UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UN SISTEMA DE REALIMENTACIÓN DE CORRIENTE VECTORIAL PARA BANCO DE ENSAYOS DE ACCIONAMIENTOS ELÉCTRICOS

AUTOR: DANIEL GARCÍA DEL BUEY

TUTOR: JAVIER ANDRÉS MARTÍNEZ ROMÁN

COTUTOR: ÁNGEL SAPENA BAÑO

Curso Académico: 2018-19

RESUMEN

Sin duda las máquinas de accionamientos eléctricos son una de las piezas claves en la industria y hasta en nuestra vida diaria, más aún si tenemos en cuenta el reciente auge de los vehículos eléctricos. Es esta la razón por la que se ha decidido poner en marcha un proyecto que implementará un control de realimentación de corriente vectorial por orientación por campo. El proyecto está pensado que para implementarse en el futuro en un banco de ensayos y se pueda usar para mostrar a los alumnos el funcionamiento de este sistema de control. Aunque en este TFG solo se realizará una parte de este proyecto

En este TFG se medirán las corrientes de un motor asíncrono y se realizarán los cálculos necesarios para transformar la información en una señal útil que pueda realimentarse al control. Estas medidas y cálculos se realizarán con un microcontrolador.

Pero además de implementar un sistema de control es necesario contar con una interfaz que nos permita interactuar con el control, para esto se ha decidido usar un dispositivo móvil gobernado por el sistema operativo Android, que es con diferencia el sistema operativo más usado para dispositivos móviles. Con la aplicación desarrollada se podrá tanto enviar órdenes al sistema de control como mostrar los datos calculados por el microcontrolador en un histórico.

Así pues, los puntos clave de este TFG son:

- Programar un microcontrolador para que capture la señal de corriente de las fases del estator y del rotor de una máquina de inducción y la procese para convertirla en la señal de realimentación del sistema de control por orientación de campo.
- Programar una aplicación de Android que pueda comunicarse con el microcontrolador para hacer de interfaz gráfica, recibiendo las órdenes del usuario para captura de corrientes y enviándolas al microcontrolador y recibiendo de éste los datos capturados y mostrándolos de forma visual al usuario.
- Construir el prototipo que nos permita llevar el proyecto la práctica, integrando sensores de corriente, preparando las conexiones con el motor y con aparatos de medida externos y con el microcontrolador.

ABSTRACT

Electric machines have become one of the most important elements of the industry and even of our daily life, even more if we keep in mind the continuous growth of the electric vehicle market. This is the main reason why we have decided to develop a project that consists on a feedback control system of vector current, a Field oriented control, to be precise. It is planned to implement this project on a test bench so the students can use it on the future and learn from it. Although only one part of the feedback control system will be covered in this final degree project.

In this final degree project we will measure the current of an asynchronous motor and operate to transform the measurements in the feedback information the control system uses. This measures and operations will be made with a microcontroller.

But a graphic interface is also needed to communicate with the microcontroller. We will use a mobile device running the Android OS to accomplish this, as Android is by far the most used OS on mobile devices. The application that we will develop will be able to send orders to the control system and also receive the information the microcontroller calculates and show it in a plot.

The main aspects of this final degree project are:

- Programming a microcontroller so it can measure the stator and rotor phase currents of an inductive machine and calculate with them the feedback signal for the Field oriented control system.
- Programming an Android application that will communicate with the microcontroller so it can be the graphical interface. It will receive the orders from the user and send them to the microcontroller and also receive the captured data of the microcontroller and show it in a plot to the user.
- Building the prototype so that the project can be put into practice. To build the prototype we will need to integrate current sensors and prepare connections with the inductive motor, external measuring devices and the microcontroller

RESUM

Sens dubte les màquines d'accionaments elèctrics són una de les peces claus en la indústria i fins en la nostra vida diària, més encara si tenim en compte el recent auge dels vehicles elèctrics. És esta la raó per la qual s'ha decidit posar en marxa un projecte que implementarà un control de realimentació de corrent vectorial per orientació per camp. El projecte està pensat que per a implementar-se en el futur en un banc d'assajos i es puga usar per a mostrar als alumnes el funcionament d'este sistema de control. Encara que en este TFG només es realitzarà una part d'esta projecte.

En este TFG es mesuraran els corrents d'un motor asíncron i es realitzaran els càlculs necessaris per a transformar la informació en un senyal útil que puga realimentar-se al control. Estes mesures i càlculs es realitzaran amb un microcontrolador.

Però a més d'implementar un sistema de control és necessari comptar amb una interfície que ens permeta interactuar amb el control, per a açò s'ha decidit usar un dispositiu mòbil governat pel sistema operatiu Android, que és amb diferència el sistema operatiu més usat per a dispositius mòbils. Amb l'aplicació desenrotllada es podrà tant d'enviar ordes al sistema de control com mostrar les dades calculats pel microcontrolador en un històric.

Així, doncs, els nuclis d'este TFG són:

- Programar un microcontrolador perquè capture el senyal de corrent de les fases de l'estator i del rotor d'una màquina d'inducció i la processe per a convertir-la en el senyal de realimentació del sistema de control per orientació de camp.
- Programar una aplicació d'Android que puga comunicar-se amb el microcontrolador per a fer d'interfície gràfica, rebent les ordes de l'usuari per a captura de corrents i enviant-les al microcontrolador i rebent d'estes les dades capturats i mostrant-los de forma visual a l'usuari.
- Construir el prototip que ens permeta portar el projecte la pràctica, integrant sensors de corrent, preparant les connexions amb el motor i amb aparells de mesura externs i amb el microcontrolador.

MEMORIA DESCRIPTIVA

Índice de la Memoria

Índice de Figuras

1 INTRODUCCIÓN

No cabe duda de que los motores eléctricos son uno de los pilares sobre los que se sustenta la sociedad moderna. Se usan en los hogares en aplicaciones como juguetes, electrodomésticos o domótica, se usan en las fábricas para casi cualquier máquina que requiera de potencia mecánica, y en la industria automovilística se están viendo cada vez más vehículos equipados exclusivamente con este tipo de motores, y es una tendencia que no parece que vaya a desaparecer. Sin embargo, los motores eléctricos no habrían llegado a tantas aplicaciones si no fuese por un aspecto de vital importancia que normalmente es pasado por alto, el sistema de control. Es gracias a los avances en sistemas de control que hoy día se pueden usar los motores eléctricos para aplicaciones tan interesantes como brazos robóticos, drones o hasta impresoras 3D. Sin embargo, aún quedan muchos métodos de control avanzados que no han sido implementados a gran escala todavía, este es el caso del control por orientación de campo o FOC (Field Oriented Control).

El control por orientación de campo es un sistema avanzado de control para motores de corriente alterna que se basa en mantener los enlaces de flujo ligados al rotor constantes. Un motor con otro sistema de control desperdicia muchos recursos variando el flujo, mientras que manteniéndolo constante se puede mejorar enormemente su respuesta ante transitorios, acelerando y decelerando lo más rápido posible y aportando gran par en un rango de velocidades mucho mayor. Esto es interesante para aplicaciones de altas especificaciones pero también para el uso general, pues aumenta las prestaciones del motor. Para llevar a cabo este sistema de control se necesita diferenciar que componente de corriente controla el par, y cuál el flujo, y, para calcular esto, necesitaremos de suficiente potencia de cálculo.

El mundo de la informática y la electrónica evoluciona a toda velocidad y actualmente podemos encontrar microcontroladores con una enorme potencia de cálculo a precios irrisorios. En este TFG se utilizará un microcontrolador con soporte en Arduino: esto permitirá hacer uso del Arduino IDE, un programa gratuito y de código abierto, para programar el microcontrolador. Además, Arduino tiene una enorme comunidad de usuarios dispuesta a ayudar en caso de tener cualquier duda, lo que facilitará el desarrollo y la implementación del código.

Tenemos un sistema de control que implementar y un hardware capaz de realizarlo, pero aún nos falta un importante detalle, una interfaz gráfica sencilla e intuitiva que permita comunicar al usuario con el sistema de control, y ahí es donde entra Android. Android es con diferencia el sistema operativo móvil más usado actualmente, está basado en Linux, cuenta con varias suite de desarrollo de aplicaciones gratuitas como Android Studio y, de forma similar a Arduino, cuenta con una enorme comunidad de desarrolladores dispuesta a resolver cualquier problema. En este TFG se usará Android Studio para desarrollar una aplicación que permita usar un dispositivo móvil con el sistema operativo Android como la interfaz gráfica del sistema de control.

En resumen, estamos ante un TFG realmente complejo que abarca una gran cantidad de temas dispares, el funcionamiento de un motor eléctrico, la implementación de un sistema de control realimentado en un microcontrolador con soporte en Arduino, el desarrollo de una aplicación de Android y por último, la comunicación entre el dispositivo Android y el microcontrolador. Debido a esta complejidad se decidió dividir la implementación del banco de ensayos en tres TFGs distintos, de los que, evidentemente, solo uno de ellos será tratado en esta memoria.

La parte que se desarrollará en este TFG consiste en la medición y cálculo de los fasores de corriente de estator y rotor del motor a ensayar. Esta parte es clave pues conocer la posición y módulo de los fasores permite saber si se está operando la máquina en las condiciones establecidas por el sistema de operación tanto en términos de par como de enlaces de flujo, es decir, las componentes de los fasores son la señal que realimenta el sistema de control. Además de esto, se diseñará la aplicación de Android ya mencionada para mostrar la evolución de los fasores al usuario y que éste pueda apreciar fácilmente los cambios en el comportamiento del motor.

Es importante destacar que todo lo explicado en este TFG ha sido llevado a la práctica en un banco de ensayos configurable de accionamientos eléctricos y montado en el laboratorio por lo que todos los datos mostrados se han obtenido de un motor real y el proyecto está listo para funcionar. El trabajo desarrollado en este TFG está previsto que se utilice a partir del curso 19-20, en una primera etapa para el aprendizaje basado en proyectos en la asignatura de Análisis dinámico y control de accionamientos eléctricos y, más adelante, para introducir esta metodología también en la asignatura Ampliación de Máquinas Eléctricas, ambas del Master en Ingeniería Industrial, en su intensificación de Electricidad.

1.1 Estructura de la memoria

La memoria se divide en varias secciones, a continuación se resumen los temas tratados en cada una de ellas.

1. En la Introducción se detalla el sistema de control que se va a implementar, cuáles son sus beneficios y por qué se va a implementar mediante un microcontrolador y un dispositivo Android.
2. En Alcance del proyecto se detallan los objetivos que se deben cumplir para dar el proyecto como finalizado y cómo se van a alcanzar esos objetivos.
3. En Elección del microcontrolador se detallan las características mínimas que se le exigen al microcontrolador empleado y cuál se ha escogido para este proyecto.
4. En Medida de corrientes se detallan las distintas tecnologías de sensores de corriente que se evaluaron para este proyecto y cuál fue el sensor escogido.
5. En Montaje del prototipo se detallan los materiales empleados para construir el prototipo y como se deben realizar las distintas conexiones y otros detalles del motaje. También se esplica el ensayo de calibración de los sensores de corriente.

6. En Cálculo de los fasores se detalla la justificación teórica del sistema de control empleado, el control por orientación de campo, y el código del microcontrolador que realiza los cálculos pertinentes, además del ensayo de calibración del codificador rotativo.
7. En Interfaz de usuario se detallan las dos alternativas de comunicación inalámbrica consideradas en el proyecto, cómo se implementa el código del microcontrolador que maneja las comunicaciones y el desarrollo de la aplicación de Android que hace de interfaz.
8. En los Resultados se detallan los datos obtenidos una vez el trabajo ha sido completado y se confirma su correcto funcionamiento.
9. En Conclusiones se repasan todos los aspectos tratados en este TFG y la utilidad que tendrá esta proyecto en el futuro cercano.
10. Finalmente, en la Bibliografía se reúnen los distintos materiales de referencia usados en la elaboración del TFG, así como las páginas oficiales de los distintos fabricantes mencionados.

2 ALCANCE DEL PROYECTO

El proyecto al que nos enfrentamos es un importante desafío que consiste en medir la posición, velocidad y corrientes del estator y rotor y usar los datos obtenidos para establecer un método de control basado en los vectores de corriente para máquinas de accionamientos eléctricos. Este método de control nos permite cambiar rápidamente el par aportado por la máquina sin perder recursos en variar el flujo magnético. Se puede aplicar a varias máquinas de accionamientos eléctricos distintas como motores asíncronos y síncronos de distintas potencias, pero en nuestro caso lo aplicaremos a un motor asíncrono de 600W.

El control se realizará con un microcontrolador utilizando el entorno de desarrollo Arduino. Y, a parte del control, también se debe desarrollar una aplicación de Android que sea capaz de comunicarse con el ya mencionado microcontrolador para transmitirle los comandos de operación del usuario y transmitir a éste el estado del accionamiento mediante una interfaz de usuario intuitiva en un dispositivo móvil.

Como es un proyecto complejo se decidió dividirlo en tres TFGs distintos. Uno de ellos se encargará de obtener la posición y la velocidad del rotor. El que se detalla en esta misma memoria se encarga de medir las corrientes y realizar los cálculos pertinentes para conseguir datos útiles de ellas: las componentes de los vectores espaciales de corriente de estator y de rotor en coordenadas de campo de rotor. Finalmente, el tercero se encarga de usar los datos obtenidos gracias a los anteriores TFGs para compararlos con las demandas de operación y así establecer el control del accionamiento. En todos ellos se desarrollará una aplicación de Android distinta adaptada al trabajo desarrollado, siendo el objetivo final del proyecto, pero no de este TFG, combinarlas en una sola aplicación que cuente con todas las funciones de las anteriores.

Más concretamente las tareas de las que se encarga este TFG consisten en:

- Elegir un sensor de corriente adecuado
- Calcular los fasores de corriente en los ejes de referencia d,q
- Medir las corrientes de rotor y estator con sensores de corriente apropiados
- Establecer un sistema de comunicación entre el microcontrolador y la aplicación Android que servirá de interfaz de usuario del sistema
- Desarrollar una aplicación de Android que muestre los fasores calculados en una gráfica en tiempo real

Finalmente hay algunas partes que es importante discutir en esta memoria aunque no sean exclusivas de este TFG, pues afectan al proyecto completo.

- Justificación teórica de cuáles son las ventajas de este tipo de control respecto a controles más sencillos
- Elegir un microcontrolador apropiado para el proyecto
- Elegir un sistema de comunicación inalámbrica entre el microcontrolador y la aplicación de Android adecuado para el proyecto

3 ELECCIÓN DEL MICROCONTROLADOR

Elegir un microcontrolador apropiado para el proyecto es realmente importante, pues todos los componentes deben ser capaces de conectarse a la placa y se necesita que realice todos los cálculos pertinentes a la suficiente velocidad.

Concretamente, el microcontrolador debe disponer de patillas suficientes (y del tipo necesario) para conectarse a todos los componentes, de un sistema de comunicación inalámbrico adecuado para conectarse a la interfaz de usuario, y de velocidad de cálculo suficiente para poder realizar el control sin que el retraso en la acción de control pueda poner en peligro los equipos por resultar un sistema con una respuesta demasiado lenta.

Se decidió usar desde el principio un microcontrolador con soporte en Arduino por que dispone de una enorme comunidad de usuarios, lo que facilita encontrar cualquier tipo de información y resolver cualquier duda rápidamente. Además, dispone del entorno de programación Arduino IDE, de código abierto y totalmente gratuito.

En un primer momento se usó la placa esp8266, creada por Espressif (Espressif Systems, 2019), que se puede encontrar fácilmente por menos de 3€, dispone de un módulo wifi integrado, 16 pines de propósito general (10 de los cuales permiten salida PWM), interfaz SPI e I2C, y un pin ADC de 10 bits, entre otras características. Esta placa tiene dos inconvenientes importantes: solo dispone de un pin ADC, por lo que obliga a usar un chip auxiliar como el MCP3208 para realizar la conversión analógico-digital entre los sensores de corriente y la placa y sólo dispone de conexión WiFi como comunicación inalámbrica, por lo que habría que adquirir un módulo Bluetooth como el HC-06 en caso de querer usar este tipo de comunicación, con el consiguiente aumento en el presupuesto y en el número de patillas usadas.

Debido a estos inconvenientes se decidió usar el microcontrolador esp32, también creado por Espressif y el sucesor directo del esp8266, sus únicos inconvenientes son que es ligeramente más caro que el anterior, aunque se puede encontrar por menos de 5€ y que al ser más nuevo es más complicado encontrar información sobre él. Pero cuenta con numerosas ventajas que lo hacen más apropiado para el proyecto. Dispone de Bluetooth incorporado, tiene un procesador más potente que además cuenta con dos núcleos y puedes usar hasta 18 pines para medir tensiones analógicas, entre muchas otras mejoras. Estas ventajas superan por mucho a las desventajas y nos permiten realizar el montaje de forma más sencilla y efectiva.

3.1 Requisitos de Velocidad de Cálculo

Como se ha mencionado anteriormente es necesario que el microcontrolador sea lo suficientemente rápido realizando los cálculos, afortunadamente el esp32 es suficientemente potente, pero conviene profundizar más en la razón por la que es necesario tener esto en cuenta.

Si tan solo tenemos en cuenta las funciones que se profundizan en este TFG no es necesaria mucha velocidad de cálculo, pues medir las corrientes de fase, calcular los fasores, y mostrarlos al usuario por medio de una interfaz no son tareas críticas y basta con enviar los datos a unos 5Hz, es decir, cada 200 milisegundos. Si queremos que la información se muestre al usuario de forma más fluida se puede tratar de reducir este tiempo, sin embargo no es muy recomendable, pues si se envían los datos demasiado rápido, la interfaz gráfica puede llegar a bloquearse. Por lo tanto se establecieron los 200 milisegundos como tiempo de espera entre un envío de datos y otro

Sin embargo, no se debe olvidar que este TFG forma parte de un proyecto más grande que debe encargarse de controlar una máquina de accionamientos eléctricos generando las señales de control al inversor de alimentación de la máquina, y evidentemente controlar la corriente cada 200 milisegundos es inadmisible. El requisito de tiempo dependerá de la máquina en concreto que deseemos controlar, en nuestro caso vamos a alimentar un motor asíncrono de 600W, teniendo en cuenta las características de este tipo de motores se ha determinado que para garantizar el correcto funcionamiento del motor se deben calcular las variables asociadas a su corriente de alimentación en, como máximo, 500 microsegundos.

Gracias a la función micros() del Arduino IDE podemos calcular el tiempo que tarda el esp32 en realizar cualquier cálculo. Según esta función el esp32 tarda unos 10-12 microsegundos en realizar una sola medida de tensión con las patillas ADC, como se explicará más adelante, es necesario realizar medidas de cuatro sensores distintos, por lo que perdemos unos 45 microsegundos solo en tomar los datos necesarios. Y en total, es decir, incluyendo la medida de posición y los cálculos detallados en la sección 6 se tardan alrededor de 150 microsegundos, eso significa que nos quedan 350 microsegundos disponibles para realizar los cálculos de velocidad y generar la señal de salida que controla el motor, estos últimos cálculos no forman parte de este TFG, pero según los encargados de su desarrollo no tardan más de 100 microsegundos. Por lo tanto, el esp32 es lo suficientemente rápido como para controlar el motor sin peligro. Además, gracias al doble núcleo de la esp32 se pueden realizar las comunicaciones con la interfaz sin producir retrasos en los cálculos, pues de los cálculos se encarga un núcleo, mientras que de las comunicaciones se encarga el otro.

En resumen, enviaremos los datos de los fasores a la interfaz cada 200 milisegundos, pues no es realmente necesario mostrar los datos al usuario a gran resolución. Sin embargo los cálculos se realizarán en menos de 500 microsegundos, pues es necesario actualizar la corriente de alimentación lo suficientemente a menudo para garantizar el correcto funcionamiento del sistema de control.

Otro punto importante respecto a la velocidad es tratar de medir todos los datos en el mismo instante, esto es imposible con el esp32, pues las medidas las realiza de forma secuencial. Sin embargo, el retraso entre la primera y la última medida es asumible. La secuencia de medidas funciona de la siguiente forma: primero se mide el tiempo, esta medida es prácticamente instantánea, luego se miden las 4 corrientes, tardando en total los 45 microsegundos mencionados anteriormente, finalmente se mide la posición del eje, esta medida se realiza con una función desarrollada en otro TFG, y tarda 10 microsegundos.

Es decir, entre la primera medida de corriente y la medida de posición se tardan alrededor de 55 microsegundos. Para saber si esto es aceptable se compara este tiempo con el ángulo de giro de los fasores que transcurre durante ese tiempo. Pensemos que el motor trabaja a su velocidad de alimentación, que son 50Hz, esto significa que el fasor tarda 20000 microsegundos en realizar una vuelta entera, y si dividimos entre 360 descubrimos que el tiempo que tarda en girar un solo grado es de 55.56 microsegundos. Curiosamente el tiempo que tarda un fasor de 50 Hz en girar un grado es muy similar al tiempo que tarda el esp32 en realizar todas las medidas, esto significa que, a menos que se usen corrientes de mayor frecuencia que las normales, el desvío ocasionado en los fasores por no ser capaz de realizar todas las medidas en el mismo instante es de como mucho un solo grado. Durante un cambio de régimen del motor los fasores pueden variar 20 grados, por lo que podemos tomar este grado como un error asumible.

Es más, esa diferencia máxima de 1º corresponde al tiempo que gira el fasor del estator en el peor de los casos durante el tiempo que pasa entre la captura de la primera señal y la de la última. En realidad, ese avance de fase sería muchísimo menor para el fasor de corriente del rotor, que gira unas 20 veces más despacio. En la práctica, si tomamos el avance de fase entre el fasor del estator y el del rotor (que prácticamente no gira) ese valor pasa a ser de sólo unos 0,4º. Esta diferencia es conveniente ponerla en contexto con las prestaciones que obtendríamos con equipos más potentes y costosos. Así, con el ESP32 llegamos a tener una desviación máxima de 0,4º con un precio por equipo de unos 5€ mientras que con equipos que permitan un muestreo simultáneo de 4 canales de corriente podríamos reducir esa desviación al tiempo necesario para tomar una única medida de posición o de corriente, unos 5 microsegundos equivalentes a 0,04º. El problema es que ese tipo de equipos tienen un coste unitario en el rango de los 1500€, lo que confirma la adecuación de la solución elegida.

4 MEDIDA DE CORRIENTES

Elegir correctamente la forma de medida de corriente más adecuada es uno de los aspectos más importantes del proyecto, pues si no se obtienen correctamente las corrientes desde el principio, todos los esfuerzos siguientes serán inútiles. Hay muchas tecnologías distintas que nos permiten medir corriente, en esta sección se repasarán las principales.

Necesitamos sensores capaces de medir a bajas frecuencias, pues mientras que el estator trabaja a la frecuencia a la que sea alimentado, 50 Hz en los casos más normales, pero también frecuencias muy bajas y hasta el doble de esta, ya que se trata de un banco de ensayos de accionamientos regulados en velocidad, el fasor del rotor trabaja a frecuencias mucho más bajas del orden de los 1-5 Hz, en función de la carga. Además usar sensores capaces de medir corriente a bajas frecuencias da más flexibilidad al proyecto y nos permite alimentar motores a frecuencias mucho menores. También se necesita que los sensores sean capaces de responder rápidamente a los cambios de corriente, para poder realizar la función de realimentación de forma efectiva, que sean resistentes al ruido, pues se encontrarán en un entorno con

corrientes y campos magnéticos relativamente grandes, que ofrezcan aislamiento galvánico para que faciliten el diseño de la instrumentación y finalmente, que sean sencillos de implementar para que se puedan integrar de forma sencilla en un sistema basado en el esp32.

4.1 Resistencia en Serie "shunt"

Es la forma más sencilla de medir corriente, se basa en una resistencia de muy bajo valor y de la cuál sabemos con gran exactitud su valor real. Esta resistencia es atravesada por la corriente que deseamos medir, medimos la caída de tensión entre los extremos de la resistencia y obtenemos la intensidad simplemente multiplicando la caída de tensión por el valor de la resistencia. Esta forma de medida se basa en la ley de Ohm y, en principio, permite alcanzar fácilmente uno de los requisitos indicados, al ser factible conseguir una respuesta en frecuencia muy buena, desde corriente continua hasta varias decenas de kHz sin tener que preocuparse de las capacidades e inductancias parásitas.

