-glossary/tcpip-transmission-control-protocolinternet-protocol>

<sup>3</sup>HTTP (Hypertext Transfer Protocol): es una capa del nivel de aplicación para transmitir documentos de hipermédia (del inglés hypermedia) como lo es un documento HTML

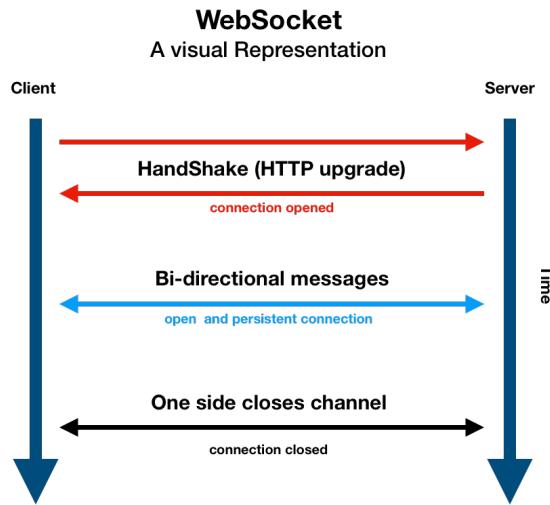


Figura 2.1: Representación visual de WebSocket  
Fuente: PubNub (2018)

Para un mayor entendimiento sobre WebSocket es necesario citar la publicación de Grigorik (2013). En el capítulo 17 define lo siguiente: "WebSocket es un conjunto de múltiples estándares: WebSocket API<sup>4</sup> que es definida por la W3C y el protocolo WebSocket (RFC 6455<sup>5</sup>) cuyas extensiones son definidas por HyBi Working Group (IETF)". En definitiva para hacer uso del protocolo se hace utilizar la API WebSocket, permitiendo abstraer la complejidad detrás del protocolo, el cual es establecido bajo el estándar RFC 6455.

La API de WebSocket es simple de usar, esto se puede observar los siguientes eventos presentes en clientes y servidores WebSocket:

- **On Open:** Este evento se activa cuando se ha abierto un nuevo socket.
- **On Message:** Este evento se activa cuando se recibe un nuevo mensaje.
- **On Error:** Este evento se activa cuando ocurre un error en la conexión WebSocket.
- **On Close:** Este evento se activa cuando ocurre cuando el socket es cerrado por el cliente o el servidor.

## 2.2 ESTADO DEL ARTE

En esta sección se busca presentar los antecedentes y el estado actual sobre los dispositivos de retroalimentación vibrotáctil disponibles en el mercado, las cuales suelen venir con herramientas

<sup>4</sup>WebSocket API estándar: <https://html.spec.whatwg.org/multipage/web-sockets.html>

<sup>5</sup>Protocolo estándar WebSocket (RFC 6455): <https://tools.ietf.org/html/rfc6455>

para el desarrollo de aplicaciones de terceros. Una de las herramientas que se le suelen facilitar a los desarrolladores es el SDK (*Software Development Kit*), el cual es un conjunto de herramientas utilizadas para desarrollar aplicaciones proporcionadas por proveedores de hardware y software. Los SDK suelen estar compuestos por interfaces de programación de aplicaciones APIs (*Application Program Interface*), código de muestra, documentación, entre otros elementos (Techopedia, 2017). En la Tabla 2.1 muestra una comparativa de distintas características de los SDKs del mercado referente a dispositivos de *haptic feedback*. Es importante señalar que el SDK no posee soporte multiplataforma, tampoco tiene una integración directa con Unity, sería necesario hacer un plugin para ello.

Tabla 2.1: Comparación SDK de distintos Guantes  
Fuente: Elaboración propia (2018)

	Lenguajes soportados	SO soportado	Información Bidireccional	Integración SDK con Unity
OpenGlove	C#, C++, Java y JavaScript	Windows	SI	SI
GloveOne	C++ y C#	Windows, MAC, Linux, Android e iOS	SI	SI
AvatarVR	C++ y C#	Windows, MAC, Linux, Android e iOS	SI	SI
Dexmo	No anunciado	No anunciado	SI	Si

## 2.2.1 OpenGlove

Es un proyecto que consiste en un dispositivo que entrega *haptics feedback*. Fue pensado para dispositivos de realidad virtual e interfaces naturales. El proyecto partió en el 2014 con el propósito de facilitar la construcción y flexibilizar uno de los recursos claves para la inmersión en ambientes de realidad virtual (InTeracTion, 2018). OpenGlove se ha pensado para que pueda ser utilizado en conjunto con otros dispositivos, como Oculus Rift, Kinect y Leap Motion. Por otra parte, los prototipos de OpenGlove, utilizan motores configurables con diferentes niveles de potencia, lo que permite la respuesta vibrotáctil en distintas áreas de la mano. Esto puede ser utilizado para representar la sensación de tocar objetos en entornos virtuales, como también, recibir retroalimentación que represente impacto, el cual sería útil en un juego de boxeo por ejemplo. Actualmente, se soporta el uso de actuadores, sensores de flexibilidad y Unidad de Medición Inercial (IMU por sus siglas en inglés).



Figura 2.2: Guantes OpenGlove  
Fuente: InTeracTion (2018)

## 2.2.2 AvatarVR

AvatarVR es otro guante diseñado por Neurodigital e incluye todas las funcionalidades presentes en GloveOne, junto a unas capacidades adicionales que ofrecen más funcionalidades. Además de los guantes se incluye un accesorio llamado TrackBand, que permite la captura de movimiento de la parte superior del cuerpo, basada en una configuración minimalista de los sensores para en brazos y manos. Además sensores para el seguimiento de dedos mediante sensores 6x 9-AXIS IMUs. El guante y las trackbands poseen un costo desde 1.100 € . Las licencias son de dos tipos profesional a 3.300 € y premium a 13.300 € para acceder a la documentación, SDK, soporte, entre otros servicios.<sup>6</sup>

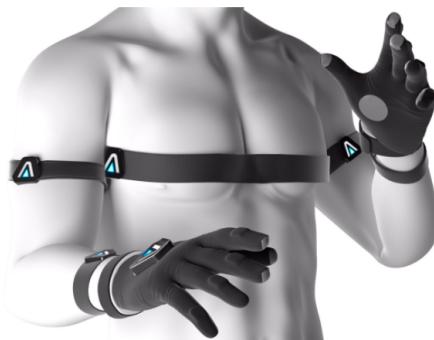


Figura 2.3: Guantes AvatarVR y TrackBand  
Fuente: Neurodigital (2018)

---

<sup>6</sup>Precio AvatarVR y licencias: <https://www.neurodigital.es/avatarvr/>

### 2.2.3 Dexmo

Dexmo es un exoesqueleto que permite *haptic feedback* en ambientes de realidad virtual. Esto lo logra mediante la capacidad de retroalimentación de fuerza, lo que permite al usuario sentir el tamaño y la forma de cualquier objeto digital, lo que mejora enormemente la inmersión. La rigidez variable se logra mediante un control preciso del motor. Con esta característica, cada objeto virtual puede tener su propia rigidez.

Por protección de propiedad intelectual, su SDK sólo es accesible a clientes que hayan comprado Dexmo DK1.



Figura 2.4: Guantes Dexmo  
Fuente: DextaRobotics (2018)

### 2.2.4 Manus VR

Manus VR es un guante que permite *haptic feedback*, seguimiento de dedos y manos para ambientes de realidad virtual. Sus especificaciones comerciales establecen que, es lavable, permite el seguimiento de los brazos, que también incluye un IMU para medir la orientación de la mano. Manus VR puede ser adquirido en dos versiones, la de desarrollador con un precio de 1.990 € y una versión profesional con un precio de 4.990 €<sup>7</sup>

---

<sup>7</sup>Precio de Manus VR y licencias <https://manus-vr.com/order.php>



Figura 2.5: Guantes Manus VR  
Fuente: ManusVR (2018)

## 2.2.5 HaptX

"HaptX presenta los primeros guantes hápticos de grado industrial del mundo. Los guantes HaptX brindan un toque realista, realimentación de fuerza y seguimiento de movimiento preciso" Haptx (2018). Anunciado el 20 de Noviembre del 2017, su principal diferencia con los demás guantes, es la tecnología textil inteligente que desarrollaron, la cual es flexible y basado en silicona. Contiene una serie de actuadores neumáticos de alto desplazamiento y canales de aire microfluídicos integrados. Los actuadores proporcionan retroalimentación háptica empujando contra la piel del usuario, desplazándola de la misma forma en que lo haría un objeto real cuando se toca (Haptx, 2018). Una segunda capa opcional de microcanales puede agregar retroalimentación de temperatura entregando variaciones de agua fría y caliente. No se han anunciado precios, porque es una herramienta de grado industrial diseñada específicamente para empresas y organizaciones.



Figura 2.6: Guantes Haptx  
Fuente: Haptx (2018)

## 2.3 RESUMEN

OpenGlove es un proyecto Open Source<sup>8</sup> que permite la retroalimentación vibrotáctil y comunicación bidireccional entre el guante y las aplicaciones mediante el uso del protocolo WebSocket. Como se ha podido ver, existen diversas alternativas en el mercado de dispositivos hápticos, siendo OpenGlove la alternativa Open Source que no requiere de costos altos de licencias ni del guante, el cual es desarrollado según las necesidades específicas requeridas. Para lograr que la comunidad de desarrolladores de VR/AR/MR pueda integrar OpenGlove en entornos móviles y realizar una fácil configuración de ellos, se desarrollará un SDK que incluye las APIs de alto nivel necesarias para el desarrollo en las plataformas móviles, la aplicación de configuración y la documentación de uso de las APIs en pruebas de concepto. Es importante señalar la importancia de las pruebas de rendimiento sobre la solución a desarrollar. esto toma relevancia cuando se busca disminuir la latencia de los dispositivos conectados. Esto se debe considerar para brindar una excelente experiencia en entornos virtuales sin que se presenten retardos perceptibles.

---

<sup>8</sup> "Open source o código abierto es el término empleado al software distribuido bajo una licencia que permite al usuario acceso al código fuente. Este tipo de licencia posibilita el estudio y la modificación del software con total libertad. Además, su redistribución está permitida siempre y cuando esta posibilidad vaya en concordancia con los términos de licencia bajo la que se adquiere el software" (Ticportal, 2018).

## **CAPÍTULO 3. ANÁLISIS**

En este capítulo se realiza un levantamiento de requisitos funcionales y no funcionales del SDK a desarrollar, considerando el estado actual del proyecto. Para luego realizar los diferentes prototipos de software basados en la metodología RAD, hasta obtener un producto final que cumpla con todos los requisitos establecidos en un comienzo.

### **3.1 LEVANTAMIENTO DE REQUISITOS DE SOFTWARE**

#### **3.1.1 Antecedentes**

En los trabajos anteriores relacionados a OpenGlove, realizados por Monsalve (2015), Meneses (2016) y Cerda (2017) se ha logrado establecer las bases de hardware y software para dar soporte a la retroalimentación vibrotáctil y el seguimiento de las manos. En el trabajo de Meneses (2016), se realizó una extensión de software a OpenGlove, desarrollando un SDK de alto nivel para la retroalimentación vibrotáctil. Por otra parte, el trabajo de Cerda (2017), se realizó una extensión de software y hardware para la captura de movimientos de la mano. En base a estos dos últimos se desprenden las diferentes funcionalidades y soporte que se debe lograr a nivel de software, para dar soporte a los actuadores, sensores de flexibilidad e IMU. Actualmente OpenGlove cuenta con un SDK para Windows, el cual incluye el software de configuración y cuatro APIs de alto nivel para los lenguajes C#, Java, C++ y JavaScript. El SDK fue desarrollado en el IDE Visual Studio.

#### **3.1.2 Requisitos**

Al realizar un análisis de los antecedentes y el problema planteado en el Capítulo 1, se capturan los siguientes requisitos funcionales del software.

Tabla 3.1: Requisitos funcionales de software  
 Fuente: Elaboración propia (2018)

ID	Síntesis del Requisito	Descripción	Origen
RF001	El sistema debe permitir al usuario ver los dispositivos OpenGlove emparejados	El sistema debe ofrecer una recopilación de los dispositivos OpenGlove emparejados con el dispositivo móvil, su estado de conexión actual y la dirección MAC del mismo.	Inicio, modificado de Meneses (2016)
RF002	El sistema debe permitir al usuario definir la configuración de hardware de cada guante	Cada placa LilyPad posee pines programables, en los cuales el usuario puede conectar actuadores, sensores de flexibilidad e IMU para crear su propio OpenGlove. Es necesario que se establezca esta configuración en el sistema para cada guante, ya que de ella depende la activación de actuadores y la lectura de datos de los flexores e IMU. Se debe diferenciar pines digitales y análogos de la placa Arduino	Inicio, modificado de Meneses (2016) y RF001 de Cerda (2017)
RF003	El sistema debe permitir al usuario guardar una configuración de hardware	Al crear un nuevo perfil de hardware para un guante, el sistema debe contar con un mecanismo para la persistencia de esta configuración.	Inicio, Meneses (2016)
RF004	El sistema debe permitir al usuario abrir una configuración de hardware previamente almacenada por el sistema	Una vez guardada una configuración, esta debe ser reconocible por el sistema para su uso posterior en otro guante.	Inicio, Meneses (2016)

RF005	El sistema debe permitir al usuario definir la configuración de actuadores de cada guante	Dependiente de la configuración de hardware, la configuración de actuadores es una representación de la distribución física de los actuadores LilyPad en una mano virtual. Esta representación permite establecer mapeos región-actuador usables en una API de alto nivel. Se debe ofrecer al usuario una solución gráfica que permita ordenar la posición de los actuadores en una representación de la mano. También debe permitir la posibilidad de agregar su propia imagen que represente el mapeo.	Inicio, modificado de Meneses (2016)
RF006	El sistema debe permitir al usuario guardar una configuración de actuadores	Al crear un nuevo perfil de actuadores para un guante, el sistema debe contar con un mecanismo para la persistencia de esta configuración.	Inicio, Meneses (2016)
RF007	El sistema debe permitir al usuario abrir una configuración de actuadores previamente almacenada por el sistema	Una vez guardada una configuración de actuadores, esta debe ser reconocible por el sistema para su uso posterior en otro guante.	Inicio, Meneses (2016)
RF008	El sistema debe permitir al usuario establecer conexión con un dispositivo OpenGlove emparejado	Una vez emparejado un guante OpenGlove, el sistema debe permitir que el usuario inicie la conexión Bluetooth. No es necesario que el usuario especifique la dirección MAC del guante.	Inicio, modificado de Meneses (2016)

RF009	El sistema debe permitir al usuario activar una región de un guante con intensidad a voluntad	El sistema debe exponer al usuario una sección que le permita activar una región de un guante con la intensidad (entre 0 y 255) que él desee para probar el hardware. Esta región esta predefinida y debe ser independiente de la configuración de hardware (actuadores) presente en el guante. Esta función debe estar disponible para uno o varios actuadores en un guante.	Inicio, modificado de Meneses (2016)
RF010	El sistema debe permitir al usuario operar con distintas implementaciones de OpenGlove	Al momento de crear un nuevo perfil de hardware, el sistema debe proveer un mecanismo para que el usuario genere su propia placa, lo que se traduce en un nombre y una cantidad de pines para poder usarla en su configuración.	Inicio, Meneses (2016)
RF011	El sistema actual debe permitir al usuario guardar y cargar una configuración de hardware incluyendo los flexores	El SDK debe ser capaz de guardar y cargar una configuración de hardware, compuesta por actuadores y/o sensores de flexibilidad.	Inicio, modificado de Cerda (2017)
RF012	El sistema debe permitir al usuario crear una configuración de los sensores de flexibilidad y del sensor de rastreo IMU	El software debe ser capaz de dar soporte para la creación de nuevas configuraciones de los sensores de flexibilidad y el IMU.	Inicio, modificado de Cerda (2017)

RF013	El sistema debe permitir al usuario seleccionar un sensor de flexibilidad y relacionarlo a una región del guante	La configuración correspondiente a los sensores de flexibilidad, debe ser capaz de seleccionar una región del guante y relacionarlo con un sensor de flexibilidad.	Inicio, Cerda (2017)
RF014	El sistema debe permitir al usuario eliminar un sensor de flexibilidad de una región del guante	La configuración de los sensores de flexibilidad debe ser capaz de eliminar un flexor de una región del guante, de tal manera que la región quede libre y el flexor pueda asignarse a una nueva región.	Inicio, Cerda (2017)
RF015	El sistema debe enviar los datos provenientes de los sensores de flexibilidad automáticamente	Cuando un sensor de flexibilidad es asignado a una región del guante, el sistema debe ser capaz de transmitir los datos de dicho sensor de manera automática, especificando el tipo de dato, región y el valor leído.	Inicio, Cerda (2017)
RF016	El sistema debe parar de enviar los datos provenientes de los sensores de flexibilidad automáticamente	Cuando un sensor de flexibilidad es eliminado de una región del guante, el sistema debe ser capaz de parar la transmisión de datos de dicho flexor automáticamente.	Inicio, Cerda (2017)
RF017	El sistema debe permitir al usuario definir un threshold al momento de enviar datos de los sensores de flexibilidad	El guante enviará el dato de un flexor, solo si este dato posee una diferencia mayor o igual al valor definido como threshold, con respecto al último valor enviado por dicho flexor.	Inicio, Cerda (2017)
RF018	El sistema debe permitir al usuario la opción de calibrar los sensores de flexibilidad	Los datos provenientes de los sensores de flexibilidad deben ser calibrados con respecto al máximo y mínimo valor leído de la articulación de un dedo.	Inicio, Cerda (2017)

RF019	El sistema debe permitir al usuario probar los sensores de flexibilidad	Luego de asignar uno o más sensores de flexibilidad a una región del guante, se debe habilitar un botón que permita probar si la configuración es correcta, visualizando el valor entregado por cada flexor en dicha región.	Inicio, Cerda (2017)
RF020	El usuario debe permitir al usuario poder obtener datos desde un sensor de rastreo IMU	La configuración correspondiente al sensor de rastreo IMU, debe ser capaz de activar o desactivar el envío de datos de ésta.	Inicio, Cerda (2017)
RF021	El sistema debe permitir al usuario poder obtener datos crudos o procesados desde el sensor de rastreo IMU	La configuración del sensor de rastreo IMU, debe ser capaz de definir si los datos enviados por ésta son procesados o no.	Inicio, Cerda (2017)
RF022	El sistema debe permitir al usuario poder probar el sensor de rastreo IMU	Luego de activar el envío de datos del sensor de rastreo IMU, se debe activar un botón que active la visualización de todos los datos entregados por el sensor.	Inicio, Cerda (2017)
RF023	El sistema debe permitir al usuario calibrar el sensor de rastreo IMU	Los datos provenientes del IMU deben ser calibrados bajo una posición de referencia, para poder determinar la posición y orientación de la mano.	Nuevo

La Tabla 3.2, muestra los requisitos no funcionales obtenidos mediante las reuniones con el profesor guía. Al igual que lo señalado por Monsalve (2015), de esta lista de requisitos para el SDK, se debe dar especial importancia a RNF004, el cual se refiere al umbral de tiempo que no se debe superar para ofrecer la retroalimentación táctil. Experimentos enfocados en medir los tiempos en que una persona puede percibir inconsistencias entre una interacción táctil con un objeto virtual y un estímulo táctil recibido, obtienen los siguientes resultados: 54 ms para retroalimentación kinestésica (Batteau et al., 2004), 60 ms en estudios de percepción (Okamoto et al., 2009) y 60 ms simulaciones quirúrgicas simulaciones (Wu et al., 2015).

Tabla 3.2: Requisitos no funcionales de software  
 Fuente: Elaboración propia (2018)

ID	Síntesis del requisito	Descripción	Origen
RNF001	Sistema multiplataforma	El software a desarrollar debe ser ejecutado inicialmente en el sistema operativo Android, sin embargo, el proyecto en un futuro debe poder dar soporte al sistema operativo iOS.	Inicio
RNF002	Soportar comunicación Bluetooth	El sistema debe ser capaz de comunicarse con los dispositivos OpenGlove mediante Bluetooth Clásico.	Inicio
RNF003	Interoperabilidad	El SDK debe permitir la interoperabilidad entre los lenguajes de programación C# y Java.	Inicio
RNF004	Umbral latencia aceptada	El SDK no debe superar el umbral de 60 ms de latencia perceptible por el usuario.	Inicio

## 3.2 PROTOTIPOS

A continuación se muestran los diferentes prototipos desarrollados durante la escritura de esta memoria. Los prototipos serán detallados utilizando una tabla resumen del mismo, la que incluye, los Objetivos del prototipo, una Descripción, los requisitos funcionales y no funcionales que aborda el prototipo. Se incluyen las capturas del trabajo realizado, el análisis y conclusión del avance. Luego de la revisión de los prototipos, finalmente se detalla el desarrollo de las APIs, las cuales fueron desarrolladas en paralelo a los prototipos de la aplicación de configuración.

### 3.2.1 Primer prototipo: Activación de motores

Este primer prototipo tiene como principal objetivo establecer las bases necesarias para conectar OpenGlove con un dispositivo Android y establecer una comunicación básica. Para ello se utiliza la documentación oficial de Android sobre las conexiones bluetooth <sup>1</sup>, permitiendo en primera instancia la obtención de dispositivos vinculados y la conexión con alguno de los mismos. No es necesaria la modificación de código cargado en la placa arduino, puesto que los protocolos de comunicación (mensajes) ya han sido establecidos con anterioridad. Por tanto se procede a hacer uso de la API en Java de bajo nivel desarrollada por Monsalve (2015). El prototipo se resume en la Tabla 3.3.

---

<sup>1</sup>Documentación oficial android: <https://developer.android.com/guide/topics/connectivity/bluetooth>

Tabla 3.3: Primer prototipo  
Fuente: elaboración propia (2018).

ID del prototipo	P001
Nombre	Activación de motores
Objetivos	Verificar la correcta activación de motores en una aplicación de Android nativo, utilizando las APIs de bajo nivel disponibles.
Descripción	El primer prototipo desarrollado hace uso de la API de bajo nivel de Java desarrollada por Monsalve (2015), sumándose modificaciones realizadas para establecer la conexión en Android. Se obtiene un prototipo capaz de establecer una conexión Bluetooth con dispositivos previamente vinculados, permitiendo activar y desactivar un motor de manera remota.
Requisitos funcionales	RF001
Requisitos no funcionales	RNF002, RNF004

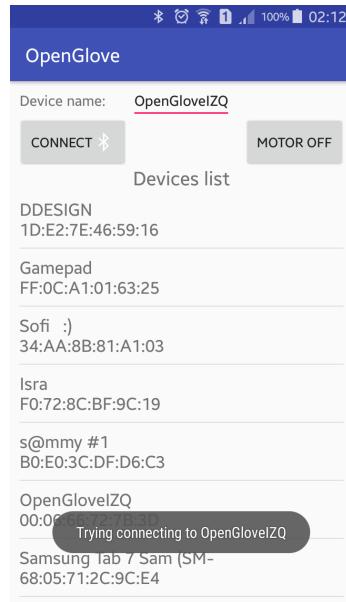


Figura 3.1: Primer prototipo: Activación de motores  
Fuente: elaboración propia (2018).

Para lograr el objetivo propuesto, luego obtener los dispositivos vinculados y de la conexión con el guante mediante las APIs de bluetooth de Android, se procedió a enviar mensajes bajo el protocolo establecido en el desarrollo de Monsalve (2015). Dicho de una manera más detallada se utilizó la clase *Message Generator* de la API de bajo nivel y la implementación nativa en Android para la escritura serial por medio de bluetooth. La Figura 3.1 consiste en el primer prototipo que muestra el listado de los dispositivos vinculados, la posibilidad de la conexión con el dispositivo deseado, permitiendo finalmente activar y desactivar el motor seleccionado para las pruebas. Para administrar la conexión entre los dispositivos, se requiere de un hilo<sup>2</sup> encargado de ello

<sup>2</sup>Hilo o Thread: En palabras muy resumidas, un hilo es una secuencia instrucciones dentro de un programa que se puede ejecutar independientemente de otro código, permitiendo así la ejecución en paralelo de instrucciones.

(*ConnectedThread*), el cual se comunica con el hilo de la *interfaz de usuario* (UI desde ahora) mediante mensajes. El uso de hilos, permite administrar multiples conexiones con dispositivos Bluetooth de manera simultánea. En conclusión es posible realizar envio de mensajes bajo el protocolo que acepta OpenGlove desde una aplicación Android nativa en Java.

### 3.2.2 Segundo prototipo: Obtención de datos desde los flexores

En el segundo prototipo se mantiene lo desarrollado previamente, añadiendo en esta iteración el soporte para los flexores. En este caso es necesaria la lectura desde el dispositivo OpenGlove. En la Tabla 3.4 se muestra el resumen del prototipo ya mencionado. El RNF004 es cubierto parcialmente, debido a que se consulta continuamente el valor del flexor mediante funciones de bajo nivel, aumentando la latencia debido a las continuas consultas. No se presentan problemas de latencia para la activación y desactivación de motores.

Tabla 3.4: Segundo prototipo  
Fuente: elaboración propia (2018).

ID del prototipo	P002
Nombre	Obtención de datos desde los flexores.
Objetivos	Verificar la correcta obtención de datos desde los flexores en la aplicación de Android nativo, utilizando las APIs de bajo nivel disponibles.
Descripción	El segundo prototipo desarrollado agrega los métodos disponibles de los flexores de la API de bajo nivel C# hecha por Cerda (2017). De esta manera, se obtiene un prototipo capaz de obtener los datos del flexor.
Requisitos funcionales	RF001
Requisitos no funcionales	RNF002, RNF004 (parcialmente)

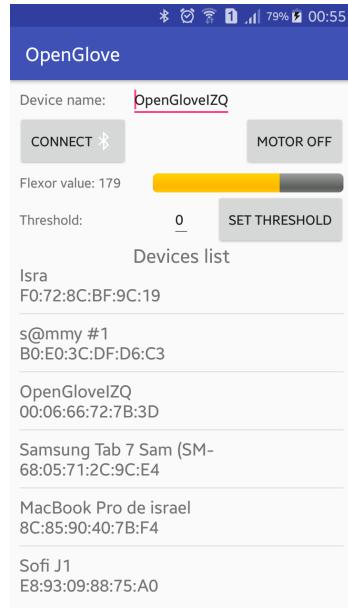


Figura 3.2: Segundo prototipo: obtención de datos desde flexores  
Fuente: elaboración propia (2018).

Para realizar la captura de datos del flexor, se obtiene valor actual del pin al cual el está conectado, esto se logra con el desarrollo de la función `analogRead(pin)` en Java basada en la API C# de bajo nivel Cerda (2017). El hilo encargado de la administrar conexión (*ConnectedThread*), tiene la responsabilidad de leer los mensajes desde OpenGlove y actualiza la UI enviando como un mensaje entre hilos el valor obtenido del flexor. Además se agregan las demás funciones de generación de mensajes relacionadas a los flexores en la API C# ya mencionada. En la figura 3.2 se puede ver el estado actual del flexor el cual varía en un rango de entre 60 a 300 y 170 el valor medio del flexor sin aplicar fuerza.

### 3.2.3 Tercer prototipo: Activación de motores y obtención de datos desde los flexores

En el segundo prototipo fue posible la activación del motor y obtener la información del flexor, permitiendo así comprobar la factibilidad de un desarrollo nativo en Android. En este tercer prototipo, se buscó dar soporte multiplataforma al proyecto, considerando la importancia de mantener umbrales de latencia, se optó por Xamarin. En concreto Xamarin.Forms, que es una tecnología de desarrollo móvil multiplataforma <sup>3</sup>, el cual permite desarrollar aplicaciones nativas para Android e iOS en C#. La tabla 3.5 muestra el resumen del tercer prototipo. El RNF004 es cubierto parcialmente, debido a que se consulta continuamente el valor del flexor al igual que el

<sup>3</sup>Traducción libre

segundo prototipo. De igual manera no se presentan problemas de latencia para la activación y desactivación de motores.

