Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Diseño e implementación de un robot cuadrupedo 3GDL con microcontrolador de 32bits STM

Juan Daniel Miret Rubio

Instituto ITACA

Tutor: Juan José Serrano Martín

Trabajo Fin de Master para la obtención de Master en Ingenieria Mecatrónica

Curso académico 2016-2017

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Diseño e implementación de un robot cuadrupedo 3GDL con microcontrolador de 32bits STM

Juan Daniel Miret Rubio

Instituto ITACA

Tutor: Juan José Serrano Martín

Trabajo Fin de Master para la obtención de Master en Ingenieria Mecatrónica

Curso académico 2016-2017

I.RESUMEN

Aunque los vehículos de ruedas son muy eficientes energéticamente, carecen de la posibilidad de moverse en terrenos accidentados. Lo contrario es cierto para las criaturas con patas. La mayoría de ellos son muy ágiles y algunos pueden alcanzar velocidades muy altas. Los guepardos, por ejemplo, pueden correr hasta 120 km / h. Esta tesis de maestría incluía el diseño y programación de un robot cuadrúpedo 3GDL. Donde se imitó la forma de diseño de la naturaleza, también llamada biocinética, y se entró en el asombroso mundo de la robótica. Antes de crear el cuadrúpedo 3GDL se hizo un robot 2GDL para aprender a programar los microcontroladores STM32. Un estudio de literatura del lenguaje de programación C fue necesario para este proyecto. Al diseñar el cuerpo y las piernas, se optó por un robot ligero pero robusto y fabrique cada pieza con una impresora 3D. Además, la soldadura de la electrónica fue una parte de este proyecto, ya que se diseño e implemento àra el robot de 3GDL una PCB JDMR0617 que acoplaba con la placa Nucleo del microcontrolador de STMF334R8. De esta manera las conexiones y la variedad de pines que ofrecia esta PCB fue de gran ayuda para conectar todos los elementos del robot, tales como servomotores, sensores IMU o de distancia etc…Para este diseño se utilizo el Porgrama Altium Designer. El objetivo de este robot era hacerlo capaz de moverse sobre superficies rugosas. En primer lugar, se necesitaba entender el movimiento del robot cuadrúpedo. Para ello, se realizó un estudio exhaustivo de su cinemática y de 4 patas de animales en movimiento, como perros, gatos y caballos. Con este conocimiento se obtuvo como seria el movimiento de las piernas y el cuerpo de cualquier manera posible. A continuación, se investigó en una manera de hacer que el cuadrúpedo se moviera en un terreno irregular. Se hizo esto combinando la cinemática inversa del cuerpo, estas son las cinemáticas que hacen que el cuerpo se mueva en todos los ángulos posibles, con una unidad de medición inercial (IMU). De esta manera, cuando el cuadrúpedo detecta que su cuerpo no es horizontal, puede reajustar su cuerpo de nuevo a una posición horizontal lo que da como resultado una mayor estabilidad y por lo tanto la capacidad de moverse sobre terreno accidentado. Con ello el autor de esta memoria espera que este estudio sirva a todos aquellos que estén interesados en la robótica o quieran hacer su propio cuadrúpedo, el principio básico de la cinemática utilizada y la forma de programar un robot y diseñarlo. Se es consciente que todavía hay un montón de mejoras necesarias para caminar y estabilizarse antes de que incluso se pueda pensar en hacer un robot cuadrúpedo como un vehículo fiable.

II.Prefacio / Reconocimientos

Soy un estudiante de 25 años y en este último curso como ingeniero mecatrónico), decidí hacer mi tesis de maestría en el instituto Tecnológico ITACA. No sólo trabajando en la tesina sino también realizando prácticas de empresa, y posteriormente supliendo en dicha empresa, ha hecho que este curso fuera para mi muy enriquecedor para coger experiencia de cara al futuro.

Este documento es el informe de un proyecto duradero de 5 meses. No puede expresar los largos días pasados en el laboratorio de ITACA, la esperanza de buenos resultados y el cansancio y la tristeza con cada intento fallido. Espero que todos los científicos que consulten este documento para crear su propio robot encuentren las respuestas a sus preguntas.

"El manejo de un robot cuadrúpedo" sólo pudo lograrse gracias al apoyo y la asistencia de mi promotor y también supervisor, Juan José Serrano Martín. Por lo tanto, quisiera agradecerle en particular por proporcionar información indispensable y por evaluar críticamente el texto.

También quiero agradecer a mi novia por apoyarme cada día, y a mis padres por darme una educación que me hizo alcanzar metas como la realización de esta tesis de maestría.

Juan Daniel Miret Rubio.

III.Responsabilidad

Estos resultados pueden ser utilizados por ITACA o su designado, bajo su propia responsabilidad. Sin embargo, ni la UPV, ni sus empleados, ni los autores de esta tesis de maestría, pueden en modo alguno ser responsables de las consecuencias del uso de los resultados de este trabajo, directa o indirectamente. En particular, no hay garantía de que la seguridad esté adecuadamente asegurada sobre la base de este texto y no se garantice el cumplimiento de la legislación.

Valencia, 22/07/17

IV.Descripción de la asignación

La dirección de un robot cuadrúpedo exige un gran conocimiento sobre varios temas de ingeniería. El proyecto abarca temas de esta rama como la mecánica, la electrónica, la informática, la física, la cinemática, la matemática, ... y es por lo tanto un tema perfecto para un trabajo final como ingeniero mecatrónico.

En primer lugar, hay una necesidad de entender cómo los programas que se va a utilizar para llevar este proyecto a un buen final trabajan juntos y cómo usarlos. Los programas relacionados son: STM32CubeMX(programa para la configuración de los periféricos del microcontrolador a utilizar y posteriormente creación del código configurado), Keil Uvision5(programa que se necesita para editar el código predeterminado generado en STM32CubeMX) y STM32 ST-LINK Utility (conecta el microcontrolador al ordenador), SolidWorks, para el diseño de las piezas 3D, Z-SUITE para la impresión de las mismas y Altium para el diseño de PCB. Una vez que los programas se instalen y trabajen juntos correctamente este científico debe mejorar sus habilidades de codificación. Basado en sus habilidades de programación ya adquiridas durante el curso, tendrá que aprender un nivel avanzado de codificación en lenguaje C. Por otra parte, una auto-enseñanza sobre la programación de microcontroladores será necesaria para esta tesis de maestría. Una vez que se haya dominado el lenguaje C y una vez que se tenga suficiente conocimiento de los microcontroladores, podemos comenzar a trabajar en este proyecto.

Para crear el robot cuadrúpedo no sólo será necesario estudiar la marcha de los animales, sino también un estudio profundo de la cinemática. Encontrar las ecuaciones matemáticas, que moverán los servomotores y eventualmente el robot mismo, y la transposición en código será la parte más difícil de este proyecto. El mecanismo será alimentado por una batería y dirigido con un joystick.

En relación con la electrónica, se utilizarán placas nucleo de STM ya que tienen unos micrcontroladores muy potentes con librerias predeterminadas. Mucho mas rapidos que cualquier PIC. Ademas el uso de estos microcontroladores Cortex M abre un abanico de trabajo con otras marcas parecidas a STM que trabajen con placas parecidas tales como ARDUNIO o RASPBERRY. Ademas de esto, se realizará el diseño y la creación de una PCB con el programa de diseño ALTIUM. Esta PCB servirá como acopladora de la STMF334 para tener unas conexiones mas sencillas del robot.
Además, hay una parte mecánica en esta asignación. Es decir, diseñar el robot en SolidWorks, imprimirlo con una impresora 3D y ensamblar los diferentes componentes (servomotores, piezas impresas, tableros, cables, ...). Estas ultimas tareas sera muy enriquecedora para el autor de este proyecto

Finalmente, el robot será probado y mejorado donde sea necesario.

INDICE

Lista de Figuras

Lista de tablas

Simbolos y abreviaturas

ADC Convertidor analogio a digital

CAD Diseño asistido por ordenador

CPU Unidad central de procesamiento

GDL grados de libertad

GPIO Entrada / Salida de uso general

HCLK High-speed Clock - Reloj de alta velocidad

I2C Inter-Integrated Circuit - Circuito inter-integrado

TIC Tecnología de la información y las comunicaciones

IDE Integrated Development Environment - Entorno de desarrollo integrado

IMU Inertial Measurement Unit - Unidad de medida Inercial

ITACA Instituto de Tecnologías de la Información y la Comunicación

LED Light-Emitting Diode - Diodo emisor de luz

MEMS MicroElectroMechanical Systems - Sistemas MicroElectromecánicos

MISO Master In Slave Out - Maestro dentro esclavo fuera

MOSI Master Out Slave In – Maestro fuera esclavo dentro

N/A Not applicable – No aplicable

PCB Printed Circuit Board - Placa de circuito impreso

PLL Phase-Locked Loops

PWM Pulse-Width Modulation - Modulación de ancho de pulso

SCK and SCL Serial Clock - Reloj serie

SD Security Digital card - Tarjeta digital de seguridad

SDA Serial Data - Datos en serie

SPI Serial Peripheral Interface - Interfaz Periférica Serial

SYSCLK System Clock – reloj del sistema

STM STMicroelectronics

TIM Timer

UART Universal Asynchronous Receiver/Transmitter - Receptor / Transmisor Asíncrono Universal

UPV Universidad Politecnica de Valencia

USB Universal Serial Bus - Bus serie universal

USART Universal Synchronous Asynchronous Receiver/Transmitter -

Receptor / transmisor asíncrono síncrono universal

1 Introducción

"Creo que, con el tiempo, la gente tendrá, en cierto modo, relaciones con ciertos tipos de robots - no todos los robots, sino ciertos tipos de robots - en los que podrían sentir que es una especie de amistad, pero va a ser de Un robot de tipo humano. "- Cynthia Breazeal

Como en este proyecto se comparte el mismo sentimiento que Cynthia Breazeal y como este asunto es orientado al futuro y muy interesante, parece un tema perfecto para profundizar. Además, el proyecto abarca muchas materias que se aprenden en el Master de ingeniería Mecatrónica tales como mecánica, electrónica, informática, física, cinemática, matemáticas, ... Demandando este gran conocimiento sobre varios asuntos de ingeniería hace que el "Diseño e implementación de un robot cuadrúpedo "sea un tema perfecto para un trabajo final como ingenieros industriales mecatrónicos.

En esta tesina de master, se descubre cómo se diseñaron, ensamblaron y programaron robots cuadrúpedos con diferentes pasos. Debido a que el conocimiento de la electrónica estaba restringida, se descubrió principalmente la placa STM32F3Discovery y se hizo un robot de 2 GDL. Una vez que se dominó este tipo de robot, se amplió el conocimiento para crear un robot de 3 GDL. Con el fin de mantener una buena visión general, por lo tanto, manteniendo a los dos tipos diferentes de robots separados. A menudo la explicación del robot con 3GDL se basa en la del robot de 2 GDL con el fin de evitar la repetición. Saltar directamente al robot de 3GDL puede causar confusión o dificultad y por lo tanto no se recomienda. Los robots fueron creados para uso didáctico en el laboratorio del Instituto de Tecnologías de la Información y la Comunicación (ITACA), entidad de investigación y desarrollo de la Universidad Politécnica de Valencia (UPV).

Figura 1: Instituto tecnologico ITACA, UPV

La siguiente enumeración da una visión general de los diferentes capítulos de esta tesis.

• Al principio de este trabajo, se agregó una marca teórica para dar al lector interesado alguna información de antecedentes y posibles aplicaciones en el uso de robots.

• Con el fin de crear nuestro robot, primero se instaló y se aprendió a trabajar con el entorno de programación para entender la electrónica y los componentes que se iban a utilizar. Dado que esto fue la base del resto del trabajo, se incluyó un capítulo en el que los componentes usados se enumeran y se explican ampliamente. En cuanto a los programas utilizados, se explican en los capítulos en los que se utilizan.

• Los dos capítulos siguientes contienen el método utilizado para el diseño de los robots. Primero se describieron los diseños mecánicos, luego los diseños electrónicos.

• Cómo configurar el software con el hardware y realizar las conexiones con los cables se puede encontrar en el capítulo 6.

• Se dan algunas imágenes de los diseños montados y cableados de los robots y controladores inalámbricos.

• Antes de que la programación en uVisión5 pudiera comenzar diferentes estudios profundos eran necesarios. En primer lugar, se analizó la cinemática para tener el control total de los robots cuadrúpedos.

• Para el diseño de una PCB hubo que autoenseñarse a utilizar el programa Altium Designer que fue de gran utilidad para este diseño. Para ellos la pagina web de este programa facilitaba unos tutoriales de los pasos a seguir para crear la PCB deseada, y las mejores formas o opciones de hacerlo en este programa informatico.

• Creación de una PCB para nuestra placa base STMF334R8 para obtener las entradas y salidas de forma diferente, para facilitar la conexión de los periféricos del robot tales como servomotores sensor acelerómetro, sensor infrarrojo de distancia, módulo de radiofrecuencia etc..

• Cómo se realizó la calibración de los robots se explica en el siguiente capítulo.

• Se utilizó una IMU para el robot de 3GDL para estabilizarlo en terrenos accidentados.

• Para ver si los servomotores podían soportar el peso de los componentes, se realizó un segundo estudio, un estudio de par.

• Todo lo relacionado con la comunicación del controlador inalámbrico con los robots se puede encontrar en el capítulo 12.

• El tercer estudio fue acerca de los diferentes pasos que se iba a dar a los robots.

• Finalmente, una auto-enseñanza del lenguaje C fue hecha por necesidad. Después de este estudio, la codificación podría comenzar. El código se encuentra en el capítulo 14.

• El último capítulo de esta tesis de maestría consiste en la conclusión que podríamos extraer.

Este informe viene con anexos electrónicos en los que puede encontrar las piezas hechas en SolidWorks y los programas en STM32CubeMX y uVisión5. También se agregaron algunos videos de los robots.

2 Marco teórico

2.1 Historia de los robots

Por siglos el ser humano ha construido máquinas que imiten las partes del cuerpo humano. Los antiguos egipcios unieron brazos mecánicos a las estatuas de sus dioses. Estos brazos fueron operados por sacerdotes, quienes clamaban que el movimiento de estos era inspiración de sus dioses. Los griegos construyeron estatuas que operaban con sistemas hidráulicas, los cuales se utilizaban para fascinar a los adoradores de los templos.

Durante los siglos XVII y XVIII en Europa fueron construidos muñecos mecánicos muy ingeniosos que tenían algunas características de robots.

Jacques de Vauncansos construyó varios músicos de tamaño humano a mediados del siglo XVIII. Esencialmente se trataba de robots mecánicos diseñados para un propósito específico: la diversión.

En 1805, Henri Maillardert construyó una muñeca mecánica que era capaz de hacer dibujos. Una serie de levas se utilizaban como ' el programa ' para el dispositivo en el proceso de escribir y dibujar. Éstas creaciones mecánicas de forma humana deben considerarse como inversiones aisladas que reflejan el genio de hombres que se anticiparon a su época. Hubo otras invenciones mecánicas durante la revolución industrial, creadas por mentes de igual genio, muchas de las cuales estaban dirigidas al sector de la producción textil. Entre ellas se puede citar la hiladora giratoria de Hargreaves (1770), la hiladora mecánica de Crompton (1779), el telar mecánico de Cartwright (1785), el telar de Jacquard (1801), y otros.

El desarrollo en la tecnología, donde se incluyen las poderosas computadoras electrónicas, los actuadores de control retroalimentados, transmisión de potencia a través de engranes, y la tecnología en sensores han contribuido a flexibilizar los mecanismos autómatas para desempeñar tareas dentro de la industria. Son varios los factores que intervienen para que se desarrollaran los primeros robots en la década de los 50's. La investigación en inteligencia artificial desarrolló maneras de emular el procesamiento de información humana con computadoras electrónicas e inventó una variedad de mecanismos para probar sus teorías.

No obstante las limitaciones de las máquinas robóticas actuales, el concepto popular de un robot es que tiene una apariencia humana y que actúa como tal. Este concepto humanoide ha sido inspirado y estimulado por varias narraciones de ciencia ficción.

Una obra checoslovaca publicada en 1917 por Karel Kapek, denominada Rossum's Universal Robots, dio lugar al término robot. La palabra checa 'Robota' significa servidumbre o trabajador forzado, y cuando se tradujo al ingles se convirtió en el término robot.

En 1920 el escritor checoslovaco ***Karel Capek*** *publicó su novela RUR (Rossum's Universal Robots). En la palabra* ***robot*** *significa* ***siervo***, ***fuerza de trabajo****. Así pues, la robótica era un término de ciencia ficción que surgió en este libro.*

En 1926 Metrópolis, de Fritz Lang, es la primera película en la que aparecen robots.

En 1946 se creó la primera computadora electrónica "ENIAC", que estaba construida a base de válvulas.

Por lo general, hasta mediado de siglo los robots eran servomecanismos teleoperados, es decir, controlados por un ser humano. En este sentido, no disponían de sistema de control propio.

Primeras implementaciones

En 1948, el neurólogo Grey Walter construye los primeros robots móviles utilizando el sentido común y algo de bricolaje, que se llamaron las tortugas de Bristol. Estaban construidos utilizando válvulas, sensores de luz y detectores de contacto. Tenían dos ruedas motrices y un foto-tubo como ojo. Se recarga al detectar descarga.

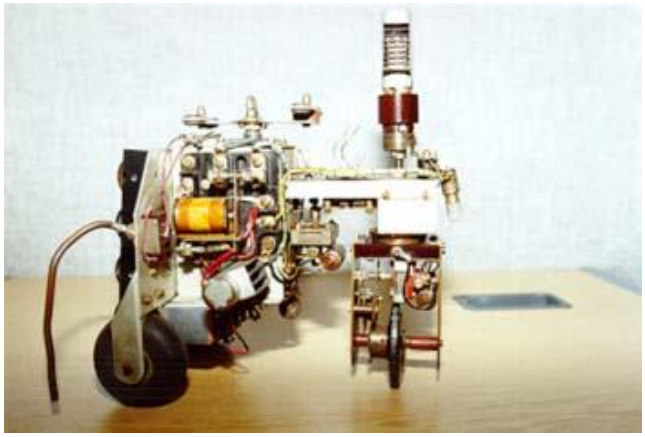


Figura 2: La tortugas de Bristol

En 1961 se instala en una planta de General Motors el primer robot de Unimate. Fue utilizado para la manipulación de material en una máquina de fundición. En 1978 Unimation desarrolla el robot ***Puma*** *(Programmable Universal Machine for Assembly) con el apoyo de General Motors.*

En 1970 se desarrolla en la Universidad de Stanford el robot Shakey.

En Standford, Nils Nilsson desarrolló el robot móvil SHAKEY en 1969. Este robot estaba equipado con un láser de telemetría, una cámara y sensores táctiles, encontrándose conectado a un ordenador DEP PDP 10 vía radio. Entre las tareas de SHAKEY se encontraban tanto la evasión de obstáculos como el movimiento de objetos dentro de un entorno altamente estructurado. Todos los obstáculos eran simples bloques y cuñas uniformemente coloreados. SHAKEY
mantenía una lista de los objetos de su entorno y, usando un teorema de resolución por sondeo llamado STRIPS, determinaba planes de acción que posteriormente ejecutaba.

En el año 2000, Honda sorprende al mundo entero con el lanzamiento del robot Asimo, (acrónimo de "Advanced Step in Innovative Mobility"- paso avanzado en movilidad innovadora), es un robot humanoide que pretende ayudar a las personas que carecen de movilidad completa en sus cuerpos, así como para animar a la juventud para estudiar ciencias y matemáticas.

En 2004, el robot Spirit es el primero de los dos robots que forma parte del Programa de Exploración de Marte de la NASA. El robot 3 aterrizó con éxito el 3 de enero de 2004 y su gemelo Opportunity aterrizó con éxito en Marte el 24 de enero de 2004.

Figura 3: El robot Asimo

Hoy en día los robots son un imaginablemente importante en nuestras vidas, ya que, en la mayoría de los casos, sin su existencia muchas cosas no serían posibles de lograr.

2.2 Varias aplicaciones en el uso de robots

Uno de los campos más importantes en los que se integran los robots es la fabricación industrial. Se utilizan en todo el mundo para satisfacer las necesidades de producción y aumentar la calidad. Varias aplicaciones son: robots de paletización, aplicaciones de picking y embalaje, robots de montaje, ...

Una de las utilidades más significativas de los robots es que son capaces de acceder a lugares que son demasiado peligrosos para los seres humanos. Piense en los reactores nucleares que se derritieron en una planta nuclear en Fukushima, Japón en 2011. Dado que el medio ambiente no era adecuado para los seres humanos para acceder y limpiar el daño, había una flota de robots que se montó. También para otras catástrofes como terremotos o incendios, los robots pueden ser de gran ayuda y asegurarse de que las vidas humanas no estén en peligro.

Los robots también se pueden utilizar para realizar tareas domésticas. El ejemplo más conocido aquí es la aspiradora robótica. Los robots del futuro probablemente no solo limpiarán nuestras casas, sino que también cocinarán para nosotros, trabajarán en el jardín, lavarán nuestra ropa, ... Dado que muchos juguetes no interactúan con su entorno, los juguetes de robot lo llevan al siguiente nivel. En este campo, los robots son simples y no tienen que cumplir con un montón de especificaciones que los hacen relativamente baratos. Perro robots de juguete, drones, ... pertenecen a esta categoría.

Además, el área de la atención sanitaria está en aumento. Los sistemas de asistencia y rehabilitación ayudan a los seres humanos en tareas complejas, agotadoras y a menudo repetidas. Este apoyo puede tener lugar por robots de servicio realizando la tarea respectiva pero también utilizando sistemas que el humano está usando como exoesqueletos u ortesis.

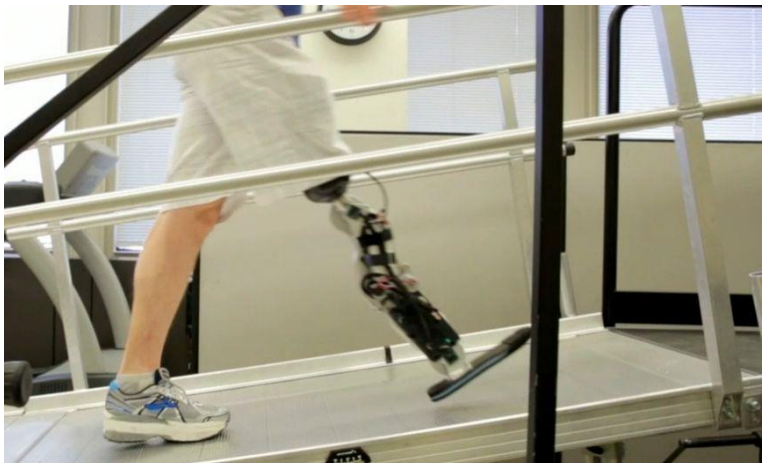


Figura 5: Pierna robótica

Sin la existencia de los robots no podríamos explorar otros planetas. La misión Mars Exploration Rover (MER) de la NASA es un buen ejemplo de esto. La misión espacial robótica en curso involucró a dos exploradores de Marte, Spirit y Opportunity, que estaban explorando el planeta. Hoy en día sólo Opportunity sigue activo. Para estos entornos extraterrestres, los sistemas de robots deben desarrollarse para terrenos irregulares basados en conceptos innovadores de locomoción de inspiración biológica.

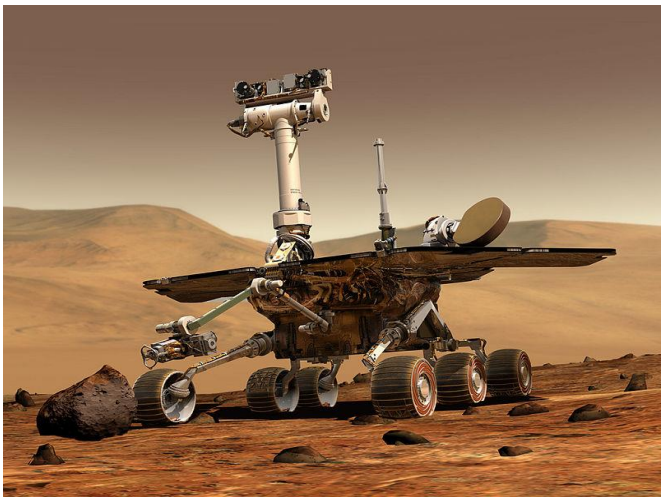


Figura 4: NASA Mars Exploration Rover

3 Selección de los componentes

Con el fin de comprender los capítulos siguientes en este documento, una descripción y explicación de los componentes utilizados se da.

3.1 Robot de 2GDL y Mandos inalámbricos: STM32F3DISCOVERY junto con ITACA-RIS RMR 3815

La placa STM32F3DISCOVERY es en combinación con ITACA-RIS RMR 3815 la parte más importante de los cuadrúpedos de 2GDL y su control inalámbrico. Es una completa plataforma de demostración y desarrollo para STMicroelectronics ARM Cortex-M4 basado en el núcleo del microcontrolador STM32F303VCT6.

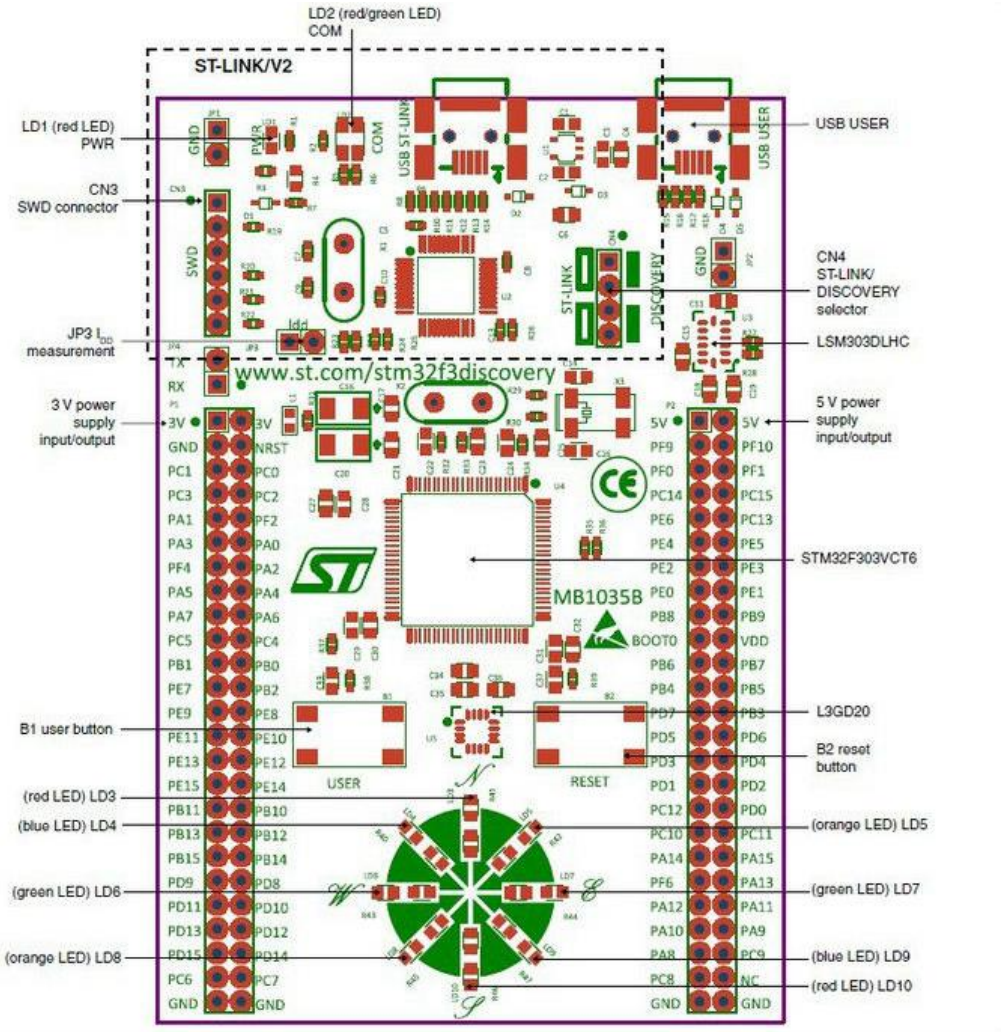


Figura 5: Layout de la placa STM32F3DISCOVERY

El kit DISCOVERY incluye una interfaz de herramienta de depuración integrada ST-LINK / V2, un giroscopio ST MEMS, una brújula electrónica ST MEMS, LEDs, pulsadores y un conector USB Mini-B y se coloca en una PCB (placa de circuito impreso) Creada por ITACA,

denominada ITACA-RIS RMR 3815, realizado para obtener las entradas y salidas en una forma diferente, para facilitar la conexión entre otros a los servomotores. Facilitar las conexiones no es la única función de la PCB. También hay entradas para la alimentación de los servomotores y pulsadores incluidos en la placa ITACA-RIS RMR 3815.

La figura siguiente muestra la configuración de las tarjetas electrónicas.

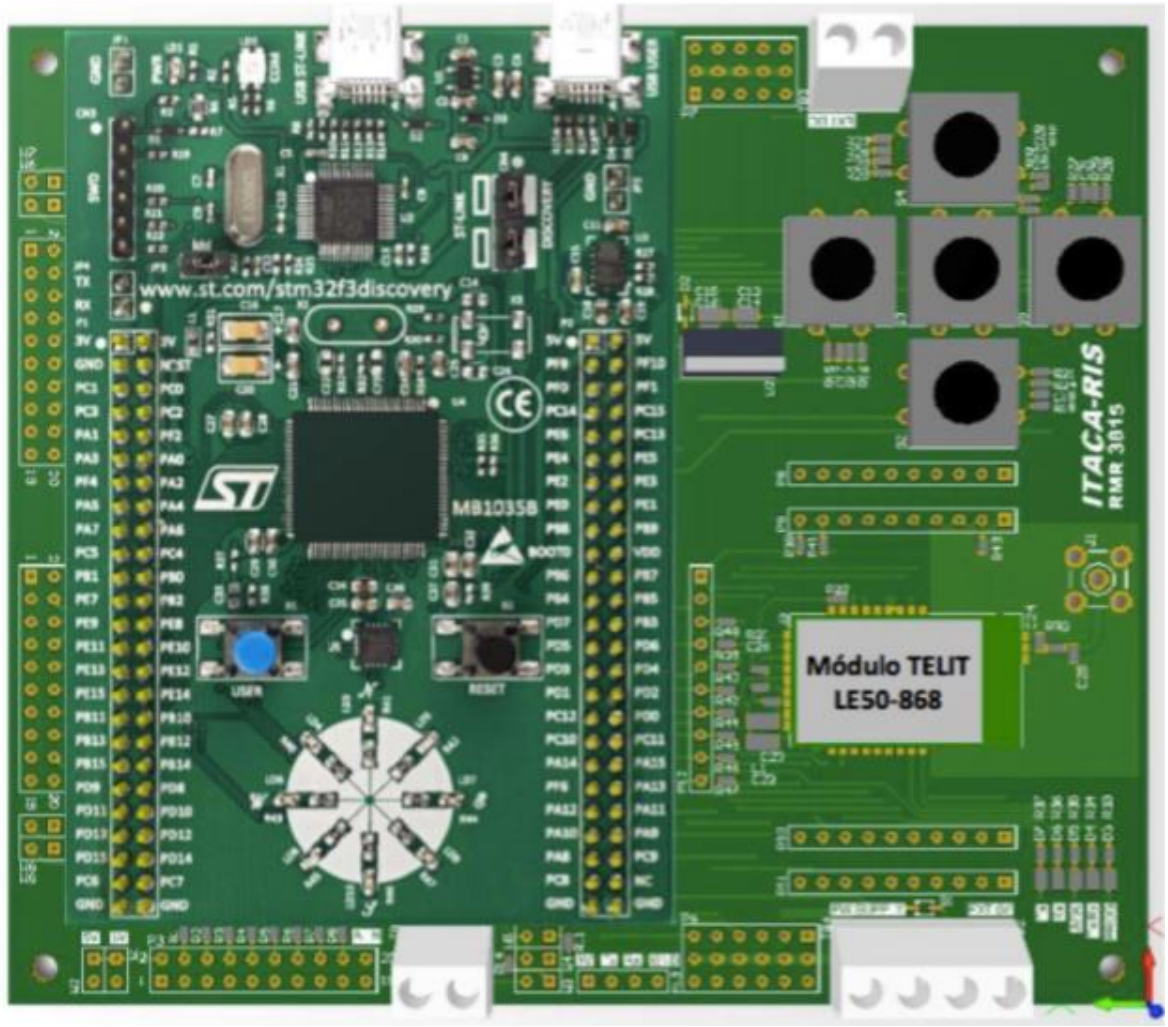


Figura 6: PCB ITACA-RIS RMR 3815 utilizada para el cuadrúpedo de 2GDL y para el mando.

También utilizamos esta configuración para el control inalámbrico pudiendo dirigir el robot a cierta distancia.

3.2 Robot 3 DOF: El escudo NUCLEO-F334R8, PCB JDMR0617 y STEVAL-MKI124V1

Para el cuadrúpedo de 3GDL utilizamos una placa Nucleo-32, la NUCLEO-F334R8. Esta placa tiene un microcontrolador STM32 en el paquete LQFP64. Al igual que la STM32F3DISCOVERY, nuestra tarjeta STM32 Núcleo no requiere sonda independiente, ya que integra el depurador y programador ST-LINK / V2-1. Los encabezados ST Morpho y el soporte de conectividad Arduino Uno V3 facilitan la ampliación de la funcionalidad de la plataforma de desarrollo abierto con una amplia variedad de escudos especializados o PCB's. El apilable escudo que utilizamos la PCB JDMR0617 diseñada por el autor del proyecto y fabricado en el exterior) tiene la misma función incluso que el PCB, ITACA-RIS RMR 3815, del robot de 2 GDL y por lo tanto es indispensable. Además de la conexión de los servomotores, también llevará pines para UART y para un chip llamado STEVAL-MKI124V1. Se necesita este chip ya que contiene entre otras cosas un giroscopio-brújula electrónica ST MEMS ST MEMS que hace posible caminar sobre superficie rugosa.

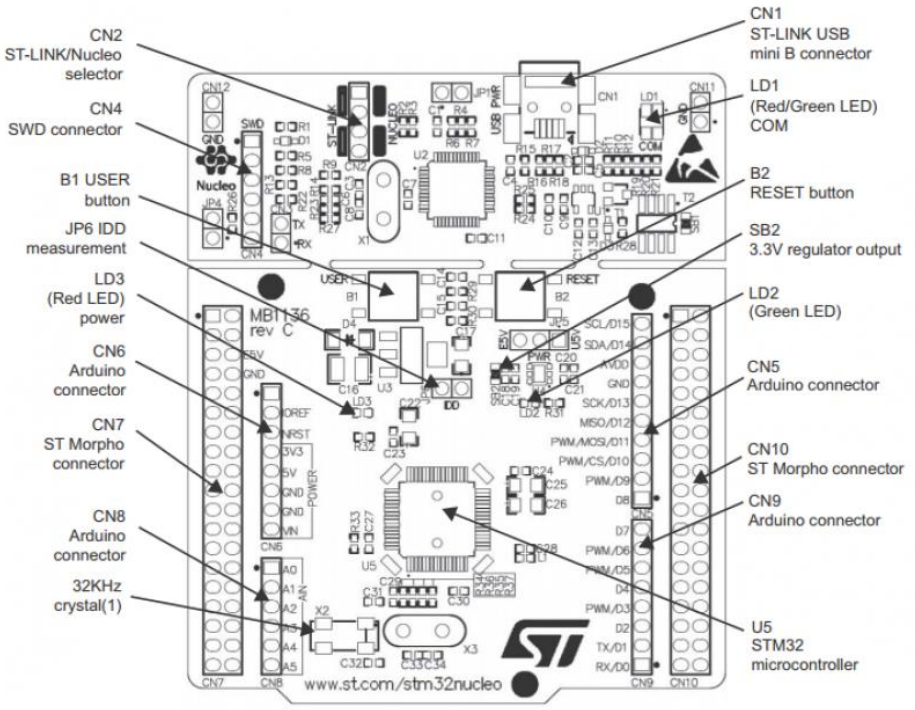
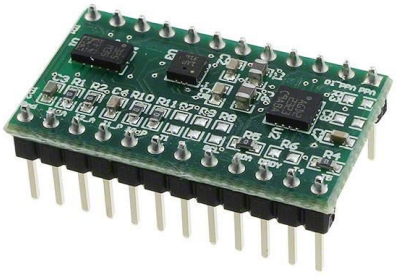
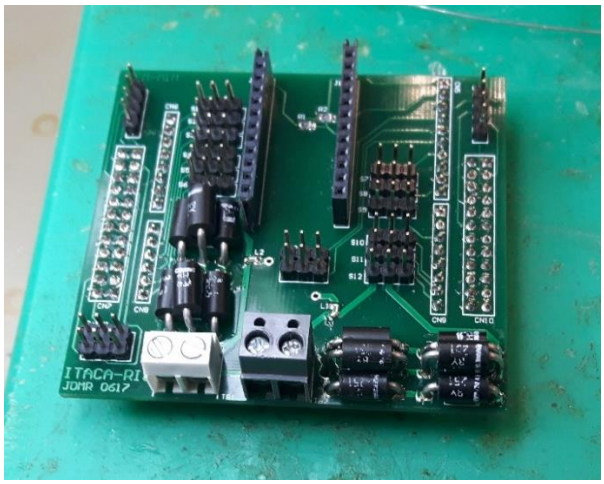
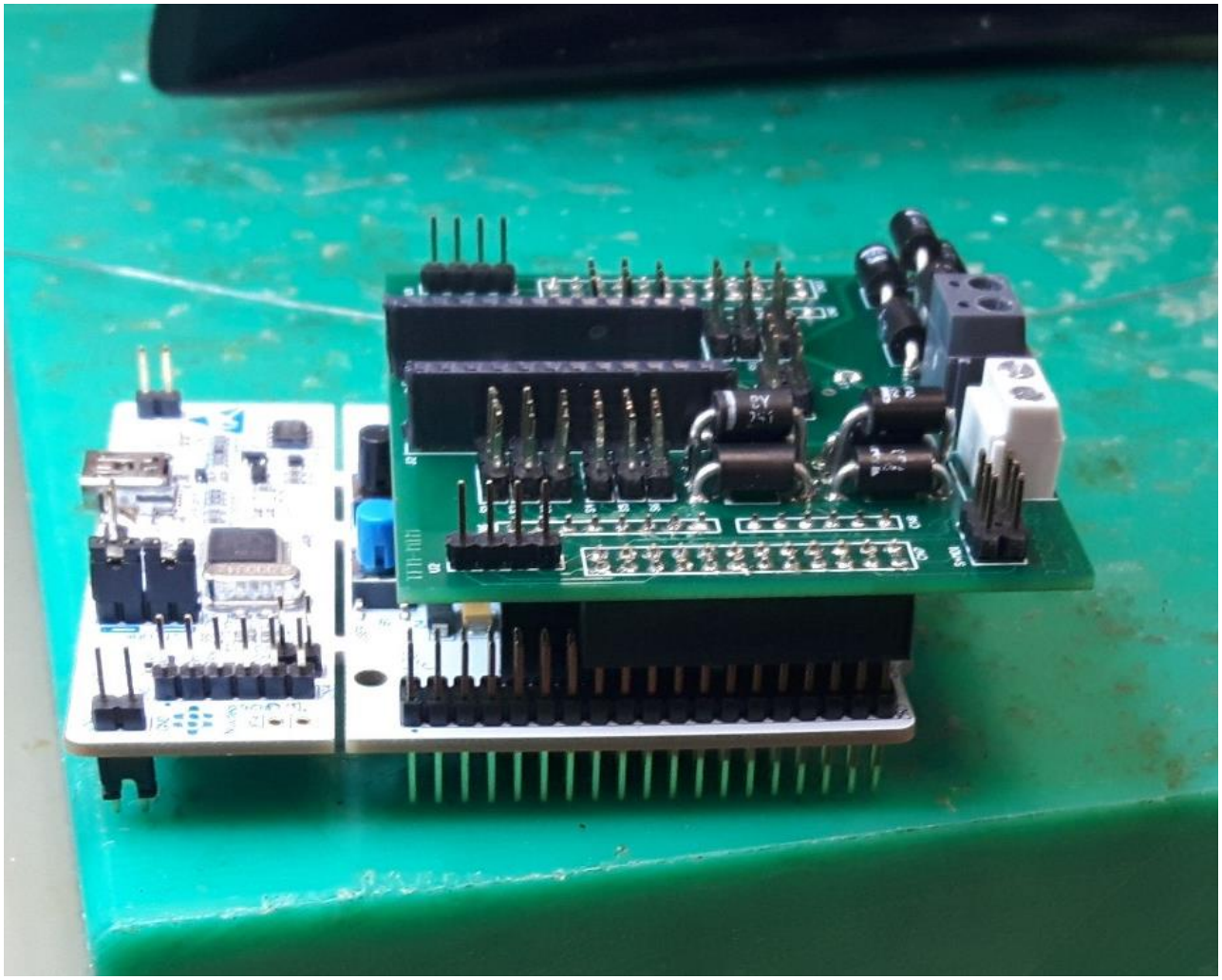


Figura 7: Disposición de la placa STMF334R8







Figuras 8 9 y 10: STEVAL-MKI124V1 - PCB JDMR 0617- *Bloque electrónico del cuadrúpedo 3GDL*

3.3 Servomotores

Los servos usados están compuestos por un motor eléctrico mecánicamente unido a un potenciómetro. La electrónica esta compuesta por un microcontrolador y dos chips semipuente en H. Esta estructura del servo traduce la anchura del pulso en una posición determinada. Cuando se ordena que el servo gire, el motor es accionado hasta que el potenciómetro alcance el valor correspondiente a la posición ordenada.

Los motores que utilizamos son los motores DM-RS0613MD (para el 2GDL) y DM-RS1513MD (para el de 3GDL) creados por DOMAN RC Hobby. Estos motores pueden girar 270 ° (PWM500-2500us). El par que estos motores pueden ofrecer son para el DM-RS0613MD entre 6.5kg.cm (6V) y 7.5kg.cm (7.4V) y para el DM-RS1513MD entre 14.4kg.cm (6V) y 16.5kg.cm (7,4 V). Ambos tipos tienen un voltaje de funcionamiento de 5.5V ~ 7.4V. Es muy importante no superar los 7.4V, esto quemará la electrónica y dañará los servos. Bajo 5.5V los servos no se moveran. Se pueden encontrar más especificaciones de estos servomotores en el sitio web de DOMAN RC Hobby.

Figuras 11 y 12: robot servo motor DM-RS0613MD y DM-RS1513MD

características:

-bajo Perfil digital servo con engranajes de titanio 25 T servo, 2BB
-diseño especial para 17DOF robot Humanoide,
-Voltaje De funcionamiento: 6 V ~ 7.4 V
-anchura de banda muerta: 2usc
-interfaz: (JR)
-longitud del cable: 30 cm

especificaciones:

N ° del artículo: DM-RS1513MD
peso: 51g
tamaño: 44mm X 23mm X 26.5mm
velocidad: 0.25sec/60deg (6 v) 0.23sec/60deg (7.4 v)
Par máximo: 14.4kg. cm (6 v) 16.5kg.cm (7.4 v)
(nota: necesita UBEC si utilizar 2 S Lipo, 2 S Lipo max es 8.4 v, se quemará el servo)
Voltaje De funcionamiento: 6-7.4Volts
270 grados de rotación (PWM500-2500us)
Futaba, JR, SANWA, Hitec compatibles.

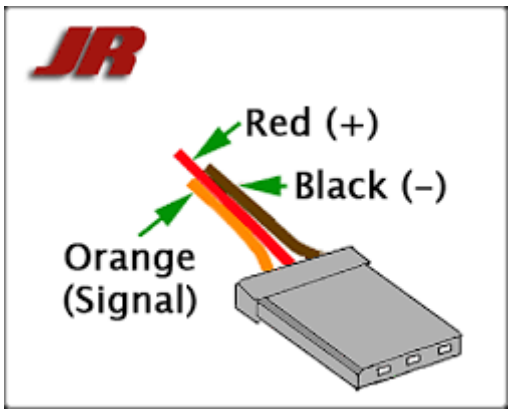
tamaño (mm)			peso		alambre	6 V			7.4 V			ángulo de rotación
						velocidad	Torque		velocidad	Torque		
L	W	H	g	oz	cm	sec/60°	$ number kg-cm	oz-en	sec/60°	$ number kg-cm	oz-en	
44	23	26.5	51.0		30.0	0.25	14.4		0.23	16.5		270 grados (PWM500-2500us),

el embalar incluye:

1pcsXDM-RS1513MD servo
Accessaries (cada uno):
1 x servo redonda
1 x largo u-brazo Marco
1 x corto u-brazo Marco
1 x cojinete de brida
1 x lavadora
7 x tornillos de montaje (2.6x6 5 unids, 2x6 2 unids)
1 x tornillo de montaje M3 x 8mm para servo cuerno

Notas importantes:

1, el tipo de conector es JR.
2, Consulte la figura al instalar los accesorios servos. conecte la U cuadro primero, luego instal en servo,
3, Para aviones propulsados por motores y barcos, de goma debe ser utilizado para reducir la vibración.
4, Por Favor elegir el modelo correcto para su aplicación.
5, esfuerzo de Torsión sobrecargado dañarán el mecanismo de servo-.
6, Mantenga el servo limpio y libre de polvo, gas corrosivo y aire húmedo.
7, elija por favor el voltaje de alimentación correcto y asegúrese de que la corriente



Los servomotores utilizan un conector de tipo JR universal (un conector de 3 pines):

- Rojo: Alimentación + 5.5V ~ 7.4V
- Marrón / negro: Tierra
- Naranja: Señal (PWM)

Figuras 13 y 14: Hoja de características de los motores - Conector Universal tipo JR

Figura 15: Robot humanoide de 17GDL construido con servomotores DOMAN 1513MD

3.4 Joystick

Para mover y controlar los robots, se utilizó un SunFounder Joystick PS2 Module de dos ejes XY para Arduino, Raspberry Pi etc... Este joystick se conectó al controlador inalámbrico. También se utilizaron la botonera en el PCB de ITACA RIS RMR3815 para maniobrar.

El joystick contiene 5 pines:

Y (eje)

X (eje)

B (Botón pulsador)

VCC (+ 3,3V)

GND (tierra).

Figura 16: Joystick

3.5 Comunicación inalámbrica

Para la comunicación inalámbrica del joystick con el microcontrolador del robot, utilizamos un módulo inalámbrico: el SP1ML-SUB 1GHz MODULE de STMicroelectronics . El protocolo UART, receptor / transmisor asíncrono universal, es responsable de la ejecución de las tareas principales en la comunicación serial. El módulo SP1ML está recibiendo datos en serie de la placa STM32F3, que transmite a otro SP1ML usando RF. El segundo SP1ML transmite estos datos a la placa STM del cuadrúpedo. El protocolo UART realiza todas las tareas necesarias, tales como cronometraje, comprobación de paridad, etc. que son necesarias para la comunicación. En otras palabras, el UART permite enviar los datos de un módulo y recibir los datos en el otro módulo.

El Módulo de radiofrecuencia SP1ML es un módulo de RF ultra-bajo y totalmente integrados que funciona respectivamente en las bandas 868 MHz SRD y 915 MHz ISM. Es un módulo de tamaño compacto, que integra una antena de a bordo y microcontrolador interno con una interfaz fácil de usar, permitiendo a los usuarios agregar fácilmente conectividad inalámbrica en diseños sin necesidad de experiencia en RF profunda, y tiene todas las aprobaciones FCC modulares y CE Cumplimiento.

Este módulo va soldado íntegramente en la PCB de ITACA RIS RMR 4616(SP1ML- ADAPTER) de pequeño tamaño diseñada especialmente para este tipo de chips de radiofrecuencia (también válida para módulos de TELIT), para conectar de forma fácil los pines de la UART TX (Transmisor), RX(Receptor), VCC y GND a las respectivas placas STM32F3.

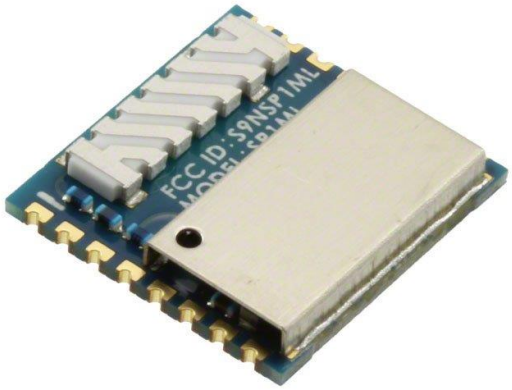


Figura 17: Modulo de radiofrecuencia SP1ML de STM

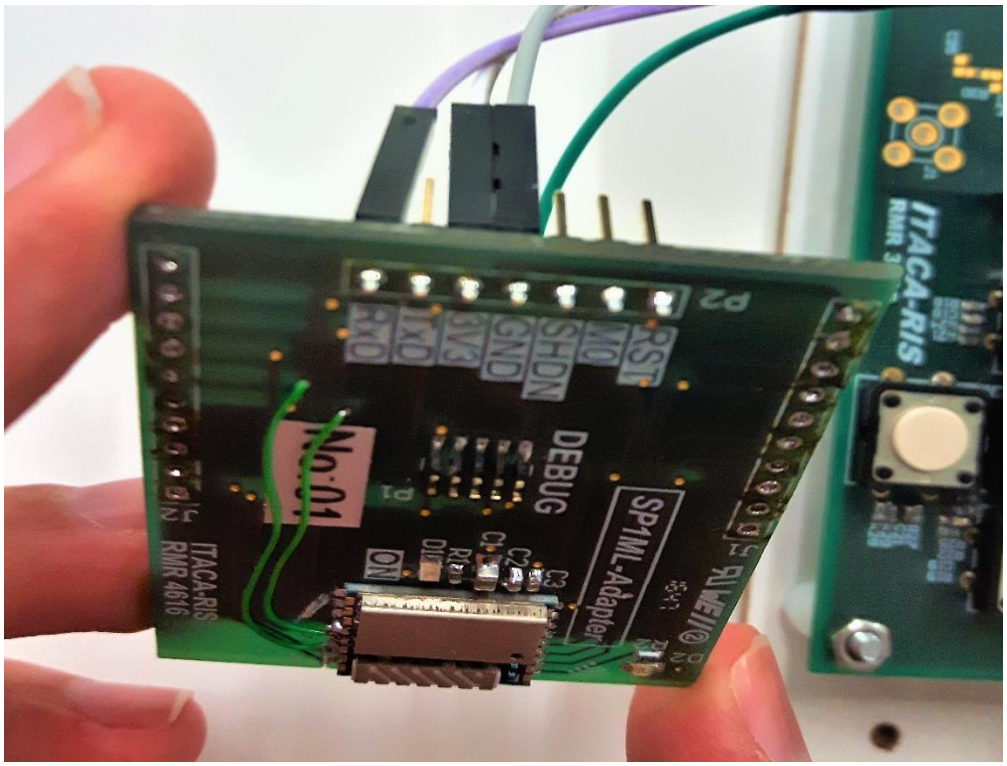


Figura 18: PCB ITACA-RIS RMR 4616 SP1ML- ADAPTER

3.6 Adaptador USB hyperterminal

Para mostrar valores en la computadora (en un HyperTerminal) se utilizó un Adaptador USB a Serial de Prolific. El adaptador contiene 5 pins: + 3,3V, TXD (Transmisor), RXD (Receptor), GND (Tierra) y + 5V. La conexión de los pines a las conexiones UART de nuestra placa hizo posible la visualización de la salida. Para trabajar a cierta distancia de nuestros robots, usamos un cable de extensión USB.

Figura 19: hiperterminal USB más cable de extensión

3.7 Baterías

Para tener una conexión inalámbrica real con los robots, utilizamos baterías recargables (Gens ace 1800mAh 7.4V 40C 2S1P Lipo Battery Pack y una PowerBank). Para el robot de 2GDL, se utilizó uno para suministrar los servos de 5.5V ~ 7.4V. La otra batería era necesaria para suministrar nuestros tableros (tablero STM32F3DISCOVERY y ITACA-RIS RMR 3815) de 5V. El robot de 3GDL sólo está equipado con una batería, el Gens ace 1800mAh 7.4V 40C 2S1P Lipo batería Pack. Para suministrar nuestra tarjeta de corriente Núcleo, se utilizó de primera hora un módulo de fuente de alimentación. Pero más adelante se analizo que no era necesario este módulo. La Power Bank se utilizó para alimentar el mando de control a distancia.

Figura 20: Baterias

3.8 Módulo Buck (NO UTILIZADO) y circuito regulador Integrado en STM32F334R8

En el cuadrúpedo 3DOF vimos que no hacía falta un módulo de alimentación. El módulo LM2596 DC-DC Buck es un regulador de tensión que ofrece la posibilidad de ajustar el voltaje de entrada en un rango de 1.25V DC a 35V DC. Ajustando la perilla del potenciómetro azul podemos aumentar (hacia la derecha) y bajar (a la izquierda) la tensión de salida a nuestras necesidades. Para monitorear esto, usamos un multímetro. La tensión requerida para nuestra placa Núcleo fue de 5V.

Figura 21: Módulo BuckLM2596 DC-DC

Viendo el manual de nuestra placa STM32F334R8 se vio que era innecesario este módulo suplementario de fuente de alimentación ya que los 7V que venían de la batería podía ser conectados directamente a Vin en nuestra placa ya que ella se encargaba mediante un circuito interno de regular hasta 12V para que el funcionamiento del microcontrolador le llegaran los 5 y 3.3 Voltios necesarios para su correcto funcionamiento.

power capability.

6.3.1 Power supply input from the USB connector

The ST-LINK/V2-1 supports USB power management allowing to request more than 100 mA current to the host PC.

All parts of the STM32 Nucleo board and shield can be powered from the ST-LINK USB connector CN1 (U5V or VBUS). Note that only the ST-LINK part is power supplied before the USB enumeration as the host PC only provides 100 mA to the board at that time. During the USB enumeration, the STM32 Nucleo board requires 300 mA of current to the host PC. If the host is able to provide the required power, the targeted STM32 microcontroller is powered and the red LED LD3 is turned ON, thus the STM32 Nucleo board and its shield can consume a maximum of 300 mA current, not more. If the host is not able to provide the required current, the targeted STM32 microcontroller and the MCU part including the extension board are not power supplied. As a consequence the red LED LD3 remains turned OFF. In such case it is mandatory to use an external power supply as explained in the next *Section 6.3.2: External power supply inputs: VIN and E5V*.

When the board is power supplied by USB (U5V) a jumper must be connected between pin 1 and pin 2 of JP5 as shown in *Table 8*.

JP1 is configured according to the maximum current consumption of the board when powered by USB (U5V). JP1 jumper can be set in case the board is powered by USB and maximum current consumption on U5V does not exceed 100 mA (including an eventual extension board or Arduino shield). In such condition USB enumeration will always succeed since no more than 100mA is requested to the PC. Possible configurations of JP1 are summarized in *Table 6*.

Table 6. JP1 configuration table

Jumper state	Power supply	Allowed current
JP1 jumper OFF	USB power through CN1	300 mA max
JP1 jumper ON		100 mA max

Warning: **If the maximum current consumption of the NUCLEO and its extension boards exceeds 300 mA, it is mandatory to power the NUCLEO using an external power supply connected to E5V or VIN.**

Note: *In case the board is powered by an USB charger, there is no USB enumeration, so the led LD3 remains set to OFF permanently and the target STM32 is not powered. In this specific case the jumper JP1 needs to be set to ON, to allow target STM32 to be powered anyway.*

6.3.2 External power supply inputs: VIN and E5V

The external power sources VIN and E5V are summarized in the *Table 7*. When the board is power supplied by VIN or E5V, the jumpers configuration must be the following:

- Jumper on JP5 pin 2 and pin 3
- Jumper removed on JP1

Table 7. External power sources

Input power name	Connectors pins	Voltage range	Max current	Limitation
VIN	CN6 pin 8 CN7 pin 24	7 V to 12 V	800 mA	From 7 V to 12 V only and input current capability is linked to input voltage: 800 mA input current when VIN=7 V 450 mA input current when 7 V<VIN (< or =) 9 V 250 mA input current when 9 V<VIN (< or =) 12 V
E5V	CN7 pin 6	4.75 V to 5.25 V	500 mA	-

Table 8. Power-related jumper

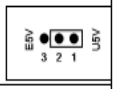
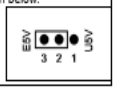
Jumper	Description
JP5	U5V (ST-LINK VBUS) is used as power source when JP5 is set as shown below (Default setting) 
	VIN or E5V is used as power source when JP5 is set as shown below. 

Figura 22: Circuito interno regulador de hasta 12V

3.9 Medidor de distancia SHARP GP2D12

El medidor de distancia se ha utilizado para que los robots pudieran visualizar obstáculos en su dirección y evitar que chocaran con ellos, en caso de que la persona que estuviera al control del robot se pudiera despistar en cualquier momento y mandarle ordenes perjidiciales para la integridad del cuadrúpedo.

El sensor utilizado ha sido un SHARP GP2D12. Este es un sensor medidor de distancias por infrarrojos que indica mediante una salida analógica la distancia medida. La tensión de salida varia de forma no lineal cuando se detecta un objeto en una distancia entre 10 y 80 cm. La salida esta disponible de forma continua y su valor es actualizado cada 32 ms. Normalmente se conecta esta salida a la entrada de un convertidor analógico digital el cual convierte la distancia en un numero que puede ser usado por el microprocesador. La salida también puede ser usada directamente en un circuito analógico. Hay que tener en cuenta que la salida no es lineal. El sensor utiliza solo una línea de salida para comunicarse con el procesador principal. El sensor se entrega con un conector de 3 pines. Tensión de funcionamiento 5V, Temperatura funcionamiento: -10 a 60ºC, Consumo Medio: 35 mA. Margen de medida 10cm a 80 cm.

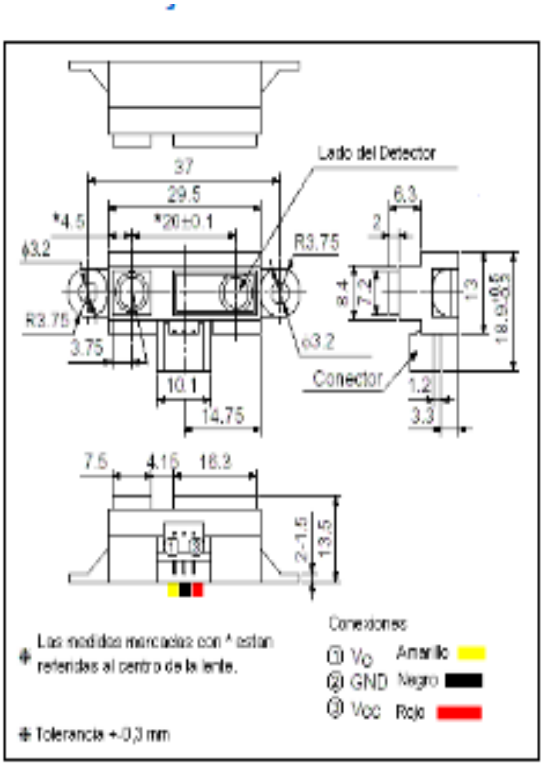


Figura 23: hoja de características mas imagen del SHARP GP2D12

3.9.1 Miniservomotores

Estos miniservomotores serán utilizados para que el medidor de distancia puedo girar en la dirección en la que el robot se mueva en cada momento. Estos servomotores al ser mas pequeños y menos potentes que los de las patas del robot van conectados a un voltaje no superior a 5 V si no se quemarían. Por tanto, la señal de 5 V se adquiere no de la batería si no de la salida de 5 V que ofrece la placa F334 R8.

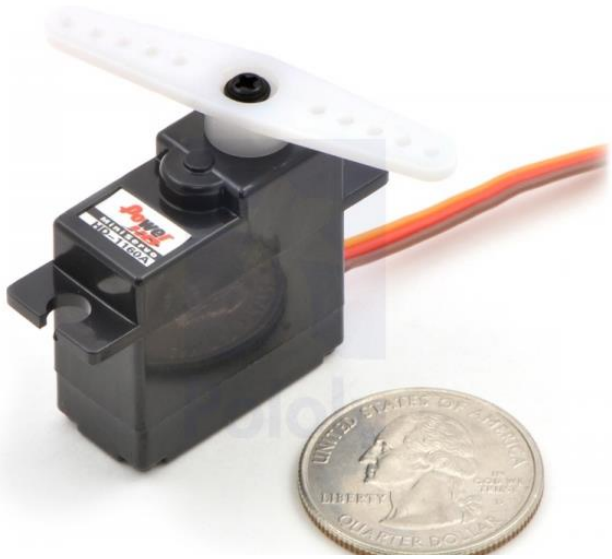


Figura 24 : mini servomotor

4 Diseños mecánicos

4.1 El Cuadrúpedo de 2GDL

Empezando por un Cuadrúpedo sencillo de 2 Grados de Libertad por pierna(8servomotores), se encontró en el laboratorio ITACA dos piezas que podían ser aprovechables para tener el cuerpo del robot listo.

Gracias a estos marcos, el cuerpo del robot de 2GDL fue fácil y rápido de construirse ya que estas bases tenían una forma apropiada para el cuerpo de un cuadrúpedo. Con estos dos marcos se trabajó para hacerles perforaciones para unirlos y hacer que la parte inferior y la parte superior del cuerpo quedaran a una buena distancia para contener las cuatro piernas(servos) al cuerpo en Angulo de 45 Grados. Se perforaron unos agujeros para colocar la placa, los servomotores y los interruptores. Con esto ya se podía empezar a montar el cuadrúpedo.