Es un método de medida muy barato pero que no ofrece aislamiento galvánico y obliga a llegar a un compromiso entre potencia consumida y calidad de la medida, ya que resistencias de bajo valor suponen unas bajas pérdidas pero tienen asociada una caída de tensión menor que será más sensible al ruido. Además se necesitarían 4 resistencias, todas ellas con referencias de tensión distintas, y del orden de los 200V: el esp32, al no disponer de entradas aisladas, es incapaz de realizar esta clase de medidas sin electrónica extra de apoyo (serían necesarias etapas de aislamiento entre cada sensor resistivo y las entradas analógicas del esp32), que encarecería y haría mucho más complejo el proyecto, por lo que se descartó esta tecnología.

Figura 1. Resistencia de medida "shunt" de RC Electronics

En la Figura 1 se muestra una resistencia especialmente diseñada para este uso, este modelo en concreto aguanta hasta 100 Amperios y la produce RC Electronics (RC Electronics, 2019), aunque ya se ha descontinuado su venta.

4.2 Transformador de Corriente

Otra forma de medir corrientes es usar un transformador para reducir el valor de la corriente, pues medir corrientes más bajas es mucho más sencillo. Este método, además, sí ofrece aislamiento galvánico y tan sólo introduce una inductancia normalmente despreciable en el circuito primario.

Consiste en un núcleo metálico atravesado por la corriente primaria un número reducido de veces, normalmente solo una vez, y atravesado por el circuito secundario muchas más veces. El circuito secundario se cierra con una resistencia de bajo valor en la que se mide la diferencia de potencial. Al ser la resistencia del secundario pequeña, el transformador se comporta prácticamente como si estuviera en

cortocircuito (corriente de magnetización casi nula), por lo que las corrientes de primario y secundario son prácticamente proporcionales. De esta forma se reduce la corriente que circula por el secundario en relación a la que circula por el primario según la relación inversa de espiras. El núcleo metálico puede ser continuo (para reducir los entrehierros y la corriente magnetizante necesaria) o partido, para poder colocar el sensor sin desconectar el circuito primario. Para el margen de corrientes que se van a medir en este proyecto se pueden encontrar este tipo de sensores por unos pocos euros, normalmente menos de 10 €.

Desgraciadamente todas estas tecnologías sirven únicamente para corrientes alternas, pues se basan en la ley de Faraday y aunque en nuestra aplicación usamos corrientes alternas, la frecuencia de la corriente de rotor es muy baja, por lo que se descartó ésta tecnología en favor de otras que sean capaces de medir corriente continua o bajas frecuencias. En el otro extremo del espectro de frecuencias la respuesta de los transformadores de corriente también sufre una fuerte degradación progresiva, ya que las pérdidas en el núcleo ferromagnético crecen muy rápido con el aumento de la frecuencia y eso hace que aumente la corriente necesaria para magnetizarlo, incidiendo negativamente en la precisión y produciendo un retraso entre las corrientes del primario y del secundario que afecta negativamente al sistema de control, pues se introduce un desfase en el sistema de control que afecta a sus prestaciones y puede reducir su margen de estabilidad

Figura 2. Transformador de corriente de núcleo partido de Magnelab

En la Figura 2 se puede ver un transformador de núcleo partido producido por Magnelab (Magnelab, 2019), aguanta hasta 200 amperios, pero su rango de frecuencia es de 30Hz a 1000Hz, por lo que no es apto para nuestra aplicación.

4.3 Bobina de Rogowski

Otra tecnología interesante para medir corriente es la bobina de Rogowski. Aunque también se pueden construir planas la forma clásica es una bobina en forma de hélice que rodea al cable por el cuál circula la corriente que se quiere medir. La bobina tiene un núcleo de aire y gracias a su flexibilidad y a que ambos terminales se encuentran en el mismo extremo de la hélice, se puede colocar de forma sencilla alrededor del

cable primario sin tener que desmontarlo. Sin embargo esta flexibilidad puede ser un problema, pues la sensibilidad del sensor varía si se mueve la bobina, por lo que debe dejarse completamente fija. Para evitar este efecto, las sondas Rogowsky modernas utilizan una vuelta adicional especial en sentido contrario que reduce su incidencia al mínimo proporcionando una buena precisión en cualquier posición alrededor del cable primario.

Gracias a que el núcleo de aire tiene muy baja inductancia, la bobina de Rogowski responde muy bien a corrientes de alta frecuencia y como no puede saturar tiene, a diferencia de los transformadores de corriente, una gran linealidad incluso para corrientes de muy alta frecuencia, hasta el margen de varias decenas de kHz típicamente. También son capaces de medir corrientes de varios miles de amperios sin problemas incluso en sus gamas más bajas.

La bobina opera bajo el mismo principio que los transformadores de corriente, se aprovecha de la inducción y la ley de Faraday para producir una tensión entre sus terminales proporcional a la variación de la corriente medida, por lo que debe ser integrada para poder ser leída. Esto significa que no puede medir corriente continua ni de baja frecuencia, por lo que esta tecnología también se rechaza para su uso en nuestro proyecto. Por otra parte, el importe típico de una única sonda Rogowsky es de varios cientos de euros, entre 20 y 100 veces superior a las otras posibles soluciones consideradas.

Figura 3. Bobina Rogowski producida por Magnelab

En la Figura 3 se puede ver un ejemplo de bobina Rogowski producida por Magnelab (Magnelab, 2019).

4.4 Sensor de Efecto Hall

Los sensores de efecto Hall funcionan gracias a la fuerza de Lorentz, que dice que una carga eléctrica en movimiento sufre una fuerza al atravesar un campo magnético. Como la fuerza de Lorentz depende del campo magnético, a diferencia de la ley de Faraday que depende de la variación del campo, un sensor de efecto Hall puede medir corriente continua o de baja frecuencia sin ningún problema. Hay dos tipos distintos de sensores de efecto hall, de lazo abierto y de lazo cerrado, aunque son muy similares y se podría decir que los de lazo cerrado son la evolución de los de lazo abierto.

El sensor cuenta con un núcleo magnético que rodea al circuito principal para concentrar las líneas de flujo magnético. En una sección del núcleo se coloca una fina placa (el denominado sensor Hall) de forma que el campo la atraviese perpendicularmente, y se hace circular una corriente de control a través de la placa. Como la corriente está atravesando un campo se desviará hacia uno de los lados de la placa por efecto de la fuerza de Lorentz. Esta desviación produce un gradiente de tensión entre los dos extremos de la placa que puede medirse de forma sencilla y es proporcional al campo magnético, que a su vez es proporcional a la corriente del circuito principal que lo ha generado. Midiendo esta tensión una vez correctamente acondicionada se puede saber la corriente que circula por el circuito primario. Los sensores de lazo abierto operan bajo este principio.

Los sensores de lazo cerrado, más caros pero de mejores prestaciones, cuentan con todo lo anterior, pero no miden directamente la tensión creada por la fuerza de Lorentz. En vez de eso, disponen de una tercera corriente que atraviesa múltiples veces el núcleo magnético y se encarga de anular el campo magnético producido por la corriente principal de forma que la tensión medida por la fuerza de Lorentz sea nula. En estos sensores la corriente del primario es directamente proporcional y de sentido contrario a la corriente del tercer circuito que se usa para anular el campo magnético, por lo tanto se usa esta corriente para conocer el valor de la corriente principal. El origen de esa tercera corriente es un amplificador diferencial de gran ganancia a cuya entrada se conecta directamente la salida del sensor Hall: la ganancia del amplificador es tan grande que establece a su salida, muy rápidamente, la tensión necesaria para que por el tercer circuito circule una corriente que compense el efecto magnético de la corriente primaria, es decir, que introduzca la misma fuerza magneto motriz.

Implementar este tercer circuito hace a estos sensores más caros y grandes que su contraparte de lazo abierto, sin embargo, ofrecen una serie de ventajas muy interesantes, la más interesante es que estos sensores pueden trabajar a frecuencias mucho mayores que los de lazo abierto, primero, porque a frecuencias elevadas en los sensores de lazo abierto, las pérdidas en el núcleo, que si está magnetizado, producen un retraso en la inducción y, por tanto, en la salida del sensor Hall, respecto de la corriente primaria, y segundo, porque cuando la frecuencia es muy alta la electrónica de los sensores pierde respuesta y es incapaz de seguir correctamente la variación de la entrada, este fallo vuelve inútil al sensor de lazo abierto a frecuencias muy elevadas. Sin embargo, el sensor de lazo cerrado puede seguir funcionando sin problemas, pues el núcleo metálico y el circuito terciario pasan a comportarse como un transformador de corriente común, como los mencionados en la sección 4.2. Otra ventaja que se obtiene es eliminar la dependencia de la temperatura. El precio se mantiene en algo por encima de la decena de euros para las corrientes que se necesitan medir en este proyecto.

4.5 Conclusión

En esta tabla se recogen las principales características que se han tenido en cuenta para valorar las tecnologías analizadas, los datos se han obtenido de sensores típicos para este tipo de proyectos, pero se pueden encontrar sensores con características muy distintas. El rango de frecuencias de una resistencia "shunt" está limitado por las capacidades e inductancias parásitas y no lo suelen indican los fabricantes, mientras que el rango de frecuencias superior del sensor de efecto Hall varía dependiendo de si es de lazo abierto o de lazo cerrado.

	Rango de Frecuencia	Aislamiento o Galvánico	Precio	Requerimientos especiales
Resistencia "shunt"	Cualquiera*	No	0.5-3 €	Aislamiento auxiliar para proteger el esp32
Transformador de corriente	30Hz-1kHz	Si	5-20 €	Ninguno
Bobina Rogowski	2Hz-100kHz	Si	50-100 €	Integrador
Sensor de efecto Hall	0-25/200kHz	Si	5-15 €	Ninguno

Finalmente se llegó a la conclusión de usar sensores de efecto Hall por ser la opción más sencilla de implementar con un precio contenido y que permite la medida de corrientes de baja frecuencia y disponer de una buena respuesta a altas frecuencias (bajo retraso).

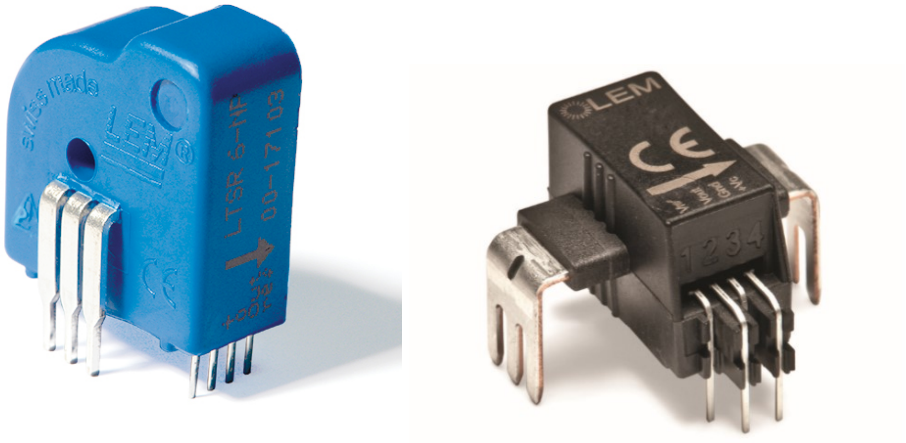

Figura 4. Sensor efecto Hall de lazo abierto (derecha) y lazo cerrado (izquierda)

En la Figura 4 se pueden observar los dos sensores hall que se probaron en este proyecto, ambos fabricados por LEM (LEM, 2019), a la derecha un HLSR-P de lazo abierto y a la izquierda el LTSR 6-NP de lazo cerrado, que fue el finalmente empleado en la construcción del prototipo.

5 MONTAJE DEL PROTOTIPO

En esta sección se explicarán los distintos detalles que se tuvieron en cuenta a la hora de montar el prototipo y que materiales se usaron para hacerlo. Puesto que se trata tan solo de un prototipo el diseño no es aún definitivo y los materiales usados son solo un ejemplo de materiales que cumplen los requisitos necesarios, en la versión final del proyecto se podrían o no cambiar los materiales y definitivamente se hará un diseño más compacto.

5.1 Materiales

En un primer momento se realizó el montaje con sensores de efecto Hall de lazo abierto de la serie HLSR-P fabricados por LEM y se conectaron con cables de cobre comunes, sin embargo, a la hora de realizar las pruebas de medida con un osciloscopio se observó una gran cantidad de ruido que complicaba la medida de corriente del estator y hacía prácticamente imposible la medida de corriente del rotor, debido a que su amplitud es menor y el ruido era del mismo orden de magnitud.

Para solucionar estos problemas se cambiaron los cables de cobre comunes por unos cables de señal apantallados de 0.35mm^2 que aíslan mucho mejor del ruido. Se debe encontrar el equilibrio entre aislamiento y rigidez del cable, pues, debido a la disposición de los componentes, usar cables demasiado rígidos dificulta enormemente el montaje, aunque si se cambia la disposición de componentes en la placa podría ser viable usar cables con mejor apantallamiento. También se cambiaron los sensores por el modelo LTSR 6-NP, igualmente fabricados por LEM, que aunque son más caros, pues son de lazo cerrado, también son más resistentes al ruido y tienen la ventaja adicional de poder cambiar el número de vueltas atraviesan al sensor, permitiendo así aprovechar mejor todo el rango de salida del sensor, poniendo más vueltas en la corriente de rotor y menos en la de estator y consiguiendo así una mejor relación señal/ruido.

Finalmente, se añadió un filtro de paso bajo para eliminar el ruido de más alta frecuencia. El filtro diseñado se muestra en la Figura 5 y se rige por la Ecuación 1, siendo R1 y R2: 2,2kΩ y 4,7kΩ respectivamente, y C: 212 nF. Este filtro tiene una ganancia a frecuencias bajas de 0,68 (que se utiliza para adaptar el rango de salida del LTSR 6-NP que es de 0-5 V, al rango de entrada del ESP32, que es de sólo 0-3,3V), y la frecuencia de corte se encuentra en 3147 ,7rad/s, o lo que es lo mismo, a 500,97Hz. Se eligió esta frecuencia de corte teniendo en cuenta que la frecuencia más alta que pretendemos medir es la del estator, a 50Hz, y la frecuencia de corte debe estar lo suficientemente separada para evitar que se atenúe nuestra señal de forma involuntaria y que se introduzca un desfase en el filtro que afecte al control de corrientes. También se tuvo en cuenta que la frecuencia de muestreo del procesador es de 2kHz, por lo que siguiendo el criterio de Nyquist se debe eliminar como mínimo todas las frecuencias por encima de 1kHz para evitar el posible “aliasing”, se optó por la mitad de esta frecuencia por seguridad y para poder usar el prototipo más adelante a mayores frecuencias.

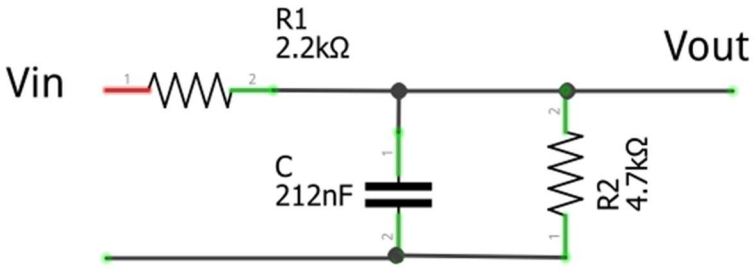

Figura 5. Filtro paso bajo. Imagen creada con Fritzing

$$Vout = Vin * \frac{R2}{R1 + R2} * \frac{1}{1 + \frac{\omega j}{\frac{R1 + R2}{R1 * R2 * C}}}$$

Ecuación 1

En cuanto al circuito primario, se usaron cables con terminales de seguridad en los extremos que conectan a la alimentación y al propio motor, y los extremos que se deben conectar a los sensores se soldaron con estaño.

Todos los componentes del prototipo se colocan sobre una perfboard. Las perfboard son placas perforadas en forma de matriz con secciones de material conductor, normalmente cobre o estaño, alrededor de los agujeros, estas secciones pueden o no estar unidas entre ellas. Para fijar los componentes y realizar las conexiones se debe usar un soldador para unir las secciones de material conductor. Este tipo de placas requieren de más tiempo de montaje que las protoboard, cuyas conexiones se realizan por fricción, pero son más baratas y el montaje es mucho más rígido y estable.

El esp 32, los cables de señal y otras conexiones del prototipo que no forman parte directa de este TFG como las relativas al codificador rotativo se conectan a la placa a través de conectores para Arduino de 2.54mm, estos conectores se sueldan a la placa por un extremo y por el otro se conectan por fricción.

Finalmente el prototipo se fija por medio de tornillos a una carcasa de plástico que lo cubre para no dejar al descubierto las conexiones y evitar posibles accidentes. La carcasa debe tener agujeros para permitir la entrada de alimentación del esp32 y los cables de alimentación del motor.

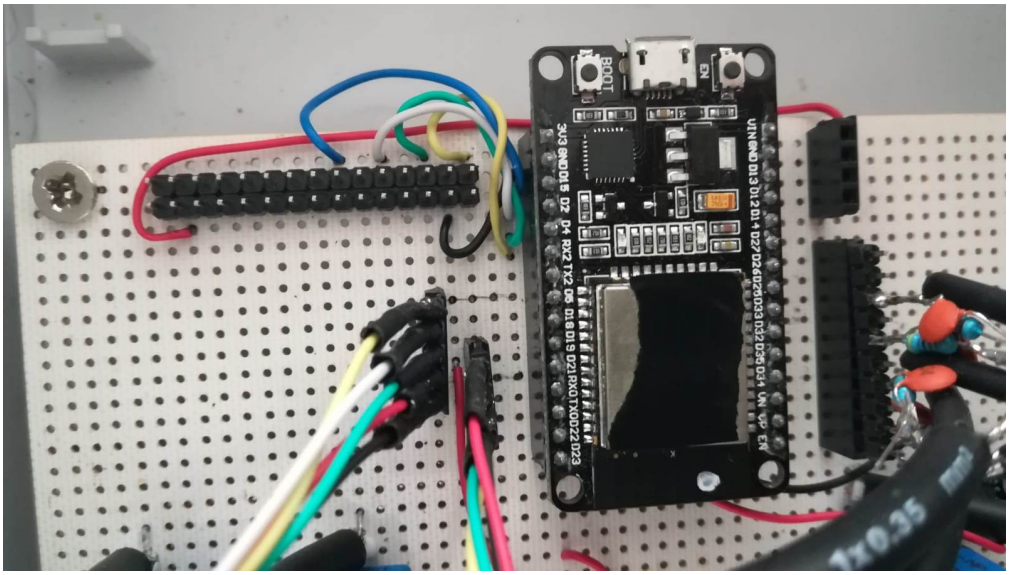

Figura 6. Detalle de las conexiones del prototipo

En la Figura 6 se puede apreciar el esp32 conectado a la placa perfboard a través d elos conectores de 2.54 mm, a su derecha se pueden observar los filtros de paso bajo soldados a los cables de señal y a la izquierda se encuentran el resto de conexiones, se usan para comunicarse con un inversor, con una pantalla LCD y con el codificador rotativo.

5.2 Conversión Analógica-digital y calibración de los sensores

Los sensores dan como salida una señal analógica entre 0 y 5V, en un primer momento se planteó usar un chip mcp3208 para la conversión analógica-digital, pero finalmente se usaron directamente las patillas adc del esp32, pues son suficientemente rápidas en su lectura, 10-12 microsegundos, y a diferencia del chip no requieren una inversión adicional ni montaje extra. Estas patillas tienen 4096 bits de resolución y son capaces de medir de 0 a 3,3 V, por lo que se debe tener cuidado para evitar la saturación, una vez se colocó el número de vueltas apropiado en los sensores Hall para aprovechar al máximo el rango de entrada sin que la señal llegue a saturar, se realizó el ensayo de calibración de corriente.

El ensayo de calibración de corriente consiste en hacer pasar una corriente por los cuatro sensores y una carga cualquiera en serie, medir los datos obtenidos durante un periodo determinado con el esp32, anotar el valor eficaz de la corriente que nos indica la fuente de alimentación y finalmente procesar esos datos con la ayuda de MATLAB para obtener los coeficientes de conversión que nos permiten realizar la conversión de bits medidos a Amperios. El ensayo se realizó 2 veces a 2,221A y a 1,464A y los coeficientes utilizados finalmente son la media de ambos ensayos. En la Figura 7 se muestran los datos obtenidos al medir la corriente de 2,221A de valor eficaz. Los sensores 1 y 2 corresponden a los sensores del estator mientras que los del rotor son los 3 y 4. Se puede observar que los sensores del rotor son más sensibles a la corriente, debido al mayor número de vueltas del primario, y que el sensor 4 recibe la corriente en sentido contrario al resto de sensores, esto último solo ocurre en este ensayo y es debido a la disposición especial de los sensores 3 y 4, pues tienen un extremo unido.

Para calcular los coeficientes basta con restar el valor medio a las corrientes para que oscilen alrededor del cero, calcular el valor eficaz y dividir la corriente mostrada por la fuente de alimentación (2,221A y 1,464A) entre el valor eficaz obtenido.

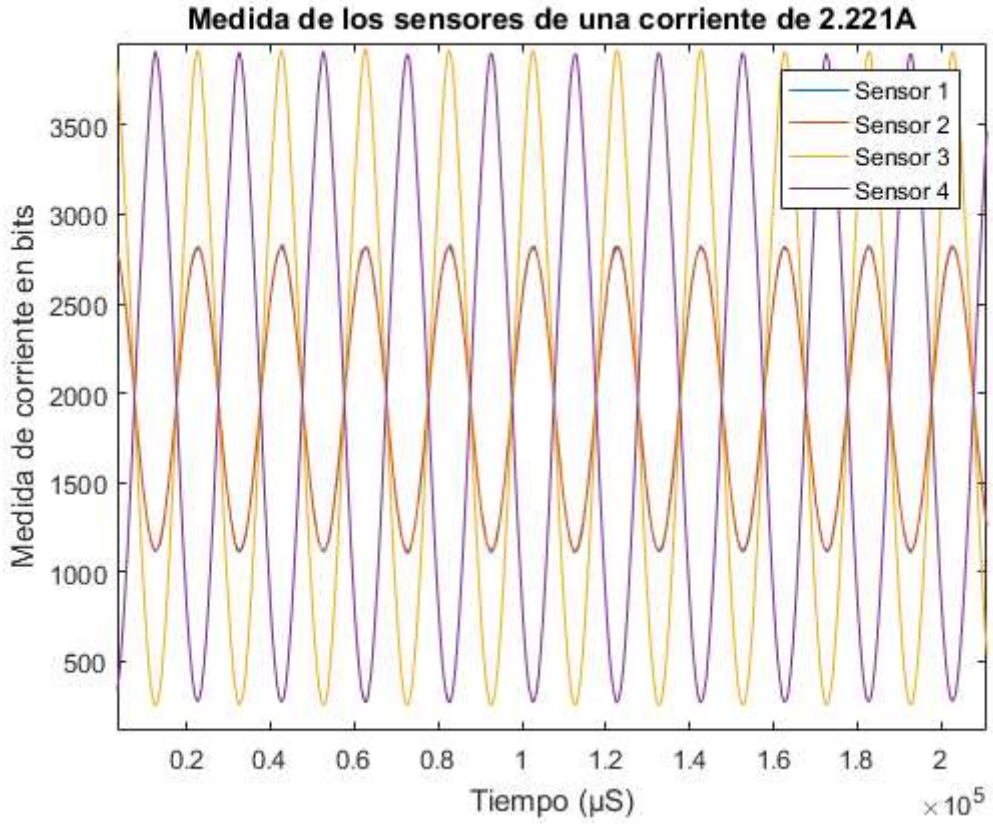

Figura 7. Medidas obtenidas por los sensores en el ensayo de calibración. Imagen creada con MatLAB

	Sensor 1 (rotor)	Sensor 2 (rotor)	Sensor 3 (rotor)	Sensor 4 (rotor)
Factor de conversión bit-Amperio	0.003726	0.003725	0.001803	0.001817

Los coeficientes de conversión obtenidos son los mostrados en la tabla.

