Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# Diseño e implementación de sistema para evaluación de la ergonomía de herramientas

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Guillermo Olmeda Zurriaga

**Tutor**: Carlos Tavares Calafate

**Cotutor**: Rafael Juarez Ortiz-Villajos

Curso 2018-2019

*A mi familia y amigos, por apoyarme y darme consejo en este proyecto que es la vida,*

*a Carlos, por su inestimable ayuda durante la creación de este trabajo,*

*y a Ford España y mis compañeros de trabajo, por acogerme como uno más.*

# Resumen

Los procesos de producción en cadena y automatización de operaciones han permitido la producción en grandes cantidades de todo tipo de materiales y dispositivos desde hace ya décadas. La industria automovilística ha sido una de las grandes impulsoras de dichas técnicas de automatización, permitiendo construir y poner a la venta vehículos de forma rápida y eficiente. En la actualidad, todo proceso de producción consta de una serie de operaciones a realizar que comprenden desde la colocación de elementos de forma manual, a la incorporación de elementos con brazos robóticos de forma automática. La mayoría de estas operaciones son realizadas por operarios con herramientas diseñadas para ello.

En una línea de producción, las operaciones de apriete de componentes representan gran parte de este proceso. En ellas, se especifica cada uno de los pasos a seguir por el operario, indicando el tornillo a colocar, y configurando herramientas para realizar dichos aprietes según las especificaciones de diseño. Pese a disponer de metodologías de monitorización y supervisión de los aprietes realizados, no existen especificaciones concretas que determinen *a priori* si un apriete realizado en unas condiciones concretas cumple con los requisitos mínimos de ergonomía, es decir, no causa daños al operario a corto o largo plazo.

La solución que se plantea en este documento trate del diseño e implementación de un sistema que realice una estimación de este impacto. Para ello se aúnan distintas metodologías de análisis y diseño, con tal de encontrar una solución posible. El sistema diseñado finalmente hace uso de distintas tecnologías para implementar los componentes. Se hace uso de Arduino para el control de sensores que midan dicho impacto, conjuntamente con una aplicación en Android que recaba los datos recogidos por el microcontrolador, los analiza, y los almacena en una base de datos en la nube.

Finalmente, el sistema ha podido ser puesto a prueba para cerciorarse de que los objetivos planteados se cumplen satisfactoriamente.

**Palabras clave:** Acelerómetro, giroscopio, Arduino, par residual, ergonomía, PowerTool, sensores, fuerza de reacción.

# Abstract

Chain production processes and the automation of operations have enabled the large-scale production of all types of materials and devices for several decades. The automotive industry has been one of the major drivers of these automation techniques, enabling vehicles to be built and sold quickly and efficiently. Nowadays, every production process consists of a series of operations that go from screwing elements manually, to the automatic incorporation of elements using robotic arms. Most of these operations are made by operators with tools designed for this purpose.

In this production line, component tightening operations represent a large part of this process. In them, each of the steps to be followed by the operator is specified, indicating the screw to be placed, and configuring specific tools to accomplish these tightening operations according to design specifications. Despite having methodologies for monitoring and supervising those tightening processes, there are no specifications that determine a priori whether a tightening executed under specific conditions meets the minimum ergonomic requirements, that is, making sure it does not cause damage to the operator in the short or long term.

The solution proposed in this document outlines the design and implementation of a system capable of making an estimation of this impact. For this purpose, different analysis and design methodologies are combined in order to find a possible solution. The designed system makes use of different technologies to implement the components. An Arduino board is used to control sensors that trace this impact, together with an Android application that collects the data read by the microcontroller, analyzes them, and stores them in a cloud database.

Finally, the system has been tested to make sure that the objectives are accomplished satisfactorily.

**Keywords:** Accelerometer, gyroscope, Arduino, residual torque, ergonomy, PowerTool, sensors, reaction force.

# Resum

Els processos de producció en cadena i automatització d'operacions han permès la producció en gran quantitat de tot tipus de materials i dispositius des de fa ja dècades. L'industria automobilística ha sigut una de les grans impulsores d'aquestes tècniques d'automatització, permetent construir i posar a la venta vehicles de forma ràpida i eficient. A l'actualitat, tot procés de producció consta d'una sèrie d'operacions a realitzar, que comprenen des de la col·locació d'elements de forma manual, a la incorporació d'elements mitjançant braços robòtics automàticament. La majoria d'aquestes operacions són realitzades per operaris amb ferramentes dissenyades per a aquesta feina.

En una línia de producció, les operacions d'estrenya de components representen gran part d'aquest procés. En aquestes, s'especifica cada un dels passos a seguir per l'operari, indicant el caragol a col·locar, i configurant la ferramenta per a realitzar aquestes estrenyes segons les especificacions de disseny. A pesar de disposar de metodologies de monitorització i supervisió de les estrenyes realitzades, no existeixen especificacions concretes que determinen *a priori* si una estrenya realitzada en unes condiciones concretes compleix amb els requisits mínims d'ergonomia, es a dir, no causa malestar a l'operari a curt o llarg termini.

La solució que es planteja a aquest document tracte del disseny i implementació d'un sistema que realitza una estimació d'aquest impacte. Per a açò s'uneixen distintes metodologies d'anàlisi i disseny, amb l'objectiu de trobar una solució possible. El sistema dissenyat finalment fa ús de distintes tecnologies per a implementar els seus components. Es fa ús d'Arduino per al control de sensors que mesuren l'impacte, conjuntament amb una aplicació Android que recapta la informació recollida pel microcontrolador, l'analitza, i l'emmagatzema a una base de dades al núvol.

Finalment, el sistema ha pogut ser posat a proba per a cerciorar que els objectius plantejats es s'acompleixen satisfactòriament.

**Paraules clau:** Acceleròmetre, giroscopi, Arduino, par residual, ergonomia, PowerTool, sensors, força de reacció.

# Índice de contenidos

# Índice de figuras

# Índice de tablas

# 1. Introducción

La planta de ensamblaje de *Ford Almussafes* produce vehículos en cadena desde su apertura en 1976 (contributors, 2019). Esta planta de gran extensión tiene una característica peculiar que no muchas plantas del sector alrededor del mundo comparten; en su interior confluye la fabricación de varios modelos de vehículos de la marca (actualmente son cinco los modelos), cada uno con su abanico de configuraciones de asientos, ruedas y otros elementos.

Esta variedad de modelos dota a la línea de producción de una complejidad notable, ya que cualquier cambio en el diseño de los propios vehículos, en las instalaciones, o la incorporación de nuevas tecnologías propias del sector, implica adaptar la línea de producción para que siga produciendo como si ningún cambio se hubiese llevado a cabo, sin afectar a la producción de ningún modelo.
Dentro de Ford Almussafes encontramos a su vez distintas plantas. Esto es debido a que la producción de vehículos en cadena divide la fabricación del vehículo en una serie de procesos, cada uno de estos con requisitos de material, personal e instalaciones distintos.

Este trabajo ha sido concebido en la planta de *Manufacturing*, dónde se recibe el chasis del vehículo pintado en la planta de Pinturas, y se ensamblan todos los componentes del vehículo, dando como resultado el vehículo terminado y listo para su funcionamiento. Este proceso de ensamblaje se divide en un conjunto de operaciones que, en su mayoría, consisten en la incorporación de las distintas piezas al cuerpo del vehículo mediante aprietes, haciendo los operarios uso de *PowerTools* para estas tareas. Dichas herramientas de apriete deben ser configuradas correctamente para realizar la tarea deseada respetando las características del apriete y la ergonomía definida para la operación.

## 1.1 Motivación

Como se ha mencionado anteriormente, el uso de *PowerTools* está extendido en el proceso de producción, contando con una amplia variedad de herramientas que se ajustan a los requisitos de cada operación. Cada uno de los aprietes efectuados por las herramientas es categorizado previamente dependiendo de lo crítico que resulte para el vehículo, por lo que se cuenta con sistemas que monitorizan y comprueban que, en cada apriete, la herramienta entrega el par deseado a la velocidad correspondiente, teniendo los aprietes más críticos una monitorización más exhaustiva.

Otro aspecto de estas herramientas a tener en cuenta es el impacto que generan en el operario que las usa. La motivación de este trabajo surge de la necesidad de un sistema que se encargue de comprobar que el apriete que realiza una *PowerTool* cumple con la ergonomía deseada, evitando posibles daños en el operario que la maneja tanto a corto como a largo plazo.

Toda operación de la línea tiene asignada una hoja de proceso dónde se especifica cada uno de los pasos a realizar, describiendo con detalle los movimientos realizados por el operario junto con las piezas y las herramientas necesarias para cada uno de ellos. En las operaciones con *PowerTool* se sigue habitualmente un mismo procedimiento:

1. El operario coloca el tornillo en la boca de la herramienta, o bien en la ranura indicada.
2. Se ubica la herramienta en posición, y se activa el gatillo para empezar a apretar.
3. El cabezal de la herramienta gira para enroscar el tornillo entregando el par deseado a la velocidad indicada (configurados previamente en la herramienta).
4. Cuando el cabezal no puede girar más, la herramienta se bloquea y detiene el giro del cabezal. El bloqueo genera una reacción opuesta al sentido del apriete, que se transmite al operario que sujeta la herramienta.

Las condiciones en que el operario sujeta la herramienta varían en cada operación, ya que son especificadas en el momento de creación de la operación. La altura a la que se ubica la *PowerTool*, la distancia de esta con el operario, el par necesario para el apriete, etc. son factores que hacen que la reacción generada por el apriete antes mencionada afecte de forma distinta al operario, por lo que no se puede asegurar que una operación especificada dentro de los límites de la ergonomía nunca se desvíe respecto a este planteamiento inicial.

### 1.1.1 Motivación personal

Desde un punto de vista más personal, este proyecto forma parte de una idea que surgió desde el momento en que pensé en hacer prácticas curriculares en una empresa: no limitar mis objetivos a aquello que se requiere en el trabajo, sino hacer uso de las herramientas que la carrera me había proporcionado para ver si podía dejar mi huella. Por esta razón, tratar de cubrir una necesidad estrechamente ligada con la salud y las condiciones en que otras personas trabajan resulta un factor de peso.

## 1.2 Objetivos

Este proyecto pretende cubrir la necesidad expuesta anteriormente, diseñando un sistema que garantice que una *PowerTool* a la que se le ha asignado un apriete concreto, permitiendo discernir si dicha operación se va a realizar dentro de los límites ergonómicos deseados, y evitando así posibles problemas de salud al operario que maneje la herramienta. Para ello, se han definido los siguientes objetivos intermedios:

- Analizar el problema a solucionar.
- Identificar las necesidades para el diseño de una solución.
- Diseñar el sistema en base a la solución propuesta.
- Especificar la tecnología que compondrá el sistema.
- Implementar el sistema diseñado previamente.
- Comprobar el correcto funcionamiento de este.

## 1.3 Impacto esperado

Tras la implantación del sistema, el encargado de instalar la *PowerTool* en la estación pertinente podrá informarse con suficiente antelación acerca de la ergonomía de la operación. Esto hará posible proponer cambios en dicha operación, evitando posibles molestias u daños al operario que hace uso de la *PowerTool*, mejorando así las condiciones de ciertos trabajos de la producción.

## 1.4 Metodología

Para conseguir el cumplimiento de los objetivos propuestos, se plantea la siguiente metodología que estructurará el procedimiento a seguir para la creación del sistema.

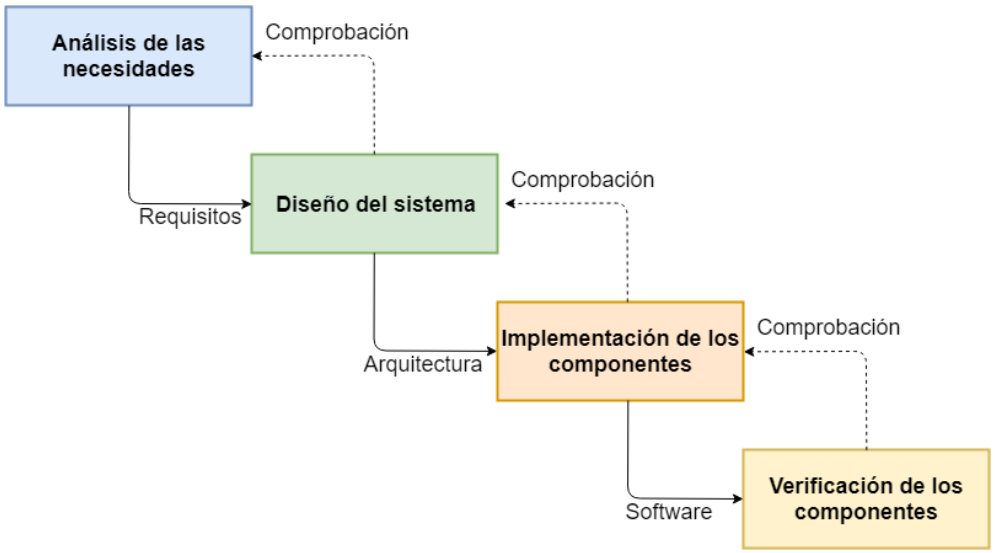


*Figura 1. Metodología en cascada.*

Como se puede observar en la Figura 1, se plantea una fase inicial donde se identifican los requisitos del sistema. La siguiente fase parte de estos requisitos para proponer un diseño que intente satisfacer todas las necesidades planteadas. Tras diseñar la estructura, se pasa a su implementación, usando la tecnología especificada anteriormente para ello. Finalmente se encuentra una última fase de verificación o *testing,* donde se comprueba que los objetivos y requisitos del sistema han sido cumplidos.

Este modelo de metodología secuencial se identifica como el modelo en cascada, y permite estructurar el proyecto de forma clara y concisa, estableciendo los requisitos en la fase inicial del proyecto. Al tratarse de un proyecto en el que los requisitos no cambian en gran medida a lo largo del proceso de desarrollo, dicha metodología se ve beneficiada en contraposición a otras como puede ser la metodología en espiral o el modelo incremental (Tinoco Gómez, Rosales López, & Salas Bacalla, 2010).

## 1.5 Estructura

La memoria de este proyecto adquiere la siguiente estructura:

- **Capítulo 1:** en este capítulo se hace una introducción a la problemática a resolver, planteando las motivaciones que definen los objetivos, y previendo el impacto del proyecto.
- **Capítulo 2:** este capítulo desarrolla las bases e investigaciones previas desde las que parte este proyecto, además de remarcar el ámbito del proyecto dentro de Ford Almussafes.
- **Capítulo 3:** en este capítulo se analiza el problema en profundidad, desgranando los requisitos que el sistema debe cumplir mediante distintas técnicas de análisis.
- **Capítulo 4:** en este capítulo se diseña la estructura del sistema y de cada uno de sus componentes, presentando las tecnologías que formarán cada componente.
- **Capítulo 5:** en este capítulo se explica cómo se han desarrollado los distintos módulos que componen el sistema para completar los objetivos propuestos.
- **Capítulo 6:** en este capítulo se introduce el sistema en el entorno final de uso, tras el desarrollo, realizando las comprobaciones necesarias para garantizar su correcto funcionamiento.
- **Capítulo 7:** en este capítulo se relacionan los objetivos inicialmente planteados, con aquellos que han sido completados; además, se plantean nuevas vías en las que el proyecto podría ser ampliado, descartando a su vez otras opciones.

## 1.6 Convenciones

La memoria contiene texto con diferente tipografía con la intención de dotarlo de un significado adicional.

- El código fuente se muestra en Monospac821 BT tamaño 10, empleando exclusivamente esta tipografía para este tipo de contenido.
- Las palabras extranjeras se marcarán en cursiva.
- Se entrecomillarán las citas textuales externas a obras.
- Los enlaces a páginas web serán subrayados y con texto de color azul.

# 2. Estado del arte

Al tratar de abordar los objetivos planteados para el proyecto, es necesario un proceso de investigación previo que trate de cerciorar la viabilidad no solo de los objetivos, sino del proyecto en sí. Además, resulta de vital importancia conocer las tecnologías y estudios previos ya realizados en el campo, ya que estos pueden llegar a ser el punto de partida para el desarrollo del sistema.

Para la realización de este proyecto es necesario conocer la función que realizan las *PowerTools* en la línea de producción, además de posibles investigaciones en la ergonomía de estas.
Para ello, se analiza el funcionamiento de las herramientas de apriete además de las distintas opciones disponibles hoy en día en el mercado.

## 2.1.1 Métricas de un apriete

Con tal de comprender el funcionamiento de las herramientas de apriete, se pretende analizar las fuerzas que intervienen al apretar un tornillo, así como las reacciones que dicho apriete provoca en el operario que maneja la herramienta.

El tornillo se somete a las siguientes tensiones durante el apriete (ver Figura 2):

- Tensión de rotura (*tensile load*): máxima tensión que un material puede soportar al ser estirado antes de fallar o romperse.
- Torsión: resultante de la fricción ente el tornillo y la tuerca.
- Tensión cortante (*shear load*): esfuerzo interno al tornillo causado por las tensiones paralelas a la sección transversal. Habitualmente es mitigada por la fricción entre los componentes si la fuerza de sujeción es suficiente.

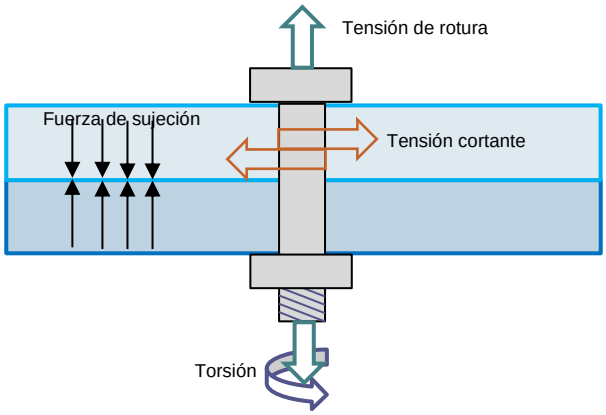


*Figura 2. Métricas del apriete.*

En las uniones se considera al tornillo como la parte más débil, por lo que el material y dimensiones de este deben ser los adecuados para entregar una fuerza de sujeción suficiente. En caso de no entregar la suficiente fuerza, la unión puede quedar suelta, exponiéndose a una posible ruptura. Pese a ser un factor de gran relevancia, no hay forma práctica de medir la fuerza de sujeción, por lo que es habitual usar el par de apriete en su lugar, con algunas consideraciones, ya que parte de la energía entregada en el par se pierde contrarrestando la fricción con la tuerca y otros elementos (AtlasCopco, 2015).

El par de apriete o momento de la fuerza, puede medirse de forma dinámica al realizar el apriete, o de forma estática una vez el apriete haya sido realizado. Como ya se ha mencionado anteriormente, en la producción de un vehículo hay aprietes considerados más críticos que otros, por lo que necesitan de un seguimiento minucioso que, por optimización del tiempo, se realiza de forma dinámica (tomando medidas en el momento en que se usa la herramienta), midiendo el par con el que se realiza, e incluso el ángulo recorrido en el apriete. Las herramientas destinadas a dichos aprietes incorporan un sensor, habitualmente ubicado cercano al motor (haciendo más compacta la herramienta) que mide de forma constante el par entregado. Así pues, la diferencia entre un mismo apriete realizado por una herramienta o realizado a mano es altamente notable (AtlasCopco, 2016).

Sin embargo, el uso de *PowerTools* plantea la siguiente problemática: una vez la herramienta detecta que se ha llegado al par deseado, el apriete finaliza, pero no de forma inmediata. La energía acumulada durante el apriete provoca una reacción en sentido opuesto al momento de la fuerza aplicada (ver Figura 3), resultando en un pequeño impulso que el operario que maneja la herramienta mitiga. Este impulso, se repite en cada apriete, pudiendo provocar daños en el operario (Aguado Ullate & Ursua Rubio, 2015).

*Figura 3. Reacción tras el apriete – Herramienta angular Atlas Copco.*

Una posible solución para mitigar esta reacción la encontramos en la velocidad de apriete. Actualmente existe herramientas que varían la velocidad angular de forma dinámica, ofreciendo una mayor velocidad al inicio del apriete, disminuyéndola paulatinamente conforme la resistencia que opone el tornillo aumenta junto al par necesario para seguir apretando. De esta forma, en la última etapa del apriete la velocidad es menor, disminuyendo la inercia de la herramienta, y provocando que la reacción antes mencionada sea menor.

Otra alternativa se encuentra en las herramientas atornilladoras de impacto, cuya forma de uso es igual a la de una atornilladora normal, y la diferencia radica en cómo este tipo de herramientas entrega el par de apriete. En una atornilladora normal la cabeza gira sin interrupciones, a una velocidad constante o variable. En las atornilladoras de impacto el motor se comporta de la misma forma, pero estas incorporan un embrague a la salida de este, que hace que en cada giro del motor la cabeza que atornilla se mueva solo una porción del giro, dejando de moverse hasta la siguiente vuelta del motor. De esta forma el giro se transmite “a golpes”, permitiendo

reducir al mínimo la inercia de las piezas en movimiento y, por consecuencia, la reacción que recibe el operario es casi nula.
Esto permite tener herramientas compactas que entreguen un par más alto que el de las atornilladoras normales. Sin embargo, no hay forma práctica de medir el par entregado por estas herramientas, por lo que la monitorización de aprietes en las atornilladoras de impacto no se puede realizar de forma precisa.

### *2.1.2* Ergonomía en *PowerTools*

El estudio de la ergonomía de *PowerTools* permite ver la relación entre el operario y el entorno en el que trabaja; diseñar operaciones ergonómicas permite mejorar la producción además de reducir costes, mejorando la calidad de trabajo de los operarios.

Los estudios ergonómicos definen las formas de uso más adecuadas para cada herramienta, incluyendo indicaciones sobre cómo mantener la muñeca recta, cómo reducir las fuerzas aplicadas sobre ésta, o el diámetro aconsejable que deben tener los mangos de las herramientas, son aspectos relevantes para reducir lesiones al usar *PowerTools*. En este proyecto es necesario centrar la atención en el par de reacción resultante en las operaciones de apriete de tornillos (ver Figura 4) (Atlas Copco, 2015).

