

UNIVERSIDAD DE MURCIA



Facultad de Informática

Trabajo fin de Grado

Convocatoria de Febrero de 2016

Control de Motores con Intel Galileo

Alumno:

José Francisco Nicolás Robles

Profesor:

Benito Úbeda Miñarro

Resumen

El objetivo principal de este trabajo fin de grado es el estudio de las características y posibilidades de la plataforma Intel Galileo, destacando su uso en el ámbito del control de motores.

La primera parte de este documento trata de enmarcar en el contexto actual los principales componentes de los que es objetivo, comenzando por los diferentes tipos de motores eléctricos existentes en la actualidad, y los controladores asociados a cada uno de ellos. De la misma manera, ponemos en contexto los microcontroladores actuales y estudiamos las características de la placa Intel Galileo y el abanico de posibilidades que nos ofrece. También se realiza una comparativa con las plataformas disponibles de características similares, destacando los puntos fuertes y débiles de la plataforma estudiada frente a las demás.

En el apartado final de esta sección de estado del arte se explican algunos de los mecanismos más conocidos del control de motores, como lo son PWM y PID, y unas pocas de las aplicaciones actuales del control de motores eléctricos, como el giro de placas fotovoltaicas en función de la orientación del sol, o el control de los motores de un dron.

El siguiente paso será construir un escenario práctico en el cual se muestren las posibilidades de la placa de Intel. Este proceso incluye la selección de los componentes, el ensamblaje de los mismos y la configuración del escenario para poder establecer la comunicación entre los componentes. Tras la toma de contacto con el microcontrolador, se construye un programa en lenguaje Arduino usando el entorno de desarrollo homónimo, para que el microcontrolador coordine todos los componentes,

Además, sé ha desarrollado una aplicación móvil en el lenguaje de programación Android, que es la encargada de controlar el motor a través de la red local, así como de mostrar la información de realimentación del sensor de velocidad de giro. El hecho de que la comunicación se realice a través de la red implica la configuración manual del dispositivo de Intel.

Por último se exponen las apreciaciones personales del trabajo realizado en función de los resultados obtenidos. Este trabajo puede usarse como punto de partida para la realización de tareas prácticas mas complejas, en cualquiera de los ámbitos del control de motores eléctricos.

Abstract

The **Intel Galileo** board is an Intel microprocessor compatible with the Arduino development environment. The main objective of this final grade project is the study of the characteristics and possibilities of the Intel Galileo platform, highlighting its use in the field of motor control. In addition to including a broad theoretical framework of each of the components, a practical scenario will be built, composed of a motor, a motor driver, an encoder to measure motor speed, a microcontroller and an Android application.

By definition, an engine is a machine that transforms energy into motion. Likewise, an electric motor is a machine that transforms **electrical** energy into motion. With the invention of the battery by Alessandro Volta in 1800, the generation of a magnetic field from the electric current in 1820, and the electromagnet in 1825, the bases for the construction of electric engines were established. However, the DC motor was created from the development of power generators, also known as dynamometers. The foundations were laid by William Ritchie and Hippolyte Pixii in 1832 with the invention of the switch and, more importantly thanks to Werner Siemens in 1856 with the double-T anchor.

Currently the most known DC motors are as follows. Brushed motors, whose most remarkable feature lies in its simplicity, since they only consist of a rotor, a switch, brushes, a shaft and a magnet. Brushless motors or BLDC are a little less simple but have a number of notable features that make them superior of their peers with brushes, including higher speed, torque and efficiency. Stepper motors are a specific type of brushless motors, but whose characteristics make them very special. Some of the most notable are the open loop positioning, which allow them to know the exact position of the shaft without the need of feedback information and the conservation of torsion force, even with the shaft immobile. Finally, the servo motors are a type of motors that are characterized by their angular precision, in other words, only rotates in a precise way and in a certain angle.

On the other hand, AC motors have the advantage that the current that feeds them can travel great distances and provides them with a great power, and that makes this type of motors ideal for industrial and household applications. There are two types of AC motors, Synchronous AC motors, which are named like this because the rotor speed of this motor is the same as that of the rotating magnetic field, which causes them to maintain a constant speed regardless of the load, and asynchronous motors or induction motors, which always move at slower speed than the synchronization speed, although its operating principle is simpler than the previous ones.

Since the study case is the Intel Galileo board, I've examined the platforms with the most similar features, offered by the most popular companies. The boards we will discuss are Raspberri Pi, from Raspberri Pi Foundation, BeagleBone Black, from Texas Instruments, Arduino Mega 2560, from Arduino and Intel Galileo, from Intel Corporation.

The results shows the strengths of each platform. Both Beaglebone Black and Raspberry Pi

have a powerful processor and support audio and video output, while the Arduino Mega and the Intel board have greater compatibility with all Arduino accessories. As for input/output pins the clear winner is the Arduino Mega 2650, followed very closely by the BBB plate. Intel Galileo has, for example, certain advantages that other platforms do not have, such as a real-time clock and a built-in Wi-Fi card slot. Finally, the most economical platform is the Arduino board, although in terms of features/price, the clear winner is the Raspberry Pi.

Therefore, depending on the task that we are going to perform we will use one board or another. In our case, the control of motors can be done with any of these 4 platforms without any problem. But Intel Galileo being the platform chosen from the beginning, we will try to get the most out of it.

We also examined various motor control techniques using electrical circuits. The first of these is Pulse Width Modulation (PWM) a powerful technique for controlling analog circuits with the digital outputs of a microprocessor. This technique is used in a wide variety of applications, ranging from measurement and communications to power control and conversion. In this case I have tested the PWM to control a BLDC motor, but this case is not included here since in the practical case a stepper motor is used instead.

On the other hand, the Proportional-Integral-Derivative Control (PID) technique is the most common control algorithm. Is used in industry and has been universally accepted in industrial control. The popularity of PID controllers can be attributed to their robust performance over a wide range of operating conditions, as well as to their functional simplicity, which allows engineers to operate with them in a simple and straightforward manner.

Engine control is very important in today's society. Motor control systems are used throughout the industrial, automotive, medical, aviation and defense sectors, and consume 45 % of the world's energy. Examples of engine control include factory automation processes, assembly and packaging, elevation control, robotics, appliances, flight control and many more.

Once finished the theoretical background I begin with the choice of the components that would form part of the scenario to be assembled. These components are a stepper motor, which provides a very precise control of the position of the motor at all times, a driver whose main functions are to operate the stepper, while increasing its step precision and protect it against current spikes, an encoder, whose function is measuring the rotation speed of the motor, and the controller Intel Galileo. The motor connects to the driver directly, and the driver to the microcontroller. The encoder is the only component that needs an external circuitry for its correct operation.

As for the configuration of the Intel Galileo board, although it comes with pre-installed Linux distribution, it should be noted that if we want the program loaded in the memory to be persistent after shutdown we must boot from the SD card. Otherwise the sketch loaded in the memory will be erased. In this case we have installed a Yocto distribution of Linux since it has more libraries and utilities than the basic one. There are also versions of Windows available to install on this platform. The installation process is very easy to understand, simply save

the image of the chosen operating system on a micro SD card with the help of an installation program, and insert the card into the Intel device. On the next boot, the microcontroller will boot from the micro SD card.

Last but not least, we need to install an Integrated Development Environment. This will depend on the chosen programming language. In this case the available programming languages are Arduino, JavaScript + Node.js, C++ and Java. After studying the different possibilities, the chosen one is Arduino. One of the reasons to make my choice has been the familiarity with the environment, since I had worked on the subject of Programming Embedded Systems. Although there are other languages I am also familiar with, especially with Java, the fact that most of the literature and documents of this type of devices are written using the language of Arduino is the main reason of my choice.

Once I have installed the Arduino IDE and downloaded the Intel SDK for that platform, we can start programming for the Intel Galileo in the same way we would with any other Arduino device. The structure of an Arduino language program consists of three parts, the declaration of the libraries and global variables, the setup section, where you declare and initialize local variables, and the main execution loop, where the repeatable code goes, in that order.

The control of the motor driver is carried out by means of 3 cables, which are responsible for the start and stop signal, the direction of rotation and the clock signal. One of the bigger disadvantages of Intel Galileo is that the read/write frequency of all its pins(except 2 and 3) is only 230Hz. In order to enable a faster writing mode(up to 2.93Mhz of write frequency) we will have to use a special function.

The encoder is responsible for sending the motor rotation signals to the microcontroller. The encoder warning mechanism will be the interruptions mechanism. This is because the pin reading is 230Hz and would slow down the execution of the main loop. On the other hand, the only pins able to activate interrupts are 2 and 3. Considering that pin 3 is already occupied by the driver, we only have pin 2 to perform this task.

The main execution loop consists of 3 parts, a section to read and manage commands received from the mobile application, if any. A holding loop to reduce the speed of the main loop, and sending the signal for the motor to advance one step.

We also have auxiliary functions whose functionalities are to stop and start the stepper motor progressively, change the direction of rotation(clockwise or counter clockwise), and set a fixed speed.

On the side of the mobile application, the platform selected to develop a mobile application is Android. Android currently dominates the market with an overwhelming 86.8 % of the world devices in November 2016. Also I am already familiar with this technology thanks to the subject of Mobile Computing. Also the only terminal I have at the moment is an Android terminal.

The code of this application is quite simple, since the control complexity resides in the microcontroller. The purpose of the application is to display a simple interface, send the commands to the microcontroller, and read the data back to show the speed read by the sensor. The development environment used has been the Eclipse platform.

The Android application interface is defined using XML markup language. Although some IDE's like Eclipse have graphic editors, it is easier for me to edit the XML code directly because it offers more control. The elements organization has been done by rows and columns, with a table(TableLayout). The size of the components adapts to the size of the screen that contains them regardless of the orientation, vertical or horizontal, since the design is responsive.

The main execution thread is responsible for managing the application and implements the Activity interface. An Activity is a component of the application that contains the screen that we will use to perform the actions.

Through the completion of this work I have learned to configure a complete scenario focused on the control of motors wirelessly, from the assembly to the programming of each of these pieces of hardware, and their configuration and communication through the network.

Among the possible extensions of work are the practical application of the use and control of motors. One of the possible works is the height control and orientation of antennas with several motors for the measurement of interferences within a measuring chamber. For the accomplishment of such work it would be necessary an arduino shield that expanded the number of fast writing pins.

Another future path would be the creation of a mobile application on other platforms, such as Windows Phone, IOs or the creation of a web interface. Finally, we could also add wireless connectivity by inserting and setting a PCI-Express(Wifi) card, or an arduino shield that provides 3G connectivity.

Índice

1. Introducción	8
2. Estado del Arte	11
2.1. Motores Eléctricos	11
2.1.1. Motores de Corriente Continua	11
2.1.2. Motores de Corriente Alterna	21
2.2. Microcontroladores	24
2.2.1. Raspberri Pi	24
2.2.2. Beaglebone black	25
2.2.3. Arduino Mega 2560	25
2.2.4. Intel Galileo	26
2.2.5. Comparativa entre las diferentes plataformas	28
2.2.6. Ventajas e inconvenientes de las diferentes plataformas	28
2.3. Control de motores eléctricos	29
2.3.1. Pulse Width Modulation - PWM	29
2.3.2. Controlador PID	31
2.4. Aplicaciones actuales	38
2.4.1. Electrodomésticos	38
2.4.2. Sistema de Seguimiento Solar	39
2.4.3. Robótica	41
3. Análisis de Objetivos	45
4. Diseño y resolución del trabajo realizado	47

4.1.	Toma de contacto con Intel Galileo	49
4.2.	Programación en Intel Galileo	51
4.2.1.	Configuración y manejo del motor	51
4.2.2.	Configuración del encoder	52
4.2.3.	Configuración de la red	53
4.2.4.	Explicación del código Arduino	54
4.3.	Aplicación Android	59
4.3.1.	Interfaz de la aplicación	59
4.3.2.	Hilo principal de ejecución	61
4.3.3.	Cliente y Servidor UDP	62
4.3.4.	Funciones auxiliares	66
5.	Conclusiones y vias futuras	68
	Referencias	69
	Anexos	71
A.	Código	71
A.1.	Código Completo Arduino	71
A.2.	Código Completo Android	74

1. Introducción

Por definición, un **motor** es una máquina que transforma una energía en movimiento. Así mismo, un **motor eléctrico** es una máquina que transforma la energía **eléctrica** en movimiento.

Con la invención de la batería(Allessandro Volta, 1800), la generación de un campo magnético a partir de la corriente eléctrica(Hans Christian Oersted, 1820) y el electroimán(William Sturgeon, 1825) se establecieron las bases para la construcción de motores eléctricos. El primer dispositivo rotatorio impulsado por electromagnetismo fue construido por el inglés Peter Barlow en 1822(Rueda de Barlow), aunque no fué hasta 1834 cuando el prusiano de habla alemana Moritz Jacobi creó el primer motor eléctrico rotatorio real, que realmente desarrolló una potencia mecánica de salida notable.

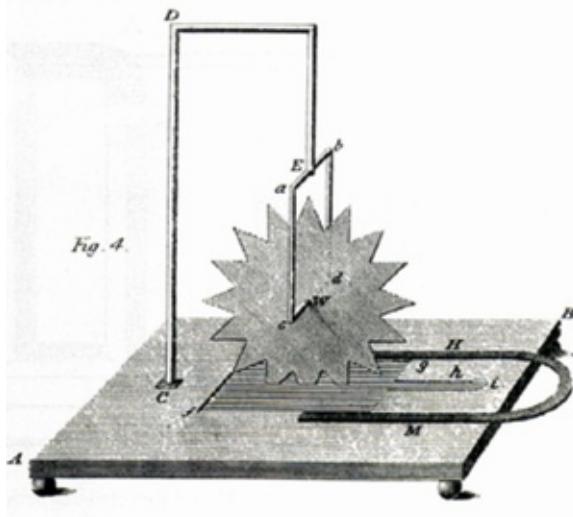


Figura 1: Rueda de Barlow - Peter Barlow

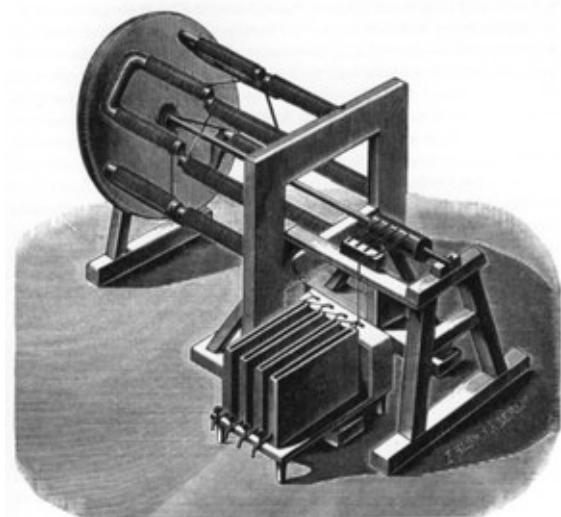


Figura 2: Primer motor eléctrico real

Sin embargo, el motor de corriente continua no fué creado a partir de estos motores, si no más bien del desarrollo de generadores de potencia, también conocidos como dinamómetros. Los fundamentos fueron puestos por William Ritchie e Hippolyte Pixii en 1832 con la invención del conmutador y, lo más importante, por Werner Siemens en 1856 con el ancla de doble-T y por su ingeniero jefe, Friedrich Hefner-Alteneck, en 1872 con la armadura de tambor.[\[1\]](#)

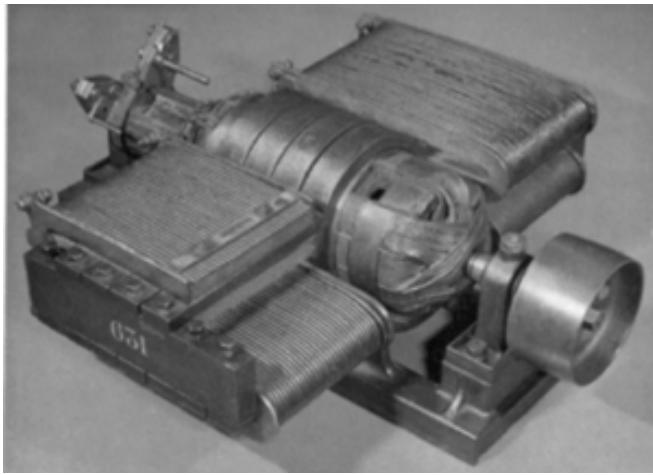


Figura 3: Armadura de tambor - Siemens

Siguiendo esta continua línea de evolución de los motores se hizo necesario un control más preciso de los mismos. Así nacieron los controladores de motores eléctricos. Un **controlador de motores** es un dispositivo(o conjunto de dispositivos) que sirven para gobernar, de una manera predeterminada, la operación de un motor de corriente continua o alterna al que están conectados.

Existen cuatro categorías principales de control de motor a considerar. Cada una de estas tiene a su vez varias subcategorías. Algunas de las piezas de un controlador de motores pueden realizar múltiples funciones:

- **Arranque del motor:** Desconexión de medios, arranque a través de la línea, arranque de voltaje reducido.
- **Protección del motor:** Protección contra la sobretensión, protección contra la sobrecarga, otras protecciones(voltaje, fase, etc.), protección ambiental.
- **Parada del motor:** Parada de cabotaje, frenado eléctrico, frenado mecánico.
- **Control operacional del motor:** Control de velocidad, cambio de dirección, control de secuencia, motor jogging¹.

Con la invención del **circuito integrado** a finales de los años cincuenta y su rápidos avances tecnológicos se hizo posible que todos estas funcionalidades del control de motores se pudieran llevar a cabo por dispositivos cada vez más pequeños. Un **circuito integrado o microchip** es

¹El *motor jogging* o trote de motor se refiere a la repetición de intervalos de arranque y parada para llevar a cabo un determinado movimiento, como mover una grúa hasta un determinado lugar.

un grupo de pequeños circuitos electrónicos que trabajan juntos en una pieza muy pequeña de material sólido(por ejemplo, silicio).

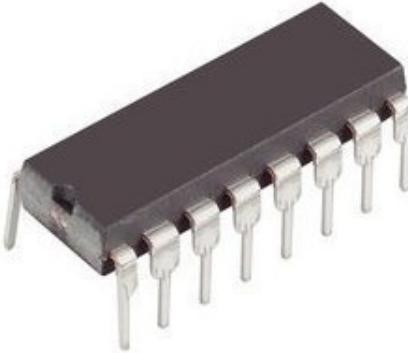


Figura 4: Ejemplo de circuito integrado - Memoria RAM 64K

Los circuitos integrados continuaron mejorando durante las décadas venideras, cada vez eran más pequeños, potentes y asequibles. Y esto nos lleva hasta el punto en el que nos encontramos hoy en día, en el que disponemos de microcontroladores de uso general, ordenadores funcionales en miniatura, capaces de controlar varios dispositivos a la misma vez, y a un precio asequible por cualquier ciudadano de a pie.

Motivación

El objetivo de este trabajo trata de explotar al máximo las posibilidades de la placa Intel Galileo en el ámbito del control de motores principalmente, además de la comunicación con otros dispositivos a través de Ethernet o Wifi, así como el desarrollo de una aplicación móvil en lenguaje de programación Android.

Por otro lado, el principal motivo por el cual elegí desarrollar este tipo de trabajo fué el escaso conocimiento que se ofrece del apartado hardware durante la carrera(sólo en una asignatura semestral) y me parece un ámbito muy interesante y necesario. Además con la reciente proliferación del llamado *internet de las cosas*, hace que estos conocimientos sean muy valorados.

Esta memoria se compone de 4 apartados. En primer lugar estudiaremos el estado actual de las diferentes tecnologías usadas, seguido de un análisis de los objetivos que se desean conseguir durante la realización de este trabajo. En tercer lugar se llevará a cabo el diseño y construcción de un escenario práctico en el cual se muestren las diferentes capacidades de control de motores, así como el desarrollo de una aplicación móvil y las comunicaciones entre los diferentes dispositivos. Por último, en el cuarto apartado se expondrán las conclusiones y las posibles vías futuras de este trabajo.

2. Estado del Arte

Con el objetivo de tener una visión más global analizaremos la tecnología actual de microcontroladores y motores disponibles en el mercado. Asimismo, veremos también varios mecanismos de control de motores eléctricos y algunas de sus aplicaciones más importantes en entornos de producción reales.

2.1. Motores Eléctricos

Como hemos dicho antes, un **motor eléctrico** es una máquina que transforma la energía **eléctrica** en movimiento. La idea básica del funcionamiento de un motor eléctrico es muy sencilla: se pone electricidad en uno de los extremos y un eje rota en el extremo contrario, proporcionando la energía necesaria para accionar una máquina de algún tipo.

La parte interesante reside en el intercambio de energías. Cuando una corriente eléctrica pasa a través de un cable, esta crea un campo magnético alrededor del mismo. Si situamos el cable cerca de un imán, el campo magnético temporal generado por la corriente interaccionará con el del imán. También sabemos que dos imanes se atraen o se repelen dependiendo de signo de sus polos. De esta manera, el magnetismo temporal causado por el cable atrae o repele el magnetismo permanente del imán, y esto es lo que causa el giro del cable.[\[2\]](#)

Estas son las bases de los motores eléctricos, los detalles sobre cómo se produce el movimiento dentro de estas máquinas dependerá del tipo de motor. Asimismo, a cada tipo de motor le corresponderá un tipo de controlador o *driver*, adaptado a sus necesidades y que también examinaremos en cada punto.