5.3 Conexiones del circuito

El estator del motor se conecta en triángulo, una de sus fases está conectada directamente a la alimentación, mientras que las otras dos tienen conectadas un sensor en serie. El rotor por el contrario se conecta en estrella internamente y debe quedar cortocircuitado externamente, por lo que los sensores tienen una de sus entradas conectadas entre ellas y a la tercera fase. En la Figura 8 se muestra la conexión de los primarios de los sensores Hall con el rotor y el estator del motor. El sensor 1 mide la fase R del estator, el 2 mide la fase S, y el 3 y el 4 hacen lo propio con las fases R y S del rotor.

Además de las entradas por las que circula la corriente principal los sensores disponen de cuatro patillas más. Una de ellas es la tensión de referencia de 2,5V y la dejaremos

sin conectar, pues no es necesario cambiar la referencia en nuestro proyecto. Otra sirve para la alimentación, estas patillas se conectarán entre ellas y a la patilla de 5 V del esp32, esta patilla es especial pues permite alimentar componentes electrónicos directamente desde la propia alimentación del esp32. Otra de las patillas es la referencia a masa, éstas como era de esperar también se conectan entre ellas y a una de las patillas de masa del esp32. Por último tenemos la patilla de salida, Esta patilla es la que produce una tensión directamente proporcional a la corriente medida por el circuito principal, estas patillas, lógicamente, no se unen entre ellas, sino que se conectan a cuatro patillas distintas de conversión analógica-digital del esp32 a través de los cables apantallados y los filtros paso bajo mencionados previamente.

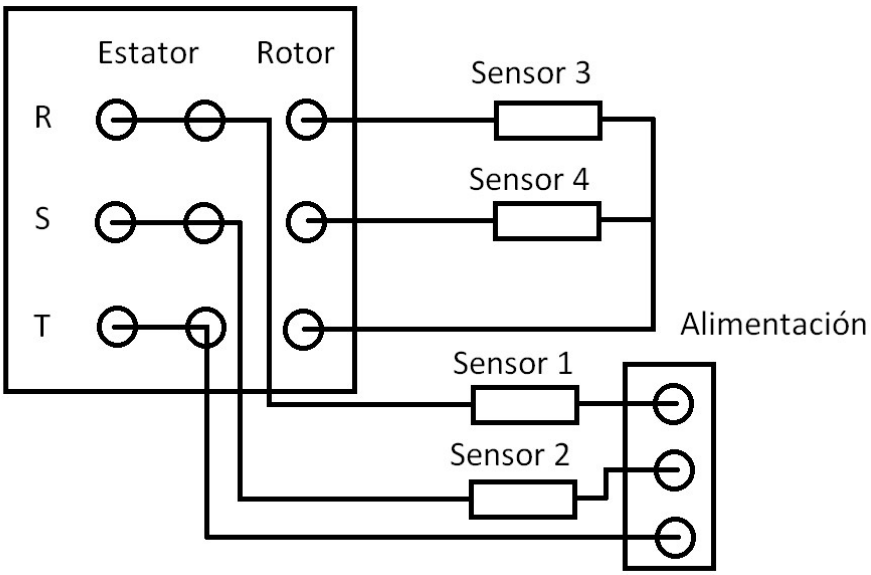

Figura 8. Conexión de los sensores de efecto Hall con respecto al motor.

En la Figura 9 se puede ver el prototipo completo, los dos sensores de la derecha son los del rotor y los dos de la izquierda son los del estator. En la imagen se puede apreciar como efectivamente el número de vueltas del primario de los sensores del rotor es mayor al número de vueltas de los sensores del estator Las conexiones de las patillas que no llevan la señal analógica de salida están conectadas directamente con caminos de soldadura por debajo de la placa, pues apenas les afecta el ruido, las conexiones analógicas también tienen un pequeño tramo de soldadura, pero es lo más pequeño posible para tratar de reducir el ruido.

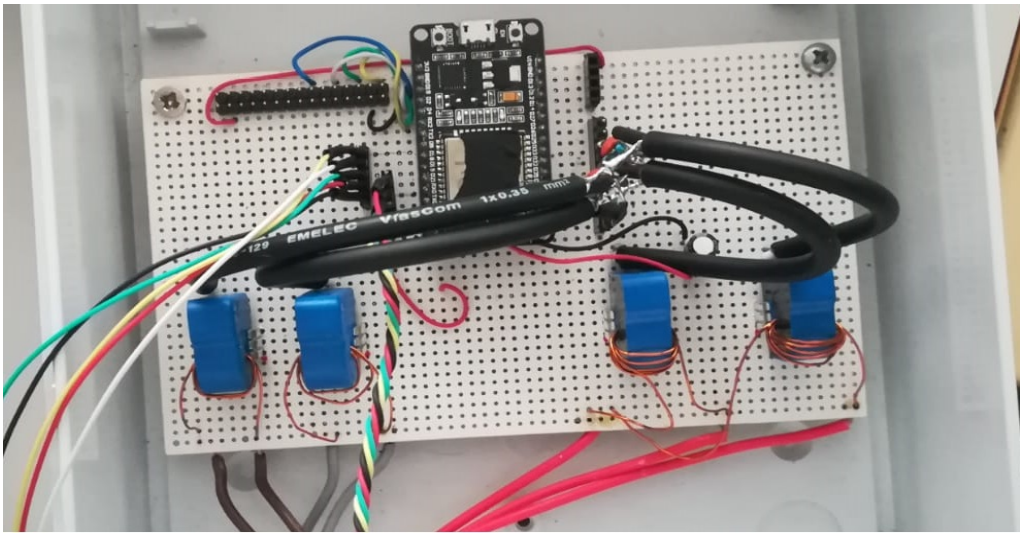

Figura 9. Prototipo

5.4 Codificador rotativo

Aunque la medida de posición y velocidad no forma parte de este TFG, es completamente necesario conocer la posición del eje para poder realizar los cálculos, por lo que aquí se indican algunos detalles del codificador usado y como instalarlo.

El codificador se trata de un EMS22A50-M25-LD6 producido por el fabricante Bourns (Bourns, 2019). El eje del codificador se debe colocar solidario al eje del motor a ensayar, mientras que el cuerpo debe fijarse para evitar su giro, esto se puede lograr con una canaleta rígida de PVC y un par de tornillos. Para la conexión al esp32 se usa el protocolo de comunicación SPI, por lo que se usarán 2 cables para la alimentación, uno para la velocidad de reloj, otro para indicar que se quiere leer del codificador y finalmente uno más para recibir los datos codificados en binario. En este caso las conexiones serán con cables básicos de cobre, pues la comunicación digital es mucho más resistente al ruido.

En la Figura 10 se puede apreciar el codificador rotativo colocado en el eje del rotor y con los cables ya soldados. También se puede ver la canaleta de PVC que sujeta el codificador para evitar que gire.

Figura 10. Codificador rotativo

6 CÁLCULO DE LOS FASORES

El sistema de control que se va a implementar se llama de orientación por campo o FOC (Field Oriented Control) y es, junto con el control directo de par, un control de altas prestaciones.

Cómo ya se ha explicado el objetivo de este TFG es calcular los fasores de corriente del rotor y del estator y obtener su posición en los ejes de referencia d,q (de magnetización y transversal del rotor). Para obtener esta información será necesario conocer la posición del eje del rotor y las corrientes de las tres fases tanto del rotor como del estator. Esta información nos permitirá crear un sistema de control para máquinas de accionamientos eléctricos mucho más avanzado que los usados normalmente, aunque la implementación del propio control no forma parte de este TFG.

6.1 Justificación teórica

La justificación completa de este sistema de control es altamente compleja, por lo que esta sección se limitará a ofrecer los principales conceptos que la hace interesante, para información más completa sobre los sistemas de control basados en corrientes vectoriales consultar en (Martínez Román, 2014).

El principal objetivo de este sistema de control es realizar los cambios de par a la máxima velocidad posible sin desperdiciar recursos, es decir, que responda rápidamente a cambios de consigna utilizando de la forma más efectiva posible los recursos disponibles para cambiar las corrientes del motor.

La corriente de un motor se invierte principalmente en generar el flujo magnético en el circuito magnético principal que, al interactuar con las corrientes de estator y rotor produce el propio par del motor, que permite al motor girar. Cuando se produce un cambio de régimen en el motor parte de los recursos puestos en juego se pueden invertir en variar el flujo magnético del motor, esto es una pérdida de recursos, por lo que el sistema de control se basará en mantener el flujo magnético constante para poder variar el par más rápidamente. Es fácil evitar que el flujo del estator no varíe, pues lo estamos alimentando nosotros directamente, pero conseguirlo para el rotor es más complicado, por lo que deberemos analizar el comportamiento de los fasores en las coordenadas del rotor.

En la Figura 11 se pueden ver los fasores de corriente en las coordenadas d,q así como el fasor espacial de enlaces de flujo totales del rotor, $\Psi_{rot,total}$. El flujo total del rotor define el eje d y se encuentra exactamente 90º por delante del fasor de corriente del rotor. La corriente del rotor se encuentra en el eje –q. Los ejes a,b son los ejes fijos del rotor, que a diferencia de los d,q no son solidarios al fasor de corriente del rotor sino que se mueven con el propio rotor.

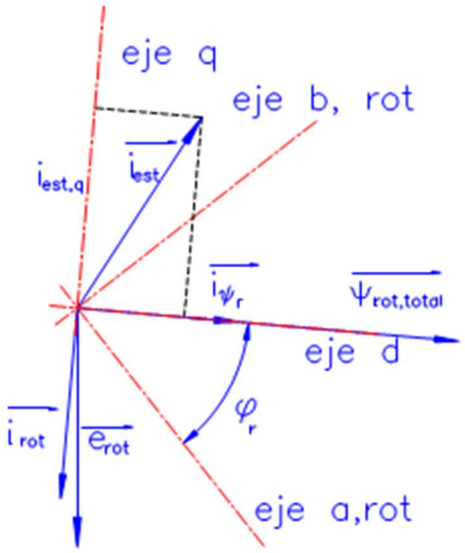

Figura 11. Fasores de corriente, flujo magnético y fem en los ejes d,q y los ejes a,b del rotor

La Ecuación 2 define el flujo magnético total del rotor, el primer miembro de la suma corresponde al flujo de corona mientras que el segundo miembro pertenece a las líneas de flujo de dispersión del propio rotor.

$$\bar{\Psi}"_{rot,total} = L_{\mu,maq} * (\bar{\imath}_{est} + \bar{\imath}'_{rot}) + L'_{\sigma,rot} * \bar{\imath}'_{rot} = L_{\mu,maq} * (\bar{\imath}_{est} + (1 + k_{\sigma,rot}) * \bar{\imath}'_{rot})$$

Ecuación 2

En la Ecuación 3 se simplifica la fórmula anterior definiendo dos nuevas variables, especialmente la llamada corriente magnetizante total del rotor, $i_{\psi r}$, esto nos permitirá simplificar los cálculos más adelante.

$$\bar{\imath}_{\psi r} = \bar{\imath}_{est} + (1 + k_{\sigma,rot}) * \bar{\imath}'_{rot} \qquad k_{\sigma,rot} = \frac{L'_{d,rot}}{L_{\mu,maq}} \qquad \bar{\Psi}"_{rot,total} = L_{\mu,maq} * \bar{\imath}_{\psi r}$$

Ecuación 3

La fuerza electromotriz inducida en el rotor se define como en la Ecuación 4. El último miembro de la ecuación es igual al anterior pero usando las definiciones de la Ecuación 3.

$$\bar{e}"_{rot,total} = -L_{\mu,maq} * \frac{d}{dt}(\bar{\imath}^{rot}_{\psi r}) = R'_{rot} * \bar{\imath}'_{rot} = R'_{rot} \frac{\bar{\imath}_{\psi r} - \bar{\imath}_{est}}{1 + k_{\sigma,rot}}$$

Ecuación 4

La derivada debe hacerse en las propias coordenadas del rotor, en vez de en los ejes d,q. Llamamos φ_r al ángulo entre los ejes del rotor y los ejes d,q, por lo que el el fasor de corriente en coordenadas de rotor se puede expresar como en Ecuación 5 y la derivada sería como se expresa en Ecuación 6, en la última expresión se retira el ángulo para representar a las componentes de los ejes d y q.

$$\bar{\imath}^{rot}_{\psi r} = i_{\psi r} * e^{j\varphi r}$$

Ecuación 5

$$\frac{d}{dt}(\bar{i}_{\psi_r}^{rot}) = \frac{di_{\psi_r}}{dt}e^{j\varphi r} + j * i_{\psi_r} * e^{j\varphi r}\frac{d\varphi r}{dt} = (\frac{di_{\psi_r}}{dt}, i_{\psi_r}\frac{d\varphi r}{dt})_{d,q}$$

Ecuación 6

Si combinamos las componentes de los ejes d,q obtenidas en Ecuación 6 con la Ecuación 4 teniendo en cuenta que la corriente $i_{\psi r}$ definida en Ecuación 3 está alineada con el eje d y por tanto su componente en el eje q es 0, obtenemos las Ecuaciones 7

$$\begin{cases} d: -L_{\mu,maq}\dfrac{di_{\psi_r}}{dt} = \dfrac{R'_{rot}}{1+k_{\sigma,rot}}(i_{\psi_r} - i_{est,d}) \\ q: -L_{\mu,maq} * i_{\psi_r}\dfrac{d\varphi r}{dt} = -\dfrac{R'_{rot}}{1+k_{\sigma,rot}}i_{est,q} \end{cases}$$

Ecuaciones 7

Finalmente, recolocando los términos podemos expresar las Ecuaciones 7 como en las Ecuaciones 8.

$$d: i_{est,d} = \frac{L_{\mu,maq}(1+k_{\sigma,rot})}{R'_{rot}}\frac{di_{\psi_r}}{dt} + i_{\psi_r} \qquad q: \frac{d\varphi r}{dt} = \frac{R'_{rot}}{(1+k_{\sigma,rot})L_{\mu,maq}}\frac{i_{est,q}}{i_{\psi_r}}$$

Ecuaciones 8

En la ecuación del eje d se puede apreciar que la componente de corriente del estator del eje d afecta al campo y además lo hace con retraso debido a la derivada y al término de inductancia entre resistencia, que es la constante de tiempo de magnetización del rotor. Por otro lado en el eje q se aprecia que la componente de corriente del estator del eje q afecta inmediatamente a la derivada del ángulo φ_r. Este ángulo es el que separa los ejes d,q de los ejes a,b, los ejes d,q son solidarios al fasor de enlaces de flujo totales del rotor, mientras que los ejes a,b son solidarios al eje del rotor. Es decir, que la derivada de φ_r es igual a la velocidad del fasor de corriente del rotor, y como se explica en (Martínez Román, 2014), esta velocidad es igual al deslizamiento, y el deslizamiento, a velocidades de giro cercanas a la nóminal, es proporcional al par otorgado por el motor.

En resumen, variar la componente q del fasor de corriente del estator afecta directa e inmediatamente al par del motor y no afecta al flujo, mientras que variar la componente d afecta lentamente al flujo y a $i_{\psi r}$, por lo que también afectará al par de forma indirecta a través de $i_{\psi r}$ pero lo hará de forma mucho más lenta. Es decir, se ha logrado separar el fasor de corriente en dos componentes, de las cuales una controla el par y otra controla el flujo.

Por lo tanto, el objetivo del control será variar únicamente la componente q del estator y mantener la componente d constante. No se entrará en detalles de cómo se logra este control porque no es parte de este TFG, pero es completamente imposible establecer el control si no se conocen las componentes d,q del fasor, pues es precisamente lo que se desea controlar.

El mayor punto en contra de este tipo de control es que se necesita conocer con exactitud los fasores de corriente, por lo que solo se puede realizar de forma precisa

en motores con los terminales del rotor disponibles para poder medir las corrientes del rotor (en motores en los que no se dispone de esa medida, como cualquier motor de jaula de ardilla, se puede realizar lo que se conoce como un control indirecto de par, imponiendo las condiciones para que se cumpla la condición de que i_{sd} se mantenga constante, o un control basado en la estimación o medida del eje de magnetización total del rotor) y además requiere de bastante potencia de cálculo y de conocer la posición y velocidad del rotor. Afortunadamente, el esp32 es capaz de realizar estos cálculos sin problemas

6.2 Implementación en el código

Pasar de la teoría a la práctica es algo más complicado de lo que parece, pues aparecen problemas que se podían ignorar en la teoría, pero que hay que afrontar en la práctica. Por ejemplo, en la práctica se debe asegurar que todos los ángulos tengan la misma referencia, que los fasores giren en el mismo sentido y que las conexiones no se cambien de sitio, pues modificaría la posición de las fases. Otro problema importante que no se contempla en la teoría es el ranurado del motor, que provoca ciertos problemas en los ensayos a baja potencia: en el apartado 8.2 se detalla el problema más a fondo.

Para programar el esp32 se usará el software de desarrollo gratuito y de código abierto Arduino IDE. Este software ha sido creado por Arduino (Arduino, 2019) y permite programar en C/C++ cualquier microcontrolador para el que se haya desarrollado una librería específica. En nuestro caso debemos programar el esp32, para ello se debe instalar un paquete especial que permite al Arduino IDE comunicarse con el esp32, el paquete lo ha desarrollado Espressif Systems y se puede descargar desde su página de GitHub (Espressif Systems, 2019). Una vez instalado el paquete programar el esp32 es tan sencillo como conectarlo a un puerto del ordenador a través de un cable USB-miniUSB, seleccionar el puerto al que se ha conectado desde el Arduino IDE, presionar el botón de upload y una vez se haya compilado el programa presionar el botón de boot presente en el esp32 para que se inicie la subida del programa.

Durante el desarrollo se probaron todos los cálculos en MatLab antes de introducirlos al esp32. MatLab es un software de cálculo matemático diseñado especialmente para ingenieros creado por MathWorks (MathWorks, 2019). Este software permite realizar innumerables funciones enfocadas a muchos campos distintos, sin embargo, aquí se usará simplemente por su capacidad para recoger datos, realizar cálculos y representarlos cómodamente en gráficas. El esp32 es incapaz de realizar gráficos directamente y cualquier cambio en el código exige recompilar y resubir el código al microcontrolador, de forma que trabaja con MatLab es mucho más sencillo y rápido. De esta forma se desarrollarán todos los cálculos primero en MatLab con el uso de scripts y una vez se esté seguro de que funcionan correctamente se trasladarán a el esp32. MatLab, a diferencia del resto de programas usados en el TFG, es un software de pago, afortunadamente disponen de licencias para estudiantes, este TFG se ha podido realizar gracias a una de estas licencias para estudiantes.

Una vez se comprobó que los cálculos funcionaban en MatLab se trasladaron a el esp32 mediante Arduino IDE. Hay algunas diferencias entre los cálculos realizados en MatLab y en el Arduino IDE, principalmente debido a que el Arduino IDE no soporta

números imaginarios directamente, por lo que se decidió operar con las componentes real e imaginaria por separado, aunque también se podría haber usado una librería que permita los números imaginarios. Otra diferencia importante es que mientras la esp32 trabaja leyendo los datos de los sensores en tiempo real, MatLab trabaja con paquetes de datos leídos previamente por el esp32 y guardados en forma de texto en archivos .txt.

6.2.1 Cálculo de fasores de corriente en su propio sistema de referencia

Necesitamos conocer la corriente de las tres fases del rotor y las tres del estator para poder calcular sus fasores de corriente. Sin embargo solo se necesitan cuatro sensores, dos para las corrientes del rotor y dos para las corrientes del estator, pues la tercera componente del fasor se puede calcular mediante la Ecuación 9 (la suma de las tres componentes de corriente trifásica alimentando una carga sin conexión de neutro es cero), pues el motor utiliza una conexión a tres hilos sin neutro.

Ecuación 9

$$it(t) = -ir(t) - is(t)$$

El esp32 lee la señal de los sensores como un valor entre 0 y 4095, para convertirlo en una señal que se pueda usar debemos restarle el valor medio para que el valor oscile alrededor del 0. Calculándolo con ayuda de Matlab y el ensayo explicado en el apartado 5.2 se obtiene que los valores medios para cada sensor son 1960, 1968, 1980 y 1986 para los sensores 1, 2, 3 y 4 respectivamente, aunque estos valores no son estrictos y varían entre distintos ensayos, en general basta con usar un valor entre 1950 y 2000. Una vez puestos a cero, se deben cambiar las unidades a Amperios, para ello primero se deben obtener primero los coeficientes usando el ensayo mencionado previamente, los coeficientes obtenidos son 0.0037263 y 0.0037252 para los sensores 1 y 2, y 0.0018037 y 0.0018178 para los sensores 3 y 4, la gran diferencia entre ellos se debe a que los sensores 3 y 4 se usan para medir la corriente del rotor, que tiene un módulo mucho menor a la del estator, por lo que estos sensores son más sensibles a la corriente (debido a que tienen más vueltas en su primario).

A continuación, se muestra cómo se realizan todos los pasos anteriores en el esp32, el código está escrito con Arduino IDE. analogRead() es la función que se encarga de medir la tensión en alguna de las patillas ADC del esp32 y devolverlo como un valor digitalizado entre 0 y 4095 bits. pinx son cuatro constantes que indican que patilla quieres medir, en nuestro caso son las patillas 36, 39, 34 y 35 respectivamente. ixy son variables tipo double que guardan el valor de la corriente, la primera letra indica si es del estator o del rotor, s y r, y la segunda letra indica la fase, también s y r. offsetx son cuatro constantes que guardan el valor medio de la corriente mientras que coeficentex hace lo propio con los coeficientes que realizan el cambio de escala de bits a Amperios.

En MatLab se procede de forma muy similar, salvo que las corrientes se obtienen de vectores cargados previamente de un archivo de texto en vez de usar el analogRead().

```
isr=analogRead(pin1);
iss=analogRead(pin2);
irr=analogRead(pin3);
```

```
irs=analogRead(pin4);

isr-=offset1;
iss-=offset2;
irr-=offset3;
irs-=offset4;

isr*=coeficiente1;
iss*=coeficiente2;
irr*=coeficiente3;
irs*=coeficiente4;
```

Una vez hecho esto se debe hacer uso de la Ecuación 9 para obtener la tercera fase del rotor y el estator y calcular los dos fasores, aquí es donde aparecen la diferencia más importante entre el Arduino IDE y MatLab, pues con MatLab se usará la expresión de Euler, mientras que en Arduino IDE se usarán el seno y el coseno de 60º para calcular la parte imaginaria y la real por separado y la función atan2() para obtener el ángulo del fasor, en radianes, calculando la arcotangente. Se debe usar la función atan2() en lugar de la función atan() por que la función atan() no distingue en qué cuadrante se encuentra el ángulo, ya que atan() solo pide el valor de la tangente mientras que atan2() pide la componente real e imaginaria, y la propia función se encarga de realizar la división. La elección de qué fase es la R, cuál es la S y cuál es la T es en principio arbitraria, pero una vez elegidas se deben usar siempre las mismas, y conviene asegurarse que los fasores de estator y rotor giren en el mismo sentido.