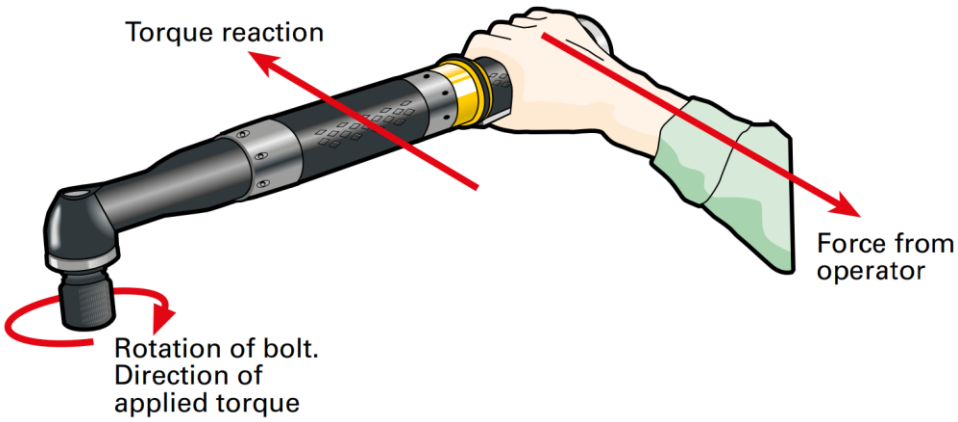


*Figura 4. Par de reacción - Herramienta angular Atlas Copco.*

La capacidad del operario para absorber la fuerza efectuada por el par de reacción depende de factores como la propia fuerza muscular del operario, la postura en que se realiza el trabajo, o la velocidad del apriete.
Según la normativa ISO 11148-6 (AENOR, 2013), que comprende las herramientas objetivo de este proyecto, se establecen los siguientes pares de apriete máximos:

- *PowerTools* de pistola: 10Nm (Newtons metro).
- *PowerTools* angulares: 60Nm (Newtons metro).

Sin embargo, este máximo varía en función de cómo se realiza cada operación, ya que pueden resultar valores demasiado altos como para ser absorbidos por el operario. Por esta razón, la normativa ergonómica no se puede aplicar por defecto a toda operación de apriete; no obstante, sí resulta de utilidad como punto de partida (Delgado Díaz & Bolufer Catalá, 2011).

## 2.2 Crítica al estado del arte

Las *PowerTools* son ampliamente usadas en el proceso de montaje de vehículos, y desde hace décadas los fabricantes siguen añadiendo mejoras al rendimiento y ergonomía de estas herramientas. Esto no impide que sigan surgiendo operaciones en el proceso de montaje que se encuentren en los límites de la ergonomía, situaciones difíciles de identificar con antelación; no es hasta más tarde (una vez el operario reporta posibles molestias), cuando se identifica la necesidad de hacer algún cambio.

Tampoco se incorpora un sistema que garantice la adecuación de las herramientas a las operaciones a las que han sido asignadas más allá de la capacidad que especifica el fabricante de la *PowerTool* para entregar un par de apriete y velocidad determinados. Por esta razón, lo que la ergonomía no siempre puede quedar cubierta.

# 3. Análisis del problema

Partiendo del problema planteado anteriormente, se desea implementar un sistema que evalúe la ergonomía de las diferentes herramientas de apriete. Dado que se quiere abordar el problema de la forma más completa posible, se va a presentar una estructura previa para el procedimiento de esta etapa de análisis. El esquema resultante es el siguiente:

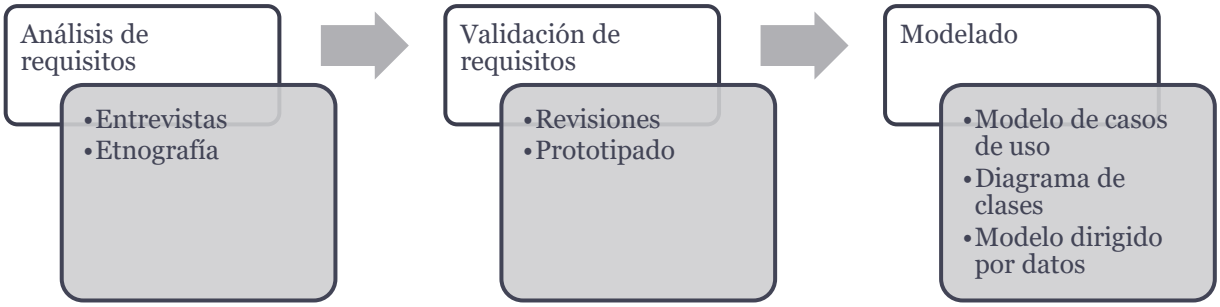


*Figura 5. Esquema de análisis del problema.*

Cabe destacar que esta primera etapa del proyecto, como es habitual en el desarrollo de software, se realiza de forma iterativa, ya que a medida que se analizan y validan los requisitos, es necesario hacer modificaciones con tal de añadir nuevos requisitos u modificar los actuales (Sommerville, 2011).

## 3.1 Análisis de requisitos

En esta fase, se pretende discernir cuáles van a ser las necesidades que el sistema va a tener que suplir. Para ello, se hace uso de distintas técnicas de análisis, que ayudarán a identificar los distintos tipos de requisitos posibles.

Estos requisitos no siempre se encuentran descritos con el mismo nivel de detalle por parte del usuario, por lo que es necesario clasificarlos con tal de evitar posibles problemas en fases posteriores (Ramos Cardozzo, 2016). La clasificación es la siguiente:

- ***Requisitos del usuario***: Son aquellas indicaciones que el usuario proporciona, y se caracterizan por describir qué servicios espera éste del sistema.
- ***Requisitos del sistema:*** Son descripciones más detalladas de las funciones y restricciones del sistema. A su vez, dentro de esta categoría, encontramos dos subcategorías:
  - ***Requisitos funcionales:*** Describen el funcionamiento del sistema en situaciones concretas.

- ***Requisitos no funcionales:*** Describen propiedades emergentes del sistema, es decir, no están relacionados directamente con las funcionalidades, por lo que afectan más a la descripción global del sistema que a las funciones concretas.

### 3.1.1 Entrevistas

La primera metodología usada para captar los requisitos son las entrevistas. Se ha elegido esta metodología por las facilidades que aporta respecto al entorno en que se desarrolla el proyecto. Dicho entorno requiere hacer uso de metodologías que no sustraigan mucho tiempo al usuario, debido a que se realiza en horario laboral; sumado a esto, las entrevistas aportan una visión sobre la idea preconcebida que el usuario pueda tener del sistema, definiendo posibles formas de interacción con este y restricciones aplicables.

Para la realización de las entrevistas, se han realizado preguntas aportando primero un contexto entorno a estas, con el objetivo de facilitar al entrevistado las respuestas; se evitan además preguntas demasiado amplias, ya que pueden llevar a respuestas no concluyentes. Con carácter general, las entrevistas han seguido una línea distendida e informal, mayormente debido a que los entrevistados eran compañeros de trabajo con los que se convive a diario; además, se ha combinado la rigidez de la entrevista cerrada (que optimiza la adquisición de información) con la flexibilidad de la entrevista abierta, evitando ideas o conceptos preconcebidos por parte del entrevistador (Zapata & Carmona, 2010).

Sin embargo, las entrevistas presentan dos contrapartidas, la primera viene dada si el entrevistado da por conocidos algunos conceptos u procedimientos, evitando comentarlos en la entrevista; por otra parte, el entrevistado puede usar lenguaje propio del trabajo que haga que los requisitos no se definan correctamente.

### 3.1.2 Etnografía

La segunda metodología usada es la etnografía. Esta técnica se basa en observar y analizar, tomando notas, el entorno en el que el sistema a desarrollar será ubicado finalmente. Para el correcto desarrollo del sistema, no solo es importante tener en cuenta los usuarios finales, sino también el entorno en el que será usado. En este proyecto el entorno se ubica en las diferentes estaciones que forman parte de la línea de producción. En base a observar cómo las *PowerTools* son usadas actualmente por los operarios, se definen requisitos y restricciones más allá de lo que los estándares de trabajo indican que sucede en la línea. Además, aunque cada operación sea definida de forma individual, en la misma estación suelen coincidir diversas operaciones a realizar por uno o más operarios; esto condiciona los movimientos a realizar, las posiciones y posturas de los operarios, y demás factores que desde la definición teórica no son observables.

Cabe destacar que esta metodología resulta más intrusiva que otras, por lo que puede llevar a variaciones en el comportamiento de las personas en su trabajo, aunque en este proyecto es poco probable, ya que cada operación es un conjunto de acciones que se realiza de forma repetitiva. Por otra parte, sí es importante destacar que la etnografía en sí misma no puede sustentar todo el análisis de requisitos, por lo que siempre es aconsejable combinarla con otras técnicas como bien se ha hecho, en este caso, con las entrevistas.

### 3.1.3 Requisitos obtenidos

Requisitos funcionales de proceso:

- El sistema permitirá analizar la ergonomía de las herramientas de la línea de producción actual.
- El sistema permitirá almacenar un listado de las herramientas analizadas y sus características.
- Al dar de alta una herramienta en el sistema, será necesario introducir el identificador que se usa en Ford para la herramienta.
- El identificador antes mencionado es único para cada herramienta.
- Entre las características adicionales de la herramienta deben encontrarse la línea y estación a las que pertenece.
- El sistema permitirá analizar nuevas herramientas que se necesiten incorporar a la línea.
- Todo análisis realizado debe estar asociado a una *PowerTool*.
- El sistema permitirá actualizar las características de cada herramienta analizada.
- El sistema permitirá volver a analizar una herramienta sin tener que crearla de nuevo.
- El sistema permitirá eliminar *PowerTools* dadas de alta previamente.
- El sistema permitirá visualizar toda la información en un smartphone o en un navegador web.
- El sistema debe permitir seleccionar qué herramienta va a usarse como referencia para el análisis.

Requisitos funcionales de interfaz gráfica:

- Al introducir el ID de una *PowerTool*, este solo podrá ser numérico.
- Las características adicionales a introducir deben permitir caracteres alfanuméricos.
- El sistema dispondrá de una ventana que muestre todas la *PowerTools* analizadas, indicando en cada entrada el resultado del análisis de su ergonomía.
- La ventana principal permitirá al usuario acceder al análisis de una nueva herramienta.

- El software dispondrá de una ventana donde visualizar todas las características de la herramienta.

Requisitos funcionales de la tecnología:

- El software podrá usarse en el sistema operativo Android.
- El software debe poder usarse sin requerir instalar software adicional.

Requisitos no funcionales:

- El usuario debe poder usar el sistema sin requerir la ayuda de otras personas.
- El sistema debe ser ligero, permitiendo a una persona analizar las herramientas en las propias estaciones.
- La interfaz gráfica debe ser sencilla, permitiendo comprender a primera vista las opciones disponibles.
- El tiempo requerido para analizar una herramienta debe ser inferior a 30 segundos.
- El sistema debe permitir almacenar un mínimo de 300 herramientas.
- La lista de herramientas analizadas y sus características debe estar actualizada en todo momento.
- El sistema a desarrollar no debe tener un coste superior a 50€.
- El tiempo de aprendizaje del usuario para usar el sistema será inferior a 1h.
- El sistema debe mostrar su estado al usuario en todo momento.
- El software no debe ocupar mucho espacio en la memoria interna del terminal donde se instale.

## 3.2 Validación de requisitos

Una vez definidos los requisitos, el siguiente paso consiste en validar su correcta definición. Este paso es de vital importancia, ya que un requisito mal definido o directamente sin definir puede conllevar cambios importantes en el diseño y la implementación del sistema, por lo que toda modificación que se haga en la fase actual será más sencilla y consumirá menos recursos (tiempo, personal, dinero, etc.) que en fases posteriores del proyecto.

Para la validación se van a analizar las siguientes características en los requisitos definidos:

- Validez: diferencia entre requisitos necesarios para el sistema y aquellos que suponen funciones adicionales.
- Consistencia: los requisitos no deben contradecirse entre sí, y tampoco debemos encontrar dos requisitos que representen la misma funcionalidad.
- Totalidad: debe haber requisitos definidos para toda funcionalidad o restricción del sistema.
- Realismo: los requisitos deben poder implementarse con la tecnología actual.
- Verificabilidad: para cada requisito debe ser posible describir pruebas que validen su cumplimiento en el sistema.

### 3.2.1 Revisiones

Esta metodología consiste en analizar cómo se ha definido cada uno de los requisitos, verificando que no existen requisitos duplicados o requisitos que se contradigan entre sí. La principal desventaja encontrada en esta metodología es que resulta demasiado simple debido a que no facilita encontrar nuevos requisitos; por otra parte, tampoco ayuda a explorar posibles modificaciones en los requisitos ya definidos.

### 3.2.2 Prototipado

El prototipado representa la principal metodología de validación de este proyecto. Debido a la naturaleza del entorno en el que se desarrolla el sistema, el prototipado aprovecha la relación cercana entre el desarrollador y los usuarios, permitiendo intercambiar opiniones con respecto al desarrollo del sistema. Esto ayuda a evitar malentendidos. Sumado a esto, los sistemas desarrollados con prototipado suelen satisfacer más al usuario debido a varios factores: por una parte, implican más al usuario en el desarrollo, lo preparan en el uso del sistema, y fortalecen la especificación de los requisitos.

Por otra parte, se han de tener en cuenta ciertos riesgos de esta metodología: el prototipado es un proceso iterativo que puede alargarse en las distintas fases del proyecto. Hay que tener en cuenta que no se deben alargar estas iteraciones más de lo necesario, además, puede incentivar a los usuarios a proponer modificaciones en exceso. Sumado a esto, se corre el riesgo de no tener en cuenta características del sistema que no puedan ser representadas con esta técnica, como la robustez o la seguridad.

El prototipo usado en este proyecto consta de las siguientes características (Maner, 1997):

- **Baja fidelidad:** prototipo tipo Mockup, realizado con papel y lápiz. Resulta económico y sencillo de realizar.
- **Exploratorio:** el objetivo del prototipo es identificar requisitos y analizar alternativas posibles en el diseño.
- **Horizontal:** modela características generales del sistema sin detallar aspectos concretos.
- **Global:** modela el sistema entero, o gran parte de sus características y funcionalidades.

El prototipo resultante es el producto de varias iteraciones realizadas durante el transcurso de las fases iniciales del proyecto (análisis de requisitos y diseño del sistema), por lo que hay aspectos de la interfaz que son especificados en apartados posteriores de esta memoria.

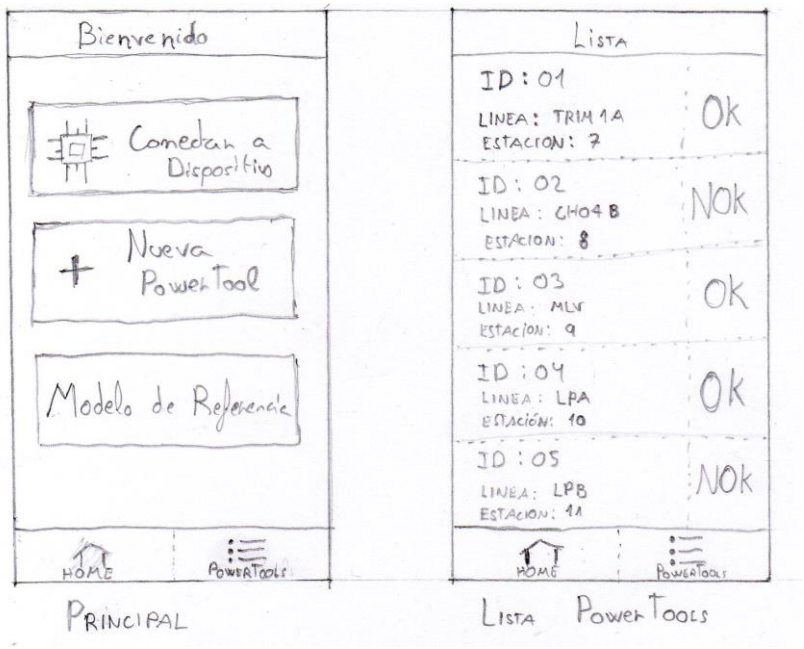


*Figura 6. Prototipo del menú principal (izquierda) y la lista de PowerTools (derecha).*

En la Figura 6 se observa un menú principal con diferentes funcionalidades, y una ventana donde visualizar las herramientas analizadas anteriormente.

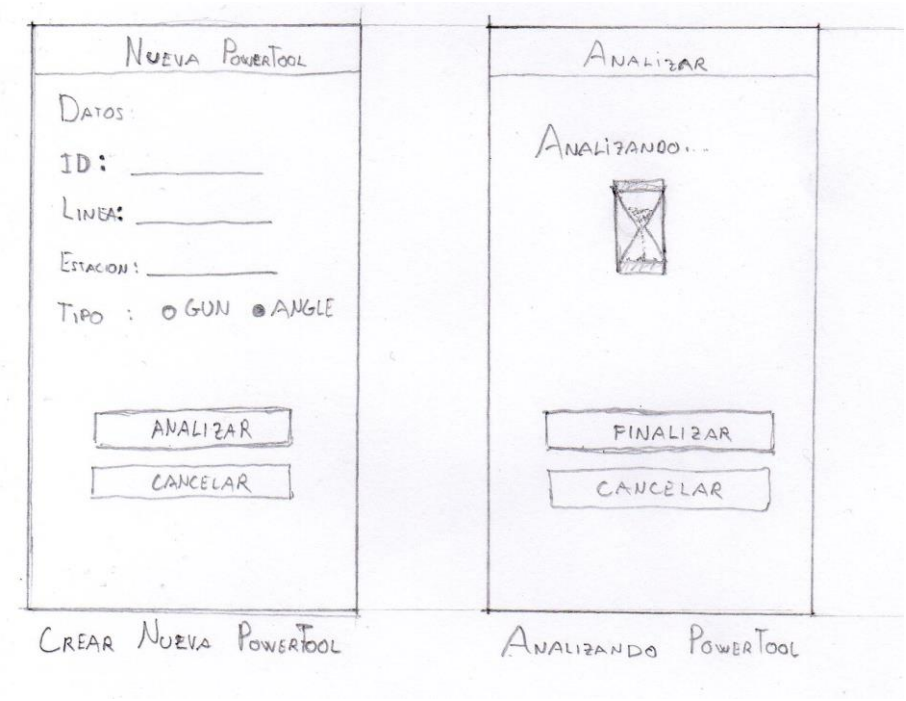


*Figura 7. Prototipo de creación (izquierda) y análisis de una Nueva PowerTool (derecha).*

En la Figura 7 se aprecia el proceso de dar de alta una nueva herramienta en el sistema para analizarla.

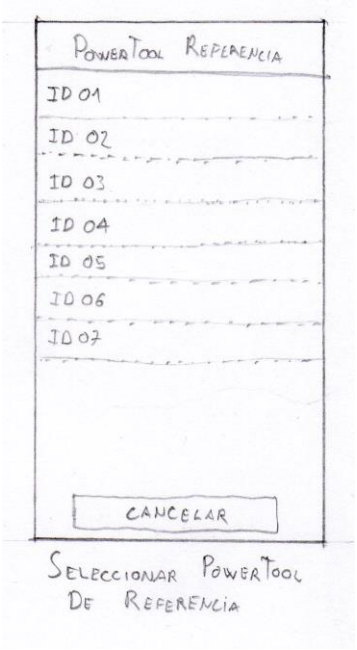


*Figura 8. Seleccionar PowerTool de referencia.*

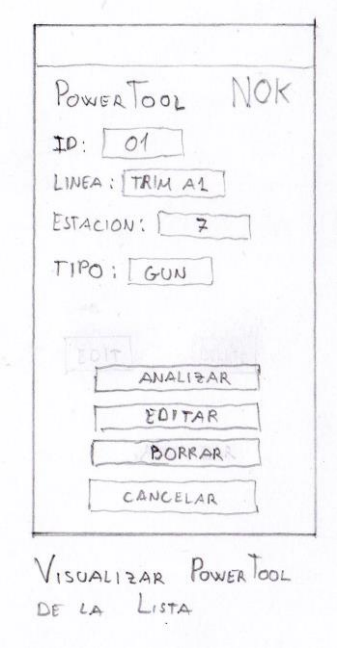


*Figura 9. Visualizar PowerTool de la lista.*

En las Figuras 8 y 9 se pueden observar dos ventanas secundarias que permiten seleccionar una herramienta como referencia para el análisis, y una ventana para visualizar una herramienta ya analizada, respectivamente.

## 3.3 Modelado del sistema

En esta fase se desarrollan distintas abstracciones del sistema que aportarán luz al sistema completo para así poder, en fases siguientes del proyecto, concretar las tecnologías a usar.

Para ello, se hace uso del lenguaje unificado de modelado (UML por sus siglas en inglés), ampliamente extendido en la actualidad. Este lenguaje hace uso de elementos gráficos para identificar y construir modelos de distintos aspectos de un mismo sistema software. La herramienta usada en este caso para la creación de los distintos diagramas es *Creately*, una herramienta disponible *online* sin necesidad de registrarse.

### 3.3.1 Modelo de casos de uso

El modelado de casos de uso ayuda a definir las interacciones entre los actores (en este proyecto son los usuarios) y el sistema de una forma sencilla. Cada caso de uso describe una tarea concreta, y para ello el diagrama se compone de distintos objetos:

- **Actor:** usuario que realiza una acción concreta.
- **Escenario:** instancia de un caso de uso en unas condiciones concretas.
- **Relación:** elemento que une un caso de uso con el actor, o con otro caso de uso.

Como se observa en el diagrama resultante (Figura 10), el sistema está compuesto por cuatro funcionalidades o casos de uso principales con las cuales el usuario puede interactuar inicialmente:

- El primero describe la funcionalidad de conectarse a un dispositivo de análisis.
- El segundo caso de uso permite al usuario dar de alta una nueva herramienta en el sistema y analizarla posteriormente.
- El tercer caso describe la visualización de la lista de herramientas analizadas en el sistema, pudiendo seleccionar cada herramienta para volver a analizarla, editar sus características, u eliminarla del sistema.
- El cuarto caso de uso permite al usuario seleccionar la *PowerTool* que hará de referencia al analizar las siguientes *PowerTools*.