2.1.1. Motores de Corriente Continua

Los motores de corriente continua se adaptan a aplicaciones que requieran operaciones precisas y estables, ya que permiten un mayor control de la velocidad y la fuerza de torsión. Dentro de los motores de corriente continua tenemos varias subcategorías.

2.1.1.1 Motor con escobillas o Brushed

Desde finales del siglo XIX, los motores *brushed* son uno de los tipos más simples de motores. Sin contar la fuente de alimentación necesaria para la operación, un motor brushed típico consta de un rotor, un commutador, escobillas, un eje, y un imán.

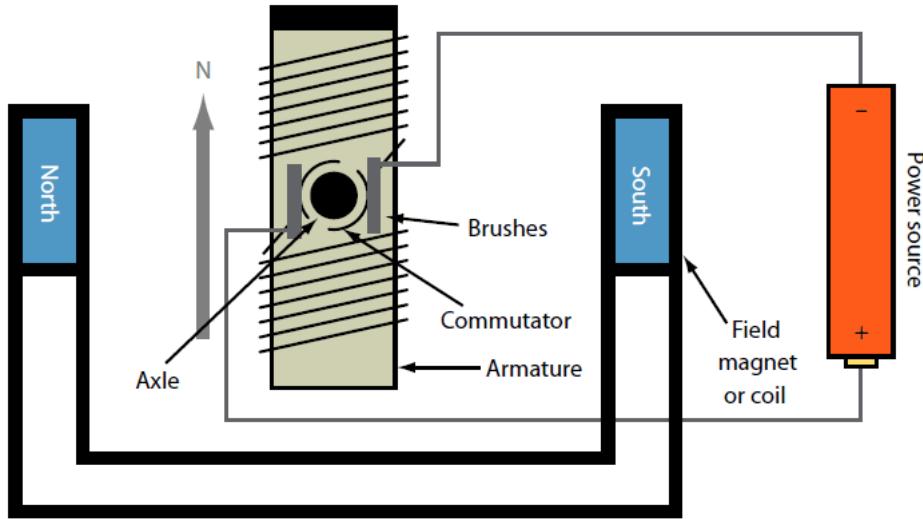


Figura 5: Esquema Motor Brushed

El rotor es un electroimán, y el imán de campo es un imán permanente. El conmutador es un dispositivo de anillo partido, que envuelve al eje, y que contacta físicamente con las escobillas, las cuales están conectadas a los polos de la fuente de alimentación.

Las escobillas cargan el conmutador con una polaridad inversa a la del imán permanente, causando que el rotor gire. La dirección de rotación puede ser invertida fácilmente invirtiendo la polaridad de las escobillas, es decir, invirtiendo la conexión de los cables de la batería. La velocidad se controla simplemente aumentando el voltaje, a mayor voltaje, mayor velocidad.^[3]

Como podemos observar, la construcción de este tipo de motores es tan sencilla que normalmente no necesitan de un controlador, pero en el caso de necesitar un **controlador de motores con escobillas**, lo normal es que sea muy simple y barato. Pero si el motor va a ser parte de una aplicación embebida será necesario un controlador y algo de lógica de control.

En la siguiente figura podemos observar un diagrama de bloques de un semiconductor LB1938FA de canal único, un controlador de motores DC reversible que proporciona salidas de baja saturación para el uso de aplicaciones de bajo voltaje. El motor está controlado por un puente en H el cual está protegido por diodos *anti-chispazos*, ya que se trata de un motor de escobillas. Los circuitos lógicos del bloque de control determinan la velocidad y dirección según lo dictado por la CPU. El LB1938FA proporciona funcionamiento en ambos sentidos, freno y modo de espera controlados por dos señales de entrada. Está diseñado para su uso en ordenadores portátiles, cámaras digitales, móviles y otros dispositivos portátiles.^[4]

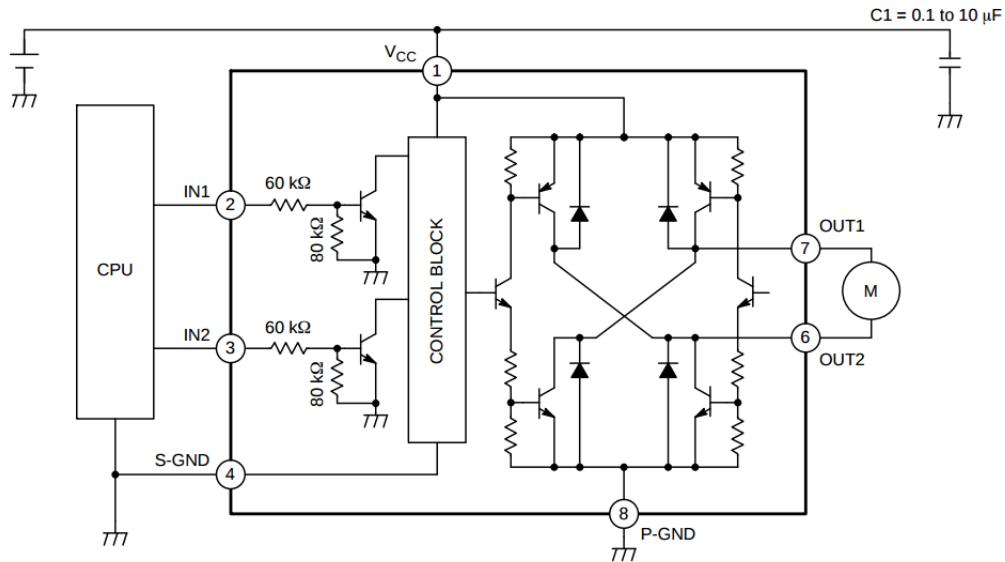


Figura 6: Esquema de control de un motor Brushed DC

2.1.1.2 Motor sin escobillas o Brushless

Los motores de corriente continua sin escobillas(BLDC) son unos de los tipos de motor que han ido ganando popularidad rápidamente. Este tipo de motores se usan en industrias tales como electrodomésticos, automoción, medicina, industria aeroespacial, automatización industrial, equipos e instrumentación. Como su propio nombre indica, los motores Brushless carecen de escobillas para realizar la conmutación; en su lugar, realizan esta conmutación de manera electrónica. Los motores BLDC tienen muchas ventajas sobre los motores con escobillas o de inducción. Algunas de ellas son:

- Mejor proporción velocidad/fuerza de torsión.
- Alta respuesta dinámica.
- Alta eficiencia.
- Vida operacional duradera.
- Funcionamiento silencioso.
- Rangos de velocidad más altos.

Además, la proporción entre la fuerza de torsión y el tamaño del motor es mayor, haciéndolos muy útiles en aplicaciones donde el espacio y el peso son factores críticos.

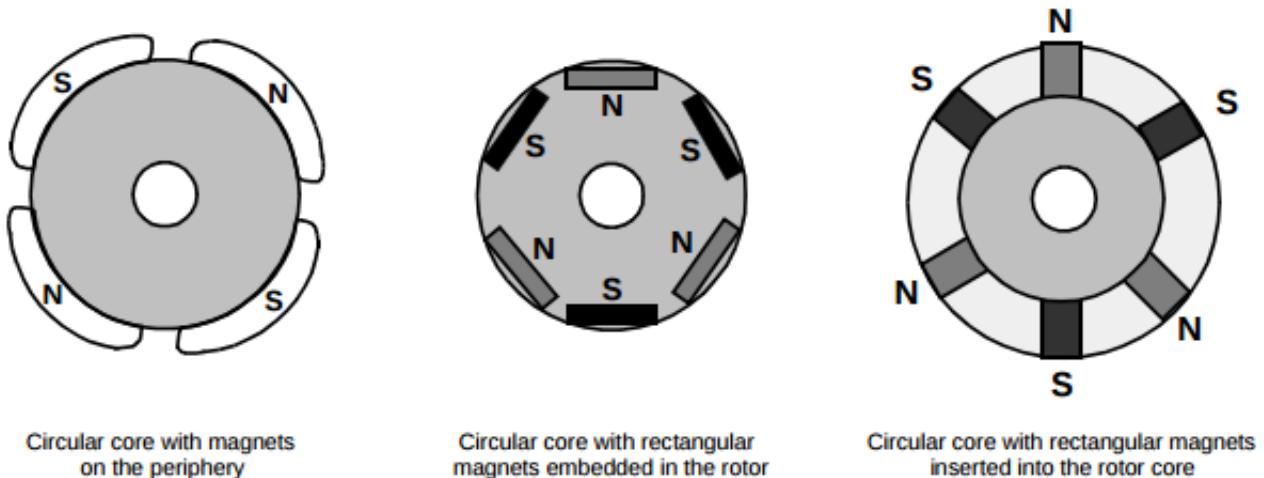


Figura 7: Sección frontal de imanes del rotor

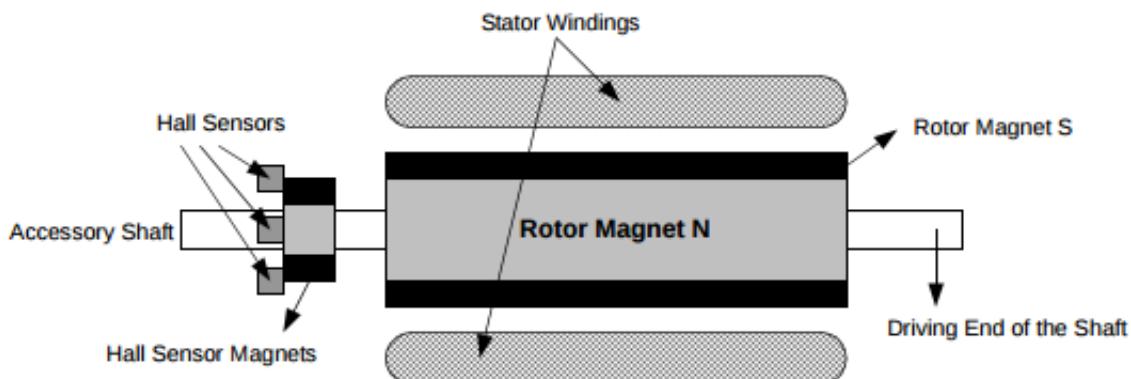


Figura 8: Sección transversal de un motor BLDC

En cuanto al modo de funcionamiento de este tipo de motores, cada secuencia de commutacion tiene uno de los estatores cargado con energía positiva(corriente entrando), el segundo estator tiene carga negativa(corriente saliendo), mientras que el tercer estator permanece sin carga. La torsión se produce debido a la interacción entre los campos generados por las bobinas del estator y los imanes permanentes. Idealmente, el pico de fuerza de rotación se produce cuando estos dos campos se encuentran a 90° y va decreciendo a medida en que se mueven juntos. Para mantener el motor en movimiento, el campo magnético producido por los estatores va cambiando, a la

misma vez que el rotor se mueve, para sincronizarse con su campo magnético. Esto se conoce como **Commutación en seis pasos** y define la secuencia de carga de los estatores.^[5]

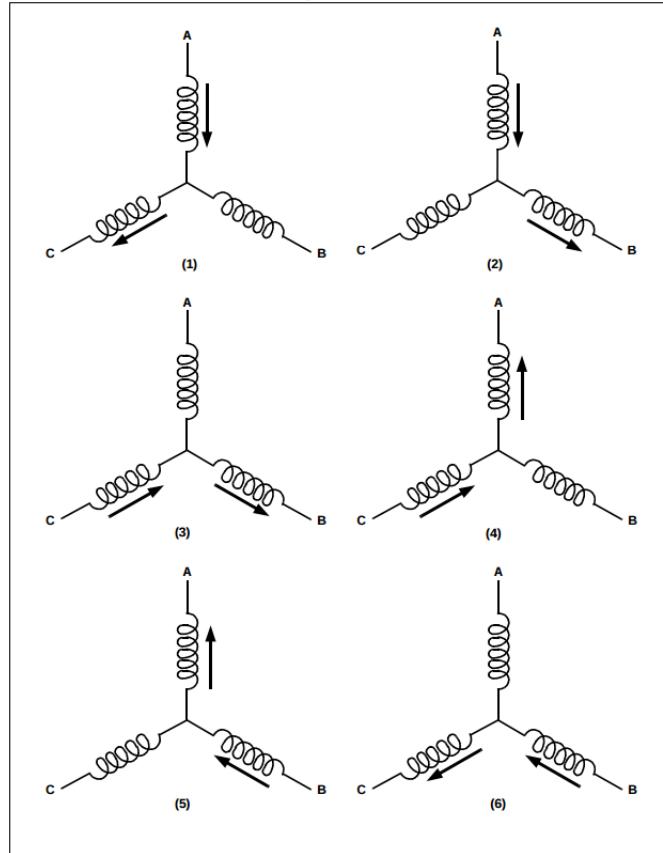


Figura 9: Secuencia de excitación de los estatores con respecto del sensor de entrada

Por último, vemos las características de un **controlador de motores BLDC**. Aunque un microcontrolador de 8 bits ligado a un inversor trifásico es un buen comienzo, no es suficiente para un sistema completo de control de motor BLDC. Para completar el trabajo se necesita una fuente de alimentación regulada para manejar el IGBT² o MOSFET³. Afortunadamente, el trabajo se hace más fácil porque varios grandes proveedores de semiconductores han diseñado chips de control integrados para esta tarea.

Estos dispositivos comprenden un convertido reductor (para alimentar el microcontrolador y otros requisitos de alimentación del sistema), control del controlador de puerta y manejo de

²El transistor bipolar de puerta aislada IGBT (del inglés Insulated Gate Bipolar Transistor) es un dispositivo semiconductor que generalmente se aplica como interruptor controlado en circuitos de electrónica de potencia.

³El transistor de efecto de campo metal-óxido-semiconductor o MOSFET (en inglés Metal-oxide-semiconductor Field-effect transistor) es un transistor utilizado para amplificar o conmutar señales electrónicas.

errores, más algo de lógica de control y tiempo. El pre-driver trifásico DRV8301 de Texas Instruments es un buen ejemplo:

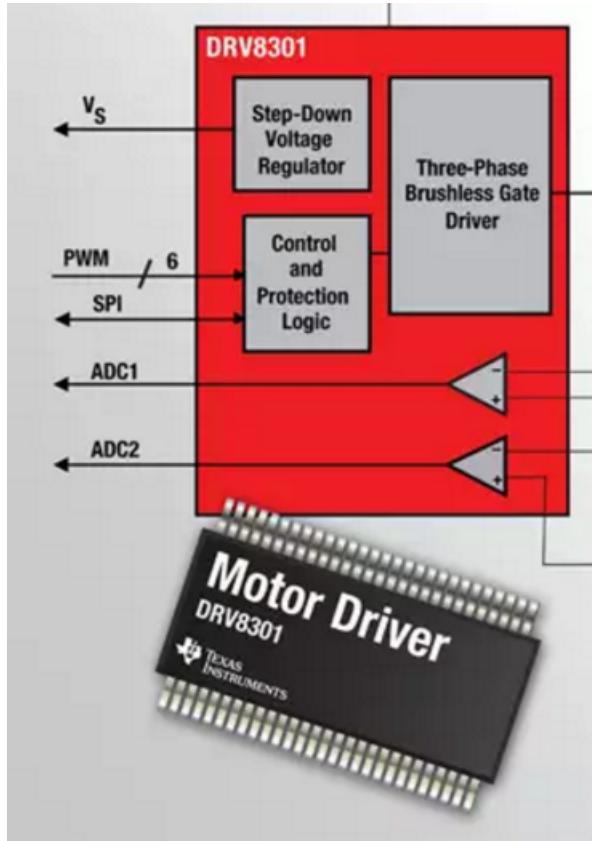


Figura 10: Driver trifásico DRV8301 de Texas Instruments

Este Pre-driver soporta hasta 2.3A de caida y 1.7A de capacidad de pico de corriente, y requiere solo una fuente de alimentación con un voltaje de entrada de entre 8 y 60V. El dispositivo usa negociación automática cuando los transistores IGBT o MOSFET están cambiando para evitar picos de corriente.[6]

2.1.1.3 Motor Paso a Paso o Stepper

Por otro lado, tenemos los motores paso a paso o Stepper que aunque pertenecen a la categoría de motores Brushless, merecen especial mención ya que tienen características especiales. Estos motores llenan un nicho único en el mundo del control de motores y se suelen usar comúnmente en aplicaciones de medida y control. Algunas de las aplicaciones incluyen impresoras de inyección de tinta, máquinas de control numérico, y bombas volumétricas. Varias

características comunes a todos los motores paso a paso hacen que se adapten perfectamente a este tipo de aplicaciones. Esas características son las siguientes:

- **Brushless** - Como ya hemos mencionado antes, este tipo de motores pertenece a la categoría Brushless. El conmutador y los cojinetes de los motores convencionales son unos de los puntos más propensos a fallos, y además crean arcos electricos que son poco deseables o incluso peligrosos en algunos escenarios.
- **Independiente de la carga** - Los motores paso a paso giraran a la misma velocidad, independientemente de la carga, siempre y cuando la carga no exceda el rango de torsión del motor.
- **Posicionamiento de lazo abierto** - Los motores paso a paso se mueven en pasos cuantificados. Siempre que el motor se mueva según su especificación de fuerza de torsión, la posición del eje será conocida en todo momento, sin que sea necesario un mecanismo de realimentación.
- **Mantiene la fuerza de torsión** - Los motores paso a paso son capaces de mantener el eje inmóvil.
- **Respuesta excelente al inicio, parada e inversión del giro.**

Dentro de los motores paso a paso tambien tenemos diferentes tipos, entre los cuales se encuentran los Motores de Reluctancia Variable, los Motores Unipolares, los Motores Bipolares, los Motores Bifilares y los Motores Híbridos. Dependiendo del tipo de uso que le vayamos a dar, la fuerza de torsión necesaria en el sistema, la complejidad del controlador o las características físicas del motor, elegiremos un tipo de motor paso a paso u otro.[\[7\]](#)

El **controlador de motor Stepper** es un poco más complejo que los controladores de motores de corriente continua normales. Los motores paso a paso necesitan un controlador para energizar las fases en una secuencia oportuna para hacer girar el motor. Como hay varios tipos de motores paso a paso, explicaremos solo uno de ellos, en este caso, los motores paso a paso de reluctancia variable. Este tipo de motores tienen multiples bobinados, normalmente entre 3 y 5, los cuales se sitúan unidos en un extremo. Los bobinados se activan de uno en uno en una determinada secuencia para hacer girar el motor. La siguiente figura muestra un circuito básico para accionar un motor de reluctancia variable.

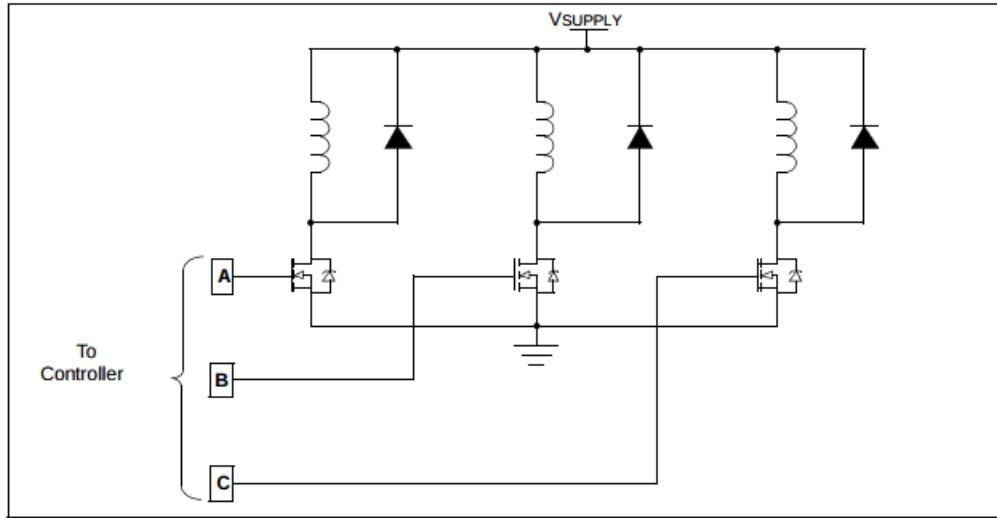


Figura 11: Circuito de control de un motor de reluctancia variable

Se pueden observar claramente los diodos junto a los bobinados. Como con todas las cargas inductivas, cuando se conecta un voltaje a través de una bobina, la corriente empieza a aumentar. Cuando el transistor MOSFET de conmutación para el bobinado se apaga, se produce un pico de voltaje que puede dañar dicho transistor. El diodo protege el MOSFET del pico de voltaje(asumiendo que el diodo sea del tamaño adecuado).

2.1.1.4 Servomotor

Un **servomotor** no es otra cosa que un simple motor eléctrico, controlado por un servomecanismo. Si el motor como dispositivo controlado, asociado con el servomecanismo es un motor de corriente continua, se conoce como servomotor DC. Si el motor controlado es accionado por corriente alterna, se denomina servomotor AC.[8]

De esta manera, un **servomecanismo** es un sistema formado por 3 componentes básicos: un dispositivo controlado, un sensor de salida y un sistema de realimentación. Esto es, un sistema automático de control de circuito cerrado⁴.