A continuación se muestra el código del Arduino IDE, dónde el coseno de 120 y de 240 se ha sustituido por -0,5 y el seno de 120 y 240 por ±√3/2, es decir, sus valores. Las variables que guardan las componentes reales e imaginarias y los ángulos son todas double.

```
ist=-isr-iss;
irt=-irr-irs;

isreal=isr-0.5*(iss+ist);
isimag=sqrt(3)/2*(iss-ist);
irreal=irr-0.5*(irs+irt);
irimag=sqrt(3)/2*(irs-irt);

angs=atan2(isimag,isreal);
angr=atan2(irimag,irreal);
```

En MatLab las seis últimas líneas se sustituyen por éstas dos, dónde exp() es una función que la exponencial del argumento que le entreges, e is e ir contienen toda la información (parte real e imaginaria) de los fasores de corriente del estator y rotor:

```
is=isr+iss*exp(1i*2*pi/3)+ist*exp(1i*4*pi/3);
ir=irr+irs*exp(1i*2*pi/3)+irt*exp(1i*4*pi/3);
```

Con éste código ya se pueden crear algunas gráficas interesantes que nos permiten comprobar si el código implementado hasta ahora funciona correctamente. Por ejemplo, cuando el motor funciona consumiendo unos 500W las corrientes de fase de estator y rotor son cómo las mostradas en la Figura 12 y la Figura 13. Como era de

esperar la amplitud y la frecuencia de la corriente de rotor son mucho menores que la del estator. Aunque no se pueden dibujar directamente los fasores de corriente porque están girando, si se puede mostrar su ángulo. En la Figura 14 se muestra el ángulo de los fasores, como era de esperar, la frecuencia del fasor del rotor es muy pequeña, de unos 2 Hz en este caso, mientras que la del estator es de 50 Hz.

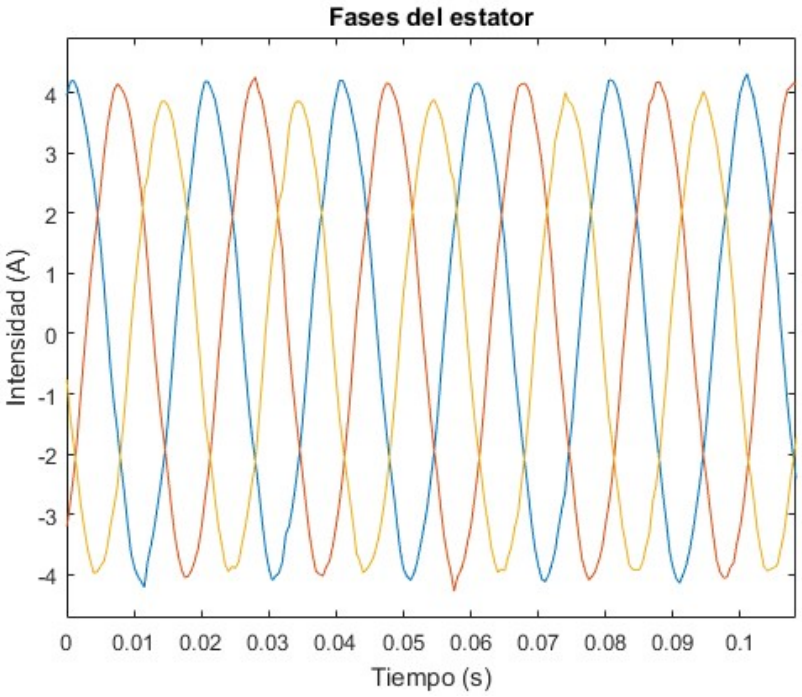

Figura 12. Fases R, S y T del estator

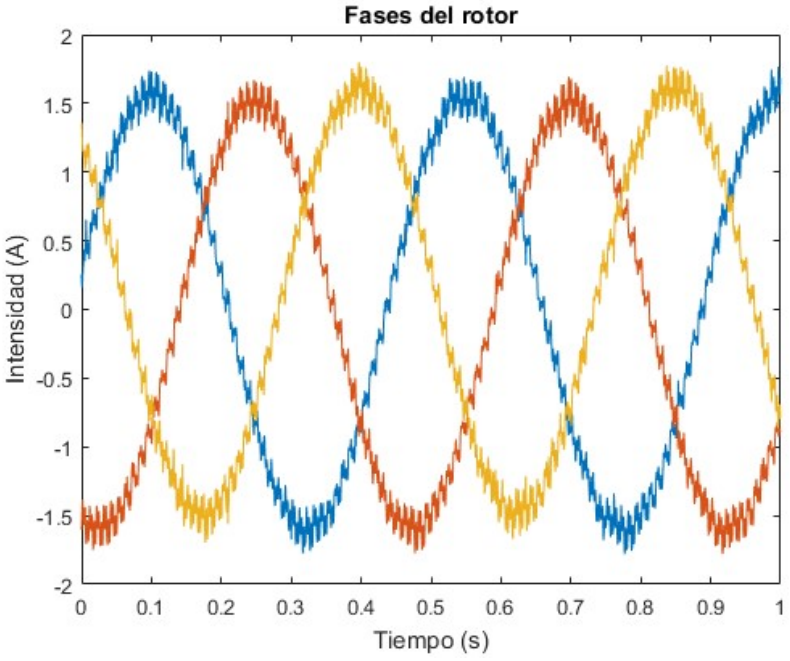

Figura 13. Fases R, S y T del rotor

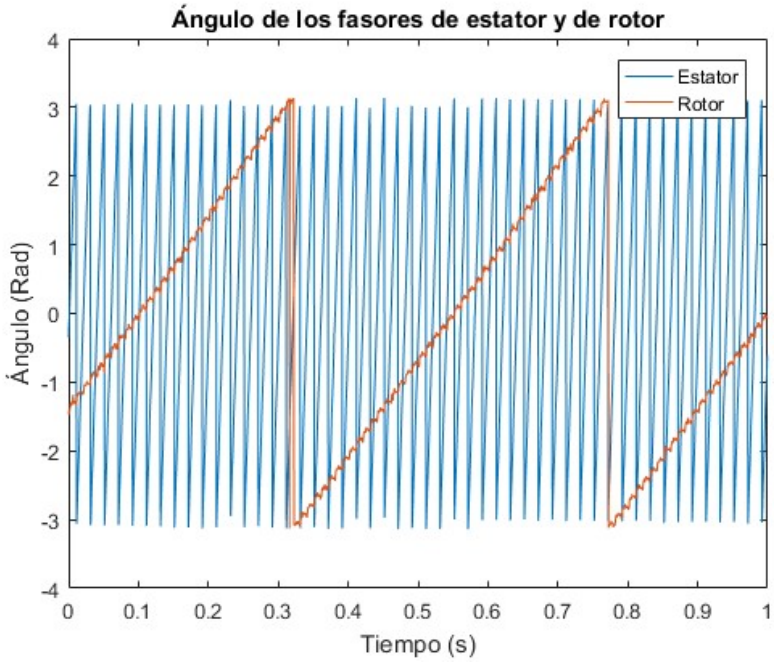

Figura 14. Ángulo de los fasores de rotor y estator

6.2.2 Cambio de sistema de referencia y ensayo de calibración

Ahora que ya tenemos los fasores en sus respectivos sistemas de referencia debemos ponerlos en el mismo sistema de referencia para poder compararlos. Para ello se le debe sumar al ángulo del fasor de corriente del rotor, el ángulo del rotor físico, así ambos fasores estarán en el sistema de referencia del estator. Para ello se debe tener en cuenta que el motor es de dos pares de polos, por lo que el ángulo físico del rotor cuenta como el doble de ángulos eléctricos para el fasor del rotor. Para saber la posición física del rotor se usará un codificador rotatorio de posición absoluta, los detalles de este sistema no forman parte de este TFG, por lo que no se entrará en detalles, basta con saber que usando la función medir() el esp32 se comunica con el codificador y nos devuelve el ángulo de posición del rotor en un rango digital de 0 a 1023. En la Figura 15 se muestra la relación entre las velocidades de giro de los fasores de corriente de estator y rotor (ω_{est} y ω_{rot}) y la velocidad de giro del propio eje del rotor (Ω), que debe multiplicarse por el número de pares de polos (p) para transformarlo a coordenadas eléctricas.

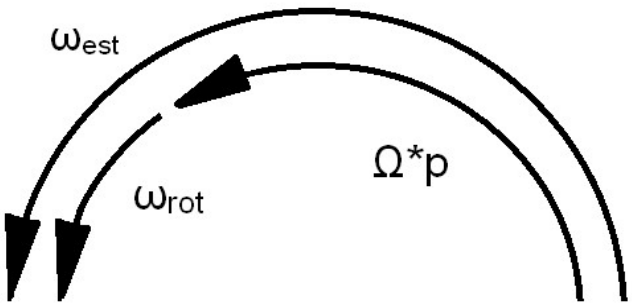

Figura 15. Velocidad de giro de los fasores de corriente y del rotor en coordenadas eléctricas

Sin embargo, no es tan sencillo como medir el ángulo y sumarlo directamente, pues necesitamos saber el ángulo que separa los ejes del estator de los ejes del rotor, de

forma que cuando las fases R del rotor y el estator estén alineadas el ángulo calculado sea cero. Como el codificador se colocó de forma aleatoria esto no se cumple, por lo tanto, es necesario un ensayo de calibración, cuyo objetivo será determinar el valor a descontar a la posición mecánica determinada por el sensor de posición del rotor de forma que esa diferencia (que indicará la posición angular del sistema de referencia ligado al rotor o móvil respecto al sistema de referencia ligado al estator o fijo) sea justo 0 cuando los ejes magnéticos de las fases R del estator y R del rotor estén alineados.

El ensayo de calibración del codificador rotatorio consiste en hacer pasar corriente alterna por una sola fase del estator y medir la tensión entre ella y los principios de las fases R del estator y R del rotor, de esta forma se puede identificar cuando se encuentran alineadas las fases del rotor y el estator (ya que la fem inducida en la fase R del rotor será máxima y la diferencia de potencial entre los principios de la fase R del estator y la fase R del rotor será mínima) y así poder descontar ese ángulo cuando se sume el ángulo físico del rotor al eléctrico del fasor. En la Figura 16 se muestra cómo se conectaron los cables para este ensayo.

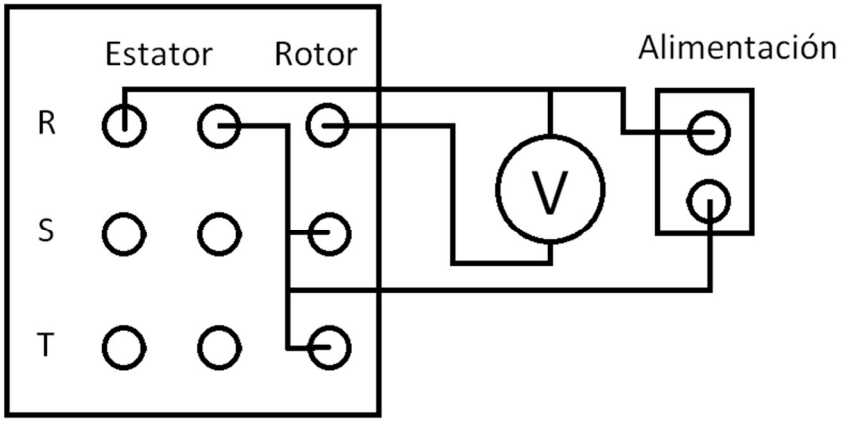

Figura 16. Conexión para el ensayo de calibración del codificador

Específicamente, en esta situación la tensión del estator sería la mostrada en la Ecuación 10, mientras que la tensión del rotor viene definida por la Ecuación 11, donde phi rotor representa el ángulo entre las fases del rotor y el estator y rt la relación de transformación entre el rotor y el estator.

Ecuación 10

$$uest(t) = \sqrt{2} * u * \cos(w * t)$$

Ecuación 11

$$urot(t) = \sqrt{2} * rt * u * \cos(w * t) * \cos(\varphi rot)$$

Como estamos midiendo la diferencia entre ambas tensiones con el voltímetro, las fases estarán alineadas cuando la tensión sea lo más baja posible, pues el coseno de φ_{rot} será 1. Se mide tensión para distintas posiciones del rotor y se anota en qué ángulo, según el codificador rotativo, se han medido. Se realizaron dos medidas, una entre las fases R y otra entre la fase R del rotor y la fase S del estator, haciendo esto

se puede comprobar que los resultados son coherentes, ya que tiene que haber una diferencia de 120º eléctricos, la separación angular entre dos fases consecutivas. Una vez se introducen los datos en MatLab se observa que la tensión tiene un tercer armónico al triple de frecuencia, asociado a la distribución de las bobinas de estator y rotor, así que se retira con la ayuda de la transformada de Fourier y obtenemos la onda mostrada en la Figura 17 y la Figura 18. Al ver los resultados se puede confirmar que la máquina es de dos pares de polos, pues con una sola vuelta del rotor las fases se alinean dos veces, de forma que podemos ignorar toda la parte por encima de los 512 bits de ángulo, pues es una copia de lo obtenido antes de los 512 bits.

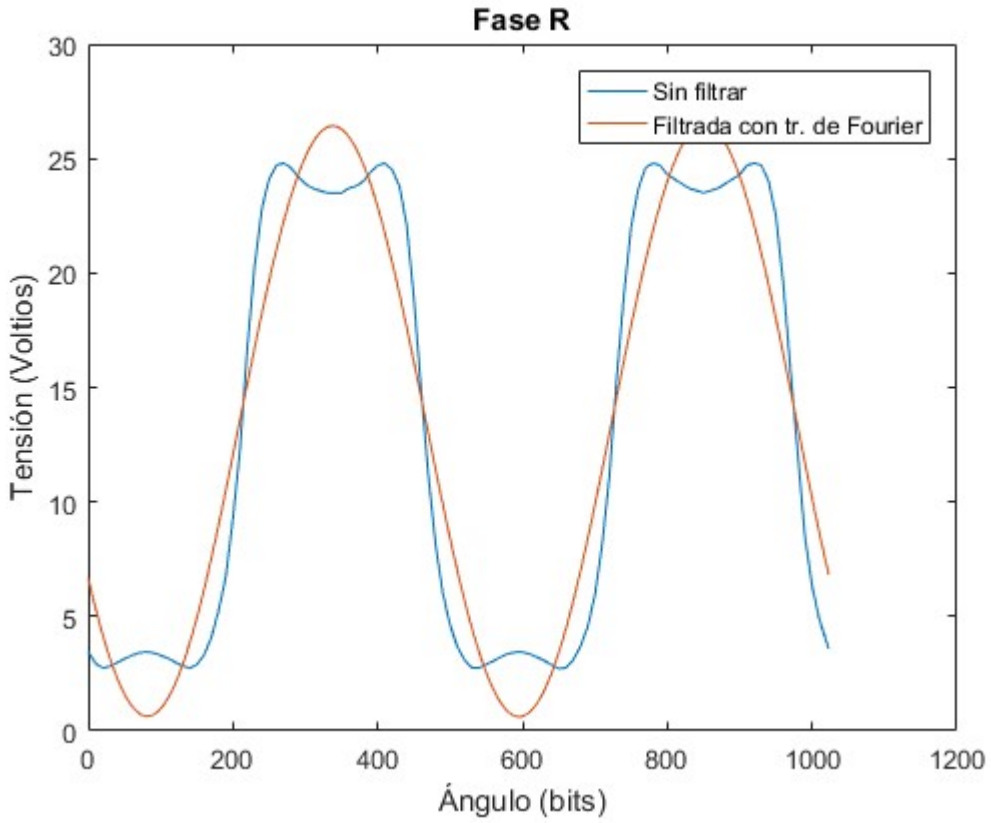

Figura 17. Calibración del codificador fase R

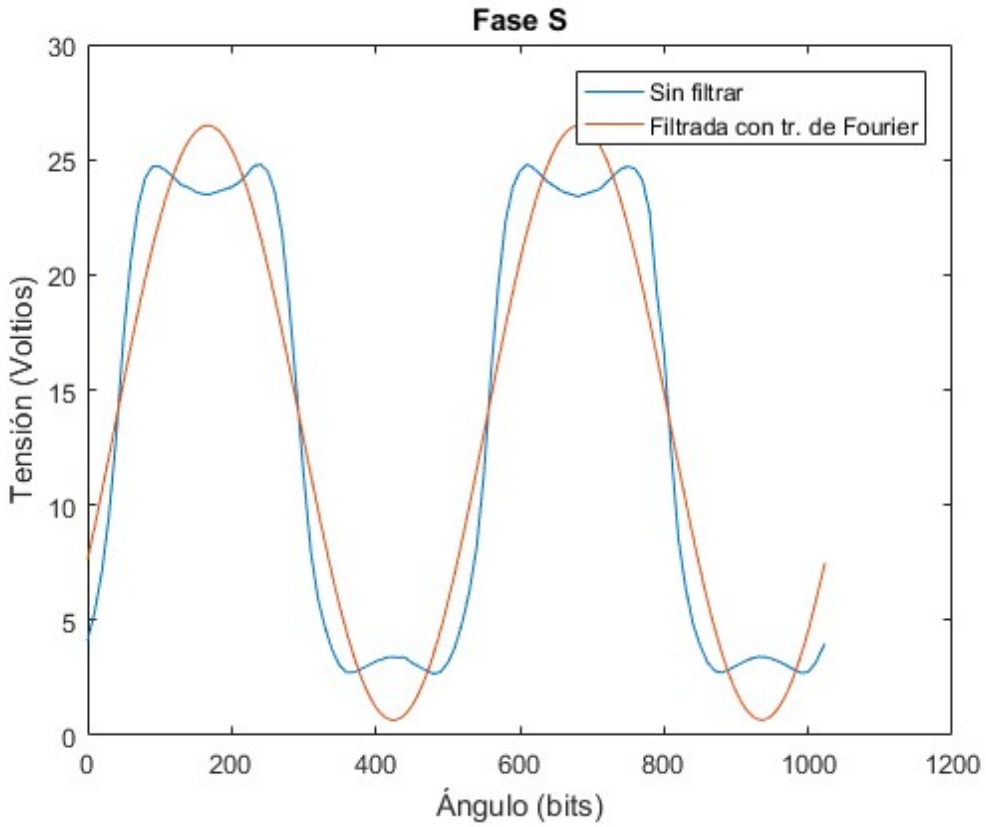

Figura 18. Calibración del codificador fase S

En cuanto a los resultados, la fase R del estator se alinea con la fase R del rotor en el ángulo 82 bits, mientras que la fase S lo hace en el ángulo 424. Estos ángulos equivalen a 57.66º y – 61.875º, se puede observar fácilmente que tienen una diferencia de casi 120º exactos, justo como era de esperar. Con estos datos se puede deducir que la fase T se alineará a los 177.66º, es decir, a unos 253 bits, también se puede observar que las fases están al revés de cómo debería, el orden en ángulo es R T S en vez de R S T, esto ocurre simplemente porque el codificador lee los ángulos al revés de cómo decidimos establecer las fases del motor. Esto significa que cuando los fasores giren en un sentido, el codificador dirá que el rotor gira en sentido contrario, por lo tanto, al añadir el ángulo físico del rotor al fasor se debe hacer restando (para cambiar el signo), en vez de sumando.

Por último, antes de hacer el cambio de sistema de referencia se debe tener en cuenta que el ensayo de calibración se realizó directamente con las fases del motor, pero en el resto de ensayos el estator del motor se conecta en estrella, y los cálculos se hacen como si la corriente de línea fuese la de la propia fase R, esto significa que hay un desfase de 30º entre la fase R real y la que usamos en el resto de ensayos. Para solucionar esto basta con sumar, o restar, esos 30º extra al fasor del estator, la diferencia de fase que existe entre corrientes de línea y de fase en conexión triángulo.

En cuanto a la implementación en código en Arduino IDE, primero obtendremos la posición del rotor con la función medir(). Después modificaremos el ángulo del fasor del rotor, guardado previamente en angr, restándole el ángulo de posición, no sin antes haberle restado los 82 bits que separan las fases R. Evidentemente, también se

debe cambiar la escala de bits a radianes, como se ha tomado una vuelta como 512 bits no hace falta multiplicar el ángulo de posición por el número de pares de polos para pasar de ángulos mecánicos a eléctricos. El número pi debe escribirse tal cual. Finalmente se le suman los 30º al ángulo del estator, que se había guardado en angs.

```
posicion=medir();
angr-=2.0*3.141592653589793*(posición-82.0)/512.0;
angs+=30.0*3.141592653589793/180.0;
```

La implementación en MatLab es ligeramente distinta, pues se sigen usando los números imaginarios. En este caso llamamos a los fasores isab e irab para simbolizar que se encuentran en el mismo sistema de referencia, el de los ejes a,b fijos del estator. Se debe usar el operando '.*' porque se está trabajando con vectores.

```
irab=ir.*exp(-1*i*2*pi/512*(pos-82));
isab=is.*exp(1*i*30*pi/180);
```

Si se quiere confirmar que el cambio de coordenadas se ha realizado como es debido se puede representar en una gráfica el ángulo de los fasores del rotor y el estator. En la Figura 19 se muestra el ángulo de ambos fasores una vez se ha realizado el cambio de coordenadas.

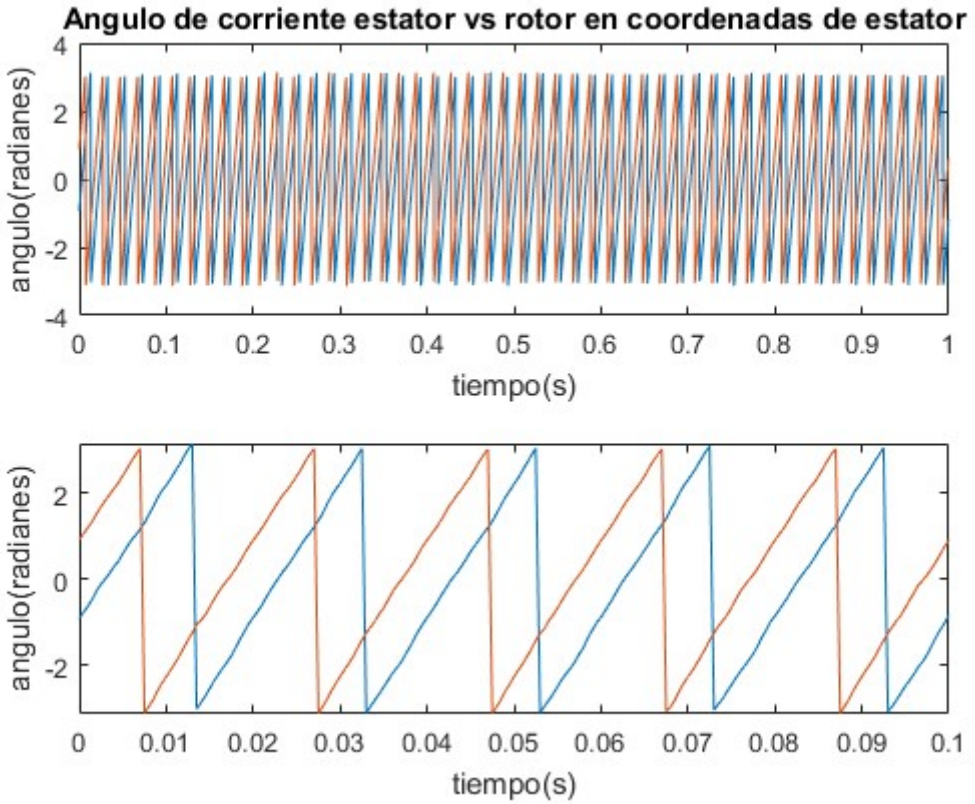

Figura 19. Ángulo de los fasores de estator t rotor en coordenadas de estator

Como era de esperar ahora los fasores, una vez ambos están expresados en el sistema de referencia del estator, giran a la misma velocidad y el ángulo entre ellos se mantiene constante, siempre y cuando el motor se mantenga en régimen permanente.

Por lo tanto, se puede confirmar que la frecuencia del rotor por el número de pares de polos de la máquina (o frecuencia asociada al movimiento) más la frecuencia del fasor del rotor es igual a la frecuencia del fasor del estator.

6.2.3 Cambio de referencia a los ejes d,q

Ahora que los fasores se encuentran en los mismos ejes y giran a la misma frecuencia solo necesitamos restar al ángulo del fasor del estator el ángulo del fasor del rotor, así el fasor deja de girar y su ángulo nos muestra el desfase entre los fasores de estator y rotor. Finalmente también debemos colocar los fasores en la posición correcta, ya que sabemos que el fasor de corriente del rotor está orientado según el eje –q, que se encuentra a -90º, por lo que colocaremos el fasor de corriente del rotor en esa posición y aplicaremos el mismo cambio al fasor del estator, es decir, restarle 90º.