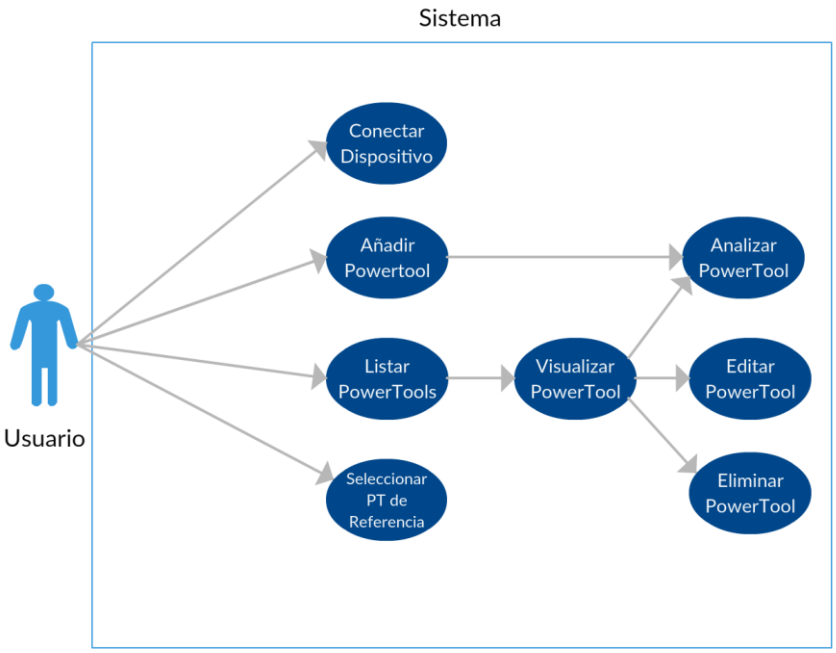


*Figura 10. Diagrama de casos de uso.*

Pese a ser un diagrama de apariencia sencilla, el diagrama de casos de uso ayuda a la adquisición de requisitos funcionales, además de proporcionar una base para el posterior desarrollo de los distintos módulos que compondrán el software (Lamar, 2005).

### 3.3.2 Diagrama de clases

La siguiente abstracción por realizar es el diagrama de clases. En este punto ya se empieza a esbozar la arquitectura del sistema, constituyendo los primeros componentes: los objetos. El sistema adquiere el modelo orientado a objetos, lo cual permite representar elementos de la realidad con objetos del modelo, donde cada clase representa un tipo de objeto del sistema. Esto ayuda a organizar los componentes del sistema en desarrollo.

Al tratarse de un modelo desarrollado en la etapa de análisis, los objetos que contiene son abstracciones del mundo real, y no es hasta llegar a fases más tardías del desarrollo cuando los objetos pueden representar elementos adicionales necesarios para el sistema, aunque pueda quedar fuera del ámbito de este proyecto.

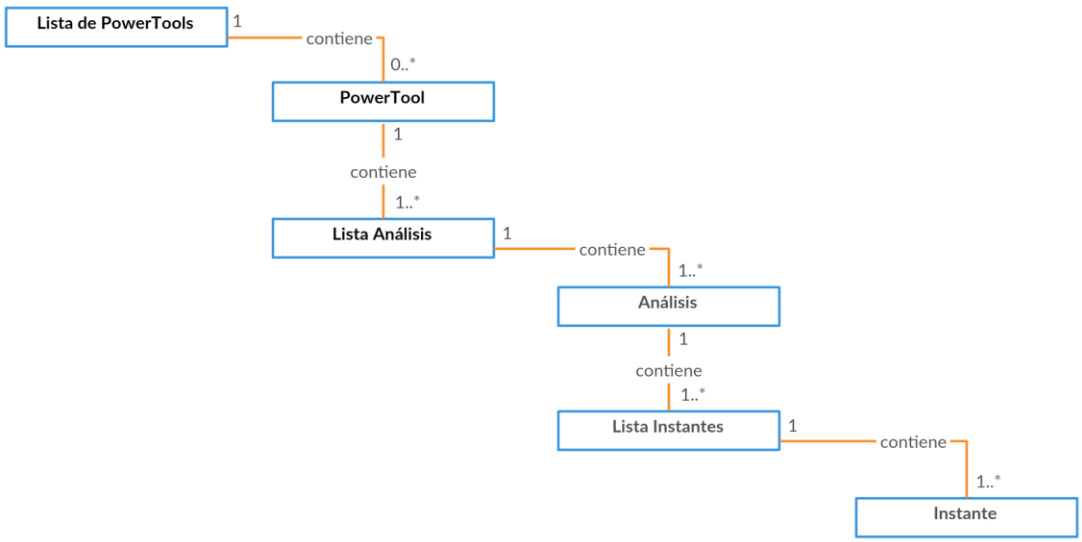


*Figura 11. Diagrama de clases.*

Como se observa en la Figura 11, el sistema almacenará una lista de *PowerTools*, cada *PowerTool* a su vez contiene una lista con los análisis realizados. Cada análisis se desgrana en una lista de instantes, que contienen el valor leído de cada sensor junto con la marca de tiempo del momento en que se ha creado (Debrauwer & Heyde, 2013).

### 3.3.3 Modelo dirigido por datos

En esta última etapa de modelaje se realiza una traza del flujo que siguen los datos a través del modelo de sistema con tal de esclarecer los diversos procesos que sufren. Este modelo apoya al análisis de requisitos debido a que se muestra el procesamiento de los datos desde un extremo a otro del sistema, identificando las funcionalidades necesarias para que el flujo de datos se lleve a cabo.

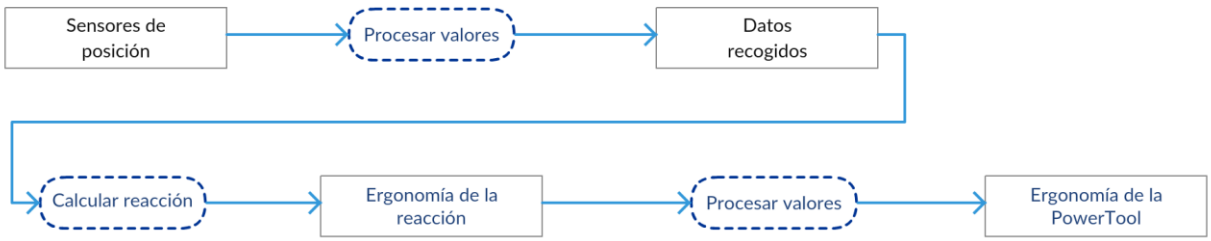


*Figura 12. Modelo dirigido por datos.*

En la Figura 12 se puede observar la secuencia seguida por los datos, desde la entrada de los sensores, su posterior procesamiento y la salida resultante determinando la ergonomía de la *PowerTool*.

## 3.4 Presupuesto

Uno de los requisitos de este proyecto es la implementación de un sistema que realice las funcionalidades deseadas, reduciendo además su coste de desarrollo. Para ello, se encuentran dos recursos principales:

- ***Recursos humanos:*** en este caso el coste del diseñador y desarrollador del sistema.
- ***Recursos materiales:*** *hardware* y *software* necesario para la realización del proyecto.

La extensión y dimensiones de este proyecto permite que una sola persona, que en este caso se trata del alumno que lo realiza, haga a la vez las funciones de diseñador y desarrollador, reduciendo así este coste a cero.
Por otra parte, los recursos materiales pueden adquirir cierto coste asumible por el mismo desarrollador, o por la empresa para la que se desarrolla el proyecto. Este coste deriva del *hardware* necesario, ya que se necesitan sensores para la recolección de información; éstos, a su vez, deberán ser controlados por microcontroladores. Por otra parte, debido a la gran cantidad de *software* de libre acceso disponible, éste no supone coste material alguno para el desarrollo del proyecto.
En apartados posteriores de esta memoria se discutirá la tecnología necesaria para el proyecto, y con ella el coste que implica.

# 4. Diseño del sistema

## 4.1 Diseño de la arquitectura

Tras el análisis de los requisitos, se pretende organizar la estructura global del sistema. Para ello, primero es necesario identificar los distintos componentes que lo forman, y las comunicaciones que estos mantienen.
Debido al tamaño de este proyecto, se presenta un modelo de abstracción del sistema llamado *arquitectura en pequeño*. Este tipo de abstracción se centra en la arquitectura de los componentes en que se separan los programas individuales del sistema, en contraposición con alternativas como la *arquitectura en grande*, orientada a sistemas formados por aplicaciones distribuidas y componentes más complejos.

La definición de la arquitectura del sistema es crítica, ya que tiene un impacto directo en factores como el rendimiento o su mantenibilidad. Además, influye directamente en el cumplimiento de los requisitos planteados anteriormente; en concreto, son los requisitos no funcionales los que se ven más afectados por un buen diseño de la arquitectura, por lo que se deben priorizar aquellos que resulten transversales a todo componente del sistema.

Dentro del *modelo en pequeño* antes mencionado, se pueden adoptar varios puntos de vista; en esta memoria se tratarán dos de ellos:

- ***Vista física:*** muestra los componentes *hardware* del sistema y las comunicaciones entre ellos.
- ***Arquitectura en capas:*** organiza los distintos componentes software en tres capas.

A continuación, se presenta un primer diagrama de la arquitectura del sistema, dónde se muestran los distintos elementos *hardware* que lo componen, además de las conexiones que estos establecen (ver Figura 13). En él, se aprecia un dispositivo de análisis formado por distintos sensores, y un microcontrolador que los controla. Este se comunica directamente con un *smartphone* para transmitir la información leída por los sensores. Por otra parte, se observa un servicio en la nube que almacena las lecturas de los sensores, y que permite visualizar dichas lecturas en el propio *smartphone*.

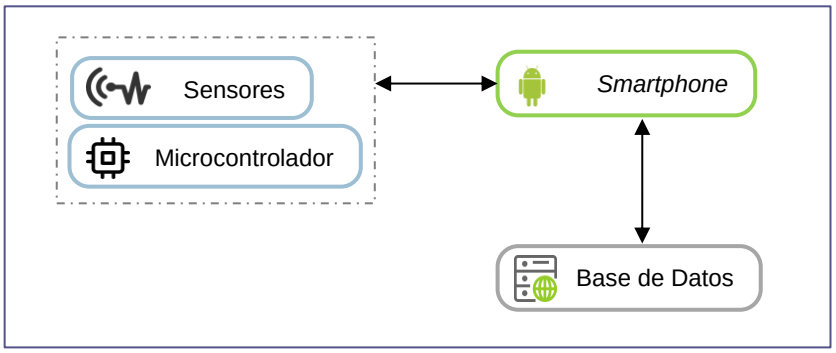


*Figura 13. Modelo Físico.*

El siguiente diagrama se basa en el modelo de arquitectura en tres capas:

- ***Capa de presentación:*** Esta capa engloba los distintos componentes software que interactúan con el usuario, como podrían ser las interfaces gráficas.
- ***Capa lógica:*** Esta capa contiene los componentes que dotan al sistema de sus funcionalidades.
- ***Capad de persistencia:*** Esta capa contiene los datos almacenados por el sistema, y la estructura en que éstos se organizan.

Además, como se observa en la Figura 14, en la capa lógica del dispositivo de análisis se incluye una subcapa de comunicación encargada de las comunicaciones con la capa lógica del smartphone.

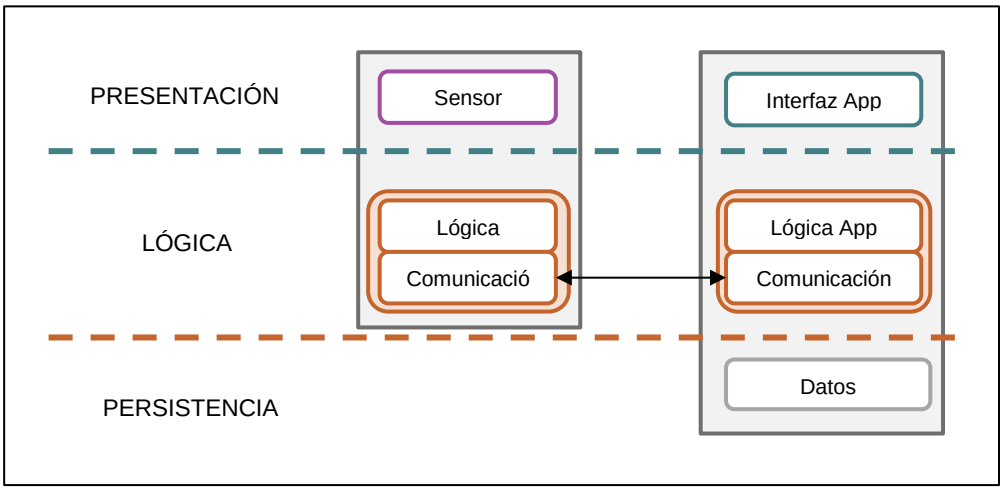


*Figura 14. Arquitectura en capas.*

Los principales beneficios que este tipo de arquitectura aporta son, por una parte, encapsular los distintos componentes del sistema con el objetivo de poder intercambiarlos libremente, facilitando la mantenibilidad y la inclusión de posibles mejoras. Por otra parte, esta independencia permite desarrollar incrementalmente las distintas capas, debido a que las necesidades de cada capa recaen en la capa inmediatamente inferior.

## 4.2 Diseño del dispositivo de análisis

El dispositivo de análisis está compuesto por los sensores encargados de medir la reacción de la herramienta en el operario, y el microcontrolador, encargado de captar las lecturas de dichos sensores, procesarlas y enviarlas al dispositivo móvil donde serán analizadas.

### 4.2.1 Sensores

Los sensores escogidos para medir la reacción son un acelerómetro y un giroscopio, mediante los cuales se pretende hacer una traza del movimiento experimentado por el operario usando la herramienta en cuestión.

El acelerómetro es un sensor que mide la aceleración experimentada por un determinado cuerpo, habitualmente compuesta por la aceleración causada por la fuerza de la gravedad y la aceleración debido a otras fuerzas externas. Hoy en día se puede encontrar este tipo de sensores incorporados en multitud de vehículos y dispositivos electrónicos distintos debido principalmente a su miniaturización mediante la tecnología MEMS (*Microelectromechanical Systems*), que permite reducir su tamaño considerablemente (Barkely Graham, 2000).

Actualmente existen diversos tipos de acelerómetros, en términos de ejes o grados de libertad se encuentran de eje único, biaxiales y triaxiales; como su nombre indica, cada tipo de acelerómetro permite medir la aceleración en uno dos o tres ejes, respectivamente. La principal ventaja que aporta el disponer de más ejes reside en la precisión de la medición del movimiento, ya que los acelerómetros de uno y dos ejes pueden ser invertidos de posición sin detectar dicho cambio al no disponer de los tres ejes.

Por otra parte, los acelerómetros se pueden clasificar en función de su mecanismo interno:

- ***Piezoresistivos:*** aprovechan la capacidad de ciertos materiales para variar su resistencia eléctrica al verse sometidos a fuerzas mecánicas.

- ***Piezoeléctricos:*** basados en el efecto piezoeléctrico de algunos materiales. Cuando se produce una aceleración, el material piezoeléctrico es presionado por una masa, generando una corriente eléctrica proporcional a la fuerza realizada por la masa.

- ***Mecánicos:*** hacen uso de una masa inerte en su interior, unida a un dinamómetro orientado en la dirección en que se desea medir. El dinamómetro se encarga de medir la deformación en la masa causada por la aceleración.

- ***Capacitivos:*** compuestos por un microcondensador sensible al desplazamiento producido por la aceleración. Dos placas paralelas separadas por un elemento no conductor forman la estructura del microcondensador; una de estas placas se mantiene fija, mientras que la otra se desplaza al verse afectada por el movimiento exterior, provocando una variación en la señal eléctrica.

Independientemente del mecanismo interno que forme el acelerómetro, éste se puede representar usando un mismo esquema: una masa inerte unida a un mecanismo que le permite desplazarse, por lo que habitualmente se representa con los principios físicos de la Ley de la elasticidad de Hooke y la Segunda Ley de Newton (ver Figura 15).

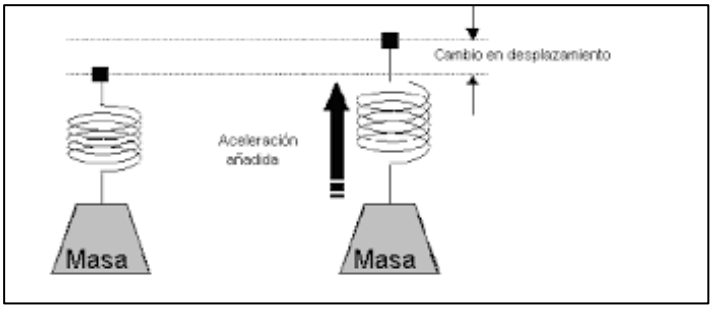


*Figura 15. Esquema de acelerómetro.*
*Fuente: Sensor medidor de Aceleración - Diseño Integrado - Universidad de Sevilla.*

El primer principio expresa la relación entre el alargamiento del material elástico en base a la fuerza aplicada sobre el mismo (este material elástico es el mecanismo que permite a la masa desplazarse):

$$F = k \times x$$

***F*** es la fuerza aplicada, ***k*** la constante de elasticidad del material y ***x*** el desplazamiento sufrido. El segundo principio relaciona la aceleración adquirida por un cuerpo con la fuerza aplicada sobre el mismo:

$$F = m \times a$$

Por lo que, igualando las dos expresiones, la aceleración se calcularía midiendo el desplazamiento de la masa. En la práctica se mide la elongación del material elástico en su lugar.

La elección de la tecnología que implementa el acelerómetro condiciona en gran medida factores como la máxima frecuencia a la que puede trabajar, o los márgenes de valores de la aceleración que admite (y con ello, la precisión). A continuación se proporciona una tabla comparativa entre los diferentes tipos de acelerómetros y sus características:

*Tabla 1. Tabla comparativa de acelerómetros.*

| Tipo | Marge de medida | Ancho de banda | Ventajas e Inconvenientes |
|---|---|---|---|
| MEMS | 1.5G – 250G | 0.1 - 1500 | - Alta sensibilidad<br>- Coste Medio<br>- Uso sencillo<br>- Bajas temperaturas |
| Piezoeléctrico | 0G – 2000G | 10 - 2000 | - Sensibilidad media<br>- Bajas temperaturas |
| Piezoresistivos | 0G – 2000G | 0 - 10000 | - Prestaciones medias<br>- Bajo coste<br>- Alta sensibilidad<br>- Tamaño y peso mínimos |
| Capacitivos | 0G - 1000G | 0 - 2000 | - Baja potencia<br>- Bajo coste |
| Mecánicos | 0G – 200G | 0 - 1000 | - Alta precisión<br>- Alto coste |

El giroscopio es un sensor que mide la variación del ángulo en función del tiempo o velocidad angular, por lo que resulta de gran utilidad para la traza de la orientación en la cual se realiza un movimiento. Al igual que los acelerómetros, la miniaturización de los giroscopios (gracias a la tecnología MEMS) ha permitido su extensión a ámbitos más comerciales (Villaluenga Morán, 2015).

En cuanto a sus características, se pueden encontrar giroscopios de uno, dos y tres ejes (al igual que ocurre con los acelerómetros). Además, dependiendo de la tecnología en la que se basan, se encuentran los siguientes tipos:

- ***MEMS:*** en su interior reside una masa que, al girar el giroscopio, se desplaza con los cambios en la velocidad angular. El movimiento resultante emite señales eléctricas que luego son interpretadas.

- ***Ópticos:*** hacen uso del efecto Sagnac para detectar la rotación; dos rayos de luz circulan en dirección opuesta en un circuito cerrado; el haz que circula en la dirección de la rotación se observa desde fuera que va a menor velocidad, por lo que el ángulo de giro es la diferencia entre los dos haces.

- ***Vibratorios:*** compuestos por un elemento vibrante que, al ser influido por una rotación, genera vibraciones secundarias de las cuales se extrapola la velocidad angular.

Cada una de estas tecnologías condiciona las características del giroscopio; cabe indicar que, cuanto mayor sea el rango de medición de la velocidad angular, menor será la precisión del giroscopio. El ámbito de este proyecto se va a limitar exclusivamente al uso de giroscopios MEMS (Figura 17), debido a su fácil integración en dispositivos de tamaño reducido, lo cual permite una mejor combinación con el acelerómetro.

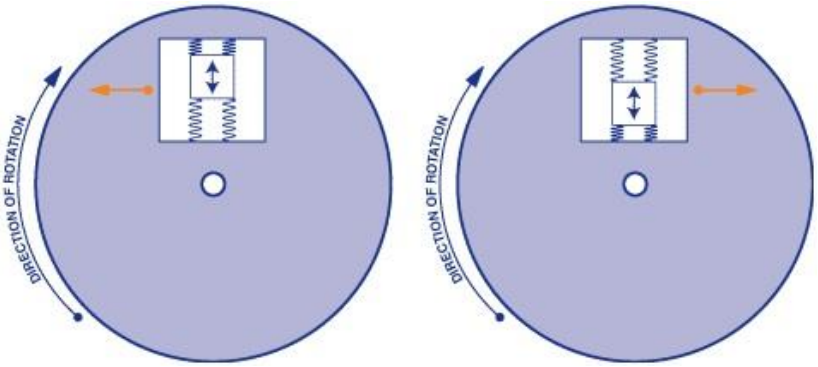


*Figura 16. Giroscopio MEMS.*
*Fuente: 5Hertz Electrónica.*

Aunque hasta ahora se ha hablado de ambos sensores como dos unidades independientes, aunque es habitual encontrarse componentes electrónicos que integran un acelerómetro y un giroscopio en una misma unidad, junto a otros sensores. Estos dispositivos electrónicos son las unidades de medición inercial, o IMU por sus siglas en inglés (*Internal Measurement Unit*), y permiten obtener el movimiento y la orientación de un cuerpo (ver Figura 18). Habitualmente están formadas por los siguientes componentes:

- Acelerómetro.
- Giroscopio.
- Magnetómetro: mide la fuerza del campo magnético que le afecta respecto al campo terrestre.
- Microprocesador: encargado de recabar las lecturas de los sensores.