⁴Esta parte la veremos con más detenimiento más adelante, en el apartado controlador PID

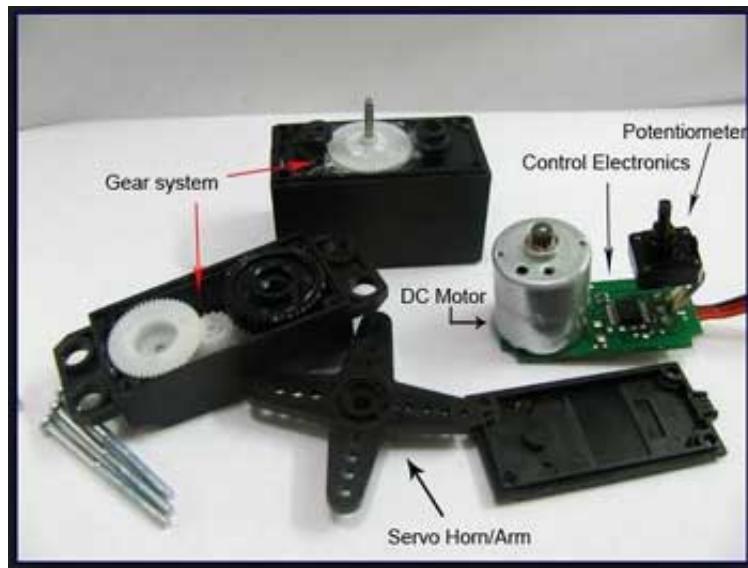


Figura 12: Componentes de un servomotor

Hay algunos tipos de aplicación de motor eléctrico donde se requiere que el motor gire solo en un cierto ángulo, y no de forma continua durante un largo período de tiempo. Aquí es donde entran en juego los motores servo. La razón principal por la que usar un servo es que provee precisión angular, es decir, solo rotará tanto como queramos y parará hasta la siguiente señal. Esto es diferente a un motor eléctrico normal, que comienza a girar cuando se le aplica potencia y la rotación continúa hasta que desconectamos la alimentación. No podemos controlar el progreso rotacional del motor eléctrico, solo podemos controlar la velocidad de rotación, encenderlo y apagarlo.

Como todos sabemos, un pequeño motor de corriente continua girará a mucha velocidad pero el torque generado no será suficiente ni siquiera para mover una carga pequeña. Por este motivo, dentro de un servomecanismo hay un sistema de engranajes. El mecanismo de engranajes recibe una gran velocidad de entrada del motor y la transforma en una velocidad mucho más lenta que la original, pero más práctica y ampliamente aplicable.

En cuanto al **controlador de servomotor**, consiste en un controlador, el servomotor y una fuente de alimentación. Un único controlador de servos puede usarse para controlar uno o varios servomotores, como en el caso de un avión RC, que utiliza muchos servos. Un servomotor tiene 3 conexiones, el cable de la señal de posición(pulsos PWM⁵), el de Vcc(de la fuente de alimentación) y la toma de tierra.

La posición angular del servomotor se controla aplicando pulsos PWM de una anchura específica. La duración de pulso varía entre 0.5ms para una rotación de 0 grados y 2.2ms para una rotación

⁵El mecanismo de control PWM está explicado de manera exhaustiva en una sección posterior.

de 180 grados. Los impulsos deben producirse a frecuencias de entre 50Hz y 60Hz.

Para generar la forma de onda PWM, como se muestra en la figura siguiente, se puede usar el módulo PWM interno del microcontrolador o los temporizadores. El uso del bloque PWM es más flexible, ya que la mayoría de las familias de microcontroladores diseñan los bloques para satisfacer las necesidades de aplicación como el servo motor. Para diferentes anchos de pulso PWM, necesitamos programar los registros internos en consecuencia.

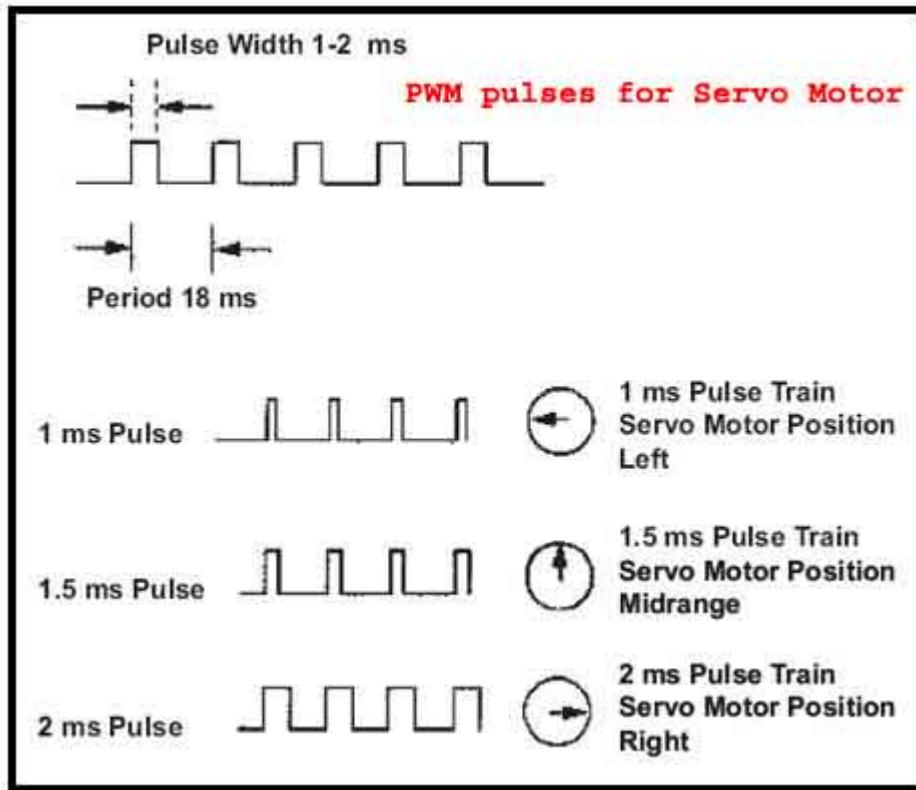


Figura 13: Pulsos PWM de un servomotor

Ahora, también tenemos que decirle al microcontrolador cuánto tiene que girar. Para ello, podemos usar un simple potenciómetro y un ADC (conversor analógico-digital) para obtener el ángulo, o para aplicaciones más complejas se puede usar un acelerómetro. Por último, por cada servomotor que se quiera controlar se necesitará un canal de PWM.

2.1.2. Motores de Corriente Alterna

Por otro lado tenemos los motores de corriente alterna. El hecho de que el tipo de corriente que los alimenta puede recorrer grandes distancias y los provee de una gran potencia, hace que este tipo de motores sean ideales para aplicaciones industriales y del hogar.

Los motores de corriente continua se pueden dividir en dos categorías principales, motores síncronos y motores asíncronos. Los motores asíncronos tambien se conocen comunmente como motores de inducción.

2.1.2.1 Motores Síncronos

Los motores síncronos se llaman así porque la velocidad del rotor de este motor es igual que la del campo magnético giratorio. Se trata básicamente de un motor de velocidad fija, ya que solo tiene una velocidad, que es la velocidad de sincronismo y por lo tanto no hay velocidad intermedia. La velocidad de sincronismo viene dada por la siguiente fórmula:

$$N_s = \frac{120f}{p}$$

Donde f es la frecuencia suministrada y p es el número de polos.

Normalmente su construcción es similar al de un motor de inducción de 3 fases, excepto porque el rotor se alimenta con corriente continua, cuya razón explicamos más adelante. En la siguiente figura se observa claramente el diseño del motor, con un estator con alimentación trifásica, y un rotor con corriente continua:

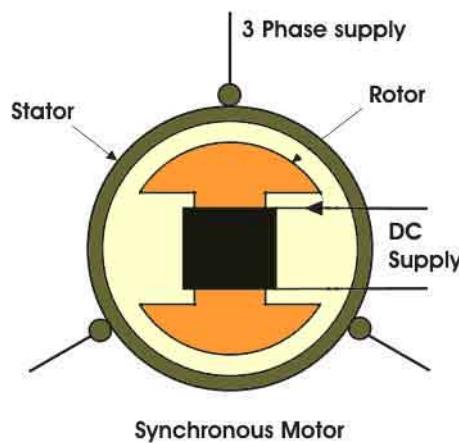


Figura 14: Sección frontal de un motor síncrono

Características principales de los motores síncronos:

- Los motores asíncronos no pueden arrancar por si mismos. Requieren de un medio externo para llevarlos hasta una velocidad cercana a la de sincronía para poder sincronizarse.
- La velocidad de funcionamiento está en sincronía con la frecuencia suministrada y por lo tanto tienen una velocidad constante independientemente de la carga.
- Este motor tiene las características únicas para operar bajo cualquier factor de potencia. Esto hace que se utilicen en la mejora del factor de potencia.

En cuanto al principio de operación, estos motores son una máquina doblemente excitada, ya que se le proporcionan dos entradas eléctricas. El bobinado del estator de 3 fases transporta corriente trifásica, produciendo un flujo magnético rotatorio de 3 fases. El rotor con corriente continua también produce un flujo constante. Teniendo en cuenta una frecuencia de 50Hz, de la relación anterior podemos observar que el flujo de rotación de 3 fases rota a 3000 vueltas/minuto o 50 vueltas/segundo.

En un instante determinado los polos del rotor y el estator deben tener la misma polaridad(N-N o S-S), haciendo que la fuerza de repulsión mueva en el rotor y en el siguiente instante se convierta la polaridad en N-S, causando fuerza de atracción. Debido a las fuerzas de atracción o repulsión que hay durante la posición de punto muerto el rotor es incapaz de rotar en ninguna dirección. Por lo tanto, no puede arrancar por si mismo.

Para superar estas fuerzas de atracción iniciales, el rotor se alimenta al principio con algún tipo de entrada mecánica que rota en la misma dirección que el campo magnético a una velocidad cercana a la velocidad de sincronía. Despues de un rato se produce el anclaje magnético y el motor síncrono rota en sincronía con la frecuencia.[\[9\]](#)

2.1.2.2 Motores de Inducción o Asíncronos

Uno de los motores eléctricos más utilizados en la mayoría de aplicaciones es el motor de inducción. Este motor tambien es conocido como motor asíncrono porque se mueve a una velocidad menor que la velocidad de sincronización⁶.

Un motor de inducción siempre se mueve a una velocidad menor que la velocidad de sincronización porque su campo magnético rotatorio, el cual es producido por un estator, generara un flujo que lo hará rotar, pero debido al retraso entre la corriente de flujo en el rotor y la del estator, el rotor nunca alcanzará su velocidad de campo magnético, o velocidad de sincronía.

⁶La velocidad de sincronización es la velocidad de rotacion del campo magnético en una máquina rotatoria y depende de la frecuencia y del número de polos de la máquina

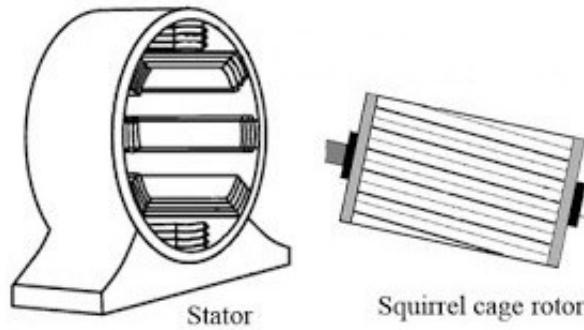


Figura 15: Motor asíncrono de caja de ardilla

Basicamente, hay dos tipos de motores de inducción que dependen del tipo de entrada - motores de una fase o motores de tres fases. Los motores de una fase no pueden arrancar por si mismos, al contrario de los de 3 fases, que si pueden.

Para hacer funcionar un motor necesitamos administrarle una doble excitación. Por ejemplo, si consideramos un motor de corriente continua, necesitaremos alimentar al estator por un lado y al rotor a través de las escobillas. Pero en el motor de inducción solo administramos un punto de alimentación, así que eso hace más interesante su modo de funcionamiento.

Es muy simple, como su propio nombre indica el proceso de inducción está presente. En realidad, cuando alimentamos el bobinado del estator, se genera un flujo en la bobina debido al paso de la corriente. Por otro lado, el bobinado del rotor esta dispuesto de tal manera que se cortocircuite a si mismo. El flujo del estator atravesará la bobina del rotor y como está cortocircuitado, de acuerdo a la ley de Faraday de inducción electromagnética, la corriente empezará a fluir por el rotor. Ahora tienen dos flujos de corriente, uno en el estator y otro en el rotor, y este último irá desfasado respecto al flujo del estator. Debido a esto, el rotor sentirá una fuerza de torsión que lo hará rotar en la dirección del flujo magnético rotatorio. Así que la velocidad del rotor dependerá de la corriente alterna y podrá ser controlada variando la alimentación entrante. Este es el principio de funcionamiento de un motor de inducción, tanto de una fase como de tres fases.[\[10\]](#)

2.2. Microcontroladores

Puesto que el caso de estudio es la placa Intel Galileo, examinaremos las plataformas con las características más parecidas, ofrecidas por las empresas más populares. Las placas que analizaremos serán **Raspberry Pi**, de la fundación Raspberry Pi, **BeagleBone Black**, de Texas Instruments, **Arduino Mega 2560**, de Arduino e **Intel Galileo**, de Intel Corporation. Por último realizaremos una comparativa entre las diferentes posibilidades, destacando los puntos fuertes de cada microcontrolador.

2.2.1. Raspberry Pi

Raspberry Pi es un mini ordenador del tamaño de una tarjeta de crédito, desarrollado por la fundación Raspberry Pi, una organización benéfica de Reino Unido, con la intención de proporcionar ordenadores de bajo costo y software gratuito a estudiantes.

Raspberry Pi 3 Modelo B es la tercera generación del microcontrolador fabricado por la firma homónima, el cual sustituyó al modelo anterior en Febrero de 2016. Cuenta con un procesador de cuatro núcleos ARMv8 de 64 bits a 1.2GHz y utiliza una distribución de Linux Raspbian como sistema operativo. Tambien cuenta con tarjeta de video integrada, 1GB de RAM y una ranura para Micro SD.

En cuanto a conectividad, esta placa dispone de un 4 puertos USB, una interfaz para cámara(CSI), una interfaz de Display(DSI), un puerto HDMI, 40 puertos GPIO, un Jack de 3.5mm con audio y video y un puerto Ethernet. Además cuenta con conectividad Wifi y Bluetooth 4.1. Si a todo esto añadimos un precio de unos 42 euros aproximadamente, tenemos una de las placas más competitivas del mercado actual. [11]



Figura 16: Raspberry Pi 3 Modelo B

2.2.2. Beaglebone black

BeagleBone Black es un pequeño ordenador de hardware y software abierto además de una plataforma de desarrollo de bajo coste que puede conectarse a casi cualquier dispositivo que tengamos en casa. Cuenta con un Procesador ARM Cortex A8 Sitara de bajo coste fabricado por Texas Instruments. Esta placa sale de fábrica con una distribución Debian GNU/Linux en la Flash para evaluación y desarrollo, aunque hay muchos otros sistemas operativos compatibles, como Ubuntu, Android o Fedora.

Entre sus especificaciones, este dispositivo cuenta con el procesador AM335x Cortex-A8 de ARM a 1GHz, una RAM DDR3 de 512MB, almacenamiento interno de 4GB, acelerador de gráficos 3D, acelerador NEON de punto flotante y 2 microcontroladores PRU de 32 bits.

Las comunicaciones en este dispositivo se llevan a cabo gracias de 2 puertos usb, uno cliente y alimentación y otro anfitrión, un puerto ethernet, un puerto HDMI y 46 pines digitales. Su precio es de unos 46 euros y está soportada por una gran comunidad de desarrolladores y aficionados.[\[12\]](#)

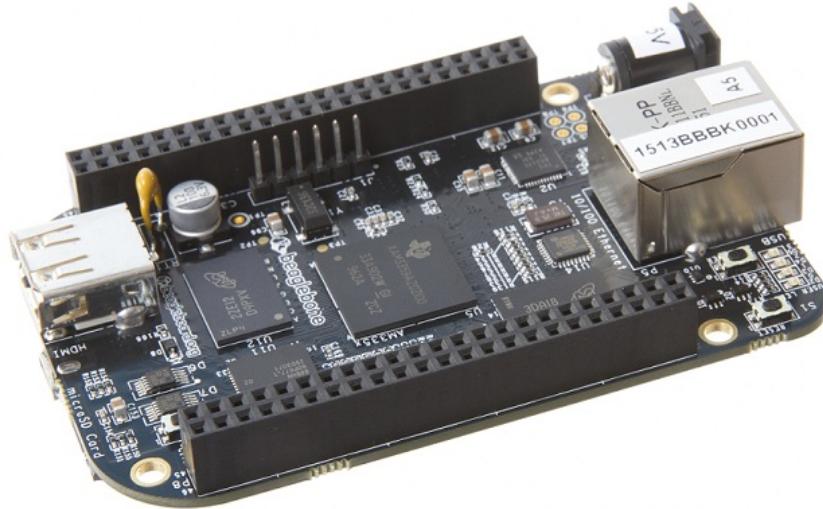


Figura 17: BeagleBone Black

2.2.3. Arduino Mega 2560

El Arduino Mega 2560 es una placa con microcontrolador basado en el ATMega2560, un núcleo de 8-bits y 16MHz de bajo consumo fabricado por Atmel. Tiene un tamaño superior al de su hermano menor, el Arduino Uno, pero a su vez dispone de 4 veces más pines(digitales, analógicos y PWM), así como una memoria flash y RAM ampliadas.

Para comunicarse consta de 54 pines digitales de entrada/salida(15 de los cuales se pueden usar como salida PWM), 16 entradas analógicas, 4 UARTs(puertos serie hardware), una conexión ICSP y un puerto USB. Se alimenta mediante un puerto Jack o USB y dispone de un botón de reset.

Además, el Arduino Mega es compatible con todos los *shields* diseñados para el Arduino Duemilanove y Diecimila y tiene un coste de tan solo 35 euros.[\[13\]](#)



Figura 18: Arduino Mega 2560

2.2.4. Intel Galileo

La placa Intel Galileo esta basada en un procesador de 32 bits y 400MHz Intel Quark SoC X1000. Es la primera placa basada en la arquitectura Intel diseñada para ser compatible en pines, tanto en hardware como en software, con los *shields* diseñados para el Arduino Uno Rev 3.

Esta placa también es compatible con el entorno de desarrollo de Arduino, lo que permite empezar a desarrollar en un abrir y cerrar de ojos.

Además de la compatibilidad en software y hardware con la plataforma Arduino, Intel Galileo cuenta con puertos de entrada/salida y otras características de la industria del PC, las cuales expanden el uso nativo y las capacidades más allá del ecosistema Arduino. Consta de una ranura mini-PCI Express, un puerto Ethernet de 100Mb, una ranura para Micro SD, un puerto serie RS-232, 2 puertos USB(Cliente/Anfitrión) y una memoria flash NOR de 8MByte.

El genuino procesador Intel junto con las capacidades de entrada/salida nativas que rodean al *SoC*⁷ ofrece un pack completo para ambas comunidades de desarrolladores y estudiantes. Además es muy útil para desarrolladores profesionales que buscan un entorno de desarrollo más simple y efectivo en costes que los diseños basados en procesadores de gama superior, como Intel Atom e Intel Core. [14]

La versión de Intel Galileo estudiada en este trabajo es la primera generación, pero desde febrero de 2014 hay una segunda generación de esta placa, que cuenta con un par de mejoras poco significativas, como la alimentación a través de ethernet(12V) y un sistema de regulación de la potencia que admite potencias entre 7 y 15V(anteriormente 5V estáticos).



Figura 19: Intel Galileo

⁷Sistem on a Chip o Sistema en un Chip

2.2.5. Comparativa entre las diferentes plataformas

En la siguiente tabla se muestran las principales características más importantes de entre todas las plataformas de estudio seleccionadas.

Espec\Nombre	Raspberry Pi	BeagleBone B	A.Mega 2650	Intel Galileo
Modelo	Modelo B Rev 3	Rev C	Rev 3	Gen 1
Dimensiones	85.60 x 56mm	86 x 54.61mm	102 x 54mm	10 x 7cm
Procesador	ARM Cortex-A53 Quad-Core	ARM Cortex-A8	ATmega2560	Intel Quark X1000
Velocidad	1.2Ghz	1Ghz	16 MHz	400Mhz
Conjunto de Ins.	64bit	32bit	24	32bit
Real Time Clock	No	No	No	Sí
Caché	32kByte L1 512kByte L2	64kByte L1 256kByte L2	No	16 KByte L1
RAM	1GB LPDDR2	512MB SDRAM	8KB SRAM	512KB SRAM 256MB DRAM
Memoria FLASH	No	4GB 8-bit eMMC	256 KB	8MB NOR Flash
GPU	Broadcom VideoCore IV	PowerVR SGX530	No	No
GPIO	40	65	54	14
Pines Analógicos	No	7	16	6
Pines PWM	No	8	14	6
USB	4 Puertos USB	2 Puertos USB Anfit. y Cliente	1 Puerto USB	2 Puertos USB Anfit. y Cliente
Networking	Ethernet	Ethernet	No	Ethernet y Slot PCI-E
Soporte Audio	Jack 3.5, HDMI	Micro HDMI	No	No
Soporte Video	HDMI y RCA	Micro HDMI	No	No
Almacenam. Ext	Tar. Micro-SD	Tar. Micro-SD	No	Tar. Micro-SD
Precio	42 euros	46 euros	35 euros	54 euros

2.2.6. Ventajas e inconvenientes de las diferentes plataformas

Los apartados anteriores muestran los puntos fuertes de cada plataforma. Tanto Beaglebone Black como Raspberry Pi tienen un procesador de gran potencia y dan soporte a salida de audio y video, mientras que el Arduino Mega y la placa de Intel tienen una mayor compatibilidad con todos los accesorios Arduino. En cuanto a pines de entrada/salida el claro vencedor es el Arduino Mega 2650, seguido muy de cerca por la placa BBB. El Intel Galileo cuenta por ejemplo con ciertas ventajas que las otras plataformas no poseen, como un reloj de tiempo real y una

ranura para tarjeta wifi integrada. Por último, la plataforma más económica es la placa Arduino, aunque en relación características/precio, la clara vencedora es la Raspberry Pi.