Los pies del cuadrúpedo están formados por unos tornillos largos metidos dentro de unos cilindros de plástico blancos como se puede observar en la imagen siguiente.

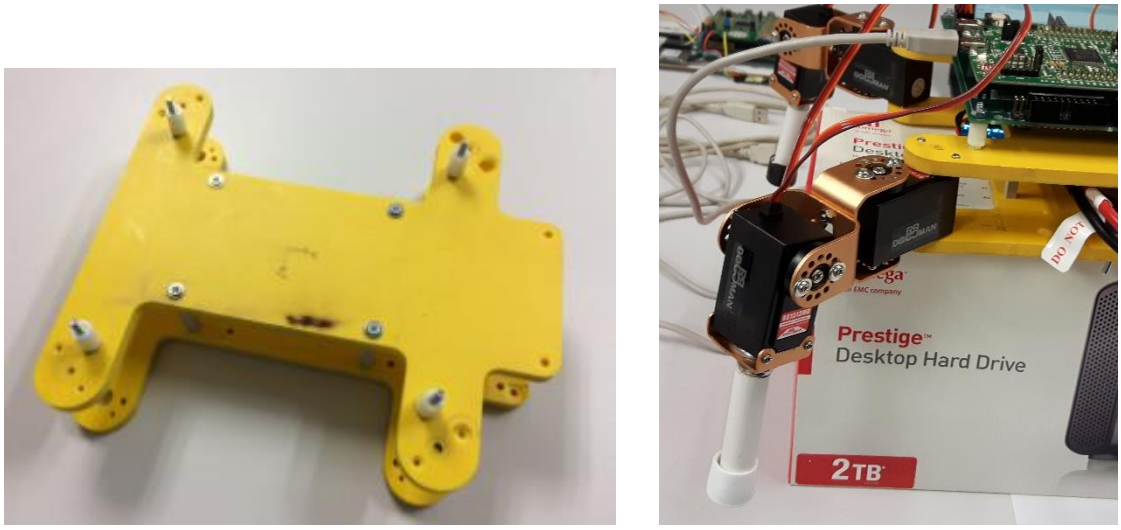


Figura 25: Partes mecánicas para el robot de 2GDL

4.2 El Cuadrúpedo de 3GDL

La mejora del diseño mecánico vendría dada por 2 aspectos que dotarían al robot de una evolución y un toque diferencial.

El primer aspecto era dotar al cuadrúpedo de un grado más de libertad en cada pierna. Es decir, dotar de un servomotor más en cada pierna pasando de tener 8 servomotores a 12. El segundo aspecto era dotar al robot de un cuerpo simétrico, envolvente y unos pies innovadores, modernos y sólidos. Esto haría que nuestro robot evolucionará de tamaño tuviera más grados de libertad, pudiera moverse sobre los 3 ejes de un entorno 3D con más facilidad. También dotaba al robot de una omnidireccionalidad a la hora de avanzar y caminar, sin necesidad de girar previamente. Además, el diseño mecánico quedaría muy bonito y muy moderno con la creación de las piezas diseñadas en una impresora 3D.

Al contrario que el robot de 2 GDL, el robot de 3GDL no tuvo un diseño predefinido y hecho. Era trabajo del propio autor diseñar y crear este robot usando las mismas medidas de motores. En este capítulo, se explica la metodología utilizada para el diseño del robot de 3GDL.

Para un buen diseño, no es posible ponerse detrás de un ordenador e inmediatamente empezar a dibujar en un programa de CAD. Se Necesitan seguir varios pasos para crear el modelo con las características que realmente se desea.

En primer lugar, es importante reunir esas características y comenzar la lluvia de ideas. Eso es exactamente lo que se hizo. Se buscaba un cuadrúpedo que fuera capaz de moverse en todas direcciones que fuera elegante y compacto. Así pues, se empezaron a dibujar algunos bocetos. Tratar de mantener nuestro robot compacto, asegurarse de que las placas quedaran todas en un núcleo, que el módulo inalámbrico pudiera encajar en un sitio donde la antena estuviera en una buena posición lo más arriba posible fue también un aspecto para tener en cuenta. Haciendo el robot con el pensamiento de ser capaz de caminar en todas direcciones, tratamos de darle un aspecto simétrico lo más posible gracias a la forma de un octágono.

Otro punto importante que se tuvo en cuenta era que las piezas que se iban a diseñar tenían que ser posteriormente imprimidas con una impresora 3D. El diseño inteligente era por lo tanto una necesidad. Cuanto más inteligente era el diseño, menos material se utilizaba y menos tiempo de impresión se necesitaba. Después de realizar todas las mediciones necesarias y todos los bocetos de las diferentes partes deseadas, se abrió SolidWorks y se empezó a dibujar.

Ser capaz de ver las partes 3D y ponerlas en un ensamblaje hizo posible ver qué partes del cuerpo se podrían diseñar mejor y dónde se debían hacer algunos cambios. El siguiente paso fue, por lo tanto, rediseñar las partes que no parecían correctas. El robot consta de 5 partes auto imprimidas: el cuerpo inferior, medio y superior, los pies y un soporte de batería.

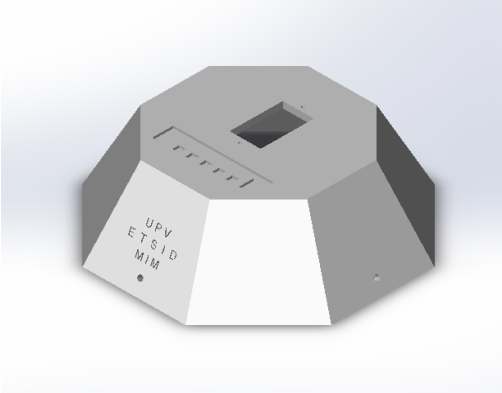


Figura 26

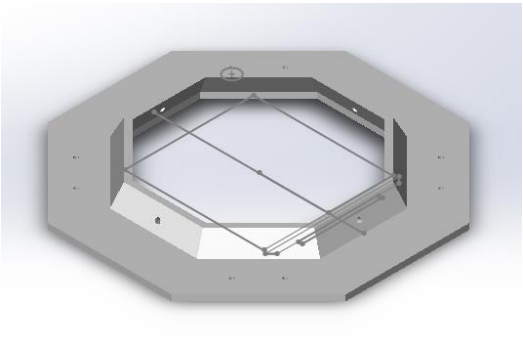


Figura 27

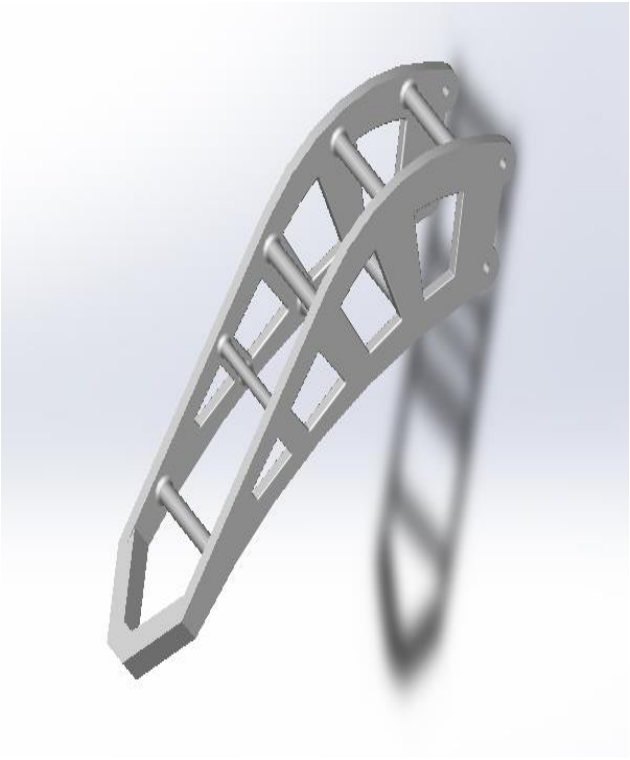


Figura 28

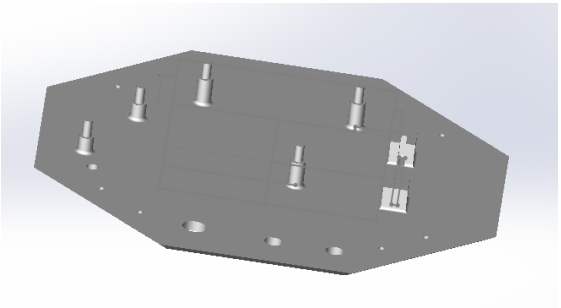


Figura 29

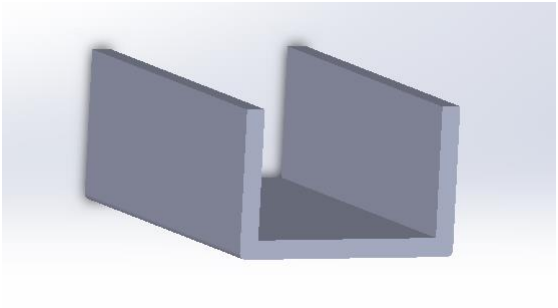


Figura 30

Figuras: 26, 27, 28, 29 y 30: Partes del robot para imprimir

Para poder leer los archivos en Z-SUITE, un programa proporcionado por Zortrax para convertir los archivos 3D en modelos, las partes creadas en el software de CAD deben guardarse como un archivo STL. Z-SUITE trabaja junto con muchos softwares en 3D, como AutoCAD de Autodesk, Cinema 4D, Adobe Photoshop, SketchUp y por supuesto SolidWorks.

Una vez que abra un archivo en Z-SUITE, obtendrá la siguiente pantalla:

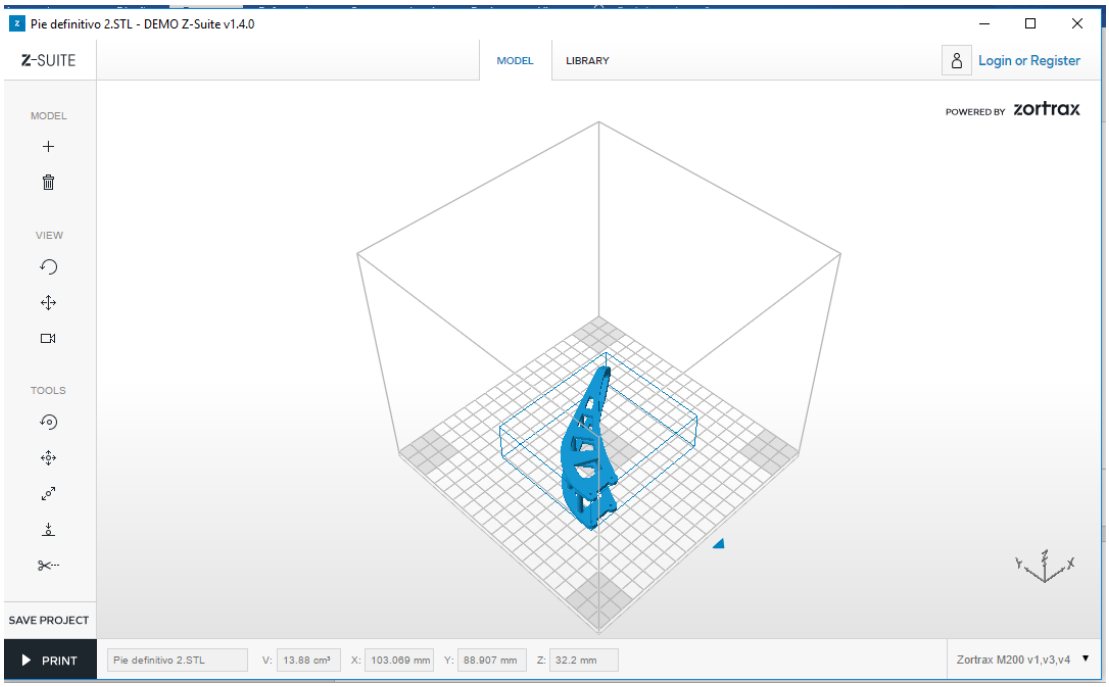


Figura 31: primera visión del archivo STL con Z-SUITE

Bajo la subdivisión 'Model' se obtiene la posibilidad de agregar y borrar modelos. De esta forma, puede imprimir varios modelos durante una sesión de impresión, lo que le permite ganar tiempo. Se puso varios modelos juntos para asegurarse de que este modelo se imprimió de la manera más rápida posible.

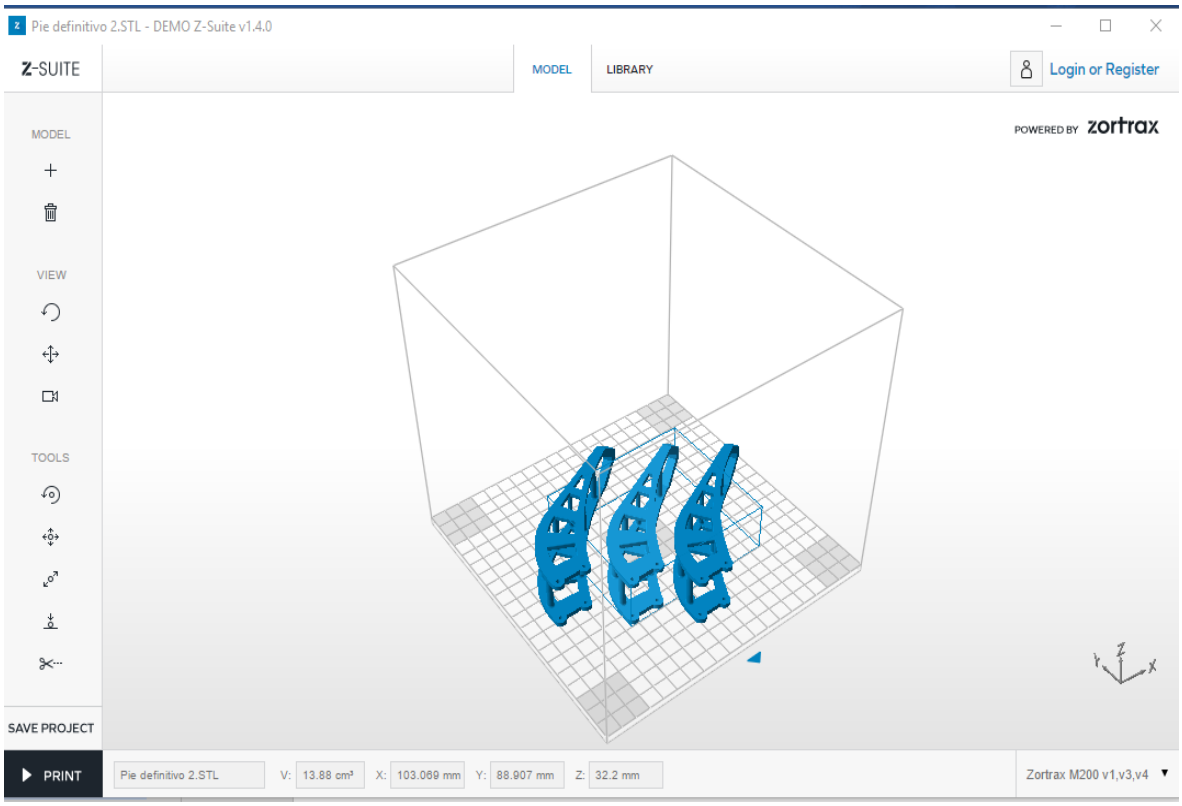


Figura 32: Varios modelos listos para ser impresos en la 1ª sesión

La sección de 'View' (vista) es únicamente para observar su modelo y por lo tanto no es tan importante.

Bajo la sección 'Tools'(herramientas) puede girar su modelo de tal manera que se utilice la dirección del filamento que más convenga para su resistencia a fuerzas. También se pueden cambiar los ajustes de impresión bajo esta subdivisión. Dado que estos ajustes son muy importantes, se han ido discutiendo uno por uno aquí.

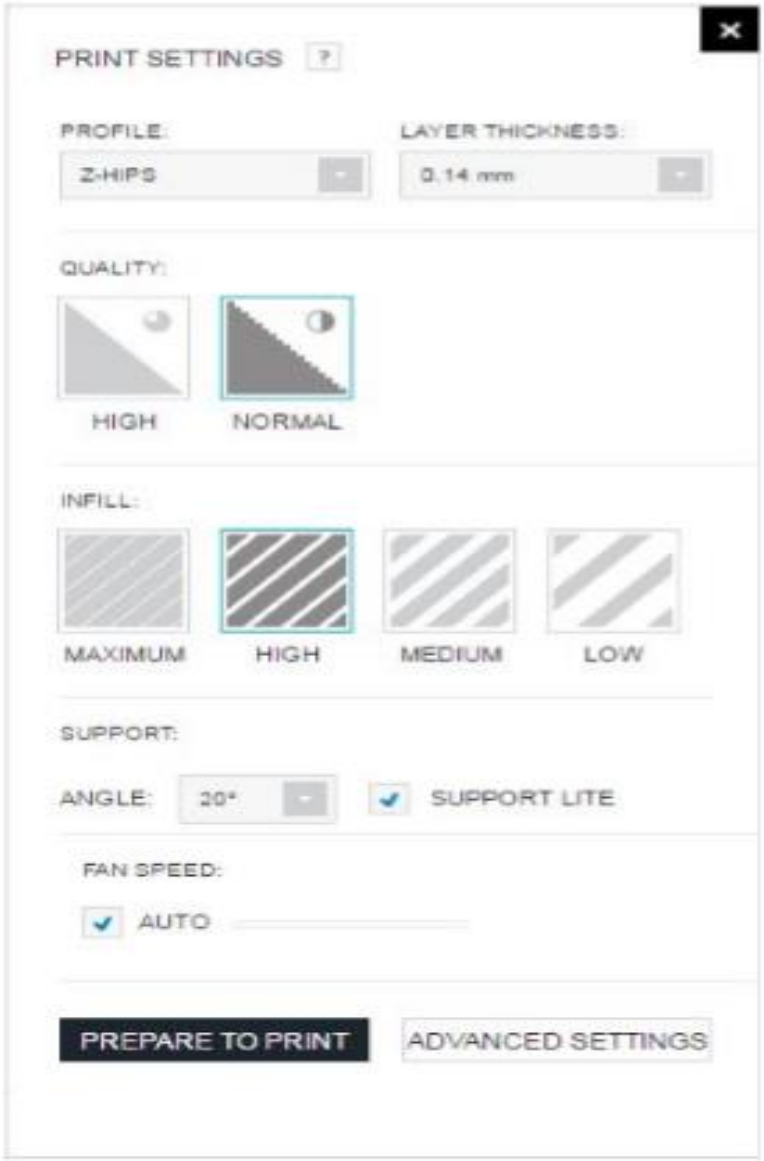


Figura 33: Ajustes de impresión

1. Perfil (Profile):

Elija el tipo de filamento (Z-ABS, Z-ULTRAT, Z-HIPS, Z-GLASS o Z-PETG). Z - HIPS tiene muy buenas características de resistencia. Puesto que esas eran las características más

importantes para nosotros y puesto que Z - GLASS y Z - PETG no estaban disponibles, elegimos este tipo de material.

Tabla de las propiedades de los diferentes materiales usados en la impresora 3D modelo ZORTRAX M200						
	Superficie	Dureza	Elasticidad	Resistencia al impacto	Resistencia a la tracción	Contracción
Z-ABS	Mate	Media	Media	Alta	Baja	Media
Z-ULTRAT	Semi-Mate	Alta	Media	Media	Baja	Baja
Z-HIPS	Mate	Baja	Media	Alta	Alta	Baja
Z-GLASS	barnizada y translucida	Media	Alta	Alta	Alta	Muy baja
Z-PETG	barnizada	Media	Alta	Alta	Alta	Muy baja

Tabla 1: Características de los diferentes tipos de materiales

2. Grosor de la capa (Layer Thickness):

Seleccione la altura de una capa (0,14, 0,19 o 0,29 mm). Las capas más finas darán como resultado una mejor calidad de superficie, pero aumentarán el tiempo de impresión. Dado que estos modelos eran muy finos y bastante pequeños, se eligió la capa más fina, por lo tanto 0,14 mm, para aumentar la resistencia y la calidad de la superficie.

3. Calidad (Quality):

Seleccione el modo de calidad (alto o normal). Una calidad superior proporciona impresiones de mejor calidad, pero aumenta el tiempo de impresión. La calidad normal fue suficiente.

4. Inserción (Infill):

Seleccione la densidad del relleno del modelo (máximo, alto, medio o bajo). Un relleno más denso dará lugar a una parte más fuerte, pero aumentará el tiempo de impresión y el uso de filamentos. Se encontró el equilibrio en el relleno alto.

5. Soporte (Support):

Seleccione el ángulo por debajo del cual se genera la estructura de soporte. Se eligió 20 °, que también es la opción por defecto, significa que para todas las superficies de modelos colgantes por debajo de 20 ° se genera estructura de soporte.

Se ha marcado la casilla de ‘Support lite’. La elección de esta opción genera una estructura de soporte sin perímetro exterior para modelos grandes. ‘Support lite’ también consume menos filamento y es más fácil de quitar.

6. Velocidad del ventilador (Fan speed):

Aquí podemos elegir la velocidad del ventilador, que enfría los modelos impresos. Ningún enfriamiento es útil para modelos grandes sin partes delgadas, mientras que el enfriamiento alto es mejor para modelos pequeños y delgados. En automático, la velocidad del ventilador se ajusta de forma inteligente según las necesidades. Esto es muy fácil y fue por lo tanto la elección más obvia.

Una vez que se haya elegido y configurado todos estos parámetros, presionamos 'prepare to print' (preparar para imprimir), se guarda el archivo en una tarjeta SD (una tarjeta de memoria) y la ponemos en la Impresora Zortrax M200 con el consentimiento de la Escuela Técnica Superior de Ingeniería del Diseño. Antes de que comience a imprimir, se debe limpiar la base de la impresora y rociar un poco de pegamento en la misma para asegurarnos de que el material permanece en su sitio durante la impresión. Además, antes de imprimir la impresora nos advertirá si la base no está bien calibrada para que demos nuestro consentimiento a que se autocalibre o nos ayude a ello ajustando los tornillos de la base.

Una vez hecho todo esto solo faltara decirle a la impresora que archivo es el que debe de imprimir, por defecto el ultimo archivo cargado en la memoria SD es el que sale por defecto para aceptar la impresión.

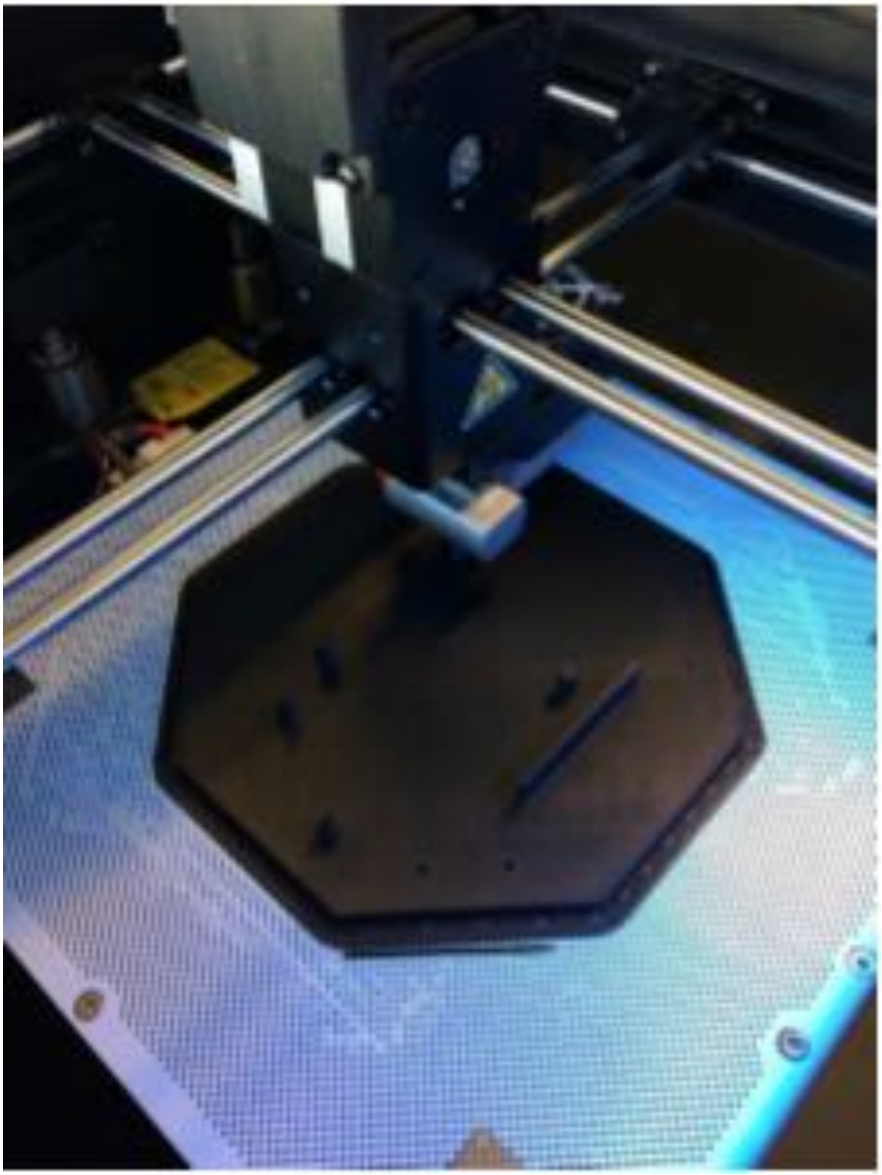


Figura 34: Impresora Zortrax M200 en acción imprimiendo la parte inferior del cuerpo

Una vez que las piezas han sido impresas, se deberá quitar los respectivos soportes de cada pieza. Además, la pieza que sujeta la batería se debe pegar a la parte inferior del cuerpo. Como se puede observar, se separaron para reducir el tiempo de impresión y el uso de filamentos.

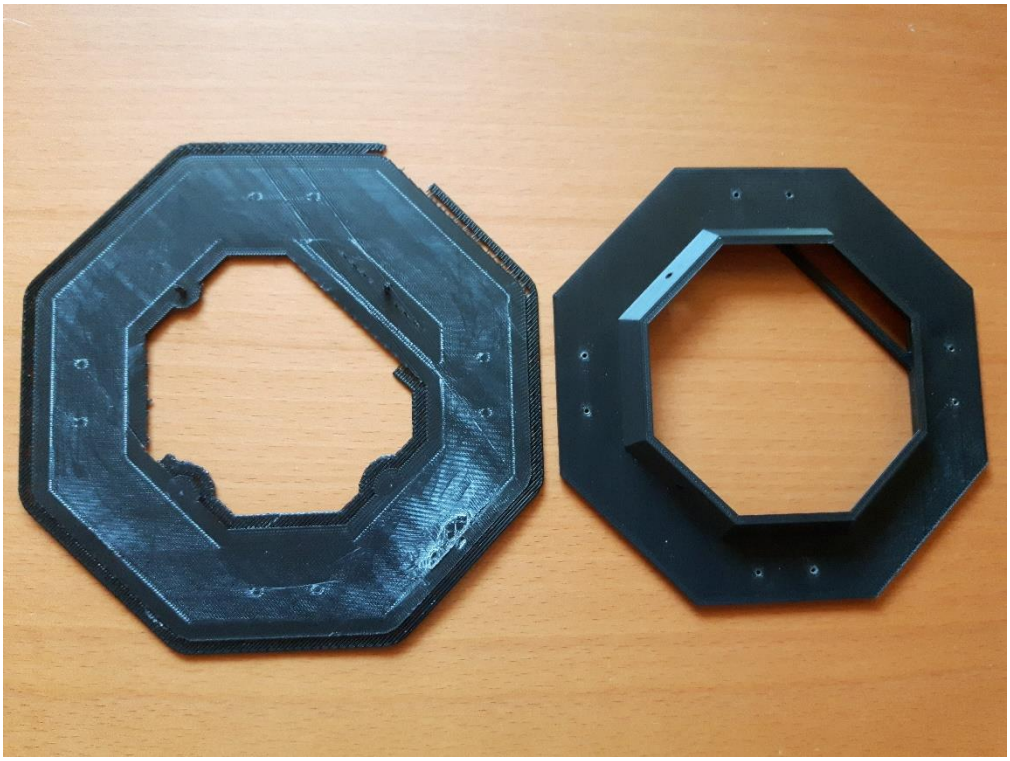


Figura 35 y 36 : piezas imprimidas con y sin soporte

5 Diseños electrónicos

5.1 El robot de 2GDL y los mandos inalámbricos

El robot de 2GDL y los controladores inalámbricos estaban equipados con un tablero STM32F3DISCOVERY y un PCB creado por ITACA (placa ITACA-RIS RMR 3815). Como ya estaba hecho y preparado para utilizar, no se tuvo que hacer ningún diseño electrónico. Para completar la información, se da una figura de las conexiones internas de dicha placa ITACA-RIS RMR 3815.

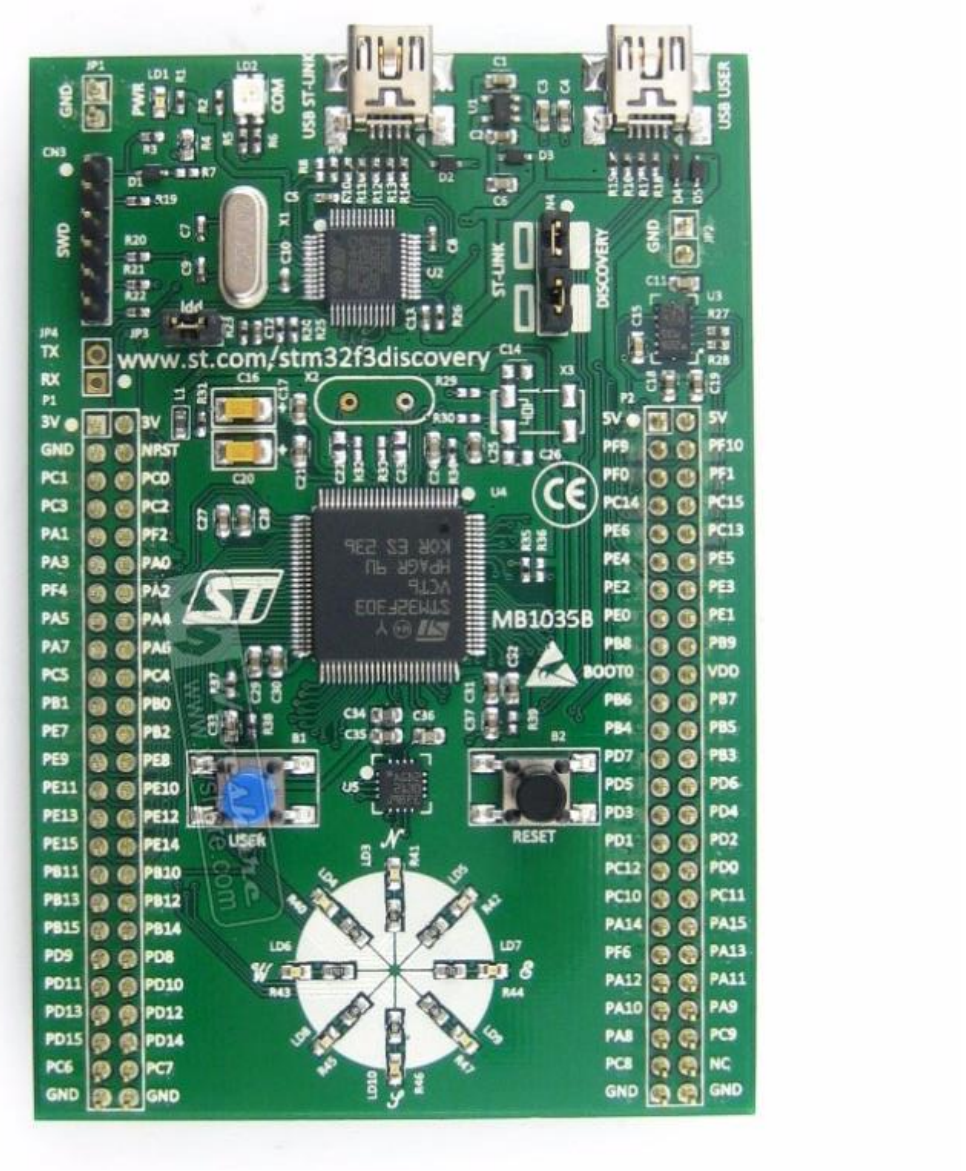


Figura 37 : Placa STM32F3DSICOVERY

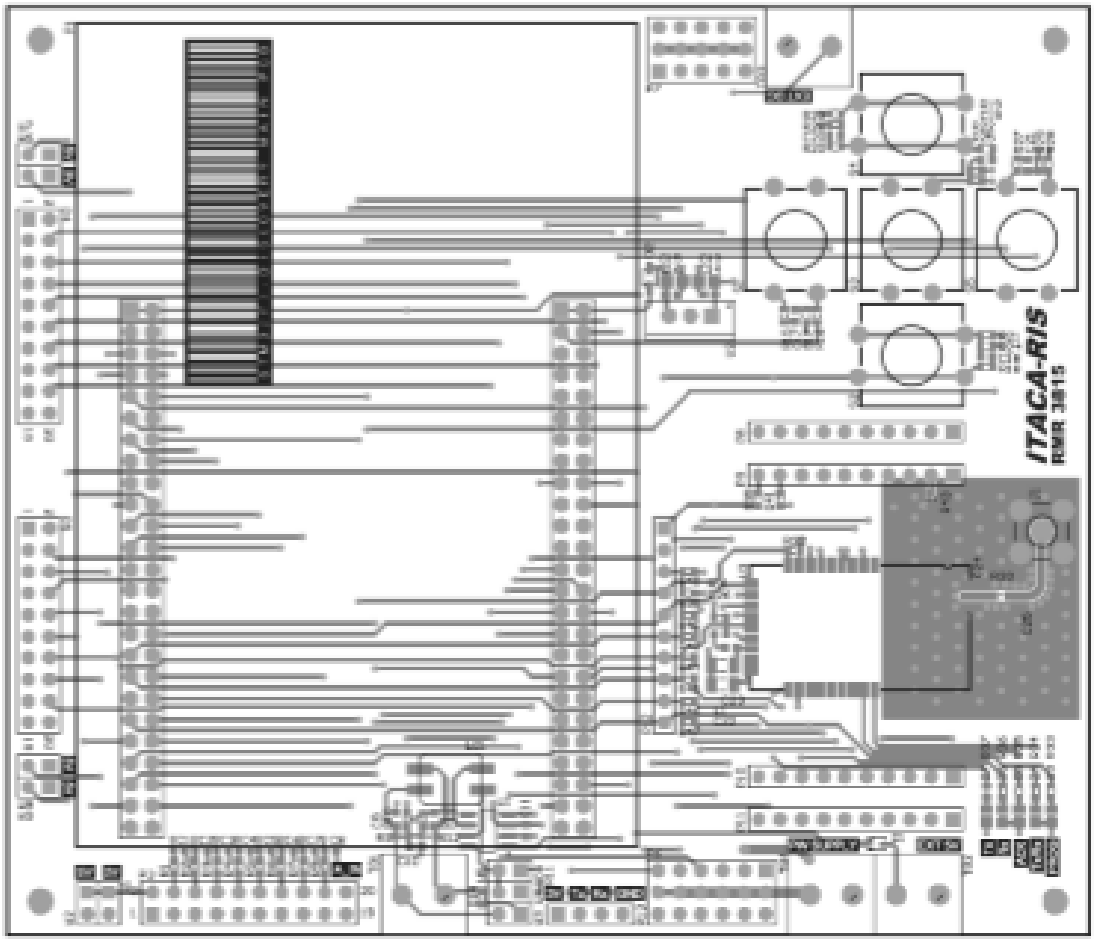


Figura 38: Conexiones interna de la PCB complementaria a la Núcleo DISCOVERY en el robot de 2GDL

5.2 El robot de 3GDL

5.2.1 Las Placas electrónicas

La placa que se utilizó en el robot 3 GDL no era la Discovery como en el anterior, pero si una placa Núcleo. El Núcleo F334R8 fue proporcionado por el laboratorio ITACA, al igual que el F3Discovery, por STMicroelectronics y fue extendido con una PCB diseñada por el autor del proyecto y creada por el laboratorio ITACA. Una PCB apilable de esta tendencia está haciendo la creación de prototipos semi-permanente fácil ya que las placas de procesamiento son para uso impermanente. Hacer esta PCB propia para este núcleo es una actividad mucho más lenta, pero se recomienda porque es muy útil y muy necesaria. La PCB JDMR0617 está diseñado para la Núcleo F334R8, pero también está diseñada para Arduino y puede así encajar en esta placa Núcleo F334R8. También para otras con su mismo formato tales como la F411 etc. puede ser válida.

(CN5, CN6, CN8 y CN9 son conectores hembra compatibles con el estándar Arduino). Esta PCB proporciona pines para:

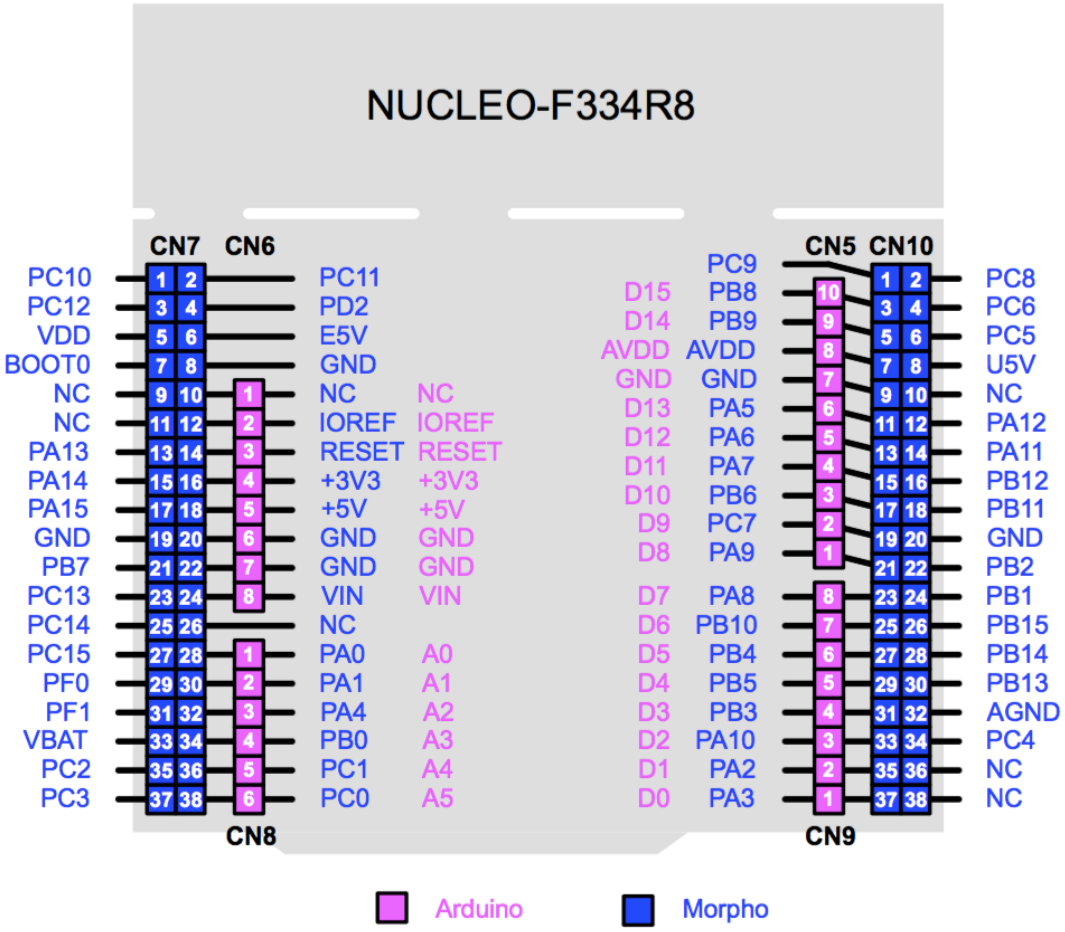
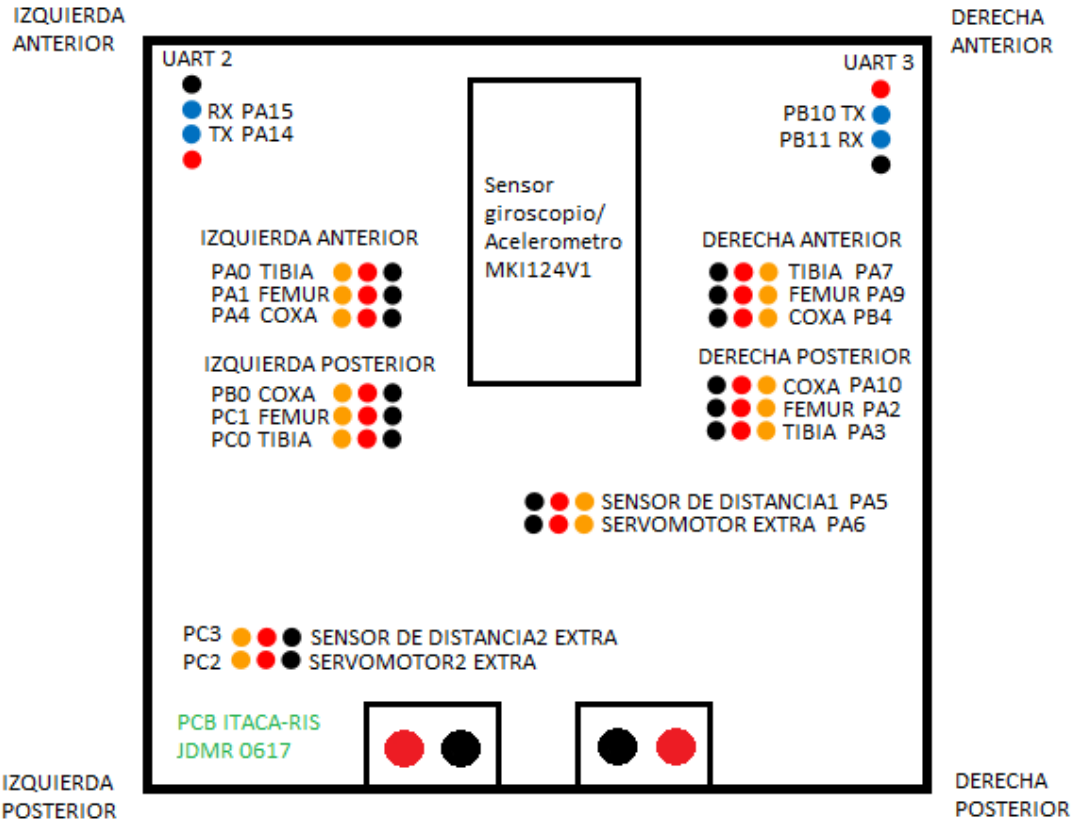


Figura 39: NUCLEO-F334R8

Dado que la placa JDMR0617 está directamente conectado a nuestra placa Nucelo F334R8 a través de los pines Arduino, tratamos de usar el mayor número de pines Arduino posible y tratamos de evitar los pines Morpho, ya que estos necesitaban ser conectados con un cable extra para hacer la conexión con la placa. Puesto que no todos los pines pueden utilizarse como temporizador, transmisor, receptor, SDA, ... hemos consultado el manual del usuario de las placas STM32 Nucleo-64 (tabla 14) [6] y la hoja de datos de los microcontroladores STM32F334x4 / 6/8 13) [14] para ver qué pines podrían cumplir con las funciones. Tenga en cuenta que se pueden asignar múltiples pines a las mismas funciones. 11 de los 14 servo pines fueron asignados como pines Arduino. Los 3 restantes, los pines UART y los de la STEVALMKI124V1 se asignaron como pines Morpho, ya que no quedaron pines Arduino.

En las tablas siguientes se da una visión general de los pines elegidos y su función.



El dibujo de la parte inferior de la JDMR0617 muestra en qué lugar se van a soldar los pines y los cabezales de tornillo (2 pines). Para crear el enlace permanente entre las 2 piezas de metal, se utilizó una aleación metálica fusible (soldadura) y un soldador eléctrico con punta gruesa que alcanza los 370 ° C para las soldaduras más grandes. Y otra punta más fina que alcanzaba los 275 º C para las soldaduras más pequeñas.

Una vez que se fijaron los pines y las cabeceras de tornillo, se hicieron las conexiones entre diferentes clavijas y pastillas. Esto podría deducirse de las tablas 2 y 3.

Como ya saben, los conectores del servomotor (conector tipo JR universal) requieren 3 pines: alimentación + 5.5V ~ 7.4V (rojo), tierra (marrón / negro) y señal PWM (naranja). Para las conexiones de la energía y de la tierra, los motores fueron divididos en 2 para reducir la corriente y el calor en las vetas eléctricas de la PCB. Por lo tanto, 2 conectores para entrada de energía fueron utilizados. El calor también se redujo mediante el uso de múltiples hilos. La razón de esto es que la corriente siempre pasa por el exterior del cable. Las conexiones para la energía y el suelo se hicieron de tal manera que estaban en la posición exterior, para asegurarse de que se impedía una conexión entre tierra y potencia. También es importante tener en cuenta que el suelo de los cabezales de tornillo debe conectarse al suelo de la placa para obtener una buena señal.

5.2.2 Diseño e implementación de la PCB JDMR-0617

En el robot de 2GDL se ha utilizado como placa principal la Nucleo F3DISCOVERY, y como acompañante la placa ITACA RMR 3815.Esta placa fue creada por un técnico de laboratorio de ITACA. Ya que él fue el creador nombro a la placa con las iniciales de su nombre y apellidos RMR y además con la fecha que fue dicha placa creada. Es decir, 3815, donde 38 fue la semana y 15 el año. Esta placa RMR3815 fue diseñada para sacar de la F3DISCOVERY conexiones de pines de manera que fuera más fácil y organizada la conexión de posibles periféricos como las de los servomotores o las de un módulo de radiofrecuencia por las USART'S como se le ha dado uso en el robot de 2GDL. Además, dota a los servomotores de una conexión distinta de las placas, con unos conectores donde se le puede conectar una señal de voltaje de hasta 12V. Así se puede elegir a que voltaje se quiere alimentar a los servomotores.

En el robot de 3GDL sin embargo la placa principal que lleva el microcontrolador cortex M4 es la llamada STM F334R8. Este tipo de placas fue una adquisición del Laboratorio ITACA posterior a la F3DISCOVERY. Una de las cosas buenas de esta placa y que por ello se eligió para el diseño del cuadrúpedo de 3GDL es su tamaño reducido. Esta placa es más pequeña en comparación con la DISCOVERY. En cuanto a su potencial de velocidad y cantidad de conexiones las dos son muy parejas y del mismo estilo. Por tanto, se escogió esta placa F334R8 para el robot de 3GDL como se ha mencionado con anterioridad. Ahora bien, esta placa no disfrutaba de otra placa acopladora como es la RMR3815 para la F3DISCOVERY.Por tanto, a la hora de conectar periféricos como servomotores o sensores a esta placa se hacía una tarea muy engorrosa y difícil por no decir imposible para implementar en un cuadrúpedo.

Llegado a este punto, el autor de este proyecto tuvo que meditar que solución se podía tomar para dotar a la placa nucleo F334R8 de una mejor conexión para los periféricos que queríamos dotar al cuadrúpedo. Una de las opciones rápidas que se encontró fue hacer uso de una placa board que permitía mediante cableado subir las conexiones, los pines y también conectar a esta board conectores necesarios. En el laboratorio ITACA algunos otros proyectos ya habían optado por esta opción la cual no gusto mucho debido a la cantidad de cableado que quedaba entre las dos placas… una imagen a continuación puede ofrecer lo que no convencía, ya que en cualquier momento las conexiones podrían romperse:

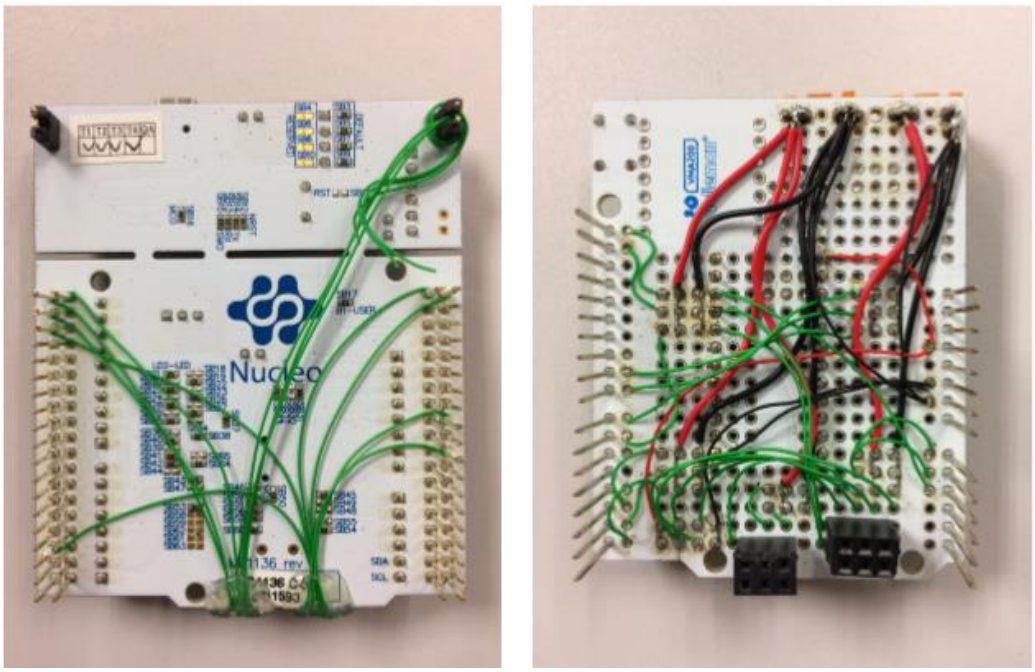


Figura 40 : proyecto antiguo de ITACA donde las conexiones del cableado dejan mucho que desear

la opción o alternativa que más convencía por ser igual a la del robot de 2GDL era la creación de una PCB con la misma funcionalidad que la mostrada por la RMR 3815 pero en este caso adaptada a la F334R8 para el robot de 3GDL. Por tanto, el autor del proyecto procedió a diseñar una PCB de tamaño similar a la F334R8 que acoplara por arriba para hacer las conexiones más fáciles. La idea era poder juntar en la PCB 12 o más conectores para los 12 servomotores que se iban a utilizar más dos o tres extras, por si más adelante se quería ampliar el robot. Además, debería estar conectado la IMU chip STEVAL-MKI124V1, 2 conectores para UART para el módulo de RF SP1ML de STM por ejemplo, 2 conectores para 1 o 2 medidores de distancia o cualquier otro sensor que se quisiera ampliar, 2 conectores para miniservomotores alimentados a 5V, dos entradas de corriente para alimentar los servos a través de la batería a 7.4V, y por último subir todos los pines posibles de la F334R8 hacia arriba a la PCB para facilitar la conexión.

Para todo ello, antes de meterse de lleno en el diseño de la PCB, se debía de conocer y familiarizarse con el entorno o programa (nunca utilizado por el autor de este proyecto) para el diseño de PCB como era el programa Altium Designer. Este programa estaba instalado en los ordenadores del laboratorio ITACA y era perfecto para este tipo de diseños.

Para familiarizarnos con el programa y su funcionamiento, se tuvo que estudiar el entorno del programa mediante unos tutoriales que se ubicaban en la página web de Altium. Esta tarea llevo bastante tiempo y bastantes horas de cara al ordenador, pero al finalizarlos, nos sirvió de gran ayuda para empezar a diseño y crear la PCB deseada. En estos tutoriales seguidos se instaba a realizar algunas prácticas y en concreto a realizar una PCB sencilla pero completa que los autores del tutorial consideraban básico para un principiante. Un tutorial es el principal y es bastante largo y completo y dentro de él se vinculó a otros tutoriales que explican algunas operaciones más engorrosas a seguir.

Dicho tutorial principal se denomina: Tutorial - Getting Started with PCB Design

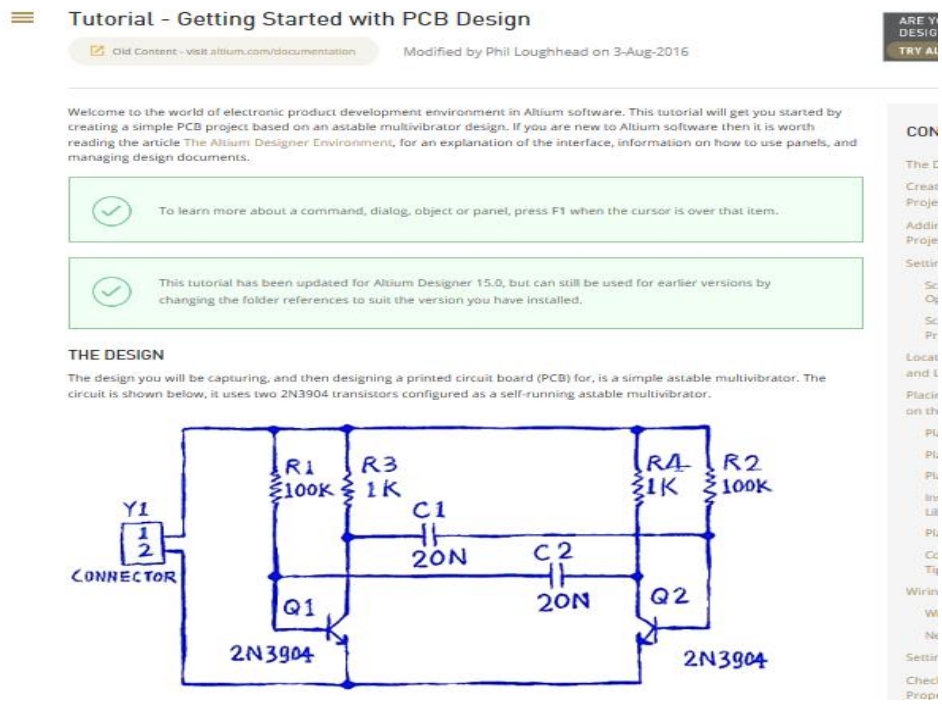


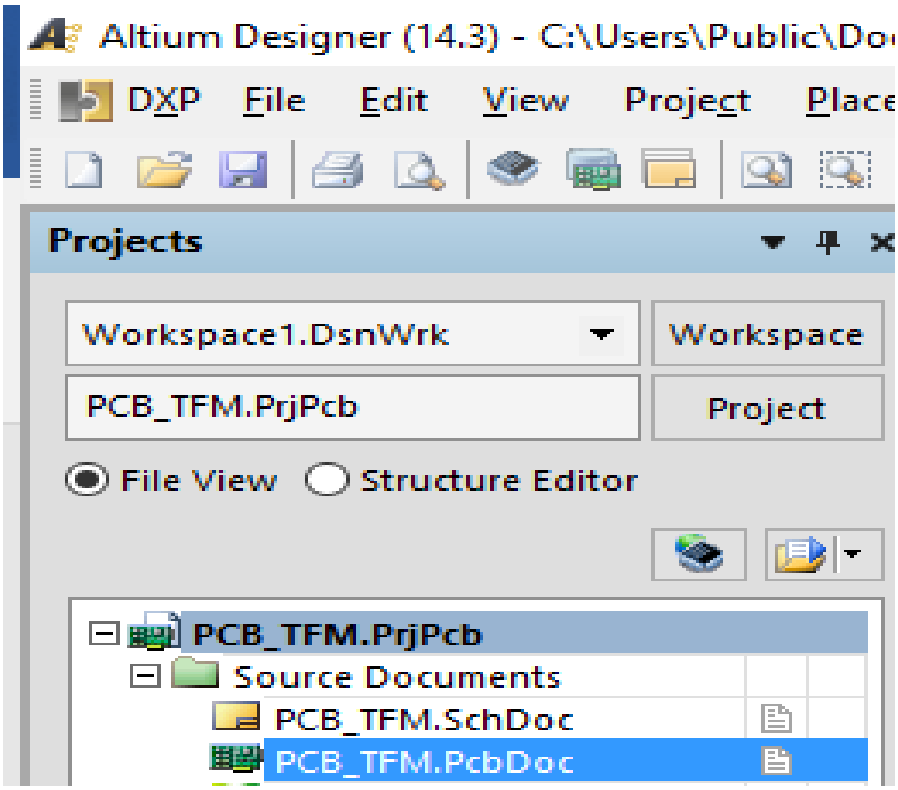
Figura 41: Tutorial - Introducción al diseño de PCB. Ejemplo miltivibrator

El ejemplo a seguir era el diseño y la creación de una simple PCB multivibrador astable con el programa anteriormente dicho Altium Designer.

Para ello se marcaban unos pasos muy importantes a seguir. Estos pasos se han seguido tanto para hacer el ejemplo del tutorial como para posteriormente crear y diseñar la PCB JDMR 0617.

Estos son los pasos que se siguieron para el diseño de la PCB explicados a continuación:

1. Crear un nuevo proyectó PCB dentro del programa. En nuestro caso se llamará PCB_TFM
2. Añadir al proyecto un esquemático del circuito. Se denominará PCB_TFM.sch



Figura

3. Localizar los componentes electrónicos que iba a llevar la PCB para ponerlos en el esquemático.
   Para localizar los componentes se pueden utilizar dos opciones:
   La primera es buscar los componentes en las librerías que vienen con el programa instaladas o que se han ido instalando por el usuario.
   La segunda es buscarlos en la Valut library, es decir una librería de Altium Online donde se encuentran los componentes actualizados gracias a los servidores. Eso si algunos componentes de esta librería son de pago para utilizar.

   Para nuestra PCb se utilizaron las dos opciones.