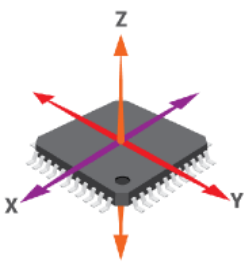


*Figura 17. IMU.*
*Fuente: Wikipedia.*

Las IMUs disponibles actualmente varían en función de las características de los sensores que incorporan (sensibilidad, velocidad de muestreo, etc.) y, al igual que ocurre con los giroscopios, a mayor rango de valores posibles, menor sensibilidad tendrán sus sensores. Además, otro condicionante que se encuentra en las IMUs es la velocidad en que el microprocesador recaba los datos de los sensores, es decir, el tiempo que transcurre entre la lectura del primer sensor, hasta el envío del último dato de la lectura del último sensor.

Para el desarrollo de este proyecto se ha usado la IMU Bosch BMI160 (Bosch, 2019), una unidad de bajo consumo que dispone de un acelerómetro y un giroscopio, ambos de tres ejes de 16 bits. A continuación se detallan las características

| Parameter | Technical data |
|---|---|
| Digital resolution | Accelerometer (A): 16 bit<br>Gyroscope (G): 16bit |
| Measurement ranges (programmable) | (A): ± 2 g, ± 4 g, ± 8 g, ± 16 g<br>(G): ± 125°/s, ± 250°/s, ± 500°/s, ± 1000°/s, ± 2000°/s |
| Sensitivity (calibrated) | (A): ±2g: 16384LSB/g<br>±4g: 8192LSB/g<br>±8g: 4096LSB/g<br>±16g: 2048LSB/g<br>(G): ±125°/s: 262.4 LSB/°/s<br>±250°/s: 131.2 LSB/°/s<br>±500°/s: 65.6 LSB/°/s<br>±1000°/s: 32.8 LSB/°/s<br>±2000°/s: 16.4 LSB/°/s |
| Zero offset (typ.): | A: ±40mg; G: ±3°/s |
| Noise density (typ.) | (A): 180 µg/√Hz<br>(G): 0.008 °/s/√Hz |
| Bandwidths (programmable) | 1600 Hz ... 25/32 Hz |
| Digital inputs/outputs | SPI, I²C, 4x digital interrupts |
| Supply voltage (VDD) | 1.71 ... 3.6 V |
| I/0 supply voltage (VDDIO) | 1.2 ... 3.6 V |
| Temperature range | -40 ... +85°C |
| Current consumption<br>- full operation<br>- low-power mode | 950 µA<br>3 µA |
| FIFO data buffer | 1024 byte |
| LGA package | 2.5 × 3.0 × 0.8 mm³ |
| Shock resistance | 10,000 g x 200 µs |

*Figura 18. Características de Bosch BMI160.*

completas:

Entre las características presentes en la hoja de especificaciones proporcionada por el fabricante (ver Figura 19), se puede observar que ambos sensores tienen rangos de valores programables, lo cuál permitirá ajustar la sensibilidad de los datos recabados.

Además, el fabricante especifica el parámetro offset para cada sensor, siendo de ±40mG para el acelerómetro, y de ±3º/s para el giroscopio. Al observar las lecturas que ambos sensores proporcionan cuando no hay ninguna clase de movimiento, es decir, cuando el valor a mostrar es cero, es habitual encontrarse que el sensor muestra valores cercanos a esta cifra, pero con una cierta desviación. Esto es debido a que las propiedades físicas del sensor varían con el tiempo de uso. Por ello, los fabricantes proporcionan el valor de este desvío medido para cada tipo de sensor, ya que cada sensor tiene su *offset* característico. La solución aplicable a este problema es la recalibración de los sensores con cierta frecuencia para evitar que el desvío sea considerable.

Con esta IMU se pretende calcular la posición relativa del dispositivo que realiza el análisis. Partiendo de la lectura que muestra el acelerómetro (la aceleración), se integramos ésta para así obtener la velocidad en cada uno de los ejes:

$$a = \frac{dv}{dt}$$

$$v = \int a(t)$$

A partir de la velocidad, se realiza una segunda integración para conseguir la posición relativa a la que se encuentra el cuerpo:

$$v = \frac{dp}{dt}$$

$$p = \int v(t)$$

Si bien puede parecer sencillo, se debe tener en cuenta que los sensores tienen errores de medida (como el *offset* antes mencionado), algunos más sencillos de contrarrestar, y otros imposibles a efectos prácticos. El integrar doblemente una medida con una pequeña desviación hace que el cálculo de la posición se desvíe substancialmente, desviación que se hace aún mayor conforme avanza el tiempo. La solución a aplicar a este problema se basa en un algoritmo que rectifique esta desviación; de entre los algoritmos más usados se encuentra el *filtro de Kalman*, del cuál se detalla su funcionamiento a continuación. Se ha optado por este algoritmo dada su precisión respecto a otras opciones.

### 4.2.2 Filtro de la señal

Como se ha indicado en el apartado anterior, se hace uso de un *filtro de Kalman* para corregir posibles desviaciones en las mediciones de la posición del cuerpo. Dicho filtro adquiere su nombre del ingeniero que lo creó, Rudolf E. Kalman; partiendo de las condiciones iniciales del sistema (posición, aceleración, velocidad, orientación, etc.) y de las mediciones de la IMU, se calcula el nuevo estado del sistema en base a los estados anteriores (recursividad).

El filtro está formado por tres etapas (ver Figura 19):

1) ***Cálculo de la ganancia:*** la ganancia es una variable cuyo valor oscila entre cero y uno, dando mayor o menor peso al valor estimado por el filtro sobre el valor medido por el sensor. Se calcula en base al error de la medida y el error de la predicción.
   Inicialmente la ganancia adquiere un valor de uno, dando más peso al valor medido por el sensor; conforme avanza el tiempo, este valor disminuye, dotando de un mayor peso al valor calculado por el filtro.
   Una ventaja de este filtro es la velocidad con la que la ganancia tiende a cero, ya que se necesitan pocas iteraciones para que la estimación gane más confianza.

2) ***Estimación del estado actual:*** tras calcular la ganancia, se estima el estado actual del sistema (tal y como se ha indicado antes), según el valor de ésta.

3) ***Error de la estimación:*** dicho error representa la proximidad entre el estado estimado por el filtro y el estado leído por el sensor. Si la ganancia tiende a cero, significa que el filtro confía más en su predicción del estado, por lo que el error es menor. Por el contrario, si la ganancia tiende a uno, el error en la estimación es mayor.

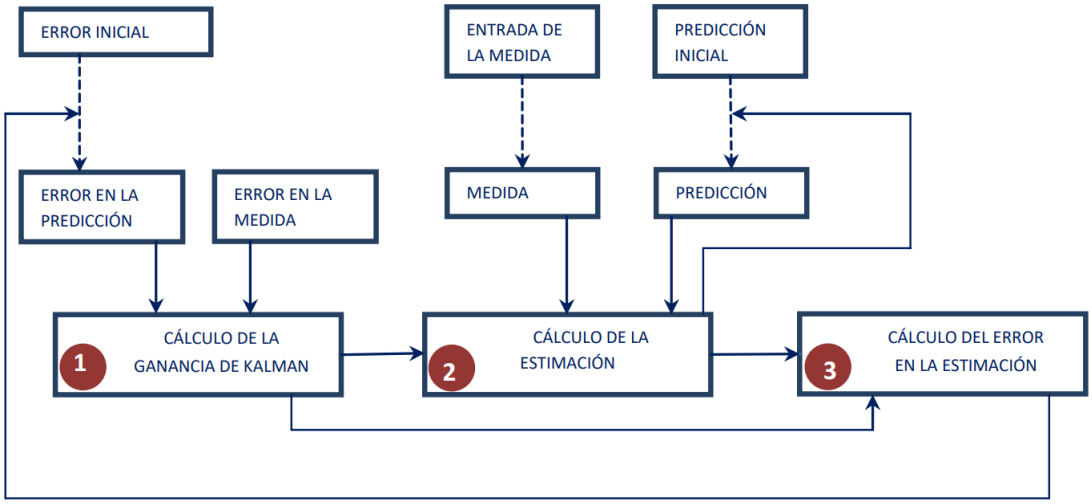


*Figura 19. Esquema del filtro de Kalman.*

El *filtro de Kalman* representa el sistema con el valor observado en la medida, y el estado actual del sistema:

$$z_k = Hx_k + v_k \qquad x_k = Ax_{k-1} + Bu_k + w_{k-1}$$

***z*** = valor observado.
***k*** = tiempo.
***x*** = estado del sistema.
***v*** = ruido de la medición.
***u*** = entrada del sistema.
***w*** = ruido del proceso.
***A*** = matriz de relación entre estado actual ***k*** y anterior ***k-1***
***B*** = matriz de relación entre ***u*** y ***x***.

El algoritmo parte de esta definición de sistema para aplicar cada una de las etapas mencionadas anteriormente. Habitualmente las etapas de *cálculo de la ganancia* y *estimación del estado actual* se representan como una sola etapa llamada predicción. Así pues, a continuación se expresa el algoritmo con dos fases definidas: *predicción* y *corrección*.

- ***Predicción***

$$x_k = Ax_{k-1} + Bu_k$$

$$P_k = AP_{k-1}A^T + Q$$

Se presenta una estimación del error actual respecto al estado anterior, además de una predicción del estado actual del sistema en base al estado anterior.

- ***Corrección***

$$K_k = P_k - H^T(HP_k - H^T + R)^{-1}$$

$$x_k = x_k + K_k\,(z_k - Hx_k)$$

$$P_k = (I - K_kH)P_k$$

En este conjunto de operaciones, el filtro corrige y se retroalimenta para conseguir resultados más precisos.
El filtro recalcula la ganancia de Kalman. Seguidamente se vuelve a calcular el estado en base a la medición del sensor y, tras esto, se recalcula la matriz que representa la varianza del error de la estimación.

Como se ha podido observar en la descripción anterior del filtro, hay dos matrices representadas que no son descritas: ***Q*** y ***R***. La matriz ***Q*** representa la dispersión de las diferencias de los valores predichos, es decir, indica cómo de acertadas han sido las predicciones de los valores respecto a la media de las iteraciones anteriores. Por otra parte, ***R*** representa la dispersión de los valores medidos, esto es, cómo de cercanos han estado los valores medidos respecto a la media de todas las iteraciones. Estos valores son usados por el filtro para que, en caso de que en la iteración actual se prediga un valor demasiado desviado de la tendencia, el resultado de esta predicción se interprete como erróneo, dotando de un mayor peso a la medición del sensor en contraposición a la predicción (Tereshkov, 2013).

### 4.2.3 Microcontrolador

Para la elaboración del dispositivo de medición se requiere de un microcontrolador que dé soporte a los sensores anteriormente descritos, y que además deberá ser capaz de incorporar la conectividad suficiente para transmitir los datos recabados a un smartphone. De entre los microcontroladores disponibles en el mercado, se ha decidido elegir un Arduino 101 para la implementación del dispositivo de análisis (Arduino, 2019). A continuación se exponen las razones principales de esta elección:

- ***Comunidad:*** la plataforma Arduino consta de una comunidad de soporte al desarrollo bastante amplia, con multitud de proyectos realizados y documentación disponible *online*.

- ***Sensores:*** la gama de dispositivos Arduino es compatible con una variedad muy amplia de sensores, debido principalmente a la capacidad de leer valores analógicos de todo tipo.

- ***Programación:*** la programación de los dispositivos Arduino se realiza en el lenguaje *Processing*, derivado de C++, lenguaje ampliamente conocido.

- ***Módulos:*** a parte de la gama de sensores, se encuentran módulos que funcionan como extensiones del dispositivo Arduino que añaden nuevas funcionalidades al mismo. Estos módulos suelen contar, además, con sus propias librerías con las que se facilita la programación de éstos.

- ***Precio:*** los dispositivos Arduino suelen ofrecer un rango amplio de precios, dependiendo de las funcionalidades que incorporen. Se pueden encontrar modelos de entre 10 y 50 euros.

El modelo Arduino101 es una derivación peculiar del Arduino UNO, ya que incorpora diversas tecnologías que lo adecúan en mayor medida a un entorno industrial. La diferencia principal del modelo 101 reside en el microcontrolador de la placa, habitualmente este procesador se basa en la arquitectura AVR (diseño de CPU tipo RISC), y en cambio este incorpora un módulo que integra el procesador y varios componentes en un mismo chip desarrollado por Intel, el cual recibe el nombre de *Intel Curie*.

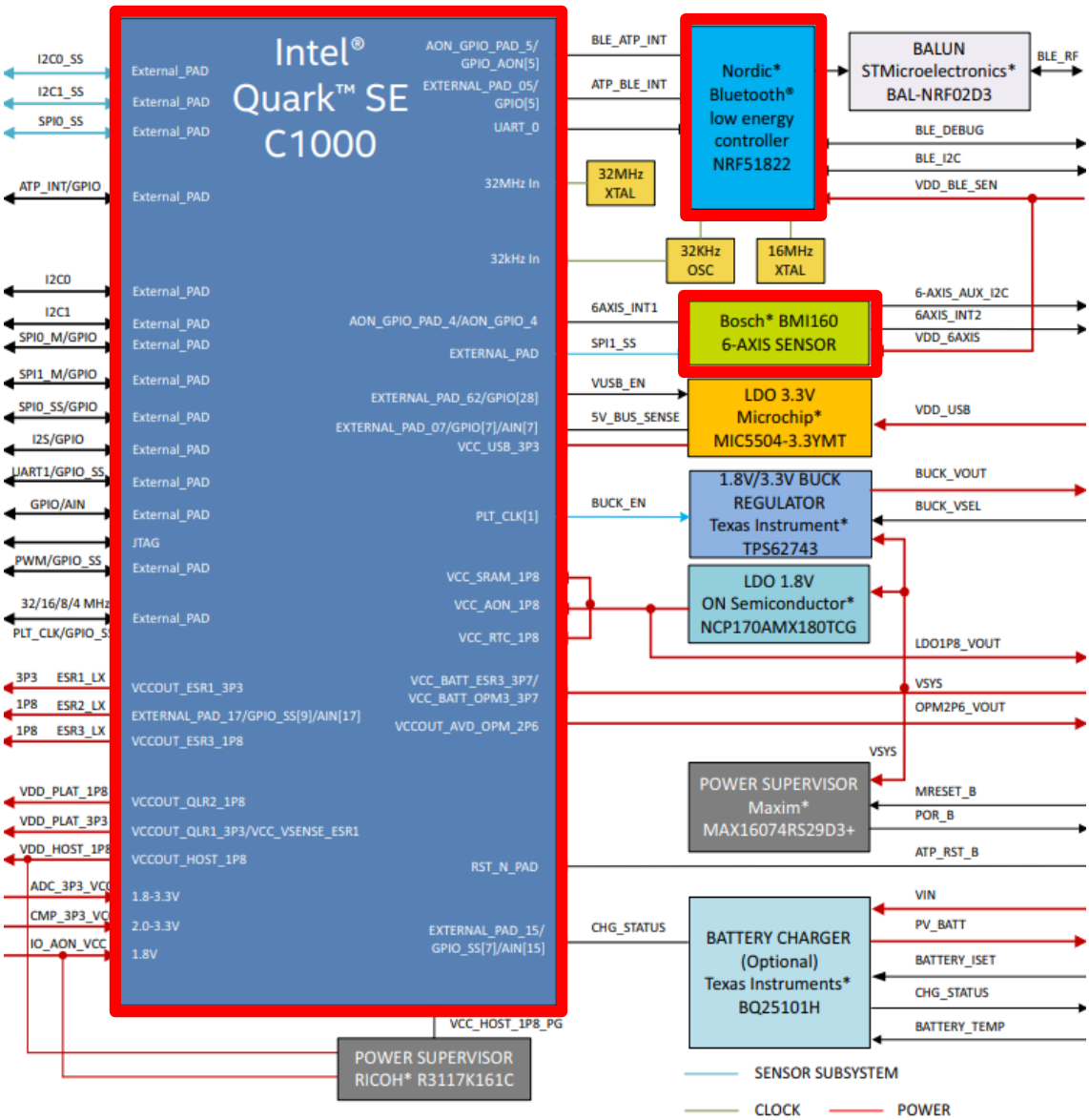


*Figura 20. Esquema del microchip Intel Curie.*
*Fuente: Intel.com.*

Como se puede observar en el esquema, el chip *Intel Curie* incorpora un procesador Quark SE C1000 y diversos módulos con funcionalidades distintas; de entre todos ellos, hay dos que resultan de especial interés para el desarrollo de este proyecto: el módulo *Bluetooth Low Energy*, y una IMU. A continuación se proporciona una descripción más detallada de dichos módulos:

- ***Intel Quark SE C1000:*** se trata del procesador del dispositivo, más potente que el procesador ATmega328 que suelen incorporar los modelos de Arduino. A continuación se puede ver una tabla comparativa entre ambos procesadores (Intel Corporation, 2017).

*Tabla 2. Comparativa entre Intel Quark SE C1000 y ATmega328.*

| Característica | Intel Quark SE C1000 | ATmega328 |
|---|---|---|
| CPU | 32bit Low Energy Pentium | 8bit AVR |
| Memoria *Flash* | 196KB | 32KB |
| SRAM | 24KB | 2KB |
| EEPROM | 8KB | 1KB |
| *Clock Speed* | 32 MHz | 16 MHz |
| Voltaje | 3.3V | 5 V |

- ***IMU:*** se trata de la Bosch BMI160, de la cual se ha hablado en apartados anteriores.

- ***Bluetooth Low Energy:*** este módulo basado en el chip nRF51822 (Semiconductor, 2019)dota a la placa de conectividad BLE. Esta tecnología resulta de gran utilidad para la conectividad con otros dispositivos. Algunas de las características de este chip son:
    - Bluetooth 2.4 GHz
    - CPU *ARM Cortex* de 32 bit
    - 128 KB memoria *flash*
    - 16 KB memoria RAM

Aunque el chip Intel Curie dote a la placa Arduino 101 de distintas funcionalidades, el diseño de la misma sigue siendo muy similar al de una paca Arduino UNO. Como se puede observar en la Figura 21, la placa conserva los mismos *pins* de lectura/escritura analógica y digital, además del puerto USB para la conexión Serial, y un puerto para la corriente de alimentación.

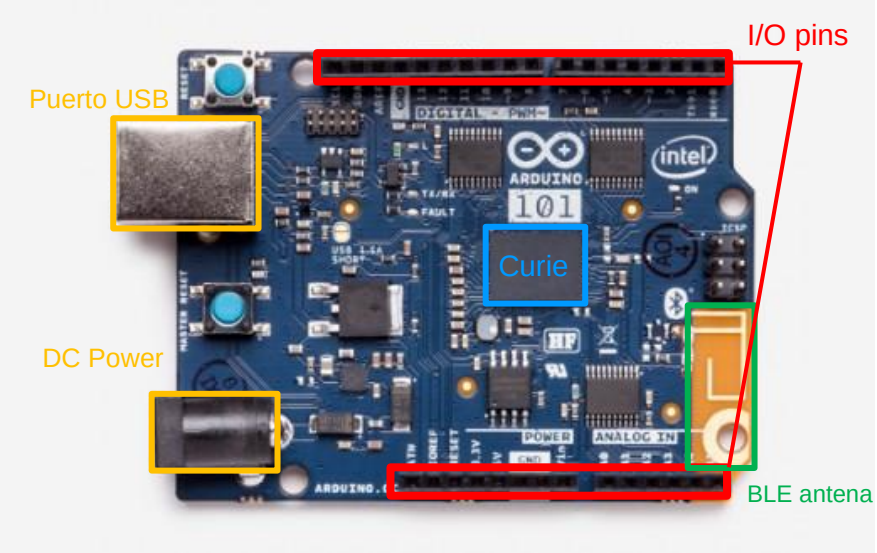


*Figura 21. Esquema placa Arduino 101.*
*Fuente: Arduino Store.*

### 4.2.4 Conectividad

La conectividad del dispositivo de análisis se basará en conexiones *Bluetooth Low Energy*. Esta tecnología inalámbrica es una variante de bajo consumo de la tecnología *Bluetooth* presente en millones de dispositivos. La tecnología BLE ha sido concebida para aplicaciones de control y monitorización con algunas diferencias respecto a su vertiente original, tal y como se detalla en la siguiente tabla:

*Tabla 3. Comparativa entre Bluetooth y BLE.*

| Característica | Bluetooth | BLE |
|---|---|---|
| Radio | 100m | >100m |
| Velocidad de conexión | 1-3Mbit/s | 125Kbit/s – 2Mbit/s |
| Seguridad | 128-bit AES | 128-bit AES |
| Latencia | 100ms | 6ms |
| Mínimo tiempo de envío de datos | 0.625ms | 3ms |
| Consumo | 1W | 0.01 – 0.5 W |

Como se puede observar, la velocidad máxima de transferencia es ligeramente inferior, pero, por otra parte, se reduce el consumo energético considerablemente. La principal razón por la que escoger la tecnología BLE frente a otras, como podría ser ZigBee o Z-Wave, reside en el amplio rango de dispositivos que son compatibles con ella.

La tecnología BLE se compone de dos partes principales (ver Figura 22) (Gomez, Oller, & Paradells, 2012):

- ***Controller***: comprende las capas Física y de Enlace del modelo OSI para esta tecnología. Suele encontrarse en un *System-on-Chip* (SoC) con la radio integrada.
  - ***Capa física:*** la tecnología BLE opera en la banda de frecuencias de 2.4GHz con 40 canales distintos de radiofrecuencia. Estos canales se dividen en *canales de descubrimiento* y *canales de datos*. Los primeros son usados para descubrir y establecer conexión con el otro extremo (de los 40 canales disponibles, se destinan 3 para el descubrimiento), mientras que el resto se usan para transferir datos. En esta capa se encuentra un mecanismo que escoge, a través de intervalos de tiempo, uno de los 37 canales restantes para intercambiar datos, seleccionando el canal menos propenso a sufrir interferencias.