Por lo tanto, dependiendo de la tarea que vayamos a realizar usaremos una placa u otra. En nuestro caso, el control de motores puede llevarse a cabo con cualquiera de estas 4 plataformas sin ningún problema. Siendo Intel Galileo la plataforma elegida desde un principio, intentaremos sacarle el mayor partido.

2.3. Control de motores eléctricos

En esta sección analizaremos diferentes técnicas de control de motores mediante circuitos eléctricos.

2.3.1. Pulse Width Modulation - PWM

La modulación de ancho de pulso(PWM) es una potente técnica para controlar circuitos analógicos con las salidas digitales de un microprocesador. Esta técnica se emplea en una amplia variedad de aplicaciones, que van desde la medición y las comunicaciones hasta el control de potencia y la conversión.

Una señal analógica tiene un valor que varía constantemente, con una resolución infinita tanto en tiempo como en magnitud. Una batería de nueve voltios es un ejemplo de dispositivo analógico, ya que su voltaje de salida no es precisamente 9V, cambia en el tiempo, y puede tomar cualquier valor real. De forma similar, la cantidad de corriente extraída de una batería no está limitada a un conjunto finito de valores posibles. Las señales analógicas son distinguibles de las señales digitales porque la última siempre toma valores de un conjunto finito de posibilidades predeterminadas, como el conjunto {0V-5V}.

Las tensiones y corrientes analógicas se pueden usar para controlar directamente las cosas, como por ejemplo el volumen de una radio. En una radio simple, la rueda de volumen esta conectada directamente a una resistencia variable. Al girarla, la resistencia sube o baja, aumentando o disminuyendo la corriente que fluye a través de la resistencia. Esto cambia la cantidad de corriente que llega a los altavoces, aumentando o disminuyendo el volumen. Un circuito analógico es uno cuya salida es linealmente proporcional a la entrada.

Aunque un circuito analógico puede parecer muy intuitivo y simple, no siempre es económicamente atractivo o práctico. Por una parte, los circuitos analógicos tienen a desplazarse en el tiempo y, por lo tanto, pueden ser difíciles de sincronizar. Los dispositivos analógicos de precisión, los cuales resuelven este problema, pueden ser muy grandes, pesados y caros. Además dichos dispositivos pueden ponerse muy calientes, la potencia disipada es proporcional al voltaje que pasa a través de los elementos activos multiplicado por la corriente. Los circuitos analógicos

también pueden ser sensibles al ruido. Debido a su resolución infinita, cualquier perturbación o ruido en una señal analógica cambia necesariamente el valor actual.

Al controlar digitalmente los circuitos analógicos, los costes del sistema y el consumo de energía se reducen drásticamente. Por ese motivo la mayoría de microcontroladores y DSPs ya incorporan controladores PWM en el chip, haciendo sencilla su implementación.

En pocas palabras, la modulación del ancho de pulso es una manera de codificación digital de los niveles de una señal analógica. Mediante el uso de contadores de alta resolución, el ciclo de trabajo de una señal cuadrada es modulado para codificar un nivel específico de señal analógica. La señal PWM sigue siendo digital porque, en cualquier instante de tiempo dado, la alimentación de corriente continua está completamente encendida o apagada. La fuente de tensión o corriente se suministra a la carga por medio de una serie repetitiva de impulsos de encendido y apagado. El tiempo de encendido es el tiempo durante el cual se aplica la alimentación de corriente continua a la carga, y el tiempo de apagado es el periodo de tiempo durante el cual la alimentación se desconecta. Dado un ancho de banda suficiente, cualquier valor analógico puede ser codificado mediante PWM.

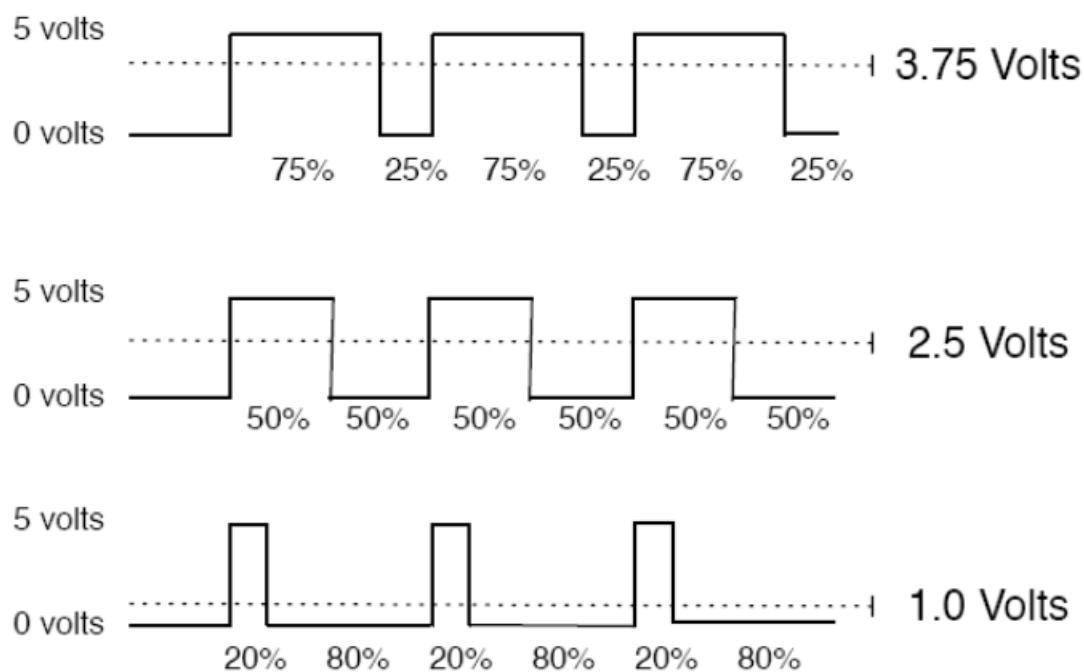


Figura 20: Varios ejemplos de señal PWM

La Figura 11 muestra tres señales PWM diferentes. La primera muestra una salida de PWM con un ciclo de trabajo del 75 %. Es decir, la señal está encendida para el 75 % del período y fuera del otro 25 %. La segunda y tercera figuras muestran las salidas PWM a ciclos de trabajo del 50 % y 20 %, respectivamente. Estas tres salidas PWM codifican tres valores de señal analógica

diferentes, al 75 %, 50 % y 20 % de la intensidad total. Como la alimentación es de 5V y el ciclo de trabajo es del 75 %, se obtiene una señal analógica de 3.75V.

Una de las ventajas del PWM es que la señal permanece digital desde el procesador hasta el sistema controlado. No es necesaria una conversión digital-analógica. Manteniendo la señal digital, los efectos causados por el ruido se minimizan. El ruido solo puede afectar a una señal digital si es lo suficientemente fuerte para cambiar un 1-lógico a un 0-lógico, o viceversa. Esta es la principal razón por la que PWM se usa a veces en comunicaciones. [15]

2.3.2. Controlador PID

El Control Proporcional-Integral-Derivativo(PID) es el algoritmo de control más común utilizado en la industria y ha sido universalmente aceptado en el control industrial. La popularidad de los controladores PID puede ser atribuida a su robusto rendimiento en un amplio rango de condiciones de funcionamiento, así como a su simplicidad funcional, la cual permite a los ingenieros operar con ellos de una forma simple y directa.[16]

Como su propio nombre indica, el algoritmo PID consiste en tres coeficientes básicos; proporcional, integral y derivativo, los cuales varían para obtener una respuesta óptima.

Sistema de control

La idea básica detrás de un controlador PID es leer datos de un sensor, para luego procesar la salida deseada del actuador calculando las respuestas proporcional, integral y derivativa y sumando estos tres componentes para calcular la salida. Antes de empezar a definir los parámetros de un controlador PID, describiremos lo que es un **sistema de bucle cerrado**.

En un sistema de control típico, la variable de proceso es el parámetro del sistema que necesita ser controlada, como la temperatura, la presión o el caudal. Se utiliza un sensor para medir la variable de proceso y proporcionar retroalimentación al sistema de control. El punto de ajuste es el valor que se desea que alcance la variable de proceso, como lo serían 100 grados Celsius en el caso de un sistema de control de temperatura. En un momento dado, la diferencia entre la variable de proceso y el punto de ajuste es utilizada por el algoritmo del sistema de control(compensador), para determinar la salida deseada del actuador para impulsar el sistema(planta). Por ejemplo, si la temperatura medida por un proceso variable son 100 °C y la temperatura deseada de ajuste son 120 °C, entonces la salida del actuador especificada por el algoritmo de control debe ser activar un calentador. Activar un actuador para encender un calentador provoca que el sistema se caliente, y esto resulta en un incremento en la variable de temperatura del proceso. Esto se conoce como **sistema de bucle cerrado**, porque el proceso de leer el sensor proporciona una retroalimentación constante y calcular la salida deseada en el actuador se repite continuamente y a una velocidad fija como se observa en la siguiente figura.

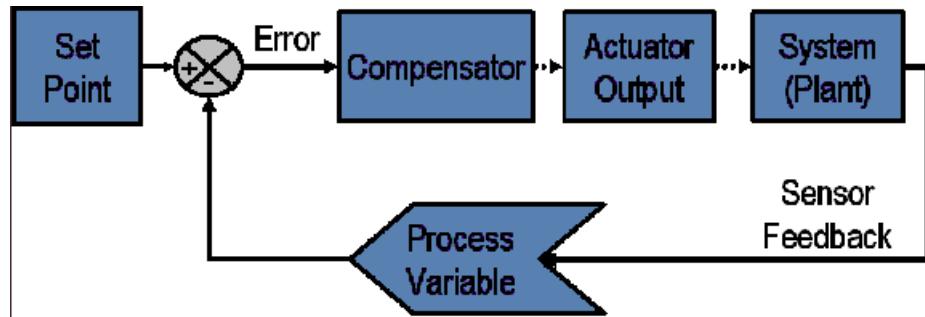


Figura 21: Diagrama de bloques de un sistema de bucle cerrado

En muchos casos, la salida del actuador no es la única señal que tiene efecto en el sistema. Por ejemplo, en una cámara de temperatura puede haber una fuente de aire frío que sopla a veces en el interior, alterando la temperatura. Este término se denomina perturbación. Normalmente trataremos de diseñar el sistema de control para minimizar el efecto de las perturbaciones en la variable de proceso.

El proceso de diseño de control empieza definiendo los requisitos de rendimiento. El rendimiento del sistema de control se suele medir aplicando una función escalonada como variable de comando de punto de ajuste y, a continuación, midiendo la respuesta de la variable de proceso. Normalmente, la respuesta es cuantificada midiendo las características de formas de onda definidas. Definiremos la siguiente terminología:

- El **tiempo de retardo (t_d)** se define como el tiempo que tarda la señal de respuesta en alcanzar por primera vez la mitad del valor final.
- El **tiempo de subida(t_r)** es la cantidad de tiempo que tarda el sistema en pasar del 10 % al 90 % del valor del estado estacionario o final⁸.
- El **tiempo pico(t_p)** es el tiempo requerido para que el sistema alcance el primer máximo de sobreelongación.
- La **sobreelongación máxima(M_s)** es la máxima cantidad que la variable de proceso sobrepasa el valor final, expresado como un porcentaje del valor final.
- El **tiempo de asentamiento(t_s)** es el tiempo requerido para que la variable de proceso se establezca dentro de un cierto porcentaje(normalmente 5 %) del valor final.
- El **error de estado estable** es la diferencia final entre la variable de proceso y el punto de ajuste.

⁸Las cantidades pueden variar entre la industria y el mundo académico.

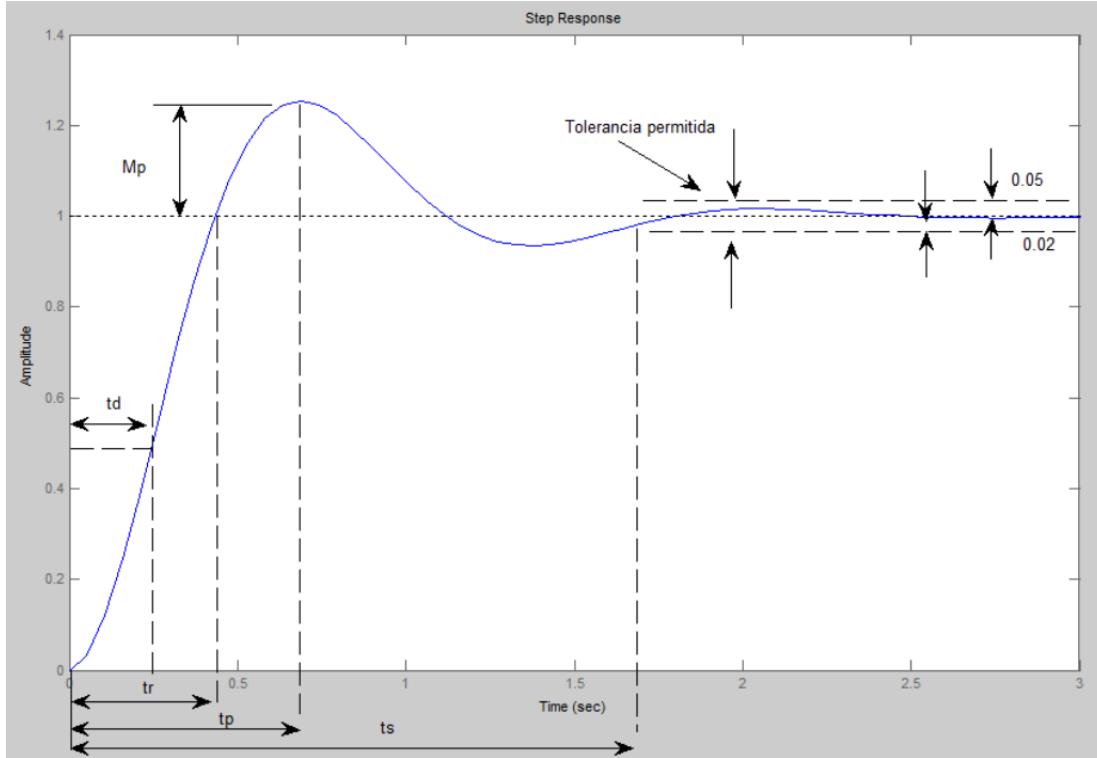


Figura 22: Respuesta de un sistema PID de bucle cerrado[17]

Después de usar una o todas estas cantidades para definir los requisitos de rendimiento para un sistema de control, es útil definir las condiciones más desfavorables en las que se espera que el sistema de control cumpla estos requisitos de diseño. Muchas veces, hay una perturbación en el sistema que afecta a la variable de proceso o a la medición de la misma. Es importante diseñar un sistema de control que rinda satisfactoriamente en los peores casos posibles. La medida de lo bien que el sistema de control es capaz de superar los efectos de las perturbaciones se conoce como **rechazo de perturbación** del sistema de control.

En algunos casos, la respuesta del sistema a una salida de control puede cambiar con el tiempo o en relación con alguna variable. Un sistema no lineal es un sistema en el que los parámetros de control que producen una respuesta deseada en un punto de operación puede no producirla en otro punto de operación diferente. Por ejemplo, una cámara parcialmente llena de fluido presentará una respuesta mucho más rápida a una modificación de temperatura cuando este casi vacía que cuando este casi llena. La medida de lo bien que el sistema tolerará las perturbaciones y las no linealidades se conoce como **robustez** de un sistema de control.

Algunos sistemas presentan un comportamiento no deseado llamado **tiempo de retardo**. El tiempo de retardo es un retraso entre cuando cambia una variable y cuando el cambio puede ser observado. Por ejemplo, si el sensor de temperatura esta muy lejos de una válvula de entrada

de agua fria, no medirá un cambio en la temperatura inmediatamente después de abrir o cerrar la válvula. El tiempo de retardo también puede ser causado por un sistema o actuador de salida que tenga una respuesta lenta al comando de control, por ejemplo, una válvula que es lenta al abrir o cerrar. Una fuente común de tiempo de retardo en las plantas químicas es el retraso causado por el flujo de fluidos a través de las tuberías.

El **ciclo de bucle** también es un parámetro importante en un sistema de bucle cerrado. El intervalo de tiempo entre las llamadas al algoritmo de control es el tiempo de ciclo de bucle. Los sistemas que cambian rápido o que tienen un comportamiento complejo requieren velocidades de bucle de control más rápidas.

Una vez definidos los requisitos de rendimiento es el momento de examinar el sistema y seleccionar un esquema de control adecuado. En la gran mayoría de aplicaciones, un control PID proporcionará los resultados necesarios.

Teoría del PID

La **componente proporcional** depende solamente de la diferencia entre el punto de ajuste y la variable de proceso. Esta diferencia se conoce como término de error. La ganancia proporcional(K_p) determina la relación entre la respuesta de salida y la señal de error. Por ejemplo, si el término de error tiene una magnitud de 10, una ganancia proporcional de 5 producirá una respuesta proporcional de 50. En general, el aumento de la ganancia proporcional incrementará la velocidad de respuesta del sistema de control. Sin embargo, si la ganancia proporcional es muy grande, el proceso variable comenzará a oscilar. Si aumentamos K_p , las oscilaciones se harán más grandes y el sistema se volverá inestable e incluso oscilará fuera de control.

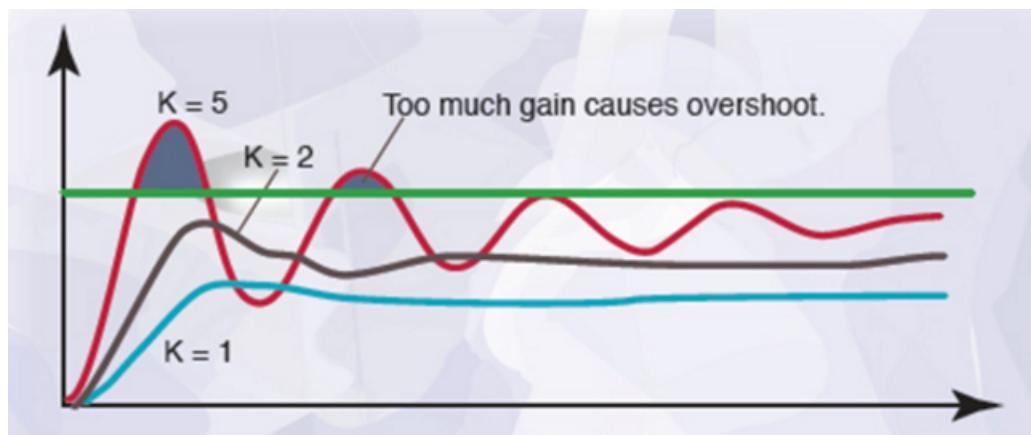


Figura 23: Señal Proporcional[18]

La componente proporcional es la encargada de acercarse al punto. Cuanto más alta es la ganancia proporcional, más pequeño será el término de error, aunque hay que llevar cuidado porque esto también causa mayor inestabilidad y da lugar a oscilaciones.

La fórmula de la componente proporcional está dada por:

$$P_{\text{sal}} = K_p e(t) \quad (1)$$

La **componente integral** suma el término de error en el tiempo. El resultado es que incluso un pequeño término de error hará que la componente integral aumente lentamente. La respuesta integral incrementará continuamente en el tiempo hasta que el error sea cero, así que el efecto es conducir el error de estado estacionario a cero. El error de estado estacionario es la diferencia final entre la variable de proceso y el punto de ajuste. Un fenómeno llamado *windup* integral resulta cuando la acción integral satura un controlador sin que el controlador conduzca la señal de error hacia cero.