En Arduino IDE es tán sencillo como restar los ángulos directamente, restarle 90º extras al estator, calcular el módulo del fasor estatórico con las funciones sqrt() y pow() y finalmente calcular las componentes real e imaginaria del estator, que como sabemos coinciden con el eje d y el eje q. Estos dos datos junto con el tiempo son los que se envían a la interfaz gráfica y son los que permiten efectuar el control vectorial por corrientes del motor.

```
angs-=angr;
angs-=90.0*3.141592653589793/180.0;
ismod=sqrt(pow(isreal,2)+pow(isimag,2));
isreal=ismod*cos(angs);
isimag=ismod*sin(angs);
```

En MatLab como siempre se calcula de forma ligeramente distinta, para obtener el módulo se usa la función abs(), y además, a diferencia del programa de Arduino IDE, aquí se fija el fasor del rotor en menos 90º, donde se encuentra el eje –q, pues nos puede ser útil para representarlo en gráficas.

```
irabfijo=abs(irab).*exp(-90*i*pi/180);
angulo=angle(irab);
isab=isab.*exp(-1*i*angulo);
isab=isab.*exp(-1*i*90*pi/180);
```

Ahora mismo contamos con datos suficientes para representar los fasores de muchas formas, podemos representar su ángulo, o sus componentes para poder observar de un vistazo el valor de cada componente, d, q y –q, pero la forma más ilustrativa de mostrar el par de fasores es en el plano complejo. En la Figura 20 se muestran el fasor del rotor y del estator en los ejes d, q de referencia. Más adelante se puede ver la representación de los componentes de los fasores en la Figura 27

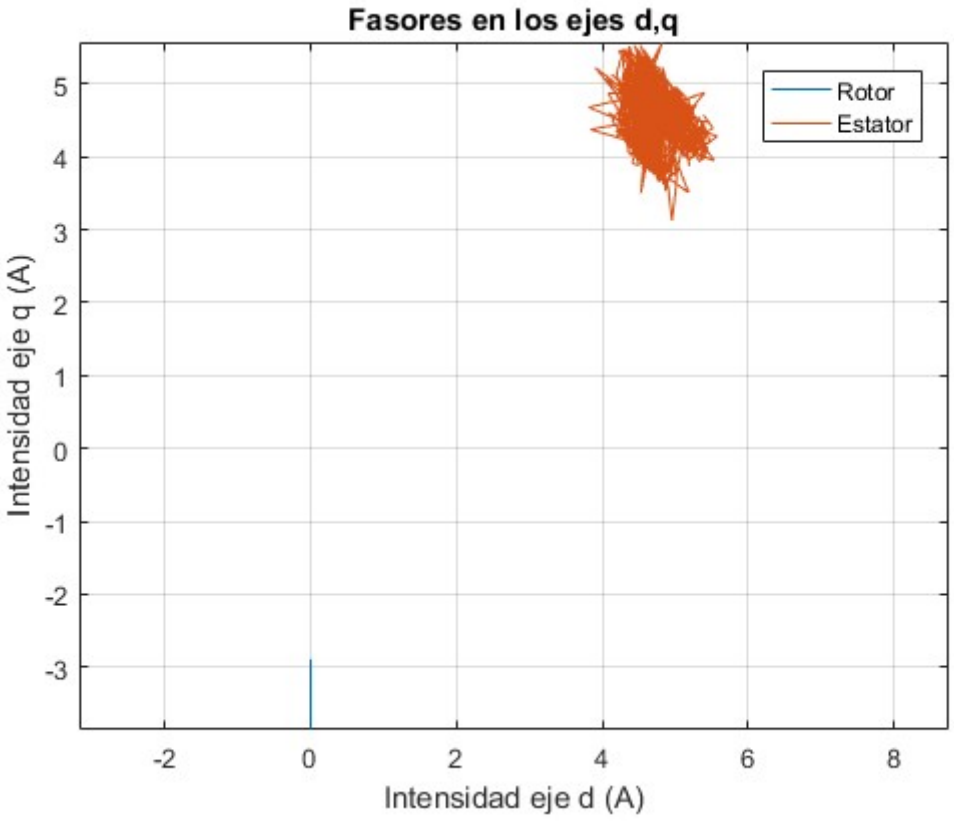

Figura 20. Fasores de corriente en los ejes d q complejos

El fasor del rotor se mantiene siempre a -90º, pues es este fasor el que define todo el sistema de referencia (al estar alineado por definición con el eje –q), mientras que el fasor del estator podemos observar que oscila entre varias posiciones, estas oscilaciones son debidas principalmente a la variación en orientación del fasor de corriente del rotor, pues como se puede observar en la Figura 13 las fases del rotor oscilan bastante. Esto es debido en parte al ranurado del rotor y se explica en más profundidad en la sección 8.2.

En resumen, no es factible, debido a factores constructivos de la máquina utilizada, conseguir que el fasor del estator se mantenga completamente fijo. Afortunadamente estas oscilaciones son asumibles y si se diera el caso de necesitar una salida más suave bastaría con aplicar un filtro a las componentes del fasor del estator.

6.2.4 Envío de datos a MatLab

Aunque no es una función necesaria para el proyecto final, es conveniente explicar de qué forma se pasaron los paquetes de datos medidos con el esp32 a MatLab.

Para comunicar el esp32 con el ordenador se usó el “Serial monitor” incluido en Arduino IDE, una sencilla ventana que permite enviar y recibir cadenas de caracteres a través del USB para comunicarse con cualquier microcontrolador que lo permita. De esta forma basta con conectar el esp32 con el ordenador a través de un cable USB-mini USB y usar la función Serial.print() para enviar los datos desde el esp32 al ordenador. Puesto que mandar los datos por el Serial es un proceso lento no se pueden enviar conforme se van leyendo, pues se perdería mucha información, por lo que primero se almacena toda la información en vectores (en la memoria RAM del

ESP32) y después se envía toda por el puerto USB. A continuación la función que se encarga de recopilar los datos y enviarlos por el USB, el vector de tiempo es de tipo int, mientras que los de tensión y posición son de tipo uint16_t, es decir, de solo 16 bits, usar variables de pocos bits nos permite crear vectores más largos y por tanto tomar más datos, puesto que la memoria de un microcontrolador no es muy grande y es fácil que el compilador te informe de un error de falta de memoria si creas vectores exageradamente grandes. En este caso los vectores usados son de 2000 elementos, que a la velocidad de toma de datos de 500 microsegundos equivale a obtener datos durante un segundo.

```
{
   i=0;
   tinit=micros();
   digitalWrite(2,HIGH);
   while (i<tam){

     currenttime=micros();

      if (currenttime-tiempoultimamedida>=500){
         tc[i]=micros()-tinit;
         V1[i]=analogRead(pin1);
         V2[i]=analogRead(pin2);
         V3[i]=analogRead(pin3);
         V4[i]=analogRead(pin4);
         pos[i]=medir();
         tiempoultimamedida=currenttime;
         i++;
     }
   }
   digitalWrite(2,LOW);
   for (int i=0; i<tam; i++){
     Serial.print(tc[i]);
     Serial.print(" ");
     Serial.print(pos[i]);
     Serial.print(" 1 ");
     Serial.print(V1[i]);
     Serial.print(" 2 ");
     Serial.print(V2[i]);
     Serial.print(" 3 ");
     Serial.print(V3[i]);
     Serial.print(" 4 ");
     Serial.println(V4[i]);
   }
}
```

Como se puede observar el código se compone de un primer bucle que se encarga de rellenar los vectores de tiempo posición y tensión. La tensión corresponde con lo salida de los sensores de corriente, por lo que el contenido de los vectores corresponde con la corriente que circula por las fases del estator y el rotor. Las medidas solo se toman cuando han pasado 500 microsegundos desde la última medida. Una vez se rellenan los vectores completamente con datos se entra en un segundo bucle que envía estos datos con Serial.print(), la función Serial.println() es igual que Serial.print() salvo por que coloca un salto de línea después de enviar su mensaje.

Con la función digitalWrite() se controla la tensión de salida de distintas patillas de la esp32, pero en este caso se usa la patilla 2 por que está unida a un LED que lleva

incorporado la esp32, de forma que el brillo del LED nos informa de cuando está leyendo datos. La variable i es la variable auxiliar que se encarga de recorrer los vectores, mientras que tam es una constante que contiene el tamaño de los vectores. La función micros() nos devuelve la cantidad de microsegundos que pasaron desde que se inició el esp32, por lo tanto para interpretar el tiempo se deben usar restas.

Los datos quedan con el siguiente aspecto, los números 1,2,3 y 4 indican a que sensor corresponde la medida mientras que la primera columna es el tiempo, este formato puede ser leído fácilmente por MatLab, pues al ser todo números se puede interpretar como una matriz.

```
500 1 755 2 1735 3 3303 4 517
1001 1 608 2 2000 3 3346 4 453
1501 1 496 2 2256 3 3367 4 428
```

Por último es importante destacar que una vez se terminó de implementar el código en el esp32 se hicieron una serie de medidas en las que se enviaban a MatLab tanto los datos tomados directamente de los sensores como los fasores calculados por el esp32, esto nos permite comparar los cálculos de MatLab con los del esp32 para comprobar que dan el mismo resultado. Efectivamente dan el mismo resultado, pero es muy importante comprobar esto para comprobar que los cálculos se han trasladado correctamente desde MatLab al esp32.

7 INTERFAZ DE USUARIO

Los requisitos básicos de la interfaz de usuario son que sea capaz de guardar un histórico de los datos de los fasores y que sea capaz de enviar órdenes al controlador. Se decidió desde el principio que la interfaz de usuario sería una aplicación de Android, pues es el sistema operativo más extendido para dispositivos móviles, es sencillo desarrollar para él por su gran comunidad y además así se pueden aprovechar las numerosas tablet Android usadas para otras prácticas de las que dispone el laboratorio del departamento de ingeniería eléctrica, para el cuál se está preparando este banco de ensayos.

La aplicación que hace de interfaz se ha desarrollado con el software gratuito Android Studio y es bastante sencilla de usar, consta de tan solo dos pantallas. En la primera de ellas se muestra una lista con los dispositivos bluetooth con los cuales puede conectarse el terminal. La segunda pantalla permite mandar al microcontrolador la orden de activar/desactivar la lectura de datos por medio de un botón, también se encarga de recibir los datos calculados por el microcontrolador y los puede mostrar tanto en forma de texto como en una gráfica, finalmente dispone de otro botón que ordena al microcontrolador el envío de datos por USB explicado en la sección 6.2.4, esta última función solo sirve para el desarrollo del tfg y por tanto será retirada en la versión final del proyecto.

7.1 Protocolo de comunicación

Elegir un protocolo de comunicación apropiado entre el dispositivo android y el microcontrolador es una cuestión importante, pues necesitamos encontrar una solución que sea compatible tanto con el dispositivo android como con el microcontrolador. Durante el desarrollo se barajaron dos alternativas.

7.1.1 Comunicación por WiFi

Al principio del desarrollo se trabajó con el microcontrolador esp8266 que dispone de wifi integrado, así que esta forma de comunicación es la más lógica. El protocolo que se decidió usar para la comunicación es el de MQTT.

El MQTT es un protocolo de comunicación por wifi diseñado para enviar mensajes simples a través de un servidor o "broker. Tiene la ventaja de ser sencillo de implementar en cualquier dispositivo con comunicación WiFi", por lo que es muy usado en el internet de las cosas, pero tiene la desventaja de requerir de un servidor externo y de conexión a la red a través de un "router" lo que lo convierte en una opción menos favorable.

7.1.2 Comunicación por Bluetooth

Tiene dos desventajas importantes respecto al MQTT: Que es una comunicación a corta distancia, y que la mayoría de microcontroladores no disponen de Bluetooth integrado. Mientras que su única ventaja es que establece una conexión directa sin necesidad de un "router" y un "broker".

Sin embargo, finalmente se decidió usar este tipo de comunicación, pues la larga distancia es totalmente irrelevante en este proyecto y el poder comunicarse directamente es una característica mucho más interesante al reducir costes.

Para solventar el problema de la ausencia de comunicación Bluetooth en la mayoría de microcontroladores se decidió optar por una solución muy simple, usar el esp32, un microcontrolador que dispone de Bluetooth integrado además de otras funcionalidades interesantes para el proyecto.

7.2 Implementación del protocolo de comunicación Bluetooth

El programa del esp32 se escribirá gracias a Arduino IDE en C/C++, mientras que la aplicación de Android se escribirá con Android Studio en Java, ambos lenguajes de programación son orientados a objetos. Los objetos pertenecen a una clase y la clase define el comportamiento del objeto a través de métodos, y sus características a partir de atributos. Los atributos son básicamente variables que definen las características de un objeto, la clase a la que pertenece el objeto define la cantidad y tipo de atributos que tiene el objeto, mientras que el valor de los atributos puede ser diferente para cada objeto de la misma clase. Por otro lado los métodos son básicamente funciones definidas en una clase, todos los objetos de una misma clase tienen los mismos métodos. Algunos métodos pueden ejecutarse directamente sin ningún objeto como una función normal, pero normalmente los métodos se ejecutan sobre un objeto concreto, la razón de esto es que normalmente los métodos se usan para alterar los atributos del objeto o devolver/calcular algún valor con ellos.

En cuanto al esp32, puesto que lleva bluetooth integrado basta con incluir la librería BluetoothSerial.h en el código y usar sus objetos y métodos para administrar la conexión. Adicionalmente, como el esp32 dispone de un procesador de dos núcleos, se ha programado para que uno de ellos realice los cálculos y medidas de los fasores mientras que el otro se encarga de enviar y recibir los datos vía bluetooth, para esto también será necesario usar objetos.

Implementar la conexión en Android es algo más complicado, principalmente porque es la aplicación la que se encarga de realizar el enlace de conexión y tanto el sistema operativo como la librería Bluetooth son bastante más complejos. Afortunadamente, Android es una plataforma con una enorme base de usuarios, por lo que es sencillo encontrar información sobre cualquier problema que pueda surgir. En concreto, la página oficial para desarrolladores de Android explica perfectamente todas las funcionalidades Bluetooth de Android y cuál es la mejor forma de implementarlas en tu código. Adicionalmente, para desarrollar la estructura básica del programa se ha recurrido a un tutorial, pues la mejor forma de aprender algo es viendo un ejemplo de cómo se hace.

La aplicación desarrollada en Android es incapaz de buscar y conectarse a dispositivos Bluetooth desconocidos por sí sola, en vez de eso, permite conectarse a cualquier dispositivo con el cual el dispositivo Android se haya vinculado previamente. Por lo tanto, la vinculación debe hacerse al menos una vez desde la propia configuración Bluetooth del dispositivo Android. Añadir la función de escaneo de dispositivos cercanos es una opción interesante que se podría añadir en el futuro, sin embargo, la aplicación se va a usar para un laboratorio de prácticas donde las Tablets se conectarán siempre con los mismos microcontroladores, por lo que es una función poco importante.

7.2.1 Código en Arduino IDE

En esta sección se explicarán de forma concreta cómo se comunica el esp32 a través del Bluetooth con la interfaz. El punto más importante, como ya se ha explicado anteriormente, es que se aprovecharán los dos núcleos del microcontrolador para que las comunicaciones no afecten a los cálculos, por lo tanto se deben usar objetos capaces de administrar tareas y asignarlas a un núcleo específico. Para que se comuniquen las tareas entre ellas se usarán “queues”, que también se deben controlar con objetos, y una estructura que contiene la información que debe ser enviada.

Las tareas son, en esencia, hilos de ejecución de código que se ejecutan en paralelo en vez de en forma secuencial. Si se tiene un solo núcleo las tareas se retrasarán entre ellas, pero si se tiene más de un núcleo y se asigna cada tarea a un núcleo distinto, los hilos de ejecución se ejecutarán en paralelo sin afectarse entre ellos, a menos que intenten acceder al mismo recurso simultáneamente.

Las “queues”, también llamadas colas, pero las continuaremos llamando “queues”, se pueden definir como un buffer donde se almacena información para que esté disponible para otras tareas. Es decir, una tarea puede dejar datos en la “queue” para que otra tarea los recoja cuando pueda, los primeros datos en dejarse son los primeros en recogerse, pues sigue la estructura “first in, first out” o FIFO. En nuestro

caso la principal razón por la que usamos “queues” es porque son una forma sencilla de avisar a otra tarea de que hay datos nuevos preparados para ser enviados.

Así se declaran los objetos y variables necesarias para las comunicaciones, las variables normales que se usan en los cálculos no están incluidas, pues ya se explica de qué tipo son en la sección 6.2.

```
#include "BluetoothSerial.h"

BluetoothSerial ESP_BT;
typedef struct{
  long tiempo;
  double dato1, dato2;
  int pos;
}datos;
datos datoscorr;
xQueueHandle enviar;
TaskHandle_t Bluetask;
```

Primero se incluye la librería BluetoothSerial.h y se crea el objeto ESP_BT que se encarga de manejar las comunicaciones. Luego se define la estructura que contiene los datos de tiempo y posición y ejes d y q de corriente y se crea una variable que sea de este tipo de estructura. Por último se crea un objeto enviar de tipo “queue” y un objeto Bluetask que permite designar una tarea.

El código del microcontrolador contiene dos partes principales, el “setup”, donde se incluye el código que solo se ejecuta una vez, y el “loop”, donde se incluye el código que se repite continuamente, ambas partes se ejecutan por defecto en el núcleo 1. Por lo tanto la tarea de comunicaciones se debe fijar en el núcleo 0. Así es como se inicializan las variables en el “setup”.

```
void setup() {
  enviar=xQueueCreate(5,sizeof(datos));
  pinMode(pin1,INPUT);
  pinMode(pin2,INPUT);
  pinMode(pin3,INPUT);
  pinMode(pin4,INPUT);
  pinMode(2,OUTPUT);
  digitalWrite(2,LOW);

xTaskCreatePinnedToCore(Bluecode,"Bluetask",10000,NULL,1,&Bluetask,0);
  Serial.begin(115200);
}
```

En la primera línea se inicializa la “queue” declarada anteriormente y se le asigna un tamaño capaz de almacenar hasta 5 variables de tipo estructura datos, que se definió anteriormente. Con pinMode() se define si los pines son de entrada o salida, los pines dedicados a leer los sensores son de salida, mientras que el pin 2, usado por el LED que lleva conectado, es de salida. Con digitalWrite() nos aseguramos que el LED está apagado. Con xTaskCreatePinnedToCore() asignamos una tarea al núcleo 0, el código de la tarea se encuentra en la función Bluecode() que se mostrará más adelante. Por último Serial.begin() nos permite usar la comunicación del USB a una velocidad de 115200 bits por segundo.

El código que se ejecutará en el núcleo 0 dentro de la función Bluecode() será el encargado de manejar las comunicaciones. Usa la “queue” anterior para saber cuándo debe enviar datos, mientras que para recibir datos se usa una variable normal llamada incoming.

```
void Bluecode( void * pvParameters){
   ESP_BT.begin("ESP32_Corrientes");
   while(1){
     if (ESP_BT.available()){
       incoming = ESP_BT.read();
     }
     else if(xQueueReceive(enviar,&datoscorr,100)==pdTRUE){
            ESP_BT.print(datoscorr.tiempo);
            ESP_BT.print('#');
            ESP_BT.print(datoscorr.dato1);
            ESP_BT.print('#');
            ESP_BT.print(datoscorr.dato2);
            ESP_BT.print('#');
            ESP_BT.print(datoscorr.pos);
            ESP_BT.print('#');
     }
   }
}
```

Con ESP_BT.begin() se inicia la comunicación Bluetooth, a partir de ahora cualquier dispositivo Bluetooth puede encontrar a la esp32 con el nombre ESP32_Corrientes. Después se entra en un bucle infinito que consta de dos partes.

En la primera parte se comprueba si se ha recibido algún dato por Bluetooth con la función ESP_BT.available(), si se han recibido datos, se lee con ESP_BT.read() y se guarda en la variable incoming. Los datos que recibe la esp32 se limitan a un número en formato ASCII que le indica qué debe hacer en ese momento, bien leer los datos de corriente, almacenarlos en vectores y enviarlos por el USB como se explica en 6.2.4 o bien realizar los cálculos de los fasores y enviarlos por Bluetooth al dispositivo Android. Debido al uso que se la da a la variable incoming no es necesario usar una “queue” para coordinar las tareas de comunicaciones y cálculo.

La otra parte del bucle se encarga de enviar los mensajes, primero espera durante 100 milisegundos a recibir datos en la “queue” enviar, si los recibe, procede a enviarlos por Bluetooth con la función ESP_BT.print, entre dato y dato se envía un símbolo ‘#’ para informarle al dispositivo Android de que se trata de un dato distinto, pues la información se envía en forma de texto y puede ser complicado de interpretar si no se usa algún código que nos ayude.

El código que se ejecuta en el núcleo que realiza los cálculos se coloca en el “loop” y no hace nada hasta que se reciba algún mensaje por Bluetooth y la variable incoming cambie a algún valor conocido.

```
void loop() {
 if (incoming == 49){
      incoming=-1;
//Aquí se encuentra el código descrito en la sección 6.2.4
}
```

Esta es la función que se encarga de mandar datos por el USB, solo es útil para realizar pruebas durante el desarrollo y se podría retirar en la versión final del programa. Una vez envía los datos sale del bucle y no vuelve a entrar a menos que se lo pidas de nuevo.

```
 else if (incoming==48){
      digitalWrite(2,HIGH);
      tinit=micros();
      lasttime=0;
      tblueacumulado=0;
      lastblue=0;
      while (incoming==48){
        currenttime=micros();
        if (currenttime-lasttime>500){

//Aquí se encuentra el código que calcula los fasores explicado en la
//sección 6.2

          lasttime=currenttime;
          tblueacumulado= micros()-lastblue;
          if (tblueacumulado>=200000){
            lastblue+=tblueacumulado;
            datoscorr.tiempo=(long)(lasttime-tinit)/1000;
            datoscorr.dato1=isreal;
            datoscorr.dato2=isimag;
            datoscorr.pos=posicion;
            xQueueSend(enviar,&datoscorr,0);
          }
        }
      }
      digitalWrite(2,LOW);
   }
```

Esta es la parte del código que se encarga de enviar los fasores calculados por el Bluetooth. Como se puede ver por el while() el esp32 se mantendrá en este modo mientras no le mandes un mensaje para que pare o haga otra función. Mientras se encuentra en este modo el LED incorporado en la esp32 y unido a la patilla 2 se mantiene encendido, esto es útil para informar al usuario, pero no es estrictamente necesario.

Como se explicó anteriormente los cálculos se realizan cada 500 microsegundos y cada vez que se realiza un cálculo se comprueba si han pasado 200 milisegundos desde que se enviaron los datos a la interfaz, si ya ha pasado el tiempo necesario se almacenan los datos calculados en esa iteración en la estructura datoscorr y se envía el puntero de la estructura a través de un “queue” para que el núcleo encargado de las comunicaciones se entere de que ya hay datos listos para ser enviados. Se envía el puntero en vez de la estructura entera por ser una opción más rápida y funcionar igual de bien.

Es importante destacar que en la función xQueueSend(), que se encarga de colocar el puntero en la “queue”, se ha colocado el número 0. Este número indica el tiempo que debe esperar el procesador para enviar el mensaje a la “queue” si ésta se encuentra llena. Es importante poner un 0 para que el procesador no pierda tiempo esperando en el caso de que se llenara la “queue”, pues es preferible que falle el envío de datos a la interfaz y se pierda información a que se retrasen los cálculos. Pues no debemos olvidar que el procesador también se debe encargar de controlar el motor, aunque no

forme parte directamente de este tfg, y un retraso en su control podría ser peligroso y dañar los equipos.

7.2.2 Código en Android Studio

Android Studio es una plataforma de desarrollo mucho más compleja que Arduino IDE, pues desarrollar para un SO como Android es bastante más complicado que hacerlo para un microcontrolador. Por esta razón gran parte del código de la aplicación referente a establecer la conexión Bluetooth se ha obtenido del tutorial que se puede encontrar en (INNOVA DOMOTICS, 2019) y en (INNOVA DOMOTICS, 2019). Aunque el código obtenido del tutorial es, en general, el mismo que se explica con mayor profundidad en el foro oficial para desarrolladores de Android, en su sección de conexión Bluetooth (Android, 2019). Para adaptar la aplicación a las necesidades del proyecto se ha cambiado el aspecto gráfico de la aplicación, se ha implementado una gráfica para representar los datos y se ha adaptado la forma en la que se interpretan los datos recibidos por Bluetooth.