  - ***Capa de enlace:*** los dispositivos que hacen *broadcast* de ciertos datos reciben el nombre de *Advertiser*. Esta transmisión se realiza a través de los canales de difusión antes comentados. También se encuentran dispositivos que sólo reciben datos a través de estos canales, y que reciben el nombre de *Scanners.*
    La conexión entre dos dispositivos es un procedimiento asimétrico. Primero, un *Advertiser* anuncia su presencia; en el momento en que otro dispositivo escucha dichas señales, envía una petición de conexión (*connection request*); este dispositivo recibe el nombre de *Initiator*.
    Una vez aceptada la petición, ambos dispositivos se comunicarán entre ellos usando los canales de datos. La conexión se identificará con un código de 32 bits generado aleatoriamente.
    Una vez establecida la conexión, se redefinen los roles; el *Initiator* pasa a ser el *Máster*, y el *Advertiser* el *Slave*. El primero puede conectarse a tantos *Slaves* como quiera, mientras que el segundo sólo puede estar conectado a un *Máster*, por lo que BLE presenta una topología de red en estrella.
    Una vez conectados con el *máster*, los *slaves* entran en un modo de ahorro energético, despertando de forma periódica según se ha establecido previamente con el máster. Así pues, el máster controla el acceso al medio, indicando a cada *slave* cuándo tiene que despertar.
    La conexión establecida entre *máster* y *slave* se establece en un canal físico, dividiendo el tiempo en *connection events*; al inicio de cada evento el *master* transmite un paquete; al recibir el paquete, el *slave* debe responder. Si desea enviar más información, activa el flag MD (*More Data*) del paquete. Mientras *master* y *slave* sigan intercambiando información, el evento se mantiene. En el momento en que ninguno de los dos tenga nada más que transmitir, el evento se cierra y no es hasta el siguiente evento cuando se pueden volver a comunicar.
    Cada paquete de la conexión se identifica con un número de secuencia y un *Next Expected Sequence Number*, que indica el siguiente paquete a recibir. Esto permite que, al recibir un paquete de forma correcta, se

incremente el *NESN* del siguiente paquete a enviar sirviendo como *ACK* al paquete recibido. Si el paquete se recibe de forma errónea, el *NESN* generado también lo será, por lo que sirve a su vez de *ACK* negativo.

- ***Host:*** comprende los protocolos de nivel superior de BLE, como pueden ser *Attribute Protocol* (*ATT*) o *Generic Access Profile(GAP).*

  - ***Protocolo L2CAP:*** este protocolo se encarga de encapsular los protocolos de capas superiores (ATT, SMP) en la capa de enlace. Se caracteriza por ser *best-effort* sin segmentación.

  - ***ATT:*** este protocolo define las comunicaciones entre clientes y servidores. Cada servidor consta de una serie de atributos, que son campos de datos que gestiona el protocolo *GATT* de la capa superior. El cliente accede a estos atributos mediante peticiones, y estas peticiones generan una respuesta del servidor.
    El servidor puede enviar adicionalmente dos tipos de mensajes, las notificaciones y las indicaciones; se diferencian principalmente en que estas últimas requieren que el cliente envíe una confirmación.
    El cliente también puede modificar los atributos del servidor mediante peticiones.

  - ***GATT:*** este *framework* hace uso del *ATT* para identificar los servicios prestados por el servidor. Estos servicios contienen características, que son los datos a intercambiar entre dispositivos. Los servicios y las características están compuestos por atributos.

  - ***GAP:*** esta tecnología especifica los distintos roles que los dispositivos pueden tomar, y, con ello, las funcionalidades que deben implementar con los protocolos anteriores. Los roles principales son *Observer, Broadcaster, Peripheral* y *Central.* En este proyecto se discutirán exclusivamente estos dos últimos. Los dispositivos *Peripheral* y *Central* son similares a los *master* y *slave* anteriormente definidos.
    Sumado a esto, se encuentran definidos varios perfiles de aplicaciones que incluyen servicios y características ya definidos; esto se debe a que hay perfiles que son comúnmente usados en entornos industriales, o en sanidad.

Además de los protocolos antes mencionados, BLE implementa otros protocolos que dan soporte a la seguridad de las conexiones. La seguridad se clasifica en dos modos excluyentes:

- ***LE Security Mode 1:*** proporciona seguridad en la capa de enlace mediante los protocolos *Chaining-Message authentication Code* y 128-bit *AES*. Al activar este modo, se añade una cabecera extra de 4 Bytes a cada mensaje para comprobar su integridad. Esta cabecera es cifrada junto al *payload* del mensaje.

- ***LE Security Mode 2:*** proporciona seguridad en la capa de *ATT* incluyendo una firma de 12 Bytes junto a cada *payload.* Dicha firma es calculada e interpretada mediante un algoritmo basado en 128-bit *AES*.

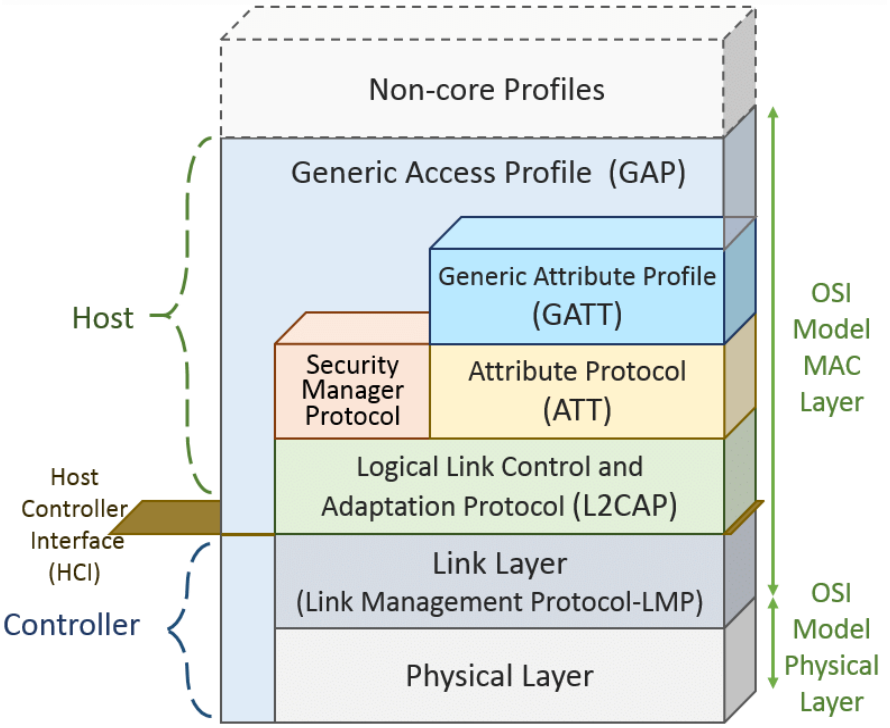


*Figura 22. Bluetooth Low Energy protocol stack. Fuente: researchgate.net.*

## 4.2.5 Software de desarrollo

La plataforma Arduino cuenta con su propio entorno de desarrollo o IDE por sus siglas en inglés (*Integrated Development Environment*). Se puede observar una captura de este software en la Figura 23. Este IDE permite programar el *sketch* o programa principal de la placa, donde se encuentra todo el código a ejecutar. Por defecto, este viene configurado para trabajar con Arduino UNO, por lo que es necesaria una configuración inicial para poder trabajar con Arduino 101.

Esta configuración consiste en la instalación de los drivers de la placa, para que así el ordenador identifique correctamente el dispositivo. Desde el propio menú del IDE se pude buscar la placa en concreto, e instalar los drivers.

El lenguaje de programación de Arduino es el *Processing*, un lenguaje derivado de C++. Una vez escrito el código, basta con indicarle al IDE que lo cargue a la placa para que este lo compile y lo descargue al dispositivo. El IDE dispone de un panel inferior donde se muestran las distintas alertas del compilador.

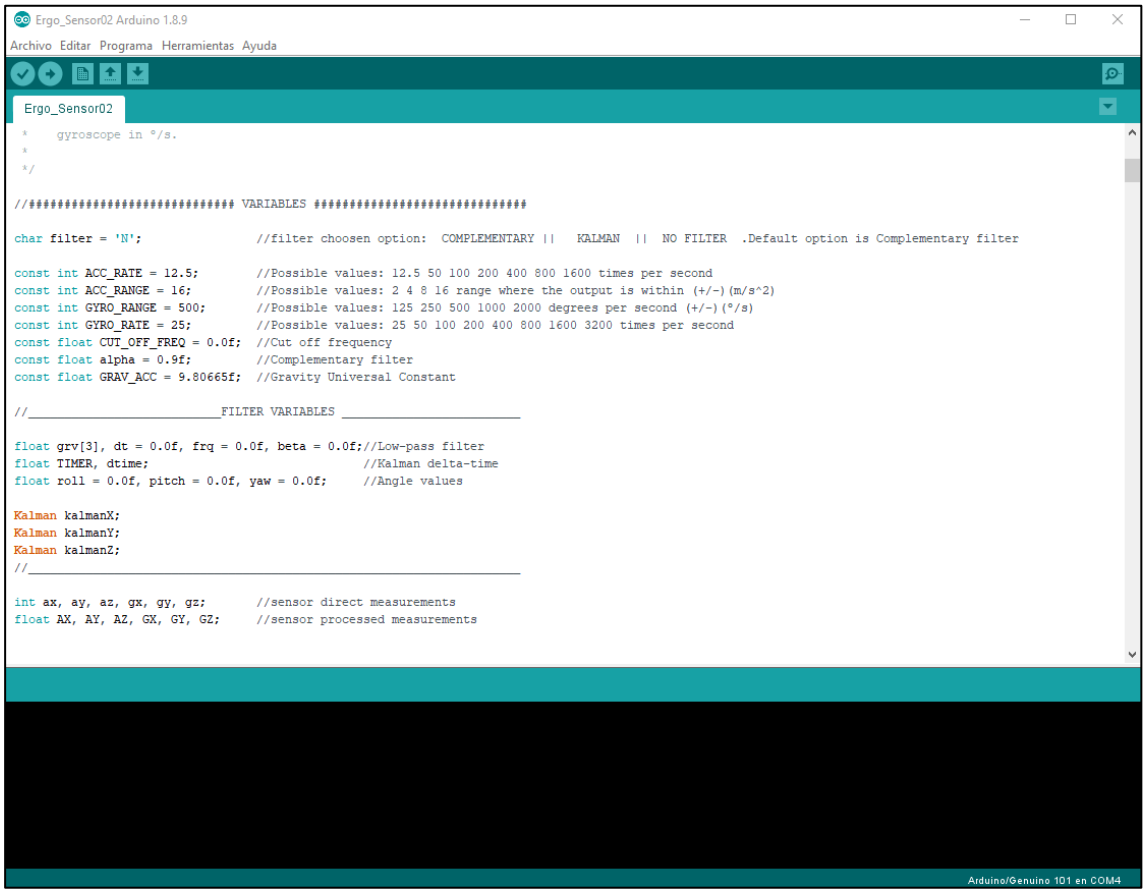


*Figura 23. Interfaz de Arduino IDE.*

## 4.2.6 Diseño del programa

Todo código de Arduino se forma entorno a una misma estructura: los dos métodos principales *setup* y *loop*. El primero de ellos se ejecuta nada ms iniciar el programa, y tras su ejecución se pasa a la ejecución del método *loop* que, como su nombre indica, se repite en bucle. A parte de estos métodos principales, el desarrollador puede crear tantos como crea conveniente. Se ha diseñado el siguiente diagrama de flujo (ver Figura 24) para describir el comportamiento del programa a desarrollar.

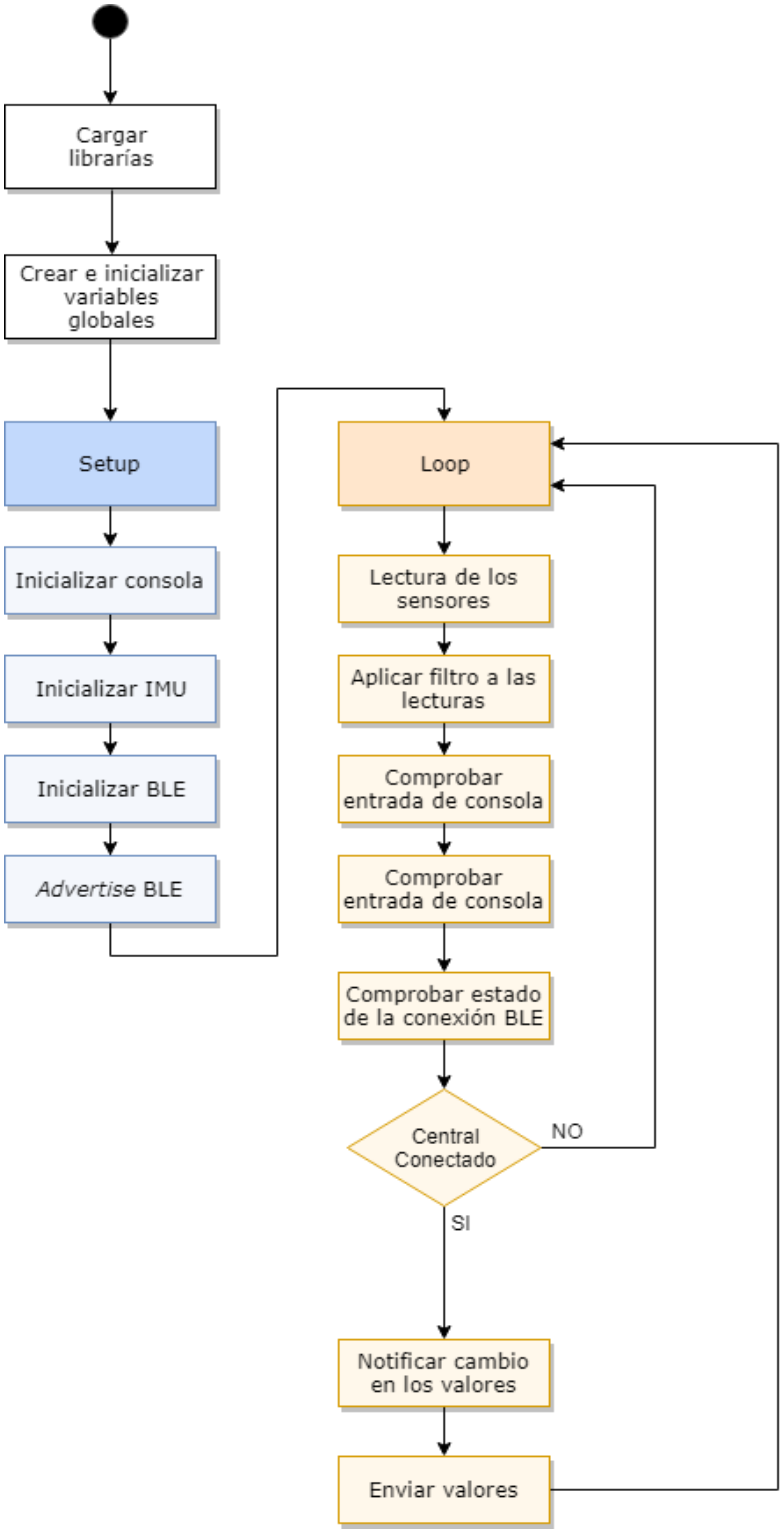


*Figura 24. Diagrama de flujo de Arduino.*

En el diagrama no se representan posibles llamadas a funciones desarrolladas para este proyecto, ya que esto será objetivo de discusión en apartados posteriores.

## 4.3 Diseño de la aplicación móvil

La aplicación móvil será la encargada de comunicarse con el dispositivo de análisis, y recabar los datos captados para luego discernir si la reacción detectada en el apriete cumple con la ergonomía establecida. Para ello, en los siguientes apartados se discutirán los distintos aspectos de dicha aplicación.

### 4.3.1 Software de desarrollo

En la fase de análisis de requisitos, se ha establecido que la aplicación a desarrollar será usada en dispositivos basados en el sistema operativo Android. Esto ha condicionado a la elección de **Java** como el lenguaje de desarrollo de la aplicación.

Java es un lenguaje de programación desarrollado por *Sun Microsystems* con la intención de ser multiplataforma, lo cual conlleva a que un código de este lenguaje compilado para una máquina puede ser ejecutado por otra máquina que también incorpore el sistema de ejecución de Java; por esto, pese a ser similar a C o C++, no comparte funcionalidades de bajo nivel que dichos lenguajes sí incorporan.

El paradigma de programación que caracteriza a este lenguaje es *orientado a objetos*, dónde cada objeto representa una entidad con atributos y funciones. De esta manera permite reutilizar código entre aplicaciones que puedan compartir funcionalidades. Además, debido a su longevidad y a su facilidad de aprendizaje, dispone de una comunidad que da soporte al desarrollo, incorporando multitud de librerías de libre acceso.

Junto a Kotlin, Java es el lenguaje oficial de Android, por lo que, al desarrollar cualquier aplicación, es preferible usar este lenguaje. Por otra parte, Kotlin es un lenguaje pensado para ser interoperable con Java, aún así; es preferible decantarse por el segundo, ya que Kotlin es un lenguaje más nuevo y, por lo tanto, menos extendido.

Una vez escogido Java como el lenguaje de desarrollo de la aplicación, se debe decidir cuál va a ser el IDE para dicho desarrollo. De entre los distintos IDEs disponibles hoy en día, se ha elegido **Android Studio** (ver Figura 25), en contraposición a alternativas como Eclipse o NetBeans, por las razones que se exponen a continuación (Android Developers, 2019).

Android Studio es un IDE desarrollado por Google y orientado a la programación de aplicaciones Android. Este IDE ofrece multitud de facilidades para desarrollar tanto las interfaces de las aplicaciones, como la lógica que hay detrás de estas. Un ejemplo de ello es el renderizado de la interfaz en tiempo real, que permite comprobar que la interfaz se visualizará como se espera. Sumado a esto, incorpora un emulador de dispositivos Android, pudiendo elegir el hardware y la versión del sistema operativo a emular (pudiendo lanzar aplicaciones en dicho emulador). Además,

permite a aquellos desarrolladores que disponen de un dispositivo con Android, enlazarlo al IDE para ejecutar directamente la aplicación desarrollada desde el mismo IDE.

A parte de las características específicas para el desarrollo de aplicaciones, Android Studio incorpora funcionalidades comunes en todos los IDEs, como pueden ser una consola de desarrollador y consejos de optimización, entre otros.

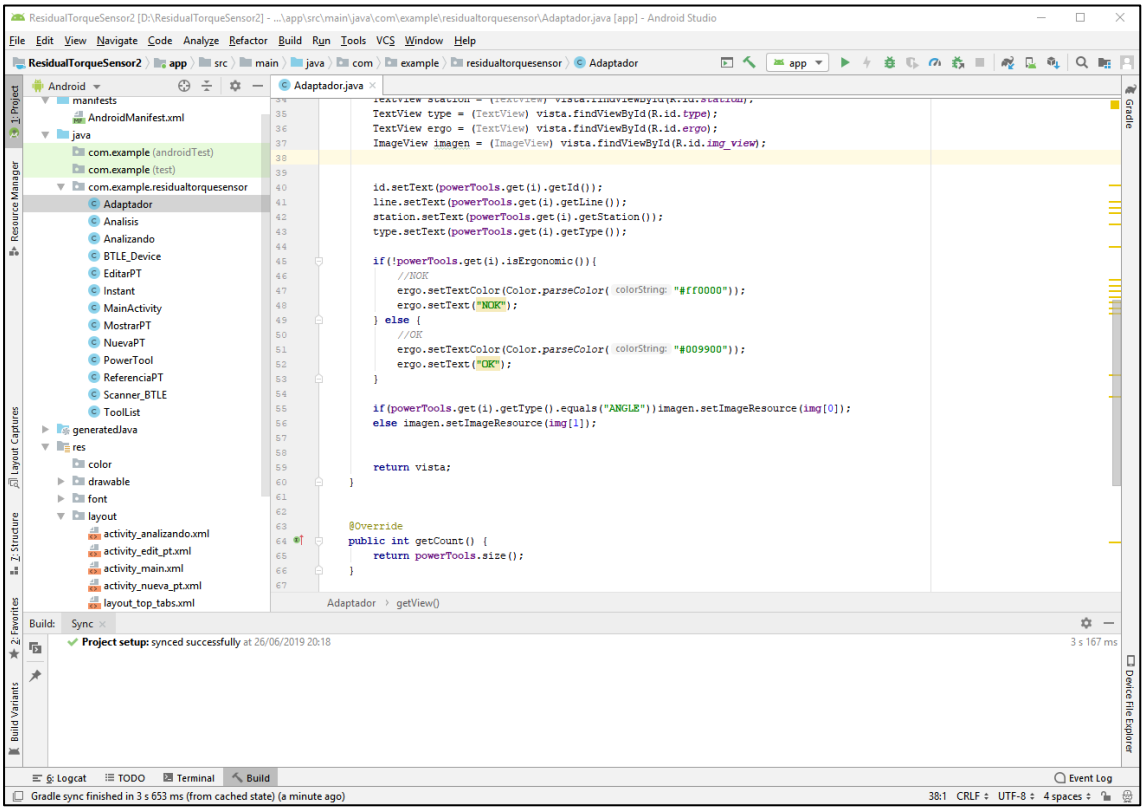


*Figura 25. Interfaz de Android Studio.*

### 4.3.2 Estructura de la aplicación

Las aplicaciones de Android se componen de Actividades, donde una actividad es una pantalla que permite a los usuarios interactuar con el dispositivo. Cada actividad a su vez está compuesta por una clase Java (que implementa la lógica) asociada a un *layout* (interfaz gráfica de la actividad). Los *layouts* se implementan con el lenguaje de marcas XML (*Extensible Markup Language*), que permite incorporar botones, listas y otros elementos interactivos de forma gráfica. Así pues, al navegar por las ventanas de la aplicación, lo que ocurre es que se activan y desactivan las distintas actividades. En la Figura 26 se puede apreciar un diagrama del ciclo de vida de las actividades; al crearse, se invoca al método *onCreate()*, que contiene la referencia al *layout* a mostrar. Justo antes de ser visible para el usuario, se llama al método *onStart()*. Si la actividad se deja en segundo plano, se invoca al método *onPause()*, tras la vuelta de la pausa, se invoca al método *onResume().* Al llamar a otra actividad, se invoca al método *onStop().* Al reanudar la actividad después de pausarla, se invoca primer a *onRestart()*. Finalmente, si la actividad va a ser finalizada, se llama al método *onDestroy()* (Rafols Montane, 2016).