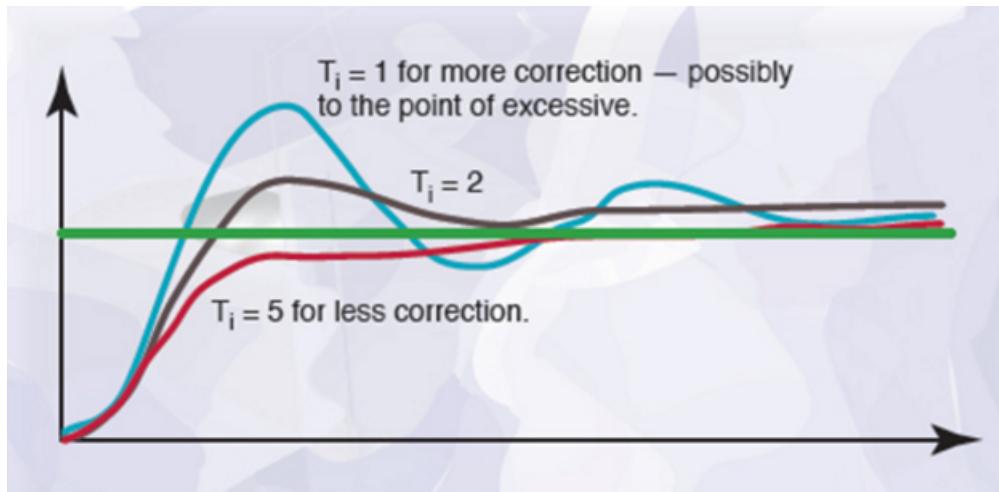


Figura 24: Señal Integral[18]

La componente integral es la encargada de corregir el error estacionario, ya que solo con la componente integral la señal no se ajusta lo suficiente. Si consideramos que el factor integral es como una cesta que guarda todo el error medido, cuando la integral esta funcionando correctamente, la cesta esta casi vacía.

La fórmula de la componente integral está dada por:

$$I_{\text{sal}} = K_i \int_0^t e(\tau) d\tau \quad (2)$$

La **componente derivativa** hace que la salida disminuya si la variable de proceso está aumentando rápidamente. La respuesta derivativa es proporcional a la tasa de cambio de la variable de proceso. Aumentar el parámetro de tiempo derivativo(T_d) hará que el sistema de control reaccione de manera más contundente a los cambios en el término de error e incrementará la velocidad de respuesta global del sistema de control. Los sistemas de control más prácticos usan un tiempo derivativo muy pequeño, debido a que la respuesta derivativa es muy sensible al ruido en la señal de la variable de proceso. Si la señal de realimentación del sensor es ruidosa o si la velocidad del bucle de control es demasiado lenta, la respuesta derivativa puede hacer que el sistema de control sea inestable.

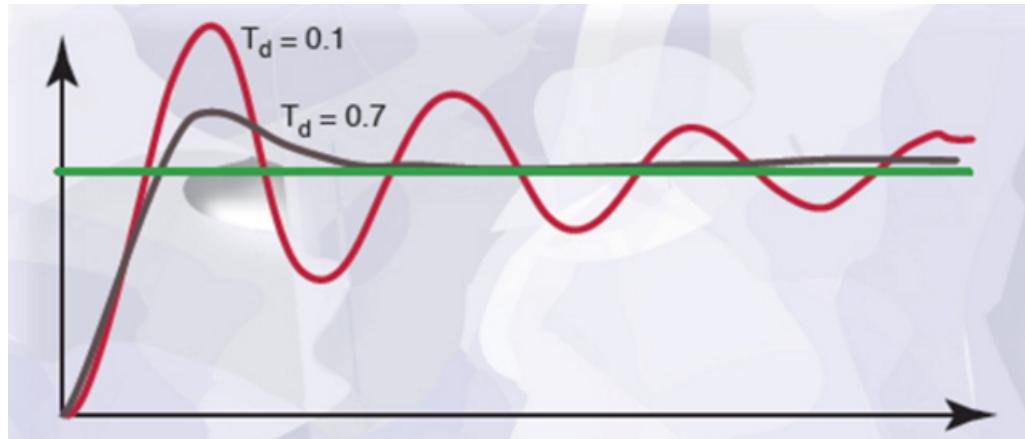


Figura 25: Señal Derivativa[18]

El factor derivativo es el más difícil de entender de los tres factores. La señal proporcional corrige instancias de error, la integral corrige la acumulación de error y la derivativa corrige el error actual frente al error la última vez que se comprobó. En otras palabras, la señal derivativa se encarga de controlar la tasa de cambio de error. El efecto que causa es el de contrarrestar la señal sobreexpuesta a causa de la componente proporcional y/o la componente integral.

La fórmula de la componente derivativa está dada por:

$$D_{\text{sal}} = K_d \frac{de}{dt} \quad (3)$$

Ajuste de la señal

El proceso que se realiza de establecer las ganancias óptimas de P, I y D para obtener una respuesta ideal de un sistema de control se llama ajuste. Es una parte muy importante del diseño ya que a veces el sistema oscila mucho, responde muy lento, tiene error de estado estacionario o es inestable.

Existen varios métodos para ajustar un bucle PID. La elección del método dependerá en gran medida de si el bucle se puede desconectar o no para la sintonización, y la velocidad de respuesta del sistema. Si el sistema puede ser desconectado el mejor método de ajuste a menudo implica someter el sistema a un cambio de paso en la entrada, medir la salida como una función de tiempo y usar esta respuesta para determinar los parámetros de control.

Si el sistema debe permanecer conectado, un método de ajuste consiste en definir primero los valores I y D a cero y aumentar P hasta que el bucle de salida oscile, y entonces incrementar I hasta que la oscilación pare e incrementar D hasta que el bucle sea aceptablemente rápido para alcanzar su referencia. Un ajuste de bucle PID rápido suele excederse un poco para alcanzar el punto de ajuste más rápidamente.

El ajuste conocido como método Ziegler-Nichols, fue introducido por John G. Ziegler y Nathaniel B. Nichols de Taylor Instruments en 1942. Esta técnica también implica ajustar las ganancias I y D a cero y luego aumentar la ganancia de P hasta que la salida del bucle comience a oscilar. Se registra la ganancia crítica K_c y el periodo de oscilación de la salida P_c y se ajustan los valores de P, I y D de la siguiente manera:

Ziegler–Nichols method			
Control type	K_p	K_I	K_d
P only	$0.5 \cdot K_c$	-	-
PI	$0.45 \cdot K_c$	$1.2 K_p / P_c$	-
PID	$0.6 \cdot K_c$	$2 K_p / P_c$	$K_p P_c / 8$

Figura 26: Tabla de valores del método Ziegler-Nichols

Aunque el hecho es que la mayoría de las instalaciones industriales ya no sintonizan bucles manualmente, sino que usan software de ajuste y optimización. Estos paquetes de software recopilan datos, desarrollan modelos de procesos, sugieren ajustes óptimos e incluso desarrollan ajustes mediante la recopilación de datos de cambios de referencia. Esto se puede hacer tanto con conexión como sin ella. También puede incluir análisis de válvulas y sensores, y simulación antes de descargar. La única pega: el software es costoso e implica cierto entrenamiento.

Ejemplo de algoritmo PID

Por último se muestra un ejemplo sencillo de controlador PID, en pseudocódigo:

```
error_anterior = 0
integral = 0
KP = Valor fijado previamente(ver seccion de ajuste)
KI = Valor fijado previamente(ver seccion de ajuste)
KD = Valor fijado previamente(ver seccion de ajuste)

while(1) {
    error = valor_deseado - valor_actual
    integral = integral + (error*tiempo_iteracion)
    derivativo = (error - error_anterior)/tiempo_iteracion
    salida = KP*error + KI*integral + KD*derivativo
    error_anterior = error
    sleep(tiempo_iteracion)
}
```

2.4. Aplicaciones actuales

Todos los motores eléctricos tienen algún tipo de controlador. Dependiendo de la complejidad de la tarea que vaya a realizar el motor, estos controladores tendrán diferentes características.

Los sistemas de control de motores son usados a lo largo del los sectores industrial, Automoción, Médico, Aviación y Defensa, y consumen el 45 % de la energía mundial. Entre los ejemplos de control de motores se encuentran los procesos de automatización en fábricas, ensamblado y empaquetado, control de elevación, robótica, electrodomésticos, control de vuelo y muchos más. A continuación pasamos a comentar algunos de ellos.

2.4.1. Electrodomésticos

La mayoría de los aparatos domésticos están impulsados por un motor eléctrico, y aunque la mayoría de ellos están controlados de una manera simple y rudimentaria, la electrónica está empezando a tomar protagonismo.

En los electrodomésticos y los aparatos para la iluminación del hogar, la parte electrónica se encuentra normalmente en la interfaz hombre-máquina(panel de control, mandos a distancia, etc), así como en la gestión de secuencias de operaciones complejas, en lavadoras por ejemplo;

sin embargo, es solo el principio del uso de la electrónica para mejorarlos. Sistemas de ahorro de energía, silencio, flexibilidad y simplicidad son requisitos con una importancia creciente: en aplicaciones como taladros, aspiradoras y frigoríficos, un control de velocidad variable es el principal medio para obtener estas características de rendimiento.

Un caso concreto de uso es la lavadora. Las lavadoras Europeas y Japonesas suelen ser máquinas de eje horizontal, con un tambor accionado por un motor brushed universal. La velocidad del tambor se suele implementar usando un rectificador de fase controlada y un microcontrolador de 8 o 16 bits. En cualquier caso los motores universales tienen problemas conocidos de desgaste de las escobillas y limitaciones a altas velocidades. En contraste, los motores de inducción de corriente alterna no usan escobillas y tienen un mayor rango de velocidades.

En esta aplicación, la velocidad de ondulación y la carga de torsión de la lavadora nos dan información muy valiosa sobre la carga de lavado. La variación de la carga de torsión junto con la rotación del tambor pueden darnos información sobre el tejido predominante en la carga de lavado. Por lo tanto, el controlador de la máquina puede seleccionar automáticamente el programa de lavado que más se ajuste y simplificar así su uso. La velocidad de las ondas se puede usar para estimar el desbalanceo de la carga antes de empezar el ciclo de giro, mejorando así la fiabilidad mecánica de la máquina.[\[19\]](#)

2.4.2. Sistema de Seguimiento Solar

Normalmente un medio para el seguimiento del sol requiere de actuadores mecánicos para manejar el sistema de seguimiento solar y así asegurar un enfoque perpendicular muy preciso del dispositivo óptico de recolección solar hacia el centro del sol para una instalación situada en una zona geográfica concreta.



Figura 27: Sistema de seguimiento solar

Cualquier sistema de seguimiento solar automático de lazo abierto necesita un algoritmo para calcular los vectores solares alrededor del disco de la circunferencia solar y orientarse en función de dichas líneas, ya sea continuamente o en intervalos de tiempo discretos. Para este tipo de sistema se usa el algoritmo SPA⁹. Para algunos procesadores o microcontroladores, existen versiones más sencillas del algoritmo SPA, adaptado para dar soporte a un conjunto reducido de cálculos bajo un conjunto de asunciones simplificadas.

La tarea principal de un mecanismo de seguimiento solar es alinear el colector solar lo más cerca posible del vector solar. Las acciones asociadas a las maniobras del colector solar con el uso de motores de corriente continua se muestran en la siguiente figura con rastreador solar de doble eje(ángulo azimuth y ángulo zenith).

⁹Uno de los algoritmos más precisos, para el cálculo de la localización del sol usando una base astronómica algebraica, que fué desarrollada bajo contrato en el Laboratorio Nacional de Energías Renovables del Departamento de Energía de los Estados Unidos(NREL) por Andreas(Reda and Andreas, 2008a).

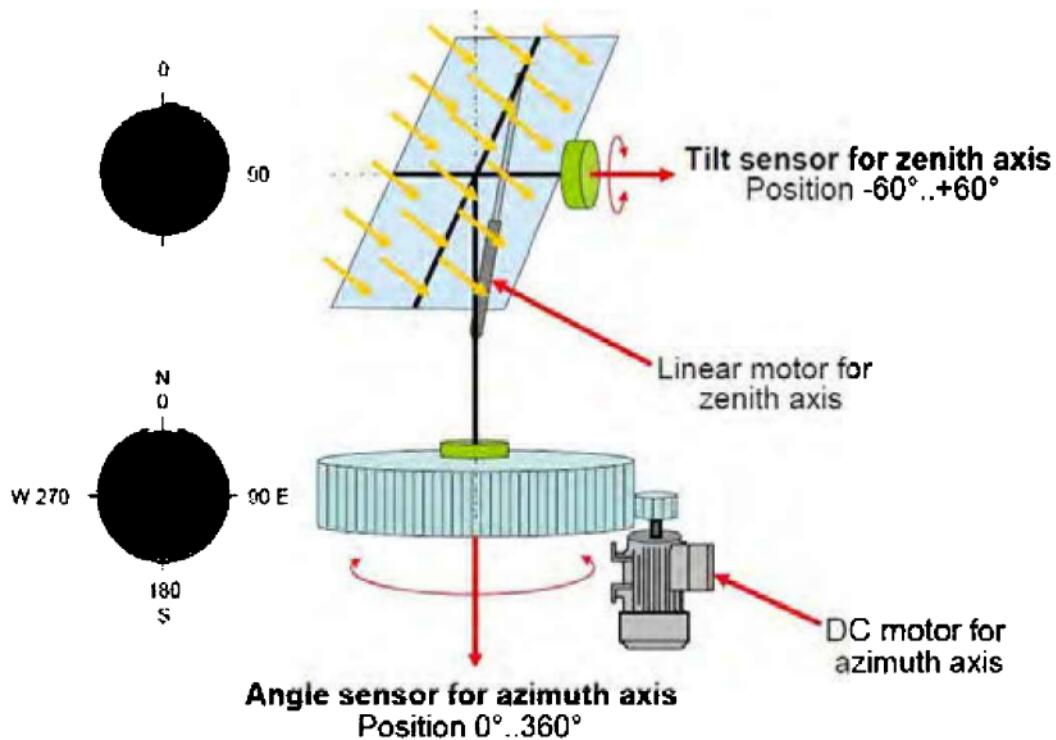


Figura 28: Esquema de motores y ángulos de giro

Durante el proceso de seguimiento del sol, un sensor de ángulo detecta la posición azimuth actual mientras que la posición zenith es detectada a través del sensor de inclinación. Con ayuda del algoritmo de posición solar astronómica, el controlador calcula ambos ángulos del vector solar en intervalos de tiempo definidos y compara la orientación actual del rastreador solar con la posición del sol.[20]

2.4.3. Robótica

Desde la antigüedad las personas han soñado con tener artilugios inteligentes capaces de ofrecernos compañía, entretenimiento o simplemente ayudarnos a realizar nuestras tareas más tediosas. La palabra robot” fue acuñada en 1920 por Karel Capek en su obra R.U.R. (Robots Universales Rossum), y proviene del término del idioma checo y muchas otras lenguas eslavas para denominar el ”trabajo duro”(robo).

Hoy en día, a pesar de los problemas de aceptación de los empleados, se emplean millones de robots a la hora de realizar las tareas más críticas, repetitivas y mundanas. No importa que tipo de automóvil tengas, lo más probable es que haya sido soldado, pintado e inspeccionado, en su gran mayoría por robots.

Los robots han llegado incluso hasta el hogar de hoy en día. Quien no ha visto un robot de limpieza pasearse de manera errática por el salón, o ha escuchado en alguna conversación lo fácil y cómodo que es cocinar con una *Thermomix*¹⁰?

2.4.3.1 Caso de ejemplo - Drones

Los drones se han popularizado mucho en los últimos años y son usados en el sector de Defensa, audio-visual, recreativo e incluso para entregar paquetes a domicilio¹¹.

El término dron, proviene del inglés, *drone* y significa "vehículo aéreo no tripulado guiado a control remoto o mediante una computadora a bordo". Si bien nosotros en esta sección nos enfocaremos más bien en un tipo concreto de dron, el cuadricóptero, cuya composición podemos observar en la siguiente figura:

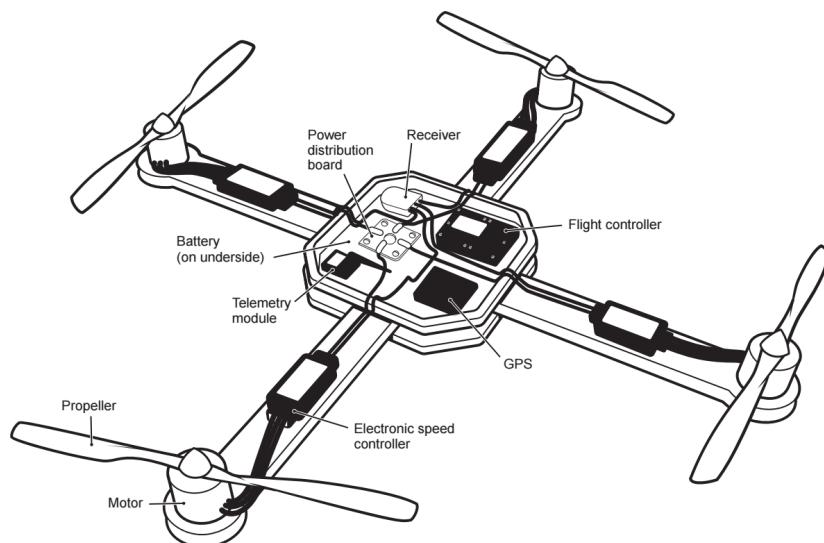


Figura 29: Esquema dron cuadricóptero

El modo de funcionamiento de un cuadricóptero es bastante sencillo, esta máquina puede moverse de manera horizontal, vertical y girar sobre si mismo, simplemente aumentando la velocidad relativa de unos motores respecto a otros. A continuación explicamos como se produce cada uno de estos movimientos.

¹⁰Marca popular de robots de cocina de la compañía Alemana Vorwerk.

¹¹Amazon Prime Air es un servicio de entrega de paquetes con drones a domicilio, actualmente en fase de pruebas.

Movimiento Vertical

El movimiento vertical del dron, es decir, hacia arriba o hacia abajo, es trivial y se produce aumentando o disminuyendo la velocidad de todos los motores simultáneamente.

Movimiento horizontal

El movimiento horizontal, ya sea hacia delante y atrás o hacia los lados, se produce aumentando la velocidad relativa de los dos motores que apuntan hacia la dirección a la que se desea moverse, con respecto de sus opuestos.

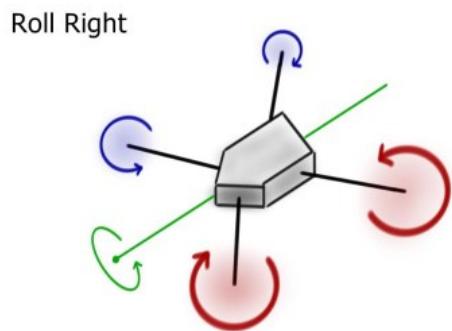


Figura 30: Cuadricóptero - Giro derecha

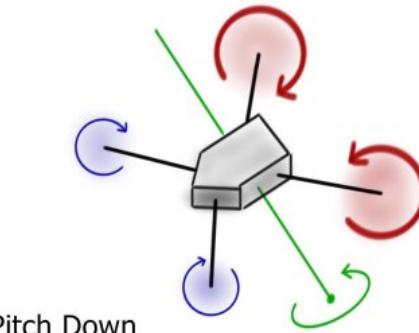


Figura 31: Cuadricóptero - Giro hacia atrás

Giro lateral

Para comprender como se produce este movimiento debemos observar antes que disponemos de 2 de los motores girando en sentido horario(rojo), y dos motores girando en sentido antihorario(azul).

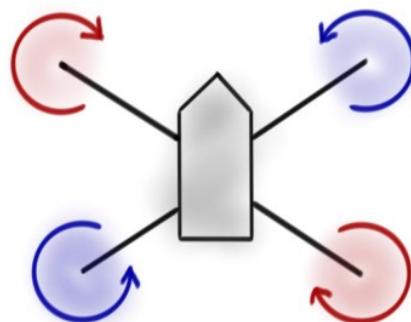


Figura 32: Dirección de giro de los motores

Como podemos observar en la figura anterior, si aumentamos la velocidad relativa de los motores que giran en sentido horario, el cuadrocóptero girará en sentido antihorario. Si por el contrario giramos los motores azules, el dron girará en sentido horario.

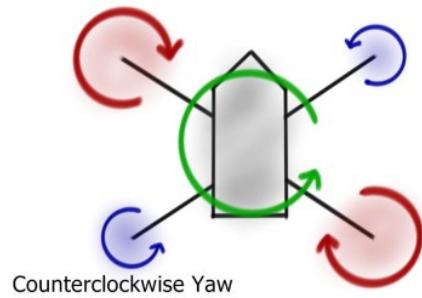


Figura 33: Dron - rotación en sentido antihorario

El control de todos estos movimientos, así como el de la estabilidad del dron, se lleva a cabo gracias al microcontrolador de abordo y los diferentes sensores de posicionamiento. Por otra parte, estos principios de funcionamiento son los mismos para cualquier dron con un número superior de motores.[\[21\]](#)

3. Análisis de Objetivos

El objetivo de este trabajo es el de evaluar las capacidades del microcontrolador Intel Galileo, entre ellas el manejo de motores de corriente continua. En este caso podremos controlar el arranque, la parada, el cambio de sentido y la velocidad. Esta última la sabremos gracias a un sensor ensamblado junto con el motor, o contando el número de pasos en el caso de un motor paso a paso.

Por otro lado nos comunicaremos con el microprocesador a través de la red, para que se asemeje lo máximo posible a un caso de uso real.

O1. Estudio y comparativa de Intel Galileo frente a otras plataformas y tecnologías actuales

Se hará una comparativa entre la plataforma seleccionada para el estudio, Intel Galileo, y las demás plataformas del mercado actual. También se analizarán las diversas tecnologías de control de motores eléctricos.

O2. Diseño de un escenario práctico

Se diseñará un escenario práctico, mediante el cual poder mostrar las capacidades de la plataforma seleccionada en el ámbito del control de motores electrónicos.

O3. Selección de los componentes necesarios

Se estudiarán y seleccionarán todos los componentes necesarios para montar el escenario diseñado.