Tabla 3.5: Tercer prototipo  
Fuente: elaboración propia (2018).

ID del prototipo	P003
Nombre	Activación de motores y obtención de datos desde los flexores.
Objetivos	Verificar la correcta activación de motores y la obtención de datos desde los flexores en la aplicación de Android nativo con Xamarin.Forms, utilizando las APIs C# de bajo nivel disponibles.
Descripción	El tercer prototipo desarrollado agrega las mismas funcionalidades que en el segundo prototipo. De esta manera, se obtiene un prototipo capaz de obtener los datos del flexor.
Requisitos funcionales	RF001
Requisitos no funcionales	RNF001, RNF002, RNF004 (parcialmente)

La figura 3.3 muestra el prototipo hecho en Xamarin.Forms, el cual es similar al segundo prototipo, con la diferencia en la forma de conectarse a un dispositivo bluetooth, el cual difiere en la forma de conectarse, este prototipo requiere presionar el dispositivo y aceptar el mensaje que explica el intento de conexión.

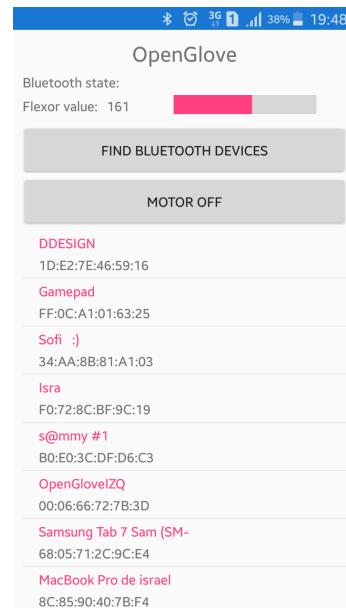


Figura 3.3: Tercer prototipo: activación de motores y obtención de datos desde flexores  
Fuente: elaboración propia (2018).

### **3.2.4 Cuarto prototipo: Navegación aplicación, administración dispositivos Bluetooth, servidor WebSocket y configuración de la placa**

Este prototipo tiene como propósito cubrir los diferentes requisitos funcionales relacionados al comportamiento de la aplicación y ser la base de las siguientes iteraciones para dar soporte a los sensores de flexibilidad, actuadores e IMU. Se establece la navegación que posee la aplicación para hacer uso de todas las funcionalidades, por medio del diseño de un prototipo usando la herramienta AdobeXD en su versión gratuita, con el cual se diseña, prototipa y se generan las diferentes imágenes e íconos que se utilizaron en el desarrollo del cuarto, quinto y sexto prototipo. La Figura 3.4, muestra el prototipo desarrollado en Xamarin.Forms para Android y iOS, los cuales comparten la interfaz de usuario diferenciándose en la representaciones nativas de los elementos de cada plataforma. La Tabla 3.6 muestra el resumen del prototipo ya mencionado.

Tabla 3.6: Cuarto prototipo  
Fuente: elaboración propia (2018)

ID del prototipo	P004
Nombre	Navegación aplicación, administración dispositivos Bluetooth, servidor WebSocket y configuración de la placa.
Objetivos	- Desarrollar Navegación de la aplicación (soporte iOS y Android). - Desarrollar Administración de dispositivos Bluetooth. - Desarrollar Administración del servidor WebSocket. - Desarrollar la Configuración de la placa.
Descripción	Este cuarto prototipo abarca los diferentes requisitos funcionales relacionados al comportamiento de la aplicación y de implementaciones de tecnología como el servidor WebSocket embebido en la aplicación.
Requisitos funcionales	RF001, RF002, RF003, RF004, RF008, RF010
Requisitos no funcionales	RNF001, RNF002, RNF004

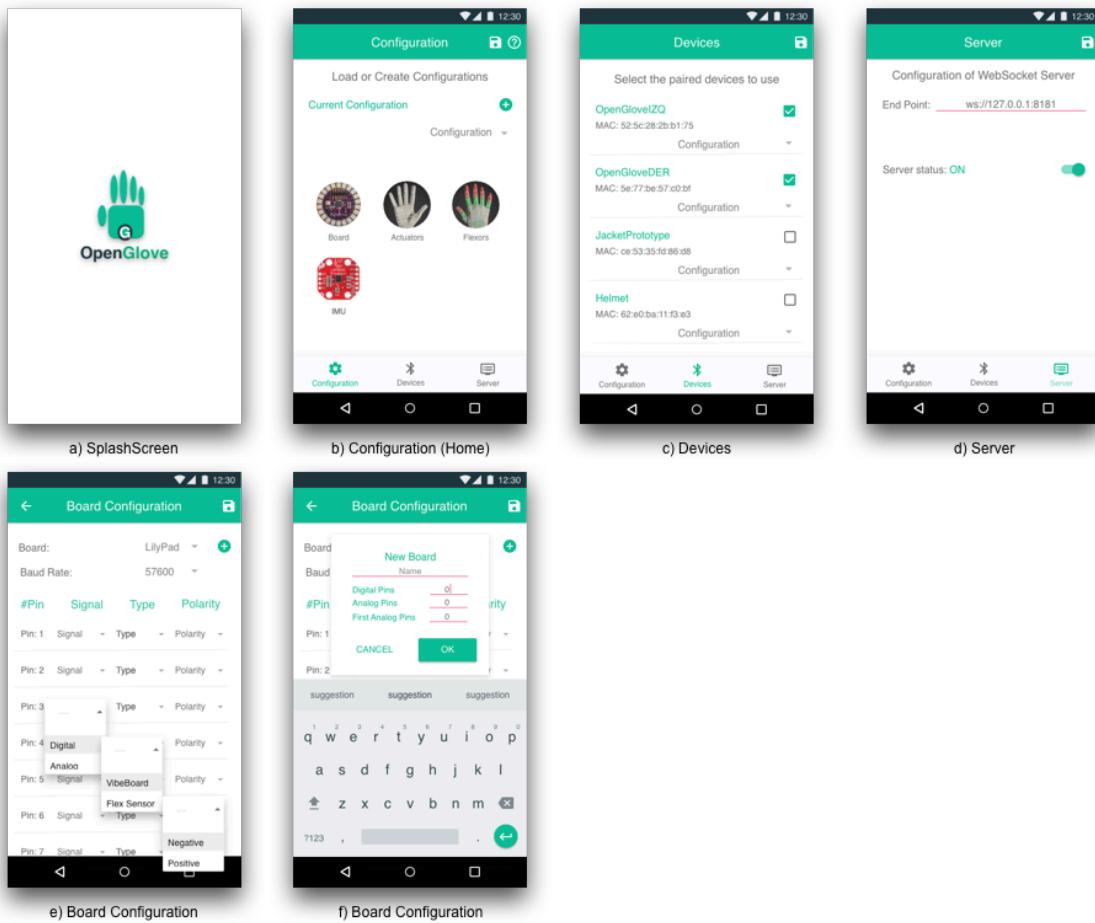


Figura 3.4: Cuarto prototipo: app navigation, devices, server and board configuration  
Fuente: elaboración propia (2018).

A continuación se describen las pantallas desarrolladas, especificando el alcance de las mismas para iOS y Android.

- La Figura 3.4.a muestra el SplashScreen de la aplicación, el cual se muestra mientras se inicia la aplicación. **[Soporte completo: iOS y Android]**
- La Figura 3.4.b muestra el menú principal de la aplicación de configuración, en el cual es posible cargar configuraciones existentes o bien crear una nueva configuración, la cual contiene la configuración de la placa, de los actuadores, de los flexores y del sensor IMU. Esta aplicación seleccionada o creada puede ser modificada del menú principal accediendo a las pantallas que permiten configurar la placa, actuadores, flexores e IMU. La aplicación posee una navegación basada en BottomNavigation, la cual corresponde a una barra de navegación inferior que permite acceder al menú de configuración general, la configuración de dispositivos Bluetooth y la configuración del servidor. **[Soporte completo: iOS y Android]**

- La Figura 3.4.c muestra una lista con los dispositivos Bluetooth emparejados, permitiendo ver el nombre y dirección MAC de ellos. También permite elegir los dispositivos Bluetooth que se utilizarán junto la configuración que se le desee aplicar. La aplicación en iOS no puede mostrar la lista de dispositivos emparejados, no se implementó la clase Communication para el proyecto iOS, porque está fuera del alcance de este trabajo **[Soporte completo: Android]**.
- La Figura 3.4.d muestra la pantalla referente al servidor WebSocket, permitiendo definir la dirección de acceso al servidor y el encendido y apagado del mismo. Para la implementación del servidor WebSocket se utilizó el paquete de software Fleck, el que se encuentra disponible en la plataforma NuGet <sup>4</sup> junto a otros paquetes que pueden ser utilizados si son compatibles con la versión .NET standar 2.0. Con este paquete se desarrolló la clase OpenGloveServer para su uso en la aplicación de configuración. En esa clase se instancia un servidor y donde se establecen los EventHandler necesarios para mandar mensajes entre los hilos que administran las conexiones de sus dispositivos Bluetooth (clase Communication) y el servidor WebSocket (clase OpenGloveServer) **[Soporte completo: iOS y Android]**.
- La Figura 3.4.e muestra la pantalla que permite definir la configuración de la placa, creando una nueva placa o utilizando alguna previamente creada, como puede ser la placa LilyPad utilizada en este trabajo. En esta pantalla de la aplicación, se puede elegir o crear una placa y seleccionar el BaudRate <sup>5</sup> con el que se trabajará. Luego de ello se procede a configurar los pines de placa considerando la configuración física de los mismos, definiendo el tipo de señal, pines digitales y análogos, si el pin corresponde a flexor o un actuador e indicando su polaridad, positiva o negativa **[Soporte completo: iOS y Android]**.
- La Figura 3.4.f muestra el diálogo utilizado para crear una nueva configuración de placa, pidiendo el nombre de la placa, el pin donde comienzan los pines análogos (depende de la placa específica (Cerda, 2017)), cantidad de pines digitales y análogos **[Soporte completo: iOS y Android]**.

### 3.2.5 Quinto prototipo: Mapeo de actuadores y prueba de actuadores

Este prototipo tiene como propósito cubrir los diferentes requisitos funcionales relacionados a los actuadores. La Tabla 3.7 muestra el resumen del prototipo antes mencionado. Este prototipo utilizó como base el cuarto prototipo desarrollado.

---

<sup>4</sup>Paquetes de software NuGet <https://www.nuget.org/packages>

<sup>5</sup>El baud rate especifica la velocidad en la que se envían los datos por una línea serial, usualmente son expresadas en unidades de bits por segundo (bps). <https://learn.sparkfun.com/tutorials/serial-communication/rules-of-serial>

Tabla 3.7: Quinto prototipo  
Fuente: elaboración propia (2018)

ID del prototipo	P005
Nombre	Mapeo de actuadores y prueba de actuadores
Objetivos	- Desarrollar el mapeo actuador-region - Desarrollar la prueba de actuadores
Descripción	Este quinto prototipo abarca los diferentes requisitos funcionales relacionados a los actuadores, sumándose a lo desarrollado en el cuarto prototipo.
Requisitos funcionales	RF001, RF002, RF003, RF004, RF005, RF006, RF007, RF008, RF009, RF010, RF011
Requisitos no funcionales	RNF001, RNF002, RNF004

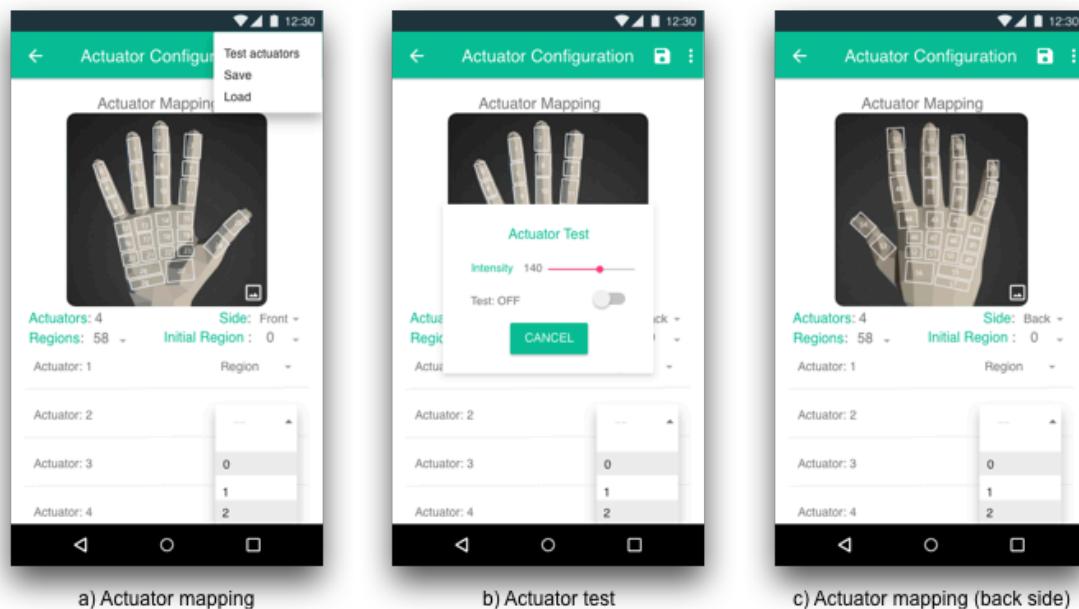


Figura 3.5: Quinto prototipo: actuator mapping and actuator test  
Fuente: elaboración propia (2018).

En paralelo al desarrollo de este prototipo fue necesario el desarrollo de la API C#, es decir el desarrollo de un cliente WebSocket utilizando el paquete de software WebSocketSharp<sup>6</sup>. El cliente es quien envía mensajes al servidor que está dentro de la aplicación de configuración, para activar los actuadores deseados. Para ello se incluyen métodos para la activación de motores, en el cual se especifica el dispositivo Bluetooth y una lista de regiones a activar, las cuales son traducidas a los mensajes generados por la API de bajo nivel de OpenGlove.

A continuación se describen las pantallas desarrolladas, especificando el alcance de las mismas para iOS y Android.

- La Figura 3.5.a muestra la pantalla que permite el mapeo actuador-región, mostrando una

<sup>6</sup>WebSocketSharp: <https://www.nuget.org/packages/WebSocketSharp/1.0.3-rc11>

lista de los actuadores identificados desde la configuración de la placa. Es posible cambiar la imagen que representa el mapeo del dispositivo Bluetooth, para ello se requiere definir la cantidad de regiones que posee el lado frontal y trasero al momento de definir la cantidad de regiones, especificando también el número inicial que identifica a las regiones (iniciando en 1 o en 0 por ejemplo). El mapeo se realiza seleccionando la región desde una lista desplegable en cada actuador, la cual es generada dependiendo de la configuración inicial de la cantidad de regiones. **[Soporte completo: iOS y Android]**.

- La Figura 3.5.b muestra el diálogo que permite realizar las pruebas de los actuadores mapeados, permitiendo iniciar y detener la prueba con la intensidad seleccionada. La prueba se termina una vez se cierra el diálogo o se desactiva dentro del diálogo. La aplicación en iOS no puede activar regiones mapeadas, porque no se implementó la clase Communication para el proyecto iOS, porque está fuera del alcance de este trabajo **[Soporte completo: Android]**.
- La Figura 3.5.c muestra el segundo lado de la imagen de mapeo de actuadores, de la pantalla de mapeo mostrada en la Figura 3.5.a.

### 3.2.6 Sexto prototipo: Mapeo de flexores, prueba de flexores y la configuración del IMU.

Este prototipo tiene como propósito cubrir los diferentes requisitos funcionales relacionados a los flexores e IMU. Se utilizó como base el quinto prototipo ya descrito.

Este prototipo tiene como propósito cubrir los diferentes requisitos funcionales relacionados a los flexores e IMU. La Tabla 3.8 muestra el resumen del prototipo antes mencionado. Este prototipo utilizó como base el quinto prototipo desarrollado.

Tabla 3.8: Sexto prototipo  
Fuente: elaboración propia (2018)

ID del prototipo	P006
Nombre	Mapeo de flexores, prueba de flexores y la configuración del IMU.
Objetivos	- Desarrollar el mapeo flexor-region - Desarrollar la prueba de flexores - Desarrollar la configuración del IMU
Descripción	Este sexto prototipo abarca los diferentes requisitos funcionales relacionados a los flexores e IMU, sumándose a lo desarrollado en el quinto prototipo. Esto es posible gracias al desarrollo de la API C#, la cual es replicada para el lenguaje de programación Java.
Requisitos funcionales	RF001, RF002, RF003, RF004, RF005, RF006, RF007, RF008, RF009, RF010, RF011, RF012, RF013, RF014, RF015, RF016, RF018, RF019, RF020, RF021, RF022, RF023
Requisitos no funcionales	RNF001, RNF002, RNF003, RNF004, RNF005, RNF006

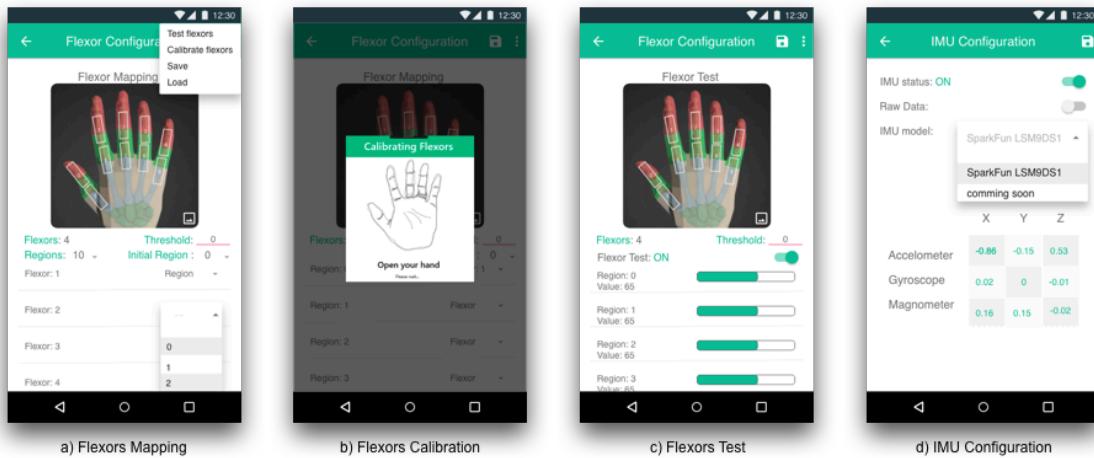


Figura 3.6: Sexto prototipo: flexor mapping, flexor test and IMU configuration  
Fuente: elaboración propia (2018).

De igual manera al quinto prototipo, se agregan métodos que permiten el desarrollar las funcionalidades referentes a los flexores e IMU, permitiendo definir el Threshold (límite en que, se enviará un dato de un flexor si la diferencia es mayor o igual a este valor), agregar o remover flexores de una región, calibrar los flexores y confirmar la calibración. También permite iniciar o detener el IMU y definir si recibir datos crudos o procesados.

A continuación se describen las pantallas desarrolladas, especificando el alcance de las mismas para iOS y Android.

- La Figura 3.6.a muestra la pantalla de mapeo flexor-region de igual manera que el de actuadores, agregando la asignación del Threshold que se utilizará cuando en la prueba

de flexores y cuando se le asigne la configuración al o los dispositivos Bluetooth. **[Soporte completo: iOS y Android].**

- La Figura 3.6.b muestra una pantalla sobrepuerta con una animación, la cual indica que es necesario mover las articulaciones para calibrar los flexores. La aplicación en iOS no puede calibrar los flexores, porque no se implementó la clase Communication para el proyecto iOS, porque está fuera del alcance de este trabajo **[Soporte completo: Android].**
- La Figura 3.6.c muestra la pantalla que permite probar los flexores previamente mapeados, también se puede definir el Threshold antes de activar o desactivar la prueba de los flexores. La aplicación en iOS no puede obtener datos de los flexores, porque no se implementó la clase Communication para el proyecto iOS, porque está fuera del alcance de este trabajo **[Soporte completo: Android].**
- La Figura 3.6.d muestra la pantalla de configuración del sensor de rastreo IMU, en la cual se puede iniciar o detener el mismo, configurar el envío de datos crudos o procesados. También permite ver los datos recibidos por el acelerómetro, giroscopio y magnetómetro para los ejes *X*, *Y* y *Z*. Agregar soporte a otros modelos de sensores de rastreo IMU. No es posible seleccionar otro modelo de IMU aparte de SparkFun LSM9DS1, dado que requiere bibliotecas de software específicas en el código de la placa Arduino. Podría definirse que la selección de modelo de placa IMU, permita cargar código en la placa Arduino, incluyendo las bibliotecas y código específico del modelo del IMU. Esto podría ser realizado utilizando la conexión Bluetooth o por medio de conexión USB adecuada. Por ejemplo, la herramienta Bluino Loader - Arduino IDE<sup>7</sup> permite cargar código a la placa Arduino por Bluetooth o USB. La aplicación en iOS no puede calibrar obtener datos desde el IMU, porque no se implementó la clase Communication para el proyecto iOS, porque está fuera del alcance de este trabajo **[Soporte completo: Android].**

### 3.2.7 APIs

Para que los desarrolladores puedan hacer uso de las funcionalidades disponibles en el SDK, se requiere del soporte de distintos lenguajes de programación permitiendo abarcar a las más importantes plataformas tecnológicas de Realidad Virtual, Aumentada y Mixta. Por lo tanto se considera la interoperabilidad para los lenguajes C# y Java, como se especificó en el Requisito no funcional RNF003. Estas APIs fueron desarrolladas paralelamente a la aplicación de configuración pues se hace uso de ella, iniciando con la API de alto nivel de C#. Una vez que la API C#

---

<sup>7</sup>Bluino Loader: <https://www.hackster.io/mansurkamsur/upload-sketch-arduino-over-bluetooth-using-android-f1ce55>

fue completada, se replicó utilizando el lenguaje Java. Las mencionadas APIs son clientes WebSockets que establecen conexión con el Servidor WebSocket provisto y administrado por la aplicación de configuración. El primer, segundo y tercer prototipo no incluyen el desarrollo de las APIs de alto nivel, porque estos prototipos abarcaron permitieron evaluar la factibilidad de distintas tecnologías y definir el trabajo a seguir en los siguientes prototipos.

El capítulo Diseño e Implementación especifica en mayor detalle los aspectos arquitecturales de la solución y del comportamiento en los cuales las APIs están involucradas.

### *3.2.7.1 Cuarto prototipo*

Este prototipo no incluye desarrollo de la APIs de alto nivel, pero si la implementación de distintas tecnologías y la definición del protocolo de mensajes a utilizar por las APIs de los siguientes prototipos. En este prototipo se implementó un servidor WebSocket que recibe conexiones entrantes de clientes WebSocket, los cuales pueden ser la aplicación de configuración y una aplicación de VR por ejemplo. El servidor WebSocket puede activarse o desactivarse desde la aplicación de configuración, permitiendo además definir el punto de acceso o EndPoint del Servidor. El servidor WebSocket se comunica con la API de Bajo nivel mediante el uso de EventHandlers, los cuales permiten por una parte al Servidor recibir los mensajes provenientes de los dispositivos Bluetooth (datos de flexores e IMU) como también el enviar mensajes para la activación de actuadores, obtención de lista de dispositivos vinculados y la conexión de a los mismos. Es importante destacar que la administración de la conexión de dispositivos Bluetooth está implementada para el Sistema Operativo Android, por lo que es necesario realizar la implementación específica para iOS.

### *3.2.7.2 Quinto prototipo*

En este prototipo se abarcan las funcionalidades referidas a los actuadores, por tanto se aplicó el protocolo de comunicación definido en el cuarto prototipo, entre los clientes y el servidor WebSocket. Con ello se logró cubrir la inicialización, activación y mapeo de los actuadores mediante la API de alto nivel en C#. Gracias a estas funcionalidades, la aplicación de configuración permite realizar el mapeo y pruebas de activación de los actuadores de los distintos dispositivos OpenGlove conectados.

### *3.2.7.3 Sexto prototipo*

Este prototipo incluye todas las funcionalidades de la API C# para el uso de OpenGlove en dispositivos móviles utilizando este lenguaje de programación. Esto se logró desarrollando las funcionalidades que no fueron cubiertas en el quinto prototipo, las referidas a los sensores de flexibilidad e IMU. Las funcionalidades implementadas respecto a los sensores de flexibilidad , permiten agregar agregar un flexor a una región para iniciar la transmisión de datos, remover un flexor de una región, calibrar los flexores, testear los flexores asignados a una región y la asignación de un umbral o Threshold (el dispositivo OpenGlove, enviará el dato de un flexor si la diferencia es mayor o igual a este valor). Respecto a las funcionalidades referidas al IMU, se permite iniciar el IMU, asignar el status del IMU, recibir datos crudos o procesados del IMU. Inicio y suspensión de lectura de datos es una funcionalidad que involucra a ambos sensores.

## **3.3 RESUMEN**

Durante el desarrollo del SDK, fue necesario un desarrollo de seis prototipos, los primeros tres que permitieron la validando la factibilidad de un desarrollo nativo en Android comparado al desarrollo multiplataforma con Xamarin.Forms. Los siguientes prototipos corresponden al desarrollo de la aplicación de configuración y las APIs para lograr cumplir con los requisitos funcionales y no funcionales del SDK. La Tabla 3.9 muestra el resumen de los requisitos funcionales, en el cual se aprecia la cobertura de los mismos en cada prototipo anteriormente detallado. La Tabla 3.10, muestra de la misma forma la cobertura de los requisitos no funcionales del SDK.

Tabla 3.9: Matriz de prototipos de software vs requisitos funcionales  
 Fuente: Elaboración propia (2018)

	P001	P002	P003	P004	P005	P006
RF001	X	X	X	X	X	X
RF002				X	X	X
RF003				X	X	X
RF004				X	X	X
RF005					X	X
RF006					X	X
RF007					X	X
RF008	X	X	X	X	X	X
RF009					X	X
RF010				X	X	X
RF011					X	X
RF012						X
RF013						X
RF014						X
RF015						X
RF016						X
RF017						X
RF018						X
RF019						X
RF020						X
RF021						X
RF022						X
RF023						X

Tabla 3.10: Matriz de prototipos de software vs requisitos no funcionales  
 Fuente: Elaboración propia (2018)

	P001	P002	P003	P004	P005	P006
RNF001			X	X	X	X
RNF002	X	X	X	X	X	X
RNF003						X
RNF004	X	X		X	X	X

# CAPÍTULO 4. DISEÑO E IMPLEMENTACIÓN

## 4.1 ARQUITECTURA GENERAL

La Figura 4.1, muestra un diagrama conceptual general de OpenGlove, donde es posible ver el trabajo previo realizado por Monsalve (2015), Meneses (2016) y Cerda (2017), como también el trabajo que resulta de este proyecto y el que se realizará en un futuro.