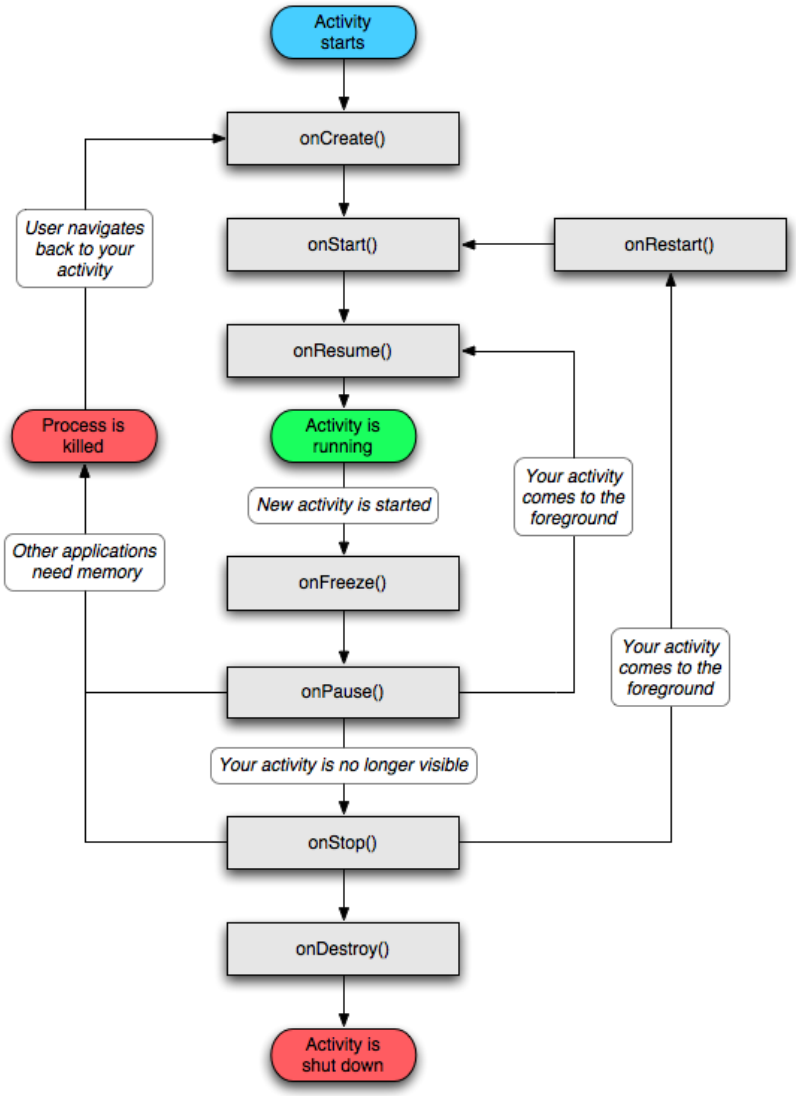


*Figura 26. Ciclo de vida de una actividad.*
*Fuente: devacademy.*

El ciclo de vida antes presentado repercute en el diseño de las actividades de la aplicación, ya que hay funciones que pueden permanecer en segundo plano para así conservar su estado, como puede ser la conexión BLE.

Partiendo de los casos de uso especificados en la fase de análisis de requisitos y del prototipo generado, se van a diseñar dos diagramas: un primer diagrama de clases con el lenguaje de modelado UML (ver Figura 27), y un diagrama de flujo de las actividades de la aplicación (ver Figura 28).

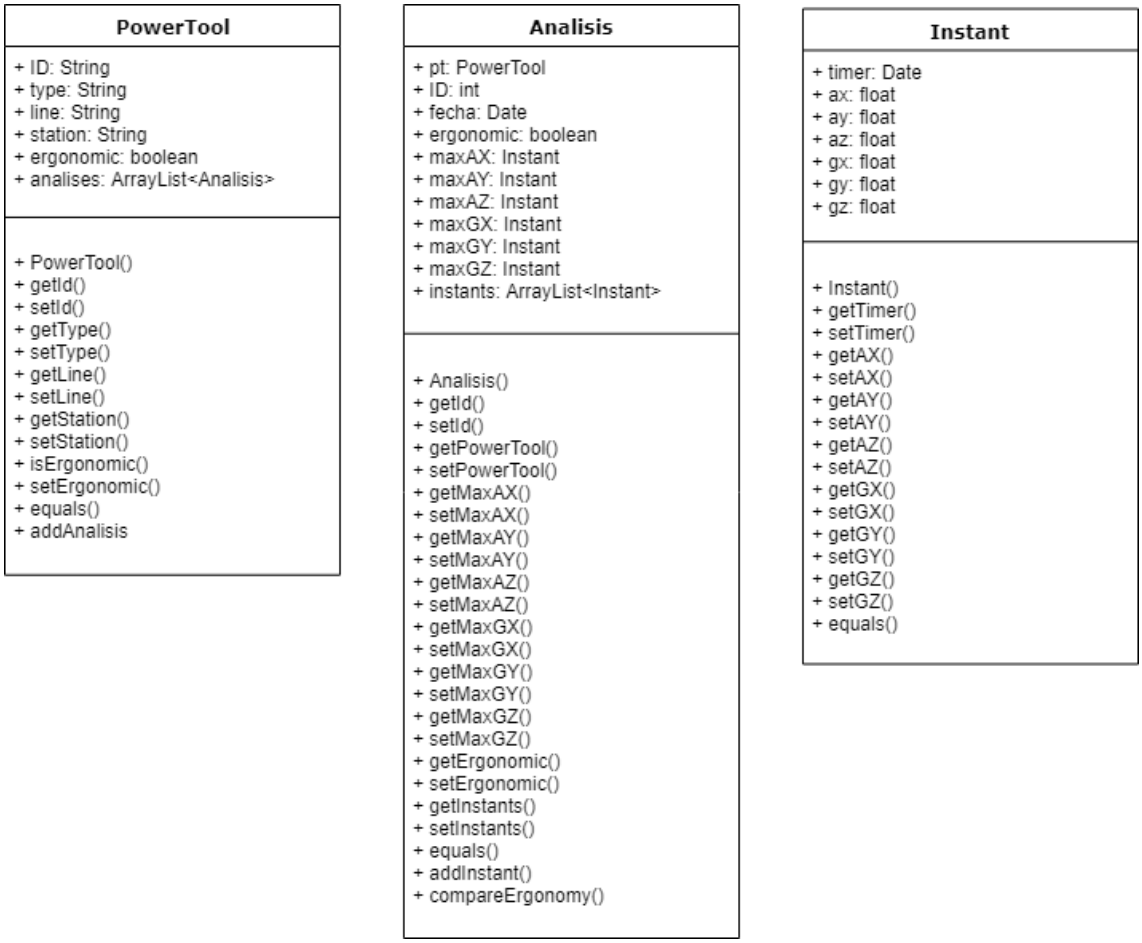


*Figura 28. Diagrama de clases UML.*

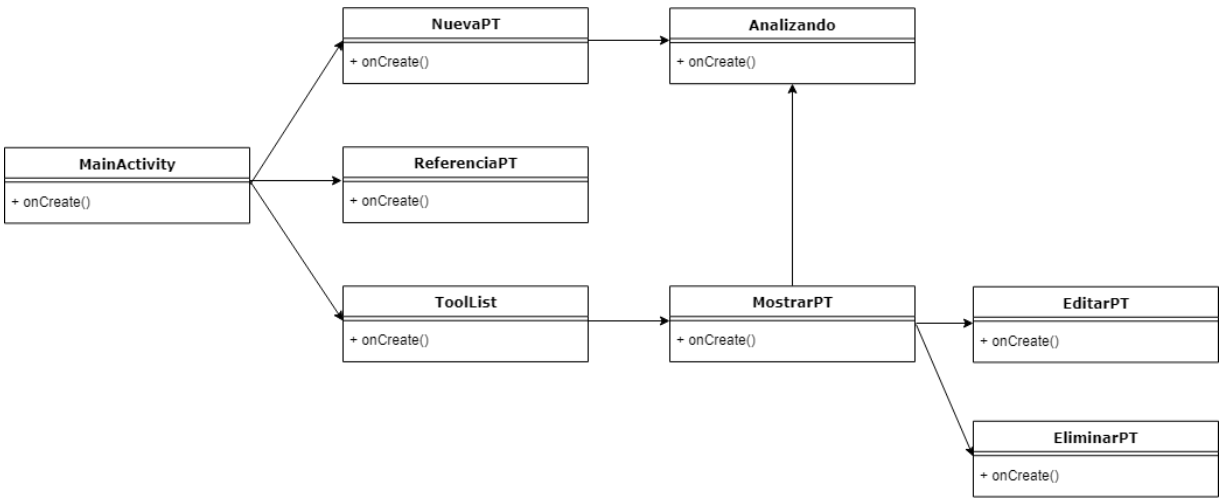


*Figura 27. Diagrama de flujo de actividades.*

El diagrama de clases diseñado presenta las tres clases principales que compondrán la aplicación.

Por un lado se encuentra la clase PowerTool, con sus métodos de consulta e inicialización de variables; dicha clase tiene una variable tipo *ArrayList<Análisis>* que almacena los análisis realizados a la PowerTool.

La segunda clase se trata de Analisis, y ésta, al igual que la clase PowerTool, contiene los métodos de consulta e inicialización de variables. Esta clase almacena una lista de objetos del tipo Instant en un ArrayList<Instant>, los cuales son cada uno de los instantes o medidas tomadas en cada momento determinado del análisis de la herramienta.

Finalmente se encuentra la clase Instant que, como se acaba de señalar, especifica los valores de los sensores para una marca temporal concreta.

El diagrama de actividades muestra el flujo de actividades que presentará la aplicación. Las actividades incluyen el método onCreate() ya que este método es necesario para lanzar la interfaz de usuario al inicio de la misma.

## 4.3.3 Conectividad

La aplicación móvil deberá conectarse con dos servicios. El primero es el dispositivo de análisis, del cuál obtendrá las mediciones en tiempo real. Para ello, tal y como se ha explicado anteriormente en esta memoria, hará uso de la tecnología Bluetooth Low Energy. El segundo servicio al que se conectará es a la base de datos en la nube; los detalles de esta conexión serán discutidos junto con el diseño de la base de datos.

#### *4.3.3.1 Conexión smartphone-dispositivo de análisis*

La conexión BLE toma un modelo publicador/subscriptor donde el dispositivo Arduino se encarga de notificar los cambios en los valores de las variables a los dispositivos conectados. En este esquema de red, el dispositivo Arduino recibe el nombre de *Peripheral,* y los dispositivos que se conectan a él adquieren el nombre de *central.* Como se especifica en apartados anteriores de esta memoria, a un mismo *peripheral* pueden conectarse varios *centrals*, pero un mismo *central* sólo puede encontrarse conectado a un *peripheral*; aunque esto no origina ninguna restricción en este proyecto, ya que se pretende establecer comunicación entre un solo *peripheral* y un *central* (Davidson, Townsend, & Cufí, 2014).

El dispositivo Arduino debe transmitir las lecturas del acelerómetro y del giroscopio, lo que resulta un total de seis valores conformados por: las lecturas del acelerómetro en los ejes X, Y, Z, y las lecturas del giroscopio en los ejes X, Y, Z. La tecnología

GATT (*General Attribute Profile*) de BLE es la encargada de definir los siguientes aspectos de la conexión:

- ***Roles:*** como se acaba de indicar, el dispositivo Arduino ejerce el rol de *peripheral*, mientras que el smartphone ejerce el rol de *central*. Los roles sirven para definir qué funcionalidades realizará cada dispositivo.
- ***Servicios:*** el dispositivo *peripheral* publicita mediante éstos, qué funciones ofrece. Un servicio está compuesto por los siguientes atributos:
  - ***Nombre***
  - ***UUID:*** identificador numérico de 128 bits (*Universal Unique Identifier*).
- ***Características:*** son el contenido de los servicios. Una característica contiene los siguientes campos:
  - ***Nombre.***
  - ***UUID.***
  - ***Valor a transmitir.***
  - ***Permisos:*** para cada característica se pueden indicar las operaciones posibles. Estas son: lectura, escritura y notificación de cambios.

Por lo que el perfil GATT del dispositivo Arduino tomaría la siguiente estructura:

*Tabla 4. Esquema del servicio GATT.*

| SERVICIO | | | | | |
|---|---|---|---|---|---|
| Nombre<br>UUID | | | | | |
| | Característica AX | Nombre | UUID | Valor | Read, Notify |
| | Característica AY | Nombre | UUID | Valor | Read, Notify |
| | Característica AZ | Nombre | UUID | Valor | Read, Notify |
| | Característica GX | Nombre | UUID | Valor | Read, Notify |
| | Característica GY | Nombre | UUID | Valor | Read, Notify |
| | Característica GZ | Nombre | UUID | Valor | Read, Notify |

Como se observa, los permisos sobre las características deben indicarse como *read* para dar permisos de lectura al *central*, y *notify*, para que de esta forma, cada vez que cambie el valor de la característica (cada vez que se realice una nueva lectura del sensor) se notifique al *central* de dicho cambio (Arduino, 2019).

Finalmente, la conexión BLE constaría de las siguientes fases (ver Tabla 5)

*Tabla 5. Fases de la conexión entre Peripheral y Central.*

| Peripheral | Central |
|---|---|
| El dispositivo **publicita** los servicios de los que dispone. | |
| | **Detecta** el servicio que le interesa y pide conectarse. |

| | |
|---|---|
| **Acepta** la conexión. | |
| | Se **suscribe** a las características. |
| | |
| **Notifica** cambio en alguna característica. | |
| | **Lee** los valores de las características. |
| | |
| | **Cierra** la conexión. |

## 4.4 Diseño de la base de datos

La base de datos del sistema almacenará cada análisis realizado, junto a un conjunto de valores que permitan identificar a qué herramienta pertenece. De las tecnologías disponibles hoy en día, se ha buscado una que pueda satisfacer los requisitos del proyecto en cuanto a coste y complejidad.

### 4.4.1 Diseño de la estructura de datos

Antes de estudiar las distintas tecnologías disponibles actualmente para implementar bases de datos, se pretende realizar un diseño de los datos que dicha base de datos debe albergar, así como sus relaciones.

La base de datos tendrá que almacenar cada uno de los análisis realizados, por lo que se deberá almacenar primero de todo la *PowerTool* analizada junto con todas aquellas características que se conozcan de ella. Una vez se crea una nueva herramienta, se le pueden asignar los análisis realizados, por lo que no puede existir un análisis que no esté asociado a una herramienta existente.

De cada análisis se deben almacenar a su vez los instantes identificados por marcas temporales, ya que, si se almacena exclusivamente el valor del resultado, todos los elementos de tipo *Instant* generados serán eliminados al cerrar la aplicación. De la misma forma que no puede existir un análisis que no tenga una herramienta signada, no puede haber un instante que no esté asociado a un análisis y, por ende, asociado a una PowerTool.

Así pues, partiendo del diseño de clases visto en la Figura 28, se crea un nuevo diseño que represente exclusivamente los datos a almacenar, junto a las relaciones que estos mantienen (ver Figura29).

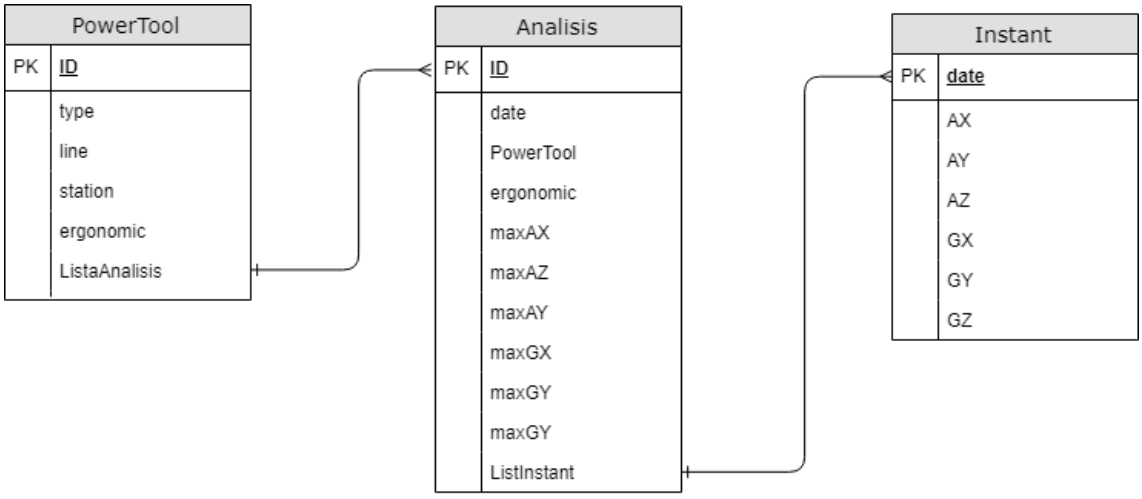


*Figura 29. Diagrama de datos.*

## 4.4.2 Tecnología utilizada

La tecnología de base de datos escogida para el proyecto es Firebase. Este servicio web creado por Google se basa en el modelo de *Software as a Service*; consiste en ofrecer aplicaciones o servicios accesibles a través de Internet, alojando estos en servidores distribuidos accesibles desde la red. Esto facilita la implementación del sistema ya que el desarrollador evita tener que preocuparse por aspectos como el alojamiento en uno o varios servidores, el mantenimiento de los mismos, o el despliegue del servicio en sendos servidores.

Firebase dispone de un catálogo amplio de funcionalidades que dan soporte a distintos tipos de aplicaciones. A continuación se detallan dichas funcionalidades:

- ***Cloud Firestore***: una base de datos NoSQL alojada en la nube que almacena y sincroniza datos entre usuarios y dispositivos. Ofrece sincronización en vivo y soporte sin conexión.

- ***Kit de AA***: API de desarrollo de aplicaciones con inteligencia artificial, conjuntamente con espacio de alojamiento en la nube para dichas aplicaciones.

- ***Cloud Functions***: servicio de alojamiento de funcionalidades *backend* para servidores externos.

- ***Authentication:*** API para la gestión de la autenticación de los usuarios en todo tipo de aplicaciones, con capacidad de autenticar con diferentes medios como pueden ser correo electrónico o cuentas de usuario de distintas redes sociales.

- ***Hosting:*** servicio de *hosting* para páginas web, con baja latencia y certificación SSL.

- ***Cloud Storage:*** espacio de almacenamiento para contenido multimedia generado por los usuarios en el uso de aplicaciones.

- ***Realtime Database:*** Base de datos con baja latencia y sincronización en tiempo real diseñada para almacenar lecturas de dispositivos en tiempo real. Ofrece soporte sin conexión, es decir, almacena localmente en el dispositivo los datos cuando no se consigue conectar con el servidor para, posteriormente, sincronizar los datos locales en momento la conexión con el servidor vuelve a estar disponible.

De entre los servicios que esta plataforma proporciona, se ha elegido Realtime Database como opción para el almacenamiento de los datos. Por una parte, al tratarse de una base de datos que se sincroniza de forma automática en tiempo real, permite mantener una lista actualizada del estado de cada PowerTool en todo momento. Sumado a esto, la base de datos traduce de forma automática el formato de los datos a almacenar, por lo que, si se envía un objeto o lista de objetos, dicha base de datos realiza de forma transparente al cliente una operación de traducción de los datos a formato JSON. Sumado a esto, la consulta de datos de la nube se realiza de forma sencilla, ya que dispone de complementos para aplicaciones Android que facilitan esta tarea.

Las características antes mencionadas simplifican el desarrollo de aplicaciones que requieran de una base de datos sincronizada en la nube, y proyectos de una envergadura relativamente pequeña, como es el caso que se trata en esta memoria, se ven beneficiados por estos servicios.

### 4.4.3 Comunicaciones

El intercambio de información entre el dispositivo Android y la base de datos se realiza referenciando primero la API del servicio proporcionado por Google en la aplicación Android. Una vez referenciado es suficiente con usar las funciones de la API para enviar y recibir los datos. Aunque resulte transparente a la vista del desarrollador, las comunicaciones subyacentes al código de la API se basan en peticiones HTTP, mediante los métodos que implementa dicho protocolo (GET, POST, PUT, DELETE y HEAD). Una vez los datos se reciben en el servidor de base de datos, el servicio los transforma a formato JSON (Google Inc, 2019).

# 5. Implementación

En este capítulo se muestra la implementación de los distintos módulos del sistema. Los lenguajes y entornos de desarrollo usados en este capítulo han sido discutidos en el apartado de diseño, por lo que el objetivo de esta sección es explicar al lector de esta memoria aquellos puntos clave del software desarrollado, dejando a un lado operaciones o funciones que se usen de forma frecuente en la implementación de aplicaciones similares.

## 5.1 Dispositivo de análisis

En el programa desarrollado se importan las siguientes librerías:

- ***CurieBLE:*** librería que incluye las funciones para comunicarse con el chip BLE, junto a esta se incluye la librería *BMI160*.
- ***CurieIMU:*** librería que incluye las funciones para comunicarse con la IMU.
- ***Kalman:*** librería desarrollada para este proyecto. su implementación será discutida en apartados posteriores.