O4. Montaje del escenario

Se ensamblarán todas las piezas para el correcto funcionamiento del escenario.

O5. Control de motores usando Intel Galileo

Se desarrollará un programa usando el lenguaje de programación Arduino, capaz de controlar un motor eléctrico de manera precisa usando la plataforma Intel Galileo. El control del motor constará de un mecanismo de cambio de sentido, arranque y parada progresivos, y un aumento y reducción de la velocidad.

O6. Diseño de una aplicación móvil

Diseño de una aplicación Android sencilla que llevará a cabo la comunicación con el microcontrolador a través de la red. Esta comunicación servirá tanto para controlar el motor a través del microcontrolador como para monitorizar la información del sensor de velocidad.

O7. Representación y comparación de la velocidad actual usando información de realimentación

Se desarrollará un método para representar de manera precisa la velocidad actual del motor por dos vías diferentes, por un lado usando la información de realimentación provista por un sensor de giro, y por otro lado gracias a la información del número de pasos que obtenemos del motor *stepper*.

O8. Manejo a través de la red

Preparación de la plataforma Intel Galileo para que pueda enviar y recibir datos a través de la red. Los datos son necesarios tanto para controlar el microprocesador de manera remota, como para recibir información de parte de este.

4. Diseño y resolución del trabajo realizado

El escenario montado para la realización de la parte práctica de este trabajo dispone de los siguientes componentes:

- **Motor paso a paso:** El motor modelo 17HS-240E[22] es un motor paso a paso híbrido de alta eficiencia con 400 pasos por vuelta. Este motor esta conectado a un driver, encargado de mandar las señales de control, y por la parte anterior, a un sensor(encoder) encargado de medir la velocidad de giro.
- **Controlador:** El driver modelo SMC42[23] es un controlador compacto de corriente constante. Es el encargado de transformar las señales enviadas por el microcontrolador hacia el motor, duplicar la precisión de paso y protegerlo en caso de una subida de tensión. El control se realiza mediante 3 cables, el cable amarillo(ENABLE), es el encargado de activar o parar el motor, el cable marrón(DIRECTION) es el encargado de marcar la dirección de giro, y por último el cable azul(CLOCK) transporta la señal de reloj para activar los pasos del motor.
- **Sensor de velocidad:** El encoder modelo E6CP-AG5C[24] es un sensor de velocidad con 8 líneas de señal, lo cual indica una precisión de giro desde 2^0 a 2^7 señales por vuelta. En nuestro caso solo hemos conectado el cable correspondiente a la señal 2^6 (Gris), lo cual implica que mandará una señal hacia el microcontrolador cada media vuelta. En la siguiente figura podemos observar que es necesario añadir 3 resistencias al circuito para poder medir de manera correcta la señal enviada por este sensor.
- **Microcontrolador:** El microcontrolador Intel Galileo es la parte central del escenario. Es el encargado de mandar las señales de control hacia el encoder, recibir los datos del sensor de velocidad y mantener la comunicación a través de la red con la aplicación móvil.
- **Aplicacion Móvil:** La aplicación móvil escrita en lenguaje Android es la encargada de mostrar de manera amigable los mandos de control para operar el motor a través del microcontrolador, así como mostrar los datos medidos por el sensor de velocidad.

A continuación se muestran un diagrama básico de bloques y un esquema gráfico del escenario montado, para hacernos una idea de la disposición de cada elemento:

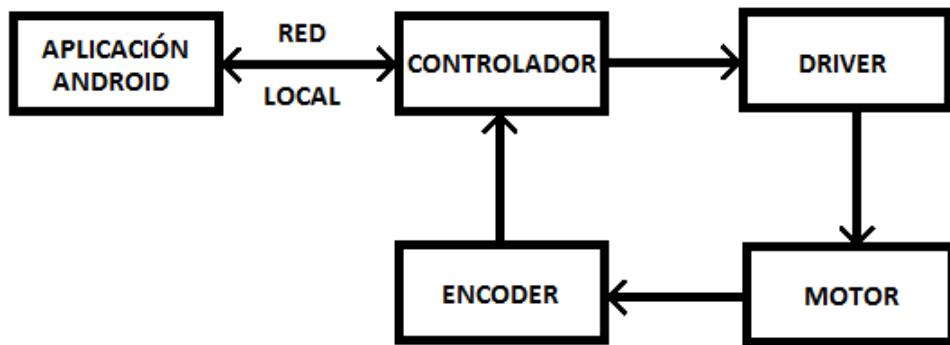


Figura 34: Diagrama de bloques del escenario montado

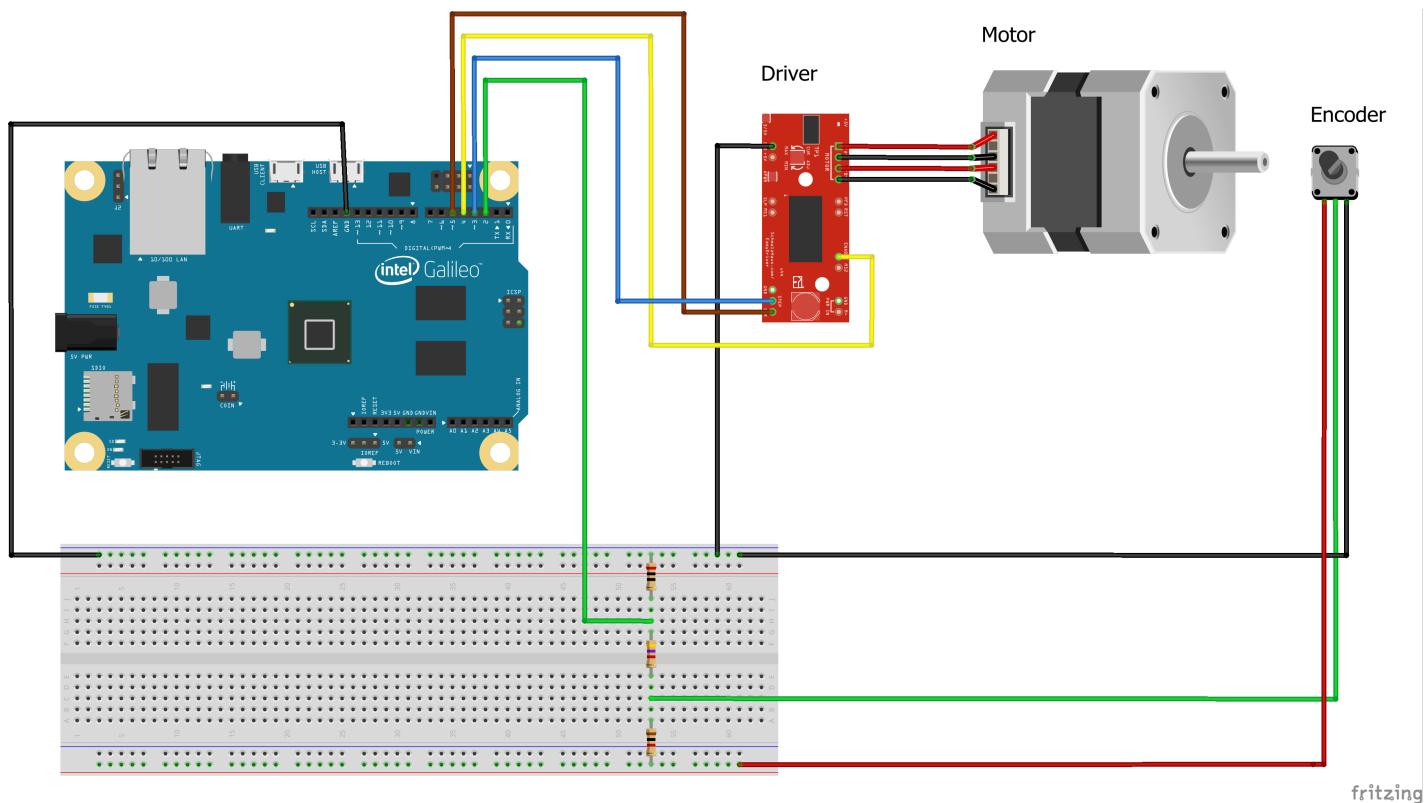


Figura 35: Esquema visual del escenario montado

Finalmente este es el escenario real tras el montaje:

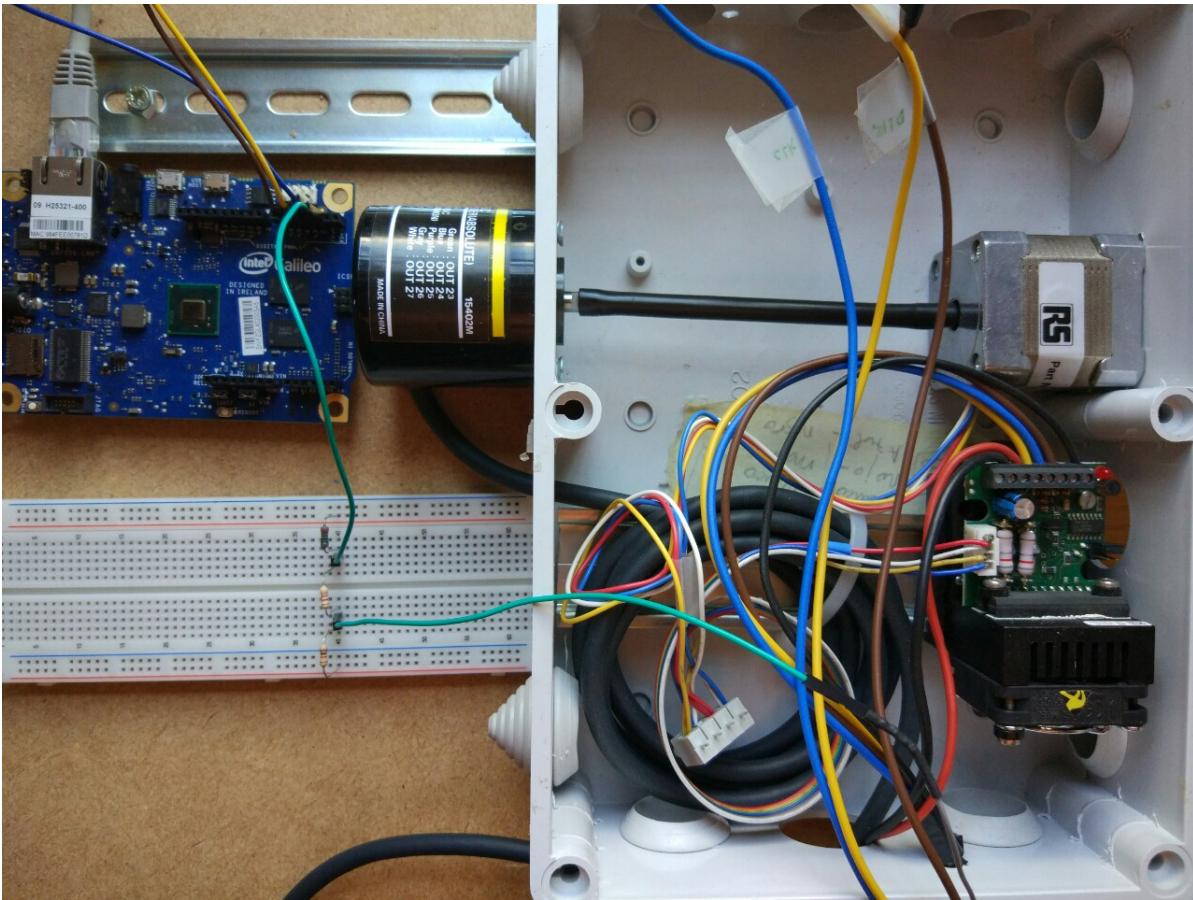


Figura 36: Escenario montado

Como podemos observar, el microcontrolador se sitúa arriba a la izquierda, junto al encoder justo en el centro de la pantalla. A la derecha tenemos el motor paso a paso, y justo debajo se sitúa el driver encargado de manejar el motor. Abajo a la izquierda tenemos la placa de conexiones la cual contiene el circuito de resistencias necesarias para recibir la señal del encoder.

4.1. Toma de contacto con Intel Galileo

Los primeros pasos con Intel Galileo son los propios de la configuración de un dispositivo de estas características. Aunque ya viene con una **distribución de linux preinstalada**. Hay que tener en cuenta que si deseamos que el programa cargado en memoria sea persistente tras el apagado debemos arrancar desde la tarjeta. En caso contrario el sketch cargado en memoria se borrará.

En este caso hemos instalado una distribución Yocto de linux ya que dispone de más librerías y utilidades que la básica. También hay versiones de Windows disponibles para instalar en esta plataforma. El proceso de instalación es muy sencillo, simplemente hay que grabar la imagen del sistema operativo elegido en una tarjeta micro SD con ayuda de un programa de instalación, e introducir la tarjeta en el dispositivo de Intel. En el próximo arranque, el microcontrolador arrancará desde la tarjeta micro SD.

Para comunicarnos con el dispositivo a la hora de configurar el sistema necesitaremos un cable serial-a-USB. Este cable nos permitirá abrir un terminal en el microcontrolador desde el ordenador, como veremos más adelante a la hora de configurar la conexión de red del dispositivo.

Por último, necesitamos instalar un entorno de desarrollo. Este dependerá del lenguaje de programación que se elija. En este caso los lenguajes de programación disponibles son los siguientes[25]:

- **Arduino:** Es un entorno de programación basado en C++ fácil de aprender y de código abierto. Es eficaz a la hora de añadir rápidamente sensores, ya que hay un montón de código distribuido por la red. Además, debido que este dispositivo es compatible con los pines de Arduino, hay un montón de *shields* Arduino para elegir. El IDE de arduino es la aplicación a elegir para programar con Arduino.
- **JavaScript y Node.js:** Estos lenguajes de programación son idóneos a la hora de crear interfaces web así como a la hora de comunicarse con otros dispositivos en la nube. En este caso el IDE a elegir sería XDK de Intel, el cual viene con plantillas de proyecto enfocadas a IoT.
- **C++:** Este lenguaje de programación provee de potencia y total control del sistema, y además aprovecha una gran cantidad de bibliotecas de código disponibles. Intel System Studio IoT Edition es un IDE que viene con una capacidad incorporada para integrar sensores de manera sencilla desde nuestra biblioteca de GitHub.
- **Java:** Este famoso y sencillo lenguaje también ha sido aceptado recientemente en el Intel System Studio IoT Edition.

Tras exponer y estudiar las distintas posibilidades, la opción elegida es Arduino. Uno de los motivos de mi elección ha sido la familiaridad con el entorno, ya que lo había trabajado en la asignatura de *Programación de Sistemas Embebidos en Red*. Aunque hay otros lenguajes con los que también estoy familiarizado, sobre todo con Java, el hecho de que la mayor parte de la literatura y documentos de este tipo de dispositivos esté redactada usando el lenguaje de Arduino es el motivo principal de mi elección.

Una vez instalado el IDE de Arduino y descargado el SDK de Intel para dicha plataforma, ya podemos empezar a programar para Intel Galileo, de la misma manera que haríamos con cualquier otro dispositivo Arduino.

4.2. Programación en Intel Galileo

La estructura de un programa en lenguaje Arduino se compone de 3 partes, la declaración de las librerías y variables globales, el apartado de inicialización(*setup*), y el bucle de ejecución principal, en ese orden:

```
//Librerias y declaracion de variables globales

void setup(){
//Declaracion de variables locales
//Inicializacion de variables
}

void loop(){
//Bucle de ejecucion principal
}
```

Por otra parte, los componentes que tendremos que manejar son, el motor(a través del driver), el encoder, y la red.

4.2.1. Configuración y manejo del motor

Como hemos comentado antes, el control del driver del motor se lleva a cabo mediante 3 cables, los cuales se encargan de la señal de arranque y parada, de la dirección de giro, y de la señal de reloj. La inicialización de los pines la lleva a cabo el siguiente código:

```
pinMode(3, OUTPUT_FAST); //CLK
pinMode(4, OUTPUT); //ENAB
pinMode(5, OUTPUT); //DIR

digitalWrite(4, LOW); // Pone a 0 ENAB
digitalWrite(5, HIGH); // Pone a 1 DIR (clockwise)
```

Como podemos observar, primero se establece el tipo de pin de cada uno de los 3 cables, en este caso los 3 pines son de salida. Tras indicar el tipo de pines les asignamos un valor, en este caso, el programa comenzará con el motor parado(*ENAB* a 0), y la dirección establecida en sentido horario(*DIR* a 1).

También se puede ver en este fragmento que el pin 3, encargado de la señal de reloj, no está marcado como *OUTPUT*, sino como *OUTPUT_FAST*. Esto se debe a que una de las limitacio-

nes más importantes de la placa Intel Galileo es que la frecuencia de lectura/escritura de todos sus pines(a excepción del 2 y el 3) es de solo 230Hz. La configuración OUTPUT_FAST de este pin nos permite una frecuencia de señal de entre 477KHz y 2.93KHz, dependiendo de la función que usemos para escribir:

```
digitalWrite(3, x); //477KHz  
fastGpioDigitalWrite(3, x); //683KHz  
fastGpioDigitalWriteDestructive(latchValue); //2.93MHz
```

En nuestro caso, con una velocidad de 477KHz será suficiente.

4.2.2. Configuración del encoder

El encoder es el encargado de mandar las señales de rotación del motor hacia el microcontrolador. La configuración es la siguiente, también en el apartado setup():

```
pinMode(2, INPUT_PULLUP);  
attachInterrupt(2, InterruptISR, RISING);
```

El mecanismo de aviso del encoder se realizará mediante interrupciones. Esto se debe a que la lectura del pin es de 230Hz y ralentizaría muchísimo la ejecución del bucle principal. Por otra parte, los únicos pines capaces de activar interrupciones son el 2 y el 3. Teniendo en cuenta que el pin 3 ya está ocupado por el driver, ya solo nos queda el pin 2 para realizar esta tarea.

En este caso, el pin 2 es de lectura, ya que los datos nos los enviará el encoder. La función `attachInterrupt()` tiene 3 parámetros. El primero es el pin que va a causar la interrupción, el segundo es la rutina de interrupción a ejecutar cuando se reciba una señal, y el tercer parámetro es el tipo de señal que provocará la interrupción(flanco de subida, flanco de bajada, cambio de valor o valor 0). En nuestro caso registramos la interrupción con cada flanco de subida.

Este encoder está configurado para mandar una señal de interrupción cada media vuelta. La rutina de interrupción `InterruptISR` se verá más adelante.

4.2.3. Configuración de la red

En el apartado de variables globales declaramos las diferentes IP's, dirección MAC, y puertos. Además tambien son necesarias dos bibliotecas de código.

```
#include <Ethernet.h>
#include <EthernetUdp.h>

// ETHERNET MANAGING
byte mac[] = {
    0x98, 0x4F, 0xEE, 0x00, 0x78, 0x1D
};
IPAddress ip(192, 168, 1, 177); // IP Controlador
IPAddress ip2(192, 168, 1, 20); // IP Android
unsigned int localPort = 8888; // Puerto local
unsigned int remotePort = 6000; // Puerto remoto
```

Una vez declaradas las variables y establecidos sus valores, es necesario iniciar la conexión. En el siguiente fragmento de código se muestra el inicio de la conexión de red, y el socket UDP, dentro de la sección setup():

```
Ethernet.begin(mac, ip);
Udp.begin(localPort);
```

Por último, para enviar y recibir datos mediante UDP usaremos el siguiente código:

```
//Envio de datos
Udp.beginPacket(ip2, remotePort);
Udp.write(buf);
Udp.endPacket();

//Recepcion de datos
int packetSize = Udp.parsePacket(); // Comprueba si hay datos
Udp.read(packetBuffer, 8);
```

4.2.4. Explicación del código Arduino

A continuación explicaré los bloques de código más importantes del programa, como lo son el bucle de ejecución principal, la rutina de interrupción del encoder, y las funciones auxiliares.

4.2.4.1 Bucle de ejecución principal

El bucle de ejecución principal se compone a su vez de **3 partes**, un apartado para leer y gestionar los comandos recibidos desde la aplicación móvil, en caso de que los haya. Un bucle de espera para reducir la velocidad del bucle principal, y el envío de la señal para que el motor avance un paso.

```
void loop()
{
    // Si hay datos disponibles para leer
    int packetSize = Udp.parsePacket();
    if (packetSize) {

        Udp.read(packetBuffer, 8);

        String packet(packetBuffer);
        if (packet.equals("0")) slowStop();
        else if (packet.equals("1")) slowStart();
        else if (packet.startsWith("v"))
            setVelocity(packet.substring(1));
        else if (packet.startsWith("cd"))
            changeDirection(packet.substring(2));

        memset(packetBuffer, 0, sizeof(packetBuffer));
    }
}
```

La parte inicial del bucle comprueba si hay datos de entrada en el socket UDP. En caso de que el buffer contenga datos, lo primero que haremos será convertir el buffer en una cadena para facilitar su manejo. En función del contenido de la cadena se tomarán las diferentes decisiones. En este caso, si recibimos un **0**, llamaremos a la función de **parada progresiva**, si recibimos un **1**, la función de **arranque progresivo**, si recibimos una cadena del tipo **vX**, donde X simboliza la velocidad, llamaremos a la función de **establecimiento de velocidad**, y en el caso de recibir una cadena con el formato **cdX**, llamaremos a la función de **cambio de sentido**. Por último limpiaremos el contenido del buffer antes de su siguiente lectura. Las diferentes funciones están descritas más adelante.

```
//Bucle delay
cont=0;
while(cont<(velocity))
{
    cont++;
}

// CLK driver motor
x=!x;
digitalWrite(3, x);
}
```

La siguiente parte es la que se encarga de **ralentizar la frecuencia del bucle principal**, mediante la ejecución de otro bucle interno. Arduino nos provee de una función **delay(milisec)**, pero la espera mínima es de 1 milisegundo, lo cual implica como mucho 1000 escrituras por segundo, y teniendo en cuenta que nuestro motor necesita 800 impulsos para dar una sola vuelta la velocidad sería de aproximadamente una vuelta por segundo.