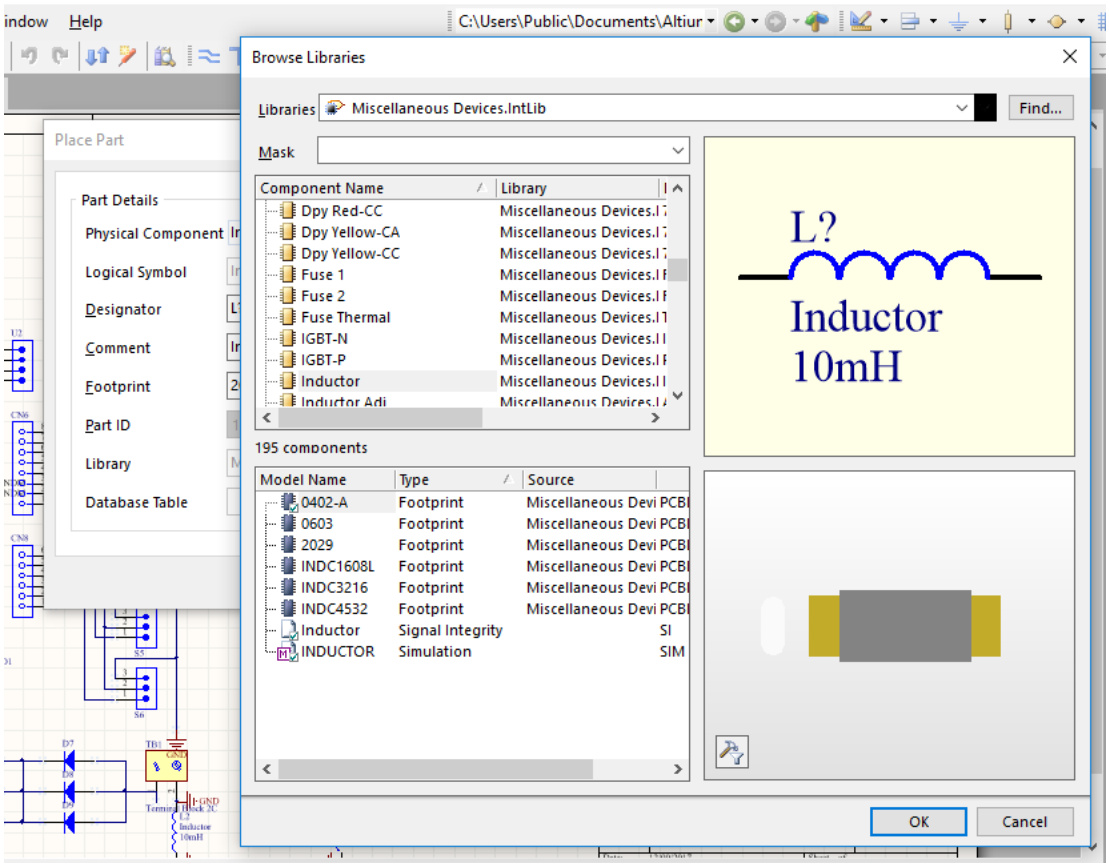


Figura 42 : Librerías instaladas en Altium

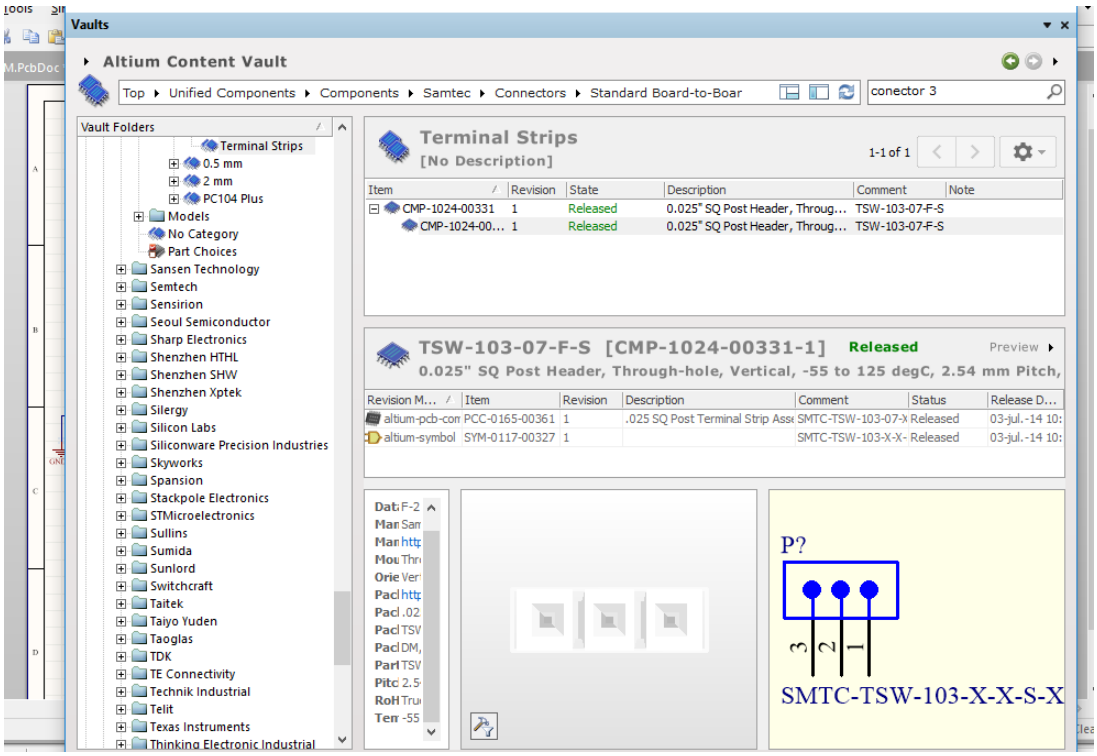


Figura 43 : Librería Interactiva Vault de Altium

4. Colocación de los componentes en el esquemático (transistores, condensadores, conectores, resistencias…).

5. Cablear el circuito en el esquemático. Además de etiquetar las redes, señales de voltaje o de Massa.

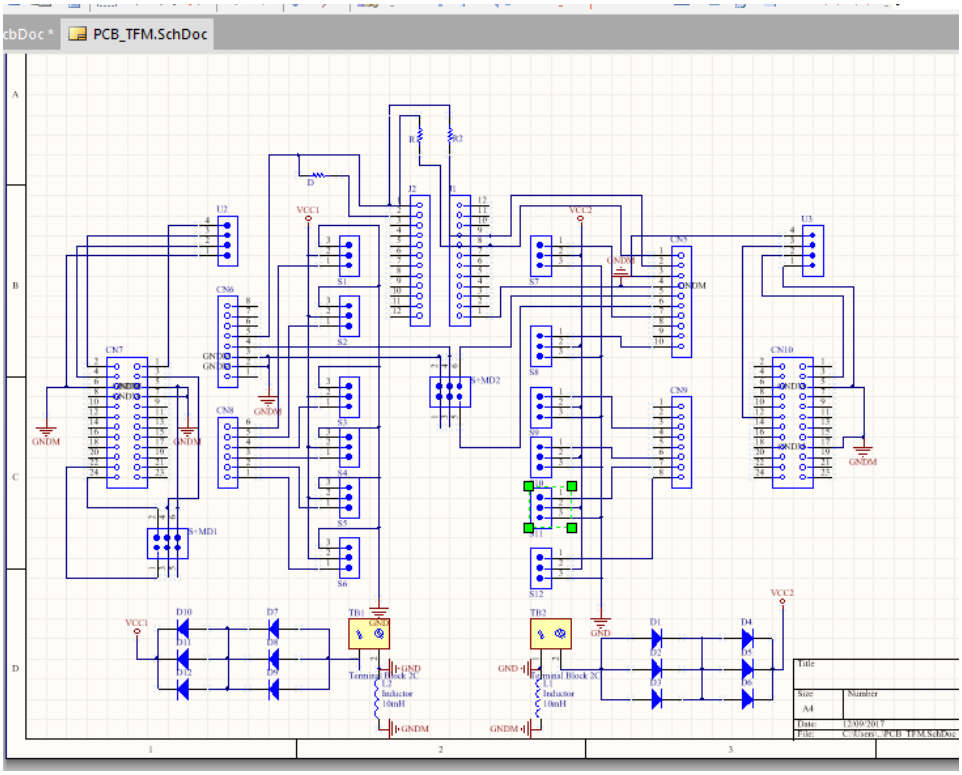
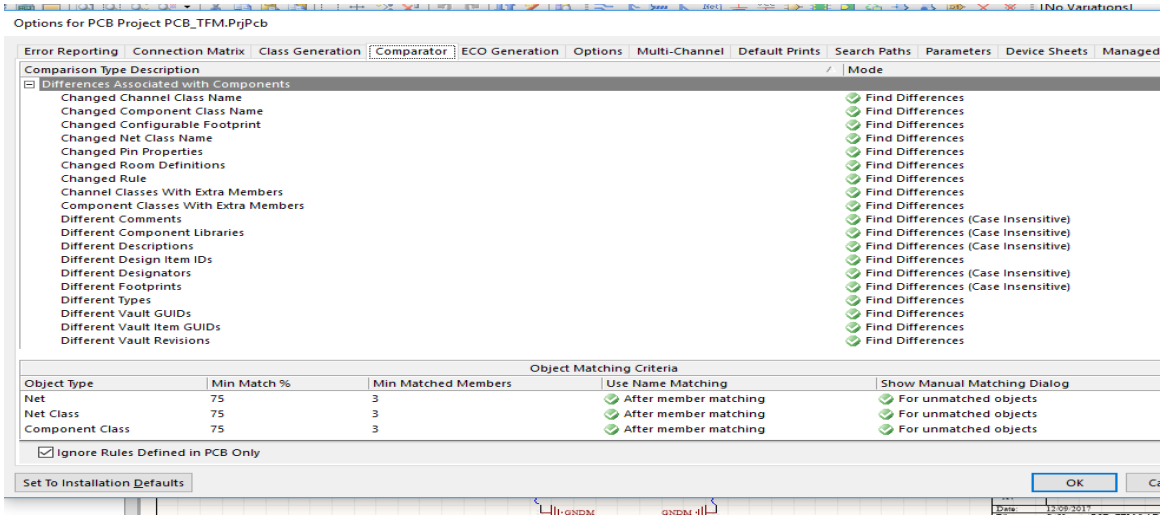
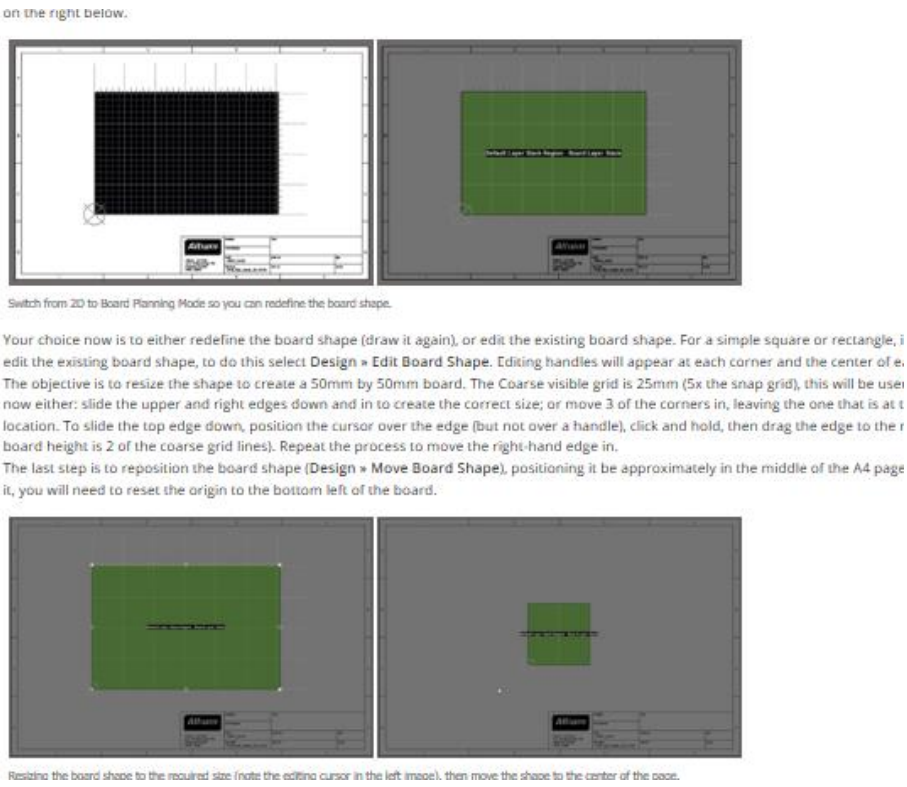


Figura 44 : Esquemático final

6. Comprobación de las propiedades eléctricas de su esquema. El programa tiene un compilador de errores de conexiones o demás que nos avisa al ejecutarlo. Si todas las conexiones están bien hechas nos dará el Okey para continuar el proceso.



7. Una vez este verificado el archivo Esquemático, se pasará a la creación de un nuevo archivo PCB. En nuestro caso llamado PCB_TFM.pcb
8. En este archivo PCb lo primero que se realizará será la creación de una nueva tabla a partir de una plantilla. Se le dará las dimensiones deseadas a la tabla PCB, así como la forma. También se le asignara un punto de origen a la tabla (0.0).
   En nuestra PCB se le dio unas dimensiones de 65 cm de alto x 70 cm de ancho. La forma fue rectangular. El Origen se situó en la esquina de abajo Izquierda.


on the right below.

Switch from 2D to Board Planning Mode so you can redefine the board shape.

7. Your choice now is to either redefine the board shape (draw it again), or edit the existing board shape. For a simple square or rectangle, i edit the existing board shape, to do this select **Design » Edit Board Shape**. Editing handles will appear at each corner and the center of e
8. The objective is to resize the shape to create a 50mm by 50mm board. The Coarse visible grid is 25mm (5x the snap grid), this will be use now either: slide the upper and right edges down and in to create the correct size; or move 3 of the corners in, leaving the one that is at t location. To slide the top edge down, position the cursor over the edge (but not over a handle), click and hold, then drag the edge to the r board height is 2 of the coarse grid lines). Repeat the process to move the right-hand edge in.
9. The last step is to reposition the board shape (**Design » Move Board Shape**), positioning it be approximately in the middle of the A4 page it, you will need to reset the origin to the bottom left of the board.

Resizing the board shape to the required size (note the editing cursor in the left image), then move the shape to the center of the page.

Figura 45 :Origen de coordenadas y dimensiones del tablero

9. A continuación, se procederá a transferir el circuito diseñado en el archivo Esquemático al archivo PCB. El programa tiene una opción de importación del Esquemático al PCBdoc. El solo validara los cambios ejecutara los cambios y mostrara en el documento PCB las huellas digitales de todos los componentes electrónicos que estaban en el Esquemático, así como su cableado.

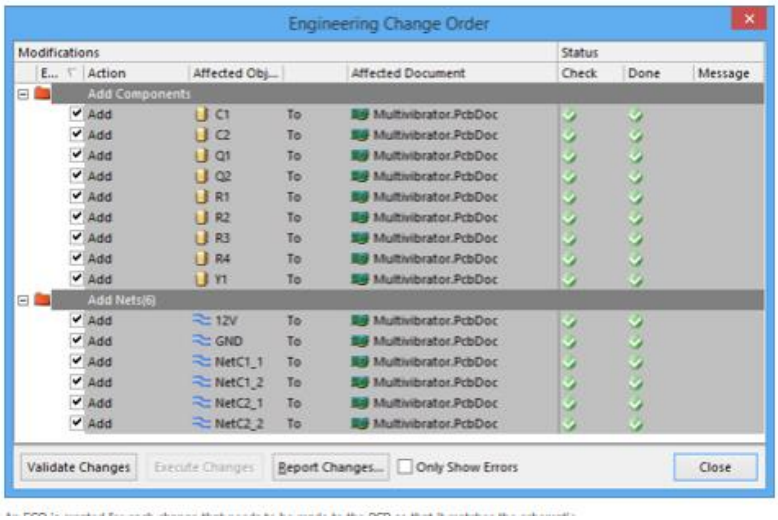


Figura 46: Validación

10. Es ahora pues cuando se estará listo para comenzar con el progreso de diseño de la PCB. Al transferir las huellas de los componentes, el programa deja las huellas y el cableado no en la tabla PCb sino justamente al lado de ella, para que sea el usuario el encargado de situarlo todo en la tabla según su criterio de diseño deseado.

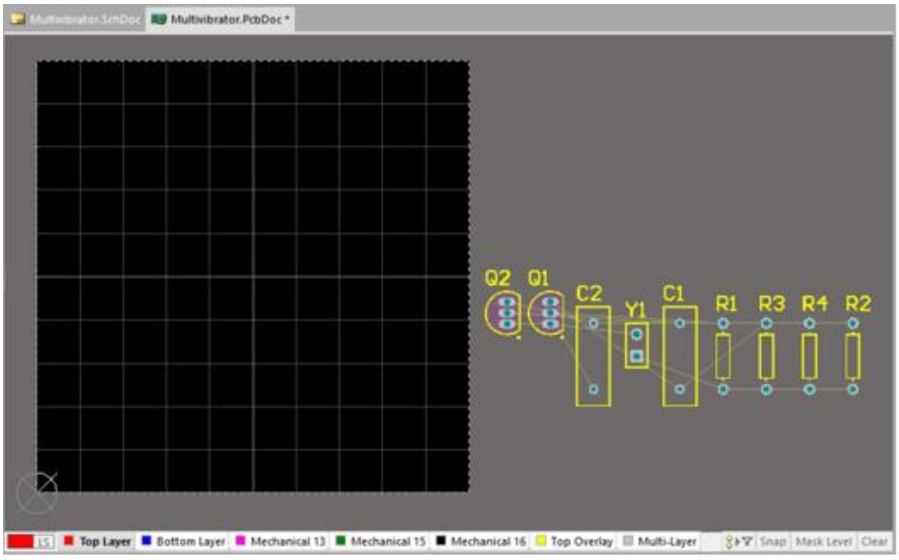


Figura 47: Posicionamiento de elementos cargados del Esquemático

11. Este documento o archivo PCB utiliza un Layer Stack Manager o lo que es lo mismo un Administrador de capas. Se debe tener en cuenta que las PCB's esta fabricadas por varias capas (las capas de serigrafia, la capa para soldar, la capa de metal, la capa dieléctrica etc... Se deberá elegir pues en que capa van situadas las huellas. Normalmente y casi siempre en la capa llamada por el programa 'Top Layer'.

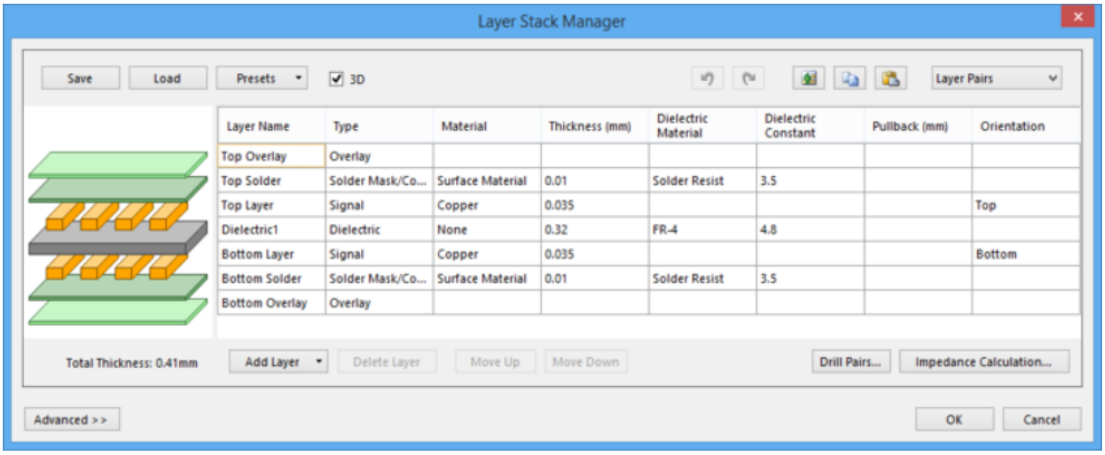


Figura 48: Selector de capas

12. Colocar las huellas puede ser muy fácil si se desea un sitio aleatorio en la tabla o muy complicado y exacto si por el contrario el sitio para dicha huella debe ser uno concreto y exacto ya que debe coincidir para que acople con otra placa (como era nuestro caso).
    Para ello sirvió de ayuda el origen asignado anteriormente, las medidas de la placa Nucleo F334R8 y las herramientas que ofrecía el programa Altium para escribir exactamente la distancia concreta (coordenadas X e Y) a la que debería de ir cada huella digital en la tabla sobre un plano de trabajo 2D con el que trabaja Altium Designer. También ofrece la posibilidad de elegir las unidades de medida con las que se prefiere trabajar: en medida Imperial o Métrica. Otra herramienta que puede ser de gran ayuda es la de Alineación de objetos.

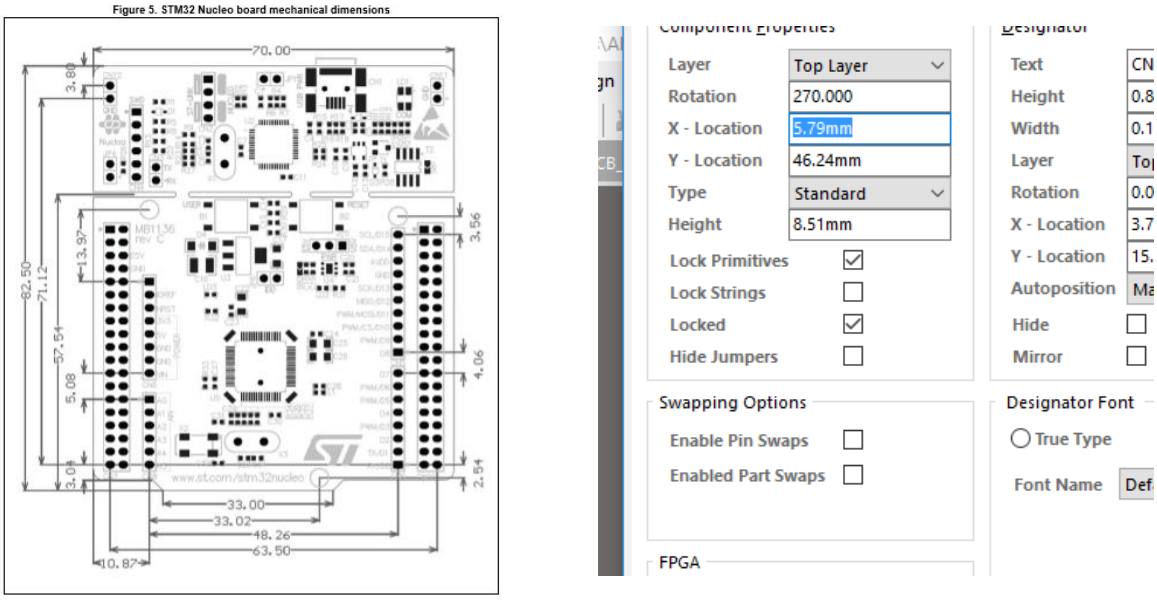


Figura 49: Medidas de la Núcleo F334 y Opción en altium de escribirlas.

13. Una vez esté bien colocadas todas las huellas digitales, se procederá a enrutar las vías o vetas (por donde pasara la corriente) que seguirán el camino del cableado tal y como se conectó en el Esquemático. Antes de ello el tutorial nos insta a repasar las reglas de díselo del enrutamiento en el 'Editor PCB rules &constraints'. En este Editor podremos configurar varias opciones como: el ancho de enrutamiento de las vetas, la prioridad a la hora de hacer los caminos de enrutamiento, poder rodear de vetas una huella de un elemento…etc. Estas opciones podemos activarlas o desactivarlas según convenga para facilitar y hacer más rápido el enrutamiento.
14.

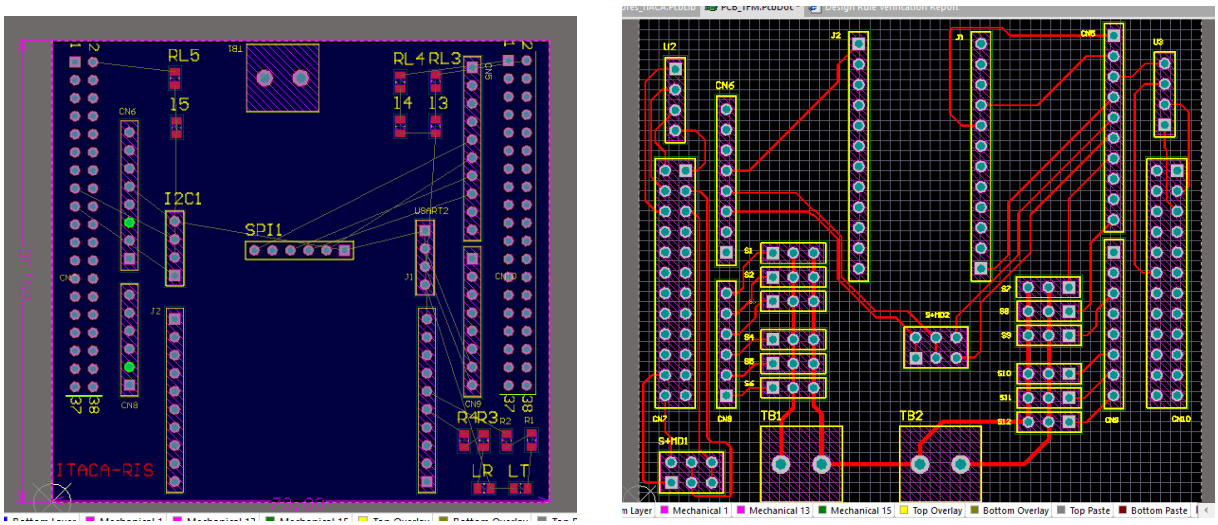


Figura50 y 51: Posicionamiento en tablero y enrutamiento de vetas

15. También es importante prestarle atención al ancho de enrutamiento de las redes principales de alimentación, ya que por ellas es conveniente que pase la mayor intensidad de corriente por las vetas para que los elementos a conectar en este caso los servomotores trabajen con el voltaje y la intensidad de corriente necesaria.
En nuestro caso las vetas de alimentación de los servos tenían un ancho de 1mm.Mientras que las otras vetas de señal tenían 0.25mm. Además, las vetas de alimentación estaban duplicadas por la capa inferior de la placa con la misma anchura de 1mm.

16. Otras opciones que son conveniente observar y estudiar son: la definición de la restricción de separación eléctrica y la definición del enrutamiento a través del estilo (redondo cuadrado).

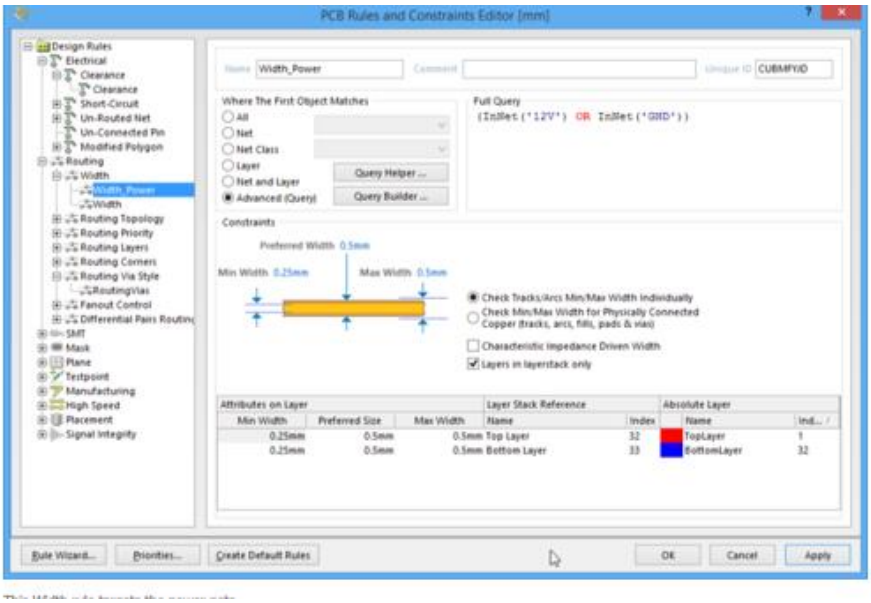


Figura52 : Ventana donde poder cambiar el ancho de las vetas.

17. Terminado pues el repaso a las reglas de enrutamiento con el Editor se procederá al enrutamiento de las vetas en la tabla. Esta tarea tiene como opciones dos alternativas: o bien el Enrutamiento automático interactivo del tablero (realizado por el programa) o bien el Enrutamiento interactivo del tablero realizador por el propio usuario. Normalmente los enrutamientos hechos por el usuario y hechos por el programa no suelen coincidir, ahora bien, los dos pueden ser igualmente correctos.
El enrutamiento de nuestra placa fue hecho por el propio usuario quedando de la siguiente manera:

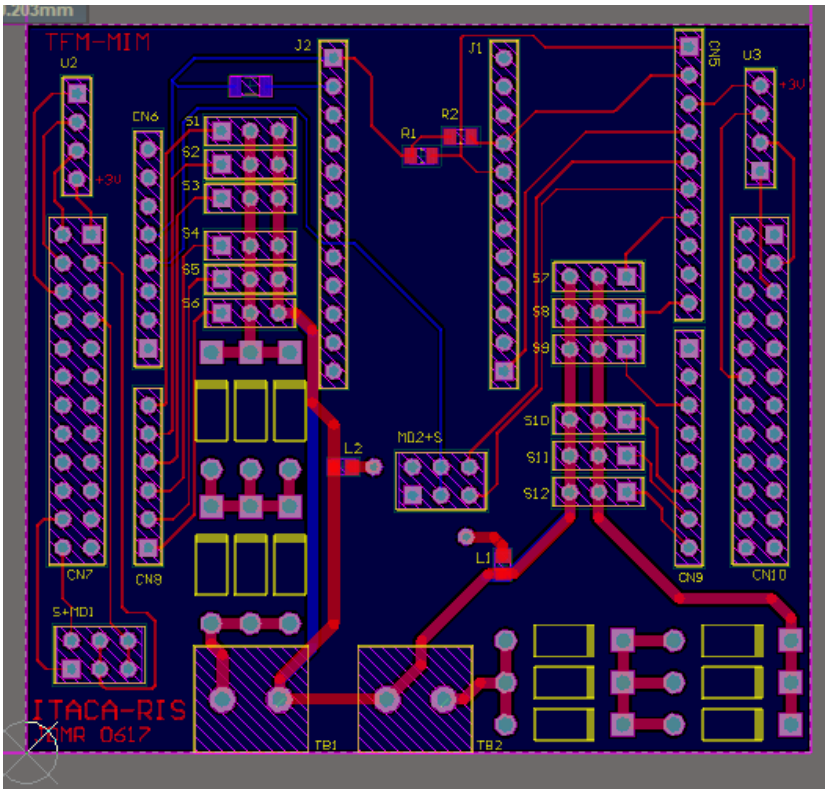


Figura 53: PCB JDMR0617

18. Una vez hecho el enrutamiento de las vías completo, habrá que verificar el diseño de la placa. Ver si aparece alguna violación de regla (aparecerá de color verde claro) e intentar solucionarla. Las violaciones suelen venir cuando alguna ruta no está bien conectada a la huella digital o cuando Net toca con algún elemento o etiqueta de serigrafia…etc.

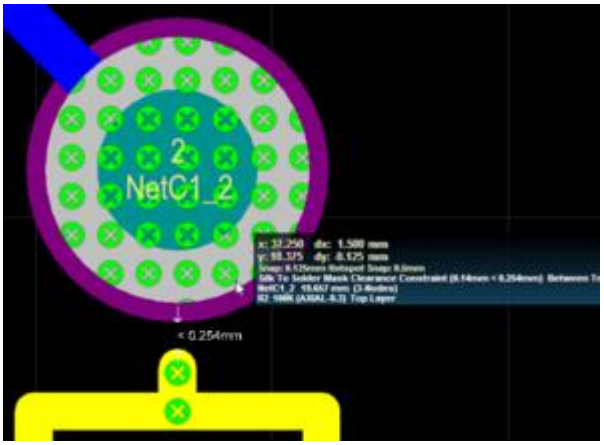


Figura 54: Infracción se muestran en verde

19. A continuación, un apartado que no venía en el tutorial pero que se aconsejaba para esta PCB era la creación de un plano de masa por toda la placa. De esta forma nos serviría también para quitar posibles interferencias o ruidos en las señales. Para este punto el programa ofrecía una opción para crearlo 'Poligon POUR. Lo que debíamos de decirle era cual de nuestras etiquetas no era nuestra masa. En este caso llamada GNDM. El programa también nos ofrece distintas opciones para crea el plano de masa tipo solido (todo), por mallas o por redes.

    Se eligió de tipo Solido ya que era el mejor. Su ubicación fue la capa Inferior Bottom Layer.

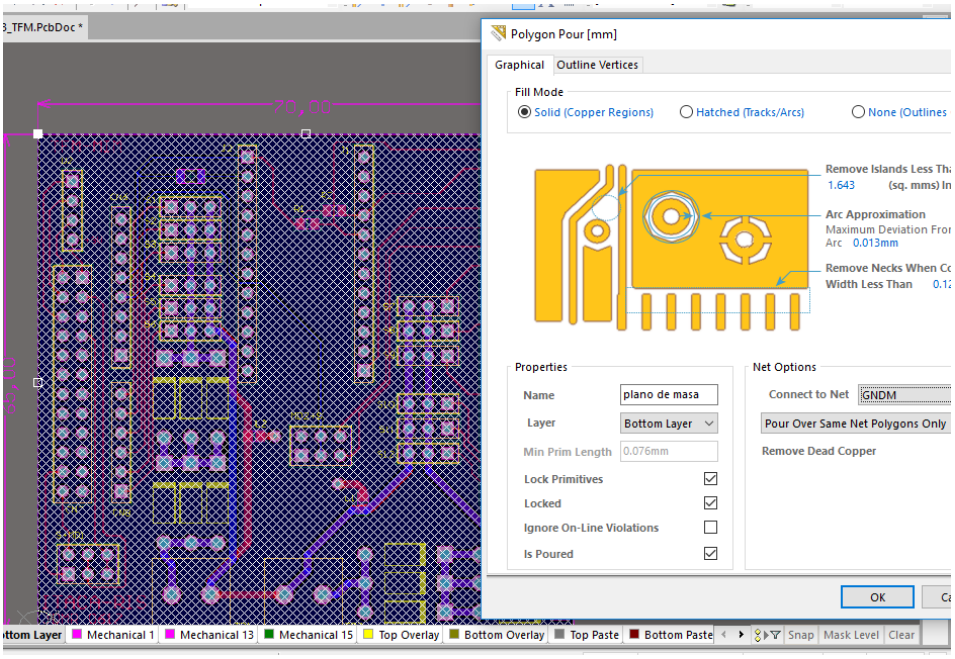


Figura 56: plano de masa

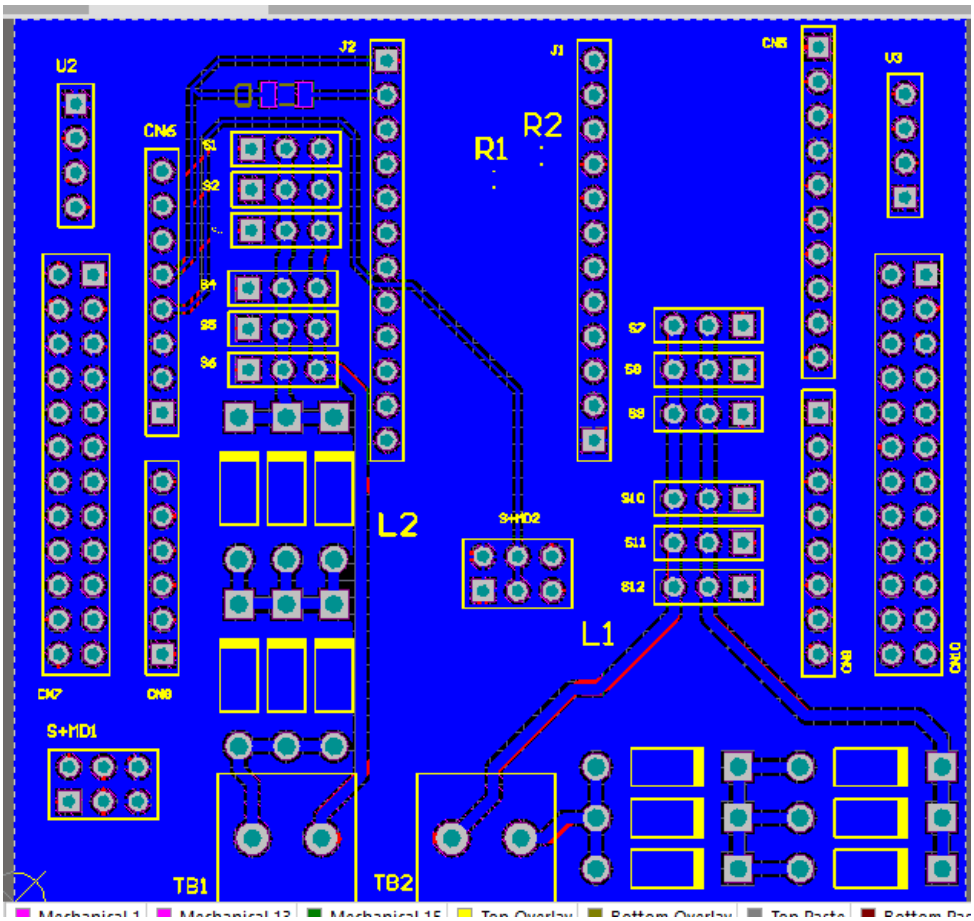


Figura57: capa inferior con vetas inferiores de alimentación y plano de masa sólido.

20. Por ultimo y para mejorar la PCB sobre posibles interferencias y ruidos, se pusieron dos huellas digitales conectadas a masa para soldar dos bobinas diminutas. Se pusieron en el Esquemático y se actualizo a la PCBdoc. Con la opción de actualizar que tiene Altium.

21. También se pusieron 6 diodos rectificadores en estructura de 3 en paralelo en serie con otro 3 más en paralelo, lo que nos ofrecía una reducción del voltaje de 0.8 a 1 V, y así también para prevenir posibles cortocircuitos o subidas de tensiones. Estos fueron puestos y actualizados entre la entrada de alimentación que viene de la batería y los pines conectores para los servomotores JR.

22. Por acabar el diseño en cuanto a la serigrafia se doto de nombres a los conectores. Se le doto de Nombre a la PCB ITACA -RIS como JDMR (iniciales del nombre del autor de este proyecto) 0617 (mes y año en que se fabricó).

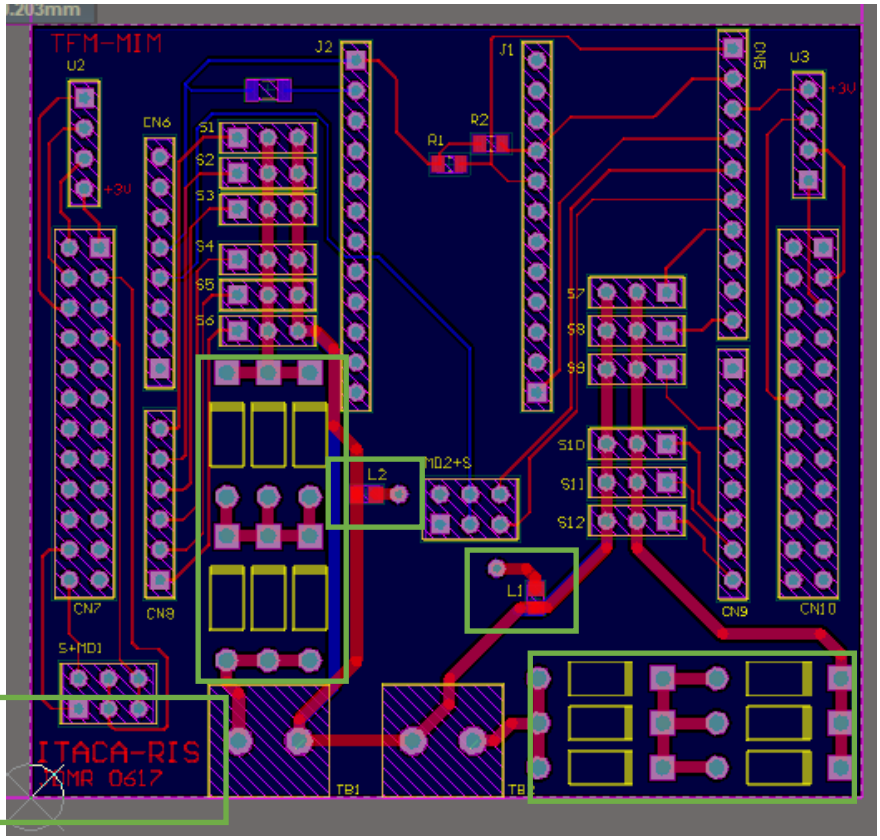


Figura 58:Bobinas más diodos rectificadores más serigrafia

23. Por ultimo y gracias al Altium una visualización en 3D podía hacerse visible antes de mandar la PCB a fabricar.
24. Una vez acabado con el diseño de la PCB, lo último que quedaba por hacer era generar con Altium los archivos GERBER de cada capa que compone la PCB. que se le enviarían al fabricante. Una vez generados el laboratorio ITACA se encargó de enviar la información al fabricante.

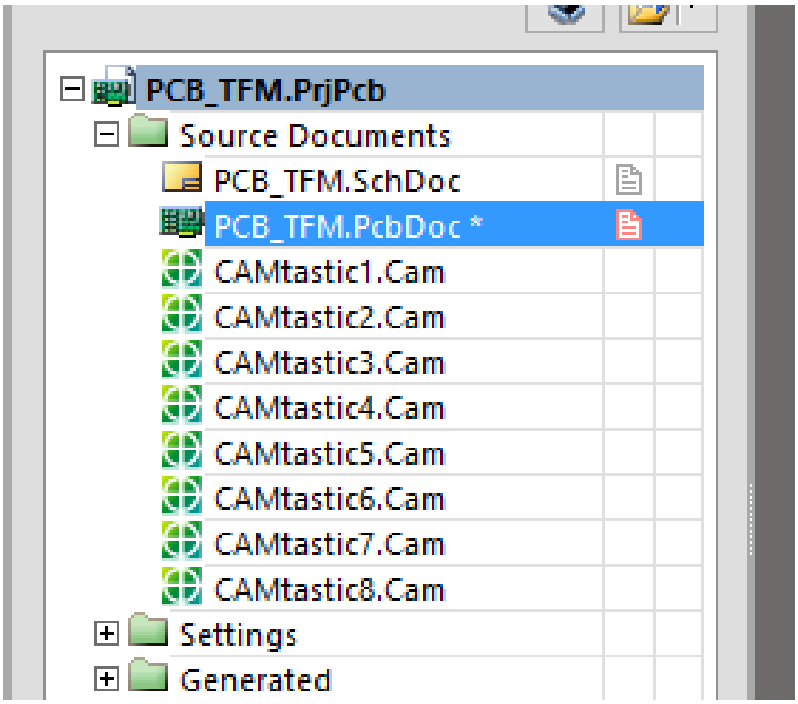


Figura 59: Archivos Gerber de cada capa.

Una vez ya teníamos la PCB fabricada, solo faltaba soldar todos los componentes que iban en ella en las huellas correspondientes con mucho cuidado. Soldarlos fue posible gracias a los puestos de soldadura que había en el laboratorio, además de todos los componentes que necesitábamos como diodos, conectores de pines conectores de alimentación, bobinas resistencias etc...

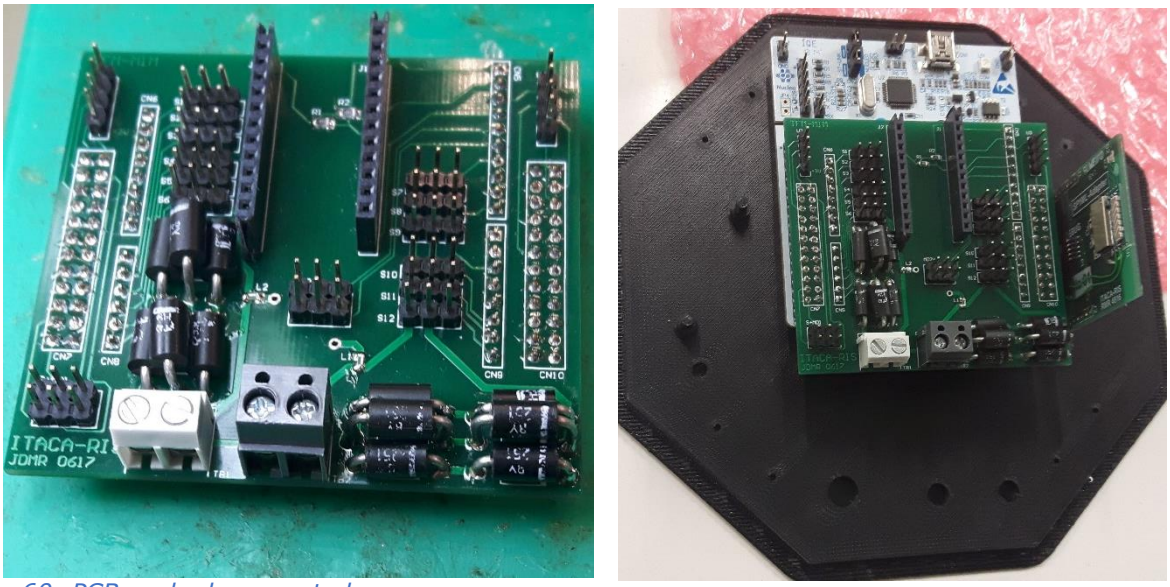


Figura 60: PCB acabada y montada

5.2.3 Conexiones adicionales para la fuente de alimentación

Con el fin de conectar la batería al módulo a la F334R8 y a la JDMR 0617 para alimentar los pines a de los servomotores a su voltaje de 5.5V a 7.4V se hicieron las conexiones como se puede ver en las imágenes. Ambos están conectados a un interruptor. La razón de esto es que a veces se prefiere el no alimentar los servomotores, pero en cambio si alimentar las placas por ejemplo para cargar programas y o realizar ajustes de configuración. De esta manera podemos, mantener el interruptor de los servomotores apagado y conectar el interruptor para las placas encendido.

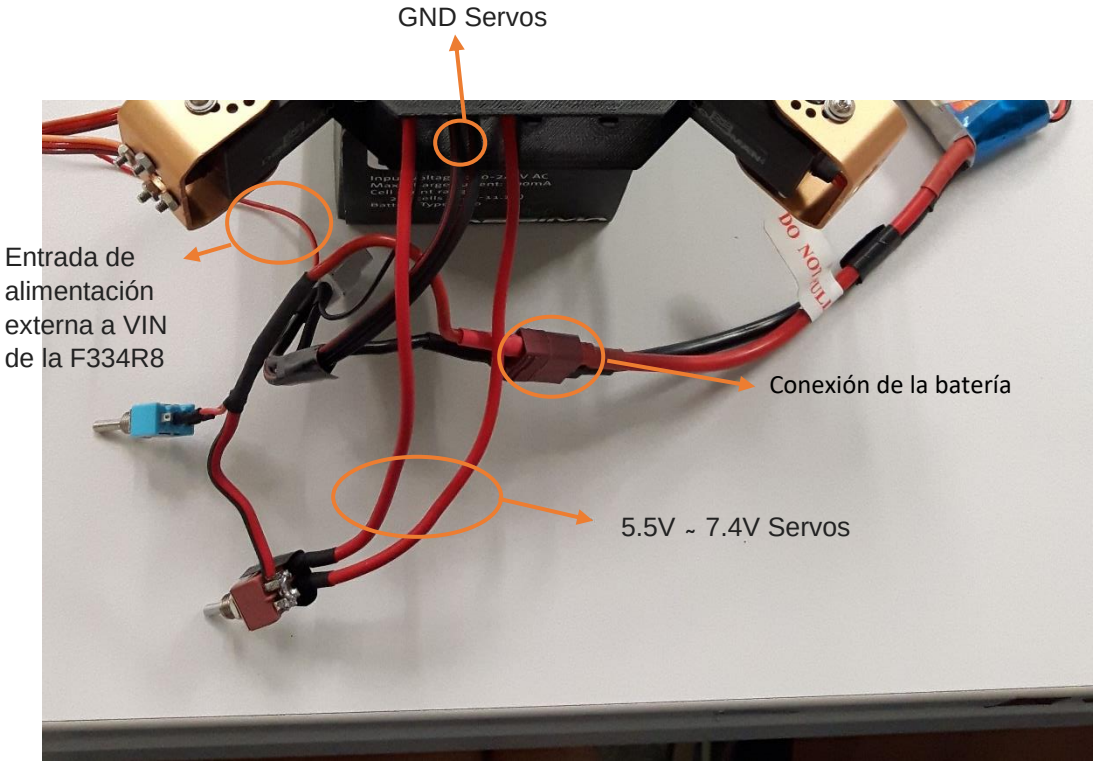


Figura 61: Conexiones para la fuente de alimentación

6 Programación en STM32CubeMX y conexiones

Los programas STM32CubeMX y µVision5 se han utilizado para programar el robot, Estos son software libre suministrado respectivamente por las empresas STMicroelectronics y KEIL. En este capítulo se explica el CubeMX. La programación en µVision5 se explica en el último capítulo, ya que el fondo teórico es esencial para la comprensión de los algoritmos utilizados.

Con STM32CubeMX, se puede generar un código que activa las entradas y salidas del procesador con el que se trabajara. En otras palabras, configuramos el software para que coincida con nuestro hardware. La configuración correcta del reloj toma una parte importante en este asunto. Por lo tanto, este capítulo se divide en 3 subcapítulos: Configuración del reloj, el robot de 2 GDL y el de 3GDL.

En primer lugar, se tiene que indicar nuestro procesador, en nuestro caso serán STM32F3DISCOVERY y el NUCLEO-F334R8.

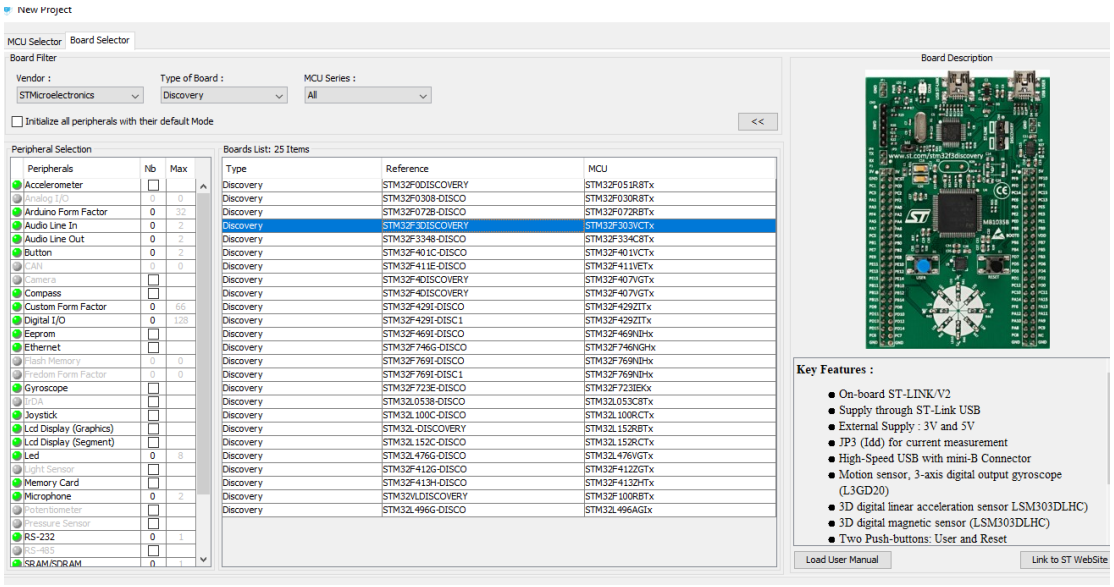


Figura 62: Eligiendo la Placa Núcleo F3DISCOVERY y su microcontrolador

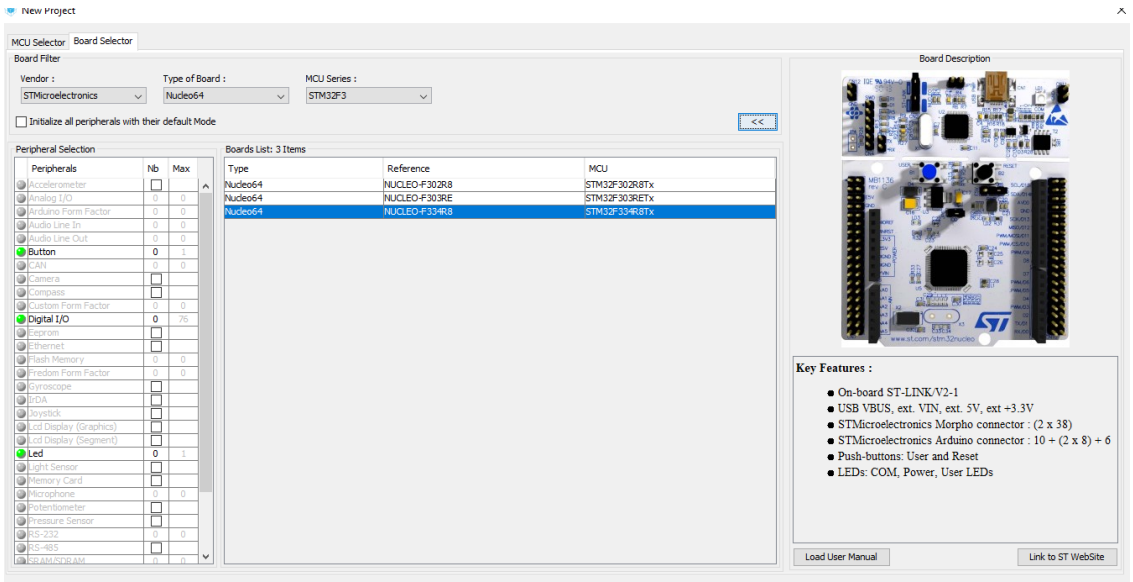


Figura 63: Eligiendo la Placa Núcleo F334R8 y su microcontrolador

Una vez que se ha seleccionado nuestra placa, se procede a configurar el reloj y a elegir las salidas que se van a usar; Los TIM para los servomotores, la UART para el módulo de radiofrecuencia de STM SP1ML, la UART para ver en la pantalla en el ordenador las salidas o variables, el ADC para el joystick y el I2C para el STEVAL-MKI124V1.

6.1 Configuración del reloj

La configuración del reloj es muy importante, casi todos los circuitos digitales utilizan una señal de reloj para sincronizarse con otros circuitos o con ella misma. Los diferentes componentes / periféricos utilizan frecuencias de reloj diferentes. También el consumo de energía está estrechamente conectado a la velocidad del reloj.

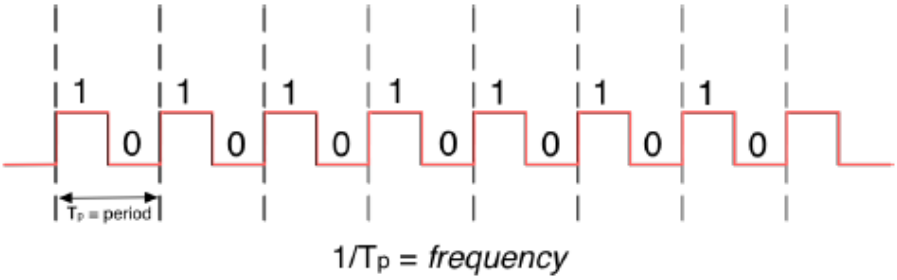


Figura 64: Frecuencia de un reloj

La STM32 puede ser sincronizada por un oscilador interno (HSI) o un oscilador de cristal externo (HSE). Con un árbol de reloj complejo, la señal de reloj se distribuye. Utilizando Prescalers y PLL (PhaseLocked Loops), la frecuencia de la señal de reloj puede ser multiplicada o dividida por un valor dado para alcanzar frecuencias distintivas. La función de PLL y Prescalers se muestra debajo.

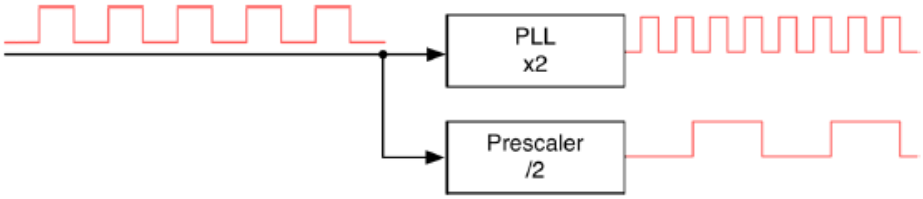


Figura 65: Función de una PLL y un Prescaler

Este es el árbol del reloj, como se puede ver en CubeMX para una placa Núcleo F334R8. Sólo se da una breve explicación ya que comprender todo el árbol del reloj es una materia compleja y que consume mucho tiempo. Como podemos observar en CubeMX en la pestaña de 'CLock Configuration' podemos configurar la frecuencia de trabajo del reloj:

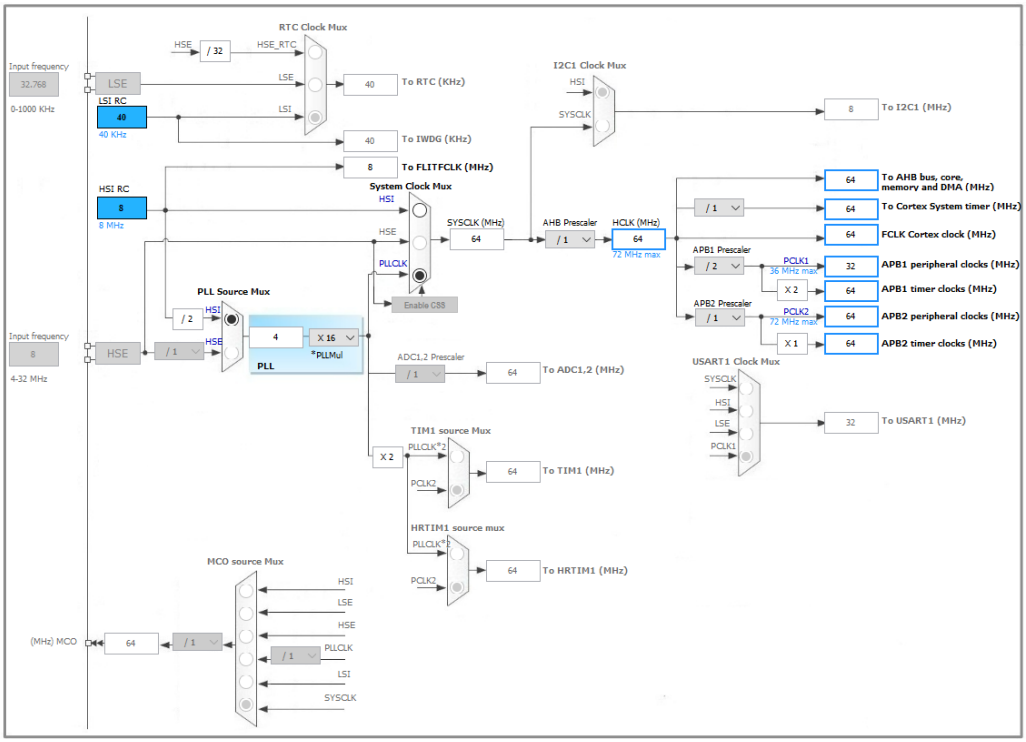


Figura 66: Árbol del reloj para el NUCLEO-F334R8

Se empieza por el RC de HSI de 8MHz, que va al Mux. del reloj del sistema (SCM). Se puede ver que podemos decidir en el SCM si usamos y reloj interno o externo y si estamos pasando por el PLL antes de ir al SCM. Desde el SYSCLK la señal de reloj se da a todos los periféricos usando PLL y prescalers para obtener la frecuencia necesaria. Tenga en cuenta que no todas las frecuencias funcionarán correctamente, se ha intentado por ejemplo 48MHz en el HCLK, y en este caso el microcontrolador no estaba funcionando correctamente.

6.2 El robot de 2GDL

Para el robot de 2GDL se utilizó 2 microcontroladores, uno para el robot y otro para el controlador o mando. Puesto que (como se dijo en el capítulo 5) la electrónica no fue diseñados por el autor de este proyecto sino por ITACA, se recibió información de un archivo el cual se podía deducir la configuración de los pines en la PCB RMR 3815. Esta configuración eléctrica es válida para los dos microcontroladores (robot y mando) que son el mismo de la F3DISCOVERY.

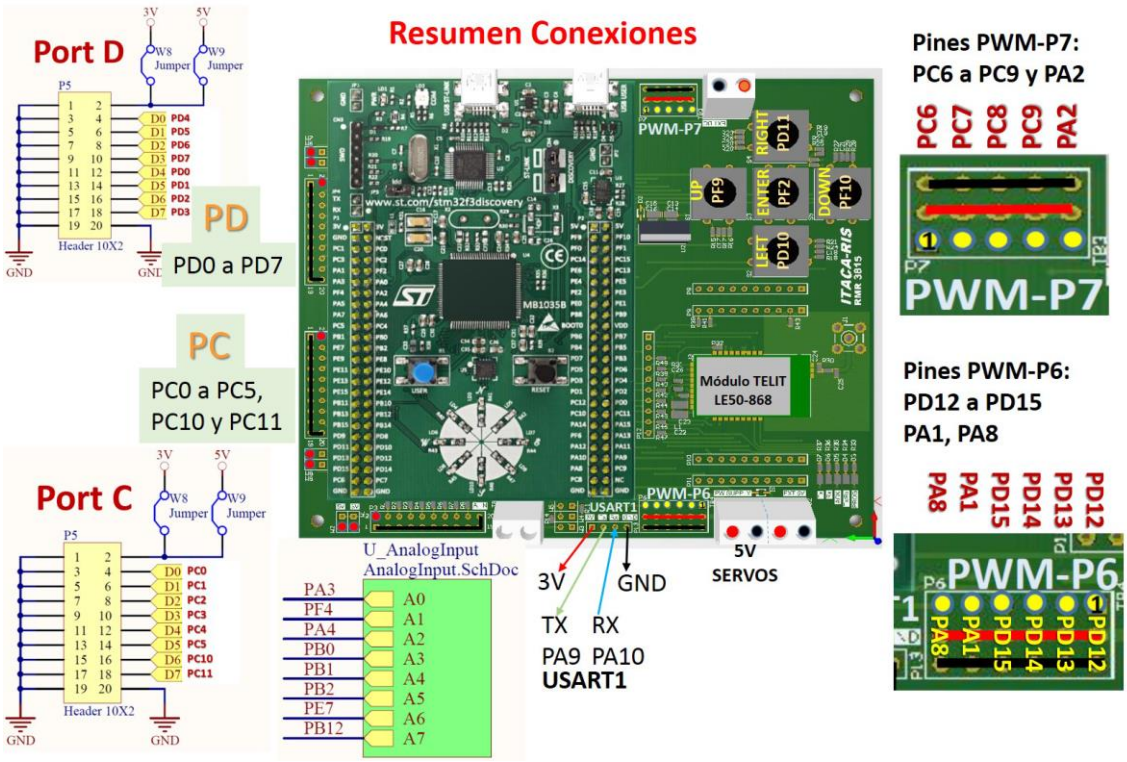


Figura 67: Conexiones de la configuración PCB & F3DISCOVERY

6.2.1 Microcontrolador del robot

En la PCB de robot se conectan los servomotores. Sólo se necesitan 8 en este caso, 2 motores por cada pierna de los 11 que hay disponibles en la RMR3815 para elegir. La elección se hizo totalmente arbitraria, aunque se buscaba la proximidad para no estirar los cables con los conectores JR. Para esta configuración de placas, también utilizamos una UART para el STM SP1ML y una UART para la conexión en serie al ordenador también. Estas conexiones se realizaron directamente en la F3DISCOVERY tanto del robot como del mando. También se podría haber utilizado una USART1 disponible en el PCB si fuera necesario.

En la figura 49 también se puede ver dónde conectar los JR a 5.5V ~ 7.4V para los servos. Para las conexiones 3,3V y GND, había multitud de pines disponibles. Una vez que estaba claro que pines se debían elegir según para qué propósito, era pues la hora de realizar la configuración en CubeMX.