Tras incluir las librerías, el siguiente paso es declarar e inicializar las variables globales del programa, entre estas se encuentran los parámetros *rate* y *range* del acelerómetro y del giroscopio. El parámetro *rate* indica la frecuencia de análisis del sensor, cada sensor dispone de un rango distinto de valores posibles. Por otro lado, el parámetro *range* indica el rango de valores en los cuales el sensor va a medir, este parámetro afecta directamente a la sensibilidad del sensor, ya que a mayor sea el rango, menor es la precisión de las medidas. El *range* para el acelerómetro se expresa en $m/s^2$, mientras que en el giroscopio se expresa en *º/s*. Los valores posibles de *rate* y *range* para cada sensor son especificados por el fabricante, en este caso se han adoptado los valores siguientes:

Seguidamente se declara el rol del dispositivo Arduino como *peripheral*. El servicio BLE precisa de una declaración inicial del identificador o UUID del mismo, además, cada una de las características se inicializa como tipo *float* con su UUID y los parámetros para habilitar la lectura y las notificaciones.

Los UUIDs asignados se han generado aleatoriamente haciendo uso de la herramienta online *UUID generator*.

A continuación se detalla una lista con los métodos secundarios implementados. Estos métodos realizan funciones secundarias para simplificar el código del método principal *loop()*.

- ***void filterData():*** la llamada a este método provoca a su vez una llamada a la librería *Kalman* donde se implementa dicho filtro.

```
kalmanX.setQangle(0.003f);
kalmanY.setQbias(0.003f);

yaw = kalmanZ.getAngle(yaw, dtime, GZ);
```

- ***printAcc():*** muestra en la consola *Serial* los valores de las lecturas del acelerómetro. Este método es usado para el *testing* del software cuando el dispositivo Arduino se encuentra conectado a un ordenador mediante conexión USB.
- ***printGyro():*** muestra en la consola *Serial* los valores de las lecturas del giroscopio. Al igual que el método anterior, este método es usado para el *testing* del software.
- ***calibrateSensors():*** este método calibra los sensores mediante la llamada a los métodos propios de calibración de cada sensor que se encuentran en la librería *CurieIMU.* Durante la calibración, es crítico mantener el dispositivo inmóvil y paralelo al suelo en una superficie plana, esto es debido a que la calibración de la aceleración en el eje Z se realiza en base a la aceleración de la fuerza gravitacional.

```
CurieIMU.autoCalibrateGyroOffset();
```

- ***consoleGuide():*** este método imprime por la consola *Serial* un menú con las entradas de teclado que activan distintas funciones en la placa, como puede ser los métodos *printAcc()* y *printGyro()*, o la calibración de los sensores.
- ***printConfig():*** este método imprime por la consola *Serial* los valores de las variables globales que configuran los sensores.
- ***connectHandler():*** este método es invocado cada vez que un dispositivo BLE *central* se conecta. La llamada a este método provoca la activación de la escritura en las características del servicio BLE, es decir, activa la transmisión de datos.
- ***disconnectHandler():*** este método es invocado cada vez que un dispositivo BLE *central* se desconecta. Este método realiza la funcionalidad opuesta al método anterior, desactiva la transmisión de datos.

El método principal *setup()* empieza inicializando la consola *Serial* a una frecuencia de 9600 baudios o señales por segundo. Seguidamente se inicializa la unidad IMU, indicando los valores de *rate* y *range* para ambos sensores.

```
CurieIMU.setGyroRate(GYRO_RATE);
```

A continuación, se activa el módulo BLE con la llamada al método *BLE.begin()*. El siguiente paso es inicializar el servicio, para ello, se le dota de un nombre (“ERGO”), se le incluye en la lista de difusión de servicios, y se le añaden las características. Una vez hecho esto, se añade el servicio al dispositivo mediante la llamada a *addService().*

La última configuración del módulo BLE consta de la llamada al método *advertise()*, que inicia el broadcast del servicio ofrecido, y la asignación de dos manejadores de eventos para cunado un *central* se conecta o se desconecta.

```
BLE.setEventHandler(BLEDisconnected, disconnectHandler);
```

El método principal *loop()* comienza leyendo los valores de ambos sensores mediante la llamada al método *readMotionSensor()*, que devuelve el valor leído en cada eje. Es necesario convertir el valor leído para poder expresarlo en $m/s^2$ en el caso del acelerómetro, y en º/s en el caso del giroscopio.

```
AZ = (az/32768.0f)*CurieIMU.getAccelerometerRange();
AZ = AZ/1000 * GRAV_ACC;
```

Tras leer los valores de los sensores, el bucle pasa a hacer una llamada al método *filterData()*, que aplica el filtro a los datos recién leídos.

Tras esto, el bucle comprueba si existe alguna entrada en el puerto *Serial* de la consola, en caso afirmativo identifica el carácter introducido, de esta forma, se pueden hacer llamadas a funciones como la calibración de los sensores desde la consola *Serial*. Esto resulta de gran utilidad para la depuración de código y las pruebas de *software*.

Finalmente, si se cumple la condición de que hay un *central* conectado al dispositivo Arduino, este último cambia los valores de las características, enviando así notificación de dicho cambio.

```
if(centralConnected){
   gyroZ.setValue(GZ);
}
```

## 5.2 Librería Kalman

La librería de Arduino que implementa el filtro de Kalman, se ha realizado con el lenguaje de programación C++ y el editor de texto avanzado NotePad++. Toda librería de Arduino se estructura como una clase en C++, compuesta por un archivo con extensión .h, que contiene las cabeceras de la clase, y un archivo .cpp que contiene las definiciones junto con el código que añade la funcionalidad a la librería. Dado que se trata de un desarrollo relativamente sencillo, se ha optado por la herramienta NotePad++ debido principalmente a su sencillez y a la amplia variedad de lenguajes que soporta.

El archivo .cpp, consta de dos partes, una primera parte que crea el objeto Kalman; dicho objeto contiene cada una de las variables que se usan para hacer el cálculo, como pueden ser las matrices, los errores y el estado actual (entre otros). Toda variable usada en esta libraría almacena datos en formato coma flotante o *float* ya que las mediciones se realizan en dicho formato.
La segunda parte implementa las funciones del objeto, a excepción de la función *getAngle(),* el resto de funciones han sido creadas para obtener o cambiar los valores de ciertas variables de la clase (funciones *get* y *set*). A continuación se describe el procedimiento seguido por la función principal.

- **getAngle()**: esta función es la encargada de realizar el calculo del filtro. Los valores de entrada de la función son la medida del sensor, la marca temporal y el parámetro *rate* del sensor. Este cálculo se descompone en tres bloques: *predict*, que calcula una predicción del nuevo estado *measure,* que calcula el nuevo estado en base a la medida del sensor, y *update*, que combina estos dos estados calculados en base al grado de confianza que el filtro mantenga sobre los valores reales medidos.

## 5.3 Aplicación Android

En el desarrollo de la aplicación Android, se ha perseguido el realizar una aplicación sencilla de usar, y que aporte la funcionalidad deseada al sistema. Se ha mantenido una estética unificada para todas las actividades que la componen, usando un mismo conjunto de fuentes de texto y gama de colores, simplificando el aspecto de la interfaz.

A continuación se describen las distintas actividades desarrolladas que componen la aplicación Android.

### 5.3.1 Actividad principal

La actividad principal (ver Figura 30) permite conectarse al dispositivo de análisis, informando al usuario del estado de la conexión con el dispositivo. Desde esta ventana se accede a la creación y posterior análisis de una nueva *PowerTool*, además de a la ventana de selección de la *PowerTool* de referencia.
Desde el menú inferior se puede navegar entre la ventana principal, y la ventana que muestra la lista de herramientas analizadas.

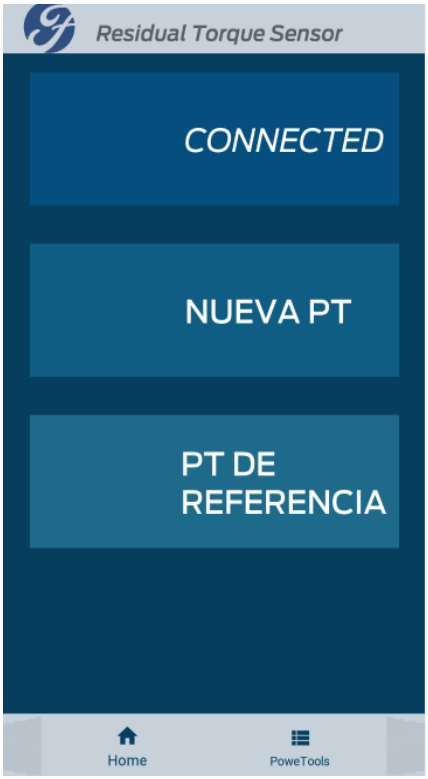


*Figura 30. Ventana Principal.*

### 5.3.2 Actividad Listar PowerTools

Esta ventana accesible desde el menú de navegación inferior de la aplicación muestra una lista de todas las herramientas analizadas. Si se selecciona cualquier herramienta de la lista, se muestra una vista más detallada de la misma. Esta lista se encuentra sincronizada en tiempo real con la base de datos Firebase, por lo que cada vez que se inicia esta actividad, la lista se actualiza con los datos del servidor.

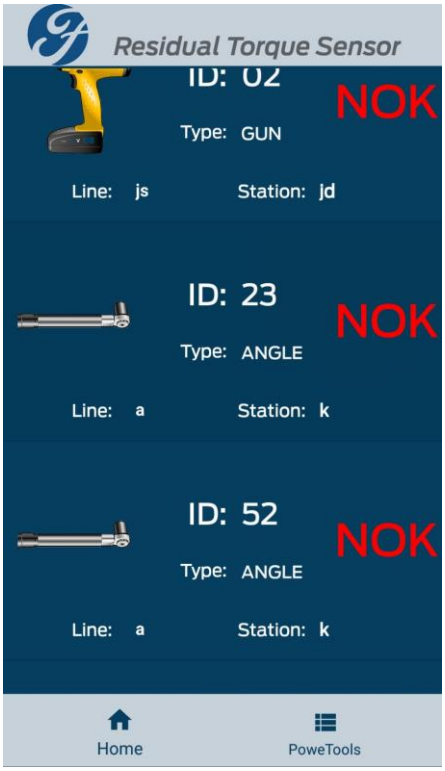


*Figura 31. Lista de PowerTools.*

### 5.3.3 Actividad Mostrar PowerTool

Esta ventana se muestra al seleccionar una herramienta de la lista de herramientas. En ella se visualiza la información completa de la misma y, además, permite volver a realizar un análisis, eliminar la herramienta del sistema, o modificar los datos de la misma a través de la ventana de modificación.

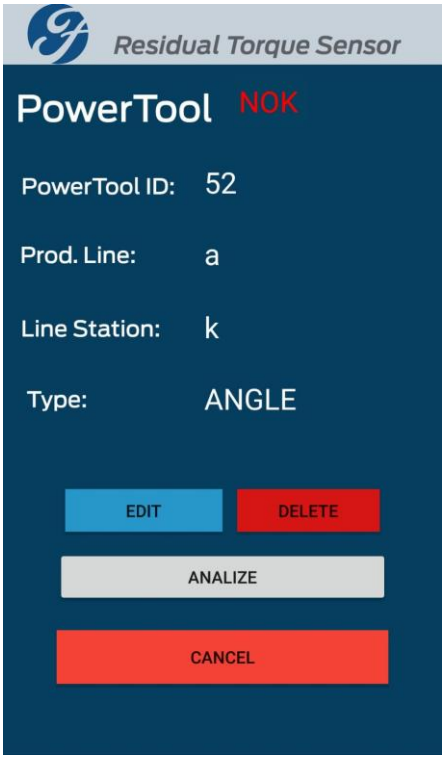


*Figura 32. Interfaz mostrar PowerTool.*

### 5.3.1 Actividad Editar PowerTool

Esta ventana, accesible a través del botón de editar de la interfaz *mostrar PowerTool*. permite editar los datos de la *PowerTool* almacenados en el sistema y guardar las modificaciones al instante.

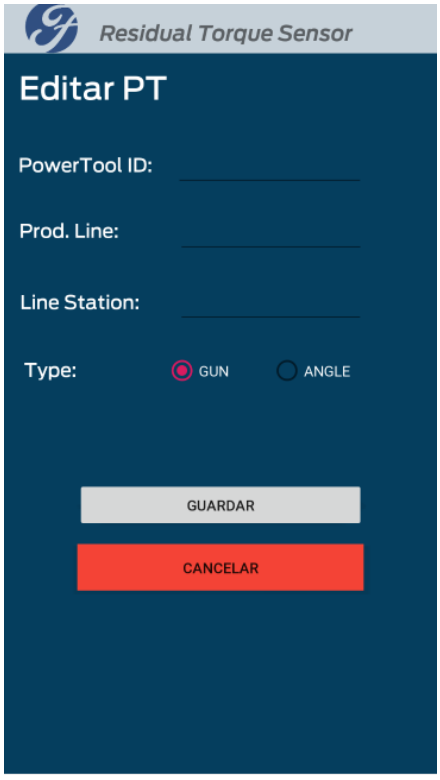


*Figura 33. Interfaz edición de PowerTool.*

### 5.3.2 Actividad Seleccionar Referencia

Esta ventana (ver Figura) es accesible desde el menú principal, y permite al usuario seleccionar de la lista de herramientas una *PowerTool.* El análisis de dicha herramienta será usado en la realización de los posteriores análisis a herramientas como una traza de referencia de la reacción de apriete, es decir, la herramienta que seleccione el usuario servirá como base para analizar las nuevas herramientas.


*Figura 34. Interfaz de selección de referencia.*

### 5.3.3 Actividad Nueva PowerTool

Para dar de alta y analizar una nueva herramienta en el sistema, se selecciona en la actividad principal dicha opción, lo cuál hace que se muestre la ventana que se presenta en la Figura 35. Esta ventana requiere que el usuario introduzca en cada campo el dato requerido para poder seguir a la siguiente actividad (Analizar PowerTool). El usuario puede cancelar el proceso de creación pulsando el botón diseñado para ello.

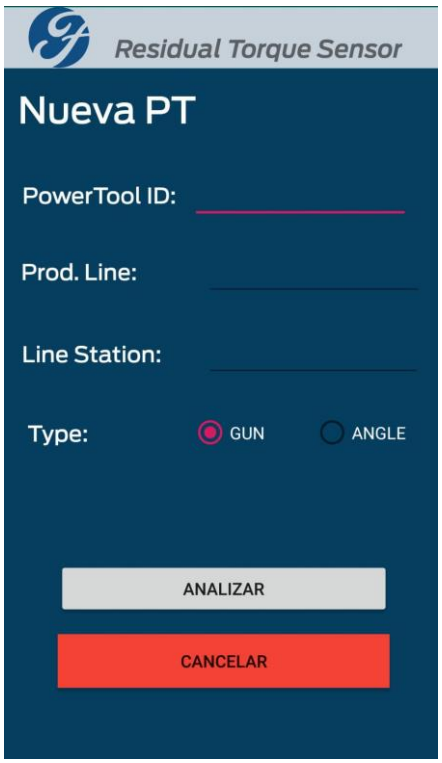


*Figura 35. Ventana de Nueva PowerTool.*

### 5.3.4 Actividad Analizar PowerTool

Esta actividad es accedida al crear una nueva PowerTool, y al volver a analizar una herramienta ya existente. En esta actividad se realiza un análisis de los datos transmitidos por el dispositivo de análisis, se comprueba la traza del movimiento de la herramienta mientras realiza el apriete, y se compara la reacción detectada con el análisis almacenado de la herramienta seleccionada como referencia.

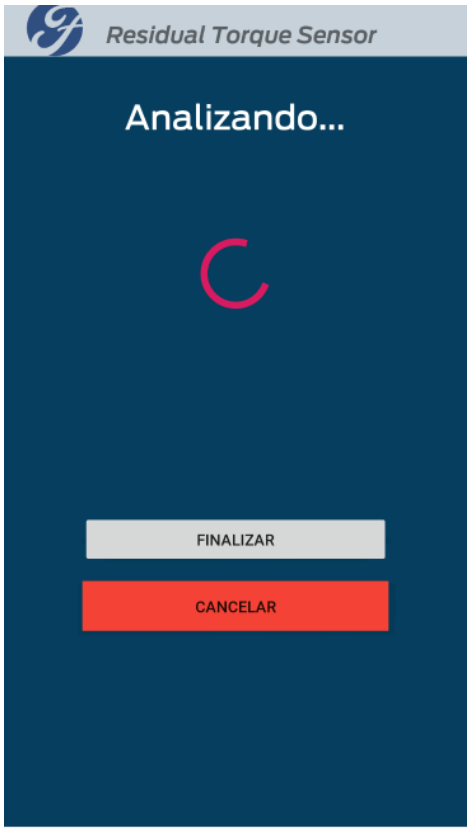


*Figura 36. Ventana de Analizar PowerTool.*

### 5.3.5 Funcionalidad de análisis de PowerTool

En la actividad de *analizar PowerTool* se realiza el análisis de la reacción que es leída en el momento por los sensores. Para ello se compara cada *Instant* recibido, es decir, cada una de las cotas de los tres ejes en ambos sensores, con los *Instant* de la herramienta seleccionada para funcionar como referencia en los análisis del sistema en ese momento.

```
ax = Float.intBitsToFloat(data[0] ^ data[1] << 8 ^ data[2] << 16 ^ data[3] << 24);
...
i.isGreaterThanAX(this.maxAX)
```

La selección de la herramienta es realizada por el usuario, por lo que es relevante que este haga una selección adecuada; el término adecuada en este caso se entiende como toda herramienta del proceso de producción que esté asignada a un apriete realizado a un mismo par, y con descripción del proceso de realización de la operación similar. La descripción del proceso de realización de la operación, como su propio nombre indica, describe cada uno de los movimientos que el operario debe hacer para

completar la operación, desde una indicación del tornillo que hay que seleccionar, a la posición en relación con el vehículo en la que se encuentra el operario que maneja la *PowerTool*.

## 5.4 Base de datos

La creación de la base de datos es un proceso sencillo, ya que el servicio proporcionado por Google adapta la estructura de los datos conforme el cliente envía la información a almacenar. Este proceso parte de la creación de un proyecto desde la página web del servicio. Para la creación del proyecto sólo es necesario dotarle de un nombre, e introducir la región o país en el que se ubica el contexto de la aplicación. Una vez se crea el proyecto, la plataforma proporciona la API que se debe referenciar en la aplicación Android, la cual permite hacer consultas y modificaciones a los datos almacenados.

La estructura de datos creada presenta una estructura que puede ser interpretada en un fichero de formato JSON (ver Figura 37) donde se identifican dos campos principales: *PowerTool*, que lista todas las herramientas del sistema, y *Referencia,* que contiene la herramienta seleccionada como referencia para los análisis. Los campos que contiene cada entrada de *PowerTool* son equiparables a los atributos de las mismas; se aprecian los campos *id*, *line*, *station* y *type*, y la lista de análisis *analises*, tal y como se ha especificado en la fase de diseño.

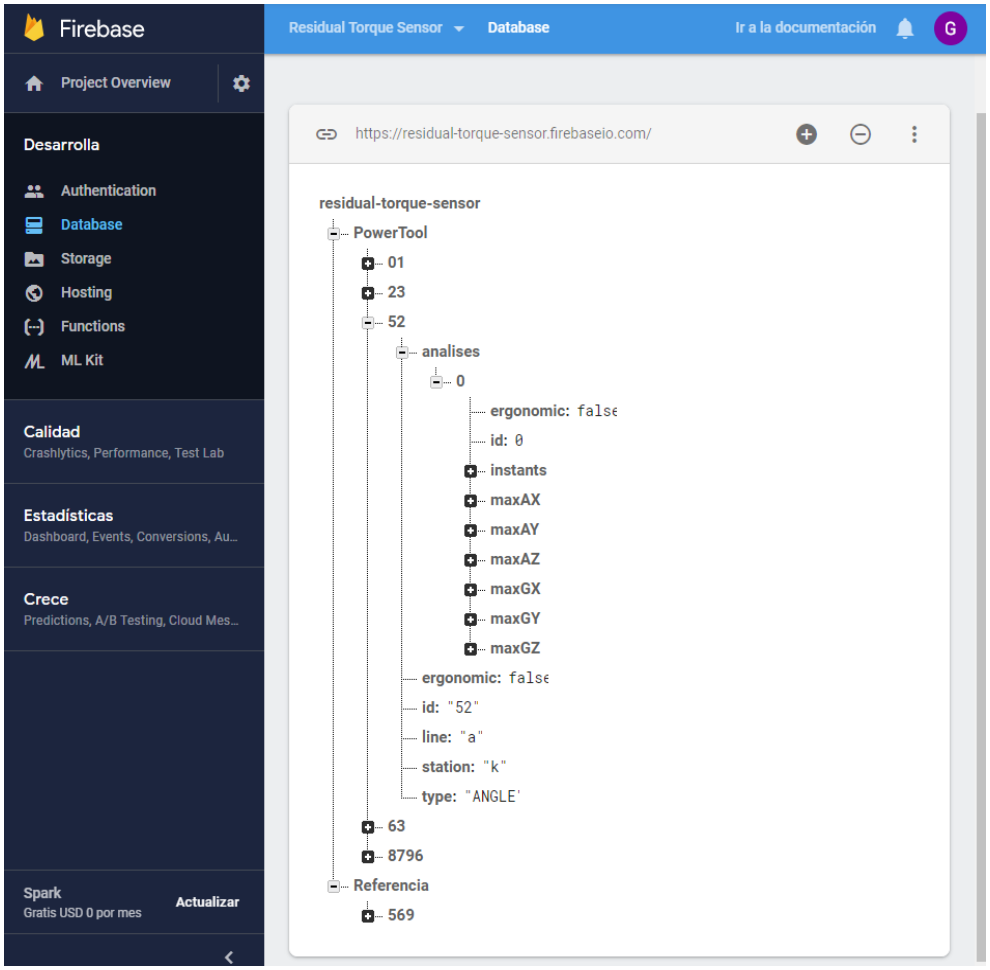


*Figura 37. Panel principal de Firebase Realtime Database.*

# 6. Validación del producto

En esta fase del proyecto se van a realizar las pruebas necesarias con el objetivo de validar que el sistema desarrollado, por una parte, cumple con los requisitos establecidos en fases más tempranas del proyecto, y, por otra parte, se quiere buscar puntos de fallo o debilidades del mismo.