Android funciona con un sistema basado en actividades, las actividades son objetos especiales que representan una pantalla de la aplicación, por lo que se debe crear una actividad distinta para cada pantalla.

Una aplicación de Android Studio se divide principalmente en cuatro partes distintas:

1. El Manifiesto, es un pequeño archivo en formato xml que define las características principales de la aplicación y sus actividades. Algunos ejemplos son el nombre y el icono de la aplicación, el estilo de las actividades o la orientación que debe tener la pantalla en cada actividad. También contiene los permisos adicionales que pide la aplicación al iniciarse.
2. Los “layout” de las actividades, son archivos en formato xml que definen el aspecto gráfico inicial de cualquier actividad, aunque también se puede definir y cambiar el aspecto gráfico a través del código una vez iniciada la aplicación. Estos archivos .xml se pueden escribir directamente, pero es mucho más sencillo editar el diseño directamente gracias a una ventana de Android Studio que permite colocar todos los elementos y definir sus propiedades gráficas como color, tamaño y posición sobre una representación de cómo se verá una vez ejecutado.
3. Los archivos .java, que incluyen todo el código java de la aplicación. Normalmente se usa un archivo .java distinto para definir cada actividad y archivos .java auxiliares para definir clases nuevas.
4. Los recursos de la aplicación, aquí se incluyen los distintos archivos externos que necesita la aplicación y no se crean con Android Studio, los ejemplos más comunes son las imágenes o la música de la aplicación. En nuestro caso solamente almacenaremos el icono de la aplicación. En los recursos también se puede almacenar el texto de la aplicación, los colores o el estilo, lo que permite cambiar de forma sencilla el aspecto o el idioma de la aplicación, pero esto es algo necesario solo para aplicaciones profesionales diseñadas para lanzarse al gran público, por lo que lo ignoraremos en este TFG.

Las aplicaciones de Android no siguen una secuencia ordenada de instrucciones si no que reaccionan a las acciones del usuario, por lo que puede ser un poco complicado saber dónde se debe colocar el código a ejecutar. Hay tres formas principales de ejecutar código en Android, a través del ciclo de vida de las actividades, a través de “listeners” o a través de “Threads”.

Todas las actividades sufren un ciclo de vida con distintas etapas que se puede resumir en crear la actividad, iniciar la actividad, parar la actividad y destruir la actividad, además de algunas etapas auxiliares como pausar o volver a una actividad ya iniciada. Para pasar de una etapa a otra del ciclo de vida se ejecuta un método concreto, y en cada uno de estos métodos se puede ejecutar código. Principalmente usaremos el método onCreate(), que se ejecuta nada más crear la actividad, para cargar el xml que sirve de aspecto gráfico de la actividad, también se puede usar para inicializar las variables, objetos, “listeners” y “threads” que permitirán a la aplicación seguir funcionando, aunque también se puede hacer esto en métodos más avanzados como onResume(), que se ejecuta justo antes de mostrar la actividad al usuario. En la Figura 21 se explica de forma gráfica el ciclo de vida de las actividades, la imagen procede de (Android, 2019), donde se puede encontrar mucha más información al respecto de cómo hacer usar el ciclo de vida de las actividades de forma efectiva.

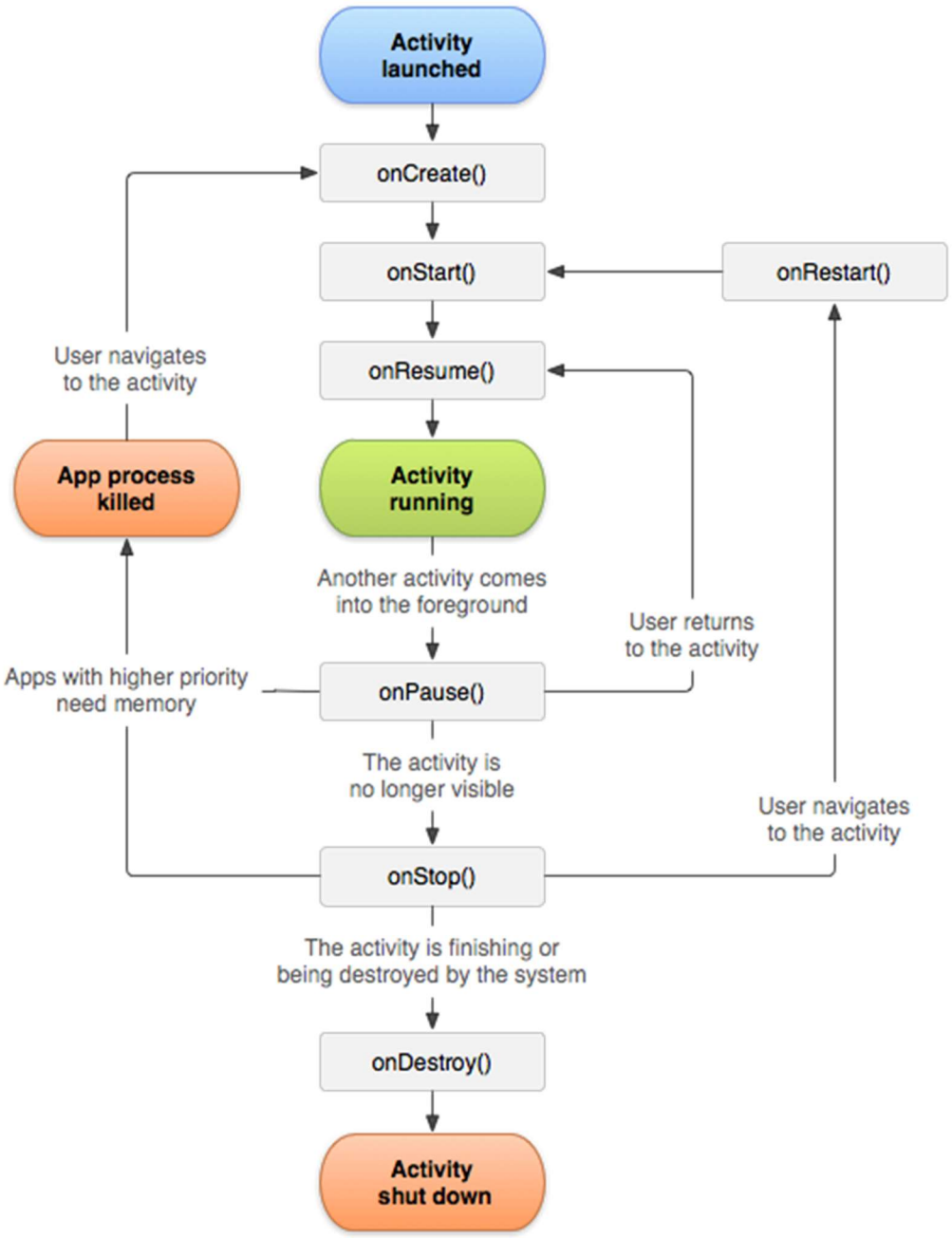

Figura 21. Diagrama del ciclo de vida de una actividad

Los “listeners” son métodos especiales de los objetos que están asociados a la parte gráfica de la aplicación, ejecutan código cuando el usuario realiza alguna acción sobre el objeto en cuestión, cómo apretar un botón, tocar la pantalla o desplazar una barra de progreso. Los “listeners” son la forma en la que el usuario puede interactuar con la aplicación

Finalmente los “threads” nos permiten ejecutar código continuamente de forma similar a como lo hace el “loop” del Arduino IDE, para usarlos se deben crear objetos especiales que heredan métodos de la clase “Thread” en nuestro caso usaremos un thread para revisar cuando llegan mensajes vía Bluetooth.

7.2.2.1 Manifiesto

Normalmente el manifiesto no es necesario editarlo y podemos dejar que Android Studio se encargue de crearlo por sí mismo, pero en este caso necesitamos añadir un par de líneas para indicar que la aplicación requiere permisos para usar el módulo Bluetooth del dispositivo.

Así sería el manifiesto completo, donde las dos primeras líneas son las que informan al dispositivo de que la aplicación requiere permiso para usar Bluetooth. Estas líneas es necesario incluirlas para que la aplicación funcione correctamente.

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>

<application
    android:allowBackup="true"
    android:icon="@mipmap/icono"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/icono"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">

    <activity android:name=".DispositivosBT"
        android:screenOrientation="portrait">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER"
/>
        </intent-filter>
    </activity>

    <activity android:name=".UserInterfaz"
        android:screenOrientation="portrait">

    </activity>

</application>
```

También se ha editado las líneas “android:icon="@mipmap/icono"” y “android:roundIcon="@mipmap/icono"”, pues son las que indican que archivo se debe cargar como icono de la aplicación. Y se ha añadido la línea “android:screenOrientation="portrait"” en la sección de ambas actividades para evitar que la imagen gire cuando se gire el dispositivo.

7.2.2.2 Primera actividad

La primera actividad del programa es simplemente una lista con los distintos dispositivos Bluetooth con los que se ha conectado el dispositivo Android previamente, cuando se selecciona un dispositivo de la lista la aplicación establece la conexión y lanza la segunda actividad. Esta actividad es prácticamente igual a la que se describe en (INNOVA DOMOTICS, 2019), pues no es necesario añadir ninguna función extra.

El diseño de esta actividad consiste tan solo en un TextView y un ListView, El TextView tan solo nos informa de que estamos viendo una lista de los dispositivos

vinculados. El ListView por el contrario es bastante más complejo y se deberá terminar de definir en la parte de código de Java.

El aspecto de la actividad se puede ver en la Figura 22, a la izquierda se puede ver su aspecto en el editor de Android Studio y a la derecha su aspecto cuando se ejecuta la aplicación en un dispositivo Android real

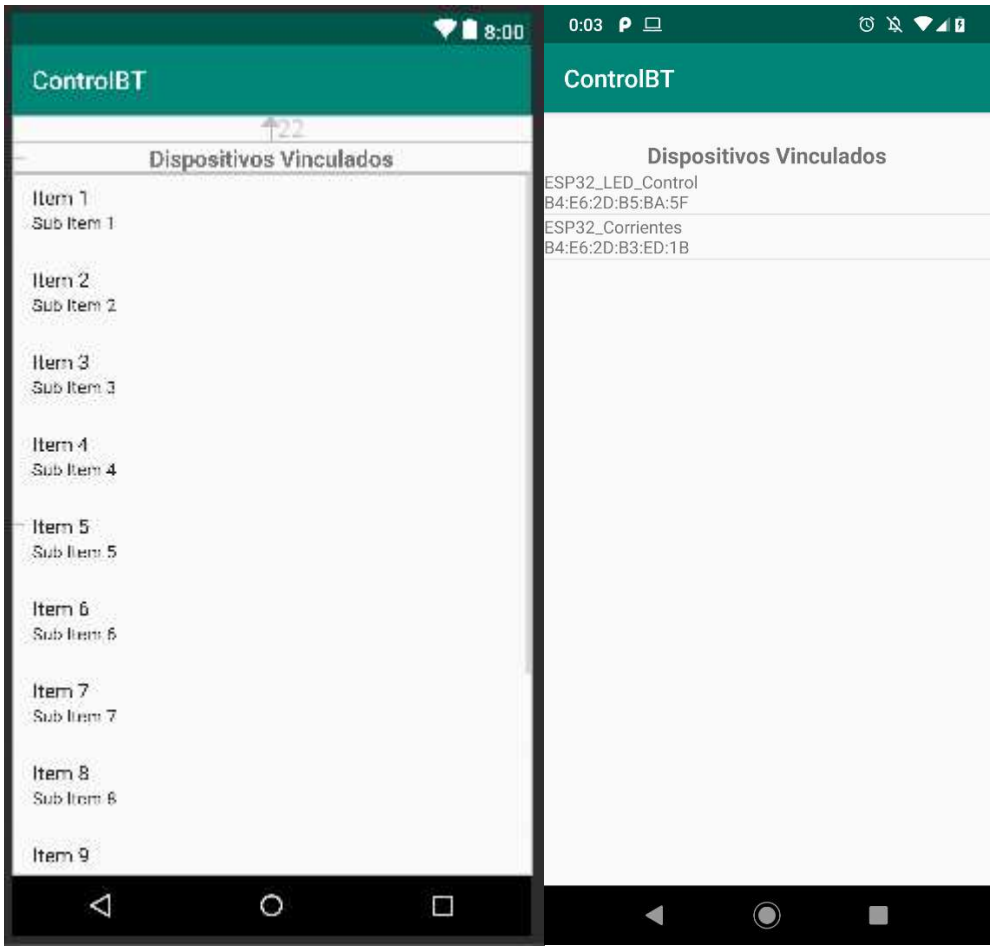

Figura 22. Aspecto de la actividad 1 en Android Studio (izq.) y Teléfono móvil (der.)

Para crear el ListView se deben definir en un xml auxiliar cómo son los elementos que van a aparecer en la lista, como nuestros elementos será solo texto el xml auxiliar será así:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
    android:layout_height="match_parent">

</TextView>
```

Como se puede ver es un simple TextView, es importante saber que el archivo se llama nombre_dispositivos.xml.

El código de Java de esta actividad, como en toda actividad empieza importando las librerías necesarias, se puede ver que hay un par de librerías de Bluetooth, mientras que la de Arrayadapter y AdapterView se encargan de realizar la lista.

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import java.util.Set;
```

A continuación se define nuestra actividad, que como todo en Android Studio, es un objeto. La actividad se llama DispositivosBT. A continuación se declaran los objetos. También se declara una String que nos ayudará más adelante a enviar información a la segunda actividad.

```
public class DispositivosBT extends AppCompatActivity {
    ListView IdLista;
    private BluetoothAdapter mBtAdapter;
    private ArrayAdapter mPairedDevicesArrayAdapter;
    public static String EXTRA_DEVICE_ADDRESS = "device_address";
```

A continuación se define el método onCreate que vincula la actividad con su archivo xml que define su aspecto gráfico.

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_dispositivos_bt);
    }
```

En el método onResume se llama al método VerificarEstadoBT, que se encarga de asegurarse que el dispositivo tiene un módulo Bluetooth funcionando. A continuación se asocia el ArrayAdapter con el xml “nombre_dispositivos” mencionado anteriormente y se coloca el Adapter en la lista. Además se establece un “listener” en la lista. Finalmente se crea el objeto BluetoothAdapter y mediante un método de este objeto se recuperan todos los dispositivos Bluetooth vinculados en el dispositivo y sus direcciones MAC y se colocan en la lista a través del ArrayAdapter gracias a un bucle for.

```
    @Override
    public void onResume()
    {
        super.onResume();

        VerificarEstadoBT();

        mPairedDevicesArrayAdapter = new ArrayAdapter(this,
R.layout.nombre_dispositivos
```

```
        IdLista = (ListView) findViewById(R.id.IdLista);
        IdLista.setAdapter(mPairedDevicesArrayAdapter);
        IdLista.setOnItemClickListener(mDeviceClickListener);
        mBtAdapter = BluetoothAdapter.getDefaultAdapter();

        Set <BluetoothDevice> pairedDevices =
mBtAdapter.getBondedDevices();


        if (pairedDevices.size() > 0)
        {
            for (BluetoothDevice device : pairedDevices)
mPairedDevicesArrayAdapter.add(device.getName() + "\n" +
device.getAddress());
            }
        }
    }
```

Ya fuera del método onCreate se define el método VarificarEstadoBT(), este método trata de crear un objeto BluetoothAdapter, si es incapaz de crearlo informa al usuario de que el dispositivo no es compatible, mientras que si el bluetooth no está activado crea un Intent que pregunta al usuario si quiere activarlo hasta que acepte.

```
    private void VerificarEstadoBT() {
        mBtAdapter= BluetoothAdapter.getDefaultAdapter();
        if(mBtAdapter==null) {
            Toast.makeText(getBaseContext(), "El dispositivo no
soporta Bluetooth", Toast.LENGTH_SHORT).show();
        } else {
            if (mBtAdapter.isEnabled()) {
                Log.d("DispositivosBT", "...Bluetooth Activado...");
            } else {
                Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent, 1);

            }
        }
    }
```

El último método de esta actividad es el “listener” de la lista, en resumen este listener se encarga de, una vez el usuario toca alguno de los dispositivos de la lista, obtiene en un String la dirección MAC del dispositivo Buetooth seleccionado, lanza la segunda actividad con un Intent e incluye en el Intent la dirección MAC por medio de un Extra.

```
    private AdapterView.OnItemClickListener mDeviceClickListener =
new AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView av, View v, int arg2, long
arg3) {

            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);

            Intent i = new Intent(DispositivosBT.this,
UserInterfaz.class);
            i.putExtra(EXTRA_DEVICE_ADDRESS, address);
            startActivity(i);
        }
    };
```

En resumen, esta actividad simplemente pide al usuario que active el Bluetooth y envía a la segunda actividad la dirección MAC del dispositivo al que se desea vincular, pues la dirección MAC es la única información necesaria para establecer una conexión Bluetooth.

7.2.2.3 Segunda Actividad

La segunda actividad es la que nos permite visualizar las componentes de los fasores calculados por la esp32 y mandarle órdenes, aunque también cuenta con código de (INNOVA DOMOTICS, 2019) como la primera actividad, en este caso se han añadido varios cambios y funciones nuevas, en concreto se ha ideado un sistema para interpretar los distintos mensajes que manda la esp32 correctamente, también se ha cambiado el diseño gráfico, y lo más importante, se ha añadido una gráfica que recoge los datos enviados por el esp32, esta gráfica se ha realizado con la librería AChartEngine.

AChartEngine es una potente librería de código abierto y gratuita que permite crear todo tipo de gráficas y personalizarlas de diversas formas, la librería y la información sobre cómo usarla se puede encontrar en Github (ddanny, 2019). Para añadir la librería al proyecto de Android Studio basta con descargar el archivo .jar de la librería, pegarlo en la carpeta libs dentro de la carpeta app clicar botón derecho y hacer clic en añadir librería.

El diseño de esta actividad es más complejo que el de la actividad uno, consta de tres botones (“Button”), la gráfica, y cuatro campos de texto (“TextView”).

Los botones superiores sirven para dar órdenes a la esp32, el primero activa o desactiva el envío y cálculo de datos de la esp32, mientras que el segundo activa el envío de datos por el USB para poder estudiarlos en MatLab, como se explicó en 6.2.4. El tercer botón se encarga de parar el envío de datos, cerrar la comunicación Bluetooth y terminar la actividad.

Los campos de texto sirven para mostrar los últimos datos recibidos de la esp32, muestran el tiempo, la componente d y la componente q del fasor del estator y la posición del eje. La posición no es realmente útil recibirla salvo para asegurarnos de que el codificador rotativo funciona correctamente, mientras que la componente –q no se incluye por comportarse de forma muy similar a la componente q. Estos datos se pueden cambiar más delante cuando el proyecto este completo de forma sencilla, por ejemplo, sería mucho más interesante recibir un dato como la velocidad en vez de la posición.

El aspecto de la actividad se puede ver en la Figura 23, a la izquierda en el editor de Android Studio y a la derecha en un teléfono móvil, la gráfica no es visible porque se ha decidido mantenerla invisible al inicio la actividad, se hace visible al empezar a recibir datos, se puede ver el aspecto de la gráfica en la Figura 24.

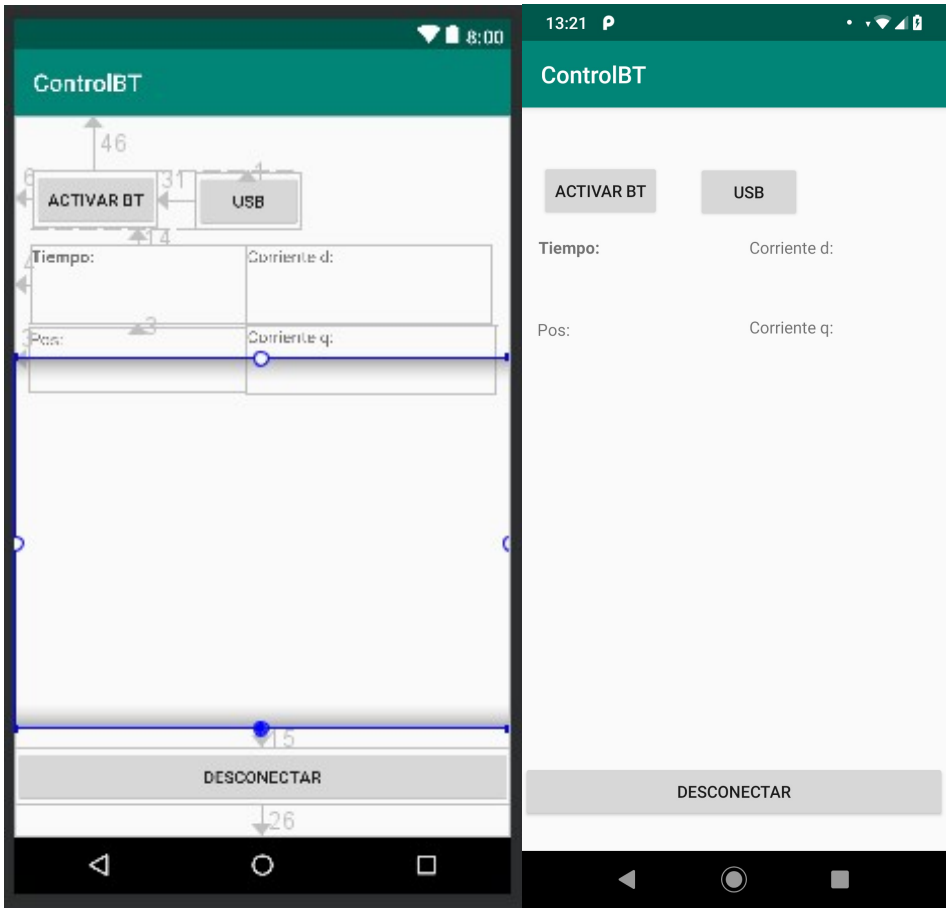

Figura 23. Aspecto de la Actividad 2 en Android Studio (izq.) y teléfono móvil (der.)

En cuanto al código de Java, primero se empieza importando las librerías pertinentes, en esta ocasión varias de las librerías pertenecen a AChartEngine.

```
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.graphics.Color;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import org.achartengine.ChartFactory;
import org.achartengine.GraphicalView;
import org.achartengine.model.XYMultipleSeriesDataset;
import org.achartengine.model.XYSeries;
import org.achartengine.renderer.XYMultipleSeriesRenderer;
import org.achartengine.renderer.XYSeriesRenderer;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;
```

A continuación se crea la clase de la actividad, llamada UserInterfaz, y se declaran las diferentes variables y objetos que se van a necesitar

```
public class UserInterfaz extends AppCompatActivity {

    private static final UUID BTMODULEUUID =
UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
    private static String address = null;
    final int handlerState = 0;
    Button Idbt, IdUSB,IdDesconectar;
    TextView IdTiempo,IdReal,IdImag,IdPos;
    double tiempo,Iq,Id;
    boolean enviodatos=false;
    int contador=0;
```

El objeto UUID es un número de identificador único universal, y es necesario elegir uno para establecer la conexión Bluetooth. Este número se usa en el ámbito de la comunicación Bluetooth para saber qué servicios ofrece un dispositivo, más información al respecto en (Stackoverflow, 2019).

Las variable enviodatos y contador son variables auxiliares que nos ayudarán más adelante.

```
    Handler bluetoothIn;
    private BluetoothAdapter btAdapter = null;
    private BluetoothSocket btSocket = null;
    private StringBuilder DataStringIN = new StringBuilder();
    private ConnectedThread MyConexionBT;

    private LinearLayout IdGraph;
    private GraphicalView graph;
    private XYMultipleSeriesRenderer renderer =new
XYMultipleSeriesRenderer();
    private XYMultipleSeriesDataset dataset=new
XYMultipleSeriesDataset();
    private XYSeries serie1, serie2;
    private XYSeriesRenderer serie1r, serie2r;
```

Se deben declarar un BluetoothAdapter y un BluetoothSocket para crear la conexión Bluetooth. El objeto ConnectedThread se explicará más adelante, pero básicamente hereda la clase “Thread” para poder recibir y enviar los mensajes Bluetooth continuamente. El objeto Handler funciona de forma similar los “queues” del Arduino IDE, permite dejar un mensaje y el Handler realiza una serie de instrucciones como respuesta.