La velocidad del bucle principal debería ser, según las especificaciones técnicas, de 400MHz. Sin embargo, tras medir la frecuencia con un osciloscópio muestra una frecuencia 40MHz, 10 veces menor. Supongo que esto será debido al tiempo que se tarda en comprobar si el buffer UDP contiene datos o no. Por lo tanto, para que el motor vaya a una velocidad de, por ejemplo, 120 vueltas por minuto(RPM), obtenemos el número de ciclos que hay que esperar en el bucle de reducción de la siguiente manera:

$$40.000.000\text{Hz}/800 \text{ pasos por vuelta} = 50.000 \text{ vueltas por segundo}$$

$$50.000 \text{ vueltas por 60 segundos} = 3.000.000 \text{ vueltas por minuto(RPM)}$$

Por lo tanto, el bucle tendrá que esperar ($3.000.000/\text{velocidad en RPM}$) ciclos en cada ciclo del bucle principal, para que el motor vaya a dicha velocidad. En nuestro caso, si queremos que la velocidad inicial sea de 120RPM, el bucle tendrá que esperar ($3.000.000/120 = 25.000$) ciclos, en cada ciclo del bucle principal.

El último paso del bucle principal envía la señal de reloj al driver del motor, para que avance un paso.

4.2.4.2 Funciones auxiliares

La función **slowStop()** es la encargada de hacer una parada progresiva, el código correspondiente a dicha función es el siguiente:

```
void slowStop(){
    int speedFactor = 12000000/velocity;
    for(int i = 0; i<speedFactor; i++){
        cont=0;
        while(cont<velocity+((velocity/speedFactor)*i)) cont++;

        x=!x;
        digitalWrite(3, x);
    }
    digitalWrite(4, LOW);
}
```

Como podemos observar, este código consta de dos bucles, el bucle exterior marca el número de vueltas que se tardara en parar, mientras que el bucle interior es el encargado de variar el espacio entre las señales enviadas al motor.

El algoritmo puede parecer enrevesado. En un principio, **speedFactor** tenía un valor fijo de 3200, o lo que es lo mismo 4 vueltas del motor. Esto significa que el motor tardaba en parar 4 vueltas siempre. Para una velocidad de 120RPM, parar en 4 vueltas es un comportamiento lógico, pero a una velocidad de 1 revolución por minuto tardaríamos 4 vueltas en parar. De esta manera, cambié el valor de **speedFactor** para que fuese proporcional a la velocidad actual del motor, de tal manera que se tarde medio segundo en frenar, en lugar de 4 vueltas, lo cual tiene más sentido. Los cálculos realizados son los siguientes:

A una velocidad de 120RPM, en medio segundo se completa 1 vuelta.

Número de vueltas para frenar en 0.5 segundos = $X/\text{velocidad}(\text{en ciclos})$

$$1 \times 800 = X / 25.000 \text{ ciclos} (120 \text{ RPM})$$

$$X = 20.000.000$$

int speedFactor = 20000000/velocity;

De esta manera obtenemos la fórmula de el número de vueltas que necesitamos para frenar el motor en medio segundo, independientemente de la velocidad del motor.

La función **slowStart()** es exactamente igual a la anterior función, **slowStop()**, pero en lugar de decelerar, acelera hasta la velocidad actual.

```
void slowStart(){
    digitalWrite(4, HIGH);
    int speedFactor = 20000000/velocity;
    for(int i = 0; i<speedFactor; i++){
        cont=0;
        while(cont<velocity*2-((velocity/speedFactor)*i)) cont++;
        x=!x;
        digitalWrite(3, x);
    }
}
```

La función **setVelocity(String velocityString)** es la encargada de establecer la velocidad actual del motor. Recibe como parámetro la velocidad en formato cadena, la convierte en un entero, y hace la conversión de RPM a ciclos de espera, como ya hemos mostrado anteriormente, a razón de 3.000.000/RPM:

```
int setVelocity(String velocityString){
    int velocity = velocityString.toInt();

    return 3000000/velocity;
}
```

Por último, la función **changeDirection(String directionString)** es la encargada de cambiar la dirección de giro:

```
void changeDirection(String directionString){
    int dir = directionString.toInt();
    slowStop();
    delay(500);
    digitalWrite(5, dir);
    slowStart();
}
```

Esta función recibe como parámetro un string, indicando la dirección de giro, 0 en caso de giro anti-horario, y 1 en caso de sentido horario. Primero convertimos la cadena recibida a entero, después realizamos una parada progresiva, esperamos medio segundo, cambiamos la dirección, y hacemos un arranque progresivo.

4.2.4.3 Rutina de interrupción del sensor

La rutina de interrupción es la encargada de enviar los datos del sensor a la aplicación Android, el código es el siguiente:

```
void InterruptISR()
{
    if(RPMCounter >= 20){
        currentTime = millis();

        //Tiempo en segundos
        difference = (currentTime - lastTime) / 1000;
        sprintf(buf, 8, "%f", 600 / difference);

        Udp.beginPacket(ip2, remotePort);
        Udp.write(buf);
        Udp.endPacket();

        lastTime = currentTime;
        RPMCounter = 0;
    }
    RPMCounter++;
}
```

El pin 2, encargado de mandar la señal de activación de la interrupción se activará cada media vuelta del motor registrada por el encoder. En el caso de nuestra rutina, cada 20 señales del encoder se enviará un mensaje con la velocidad, o lo que es lo mismo, cada 10 vueltas.

Al principio de la rutina se guarda el instante de tiempo actual, seguidamente, se calcula el tiempo transcurrido desde la última vez que se mandó un mensaje. Una vez que sabemos que hemos realizado 10 vueltas en T segundos, la fórmula es la siguiente:

$$10 \text{ vueltas} \times 60 \text{ segundos} / T = \text{velocidad en RPM}$$

Por último, se envía el mensaje vía Ethernet a la IP y puerto especificados anteriormente. El código completo está disponible en Anexos, en la sección Código Completo Arduino.

4.3. Aplicación Android

La plataforma seleccionada para desarrollar una aplicación móvil es Android. Como podemos observar en la siguiente tabla, Android domina el mercado de manera aplastante.

Period	Android	iOS	Windows Phone	Others
2015Q4	79.6%	18.7%	1.2%	0.5%
2016Q1	83.5%	15.4%	0.8%	0.4%
2016Q2	87.6%	11.7%	0.4%	0.3%
2016Q3	86.8%	12.5%	0.3%	0.4%

Figura 37: Porcentaje de dispositivos por plataforma(Fuente: IDC, Nov 2016)

Además ya estoy familiarizado con esta tecnología gracias a la asignatura de Computación Móvil, y el único terminal del que dispongo ahora mismo es un terminal Android.

El código de esta aplicación es bastante sencillo, ya que la complejidad de control reside en el microcontrolador. El objetivo de la aplicación es mostrar una interfaz sencilla, mandar las órdenes hacia el microcontrolador, y leer los datos de vuelta para mostrar la velocidad leída por el sensor. El entorno de desarrollo utilizad ha sido la plataforma Eclipse. A continuación se explican las partes más importantes del código.

4.3.1. Interfaz de la aplicación

La interfaz Android está definida usando el lenguaje de marcado XML. Pese a que algunos editores como Eclipse poseen editores gráficos, es más sencillo editar directamente el código XML ya que ofrece un mayor control. El resultado final de la interfaz es el siguiente:

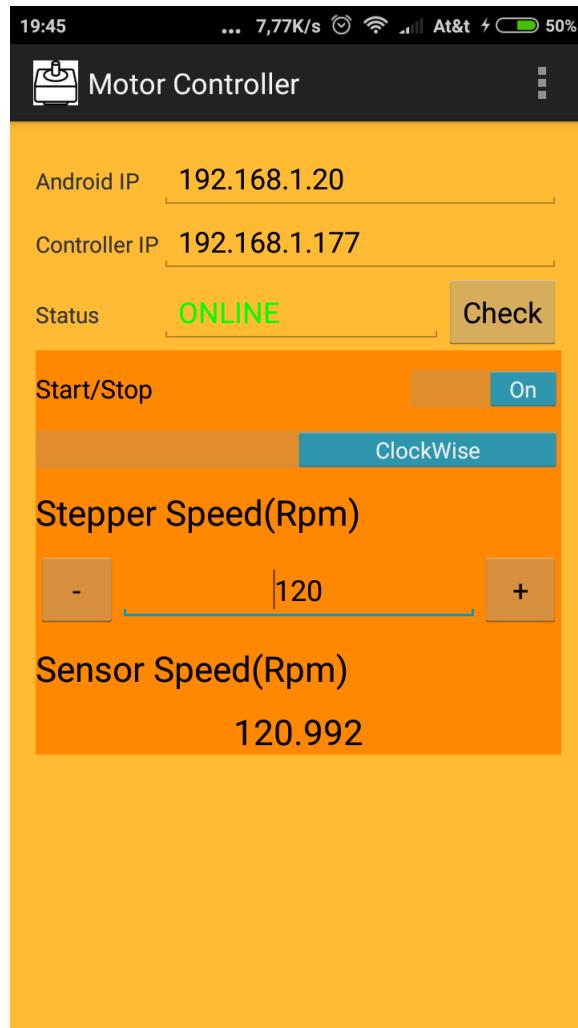


Figura 38: Interfaz de la aplicación Android desarrollada

La organización de los elementos se ha hecho por filas y columnas, con una tabla(TableLayout) en este caso. El tamaño de los componentes se adapta al tamaño de la pantalla que los contiene independientemente de la orientación, vertical u horizontal, ya que el diseño es *responsive*.

El código de todos los componentes es muy parecido, así que comentaremos un bloque sencillo, en este caso el botón Check.

```
<Button android:id="@+id/check_btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="checkButtonClick"
        android:text="@string/check">
</Button>
```

El contenido del **campo Id** se usa para acceder al elemento en cuestión desde el código Android a la vista. Los campos **ancho** y **alto** están establecidos con el valor `wrap_content`, esto significa que se adaptarán al contenido del elemento. El **campo onClick** establece el nombre del evento que se disparará al hacer click sobre el botón, el hilo principal de ejecución será el encargado de tratar este evento. Por último, el campo `text` no contiene directamente el texto del elemento, si no el nombre de la que contiene el texto, almacenado en otro fichero. Este mecanismo se utiliza básicamente para que sea más sencillo mostrar una aplicación en varios idiomas, ya que lo único que habría que hacer es cambiar el fichero dependiendo del lenguaje a utilizar.

4.3.2. Hilo principal de ejecución

Este código es la columna vertebral de la aplicación. Es el punto de inicio del programa y el encargado de comunicarse con la interfaz y las funciones auxiliares. En nuestro caso el nombre de la clase principal es `ActivityMain`:

```
public class MainActivity extends Activity {
    /*Definicion de variables globales*/
    public final int DEFAULT_SEND_PORT = 8888;
    ...

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

El código principal encargado de manejar la aplicación implementa la interfaz **Activity**. Una Actividad es un componente de la aplicación que contiene la pantalla con la que realizamos las acciones. El código dentro de la función `onCreate(...)` es el encargado de crear la actividad cuando arrancamos la aplicación. En nuestro caso establecemos valores por defecto de algunos de los campos y definimos eventos. Un ejemplo de evento es el del interruptor encargado de cambiar la dirección de giro del motor:

```
Switch directionswitch = (Switch)
    findViewById(R.id.directionswitch);
directionswitch.setOnCheckedChangeListener(new
    CompoundButton.OnCheckedChangeListener() {
        public void onCheckedChanged(CompoundButton buttonView,
            boolean isChecked) {
            if(isChecked){
                SendMessage("cd1");
            }
            else{
                SendMessage("cd0");
            }
        }
});
```

La primera linea se encarga de recuperar el componente a partir de su Id, como habíamos comentado antes. Una vez recuperado el elemento, se le asigna un evento, en este caso `OnCheckedChangeListener(...)`. Este evento se disparará cada vez que cambie el estado del interruptor. En función del valor del interruptor, se mandará la cadena `cd1` o `cd0` via UDP, haciendo uso de la función `SendMessage()`, la cual explicaremos más adelante.

Los elementos de aumento y reducción de velocidad, y de arranque y parada son muy parecidos a este. Simplemente llaman a la función `SendMessage` con las cadenas correspondientes.

4.3.3. Cliente y Servidor UDP

Para encapsular los comandos encargados de controlar el motor se usan las clases `UDP_Client` y `UDP_Server`. Ambas clases son un código adaptado extraido de [este hilo](#).

```
public class UDP_Client{
    private AsyncTask<Void, Void, Void> async_cient;

    @SuppressLint("NewApi")
    public void Send(final String message, final String address,
        final int port){
        async_cient = new AsyncTask<Void, Void, Void>() {
```

Esta clase contiene una función `Send(message, address, port)`, encargada de mandar de manera *asíncrona* el mensaje `message` a la dirección `address:port`.

```
@Override
protected Void doInBackground(Void... params)
{
    DatagramSocket ds = null;

    try
    {
        InetAddress inetAddress = InetAddress.getByName(address);
        ds = new DatagramSocket();
        DatagramPacket dp;
        dp = new DatagramPacket(message.getBytes(),
                               message.length(), inetAddress, port);
        ds.send(dp);
        System.out.println("Message sent: " + message);
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        if (ds != null)
        {
            ds.close();
        }
    }
    return null;
}
```

El método `doInBackground(...)` es el encargado de enviar el paquete en segundo plano. Primero se crea un socket, que será el encargado de enviar el datagrama, y después se crea el datagrama usando el mensaje, la dirección Ip y el puerto. Por último, se envía el datagrama haciendo uso de la función `DatagramSocket.send(DatagramPacket)`.

El modo de uso del cliente es el siguiente:

```
public UDP_Client Client = new UDP_Client();
Client.Send(message, address, port);
```

El código del **servidor UDP** es un poco más complejo, ya que tiene que mandar información de vuelta hacia la actividad principal. Es el siguiente:

```
public class UDP_Server
{
    private AsyncTask<Void, Void, Void> async;
    private boolean Server_running = true;
    public static String sensorspeed;

    @SuppressLint("NewApi")
    public void runUdpServer(final Handler mHandler, final
        Runnable mUpdateResults, final int port)
    {
        async = new AsyncTask<Void, Void, Void>()
        {
            @Override
            protected Void doInBackground(Void... params)
            {
                byte[] lMsg = new byte[8];
                DatagramPacket dp = new DatagramPacket(lMsg,
                    lMsg.length);
                DatagramSocket ds = null;

                try
                {
                    ds = new DatagramSocket(port);

                    while(Server_running)
                    {
                        ds.receive(dp);
                        String sensorSpeed = new String(lMsg,
                            "UTF-8");
                        System.out.println(sensorSpeed);
                        sensorspeed = sensorSpeed;
                        mHandler.post(mUpdateResults);
                    }
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            finally
```

```
{  
    if (ds != null)  
    {  
        ds.close();  
    }  
    return null;  
}  
};  
if (Build.VERSION.SDK_INT >= 11)  
    async.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);  
else async.execute();  
  
}  
  
public void stop_UDP_Server()  
{  
    Server_running = false;  
}  
}
```

Podemos observar que la estructura del código servidor es muy parecida al del cliente. Sin embargo, los argumentos que se le pasan son un tanto diferentes. En este caso al arrancar el servidor, pasamos como argumentos, un objeto `Handler`, un objeto `Runnable`, y un puerto. Esto se debe a que el valor a publicar, en este caso la velocidad del sensor, debe ser enviada hacia la vista, pero el único componente que puede mostrar dichos datos es el hilo principal. De esta manera, el objeto `Runnable` hace referencia a una la función del hilo principal encargada de publicar el valor recibido por la red, mientras que el objeto `Handler` es el encargado de llamar a dicha función, desde la clase `UDP_Server`.

La estructura principal consta de un bucle que se ejecuta mientras que el servidor esté activo. El mecanismo de recepción consta de los mismos elementos, un socket y un datagrama, y la recepción de los datos se realiza mediante la función `DatagramSocket.receive(DatagramPacket)`. Cada vez que se recibe un packete, se publica el mensaje con el mecanismo antes mencionado.

Por último, para arrancar el servidor se usan las siguientes líneas de código:

```
public UDP_Server Server = new UDP_Server();  
Server.runUdpServer(Handler, RunnableMethod, port);
```

4.3.4. Funciones auxiliares

La clase Utils.java contiene varios métodos de clase auxiliares utilizados por la actividad principal, para **leer la ip local**, y para **mandar un ping**. Ambas son funciones adaptadas de este hilo y este hilo respectivamente.

```
public static String getIPAddress(boolean useIPv4) {  
    try {  
        List<NetworkInterface> interfaces =  
            Collections.list(NetworkInterface.getNetworkInterfaces());  
        for (NetworkInterface intf : interfaces) {  
            List<InetAddress> addrs =  
                Collections.list(intf.getInetAddresses());  
            for (InetAddress addr : addrs) {  
                if (!addr.isLoopbackAddress()) {  
                    String sAddr = addr.getHostAddress();  
                    boolean isIPv4 = sAddr.indexOf(':') < 0;  
  
                    if (useIPv4) {  
                        if (isIPv4)  
                            return sAddr;  
                    }  
                    else {  
                        if (!isIPv4) {  
                            int delim = sAddr.indexOf('%'); // drop ip6 zone  
                            suffix  
                            return delim < 0 ? sAddr.toUpperCase() :  
                                sAddr.substring(0, delim).toUpperCase();  
                        }  
                    }  
                }  
            }  
        }  
    }  
    catch (Exception ex) {} // for now eat exceptions  
    return "";  
}
```

Este método se encarga de leer las interfaces de una la lista de interfaces de red, y devolver la IPv4 o IPv6 local, en función del parámetro recibido.