La figura y las tablas siguientes le dan una visión general de la configuración de los pines de salida que tendrá el programa en STM32CubeMX:

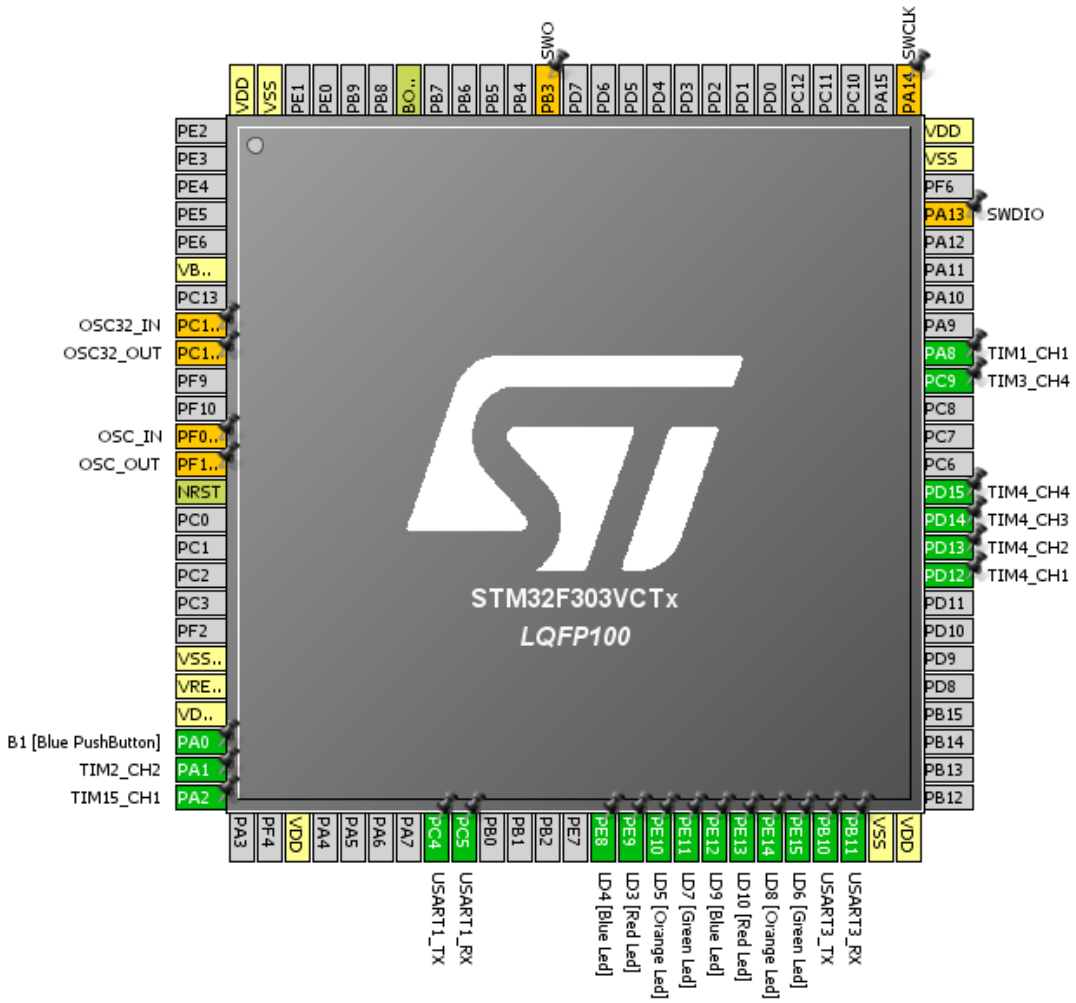


Figura 68: Configuración de los pines elegidos para el robot de 2GDL

SERVOMOTORES	SOFTWARE	HARDWARE
Derecha anterior-Coxa	TIM4_CH2	PD13
Derecha anterior-Fémur	TIM4_CH1	PD12
Derecha posterior-Coxa	TIM4_CH4	PD15
Derecha posterior-Fémur	TIM4_CH3	PD14
Izquierda anterior-Coxa	TIM3_CH4	PC9
Izquierda anterior_-Fémur	TIM1_CH1	PA8
Izquierda posterior-Coxa	TIM15_CH1	PA2
Izquierda posterior-Fémur	TIM2_CH2	PA1

Tabla 2: Temporizadores utilizados para los servomotores

UART 1	SOFTWARE	HARDWARE
GND	GND	GND
3V	3V	3V
TX (transmisión)	USART1_TX	PC4
RX (recepción)	USART1_RX	PC5

Tabla 3: UART 1 utilizada para el módulo de radiofrecuencia de STM SP1ML

UART 3	SOFTWARE	HARDWARE
GND	GND	GND
3V	3V	3V
TX (transmisión)	USART3_TX	PB10
RX (recepción)	USART3_RX	PB11

Tabla 4: UART 3 utilizada para la visualización en el ordenador con el Hyperterminal USB y TERMITE

Después de marcar las salidas y entradas del microprocesador se deben configurar los TIMER's y las UART's. Para ello en la pestaña de 'Configuration' del STM32CubeMX podemos configurar todos los parámetros a los que se les ha asignado anteriormente los pines de salida.

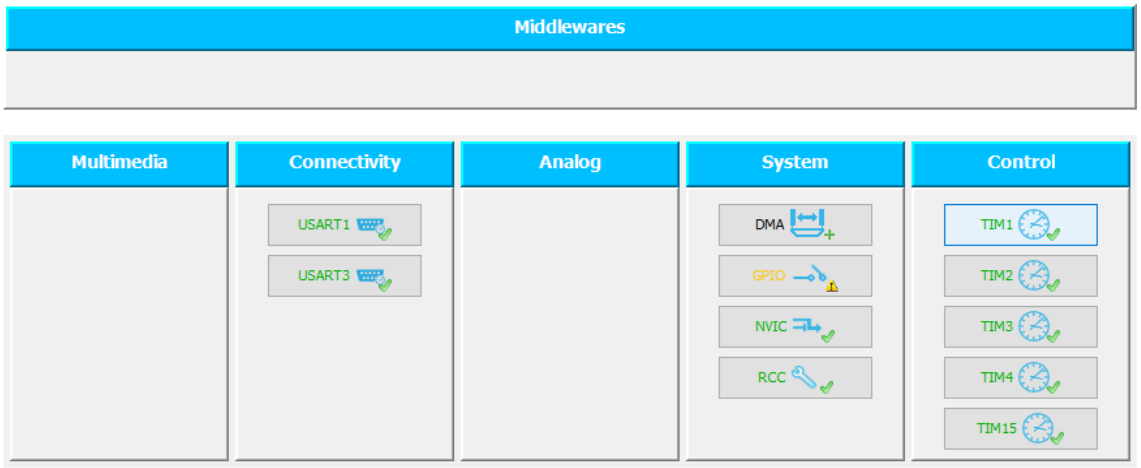


Figura 51: Ajustes

Al hacer clic en el TIM1, aparece la siguiente pantalla:

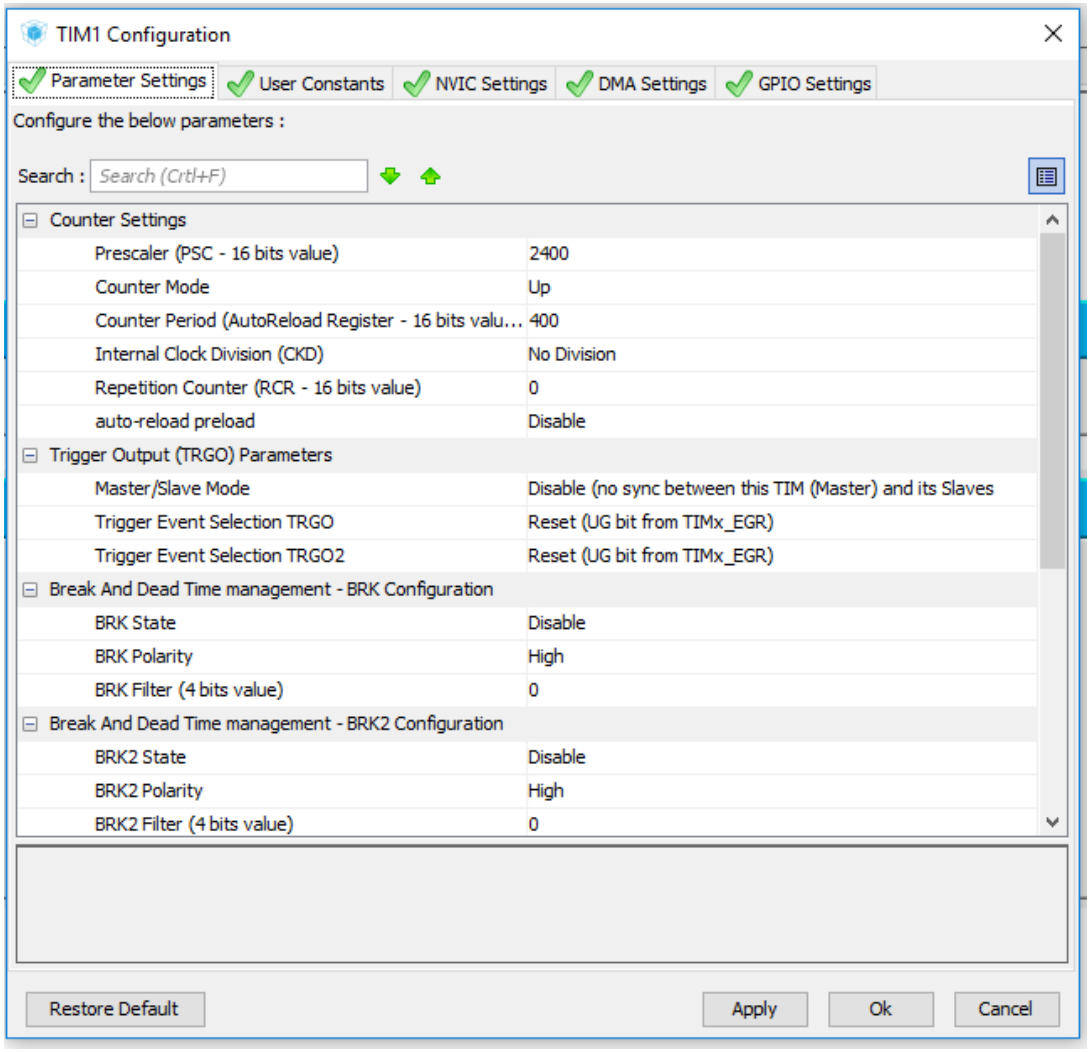


Figura 69: Configuración del TIMER 1

Puesto que se quería enviar una señal pulsada de 50 Hz al servomotor con un ancho de pulso variable, sé tuvo que cambiar los ajustes del contador. El Prescaler se puso en 2400, el período de contador en 400. El cómo se llegó a estos valores se describe en el capítulo 9. Para los demás temporizadores (T2, T3, T4), se ajustaron los mismos parámetros a las mismas cifras elegidas.

Al hacer clic en la USART1, tenemos la siguiente pantalla:

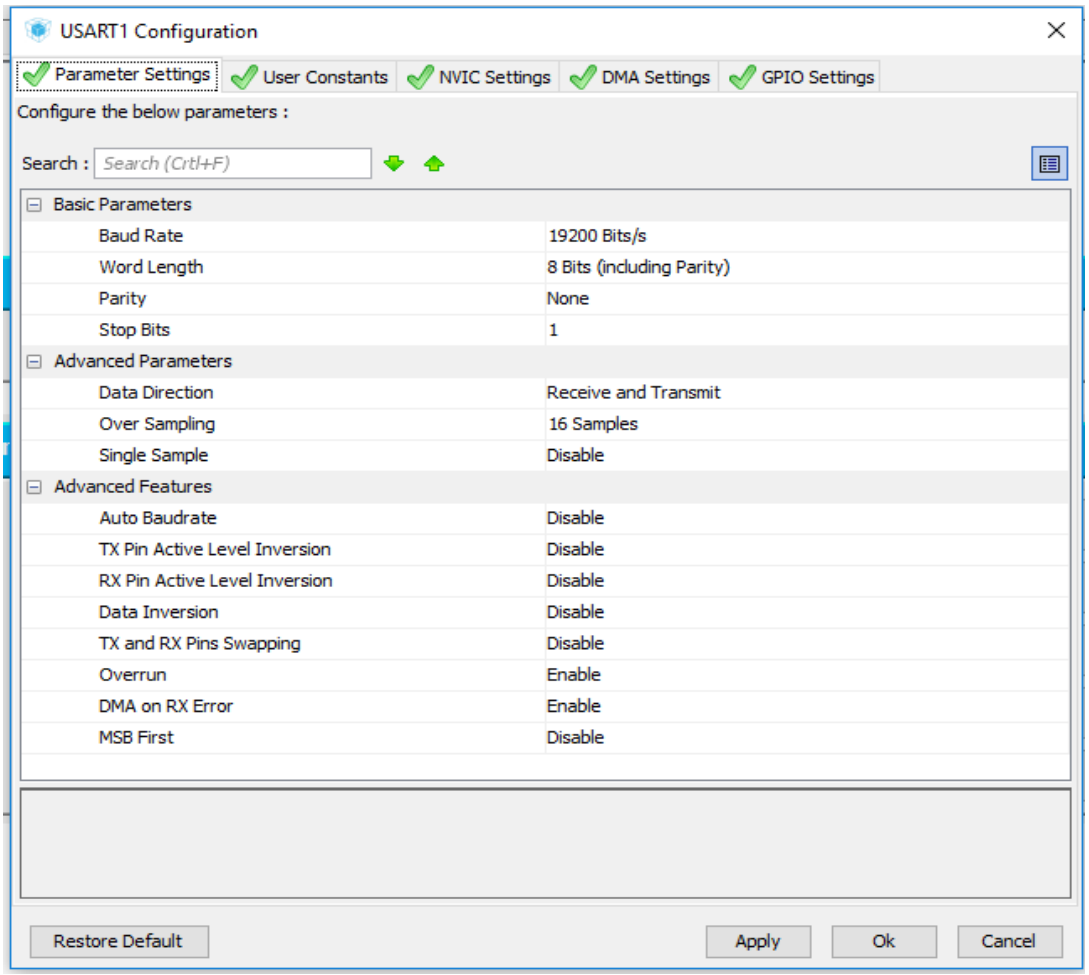


Figura 70: Configuración de parámetros de USART1

En los ajustes de parámetros, la velocidad en baudios especifica la rapidez con la que se envían los datos. Se eligen por una velocidad en baudios de 19200 bits / s. Se selecciona una longitud de palabra de 8 bits ya que se están enviando caracteres de 8 bits. Puede encontrar más información al respecto en el capítulo 12.

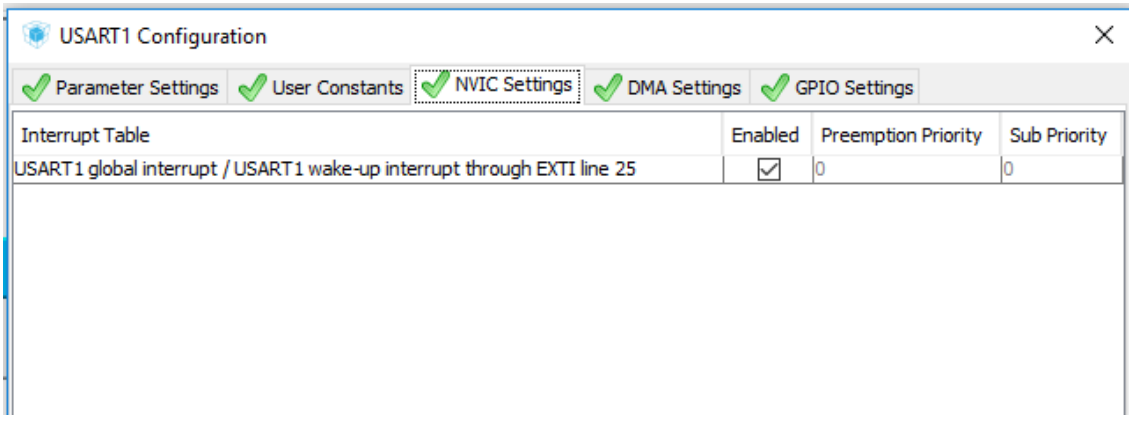


Figura 71: Configuración de NVIC de USART1

También se debe activar la alarma de activación, lo que significa que cuando se reciben datos se llama a una petición de interrupción, la cual tendrá prioridad sobre el programa principal. En el cuadrúpedo 2GDL no se leen los datos a través de las interrupciones, pero sólo se utiliza la interrupción como despertador. En cuanto al USART3, los ajustes son los mismos. Aquí la interrupción no estaba habilitada.

Lo único que queda por hacer es conectar los cables correctamente.

La conexión de los servomotores y la fuente de alimentación debe ser bastante sencilla con la ayuda de la figura 49. Por consiguiente, no se da más explicación.

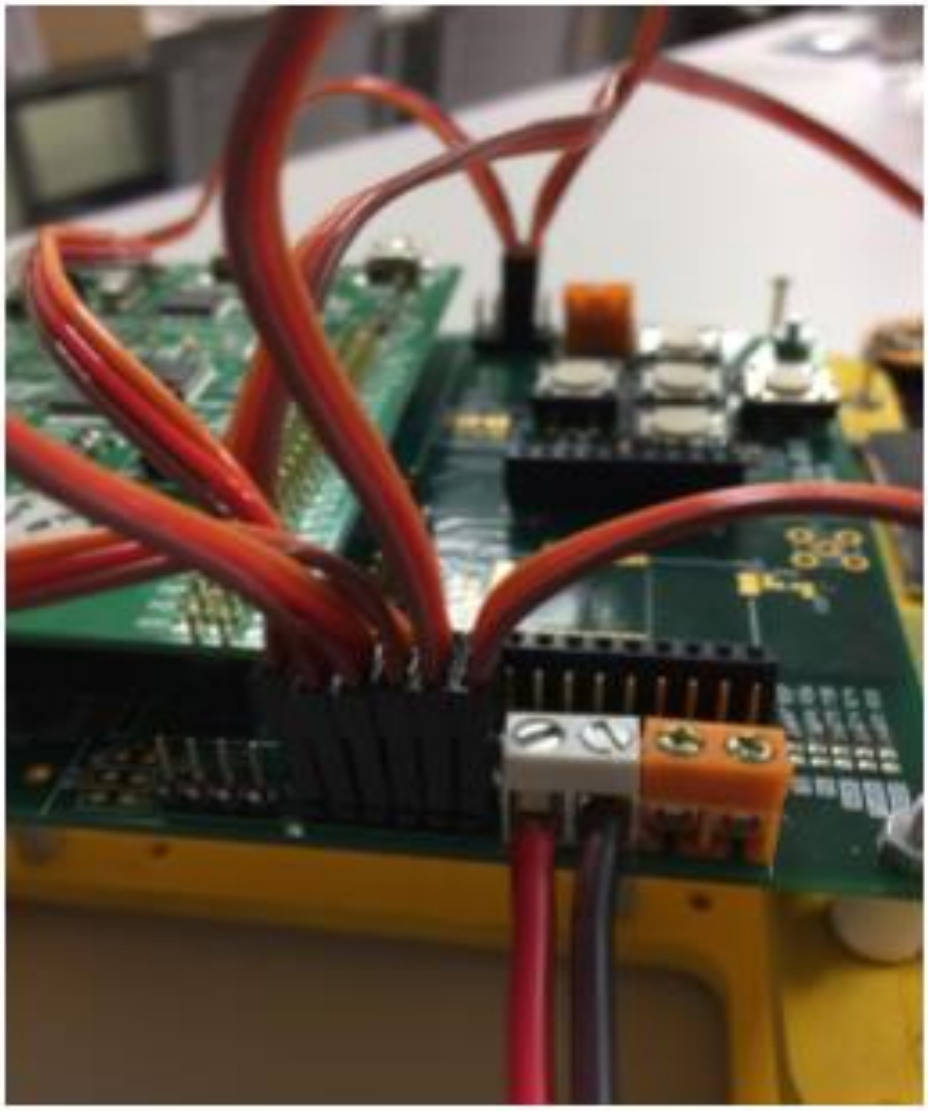


Figura 72: Conexión de los servomotores

Para el Hyperterminal Serie a USB se puede ver que ni el pin de 3V ni el pin 5V estaban conectados a la placa. La razón que está detrás de esto es que el dispositivo ya estaba alimentado por el ordenador mediante el cable USB. Es importante señalar que necesitamos conectar el receptor del hyperterminal con el transmisor de la placa Núcleo PB10 y viceversa con el transmisor del hyperterminal con el receptor de la placa PB11. Además de conectar las masas GND.

Figura 73: Conexión de la serie a USB

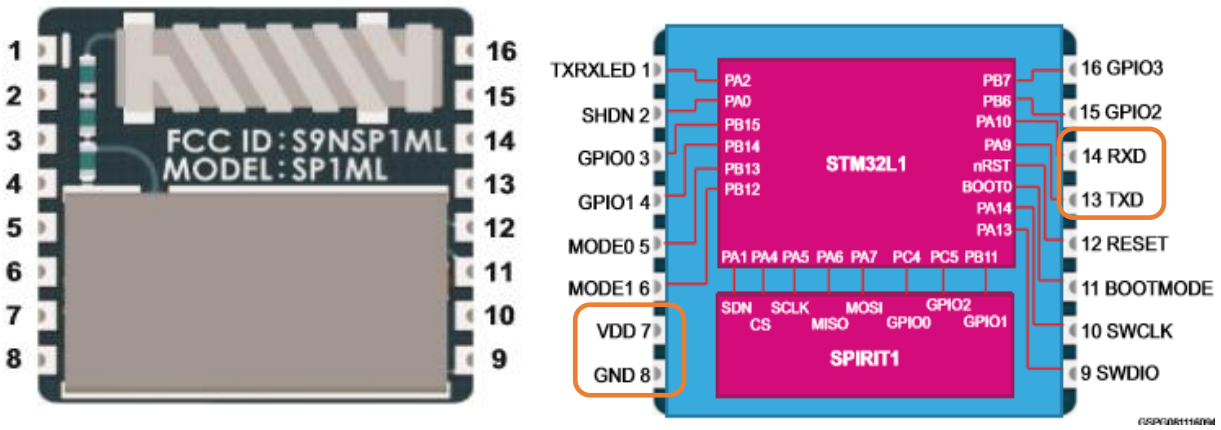
Con la ayuda de las figuras 57 - 59 debe estar claro cómo hacer las conexiones del módulo SP1ML. La explicación es similar a la explicación del Hyperterminal serial a USB.

El módulo de RF STEVAL SP1ML de STMicroelectronics se debe soldar a la PCB Adaptadora proporcionada por ITACA. Esto hace que las conexiones queden más claras y más fáciles para conectar. Además, esta PCB RMR4616 es de tamaño pequeño 3*4 cm y hace que sea perfecta para situar en el cuerpo de los robots. Las conexiones por tanto de Vcc, GND, TX y RX están sacadas con pines por la PCB.

A continuación, solo faltaría conectarlas del módulo a la placa del microcontrolador. Las conexiones se realizarían de la siguiente forma:

Conexiones del SP1ML a la UART elegida en la Placa Nucleo, por ejemplo, la UART1 de la F3Discovery:

- Pin 7 VDD : 3V
- Pin 8 GND : GND
- Pin 13 TXD : PC5 (UART1)
- Pin 14 RXD : PC4 (UART1)



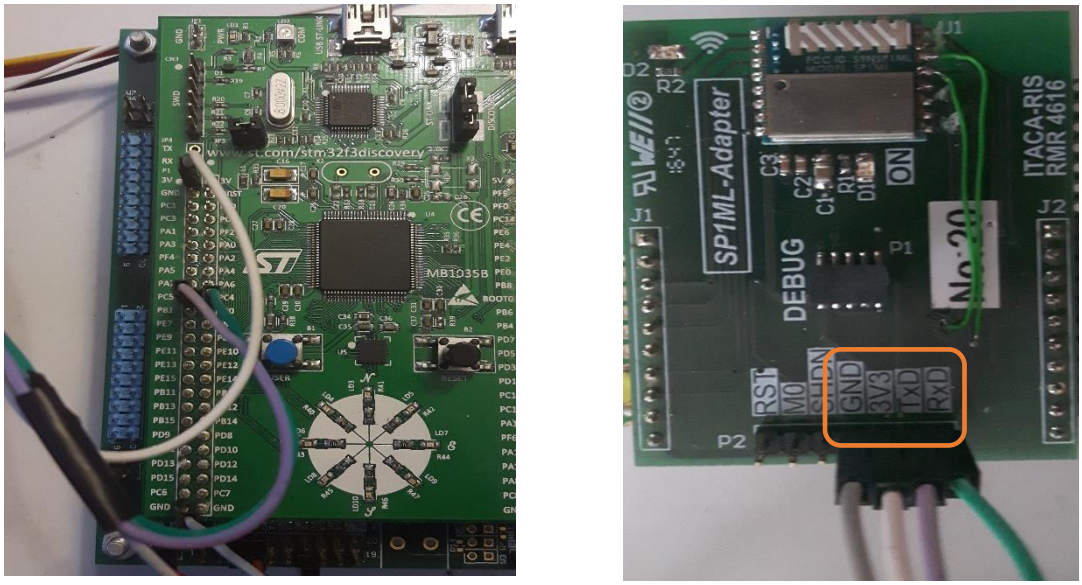


Figura73, 74, 75 y 76: Conexión del módulo de RF SP1ML

6.2.2 Microcontrolador del Mando inalámbrico

En el microcontrolador del controlador o mando también están las conexiones UART. Se eligieron para los mismos pines que para el microcontrolador del robot 2GDL. Una vez más, 3,3V y GND estaba disponible en muchos alfileres. En este microcontrolador, también era necesario una conexión ADC para el joystick. Estos pines también fueron elegidos al azar. Siempre que pudieran utilizarse como conexiones ADC, no importaba qué pines fueran.

La figura y las tablas siguientes le dan una visión general de la configuración de los pines en STM32CubeMX:

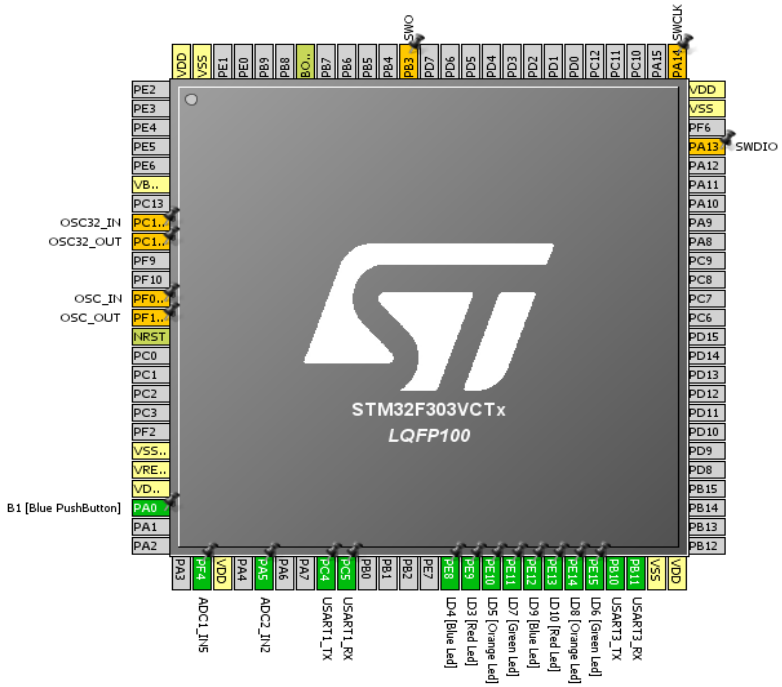


Figura 77: Configuración de los pines del mando inalámbrico del cuadrúpedo de 2GDL

ADC	SOFTWARE	HARDWARE
GND	GND	GND
3V	3V	3V
Eje X	ADC1_IN5	PF4
Eje Y	ADC2_IN2	PA5
Botón B	N/A	N/A

Tabla 5: ADC utilizado para el joystick

UART 1	SOFTWARE	HARDWARE
GND	GND	GND
3V	3V	3V
TX (transmisión)	USART1_TX	PC4
RX (recepción)	USART1_RX	PC5

Tabla 6: La UART utilizada para el SP1ML

UART 3	SOFTWARE	HARDWARE
GND	GND	GND
3V	3V	3V
TX (transmisión)	USART3_TX	PB10
RX (recepción)	USART3_RX	PB11

Tabla 7: UART utilizado para la visualización en el ordenador

Después de marcar las salidas y entradas del microprocesador se configuro los ADC's y las **UA**RT's. Las UART's reciben los mismos ajustes que para el robot 2GDL. Los ADC's quedaron en la configuración predeterminada.

Lo único que quedaba por hacer era conectar los cables correctamente. Puesto que esto era igual que para el robot 2GDL, no se va a explicar esto otra vez, excepto las conexiones del Joystick que son las siguientes:

6.3 El robot de 3GDL

También para el cuadrúpedo de 3GDL se utilizaron 2 microcontroladores, uno para el robot y otro para el controlador. Para el microcontrolador del robot se definió la configuración de los pines en el diseño la PCB JDMR 0617. La configuración de las clavijas del microcontrolador del controlador era, como se ha dicho anteriormente, definida por la PCB de ITACA (aunque también se hubiera podido crear con la configuración de una F334R8, por ejemplo).

6.3.1 Microcontrolador del robot

Para la configuración de pin de este microcontrolador nos basamos en las tablas 2 y 3. Se utilizaron 12 motores para las patas: 3 motores por pierna para controlar los 3 ejes. Además de los 12 servo pines para los motores 2 más fueron añadidos para otros fines. Si por ejemplo quisiéramos añadir un sensor de infrarrojos al robot en el futuro, no tendríamos que sacar nuestras herramientas de soldadura, pero sólo tendríamos que hacer la conexión con las clavijas ya instaladas y hacer la configuración en CubeMX . Además de los servomotores, hemos utilizado nuevamente 2 UARTs, uno para el SP1ML y otro para la conexión en serie al ordenador. Para el giroscopio y el E-Compass I2C las conexiones eran necesarias y configuradas así.

La figura y las tablas siguientes le dan una visión general de la configuración de los pines en STM32CubeMX:

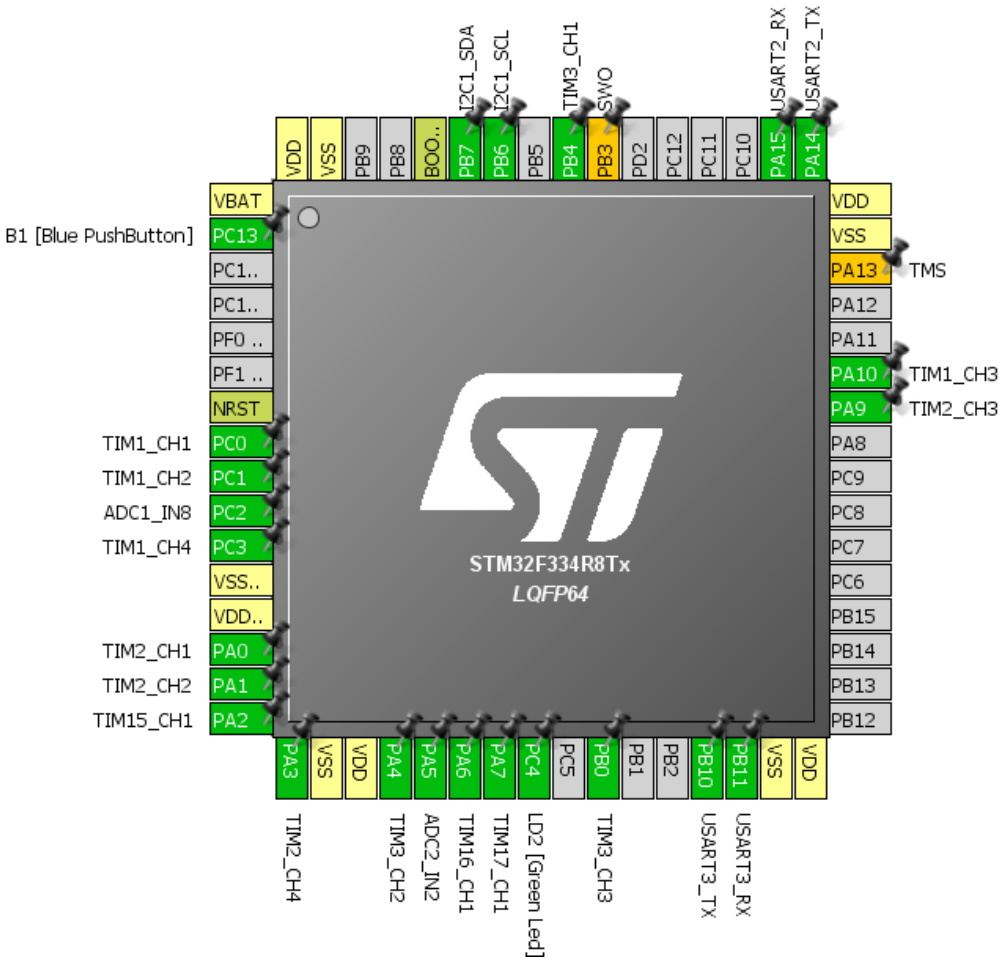


Figura 78: Configuración de los pines del robot de 3GDL

SERVOMOTORES	SOFTWARE	HARDWARE
Piernas		

Derecha anterior	Coxa	TIM3_CH1	PA6 DEBERIA SER PB4
	Femur	TIM2_CH3	PA9
	Tibia	TIM17_CH1	PA7
Derecha posterior	Coxa	TIM1_CH3	PA10
	Fémur	TIM15_CH1	PA2
	Tibia	TIM2_CH4	PA3
Izquierda anterior	Coxa	TIM3_CH2	PA4
	Fémur	TIM2_CH2	PA1
	Tibia	TIM2_CH1	PA0
Izquierda posterior	Coxa	TIM3_CH3	PB0
	Fémur	TIM1_CH2	PC1
	Tibia	TIM1_CH1	PC0
Pines Extras para servomotores			
Pin Extra para servomotor 1		TIM16_CH1	PA6
Pin extra para servomotor 2		TIM1_CH4	PC3

Tabla 8: Temporizadores utilizados para los servomotores

UART 2	SOFTWARE	HARDWARE
GND	GND	GND
3V	3V	3V
TX (transmisión)	USART2_TX	PA14
RX (recepción)	USART2_RX	PA15

Tabla 9: La UART utilizada para el SP1ML

UART 3	SOFTWARE	HARDWARE
GND	GND	GND
3V	3V	3V
TX (transmisión)	USART3_TX	PB10
RX (recepción)	USART3_RX	PB11

Tabla 10: UART utilizado para la visualización en el ordenador

SENSOR GIRO./ACEL. STEVAL-MKI124V1	SOFTWARE	HARDWARE
GND	GND	GND
3V	3V	3V
SDA	I2C1_SDA	PB7
SCL	I2C1_SCL	PB6

Tabla 11: I2C utilizado para el chip-sensor Giroscopio y Acelerómetro STEVAL-MKI124V1

ADC's para el sensor de distancia SHARP	SOFTWARE	HARDWARE

GND	GND	GND
5V	5V	5V
Señal 1 para sensor de distancia Sharp	ADC2_IN2	PA5
Señal 2 para sensor extra	ADC1_IN8	PC2

Tabla 12: ADC utilizado para el medidor de distancia SHARP

Después de marcar las salidas y entradas del microprocesador se tuvo que configurar los TIMERS, UARTs e I2C.

De nuevo se buscó una señal pulsada de 50 Hz para el servomotor con un ancho de impulso variable. Como antes, se cambió la configuración del contador. Esta vez el Prescaler se puso en 1600, el período de contador en 800. **Estos valores dieron un control más preciso sobre los motores y más rango de exactitud en los ángulos que los servomotores podían llegar a fijar**. Para obtener más información, consulte el capítulo 9.

En los ajustes de parámetros de las USART nuevamente se cambió la velocidad en baudios. Elegimos por una velocidad en baudios de 115200 bits / s. **Esto nos dio una comunicación más rápida que antes y con menos interferencias y posibles ruidos**. Se selecciono de nuevo una longitud de palabra de 8 bits ya que se estaba enviando caracteres de 8 bits. Para UART1 se tuvo que, como antes, activar la interrupción.

Para el SDA y SCL no se realizaron cambios.

Lo único que quedaba por hacer era conectar los cables correctamente. Con el siguiente esbozo del escudo JDMR 0617 compatible las conexiones deben quedar claras. Después de todo es cómo se diseñó esta PCB en el capítulo 5.2.2. Las conexiones para las UART's se hicieron de la misma manera que para el robot de 2GDL y por lo tanto no se explican de nuevo.

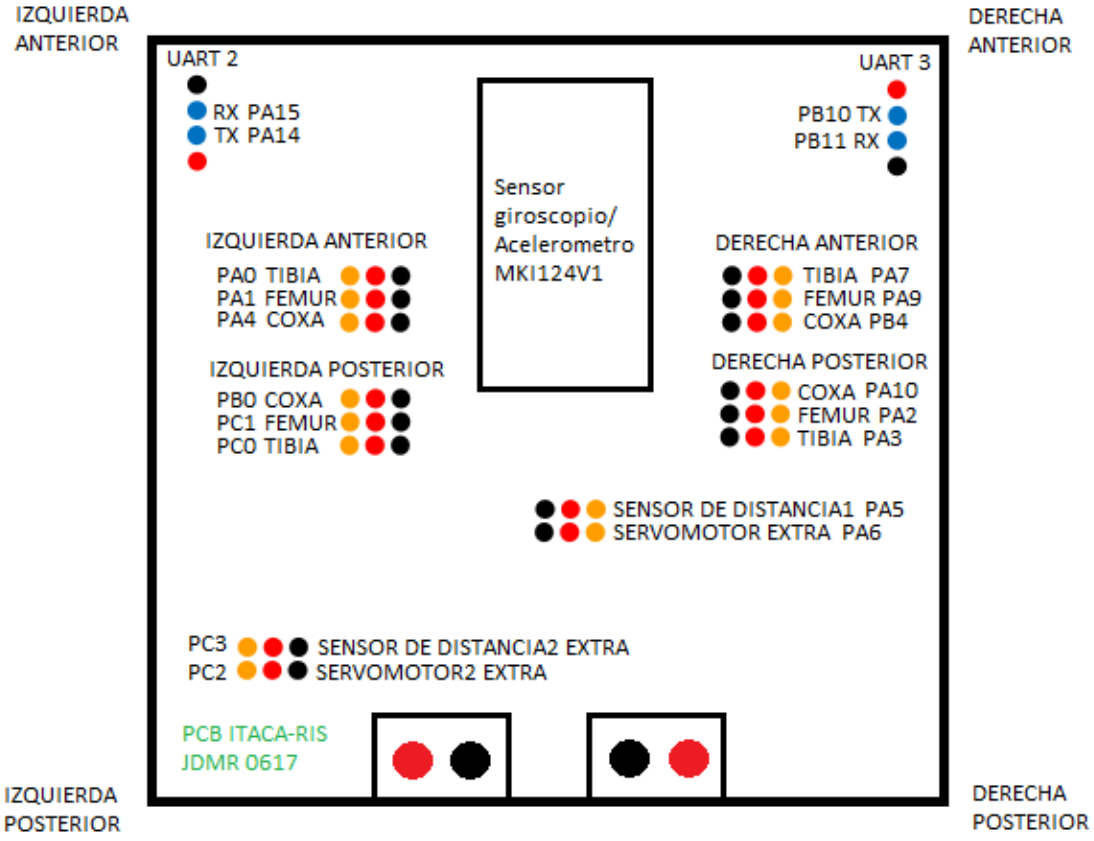


Figura 79: Esquema de las conexiones para la PCB JDMR 0617

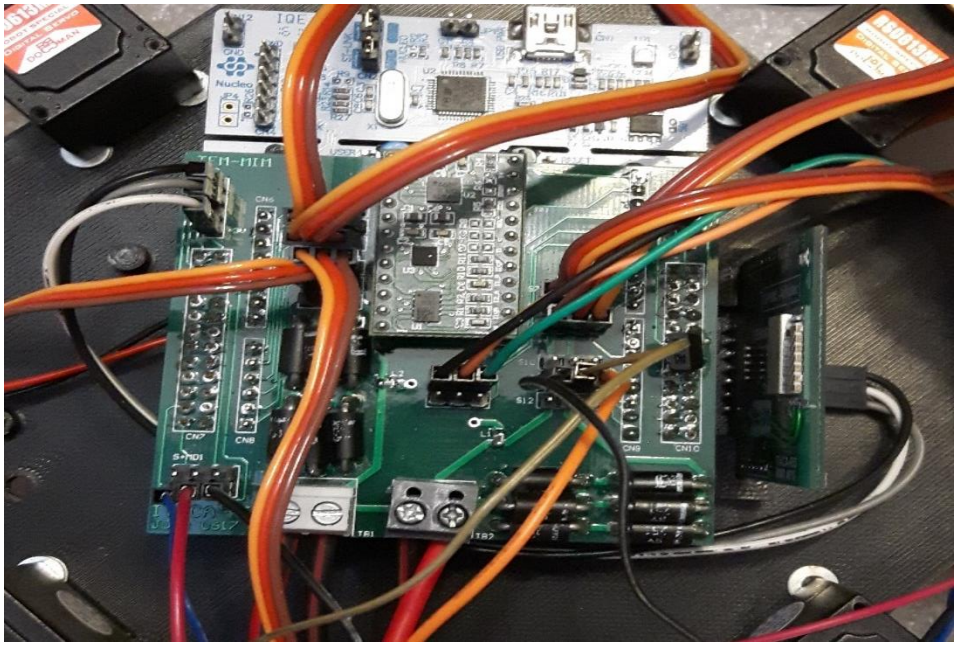


Figura 80: Conectado los servos, el módulo de RF y el chip sensor MEMS en el robot de 3GDL

6.3.2 Microcontrolador del mando inalámbrico

El microcontrolador del controlador inalámbrico del cuadrúpedo 3GDL se basó nuevamente en la figura 49. Se hicieron algunas mejoras con respecto a los pines utilizados. Elegimos solamente los pines de la placa ITACA ya que los pines son más largos y los cables no caen tan rápido como en la placa Discovery. Además, se añadieron los pulsadores para elegir en qué opción queremos operar el robot. Más información sobre esto se puede encontrar en el capítulo 12. Al igual que el microcontrolador del controlador inalámbrico del robot 2GDL, UARTs y ADCs eran necesarios. Pero como hemos añadido diferentes opciones de operación, también necesitamos GPIOs.

La figura y las tablas siguientes le dan una visión general de la configuración de los pines en STM32CubeMX:

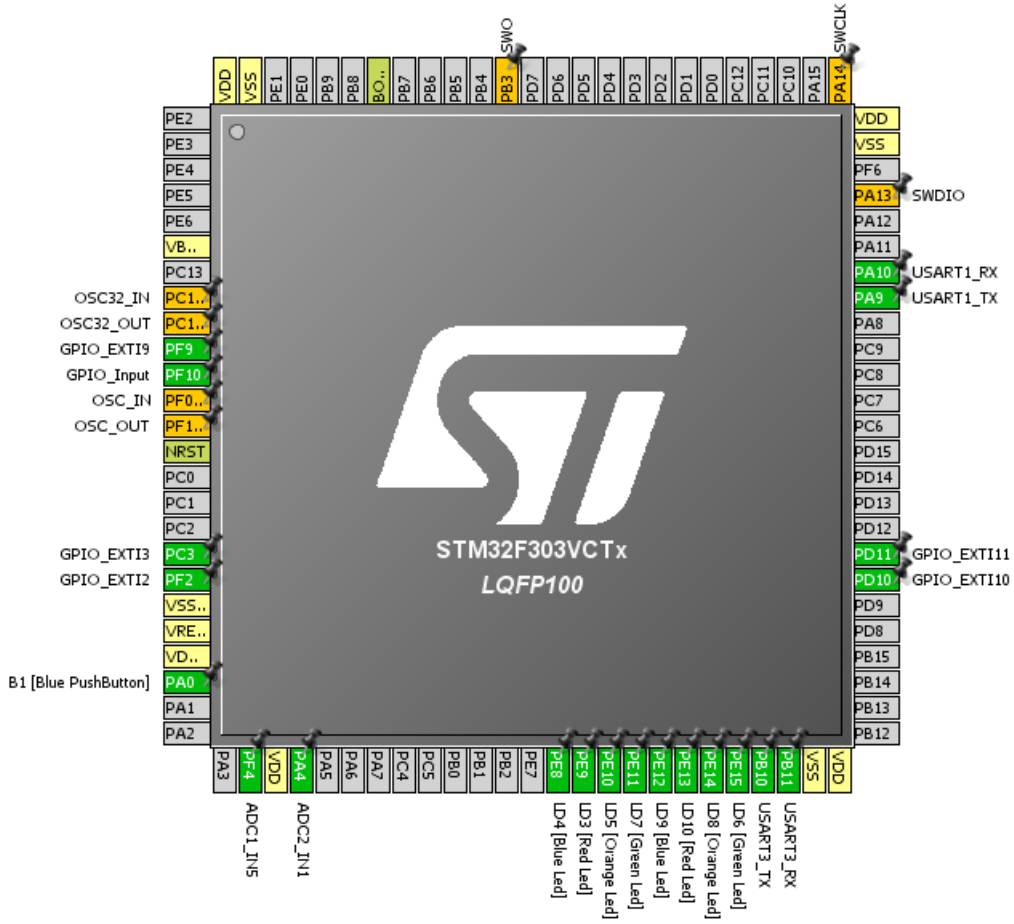


Figura 81: Configuración de los pines del controlador inalámbrico del robot de 3GDL

ADC	SOFTWARE	HARDWARE
GND	GND	GND
3V	3V	3V
Eje X	ADC1_IN5	PF4
Eje Y	ADC2_IN2	PA5
Boton B	N/A	N/A

Tabla 13: ADC y GPIO utilizados para el joystick

UART 1	SOFTWARE	HARDWARE
GND	GND	GND
3V	3V	3V
TX (transmisión)	USART1_TX	PA9
RX (recepción)	USART1_RX	PA10

Tabla 14: La UART utilizada para el SP1ML

UART 3	SOFTWARE	HARDWARE
GND	GND	GND
3V	3V	3V
TX (transmisión)	USART3_TX	PB10
RX (recepción)	USART3_RX	PB11

Tabla 15: UART utilizado para la visualización en el ordenador

GPIO para la botonera	SOFTWARE	HARDWARE
Rotación en X, Y	GPIO_EXTI10	PD10
Caminar en X y Rotación	GPIO_EXTI11	PD11
Posición de contracción y guardado	GPIO_EXTI2	PF2
Movimiento del cuerpo en X	GPIO_EXTI9	PF9
Movimiento del cuerpo en Z y rotación en Z	GPIO_Input	PF10

Tabla 16: GPIOs utilizados para la botonera del mando

Después de marcar las salidas y entradas del microcontrolador hemos configurado los ADC, UARTs y GPIOs. A la UART1 se dio una velocidad en baudios de 115200 bits / s. Como se puede notar, no se dio una velocidad inferior o superior a la que se le dio al robot. Las dos fuerón iguales a una velocidad de 115.200 bits / s que mejora la comunicación. De nuevo para UART3 elegimos una velocidad en baudios aleatoria también de 115200 Bits / s. Esta velocidad en baudios realmente no importaba. Lo único que era importante para el Hyperterminal Serial a USB es que se seleccione la misma velocidad en baudios tanto en el TERMITE (programa para el HyperTerminal) como se define para la UART, para poder visualizar los datos que se envían y se reciben.

Los ADC's quedaron en las opciones predeterminadas. Además de los GPIO no se modificaron.

Lo único que quedaba por hacer era conectar los cables correctamente. Dado que el método es el mismo que para el mando inalámbrico del 2GDL, no se volverá a explicar esto de nuevo.

Finalmente, una vez que todos los ajustes deseados habían sido establecidos y las conexiones fueron hechas, se utilizó el programa para generar el código en lenguaje C. Con la ayuda de "µVision 5 de KEIL", que es un intermediario para STM32CubeMX, el código fuente que se recibe de las configuraciones hechas en el STM32CubeMX fue importado inmediatamente al µVision 5 para así poder iniciar la codificación que ser explica más adelante en el capítulo 14.

7 Implementaciones

7.1 El Cuadrúpedo de 2GDL

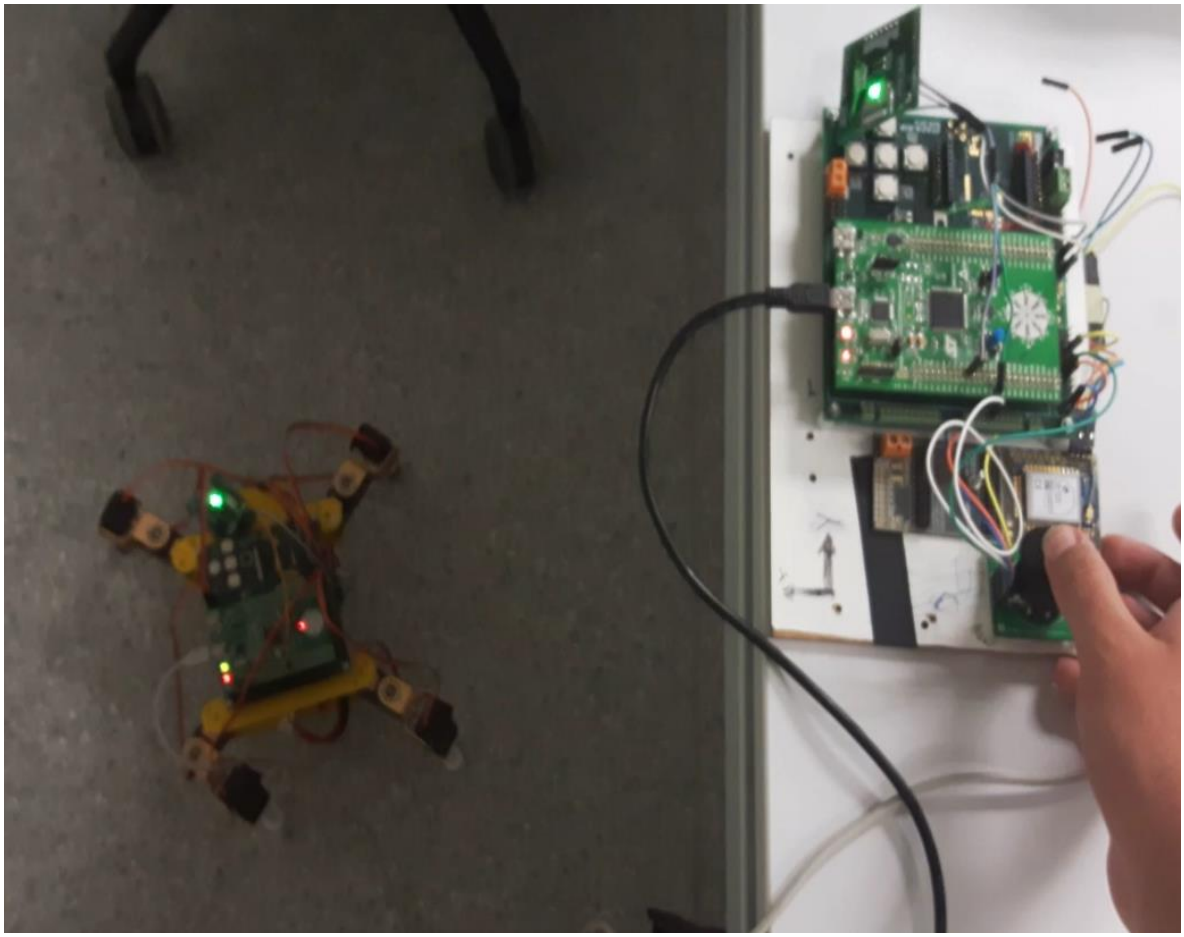


Figura 82: Montaje del robot cuadrúpedo 2DOF y su controlador

7.2 El Cuadrúpedo de 3 GDL

FIGURA 83

8 Cinemática

Antes de empezar a discutir la cinemática utilizada en el proyecto, se necesitó distinguir la cinemática inversa y directa. La diferencia entre estas dos es bastante sencilla. La cinemática de avance calcula las posiciones dadas los ángulos de articulación, mientras que la cinemática inversa calcula los ángulos de articulación dados las posiciones, la misma se puede extender a las velocidades. Además, se ha utilizado un sistema de coordenadas cartesianas, por lo que las posiciones son X, Y y Z valores. La siguiente figura muestra la diferencia entre los dos tipos de cinemática.

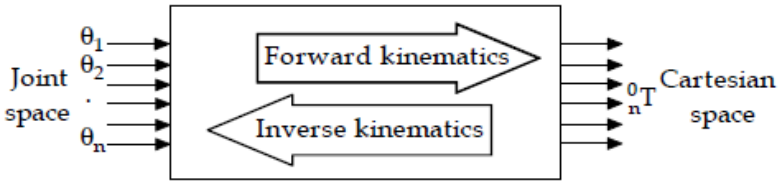


Figura 84: La representación esquemática de la cinemática hacia adelante y hacia la inversa

8.1 Cinemática cuadrúpeda 2 GDL

En primer lugar, se hablará de la cinemática para el cuadrúpedo 2GDL que obviamente es mucho más directa que la cuadratura 3GDL puesto que hay 1GDL menos. Para derivar las ecuaciones necesarias para el cuadrúpedo 2GDL se utilizó cinemática inversa, así como se explicó anteriormente se utilizó las coordenadas cartesianas que son las posiciones finales y se calcularan los ángulos de articulación. Para hacer la derivación más comprensible se puede ver la vista superior del 2GDL cuadrúpedo en la siguiente figura. Como puede ver, se utilizó dos sistemas de coordenadas cartesianas, un sistema cartesiano del cuerpo y un sistema cartesiano de la pierna. Para el movimiento total del robot se necesitó usar el sistema cartésico del cuerpo. Para la cinemática inversa de cada pierna se utilizó el sistema cartesiano de la pierna.

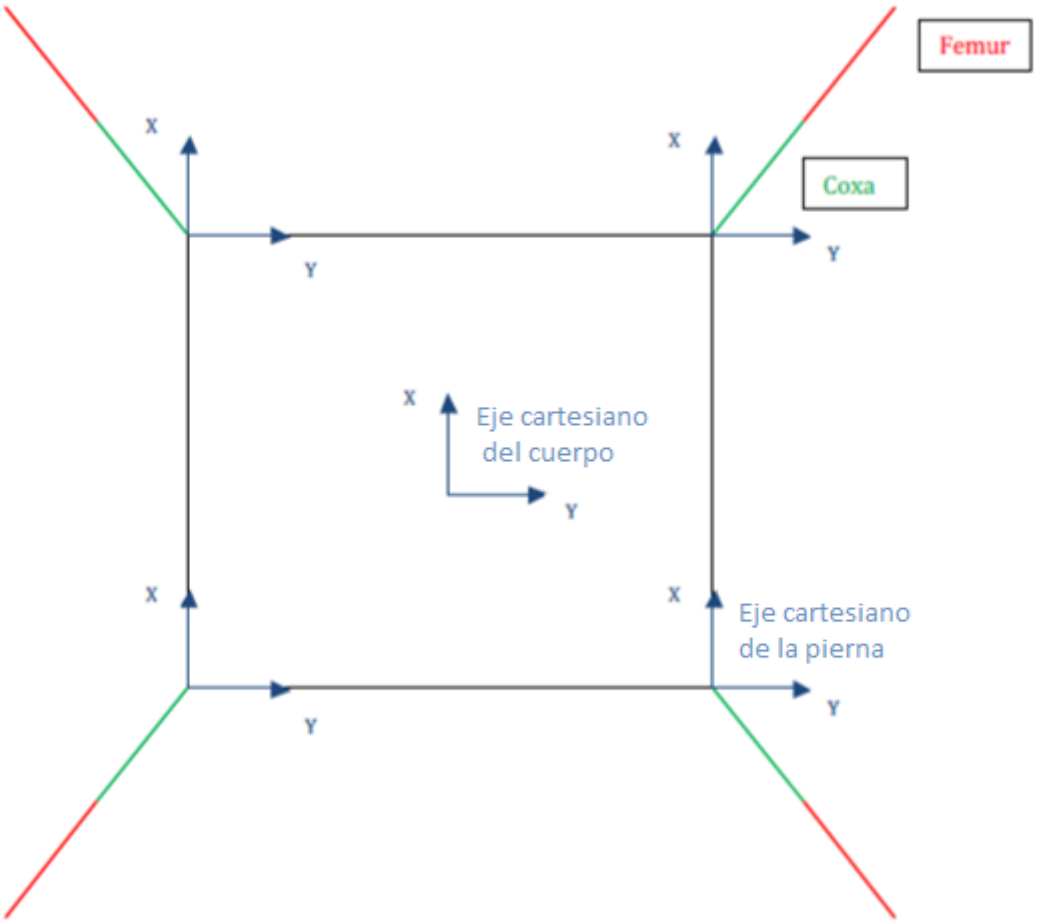


Figura 85: Cuadrúpedo 2 GDL - vista superior

Sabiendo que los nombres coxa y fémur son nombres latinos en la ciencia biológica para indicar las diferentes partes de la pierna de un insecto. Estos nombres son adoptados por la robótica y a menudo serán usados a través de este papel. La coxa es igual a la cadera, el fémur es igual al muslo y la tibia que se utiliza adicionalmente en el cuadrúpedo 3GDL es igual a la espinilla. Estos nombres se van a utilizar para indicar las articulaciones y la parte de la pierna misma.

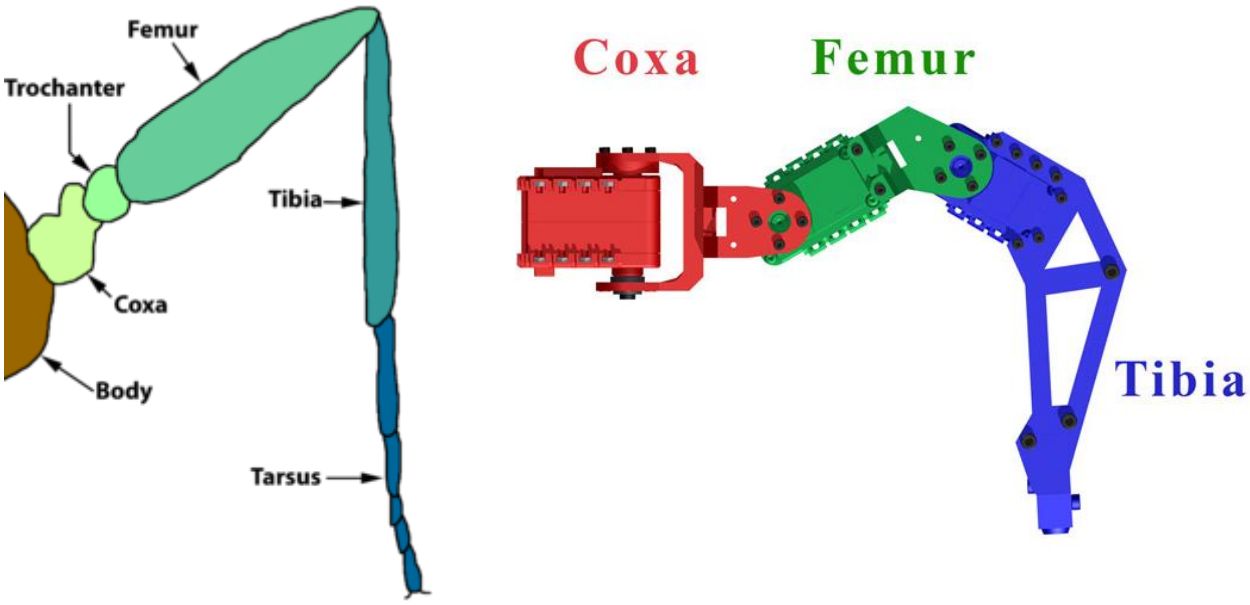


Figura 86: La anatomía de la pierna de un insecto ajustada a la de una pierna del robot cuadrúpedo

Lo primero que hay que aprender es que como el robot tiene 2 servomotores en cada pierna, por lo tanto, sólo 2 grados de libertad, sólo se podría controlar los 2 ejes siendo en este caso el eje X e Y. Si necesitamos por ejemplo mover una pierna adelante y así aumentar el valor de X que inherentemente también cambiaría el valor de Z. Sin embargo, si se utilizan 3 servomotores en cada pierna se debe tener un control total sobre los ejes X, Y y Z, y así ser capaces de mover el robot en todas las direcciones que se desee. Además, al utilizar la cinemática inversa del cuerpo derecho con un robot 3 GDL, se podría inclinar el cuerpo en todas las direcciones que se quisiera. Los nombres científicos para estas rotaciones que son la guiñada, el tono y el rollo se ilustran en la figura abajo.