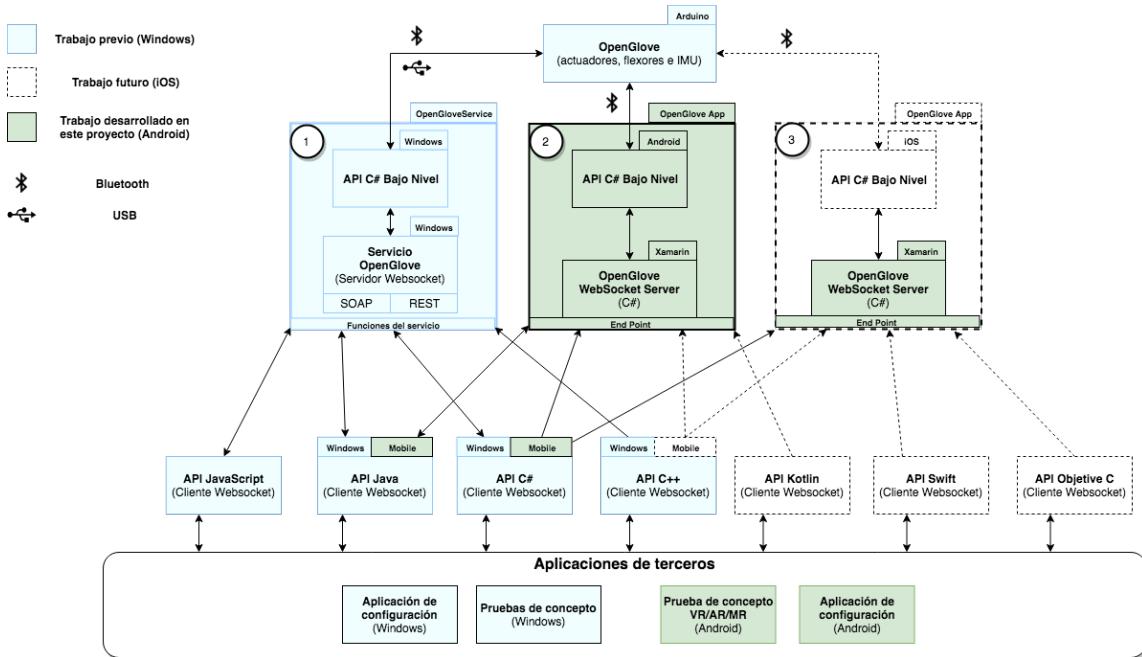


Figura 4.1: Diagrama conceptual general de OpenGlove  
Fuente: Elaboración propia (2018)

La aplicación de configuración de OpenGlove desarrollada en este proyecto, permite comunicarse con dispositivos OpenGlove a través de comunicación serial Bluetooth. Para acceder a sus funcionalidades los desarrolladores hacen uso de las APIs de alto nivel, las cuales corresponden a clientes WebSocket. Éstas pueden ser utilizadas en cualquier aplicación compatible con el lenguaje de la API a utilizar. Dado que se utiliza Xamarin.Forms para desarrollar aplicaciones nativas multiplataforma, para iOS y Android, el servidor Websocket es desarrollado utilizando bibliotecas de software de C#, para que ambas plataformas comparten el código. La API de bajo nivel, debe ser específica en cada sistema operativo, por las diferencias de comunicación presentes en cada uno. Por esta misma razón se desarrollan las APIs de alto nivel en Java y C#. La primera permite hacer uso de OpenGlove en proyectos Android y la segunda permite hacer aplicaciones para iOS y Android utilizando Xamarin. El uso de Xamarin.Forms permite reutilizar

código de la Interfaz de Usuario y desarrollar de manera específica para cada plataforma utilizando un mismo lenguaje de programación. Cabe destacar que la comunicación realizada entre las APIs y la aplicación es mediante WebSockets considerando que OpenGlove es una aplicación que transmite datos en tiempo real.

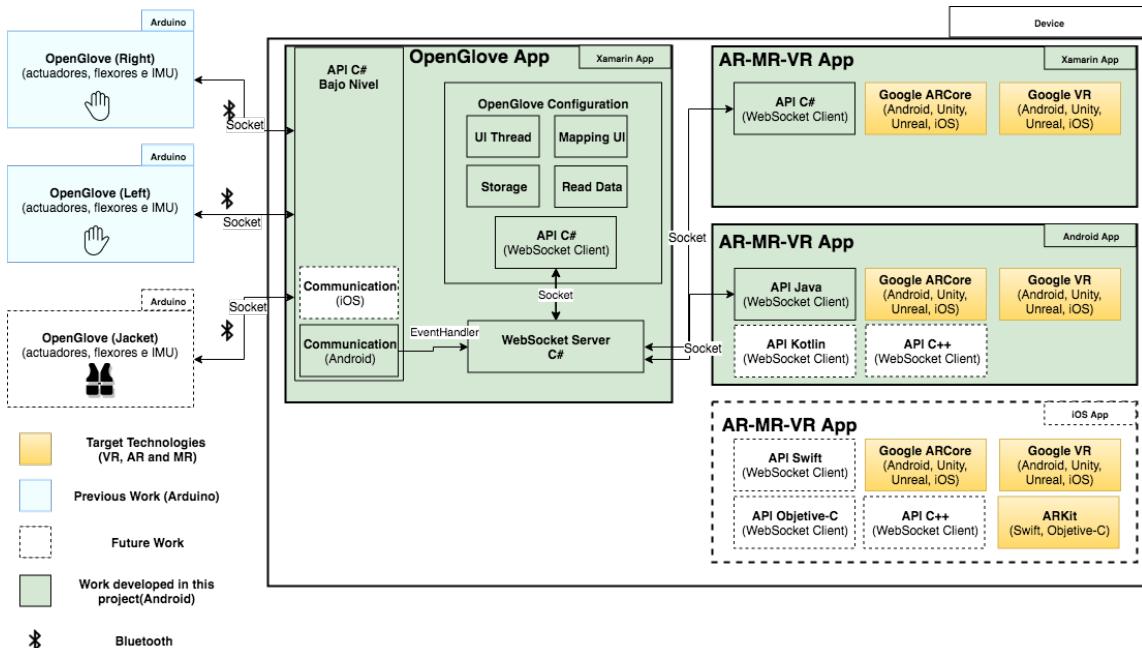


Figura 4.2: Detalle Arquitectura de OpenGlove

Fuente: Elaboración propia (2018)

La Figura 4.2 muestra un diagrama conceptual de la arquitectura en detalle, donde se consideró el objetivo de uso de las APIs de alto nivel en Java y C#. Las APIs de alto nivel mencionadas permiten el desarrollo de aplicaciones de terceros en Android nativo usando Java o con C# utilizando Xamarin. Para realizar aplicaciones de AR o VR, se puede utilizar Google ARCore y Google VR respectivamente. Ambas herramientas de Google están disponibles para Android, iOS, Unity y Unreal.

Esta arquitectura establece al servidor WebSocket como punto de acceso a las funcionalidades referentes a los actuadores, flexores e IMU. Por tanto la aplicación de configuración y las aplicaciones de terceros utilizan las mismas APIs de alto nivel para su desarrollo, evitando replicación de código y aprovechando todas las funcionalidades provistas por la API de alto nivel. La aplicación de configuración de OpenGlove desarrollada en Xamarin.Forms, utiliza la API de alto nivel C#.

Dado el alcance de la solución, no se ha desarrollado APIs de alto nivel que tengan como objetivos aplicaciones de terceros para iOS, para lo cual requiere un dispositivo iPhone real para realizar las pruebas de Bluetooth y WebSockets, o en su defecto un MacBook para poder visualizar

la interfaz de la aplicación, pero sin hacer uso de las funcionalidades Bluetooth ni WebSocket. También es importante recalcar la necesidad de implementar parte de la API de bajo nivel en C# ( Communication ) específica para iOS para gestionar las conexiones de los dispositivos Bluetooth y la administración de mensajes entre la API de bajo nivel y el servidor WebSocket. Este es un trabajo que se puede realizar a futuro para dar soporte a OpenGlove en iOS.

## 4.2 ESTRUCTURA

En esta sección se detalla la estructura de todos los componentes de software de OpenGlove, los cuales son: el software de control Arduino, la API de bajo nivel C#, las APIs de alto nivel y la aplicación de configuración.

### 4.2.1 Software de control Arduino

La estructura del software de control Arduino corresponde a la desarrollada por Cerda (2017), esta estructura se mantiene para este proyecto, solamente se agrega el soporte para obtener la versión actual del Software de configuración, siguiendo el versionamiento semántico propuesto por Preston-Werner (2018). A continuación se muestra la forma de aplicar este versionamiento en resumidas palabras:

“Dado un número de versión MAJOR.MINOR.PATCH incrementar

- MAJOR cuando se tienen cambios incompatibles en la API
- MINOR cuando se agrega funcionalidad compatible con anteriores versiones, y
- PATCH cuando se hacen correcciones de errores compatibles con versiones anteriores.

El uso de etiquetas adicionales para pre-lanzamientos y metadatos, son extensiones al formato MAJOR.MINOR.PATCH.” (Preston-Werner, 2018).

La versión 1.0.0 fue desarrollada en el trabajo de Monsalve (2015) permitiendo la activación de actuadores. Luego en el trabajo de Cerda (2017) se desarrolla la versión 1.1.0, porque se mantiene la compatibilidad luego de agregar la funcionalidad de lectura de datos de flexores e IMU en la placa Arduino. Por tanto la versión modificada en este proyecto corresponde a la versión 1.2.0. La Figura 4.3 muestra la estructura del software de control, donde se aprecia que OpenGloveArduino es el código principal, el cual incluye a FunctionsSwitch que contiene todas referencias a las funciones de la placa y a los componentes de hardware conectados a ella (actuadores, flexores

e IMU). La Tabla 4.1 contiene la descripción de cada clase del diagrama, su origen y el estado actual de desarrollo.

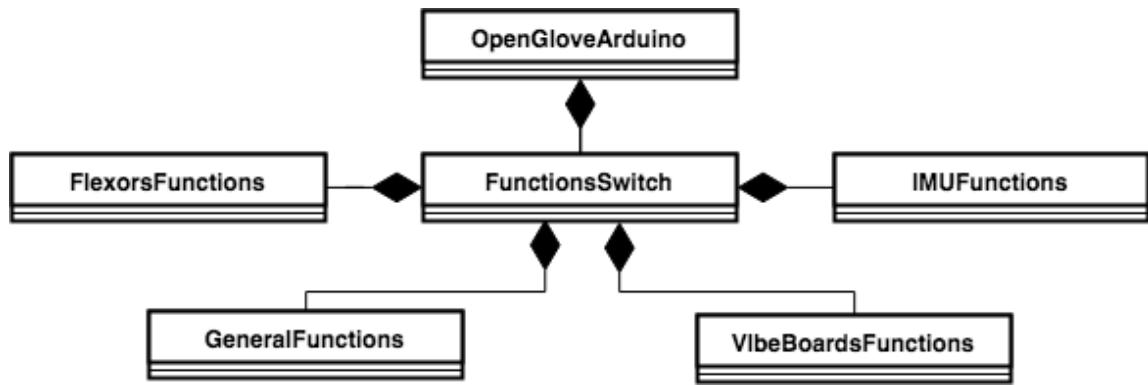


Figura 4.3: Estructura del software de control Arduino  
Fuente: Elaboración propia (2018)

Tabla 4.1: Descripción estructura del software de control Arduino  
 Fuente: Elaboración propia (2018)

Nombre	Descripción	Origen	Estado Actual
OpenGloveArduino	"Contiene el código principal de Arduino, desde el cual se realizan los llamados de las funciones disponibles en FunctionsSwitch. Algunas de éstas son invocadas solo cuando hay datos disponibles, mientras que las funciones que requieren un llamado constante como las dedicadas a la lectura de un sensor, son llamadas sin necesidad de datos entrantes" (Cerda, 2017).	Cerda (2017)	Modificado
FunctionsSwitch	"Se encarga de evaluar e invocar el tipo de función solicitada, con el fin de realizar cambios en la instancia de la biblioteca correspondiente a dicha función" (Cerda, 2017).	Cerda (2017)	Sin modificar
GeneralFunctions	Contiene las funciones generales de Arduino (lecturas/escrituras y digitales/análogas). Se destaca la función setLoopDelay() para definir el retraso de los ciclos de la placa en milisegundos. Por defecto es 60 ms, por tanto para obtener la mayor velocidad de la placa Arduino, se debe utilizar la función SetLoopDelay disponible en la API de alto nivel.	Monsalve (2015)	Sin modificar
FlexorsFunctions	"Contiene todas las variables y funciones necesarias para representar los flexores del guante y su configuración." (Cerda, 2017). Si se requiere aumentar la cantidad máxima de flexores por placa (10) debe realizarse a nivel del software de control Arduino.	Cerda (2017)	Sin modificar
VibeBoardFunctions	Contiene todas las funciones necesarias para inicializar, activar y desactivar los actuadores.	Monsalve (2015)	Sin modificar
ImuFunctions	"Contiene todas las variables y funciones necesarias para el funcionamiento y configuración del sensor de rastreo IMU." (Cerda, 2017).	Cerda (2017)	Sin modificar

#### 4.2.2 API C# de bajo nivel

Para establecer la comunicación entre la API C# de bajo nivel y el software de configuración Arduino, fue necesario rehacer la clase Communication, puesto que estaba pensada solamente para un ambiente de escritorio en Windows. Por ello se define una interfaz ICommunication, que define los métodos que deben ser implementados en las diferentes plataformas objetivo.

Los métodos de la interfaz son: la obtención de la lista de dispositivos emparejados, la conexión/desconexión de dispositivos y la escritura/lectura del socket Bluetooth creado. La Figura 4.4 muestra el diagrama de clases de la API C# de bajo nivel desarrollada. La Tabla 4.2 contiene la descripción de cada clase que compone el diagrama de clases, su origen y el estado actual de desarrollo.

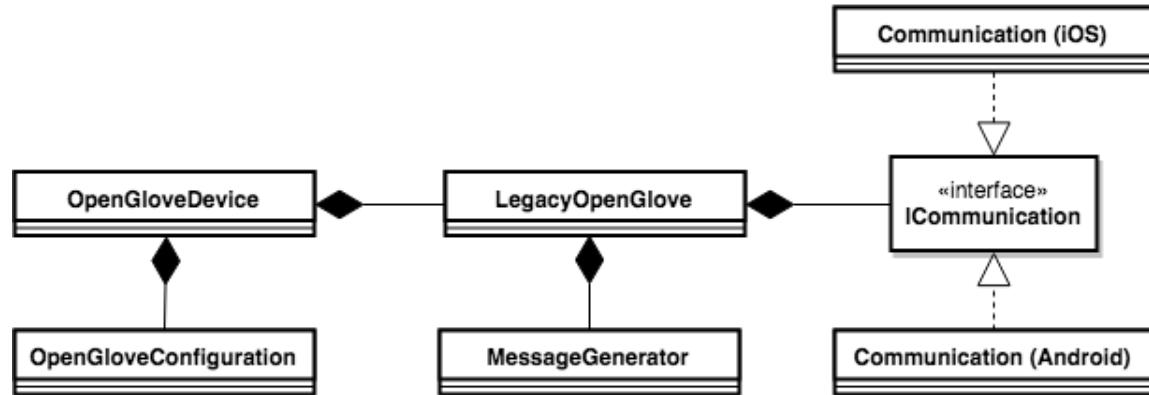


Figura 4.4: Diagrama de clases API C# de bajo nivel

Fuente: Elaboración propia (2018)

Tabla 4.2: Descripción del diagrama de clases API C# de bajo nivel  
 Fuente: Elaboración propia (2018)

Nombre	Descripción	Origen	Estado Actual
OpenGloveDevice	Clase que representa un dispositivo OpenGlove, la cual incluye su identificación, configuración y todos los métodos que permiten la comunicación con el software de control Arduino y la inicialización de la configuración en la placa Arduino.	Nuevo	Completado
OpenGloveConfiguration	Representa la configuración global de un dispositivo OpenGlove. Esta incluye la configuración de la placa Arduino, el mapeo de actuadores, el mapeo de flexores y la configuración del IMU. Dicha configuración global puede estar o no inicializada en el dispositivo físico.	Nuevo	Completado
LegacyOpenGlove	Clase que representa la API C# de bajo nivel desarrollada por Monsalve (2015) y Cerda (2017), modificada para obtener la versión del software de control Arduino.	Cerda (2017)	Modificado
MessageGenerator	Clase que permite la generación de mensajes bajo el protocolo establecido por Monsalve (2015) para los actuadores y la extensión realizada por Cerda (2017) para dar soporte a los flexores e IMU. Esta clase fue modificada para obtener la versión del software de control Arduino.	Cerda (2017)	Modificado
ICommunication	Interfaz que contiene la definición de métodos que deben ser implementados para obtener: la lista de dispositivos emparejados, la conexión/desconexión de los mismos y la escritura/lectura de mensajes. Esta interfaz debe ser implementada en cada sistema operativo que desee utilizar OpenGlove, para ello se debe consultar las plataformas soportadas por Xamarin.Forms, este proyecto incluye el soporte para Android.	Nuevo	Completado
Communication (Android)	Implementación de la interfaz ICommunication en el proyecto Android de Xamarin.Forms. Hace uso de las APIs Bluetooth de Android permitiendo obtener la lista de dispositivos emparejados, la conexión/desconexión de dispositivos y la escritura/lectura del socket Bluetooth creado. Esta clase crea un hilo que administra cada conexión Bluetooth de manera independiente, este hilo permite que otros objetos se suscriban para recibir los mensajes que envie el software de control Arduino.	Nuevo	Completado
Communication (iOS)	Implementación de la interfaz ICommunication en el proyecto iOS de Xamarin.Forms. Los métodos de esta clase arrojan la excepción NotImplementedException. Es necesario implementar los métodos de esta clase para dar soporte a OpenGlove en iOS, para ello debe utilizarse un dispositivo físico.	Nuevo	Incompleto, se requiere implementar los métodos en iOS

#### 4.2.3 APIs de alto nivel

Para permitir la interoperabilidad de OpenGlove, se deben soportar diferentes lenguajes de programación, que son utilizados en el desarrollo móvil de diferentes plataformas, como lo es iOS y Android, por tanto se desarrollaron las APIs C# y Java como se especifica en el alcance del proyecto. Estas APIs poseen un mismo objetivo, el cual consiste en abstraer la complejidad de conexiones y protocolos de comunicación necesarios, para tener una instancia OpenGlove que permita la interacción en tiempo real con el dispositivo físico. Esto se consigue utilizando el diseño presentado en la Figura 4.5 y 4.6. Se destaca la diferencia en la implementación de WebSocketClient en la clase Communication, debido a las diferencias en el lenguaje de programación y la implementación de las bibliotecas WebSocket utilizadas en cada API. Esto se debe a que en ambos casos, se busca que la clase Communication sea el acceso a los eventos relacionados con la obtención de mensajes desde el servidor (valores de los flexores, el IMU y el estado de conexión con el dispositivo Bluetooth). La Tabla 4.3 contiene la descripción de cada clase que compone el diagrama de clases.

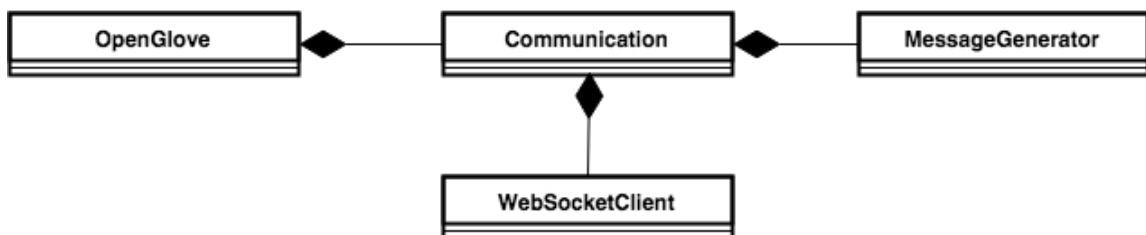


Figura 4.5: Diagrama de clases API C# de alto nivel  
Fuente: Elaboración propia (2018)

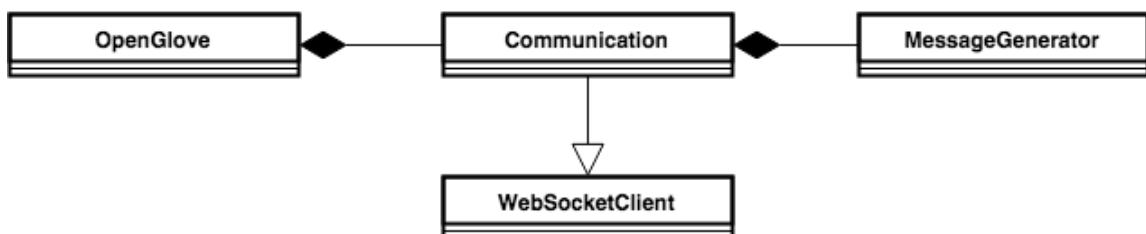


Figura 4.6: Diagrama de clases API Java de alto nivel  
Fuente: Elaboración propia (2018)

Tabla 4.3: Descripción del diagrama de clases APIs de alto nivel  
 Fuente: Elaboración propia (2018)

Nombre	Descripción
OpenGlove	Clase que representa un dispositivo OpenGlove, la cual incluye su identificación y todos los métodos que permiten la comunicación con el servidor WebSocket, la creación/asignación de la configuración en la instancia OpenGlove del servidor y la inicialización de la configuración en la placa Arduino.
MessageGenerator	Clase que permite la generación de mensajes bajo el protocolo creado por el memorista, para la comunicación Cliente WebSocket y Servidor WebSocket.
Communication	Clase que permite la comunicación con el Servidor WebSocket, permitiendo el envío/recepción de mensajes bajo el protocolo creado por el memorista. Se utiliza esta clase para la obtención de datos mediante la invocación de métodos, permitiendo crear soluciones en tiempo real suscribiéndose/implementando los siguientes eventos/interfaces: <ul style="list-style-type: none"> <li>- OnTimeTestServerLatencyActivateActuatorsReceived</li> <li>- OnTimeTestArduinoLatencyActivateActuatorsReceived</li> <li>- OnFlexorValueReceived</li> <li>- OnAccelerometerValuesReceived</li> <li>- OnGyroscopeValuesReceived</li> <li>- OnMagnetometerValuesReceived</li> <li>- OnAllIMUValuesReceived</li> <li>- OnBluetoothDeviceConnectionStateChanged</li> <li>- OnWebSocketConnectionStateChanged</li> <li>- OnInfoMessagesReceived</li> </ul>
WebSocketClient	Clase que permite utilizar la API WebSocket, la cual que implementa el protocolo RFC 6455. Se utilizan las bibliotecas disponibles para cada lenguaje de programación.

#### 4.2.4 Aplicación de configuración

La aplicación de configuración utiliza la API C# de alto nivel (*OpenGlove*), accediendo a todas las funcionalidades provistas en ella. La estructura de la aplicación se compone de una barra de navegación inferior, que contiene el acceso a las configuraciones como pantalla principal, los dispositivos emparejados y el servidor WebSocket. Esta estructura permite agregar nuevas creaciones de interfaces de usuario, para la configuración de los distintos componentes que pueda soportar OpenGlove en un futuro. La Figura 4.7 muestra el diagrama de clases de la aplicación de configuración desarrollada. La Tabla 4.4 contiene la descripción de cada clase que compone

el diagrama de clases de la aplicación de configuración de OpenGlove.

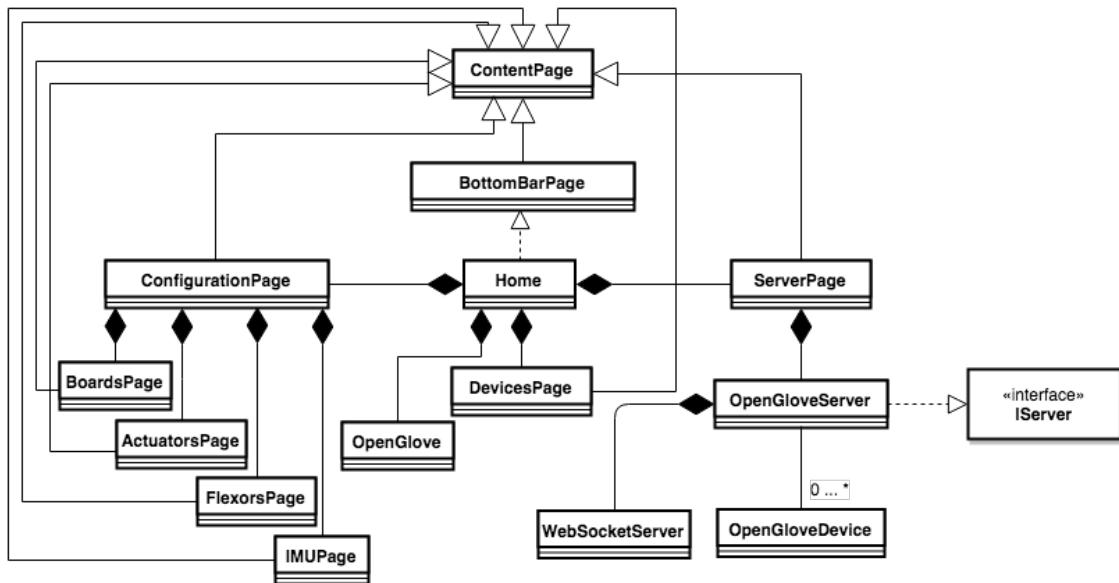


Figura 4.7: Diagrama de clases aplicación de configuración  
Fuente: Elaboración propia (2018)

Tabla 4.4: Descripción del diagrama de clases aplicación de configuración  
Fuente: Elaboración propia (2018)

Nombre	Descripción
ContentPage	Xamarin.Forms.ContentPage es una clase y un elemento visual que hereda de la clase base Page en Xamarin.Forms. Otras clases derivadas de Page son MasterDetailPage, NavigationPage, TabbedPage, TemplatedPage y CarrouselPage.
BottomBarPage	BottomBar.XamarinForms.BottomBarPage es una clase que implementa la representación visual de la barra inferior en iOS y Android. Permite agregar Pages para representarlos como botones inferiores de navegación.
Home	Clase que hereda de BottomBarPage (biblioteca de terceros). Esta clase se instancia como primer Page del NavigationPage, esto permite la navegación jerárquica en la aplicación. La clase NavigationPage implementa la navegación como una lista LIFO (last-in, first-out) de objetos Page.

ConfigurationPage	Clase que contiene el menú de acceso a las diferentes interfaces gráficas que permiten la actualización de los actuadores, los flexores e IMU. Permite la creación, lectura, actualización y eliminación (CRUD) de configuraciones globales de OpenGlove.
BoardsPage	Clase que provee la interfaz gráfica para modificar la configuración de la placa Arduino, es decir, la configuración de los pines de la placa, cantidad, tipo de señal (análoga/digital), tipo de componente asociado (VibeBoards, Flexores) y polaridad del pin (positiva/negativa). Permite la creación de nuevos modelos de placas, especificando: nombre de la placa, cantidad de pines digitales, cantidad de pines análogos y el primer pin análogo.
ActuatorsPage	Clase que provee la interfaz gráfica para modificar la configuración de los actuadores, es decir, el mapeo región-actuador, el rango de regiones disponible es dinámica según se requiera, permitiendo cambiar la imagen referencial del mapeo utilizar. Además permite testear los actuadores, especificando la intensidad.
FlexorPage	Clase que provee la interfaz gráfica para modificar la configuración de los flexores, es decir, el mapeo region-flexor, el rango de regiones disponible es dinámica según se requiera, esto permite cambiar la imagen referencial del mapeo a utilizar. Además permite testear los flexores, mostrando el valor leído desde la placa Arduino con el Threshold permite asignado.
IMUPage	Clase que provee la interfaz gráfica para modificar la configuración del IMU (SparkFun LSM9DS1). Además permite testear el IMU, mostrando los valores leídos desde la placa Arduino según se especifique.
DevicePage	Clase que muestra todos los dispositivos emparejados, permitiendo asignar la configuración global OpenGlove que se quiera inicializar en la placa Arduino.
ServerPage	Clase que provee la interfaz gráfica para modificar la configuración del servidor, permitiendo la modificación de la dirección del servidor WebSocket, además del encendido y apagado del mismo.
IServer	Interfaz que define los métodos Start, Stop y ConfigureServer.