También se deben declarar muchos objetos para hacer funcionar la gráfica, en resumen, los renderer definen el aspecto de la gráfica completa o de una serie de valores, mientras que las propias series y el dataset incluyen los propios valores de la gráfica.

En el método onCreate() se vinculan los objetos gráficos con su contraparte en código mediante el método findViewById().

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_user_interfaz);
        Idbt = (Button) findViewById(R.id.idbt);
        IdUSB = (Button) findViewById(R.id.idusb);
        IdDesconectar = (Button) findViewById(R.id.IdDesconectar);
        IdTiempo = (TextView) findViewById(R.id.Idtiempo);
        IdReal = (TextView) findViewById(R.id.Idreal);
        IdImag = (TextView) findViewById(R.id.Idimag);
        IdPos = (TextView) findViewById(R.id.Idpos);
        IdGraph=findViewById(R.id.Idgraph);
```

A continuación se definen todas las características de aspecto de la gráfica y se crean y asocian los distintos objetos de la gráfica entre ellos. Las características de la gráfica que se han editado son: el color de los márgenes se ha cambiado a blanco, las leyendas del eje x e y se han cambiado a negro, se ha activado una malla, se ha aumentado el tamaño del texto y el grosor de las líneas y se le ha asignado el color rojo a la componente d y el azul a la componente q.

```
        serie1=new XYSeries("componente d");
        serie2=new XYSeries("componente q");
        dataset.addSeries(serie1);
        dataset.addSeries(serie2);
        serie1r=new XYSeriesRenderer();
        serie2r=new XYSeriesRenderer();
        renderer.addSeriesRenderer(serie1r);
        renderer.addSeriesRenderer(serie2r);
        graph = ChartFactory.getLineChartView(this,dataset,renderer);
        IdGraph.addView(graph);
        renderer.setMarginsColor(Color.WHITE);
        renderer.setShowGrid(true);
        renderer.setXLabelsColor(Color.BLACK);
        renderer.setYLabelsColor(0,Color.BLACK);
        renderer.setLegendTextSize(30);
        renderer.setLabelsTextSize(30);
        serie1r.setColor(Color.RED);
        serie2r.setColor(Color.BLUE);
        serie1r.setLineWidth(6);
        serie2r.setLineWidth(6);
```

También se crea el objeto Bluetooth adapter y se llama a la función VerificarEstadoBT(), que es exactamente igual a la que se explica en 7.2.2.2.

```
        btAdapter = BluetoothAdapter.getDefaultAdapter();
        VerificarEstadoBT();
```

En el método onResume() se recupera la dirección MAC del dispositivo al que quiere conectarse por medio del Extra guardado en el intent. Después se crea un objeto BluetoothDevice con la dirección MAC, y finalmente se intenta crear el objeto BluetoothSocket, que es el encargado de establecer la conexión Bluetooth con el esp32.

```
    @Override
    public void onResume()
    {
        super.onResume();
        Intent intent = getIntent();
        address =
intent.getStringExtra(DispositivosBT.EXTRA_DEVICE_ADDRESS);
        BluetoothDevice device = btAdapter.getRemoteDevice(address);
```

```
        try
        {
            btSocket = createBluetoothSocket(device);
        } catch (IOException e) {
            Toast.makeText(getBaseContext(), "La creacción del Socket
fallo", Toast.LENGTH_LONG).show();
        }
        try
        {
            btSocket.connect();
        } catch (IOException e) {
            try {
                btSocket.close();
            } catch (IOException e2) {}
        }
        MyConexionBT = new ConnectedThread(btSocket);
        MyConexionBT.start();
    }
```

Para crear el objeto btSocket se ha usado el método createBluetoothSocket, que simplemente llama al método createRfcommSocketToServiceRecord() con el UUID mencionado anteriormente

```
private BluetoothSocket createBluetoothSocket(BluetoothDevice device)
throws IOException
    {
        return device.createRfcommSocketToServiceRecord(BTMODULEUUID);
    }
```

Al final del método onResume() se crea el objeto ConnectedThread y se inicia. El objeto ConnectedThread es bastante complejo, hereda la clase Thread para poder ejecutarse continuamente, y declara objetos de la clase InputStream y OutputStream, que son los encargados de enviar y recibir mensajes a través de la conexión Bluetooth.

```
    private class ConnectedThread extends Thread
    {
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;
```

El constructor de la clase, que es un método especial que solo se puede ejecutar durante la creación de un objeto de esa clase, se encarga de enlazar los InputStream y OutputStream con el Bluetooth Socket creado anteriormente en el onResume().

```
        public ConnectedThread(BluetoothSocket socket)
        {
            InputStream tmpIn = null;
            OutputStream tmpOut = null;
            try
            {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) { }
            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }
```

A continuación se define el método run(), que se ejecutará continuamente de forma paralela a la ejecución de la actividad principal para revisar si llegan datos nuevos. Los datos que recibe se envían por medio de un mensaje al Handler, que se encargará de usarlos como convenga. El vector de bytes, buffer, incluye los propios datos, mientras que la variable bytes es el tamaño del vector. Puesto que los datos se envían en código ASCII, y en código ASCII cada carácter es un byte, cada miembro del vector buffer corresponde a un carácter.

```
        public void run()
        {
            byte[] buffer = new byte[4096];
            int bytes;
            while (true) {
                try {
                    bytes = mmInStream.read(buffer);
                    String readMessage = new String(buffer, 0, bytes);
                    bluetoothIn.obtainMessage(handlerState, bytes, -1,
readMessage).sendToTarget();
                } catch (IOException e) {
                    break;
                }
            }
        }
```

Además del método run(), la clase ConnectedThread tiene el método write(), que como se puede ver a continuación simplemente se encarga de enviar los datos por Bluetooth a través del OutputStream, Si hay algún error de conexión la actividad se cierra y se avisa al usuario.

```
        public void write(String input)
        {
            try {
                mmOutStream.write(input.getBytes());
            }
            catch (IOException e)
            {
                Toast.makeText(getBaseContext(), "La Conexión fallo",
Toast.LENGTH_LONG).show();
                finish();
            }
        }
    }
```

Pasemos ahora a los “listeners” de los botones, el botón del USB lo único que hace es enviar un carácter 1 a la esp32 a través del objeto ConnectedThread (que se llama MyConexionBT). El número uno en código ASCII corresponde con el 49, por eso en la sección 7.2.1 se espera a recibir el valor 49 para activar el envío de datos por USB. También se hace falsa la variable enviodatos, pues enviodatos nos ayuda a saber cuándo está la esp32 enviando datos en el modo Bluetooth.

```
        IdUSB.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                MyConexionBT.write("1");
                enviodatos=false;
            }
        });
```

El “listener” del botón Bluetooth es un poco más complejo, primero se comprueba si se están enviando datos gracias a la variable enviodatos mencionada anteriormente, si no se están enviando datos (la variable es falsa), se envía un cero a el esp32, el carácter cero equivale en ASCII al número 48, que es el valor que comprueba el esp32 en la sección 7.2.1. Si ya se están enviando datos se envía un ‘#’ (aunque valdría cualquier símbolo que no fuese el 1 o el 0) para que el esp32 deje de enviar datos. Además cuando empieza el envío de datos se eliminan los datos de la gráfica que pudiesen haber quedado almacenados en los objetos serie1 y serie2. También se hace visible la gráfica, pues al inicio de la actividad la gráfica es invisible (se ha definido así en el xml). Finalmente también se cambia el texto del botón para indicar que hará al pulsarlo, si activar o desactivar el envío de datos.

```
        Idbt.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                if(enviodatos){
                    Idbt.setText("Activar BT");
                    MyConexionBT.write("#");
                    enviodatos=false;
                }else {
                    Idbt.setText("Desactivar BT");
                    MyConexionBT.write("0");
                    IdGraph.setVisibility(View.VISIBLE);
                    enviodatos=true;
                    serie1.clear();
                    serie2.clear();
                }
            }
        });
```

En el “listener” del botón desconectar se envía también un ‘#’ para evitar que el esp32 continúe enviando y calculando datos una vez desconectado, además se intenta cerrar la conexión mediante un método del objeto btSocket y se termina la actividad, volviendo a la actividad 1.

```
        IdDesconectar.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                MyConexionBT.write("#");
                if (btSocket!=null)
                {
                    try {btSocket.close();}
                    catch (IOException e)
                    { Toast.makeText(getBaseContext(), "Error",
Toast.LENGTH_SHORT).show();}
                }
                finish();
            }
        });
```

En el caso de que se cambie de aplicación se llamará al método onPause() que se encarga también de cerrar la conexión como hace el botón de desconexión.

```
    @Override
    public void onPause()
    {
        MyConexionBT.write("#");
        super.onPause();
        try
        {
            btSocket.close();
        } catch (IOException e2) {}
    }
```

Finalmente solo queda mostrar el Handler, que es posiblemente la parte más interesante de la aplicación, pues es la que se encarga de recoger los datos y mostrarlos al usuario. El objeto se declara en onCreate(), primero espera a que le llegue un mensaje, si el mensaje has sido enviado por el objeto ConnectedThread procede a almacenar los caracteres en un StringBuilder.

```
bluetoothIn = new Handler() {
public void handleMessage(android.os.Message msg) {
    if (msg.what == handlerState) {
        String readMessage = (String) msg.obj;
        DataStringIN.append(readMessage);
```

Ahora debemos recordar cómo envía los datos el esp32, como se puede ver en la sección 7.2.1 los datos se envían siguiendo una secuencia ordenada de tiempo, componente d, componente q y posición, y entre cada dato se intercala el carácter '#'. Además se envían en formato ASCII, por lo que ya están listos para almacenarse en Strings y colocarse como texto, mientras que para integrarlos en la gráfica se deberán usar los métodos parseDouble para transformar el texto en valores numéricos.

Para interpretar los datos primero se comprueba donde se encuentra el primer carácter '#', en el caso de que no se encuentre al principio del StringBuilder se almacenan todos los caracteres desde el principio hasta el '#' en una String y se procede a interpretar los datos, finalmente se eliminan todos los caracteres del StringBuilder desde el principio hasta el primer '#' incluido, haciéndolo de esta forma nos aseguramos de que no se pierda información.

```
        int endOfLineIndex = DataStringIN.indexOf("#");
        if (endOfLineIndex > 0) {
            String dataInPrint = DataStringIN.substring(0,
endOfLineIndex);
            if(contador==0){
                IdTiempo.setText("Tiempo: " + dataInPrint+"ms");
                contador=1;
                tiempo = Double.parseDouble(dataInPrint) / 1000;
            }
            else if(contador==1){
                IdReal.setText("Corriente d: " + dataInPrint+"");
                contador=2;
                Id = Double.parseDouble(dataInPrint);
                serie1.add(tiempo,Id);
            }
            else if(contador==2){
                IdImag.setText("Corriente q: " + dataInPrint+"");
                contador=3;
                Iq = Double.parseDouble(dataInPrint);
                serie2.add(tiempo,Iq);
```