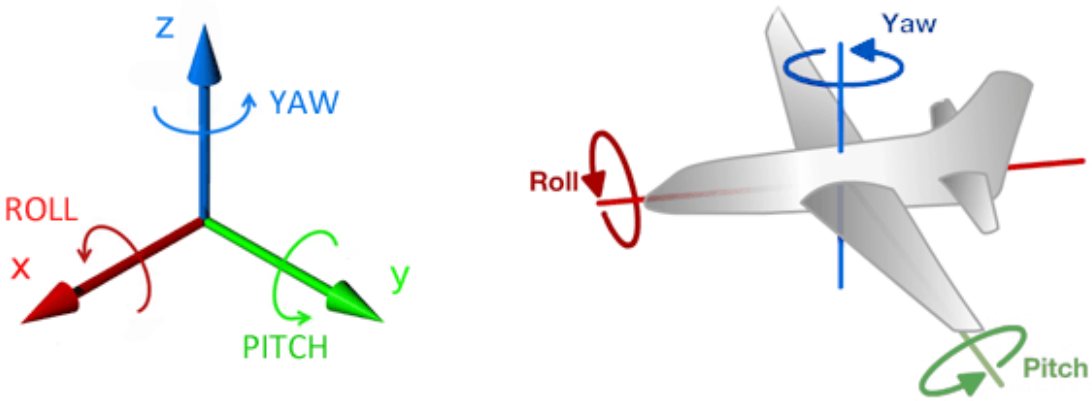


Figura 87: Viraje (yaw), inclinación (pitch) y balanceo (roll)

8.1.1 Cinemática inversa de las piernas

Para las patas del robot de 2 GDL, la cinemática inversa es bastante sencilla y se podría utilizar funciones trigonométricas simples para resolver los ángulos para un valor X, Y dado. La imagen de abajo muestra la vista desde arriba de la pierna anterior derecha, es obvio que los siguientes cálculos son los mismos para cada pierna. La variable φ_{coxa} es el ángulo que el servo motor coxa se mueve para que se convierta en una determinada posición.

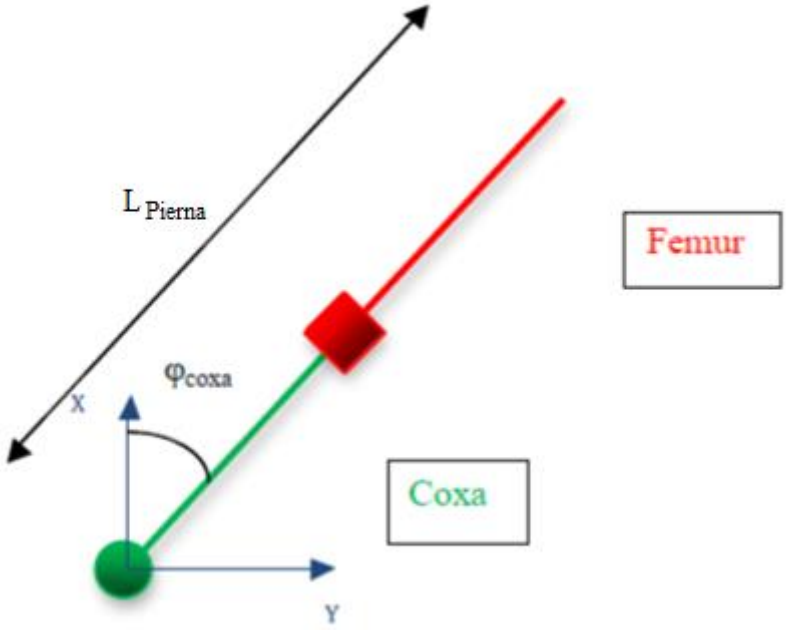


Figura 88: Pierna anterior derecha - vista superior

Cálculos:

$$\varphi_{coxa} = \tan^{-1}\left(\frac{y}{x}\right) \quad \textbf{(1)}$$

Teniendo en cuenta que es muy importante mantenerse consecuente con el φ_{coxa} ángulo. Se debe prestar mucha atención a los cuadrantes de tal manera la pierna trasera derecha y todas las otras patas tendrían un valor φ_{coxa} de 0 ° cuando la pata coincida con el eje X positivo, lo que significa que por ejemplo la posición inicial como se ve en la figura 72 va a ser 45 ° para la derecha anterior y 135 ° para la En la figura de abajo se puede ver el círculo geométrico con sus cuadrantes.

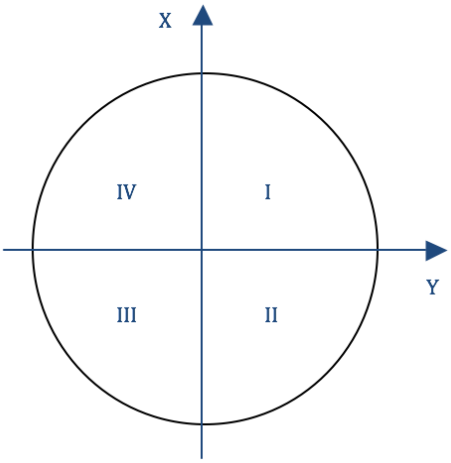


Figura 89: Círculo geométrico

En la figura de abajo se puede ver una vista lateral de la pierna. Se debe calcular φ_{femur} con la coordenada X. Antes de poder calcular el ángulo primero se debe determinar el valor L_{pierna}, este valor depende de los valores X e Y y puede ser visto como la longitud de la pierna vista desde la vista superior.

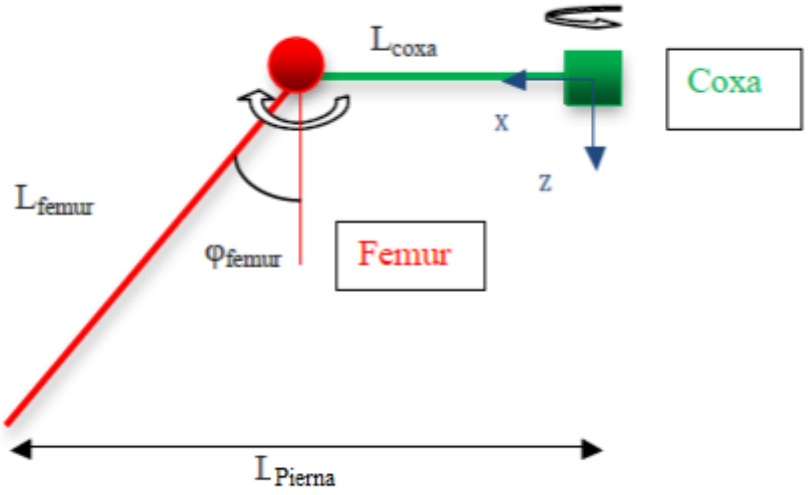


Figura 90: Vista lateral de la pierna derecha anterior

Cálculos:

$$L_{Pierna} = \sqrt{x^2 + y^2} \quad \textbf{(2)}$$

$$\varphi_{femur} = sen^{-1}\left(\frac{Lpierna - Lcoxa}{Lfemur}\right) \quad \textbf{(3)}$$

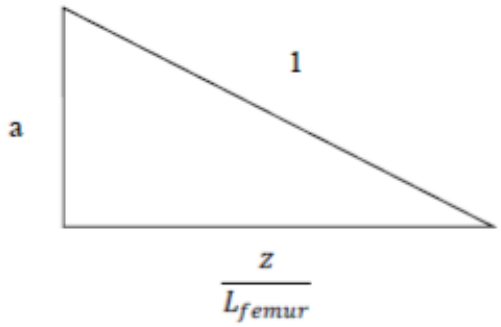
Con estos valores también se podría obtener la coordenada Z que, como ya se ha dicho, se cambia de forma inherente al cambiar las coordenadas X e Y.

$$z = L_{femur} \cdot \cos(\varphi_{femur}) \quad (4)$$

Dadas las ecuaciones 3 y 4 se podría derivar las asociaciones entre los valores de coordenadas Z y X, Y.

$$\cos^{-1}\left(\frac{z}{Lfemur}\right) = sen^{-1}\left(\frac{\sqrt{x^2+y^2}-Lcoxa}{Lfemur}\right) \quad (5)$$

Si se dibujara un triángulo rectángulo con un ángulo de $\cos^{-1}\left(\frac{z}{Lfemur}\right)$ Y el lado largo del rectángulo fuera 1, se podría encontrar fácilmente la simplificación de $sen(\cos^{-1}\left(\frac{z}{Lfemur}\right))$.



De Pitágoras se podría encontrar a, (suponiendo que el φ_{femur} permanece en el primer cuadrante):

$$a = \sqrt{1^2 - \left(\frac{z}{L_{femur}}\right)^2} \quad (6)$$

Eventualmente encontramos:

$$\sqrt{1^2 - \left(\frac{z}{L_{femur}}\right)^2} = \frac{\sqrt{x^2+y^2} - L_{coxa}}{L_{femur}} \quad (7)$$

$$z = \sqrt{{L_{femur}}^2 \cdot \left(1 - \frac{x^2+y^2-2L_{coxa}\sqrt{x^2+y^2}+{L_{coxa}}^2}{{L_{femur}}^2}\right)} \quad (8)$$

Con estas ecuaciones se podría hacer todas las combinaciones de ejes dominantes XZ, YZ, XY. Se hubiera podido haber programado esto en el cuadrúpedo de 2GDL para poder tener algunos grados de libertad más en algunos casos, sin embargo, debido al hecho de que el cuadrúpedo de 2GDL fue destinado principalmente a aprender a conocer el microcontrolador STM32, se eligió el XY como ejes dominantes con obviamente el eje Z siendo el obediente.

8.1.2 Cinemática inversa del cuerpo

Se podría ver la cinemática inversa del cuerpo como el pegamento que une las piernas y hace que el cuerpo se mueva en cierta posición. Es muy importante tener en cuenta que se ha usado el sistema de coordenadas global cuerpo X, Y, Z en lugar del sistema de coordenadas X, Y, Z de la pata. Puesto que se ha elegido el XY como ejes dominantes, sólo se debe tener una rotación del cuerpo alrededor de los ejes Z, como se ha visto anteriormente se denomina "Yaw" Viraje. Se puede observar la rotación del cuerpo como un cambio de los ejes X e Y presentados en la figura siguiente.

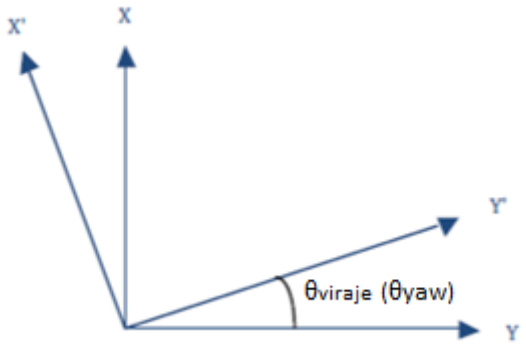


Figura 91: Sistema de coordenadas XY a X'Y '

El sistema de coordenadas rotado se designa con el exponente 'como X'Y'. Ahora se podría calcular los nuevos valores X 'e Y' dados los valores X, Y antes de la rotación y el ángulo θ_{viraje} o θ_{yaw}.

Cálculos:

Nuevamente utilizando matemáticas geométricas básicas, se podría derivar los valores X 'e Y'.

$$y' = y \,.\cos\theta_{yaw} + x \,.\operatorname{sen}\theta_{yaw} \qquad \textbf{(9)}$$

$$x' = -y \,.\operatorname{sen}\theta_{yaw} + x \,.\cos\theta_{yaw} \qquad \textbf{(10)}$$

Esto pone en una matriz las ecuaciones:

$$\begin{bmatrix} y' \\ x' \end{bmatrix} = \begin{bmatrix} \cos\theta_{yaw} & sen\,\theta_{yaw} \\ -sen\,\theta_{yaw} & \cos\theta_{yaw} \end{bmatrix} . \begin{bmatrix} y \\ x \end{bmatrix} \qquad \textbf{(11)}$$

Tenga en cuenta que, si se hubiera escogido otros ejes dominantes, los cálculos serían analógicos, pero el valor de la rotación ahora sería inclinación (pitch) o balanceo (roll). Otro comentario que se va a realizar es que dado que se tiene una rotación, que es un movimiento

circular, el eje compatible no cambia, ya que se sabe que la ecuación circular que está en este ejemplo con los ejes X e Y es $x^2 + y^2 = R^2$.

En lo que sigue se trata de dar una explicación completa de lo que implica esta matriz. Imagine el robot con sus 4 patas e imagine que su cuerpo gire un cierto ángulo de viraje (yaw) sin cambiar la posición de las piernas. Los ejes XY también habrán girado este cierto ángulo. Los ejes de rotación se presentan en la figura anterior con los ejes X'Y '. Si se quiere seguir utilizando los ejes XY globales, sólo se podría restar los valores X'Y 'obtenidos de los valores XY iniciales antes de la rotación. De esta manera se obtendrían los cambios en el sistema global de coordenadas globales cuerpo.

8.2 Cinemática del cuadrúpedo de 3 GDL

La cinemática del cuadrúpedo 3GDL es un poco más exigente que los del cuadrúpedo 2GDL, pero los principales siguen siendo los mismos, esta es la razón por la que cubriremos este parte un poco más rápido. En la figura de abajo se puede encontrar de nuevo el cuadrúpedo sin embargo ahora combinado con la pierna de la tibia. Evidentemente, los ejes permanecen iguales.

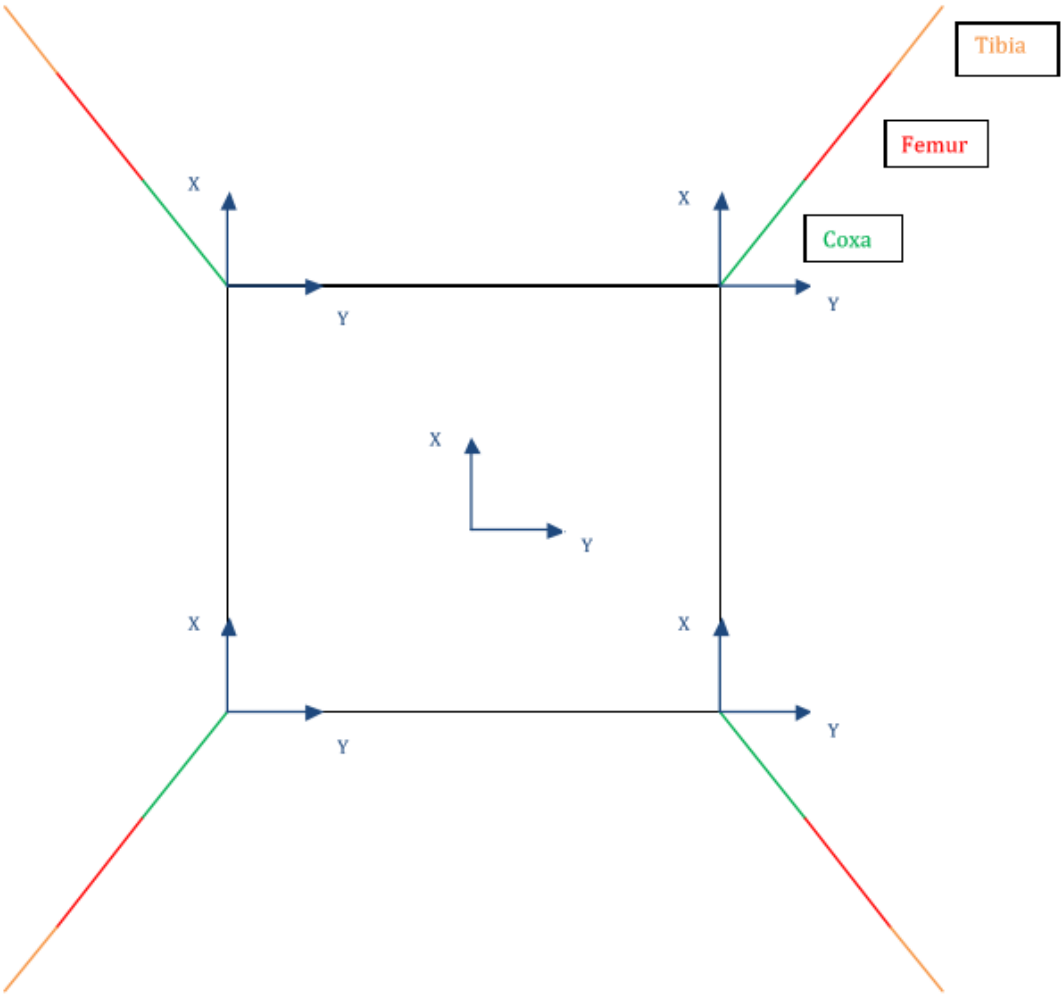


Figura 92: 3DOF cuadrúpedo - vista superior

8.2.1 Cinemática inversa de pierna

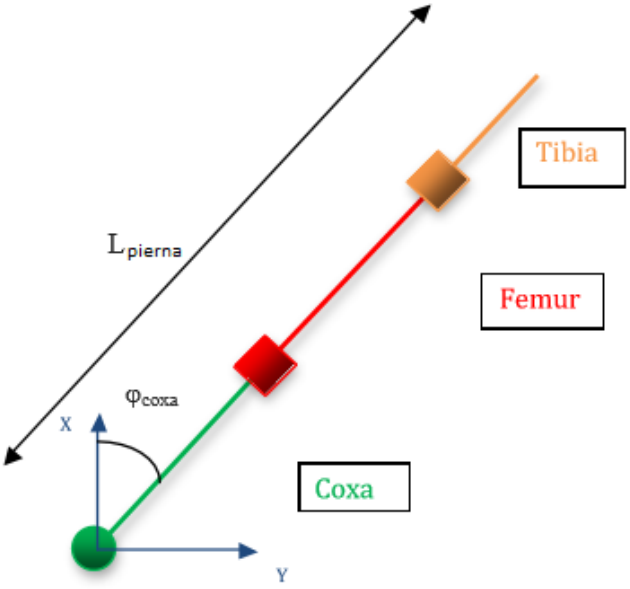


Figura 93: Pierna anterior derecha - vista superior

En la figura de arriba se puede completamente análogo como para el cuadrúpedo 2GDL encuentre la vista superior de la pierna con la parte de la tibia agregada.

Cálculos:

$$\varphi_{coxa} = tan^{-1}\left(\frac{y}{x}\right) \quad \textbf{(1)}$$

Por lo tanto, se podría ver que los cálculos para φ_{coxa} son exactamente los mismos que para el cuadrúpedo 2GDL.

La vista lateral de la pata puede verse a continuación, calculando el φ_{femur} y φ_{tibia} es más complejo que para el cuadrúpedo 2GDL, pero todavía se podría hacer con relaciones geométricas básicas.

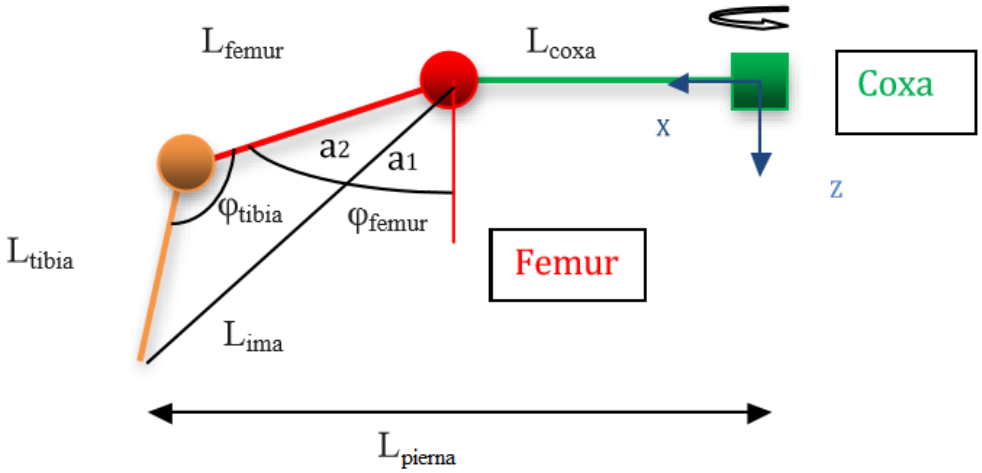


Figura 94: Vista lateral de la pierna anterior derecha

Cálculos:

$$L_{pierna} = \sqrt{x^2 + y^2} \quad \textbf{(2)}$$

Dividimos el φ_{femur} en dos partes denominadas a_1 y a_2, lo que facilita el cálculo. Primero se calcula la longitud de la línea negra imaginaria "L_{ima}" y desde allí se debe empezar a calcular los ángulos.

$$L_{ima} = \sqrt{z^2 + \left(L_{pierna} - L_{coxa}\right)^2} \quad \textbf{(12)}$$

$$a_1 = cos^{-1}\left(\frac{z}{L_{ima}}\right) \quad \textbf{(13)}$$

Ahora con la línea imaginaria trazada entre el fémur y el extremo de la pierna de la tibia, como se ve en la figura de abajo, se podría calcular a_2 y φ_{tibia} usando la regla del coseno.

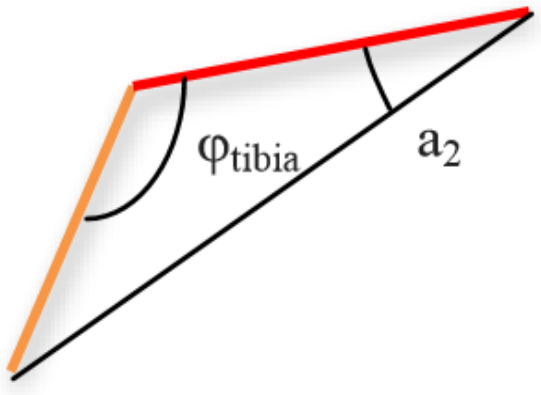


Figura 95: línea imaginaria entre la articulación de la coxa y el extremo de la pierna de la tibia

- $L_{tibia}^2 = L_{femur}^2 + L_{ima}^2 - 2 \cdot L_{femur} \cdot L_{ima} \cdot cos(a_2)$ **(14)**

- $a_2 = cos^{-1}\left(\frac{L_{femur}^2 + L_{ima}^2 - L_{tibia}^2}{2 \cdot L_{femur} \cdot L_{ima}}\right)$ **(15)**

- $\varphi_{femur} = a_1 + a_2$ **(16)**

Completamente análogo φ_{tibia} puede ser calculado como:

- $\varphi_{tibia} = cos^{-1}\left(\frac{L_{femur}^2 + L_{tibia}^2 - L_{ima}^2}{2 \cdot L_{tibia} \cdot L_{femur}}\right)$ **(17)**

Tenga en cuenta que esta solución no es generalmente aplicable a todos los tipos de patas de robots ya que el ángulo entre las diferentes partes de la pierna fémur, tibia también puede diferir. Se puede imaginar, por ejemplo, la parte de la tibia dividida en dos partes con diferentes longitudes y un ángulo entre estas dos partes. De esta manera se podría ver la solución anterior como una solución específica con el ángulo de las partes del fémur y la tibia como 180 grados. Por supuesto, con este tren de pensamiento se puede imaginar el ángulo infinito en la tibia y el fémur, sin embargo, es obvio que la funcionalidad de estos ángulos termina en 1 ángulo en cada parte. La función de estos ángulos es dar una mayor gama de ángulos de motor útil ya que dependiendo de la configuración de la pierna algunos ángulos serán inútiles. En el robot utilizamos un ángulo para la parte de la tibia, los cálculos para este ángulo extra conocido se pueden encontrar a continuación. El cálculo del ángulo de coxa no se repetirá ya que es el mismo.

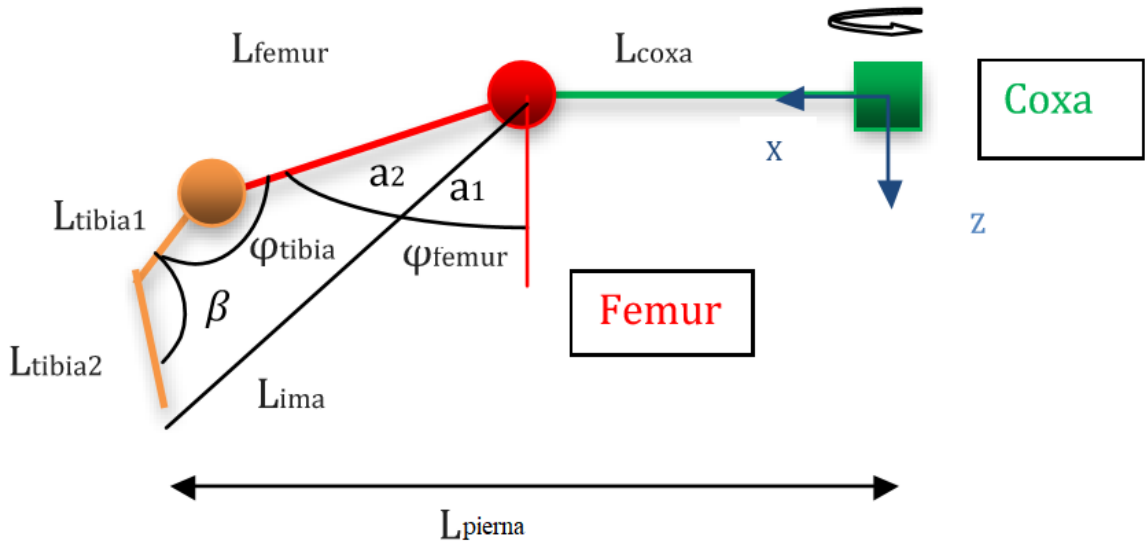


Figura 96: Parte anterior de la pierna anterior - vista lateral - tibia de dos partes

Cuando dibujamos la línea imaginaria similar a los cálculos anteriores se encuentra un cuadrilátero irregular. Que se podría dividir en dos triángulos irregulares a partir de los cuales los ángulos se pueden calcularse con la regla del coseno.

Cálculos:

Los cálculos 1, 2, 12, 13 permanecen iguales.

Ahora se dibuja otra línea imaginaria que se llamará "L_{ayu}" (línea de ayuda) en el cuadrilátero irregular que como el nombre predice nos ayuda a calcular los ángulos. Esta línea imaginaria divide φ_{tibia} en b_1 y b_2.

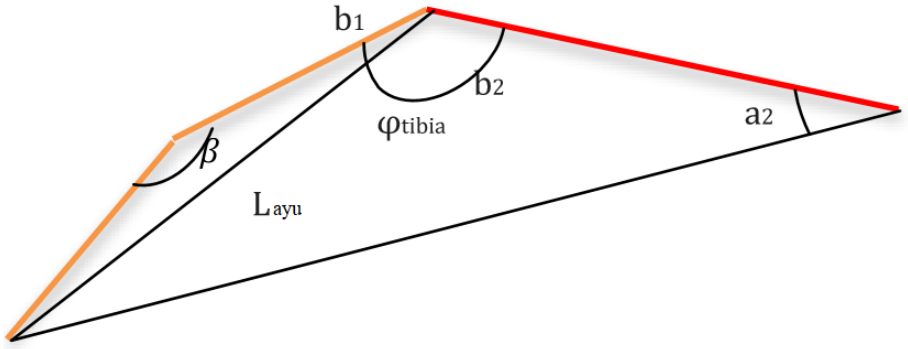


Figura 97: líneas imaginarias entre las articulaciones de la coxa y el extremo de la pierna de la tibia – tibia en dos partes

- $L_{ayu} = \sqrt{L_{tibia1}^2 + L_{tibia2}^2 - 2\,.\,L_{tibia2}\,.\,L_{tibia1}\,.\cos\beta}$ **(18)**

usando la regla del seno:

- $b_1 = sen^{-1}\left(\frac{L_{tibia2}\,.\,\text{sen}\,\beta}{L_{ayu}}\right)$ **(19)**

usando la regla del coseno:

- $b_2 = cos^{-1}\left(\frac{L_{ayu}^2 + L_{femur}^2 - L_{ima}^2}{2\,.\,L_{ayu}\,.\,L_{ima}}\right)$ **(20)**

- $\varphi_{tibia} = b_1 + b_2$ **(21)**

De igual manera para el fémur:

- $a_2 = cos^{-1}\left(\frac{L_{femur}^2 + L_{ima}^2 - L_{ayu}^2}{2 \cdot L_{femur} \cdot L_{ima}}\right)$ **(22)**

- $\varphi_{femur} = a_1 + a_2$ **(16)**

También es posible un ángulo en la parte del fémur. Sin embargo, los ángulos en el fémur y en la parte de la tibia combinados resultan en un pentágono irregular muy difíciles de calcular manualmente hasta llegar al punto donde se alcanzarán funciones geométricas complejas. En ese momento se aconseja consultar software matemático.

8.2.2 Cinemática inversa de la pierna

El autor de este proyecto quiere ser capaz de hacer que el robot entre en contracción y en una posición de almacenamiento o guardado que implica poner las piernas junto a su cuerpo para ocupar el mínimo espacio posible. Se desee utilizar un booleano simple para cambiar de la cinemática de inversión normal a la cinemática de inversión inversa. Se tuvo que retomar la cinemática inversa y ver qué cambios debían hacerse.

La cinemática normal discutida anteriormente:

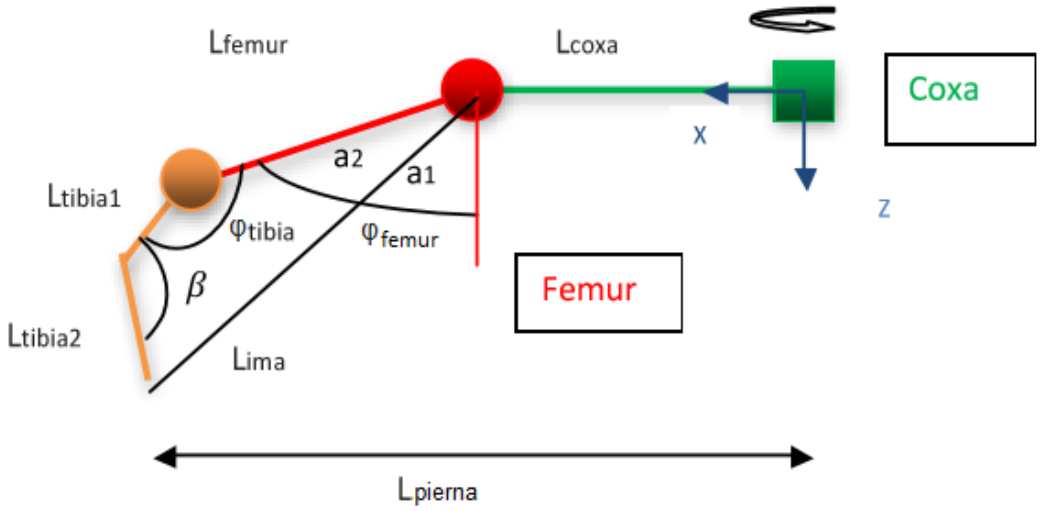


Figura 98: Cinemática normal discutida anteriormente

Cinemática inversa:

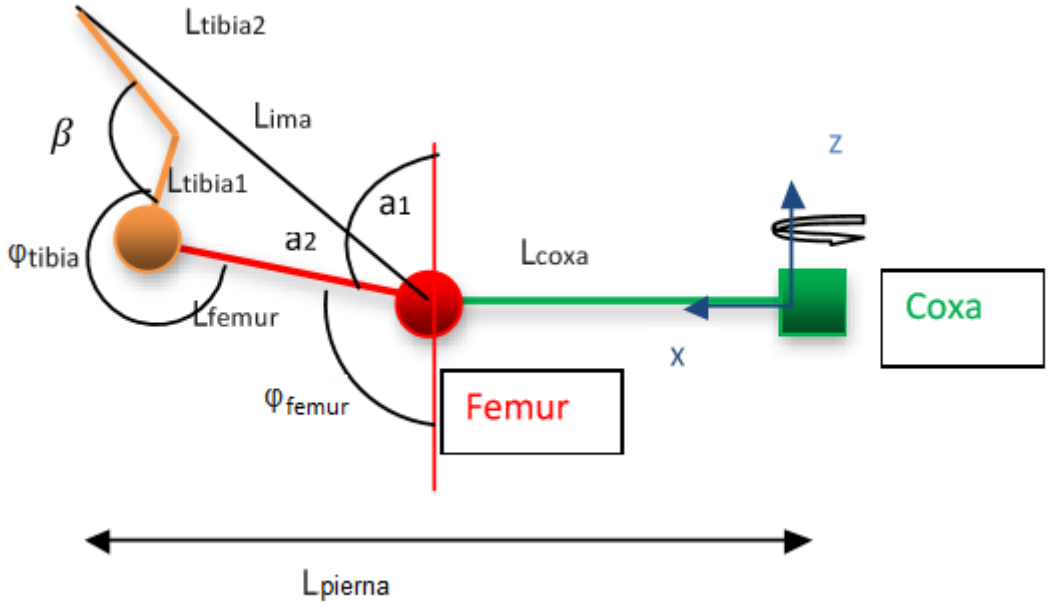


Figura 99: Cinemática inversa

Podemos otra vez igual que antes calcular a_1, a_2, b_1 y b_2.

- $a_1 = cos^{-1}\left(\frac{z}{L_{ima}}\right)$ **(13)**
- $a_2 = cos^{-1}\left(\frac{L_{femur}^2 + L_{ima}^2 - L_{ayu}^2}{2\,.\,L_{femur}\,.\,L_{ima}}\right)$ **(22)**
- $b_1 = sen^{-1}\left(\frac{L_{tibia2}\,.\,\text{sen}\,\beta}{L_{ayu}}\right)$ **(19)**
- $b_2 = cos^{-1}\left(\frac{L_{ayu}^2 + L_{femur}^2 - L_{ima}^2}{2\,.\,L_{ayu}\,.\,L_{ima}}\right)$ **(20)**

Observe cómo los cálculos de estos ángulos son exactamente iguales. Pero estos ángulos ahora tienen un significado diferente. Para calcular el ángulo real los servomotores tienen que girar como se ve en la representación anterior de la figura 83. Por tanto, ahora sólo se deben usar las siguientes ecuaciones.

- $\varphi_{femur} = \pi - a_1 - a_2$ **(23)**
- $\varphi_{tibia} = 2\,.\pi - b_2 + b_1$ **(24)**

8.2.3 Cinemática inversa del cuerpo

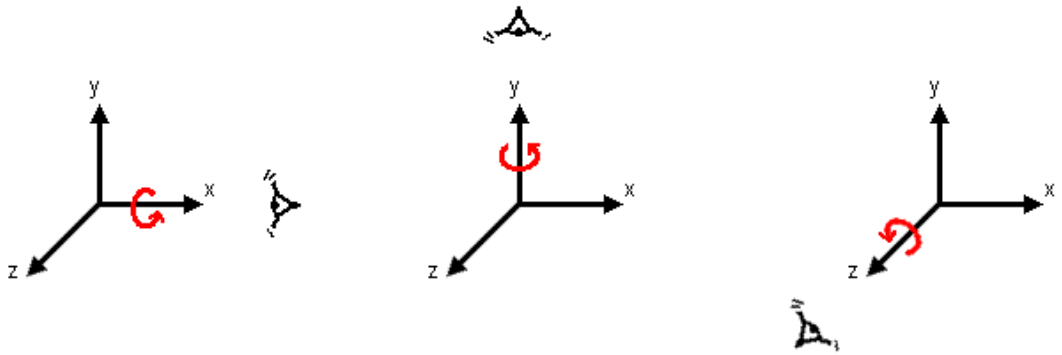


Figura 100: Rotación de los ejes

Con los tres valores de ángulo del motor se tiene el control total de los ejes X, Y y Z y así se puede mover y girar el cuerpo en cualquier posición deseada dentro de los límites físicos de las piernas del robot, por supuesto. El concepto de los cálculos es el mismo que el descrito anteriormente para el cuadrúpedo 2GDL, pero ahora las rotaciones son posibles a lo largo de los ejes X balanceo-roll, Y inclinación-pitch y Z viraje-yaw. Los cálculos para obtener todas las matrices de rotación son los mismos.

Es necesario tener en cuenta que estas matrices de rotación sólo son válidas en un sistema de coordenadas a derechas y que los ángulos positivos están en sentido contrario a las agujas del reloj.

Rotación alrededor del eje Z viraje - yaw:

se puede observar cómo la coordenada Z obviamente no cambia ya que la rotación está por encima del eje Z. Esto es lo mismo para las otras matrices de rotación.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta_{yaw} & -sen\,\theta_{yaw} & 0 \\ sen\,\theta_{yaw} & \cos\theta_{yaw} & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \textbf{(25)}$$

Rotación alrededor del eje Y inclinación - pitch:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta_{pitch} & 0 & sen\,\theta_{pitch} \\ 0 & 1 & 0 \\ -sen\,\theta_{pitch} & 0 & \cos\theta_{pitch} \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \textbf{(26)}$$

Rotación alrededor del eje X balanceo - roll:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_{roll} & -sen\,\theta_{roll} \\ 0 & sen\,\theta_{roll} & \cos\theta_{roll} \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \textbf{(27)}$$

Ahora se podría multiplicar todas estas rotaciones de matrices para obtener la matriz de rotación total M totR de 3x3, tenga en cuenta que la matriz de rotación general depende del orden de rotaciones, pero esto no importa ni altera el resultado final.

$$M_{totR} = M_{yaw} \cdot M_{pitch} \cdot M_{roll} \quad \textbf{(28)}$$

$$M_{totR} =$$

$$\begin{bmatrix} \cos\theta_{pitch}\cos\theta_{yaw} & \cos\theta_{yaw}\, sen\,\theta_{roll} sen\,\theta_{pitch} - sen\,\theta_{yaw}\cos\theta_{roll} & \cos\theta_{roll}\cos\theta_{yaw}\, sen\,\theta_{pitch} + sen\,\theta_{roll} sen\,\theta_{yaw} \\ \cos\theta_{pitch}\ sen\,\theta_{yaw} & \cos\theta_{roll}\cos\theta_{yaw} + sen\,\theta_{roll} sen\,\theta_{pitch}\, sen\,\theta_{yaw} & -\cos\theta_{yaw}\, sen\,\theta_{roll} + \cos\theta_{roll}\,\theta_{roll} sen\,\theta_{pitch} sen\,\theta_{yaw} \\ -sen\,\theta_{pitch} & \cos\theta_{pitch}\, sen\,\theta_{roll} & \cos\theta_{roll}\cos\theta_{pitch} \end{bmatrix}$$

9 Calibración de los servomotores

Después de estudiar la cinemática inversa se debe conocer el ángulo que los servomotores necesitan para moverse para obtener una determinada posición. Ahora bien, los servomotores tienen valores de servo como entrada en lugar de ángulos. Por lo tanto, se requiere una transformación de estos ángulos en valores de servo. Los servomotores se controlan mediante una señal PWM modulación de ancho de pulso. Enviando una señal pulsada de 50 Hz al servomotor con un ancho de impulso variable. Dependiendo de esta señal pulsada el servomotor mantendrá una determinada posición. Se puede encontrar en la lista de componentes usados los servomotores 270 ° PWM500-2500us. Esto significa que la señal pulsada variará entre 0,5 y 2,5 milisegundos. La velocidad de reloj de la STM32F3Discovery es de 48MHz, esto significa que se obtiene una señal pulsada con un período de 1/48000000 segundos, que es sin duda una forma fácil de configurar el microcontrolador. Por lo tanto, se tendría un pre-escalador de 16 bits y un período de contador para cambiar la señal. Los 48MHz se transformarán así en una señal de 50Hz. Los cambios para el prescaler y el período del contador se pueden cambiar, como se ha visto con anterioridad, en el programa STM32CubeMX.

$$prescaler\ .counterperiod = \frac{48000000}{50Hz} = 960000$$

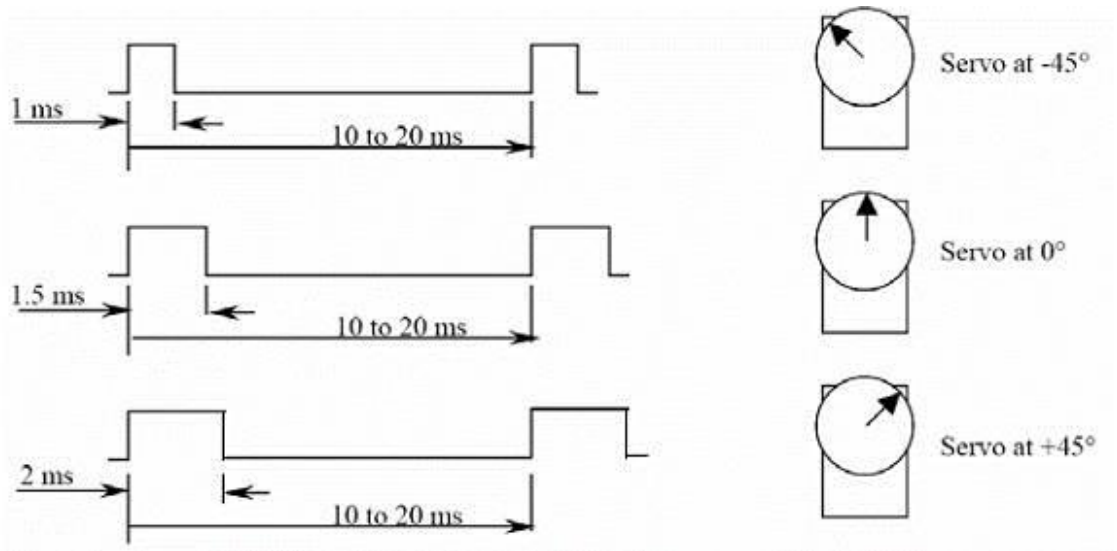


Figura 101: PWM - la figura es puramente ilustrativa

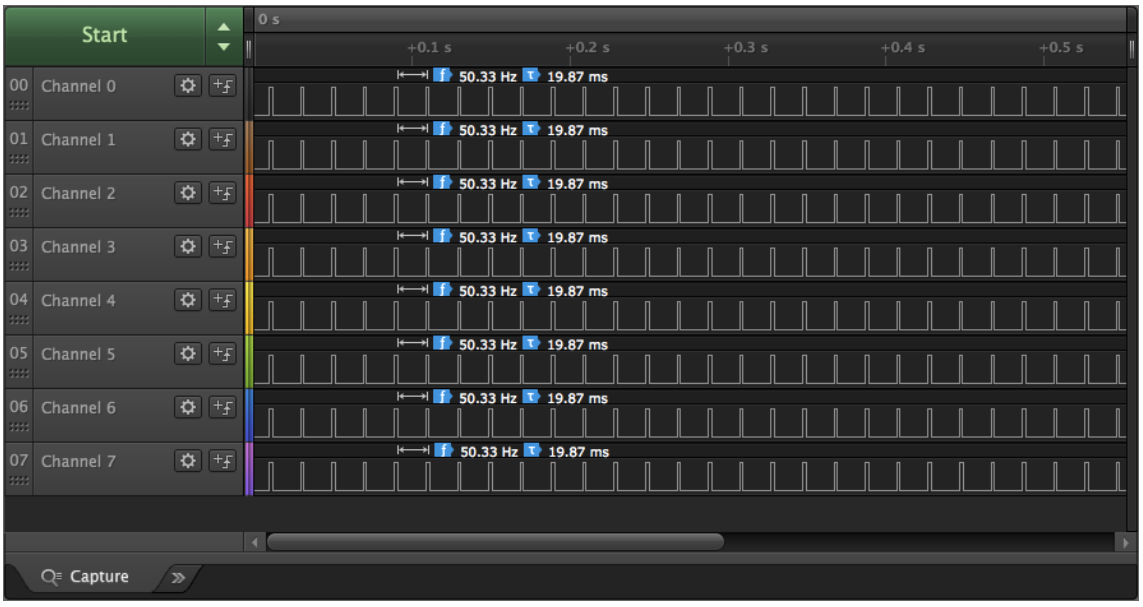


Figura 102: Frecuencia de las señales dadas a los 8 servomotores del robot de 2GDL

Ahora se debe encontrar valores de ajuste para el período de prescaler y counter, se sabe que el valor para el contador es el valor que se cuenta en 20 milisegundos, porque Una señal de 50 Hz tiene un período de 20 milisegundos. En el programa se compara el período de contador con un cierto valor de servo, que a su vez hará que el servomotor se mueva un cierto ángulo.

La mejor manera de explicar esto es con un ejemplo:

En el cuadrúpedo de 2GDL se toma un valor de prescaler de 2400 y un periodo de contador de 400, estos dos juntos obviamente igual a 960000 y así dar una señal de 50 Hz. Cada 20 milisegundos el contador cuenta a 400. Si se compara el periodo de contador de 400 con un valor de 10 entonces se observaria que contar hasta 10 toma 20 x 10/400 milisegundos, siendo 0,5 milisegundos. El ángulo del motor sería entonces de 0 ° ya que se obtiene un PWM de 500-2500us. Si se compara el periodo del contador con un valor de 5, el servomotor no se movería, ya que esto sería igual a 0,25 milisegundos y, por lo tanto, caería fuera del rango para el servo.

Debido a que ponemos en el servomotor un brazo con estructura en forma de U el ángulo del motor no será igual al ángulo de salida hacia la referencia en la cinemática, ya que hay muchas posibilidades para poner el motor en el marco. Por ejemplo, el ángulo del motor de 0 ° dado con el valor de servo 10 puede ser un valor de coxa de 45º. Por lo tanto, para el resto de este artículo nos olvidamos de los ángulos del motor y sólo usamos los ángulos de salida dados para un cierto valor de servo. Es este ángulo que se deberá transformar en un valor servo.

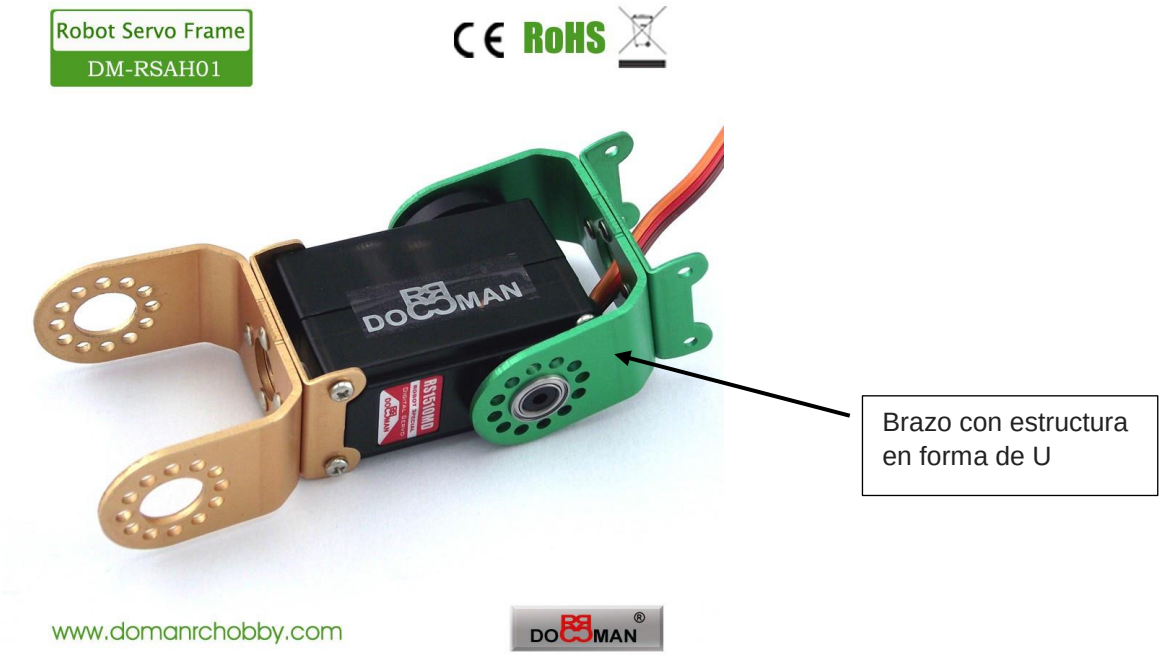


Figura 103: Servomotor con brazos con marco en forma de U

La calibración real es realmente fácil. Se mide el ángulo de dos puntos junto con sus valores de servo ahora que solo se ajustaría a la ecuación de una recta **y = mx + b** sobre estos dos puntos y se debe tener nuestra función para ir de ángulos a servos. Esta ecuación es diferente para cada motor. Debido a que todas las calibraciones son completamente analógicas y sus funciones se pueden encontrar en el programa dado en el capítulo 14 sólo se cubrira 1 ejemplo de calibración.

Calibración:

Medición:

- 90 º = $\pi/2$ rad. equivalen a un valor del servo de 30
- 180 º = π rad. Equivalen a un servo-valor de 44

Por lo tanto, se obtiene:

La pendiente m = $\frac{y_2 - y_1}{x_2 - x_1}$ siendo y_1=30, y_2=44, x_1= $\pi/2$, x_2= π.

$$\boldsymbol{m} = \frac{44 - 30}{\pi - \frac{\pi}{2}} = \mathbf{8,9126} \ldots$$

El intercepto en el eje Y llamado $b = y_1 - m \cdot x_1$

También puede calcularse como $b = y_2 - m \cdot x_2$ como se ha hecho en este caso:

$$\boldsymbol{b} = 44 - 8{,}9126 \cdot \pi = \mathbf{16}$$

La ecuación de la recta queda de tal forma:

- **Servo valor = 8.912φ + 16**

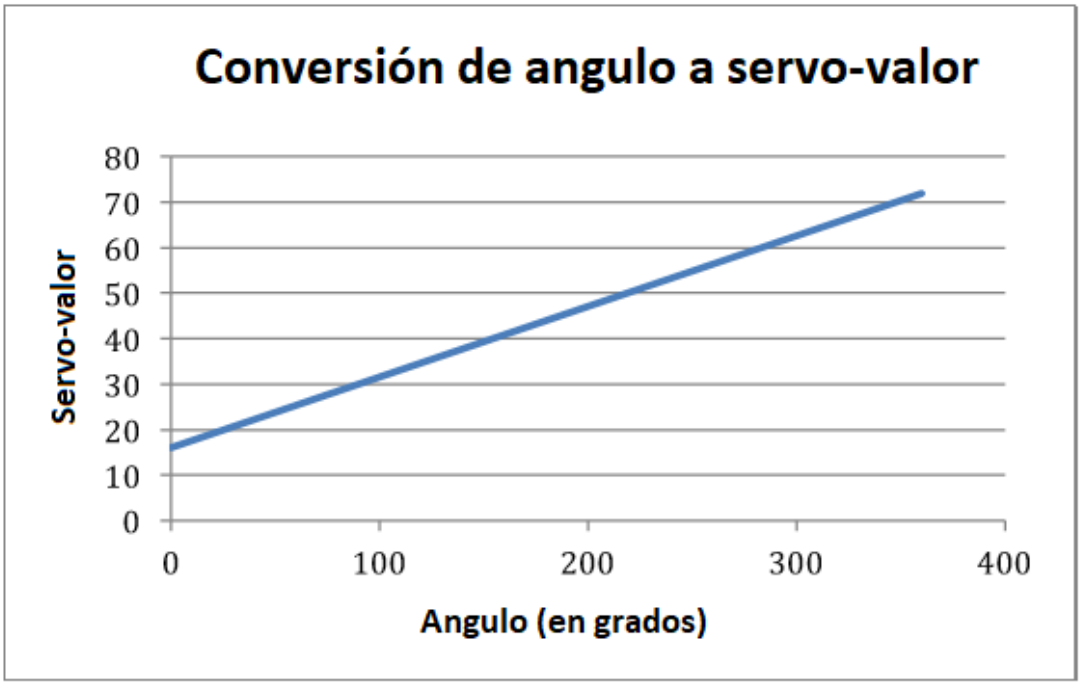


Figura 104: Ángulo al valor de servo

Tenga en cuenta que sólo los ángulos dentro de 0 ° a 360 ° tienen una solución analítica. Los cambios de 360 ° a 0 ° no se podrían calcular correctamente ya que esto es discontinuo. Se conseguiría un valor de servo que está fuera de los límites. Para la configuración angular (dirección usada del ángulo) sólo tendría estos problemas con los ángulos de coxa de las patas anteriores derechas e izquierda ya que éstas pueden ir de 360 ° a 0 °. Se podría resolver este problema calculando dos ecuaciones para transformar los ángulos de servo en valores de servo, en lugar de una sola ecuación. Por ejemplo, para la pierna anterior izquierda se podría generar una ecuación para calcular los valores de servo para los ángulos 225 °- 360 ° y uno para los ángulos 0 °- 45 °.

Programa de calibración usado

En el código programado a continuación se debe encontrar un bucle “for” que va de 10 a 55 para cubrir todos los valores de servo con un valor de prescaler de 2400 y un periodo de contador de 400. Se utiliza este programa de calibración para el cuadrúpedo 2GDL, Para el cuadrúpedo 3GDL es completamente analógico, sólo los valores serán diferentes, ya que se usaron otro valor de prescaler y otro valor para el período de contador. La salida del programa es muy simple, cada 500 ms, el valor “i” se incrementará, debido a esto el servo motor se moverá cada 500 ms ya que el valor y se transmite como valor servo que se puede ver en el "__HAL_TIM_SetCompare" . Obviamente esto por sí solo no es suficiente para calibrar el servo motor ya que también se necesita saber qué posición se alcanza en un momento determinado. Para ello se utiliza un canal UART en el microcontrolador. Se envía el valor y a través de un cable serie a USB a nuestro ordenador cada 500ms que se muestra en nuestro HyperTerminal llamado "Termite 3.3". El valor “i” no se puede enviar directamente ya que es una variable entera “int”. Es necesario transformar esto en una variable cadena de caracteres “char” primero, como lo hace la función “sprintf” que escribe la cadena de caracteres donde se enviara por el canal UART y se visualizara por el “Termite 3.3”. “Mensa_debug_Usart3” es la función o el método en el codigo utilizado para enviar los datos, el código de este método también se muestra a continuación. Como se ve en el capítulo 6, la inicialización del canal serie y los temporizadores HAL se realizan en STM32CubeMX.

```
for (int i = 10; i < 55; i++)
{

__HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_1,(uint16_t) i);
char buffer[5];
sprintf(buffer, "i%.2u", i);
Mensa_debug_Usart3(buffer);
Mensa_debug_Usart3("\n");
HAL_Delay(500);

}


void Mensa_debug_Usart3(char text[32])
{

if (HAL_UART_Transmit(&huart3, (uint8_t*) text, strlen(text), 1000)!= HAL_OK)
       {
       Error_Handler();
       }

HAL_Delay(50);
return;

}
```

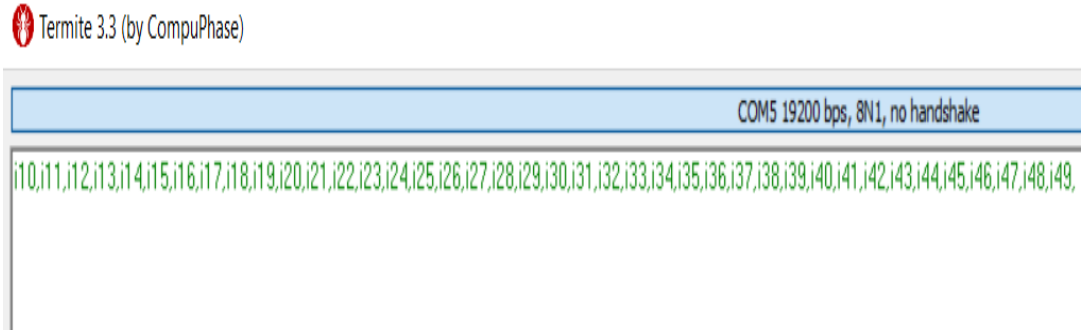


Figura 105: Los valores de servo dados en nuestro Hyper-Terminal (Termite 3.3)

10 IMU - MEMS

Una IMU o unidad de medida inercial recoge la aceleración lineal y la velocidad angular. Para recoger la aceleración lineal la IMU utiliza un acelerómetro y para recoger la velocidad angular se utiliza un giroscopio. De esta manera se podría ver que existe una IMU de 2 sensores diferentes. La fuerza de estos 2 sensores juntos es que las mediciones de velocidad angular no son afectadas por la aceleración lineal. Poniendo estos dos sensores juntos la IMU nos proporciona la orientación que no se ve afectada por aceleraciones lineales. Las IMU se usan en aviones, drones, misiles, ...

Figura 106 y 107 : Un Dron y un caza tirando un un misil

Estos pequeños productos electrónicos pueden proporcionarnos la posición de los aviones, drones o cualquier robot terrestre en este caso del cuadrúpedo. Más específicamente en nuestro caso las rotaciones que serán las de pitch-inclinación y roll- balanceo.

10.1 Uso de la IMU en el cuadrúpedo 3GDL

Se utiliza la IMU en el cuadrúpedo 3GDL para estabilizarlo en terrenos accidentados, esto no fue ciertamente la mejor posibilidad. Una combinación de diferentes sensores, por ejemplo, una cámara que filma el terreno combinado con un IMU y un algoritmo de estabilización habría sido más óptima para estabilizar el cuadrúpedo. Debido a la falta de tiempo y de material, esto no fue posible, ya que encontrar un algoritmo de estabilización, etc. Es tarea muy difícil y requiere mucho conocimiento cinemático y sobre todo dinámico. La IMU detecta cuando el cuerpo del cuadrúpedo no es horizontal, por ejemplo, cuando hay una pierna de pie sobre una piedra u otra rugosidad de un cierto terreno. Cuando esto sucede, la IMU dará los ángulos de inclinación y de balanceo, estos ángulos serán utilizados inversamente en la cinemática inversa del cuerpo. De esta forma, el cuerpo volverá a ser horizontal y permanecerá estable.

10.2 I²C y SPI

Antes de explicar el acelerómetro y el giroscopio, daremos primero una breve introducción a I²C y SPI, que son dos protocolos utilizados para la comunicación entre circuitos integrados. Ambos tipos de comunicación están disponibles en los microcontroladores STM32F3.

10.2.1 I²C

I²C es un protocolo maestro-esclavo que puede tener múltiples esclavos. La comunicación es semidúplex y orientada a 8 bits. A continuación, se muestra una representación de la comunicación.

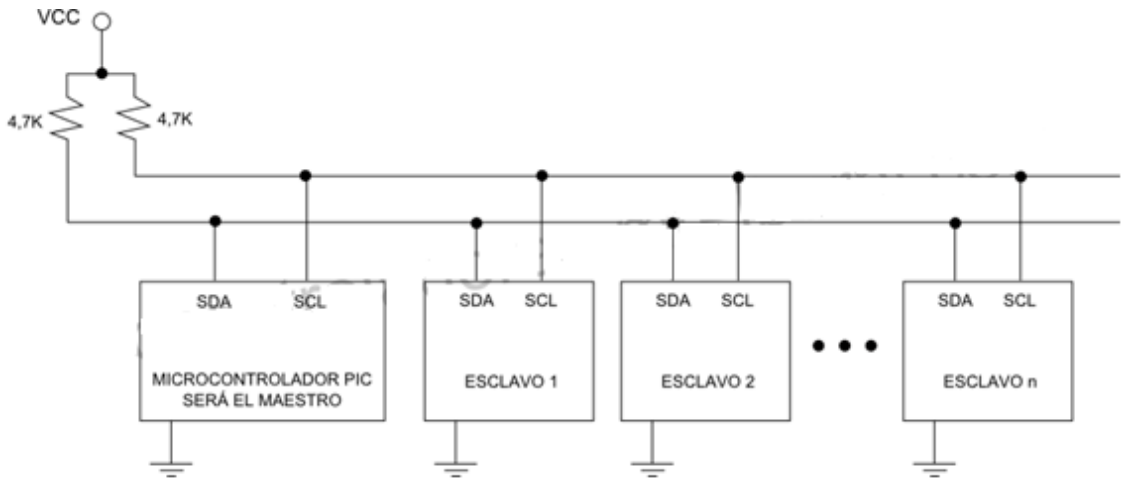


Figura 108: Protocolo I2C

Se puede observar que tenemos 2 líneas, una línea SDA y SCL. SDA o Serial Data es como el propio nombre revela la línea para enviar datos en serie. El reloj SCL o Serial es la señal de reloj que el maestro envía con los datos. Puesto que se tiene una señal de reloj se obtiene una comunicación síncrona. Primero el maestro envía un bit de inicio que indica el inicio de la comunicación. Después de este bit de inicio, el maestro envía una dirección del esclavo con el que desea comunicarse. Después del bit de dirección se sigue un bit de lectura o escritura para indicar qué tipo de función ejecutará el maestro. Ahora el esclavo dirigido reaccionará si está activo al maestro con un acuse de recibo. Después de esto la comunicación puede comenzar, 8 bits son enviados a la vez más un acuse de recibo.

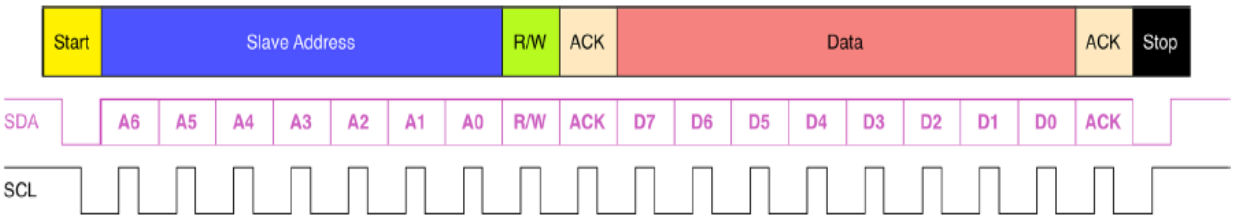


Figura 109: Ejemplo de transferencia de datos - I2C

10.2.2 SPI

Aunque tiene la misma función que I²C, el principio de funcionamiento es totalmente diferente. I²C sólo utiliza dos líneas mientras SPI utiliza n · 4 líneas, siendo n el número de esclavos. La comunicación es full-duplex (también puede ser half-duplex dejando una línea fuera).