OpenGloveServer	Clase que provee el acceso a todas las funcionalidades de la aplicación de configuración. Posee todas las instancias de OpenGloveDevice, de los clientes WebSocket conectados y recibe los mensajes de los hilos que administran las conexiones con los dispositivos Bluetooth.
WebSocketServer	Clase que permite utilizar la API WebSocket, la cual implementa el protocolo RFC 6455. Se utiliza la biblioteca Fleck.
OpenGlove	Clase correspondiente a la API C# de alto nivel de OpenGlove, la cual ya fue descrita en la Tabla 4.2.
OpenGloveDevice	Clase correspondiente a la API C# de bajo nivel de OpenGlove, la cual ya fue descrita en la Tabla 4.3.

### 4.3 COMPORTAMIENTO

Para comprender el comportamiento de las APIs de alto nivel, es necesario complementar lo detallado sobre la clase OpenGlove en la Subsección 4.2.3, en la cual se habló sobre la estructura de las mismas. Cada instancia de la clase OpenGlove de alto nivel, permite ejecutar acciones sobre su cliente WebSocket, el servidor WebSocket al que se conecta, a las configuraciones de su instancia de la clase OpenGloveDevice alojada en el servidor y realizar acciones sobre el software de control Arduino. Además, cada instancia provee un fácil consumo de los datos provenientes de Arduino. La suma de esto es lo que hace posible una representación de los dispositivos OpenGlove en las APIs de alto nivel en tiempo real, incluyendo el envío de mensajes de manera bidireccional y full-duplex para los lenguajes de programación C# y Java. Con esto se facilita que otras APIs de alto nivel pueden ser desarrolladas, utilizando lenguajes de programación que permitan: el manejo de eventos o funciones lambda o el envío de mensajes entre hilos, además del uso de Clientes WebSocket (API WebSocket y protocolo RFC 6455). De esta forma, se reduce la necesidad de agregar lógica compleja en la API de alto nivel.

A continuación se describe el comportamiento general de los métodos disponibles en las APIs de alto nivel, utilizando diagramas de actividades. Se agruparon estos métodos según su nivel de interacción con las diferentes capas de abstracción del SDK. Las capas de abstracción corresponden a: la API de alto nivel (Cliente WebSocket), el Servidor WebSocket, las instancias de la clase OpenGloveDevice, las instancias de LegacyOpenGlove (API C# de bajo nivel) y el software de control Arduino.

### 4.3.1 Métodos aplicados sobre cliente WebSocket

Los métodos que pueden ser aplicados sobre el cliente WebSocket se muestran en el Código ???. Estos métodos son ejecutados por la aplicación que utiliza una instancia de OpenGlove, realizando la acción sobre el cliente WebSocket. Esto se detalla en el diagrama de actividades de la Figura 4.8. Adicionalmente, en el caso que se logre la conexión con el servidor WebSocket, el evento OnOpen del cliente es invocado. Cuando se invoca ese evento, se agrega la instancia OpenGlove al servidor y se inicia la captura de datos. Eso se realizó utilizando los métodos AddOpenGloveDeviceToServer y StartCaptureDataFromServer respectivamente. El comportamiento de estos dos métodos mencionados se incluyen en la Subsección 4.3.2.

---

**Código 4.1:** Métodos aplicados sobre el cliente WebSocket

Fuente: Elaboración propia (2018)

---

```
1 // OpenGloveActions: OpenGlove High level API (WebSocketClient)
2 public void ConnectToWebSocketServer();
3 public void DisconnectFromWebSocketServer();
```

---

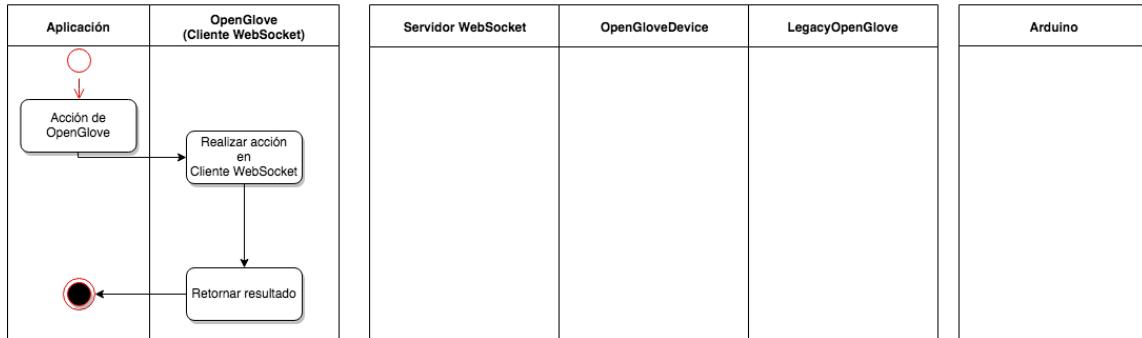


Figura 4.8: Diagrama de actividades de los métodos aplicados sobre el cliente WebSocket  
Fuente: Elaboración propia (2018)

### 4.3.2 Métodos aplicados sobre servidor WebSocket

Los métodos que pueden ser aplicados sobre el servidor WebSocket se muestran en el Código 4.2. Estos métodos son ejecutados por la aplicación que utiliza una instancia de OpenGlove, realizando la acción sobre el servidor WebSocket, pasando por la capa intermedia del cliente WebSocket. El comportamiento de estos métodos se aprecia en el diagrama de actividades de la Figura 4.9.

---

**Código 4.2:** Métodos aplicados sobre el servidor WebSocket

Fuente: Elaboración propia (2018)

```
1 // OpenGloveActions : WebSocket Server
2 public void AddOpenGloveDeviceToServer();
3 public void RemoveOpenGloveDeviceFromServer();
4 public void SaveOpenGloveConfiguration();
5 public void StartCaptureDataFromServer();
6 public void StopCaptureDataFromServer();
```

---

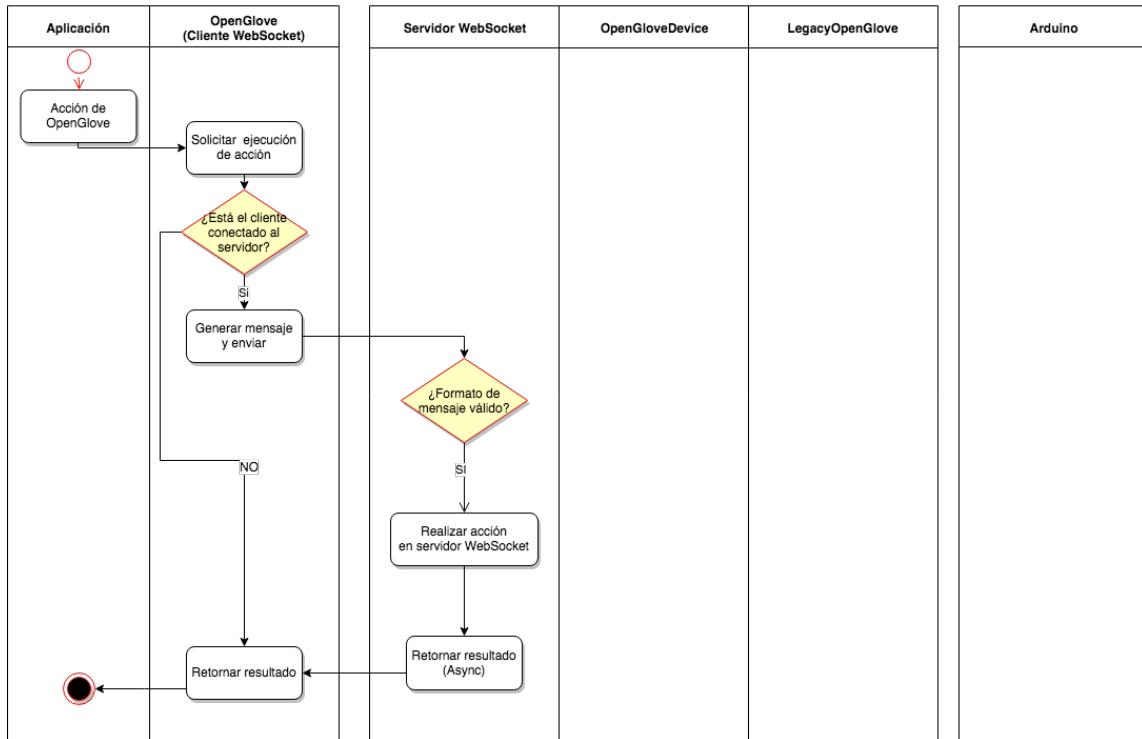


Figura 4.9: Diagrama de actividad de los métodos aplicados sobre el servidor WebSocket

Fuente: Elaboración propia (2018)

#### 4.3.3 Métodos aplicados sobre instancias de OpenGloveDevice

Los métodos que pueden ser aplicados sobre la instancia de la clase OpenGloveDevice en el servidor WebSocket se muestran en el Código 4.3. Estos métodos son ejecutados por la aplicación que utiliza una instancia de OpenGlove, realizando la acción sobre la instancia de OpenGloveDevice en el servidor WebSocket, pasando por las capas intermedias del cliente y el servidor WebSocket. El comportamiento de estos métodos se puede apreciar en el diagrama

de actividades de la Figura 4.10.

---

**Código 4.3: Métodos aplicados sobre instancias de OpenGloveDevice**

Fuente: Elaboración propia (2018)

---

```
1 // OpenGloveActions: OpenGlove Instances
2 public void ConnectToBluetoothDevice();
3 public void DisconnectFromBluetoothDevice();
4 public void AddActuator(int region, int positivePin, int negativePin);
5 public void AddActuators(List<int> regions, List<int> positivePins, List<int>
   negativePins);
6 public void AddFlexor(int region, int pin);
7 public void AddFlexors(List<int> regions, List<int> pins);
8 public void SetThreshold(int value);
9 public void SetIMUStatus(bool status);
10 public void SetRawData(bool status);
11 public void SetIMUCHoosingData(int value);
12 public void SetLoopDelay(int value);
```

---

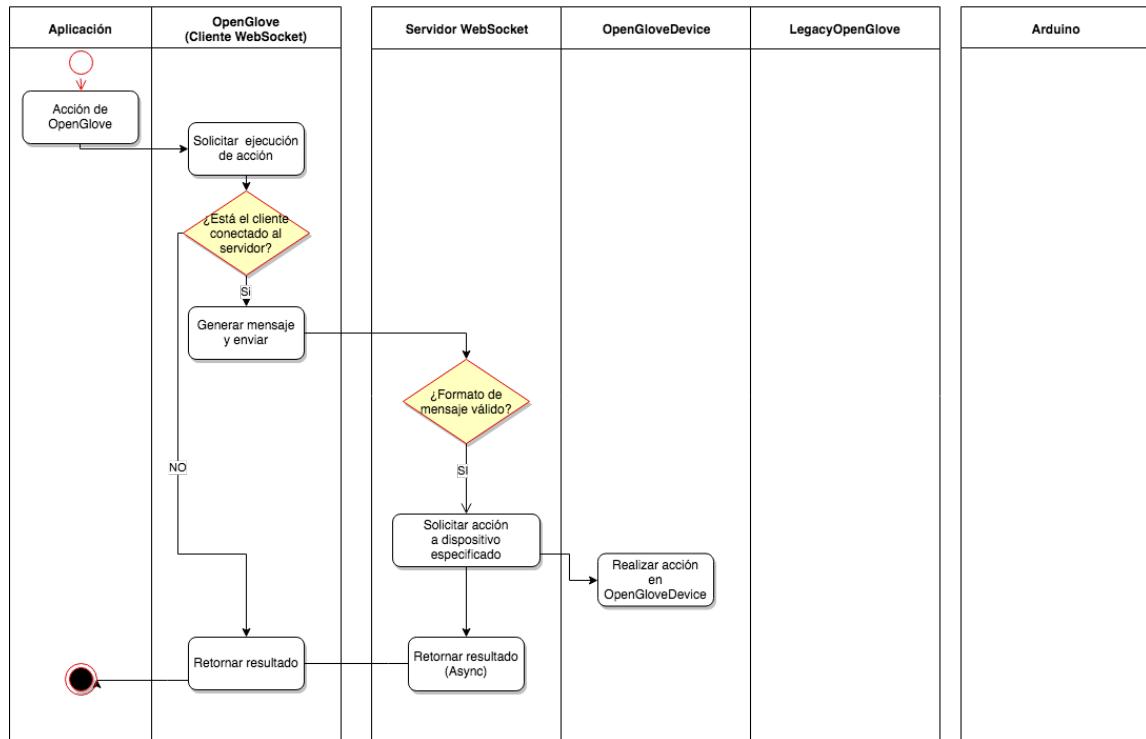


Figura 4.10: Diagrama de actividades de las acciones OpenGlove sobre instancias de OpenGloveDevice en servidor  
Fuente: Elaboración propia (2018)

#### 4.3.4 Métodos aplicados sobre el software de control Arduino

Los métodos que pueden ser aplicados sobre el software de control Arduino se muestran en el Código 4.4. Estos métodos son ejecutados por la aplicación que utiliza una instancia de OpenGlove, realizando la acción sobre el software de control Arduino. Estos métodos generan cambios en el dispositivo Arduino. Para realizar la acción se pasa por las capas intermedias del cliente, el servidor WebSocket y la instancia de OpenGloveDevice que administra la conexión con Arduino. El comportamiento de estos métodos se puede apreciar en el diagrama de actividades de la Figura 4.11.

---

**Código 4.4:** Métodos aplicados sobre el software de control Arduino

Fuente: Elaboración propia (2018)

---

```
1 // OpenGloveActions: Software of Control Arduino
2 public void Start();
3 public void Stop();
4 public void RemoveActuator(int region);
5 public void RemoveActuators(List<int> regions);
6 public void ActivateActuators(List<int> regions, List<string> intensities);
7 public void ActivateActuatorsTimeTest(List<int> regions, List<string>
     intensities);
8 public void TurnOnActuators();
9 public void TurnOffActuators();
10 public void TurnOnFlexors();
11 public void TurnOffFlexors();
12 public void ResetActuators();
13 public void RemoveFlexor(int region);
14 public void RemoveFlexors(List<int> regions);
15 public void CalibrateFlexors();
16 public void ConfirmCalibration();
17 public void ResetFlexors();
18 public void StartIMU();
19 public void ReadOnlyAccelerometerFromIMU();
20 public void ReadOnlyGyroscopeFromIMU();
21 public void ReadOnlyMagnetometerFromIMU();
22 public void ReadOnlyAttitudeFromIMU();
23 public void ReadAllDataFromIMU();
24 public void TurnOnIMU();
25 public void TurnOffIMU();
26 public void GetOpenGloveArduinoVersionSoftware();
```

---

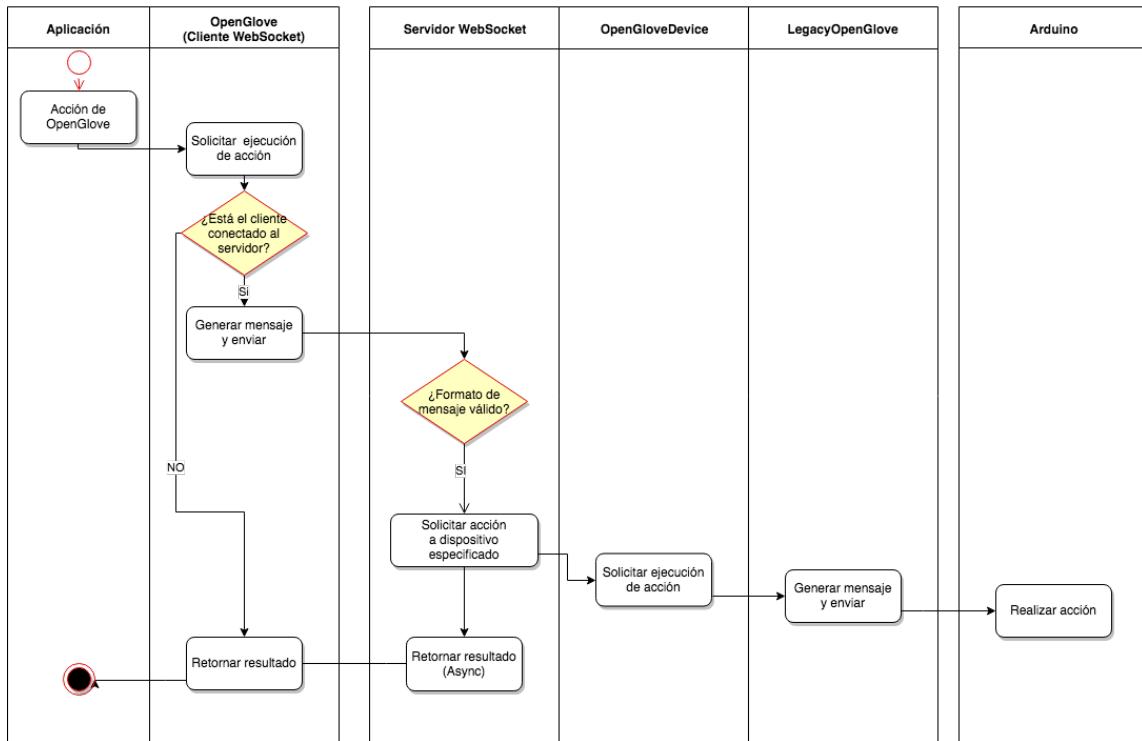


Figura 4.11: Diagrama de actividades de los métodos aplicados sobre el software de control Arduino

Fuente: Elaboración propia (2018)

#### 4.3.5 Escuchando mensajes provenientes de Arduino

Este comportamiento ocurre cuando se ha inicializado por lo menos un flexor y/o se ha inicializado e iniciado la IMU. El código 4.5, muestra el formato de los métodos que deben ser implementados por los objetos que utilicen instancias de la clase OpenGlove. El comportamiento de estos métodos se aprecia en el Diagrama de actividades de la Figura 4.12

---

**Código 4.5:** Escuchando mensajes provenientes de Arduino

Fuente: Elaboración propia (2018)

```
1 // OpenGloveActions: Listening messages from Arduino Control Software
2 // For use in API C#: subscribe to event with your own method
3 // For use in API Java: implements public interface with lambdas or method
   references
4 public void OnTimeTestServerLatencyActivateActuatorsReceived(long nanoSeconds);
5 public void OnTimeTestArduinoLatencyActivateActuatorsReceived(long
   microSeconds);
6 public void OnFlexorValueReceived(int region, int value);
7 public void OnAccelerometerValuesReceived(float ax, float ay, float az);
8 public void OnGyroscopeValuesReceived(float gx, float gy, float gz);
9 public void OnMagnetometerValuesReceived(float mx, float my, float mz);
10 public void OnAttitudeValuesReceived(float pitch, float roll, float yaw);
11 public void OnAllIMUValuesReceived(float ax, float ay, float az, float gx,
   float gy, float gz, float mx, float my, float mz);
12 public void OnInfoMessagesReceived(string message);
```

---

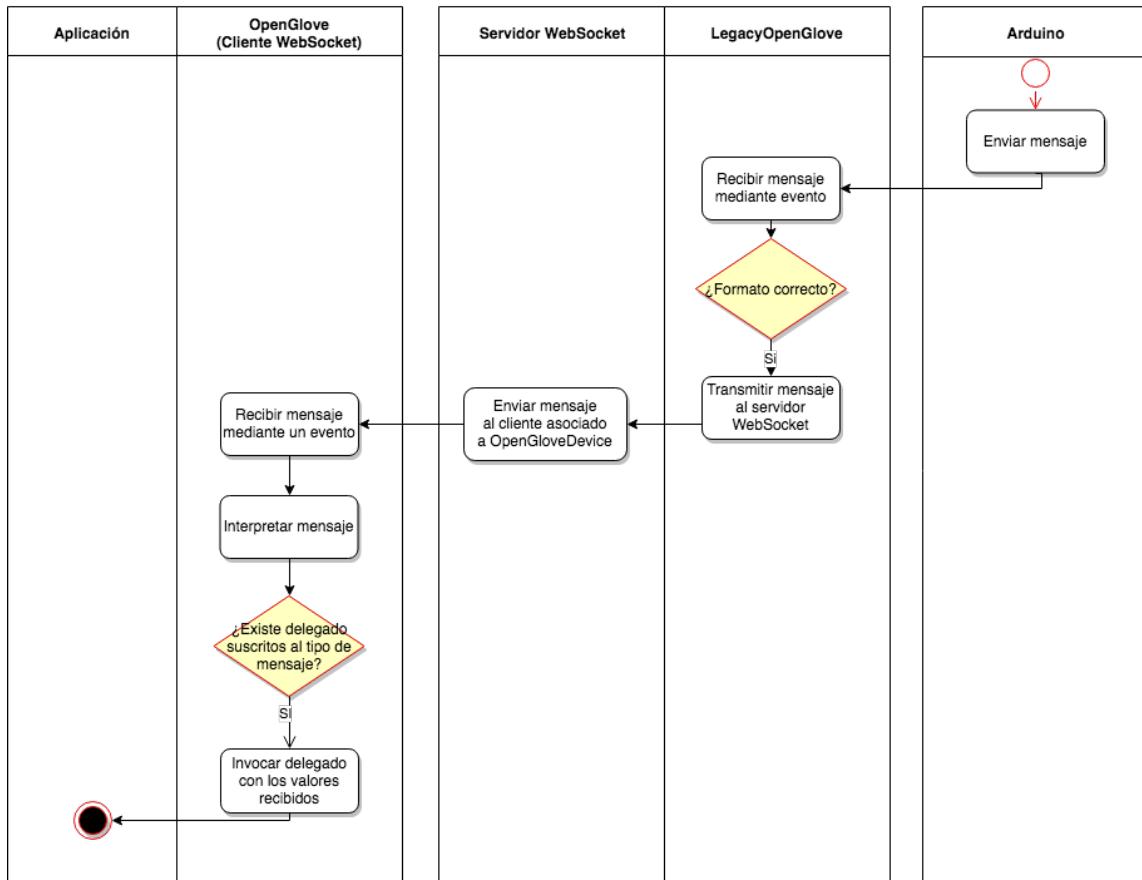


Figura 4.12: Diagrama de actividades cuando se escuchan mensajes provenientes de Arduino

Fuente: Elaboración propia (2018)

#### 4.3.6 Escuchando el estado de conexión del dispositivo Bluetooth

Para actualizar el estado de la conexión Bluetooth en las APIs de alto nivel, el hilo que administra dicha conexión, debe enviar actualizaciones del estado de conexión o desconexión con los dispositivos. Cuando el estado del socket está conectado se envía un mensaje “*b, True*” y cuando no, se envía “*b, False*”. La Figura 4.6, muestra el método que se debe implementar para obtener los cambios de estado de la conexión Bluetooth. El comportamiento de activación de estos métodos se puede apreciar en el diagrama de actividades de la Figura 4.13.

---

**Código 4.6:** Escuchando el estado de conexión del dispositivo Bluetooth

Fuente: Elaboración propia (2018)

---

```
1 // OpenGloveActions: Listening Bluetooth device connection state messages
2 // For use in API C#: subscribe to event with your own method
3 // For use in API Java: implements public interface with lambdas or method
   references
4 public void OnBluetoothDeviceConnectionStateChanged(bool isConnected);
```

---

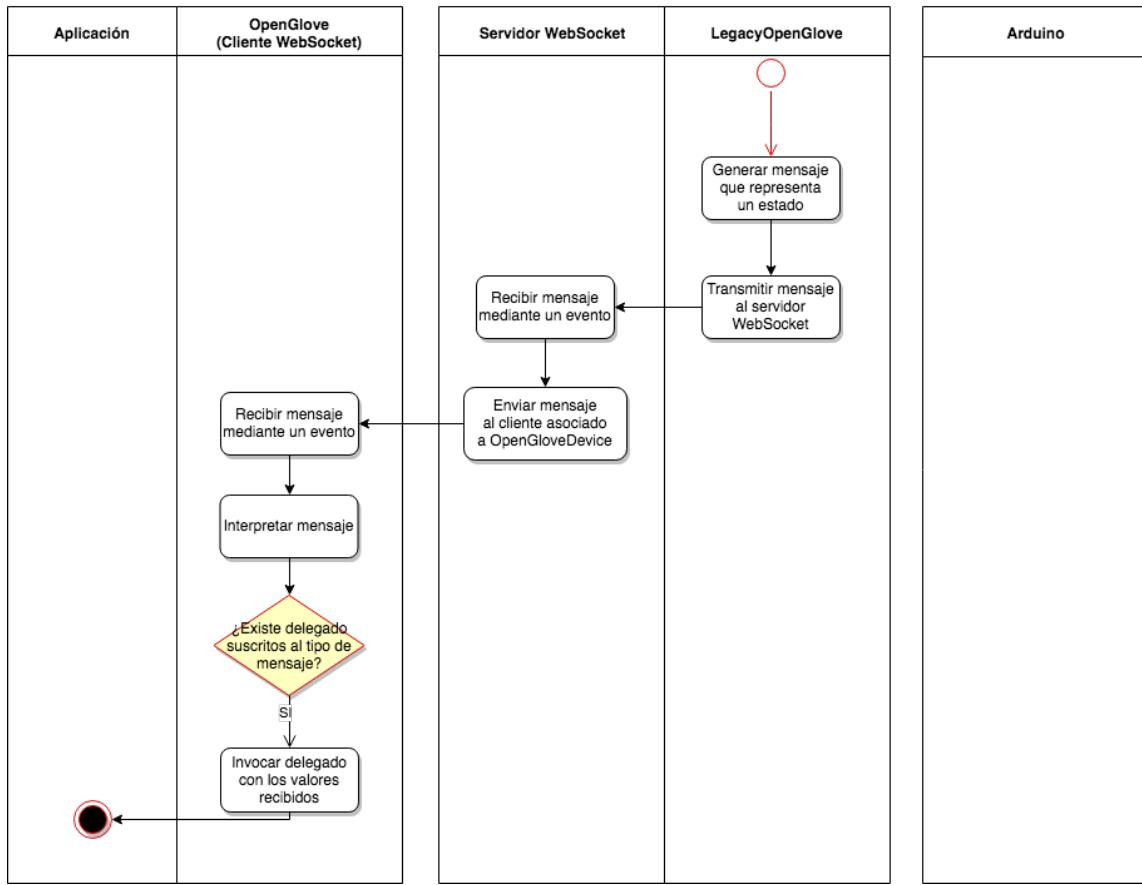


Figura 4.13: Diagrama de actividades cuando se escucha el estado de la conexión del dispositivo Bluetooth

Fuente: Elaboración propia (2018)

#### 4.3.7 Escuchando el estado de conexión del cliente WebSocket

Para notificar el estado de la conexión del cliente WebSocket en las APIs de alto nivel, se invoca el método `OnWebSocketConnectionStateChanged`, cada vez que ocurre un cambio utilizando `OnOpen` y `OnClose` de la API `WebSocket`. La Figura 4.7, muestra el método que debe ser implementado para escuchar el estado de conexión del cliente WebSocket. El comportamiento de este método se puede apreciar en el diagrama de actividades de la Figura 4.14.