```
                graph.repaint();
            }
            else if(contador==3){
                IdPos.setText("Pos: " + dataInPrint+"");
                contador=0;
            }
        }DataStringIN.delete(0, endOfLineIndex+1);
    }
}
```

Para interpretar los datos del String necesitaremos la ayuda de la variable contador, que nos informa de que dato toca interpretar en este momento. El primer dato que nos llega es el tiempo, que viene representado en milisegundos, se coloca el valor como texto y se almacena en una variable en forma de segundos. Los siguientes dos datos son las componentes de corriente, también se escriben en forma de texto y se almacena el valor en una variable. A continuación se añaden los datos numéricos a su correspondiente serie de la gráfica y finalmente se llama al método repaint() para actualizar el aspecto de la gráfica. El último dato que se recibe es la posición, que simplemente se escribe en forma de texto.

En la Figura 24 se puede ver el aspecto de la actividad cuando se reciben datos, aunque en este caso son datos aleatorios que no provienen del cálculo de fasores de corriente del motor.

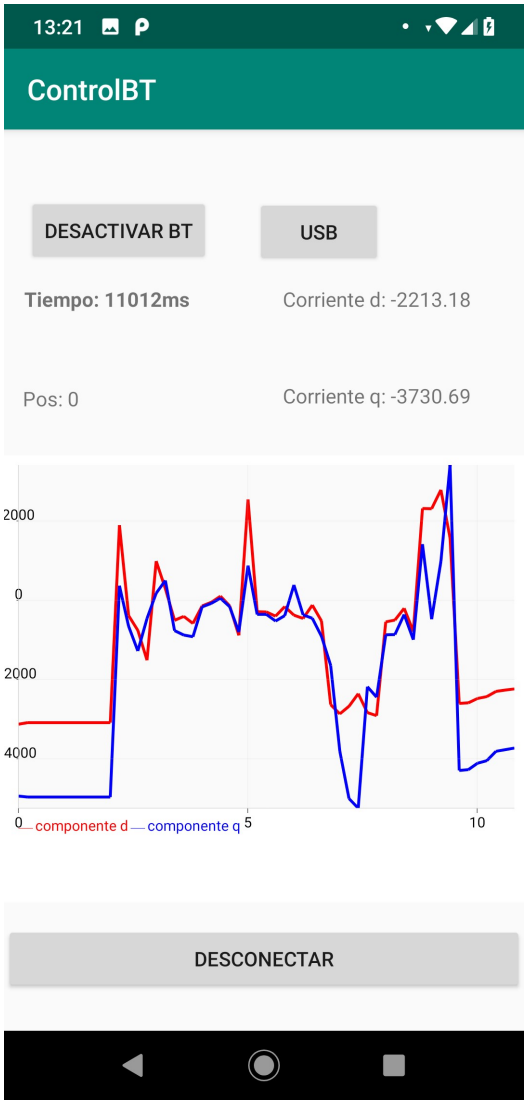

Figura 24. Aspecto de la actividad 2 cuando se reciben datos

Este es el resumen de cómo funciona la aplicación de Android sin entrar en demasiados detalles de programación. Cuando el proyecto completo esté terminado la aplicación se actualizará para mostrar también la velocidad de giro y permitir elegir el par al que se desea que trabaje el motor, funciones que no se han incluido todavía por no formar parte de este TFG.

Aunque la aplicación usa bastante código de (INNOVA DOMOTICS, 2019) para controlar la conexión Bluetooth, gran parte de la información sobre el funcionamiento básico de Android Studio y Java que se ha puesto en esta memoria ha sido adquirido gracias a la asignatura de Desarrollo de Aplicaciones para Dispositivos Móviles impartida en la UPV para el grado en Ingeniería en Tecnologías Industriales y sus diapositivas (García García & Gil Gómez, 2018-2019).

8 RESULTADOS

Una vez montado finalmente el prototipo y terminado el programa de la esp32 y la aplicación de Android sólo nos queda comprobar que funciona correctamente. Aunque este TFG sólo comprende una parte del proyecto de control y no se puede comprobar si el control por orientación de campo funciona correctamente, podemos comprobar el correcto funcionamiento de la medida de corrientes haciendo funcionar el motor conectado a la red y observando el comportamiento de sus corrientes. Si provocamos un transitorio se puede observar el cambio en las componentes de corriente asociadas al par y a la magnetización.

Los ensayos consisten en poner el motor a trabajar en carga a su tensión y velocidad nominal y calcular sus componentes fasoriales con la esp32. Para simular una carga estable en el motor se usará el mismo método que el usado en los ensayos de carga de la asignatura de Máquinas Eléctricas.

El método que se usará consiste en unir el eje del motor que se está ensayando a un segundo motor auxiliar. Este segundo motor en vez de alimentarse de la red eléctrica se alimentará a través de un variador de frecuencia, concretamente un Micromaster Vector producido por Siemens (Siemens, 2019). El variador de frecuencia nos permite alimentar el motor a la frecuencia que deseemos, en vez de los 50Hz de la red eléctrica, aprovecharemos esto para alimentar a este segundo motor a una menor frecuencia, de esta forma el eje acabará girando a una velocidad intermedia entre la velocidad de sincronismo de ambos motores. Cuanto menor sea la frecuencia elegida, mayor el deslizamiento del motor principal y mayor su grado de carga. Para reproducir una carga variable se utiliza este método también, pero con un control adicional que permite cambiar bruscamente la frecuencia de alimentación del motor auxiliar.

El eje girará más lento de lo normal para el motor que se está ensayando, lo que aumentará el par que debe proporcionar y su consumo de potencia, el segundo motor, por el contrario, girará más rápido de lo normal. Esto significa que empezará a funcionar como generador y devolverá corriente a la red, este fenómeno hará saltar la protección de la red a menos que esté preparada para ello. Afortunadamente se puede evitar el envío de energía a la red gracias al Micromaster Vector y a un equipo de resistencias que pueden disipar la potencia generada. Las resistencias se conectan al variador de frecuencia y este se encarga de desviar la energía generada por el segundo motor a las resistencias, donde se disipa sin peligro en forma de calor.

Antes de aportar tensión al motor a ensayar se debe cerciorar de que el eje gira a la velocidad nominal del motor. El motor a ensayar es dos pares de polos, por lo que girará a 1500rpm al conectarlo a la red, por el contrario el segundo motor es de un solo par de polos, por lo tanto hay que alimentarlo a 25Hz para que gire a 1500rpm, también hay que tener especial cuidado en que el sentido de giro de ambos motores sea el mismo, pues conectarlos con sentidos de giro distintos podría dañar los motores y poner en peligro a las personas presentes y a dispositivos cercanos si no actuaran las protecciones.

Figura 25. Montaje completo del banco de ensayos.

En la Figura 25 se puede observar el montaje completo del banco de ensayos, a falta de la serie de resistencias empleadas para disipar energía y el autotransformador que regula la tensión de alimentación del motor a ensayar, que se encuentran bajo el banco. El motor azul es el motor que se pretende ensayar, mientras que el gris es el motor auxiliar que simula la carga. El equipo de color gris oscuro es el variador de frecuencia Micromaster Vector que alimenta el motor secundario (gris) a la frecuencia deseada, con el interruptor de su derecha se provocan los transitorios. El aparato transparente del centro es un sistema de medida externo que nos permite observar diferentes características de la línea de alimentación del motor a ensayar (azul), como la tensión, la corriente, o la potencia eficaz de cada una de las fases de línea. Finalmente, a la izquierda se encuentra el prototipo ya montado en su carcasa de protección (aunque sin la tapa) y justo debajo el inversor encargado de generar la corriente de alimentación del motor según le indica el microcontrolador, aunque este último dispositivo no forma parte de este TFG, si es muy importante en el desarrollo del sistema de control.

Se usará el modo USB para pasar los datos de los ensayos a MatLab, pues las gráficas de MatLab son mucho más versátiles que la diseñada en la aplicación, y usa vectores de almacenamiento en vez de medidas en tiempo real, por lo que podemos representar datos cada 500 microsegundos frente a los 200 milisegundos del modo Bluetooth. Por lo tanto es bastante más apropiado para comprobar el correcto funcionamiento del sistema. Este modo tiene el inconveniente de que solo puede medir durante un tiempo muy limitado debido a la falta de memoria donde guardar vectores, pero es un problema asumible, pues podemos grabar datos durante un segundo, tiempo suficiente para poder capturar el estado inicial del sistema y el transitorio completo hasta que el sistema se estabiliza de nuevo.

Se hicieron numerosos ensayos en distintas situaciones de carga del motor, se pueden dividir en dos categorías distintas:

- Ensayos a régimen permanente, se mantiene la velocidad de giro/potencia consumida constante. Se realizó el ensayo a 100, 200, 350, 500 y 635 Vatios.
- Ensayos a con un transitorio, se cambia la velocidad de giro/potencia durante el ensayo. Se realizaron ensayos con escalones tanto de aumento como de bajada de potencia, en concreto de 23.8Hz a 22.3Hz (de 400W a 600W), de 23.5Hz a 22.5Hz (de 450W a 550W) y sus contrapartes, de 22.3Hz a 23.8Hz y de 22.5Hz a 23.5Hz.

Nos centraremos en uno de los ensayos con transitorio, pues los transitorios son más ilustrativos al incluir dos estados distintos de funcionamiento en el mismo ensayo y permitir observar el transitorio de uno a otro.

8.1 Ensayo de escalón

En estos ensayos se cambia la velocidad de alimentación del segundo motor lo más rápido posible gracias al variador de frecuencia, esto provoca un cambio repentino en el par y en la potencia consumida por el motor a ensayar. En este caso el cambio de frecuencia del motor auxiliar se hará de 23.8Hz a 22.3Hz, este cambio provoca un transitorio en el motor principal que pasa de consumir una potencia de 400W a 600W, girará más lento, por lo que aumenta el deslizamiento y con ello el par aportado.

Al producirse el escalón la amplitud de las fases del estator aumenta ligeramente, pero el efecto es mucho más significativo si miramos a las fases del rotor. En la Figura 26 se puede ver la respuesta de las tres fases del rotor cuando se aumenta la potencia, tal y como dice la teoría, tanto la amplitud como la frecuencia aumentan de forma significativa, pues la amplitud está relacionada con el par y la frecuencia con el deslizamiento.

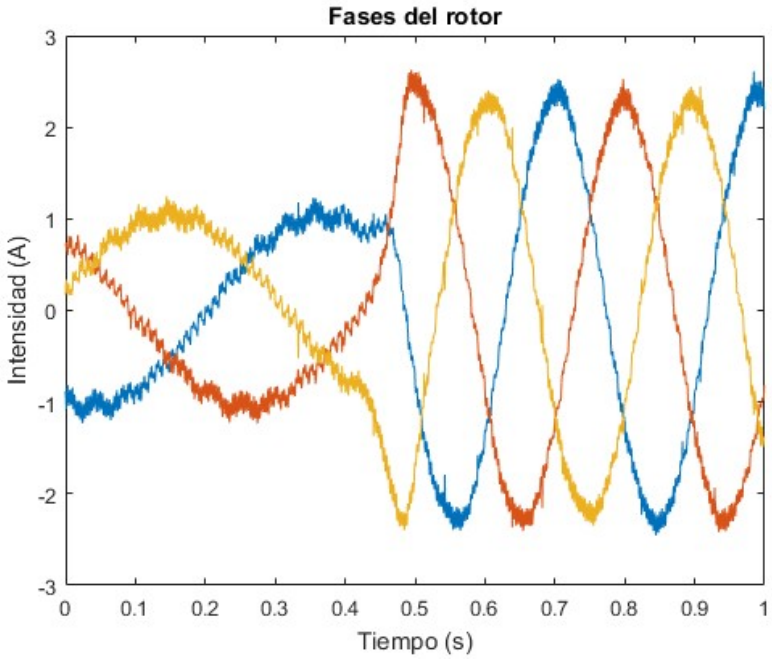

Figura 26. Transitorio de las fases del rotor al aumentar el par

Si miramos a los fasores de corriente en los ejes d,q podemos observar que el aumento en la carga del motor incrementa significativamente las componentes q y –q de los fasores de corriente. En el caso del eje –q está claro, pues el eje –q es igual al fasor de corriente del rotor, por lo que si aumenta la amplitud de sus fases también debe aumentar el módulo de su fasor. En el caso del fasor del estator sin embargo no funciona exactamente igual, pues aunque la componente q aumenta de igual forma que la –q, la componente d apenas varía, y de hecho hasta se reduce ligeramente. Este es justo el comportamiento que se debía esperar ante un cambio repentino de la carga de un motor conectado a la red de alimentación trifásica de tensión y frecuencia constante: un aumento brusco de carga se traduce en un aumento brusco de la corriente del rotor que, por trabajar con deslizamientos muy bajos, es prácticamente toda activa (productora de par). Ese aumento de la corriente del rotor se traduce en un aumento similar de la componente activa (q) de la corriente de estator y, como el motor está conectado a la red de tensión constante, apenas si produce cambios en la corriente magnetizante (de eje d) del estator. Es cierto que se produce una pequeña reducción de la corriente de eje d debido a que un aumento de carga supone una mayor corriente de estator y una mayor caída de tensión en la resistencia e inductancia de dispersión del estator, es decir, una ligera desmagnetización de la máquina. Este comportamiento se puede apreciar en la Figura 27 y en la Figura 28, donde se representan los fasores por componentes y en el plano complejo.

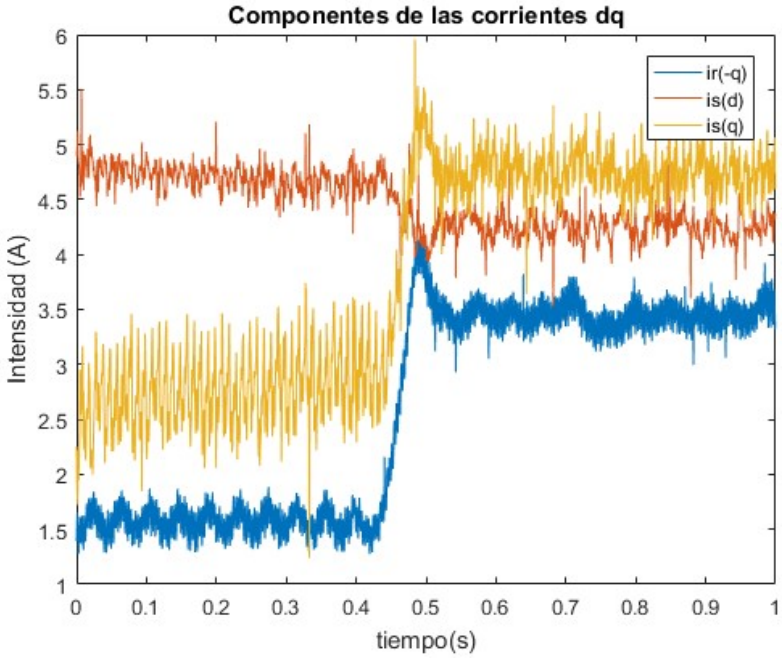

Figura 27. Componentes fasoriales en los ejes d,q durante un escalón

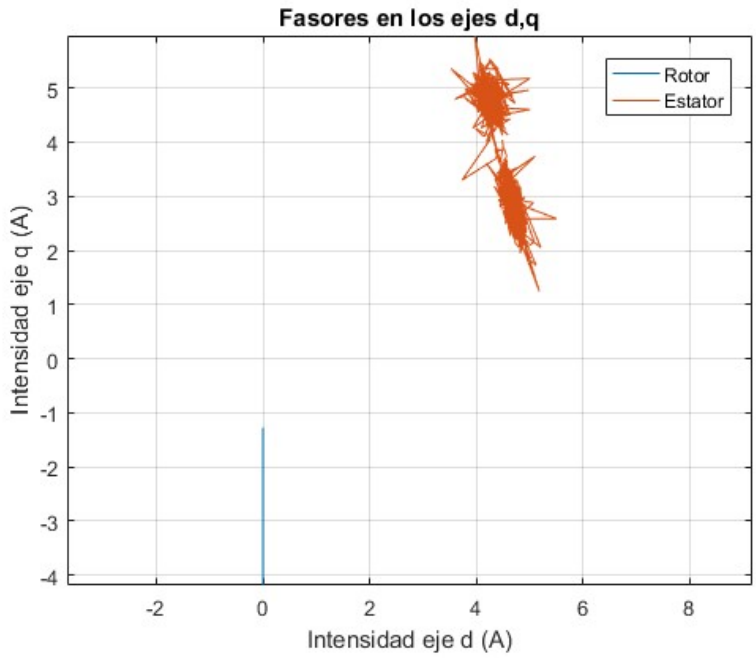

Figura 28. Fasores en los ejes d,q durante un transitorio en el plano complejo

El ángulo del fasor del estator en los ejes d,q aumenta con la potencia, en este caso el ángulo varía de 30º a 48º entre antes y después de producirse el salto.

8.2 El problema de los ensayos a baja potencia

Una de las primeras limitaciones que se descubrieron al realizar las primeras medidas de corriente es que las fases de corriente de rotor no siguen una senoidal limpia, tienen una oscilación de menor frecuencia acoplada, esta oscilación no es ruido, proviene de la distribución en ranuras de los devanados en el interior del motor. Si los devanados se pudiesen distribuir de uniformemente a lo largo del motor no ocurriría este problema, pero eso complicaría en exceso la construcción del motor.

Cuando la amplitud de corriente del rotor es lo suficientemente grande las oscilaciones no suponen ningún inconveniente importante, aunque dificultan obtener el ángulo del fasor del rotor. Esto se puede ver en la Figura 14, donde el ángulo del rotor no avanza de forma lineal como lo hace el del estator, si no que avanza “a saltos”. No saber con absoluta precisión el ángulo del rotor afecta directamente a la representación del fasor del estator en los ejes d,q, y es la principal razón por la que el fasor oscila incluso cuando el motor se encuentra en régimen permanente.

Sin embargo, cuando se realizan ensayos a baja potencia, por debajo de los 200 vatios en el motor ensayado, la amplitud de corriente del rotor puede ser tan pequeña que las oscilaciones causadas por el ranurado sean del orden de magnitud de la propia amplitud de la corriente. Esto causa que el ángulo del fasor del rotor empiece a variar de forma aleatoria y no se pueda realizar la representación de los fasores en los ejes d,q.

En resumen, aunque el cálculo de las componentes del fasor espacial de corriente del estator tal y como está ahora mismo implementado no proporciona valores adecuados cuando el motor se encuentra en potencias bajas (de 100 a 200 vatios en un motor de

600 vatios de potencia nominal) esta limitación no es realmente importante, ya que puede suplirse fácilmente con una estimación muy precisa de ambas componentes a partir del módulo del fasor espacial de corriente del rotor (para i_{sq}) y a partir de las condiciones de alimentación (para i_{sd}). Además, este problema es menos crítico pues el principal interés del sistema de control es para usarse en potencias medias o altas y este problema surge en potencias menores al 33% de la nominal. Por otro lado este problema surge en parte por la baja potencia del motor ensayado, pues su corriente de rotor es muy pequeña: en motores más potentes con corrientes de rotor mayores esta limitación es mucho menos apreciable.

Una posible solución a este problema y que además ayudaría a reducir la oscilación de los fasores también en potencias altas consiste en aplicar un filtro de paso alto en el código del programa. Esta idea parece una buena solución a simple vista, pero no se debe olvidar que los filtros producen retraso en los datos, como todos los datos deben ir coordinados se debería aplicar el mismo filtro a todos los datos de entrada para producir el mismo retraso, o, más sencillo, aplicarse a los datos de salida. Finalmente se decidió no aplicar ningún filtro hasta que se pueda comprobar cómo responde el sistema de control ante el filtro, pues no debemos olvidar que cuanto más rápido trabaje, más efectivo será el control, por lo que aplicar un filtro podría ser una medida contraproducente que redujera la estabilidad del sistema de control de forma innecesaria.

8.3 El fasor del rotor y su módulo

Cómo ya se ha podido observar la principal utilidad del fasor del rotor es su ángulo, pues gracias a él se puede colocar el fasor del estator en la posición correspondiente para obtener sus componentes d y q que se usan para establecer el control.

Sin embargo, el módulo del rotor no es totalmente inútil, pues este módulo nos da el eje –q y según la teoría que se puede encontrar en (Martínez Román, 2014) la componente –q, al igual que la q es directamente proporcional a la potencia del motor, de hecho cuando se representan en el mismo sistema de referencia las componentes q y –q deben ser iguales y anularse entre ellas. El único problema es que los módulos de los fasores nunca han estado en el mismo sistema de referencia, pues para hacer el cambio de referencia se necesita saber la relación de transmisión entre el estator y el rotor y para saber esta relación se debe hacer un ensayo adicional.

8.3.1 Ensayo de relación de transformación

Este ensayo es bastante sencillo y rápido de realizar. Consiste en alimentar al estator en forma de triángulo y medir la tensión en los bornes del rotor, estando el rotor desconectado. Si se divide la tensión de alimentación entre la tensión medida en el rotor se obtiene la relación de transformación. Esta relación de transformación no es entre las tensiones de fase reales, si no entre las tensiones de línea, pero eso es precisamente lo que necesitamos, pues en todo momento se ha trabajado con las corrientes de línea.

Se realizó el ensayo a tres tensiones distintas y se calculó su media, aunque en todas las medidas se obtuvo el mismo valor salvo por un par de centésimas, la media de la relación de transformación resultó ser 1,044. Esto significa que para pasar la corriente

del rotor a coordenadas del estator se debe multiplicar la corriente por la inversa de la relación de transformación, que en nuestro caso es 0,958.

8.3.2 Conclusión

Una vez se multiplica la corriente del rotor por 0,958 para cambiar su referencia al estator, ya se puede comparar con la corriente del estator y se observa que aunque las corrientes responden de la misma forma ante los cambios de régimen del rotor, las componentes de corriente q y –q no tienen exactamente el mismo módulo, siendo la componente –q alrededor de 1 amperio menor que la componente q del estator. No está muy clara la razón de este error, pues podría deberse tanto a alguna simplificación de la teoría que no puede aplicarse a un motor de estas características como a un error de cualquiera de los ensayos de calibración realizados, es decir, podría deberse a un error acumulado debido a inexactitudes del ensayo de calibración de posición (6.2.2), del ensayo de calibración de los sensores de corriente (5.2) y del ensayo de relación de transformación (8.3.1), sumados a las inexactitudes causadas por el ranurado del devanado del motor (8.2) o incluso del hecho de no poder realizar todas las medidas pertinentes en el mismo instante (3.1).

En resumen, no se cumple este detalle de la teoría, y la causa de esto podría encontrarse en cualquier aspecto del TFG incluyendo la propia teoría, afortunadamente no es un detalle importante y si de verdad se quisiera eliminar este problema bastaría con ignorar el ensayo de calibración de posición y elegir el ángulo de calibración directamente para que las componentes q y –q se anulen, aunque no es una solución muy ortodoxa y podría traer otros problemas.

9 CONCLUSIONES

Tras terminar este TFG se puede afirmar que pese a tener alguna limitación como la detallada en 8.2, se han alcanzado satisfactoriamente todos los objetivos que se propusieron al inicio. Y por lo tanto, está listo para combinarse con el resto de TFGs del proyecto de control de máquinas de accionamientos eléctricos, aunque deberán realizarse antes los ajustes pertinentes.

Gracias a la realización de este TFG se han podido poner en práctica varios de los conocimientos adquiridos durante la carrera. Principalmente ha servido para alcanzar un conocimiento más profundo del funcionamiento de los motores asíncronos y de la representación fasorial de corrientes, pero también se han desarrollado muchas más habilidades.

Se ha profundizado en la programación orientada a objetos y el lenguaje Java. Se ha programado en dos plataformas muy distintas, un microcontrolador con soporte en Arduino, y un dispositivo Android, además se ha aprendido sobre comunicación Bluetooth y MQTT y se ha aprendido a usar la magnífica librería de achartengine.

También se ha aprendido sobre distintos sensores de medición de corriente y cómo elegir el más apropiado para una aplicación determinada, además de como modificar el circuito para reducir el ruido, añadiendo un filtro y usando cables con mejor aislamiento.

Se ha aprendido a diseñar y depurar algoritmos con la inestimable ayuda que nos ofrece MatLab. Pues nos permite comprobar el correcto funcionamiento de los algoritmos forma rápida y sencilla mediante el uso de scripts y gráficas, de hecho todas las gráficas incluidas en esta memoria se han realizado con MatLab.

También se ha aprendido todo lo necesario para desenvolverse con relativa soltura en un laboratorio de máquinas eléctricas, desde las distintas normas de seguridad que se deben respetar en todo momento hasta la realización de todo tipo de ensayos sobre motores. Sin olvidarse del montaje del prototipo, para el que es necesario tanto soldar las distintas conexiones con estaño como preparar una carcasa contenedora cerrada donde colocarlo para evitar que se pueda acceder a los circuitos de 220V directamente.

En resumen, aunque este TFG se trata de una parte de un proyecto aún más grande, se puede afirmar con total confianza que durante su desarrollo se trataron una gran variedad de temas referentes al grado en ingeniería industrial y, cómo ya se mencionó anteriormente, gracias a este TFG se podrá disponer en el futuro cercano de un sistema de control de corriente vectorial en los laboratorios del Departamento de Ingeniería Eléctrica, y permitirá ofrecer mejores prácticas a los futuros alumnos de las asignaturas de Análisis dinámico y control de accionamientos eléctricos y Ampliación de Máquinas Eléctricas.

10 BIBLIOGRAFÍA

Android. (17 de 06 de 2019). *Android Developers Activity lifecycle*. Obtenido de https://developer.android.com/guide/components/activities/activity-lifecycle

Android. (17 de 06 de 2019). *Android Developers Bluetooth*. Obtenido de https://developer.android.com/guide/topics/connectivity/bluetooth?hl=es-419

Arduino. (21 de 06 de 2019). *Página oficial de Arduino*. Obtenido de https://www.arduino.cc/en/Main/Software

Bourns. (30 de 06 de 2019). *Fabricante*. Obtenido de https://www.bourns.com/home

ddanny. (17 de 06 de 2019). *Github, página de achartengine*. Obtenido de https://github.com/ddanny/achartengine

Espressif Systems. (21 de 06 de 2019). *Github*. Obtenido de https://github.com/espressif/arduino-esp32

Espressif Systems. (30 de 06 de 2019). *Página oficial del fabricante Espressif*. Obtenido de https://www.espressif.com/

Fritzing. (30 de 06 de 2019). *Software de diseño de circuitos*. Obtenido de http://fritzing.org/home/

García García, I., & Gil Gómez, J. A. (2018-2019). *Apuntes de la Asignatura Desarrollo de Aplicaciones para Dispositivos Móviles.*

INNOVA DOMOTICS. (17 de 06 de 2019). *Canal de Youtube*. Obtenido de https://www.youtube.com/watch?v=XRLuEgzIk7A

INNOVA DOMOTICS. (17 de 06 de 2019). *Página oficial de INNOVA DOMOTICS*. Obtenido de http://www.innovadomotics.com/mn-tuto/mn-android/proyectos/23-and-cnm-bt.html

LEM. (28 de 06 de 2019). *Fabricante*. Obtenido de https://www.lem.com/en/ltsr-6np

Magnelab. (28 de 06 de 2019). *Fabricante*. Obtenido de https://www.magnelab.com/

Martínez Román, J. A. (2014). *Apuntes de Ampliación de Equipos e Instalaciones Eléctricas.* Valencia.

MathWorks. (21 de 06 de 2019). *Página oficial de MathWorks*. Obtenido de https://es.mathworks.com/

RC Electronics. (28 de 06 de 2019). *Fabricante*. Obtenido de https://www.rc-electronics-usa.com/buy-current-shunt.html

Serrano Iribarnegaray, L., & Martínez Román, J. A. (2013). *Máquinas Eléctricas.* Valencia: Universitat Politècnica de València.

Siemens. (30 de 06 de 2019). *Fabricante*. Obtenido de https://new.siemens.com/global/en.html

Stackoverflow. (17 de 06 de 2019). *Android: How do bluetooth UUIDs work?* Obtenido de https://stackoverflow.com/questions/13964342/android-how-do-bluetooth-uuids-work

PRESUPUESTO

En el presente documento se recogerá el coste total del proyecto, pues como en todo proyecto de ingeniería resulta de suma importancia conocer la inversión necesaria para llevar a cabo el proyecto, ya que la viabilidad económica es tan importante como la viabilidad técnica. El presupuesto recoge los costes asociados a desarrollar la parte contemplada en este TFG (el sistema de realimentación vectorial de corriente) de un banco de ensayos configurable de accionamientos eléctricos.

El presupuesto se dividirá en distintos capítulos, el diseño del código de cálculo fasorial, el diseño de la aplicación de Android y del sistema de comunicación, la construcción del prototipo y finalmente la realización de la memoria del TFG.

En el presupuesto solo se incluirá el coste derivado de la elaboración del sistema de medida, es decir, no se incluirá el coste de los motores que permiten realizar los ensayos y sobre los que se realizan las medidas.

Capítulo 1 Diseño e implementación del algoritmo de cálculo de los fasores				
Recursos	**Concepto**	**Cantidad**	**Precio unitario**	**Coste**
Materiales	Ordenador personal	2 sem.	2.30 €/sem.	4.60 €
	Arduino IDE	1 u.	0.00 €	0.00 €
	MatLab	1 Lic. est.	0.00 €	0.00 €
Humanos	Estudios previos	20h	17 €/h	340 €
	Diseño e implementación en MatLab	12h	17 €/h	204 €
	Implementación en Arduino	8h	17 €/h	136 €
	Pruebas	20h	17 €/h	340 €
Total Capítulo 1:				1024.60 €

Al poseer una licencia de estudiante, el uso de MatLab es gratuito, si no se dispone de una se deberá adquirir una al precio correspondiente o prescindir de la ayuda de MatLab. El Arduino IDE, sin embargo, es gratuito y de código abierto para todo el mundo.

En los estudios previos se incluyen tanto el estudio de la teoría de los fasores como el aprendizaje mínimo de programación para ser capaz de implementar los algoritmos en Arduino IDE y en MatLab. En las pruebas se incluyen todos los ensayos dedicados a asegurar el correcto funcionamiento del algoritmo. En cuanto a la remuneración, se ha tenido en cuenta el sueldo medio de un Ingeniero Industrial con 5 años de experiencia.

Capítulo 2 Montaje del prototipo				
Recursos	**Concepto**	**Cantidad**	**Precio unitario**	**Coste**
Materiales	Microcontrolador Esp32	1 u.	4.49 €/u.	4.49 €
	Carcasa de plástico	1 u.	3 €/u	3 €
	Cable cobre flexible 3x2.5 mm^2	3 m	0.95 €/m	2.85 €
	Conector Banana 4 mm	9 u.	0.49 €/u	4.41 €
	Cable cobre flexible 0.6 mm	3 m	0.15 €/m	0.45 €
	Sensor efecto Hall LTSR 6-NP	4 u.	12.85 €/u	51.40 €
	Cable apantallado de 0.35mm^2	1 m	0.51 €/m	0.51 €
	Perfboard	1 u.	2 €/u	2 €
	Soldador de estaño 40W	1 día	0.01 €/día	0.01 €
	Estaño para soldar	10 g	0.0985 €/g	0.98 €
	Resistencias de 1/2W	8 u.	0.01 €/u	0.08 €
	Condensador de 212nF	4 u.	0.01 €/u	0.04 €
	Codificador EMS22A50-M25-LD6	1 u.	33.43 €/u	33.43 €
	Conector Arduino 2.54 mm	50 u.	0.00874 €/u	0.44 €
	Canaleta para cableado PVC	0.2 m	1.45 €/m	0.29 €
	Tornillería	4 u.	0.20 €/u	0.80 €
Humanos	Diseño previo	15h	16 €/h	240 €
	Montaje	16h	16 €/h	256 €
	Pruebas	4h	16 €/h	64 €
Total Capítulo 2:				665.18 €

La mayoría de materiales de este apartado son absurdamente baratos si se compran en paquetes grandes, especialmente las resistencias y condensadores, pero se deben aprovechar el resto de los componentes para otros proyectos. Se debe disponer de un soldador, pero al poder encontrarse por menos de 10 euros y tener una vida media de 5 años su amortización es prácticamente despreciable. Cómo ya se explicó en la memoria los precios de casi todos los materiales del prototipo son orientativos y se pueden cambiar por otros de características similares sin mayor problema, los únicos materiales con los que hay que tener cuidado si se deciden cambiar son los sensores hall, el codificador, y especialmente el microcontrolador esp32.

En el diseño previo se incluye el estudio de cuál es el sensor más conveniente para el proyecto y el diseño del filtro. En las pruebas se incluye el tiempo destinado a comprobar el correcto funcionamiento del montaje, incluyendo la ausencia de ruido en las medidas. En este caso la remuneración por horas ha tenido en cuenta el sueldo medio de un ingeniero electrónico.

Capítulo 3 Sistema de comunicación y código de Arduino				
Recursos	**Concepto**	**Cantidad**	**Precio unitario**	**Coste**
Materiales	Ordenador personal	1 sem.	2.3 €/sem.	2.30 €
	Arduino IDE	1 u.	0 €/u	0.00 €
Humanos	Estudios previos	30h	16 €/h	480 €
	Diseño de la aplicación de Arduino	10h	16 €/h	160 €
	Pruebas	5h	16 €/h	80 €
Total Capítulo 3:				722.30 €

En este caso los estudios previos recogen el tiempo dedicado a escoger la mejor forma de comunicación y a idear de qué forma se puede usar para transmitir los datos que deseamos. En la parte de diseño de la aplicación Arduino se incluye implementar todo el código que no es el propio algoritmo de cálculo de los fasores del capítulo 1, esto incluye principalmente el protocolo de comunicación, y la división de las tareas del procesador en sus dos núcleos. La remuneración de las horas se corresponde al sueldo medio de un ingeniero informático con 5 años de experiencia.

Capítulo 4 Desarrollo de la aplicación Android				
Recursos	**Concepto**	**Cantidad**	**Precio unitario**	**Coste**
Materiales	Ordenador personal	1.5 sem.	2.3 €/sem.	3.45 €
	Android Studio	1 u.	0 €/u	0.00 €
	Dispositivo Android	1.5 sem.	1 €/sem	1.50 €
Humanos	Estudios previos	40h	16 €/h	640 €
	Diseño de la aplicación Android	15h	16 €/h	240 €
	Pruebas	5h	16 €/h	80 €
Total Capítulo 4:				964.95 €

Dispositivo Android se refiere a la plataforma sobre la que hacer las pruebas de desarrollo, durante el TFG se usó un teléfono móvil personal, pero en la versión final se debería usar un dispositivo que se pueda usar durante las prácticas, como una tablet.

En la etapa de estudios previos se recoge la adquisición de los conocimientos para ser capaz de desenvolverse en Android Studio satisfactoriamente y el aprendizaje sobre el uso el protocolo de comunicación del capítulo 3 y de una librería que permita representar gráficas, en este caso se usó achartengine. La remuneración de las horas se corresponde a las de un Ingeniero informático con 5 años de experiencia. Para el diseño de la aplicación de Android se debe tener en cuenta de que forma envía los datos el microcontrolador y editarla en el caso de que no sea una forma conveniente para el dispositivo Android.

Capítulo 5 Memoria				
Recursos	**Concepto**	**Cantidad**	**Precio unitario**	**Coste**
Materiales	Ordenador personal	6 sem.	2.30 €/sem.	13.80 €
Humanos	Redacción de la memoria	100h	17 €/h	1700 €
Total Capítulo 5:				1713.80 €

En este Capítulo se ha tomado como referencia el sueldo medio de un ingeniero Industrial con 5 años de experiencia.

Para la amortización del ordenador personal se ha tenido en cuenta un ordenador de 600 € y una vida útil de 5 años. Si cada año tiene 52 semanas, se deben ganar unos 2.30 € por semana para amortizarlo. Para la amortización del dispositivo Android se ha tenido en cuenta un precio de 250 € para una vida útil de también 5 años, resultando en 1 € a la semana para poder amortizarlo.

Resumen	
Concepto	**Coste**
Capítulo 1. Diseño e implementación del algoritmo	1024.60 €
Capítulo 2. Montaje del prototipo	665.18 €
Capítulo 3. Sistema de comunicación y código de Arduino	722.30 €
Capítulo 4. Desarrollo de la aplicación Android	964.95 €
Capítulo 5. Memoria	1713.80 €
Presupuesto de ejecución material	5090.83 €
Gastos generales (12%):	610.90 €
Beneficio industrial (6%):	305.45 €
Presupuesto de ejecución contrata	6007.18 €
IVA (21%):	1261.51 €
Presupuesto base de licitación	7268.69 €

El presupuesto total asciende a la cantidad de 7268.69 € **"SIETE MIL DOSCIENTOS SESENTA Y OCHO EUROS CON SESENTA Y NUEVE CÉNTIMOS DE EURO"**