```
public static String ping(String url) {  
    String str = "";  
    try {  
        Process process =  
            Runtime.getRuntime().exec("/system/bin/ping -c 3 " + url);  
        BufferedReader reader = new BufferedReader(new  
            InputStreamReader(process.getInputStream()));  
        int i;  
        char[] buffer = new char[4096];  
        StringBuffer output = new StringBuffer();  
        while ((i = reader.read(buffer)) > 0)  
            output.append(buffer, 0, i);  
        reader.close();  
        str = output.toString();  
    }  
    catch (IOException e) {  
        e.printStackTrace();  
    }  
  
    System.out.println("Ping Response "+str);  
    return str;  
}
```

Este método recibe como parámetro una url, a la cual envía el comando `ping -c 3`, dirigido a la url recibida como parámetro, lee la respuesta recibida tras ejecutar el comando y la devuelve como resultado en forma de cadena.

5. Conclusiones y vias futuras

En este trabajo se han explorado algunas de las muchas posibilidades de la plataforma Intel Galileo, entre las cuales se encuentran la configuración de la red mediante un terminal linux, la comunicación a través de la red, el control de un motor paso a paso de manera precisa, y la lectura de datos emitidos por un encoder gracias al mecanismo de interrupciones.

Durante la realización de estas tareas, he podido observar los diferentes puntos fuertes y débiles de este microcontrolador.

Entre sus puntos fuertes, destacaría la compatibilidad total con la escena Arduino. Además de la posibilidad de instalar Windows de manera nativa y trabajar con un entorno de desarrollo Intel. También es de las pocas plataformas que dispone de slot para insertar una tarjeta WiFi.

En el otro extremo, algunos de los puntos flojos son la escasez de entrada salida a una velocidad útil, ya que de los 14 pines disponibles, solo 2 soportan interrupciones y una velocidad de escritura considerable y los otros 12 pines solo soportan 230Hz de frecuencia. Además no dispone de tarjeta gráfica ni de sonido, y la potencia de procesamiento de otras plataformas superan a esta en más del doble. Si a todo esto añadimos que el precio es el mayor de todas las plataformas de la misma categoría, tenemos una placa que no es lo suficientemente competitiva.

Los microcontroladores como BeagleBone Black son superiores al modelo estudiado y sobre todo Raspberry Pi, que es la placa líder del mercado en este momento, tanto por el precio como por sus prestaciones.

Mediante la realización de este trabajo he aprendido a configurar un escenario completo enfocado al control de motores de manera inalámbrica, desde el montaje, a la programación de cada una de estas piezas de hardware, y su configuración y comunicación a través de la red.

Entre las posibles ampliaciones de este trabajo se encuentran la aplicación práctica del uso y control de motores. Uno de los posibles trabajos es el control de altura y orientación de antenas con varios motores para la medición de interferencias dentro de una cámara de medición. Para la realización de dicho trabajo sería necesario un shield arduino que expandiese el número de pines de escritura rápida.

Otra de las vias futuras sería la creación de una aplicación móvil en otras plataformas, como Windows Phone, IOs o la creación de una interfaz web.

Por último, también podríamos añadir conectividad inalámbrica mediante la inserción de una tarjeta PCI-Express(Wifi), o un shield arduino que provea de conectividad 3G.

Referencias

- [1] MARTIN DOPPELBAUER - *The invention of the electric motor*, KIT – The Research University in the Helmholtz Association. Disponible en: <https://www.eti.kit.edu/english/1376.php>
- [2] CHRIS WOODFORD, *Explain that Stuff/Electric Motors - online book*. Disponible en: <http://www.explainthatstuff.com/electricmotors.html>
- [3] MAT DIRJISH, *Difference Between Brush DC And Brushless DC Motors*, Febrero 2012. Disponible en: <http://electronicdesign.com/electromechanical/what-s-difference-between-brush-dc-and-brushless-dc-motors>
- [4] *DC Motor Driver Fundamentals*, Semiconductor Components Industries, Marzo 2014. Disponible en: http://www.onsemi.com/pub_link/Collateral/TND6041-D.PDF
- [5] PADMARAJA YEDAMALE, *Brushless DC (BLDC) Motor Fundamentals*, Microchip Technology Inc. Disponible en: <http://electrathonoftampabay.org/www/Documents/Motors/Brushless%20DC%20%28BLDC%29%20Motor%20Fundamentals.pdf>
- [6] STEVEN KEEPING (2013). *An Introduction to Brushless DC Motor Control - Driving a BLDC motor*, Digi-Key Electronics. Disponible en: <http://www.digikey.com/en/articles/techzone/2013/mar/an-introduction-to-brushless-dc-motor-control>
- [7] RESTON CONDIT AND DR. DOUGLAS W. JONES, *Stepping Motors Fundamentals*, Microchip Technology Inc. Disponible en: <http://homepage.cs.uiowa.edu/~jones/step/an907a.pdf>
- [8] *What is a servo Motor?*, electrical4u, 2011-2017. Disponible en: <http://www.electrical4u.com/what-is-servo-motor/>
- [9] *Synchronous Motor Working Principle*, electrical4u, 2011-2017. Disponible en: <http://www.electrical4u.com/synchronous-motor-working-principle/>
- [10] *Induction Motor Working Principle*, electrical4u, 2011-2017. Disponible en: <http://www.electrical4u.com/induction-motor-types-of-induction-motor/>
- [11] *Documentación Raspberry Pi 3 Model B*, Raspberry Pi Foundation. Disponible en: <https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/>
- [12] *Documentación BeagleBone Black*, Organización BeagleBoard. Disponible en: https://cdn.sparkfun.com/datasheets/Dev/Beagle/BBB_SRM_C.pdf
- [13] *Documentación Arduino Mega 2560*, Arduino. Disponible en: <https://www.arduino.cc/en/Main/ArduinoBoardMega2560>

- [14] *Documentación Intel Galileo*, Intel Corporation. Disponible en: http://www.intel.com/newsroom/kits/quark/galileo/pdfs/Intel_Galileo_Datasheet.pdf
- [15] MICHAEL BARR. (2012). *Introduction to Pulse Width Modulation*, embedded.com. Disponible en: <http://www.embedded.com/electronics-blogs/beginner-s-corner/4023833/Introduction-to-Pulse-Width-Modulation>
- [16] NATIONAL INSTRUMENTS (2011). *PID Theory Explained*. Disponible en: <http://www.ni.com/white-paper/3782/en/>
- [17] MAGNO GUERRERO NABOA. (2001). *Determinación de los parámetros de un controlador PID para una planta con función de transferencia conocida*, pg 29, Universidad Veracruzana, Facultad de Ingeniería Mecánica Eléctrica. Disponible en: <https://core.ac.uk/download/pdf/16308150.pdf>
- [18] PAUL AVERY (2009). *Introduction to PID control - Motion System Design*, Yaskawa Electric America, Inc. Disponible en: <http://machinedesign.com/sensors/introduction-pid-control>
- [19] *DSP Motor Control Boosts Efficiency In Home Appliances*, Electronic Design. Disponible en: <http://electronicdesign.com/dsp/dsp-motor-control-boots-efficiency-home-appliances>
- [20] PRINSLOO, G.J., DOBSON, R.T. (2015). *Solar Tracking*. SolarBooks. ISBN 978-0-620-61576-1, páginas 13 y 131. Disponible en: https://www.researchgate.net/publication/263128579_Solar_Tracking_Sun_Tracking_Sun_Tracker_Solar_Tracker_Follow_Sun_Sun_Position
- [21] TOGLEFRITZ (29 Abril 2014). *The Physics of Quadcopter Flight*. Disponible en: <http://blacktieaerial.com/the-physics-of-quadcopter-flight/>
- [22] *Especificaciones del motor paso a paso modelo 17HS-240E* - RS, Amidata S.A. Disponible en: <http://es.rs-online.com/web/p/products/4328273>
- [23] *Especificaciones del driver modelo SMC42* - Nanotec. Disponible en: <http://www.farnell.com/datasheets/23986.pdf>
- [24] *Especificaciones del encoder modelo E6CP-AG5C* - OMRON Corporation. Disponible en: https://www.ia.omron.com/data_pdf/cat/e6cp-a_ds_e_6_3_csm496.pdf
- [25] *Getting Started with the Intel® Galileo Board on Windows* - Setting up the IDE. Intel. Disponible en: <https://software.intel.com/en-us/get-started-galileo-windows-step5>

A. Código

A.1. Código Completo Arduino

```
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>

char buf[8];
float lastTime, currentTime, difference;
const int pinInt0 = 2;
int cont=0;
int RPMCounter=0;
int incomingByte = 0;
int velocity = 25000;
int x = 0;

byte mac[] = {
  0x98, 0x4F, 0xEE, 0x00, 0x78, 0x1D
};
IPAddress ip(192, 168, 1, 177);
IPAddress ip2(192,168,1,20);
unsigned int localPort = 8888;
unsigned int remotePort = 6000;

char packetBuffer[8];
EthernetUDP Udp;

void setup()
{
  //Pines Driver
  pinMode(3, OUTPUT_FAST); //CLK
  pinMode(4, OUTPUT); //ENAB
  pinMode(5, OUTPUT); //DIR

  //Encoder
  pinMode(pinInt0, INPUT_PULLUP);
  attachInterrupt(pinInt0, InterruptISR, RISING);

  digitalWrite(4, LOW); // Pone a 0 ENAB
  digitalWrite(5, HIGH); // Pone a 1 DIR (clockwise)
```

```
currentTime = millis();

Ethernet.begin(mac, ip);
Udp.begin(localPort);

Serial.begin(9600);
Serial.println("--- Inicio de programa ---");
Serial.println();
}

void loop()
{
    int packetSize = Udp.parsePacket();
    if (packetSize) {

        Udp.read(packetBuffer, 8);

        String packet(packetBuffer);
        if (packet.equals("0")) slowStop();
        else if (packet.equals("1")) slowStart();
        else if (packet.startsWith("v")) velocity =
            setVelocity(packet.substring(1));
        else if (packet.startsWith("cd"))
            changeDirection(packet.substring(2));

        memset(packetBuffer, 0, sizeof(packetBuffer));
    }

    cont=0;
    while(cont<(velocity)) //Bucle delay
    {
        cont++;
    }
    x=!x;
    digitalWrite( 3, x );
}

int setVelocity(String velocityString){
    int velocity = velocityString.toInt();
    return 2900000/velocity;
```

```
}

void slowStart(){
    digitalWrite(4, HIGH);
    int speedFactor = 20000000/velocity;
    for(int i = 0; i<speedFactor; i++){
        cont=0;
        while(cont<velocity*2-((velocity/speedFactor)*i)) cont++;

        x=!x;
        digitalWrite(3, x);
    }
}

void slowStop(){
    int speedFactor = 20000000/velocity;
    for(int i = 0; i<speedFactor; i++){
        cont=0;
        while(cont<velocity+((velocity/speedFactor)*i)) cont++;

        x=!x;
        digitalWrite(3, x);
    }
    digitalWrite(4, LOW);
}

void changeDirection(String directionString){
    int dir = directionString.toInt();
    slowStop();
    delay(500);
    digitalWrite(5, dir);
    slowStart();
}

void InterruptISR()
{
    if(RPMCounter>=20){
        currentTime = millis();
        difference = (currentTime-lastTime)/1000;

        sprintf(buf, 8, "%f", 600/difference);
    }
}
```

```
    Udp.beginPacket(ip2, remotePort);
    Udp.write(buf);
    Udp.endPacket();

    lastTime = currentTime;
    RPMCounter=0;
}
RPMCounter++;
}
```

A.2. Código Completo Android

Fichero `activity_main.xml`:

```
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_orange_light"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.motorcontroller.MainActivity" >

    <TableLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:stretchColumns="1" >

        <TableRow
            android:id="@+id/tableRow1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" >

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/androidip" />
    
```

```
<EditText
    android:id="@+id/androidip_tb"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_span="2"
    android:focusable="false"
    android:inputType="text" >

    <!-- <requestFocus /> -->
</EditText>
</TableRow>

<TableRow
    android:id="@+id/tableRow2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/controllerip" />

<EditText
    android:id="@+id/controllerip_tb"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_span="2"
    android:inputType="text" >
</EditText>
</TableRow>

<TableRow
    android:id="@+id/tableRow3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/status" />
```

```
<EditText
    android:id="@+id/status_tb"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:focusable="false"
    android:inputType="textNoSuggestions" >
</EditText>

<Button
    android:id="@+id/check_btn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="checkButtonClick"
    android:text="@string/check" >
</Button>
</TableRow>

<TableRow
    android:id="@+id/tableRow4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >

    <TableLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_span="3"
        android:background="@android:color/holo_orange_dark"
        android:stretchColumns="*" >

        <TableRow
            android:id="@+id/tableRow5"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingTop="10dp" >

            <Switch
                android:id="@+id/onoffswitch"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/startstop"
                android:textOff="Off"
                android:textOn="On" />

```

```
</TableRow>

<TableRow
    android:id="@+id/tableRowX"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingTop="10dp" >

    <Switch
        android:id="@+id/directionswitch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:enabled="false"
        android:checked="true"
        android:textOff="Counter-ClockWise"
        android:textOn="ClockWise" />
</TableRow>

<TableRow
    android:id="@+id/tableRow6"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingTop="10dp" >

    <TextView
        android:id="@+id/stepperspeed_txv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/stepperspeed"
        android:textAppearance="?android:attr/textAppearanceLarge" />
</TableRow>

<TableRow
    android:id="@+id/tableRow7"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingTop="10dp" >

    <LinearLayout
        android:orientation="horizontal"
        android:weightSum="10" >
```

```
<Button
    android:id="@+id/minus_btn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:enabled="false"
    android:onClick="minusButtonClick"
    android:text="-" />

<EditText
    android:id="@+id/stepperspeed_tb"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="8"
    android:enabled="false"
    android:gravity="center"
    android:inputType="number" >
</EditText>

<Button
    android:id="@+id/plus_btn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:enabled="false"
    android:onClick="plusButtonClick"
    android:text"+" />
</LinearLayout>
</TableRow>

<TableRow
    android:id="@+id/tableRow8"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingTop="10dp" >

<TextView
    android:id="@+id/sensorspeed_txv"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/sensorspeed"
```

```
        android:textAppearance="?android:attr/  
        textAppearanceLarge" />  
    </TableRow>  
  
    <TableRow  
        android:id="@+id/tableRow9"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:paddingTop="10dp" >  
  
        <TextView  
            android:id="@+id/sensorspeedresult_txv"  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:gravity="center"  
            android:text="0"  
            android:textAppearance="?android:attr/  
            textAppearanceLarge" />  
    </TableRow>  
    </TableLayout>  
    </TableRow>  
    </TableLayout>  
</ScrollView>
```

Fichero **MainActivity.java**:

```
package com.example.motorcontroller;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Handler;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.Switch;
import android.widget.TextView;

public class MainActivity extends Activity {

    public final int DEFAULT_SEND_PORT = 8888;
    public final int DEFAULT_RECEIVE_PORT = 6000;
    public final String DEFAULT_LOCAL_IP = "192.168.1.177";

    public UDP_Client Client = new UDP_Client();
    public UDP_Server Server = new UDP_Server();
    public int currentVelocity = 120;
    public String currentAndroidIP = "0.0.0.0";

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

        EditText stepperspeed_tb = (EditText)
            findViewById(R.id.stepperspeed_tb);
        stepperspeed_tb.setText("0");

        EditText androidip_tb = (EditText)
            findViewById(R.id.androidip_tb);
        currentAndroidIP = Utils.getIPAddress(true);
        androidip_tb.setText(currentAndroidIP);
    }
}
```

```
EditText controllerip_tb = (EditText)
    findViewById(R.id.controllerip_tb);
controllerip_tb.setText(DEFAULT_LOCAL_IP);

Switch onoffswitch = (Switch)
    findViewById(R.id.onoffswitch);
onoffswitch.setOnCheckedChangeListener(new
    CompoundButton.OnCheckedChangeListener() {
public void onCheckedChanged(CompoundButton buttonView,
    boolean isChecked) {
    EditText stepperspeed_tb = (EditText)
        findViewById(R.id.stepperspeed_tb);
    Button minus_btn = (Button)
        findViewById(R.id_MINUS_BTN);
    Button plus_btn = (Button) findViewById(R.id.PLUS_BTN);
    Switch directionswitch = (Switch)
        findViewById(R.id.directionswitch);
    if(isChecked){
        stepperspeed_tb.setEnabled(true);
        plus_btn.setEnabled(true);
        minus_btn.setEnabled(true);
        directionswitch.setEnabled(true);
        stepperspeed_tb.setText(Integer
            .toString(currentVelocity));

        SendMessage("v"+currentVelocity);
        SendMessage("1");
    }
    else{
        stepperspeed_tb.setEnabled(false);
        plus_btn.setEnabled(false);
        minus_btn.setEnabled(false);
        directionswitch.setEnabled(false);
        stepperspeed_tb.setText("0");

        SendMessage("v"+currentVelocity);
        SendMessage("0");
    }
}
});

Switch directionswitch = (Switch)
```

```
        findViewById(R.id.directionswitch);
directionswitch.setOnCheckedChangeListener(new
    CompoundButton.OnCheckedChangeListener() {
        public void onCheckedChanged(CompoundButton
            buttonView, boolean isChecked) {
            if(isChecked){
                SendMessage("cd1");
            }
            else{
                SendMessage("cd0");
            }
        }
    });
}

public void checkButtonClick(View view) {

    EditText status_tb = (EditText)
        findViewById(R.id.status_tb);
    EditText controllerip_tb = (EditText)
        findViewById(R.id.controllerip_tb);
    String response =
        Utils.ping(controllerip_tb.getText()
            .toString());
    if(response.contains("Unreachable")){
        status_tb.setText("UNREACHABLE");
        status_tb.setTextColor(Color.RED);
    }
    else if(response.contains("0 received")){
        status_tb.setText("OFFLINE");
        status_tb.setTextColor(Color.RED);
    }
    else{
        status_tb.setText("ONLINE");
        status_tb.setTextColor(Color.GREEN);

        Server.runUdpServer(mHandler,
            mUpdateResults, 6000);
    }
}

public void setSensorSpeedInView(String speed){
```

```
        TextView sensorspeedresult_txv = (TextView)
            findViewById(R.id.sensorspeedresult_txv);
        sensorspeedresult_txv.setText(speed);
    }

    final Handler mHandler = new Handler();

    final Runnable mUpdateResults = new Runnable() {
        public void run() {
            setSensorSpeedInView(UDP_Server.sensorspeed);
        }
    };

    public void minusButtonClick(View view) {

        EditText stepperspeed_tb = (EditText)
            findViewById(R.id.stepperspeed_tb);
        int velocity = Integer.parseInt(stepperspeed_tb
            .getText().toString());
        if (velocity > 0) {
            velocity--;
            currentVelocity = velocity;
            stepperspeed_tb.setText(Integer
                .toString(velocity));

            SendMessage("v"+velocity);
        }
    }

    public void plusButtonClick(View view) {

        EditText stepperspeed_tb = (EditText)
            findViewById(R.id.stepperspeed_tb);
        int velocity =
            Integer.parseInt(stepperspeed_tb.getText()
                .toString());
        if (velocity < 500) {
            velocity++;
            currentVelocity = velocity;
            stepperspeed_tb.setText(Integer
                .toString(velocity));
    }}
```

```
                SendMessage("v"+velocity);
            }
        }

    public void SendMessage(String message){

        //Send the message
        EditText controllerip_tb = (EditText)
            findViewById(R.id.controllerip_tb);
        String address = controllerip_tb.getText()
            .toString();
        Client.Send(message, address,
                    DEFAULT_SEND_PORT);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the
        // action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The
        // action bar will
        // automatically handle clicks on the Home/Up
        // button, so long
        // as you specify a parent activity in
        // AndroidManifest.xml.
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

}
```

Fichero UDP_Client.java:

```
package com.example.motorcontroller;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

import android.annotation.SuppressLint;
import android.os.AsyncTask;
import android.os.Build;

//http://stackoverflow.com/a/19541474/5147720
public class UDP_Client
{
    private AsyncTask<Void, Void, Void> async_client;
    public String Message;

    @SuppressLint("NewApi")
    public void Send(final String message, final String address,
                     final int port)
    {
        async_client = new AsyncTask<Void, Void, Void>()
        {
            @Override
            protected Void doInBackground(Void... params)
            {
                DatagramSocket ds = null;

                try
                {
                    InetAddress inetAddress =
                        InetAddress.getByName(address);
                    ds = new DatagramSocket();
                    DatagramPacket dp;
                    dp = new DatagramPacket(message.getBytes(),
                                           message.length(), inetAddress, port);
                    ds.send(dp);
                    System.out.println("Message sent: " +
                                       message);
                }
            }
        };
        async_client.execute();
    }
}
```

```
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            if (ds != null)
            {
                ds.close();
            }
        }
        return null;
    }

    protected void onPostExecute(Void result)
    {
        super.onPostExecute(result);
    }
};

if (Build.VERSION.SDK_INT >= 11)
    async_client.executeOnExecutor(
        AsyncTask.THREAD_POOL_EXECUTOR);
else
    async_client.execute();
}

}
```

Fichero UDP_Server.Java:

```
package com.example.motorcontroller;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import android.annotation.SuppressLint;
import android.os.AsyncTask;
import android.os.Build;
import android.os.Handler;

//http://stackoverflow.com/a/19541474/5147720
public class UDP_Server
{
    private AsyncTask<Void, Void, Void> async;
    private boolean Server_running = true;
    public static String sensorspeed;

    @SuppressLint("NewApi")
    public void runUpdServer(final Handler mHandler, final
        Runnable mUpdateResults, final int port)
    {
        async = new AsyncTask<Void, Void, Void>()
        {
            @Override
            protected Void doInBackground(Void... params)
            {
                byte[] lMsg = new byte[8];
                DatagramPacket dp = new DatagramPacket(lMsg,
                    lMsg.length);
                DatagramSocket ds = null;

                try
                {
                    ds = new DatagramSocket(port);

                    while(Server_running)
                    {
                        ds.receive(dp);

                        String sensorSpeed = new String(lMsg,
                            "UTF-8");
                    }
                }
            }
        };
    }
}
```

```
        System.out.println(sensorSpeed);
        sensorSpeed = sensorSpeed;

        mHandler.post(mUpdateResults);
    }
}

catch (Exception e)
{
    e.printStackTrace();
}
finally
{
    if (ds != null)
    {
        ds.close();
    }
}

return null;
}
};

if (Build.VERSION.SDK_INT >= 11)
    async.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
else
    async.execute();
}

public void stop_UDP_Server()
{
    Server_running = false;
}
}
```

Fichero Utils.java

```
package com.example.motorcontroller;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.InetAddress;
import java.net.NetworkInterface;
import java.util.Collections;
import java.util.List;

public class Utils {

    //http://stackoverflow.com/a/13007325/5147720
    public static String getIPAddress(boolean useIPv4) {
        try {
            List<NetworkInterface> interfaces =
                Collections.list(NetworkInterface
                    .getNetworkInterfaces());
            for (NetworkInterface intf : interfaces) {
                List<InetAddress> addrs =
                    Collections.list(intf.getInetAddresses());
                for (InetAddress addr : addrs) {
                    if (!addr.isLoopbackAddress()) {
                        String sAddr = addr.getHostAddress();
                        boolean isIPv4 = sAddr.indexOf(':')<0;

                        if (useIPv4) {
                            if (isIPv4)
                                return sAddr;
                        } else {
                            if (!isIPv4) {
                                int delim = sAddr.indexOf('%');
                                return delim<0 ?
                                    sAddr.toUpperCase() :
                                    sAddr.substring(0,
                                        delim).toUpperCase();
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
        } catch (Exception ex) { }
        return "";
    }

//http://stackoverflow.com/a/14576801/5147720
public static String ping(String url) {
    String str = "";
    try {
        Process process = Runtime.getRuntime()
            .exec("/system/bin/ping -c 3 " +
                  url);
        BufferedReader reader = new
            BufferedReader(new InputStreamReader(
                process.getInputStream()));
        int i;
        char[] buffer = new char[4096];
        StringBuffer output = new StringBuffer();
        while ((i = reader.read(buffer)) > 0)
            output.append(buffer, 0, i);
        reader.close();
        str = output.toString();
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    System.out.println("Ping Response "+str);
    return str;
}
}
```