---

**Código 4.7:** Escuchando el estado de conexión del cliente WebSocket  
Fuente: Elaboración propia (2018)

---

```
1 // OpenGloveActions: Listening WebSocket client connection state messages
2 // For use in API C#: subscribe to event with your own method
3 // For use in API Java: implements public interface with lambdas or method
   references
4 public void OnWebSocketConnectionStateChanged(bool isConnected);
```

---

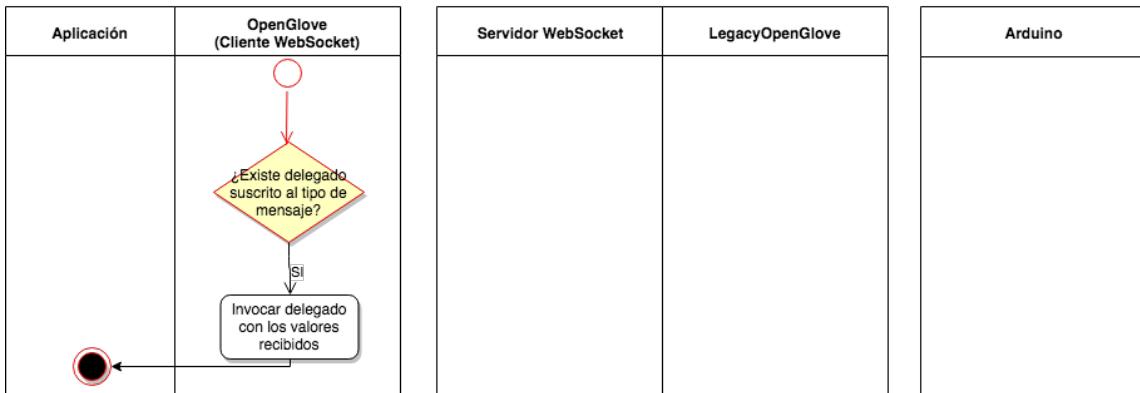


Figura 4.14: Diagrama de actividades cuando se escucha la conexión del dispositivo Bluetooth  
Fuente: Elaboración propia (2018)

## 4.4 ASPECTOS DE IMPLEMENTACIÓN

### 4.4.1 Desarrollo multiplataforma en Xamarin.Forms

De los resultados de la comparación en la Sección 5.1, entre desarrollo nativo en Android con Android Studio y Android con Xamarin.Forms, se obtuvo que los rendimientos no eran muy diferentes entre si, por tanto, se optó por el uso de Xamarin.Forms, ya que provee soporte multiplataforma y permite que el proyecto pueda ser utilizado en otras plataformas en un futuro, utilizando las APIs y bibliotecas nativas que permitan comunicación tanto Bluetooth como WebSocket.

#### **4.4.2 API C# bajo nivel**

En la Sección 4.2.2, se detalla sobre la estructura de OpenGloveDevice que corresponde a la API de bajo nivel C#. Respecto a la implementación, se debe mencionar el uso de *DependencyService*<sup>1</sup> de Xamarin.Forms, el cual permite realizar diferentes implementaciones de una interfaz referenciada en el proyecto principal, a los demás proyectos asociados (iOS, Android, UWP, etc.). Para ello se requiere la declaración de una interfaz en el código compartido (proyecto principal), segundo, realizar una implementación de la interfaz en cada proyecto por plataforma, cada implementación debe ser registrada en *DependencyService* para que esta sea cargada en tiempo de ejecución. Finalmente en el código compartido, se debe llamar explícitamente a *DependencyService* para hacer uso de ella. El Código 4.8 muestra esta implementación, utilizando como ejemplo el método Write, que permite la escritura en el socket serial Bluetooth.

#### **4.4.3 Servidor WebSocket**

#### **4.4.4 Software de configuración**

#### **4.4.5 APIs**

##### **4.4.5.1 Java**

##### **4.4.5.2 C#**

### **4.5 RESUMEN**

---

<sup>1</sup>DependencyService: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/dependency-service/introduction>

---

**Código 4.8:** Implementación ICommunication en diferentes plataformas

Fuente: Elaboración propia (2018)

---

```
1 // On OpenGloveApp main project
2 public interface ICommunication
3 {
4     List<BluetoothDevices> GetAllPairedDevices();
5     void OpenDeviceConnection(string bluetoothDeviceName);
6     void CloseDeviceConnection();
7     void Write(string message);
8     string ReadLine();
9 }
10
11 // On OpenGloveApp.Android project
12 [assembly: Xamarin.Forms.Dependency(typeof(Communication))]
13 namespace OpenGloveApp.Droid.Bluetooth
14 {
15     public class Communication : ICommunication
16     {
17         // ... More code ...
18         public void Write(string message)
19         {
20             if (mBluetoothManagementThread != null)
21                 if (mBluetoothManagementThread.IsAlive)
22                     mBluetoothManagementThread.Write(message);
23         }
24         // ... More code ...
25
26     // On OpenGloveApp.iOS project
27     [assembly: Xamarin.Forms.Dependency(typeof(Communication))]
28     namespace OpenGloveApp.iOS.Bluetooth
29     {
30         public class Communication : ICommunication
31         {
32             // ... More code ...
33             public void Write(string message)
34             {
35                 throw new NotImplementedException();
36             }
37             // ... More code ...
38
39     // Call Dependency Service on Shared Code, and use the methods
40     public class LegacyOpenGlove
41     {
42         public ICommunication communication =
43             DependencyService.Get<ICommunication>();
44         // ... More code ...
45         public void ActivateMotor(IEnumerable<int> pins, IEnumerable<string>
46             values)
47         {
48             string message = messageGenerator.ActivateMotor(pins, values);
49             communication.Write(message);
50         }
51         // ... More code ...
52     }
53 }
```

## CAPÍTULO 5. EVALUACIÓN TÉCNICA Y PRUEBAS DE CONCEPTO

En este capítulo se detallan las evaluaciones técnicas realizadas en el proyecto. Primero se muestran los resultados de las evaluaciones de tiempos de activación de motores y tiempos de lectura de flexores de la aplicación nativa (tercer prototipo) y multiplataforma (cuarto prototipo), con los cuales se decide el uso de Xamarin.Forms. En segundo lugar se muestran y analizan los resultados de los tiempos de activación de los motores para las APIs desarrolladas en C# y Java. En tercer lugar se muestran y analizan los resultados obtenidos de los tiempo de lectura de datos desde los flexores e IMU juntos para las APIs antes mencionadas. En Cuarto lugar se muestran las pruebas de concepto utilizando las APIs desarrolladas en ambientes de realidad VR, AR y MR. Finalmente se realiza un resumen sobre los resultados obtenidos en todas las evaluaciones técnicas del proyecto.

### 5.1 EVALUACIÓN APPLICACIONES NATIVA Y MULTIPLATAFORMA

A continuación se detalla la evaluación de los prototipos 3 y 4 ya expuestos en la Sección 3.2, los cuales poseen las mismas funcionalidades pero desarrollados con distintas herramientas, el primero utilizando el IDE Android Studio para desarrollar con el SDK nativo de android (desde ahora Droid) y el segundo prototipo utilizando el IDE VisualStudio Community en un proyecto Xamarin.Forms (desde ahora Xamarin). Ambos prototipos fueron modificados para realizar la siguiente evaluación que consta de 1,000 muestras y su almacenamiento en la memoria interna para la cantidad de motores y flexores del hardware disponible. Las pruebas fueron realizadas en el dispositivo Samsung Galaxy S5 mini y Nexus 5 ya señalados en el Capítulo 1. Para el análisis de latencia de los motores, se consideró su tiempo de activación y desactivación con una precisión de nanosegundos ( $ns$ ), transformando los resultados a microsegundos ( $\mu s$ ). Por tanto el umbral límite de 60 milisegundos ( $ms$ ) es equivalente a  $60,000 \mu s$  ó  $60,000,000 ns$ .

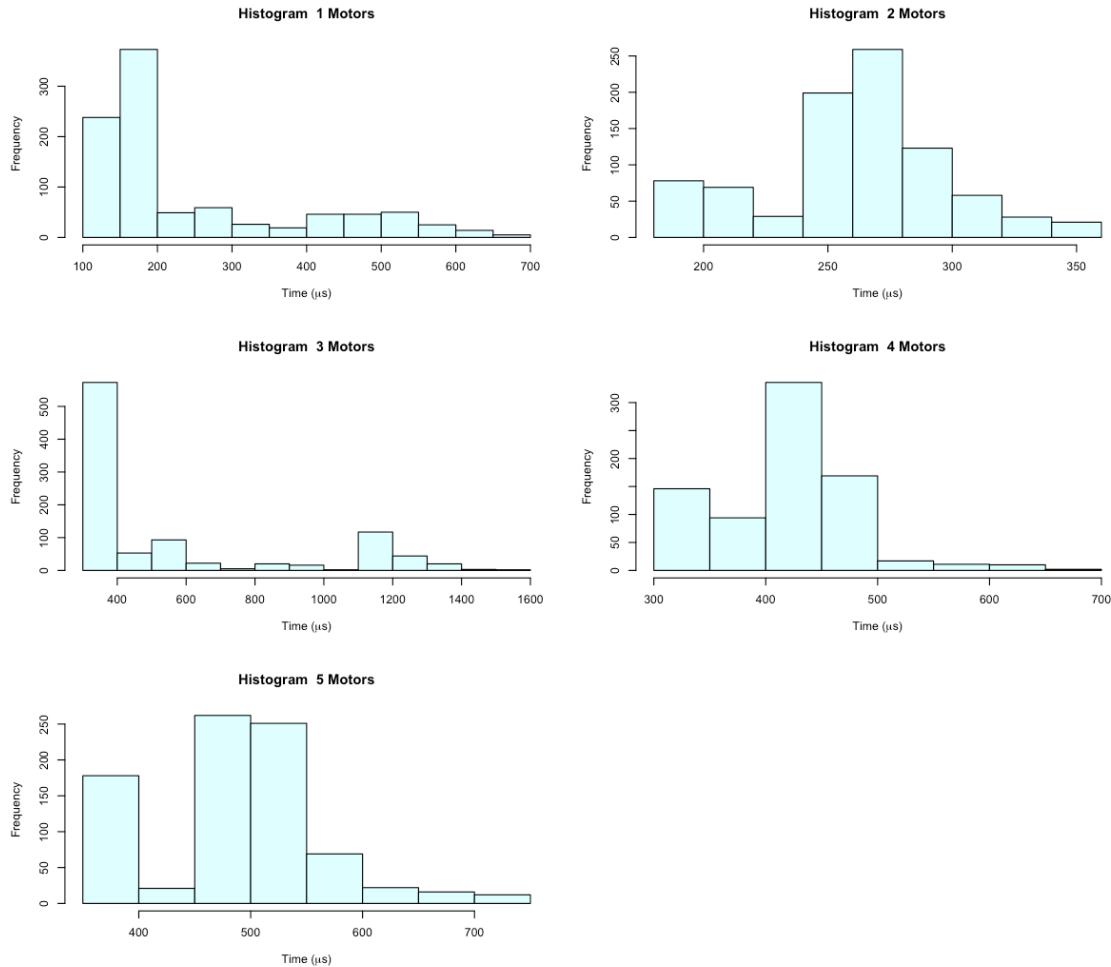
### **5.1.1 Prototipo 3 : Droid - Galaxy**

#### **5.1.1.1 Motores**

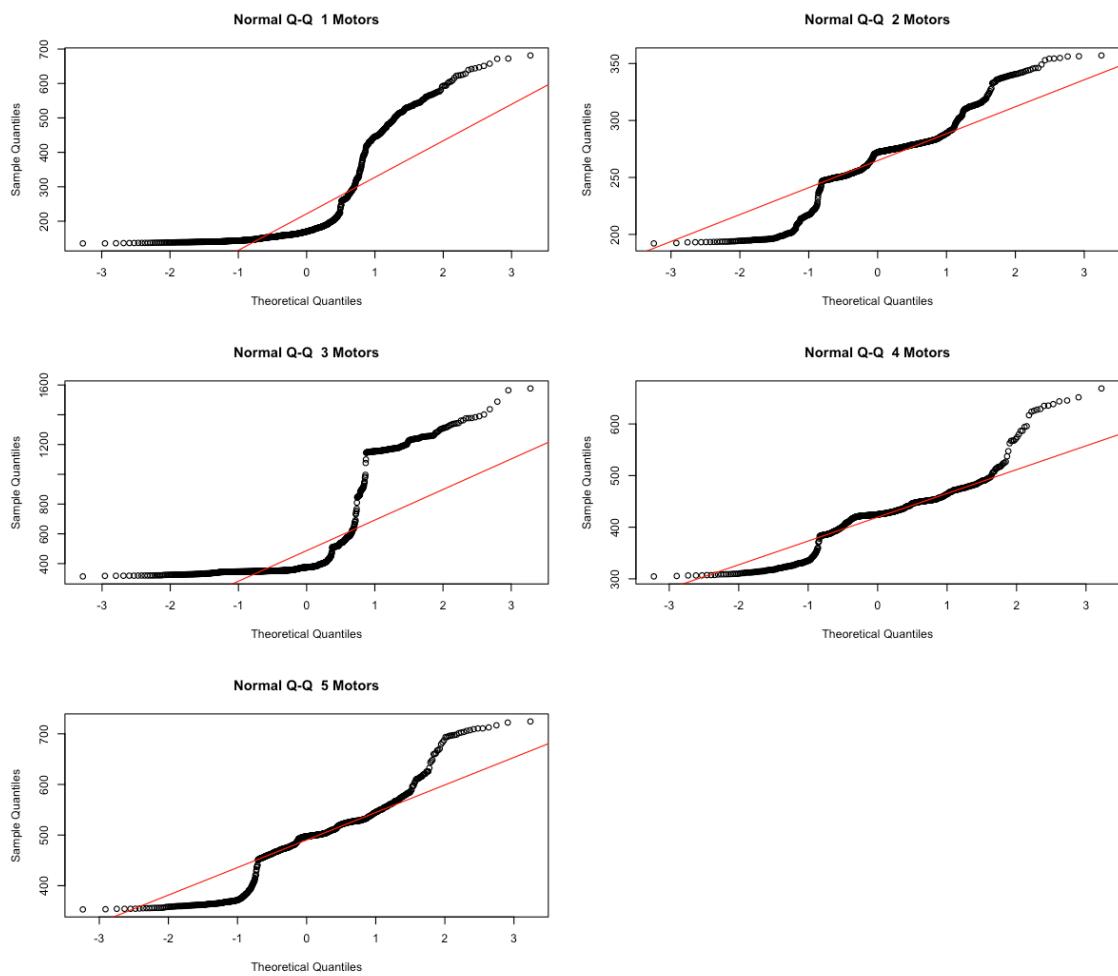
Se realizaron pruebas desde uno a cinco motores, el resumen de los resultados para las pruebas de activación de los motores se encuentra en la Tabla 5.1. La Figura 5.1, muestra los histogramas de las latencias obtenidas al mandar los mensajes de activación y desactivación, siendo la gráfica para dos motores la única con una asimetría (Skewness) negativa, lo que indica que la cola de la distribución de los datos está orientada a la izquierda, por tanto la media de la distribución se encuentra desplazada a la derecha. Para las demás pruebas realizadas se obtuvieron asimetrías positivas, lo que indica que la cola de la distribución se encuentra a la derecha, por tanto su media se encuentra desplazada a la izquierda. Esto se ve reflejado en los gráficos de cajas de la Figura 5.3. Las gráficas de Q-Q (quantile-quantile) mostradas en la Figura 5.2, demuestran que la curva no sigue estrictamente una distribución normal, pero los gráfico Q-Q de la misma figura, para dos, cuatro y cinco motores muestra resultados cercanos a la normal exceptuando los valores atípicos, los cuales pueden ser apreciados también en sus respectivas gráficas de cajas. Finalmente la Curtosis (Kurtosis) de todas las pruebas fue positiva, lo que indica que las colas de estas distribuciones concentran una cantidad mayor de datos en las colas, que en una distribución normal.

Tabla 5.1: Resumen de los resultados de los tiempos de activación de motor Droid-Galaxy  
Fuente: Elaboración propia (2018)

Motors	Mean $\mu s$	Median $\mu s$	Min $\mu s$	Max $\mu s$	Std. Dev. $\mu s$	Skewness	Kurtosis
1	245.345	170.156	135.469	681.354	141.369	1.336	3.376
2	263.026	272.031	192.084	357.083	36.334	-0.114	2.965
3	570.168	375.026	313.802	1,576.875	337.726	1.257	2.911
4	418.957	424.843	304.688	668.750	61.357	0.417	4.613
5	484.390	497.240	353.073	724.479	78.401	0.221	3.303



**Figura 5.1: Histogramas de los tiempos de activación de motores Droid-Galaxy**  
**Fuente:** elaboración propia (2018)



**Figura 5.2: Gráficos QQ de los tiempos de activación de motores Droid-Galaxy**  
**Fuente:** elaboración propia (2018)

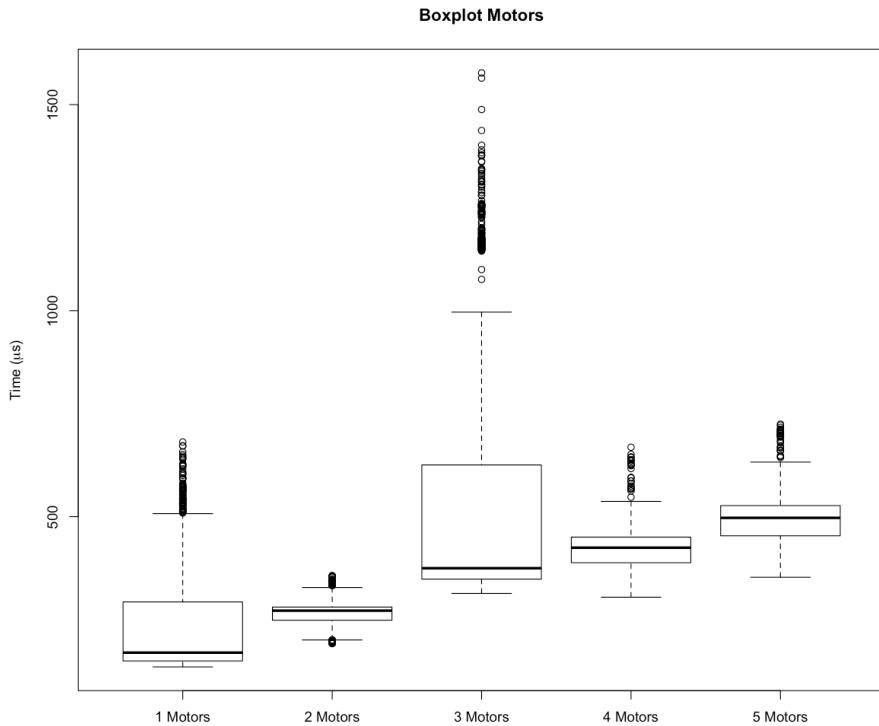


Figura 5.3: Gráficos de cajas de los tiempos de activación de motores Droid-Galaxy  
Fuente: elaboración propia (2018)

#### 5.1.1.2 Flexores

En el caso de los flexores, se realizó solamente una prueba para la obtención de datos desde la placa Arduino, con el único flexor físico disponible. Se utilizó una API de bajo nivel en Java desarrollada, en base a la de C# de bajo nivel. El resumen de los resultados de la prueba se muestra en la Tabla 5.2. De la tabla se puede observar que la muestra presenta una asimetría positiva, lo que implica que la cola de la distribución se encuentra a la derecha, por tanto, la media se encuentra desplazada hacia la izquierda, su Curtosis al ser positiva indica que posee mayor concentración de datos en las colas que una distribución normal. Esto puede apreciarse en el histograma de la Figura 5.4 en conjunto con el diagrama de cajas de la Figura 5.6. En el gráfico Q-Q de la Figura 5.5, se puede apreciar que la curva no sigue una distribución normal de manera estricta, sino que presenta acercamientos a ella, en forma sinusoidal.

Tabla 5.2: Resumen de los resultados de los tiempos de lectura de flexores Droid-Galaxy en  $\mu s$   
 Fuente: Elaboración propia (2018)

Motors	Mean $\mu s$	Median $\mu s$	Min $\mu s$	Max $\mu s$	Std. Dev. $\mu s$	Skewness	Kurtosis
1	121,879.900	108,551.400	34,473.860	252,435.200	38,318.940	0.658	2.637

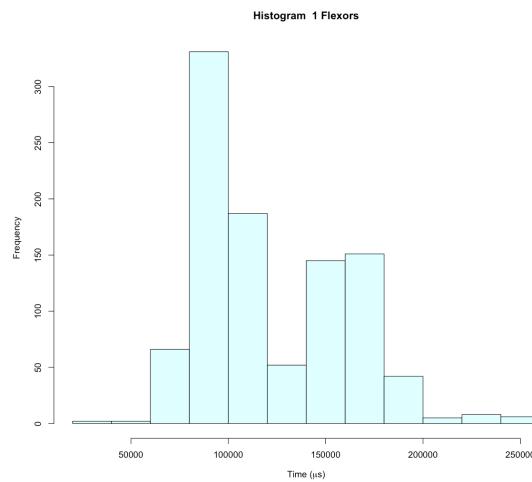


Figura 5.4: Histogramas de los tiempos de lectura de flexores Droid-Galaxy  
 Fuente: elaboración propia (2018)

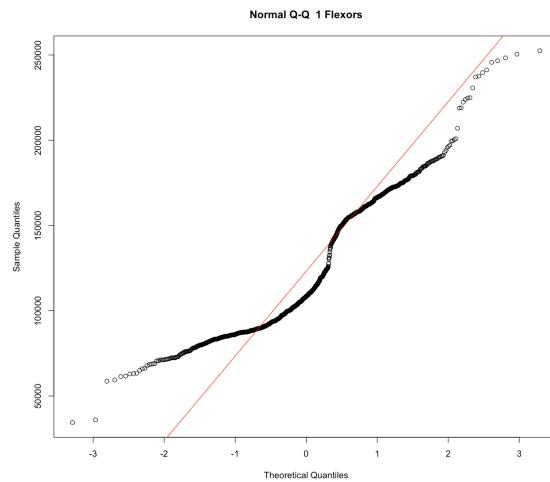


Figura 5.5: Gráficos QQ de los tiempos de lectura de flexores Droid-Galaxy  
 Fuente: elaboración propia (2018)

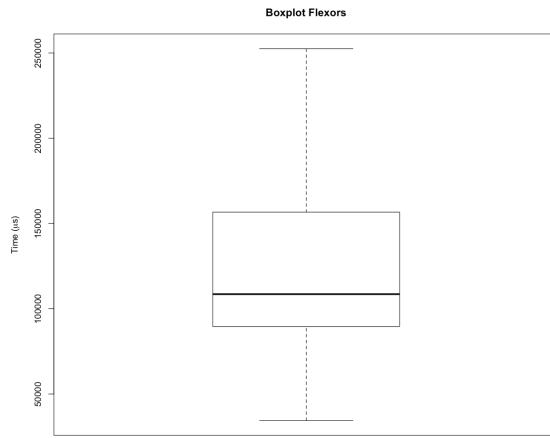


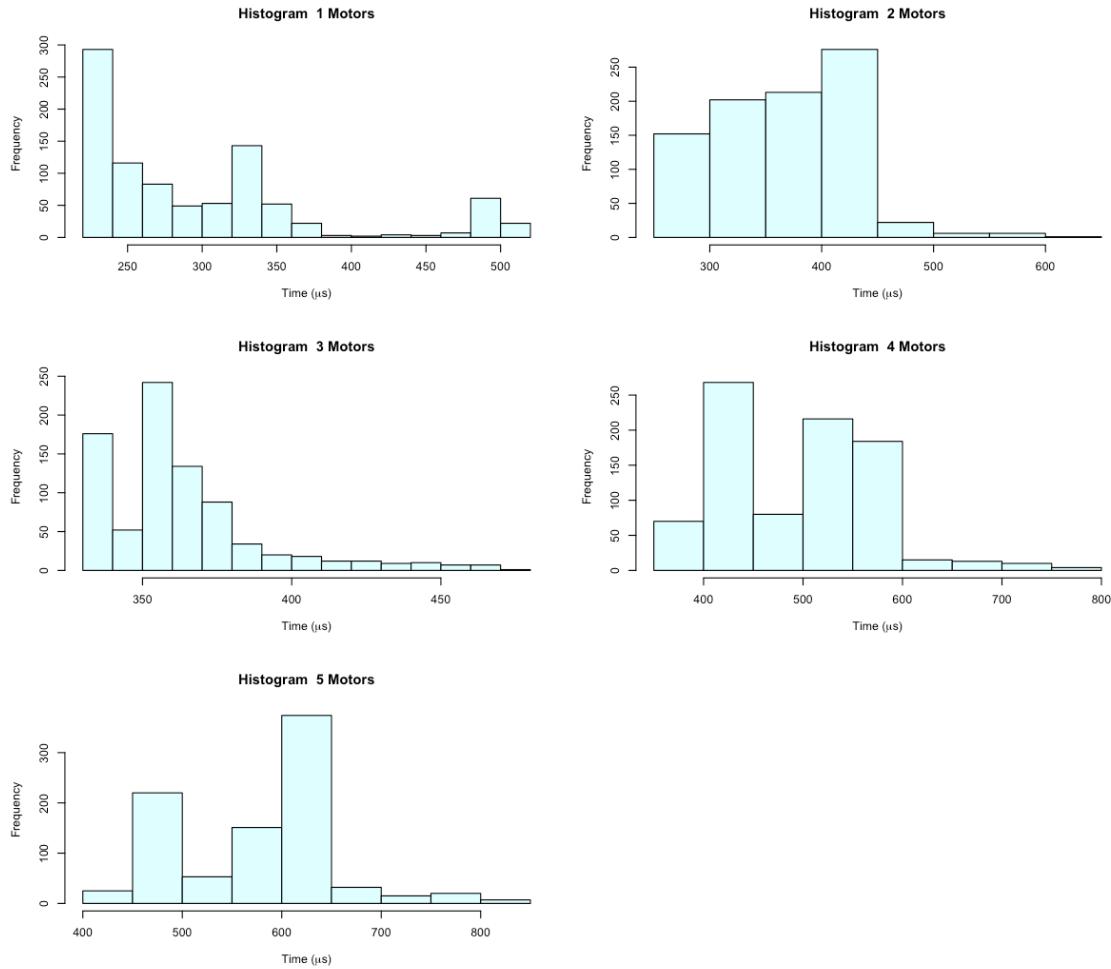
Figura 5.6: Gráficos de cajas de los tiempos de lectura de flexores Droid-Galaxy  
 Fuente: elaboración propia (2018)

### 5.1.2 Prototipo 4: Xamarin - Galaxy

#### 5.1.2.1 Motores

Tabla 5.3: Resumen de los resultado de los tiempos de activación de motores Xamarin-Galaxy  
 Fuente: Elaboración propia (2018)

Motors	Mean μs	Median μs	Min μs	Max μs	Std. Dev. μs	Skewness	Kurtosis
1	299.835	268.800	231.500	512	77.214	1.376	4.065
2	367.823	387.100	282	600.800	59.888	0.290	2.832
3	363.318	358.200	333	472.100	26.389	1.707	6.343
4	491.245	506.050	380.800	785.600	78.420	0.503	3.042
5	574.120	600	429.200	832.200	84.596	0.045	2.703



**Figura 5.7: Histogramas de los tiempos de activación de motores Xamarin-Galaxy**  
**Fuente:** elaboración propia (2018)