Las pruebas a realizar sobre el sistema se clasifican en dos grupos:

- ***Pruebas de validación:*** comprueban que el sistema cumple correctamente ciertos casos de prueba previamente definidos, que simulan el uso previsto del sistema. En estas pruebas, el sistema se interpreta como una caja negra de la que solo se pueden comprobar los datos de salida frente a unos datos de entrada concretos.
- ***Pruebas de defectos:*** pruebas diseñadas para encontrar defectos o fallos en el sistema.

Aunque ambos enfoques persiguen metas distintas, las pruebas realizadas en la validación pueden descubrir defectos en el sistema; esto también puede pasar de forma contraria. Cabe destacar que, todo y que las pruebas permiten solucionar errores inesperados en el comportamiento del sistema, nunca podrán demostrar que el software no tiene ningún fallo, ya que existen infinidad de condiciones que provocan respuestas distintas.

A parte de las pruebas, se ha realizado una inspección del software para revisar el código desarrollado en busca de posibles mejoras. La inspección del código se considera un proceso estático, ya que no se necesita ejecutar el código a revisar. De forma complementaria a las pruebas, las inspecciones evitan que unos errores enmascaren a otros durante la ejecución del programa, por lo que no se tienen en cuenta las interacciones entre los posibles errores. Por otra parte, las inspecciones pueden orientarse a la búsqueda del cumplimiento de estándares en el código del sistema, su mantenibilidad, u otros aspectos no funcionales.

Las pruebas se realizarán en tres fases consecutivas: una primera fase llamada *pruebas de unidad*, donde se pone a prueba cada componente individual del sistema; una segunda fase llamada *pruebas de componente*, que pone a prueba el funcionamiento conjunto de varios componentes; y, por último, las *pruebas de sistema*, donde se comprueban todos los componentes trabajando juntos.

## 6.1 Pruebas de unidad

Estas pruebas están pensadas para probar el funcionamiento de componentes individuales como pueden ser métodos u objetos del sistema. En este proyecto se ha decidido probar las actividades que componen la aplicación Android, y el código desarrollado para el dispositivo de análisis.

### 6.1.1 Pruebas del dispositivo de análisis

Como se ha comentado en el apartado de implementación, el código desarrollado para el dispositivo de análisis permite comunicarse con él mediante la consola *Serial* del entorno de programación. Así pues, mediante la introducción de caracteres especiales en el *input* de la consola, se pueden activar ciertos métodos que realizan funciones especiales como puede ser la calibración de los sensores.

Los métodos del código a los que se les han realizado las pruebas son: el método de calibración de los sensores, el método que aplica el filtro de Kalman a las lecturas de los sensores, y ambos métodos *listeners* que se encargan del estado de la conexión BLE. El procedimiento para comprobar el funcionamiento de estos dos últimos consiste en conectar un dispositivo Android con una aplicación que gestione la conectividad BLE, y observar el estado de la conexión mostrado en la consola. La aplicación elegida para ello ha sido nRF Connect, una aplicación gratuita que permite conectarse a todo tipo de dispositivos Bluetooth y visualizar sus servicios y características.

Por otra parte, el correcto funcionamiento del filtro de Kalman y la calibración de los sensores se ha realizado a través de la consola *Serial*. La comprobación del primero consiste en realizar un movimiento con el dispositivo, observando las lecturas en la consola, para luego realizar el mismo movimiento, pero aplicando el filtro a las lecturas, contrastando así los datos observados y cerciorando que las lecturas son correctas. Es importante que ambos movimientos sean idénticos, o, al menos, lo más parecidos posibles con tal de no generar errores.
La comprobación de la calibración de los sensores se ha realizado de forma muy parecida a la del filtro. Se han analizado las lecturas antes y después de aplicar el calibrado de los sensores, mientras se realiza un mismo movimiento, con tal de cerciorarse si los valores leídos son coherentes con lo esperado.

### 6.1.2 Pruebas de la aplicación Android

Las pruebas realizadas a la aplicación Android consisten en la comprobación del correcto funcionamiento de las distintas Actividades. Este proceso se ha centrado en la comprobación de los elementos interactivos presentes en las actividades, como pueden ser campos de formularios y botones. Para ello, se han usado todos los elementos interactivos de cada actividad; en el caso de los campos de formulario, además, se ha comprobado que el usuario no puede introducir caracteres que no corresponden, y que los datos introducidos para cada herramienta se guardan correctamente.

## 6.2 Pruebas de componente

Estas pruebas han sido pensadas para comprobar las interacciones entre los distintos componentes del sistema. Principalmente, se han comprobado dos aspectos del sistema:

- ***Intercambio de información:*** componentes como actividades o dispositivos intercambian información entre ellos; se debe comprobar que estos intercambios se efectúan de forma correcta.
- ***Navegabilidad entre componentes:*** estas pruebas son un caso particular de la aplicación Android. La navegabilidad se basa en actividades que se invocan entre ellas. Si la navegabilidad entre ventanas de la interfaz de usuario no funciona correctamente, impedirá que el usuario realice las tareas, dejando al sistema sin utilidad.

### 6.2.1 Intercambio de información

En el sistema diseñado, los intercambios de información entre componentes se dividen en tres ámbitos, el primero, los intercambios entre el dispositivo de análisis y la aplicación Android, el segundo, los intercambios entre actividades de la aplicación Android, y el tercero, los intercambios entre la aplicación Android y la base de datos en la nube.

Los intercambios de datos producidos entre el dispositivo de análisis y la aplicación Android se producen en primera instancia, cuando ambos dispositivos establecen conexión entre ellos en el momento en que el usuario activa la búsqueda de dispositivos. Este intercambio es sencillo de comprobar ya que, si ambos programas son correctos, los dispositivos consiguen reconocerse entre sí, y establecen la conexión en apenas unos instantes. Por otra parte, el segundo contexto en que se establece un intercambio de datos es durante el proceso de análisis de la herramienta, es decir, durante la recepción de los valores de los sensores. Para su

comprobación, en el entorno de programación de ambos dispositivos se dispone de una consola que muestra los valores locales de los sensores, por lo que se pueden comparar los valores leídos por el dispositivo de análisis con los valores recibidos en el dispositivo Android, y comprobar que son iguales.

En la aplicación Android, existen dos casos en que una actividad invoca a otra transmitiendo información en el momento de la invocación. Estos casos son:

- ***Creación de una nueva herramienta:*** Al crear una nueva herramienta, la actividad que genera la ventana de formulario recoge los datos introducidos en éste y lanza la actividad que lee y analiza los valores de los sensores, enviándole al momento de lanzarla dichos datos. Este intercambio de datos se ha comprobado creando varias herramientas de prueba, y verificando que los datos se han recibido correctamente leyendo los valores en la consola del IDE(ver Figura 38).

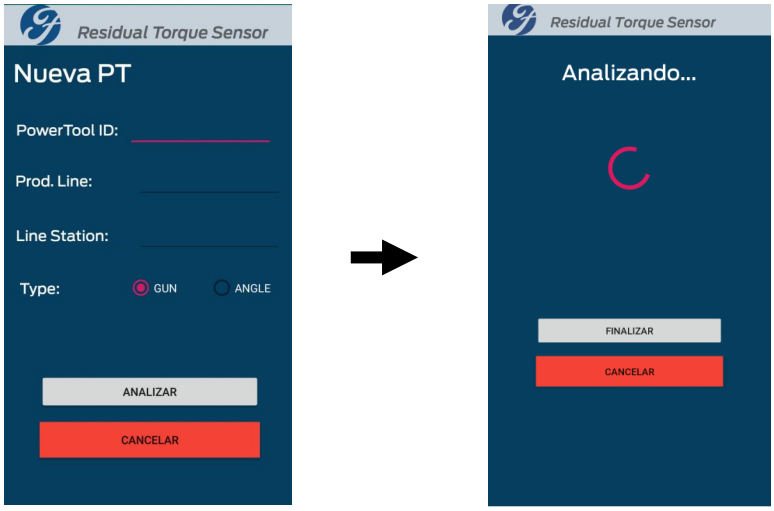


*Figura 38. Prueba de creación de nueva herramienta.*

- ***Modificación de una herramienta:*** en este segundo caso, el procedimiento es idéntico, ya que los datos se transfieren de igual forma al lanzar la nueva actividad, y la comprobación se realiza a través de la consola del IDE, al igual que se ha hecho en el caso anterior(ver Figura 39).

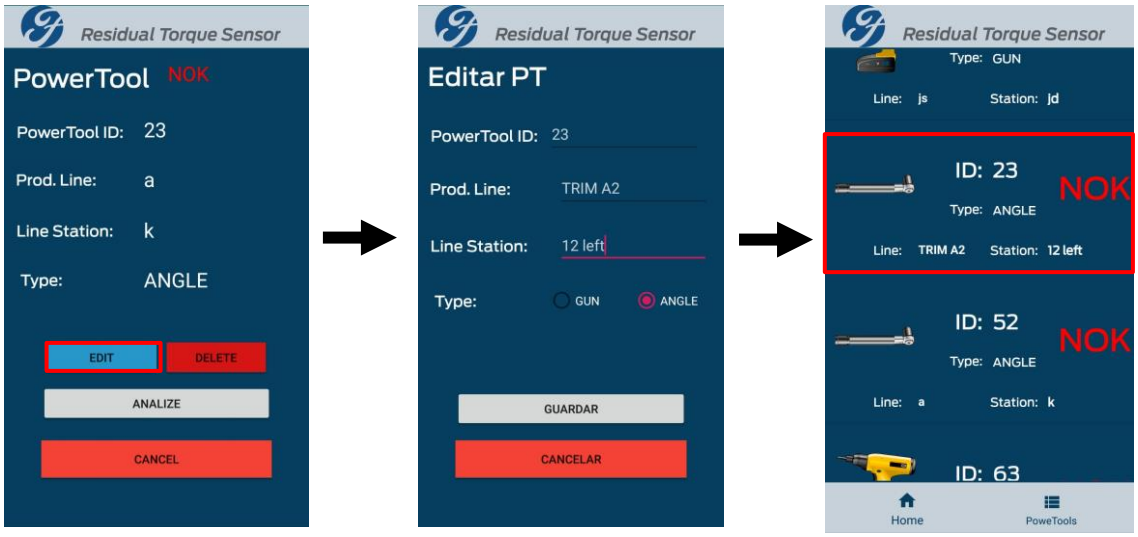


*Figura 39. Prueba de modificación de herramienta.*

Finalmente, el último contexto en que se intercambian datos es entre la aplicación Android y la base de datos. Esto sucede primero, en el momento en que finaliza el análisis de una herramienta, y la aplicación respalda el análisis realizado junto con los datos de la herramienta en la base de datos. Para su comprobación, se crean varios usos ficticios del análisis de herramientas, y se comprueba mediante el panel de control web de Firebase que los análisis se guardan con los datos indicados de forma correcta. El segundo caso se encuentra en la ventana de visualización de las *PowerTools*, en el momento en que la actividad que genera la ventana es iniciada, se sincroniza con la base de datos para recuperar cada una de las entradas (cada entrada almacena una herramienta), por lo que se han creado entradas manualmente en la base de datos, y se ha comprobado que las *PowerTools* creadas se visualizan en la interfaz de la aplicación.

## 6.3 Pruebas del sistema

En esta fase final del proyecto, se ha probado el funcionamiento completo del sistema, con todos sus componentes realizando las funciones que deben. Las pruebas se han realizado en el contexto final de uso del sistema, haciendo un análisis de herramientas al realizar aprietes. Tras varias iteraciones que consisten en la creación de herramientas con posterior análisis, modificación de las mismas, borrado, y nuevos análisis, se ha podido observar que el sistema tiene el funcionamiento deseado.

*Figura 41. Posición de la mano en la PowerTool.*

*Figura 40. Posición del dispositivo en el brazo.*

# 7. Conclusiones

El objetivo principal de este trabajo académico no era otro que garantizar que, en la línea de producción de Ford Almussafes, las herramientas de apriete usadas en las distintas estaciones que componen dicha línea se instalen de forma correcta, es decir, no sólo se busca que la herramienta realice la operación en el menor tiempo de ciclo posible, sino que, como se plantea al inicio de esta memoria, es necesario un estudio de la ergonomía de dichas operaciones por su impacto en la vida y salud de los operarios que las manejan.

Este objetivo principal se desgrana en seis objetivos secundarios tratados a lo largo de esta memoria. Primeramente, se ha realizado un análisis de la problemática surgida en los aprietes, donde se ha identificado un movimiento de reacción al finalizar cada apriete, que, dependiendo de las condiciones concretas en que se realiza, puede afectar en mayor o menor medida a la salud del operario. Además, se ha visto cómo las empresas que diseñan y fabrican este tipo de herramientas tratan de mejorar sus diseños, incorporando tecnología que minoriza dicho impacto, pero se concluye que no siempre es suficiente, ya que la complejidad y la gran casuística de los aprietes hace que no se pueda tratar todos los aprietes desde un estándar rígido.

El segundo objetivo planteado consistía en identificar las necesidades o requisitos que la solución a diseñar debería cumplir. Para ello, se han usado diferentes metodologías como pueden ser el prototipado o diseño de casos de uso que garantizan la correcta definición de requisitos, un proceso seguido de forma iterativa en el que se presentaban unas primeras vistas de las funcionalidades que el sistema debería incluir.

Así pues, el tercer objetivo planteado pretendía diseñar una posible solución al problema a través de un sistema que ayudase a identificar la ergonomía de cada apriete concreto en relación a aquellos aprietes de los que se conoce que generan impacto mínimo en el operario. En este proceso se ha definido una arquitectura para el sistema, dónde se separa al sistema en tres componentes físicos que albergan funciones distintas. Sobre estos tres componentes se han diseñado tres capas lógicas que separan las funcionalidades del *software* del sistema.

El siguiente objetivo consiste en la elección de la tecnología a usar; una vez desgranado el sistema en componentes, el proceso de selección de cada tecnología, tanto *hardware* como *software,* ha sido sencillo, ya que parte de la tecnología elegida ya era conocida.

Finalmente, el último objetivo especificaba realizar las comprobaciones necesarias para garantizar el correcto funcionamiento del sistema diseñado, cerciorándose además de que éste cumplía con los requisitos establecidos. Para ello se han realizado comprobaciones con distinto nivel de granularidad, probando primero componentes individuales, para posteriormente comprobar las comunicaciones entre estos, y, finalmente, el funcionamiento del sistema al completo.

Tras la realización del proyecto, se ha adquirido una perspectiva más completa de lo que significa desarrollar una aplicación que integra varias tecnologías, y de las posibles trabas que este proceso puede tener. Por una parte, el desarrollo del dispositivo Arduino, pese a conocer de antemano dicha plataforma, ha supuesto un reto al tener que programar una librería para implementar un filtro para los sensores, además de la comunicación con otros dispositivos.

Por otro lado, la aplicación Android ha supuesto la mayor carga temporal de todo el proyecto debido a varios factores; el primero viene dado por el poco conocimiento sobre el desarrollo de aplicaciones para el sistema operativo en cuestión, que si bien se basa en programación en un lenguaje estudiado en profundidad en la carrera (como es Java), la estructura de los proyectos que conforman las aplicaciones es compleja si se tiene que comprender mientras se desarrolla el propio software. Por otro lado, la aplicación Android establecía conexiones con todos los dispositivos, por lo que la creación de estas comunicaciones ha resultado ser un proceso iterativo en el que los errores eran constantes.

Respecto a la base de datos en la nube, pese al previo desconocimiento de la plataforma Firebase, ha resultado ser una herramienta muy potente que ha facilitado en gran medida el desarrollo del sistema.

Afortunadamente, todo reto presentado a lo largo del desarrollo del proyecto ha sido superado con éxito, y aún si no hubiese sido este el caso, el proyecto hubiese servido como un gran aporte de experiencia en el ámbito personal del alumno.

## 7.1 Trabajos futuros

Aún habiendo conseguido los objetivos planteados inicialmente para el proyecto, hay posibles vías de mejora del sistema, funcionalidades que no se han podido llevar a cabo debido principalmente a la falta de tiempo. A continuación se presentan algunas de las mejoras pensadas:

- ***Visualización de gráficos de análisis:*** pese a almacenar cada uno de los análisis, la aplicación no lleva incorporada una herramienta que muestre al detalle y de forma gráfica los resultados que el sensor ha proporcionado, ya que durante el desarrollo ha habido que solucionar varios errores de la interfaz de la aplicación, que han conllevado demasiado tiempo de desarrollo.
  Sería interesante en próximas actualizaciones incluir un *dashboard* o tablón de visualización de gráficos en la aplicación, ya que existen librerías destinadas a ello.
- ***Versión web***: al disponer de la base de datos en la nube, se podría crear un servicio web que permitiese listar las *PowerTools* ya analizadas, pudiendo modificar sus datos y visualizar los análisis; sin embargo, esta versión web no contaría con la opción de analizar herramientas, ya que se necesita conectarse al dispositivo de análisis mediante BLE. El propio servicio Firebase proporciona en una de sus variantes un servicio de *hosting* web que puede resultar útil para esta implementación. Además, existen otros servicios web especializados en creación de *dashboards* a partir de datos recibidos.

# Bibliografía

AENOR. (2013). *UNE-EN ISO 11148-6:2012 Hand-held non-electric power tools - Safety requirements - Part 6: Assembly power.* : .

Aguado Ullate, M., & Ursua Rubio, A. (2015). *Implantación de sistema de trazabilidad mediante herramientas electrónicas.* Pamplona: Escuela Técnica Superior de Ingeniería Industrial, Informática y de Telecomunicación.

Android Developers. (2019). *Documentation Bluetooth*. Recuperado el 20 de 06 de 2019, de https://developer.android.com/guide/topics/connectivity/bluetooth

Arduino. (2019). *Curie BLE Library*. Recuperado el 20 de 06 de 2019, de https://www.arduino.cc/en/Reference/CurieBLE

Arduino. (2019). *Genuino 101*. Recuperado el 02 de 06 de 2019, de https://store.arduino.cc/genuino-101

Atlas Copco. (2015). *The art of ergonomics.* Sweden: Atlas Copco Industrial Technique AB.

AtlasCopco. (2015). *Pocket guide to tightening technique.* Sweden: AtlasCopco Industrial Technique AB.

AtlasCopco. (2016). *Pocket guide to screwdriving.* Sweden: AtlasCopco Industrial Technique AB.

Barkely Graham, B. (2000). *Using an Accelerometer Sensor to Measure Human Hand Motion.* Massachusetts: Massachusetts Institute of Technology.

Bosch. (2019). *Bosch Sensotec*. Recuperado el 17 de 06 de 2019, de https://www.bosch-sensortec.com/bst/products/all_products/bmi160

contributors, W. (06 de 06 de 2019). *Ford Valencia Body and Assembly - Wikipedia*. Recuperado el 01 de 07 de 2019, de https://en.wikipedia.org/wiki/Ford_Valencia_Body_and_Assembly

Davidson, R., Townsend, K., & Cufí, C. (2014). *Getting Started with Bluetooth Low Energy.* O'Reilly Media.

Debrauwer, L., & Heyde, F. v. (2013). *UML 2 [Recurso electrónico-En línea] : iniciación, ejemplos y ejercicios corregidos.* Cornellà de Llobregat: Ediciones ENI.

Delgado Díaz, J., & Bolufer Catalá, E. (2011). *Ergonomía física en obra: lesiones producidas e instrumentos para mejorarla.* Valencia: Escuela Técnica Superior de Ingeniería de Edificación .

Gomez, C., Oller, J., & Paradells, J. (2012). *Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology.* Valencia: Universitat Politècnica de Catalunya/Fundació i2Cat.

Google Inc. (2019). *Firebase - Protocolo HTTP de Firebase Cloud Messaging*. Recuperado el 21 de 06 de 2019, de https://firebase.google.com/docs/cloud-messaging/http-server-ref?hl=es

Intel Corporation. (2017). *Intel Curie Module Datasheet.* Intel Corporation.

Lamar, C. (2005). *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and Iterative Development.* London: Prentice Hall PTR.

Maner, W. (1997). *Sidar: Prototipado*. Recuperado el 03 de 06 de 2019, de http://www.sidar.org/recur/desdi/traduc/es/visitable/maner/Prototipado.htm

Rafols Montane, R. (2016). *Learning Android Application Development.* Packt Publishing.

Ramos Cardozzo, D. (2016). *Desarrollo de Software: Requisitos, Estimaciones y Análisis.* It Campus Academy.

Semiconductor, N. (2019). *nRF51822*. Recuperado el 23 de 06 de 2019, de https://www.nordicsemi.com/?sc_itemid=%7BE343E4D9-21F1-4FBC-881F-10320A687576%7D

Sommerville, I. (2011). *Ingeniería de software.* Pearson Educación .

Tereshkov, V. (2013). An Intuitive Approach to Inertial Sensor Bias Estimation. *Cornell University*, 1-6.

Tinoco Gómez, O., Rosales López, P. P., & Salas Bacalla, J. (2010). Criterios de selección de metodologías de desarrollo de software. *Dyna*, 70-74.

Villaluenga Morán, J. (2015). *Uso de acelerómetros para el control de dispositivos mediante captura de movimiento.* Barcelona: Univesitat Ramon Llull.

Zapata, C., & Carmona, N. (2010). Un modelo de diálogo para la educción de requisistos de software. *Dyna*, 2019 - 2019.

# Glosario

A continuación se presentan las abreviaturas presentes en esta memoria, en orden de aparición:

ISO – International organization for Standardization

UML – Unified Modeling Language

MEMS – Microelectromechanical Systems

IMU – Inertial Measurement Unit

CPU – Central Processing Unit

RISC – Reduced Instruction Set Computer

RAM – Random Access Memory

SoC – System on Chip

AES – Advanced Encryption Standard

UUID – Universal Unique Identifyer