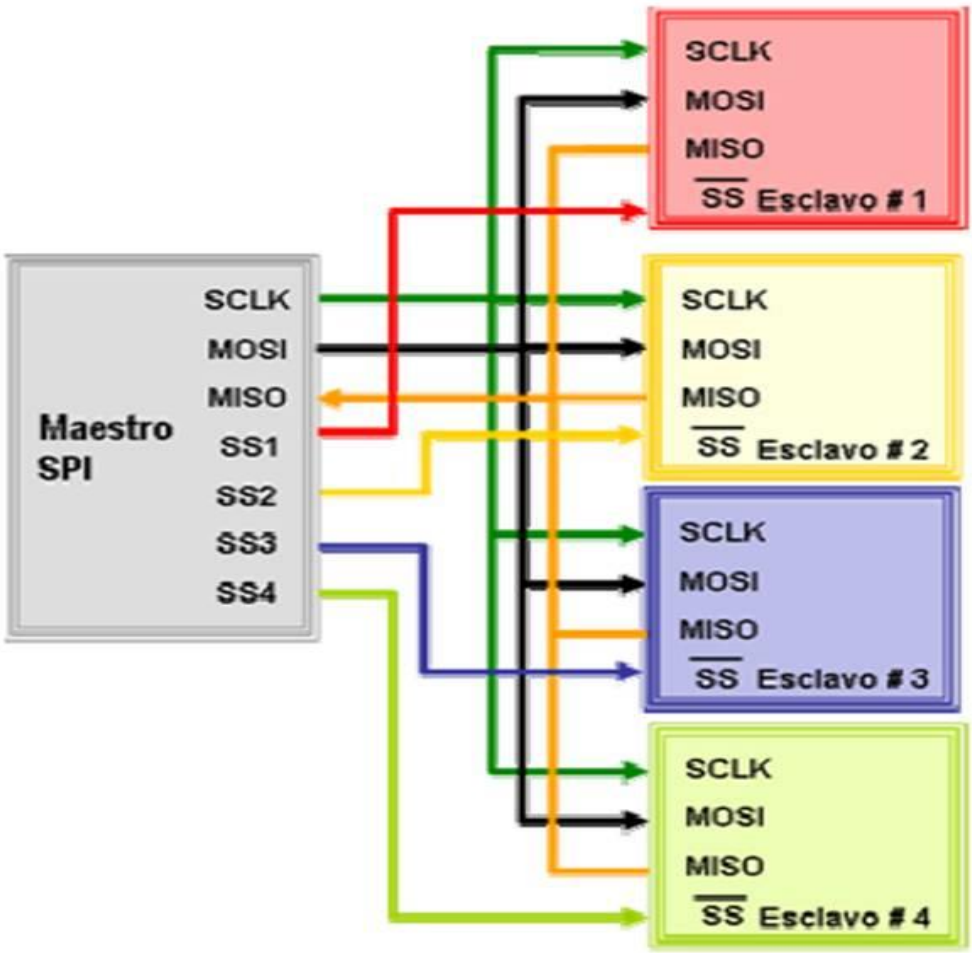


Figura 110: Protocolo SPI

En la representación anterior se pueden distinguir 4 líneas, SCK, MOSI, MISO y SS. La línea SCK es la línea de reloj que se utiliza para sincronizar la transferencia de datos. Tenga en cuenta que el reloj del SPI es mucho más rápido que el reloj del I²C. MOSI y MISO representan respectivamente la entrada de esclavo de salida maestra y la salida de esclavo de entrada maestra y son como pueden derivarse de los nombres, ya sea enviando datos de maestro a esclavo o datos de esclavo a maestro. La selección SS o Slave es una línea que puede ser alta o baja y se utiliza para seleccionar el esclavo con el que se comunicará. A continuación, puede encontrar un ejemplo de transferencia de datos.

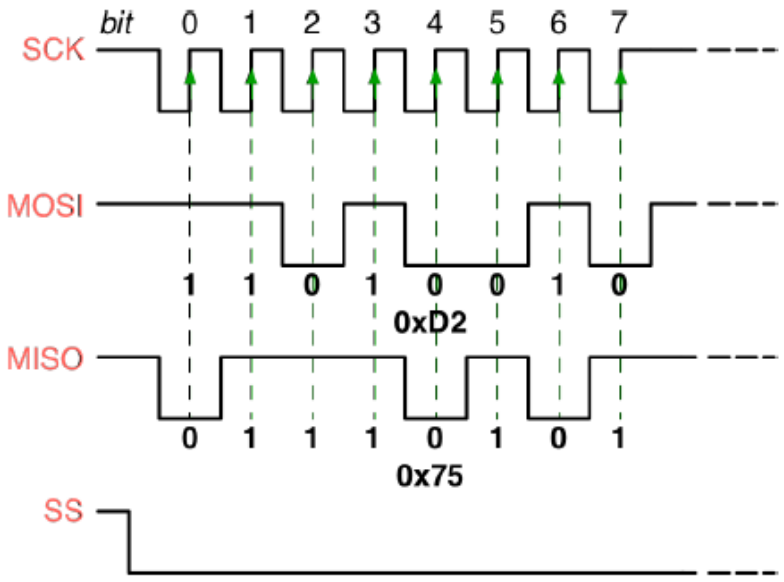


Figura 111: Ejemplo de transferencia de datos - SPI

Al comienzo de la comunicación primero se selecciona la velocidad de transferencia, siendo igual a la velocidad de reloj. Posteriormente, el esclavo con el que se debe comunicar será seleccionado por la línea SS. Ahora los datos serán transferidos usando la línea MOSI y MISO.

10.3 Acelerómetro

El acelerómetro mide, como el nombre revela, la aceleración lineal en 3 direcciones. Esto significa las aceleraciones gravitacionales, pero también las aceleraciones debidas al movimiento del robot. Se puede ver que el acelerómetro no es fiable para medir las rotaciones ya que los valores medidos consisten en dos tipos de aceleraciones y para medir las rotaciones solo se necesitaría las aceleraciones gravitatorias. Cuando tenemos una aplicación con una aceleración muy baja, la falla de las rotaciones calculadas será bastante baja. Sin embargo, en la mayoría de las aplicaciones la aceleración debida al movimiento tendrá un gran impacto. En el STM32F3DISCOVERY ya hay un acelerómetro que es el lsm303dlhc. En la placa de Nucleo STMF334R8 no hay, como se dijo antes, ningún acelerómetro implementado así que aquí se tuvo que diseñar una huella digital DIL24 y luego soldar un Socket para 24 pines en la PCB JDMR 0617 para poder instalar el chip o placa MEMS que se iba a utilizar. La placa MEMS STEVAL-MKL124V1 contiene pues un lsm303dlhc acelerómetro y l3gd20 giroscopio. Antes de que se pudiera comenzar a programar el lsm303dlhc era necesario familiarizarse con los registros del acelerómetro, para visualizar estos registros hubo que cavar en la hoja de datos. También se tuvo que buscar la dirección del lsm303dlhc. Tenga en cuenta que en el MKI124V1 tanto el acelerómetro como el giroscopio se combinan en un circuito integrado, por lo que se debe usar un protocolo de comunicación I²C o SPI para comunicarnos con ambos dispositivos esclavos.

Después de haber programado el acelerómetro todavía había que calibrarlo. Se tuvo que mirar en qué dirección apuntaban los diferentes ejes. Esto se describe en la hoja de datos, pero desde el lsm303dlhc es un cuadrado que no se sabía cómo se integró. También había una compensación debida a las imperfecciones que se necesitaban eliminar.

Aparte de la programación, se analizó la conexión interna para visualizar como debía ser la conexión externa de los pines. Una figura de los circuitos internos del STEVAL-MKL124V1 puede encontrar debajo.

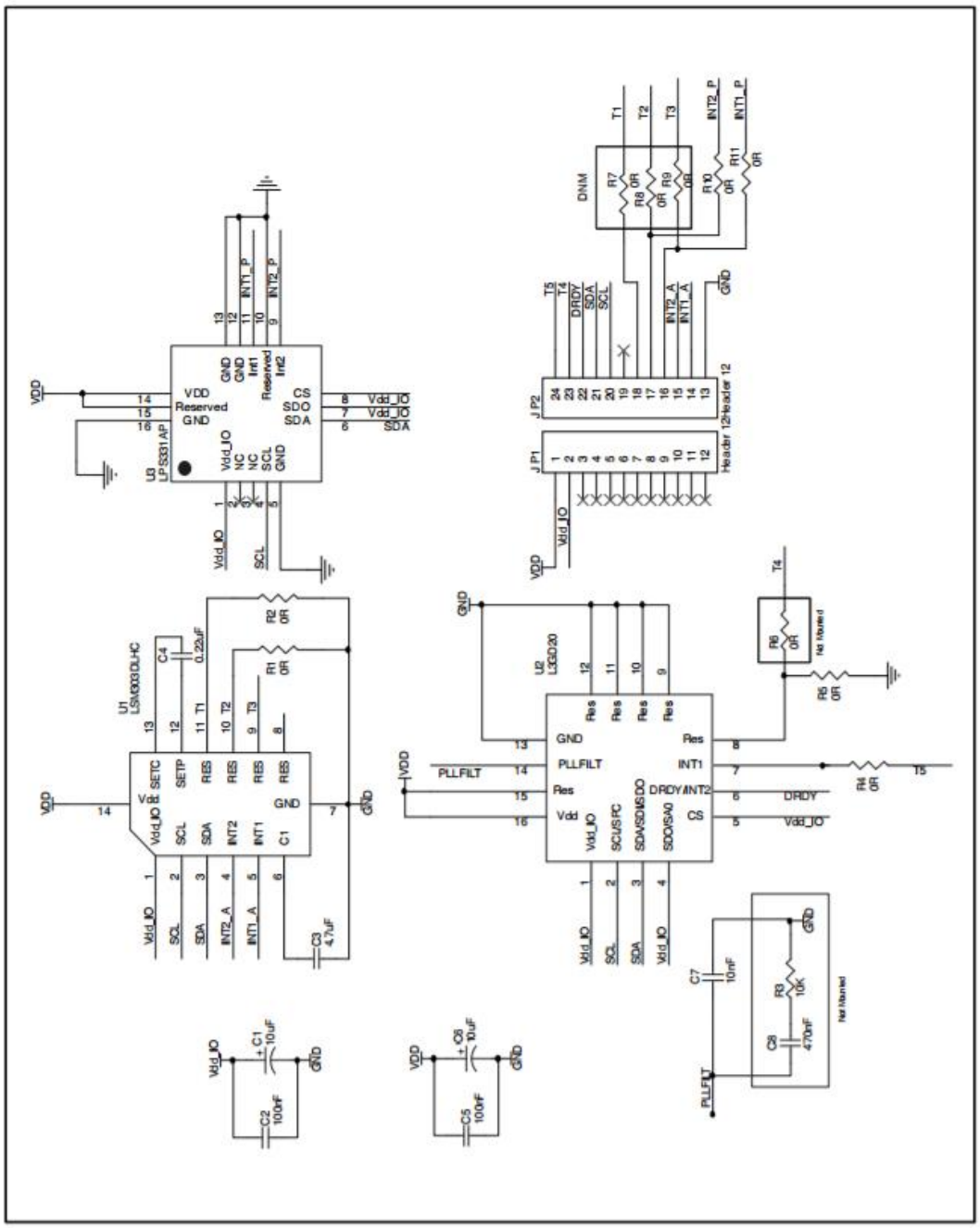
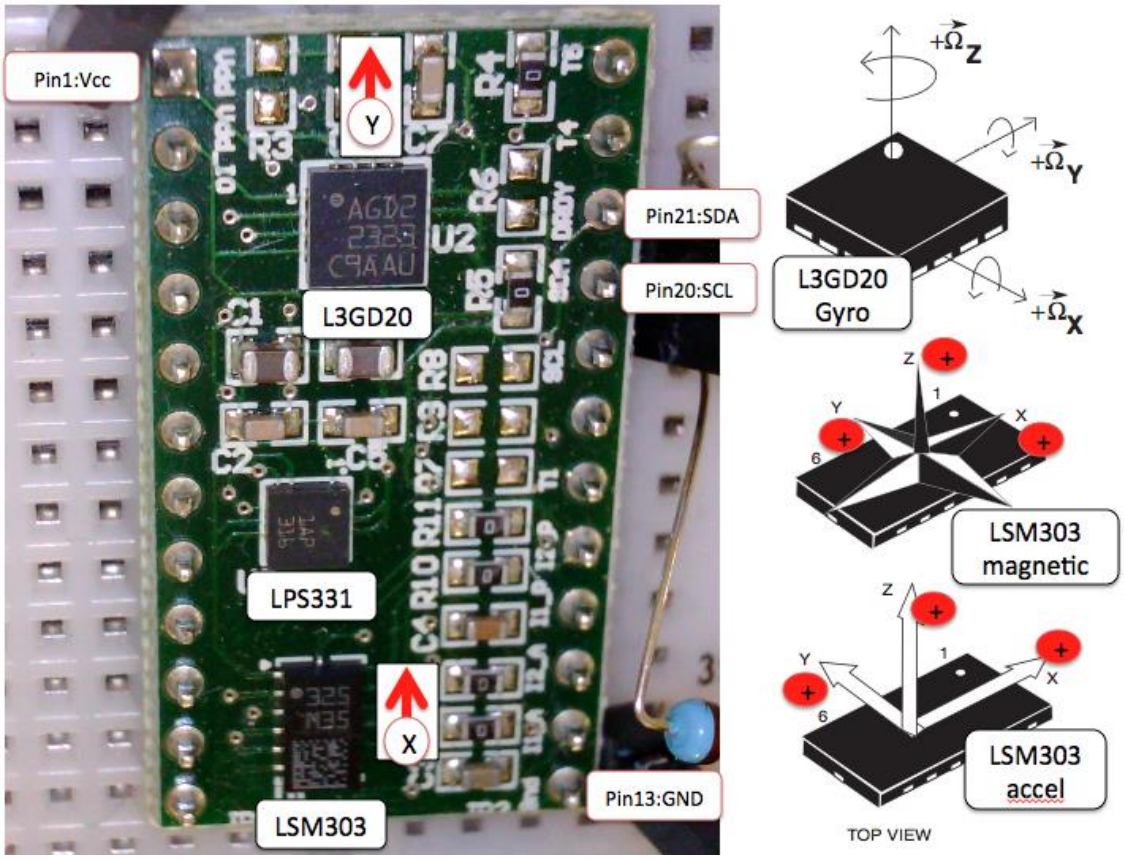
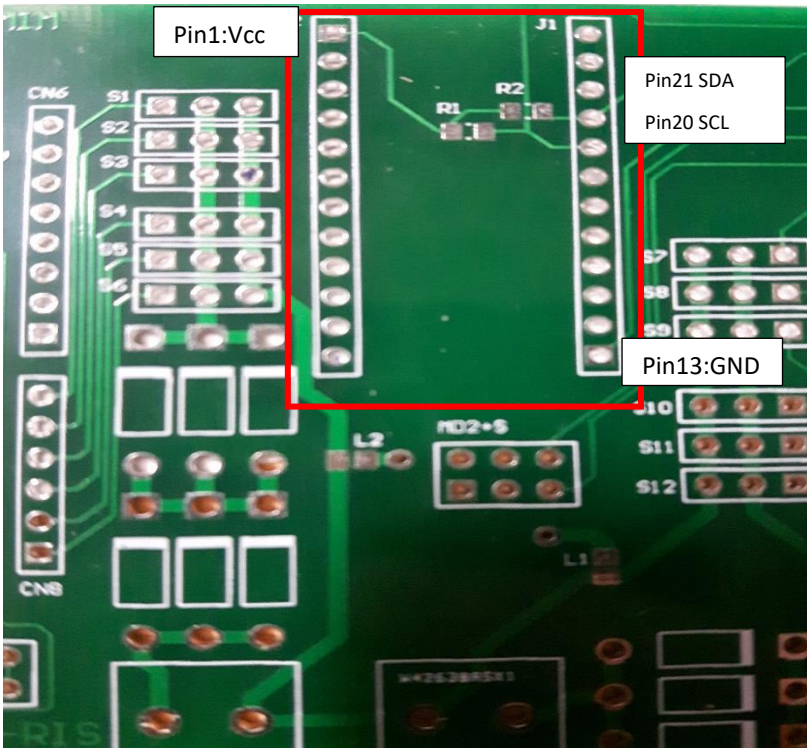


Figura 112: Esquema del circuito STEVAL-MKI124V1

Como se pudo ver en los circuitos internos lsm303dlhc las resistencias de Pull-up y Pull-down no estaban conectadas en el chip y por tanto se debía realizar la conexión de forma externa al chip. Es decir, en la PCB donde se diseñó el DIL de 24 pines también se tuvo que implementar unas huellas para dichas resistencias R1 y R2 conectadas del pin a los pines SDA y SCL a VDD del STEVAL-MKI124V1.





Figuras: 113 y 114 Configuración de los elementos del sensor IMU y conexiones y pines

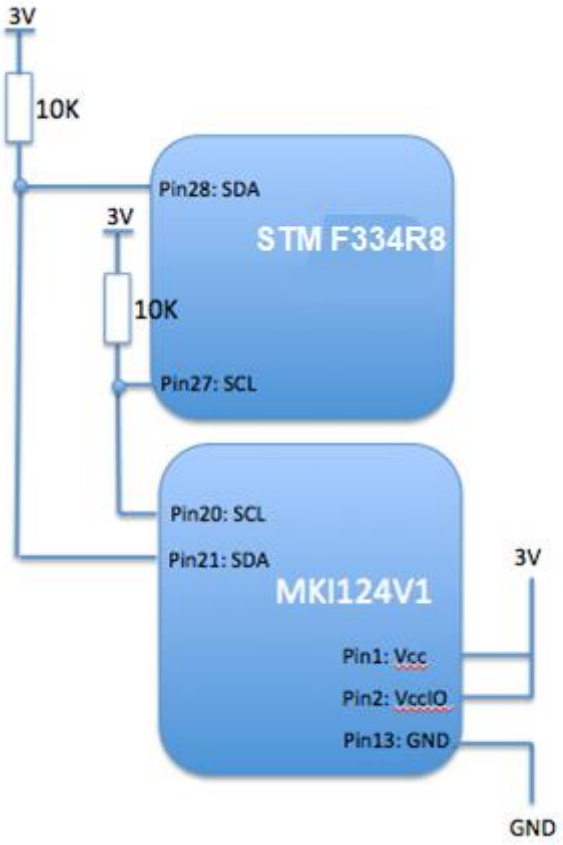
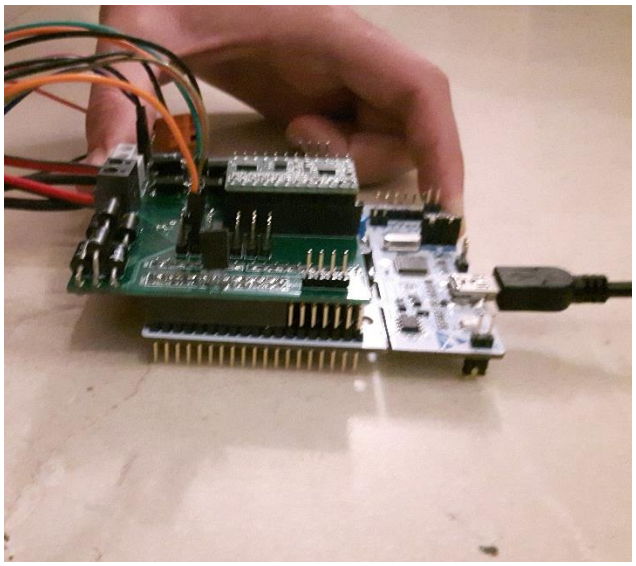
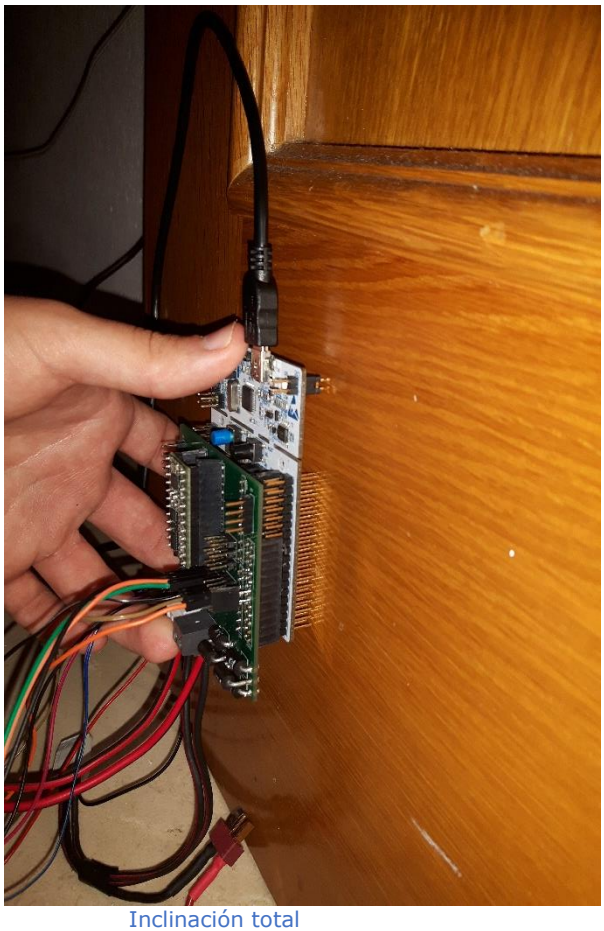


Figura 115: PCB JDMR 0617: DIL24 para el chip MEMS y huellas digitales para R1 Pull-up y R2 Pull-down

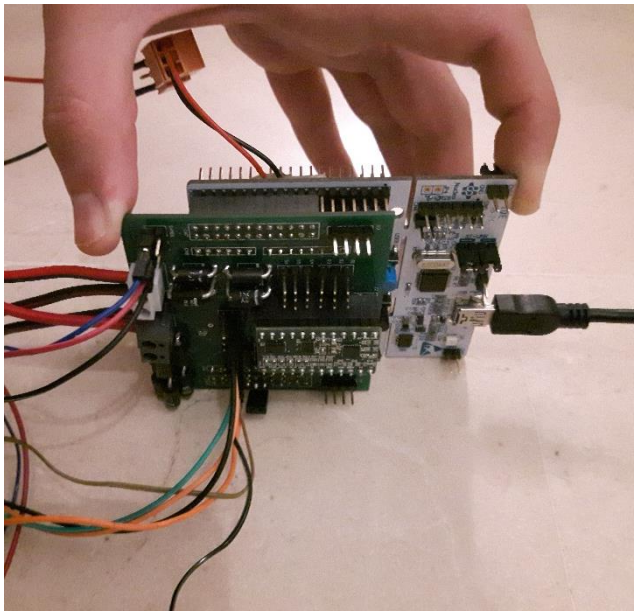
En las figuras a continuación se puede ver cómo se ha sostenido las tablas en diferentes posiciones para determinar en qué dirección estaban apuntando los diferentes ejes. Se volvió a mostrar los datos dados por la placa MEMS con el HyperTerminal "Termite 3.3".



Horizontal al suelo



Inclinación total



Balanceado total

Figuras 116, 117 y 118: Calibración del acelerómetro

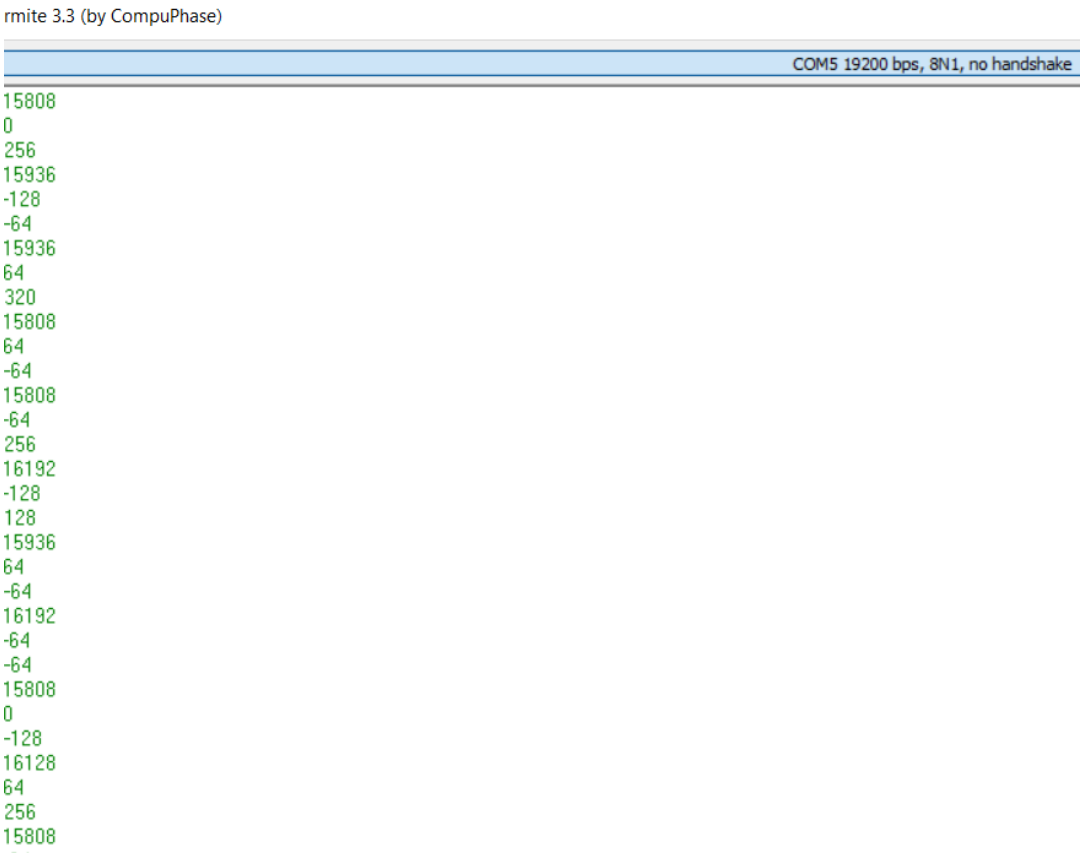


Figura 119: Valores dados para las diferentes posiciones a las que se ha expuesto con el HyperTerminal (Termite 3.3)

Dependiendo de la cantidad de bits que tenga el ADC del acelerómetro, su salida será un máximo de $2^{bits\ -1}$ - 1 y puede ser tanto positiva como negativa. De estas salidas es realmente fácil calcular la inclinación y el balanceo (pitch and roll). Estas son las únicas rotaciones que se necesita para este cuadrúpedo. Considerando las compensaciones, las rotaciones se pueden calcular mediante las siguientes ecuaciones:

$$inclinación/pitch = \ tan^{-1}\left(\frac{X_{aceleración} - X_{offset}}{Z_{aceleración} - Z_{offset}}\right) \qquad \textbf{(29)}$$

$$balanceo/roll = tan^{-1}\left(\frac{Y_{aceleración} - Y_{offset}}{Z_{aceleración} - Z_{offset}}\right) \qquad \textbf{(30)}$$

10.4 Giroscopio

El giroscopio mide la velocidad angular o de rotación también llamada la velocidad angular en 3 direcciones. En el STM32F3Discovery ya hay un giroscopio l3gd20. Sin embargo, en la placa de Nucleo no hay un giroscopio implementado, pero como se dijo antes utilizamos el STEVAL-MKL124V1. El circuito interno se encuentra en la parte del acelerómetro.

Para obtener las rotaciones a partir de la velocidad angular necesitamos integrarla con respecto al tiempo.

$$inclinación/pitch = \int \omega_y \cdot dt \qquad \textbf{(31)}$$

$$balanceo/roll = \int \omega_x \cdot dt \qquad \textbf{(32)}$$

Para obtener la velocidad angular primero se necesita transformar la salida del giroscopio que es análoga al acelerómetro $2^{bits\ -\ 1}$ - 1. Sin embargo, antes de transformar la salida de giroscopio primero se tuvo que buscar el valor de desplazamiento del giroscopio que se puede ver en el gráfico de abajo. Se encontraron estos valores de compensación de nuevo utilizando el HyperTerminal "Termite 3.3" y manteniendo el sensor inmóvil, sin moverlo y asegurándose de que no había velocidades angulares. Esto fue tarea bastante difícil ya que este sensor es muy sensible y con una pequeña vibración de la placa podia dar cambios en los valores de salida… Después de hacer esto se tomó la media de todos los valores salidos en el experimento. Así se pudo sacar la compensación o offset que se necesitaba para u funcionamiento cercano a un correcto funcionamiento del sensor.

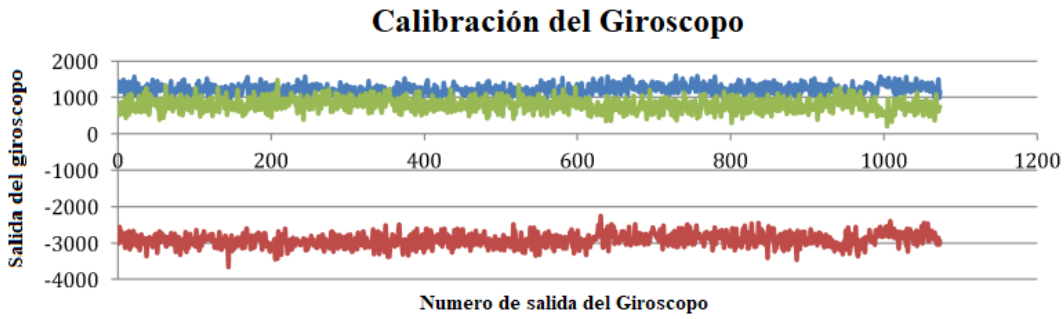


Figura 120: Calibración del giroscopio

En la hoja de datos, podríamos encontrar cómo transformar estos resultados.

Symbol	Parameter	Test condition	Min.	Typ.(2)	Max.	Unit
FS	Measurement range	User-selectable		±250		dps
				±500		
				±2000		
So	Sensitivity	FS = 250 dps		8.75		mdps/digit
		FS = 500 dps		17.50		
		FS = 2000 dps		70		
SoDr	Sensitivity change vs. temperature	From -40 °C to +85 °C		±2		%
DVoff	Digital zero-rate level	FS = 250 dps		±10		dps
		FS = 500 dps		±15		
		FS = 2000 dps		±75		
OffDr	Zero-rate level change vs. temperature	FS = 250 dps		±0.03		dps/°C
		FS = 2000 dps		±0.04		dps/°C
NL	Non linearity	Best fit straight line		0.2		% FS
Rn	Rate noise density			0.03		dps/(√Hz)
ODR	Digital output data rate			95/190/ 380/760		Hz
Top	Operating temperature range		-40		+85	°C

Figura 121: Datasheet del l3gd20

Dependiendo de nuestro rango de medición (250, 500 o 2000 grados por segundo) se debe encontrar la sensibilidad en milisegundos por segundo. (Tenga en cuenta que esto también puede calcularse fácilmente haciendo $\frac{\text{rango de medicion}}{2^{bits-1}}$.

Finalmente se podría calcular el ángulo en grados:

$$angle = angle + gyro\ output \cdot \frac{sensibilidad}{1000} \cdot dt \quad \textbf{(33)}$$

siendo dt el intervalo de tiempo entre dos mediciones. El valor de ángulo no es sólo la salida veces la sensibilidad y dt. Este término tiene que ser añadido al valor de ángulo anterior, ya que el ángulo seguirá aumentando con una determinada velocidad de giroscopio.

En un primer aspecto estos ángulos obtenidos parecían correctos, sin embargo, debido a la deriva del giroscopio y la integración continua dio ángulos erróneos después de un cierto lapso de tiempo. El giroscopio es muy preciso para tiempos cortos.

10.5 Combinación del giroscopio y el acelerómetro

Se podría combinar estos dos sensores con un filtro. Para esto se debería haber utilizado un filtro de Kalman que es muy bueno, pero es muy intensivo del procesador debido a la cantidad masiva de operaciones matemáticas. Otro filtro mucho más fácil que nos ahorró el estudio de la literatura del filtro de Kalman se llama el filtro complementario que es matemáticamente mucho más fácil y también mucho más fácil de entender.

Filtro complementario

$$angulo = a \cdot (angulo + ProporciónGiros \cdot dt) + (1 - a) \cdot anguloAcelero \quad \textbf{(34)}$$

El valor 'a' decide cuánto peso obtiene el giroscopio y el acelerómetro. Se elige un valor de 0.96, de esta manera por períodos de tiempo cortos el giroscopio tiene más peso y para largos períodos de tiempo obtenemos el valor del acelerómetro.

En la figura siguiente se puede encontrar un esquema del filtro complementario.

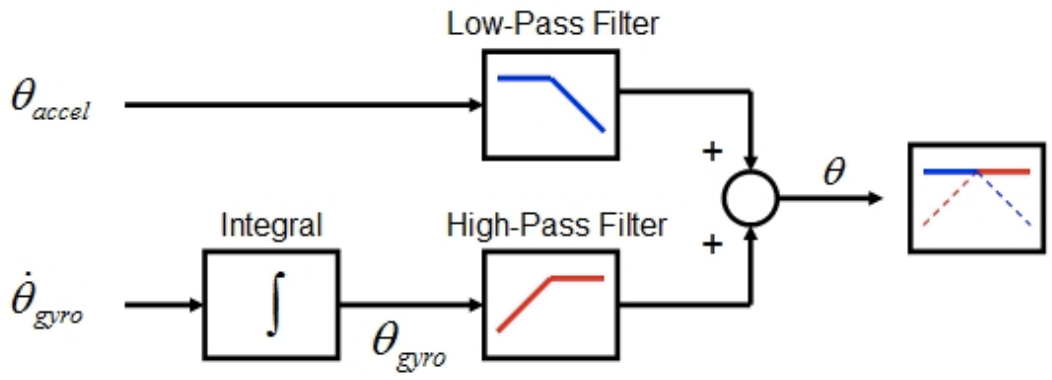


Figura 122: Filtro complementario

Se puede apreciar como el valor del acelerómetro pasa a través de un filtro Paso-bajo (Low-pass filter) y el valor del giroscopio después de la integración pasa a través de un filtro Paso-alto (High-pass filter).

11 Estudio del par

En la lista de componentes se pueden encontrar los motores "DM-RS0613MD" y "DMRS1513MD". Respectivamente, estos motores tienen un par de 6,5 kg.cm y 14,4 kg.cm cuando se accionan con 6V y un par de 7,5 kg.cm y 16,5 kg.cm respectivamente cuando se accionan con 7,4V. se hizo un estudio del par con las dimensiones de las patas de nuestros robots, ya que una sobrecarga de torque podría dar problemas de movilidad o dañar los servos. A primera vista un estudio de torque del robot cuadrúpedo parece muy difícil ya que las piernas pueden tomar una cantidad infinita de posiciones diferentes. Lo que se hizo fue buscar un escenario que diera los peores resultados. Es decir, asumir que solamente hubiera dos piernas que estabilizaran el cuerpo y así suponer que estas llevarían el peso completo del cuerpo. Además de eso, también se observó el pie como un extremo completamente fijo. Si los torques requeridos permanecieran dentro de los límites para este escenario, ciertamente permanecerían dentro de los límites para cualquier otro escenario posible imaginable.

11.1 Estudio del torque en el Robot 2GDL

Para calcular el peor de los casos, se utilizó la posición de la pierna en la que la distancia desde el centro del cuerpo hasta la articulación del fémur es máxima. Esta es la posición de la pierna en la que toma el ángulo de coxa, dependiendo de qué patas, 45 °, 135 °, 225 ° o 315 °.

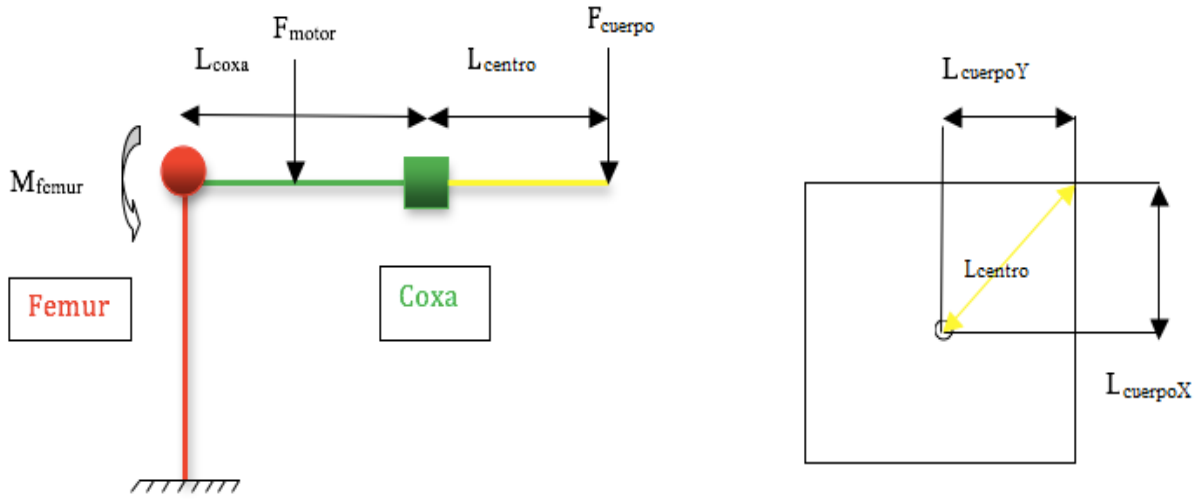


Figura 123: Estudio del Par en el 2GDL

Datos

L_{coxa} = 45 mm

$L_{cuerpoY}$ = 42 mm

$L_{cuerpoX}$ = 43 mm

F_{motor} = 51 gr

F_{cuerpo} = 500 gr

Cálculos:

$$L_{centro} = \sqrt{{L_{cuerpoY}}^2 + {L_{cuerpoX}}^2} = 60.11\ mm$$

$$M_{femur} = F_{motor} \cdot \frac{L_{coxa}}{2} + F_{cuerpo} \cdot (L_{coxa} + L_{centro}) = 53702.5\ gram \cdot mm$$

$$\approx 5.4\ kg \cdot cm$$

Como se puede ver, el par no supera los 7,5 kg.cm. Dado el hecho de que se podría tratar de uno de los peores de los casos, es muy poco probable que el par llegue a alcanzar incluso 5,4 kg.cm. Se puede concluir entonces que el DM-RS0613MD satisface para el robot cuadrúpedo de 2 GDL ya que incluso a su tensión más baja 6V nos da un par de 6.5 Kg.cm.

11.2 Estudio del torque en el robot 3GDL

Para calcular el peor escenario del robot 3GDL, utilizamos de nuevo la posición de la pierna con el ángulo de la coxa, dependiendo de la pierna, 45 °, 135 °, 225 ° o 315 °. Como puede verse en la figura, el par estará en su máximo si el fémur y la parte coxa están alineados

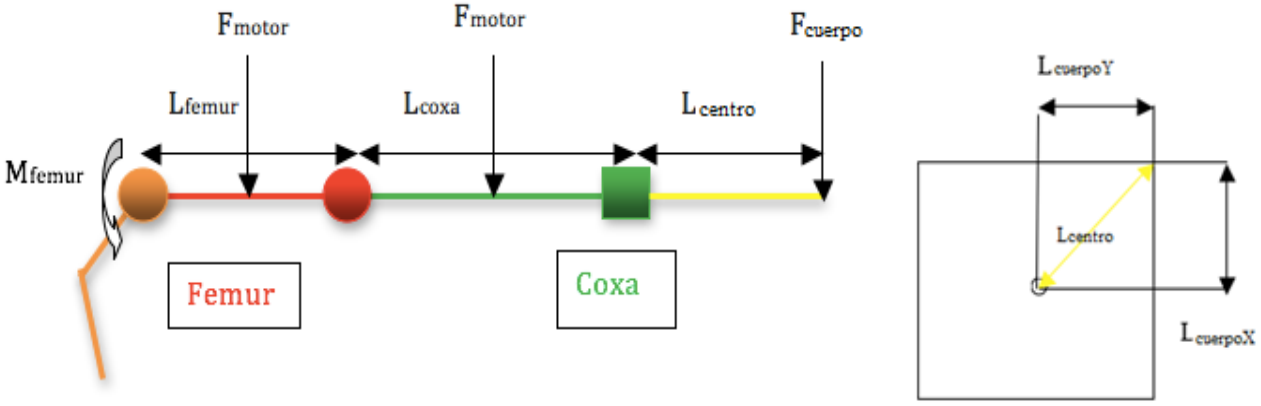


Figura 124: Representación gráfica con longitudes, fuerzas y momentos de pierna 3GDL

Datos:

L_{coxa} = 40 mm

L_{femur} = 60 mm

$L_{cuerpoY}$ = 63 mm

$L_{cuerpoX}$ = 63 mm

F_{motor} = 51 gr

F_{cuerpo} = 500 gr

Calculos:

$$L_{centro} = \sqrt{{L_{cuerpoY}}^2 + {L_{cuerpoX}}^2} = 89.1\ mm$$

$$M_{femur} = F_{motor} \cdot \left(\frac{L_{coxa}}{2} + \frac{3 \cdot L_{femur}}{2}\right) + F_{cuerpo} \cdot \left(L_{coxa} + L_{centro} + L_{femur}\right) =$$

$$= 10\ Kg \cdot cm$$

En este caso, el par supera los 6.5 kg.cm, incluso también a los 7,5 kg.cm. Sin embargo, este es el peor de los casos, y es muy poco probable que 10 kg.cm nunca se alcanzará. Por precaución utilizamos motores que pueden manejar un par de 15 kg.cm como son los DM-RS1513MD.

12 Cominicacion

La comunicación inalámbrica fue la parte más difícil porque llevó mucho tiempo para depurar los muchos errores que se tuvieron. Parecía que, para cada error resuelto, aparecieron dos nuevos. Sin embargo, finalmente se estableció una comunicación inalámbrica fiable. Se implemento un propio controlador o mando utilizando un joystick inalámbrico y un tablero STM32F3Discovery combinado con el tablero ITACA RMR como se explica en la lista de componentes. Para hacer la comunicación inalámbrica se utilizaron dos módulos de RF de STM SP1ML como módulo de radio.

Una representación esquemática de la comunicación se puede encontrar en la imagen que aparece a continuación.

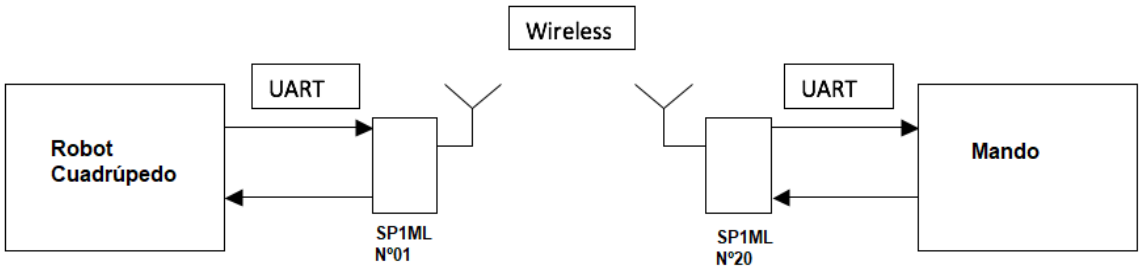


Figura 125: Representación esquemática de la comunicación inalámbrica

Para la transmisión entre los microcontroladores SP1ML y STM32 se utiliza un UART, que está disponible en los microcontroladores. Recuerde que cuando las impresoras y los ratones de ordenadores tenían un cable muy grueso hace mucho tiempo, estos dispositivos estaban usando una UART. Ahora esto está casi completamente reemplazado por un USB. En primer lugar, para poder usar una UART tuvimos que familiarizarnos con el protocolo. Se necesitarían dos cables: un receptor y un cable transmisor. El UART convierte los datos paralelos de la CPU en datos en serie y transmite datos de forma asíncrona, lo que significa que no se tiene una sincronización mutua entre los dos dispositivos que están enviando por reloj. Si ambos dispositivos con comunicación asíncrona saben cuánto tiempo se tarda en enviar un solo bit y saber cuándo iniciar o detener los bits transmitidos por muestreo, no se necesitaría una tercera línea de reloj como si que se necesita para la comunicación síncrona USART en modo síncrono. Por lo tanto, UART agrega un bit de inicio y detención al paquete de datos antes de transferir. Cuando el receptor recibe un bit de inicio, comienza a leer los bits entrantes a la velocidad de baudios seleccionada para la comunicación. Como se ha dicho antes, tanto el receptor como el transmisor deben tener la misma velocidad en baudios. La lectura se detiene cuando el UART detecta un bit de parada. Debajo se puede encontrar la diferencia entre USART en modo síncrono y UART (la más sencilla a utilizar para este proyecto) en una representación esquemática.

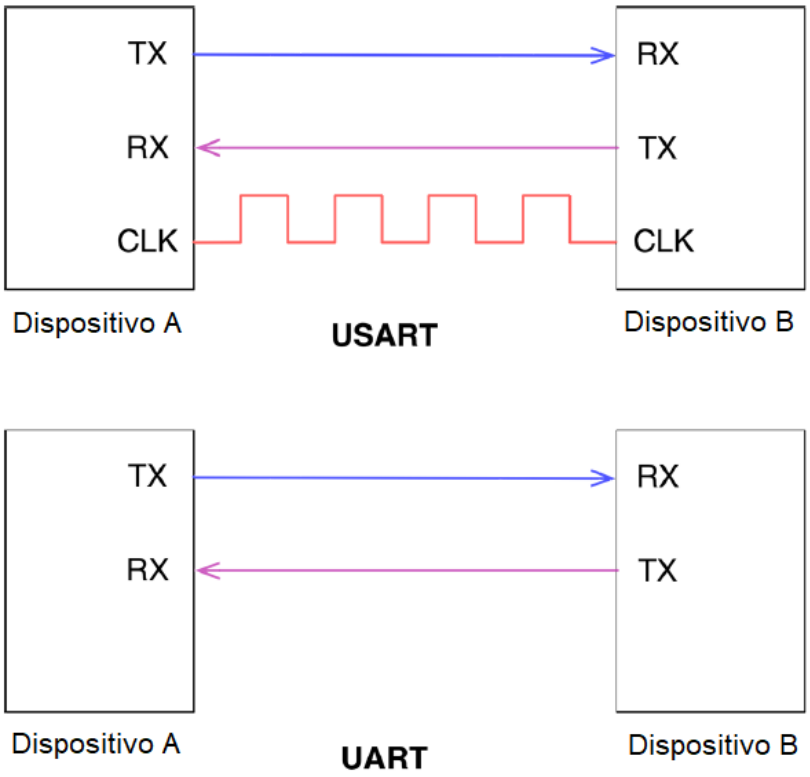


Figura 126: Presentación esquemática de USART y UART

También puede encontrar las representaciones de bit de USART y UART.

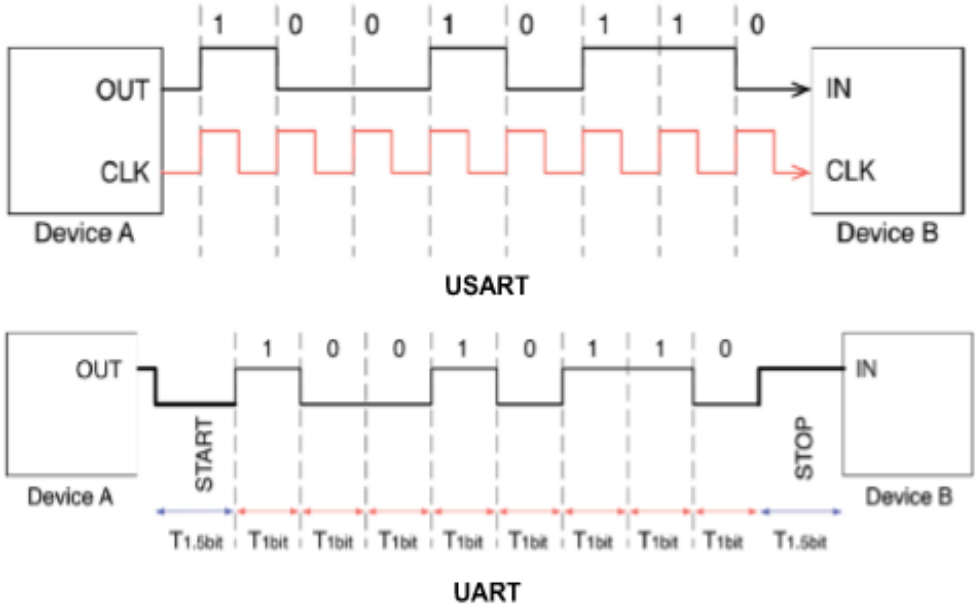


Figura 127: Representaciones de bit del USART y UART

Para el controlador sólo se utiliza la línea de transmisión de la UART y para el cuadrúpedo sólo se utiliza el receptor. No se utilizó un acuse de recibo que también implicaría el uso de la línea de recepción y transmisión de, respectivamente, el controlador y el cuadrúpedo. Tampoco se utilizó el CTS Clear To Send y el RTS Request To Send. Tanto el controlador como el SP1ML tienen un buffer que contiene los datos necesarios para ser transmitidos. Si este búfer está lleno, los datos se pierden y la comunicación incluso puede detenerse. Para evitar esto, es necesario estar seguros de que estos tapones no pudieran estar llenos calculando las velocidades de transmisión.

12.1 Configuración de los SP1ML

se necesita conocer la dirección de los dos dispositivos SP1ML para poder enviar información entre sí. También se ha tenido que configurar la velocidad en baudios las UARTs en ambos modulos SP1ML. Esta velocidad tenía que ser la mismo para ambos canales de comunicación correspondientes. Se ha configurado los SP1ML con el programa utilizado por el Hyperterminal ‘TERMITE3.3’. Primero se conecto un dispositivo a la interfaz de serie a USB. Y luego el otro.

Para este procedimiento antes se tuvo que estudiar los comandos de referencia proporcionados por STM en el manual de SP1ML ‘ SPIRIT1 868 and 915 MHz low power RF modules with integrated Microcontroller’

El módulo SP1ML está provisto de firmware que admite el cable serial inalámbrico reemplazo. Hay dos modos operativos, modo de comando y modo de funcionamiento. los El modo de comando permite la configuración del módulo y la interrogación del estado Juego de comandos 'AT'. En el modo de funcionamiento, el módulo cumple su función principal como transceptor inalámbrico. Después del encendido o reset, el módulo se inicia en el modo de funcionamiento con la configuración actual cargada desde EEPROM.

En el modo de funcionamiento, los datos recibidos del host en la interfaz UART serán inalámbricos transmitido por la radio SP1ML usando los ajustes de configuración actuales para la frecuencia, los datos velocidad, modulación y potencia de salida.

En el modo de comando, el módulo aceptará comandos para configurar los interroga el estado del módulo. Para entrar en el modo de comando, la secuencia de escape '+++' se emite al módulo del modo de funcionamiento. La secuencia de escape debe ir precedida de una retraso de 500 milisegundos en el que no se transmiten otros datos. Los tres caracteres '+' de la deben enviarse dentro de los 500 milisegundos entre sí. El módulo emite el respuesta 'OK' si el modo de comando se introduce correctamente.

Los comandos emitidos al módulo están en formato 'AT' y utilizan caracteres ASCII, con 'A' y 'T', entonces uno o más caracteres para el comando específico, seguido de cualquier datos específicos del comando adicional y termina con un retorno de carro <CR>.
El retardo entre cada carácter consecutivo de un comando debe ser inferior a 8 segundos, de lo contrario el módulo se agotará y descartará los caracteres ya recibidos.
Si el módulo recibe un comando no válido, enviará la respuesta 'ERROR'. Todas las respuestas emitidas por el módulo se terminan con una alimentación de línea y un retorno de carro,
<LF> <CR>.

A continuación, se ofrece una tabla con los comandos mas utilizados para configurar los SP1ML:

Table 9. Command reference

Command	Description
ATO	***Enter operating mode*** This command is issued to exit command mode and enter operating mode where the module fulfills its primary purpose as a wireless transceiver using the current configuration. *Response:* OK
AT/V	***Read module version information*** Reports the module hardware and firmware version information. *Response:* **SP1ML-xxx HW:Vy FW:Vy.zz** Where **xxx** is 868 or 915, **y** is a single digit major version number and **zz** is a double digit minor version number.
ATIn	***Read an information register*** Reads the current value from an information register, where **n** is the information register number. See information registers table. *Response:* **<REGISTER NUMBER>:<REGISTER NAME>=<VALUE>** **ERROR PARAM** if an invalid register is specified.
ATSnn?	***Read a configuration register*** Reads the current value from a configuration register, where **nn** is the configuration register number. See configuration registers section. *Response:* **<REGISTER NUMBER>:<REGISTER NAME>=<VALUE>** **ERROR PARAM** if an invalid register number is specified.
ATSnn=x	***Write a configuration register*** Writes a new value to a configuration register, where **nn** is the configuration register number and **x** is the value. The new configuration will be in effect until the next module reset. See configuration registers section. *Response:* **OK** if the value is written successfully. **ERROR PARAM** if an invalid register number is specified. **ERROR VALUE** if an invalid value is specified.
AT/S	***Read all configuration registers*** Reads the current values of all configuration registers. See the configuration registers section. *Response: (one line for each register)* **<REGISTER NUMBER>:<REGISTER NAME>=<VALUE>**
AT/C	***Store the current configuration*** Stores the current module configuration registers. The stored configuration will be reloaded anytime the module is reset. *Response:* **OK** if the configuration is stored successfully. **ERROR** if storing the configuration failed.

Tabla 128: Lista de comandos para configurar los SP1ML a la misma velocidad y dirección.

Visto esto sólo queda fijar la velocidad en baudios y dirección. Se utilizarán los comandos AT/S para leer todos los registros configurado. Acontinuación nos aparecerán en la pantalla todos los registros. Comprobamos que el registreo de la velocidad S00 BAUD_RATE es igual a 115200 si no es igual escribiremos el comando ATS00=115200 para que cambie el valor al que deseamos. Si el si el valor escrito ha sido cambiado satisfactoriamente responderá con un OK. Este valor de 115200 debera ser el mismo en los dos registros de BAUD_RATE de los dos SP1ML.

A continuación, se procederá a verificar las direcciónes en cada SP1ML:

Habran dos direcciónes en cada SP1ML. Una la dirección propia del SP1ML dada por el registro S15:SOURCE_ADDR. Otra la dirección de destino donde enviara o recivira información es la S16 DESTINATION_ADDR.

En el robot cuadrúpedo se ha instalado el SP1ML con dirección propia nº01. En el mando se ha instalado el otro SP1ML con dirección propia nº20.Por tanto, la dirección de destino del SP1ML nº01 sera la 0x20 (ya que recibirá información del mando). A su vez, la dirección de destino del SP1ML nº20 instalado en el mando será la de 0x01 (ya que enviará a esta dirección la información). El procedimiento para poner las direcciónes de destino correctas es el mismo que para cambiar la BAUD_RATE explicado anteriormente con el comando de AT/S16= 0x…

Una vez configurado cada modulo de RF para salir del modo comando y pasar al modo funcionamiento guardando los configuraciones o cambios realizados se escribirá en TERMITE3.3 el comando de salir y guardar ATO. Si todo ha ido bien el modulo responderá con un OK. Entonces estaremos listos para empezar la comunicación entre robot y mando.

En la figura siguiente se muestra un ejemplo de configuración del SP1ML n01:

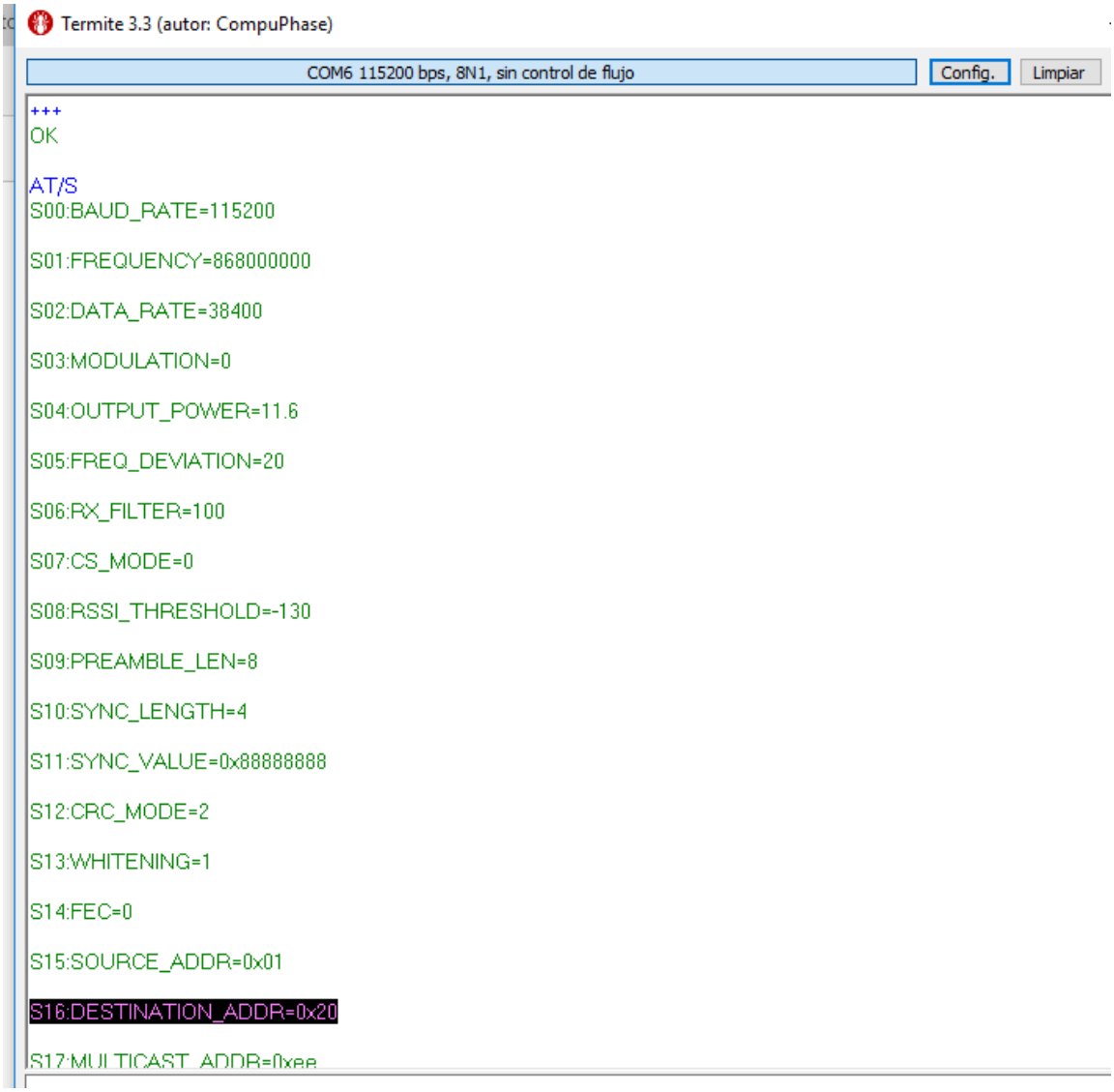


Figura 129: configuración de los SP1ML.

12.2 Joystick

El joystick se puede ver como 2 potenciómetros, uno en la X y uno en la dirección Y, por lo que tendrá que utilizar dos ADC para convertir ambas señales. El ADC o convertidor analógico a digital convertirá, como su nombre indica, la señal analógica dada por los potenciómetros a una señal digital que nuestro procesador puede poner en funcionamiento. Se utilizó un ADC de 12 bits, el principio de funcionamiento se describe a continuación con 4 bits.