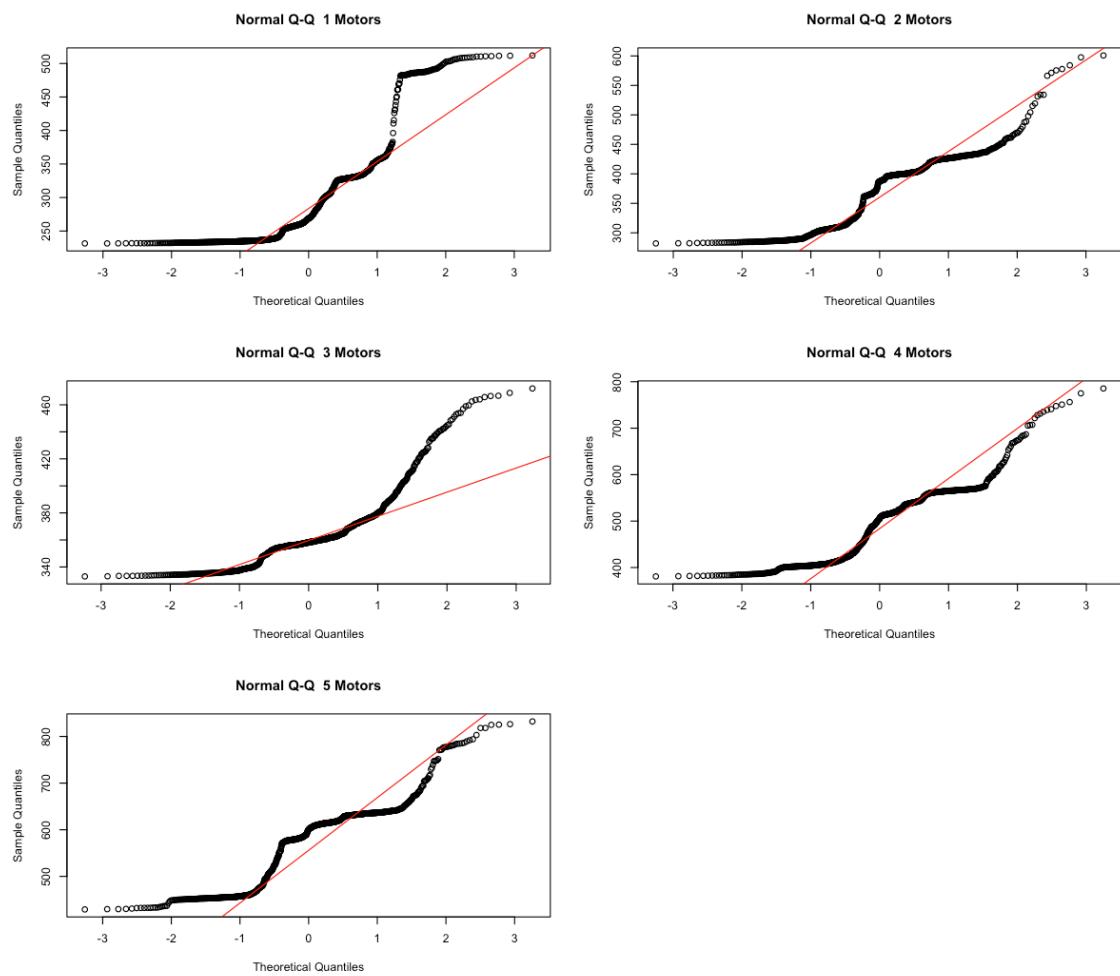


Figura 5.8: Gráficos QQ de los tiempos de activación de motores Xamarin-Galaxy  
Fuente: elaboración propia (2018)

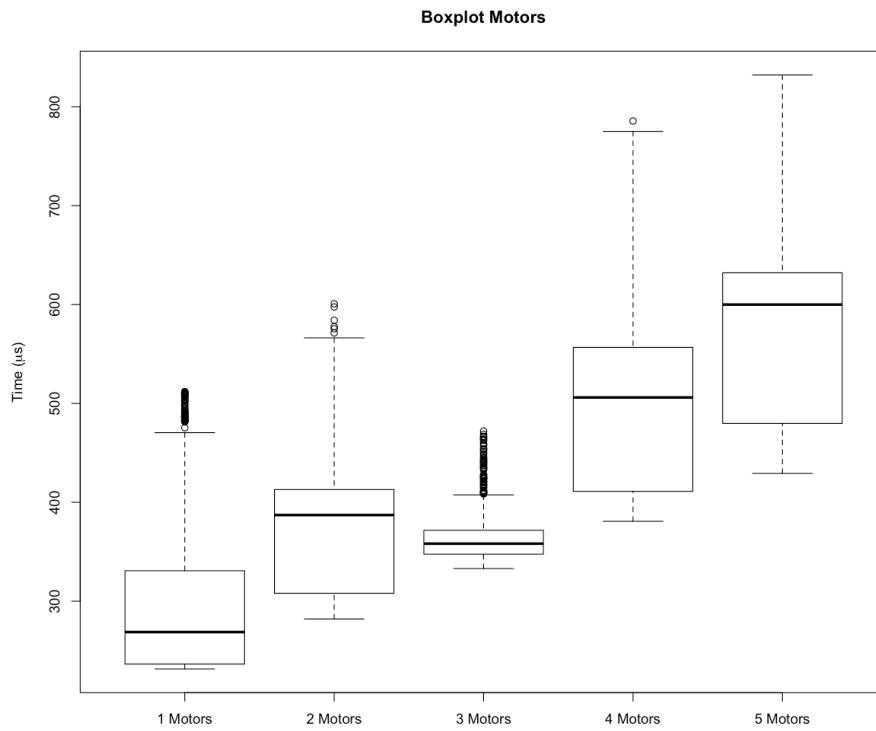


Figura 5.9: Gráficos de cajas de los tiempos de activación de motores Xamarin-Galaxy  
 Fuente: elaboración propia (2018)

### 5.1.2.2 Flexores

Tabla 5.4: Resumen de los resultados de los tiempos de lectura de flexores Xamarin-Galaxy  
 Fuente: Elaboración propia (2018)

Motors	Mean μs	Median μs	Min μs	Max μs	Std. Dev. μs	Skewness	Kurtosis
1	123,483.700	109,237.600	55,364.200	253,747.900	39,493.110	0.696	2.824

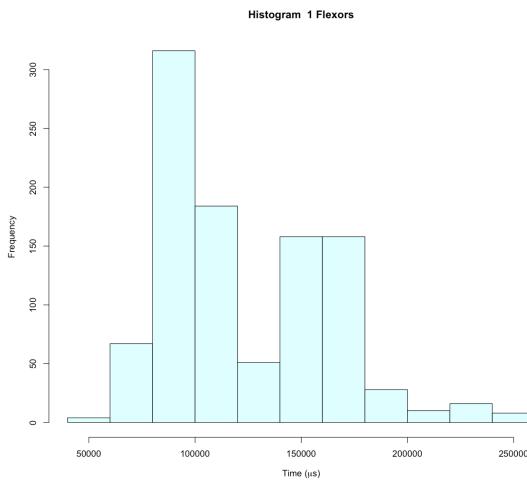


Figura 5.10: Histogramas de los tiempos de lectura de flexores Xamarin-Galaxy  
Fuente: elaboración propia (2018)

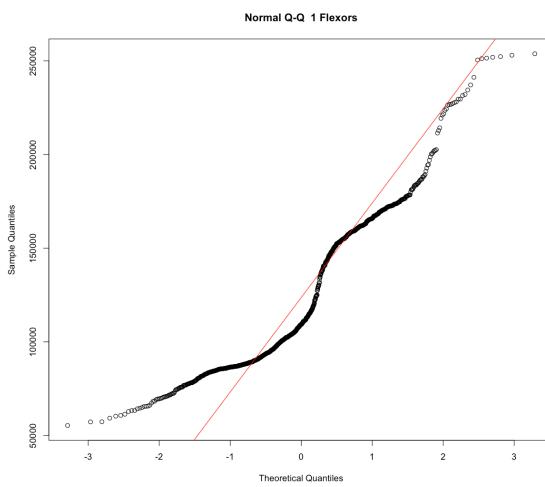


Figura 5.11: Gráficos QQ de los tiempos de lectura de flexores Xamarin-Galaxy  
Fuente: elaboración propia (2018)

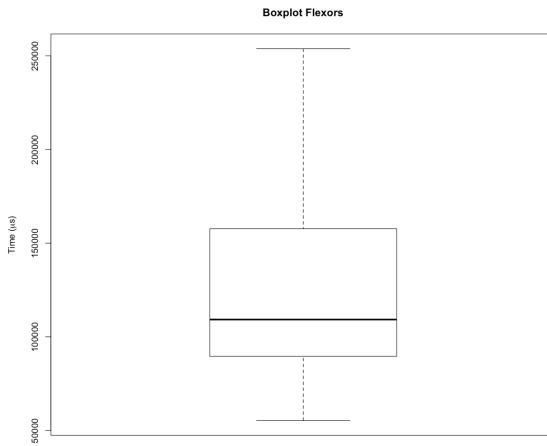


Figura 5.12: Gráficos de cajas de los tiempos de lectura de flexores Xamarin-Galaxy  
Fuente: elaboración propia (2018)

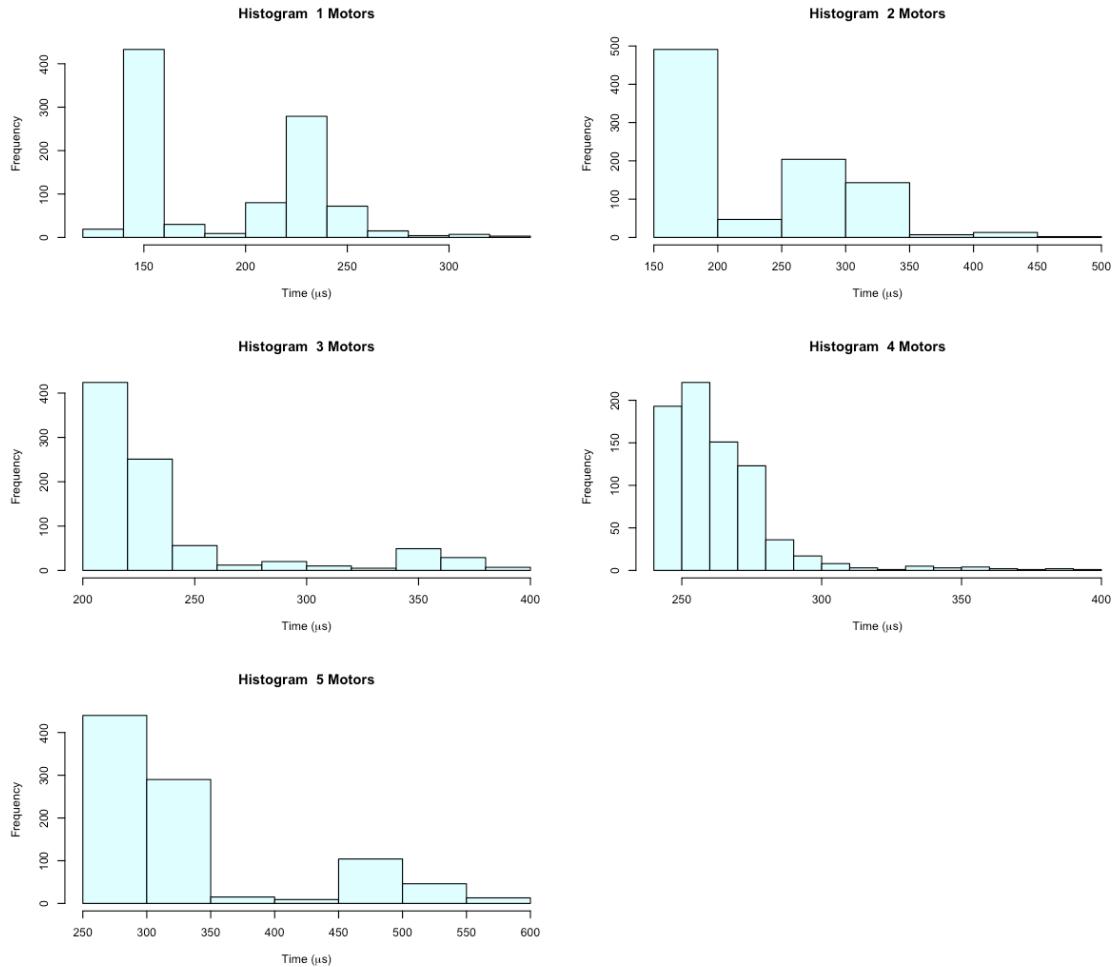
### 5.1.3 Prototipo 3 : Droid - Nexus

#### 5.1.3.1 Motores

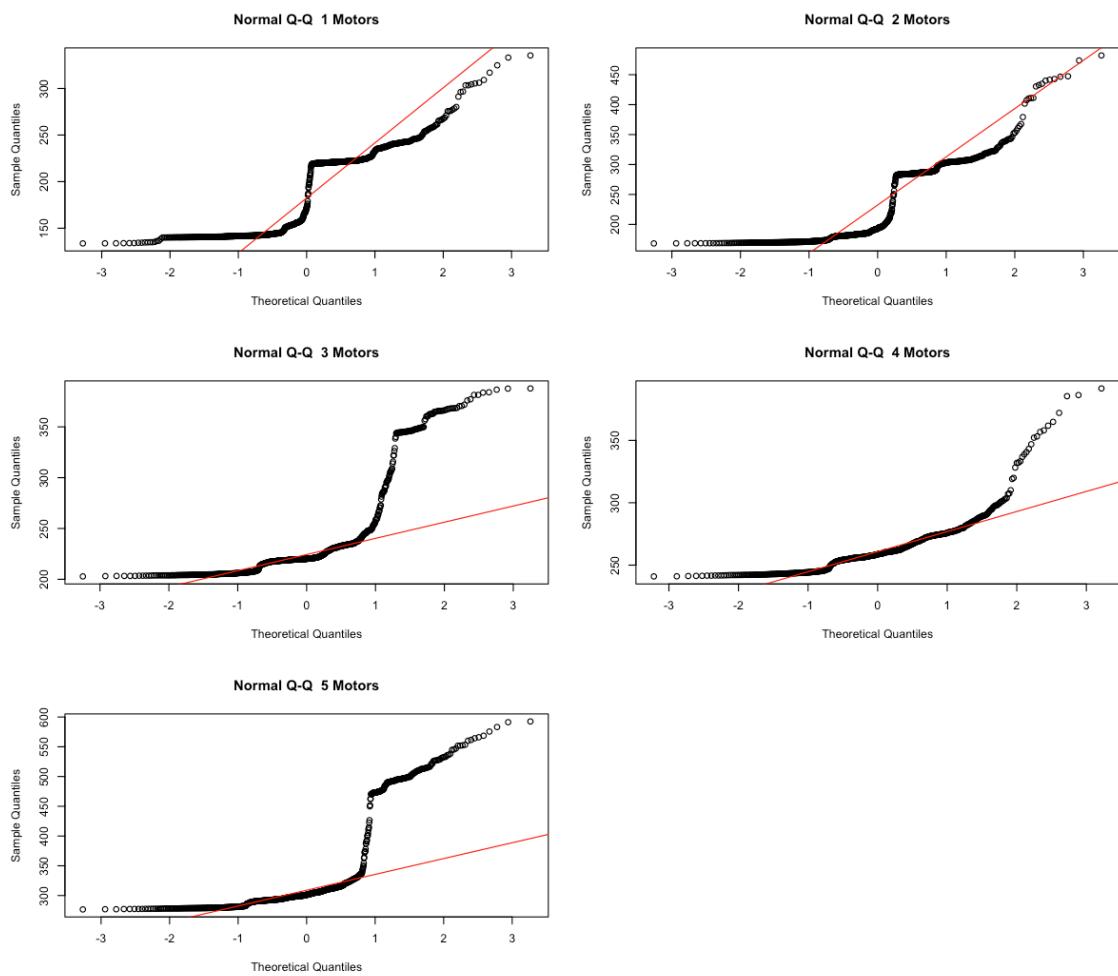
Tabla 5.5: Resumen de los resultados de los tiempos de activación de motores Droid-Nexus  
Fuente: Elaboración propia (2018)

motors	Mean $\mu s$	Median $\mu s$	Min $\mu s$	Max $\mu s$	Std. Dev. $\mu s$	Skewness	Kurtosis
1	187.574	171.979	133.646	335.416	44.577	0.337	1.905
2	231.251	192.761	168.230	482.136	63.706	0.773	2.835
3	237.638	220.105	202.812	387.813	44.517	1.965	5.687
4	262.838	258.646	240.938	391.615	20.442	2.567	13.078
5	336.917	301.250	276.667	592.553	79.910	1.566	3.872

La Figura 5.13, muestra los histogramas de las latencias obtenidas al mandar los mensajes de activación y desactivación. Se realizaron pruebas desde uno a cinco motores.



**Figura 5.13: Histogramas de los tiempos de activación de motores Droid-Nexus**  
**Fuente:** elaboración propia (2018)



**Figura 5.14: Gráficos QQ de los tiempos de activación de motores Droid-Nexus**  
**Fuente:** elaboración propia (2018)

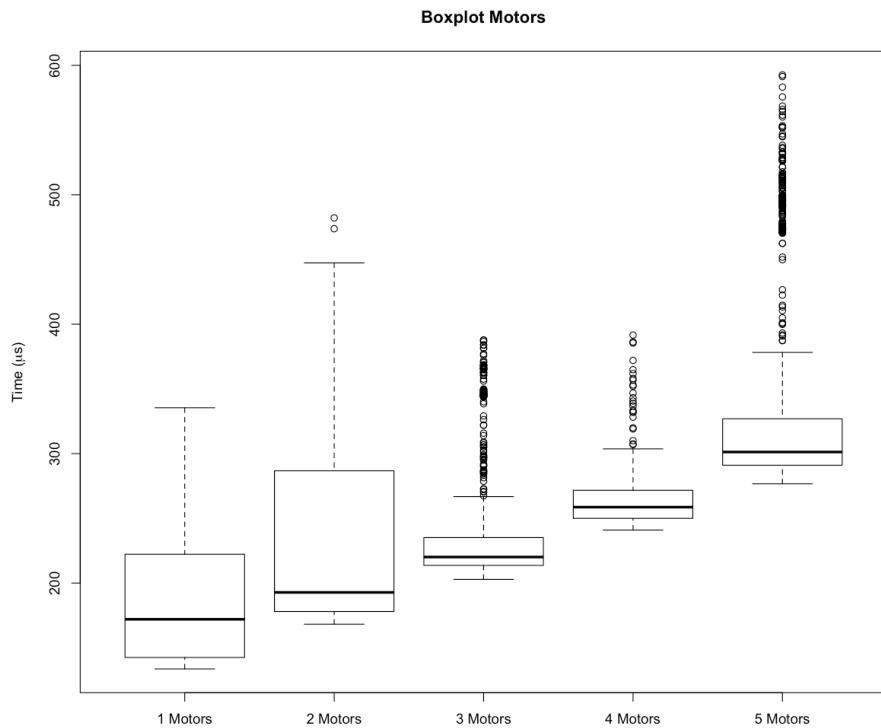


Figura 5.15: Gráficos de cajas de los tiempos de activación de motores Droid-Nexus  
Fuente: elaboración propia (2018)

### 5.1.3.2 Flexores

Tabla 5.6: Resumen de los resultados de los tiempos de lectura de flexores Droid-Nexus  
Fuente: Elaboración propia (2018)

flexors	Mean $\mu s$	Median $\mu s$	Min $\mu s$	Max $\mu s$	Std. Dev. $\mu s$	Skewness	Kurtosis
1	109,671	114,496	48,099.480	172,982.100	24,155.450	-0.304	2.621

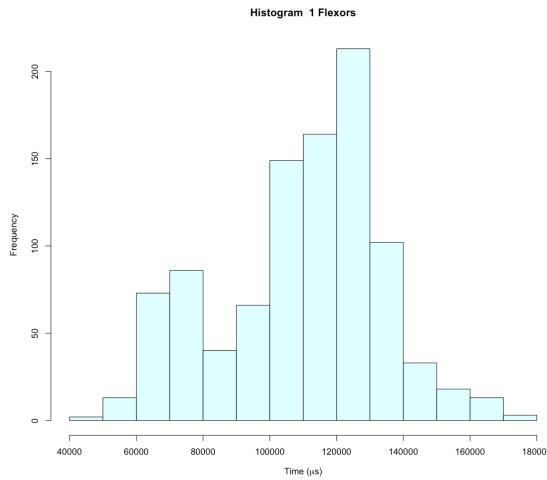


Figura 5.16: Histogramas de los tiempos de lectura de flexores Droid-Nexus  
Fuente: elaboración propia (2018)

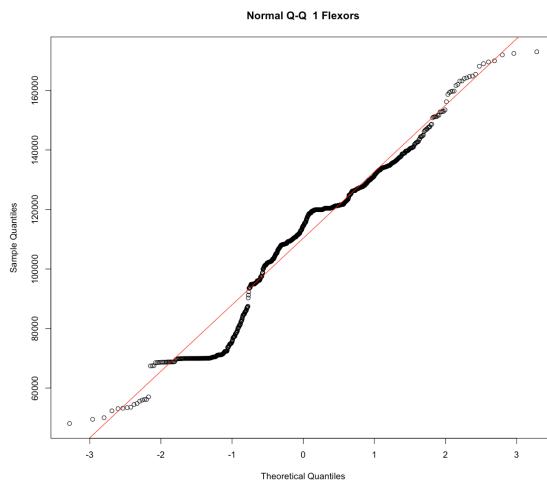


Figura 5.17: Gráficos QQ de los tiempos de lectura de flexores Droid-Nexus  
Fuente: elaboración propia (2018)

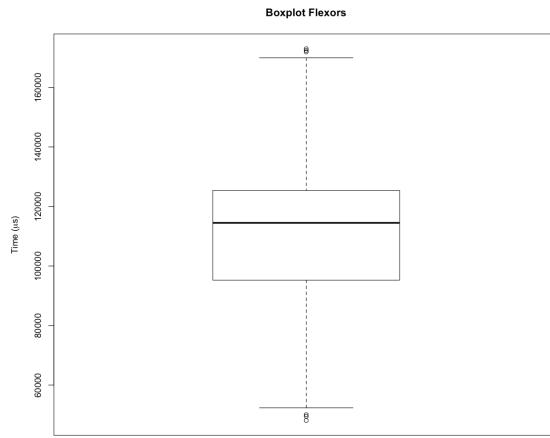


Figura 5.18: Gráficos de cajas de los tiempos de lectura de flexores Droid-Nexus  
Fuente: elaboración propia (2018)

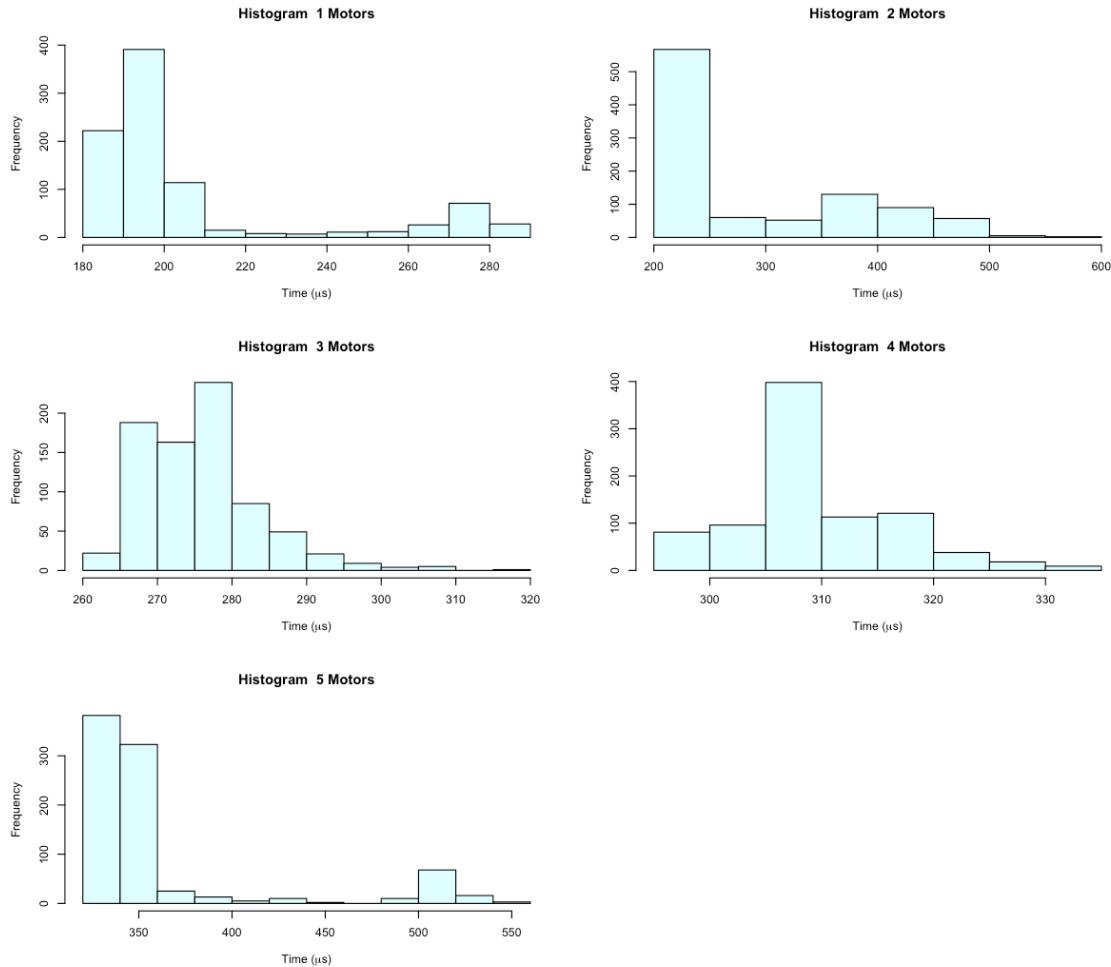
#### 5.1.4 Prototipo 4: Xamarin - Nexus

##### 5.1.4.1 Motores

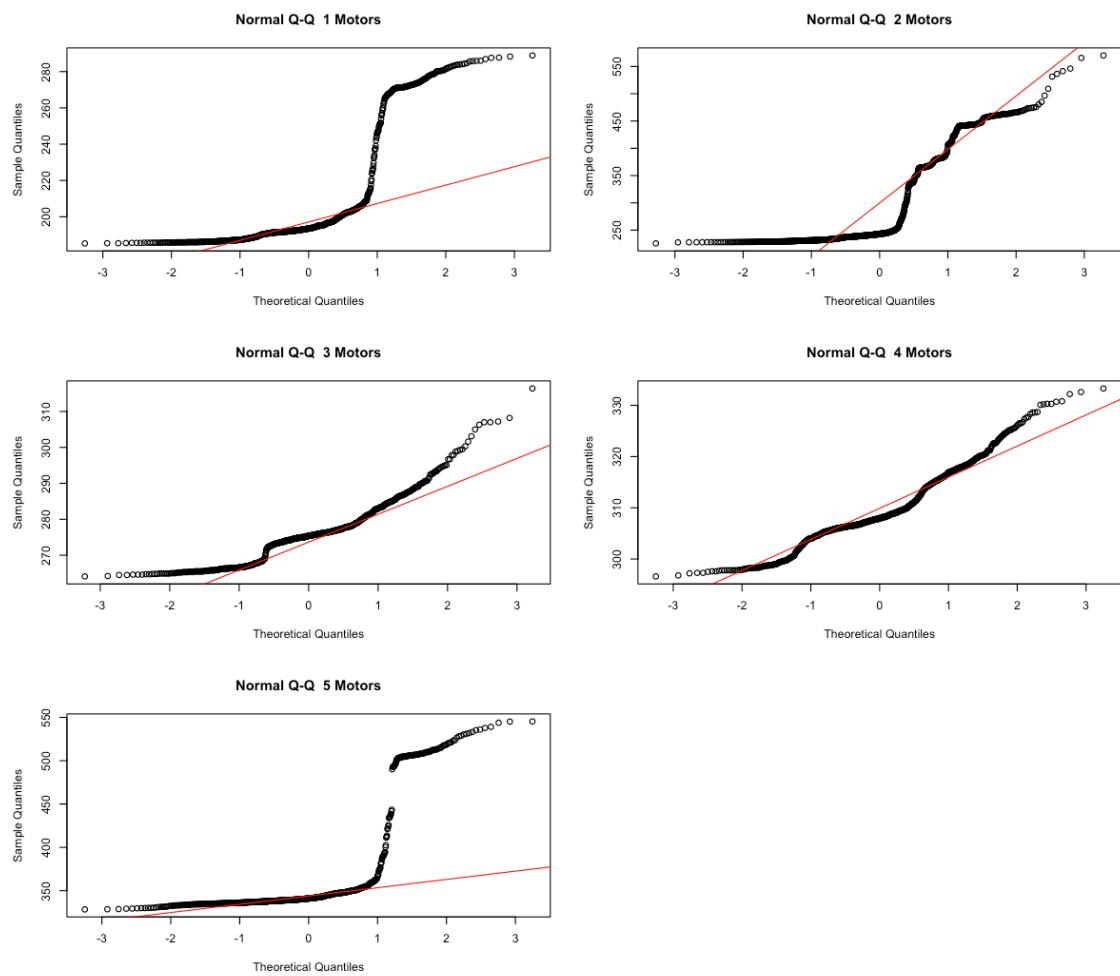
Tabla 5.7: Resumen de los resultados de los tiempos de activación de motores Xamarin-Nexus  
Fuente: Elaboración propia (2018)

motors	Mean	Median	Min	Max	Std. Dev.	Skewness	Kurtosis
	$\mu s$						
1	206.856	193.600	185.300	288.900	29.516	1.673	4.187
2	296.004	243.100	225.600	570.200	83.534	0.995	2.558
3	275.752	275.400	264.100	316.400	8.007	1.078	5.129
4	309.417	307.950	296.600	333.300	6.822	0.707	3.544
5	363.526	341.100	328.400	545.300	54.915	2.211	6.235

La Figura 5.19, muestra los histogramas de las latencias obtenidas al mandar los mensajes de activación y desactivación. Se realizaron pruebas desde uno a cinco motores.



**Figura 5.19: Histogramas de los tiempos de activación de motores Xamarin-Nexus**  
**Fuente:** elaboración propia (2018)



**Figura 5.20: Gráficos QQ de los tiempos de activación de motores Xamarin-Nexus**  
**Fuente:** elaboración propia (2018)

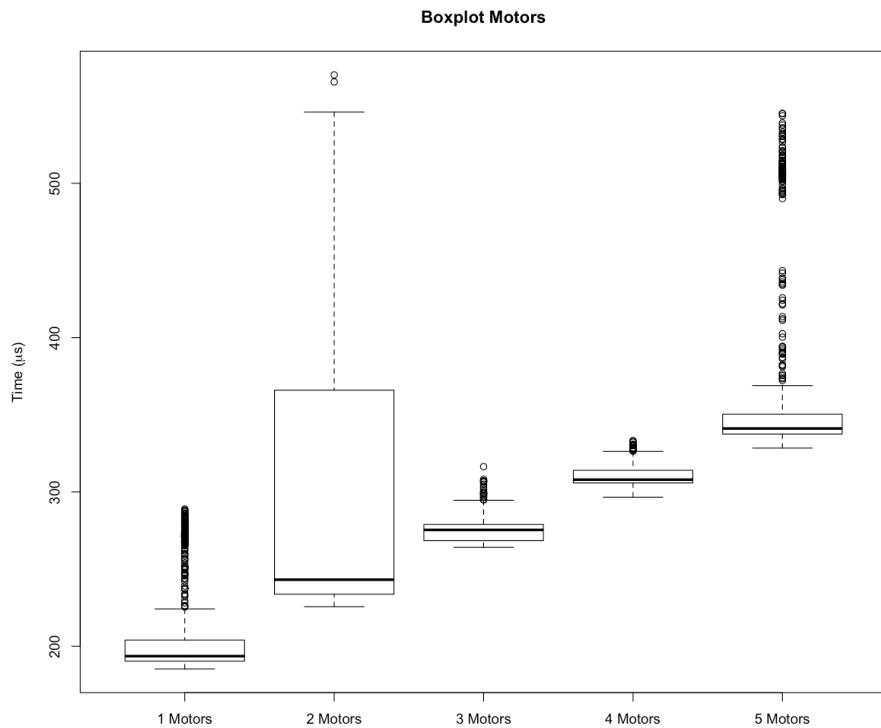


Figura 5.21: Gráficos de cajas de los tiempos de activación de motores Xamarin-Nexus  
Fuente: elaboración propia (2018)

#### 5.1.4.2 Flexores

Tabla 5.8: Resumen de los resultados de los tiempos de lectura de flexoresXamarin-Nexus  
Fuente: Elaboración propia (2018)

flexors	Mean $\mu s$	Median $\mu s$	Min $\mu s$	Max $\mu s$	Std. Dev. $\mu s$	Skewness	Kurtosis
1	109,677	110,958	67,022.400	158,703.500	18,411.270	-0.353	2.914

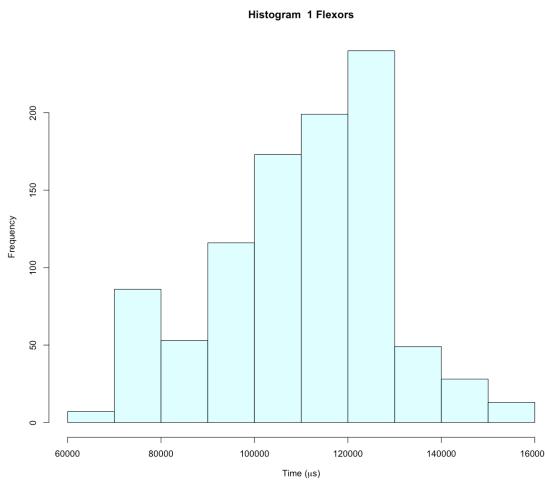


Figura 5.22: Histogramas de los resultados de los tiempos de lectura de flexores Xamarin-Nexus  
Fuente: elaboración propia (2018)

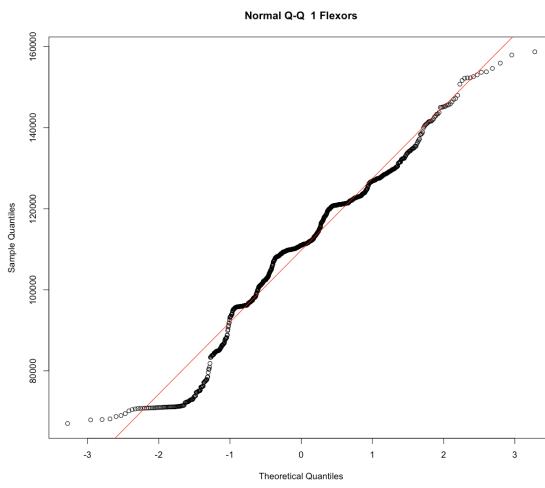


Figura 5.23: Gráficos QQ de los resultados de los tiempos de lectura de flexores Xamarin-Nexus  
Fuente: elaboración propia (2018)

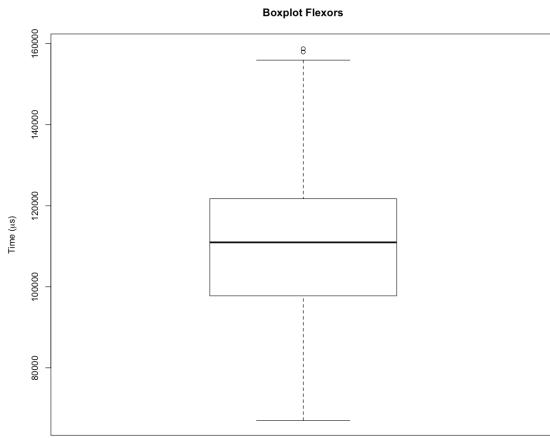


Figura 5.24: Gráficos de cajas de los resultados de los tiempos de lectura de flexores  
Xamarin-Nexus  
Fuente: elaboración propia (2018)

## 5.2 EVALUACIÓN TIEMPO DE ACTIVACIÓN USANDO APIS

El tipo de prueba que se realizó se basa en el trabajo realizado por Monsalve (2015) y Meneses (2016). En ambos trabajos se considera el tiempo de las tres etapas comprendidas por: el tiempo de la generación de mensajes, el tiempo de envío y el tiempo de activación de actuadores en el Software Arduino. En este proyecto, se considera el tiempo de generación de mensajes en la API de alto nivel (C# y Java), el tiempo que demora en llegar al Servidor WebSocket, el tiempo de la generación de mensajes desde la instancia de OpenGlove en el servidor, luego el tiempo hasta software de control en la placa Arduino, el tiempo a través del Bluetooth, para finalmente calcular el tiempo que demora el microcontrolador Arduino en realizar la activación de los actuadores. Se realizaron 2000 pruebas entre la activación y desactivación, utilizando un Baudrate de 57600 y un LoopDelay igual a 0 ms en la placa Arduino. Los tiempos entre activación y desactivación desde la API de alto nivel, están dados por la velocidad entre cada mensaje que llega desde el software de control, evitando así, los efectos indeseados por el envío de mensajes en corto tiempo.

Se realizaron pruebas desde uno a cinco motores físicos disponibles en la placa. El dispositivo utilizado para la evaluación técnica de las APIs fue el Samsung Galaxy S5 Mini. La Figura 5.25 muestra la aplicación desarrollada con Xamarin.Forms para realización de las evaluaciones técnicas de la API C# y la Figura 5.26, muestra la aplicación desarrollada para la realización de evaluaciones técnicas de la API Java.

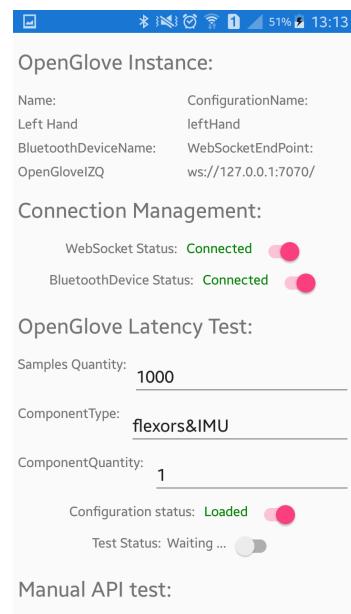


Figura 5.25: Aplicación utilizada para realizar las pruebas de la API C#  
Fuente: elaboración propia (2018)

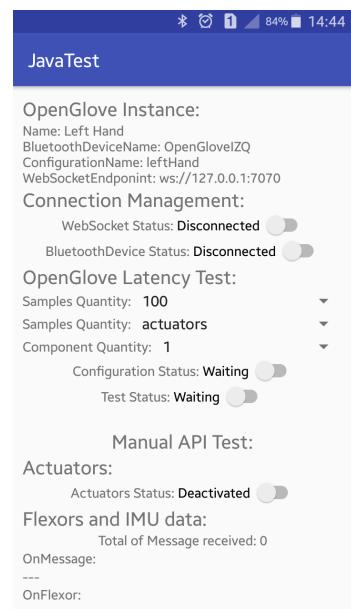


Figura 5.26: Aplicación utilizada para realizar las pruebas de la API Java  
Fuente: elaboración propia (2018)

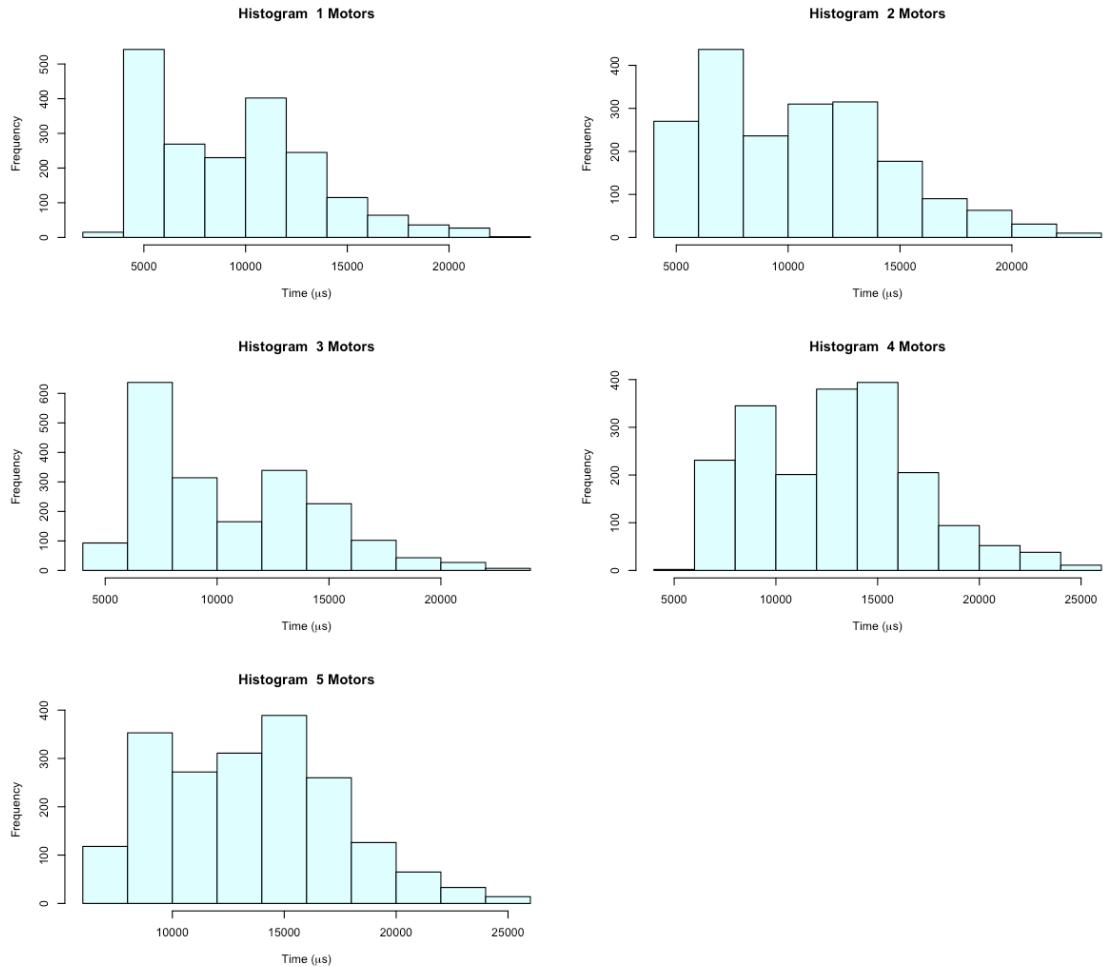
### 5.2.1 API C#

Para realizar las evaluaciones técnicas de esta API, fue necesario desarrollar una aplicación en Xamarin.Forms mostrada en la Figura 5.25, que hiciera uso de la API y registrara los tiempos de latencia obtenidos en el dispositivo Android.

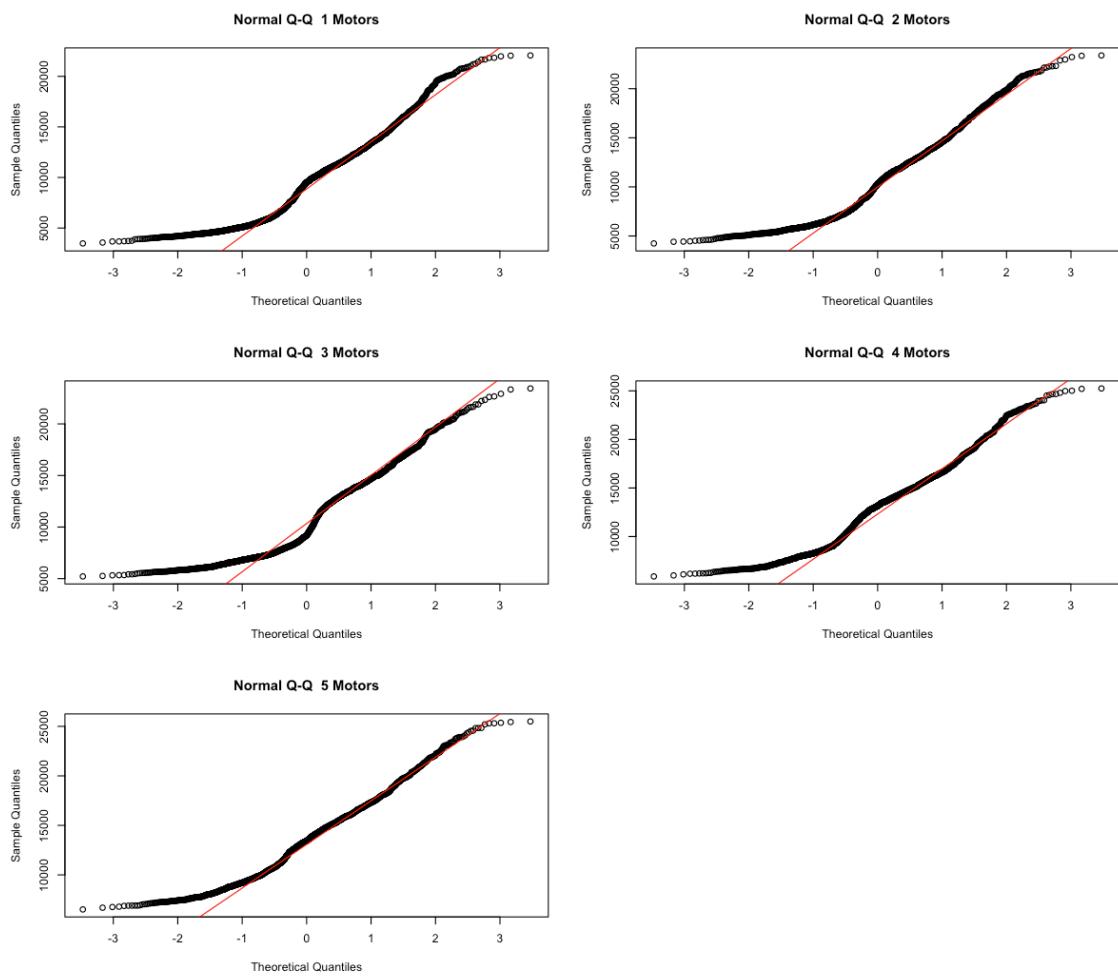
Tabla 5.9: Resumen resultados de los tiempos de activación usando API C# en  $\mu s$   
Fuente: Elaboración propia (2018)

motors	Mean	Median	Min	Max	Std. Dev.	Skewness	Kurtosis
	$\mu s$	$\mu s$	$\mu s$	$\mu s$	$\mu s$		
1	9,447.961	9,519.400	3,478.700	22,064.100	4,011.735	0.625	2.841
2	10,505.070	10,292.400	4,241.800	23,386.200	4,083.306	0.615	2.712
3	10,573.240	9,208.100	5,214.200	23,436.100	3,868.327	0.683	2.598
4	12,915.820	13,147.200	5,894.100	25,246.200	4,018.170	0.383	2.716
5	13,497.350	13,429	6,513.400	25,496	3,900.305	0.410	2.665

La Figura 5.7, muestra los histogramas de las latencias obtenidas al mandar los mensajes de activación y desactivación. Se realizaron pruebas desde uno a cinco motores.



**Figura 5.27: Histogramas de los tiempos de activación de motores usando API C#**  
**Fuente:** elaboración propia (2018)



**Figura 5.28: Gráficos QQ de los tiempos de activación de motores usando API C#**  
**Fuente:** elaboración propia (2018)

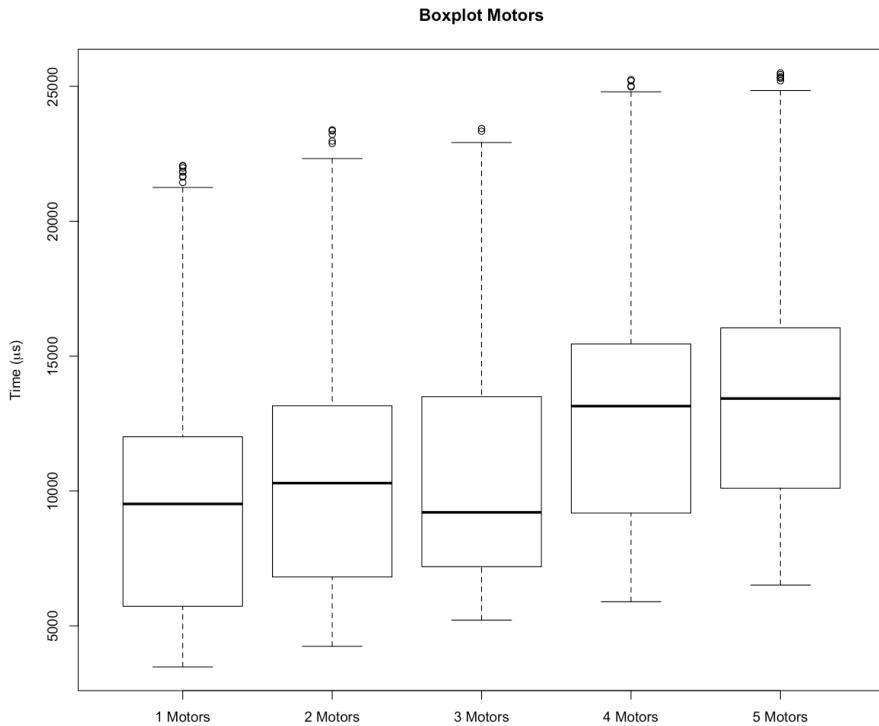


Figura 5.29: Gráficos de cajas de los tiempos de activación de motores usando API C#  
Fuente: elaboración propia (2018)

### 5.2.2 API Java

Para realizar las evaluaciones técnicas de esta API, fue necesario desarrollar una aplicación nativa en Android mostrada en la Figura 5.26, que hiciera uso de la misma para registrar las latencias obtenidas en las pruebas.

## 5.3 EVALUACIÓN TIEMPO DE LECTURA DE DATOS USANDO APIS

El tipo de prueba que se realizó es el mismo que el utilizado por Cerda (2017), por tanto se considera como ciclo de lectura cuando el software de control Arduino envía los valores de todos los flexores agregados. Esto es realizado en paralelo a la lectura de todos los datos que entrega el IMU. Cada medición considera el tiempo transcurrido desde la generación del mensaje en el software de control hasta que llega a la API de alto nivel. Se realizaron 1000 pruebas de

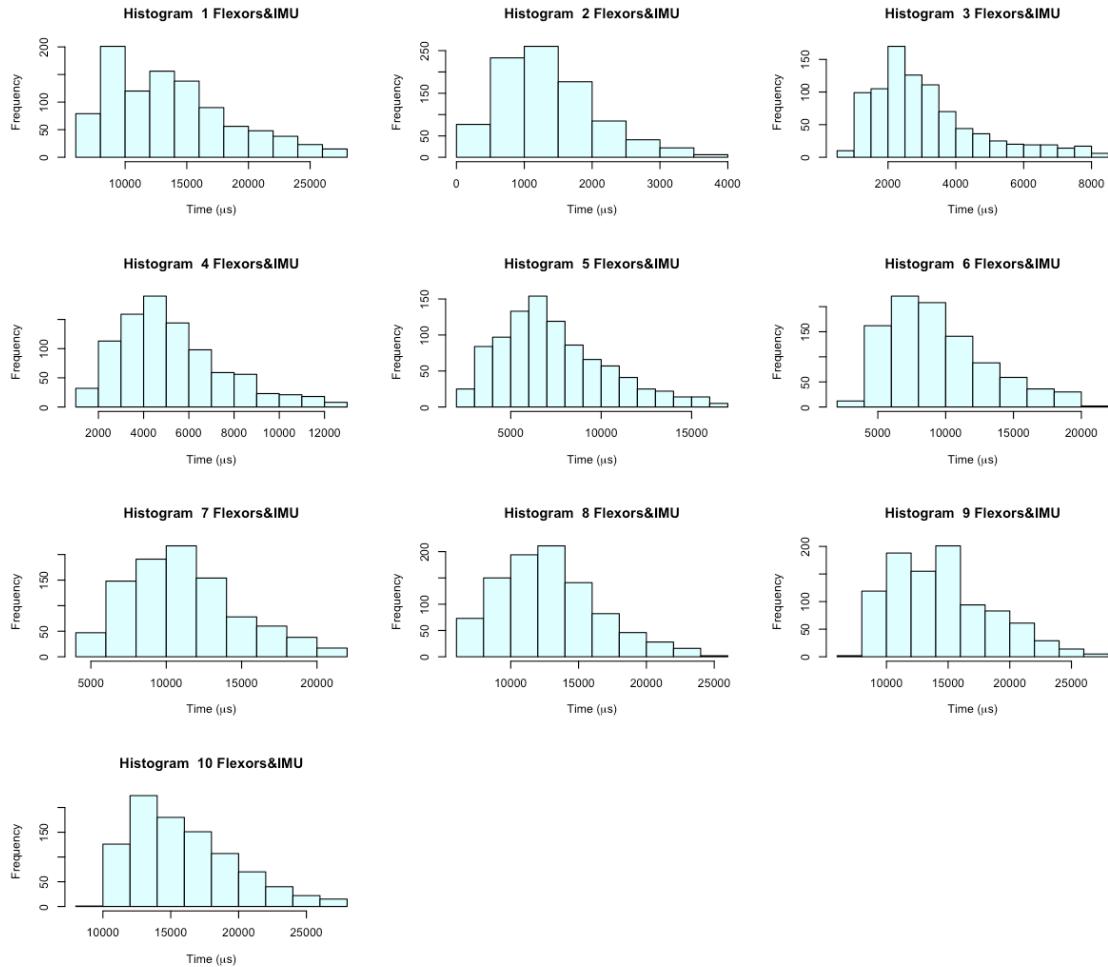
ciclos lectura utilizando un Baudrate de 57600, un LoopDelay igual a 0 ms y Threshold igual a 0 en la placa Arduino. Se realizaron pruebas con el IMU enviando la información completa (acelerómetro, giroscopio y magnetómetro) y modificando la cantidad de flexores desde uno a 10, siendo simulados solamente con un flexor físico en la placa. El dispositivo utilizado para la evaluación técnica de las APIs fue el Samsung Galaxy S5 Mini. Se utilizan las mismas aplicaciones desarrolladas para la evaluación de tiempo de activación de actuadores.