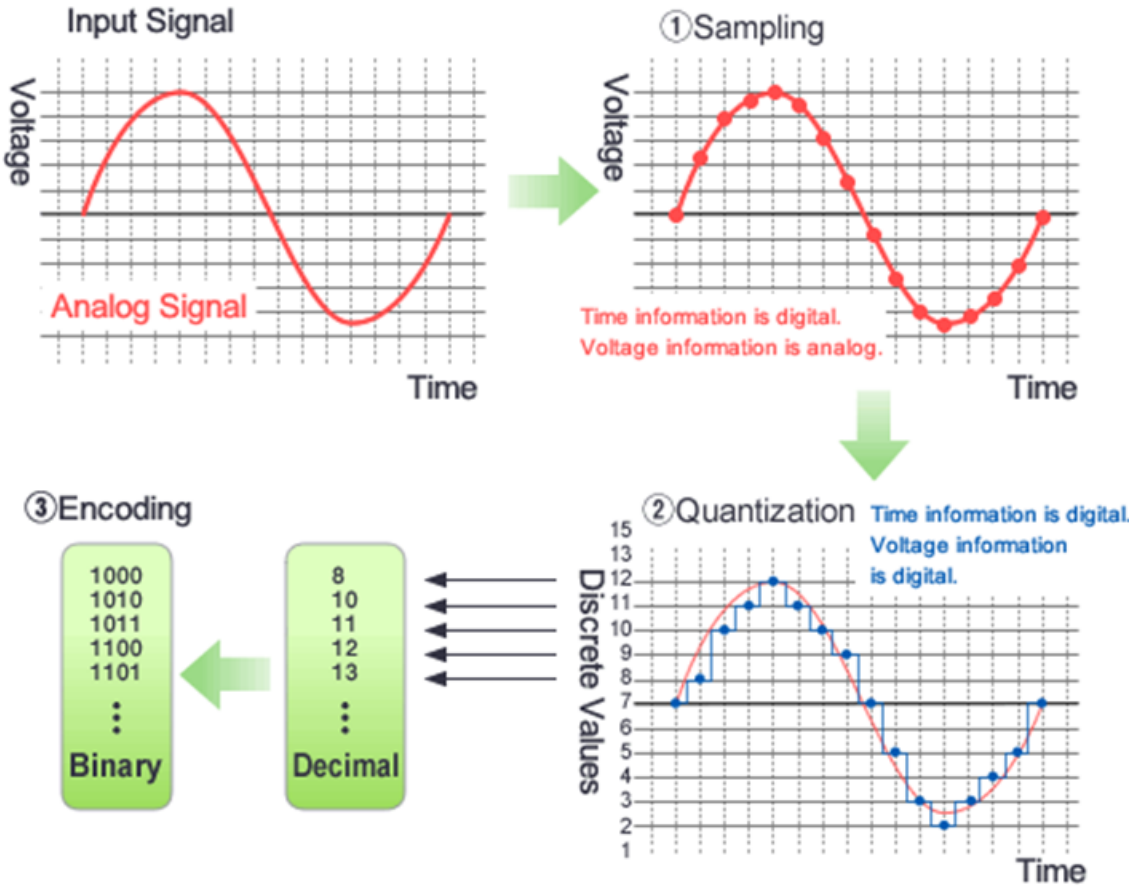
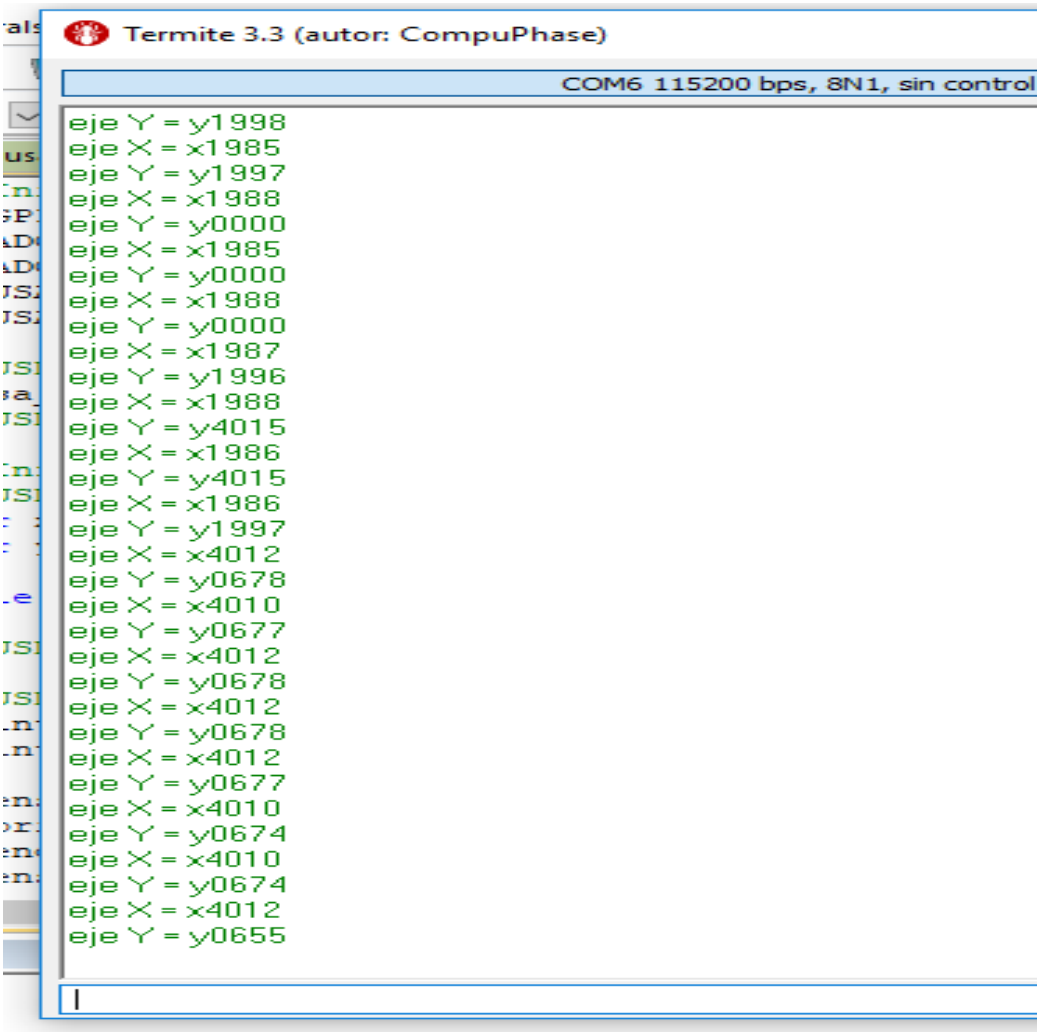


Figura 130: ADC del joystick

El valor obtenido por el joystick será un valor entre 0 y $2^{12} - 1 = 4095$. Este valor tendrá que ser posteriormente convertido en el programa a los valores deseados para X e Y o para otros valores.



Datos de salida del joystick visualizados por el HyperTerminal.

12.3 Comunicación del 2GDL

La comunicación inalámbrica del 2GDL consistió únicamente en el joystick inalámbrico. No hubo botones adicionales implementados para dar comandos adicionales al robot. La comunicación era muy poco fiable ya que en el momento en que hicimos el 2GDL no se tenía el conocimiento correcto todavía. Ambas velocidades UART eran 19200 bits / s. El hecho de que ambas velocidades fueran de un rango bajo fue muy malo ya que a veces sufría interferencias o la comunicación podía ir lenta (aunque no debería de ser asi). Mas adelante subimos la velocidad de la UART de los dos a 115200 ya que para un buen funcionamiento los dos módulos SP1ML deben trabajar a la misma frecuencia de baudios. No hizo falta ningún manejador de errores que hubiera hecho que la comunicación se detuviera. A partir de este cambio la comunicación fue la correcta y esperada.

12.4 Comunicación 3GDL

La comunicación del cuadrúpedo 3GDL fue finalmente, la misma que en el de 2GDL, una comunicación fiable continua. También se actualizo el controlador con 6 botones para dar ciertos comandos al cuadrúpedo. Uno de estos botones ya estaba implementado en el joystick, pero no lo usamos en la comunicación del 2GDL. En la comunicación del 3GDL usamos los 5 botones de la botonera de la PCB ITACA RMR mas el implementado en el propio joystick.

A continuación, puede encontrar una imagen del mando inalámbrico y sus botones.

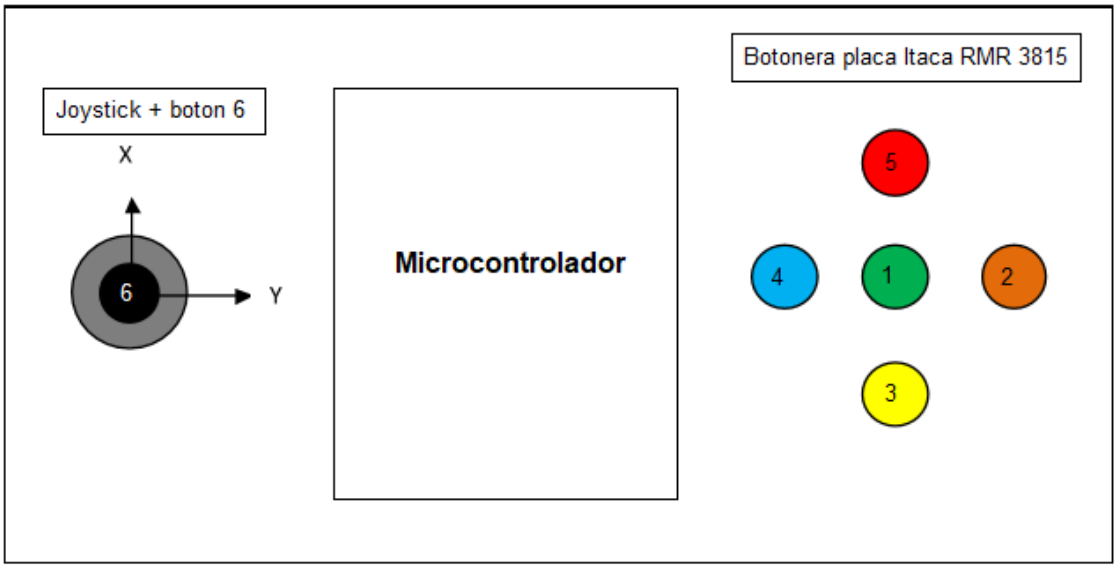


Figura 131: Mando inalámbrico con botonera para el robot 3GDL

Funcionalidad de los botones:

1. Botón verde: Ir a la posición de almacenaje o para ser guardado el robot.
2. Botón marrón: Mover el robot en la dirección Z y rotar alrededor del eje Z quedándose el cuerpo y patas en el mismo punto.
3. Botón amarillo: Rotar alrededor de los ejes X e Y (balanceo-roll e inclinación-pitrch).
4. Botón azul: Hacer que el cuerpo se mueva en la dirección X e Y con las piernas en la misma posición.
5. Botón rojo: Hacer que el robot camine en la dirección X y pueda girar.
6. **Botón negro del Joystick**: Hacer que el robot camine en dirección X e Y.

Estos botones son interrupciones externas, lo que significa que cuando se pulsan el programa entra en interrupción y procesa el código dado a la interrupción. Es una buena práctica poner los botones en interrupciones ya que de lo contrario pulsar los botones no hubiera tenido efecto ninguno si sólo fuera una entrada normal y el programa estuviera procesando otra parte del código. Tenga en cuenta que en la programación para la comunicación inalámbrica esto no habría importado mucho porque no es una programación muy grande y el bucle se ejecuta al menos cada 200ms.

Comunicación final:

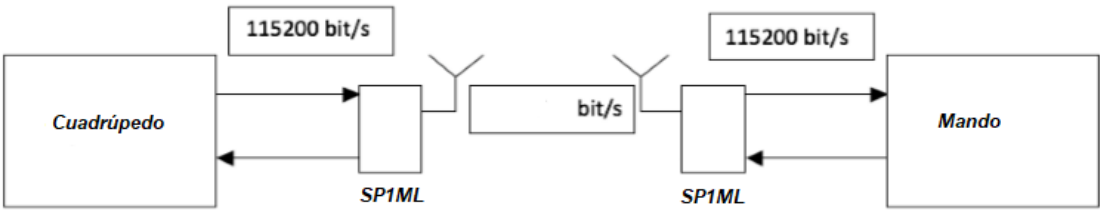


Figura 132: Representación esquemática de la comunicación fiable para el robot 3 GDL

Al elegir las velocidades de baudios vistas en la figura anterior, es casi imposible que el búfer esté lleno, ya que el cuadrúpedo recibe la información a la misma velocidad que la que el mando la transmite. Esto nos ofrece pues combinado con el uso de un manejador de errores (Interupciones) que cuando se encuentra con un error se asegura de que el programa comienza de nuevo desde su bucle inicial y, por lo tanto, se asegura de que el programa y la comunicación no se bloquea nos proporciona una comunicación bastante fiable. Además, esta vez se utilizaron interrupciones para leer los datos, esto hace que el programa sea mucho más limpio y hace que la reacción del robot sea más rápida.

En la imagen de abajo puede encontrar una captura de pantalla de los datos que enviamos.

Figura 133: Datos transmitidos

Se ha puesto una ‘x’ antes de los datos en la dirección x y una ‘y’ antes de los datos en la dirección y. De esta manera, el receptor puede distinguir ambos tipos de datos y convertirlos correctamente a los valores deseados. También se puede ver "aaaaa", "bbbbb" y "ccccc", este es el envío de datos cuando se presiona un botón. Estos caracteres son elegidos arbitrariamente en el código. Cuando el cuadrúpedo recibe caracteres diferentes de x o y, se ejecuta un comando que corresponde al carácter.

13 Formas de dar los pasos

Para determinar el paso que se va a utilizar en nuestros cuadrúpedos 2GDL y 3GDL primero se realizó un estudio exhaustivo de la literatura sobre la marcha de los animales como perros y gatos, ya que estos son animales de 4 patas. También distinguimos diferentes tipos de pasos que se explicarán a continuación.

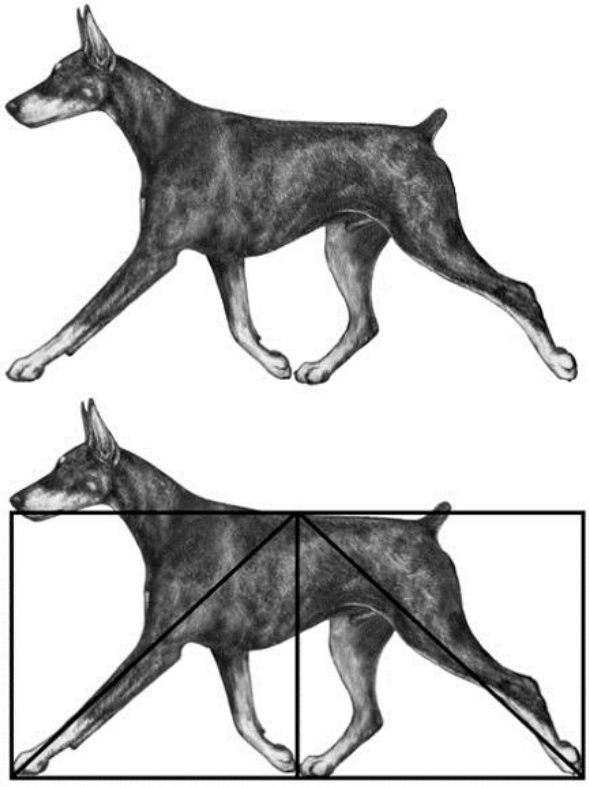


Figura 134: Paso de trote - perro

Figura 135: paso de gatear - gato

13.1 Marcha de trote

La marcha de trote que se puede ver en la figura del perro de arriba es un paso muy simple. Con el paso del trote se levantan dos patas diagonales y se mueven hacia adelante, durante este movimiento de elevación y avance de estas dos patas las otras patas diagonales se mueven hacia atrás lo que da una traslación delantera del cuerpo. Se podría intuir directamente que se trata de una marcha rápida ya que dos piernas se mueven a la vez, la marcha también tiene menos estabilidad ya que dos piernas se levantan al mismo tiempo dejando el cuerpo sólo estabilizado por las otras dos. En esta forma de movimiento se usa estabilidad dinámica,

más adelante se podrá observar también la estabilidad estática. Durante el paso de trote el cuerpo mantiene su estabilidad debido al movimiento hacia atrás de las piernas que lo hace dinámicamente estable. Otro ejemplo de animal que utiliza la marcha del trote es el caballo.

13.2 Marcha de arrastre o gatear

La marcha de arrastre (gatear) tiene estabilidad estática en lugar de estabilidad dinámica como se ve en el paso de trote. Con el paso de fluencia sólo se levanta una pierna en un momento, el cual que deja las otras 3 patas en el suelo estabilizando el cuerpo. Sin embargo, tres patas no son suficientes para estabilizar el cuerpo. Las tres patas necesitan estar en una posición que mantenga el centro de gravedad dentro del triángulo virtual formado por las tres patas, esto se llama estabilidad del trípode. Si el centro de gravedad no está en el triángulo, el cuadrúpedo caerá.

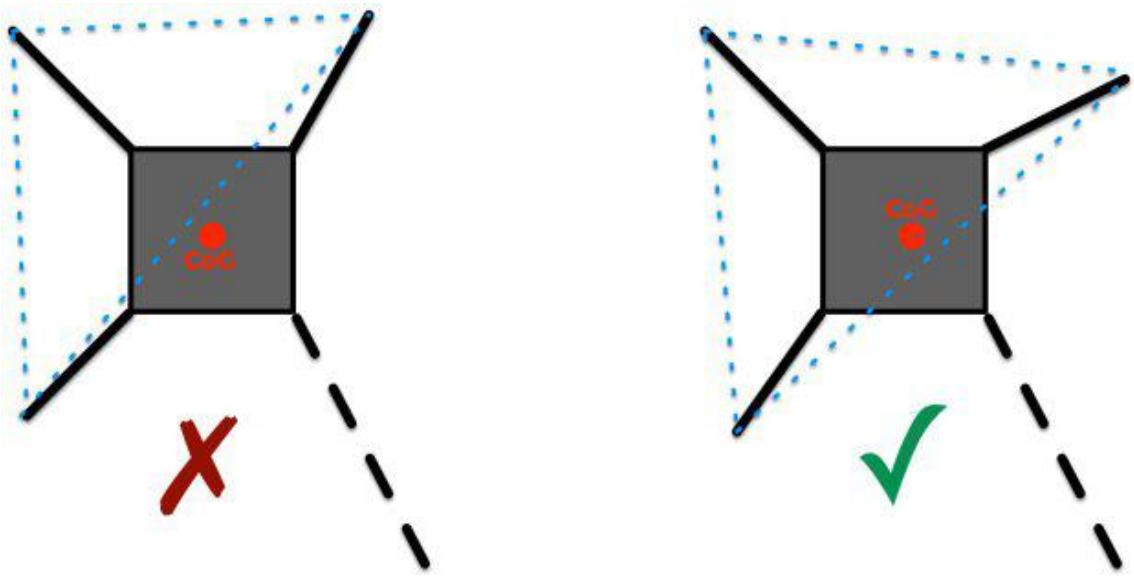


Figura 136: Estabilidad del trípode

13.3 Paso del cuadrúpedo 2GDL

Para el cuadrúpedo 2GDL se utilizó un paso de deslizamiento de arrastre o lo que se nombra como gateo, como se explicó en el párrafo anterior.

13.3.1 Movimiento para caminar

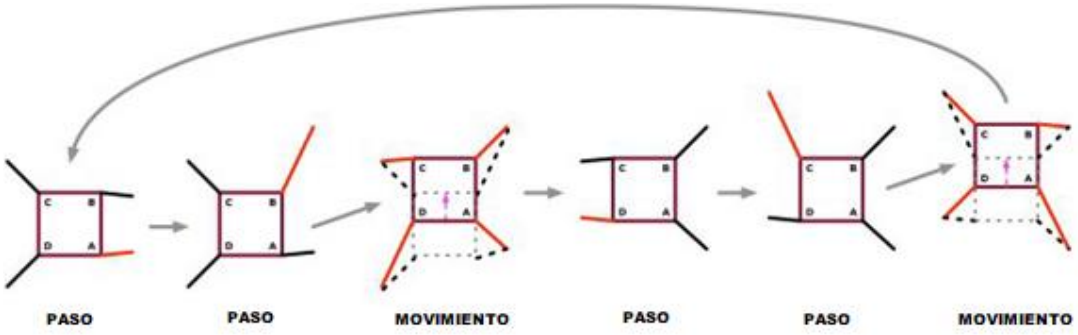


Figura 137: Secuencia de pasos de la caminata de gateo – Robot 2GDL

Los pasos de la marcha se pueden ver en la imagen de arriba:

- Primero vamos desde la posición inicial hasta el primer paso que es la posición inicial en la imagen.
- Luego levantaremos y moveremos la pierna anterior derecha hacia delante y hacia abajo después del movimiento hacia delante.
- Ahora movemos el cuerpo hacia adelante con todas las piernas en el suelo, lo hacemos moviendo todas las piernas hacia atrás sin levantarlas, es decir, tirando el cuerpo hacia delante (por pura inercia prácticamente).
- Después del movimiento hacia adelante se levantará la pierna trasera izquierda, se moverá hacia adelante y hacia abajo después del movimiento hacia delante.
- En este paso se levantará la pierna anterior izquierda y se moverá hacia adelante, después del movimiento hacia adelante se moverá la pierna otra vez hacia abajo.
- En el último paso se moverá el cuerpo nuevamente hacia adelante moviendo todas las piernas hacia atrás mientras están todavía en el suelo (por pura inercia de nuevo).
- Finalmente se moverá la pierna posterior derecha hacia adelante y luego hacia abajo para volver así a la primera posición de la secuencia y empezar a repetir los mismos pasos de nuevo.

Tenga en cuenta que este paso sólo funciona para el movimiento hacia delante, el movimiento hacia atrás será similar, pero las piernas obviamente se mueven hacia atrás y el orden de las piernas también será diferente. Lo que si se tendrá en cuenta es que también se deberán

levantar y posteriormente bajar las piernas. Como explicamos anteriormente, el robot de 2GDL no tendra control sobre el eje Z.

Se puede levantar las piernas aumentando el ángulo del fémur, esto también cambiará de sitio las posiciones X e Y, pero no significa un problema ya que la pierna está en el aire. Siempre obtenemos un trípode estable con los pasos dados. El centro de gravedad estará siempre en el triángulo formado por las 3 piernas que pisen en ese momento el suelo y así el cuerpo no caerá.

13.3.2 Rotación para el 2GDL

Para la rotación se gira primero el cuerpo. Una vez el cuerpo se ha girado se mueve cada pierna por separado (para que el robot no piedra el equilibrio) de nuevo a la posición inicial. Primero se levantaría una pierna luego se giraría el mismo ángulo que ha girado el cuerpo y finalmente se bajaría hasta tocar el suelo. Así se haría con las demás piernas. De esa manera se obtendría una rotación completa. En algunos casos, no se consiguió un trípode muy estable ya que el triángulo formado por las 3 patas estaba al límite con el centro de gravedad. Sin embargo, no causaba ningún problema, ya que en estos casos cuando el cuerpo empezaba a caer, la pierna en movimiento ya había tomado la posición correcta para estabilizar el cuerpo de nuevo.

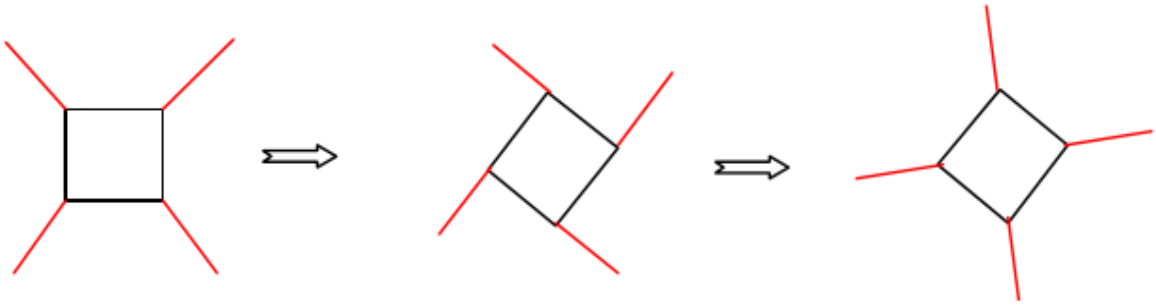


Figura 138: Rotación progresiva - rotación – 2GDL

Otro algoritmo fácil de implementar para la rotación hubiera sido la rotación ‘dos a dos’ más común en la marcha de trote. Aquí se girarían dos piernas a la vez mientras que las otras rotan con el cuerpo posteriormente. Es decir, por ejemplo, girar las piernas anterior derecha y posterior izquierda a la vez, y a continuación girar las otras dos piernas que faltan junto con el cuerpo.Esos si todas giraran en la misma dirección de giro y el mismo ángulo de giro. Esta manera hubiera podido ser también una opción de rotación en el cuadrúpedo 2GDL, pero en este caso la que si se utilizó fue la rotación progresiva explicada más arriba.

13.4 Marcha del cuadrúpedo 3GDL

13.4.1 Movimiento para caminar

Para la forma de caminar del cuadrúpedo 3GDL se utiliza una marcha de gateo y una marcha de trote. La marcha de gateo consiste en 20 pasos. El principio de la marcha es que cuando una pierna se levanta el cuerpo se mueve hacia adelante lo que resulta en un trípode estable

cuando se utiliza la orden de la pierna derecha. La marcha del trote se puede derivar de la rotación como se explica en la página siguiente en el punto 13.4.2.

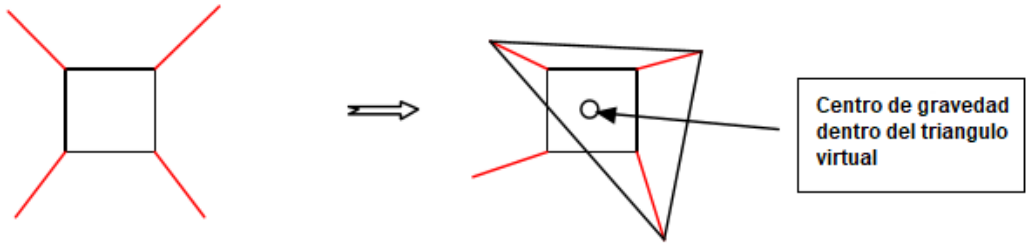


Figura 139: Caminata de gateo - caminar – 3GDL

En la siguiente tabla se presentan los movimientos consecutivos.

Avances	Anterior Izquierda	Anterior derecha	Posterior izquierda	Posterior derecha
1	Cuerpo avanzando	Cuerpo avanzando	**Pierna medio levantada**	Cuerpo avanzando
2	Cuerpo avanzando	Cuerpo avanzando	**Pierna totalmente levantada + moverla adelante**	Cuerpo avanzando
3	Cuerpo avanzando	Cuerpo avanzando	**Totalmente avanzada**	Cuerpo avanzando
4	Cuerpo avanzando	Cuerpo avanzando	**Pierna bajada**	Cuerpo avanzando
5	Cuerpo avanzando	Cuerpo avanzando	**Cuerpo avanzando**	Cuerpo avanzando
6	**Pierna medio levantada**	Cuerpo avanzando	Cuerpo avanzando	Cuerpo avanzando
7	**Pierna totalmente levantada + moverla adelante**	Cuerpo avanzando	Cuerpo avanzando	Cuerpo avanzando
8	**Totalmente avanzada**	Cuerpo avanzando	Cuerpo avanzando	Cuerpo avanzando
9	**Pierna bajada**	Cuerpo avanzando	Cuerpo avanzando	Cuerpo avanzando
10	**Cuerpo avanzando**	Cuerpo avanzando	Cuerpo avanzando	Cuerpo avanzando
11	Cuerpo avanzando	Cuerpo avanzando	Cuerpo avanzando	**Pierna medio levantada**
12	Cuerpo avanzando	Cuerpo avanzando	Cuerpo avanzando	**Pierna totalmente levantada + moverla adelante**
13	Cuerpo avanzando	Cuerpo avanzando	Cuerpo avanzando	**Totalmente avanzada**
14	Cuerpo avanzando	Cuerpo avanzando	Cuerpo avanzando	**Pierna bajada**
15	Cuerpo avanzando	Cuerpo avanzando	Cuerpo avanzando	**Cuerpo avanzando**
16	Cuerpo avanzando	**Pierna medio levantada**	Cuerpo avanzando	Cuerpo avanzando
17	Cuerpo avanzando	**Pierna totalmente levantada + moverla adelante**	Cuerpo avanzando	Cuerpo avanzando
18	Cuerpo avanzando	**Totalmente avanzada**	Cuerpo avanzando	Cuerpo avanzando
19	Cuerpo avanzando	**Pierna bajada**	Cuerpo avanzando	Cuerpo avanzando
20	Cuerpo avanzando	**Cuerpo avanzando**	Cuerpo avanzando	Cuerpo avanzando

Tabla 18: Movimientos consecutivos de la marcha del robot de 3GDL

13.4.2 Rotación

Para la rotación ‘dos a dos’ del cuadrúpedo 3GDL se utiliza la marcha de trote. Este paso es muy sencillo Aquí se girarían dos piernas cruzadas a la vez en la misma dirección de giro, una vez hayan girado las dos en la misma dirección y el mismo Angulo de giro, se procederá a girar las otras dos patas que quedan que rotan junto con el cuerpo. Es decir, por ejemplo, girar las piernas anterior derecha y posterior izquierda a la vez. Una vez giradas y puestas en el suelo, se procederá a girar las otras dos piernas restantes que rotaran junto con el cuerpo.

Los movimientos se pueden ver en la imagen de abajo. Las piernas azules son las piernas levantadas en ese momento, las rojas permanecen en el suelo.

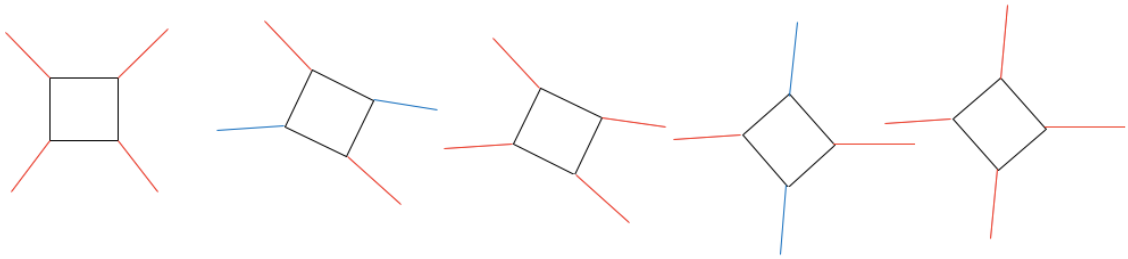


Figura 140: Marcha del trote – rotación 'dos a dos' – 3GDL

Tenga en cuenta que el paso de se usa tanto para avanzar como también para ir hacia los lados, este paso es completamente análogo a la rotación, sólo que en lugar de girar las piernas se moverán hacia adelante o hacia los lados.

14 Programa Keil µVsisón 5

Antes de que se pudiera empezar a programar, primero se tuvo repasar cómo programar en C, ya que en la asignatura de Sistemas Embebidos ya se enseñó a programar en C y concretamente en este programa llamado µVision5de Keil. Se utiliza este programa para añadir código o editar programación predeterminada que viene dada de la configuración que hacemos previamente con el STM32CubeMX. Este programa de STM crear todo el código fuente o digamos mínimo en C según la configuración dada por nosotros previamente, con los plug-ins y las librerías necesarias para programar los microcontroladores STM32 . Algunas características útiles de µVisión son el depurador que utilizamos para investigar variables dentro de nuestro programa. Se utiliza STM32 ST-LINK Utility para cargar el programa en el microcontrolador. Antes de poder cargar el programa primero se tiene que compilar, construir este programa en un archivo binario usando la función de compilación.

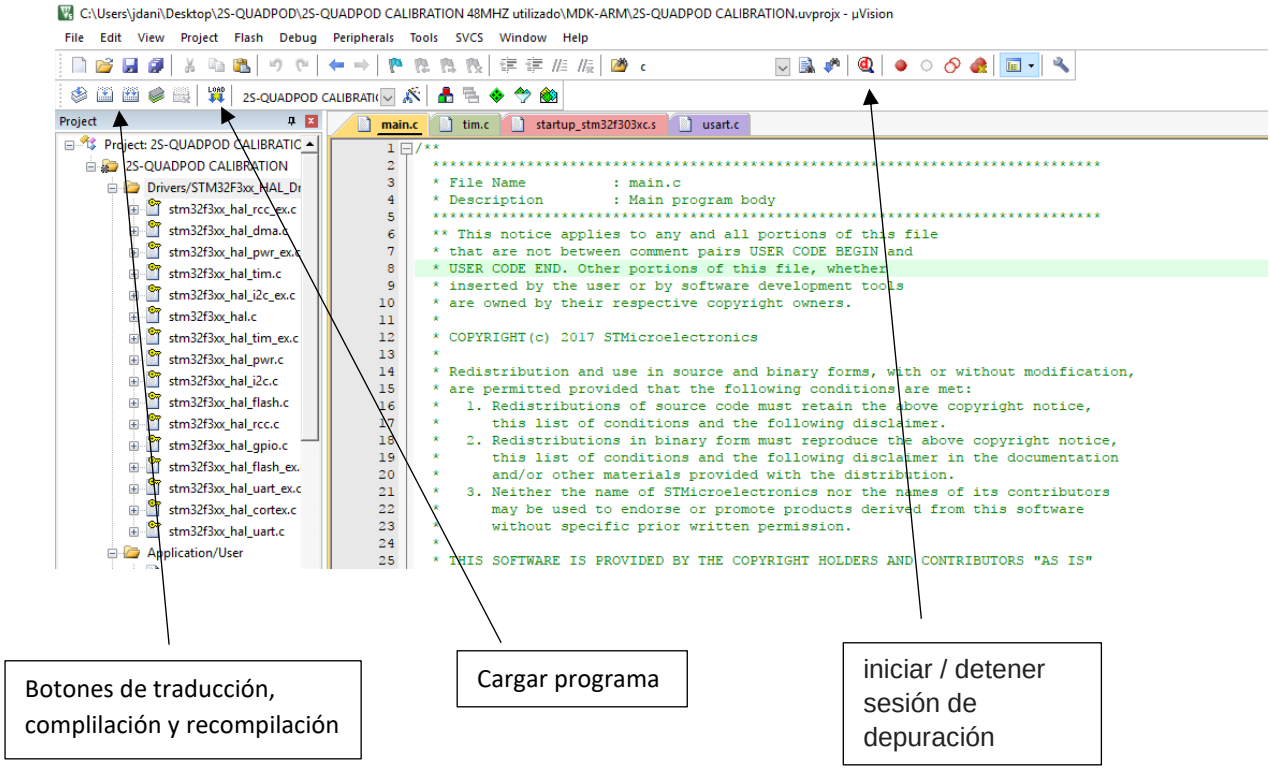


Figura 141: µVision5 (Keil) para C/C++

14.1 Explicación del programa para el cuadrúpedo de 2GDL

14.1.1 Código del Robot

Tenga en cuenta que el autor de este proyecto es ingeniero electromecánico, el programa no siempre está escrito de la manera más sencilla o más directa posible. Los programadores profesionales podrían llamar a algunas partes de este código como código sucio escrito. De

ninguna manera esta implementación es la mejor y también habrá muchas otras formas y opciones de programarlo.

```
/**

******************************************************************************
  * File Name          : main.c
  * Description        : Main program body
  * Autor              : Juan Daniel Miret Rubio

******************************************************************************
  ** This notice applies to any and all portions of this file
  * that are not between comment pairs USER CODE BEGIN and
  * USER CODE END. Other portions of this file, whether
  * inserted by the user or by software development tools
  * are owned by their respective copyright owners.
  *
  * COPYRIGHT(c) 2017 STMicroelectronics
  *
  * Redistribution and use in source and binary forms, with or without
modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above
copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above
copyright notice,
  *      this list of conditions and the following disclaimer in the
documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its
contributors
  *      may be used to endorse or promote products derived from this
software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
  *
```

```
**********************************************************************
********
  */
/* Includes ------------------------------------------------------------
--------*/
#include "main.h"
#include "stm32f3xx_hal.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"
```

Se utilizaron la biblioteca estándar, la biblioteca de matemáticas y la biblioteca de cadenas.

```
/* USER CODE BEGIN Includes */
#include <stdlib.h>
#include <math.h>
#include <string.h>
/* USER CODE END Includes */
//initialization of Uart 1 and 3
UART_HandleTypeDef UartHandle;
__IO ITStatus Uart1Ready = RESET;
__IO ITStatus Uart3Ready = RESET;
//UART_HandleTypeDef huart1;
//UART_HandleTypeDef huart3;
GPIO_InitTypeDef GPIO_InitStruct;
/* Private variables ---------------------------------------------------
--------*/

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------
--------*/
```

Se dio a cada pierna un número para identificarlas, se utilizó una definición ya que estas variables no cambiarian durante el funcionamiento del programa.

```
#define DERECHA_ANTERIOR 0
#define DERECHA_POSTERIOR 1
#define IZQUIERDA_ANTERIOR 2
#define IZQUIERDA_POSTERIOR 3
```

En esta sección, se han definido todas las longitudes de las diferentes partes de la pierna como se explica en la cinemática.

```
//longitud de la coxa (parte primera de la pierna) en mm
#define L_COXA 45
//longitud del femur ( parte segunda de la pierna) en mm
#define L_FEMUR 102
//Longitud del cuerpo en la dirección X dividida entre 2
#define X_CUERPO 43
//Longitud del cuerpo en la dirección Y dividida entre 2
#define Y_CUERPO 42
```

A partir de aquí se declaran las variables que se usarán, que cambiarán mientras el programa se está ejecutando. En los comentarios se puede encontrar cuáles son las funciones de estas variables.

```
//variables para hacer que el cuerpo avance / retroceda o izquierda /
derecha
int poscuerpoX = 0;
int poscuerpoY = 0;
//variable para ir adelante
int moviX = 0;
//variable para girar
double moviY = 0;
// Ángulo de rotación alrededor del eje Z llamado YAW o VIRAJE
double yaw = 0; // YAW o VIRAJE
//estructura autodefinida con valores x e y para dar una solicitud de
posición de pierna
typedef struct
{
      int x;
      int y;
} pedirPosicionPierna;

//Estructura autodefinida con ángulos de coxa y fémur
typedef struct
{
      double coxa;
      double femur;
} servoSalida;

//contendra las posiciones iniciales de cada pata
pedirPosicionPierna iniPos[4];
```

Utilizamos algunas funciones matemáticas auto-implementadas aquí ya que algunas funciones de la libreria de matemáticas MATH no fueron capaces de dar los resultados que se querían obtener. Por ejemplo, las funciones para el cálculo de la raíz cuadrada y la del arcoseno.

```
//raiz cuadrada
double raizCuadrada(int x)
{
      double firstGuess = x / 2;
      for (int i = 0; i < 10; i++)
      {
            firstGuess = (firstGuess + x / firstGuess) / 2;
      }
      return firstGuess;
}
//Arcoseno
double arcsin(double a)
{
      double z;
      z = a + (a * a * a) / 6 + (a * a * a * a * a * 3) / 40
                  + (a * a * a * a * a * a * a * 15) / 336;
      return z;
}
/* USER CODE END PV */

/* Private function prototypes ---------------------------------------
--------*/
void SystemClock_Config(void);
//void Error_Handler(void);
/* USER CODE BEGIN PFP */
/* Private function prototypes ---------------------------------------
--------*/
```

Puede encontrar las funciones usadas debajo. Estos son los prototipos de las funciones.

```
void posicionInicial(void);
pedirPosicionPierna CINV_cuerpo(int X, int Y, int Xcuerp, int Ycuerp);
servoSalida CINV_pierna(int X, int Y);
void derechaAnterior(pedirPosicionPierna paso);
void derechaPosterior(pedirPosicionPierna paso);
void izquierdaAnteior(pedirPosicionPierna paso);
void izquierdaPosterior(pedirPosicionPierna paso);
void caminarAdelante(void);
void caminarAtras(void);
void volverAlInicio(void);
void rotacion(void);
void Mensa_debug_Usart3(char text[32]);
void recibirDato(char text[5]);
void TransformDato(char[5]);
void transmision(void);
/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------
----------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM1_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  MX_TIM4_Init();
  MX_TIM15_Init();
  MX_USART1_UART_Init();
  MX_USART3_UART_Init();

  /* USER CODE BEGIN 2 */
```

Funciones de inicialización de los Timers con las señales PWM para la modulación de ancho de pulso, Inicializamos asi cada temporizador. Se han configurado 8 temporizadores ya que el robot está constituido por 8 servo-motores.

```
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_4);
  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);
  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);
  HAL_TIM_PWM_Start(&htim15, TIM_CHANNEL_1);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */

 /* USER CODE BEGIN 3 */
```

Aquí llamamos a la función que nos fijara los valores de la posición de inicio que debe tener el robot y comenzaremos la transmisión entre mando y robot.

```
            posicionInicial();
            transmision();
```

Aquí verificamos si los valores transformados del joystick son más altos que un cierto valor mínimo, si este es el caso empezamos a caminar, retroceder o rotar, dependiendo de hacia dónde movamos el cursor del joystick.

```
            if (moviX > 10)
            {
                  caminarAdelante();
            }
            if (moviX < -10)
                  {
                        caminarAtras();
```

Si dejamos de mover el joystick el cursor volverá a su posición de reposo y por tanto el robot volverá a su posición inicial.

```
                  }
            if ((10 > moviX) && (moviX > -10))
            {
                  volverAlInicio();
            }
            if (moviY > 0.5 || moviY < -0.5)
            {

                  rotacion();
            }

      //CALIBRACION DE SERVOMOTORES
//          double servoValor;
//          double femur;
//
//          femur = 3.14;
//
//          servoValor = -8.9127 * femur + 53;
//
```

```
//     __HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_4,(uint16_t)
servoValor); //(uint16_t) servoValor
//      HAL_Delay(6000);
//
//          __HAL_TIM_SetCompare(&htim15, TIM_CHANNEL_1, 50);
//          HAL_Delay(6000);

}
 /* USER CODE END 3 */

}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInit;

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL12;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) !=
HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  PeriphClkInit.PeriphClockSelection =
RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_USART3
                              |RCC_PERIPHCLK_TIM1;
  PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
  PeriphClkInit.Usart3ClockSelection = RCC_USART3CLKSOURCE_PCLK1;
  PeriphClkInit.Tim1ClockSelection = RCC_TIM1CLK_HCLK;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
```

```
    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* USER CODE BEGIN 4 */
```

Esto es, como se usa en el bucle while del main, el método que establece la posición inicial. Como puede ver, establecemos el valor x e y de todas las patas como 55 milímetros. Ya que es una buena distancia para que el robot quede en su posición de inicio preparado para cualquier movimiento.

```
//establece la posición inicial del cuadrúpedo
void posicionInicial(void)
{
//posición de la pierna delantera derecha
      iniPos[DERECHA_ANTERIOR].x = 55;
      iniPos[DERECHA_ANTERIOR].y = 55;

//posición de la pierna trasera derecha
      iniPos[DERECHA_POSTERIOR].x = -55;
      iniPos[DERECHA_POSTERIOR].y = 55;

// posición de la pierna delantera izquierda
      iniPos[IZQUIERDA_ANTERIOR].x = 55;
      iniPos[IZQUIERDA_ANTERIOR].y = -55;

// posición de la pierna trasera izquierda
      iniPos[IZQUIERDA_POSTERIOR].x = -55;
      iniPos[IZQUIERDA_POSTERIOR].y = -55;
}
```

En el método "CINV_cuerpo" se encuentra la implementación de la cinemática inversa del cuerpo según se calcula en el capítulo de la cinemática.

```
//cinemática inversa del cuerpo
pedirPosicionPierna CINV_cuerpo(int xPierna, int yPierna, int Xcuerp,
int Ycuerp)
{
      pedirPosicionPierna result;
      double cosA = cos(yaw);
      double sinA = sin(yaw);

      int cuerpoX = xPierna + Xcuerp;
      int cuerpoY = yPierna + Ycuerp;
      result.x = cuerpoX - cuerpoX * cosA + cuerpoY * sinA +
poscuerpoX;
      result.y = cuerpoY - cuerpoY * cosA + cuerpoX * sinA +
poscuerpoY;
      return result;
}
```

En el método "CINV_pierna" se encuentra la implementación de la cinemática inversa de la pierna según se calcula en el capítulo de cinemática.

Obsérvese cómo usamos las sentencias 'if' para asegurarnos de que los ángulos tienen los valores correctos, es decir, que cada ángulo corresponda al cuadrante que le corresponde a la pierna la cual hace referencia. ¡Como se dijo en el capítulo de cinemática es muy importante mantenerse consecuente con los cuadrantes!

```
// cinemática inversa de la pierna
servoSalida CINV_pierna(int X, int Y)
{
	servoSalida result;
	double longPierna;
	//char Buffer[5];

	if (X <= 0 && Y <= 0)
	{
		result.coxa = atan2(Y, X) + 6.28 ;
	} else
	{
		if (X >= 0 && Y <= 0)
		{
			result.coxa = atan2(Y, X) + 6.28;
		} else
		{
			result.coxa = atan2(Y, X);
		}
	}
    //Mensa_debug_Usart3(" X = ");
		//sprintf(Buffer, "%.4u \n", X);

	longPierna = raizCuadrada(X * X + Y * Y) - L_COXA;

	result.femur = arcsin(longPierna / L_FEMUR);
	return result;
}
```

En los 4 métodos siguientes se conbina la cinemática inversa del cuerpo (CINV_cuerpo) y la cinemática inversa de la pierna (CINV_pierna). Usamos estos para hacer que cada pierna se mueva. Utilizamos el __HAL_TIM_SetCompare (...) para dirigir los servomotores con el valor servo obtenido de la ecuación calibrada que transforma la salida CINV_pierna en un 'servoValor'. Estos métodos obtienen la variable 'paso' que se utilizara posteriormente para hacer que el robot caminar adelante, retroceda y gire.

```
//métodos para hacer que las piernas se muevan
void derechaAnterior(pedirPosicionPierna paso)
{
	pedirPosicionPierna result;
	servoSalida servo;
	double servoValor1;
	double servoValor2;
	result = CINV_cuerpo(iniPos[DERECHA_ANTERIOR].x,
iniPos[DERECHA_ANTERIOR].y, X_CUERPO,
			Y_CUERPO);
	servo = CINV_pierna(iniPos[DERECHA_ANTERIOR].x - result.x +
paso.x,
			iniPos[DERECHA_ANTERIOR].y - result.y + paso.y);
	servoValor1 = 8.91267 * servo.femur + 16;
	servoValor2 = -8.276 * servo.coxa + 36;
```

```
      __HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_1,(uint16_t)
servoValor1);
  __HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_2,(uint16_t) servoValor2);
}
void derechaPosterior(pedirPosicionPierna paso)
{
      pedirPosicionPierna result;
      servoSalida servo;
      double servoValor1;
      double servoValor2;
      result = CINV_cuerpo(iniPos[DERECHA_POSTERIOR].x,
iniPos[DERECHA_POSTERIOR].y, -X_CUERPO, Y_CUERPO);
      servo = CINV_pierna(iniPos[DERECHA_POSTERIOR].x + result.x +
paso.x,
                  iniPos[DERECHA_POSTERIOR].y + paso.y);
      servoValor1 = 8.91267 * servo.femur + 16;
      servoValor2 = -8.91267 * servo.coxa + 53;
      __HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_3,(uint16_t)
servoValor1);
      __HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_4,(uint16_t)
servoValor2);
}
void izquierdaAnteior(pedirPosicionPierna paso)
{
      double servoValor1;
      double servoValor2;
      pedirPosicionPierna result;
      servoSalida servo;
      result = CINV_cuerpo(iniPos[IZQUIERDA_ANTERIOR].x,
iniPos[IZQUIERDA_ANTERIOR].y, X_CUERPO,
                  -Y_CUERPO);
      servo = CINV_pierna(iniPos[IZQUIERDA_ANTERIOR].x + result.x +
paso.x,
                  iniPos[IZQUIERDA_ANTERIOR].y + result.y + paso.y);
      servoValor1 = 8.91267 * servo.femur + 15;
      servoValor2 = -8.276 * servo.coxa + 75;
      __HAL_TIM_SetCompare(&htim1, TIM_CHANNEL_1,(uint16_t)
servoValor1);
      __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_4,(uint16_t)
servoValor2);
}
void izquierdaPosterior(pedirPosicionPierna paso)
{
      pedirPosicionPierna result;
      servoSalida servo;
      double servoValor1;
      double servoValor2;
      result = CINV_cuerpo(iniPos[IZQUIERDA_POSTERIOR].x,
iniPos[IZQUIERDA_POSTERIOR].y, -X_CUERPO, -Y_CUERPO);
      servo = CINV_pierna(iniPos[IZQUIERDA_POSTERIOR].x + result.x +
paso.x,
                  iniPos[IZQUIERDA_POSTERIOR].y + result.y + paso.y);
      servoValor1 = 8.91267 * servo.femur + 17;
      servoValor2 = -8.91267 * servo.coxa + 65;
      __HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_2,(uint16_t)
servoValor1);
      __HAL_TIM_SetCompare(&htim15, TIM_CHANNEL_1,(uint16_t)
servoValor2);
}
```

Como dice el nombre del siguiente procedimiento, haremos que el robot avance. Lo hacemos de la misma manera que se explica en el capítulo 13 (marcha). Primero tomamos la posición inicial y de allí hacia el final procedemos a realizar la secuencia de pasos. Utilizamos la transmisión a través de cada paso, esto funciona como un HAL_Delay, que es un retraso entre la dirección de dos motores. Esto es necesario ya que los microcontroladores procesan el código en unos pocos milisegundos, pero los motores necesitan más tiempo para moverse. Si olvidáramos un HAL_Delay, entonces el motor obtendría sus valores de forma rápida no llegarían a su posición mandada porque ya estarían mandándole otra distinta y acatando esta última,… por tanto pasaría a un valor antes de que el comando anterior haya terminado. Además, las transmisiones funcionan obviamente también como la transmisión real que da los datos del joystick.

```
//Caminar adelante
void caminarAdelante(void)
{
      pedirPosicionPierna paso;

//PASO 1 inicialización
      paso.x = 0;
      paso.y = 0;
      izquierdaAnteior(paso);
  izquierdaPosterior(paso);
      paso.x = moviX;
      paso.y = 0;
      derechaPosterior(paso);
      paso.x = -moviX;
  paso.y = moviX / 3;
      derechaAnterior(paso);
      transmision();

//PASO 2
      paso.x = moviX / 2;
      paso.y = 0;
      derechaAnterior(paso);
      transmision();

//PASO 3
      paso.x = 0;
      paso.y = 0;
      derechaAnterior(paso);
      derechaPosterior(paso);
      paso.x = -moviX;
      paso.y = 0;
      izquierdaAnteior(paso);
      paso.x = -moviX / 2;
      paso.y = 0;
      izquierdaPosterior(paso);
      transmision();

//PASO 4
      paso.x = moviX;
      paso.y = -moviX / 3;
      izquierdaPosterior(paso);
      transmision();

//PASO 5
      paso.x = moviX / 2;
      paso.y = 0;
      izquierdaAnteior(paso);
      transmision();
```

```
//ÚLTIMO PASO
  paso.x = 0;
      paso.y = 0;
      izquierdaAnteior(paso);
      izquierdaPosterior(paso);
      paso.x = -moviX;
      paso.y = 0;
      derechaAnterior(paso);
      paso.x = -moviX / 2;
      paso.y = 0;
      derechaPosterior(paso);
      //transmision();

}
```

Este método es completamente similar a caminarAdelante.

```
// Retroceder

void caminarAtras(void)
{
      pedirPosicionPierna paso;

//PASO 1 inicialización
      paso.x = 0;
      paso.y = 0;
      derechaAnterior(paso);
      derechaPosterior(paso);
      paso.x = moviX;
      paso.y = 0;
      izquierdaAnteior(paso);
      paso.x = -moviX;
      paso.y = 0;
      izquierdaPosterior(paso);
      transmision();
//PASO 2
      paso.x = -moviX / 2;
      paso.y = 0;
      izquierdaPosterior(paso);
      transmision();
//PASO 3
      paso.x = 0;
      paso.y = 0;
      izquierdaAnteior(paso);
      izquierdaPosterior(paso);
      paso.x = -moviX / 2;
      paso.y = 0;
      derechaAnterior(paso);
      paso.x = -moviX;
      paso.y = 0;
      derechaPosterior(paso);
      transmision();
//PASO 4
      paso.x = +moviX;
      paso.y = 0;
      derechaAnterior(paso);
      transmision();
//PASO 4
      paso.x = +moviX / 2;
      paso.y = 0;
```

```
	derechaPosterior(paso);
// ÚLTIMO PASO
	paso.x = 0;
	paso.y = 0;
	transmision();
	derechaAnterior(paso);
	derechaPosterior(paso);
	paso.x = -moviX;
	paso.y = 0;
	izquierdaPosterior(paso);
	paso.x = -moviX / 2;
	paso.y = 0;
	izquierdaAnteior(paso);

}
```

En este procedimiento ponemos el paso.x y paso.y igual a cero, por lo que el robot tomaría su posición inicial.

```
// Regresar al comienzo

void volverAlInicio(void)
{
	pedirPosicionPierna paso;

	paso.x = 0;
	paso.y = 0;

	derechaAnterior(paso);
	derechaPosterior(paso);
	izquierdaAnteior(paso);
	izquierdaPosterior(paso);
}
```

El método para la rotación es el explicado anteriormente en el apartado 13 rotación progresiva.

```
// ROTAR
void rotacion(void)
{
	double rotacion;
	uint16_t servoValor;
	//Rotacion delantera derecha coxa
	rotacion = -moviY + 0.7853;
	servoValor = -8.276 * rotacion + 36;
	__HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_2,(uint16_t)
servoValor);
	//Rotacion trasera derecha coxa
	rotacion = -moviY + 2.36;
	servoValor = -8.91267 * rotacion + 53;
	__HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_4,(uint16_t)
servoValor);
	//Rotacion delantera izquierda coxa
	rotacion = -moviY + 5.5;
	servoValor = -8.276 * rotacion + 75;
	__HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_4,(uint16_t)
servoValor);
	//Rotacion trasera izquierda  coxa
	rotacion = -moviY + 3.93;
	servoValor = -8.91267 * rotacion + 65;
```

```
      __HAL_TIM_SetCompare(&htim15, TIM_CHANNEL_1,(uint16_t)
servoValor);
      transmision();

      //volverAlInicio
      //delantera izquierda
      //HAL_Delay(2000);
      //HAL_Delay(80);
      rotacion = 0.663;
      servoValor = 8.91267 * rotacion + 15;
      __HAL_TIM_SetCompare(&htim1, TIM_CHANNEL_1,(uint16_t)
servoValor);
      HAL_Delay(80);
      rotacion = 5.5;
      servoValor = -8.276 * rotacion + 75;
      __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_4,(uint16_t)
servoValor);
      HAL_Delay(80);
      rotacion = 0.327;
      servoValor = 8.91267 * rotacion + 15;
      __HAL_TIM_SetCompare(&htim1, TIM_CHANNEL_1,(uint16_t)
servoValor);
      transmision();
      HAL_Delay(80);

      //trasera derecha
      rotacion = 0.663;
      servoValor = 8.91267 * rotacion + 16;
      __HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_3,(uint16_t)
servoValor);
      rotacion = 2.3;
      servoValor = - 8.91267 * rotacion + 53;
      __HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_4,(uint16_t)
servoValor);
      HAL_Delay(80);
      rotacion = 0.327;
      servoValor = 8.91267 * rotacion + 16;
      __HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_3,(uint16_t)
servoValor);
      transmision();
      HAL_Delay(80);

      //delantera derecha
      rotacion = 0.663;
      servoValor = 8.91267 * rotacion + 16;
      __HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_1,(uint16_t)
servoValor);
      rotacion = 0.785;
      servoValor = -8.276 * rotacion + 36;
      __HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_2,(uint16_t)
servoValor);
      HAL_Delay(80);
      rotacion = 0.327;
      servoValor = 8.91267 * rotacion + 16;
      __HAL_TIM_SetCompare(&htim4, TIM_CHANNEL_1,(uint16_t)
servoValor);
      transmision();
      HAL_Delay(80);

      //trasera izquierda
      rotacion = 0.663;
```

```
	servoValor = 8.91267 * rotacion + 17;
	__HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_2,(uint16_t)
servoValor);
	rotacion = 3.93;
	servoValor = -8.91267 * rotacion + 65;
	__HAL_TIM_SetCompare(&htim15, TIM_CHANNEL_1,(uint16_t)
servoValor);
	HAL_Delay(80);
	rotacion = 0.327;
	servoValor = 8.91267 * rotacion + 17;
	__HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_2,(uint16_t)
servoValor);
	//transmision();
	//HAL_Delay(80);
}
```

Esto se utiliza con un cable serial a USB para transmitir datos que luego se muestra en un HyperTerminal, programa Termite 3.3. Este código no tiene funciones reales dentro del robot y era sólo una herramienta para ayudar a codificar y ver las variables en el codigo. Asi antes de trastear y probar con el robot se podía ver si el resultado en pantalla es el esperado para que el robot se moviera correctamente.

```
//debug mediante Uart3
void Mensa_debug_Usart3(char text[32])
{
	if (HAL_UART_Transmit(&huart3, (uint8_t*) text, strlen(text),
1000)!= HAL_OK)
	{

	}
	HAL_Delay(50);
	return;
}
```

Aquí, como se ve en el comentario, recibimos los datos del joystick. Podemos ver en el HAL_UART_Receive_IT (...) que obtenemos 5 caracteres con cada revelación de datos. Ponemos esto en la variable tipo cadena char "texto".

```
//recibir el dato del joystick
void recibirDato(char text[5])
{
	HAL_UART_Receive_IT(&huart1, (uint8_t *) text, 5);

//##-5- Esperando al final de la transferencia
##################################
```

Cuando se termina la transmisión vamos a la HAL_UART_RxCpltCallback y establecemos el Uart1Ready. Mientras la transmisión no termine, nos quedamos en el bucle while.

```
	while (Uart1Ready != SET)
	{
		HAL_GPIO_WritePin(GPIOE, LD3_Pin, GPIO_PIN_SET);
		HAL_GPIO_WritePin(GPIOE, LD3_Pin, GPIO_PIN_RESET);

	}
```

```
	//Reset de la transmisión
	Uart1Ready = RESET;

}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)
{

	Uart1Ready = SET;

	//HAL_GPIO_WritePin(GPIOE, LD4_Pin, GPIO_PIN_RESET);

}
```

Los datos recibidos están en un char ‘buffer’, para poder utilizar estos datos queremos transferirlos a un int / double. Hacemos esto utilizando la función atoi en C.

```
//Transformar el dato del joystick en un dato utilizable mediante la
funcion atoi
void TransformDato(char text[5])
{
	char result1[4];
	char result2[4];
	if (text[0] == 'x')
	{
		for (int i = 1; i < 5; i++)
		{
			result1[i - 1] = text[i];
		}
		moviX = (atoi(result1)) * 150 / 4095 - 75;
		//char buffer[32];
		//sprintf(buffer,"x%.4u", poscuerpoY);
		//Mensa_debug_Usart3(buffer);
	}

	if (text[0] == 'y')
	{
		for (int i = 1; i < 5; i++)
		{
			result2[i - 1] = text[i];
		}
		moviY = (atoi(result2)) * 3.14 / 4095 - 1.571;
		//char buffer[5];
		//sprintf(buffer,"x%.4u", poscuerpoY);
		//Mensa_debug_Usart3(buffer);

	}
}
```

En la transmisión de la siguiente función se llamaran a las funciones DataReceive y TransformData juntas, de esta manera cada vez que se llama a esta función ‘transmision’ leeremos las variables X e Y dadas por el joystick.

```
void transmision(void)
{
	char buffer[5];
	recibirDato(buffer);
	TransformDato(buffer);
	recibirDato(buffer);
```

```
      TransformDato(buffer);
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void _Error_Handler(char * file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error
return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line
number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/
/************************ (C) COPYRIGHT Escrito por Juan Daniel Miret
Rubio ***/
/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

14.1.2 Código del controlador inalámbrico

```
/**

******************************************************************************
  * File Name          : main.c
  * Description        : Main program body
  * Autor              : Juan Daniel Miret Rubio

******************************************************************************
  ** This notice applies to any and all portions of this file
  * that are not between comment pairs USER CODE BEGIN and
  * USER CODE END. Other portions of this file, whether
  * inserted by the user or by software development tools
  * are owned by their respective copyright owners.
  *
  * COPYRIGHT(c) 2017 STMicroelectronics
  *
  * Redistribution and use in source and binary forms, with or without
modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above
copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above
copyright notice,
  *      this list of conditions and the following disclaimer in the
documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its
contributors
  *      may be used to endorse or promote products derived from this
software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
```

```
  *
********************************************************************************
  */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f3xx_hal.h"
#include "adc.h"
#include "usart.h"
#include "gpio.h"
#include "string.h"

/* USER CODE BEGIN Includes */
#include <stdlib.h>
/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
__IO ITStatus Uart1Ready = RESET;
__IO ITStatus Uart3Ready = RESET;
//UART_HandleTypeDef huart1;
//UART_HandleTypeDef huart3;
GPIO_InitTypeDef GPIO_InitStruct;
/* Private variables ---------------------------------------------------------*/

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
//void Error_Handler(void);
void enviarDato(char data[5]);
uint16_t dato_x(void);
uint16_t dato_y(void);
void Mensa_debug_Usart3(char text[32]);
/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/*void TransformData(char text[5]){
 char help1[4];
 if(text[0]=='x'){
 for(int i=1;i<5;i++){
 help1[i-1]=text[i];}

 int a=(atoi(help1));
 char buffer[32];
 sprintf(buffer,"xwerkt%.4u", a);
 Mensa_debug_Usart3(buffer);
 }

 char help[4];
```

```
 if(text[0]=='y'){
 for(int i=1;i<5;i++)
 {
 help[i-1]=text[i];
 }
 uint32_t b=(atoi(help));
 char buffer[32];
 sprintf(buffer,"ywerkt%.4u", b);
 Mensa_debug_Usart3(buffer);


 }
 }*/
/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------
----------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_ADC2_Init();
  MX_USART1_UART_Init();
  MX_USART3_UART_Init();

  /* USER CODE BEGIN 2 */
      Mensa_debug_Usart3("sistema inicializado");
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
```

Lectura de los datos del joystick.

```
      char xBuffer[5];
      char yBuffer[5];

  while (1)
  {
```

```
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    uint16_t a = dato_x();
    uint16_t b = dato_y();
```

Código para enviar los datos. La transmisión se realiza a través de la interrupción, por lo tanto, después de que los bits se transmiten vamos automáticamente a la función de devolución de llamada como se ve debajo. Modificar los datos (int) en caracteres (char) y enviarlos.

```
            Mensa_debug_Usart3("");
            sprintf(xBuffer, "x%.4u \n",a);
            enviarDato(xBuffer);
        Mensa_debug_Usart3(xBuffer);
            Mensa_debug_Usart3("");
            sprintf(yBuffer, "y%.4u \n", b);
            enviarDato(yBuffer);
            Mensa_debug_Usart3(yBuffer);
            HAL_Delay(800);


  }
  /* USER CODE END 3 */

}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInit;

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL12;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
```

```
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) !=
HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  PeriphClkInit.PeriphClockSelection =
RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_USART3
                              |RCC_PERIPHCLK_ADC12;
  PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
  PeriphClkInit.Usart3ClockSelection = RCC_USART3CLKSOURCE_PCLK1;
  PeriphClkInit.Adc12ClockSelection = RCC_ADC12PLLCLK_DIV1;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* USER CODE BEGIN 4 */
```

Leyendo los datos X e Y con el ADC.