### 5.3.1 API C#

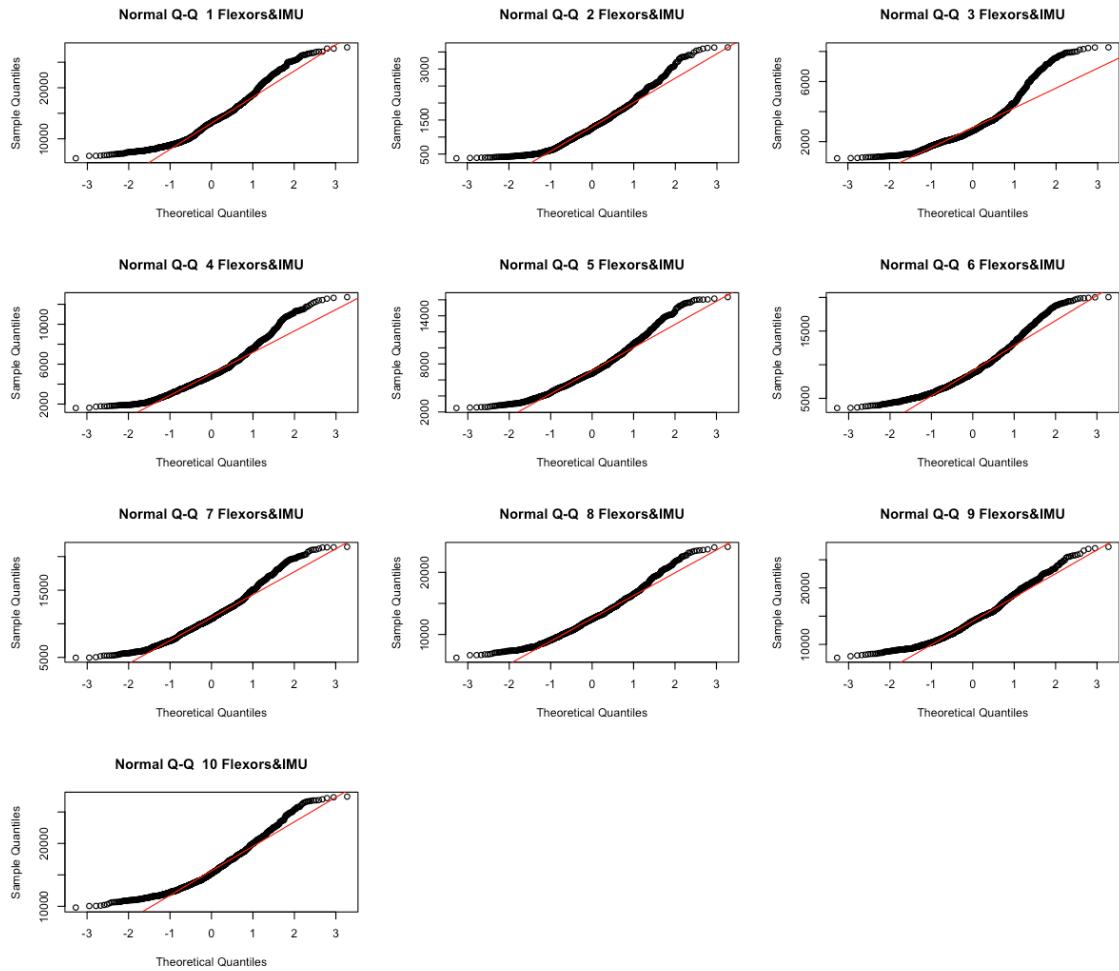
Tabla 5.10: Resumen resultados de pruebas de lectura flexores e IMU usando API C# en  $\mu s$   
 Fuente: Elaboración propia (2018)

flexors	Mean $\mu s$	Median $\mu s$	Min $\mu s$	Max $\mu s$	Std. Dev. $\mu s$	Skewness	Kurtosis
1	16,038.830	15,068.100	7,417.900	28,865.600	4,484.658	0.692	2.908
2	1,365.986	1,271.700	376.800	3,635.400	704.429	0.816	3.307
3	3,133.853	2,734.500	895.200	8,272.900	1,636.889	1.166	3.906
4	5,251.955	4,796.100	1,605	12,720.200	2,327.944	0.877	3.417
5	7,391.620	6,823.100	2,496.400	16,335.200	2,972.165	0.739	3.123
6	9,445.057	8,723.800	3,565	20,047.800	3,703.347	0.815	3.074
7	11,231.080	10,768.700	4,948.100	21,445.900	3,611.584	0.609	2.891
8	12,864.060	12,476.600	6,264.900	24,091.800	3,660.714	0.646	3.045
9	14,512.320	14,120.300	7,632.200	27,282	4,010.673	0.657	2.891
10	15,967.690	15,187.800	9,808	27,444.800	3,769.420	0.806	3.086

La Figura ??, muestra los histogramas de las latencias obtenidas al recibir los mensajes de los flexores e IMU, modificando la cantidad de flexores de uno a diez.



**Figura 5.30: Histogramas de Flexores e IMU usando API C#**  
**Fuente:** elaboración propia (2018)



**Figura 5.31: Flexores e IMU usando API C#**  
**Fuente:** elaboración propia (2018)

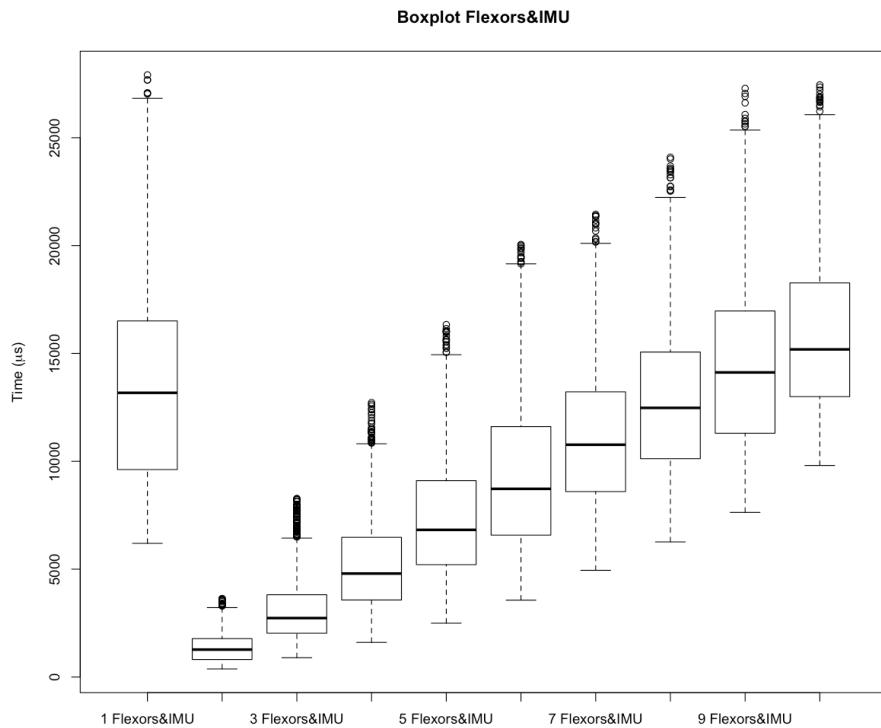


Figura 5.32: Gráficos de cajas de Flexores e IMU usando API C#  
Fuente: elaboración propia (2018)

### 5.3.2 API Java

## 5.4 PRUEBAS DE CONCEPTO

En esta sección se muestran las diferentes pruebas de concepto hechas en ambientes de (VR/AR/MR) ...

## 5.5 RESUMEN

## CAPÍTULO 6. CONCLUSIONES

Este capítulo presenta las conclusiones del proyecto realizado, abordando el cumplimiento de los objetivos del proyecto, los resultados obtenidos, los alcances y limitaciones de los mismos, propuestas de trabajo futuro y las observaciones finales referentes a todo el proyecto.

### 6.1 OBJETIVOS

En esta sección se concluye sobre el objetivo general y específicos del proyecto, detallando el nivel de completitud de cada uno.

#### 6.1.1 Objetivos específicos

*6.1.1.1 Desarrollar la aplicación de configuración de OpenGlove en Android. Permitiendo la creación, visualización, actualización y eliminación de los perfiles de configuración:*

Se logró desarrollar la aplicación de configuración utilizando Xamarin.Forms permitiendo compartir la interfaz de usuario, realizando modificaciones mínimas según los patrones de diseño para iOS y Android. Además se logró compartir el código sobre el almacenamiento y el servidor WebSocket, esto fue logrado utilizando paquetes de software compatibles<sup>1</sup> con Xamarin.Forms (.NET Standard 2.0). Con esto se logró abstraer la complejidad de las configuraciones de la placa, de la IMU, los mapeos los flexores y actuadores, gracias a esta herramienta que permite configurar y administrar los distintos perfiles de configuración que pueden ser utilizados de manera independiente en cada dispositivo Bluetooth. Las diferentes iteraciones del desarrollo de esta aplicación fueron detalladas en la Sección 3.2. La estructura y descripción puede ser detallada en la Subsección 4.2.4 de este documento.

Adicionalmente esta aplicación permite probar los actuadores, flexores e IMU previamente configurados. También permite la administración de los dispositivos Bluetooth vinculados, todo esto haciendo uso de la API C# desarrollada en paralelo a esta aplicación.

---

<sup>1</sup> Paquetes de software NuGet: <https://www.nuget.org/packages>

#### *6.1.1.2 Desarrollar APIs en Java y C# que permitan la administración de dispositivos Bluetooth en*

*segundo plano en Android permitiendo la conexión, desconexión, activación y listado de guantes OpenGlove. También permitirá la activación y control de actuadores, flexores e IMU*

El desarrollo de las APIs fue realizado completamente para los dos lenguajes de programación Java y C#, permitiendo realizar todas las funcionalidades descritas en este objetivo. Estas APIs fueron desarrolladas como clientes WebSocket que se comunican bajo un protocolo de mensajes especificado en el Anexo de este documento. Gracias a esto los desarrolladores pueden implementar diferentes funcionalidades a sus aplicaciones, como lo es el caso de la aplicación de configuración de este proyecto, el cual brinda funcionalidades que permiten probar los actuadores, flexores e IMU. La estructura y descripción de las APIs de alto nivel desarrolladas se detalla en la Subsección 4.2.3. El comportamiento de las APIs fue detallado en la 4.3.

#### *6.1.1.3 Realizar evaluaciones de rendimiento para las APIs Java y C# :*

Se lograron realizar las evaluaciones de rendimiento necesarias para las APIs, considerando también las evaluaciones que buscaban comparar el desarrollo de aplicaciones nativas en Android usando Android Studio con el uso de Xamarin.Forms como herramienta multiplataforma para aplicaciones nativas. Las evaluaciones permitieron determinar los rendimientos obtenidos en los dispositivos móviles. En comparación a los resultados obtenidos en el trabajo de Cerda (2017), los de este proyecto presentaron rendimientos inferiores, pero esto es debido a las limitaciones del dispositivo móvil utilizado, comparado con el ordenador en el que el trabajo previo fue evaluado.

El Capítulo ??

#### *6.1.1.4 Demostrar el uso del SDK en un ambiente de VR, AR o MR, utilizando las APIs*

*desarrolladas:*

Fue posible realizar una prueba de concepto simple, utilizando las herramientas de GoogleARCore/GoogleVR, generando colisiones en regiones especificadas para activar los actuadores y generando el movimiento gracias a los datos provistos por el IMU. De esta forma es posible demostrar de manera práctica el uso de las APIs en los ambientes de AR, VR o MR. la

sección de pruebas

### **6.1.2 Objetivo general**

El objetivo general del proyecto fue desarrollar un SDK que permita dar soporte a OpenGlove en dispositivos móviles. Como producto final se obtiene una aplicación de configuración para Android, con APIs en lenguajes de programación Java y C#. Se proporciona un SDK para dispositivos móviles de Código Abierto (Open Source) que incluye las APIs de alto nivel, la documentación y todo los recursos generados durante el desarrollo del proyecto (prototipos, iconos, imágenes, diagramas, etc.)

## **6.2 RESULTADOS OBTENIDOS**

Esta sección se concluye sobre los resultados obtenidos sobre el desarrollo de software y los resultados de las pruebas realizadas en el presente trabajo.

### **6.2.1 Desarrollo de software**

El desarrollo de software de este proyecto tiene como producto final el SDK para dispositivos móviles, el cual se compone de las APIs de alto nivel, la aplicación de configuración, pruebas de concepto y toda la documentación generada durante el proyecto. Estas APIs desarrolladas en Java y C# dan soporte a la gestión de dispositivos Bluetooth, los actuadores, sensores de flexibilidad y sensor de rastreo IMU. Gracias a esto es posible hacer uso de OpenGlove en ambientes de Realidad Virtual, Aumentada o Mixta en dispositivos móviles, con tecnologías que sean compatibles con los lenguajes de programación ya mencionados.

### **6.2.2 Resultados de las pruebas**

Los resultados de las pruebas de rendimiento son aceptables dado que no superan el umbral de latencia máximo que las personas pueden detectar. Esto puede ser mejorado utilizando

dispositivos con mayores prestaciones que el utilizado en las pruebas de rendimiento, por tanto, las especificaciones de Hardware del dispositivo móvil utilizado, se plantean como requerimientos mínimos recomendados para utilizar OpenGlove en Android, considerando siempre disponer de prestaciones superiores de acuerdo a los requerimientos mínimos para las aplicaciones de Realidad Virtual, Aumentada o Mixta.

### 6.3 ALCANCES Y LIMITACIONES

- El estado actual del SDK de OpenGlove desarrollado, solo permite ser utilizado en Android. Para dar soporte en iOS es necesario implementar la comunicación con dispositivos Bluetooth utilizando las APIs nativas de iOS, además de replicar por lo menos una API como las hechas en Java y C# si fuera necesario. Existe la posibilidad de usar la API C# en iOS desarrollando aplicaciones con Xamarin con C# y ver la integración con Unity (también compatible con C#).
- La aplicación para iOS solo fue probada utilizando los dispositivos virtuales provistos por el IDE XCode de Mac, los cuales fueron iPhone X, iPhone 8 y iPhone 8 Plus con la versión 11.4 de iOS. Con esto sólo fue posible probar la interfaz de usuario de la aplicación, ya que no se pudo probar la conexión con dispositivos Bluetooth ni el uso del servidor WebSocket.
- La aplicación para Android fue probada utilizando dispositivos virtuales y reales. De los dispositivos virtuales provistos por Android Virtual Device Manager, sólo se utilizó el Nexus 5X (API 23). Para pruebas en dispositivos reales, se utilizó el Samsung Galaxy S5 mini con Android 6.0.1 (API 23) y Nexus 5 Android 6.0.1 (API 23). Con los dispositivos reales, fue posible probar el funcionamiento correcto de la aplicación, lo que incluye aspectos técnicos como la conexión con dispositivos Bluetooth y el correcto funcionamiento del servidor WebSocket. En los dispositivos virtuales solo fue posible probar la interfaz de usuario, dado que no fue posible probar la conexión con dispositivos Bluetooth ni el uso del servidor WebSocket.
- No se han realizado validaciones automatizadas del software del SDK. Sólo se han realizado pruebas de aceptación con el profesor guía y las pruebas de concepto que permiten ver el uso las APIs.
- No se pudieron realizar pruebas reales con el máximo de flexores permitido, debido a las limitaciones del hardware (máximo cinco). Sin embargo basado en el trabajo de Cerda (2017),

se repitió el flexor en otra región, para simular las lecturas de otro real, obteniendo un total de diez flexores, la mitad reales y la otra mitad simulados.

- Debido al hardware disponible no fue posible realizar conexión simultánea con dispositivos Bluetooth OpenGlove, dado que solo se disponía de un módulo Bluetooth.

## 6.4 TRABAJO FUTURO

- Dar soporte al SDK en iOS, desarrollando como mínimo la comunicación y administración de dispositivos Bluetooth. Luego desarrollar APIs que soporten otros lenguajes de programación para dispositivos móviles como Kotlin, Swift y Objective C.
- Desarrollar patrones de activación de actuadores extensibles en las APIs de alto nivel, para evitar que el desarrollador deba crear sus propios patrones básicos cada vez y que pueda extender estos patrones para crear otros más complejos.
- Evaluar la posibilidad de extender este proyecto en otras plataformas<sup>2</sup>, dado que Xamarin.Forms soporta las plataformas Android, iOS, UWP (Universal Windows Platform) y Tizen en una versión estable.
- Establecer la calibración del IMU para los diferentes dispositivos OpenGlove para establecer una orientación específica. También desarrollar un algoritmo encargado de interpretar la orientación y posición de la mano (o donde se encuentre el IMU), para obtener una interpretación precisa en tiempo real. Bajo esta misma línea de trabajo, se propone evaluar el uso de los sensores de los dispositivos móviles para establecer una calibración, esto puede ser de uso complementario, ya que al existir diferentes fabricantes, estos no poseen las mismas especificaciones entre cada modelo de smartphone. De manera similar, se podrían utilizar los dispositivos smartwatch, para iOS o Android, utilizando el mismo proyecto Xamarin.Forms para dar soporte a todas las variantes que pueden ser desarrolladas.
- Agregar soporte a otros modelos de sensores de rastreo IMU. Actualmente no es posible seleccionar otro modelo de IMU aparte de SparkFun LSM9DS1, dado que requiere bibliotecas de software específicas en el código de la placa Arduino. Para ello la comunidad debe desarrollar variantes del software de control Arduino para diferentes modelos de IMU. De manera adicional se podría dar soporte una funcionalidad de carga de código en la placa Arduino utilizando la aplicación de configuración. Esto podría realizarse utilizando la conexión Bluetooth o por medio de conexión USB adecuada. Por ejemplo, existe una

---

<sup>2</sup>Plataformas soportadas por Xamarin.Forms <https://github.com/xamarin/Xamarin.Forms/wiki/Platform-Support>

herramienta llamada Bluino Loader - Arduino IDE<sup>3</sup>, la cual permite cargar código a la placa Arduino por Bluetooth o USB.

## 6.5 OBSERVACIONES FINALES

---

<sup>3</sup>Bluino Loader: <https://www.hackster.io/mansurkamsur/upload-sketch-arduino-over-bluetooth-using-android-f1ce55>

## GLOSARIO

- **Actuadores:** Un actuador es un dispositivo inherentemente mecánico cuya función es proporcionar fuerza para mover o “actuar” otro dispositivo mecánico. La fuerza que provoca el actuador proviene de tres fuentes posibles: Presión neumática, presión hidráulica, y fuerza motriz eléctrica (motor eléctrico o solenoide). Dependiendo de el origen de la fuerza el actuador se denomina “neumático”, “hidráulico” o “eléctrico” (Aie, 2017).
- **API:** *Application Program Interface* por sus siglas en inglés es código que actúa como interfaz para la programación de aplicaciones, permitiendo por ejemplo, que dos aplicaciones se comuniquen entre si, como el acceder a funcionalidades sin la necesidad de conocer la complejidad del código implementado(Rouse, 2017).
- **Augmented Reality (AR):** La realidad aumentada (AR) es el uso de información en tiempo real en forma de texto, gráficos, audio y otras mejoras virtuales integradas con objetos del mundo real. Es este elemento del “mundo real” lo que diferencia a AR de la realidad virtual. AR integra y agrega valor a la interacción del usuario con el mundo real, frente a una simulación. (Gartner, 2017a).
- **Haptic Feedback:** Haptics es una tecnología táctil o de retroalimentación de fuerza que aprovecha el sentido del tacto de una persona al aplicar vibraciones y / o movimiento a la punta del dedo del usuario. Esta estimulación puede ayudar a la tecnología en el desarrollo de objetos virtuales en la pantalla del dispositivo. En su sentido más amplio, hapticos puede ser cualquier sistema que incorpore elementos táctiles y vibre a través de un sentido del tacto (Gartner, 2017b).
- **IMU (Inertial Measurement Unit):** Los sensores iniciales, también llamados IMU (Unidad de medición inercial), son dispositivos electrónicos de medición que permiten estimar la orientación de un cuerpo de las fuerzas iniciales que el cuerpo experimenta. Su principio de funcionamiento se basa en la medición de las fuerzas de aceleración y velocidad angular ejercidas independientemente en masas pequeñas ubicadas en el interior (Technaid, 2018).
- **OpenGlove:** es un guante desarrollado por la Universidad de Santiago de Chile, por el grupo de investigación y desarrollo InTeracTion, el cual provee *haptic feedback* o retroalimentación táctil en ambientes virtuales, como también la captura de movimientos de la mano, flexiones de las articulaciones y la orientación de la mano (InTeracTion, 2018).
- **Mixed reality (MR):** La realidad mixta es el resultado de mezclar el mundo físico con el mundo digital. La realidad mixta es la siguiente evolución en la interacción entre el hombre, la computadora y el entorno, y abre posibilidades que antes estaban restringidas a nuestra imaginación. Es posible gracias a los avances en visión artificial, potencia de procesamiento gráfico, tecnología de visualización y sistemas de entrada. El término realidad mixta fue presentado originalmente en un artículo de 1994 por Paul Milgram y Fumio Kishino, “Una taxonomía de visualizaciones de realidad mixta”. Su trabajo introdujo el concepto del continuum de virtualidad y se centró en cómo se aplica la categorización de la taxonomía a las exhibiciones. Desde entonces, la aplicación de la realidad mixta va más allá de las pantallas, pero también incluye la información ambiental, el sonido espacial y la ubicación. (Microsoft, 2017).
- **SDK:** conjunto de utilidades de desarrollo para escribir aplicaciones de software, generalmente asociadas a entornos específicos (por ejemplo, el SDK de Windows) (Gartner, 2017c).
- **UX:** La “experiencia de usuario” abarca todos los aspectos de la interacción del usuario final con la empresa, sus servicios y sus productos (Norman & Nielsen, 2017).
- **Virtual Reality (VR):** La realidad virtual (VR) proporciona un entorno 3D generado por computadora que rodea al usuario y responde a las acciones de esa persona de forma

natural, generalmente a través de pantallas inmersivas montadas en la cabeza y el seguimiento de la cabeza. También se pueden usar guantes que proporcionen seguimiento de las manos y retroalimentación háptica (sensible al tacto). Los sistemas basados en sala brindan una experiencia 3D para múltiples participantes; sin embargo, son más limitados en sus capacidades de interacción. (Gartner, 2017d).

- **WebSocket:** WebSocket es un protocolo que provee un canal de comunicación bidireccional y simultáneo (full-duplex) entre un cliente y un servidor, utilizando un único socket TCP (Transmission Control Protocol). (Websocket.org, 2018a)

## REFERENCIAS BIBLIOGRÁFICAS

- Aie (2017). Actuadores. Recuperado de <http://www.aie.cl/files/file/comites/ca/abc/actuadores.pdf> Revisado el 23 de Octubre de 2017.
- Batteau, L. M., Liu, A., Maintz, J. B. A., Bhasin, Y., & Bowyer, M. W. (2004). A study on the perception of haptics in surgical simulation. In S. Cotin, & D. Metaxas (Eds.) *Medical Simulation*, (pp. 185–192). Berlin, Heidelberg: Springer Berlin Heidelberg.
- Cerda, R. A. (2017). Extensión de openglove a nivel de hardware y software para la captura del movimiento de la mano.
- DextaRobotics (2018). Dexmo. Recuperado de <http://www.dextarobotics.com/> el 10 de Marzo de 2018.
- Gartner (2017a). It glossary: Augmented reality (ar). Recuperado de <https://www.gartner.com/it-glossary/augmented-reality-ar/> Revisado el 19 de Octubre de 2017.
- Gartner (2017b). It glossary: Haptics. Recuperado de <https://www.gartner.com/it-glossary/haptics> Revisado el 19 de Octubre de 2017.
- Gartner (2017c). It glossary: Sdk. Recuperado de <https://www.gartner.com/it-glossary/?s=SDK> Revisado el 19 de Octubre de 2017.
- Gartner (2017d). It glossary: Virtual reality (vr). Recuperado de <https://www.gartner.com/it-glossary/vr-virtual-reality/> Revisado el 19 de Octubre de 2017.
- Gartner (2018). It glossary: Operating system. Recuperado de <https://www.gartner.com/it-glossary/os-operating-system> Revisado el 28 Junio de 2018.
- Grigorik, I. (2013). High performance browser networking: Websocket. Recuperado de <https://hpbn.co/websocket/> Revisado el 25 de Junio de 2018.
- Haptx (2018). Haptx. Recuperado de <https://manus-vr.com/order.php> el 28 de Junio de 2018.
- InTeracTion (2018). Openglove haptics easy to develop. Recuperado de <http://www.openglove.org/> Revisado el 10 de Marzo de 2018.
- ManusVR (2018). Manus vr. Recuperado de <https://manus-vr.com/order.php> el 10 de Marzo de 2018.
- Martin, J. (1991). *Rapid Application Development*. Indianapolis, IN, USA: Macmillan Publishing Co., Inc.
- Meneses, S. A. (2016). Openglove sdk: Apis de alto nivel para c#, c++ java y javascript.
- Microsoft (2017). Mixed reality. [https://developer.microsoft.com/en-us/windows/mixed-reality/mixed\\_reality](https://developer.microsoft.com/en-us/windows/mixed-reality/mixed_reality) Revisado el 26 de Noviembre de 2017.
- MobiLoud (2018). Native, web or hybrid apps? what's the difference? Recuperado de <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/> Revisado el 30 de Junio de 2018.
- Monsalve, R. A. (2015). Dispositivo de retroalimentación táctil para interfaces naturales y de realidad virtual.
- Neurodigital (2018). Avatarvr. Recuperado de <https://www.neurodigital.es/avatarvr/> el 10 de Marzo de 2018.
- Norman, D., & Nielsen, J. (2017). The definition of user experience (ux). Recuperado de <https://www.nngroup.com/articles/definition-user-experience/> Revisado el 21 de Octubre de 2017.

- Okamoto, S., Konyo, M., Saga, S., & Tadokoro, S. (2009). Detectability and perceptual consequences of delayed feedback in a vibrotactile texture display. *IEEE Transactions on Haptics*, 2(2), 73–84.
- Preston-Werner, T. (2018). stargazer: Well-formatted regression and summary statistics tables. Recuperado de <https://semver.org/> Revisado el 25 de Agosto de 2018.
- PubNub (2018). Pubnub websockets. Recuperado de <https://www.pubnub.com/websockets/> Revisado el 1 de Agosto de 2018.
- Rambal (2018). Sensor flex. Recuperado de <http://rambal.com/presion-peso-nivel-liquido/250-sensor-flex.html> Revisado el 23 de Abril de 2018.
- Rouse, M. (2017). application program interface (api). Recuperado de <http://searchmicroservices.techtarget.com/definition/application-program-interface-API> Revisado el 21 de Octubre de 2017.
- Statista (2016). Virtual reality (vr) - statistics and facts. Recuperado de <https://www.statista.com/topics/2532/virtual-reality-vr/> Revisado el 20 de Octubre de 2017.
- Statista (2017). The smartphone platform war is over. Recuperado de <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/> Revisado el 29 de Junio de 2018.
- Technaid (2018). Sensor de rastreo imu. Recuperado de <http://www.technaid.com/support/research imu-working-principles/> Revisado el 22 de Abril de 2018.
- Techopedia (2017). Software development kit (sdk). Recuperado de <https://www.techopedia.com/definition/3878/software-development-kit-sdk> Revisado el 19 de Octubre de 2017.
- Techterms (2010). Sdk definition. Recuperado de <https://techterms.com/definition/sdk> Revisado el 30 de Junio de 2018.
- Ticportal (2018). Open source (código abierto). Recuperado de <https://www.ticportal.es/glosario-tic/open-source-codigo-abierto> Revisado el 28 de Abril de 2018.
- WebSocket.org (2018a). About html5 websocket. Recuperado de <http://websocket.org/aboutwebsocket.html> Revisado el 5 de Junio de 2018.
- WebSocket.org (2018b). Html5 websocket: A quantum leap in scalability for the web. Recuperado de <https://websocket.org/quantum.html> Revisado el 25 de Junio de 2018.
- Wu, B., Sim, S. H., Enquobahrie, A., & Ortiz, R. (2015). Effects of visual latency on visual-haptic experience of stiffness. In *2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX)*, (pp. 1–6).

## **ANEXO A. LOW LEVEL COMMUNICATION PROTOCOL**

Historial de versiones: 1.0.0 -> 1.1.0 -> 1.2.0 ...

## **ANEXO B. HIGH LEVEL COMMUNICATION PROTOCOL**

Versión 1.0.0 del protocolo ...

## **ANEXO C. HIGH LEVEL OPENGLOVE APIs REFERENCE**

**C.1 C# API**

**C.2 JAVA API**

## **ANEXO D. PROOF OF CONCEPTS**

Agregar Imágenes ... aplicaciones prueba de APIs C# y Java. Agregar Imágenes ... aplicación(es) VR/AR/MR usando API(s) C#, Java.