```
uint16_t dato_x(void) {
      HAL_ADC_Start(&hadc1);
      if (HAL_ADC_PollForConversion(&hadc1, 10) != HAL_OK) {
      }
      uint16_t b = HAL_ADC_GetValue(&hadc1);


      HAL_Delay(50);
      HAL_ADC_Stop(&hadc1);
      return b;
}

uint16_t dato_y(void) {
      HAL_ADC_Start(&hadc2);
      if (HAL_ADC_PollForConversion(&hadc2, 10) != HAL_OK) {
      }
      uint16_t b = HAL_ADC_GetValue(&hadc2);

      HAL_Delay(50);
      HAL_ADC_Stop(&hadc2);
      return b;
}
```

Método, como se ha visto antes, utilizado para enviar datos al HyperTerminal a través del cable serial a USB (utilizando la UART3).

```
void Mensa_debug_Usart3(char text[32]) {
```

```
	if (HAL_UART_Transmit(&huart3, (uint8_t*) text, strlen(text),
1000)
			!= HAL_OK) {

	}
	HAL_Delay(50);
	return;
}

/*void enviarDato(char text[5])
 {
 if(HAL_UART_Transmit(&huart1, (uint8_t*) text, strlen(text),1000)!=
HAL_OK)
 {

 }


 return;
 }*/
void enviarDato(char data[5]) {
	HAL_UART_Transmit_IT(&huart1, (uint8_t *) data, 5);

//##-5- Esperando al final de la transferencia
###################################
	while (Uart1Ready != SET) {

		//HAL_Delay(200);
	}

	//Reset transmision
	Uart1Ready = RESET;

}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *UartHandle) {

	//transferencia completada
	Uart1Ready = SET;


}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void _Error_Handler(char * file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error
return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT
```

```
/**
   * @brief Reports the name of the source file and the source line
number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/

/************************ (C) COPYRIGHT Escrito por Juan Daniel Miret
Rubio ***/
/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

14.2 Explicación del programa para el cuadrúpedo 3GDL

14.2.1 Código del robot

```
/**

********************************************************************
********
  * File Name          : main.c
  * Description        : Main program body
  * Autor              : Juan Daniel Miret Rubio

********************************************************************
********
  ** This notice applies to any and all portions of this file
  * that are not between comment pairs USER CODE BEGIN and
  * USER CODE END. Other portions of this file, whether
```

```
  * inserted by the user or by software development tools
  * are owned by their respective copyright owners.
  *
  * COPYRIGHT(c) 2017 STMicroelectronics
  *
  * Redistribution and use in source and binary forms, with or without
modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above
copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above
copyright notice,
  *      this list of conditions and the following disclaimer in the
documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its
contributors
  *      may be used to endorse or promote products derived from this
software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
  *

******************************************************************************
  */
/* Includes ------------------------------------------------------------*/
#include "main.h"
#include "stm32f3xx_hal.h"
#include "adc.h"
#include "i2c.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* USER CODE BEGIN Includes */
#include <string.h>
#include "accelerometer.h"
#include "gyroscope.h"
```

```
#include <math.h>
#include <stdlib.h>
/* USER CODE END Includes */
char transform[32]; //matriz de caracteres utilizada para mostrar
valores a lo largo del programa
float gyroData[3];
char buffer[5];
char memory[5];
int16_t accelBuf[3];
float memX = 0;
float memY = 0;
float anguloX = 0;
float anguloY = 0;
// Posiciones Servo Sensor
#define sensor_centro 52                  // Posición media del sensor
#define sensor_izquierda 20         // Posición izquierda
#define sensor_derecha 92                 // Posición derecha
#define sensor_izquierda_45 36            // Posición izquierda 45°
#define sensor_derecha_45 72              // Posición derecha 45°

#define DERECHA_ANTERIOR 0
#define DERECHA_POSTERIOR 1
#define IZQUIERDA_ANTERIOR 2
#define IZQUIERDA_POSTERIOR 3
#define L_COXA 40
#define L_FEMUR 60
#define X_COXA 63.33
#define Y_COXA 63.33
#define b1 0.4727793374
#define ayudaLongTibia 144
uint8_t avancesAire;
uint8_t avancesSuelo;
uint8_t totalavances;
uint8_t piernaNum[4];
int8_t poscuerpx = 0;
int8_t poscuerpy = 0;
int8_t poscuerpz = 0;
int8_t moviX = 0;
int8_t moviY = 0;
int8_t ssx = 0;
int8_t ssy = 0;
double imuRotX = 0;
double imuRotY = 0;
uint8_t avance = 1;
char result[5];
uint8_t imuCounter1 = 0;
double pitch = 0; // Inclinación
uint8_t piernaInvertidaCINV = 0;
typedef struct {
      int x;
      int y;
      int z;
      double r;
} pedirPosicionPierna;

pedirPosicionPierna pasos[4];
typedef struct {
      double coxa;
      double femur;
      double tibia;
} servoSalida;
```

```
uint8_t tiempIntegra;
double cuerpRotX = 0;
double cuerpRotY = 0;
double cuerpRotZ = 0;
uint8_t contraccion = 0;
uint8_t rota = 0;
pedirPosicionPierna iniPos[4];
float ranguloX, ranguloY;
/* Private variables -----------------------------------------------
--------*/

/* USER CODE BEGIN PV */
/* Private variables -----------------------------------------------
--------*/
//__IO ITStatus Uart2Ready = RESET;
//__IO ITStatus Uart3Ready = RESET;
/* USER CODE END PV */

/* Private function prototypes -------------------------------------
--------*/
void SystemClock_Config(void);
//void Error_Handler(void);
/* USER CODE BEGIN PFP */
/* Private function prototypes -------------------------------------
--------*/
double raizCuadrada(int x);
double arcsin(double a);
double arccos(double a);
void posicionInicial(void);
pedirPosicionPierna CINVcuerpo(int X, int Y, int Z, int Xcuerp, int
Ycuerp, double rotZ);
servoSalida CINVpierna(int X, int Y, int Z);
void derechaAnterior(pedirPosicionPierna paso);
void Mensa_debug_Usart3(char text[32]);
void derechaPosterior(pedirPosicionPierna paso);
void izquierdaAnterior(pedirPosicionPierna paso);
void izquierdaPosterior(pedirPosicionPierna paso);
int TransformDato(char[5]);
void volverAlInicio(void);
void rotacion(void);
pedirPosicionPierna solicitoPosicion(int x, int y, int z, double r);


uint16_t analog_voltage(uint16_t valor);
uint16_t Read_Analog_Value(void);
unsigned int medir (void);

void mover_servo_medir(uint8_t m_s_sensor);


//Implementación de la marcha progresiva o gateo (Aunque finalmente se
usa la marcha de trote)

/*void marchaGateoX(void);
 void pasoX(void);
 void marchaGateoY(void);
 void pasoY(void);*/

void selecPierna(int selec, pedirPosicionPierna paso);
void elegirOpcion(void);
```

```
void contraccionPos(void);
void LeoIMU(void);
void trotX(void);
void trotY(void);
/* USER CODE END PFP */

/* USER CODE BEGIN 0 */
unsigned int valor_ADC = 0;              // Variable donde se almacena
las sucesivas lecturas del sensor infrarrojo
unsigned short int sensor_mV1;      // Variable donde se almacena el
valor medio de las lecturas del sensor convertido en mV
int i;

unsigned int umbral = 1500;   // Umbral Distancia mínima a la que
empieza a detectar el sensor (en mV)
/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------
----------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_ADC2_Init();
  MX_I2C1_Init();
  MX_TIM1_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  MX_TIM15_Init();
  MX_TIM16_Init();
  MX_TIM17_Init();
  MX_USART2_UART_Init();
  MX_USART3_UART_Init();

  /* USER CODE BEGIN 2 */
  BSP_ACCELERO_Init();
  BSP_GYRO_Init();

      HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
```

```
        HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
        HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
        HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2);
        HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);
        HAL_TIM_PWM_Start(&htim15, TIM_CHANNEL_1);
        HAL_TIM_PWM_Start(&htim15, TIM_CHANNEL_2);
        HAL_TIM_PWM_Start(&htim16, TIM_CHANNEL_1);
        HAL_TIM_PWM_Start(&htim17, TIM_CHANNEL_1);
        HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
        HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);
        HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);
        HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3);
        HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
        HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
        HAL_Delay(3000);
        HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);

        posicionInicial();
        volverAlInicio();
        HAL_Delay(3000);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

//    Calibración de Motores
/*  __HAL_TIM_SetCompare(&htim16, TIM_CHANNEL_1, 60);
          HAL_Delay(3000); */
//  Fin de calibración


HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);
```

Utilizamos comandos if para enviarle las ordenes al medidor de distancia en cada caso.

```
      //sensor distancia
            if ( ssx > 20 &&  (10 > ssy) && (ssy > -10))
            {
                  mover_servo_medir(sensor_derecha);
              HAL_Delay(100);
                  HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);
            }
    if ( ssx > 20 &&  ssy > 20)
            {
                  mover_servo_medir(sensor_derecha_45);
              HAL_Delay(100);
                  HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);
            }
            if ( (10 > ssx) && (ssx > -10) &&  ssy > 20)
            {
                  mover_servo_medir(sensor_centro);
              HAL_Delay(100);
                  HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);
            }
            if ( ssx < -20 &&  ssy > 20)
```

```
            {
                  mover_servo_medir(sensor_izquierda_45);
              HAL_Delay(100);
                  HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);
            }

            if ( ssx < -20 &&  (10 > ssy) && (ssy > -10))
            {
                  mover_servo_medir(sensor_izquierda);
              HAL_Delay(100);
                  HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);
            }
```

Utilizamos comandos if para decidir qué posición o función de marcha o paro se le da al robot dependiendo del mando y en algunos casos tambien del medidor de distancia.

```
      //ROBOT
            if (moviX < -20 || (moviX > 20  )) //&& medir() < umbral
            {
                  trotX();
                  HAL_Delay(100);
                  HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);
            }

            if ((moviY > 20) || (moviY < -20))
            {
                  trotY();
                  HAL_Delay(100);
                  HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);
            }

            if ((cuerpRotZ > 0.10) || (cuerpRotZ < -0.10))
            {
                  rotacion();
                  tiempIntegra = 100;
                  HAL_Delay(100);
                  HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);
            }
            if (contraccion == 1)
            {
                  contraccionPos();
            }
            if (((10 > moviX) && (moviX > -10) && (10 > moviY) &&
(moviY > -10)&& (cuerpRotZ < 0.175) && (cuerpRotZ > -0.175)&&
(contraccion == 0))||(medir() > umbral))
            {
                  volverAlInicio();
                  tiempIntegra = 30;
                  LeoIMU();
                  HAL_Delay(30);
                  HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);
            }



    }
  /* USER CODE END 3 */

}
```

```
/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInit;

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL16;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) !=
HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  PeriphClkInit.PeriphClockSelection =
RCC_PERIPHCLK_I2C1|RCC_PERIPHCLK_TIM1
                              |RCC_PERIPHCLK_ADC12;
  PeriphClkInit.Adc12ClockSelection = RCC_ADC12PLLCLK_DIV1;
  PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_HSI;
  PeriphClkInit.Tim1ClockSelection = RCC_TIM1CLK_HCLK;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}
```

```
/* USER CODE BEGIN 4 */

// Movimiento servo Sensor(PA6)
void mover_servo_medir(uint8_t m_s_sensor){          // Servo sensor
medida
      __HAL_TIM_SetCompare(&htim16,TIM_CHANNEL_1,m_s_sensor);    //
Actualizar el valor del PWM del servo del Sensor/update pwm value
      HAL_Delay(1000);
}
```

Funciónes las cuales 'Read_Analog_value' mide el valor analogico que da el sensor y mediante un convertidor analógico digital ADC, la funció 'medir' transforma dicho valor en una lectura de voltaje en miliVoltios.

```
unsigned int medir (){
      valor_ADC=0;
      for(unsigned char i=0;i<32;i++){
            valor_ADC+=Read_Analog_Value();
            HAL_Delay(1);
      }
      valor_ADC/=32;

      sensor_mV1=valor_ADC*3300/4095;     // Transformar lectura a mV
      HAL_Delay(1);
      return sensor_mV1;
}
uint16_t Read_Analog_Value(void)
{
            uint16_t valor_analog;
            HAL_ADC_Start(&hadc2);
            if (HAL_ADC_PollForConversion(&hadc2, 10)!=HAL_OK)
            {
            }
            valor_analog = HAL_ADC_GetValue(&hadc2);
            //HAL_Delay(1);
            HAL_ADC_Stop(&hadc2);
            return valor_analog;
}
```

Algunas funciones matemáticas auto-implementadas, las funciones de la biblioteca Math no compilaban.

```
double raizCuadrada(int x)
{
      double firstGuess = x / 2;
      for (int i = 0; i < 10; i++)
      {
            firstGuess = (firstGuess + x / firstGuess) / 2;
      }
      return firstGuess;
}

double arcsin(double a)
{
      double z;
      if (a < -1)
      {
            a = -1;
```

```
	}
	if (a > 1)
	{
		a = 1;
	}
	z = a + (a * a * a) / 6 + (a * a * a * a * a * 3) / 40
		+ (a * a * a * a * a * a * a * 15) / 336;
	return z;
}

double arccos(double a)
{
	double z = 1.57 - arcsin(a);
	return z;
}

void posicionInicial(void)
{
//posición de la pierna anterior derecha
	iniPos[DERECHA_ANTERIOR].x = 90;
	iniPos[DERECHA_ANTERIOR].y = 90;
	iniPos[DERECHA_ANTERIOR].z = 120;
//posición de la pierna trasera derecha
	iniPos[DERECHA_POSTERIOR].x = -90;
	iniPos[DERECHA_POSTERIOR].y = 90;
	iniPos[DERECHA_POSTERIOR].z = 130;
// posición de la pierna anterior izquierda
	iniPos[IZQUIERDA_ANTERIOR].x = 90;
	iniPos[IZQUIERDA_ANTERIOR].y = -90;
	iniPos[IZQUIERDA_ANTERIOR].z = 120;
// posición de la pierna trasera izquierda
	iniPos[IZQUIERDA_POSTERIOR].x = -90;
	iniPos[IZQUIERDA_POSTERIOR].y = -90;
	iniPos[IZQUIERDA_POSTERIOR].z = 130;

}
```

La cinemática inversa del cuerpo como se explica en el capítulo de cinemática inversa.

```
pedirPosicionPierna CINVcuerpo(int X, int Y, int Z, int Xcuerp, int
Ycuerp, double rotZ)
{
	pedirPosicionPierna result;
	double cosA = cos(cuerpRotX + imuRotX);
	double sinA = sin(cuerpRotX + imuRotX);
	double cosB = cos(cuerpRotY + imuRotY);
	double sinB = sin(cuerpRotY + imuRotY);
	double cosC = cos(rotZ + pitch);
	double sinC = sin(rotZ + pitch);
	int totalX = X + Xcuerp + poscuerpx;
	int totalY = Y + Ycuerp + poscuerpy;
	result.x = totalX - totalX * cosB * cosC - totalY * sinA * sinB
* cosC
		- Z * cosA * cosC * sinB + totalY * cosA * sinC - Z *
sinC * sinA
		+ poscuerpx;
	result.y = totalY - totalX * cosB * sinC - totalY * sinA * sinB
* sinC
		- Z * cosA * sinB * sinC - totalY * cosA * cosC + Z *
sinA * cosC
		+ poscuerpy;
```

```
	result.z = Z + totalX * sinB - totalY * sinA * cosB - Z * cosA *
cosB
			+ poscuerpz;

	return result;

}
```

La cinemática inversa de la pierna como se explica en el capítulo de cinemática inversa.

```
servoSalida CINVpierna(int X, int Y, int Z)
{

	servoSalida result;
	double longPierna;
	double imaL;
	double a1;
	double a2;
	double b2;
	double resultvalor1;
	double resultvalor2;
	if ((X <= 0 && Y <= 0) || (X >= 0 && Y <= 0))
	{
		result.coxa = atan2(Y, X) + 6.28;
	} else
	{
		result.coxa = atan2(Y, X);
	}
	longPierna = raizCuadrada(X * X + Y * Y) - L_COXA;
	imaL = raizCuadrada(longPierna * longPierna + Z * Z);
	a1 = arccos(Z / imaL);


	resultvalor1 = ayudaLongTibia * ayudaLongTibia + L_FEMUR *
L_FEMUR
			- imaL * imaL;
	resultvalor2 = 2 * ayudaLongTibia * imaL;

	b2 = arccos((resultvalor1) / (resultvalor2));
	if (piernaInvertidaCINV == 0)
	{
		result.tibia = b1 + b2;
	} else
	{
		if (piernaInvertidaCINV == 1)
		{
			result.tibia = b1 + 6.28 - b2;
		}
	}
	resultvalor1 = L_FEMUR * L_FEMUR
			+ imaL * imaL- ayudaLongTibia * ayudaLongTibia;
	resultvalor2 = 2 * imaL * L_FEMUR;
	a2 = arccos((resultvalor1) / (resultvalor2));
	if (piernaInvertidaCINV == 0)
	{
		result.femur = a1 + a2;
	} else
	{
		if (piernaInvertidaCINV == 1)
		{
			a1 = 3.14 - a1;
```

```
                if (longPierna < 0)
                {
                      a1 = 6.14 - a1;
                }
                result.femur = a1 - a2;
          }
      }

      return result;
}
```

Los 4 métodos para mover cada pierna por separado.Tambien utilizados en el robot 2GDL.

```
void derechaAnterior(pedirPosicionPierna paso)
{

      pedirPosicionPierna result;
      servoSalida servo;
      double servoValor;
      result = CINVcuerpo(iniPos[DERECHA_ANTERIOR].x + paso.x,
                  iniPos[DERECHA_ANTERIOR].y + paso.y,
                  iniPos[DERECHA_ANTERIOR].z + paso.z, X_COXA, Y_COXA,
paso.r);
      servo = CINVpierna(iniPos[DERECHA_ANTERIOR].x - result.x +
paso.x,
                  iniPos[DERECHA_ANTERIOR].y - result.y + paso.y,
                  iniPos[DERECHA_ANTERIOR].z - result.z + paso.z);

      servoValor = -17.8253 * servo.coxa + 73; // nueva formula + if y
ecuacion con  limites de 360° a 320°

      if (servo.coxa > 5.3)
      {
            servoValor = -15.27887458 * servo.coxa + 169;
      }
      if (servoValor >= 85)
      {
            servoValor = 85;
      }

      if (servoValor <= 33)
      {
            servoValor = 33;

      }
      __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, servoValor);
      servoValor = 15.91549431 * servo.femur + 37; //17.825353 *
servo.femur + 37; // nueva formula
      if (servoValor >= 96)
      {
            servoValor = 96;
      }

      if (servoValor <= 34)
      {
            servoValor = 34;

      }
      __HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_3, servoValor);
```

```
	servoValor = 17.18873385 * servo.tibia + 10;
	if (servoValor >= 94)
	{
		servoValor = 94;
	}

	if (servoValor <= 34)
	{
		servoValor = 34;

	}

	__HAL_TIM_SetCompare(&htim17, TIM_CHANNEL_1,servoValor);
}
void derechaPosterior(pedirPosicionPierna paso)
{

	pedirPosicionPierna result;
	servoSalida servo;
	double servoValor;
	result = CINVcuerpo(iniPos[DERECHA_POSTERIOR].x + paso.x,
iniPos[DERECHA_POSTERIOR].y + paso.y,iniPos[DERECHA_POSTERIOR].z +
paso.z, -X_COXA, Y_COXA, paso.r);
	servo = CINVpierna(iniPos[DERECHA_POSTERIOR].x - result.x +
paso.x,iniPos[DERECHA_POSTERIOR].y - result.y +
paso.y,iniPos[DERECHA_POSTERIOR].z - result.z + paso.z);
	servoValor = -17.72535366 * servo.coxa + 94;
	if (servoValor > 80)
	{
		servoValor = 80;
	}

	if (servoValor < 24)
	{
		servoValor = 24;

	}

	__HAL_TIM_SetCompare(&htim1, TIM_CHANNEL_3,servoValor);
	servoValor = 18.4619734 * servo.femur + 35;
//17.825353 * servo.femur + 36;		//18.4619734 * servo.femur +
35;
	if (servoValor > 96)
	{
		servoValor = 96;
	}

	if (servoValor < 32)
	{
		servoValor = 32;

	}

	__HAL_TIM_SetCompare(&htim15, TIM_CHANNEL_1,servoValor);//
tambien con T15 C1
	servoValor = 17.1887385 * servo.tibia + 12;
	if (servoValor > 96)
	{
		servoValor = 96;
	}
```

```
      if (servoValor < 36)
      {
            servoValor = 36;

      }

  __HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_4, servoValor);// tambien
con T2C4

}
void izquierdaAnterior(pedirPosicionPierna paso)
{

      double servoValor;
      pedirPosicionPierna result;
      servoSalida servo;
      result = CINVcuerpo(iniPos[IZQUIERDA_ANTERIOR].x + paso.x,
                  iniPos[IZQUIERDA_ANTERIOR].y + paso.y,
iniPos[IZQUIERDA_ANTERIOR].z + paso.z,X_COXA, -Y_COXA, paso.r);
      servo = CINVpierna(iniPos[IZQUIERDA_ANTERIOR].x - result.x +
paso.x,iniPos[IZQUIERDA_ANTERIOR].y - result.y +
paso.y,iniPos[IZQUIERDA_ANTERIOR].z - result.z + paso.z);
      servoValor = -16.55211408 * servo.coxa + 39;

      if (servo.coxa > 3.5)
      {
            servoValor = -17.82535363* servo.coxa + 149;
      }
      if (servoValor > 75)
      {
            servoValor = 75;
      }

      if (servoValor < 26)
      {
            servoValor = 26;

      }

      __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2,servoValor);
      servoValor = 16.552114 * servo.femur + 33;//17.825353 *
servo.femur + 35;
      if (servoValor > 94)
      {
            servoValor = 94;
      }

      if (servoValor < 32)
      {
            servoValor = 32;

      }

      __HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_2, servoValor);
      servoValor = 15.91549431 * servo.tibia + 16;
      if (servoValor > 94)
      {
            servoValor = 94;
      }

      if (servoValor < 38)
```

```
	{
		servoValor = 38;

	}
	__HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_1, servoValor);

}
void izquierdaPosterior(pedirPosicionPierna paso)
{

	pedirPosicionPierna result;
	servoSalida servo;
	double servoValor;
	result = CINVcuerpo(iniPos[IZQUIERDA_POSTERIOR].x + paso.x,
iniPos[IZQUIERDA_POSTERIOR].y + paso.y,iniPos[IZQUIERDA_POSTERIOR].z +
paso.z, -X_COXA, -Y_COXA, paso.r);
	servo = CINVpierna(iniPos[IZQUIERDA_POSTERIOR].x - result.x +
paso.x,iniPos[IZQUIERDA_POSTERIOR].y - result.y +
paso.y,iniPos[IZQUIERDA_POSTERIOR].z - result.z + paso.z);
	servoValor = -17.82535369 * servo.coxa + 127;
	if (servoValor >= 85)
	{
		servoValor = 85;
	}

	if (servoValor <= 32)
	{
		servoValor = 32;

	}
	__HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_3, servoValor);
	servoValor = 17.825353 * servo.femur + 36;
	if (servoValor >= 95)
	{
		servoValor = 95;
	}

	if (servoValor <= 33)
	{
		servoValor = 33;

	}
	__HAL_TIM_SetCompare(&htim1, TIM_CHANNEL_2, servoValor);
	servoValor = 15.91549431 * servo.tibia + 12;
	if (servoValor >= 90)
	{
		servoValor = 90;
	}

	if (servoValor <= 34)
	{
		servoValor = 34;

	}
	__HAL_TIM_SetCompare(&htim1, TIM_CHANNEL_1,servoValor);

}
void volverAlInicio(void)
{
	pedirPosicionPierna paso;
```

```
	paso.x = 0;
	paso.y = 0;
	paso.z = 0;
	paso.r = 0;
	izquierdaPosterior(paso);
	derechaPosterior(paso);
	izquierdaAnterior(paso);
	derechaAnterior(paso);
}
pedirPosicionPierna solicitoPosicion(int x, int y, int z, double r)
{
	pedirPosicionPierna result;
	result.x = x;
	result.y = y;
	result.z = z;
	result.r = r;
	return result;
}

void selecPierna(int selec, pedirPosicionPierna paso) {

	if (selec == 0)
	{
		derechaAnterior(paso);
	}
	if (selec == 1)
	{
		derechaPosterior(paso);
	}
	if (selec == 2)
	{
		izquierdaAnterior(paso);
	}
	if (selec == 3)
	{
		izquierdaPosterior(paso);
	}
}

void rotacion(void)
{
	//rotación trotando
	//levantar y girar 2 patas + girar las otras 2 patas en la misma
dirección
	derechaAnterior(solicitoPosicion(0, 0, -20, cuerpRotZ));
	izquierdaPosterior(solicitoPosicion(0, 0, -20, cuerpRotZ));
	izquierdaAnterior(solicitoPosicion(0, 0, 0, -cuerpRotZ));
	derechaPosterior(solicitoPosicion(0, 0, 0, -cuerpRotZ));
	HAL_Delay(100);
	//poniendo las piernas de nuevo al suelo
	derechaAnterior(solicitoPosicion(0, 0, 0, cuerpRotZ));
	izquierdaPosterior(solicitoPosicion(0, 0, 0, cuerpRotZ));
	HAL_Delay(100);
	//elevación y rotación de las otra 2 patas restantes + rotación
de las 2 primeras giradas
	izquierdaAnterior(solicitoPosicion(0, 0, -20, 0));
	derechaPosterior(solicitoPosicion(0, 0, -20, 0));
	izquierdaPosterior(solicitoPosicion(0, 0, 0, 0));
	derechaAnterior(solicitoPosicion(0, 0, 0, 0));
	//poniendo las piernas restantes de nuevo al suelo
	HAL_Delay(100);
```

```
      izquierdaAnterior(solicitoPosicion(0, 0, 0, 0));
      derechaPosterior(solicitoPosicion(0, 0, 0, 0));

}
void trotX(void)
{

      derechaAnterior(solicitoPosicion(moviX, 0, -50, 0));
      izquierdaPosterior(solicitoPosicion(moviX, 0, -50, 0));
      izquierdaAnterior(solicitoPosicion(-moviX, 0, 0, 0));
      derechaPosterior(solicitoPosicion(-moviX, 0, 0, 0));
      //LeoIMU();
      HAL_Delay(100);
      derechaAnterior(solicitoPosicion(moviX, 0, 0, 0));
      izquierdaPosterior(solicitoPosicion(moviX, 0, 0, 0));
      //LeoIMU();
      HAL_Delay(100);
      derechaAnterior(solicitoPosicion(0, 0, 0, 0));
      izquierdaPosterior(solicitoPosicion(0, 0, 0, 0));
      izquierdaAnterior(solicitoPosicion(0, 0, -50, 0));
      derechaPosterior(solicitoPosicion(0, 0, -50, 0));
      //LeoIMU();
      HAL_Delay(100);
      izquierdaAnterior(solicitoPosicion(0, 0, 0, 0));
      derechaPosterior(solicitoPosicion(0, 0, 0, 0));

}
void trotY(void)
{
      derechaAnterior(solicitoPosicion(0, moviY, -50, 0));
      izquierdaPosterior(solicitoPosicion(0, moviY, -50, 0));
      izquierdaAnterior(solicitoPosicion(0, -moviY, 0, 0));
      derechaPosterior(solicitoPosicion(0, -moviY, 0, 0));
//LeoIMU();
      HAL_Delay(100);
      derechaAnterior(solicitoPosicion(0, moviY, 0, 0));
      izquierdaPosterior(solicitoPosicion(0, moviY, 0, 0));
      //LeoIMU();
      HAL_Delay(100);
      derechaAnterior(solicitoPosicion(0, 0, 0, 0));
      izquierdaPosterior(solicitoPosicion(0, 0, 0, 0));
      izquierdaAnterior(solicitoPosicion(0, 0, -50, 0));
      derechaPosterior(solicitoPosicion(0, 0, -50, 0));
      //LeoIMU();
      HAL_Delay(100);
      izquierdaAnterior(solicitoPosicion(0, 0, 0, 0));
      derechaPosterior(solicitoPosicion(0, 0, 0, 0));
}
/*void marchaGateoX(void) {

 //comienzo del ciclo de avance para cada pierna (con referencia a la
primera etapa)
 if (moviX > 30) {

 piernaNum[DERECHA_ANTERIOR] = 16;
 piernaNum[DERECHA_POSTERIOR] = 11;
 piernaNum[IZQUIERDA_POSTERIOR] = 1;
 piernaNum[IZQUIERDA_ANTERIOR] = 6;

 } else {
 if ((moviX < -30)) {
```

```
 piernaNum[DERECHA_ANTERIOR] = 1;
 piernaNum[DERECHA_POSTERIOR] = 6;
 piernaNum[IZQUIERDA_POSTERIOR] = 11;
 piernaNum[IZQUIERDA_ANTERIOR] = 16;
 }
 }

 avancesAire = 5; // número de pasos que la pierna está en el aire
 avancesSuelo = 15; // número de pasos que la pierna está en el suelo
 totalavances = 20; // pasos totales
 }
 void marchaGateoY(void) {

 //comienzo del ciclo de avences para cada pierna (con referencia a la
primera etapa)
 if (moviY > 30) {

 piernaNum[DERECHA_ANTERIOR] = 11;
 piernaNum[DERECHA_POSTERIOR] = 1;
 piernaNum[IZQUIERDA_POSTERIOR] = 6;
 piernaNum[IZQUIERDA_ANTERIOR] = 16;


 } else {
 if ((moviY < -30)) {

 piernaNum[DERECHA_ANTERIOR] = 6;
 piernaNum[DERECHA_POSTERIOR] = 16;
 piernaNum[IZQUIERDA_POSTERIOR] = 11;
 piernaNum[IZQUIERDA_ANTERIOR] = 1;


 }
 }

 avancesAire = 5; // número de pasos que la pierna está en el aire
 avancesSuelo = 15; // número de pasos que la pierna está en el suelo
 totalavances = 20; // pasos totales
 }*/

/*void pasoX(void) {
 //Calculando las posiciones de marcha

 for (int pierna = 0; pierna < 4; pierna++) {

 if (avance == piernaNum[pierna]) {
 pasos[pierna].z = -60 / 2;
 selecPierna(pierna, pasos[pierna]);
 } else if (avance == piernaNum[pierna] + 1) {
 pasos[pierna].x = 0;
 pasos[pierna].z = -60;

 selecPierna(pierna, pasos[pierna]);
 } else if (avance == piernaNum[pierna] + 2) {
 pasos[pierna].x = moviX / 2;
 pasos[pierna].z = -60 / 2;
 selecPierna(pierna, pasos[pierna]);
 } else if (avance == piernaNum[pierna] + 3) {
 pasos[pierna].z = 0;
 selecPierna(pierna, pasos[pierna]);
 } else {
```

```
//moviendo el cuerpo hacia adelante
pasos[pierna].x -= moviX / avancesSuelo;
selecPierna(pierna, pasos[pierna]);

}

}

//Avanzar al siguiente paso
if (++avance > totalavances) {
avance = 1;

}
}
void pasoY(void) {
//Calculando las posiciones de marcha

for (int pierna = 0; pierna < 4; pierna++) {

if (avance == piernaNum[pierna]) {
pasos[pierna].z = -44 / 2;
selecPierna(pierna, pasos[pierna]);
} else if (avance == piernaNum[pierna] + 1) {
pasos[pierna].y = 0;
pasos[pierna].z = -44;

selecPierna(pierna, pasos[pierna]);
} else if (avance == piernaNum[pierna] + 2) {
pasos[pierna].y = moviY / 2;
pasos[pierna].z = -44 / 2;
selecPierna(pierna, pasos[pierna]);
} else if (avance == piernaNum[pierna] + 3) {
pasos[pierna].z = 0;
selecPierna(pierna, pasos[pierna]);
} else
{
// moviendo el cuerpo hacia adelante

pasos[pierna].y -= moviY / avancesSuelo;
selecPierna(pierna, pasos[pierna]);

}

}

//Avanza al siguiente paso
if (++avance > totalavances)
{
avance = 1;

}
}*/

void contraccionPos(void)
{

      izquierdaPosterior(solicitoPosicion(0, 0, -60, 0));
      derechaPosterior(solicitoPosicion(0, 0, -60, 0));
      izquierdaAnterior(solicitoPosicion(0, 0, -60, 0));
      derechaAnterior(solicitoPosicion(0, 0, -60, 0));
      HAL_Delay(1000);
```

```
        piernaInvertidaCINV = 1;
        izquierdaPosterior(solicitoPosicion(79, 79, 80, 0.8));
        derechaPosterior(solicitoPosicion(49, -49, 80, 0.8));
        izquierdaAnterior(solicitoPosicion(-79, 79, 80, 0.8));
        derechaAnterior(solicitoPosicion(-79, -79, 80, 0.8));
        piernaInvertidaCINV = 0;
        while (contraccion == 1)
        {
              HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);
        }

}
```

La comunicación se ejecuta en la devolución de llamada callback, se trabaja a través de la interrupción.

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
        Mensa_debug_Usart3(buffer);
        Mensa_debug_Usart3("\n");
        elegirOpcion();
        HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);

}
```

Si se produce un error de comunicación, este error se tratará aquí y el programa se iniciará de nuevo.

```
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
        if (huart->ErrorCode == HAL_UART_ERROR_ORE)
        {
              HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);
        }
        if (huart->ErrorCode == HAL_UART_ERROR_FE)
        {
              HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);

        }
        if (huart->ErrorCode == HAL_UART_ERROR_NE)
        {
              HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);

        }
        HAL_UART_Receive_IT(&huart2, (uint8_t *) buffer, 5);

}
```

En este apartado se gestionan todos los botones utilizados en la botonera del mando inalámbrico. Depende cual sea la elección las variables tomaran las formulas peustas aquí.

```
void elegirOpcion(void)
{

        contraccion = 0;
        rota = 0;
        if (buffer[0] == 'a' || memory[0] == 'a')
```

```
	{

		memory[0] = 'a';

		if (buffer[0] == 'x')
		{
			poscuerpx = TransformDato(buffer) * 50 / 4095 - 25;
		}
		if (buffer[0] == 'y')
		{

				poscuerpy = TransformDato(buffer) * 50 / 4095 -
25;


		}
	}

	if (buffer[0] == 'b' || memory[0] == 'b')
	{

		memory[0] = 'b';

		contraccion = 1;

	}
	if (buffer[0] == 'c' || memory[0] == 'c')
	{
		memory[0] = 'c';

		if (buffer[0] == 'x')
		{
			poscuerpz = -(TransformDato(buffer) * 100 / 4095 -
50);
		}
		if (buffer[0] == 'y')
		{

			pitch = TransformDato(buffer) * 1.5707 / 4095 -
0.7853981;

		}

	}
	if (buffer[0] == 'f' || memory[0] == 'f')
	{

		memory[0] = 'f';

		if (buffer[0] == 'x')
		{
			moviX = TransformDato(buffer) * 60 / 4095 - 30;
			ssx = TransformDato(buffer) * 60 / 4095 - 30;
		}
		if (buffer[0] == 'y')
		{
			moviY = TransformDato(buffer) * 60 / 4095 - 30;
			ssy = TransformDato(buffer) * 60 / 4095 - 30;
		}

	}
```

```
	if (buffer[0] == 'd' || memory[0] == 'd')
	{

		memory[0] = 'd';

		if (buffer[0] == 'x')
		{
			moviX = TransformDato(buffer) * 60 / 4095 - 30;
		}

		if (buffer[0] == 'y')
		{

			cuerpRotZ = TransformDato(buffer) * 0.2443460953 /
4095- 0.1221730476;

		}
	}

	if (buffer[0] == 'e' || memory[0] == 'e')
	{

		memory[0] = 'e';
		rota = 1;

		if (buffer[0] == 'x')
		{
			cuerpRotY = -(TransformDato(buffer) * 0.62831853 /
4095 - 0.314159265);
		}
		if (buffer[0] == 'y')
		{

			cuerpRotX = TransformDato(buffer) * 0.62831853 / 4095
- 0.314159265;

		}
	}

}


int TransformDato(char text[5])
{
	char result1[4];
	for (int i = 1; i < 5; i++)
	{
		result1[i - 1] = text[i];
	}
	return atoi(result1);

}
void Mensa_debug_Usart3(char text[32]) {
	if (HAL_UART_Transmit(&huart3, (uint8_t*) text, strlen(text),
1000)!= HAL_OK)
	{
		//código que puede ser agregado aquí para visualizar
resultados en el TERMITE3.3
	}
```

```
	return;
}
```

Este es el programa o función para la IMU, los valores de la IMU se utilizan inversamente en la cinemática inversa del cuerpo.
Como podemos apreciar en el filtro se escoge un 95% al giroscopio y un 5% al acelerómetro.

```
void LeoIMU(void)
{
	float roll, pitch, X, Y;
	BSP_ACCELERO_GetXYZ((int16_t *) accelBuf);
	BSP_GYRO_GetXYZ((float*) gyroData);
	if (accelBuf[2] > 0)
	{
		piernaInvertidaCINV = 0;
		if (rota == 0)
		{
			roll = atan2((accelBuf[1] - 128.273), (accelBuf[2] +
1277.39185))* 180 / 3.14;
			pitch=atan2((accelBuf[0] + 599.256), (accelBuf[2] +
1277.39185))* 180 / 3.14;

			X = (gyroData[1] + memX + 305.752) * (tiempIntegra +
8)/ 2000000;
			Y = (gyroData[0] + memY + 1612.06) * (tiempIntegra +
8) / 2000000;

			memX = gyroData[1] + 305.752;
			memY = gyroData[0] + 1612.06;
			anguloX = 0.95 * (anguloX + X) + 0.05 * roll;
			anguloY = 0.95 * (anguloY + Y) + 0.05 * pitch;
			++imuCounter1;
			if (imuCounter1 == 5)
			{
				imuCounter1 = 0;
				if (anguloX > 2 || anguloX < -2)
				{
					imuRotX += (anguloX * 3.14 / 180);
					if (imuRotX > 0.348889)
					{
						imuRotX = 0.348889;
					}
					if (imuRotX < -0.348889)
					{
						imuRotX = -0.348889;
					}
				}
				if (anguloY > 6 || anguloY < -6)
				{
					imuRotY += (anguloY * 3.14 / 180);
					if (imuRotY > 0.348889)
					{
						imuRotY = 0.348889;
					}
					if (imuRotY < -0.348889)
					{
						imuRotY = -0.348889;
					}
				}
			}
			//depuración o visualización con el hyperTerminal
```

```
                    sprintf(transform, "xaxis%d", (int) anguloX);
                    Mensa_debug_Usart3(transform);
                    sprintf(transform, "yaxis%d", (int) anguloY);
                    Mensa_debug_Usart3(transform);
                    Mensa_debug_Usart3("\n");
                }
            }

}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void _Error_Handler(char * file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error
return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line
number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/

/************************ (C) COPYRIGHT Escrito por Juan Daniel Miret
Rubio ***/
```

```
/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

14.2.2 Código del Mando inalámbrico

```
/**

******************************************************************************
 * File Name          : main.c
 * Description        : Main program body
 * Autor              : Juan Daniel Miret Rubio

******************************************************************************
 *
 * COPYRIGHT(c) 2017 STMicroelectronics
 *
 * Redistribution and use in source and binary forms, with or without
modification,
 * are permitted provided that the following conditions are met:
 *   1. Redistributions of source code must retain the above copyright
notice,
 *      this list of conditions and the following disclaimer.
 *   2. Redistributions in binary form must reproduce the above
copyright notice,
 *      this list of conditions and the following disclaimer in the
documentation
 *      and/or other materials provided with the distribution.
 *   3. Neither the name of STMicroelectronics nor the names of its
contributors
 *      may be used to endorse or promote products derived from this
software
 *      without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
 *
```

```
**********************************************************************
********
 */
/* Includes ------------------------------------------------------------
--------*/
#include "main.h"
#include "stm32f3xx_hal.h"
#include "adc.h"
#include "usart.h"
#include "gpio.h"

/* USER CODE BEGIN Includes */
#include <stdlib.h>
#include <string.h>
/* USER CODE END Includes */

/* Private variables ---------------------------------------------------
--------*/

/* USER CODE BEGIN PV */
__IO ITStatus Uart1Ready = RESET;
__IO ITStatus Uart3Ready = RESET;
/* Private variables ---------------------------------------------------
--------*/

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------
--------*/
void SystemClock_Config(void);
void Error_Handler(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------
--------*/
uint16_t dato_y(void);
uint16_t dato_x(void);
void enviarDato(char dato[5]);
/* USER CODE END PFP */

/* USER CODE BEGIN 0 */
/* USER CODE END 0 */

int main(void) {

      /* USER CODE BEGIN 1 */

      /* USER CODE END 1 */

      /* MCU Configuration--------------------------------------------------
--------------*/

      /* Reset of all peripherals, Initializes the Flash interface and
the Systick. */
      HAL_Init();

      /* Configure the system clock */
      SystemClock_Config();

      /* Initialize all configured peripherals */
```

```
	MX_GPIO_Init();
	MX_ADC1_Init();
	MX_USART1_UART_Init();
	MX_USART3_UART_Init();
	MX_ADC2_Init();

	/* USER CODE BEGIN 2 */

	/* USER CODE END 2 */

	/* Infinite loop */
	/* USER CODE BEGIN WHILE */
	char xBuffer[5];
	char yBuffer[5];
	char envio[5];
	uint16_t a,b;
```

1 botón no se utiliza como una interrupción externa ya que hay 2 botones en el pin 10 y sólo podemos usar una EXTI10 (interrupción externa para el pin 10. Concretamente el PF10 que equivale a'c' que equivale.

```
	while (1) {
		/* USER CODE END WHILE */

		/* USER CODE BEGIN 3 */
		if (HAL_GPIO_ReadPin(GPIOF, GPIO_PIN_10))
		{
			for (int i = 0; i < 5; i++)
			{
				envio[i] = 'c';
			}
			HAL_UART_Transmit_IT(&huart1, (uint8_t *) envio, 5);
		}
```

Leyendo los datos del joystick

```
		 a = dato_x();
		 b = dato_y();
```

Modificar los datos (int) en caracteres (char) y enviarlos.

```
		sprintf(xBuffer, "x%.4u", a);
		enviarDato(xBuffer);
		HAL_Delay(50);
		sprintf(yBuffer, "y%.4u", b);
		enviarDato(yBuffer);
		HAL_Delay(50);

		//Mensa_debug_Usart3(xBuffer);
		//Mensa_debug_Usart3(yBuffer);

	}
	/* USER CODE END 3 */

}

/** System Clock Configuration
 */
void SystemClock_Config(void) {

	RCC_OscInitTypeDef RCC_OscInitStruct;
```

```
	RCC_ClkInitTypeDef RCC_ClkInitStruct;
	RCC_PeriphCLKInitTypeDef PeriphClkInit;

	/**Initializes the CPU, AHB and APB busses clocks
	 */
	RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
	RCC_OscInitStruct.HSIState = RCC_HSI_ON;
	RCC_OscInitStruct.HSICalibrationValue = 16;
	RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
	RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
	RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL12;
	if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
		Error_Handler();
	}

	/**Initializes the CPU, AHB and APB busses clocks
	 */
	RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_SYSCLK
			| RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
	RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
	RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
	RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
	RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

	if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) !=
HAL_OK) {
		Error_Handler();
	}

	PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART1
			| RCC_PERIPHCLK_USART3 | RCC_PERIPHCLK_ADC12;
	PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
	PeriphClkInit.Usart3ClockSelection = RCC_USART3CLKSOURCE_PCLK1;
	PeriphClkInit.Adc12ClockSelection = RCC_ADC12PLLCLK_DIV1;
	if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK) {
		Error_Handler();
	}

	/**Configure the Systick interrupt time
	 */
	HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq() / 1000);

	/**Configure the Systick
	 */
	HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

	/* SysTick_IRQn interrupt configuration */
	HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* USER CODE BEGIN 4 */

uint16_t dato_y(void)
{
	HAL_ADC_Start(&hadc1);
	if (HAL_ADC_PollForConversion(&hadc1, 10) != HAL_OK)
	{
	}
	uint16_t b = HAL_ADC_GetValue(&hadc1);
	return b;
```

```
	HAL_ADC_Stop(&hadc1);
}

uint16_t dato_x(void) {
	HAL_ADC_Start(&hadc2);
	if (HAL_ADC_PollForConversion(&hadc2, 10) != HAL_OK)
	{
	}
	uint16_t b = HAL_ADC_GetValue(&hadc2);
	return b;
	HAL_ADC_Stop(&hadc2);
}

void Mensa_debug_Usart3(char text[32]) {
	if (HAL_UART_Transmit(&huart3, (uint8_t*) text, strlen(text),
1000)
			!= HAL_OK) {

	}
	HAL_Delay(50);
	return;
}

/*void enviarDato(char text[5])
 {
 if(HAL_UART_Transmit(&huart1, (uint8_t*) text, strlen(text),1000)!=
HAL_OK)
 {

 }


 return;
 }*/
```

Código para enviar los datos, la transmisión se realiza a través de la interrupción, por lo tanto, después de que los bits se transmiten vamos automáticamente a la función de devolución de llamada como se ve debajo.

```
void enviarDato(char dato[5]) {
	HAL_UART_Transmit_IT(&huart1, (uint8_t *) dato, 5);

//##-5- Esperando al final de la transferencia
###################################
	while (Uart1Ready != SET)
	{


	}

	//Reset transmision
	Uart1Ready = RESET;

}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *UartHandle) {
	//dataReceive(ack);
	//Set transmission flag: transfer complete
	Uart1Ready = SET;
	//HAL_GPIO_WritePin(GPIOE, LD5_Pin, GPIO_PIN_SET);
```

```
}


/* USER CODE END 4 */

/**
 * @brief  This function is executed in case of error occurrence.
 * @param  None
 * @retval None
 */
void Error_Handler(void) {
      /* USER CODE BEGIN Error_Handler */
      /* User can add his own implementation to report the HAL error
return state */
      while (1) {
      }
      /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
 * @brief Reports the name of the source file and the source line
number
 * where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t* file, uint32_t line)
{
      /* USER CODE BEGIN 6 */
      /* User can add his own implementation to report the file name
and line number,
       ex: printf("Wrong parameters value: file %s on line %d\r\n",
file, line) */
      /* USER CODE END 6 */

}

#endif

/**
 * @}
 */

/**
 * @}
 */
/************************ (C) COPYRIGHT escrito por Juan Daniel Miret
Rubio********/
/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

En este archivo.c estan las Interupciones externas:

```
/**
```

```
******************************************************************************
 * @file    stm32f3xx_it.c
 * @brief   Interrupt Service Routines.
******************************************************************************
 *
 * COPYRIGHT(c) 2017 STMicroelectronics
 *
 * Redistribution and use in source and binary forms, with or without
modification,
 * are permitted provided that the following conditions are met:
 *   1. Redistributions of source code must retain the above copyright
notice,
 *      this list of conditions and the following disclaimer.
 *   2. Redistributions in binary form must reproduce the above
copyright notice,
 *      this list of conditions and the following disclaimer in the
documentation
 *      and/or other materials provided with the distribution.
 *   3. Neither the name of STMicroelectronics nor the names of its
contributors
 *      may be used to endorse or promote products derived from this
software
 *      without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
 *
******************************************************************************
 */
/* Includes ------------------------------------------------------------
--------*/
#include "stm32f3xx_hal.h"
#include "stm32f3xx.h"
#include "stm32f3xx_it.h"

/* USER CODE BEGIN 0 */
```

```
char envio[5];
/* USER CODE END 0 */

/* External variables ---------------------------------------------
--------*/
extern UART_HandleTypeDef huart1;
extern UART_HandleTypeDef huart3;

/******************************************************************
*********/
/*            Cortex-M4 Processor Interruption and Exception Handlers
*/
/******************************************************************
*********/

/**
 * @brief This function handles System tick timer.
 */
void SysTick_Handler(void) {
      /* USER CODE BEGIN SysTick_IRQn 0 */

      /* USER CODE END SysTick_IRQn 0 */
      HAL_IncTick();
      HAL_SYSTICK_IRQHandler();
      /* USER CODE BEGIN SysTick_IRQn 1 */

      /* USER CODE END SysTick_IRQn 1 */
}

/******************************************************************
*********/
/* STM32F3xx Peripheral Interrupt Handlers
*/
/* Add here the Interrupt Handlers for the used peripherals.
*/
/* For the available peripheral interrupt handler names,
*/
/* please refer to the startup file (startup_stm32f3xx.s).
*/
/******************************************************************
*********/

/**
 * @brief This function handles RCC global interrupt.
 */
void RCC_IRQHandler(void) {
      /* USER CODE BEGIN RCC_IRQn 0 */

      /* USER CODE END RCC_IRQn 0 */
      /* USER CODE BEGIN RCC_IRQn 1 */

      /* USER CODE END RCC_IRQn 1 */
}

/**
 * @brief This function handles EXTI line2 and Touch Sense controller.
 */
void EXTI2_TSC_IRQHandler(void) {
      /* USER CODE BEGIN EXTI2_TSC_IRQn 0 */
```

Se Envia una cadena de 5 caracteres cuando se presiona el botón ("por ejemplo" bbbbb ").

El resto de interupciones son análogas a esta.

```
	if (HAL_GPIO_ReadPin(GPIOF, GPIO_PIN_2)) {
		for (int i = 0; i < 5; i++) {
			envio[i] = 'b';
		}
		HAL_UART_Transmit_IT(&huart1, (uint8_t *) envio, 5);

	}
	/* USER CODE END EXTI2_TSC_IRQn 0 */
	HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_2);
	/* USER CODE BEGIN EXTI2_TSC_IRQn 1 */

	/* USER CODE END EXTI2_TSC_IRQn 1 */
}

/**
 * @brief This function handles EXTI line3 interrupt.
 */
void EXTI3_IRQHandler(void) {
	/* USER CODE BEGIN EXTI3_IRQn 0 */
	if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_3)) {
		for (int i = 0; i < 5; i++) {
			envio[i] = 'f';
		}
		HAL_UART_Transmit_IT(&huart1, (uint8_t *) envio, 5);

	}
	/* USER CODE END EXTI3_IRQn 0 */
	HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_3);
	/* USER CODE BEGIN EXTI3_IRQn 1 */

	/* USER CODE END EXTI3_IRQn 1 */
}

/**
 * @brief This function handles EXTI line[9:5] interrupts.
 */
void EXTI9_5_IRQHandler(void) {
	/* USER CODE BEGIN EXTI9_5_IRQn 0 */
	if (HAL_GPIO_ReadPin(GPIOF, GPIO_PIN_9)) {

		for (int i = 0; i < 5; i++) {
			envio[i] = 'a';
		}

		HAL_UART_Transmit_IT(&huart1, (uint8_t *) envio, 5);

	}
	/* USER CODE END EXTI9_5_IRQn 0 */
	HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_9);
	/* USER CODE BEGIN EXTI9_5_IRQn 1 */

	/* USER CODE END EXTI9_5_IRQn 1 */
}

/**
 * @brief This function handles USART1 global interrupt / USART1 wake-
up interrupt through EXTI line 25.
 */
void USART1_IRQHandler(void) {
```

```
	/* USER CODE BEGIN USART1_IRQn 0 */

	/* USER CODE END USART1_IRQn 0 */
	HAL_UART_IRQHandler(&huart1);
	/* USER CODE BEGIN USART1_IRQn 1 */

	/* USER CODE END USART1_IRQn 1 */
}

/**
 * @brief This function handles USART3 global interrupt / USART3 wake-
up interrupt through EXTI line 28.
 */
void USART3_IRQHandler(void) {
	/* USER CODE BEGIN USART3_IRQn 0 */

	/* USER CODE END USART3_IRQn 0 */
	HAL_UART_IRQHandler(&huart3);
	/* USER CODE BEGIN USART3_IRQn 1 */

	/* USER CODE END USART3_IRQn 1 */
}

/**
 * @brief This function handles EXTI line[15:10] interrupts.
 */
void EXTI15_10_IRQHandler(void) {
	/* USER CODE BEGIN EXTI15_10_IRQn 0 */
	if (HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_11)) {

		for (int i = 0; i < 5; i++) {
			envio[i] = 'd';
		}

		HAL_UART_Transmit_IT(&huart1, (uint8_t *) envio, 5);

	}
	if (HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_10)) {

		for (int i = 0; i < 5; i++) {
			envio[i] = 'e';
		}

		HAL_UART_Transmit_IT(&huart1, (uint8_t *) envio, 5);

	}

	/* USER CODE END EXTI15_10_IRQn 0 */
	HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_10);
	HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_11);
	/* USER CODE BEGIN EXTI15_10_IRQn 1 */

	/* USER CODE END EXTI15_10_IRQn 1 */
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

/************************ (C) COPYRIGHT escrito por Juan Daniel Miret
Rubio ********/
```

```
/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

15 Conclusiones

Como futuro ingeniero industrial con una especialización en Electromecánica se a aprendido mucho sobre Electrónica. Se ha adquirido un gran conocimiento previo sobre la electrónica y la tecnología de la información y las comunicaciones TIC. Desde diseñar e implementar una PCB auxiliar para nuestro microcontrolador principal, pasando por la programación en C del código el cual sigue las reglas de la cinematica inversa, hasta llegar a montar y soldar elementos electrónicos diminutos. Ni siquiera mencionar que nunca se habia oído hablar de STMicroelectronics antes de cursar la asgnatura de sistemas embebidos. Del programa SolidWorks si que se había oído hablar de él, pero nunca se había utilizado, eso si ayudo bastante los conocimientos en otros programas similares como autocad o Ansys visto en las asignaturas de mecánica de este Master de Ingenieria Mecatrónica. Combinar estos elementos electrónicos con temas mecánicos como por ejemplo diseñar el robot cuadrúpedo en SolidWorks y posteriormente sacar las piezas del robot impresas en 3D, hizo que este proyecto fuera muy enriquecedor y una experiencia que no se olvidara en el futuro.

Comenzar con el robot de 2GDL (es decir 8 servomotores en total 2 por cada pierna) era bueno y necesario para aprender sobre la electrónica y los programas. Sin embargo, este no era el objetivo final y se avanzo rápidamente hacia el cuadrúpedo mas avanzado y que vemos mas videos sobre ellos el cuadrúpedo de 3GDL (es decir 12 servomotores en total, 3 por cada pata).

El robot 3GDL funciona bien. Puede caminar hacia adelante, hacia atrás y hacia los lados y puede girar. Rotar el cuerpo alrededor de los diferentes ejes, así como mover el cuerpo hacia arriba y hacia abajo. Cuando el robot se mueve hacia delante o hacia los lados puede evitar chocar con cualquier obstáculo que se le ponga por su dirección y por tanto pararse anque el controlador del robot le diga que continue (bien por despiste de la persona que maneje el mando, por ejemplo). La opción de posición de contracción y guardado integrada ofrece un almacenamiento fácil y rápido del cuadrúpedo, y además ocupando el minimo espacio. Sin embargo, se podrían hacer muchas mas mejoras. Para evitar que una pierna se quede en el aire, por ejemplo, al hacer una rotación del cuerpo, los sensores de presión podrían ser implementados en los pies del cuadrúpedo. También se podría implementar un algoritmo adicional para sincronizar el tiempo de llegada de los motores accionados a sus posiciones asignadas, es decir que todos los motores lleguen a su posición en el mismo intervalo de tiempo. La IMU tampoco funciona perfectamente. Cuando se opera con el robot en estatico nada mas empezar el robot reaccióna bien a los cambios de altura en el suelo y logra equilibrar el cuerpo, pasados unos minutos le cuesta mas reaccionar bien a estos cambios en el terreno. Ademas, la combinación de la IMU y la marcha no dan los resultados esperados. Debido a la falta de tiempo este problema no ha podido mejorarse. La consecuencia de esto es que el robot cuadrúped de 3GDL no puede caminar sobre terreno áspero como se deseaba inicialmente. El diseño mecánico del robot es pequeño y simétrico, tal como se pretendia. Cada componente necesario encaja en el diseño y por esa parte el autor del proyecto esta muy contentos con el resultado. Las piezas en 3D que corresponden a los pies del robot, se han mejorado hasta llegar a diseñarlas con mas líneas curvas y angulos para que asi cada esquina no sufran tanto impacto de rotura. De todos modos, este aspecto puede ser mejorado por algún otro alumno entusiasta mas especializado en mecánica que quiera mejorar este proyecto.

El diseño electrónico también es bastante bueno. En retrospectiva, tomó más tiempo del inicialmente pensado. El diseño e implementación de la PCB JDMR 0617 ayudo muchísimo a la hora de hacer las conexiones muy fáciles y seguras. Esto fue mejor que ir soldando miles de cables pasando los pines de la placa Nucleo principal STMF334R8 a una placa board por ejemplo utilizada en practicas. Esta alternativa desde luego no gusto.

Como se puede ver, se pueden hacer muchas mejoras. Se espera que este informe sea una guía clara para todos los estudiantes de la robótica y un buen punto de partida para los alumnos futuros que quieran construir y mejorar este proyecto en el instituto ITACA.

Estimación de tiempo Empleado

Semana	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Programa de instalación uVision 5																
Estudio de programación en C																
Conocer la electrónica																
Estudio cinemático																
2GDL mecanica y electrónica																
STM32cubeMX																
Programación de posiciones iniciales																
Calibracion motores																
Programación joystick																
Programación inalámbrica																
Programación USB a serie																
Programación de la marcha gateo																
3GDL mecanica y electrónico																
STM32cubeMX																
Programación de la posición inicial																
Motores de calibración																
Joystick de programación																
Programación inalámbrica																
Tabla de opción de programación																
Programación USB a serie																
Programación de la marcha a pie																
Prógramación de la rotación																
Programación IMU																
Programación sensor de distancia																
Diseño en SolidWorks																
Imprimir las piezas																
Ensamblaje de los componentes																
Probando el robot cuadrúpredo																
Memoria																

Bibliografia

http://www.st.com/content/ccc/resource/technical/document/datasheet/f0/6d/14/bf/7c/d8/49/91/DM00133214.pdf/files/DM00133214.pdf/jcr:content/translations/en.DM00133214.pdf

http://wiki.robotica.webs.upv.es/wiki-de-robotica/introduccion/historia/

https://es.wikipedia.org/wiki/Pata_(artr%C3%B3podos)

https://es.wikipedia.org/wiki/Insecta

https://www.google.es/search?q=coxa+femur+y+tibia&safe=off&tbm=isch&imgil=lWnoPqIwJoP45M%253A%253BSG3XM1m8acKuIM%253Bhttp%25253A%25252F%25252Flearn.trossenrobotics.com%25252Fprojects%25252F134-phantomx-quadruped-assembly-guide.html&source=iu&pf=m&fir=lWnoPqIwJoP45M%253A%252CSG3XM1m8acKuIM%252C_&usg=__jFEqQgjfAx_h9cgqOzyidM1xkYE%3D&biw=1474&bih=724&ved=0ahUKEwidjuaSpJDWAhUEzRQKHbM5CMEQyjcINA&ei=pcSvWd2nK4SaU7PzoIgM#imgrc=lWnoPqIwJoP45M:

https://es.scribd.com/doc/100646117/pasar-codigo-java-hacia-word-manteniendo-formato-y-colores-de-Netbeans

http://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf

http://techdocs.altium.com/display/ADOH/Tutorial+-+Getting+Started+with+PCB+Design

http://www.domanrchobby.com/content/?97.html

https://es.aliexpress.com/store/product/1pcs-lot-free-shipping-DM-RS1513MD-metal-gear-digital-low-profile-300degree-Humanoid-robot-servo/316597_32467096396.html

https://developer.mbed.org/users/liamg/code/MEMSdemoBoardMKI124V1/graph

http://www.sensorsmag.com/components/compensating-for-tilt-hard-iron-and-soft-iron-effects

https://io9.gizmodo.com/the-world-s-first-mind-controlled-robotic-leg-is-ready-1401103956

STMicroelectronics, "UM1570 User manual," July 2016. [Online]. [Accessed April 2017].

STMicroelectronics, "UM1724 User manual," November 2016. [Online]. [Accessed April 2017].

https://www.digikey.ca/product-detail/en/stmicroelectronics/STEVAL-MKI124V1/497-13014-ND/3306163

http://www.fatlion.com/sailplanes/images/jrconnector.png

http://www.fatlion.com/sailplanes/images/jrconnector.png

STMicroelectronics, "STM32F334x4 STM32F334x6 STM32F334x8 Manual microcontroller," September 2016. [Online]. [Accessed April 2017].

STMicroelectronics, "STEVAL-MKI124V1 manual," [Online].

https://www.researchgate.net/publication/4106039_The_inverse_kinematics_solutions_of_industrial_robot_manipulators

http://www.st.com/content/ccc/resource/technical/document/datasheet/43/37/e3/06/b0/bf/48/bd/DM00036465.pdf/files/DM00036465.pdf/jcr:content/translations/en.DM00036465.pdf

https://cdn-shop.adafruit.com/datasheets/LSM303DLHC.PDF

https://oscarliang.com/quadruped-robot-gait-study/

https://makezine.com/2016/11/22/robot-quadruped-arduino-program/

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA