



VNiVERSiDAD
DE SALAMANCA

CAMPUS OF INTERNATIONAL EXCELLENCE

DISEÑO Y REALIZACIÓN DE UN ROBOT BÍPEDO MICROCONTROLADO

GRADO EN INGENIERÍA MECÁNICA

ESCUELA POLITÉCNICA SUPERIOR DE ZAMORA

DEPARTAMENTO DE FÍSICA APLICADA

ÁREA DE ELECTRÓNICA

TRABAJO FIN DE GRADO



AUTOR: JOSÉ GARCÍA MANZANARO

TUTOR: BEATRIZ GARCÍA VASALLO

COTUTOR: MIGUEL ÁNGEL RABANILLO DE LA FUENTE

FECHA DE ADJUDICACIÓN: MARZO DE 2016

FECHA DE PRESENTACIÓN: SEPTIEMBRE DE 2016

Agradecimientos

Me gustaría empezar dedicando este trabajo a mi tutora Beatriz por haberme propuesto un proyecto tan multidisciplinar, diferente y haber estado tan atenta a todas mis cuestiones.

Especiales agradecimientos también a Pedro, por enseñarme Solid y ayudarme con cuestiones escurridizas en pleno mes de agosto. Sin sus aportaciones y explicaciones el proyecto no sería el mismo.

A todas las personas, amigos, compañeros y profesores que he conocido en esta ciudad que han hecho que este breve capítulo de mi vida haya merecido la pena. Gracias a ellos he crecido no solamente en lo profesional, sino también en lo personal.

Y a mí padre, por haber estado siempre ahí, incondicional. Sin ti, esto nunca hubiera sucedido.

Gracias

RESUMEN

Esta memoria es el resultado de la construcción de un robot bípedo controlado mediante Arduino. Consta de ocho grados de libertad y camina con la rodilla hacia atrás, tal y como lo hacen las aves. Se expondrán de forma separada cuatro capítulos: el diseño mecánico, su construcción, el funcionamiento de cada uno de sus elementos electrónicos y su código.

Una de las novedades más importantes de este proyecto es la estructura especialmente diseñada para ser impresa con una impresora 3D. Se ha comentado, aunque levemente, el estado actual de la técnica y los problemas más importantes que han ido surgiendo propios de una tecnología en desarrollo.

El resultado final ha sido un robot capaz de adquirir una posición estable en superficies horizontales y planas, desplazarse frontalmente, realizar giros, detectar obstáculos con ayuda de un giróscopo-acelerómetro para controlar el balanceo.

ABSTRACT

This thesis is the result of constructing an Arduino-controlled biped robot with eight DOF and knees facing backwards, similar to those of a bird's.

It is composed of four separate chapters: mechanical design, construction, code and an explanation of every electronic device and how they work.

One of the most important novelties in this project is the structure, especially designed to be generated by a 3D printer. It is given an overview of the current state of the art and the most important problems a technology still in development entails.

The final result has been a robot which is able to remain in a stable position on a flat horizontal surface, and which can move frontally, turn to the sides, detect obstacles and avoid them using an accelerometer-gyroscope to maintain its balance

CONTENIDO

Resumen	2
Abstrabt.....	2
Índice de ilustraciones.....	6
Índice de tablas	8
Introducción.....	9
Historia y estado actual de la técnica en la robótica	9
Historia y estado de la técnica en la impresión 3D.....	16
Capítulo 1. Mecánica del robot	19
- Elección del plástico.....	20
- Diseño.....	21
- Grados de libertad	22
- Diseño de los engranajes	23
- Unión piñones-servos	29
- Unión Servo-Fémur.....	31
- Unión Servo- Empeine	32
- Unión Servo-Espinilla	32
- Uniones cojinetes-espinillas	34
- Uniones cojinetes-ejes	34
- Unión Servo-Pie	35
- Unión Servos-Cabeza.....	36
- Retículas	37
- Diseño Final	38
Capítulo 2. Electrónica	38
- Arduino.....	39
- Batería	40
- Servos	41
- Interruptor.....	42
- Sensor de ultrasonidos.....	42
- Circuito de alimentación de los servos.....	44
- Led RGB	46
- IMU (Inertial Measurement Unit)	48

Capítulo 3. Programación y código	60
- IrA	60
- Posición de Inicio	61
- Ir Hasta	62
- Zancada derecha/izquierda	62
- Ir A Paso.....	63
- Evitar desde derecha/izquierda.....	63
- Maniobra de girar	63
- Programa Caminar1.....	64
- Programa Caminar2.....	65
Capítulo 4. Bipedestación y características finales	66
- Bipedestación	66
- Coste global.....	70
- Especificaciones técnicas	73
- Líneas de futuro.....	74

ÍNDICE DE ANEXOS

Anexo 1. Diseño mecánico	75
Anexo 1.1 Tabla de conjuntos.....	75
Anexo 1.2. Tabla de Piezas.	79
Anexo 2. Estudio del comportamiento de los ángulos.....	84
Anexo 2.1 Tabla ángulos para el primer programa	84
Anexo 2.2 Datos globales de ángulos para el primer programa.....	85
Anexo 3. Datos de entrada para el cálculo de los engranajes.....	86
Anexo 4. Parámetros recomendados de impresión	88
Anexo 5. Código.....	89
Anexo 5.1 Código completo.....	89
Anexo 5.2. Diagrama de contexto.....	106
Anexo 5.3. Organigrama y código del ultrasonidos.....	107
Anexo 5.4. Organigrama y código de la rutina “tomarDatosMPU”	108
Anexo 5.5. Organigrama y código de la rutina “Selector”.....	110
Anexo 5.6. Organigrama y código de la rutina “posicionInicio”.....	112
Anexo 5.7. Organigrama y código de la rutina “irHasta”.....	115
Anexo 5.8. Organigrama y código de la rutina “irAPaso”	117
Anexo 5.9 Organigrama y código de la rutina “Caminar1”	118
Anexo 6. Tabla de tiempos y pesos de impresión.	119
Anexo 7. Esquemas.....	120
Anexo 7.1 Esquema eléctrico.....	120
Anexo 7.2. Esquema gráfico	121

ÍNDICE DE ILUSTRACIONES

<i>Ilustración 1. Robot ASIMO</i>	10
<i>Ilustración 2. Robot Nao.....</i>	11
<i>Ilustración 3. Robot Petman.....</i>	12
<i>Ilustración 4. Robot Atlas</i>	13
<i>Ilustración 5. Robot iCub</i>	14
<i>Ilustración 6. Prusa I3 MK2</i>	16
<i>Ilustración 7. Archivo en formato .stl</i>	17
<i>Ilustración 8. Extracto de un archivo .gcode</i>	17
<i>Ilustración 9. Esquema del funcionamiento en el modelado por deposición fundida</i>	18
<i>Ilustración 10. Impresora 3D casera.....</i>	19
<i>Ilustración 11. Fenómeno Warping.....</i>	20
<i>Ilustración 12. Máquina de guerra ficticia</i>	21
<i>Ilustración 13. Modelo inicial</i>	22
<i>Ilustración 14. Diseño final del conjunto cabeza-ultrasonidos.....</i>	23
<i>Ilustración 15. Representación del diseño para calcular los esfuerzos de las articulaciones.....</i>	24
<i>Ilustración 16. Mecanismo de 4 barras</i>	24
<i>Ilustración 17. Unión espinilla-fémur</i>	26
<i>Ilustración 18. Engranaje recto con socavación</i>	28
<i>Ilustración 19. Diseño inicial del acoplamiento de los engranajes a los servos</i>	30
<i>Ilustración 20. Primera versión del piñón para la articulación R2.....</i>	30
<i>Ilustración 21. Solución final del enlace entre el piñón y el servo</i>	31
<i>Ilustración 22. Articulación de la cabeza seccionada</i>	31
<i>Ilustración 23. Detalle de la unión servo-empeine</i>	32
<i>Ilustración 24. Descomposición de fuerzas en engranajes.....</i>	32
<i>Ilustración 25. Detalle de la unión servo-espinilla y detalle del ala del servo</i>	33
<i>Ilustración 26. Detalle de fuerzas en la unión Servo-Espinilla</i>	33
<i>Ilustración 27. Diseño de los alojamientos de los cojinetes.</i>	34
<i>Ilustración 28. Solución para alojar a los cojinetes y detalle de cotas en la articulación espinilla-fémur..</i>	35
<i>Ilustración 29. Unión entre el pie y el servo.</i>	36
<i>Ilustración 30. Primer diseño de la unión entre los servos y la cabeza</i>	36
<i>Ilustración 31. Solución definitiva para fijar los servos a la cabeza</i>	37
<i>Ilustración 32. Pieza con retícula "1.04 EMPEINE".....</i>	37
<i>Ilustración 33. Pieza con retícula "1.05 ESPINILLA".....</i>	38
<i>Ilustración 34. Pieza con retícula "1.06 FEMUR".....</i>	38
<i>Ilustración 35. Logo de Arduino</i>	39
<i>Ilustración 36. Conector Jack macho 5.5 x 2.1mm</i>	40



Ilustración 37. Placa Arduino UNO.....	40
Ilustración 38. Batería Floureon 7.4V 1000mAh	41
Ilustración 39. Servo SG90 9g.....	41
Ilustración 40. Señal PWM extraída de la hoja de especificaciones.....	42
Ilustración 41. Representación CAD del interruptor.....	42
Ilustración 42. HC-SR04. Sensor de distancia por ultrasonidos	43
Ilustración 43. Representación CAD del circuito alimentador de servos	45
Ilustración 44. Fotografía de la pinza acoplada al regulador de tensión	45
Ilustración 45. Diodo Led común	47
Ilustración 46. Diodo Led RGB	47
Ilustración 47. Módulo RGB Led empleado en el proyecto.....	48
Ilustración 48. IMU MPU 5060 sobre breakboard GY521	49
Ilustración 49. El bus I2C de Arduino Uno	49
Ilustración 50. Comparativa entre un ácaro y unas piezas MEMS	50
Ilustración 51. Efecto piezoelectrónico en el cuarzo.....	51
Ilustración 52. Esquema de funcionamiento de un acelerómetro basado en el efecto piezoelectrónico.....	51
Ilustración 53. Acelerómetro piezoelectrónico.....	52
Ilustración 54. Esquema de funcionamiento de un acelerómetro MEMS capacitivo	53
Ilustración 55. Acelerómetro MEMS capacitivo real	52
Ilustración 56. Representación de la cabeza respecto tres ejes de coordenadas.....	54
Ilustración 57 Ejemplo visual del efecto Coriolis.	55
Ilustración 58. La masa vibra a causa de los campos electromagnéticos.....	56
Ilustración 59. Esquema del funcionamiento de un giróscopo MEMS	56
Ilustración 60. Giróscopo MEMS real.	57
Ilustración 61. Dirección y denominación de los ejes	59
Ilustración 62. Extracto del programa 2.....	60
Ilustración 63. Captura de pantalla del código	63
Ilustración 64. Extracto de la rutina evitarDesdeDerecha.....	64
Ilustración 65. Bucle del programa principal.....	64
Ilustración 66. La suma de ángulos menos el ángulo del empeine es igual a 360º	66
Ilustración 67. Cambio de pesos en diagonal.....	67
Ilustración 68. Cambio de pesos sobre un eje	68

ÍNDICE DE TABLAS

<i>Tabla 1. Radio de giro del robot en función de la superficie</i>	69
<i>Tabla 2. Lista y coste de los materiales.....</i>	71
<i>Tabla 3. Lista y coste de herramientas.....</i>	72
<i>Tabla 4. Coste de la mano de obra.....</i>	72
<i>Tabla 5. Desglose del coste total.....</i>	72
<i>Tabla 6. Especificaciones técnicas finales</i>	73

INTRODUCCIÓN

Desde el inicio de la robótica científicos e ingenieros ha ido dando soluciones a la bipedestación como método de desplazamiento. La proximidad ergonómica es la principal ventaja que ofrece esta forma de movimiento en un mundo donde la interacción entre los humanos en las máquinas es más notoria que nunca. Este trabajo ha pretendido dar una solución original, alejado de los clásicos robots que tratan de copiar la bipedestación humana se ha recreado una forma de andar más “robótica” cercana a los movimientos de aves como el aveSTRUZ con la rodilla articulada de forma inversa.

HISTORIA Y ESTADO ACTUAL DE LA TÉCNICA EN LA ROBÓTICA

La palabra robot fue introducida en la literatura en 1920, en la obra R.U.R. (Rossum's Universal Robots), de Karel Čapek, nacido en lo que hoy es la República Checa¹. Etimológicamente, robot viene de la palabra checa robota, que viene a significar "labor forzada", servicio, esclavo... Este nombre fue utilizado en el imperio austro-húngaro hasta 1848. La palabra inventada por Josef Čapek sirve para designar a las máquinas trabajadoras o serviles.²

El inicio de la robótica actual puede fijarse en la industria textil del siglo XVIII, cuando Joseph Jacquard inventa en 1801 una máquina textil programable mediante tarjetas perforadas. Más tarde, en la Revolución Industrial se impulsó el desarrollo de estos agentes mecánicos³

La automática tuvo y tiene un papel predominante en la industria desde finales del siglo XX. No obstante, los robots humanoides ni los bípedos han tenido nunca un papel relevante en la sociedad, a diferencia de lo que se pensaba en el siglo pasado. Sin embargo, los nuevos materiales, electrónica, los avances en computación y lo competitivos que son económicamente hace pensar que vivir en mundo donde los humanoides hagan el trabajo más duro sea posible.

La ventaja que ofrece la bipedestación como forma de movimiento en robots reside en la ergonomía. El hecho de vivir en un mundo por y para humanos, hace difícil compaginar otras formas de movimientos, como mediante ruedas. Subir escaleras, accionar el botón de un ascensor, o evitar obstáculos por la calle requieren de complicados sistemas mecánicos. Es aquí donde entra la rama de la bipedestación en la robótica: si queremos vivir en un mundo donde los robots interactúen frecuentemente con nosotros, debemos adaptar los robots al entorno de los humanos.

Aunque se hayan dado grandes pasos a nivel mundial, España es uno de los países desarrollados a la cola de esta tecnología, no obstante, el Centro de Automática y Robótica CSIC en Madrid, investiga la bipedestación orientada a la anatomía humana. Concretamente, se centra en investigar y tratar anomalías físicas en la distribución de peso al caminar, entre otras cosas.⁴

A continuación, se van a mostrar cuatro de los robots bípedos más relevantes en la actualidad:

➤ *ASIMO*

Advanced Step in Innovative MObility. Proyecto japonés desarrollado por Honda desde el año 2000. En su momento, fue el robot de referencia en interactuar con humanos, imitar gestos, subir y bajar escaleras fluida y suavemente y desplazarse caminando de forma muy humana. ASIMO sigue siendo un proyecto de referencia internacional. Cuenta con 34 grados de libertad.⁵



Ilustración 1. Robot ASIMO

➤ *NAO*

NAO Robot se presenta como un robot bípedo de propósito general. Aparte de tener aplicaciones didácticas, desde primaria hasta la universidad, de apoyo a personas discapacitadas como autistas, etcétera, está especialmente enfocado a investigadores. Se concibe como una plataforma de desarrollo en la que los investigadores puedan implementar y probar sus aplicaciones. Es uno de los robots de este tipo más relevantes en el momento. Solo pesa 4.3kg y mide 58cm, aunque su precio ronda los 7.000€, hay que destacar la gran cantidad de periféricos que integra. Algunos son:

- Dos cámaras HD
- Cuatro micrófonos
- Nueve sensores táctiles
- Dos sensores de ultrasonidos
- Ocho sensores de presión
- Un acelerómetro con giróscopo
- Dos altavoces

Cabe destacar que también tiene un procesador ATOM a 1.6GHz, 25 grados de libertad y conectividad WiFi y Ethernet⁶



Ilustración 2. Robot Nao

➤ PETMAN

Es otro robot antropomórfico desarrollado por *Boston Dynamics*. Esta especialmente diseñado para imitar movimientos humanos y probar equipos de protección química. El movimiento es especialmente ágil y realista. Las últimas versiones de PETMAN pueden reproducir un amplio repertorio de acciones con el objetivo de evaluar los trajes en las condiciones más desfavorables. También lleva sensores que controlan la temperatura y la humedad para simular la sudoración y, en definitiva, todo lo necesario para proporcionar unas condiciones realistas.

Boston Dynamics es una compañía estadounidense recientemente absorbida por Google⁷ orientada a la robótica con fines militares. Es por ello que no se revelan características importantes de sus máquinas.



Ilustración 3. Robot Petman

➤ *Atlas*

Desarrollado por la compañía estadounidense *Boston Dynamics*. Es un robot humanoide ganador de la competición *DARPA Robotics Challenge* siendo capaz de caminar por cualquier terreno ya sea, nieve, desniveles importantes, rocas, etc. Puede levantar cantidades importantes de peso, moverlas entre terrenos complicados y corregir la trayectoria si fuese necesario. Consta de 28 grados de libertad hidráulicos, una cámara estereoscópica y un telémetro laser para medir distancias. Está basado en el proyecto Petman pero mantiene claras diferencias. A principios de 2015, salió a la luz una “actualización” que modifica el 75% del diseño con el que ganó el concurso. La nueva versión se llama “ATLAS DRC Robot”⁸

El fabricante apenas revela características técnicas del robot, pero es uno de los proyectos más importantes en esta área.⁹

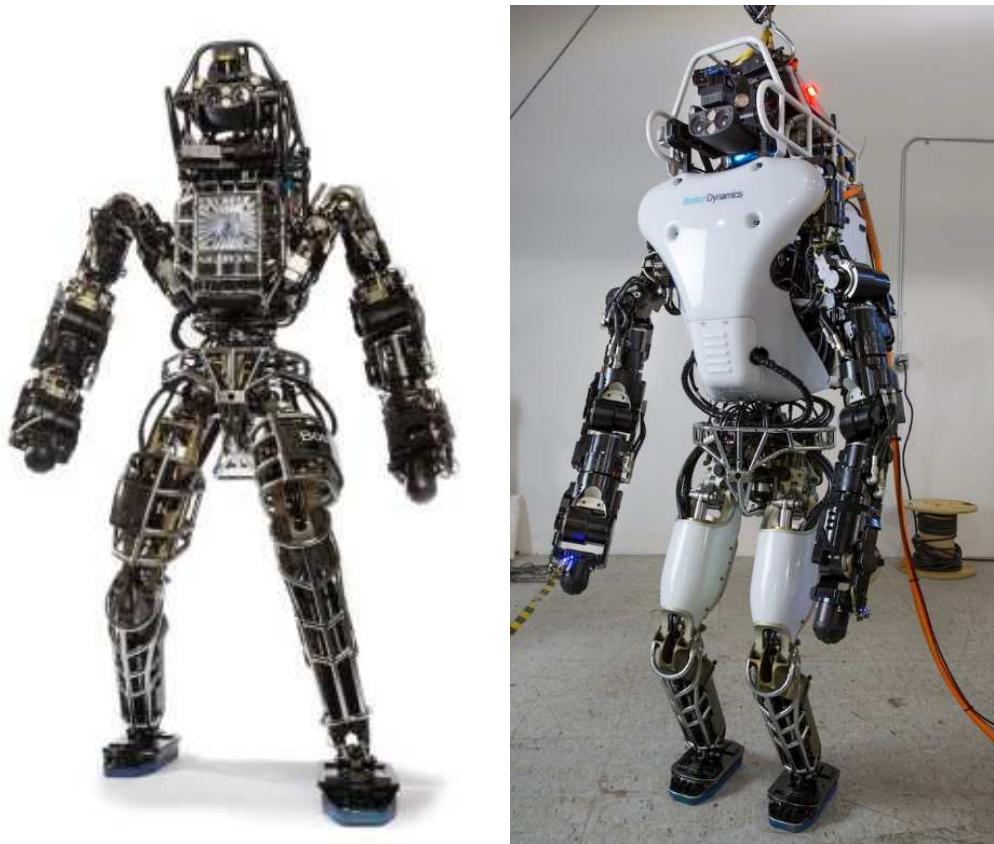


Ilustración 4. A la izquierda *Atlas Robot* original. A la derecha la actualización *ATLAS DRC Robot*

➤ *iCub*

Es uno de los proyectos líder en la Unión Europea la cual financia el proyecto en el que se ven involucrados 20 centros de investigación. A diferencia de los otros robots no destaca en su forma de caminar, por el momento, no es rápido ni fluido. Su diseño trata de imitar las actitudes de un niño de unos dos años y medio. Destaca por ser uno de los proyectos punteros en investigar e imitar la conciencia humana.

Tiene 53 grados de libertad, mide 105cm, pesa 24kg y cada unidad se valora en unos 250.000€.

10

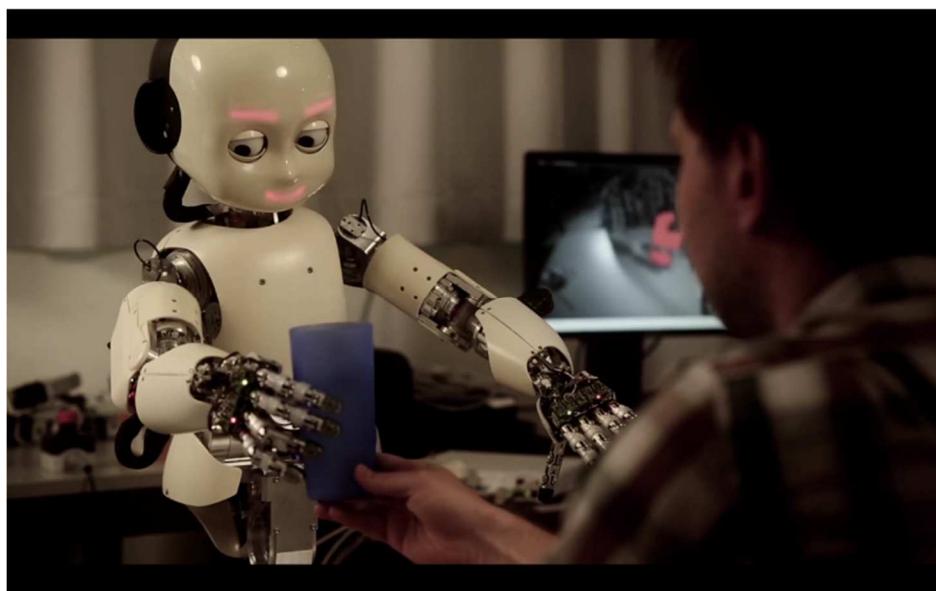


Ilustración 5. Robot *iCub*

➤ *Raptop*

Desarrollado por el Instituto de Ciencia y Tecnología Avanzada de Korea (KAIST). Es un robot bípedo capaz de alcanzar la increíble velocidad de 46km/h superando la velocidad del actual campeón del mundo en los 100m lisos Usain Bolt en 43.9km/h. Una de las novedades es su estructura extremadamente ligera fabricada con materiales compuestos en la que gira una barra estabilizadora que lo hace capaz de saltar obstáculos sin desestabilizarse a 13.5km/h. Su forma está inspirada en los Velociraptores.¹¹

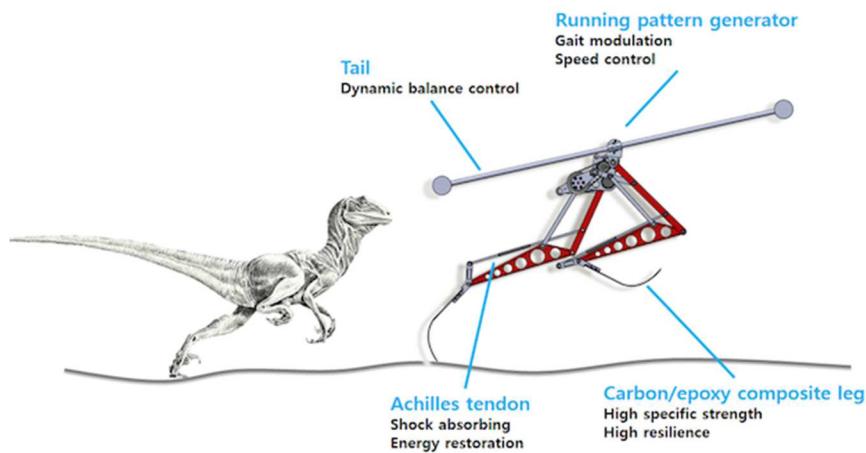


Ilustración 6. Diseño del Raptor



Ilustración 7. Raptor en los laboratorios del KAIST

HISTORIA Y ESTADO DE LA TÉCNICA EN LA IMPRESIÓN 3D

El proceso de impresión por deposición de material tiene su origen en los años 80, cuando varias compañías, registraron varias patentes. Todas ellas se basaban en la generación de piezas mediante archivos digitales, pero no fue hasta 1989 cuando se presentó la patente que explicaba el funcionamiento tal y como lo conocemos hoy en día. A lo largo de estos años estas impresoras han sido comercializadas por un precio de entre 5.000 y 15.000€

Al caducar esta patente en 2009, comienza la verdadera popularización de esta tecnología especialmente gracias al proyecto RepRap. El proyecto RepRap nace algo antes, en el 2004 en la universidad de Bath, Inglaterra. El objetivo era crear una máquina de prototipado rápido que pueda replicarse a sí misma fabricándose la mayoría de sus componentes mecánicos. Hoy en día, también es una plataforma de código abierto en la que la comunidad aporta continuas mejoras que mejorar la técnica de esta tecnología. La impresora líder en código abierto es la conocida como Prusa I3 (Diseñada por Josef Průša). Su código y piezas pueden ser descargadas desde RepRap.org



Ilustración 8. Prusa I3 MK2, uno de los últimos modelos de la Prusa I3 (a mayo de 2016)

<http://josefprusa.cz/>

Otro motivo de la popularización de la impresión 3D es su bajo precio. Una impresora construida por uno mismo con ayuda de las aportaciones de la comunidad puede costar menos de 300€.

Con la popularización de las impresoras 3D también se han lanzado al mercado numerosos modelos comerciales que no están basados en la filosofía “*Open Source*”. Algunas tienen con numerosas mejoras y diversos diseños, pero todas están basadas en el mismo funcionamiento. El coste de una impresora a septiembre de 2016 oscila entre los 350€ y 1.500€

El funcionamiento del modelado por deposición fundida consiste, primero en generar un archivo en código G a partir de un modelo tridimensional (habitualmente .STL). El archivo en .gcode contiene la información relativa a los movimientos y temperaturas que debe hacer la impresora para generar la pieza. Dos de los programas más utilizados para convertir los archivos. stl en .gcode son Repetier, relacionado con el proyecto RepRap, y Cura, gestionado por Ultimaker.

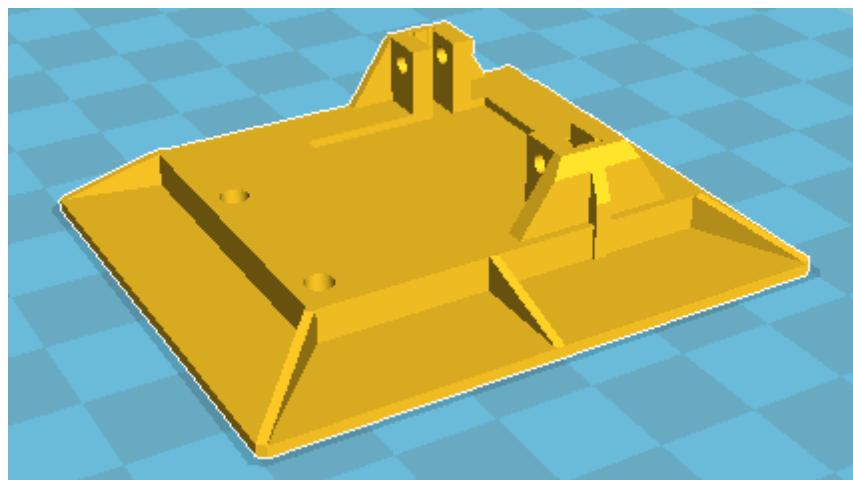


Ilustración 9. Archivo en formato .stl en el programa Cura.

```
G1 X73.395 Y84.632 E10.35706
G1 X73.476 Y84.573 E10.35862
G1 X73.277 Y84.551 E10.36174
G1 X73.166 Y84.529 E10.36350
G1 X73.037 Y84.496 E10.36558
```

Ilustración 10. Extracto de un archivo .gcode generado con el programa Cura.. G1 significa avance normal. X e Y son las coordenadas de la capa y E indica la posición a la que tiene que ir el extrusor para depositar el plástico.

Para generar la pieza, se calienta el plástico que se vaya a utilizar y se fuerza a través de una boquilla formando una capa que sirve de base para la siguiente. La primera capa se deposita sobre una superficie plana (habitualmente un espejo o cristal) a una temperatura superior a la temperatura de transición vítrea del plástico. De esta forma se fija a la base y se disminuyen las tensiones internas debidas a la contracción en el enfriamiento del plástico.¹²

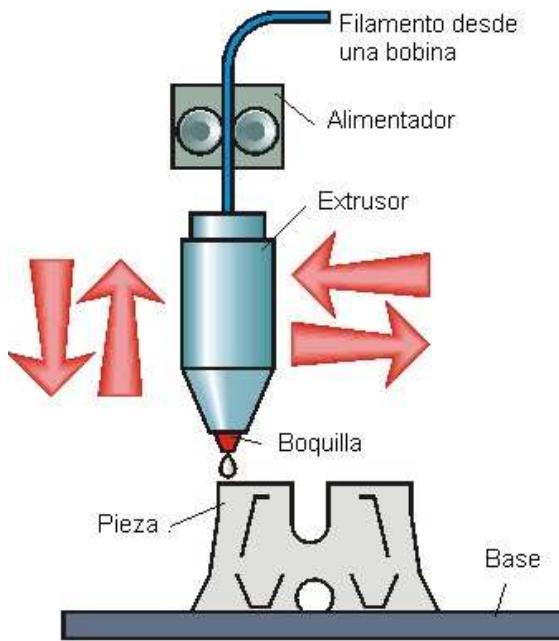


Ilustración 11. Esquema del funcionamiento en el modelado por deposición fundida
<http://tecnologiadelosplasticos.blogspot.com.es/2013/02/impresion-3d.html>

Durante la realización del proyecto se encontraron numerosos problemas de impresión, la mayoría de los cuales no se van a comentar. Estos problemas debidos a la impresión son simplemente fruto de emplear una tecnología de tan solo unos pocos años de técnica. A pesar de ello, el resultado final es satisfactorio.

CAPÍTULO 1. MECÁNICA DEL ROBOT

Como elemento fundamental y clave del proyecto se eligió la controladora Arduino Uno debido a su bajo costo, simplicidad y enormes aportaciones de la comunidad Arduino. Era sin duda el microcontrolador adecuado para el proyecto. Para mover las articulaciones se eligieron los servos SG90 9g, por tener el precio más competitivo del mercado y una relación potencia/peso adecuada para este trabajo. Para la elección de los grados de libertad se tuvo en cuenta que Arduino Uno consta de 13 salidas digitales, desde las que controlar los servos, pero en las especificaciones de la librería que se iba a utilizar figura que puede controlar un límite de 9 servos en Arduino Uno.

La estructura iba a ser impresa con una impresora 3D por deposición. La razón de ello es la versatilidad que ofrece a la hora de hacer el diseño, es posible obtener prácticamente cualquier pieza y eso era fundamental para este proyecto.

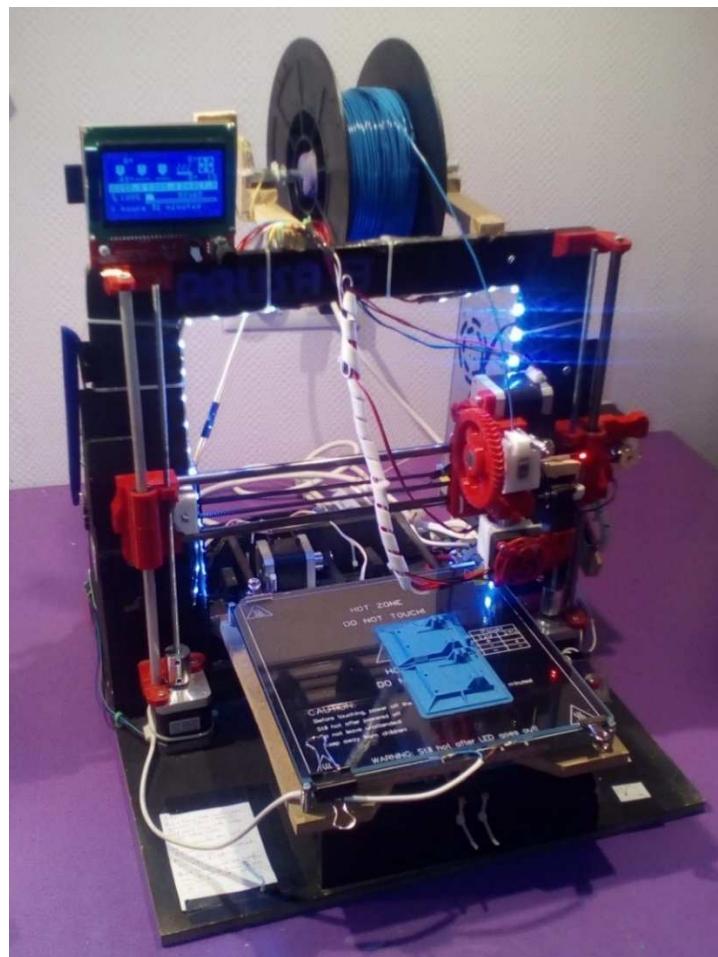


Ilustración 12. Impresora 3D casera basada en el modelo Prusa i3. Fue construida especialmente para hacer este proyecto. Las piezas que se ven impresas corresponden con los pies derecho e izquierdo del robot.

- ELECCIÓN DEL PLÁSTICO

En la actualidad existen diversos plásticos para la impresión 3D, también llamada modelado por deposición fundida. Los dos plásticos más comunes son el ABS y el PLA. El ABS (acrilonitrilo butadieno estireno) es un termoplástico plástico amorfó que se caracteriza por ser muy resistente a impactos. Es ampliamente utilizado en la industria del automóvil, la juguetería, para carcasa de televisores, ordenadores, etcétera. El PLA (ácido poliláctico) se obtiene del almidón, que a su vez procede de derivados vegetales como maíz, yuca, trigo, remolacha o caña de azúcar. Al no proceder de la transformación de hidrocarburos, tiene unas propiedades inusuales en los plásticos, como la biodegradabilidad. Por ello se suele emplear en la industria alimentaria como botellas de agua y refrescos, siempre y cuando no contengan bebidas carbonatadas.

En términos mecánicos, el ABS es más duro y resistente a impactos y el PLA más rígido y frágil¹³.

La principal ventaja que ofrece el PLA respecto al ABS es que es más fácil de utilizar en la impresión 3D, parámetro que debemos tener en cuenta cuando se va a utilizar una tecnología muy poco desarrollada. El *warping* es el principal problema que se encuentra en la impresión 3D. Sucede cuando se desprende una capa de la inferior por los vértices. Este fenómeno se produce por tensiones internas en la contracción del plástico depositado.



Ilustración 13. Fenómeno Warping. Se considera cuando se produce tanto en la base como en medio de la pieza
<http://www.trdimension.com/microblog/warping>

Imprimir en PLA también necesita mucha menos energía para funcionar. Con el ABS la cama se calienta a 100-110°C y con el PLA a tan solo 60°C, incluso con PLA es posible hacerlo en frío siempre que se use un adhesivo. Con PLA es en general, más fácil imprimir.

- DISEÑO

En cuanto a la forma de la estructura se ha tratado de ser lo más original posible. Hubiera sido muy fácil copiar el diseño de algún robot de juguete comercial. Todos tratan, en esencia, se imitar un desplazamiento antropomórfico. Hacer un droide, un robot caminante similar al humano, hubiera sido un error. Gobiernos y multinacionales de todo el mundo invierten grandes cantidades de dinero para investigar cómo andamos los humanos y cómo implementarlo en robots. Un TFG de este tipo está en otro nivel, no se trata de imitar a los humanos el objetivo es hacer un robot bípedo lo más original posible.

Por ello, el diseño elegido tiene la rodilla en la parte trasera del cuerpo, en lugar de delante como los humanos. La bipedestación de este proyecto es más parecida a los avestruces. El diseño inicial fue inspirado en el cine de ciencia ficción de la máquina de la siguiente imagen

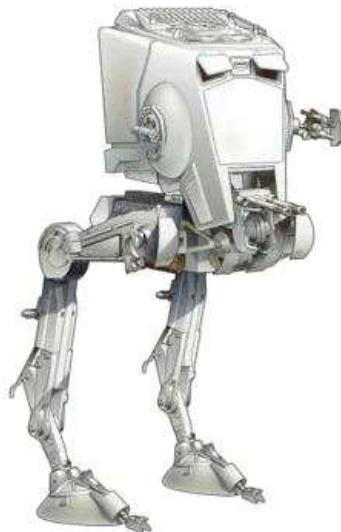


Ilustración 14. Máquina de guerra ficticia de la saga Star Wars

<http://es.starwars.wikia.com/>

- GRADOS DE LIBERTAD

La idea final respecto a las articulaciones y los grados de libertad consiste en un grado de libertad en cada pie para controlar el balanceo, otro grado en cada tobillo para mover el pie de arriba abajo, otro par de grados en la rodilla inversa para elevar o descender la espinilla y, finalmente, otro par en la cabeza orientados en el mismo eje que los dos pares anteriores. Según esto, los grados de libertad solo cubren dos ejes, concretamente los de balanceo y cabeceo. La razón por la que se ha hecho esto es que simplifica el diseño: todas las acciones que se barajaron complicaban enormemente la estructura, incrementaba la dificultad de la impresión por la complejidad de las piezas, no aseguraba que el funcionamiento fuera a ser mejor. Añadir grados de libertad al tercer eje no era necesario para que el robot avanzara y rodeara obstáculos.

El método que se iba a emplear empleado para girar consiste en desplazar las piernas en dirección opuesta (una hacia adelante y otra hacia detrás) para que las fuerzas de rozamiento, opuestas entre sí, generen un par torsor. Este es el mismo principio por el que giran los tanques.

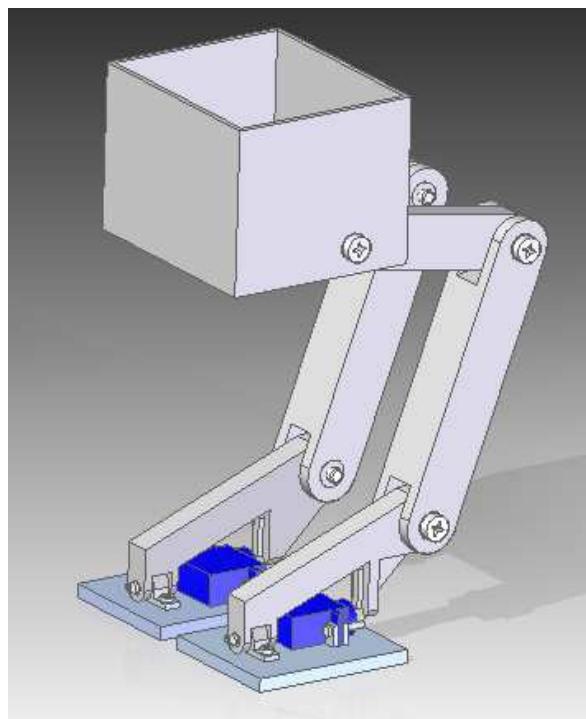


Ilustración 15. Modelo inicial

Respecto a la cara, se estudiaron diversas posibilidades para que el sensor de ultrasonidos pudiera detectar obstáculos en varias direcciones y hacer un mapeo básico de lo que tiene en frente, pero el reducido espacio de la cabeza impedía incluir un noveno servo con el sensor de ultrasonidos.

Se decide por tanto insertar simplemente el sensor en la parte frontal de la cabeza sin un servo que lo haga girar, al fin y al cabo, el mapeo no es imprescindible para detectar el obstáculo.

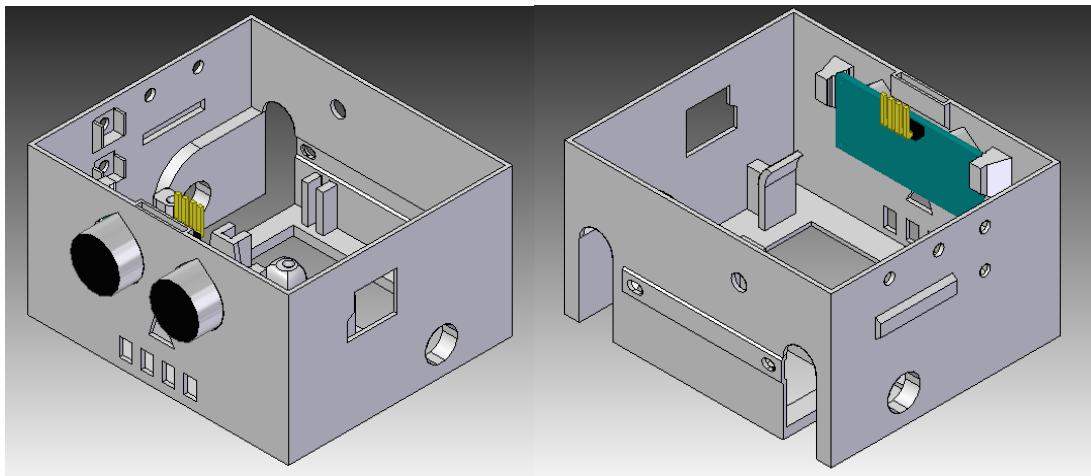


Ilustración 16. Diseño final del conjunto cabeza-ultrasonidos

- DISEÑO DE LOS ENGRANAJES

Se estimó el peso que podrían llegar a tener todos los componentes en 500g y se comprobó si los servos elegidos eran aptos para soportar esos momentos.

Pie, R1, R2 y Cabeza son las articulaciones tomadas en orden ascendente y se redondean al alza las medidas tomando así el caso más desfavorable:

$$Pie = 1.5\text{cm} \times 0.5\text{kgf} = 0.75 \text{ kgf} \times \text{cm}$$

$$R1 = 4.2\text{cm} \times 0.5\text{kgf} = 2.2 \text{ kgf} \times \text{cm}$$

$$R2 = 8.0\text{cm} \times 0.5\text{kgf} = 4 \text{ kgf} \times \text{cm}$$

$$Cabeza = 1.0\text{cm} \times 0.5\text{kgf} = 0.5 \text{ kgf} \times \text{cm}$$

La ilustración son dos vistas, alzado y perfil, obtenidas con Solid Edge para la posición que se esperaba tener en al inicio. No se barajan otras posibilidades porque en este momento no se conoce cuál va a ser el comportamiento final del robot. De todas formas, los valores se redondean al alza y el peso final es 360g, no 500g.

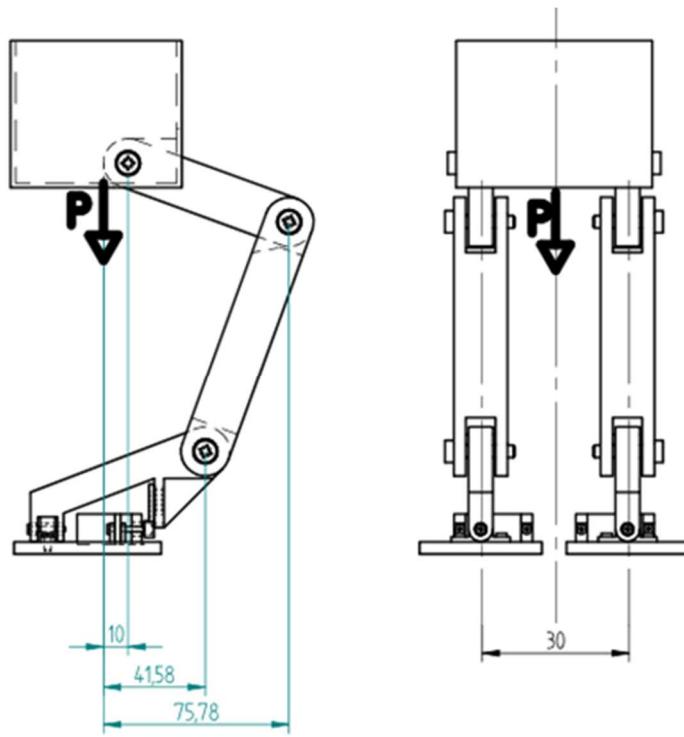


Ilustración 17. Representación del diseño preliminar del robot utilizado para calcular los esfuerzos de las articulaciones.

Según la hoja de especificaciones los servos tienen un par torsor de 1.8 kgf·cm por lo que los de la base del pie y los de la cabeza cumplen, pero los de la rodilla y el tobillo no. Se pensaron dos soluciones

1. Utilizar un mecanismo de 4 barras para aplicar una reducción.
2. Colocar ruedas dentadas en estas articulaciones.

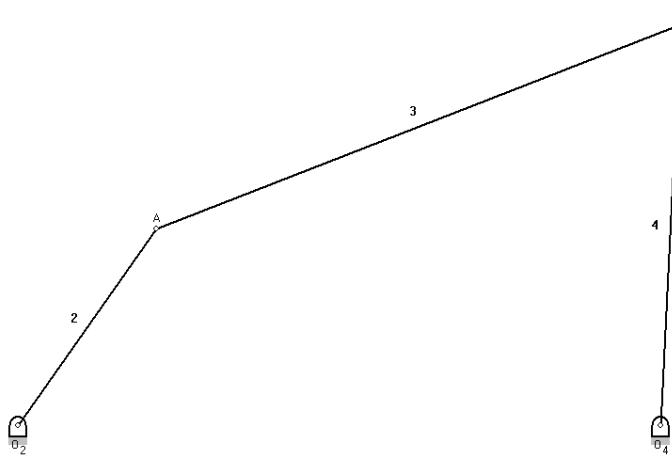


Ilustración 18. Mecanismo de 4 barras. Ilustración generada con el programa WinMec

El mecanismo de 4 barras hubiera sido una solución original a un problema de estas características. Como ventajas tiene ligereza, la originalidad. Hubiera sido una solución elegante para reducciones pequeñas como las que se dan en este problema. Además, un mecanismo de 4 barras no transmitiría el par de forma constante y podría estudiarse cómo aprovechar el par máximo en las situaciones más desfavorables.

Se estudió la posibilidad de imprimir pequeñas barras articuladas impresas con cojinetes de 1mm de diámetro interior y 3mm de diámetro exterior y utilizar como ejes alfileres. Otra opción sería la compra de rotulas de aeromodelismo empleadas en coches y helicópteros radiocontrol. La primera compleja y la segunda igual de compleja y cara. Además, se intentó obtener las ecuaciones de los pares torsores y la cinemática de las barras con el programa WinMec y se concluyó que el proyecto se iba a dilatar de forma innecesaria varias semanas.

La complejidad del diseño mecánico, los cálculos matemáticos y su posterior implementación en el *software* hicieron que la idea 1 terminara desechándose.

Las ruedas dentadas presentaban varios problemas. El tamaño de las ruedas debería de ser demasiado grande para la rueda mayor y/o demasiado pequeña la rueda menor. El tamaño de los dientes no debía quedar muy pequeño si iban a imprimirse con una impresora 3D. Los servos debían colocarse en una posición algo “ortopédica” para que entraran. La rebaba que deja el *brim* o camisa al imprimir hacia que no se respetara la geometría del diente en la primera capa y las ruedas saltaban sin engranar. Podía arreglarse “tallando” los dientes o recortando la rebaba con un cuchillo o *cutter*, pero, de esta forma, los resultados no eran los óptimos. Despues de semanas de técnica y al comprobar que la función *brim* no era imprescindible para trabajar con PLA se mejoró mucho la calidad de las ruedas.

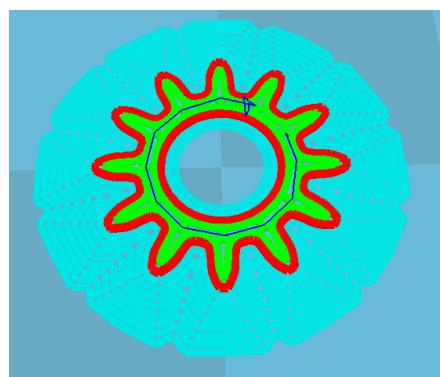


Ilustración 19. Simulación del brim mediante el Cura. El brim es utilizado para adherir los extremos de la pieza y evitar el “warping”

- Para R1 (tobillo)

$$\text{Reducción mínima } R1 = 2.2 \text{ kg} \cdot \text{cm} / 1.8 \text{ kg} \cdot \text{cm} = 1.22$$

Se toma como valor de cálculo: redu R1 = 1.4

- Para R2 (rodilla)

$$\text{Reducción mínima } R2 = 4.0 \text{ kg} \cdot \text{cm} / 1.8 \text{ kg} \cdot \text{cm} = 2.22$$

Se toma como valor de cálculo: redu R2= 2.4

Para los cálculos iniciales tomamos como número de dientes del piñón Z1=17, por ser el número mínimo para que no se produzca rebaje. El diámetro primitivo de la rueda dentada debe ser, aproximadamente, del valor de la anchura de la espinilla para no romper con la estética. Véase ilustración 20.

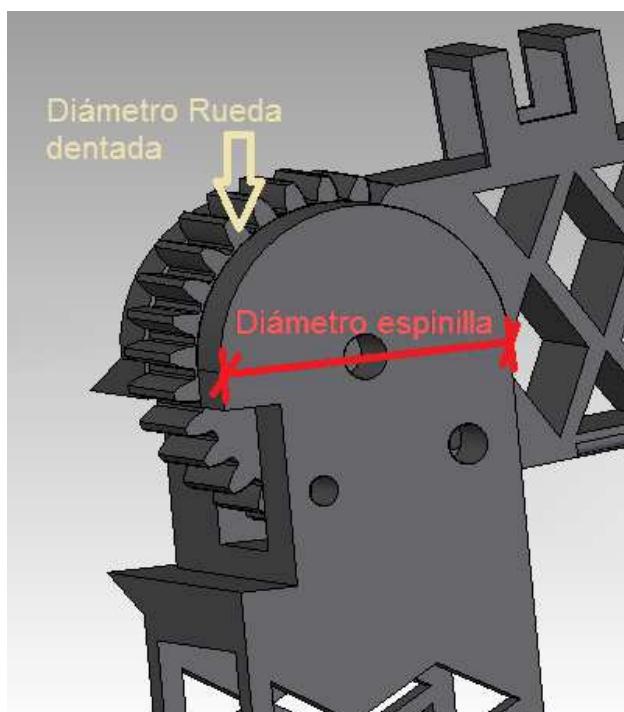


Ilustración 20. Unión espinilla-fémur

- Para R1:

Tomamos como diámetro exterior de la rueda conducida (Da) ≈ 20mm

$$Z2 = Z1 \cdot \text{redu R1} = 17 \cdot 1.4 = 23.8$$

$$Z2 = 24 \text{ dientes}$$

$$\text{redu R1} = \frac{Z2}{Z1} = 1.41$$

$$Dp = Da - 2 \cdot m$$

$$m = \frac{Dp}{z} = \frac{Da - 2 \cdot m}{z} = \frac{20 - 2 \cdot m}{24}$$

$$m = \frac{20}{26} = 0.769$$

Para tomar un valor redondo y estar del lado de la seguridad se toma m=0.8

Calculando el nuevo diámetro exterior:

$$Dp = m \cdot z = 0.8 \cdot 24 = 19.2$$

$$Da = Dp + 2 \cdot m = 20.6$$

- Para R2:

$$Z2 = Z1 \cdot \text{Redu R2} = 17 \cdot 2.4 = 40.8$$

$$Z2 = 41 \text{ dientes}$$

$$\text{redu R2} = \frac{Z2}{Z1} = \frac{41}{17} = 2.41$$

$$Dp = Da - 2 \cdot m$$

$$m = \frac{Dp}{z} = \frac{Da - 2 \cdot m}{z} = \frac{22 - 2 \cdot m}{41}$$

$$m = \frac{22}{43} = 0.512$$

El valor del módulo resultó demasiado pequeño para imprimirse con una boquilla de 0.4mm o incluso con una de 0.3mm.

La solución elegida para este problema fue reducir el número de dientes de la rueda pequeña, de 17 a 12, el mínimo aceptado por Solid Edge, y aceptar el rebaje. El rebaje es un fenómeno indeseable por socavar la base del diente (Vease ilustración 21) y reducir su resistencia. No obstante, la función *Engineering Reference* de SolidEdge con la que se diseñaron los engranajes

indicaba que no iba a haber problemas de esfuerzos al simular el movimiento con la velocidad máxima del servo de la hoja de especificaciones y ABS de densidad media como material. Es cierto que no han sido impresos en ABS, pero es un material cercano en propiedades y los esfuerzos que es capaz de soportar esta rueda son muy superiores al caso más desfavorable, con lo que no se esperaban problemas de esfuerzos por causa del rebaje y, de hecho, no ha habido.

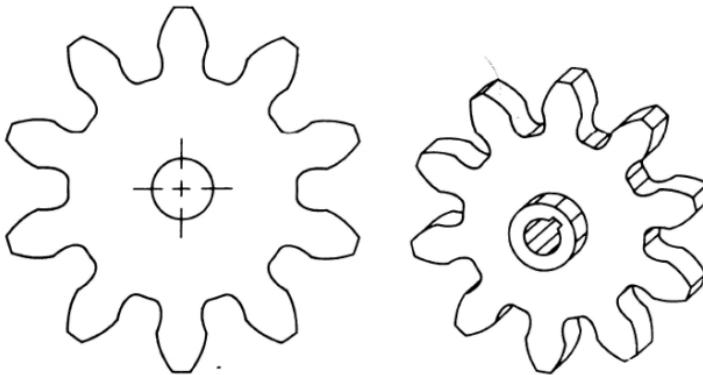


Ilustración 21. Engranaje recto con socavación para $\alpha=20$ y $Z=10$

Se calculó de nuevo la rueda R2 con los siguientes datos:

$$Z1=12$$

$$Da \text{ máximo} = 24 \text{ mm}$$

$$Z1 \cdot ReduR2 = 12 \cdot 2.4 = 28.4$$

$$Z2 = 29$$

$$Dp = Da - 2 \cdot m$$

$$Dp = m \cdot Z2 = 29 \cdot m$$

$$29 \cdot m = 24 - 2 \cdot m$$

$$m = \frac{24}{39} = 0.77$$

Se toma como módulo 0.75 porque la rueda puede sobresalir. Este diseño trata de hacer la rueda dentada en continuidad con las piezas de la que forma parte. Si la rueda es demasiado grande rompe con la estética tal y como se vio en la ilustración 20.

Los parámetros finales de la articulación R2 son:

$$Z1 = 12$$

$$Z2 = 29$$

$$redu\ R2 = 2.417$$

$$m\ R2 = 0.75$$

Más tarde, se comprobó experimentalmente que la reducción en la articulación R1 era insuficiente, el robot solo podía mover esa articulación si se colocaba manualmente en la posición de inicio. Por otra parte, se desarrolló una hoja de cálculo con *Microsoft Excel* para, entre otras cosas, ver los desplazamientos máximos y mínimos de las articulaciones (ver anexo 2.2). En el primer programa con el que el robot pudo caminar, la desviación máxima para la articulación R1 era de apenas 9° y los servos empleados tienen un barrido máximo de 180°. En una articulación con una reducción de 2.4, la articulación tendría un barrido máximo de 75°, cantidad más que suficiente que cumple con creces los requisitos. Visto esto, y visto que el sistema de engranajes en la articulación R2 funciona bien se decide poner la misma reducción con los mismos engranajes en ambas articulaciones. Los parámetros de entrada con los que se generan las piezas en la función Engineering Reference de Solid Edge (ver anexo 3).

- UNIÓN PIÑONES-SERVOS

Los brazos de los servos se acoplan a estos mediante unas pequeñas estrías situadas en el eje del servo y en el cubo del brazo. No obstante, esas estrías no son suficientes para impedir el deslizamiento cuando el par torsor es grande. Los servos incluyen tornillos autorroscantes para insertarlos en los ejes cuando el brazo está insertado y así dilatar el eje quedando atrapado dentro del cubo.

Se estudiaron la posibilidad de unir los piñones a los servos mediante el brazo que traen consigo.

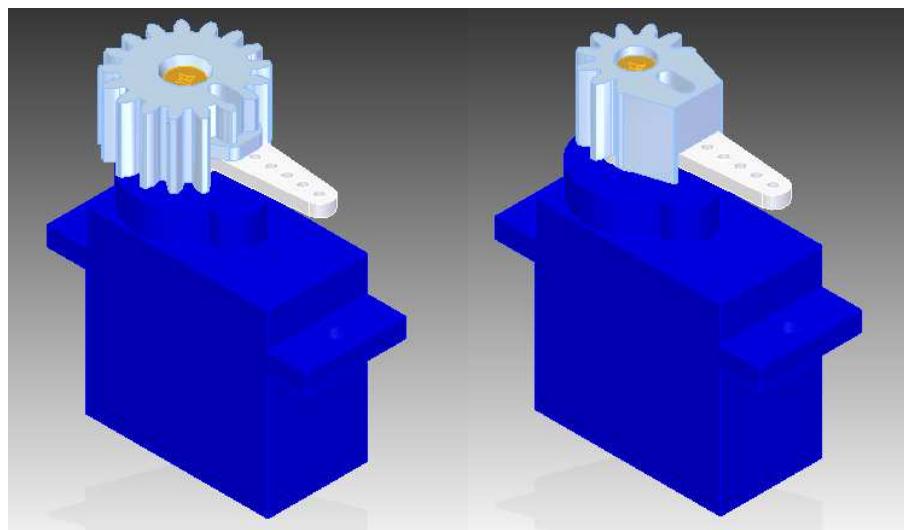


Ilustración 22. Diseño inicial del acoplamiento de los engranajes a los servos. A la izquierda, articulación R1 para reducción de 1.4. A la derecha articulación R2 para reducción 2.4

El resultado fue satisfactorio para el piñón de gran tamaño (que no se llegó a utilizar en el diseño final) pero el de la articulación con reducción 2.4 no se generaba correctamente en la impresora llegando a romperse con facilidad la parte superior o incluso no llegarse a imprimir. La razón residía en el pequeño espacio entre la base de los dientes y el alojamiento del brazo del servo.

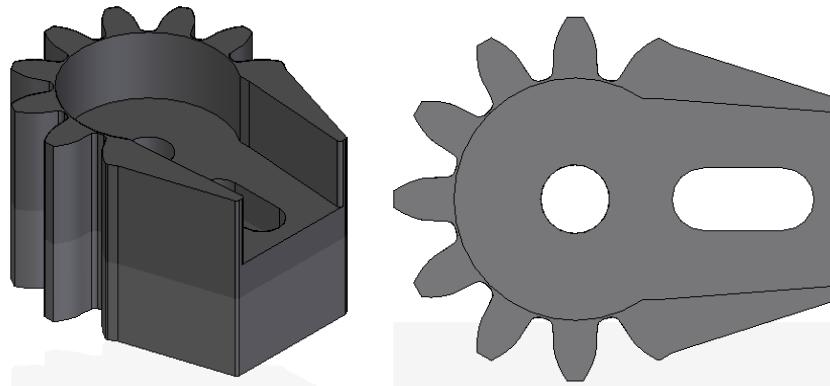


Ilustración 23. Primera versión del piñón para la articulación R2

La solución final fue quitar el brazo como medio de unión entre el servo y el piñón. El diseño final incluye unas estrías similares a las que incluye el brazo en el interior.

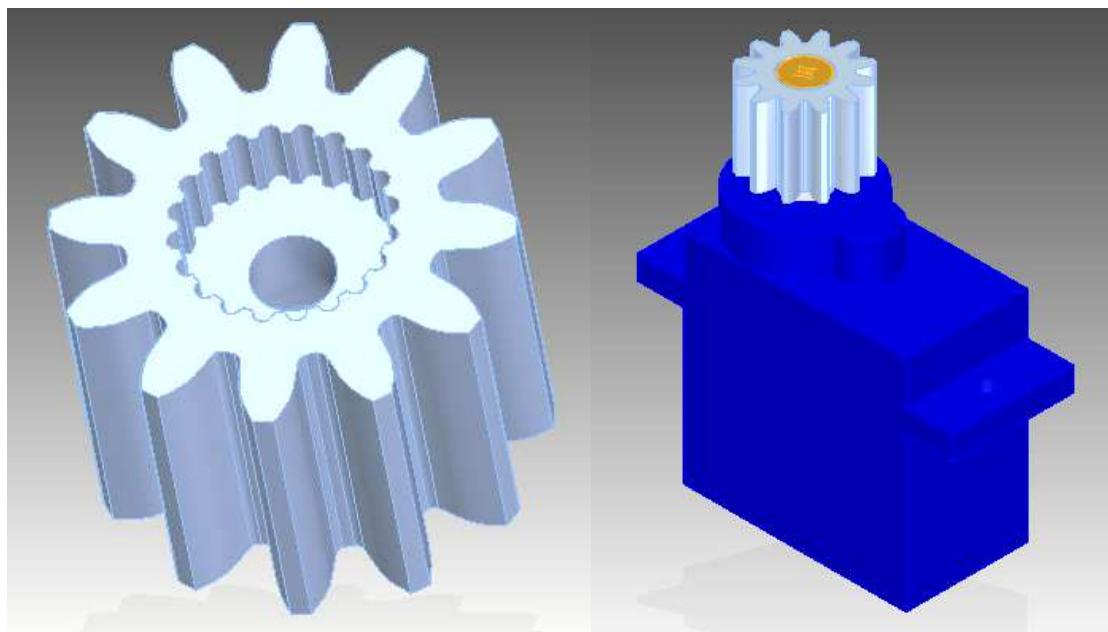


Ilustración 24. Solución final del enlace entre el piñón y el servo

- UNIÓN SERVO-FÉMUR

En el diseño final que se emplea en la cabeza un sistema parecido a la opción descartada en el caso anterior. El brazo del servo se introduce en un hueco en el que se inserta a presión para no generar holguras. Los agujeros que se observan en los laterales de la cabeza servían para fijar completamente la unión mediante un tornillo. No obstante, en la práctica se observa que no necesario reforzar esta unión al no soportar momentos importantes. No es necesario el empleo de tornillos ni otros elementos.

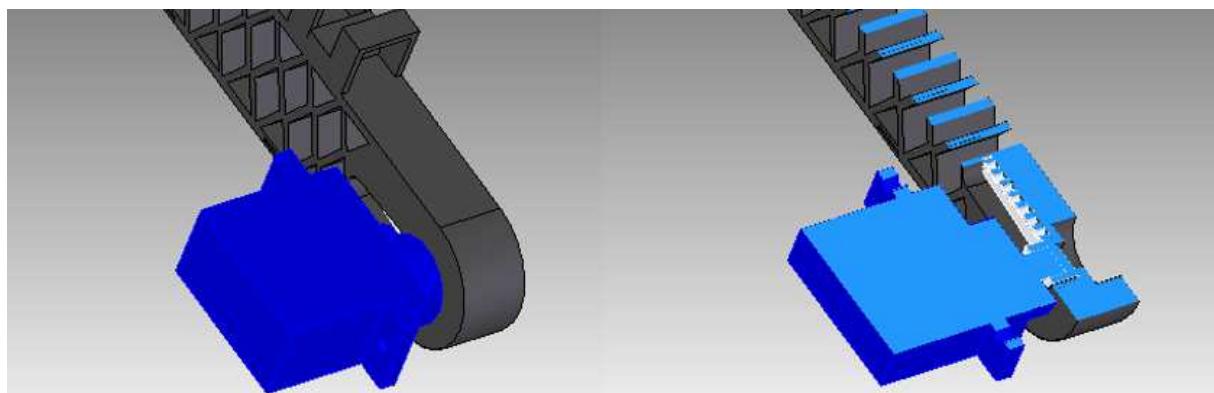


Ilustración 25. Articulación de la cabeza seccionada

- UNIÓN SERVO- EMPEINE

La articulación del pie funcionaba perfectamente introduciendo tornillos en los agujeros realizados para tal efecto pero, por estética, se decidió aplicar también este sistema.

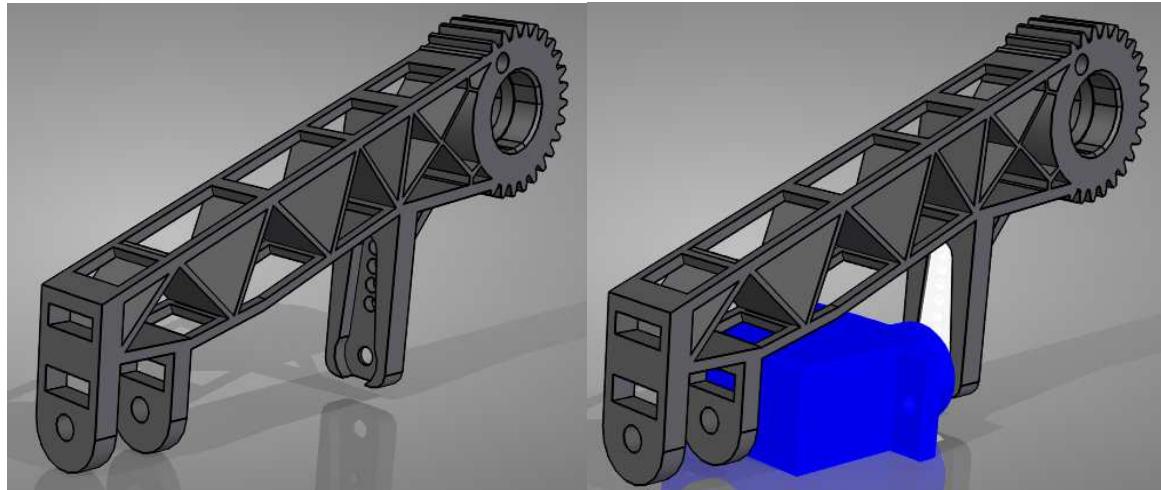


Ilustración 26. Detalle de la unión servo-empeine

- UNIÓN SERVO-ESPINILLA

Esta unión requiere de un sistema especial para que soporte las fuerzas radiales (Fr) que se dan en los engranajes.

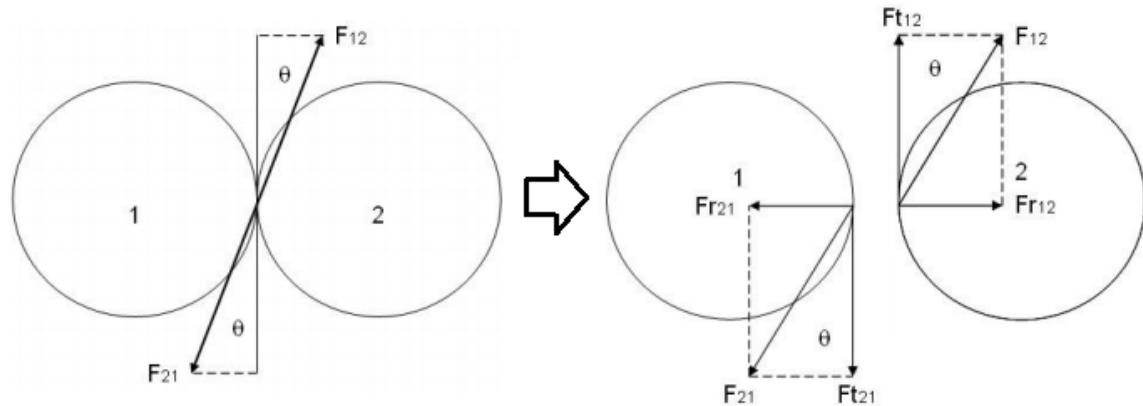


Ilustración 27. Descomposición de fuerzas en engranajes. Fr =fuerzas radiales. Ft =fuerzas tangenciales.

El servo se fija desde dos puntos diferentes:

1. El ala del servo es flexible (Véase ilustración 38) Tiene unos pequeños agujeros que, al introducir un tornillo, abren las aletas. El mecanismo diseñado emplea la fuerza de esa apertura para fijar el servo a la pieza e impedir que rote.

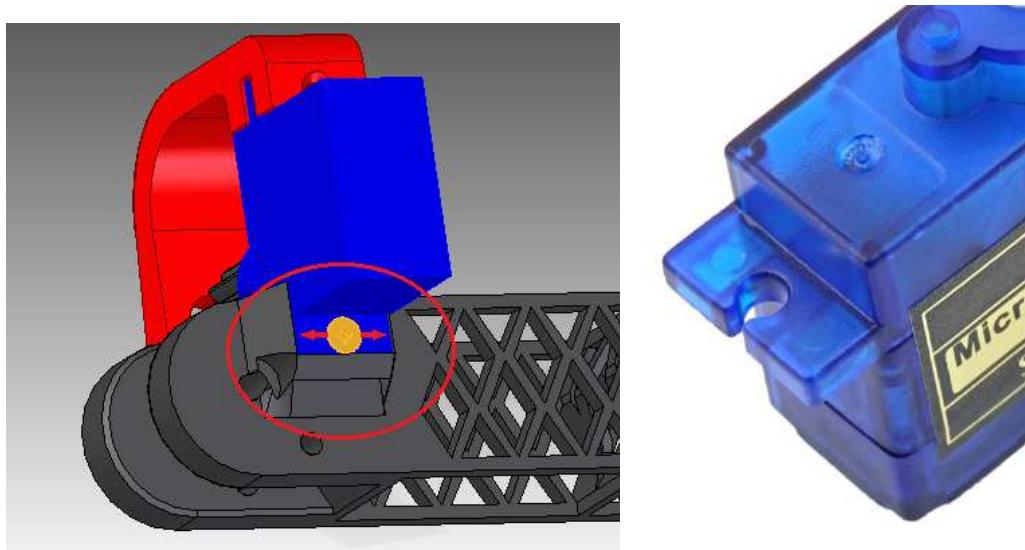


Ilustración 28. A la izquierda, detalle de la unión servo-espinilla. A la derecha, detalle del ala del servo

2. El sistema anterior no puede funcionar solo. Se experimentó que cuando el servo giraba, la fuerza radial anteriormente comentada, separaba las ruedas impidiendo la transmisión de movimiento. Por ello, se emplea la pieza “1.08 apoyos servos.par” (en rojo). Esta pieza obliga al servo a permanecer en posición y contrarresta la fuerza radial.

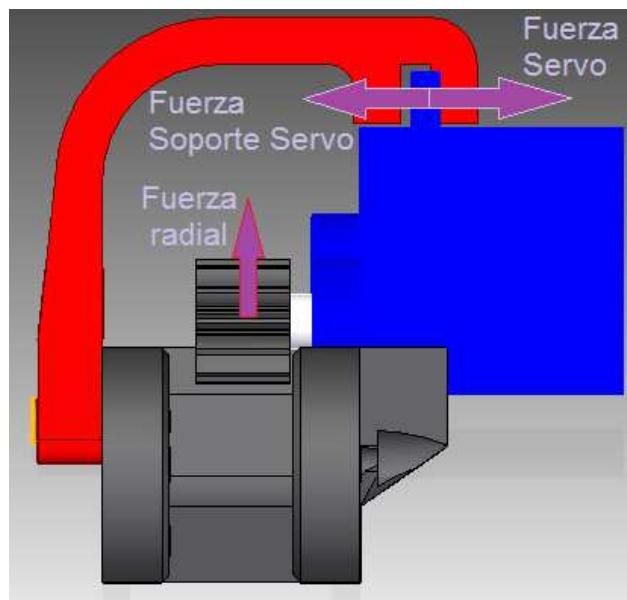


Ilustración 29. Detalle de fuerzas en la unión Servo-Espinilla

La “Fuerza radial” se transmite al servo y la “Fuerza Servo” es contrarrestada por la “Fuerza Soporte Servo” para evitar los desplazamientos indeseados.

La holgura que puede verse entre el ala superior del servo y el soporte del servo es necesaria para provocar una tracción que reduzca la fuerza radial al mínimo.

- UNIONES COJINETES-ESPINILLAS

Para fijar los cojinetes a las espinillas no se puede recurrir a los mecanismos tradicionales como los anillos de fijación. Las tolerancias de la impresora no lo permitirían. Tampoco podía introducirse el cojinete en el alojamiento circular simplemente a presión, de nuevo, porque las tolerancias de la impresora no lo permiten.

La solución debía aceptar que se está trabajando con una tecnología sin desarrollar y que no puede dar tolerancias como un torno ni una fresa. Después de muchos experimentos y ensayos se determinó que la solución más adecuada pasaba por no hacer el agujero estrictamente circular.

El diseño final es un agujero con unas marcas de medidas ligeramente menores. Este diseño consigue insertar con juego el cojinete en el círculo principal, pero con apriete en las marcas. (Ver ilustración 30). De esta forma se obtiene una solución elegante y discreta en la que se pueden insertar a presión con el dedo.



Ilustración 30. Diseño de los alojamientos de los cojinetes.

- UNIONES COJINETES-EJES

La idea original para esta articulación era emplear cojinetes de diámetro interior 3mm y tornillos de métrica 3 que se autorroscaran en plástico adyacente.

No obstante, el diámetro interno de los cojinetes resultó ser de entre 3.10 y 3.12, en función del cojinete que se midiera. Esta gran holgura hacía oscilar el eje y comprometía la estabilidad del robot.

La solución fue colocar dos cojinetes, uno en cada lado de la espinilla, de forma, que se reduce notablemente la oscilación. También se redujo el juego considerablemente entre la espinilla y las piezas adyacentes. Las piezas adyacentes (“Empeine” y “Fémur”) tienen un grosor de 10mm y el hueco de la espinilla pasó de 12 a 10.6mm.

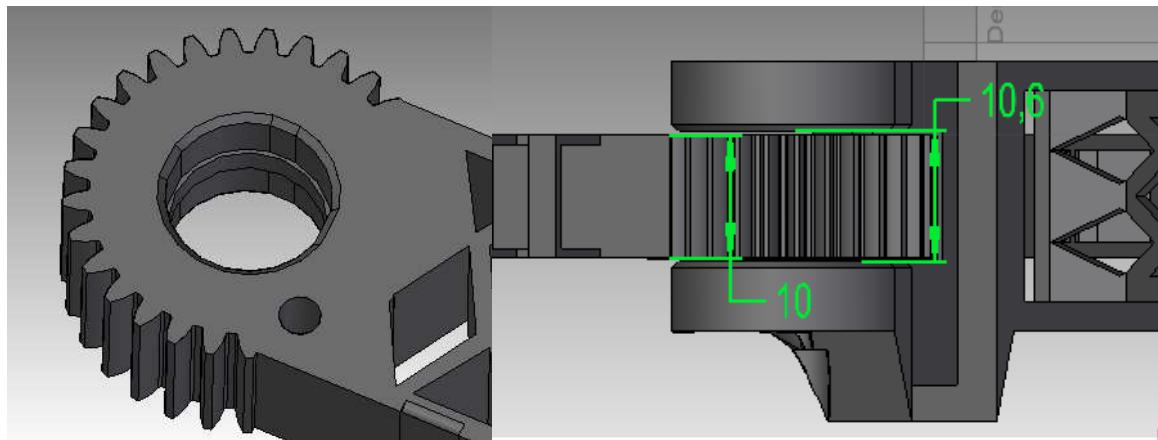


Ilustración 31. A la izquierda, solución para alojar a los cojinetes y distanciarlos entre sí para restar oscilación. A la derecha, detalle de cotas en la articulación espinilla-fémur

Esta solución elimina por completo las holguras en esta articulación.

- UNIÓN SERVO-PIE.

La idea inicial era insertar el servo en una huella para que entrara con un poco de presión y luego insertar dos tornillos autorroscantes de 2mm de diámetro en las aletas de los servos como se aprecia en la ilustración 32. En la práctica se vio que no eran necesarios los tornillos.

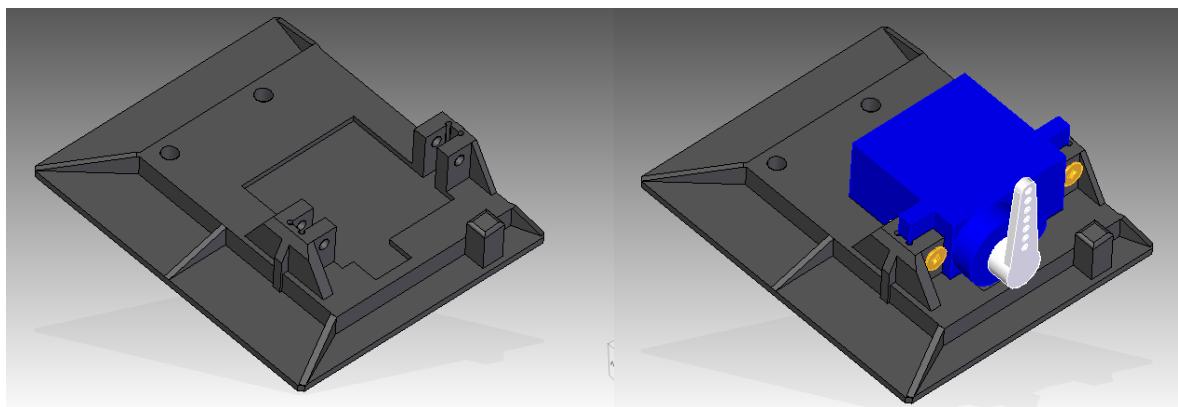


Ilustración 32. Unión entre el pie y el servo.

- UNIÓN SERVOS-CABEZA

En la cabeza se trató de implementar el mismo sistema, pero no fue posible fijar los servos con tornillos porque no se podía meter el destornillador.

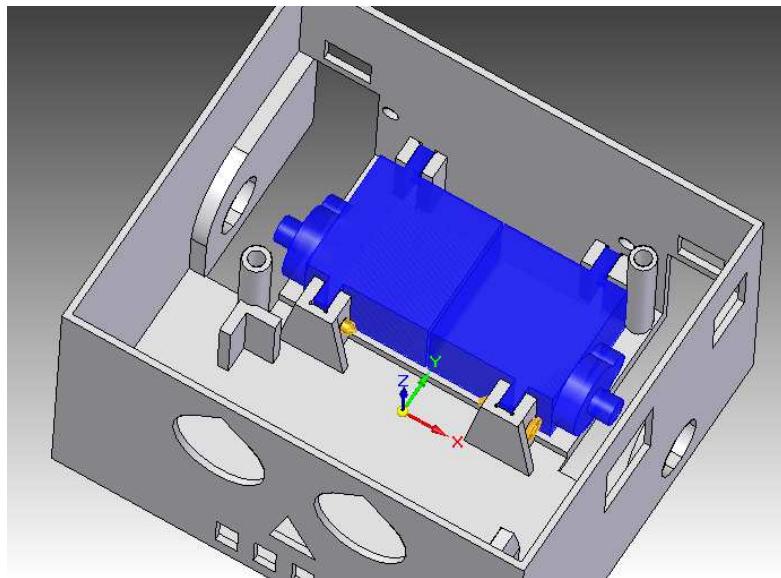


Ilustración 33. Primer diseño de la unión entre los servos y la cabeza. Nótese la dificultad para meter los tornillos.

Luego, se pensó que, si no hacían falta los tornillos en el pie, quizá tampoco hacían falta en la cabeza. Empíricamente se comprobó que esto no era cierto. En el caso del pie, la presión que lo sujetaba va en sentido favorable a la carga principal, mientras que, en la unión de la cabeza, la fuerza principal va hacia arriba y tiende a sacar los servos de su sitio.

La solución definitiva consiste en un tornillo en posición vertical con una arandela entre los dos servos para contrarrestar esa fuerza vertical y fijar definitivamente los servos.

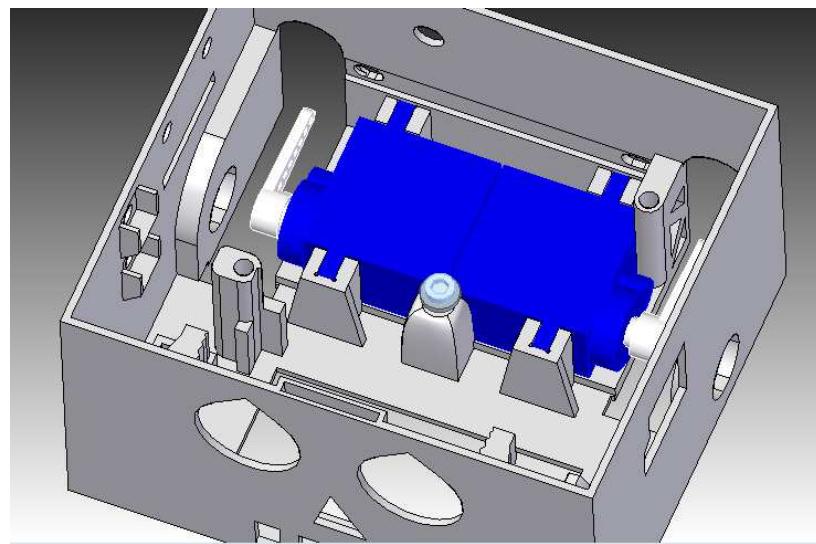


Ilustración 34. Solución definitiva para fijar los servos a la cabeza

- RETÍCULAS

Durante la construcción del robot se observaron largos tiempos de impresión y un gran consumo de material. Para acelerar el proceso, reducir peso, y mejorar ornamentalmente el robot, se aplicaron estructuras reticuladas a las piezas de las piernas, así como aberturas en la cabeza y nervios en los pies. El resultado no fue el esperado. Se ahorraba material y el robot quedaba mejor, pero apenas se ahorraba mucho tiempo en las barras reticuladas. Esto se debía a que la impresora imprime a diferentes velocidades el relleno interior y las paredes. La velocidad de relleno interior de la versión definitiva es de 70mm/s y la de la pared exterior es de 35mm/s. (véase anexo 4). Esto indica que, aunque haya que depositar menos plástico, también debe hacerse más lento, en el doble de tiempo. Las dos espinillas tardaban unas 6 horas en imprimirse sin retículas y 5.30h en imprimirse con retículas (con cabezal de 0.4mm y 40% de relleno). El ahorro de tiempo era positivo, pero no era el esperado. El ahorro en peso si fue muy positivo, sin retículas pesaba 47g y con retículas 27g.

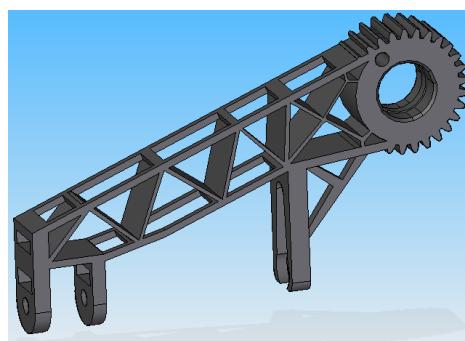


Ilustración 35. Pieza con retícula "1.04 EMPEINE"

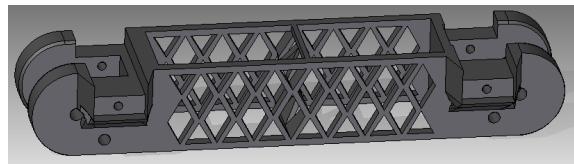


Ilustración 36. Pieza con retícula "1.05 ESPINILLA"

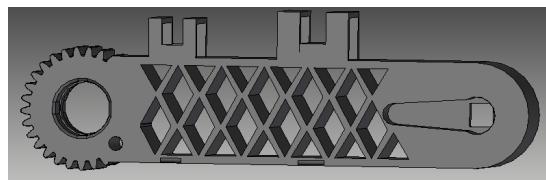


Ilustración 37. Pieza con retícula "1.06 FEMUR"

- DISEÑO FINAL

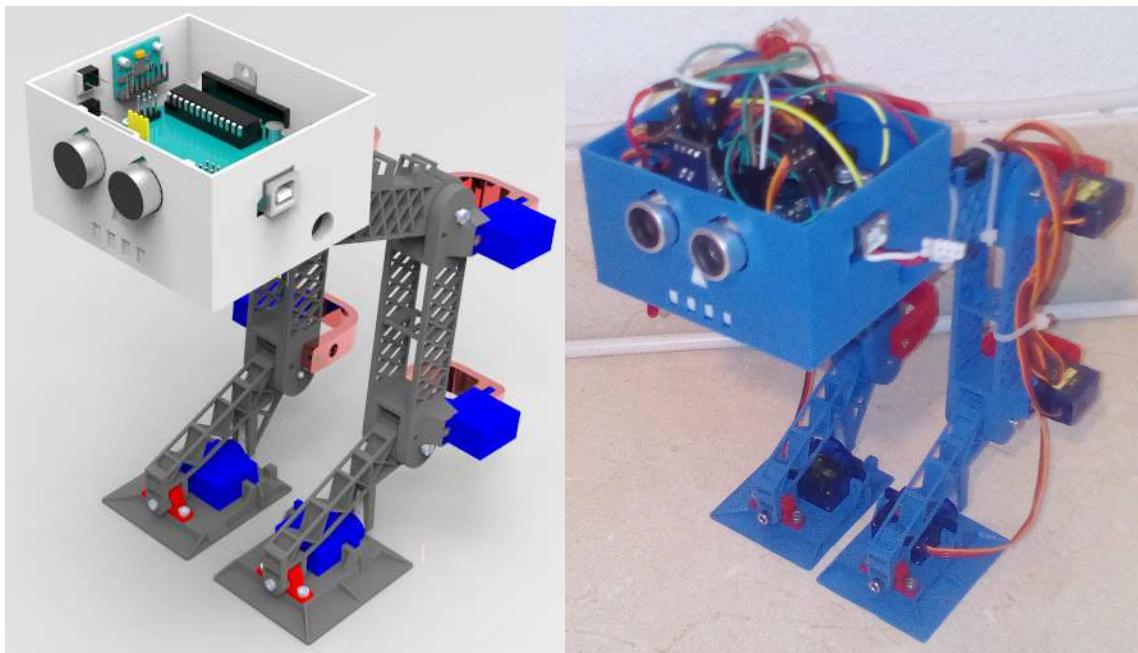


Ilustración 38. A la izquierda, diseño CAD final. A la derecha, el robot real

CAPÍTULO 2. ELECTRÓNICA

Para el proyecto se necesitaron los siguientes componentes:

1. Una controladora Arduino Uno
2. Una batería Floureon® 7.4V 1000mAh 2S 20C Lipo.
3. Ocho SG90 9g Microservos
4. Interruptor de 3 pines
5. Sensor de ultrasonidos HC-SR04

6. 9x2 zócalo de cabezal hembra
7. x2 conectores macho
8. Regulador de tensión LM7805
9. Pinza de refrigeración
10. *Protoboard* de circuito impreso
11. Módulo Led RGB
12. IMU → GY-521: Giróscopo con acelerómetro MPU 6050
13. Cables varios

- ARDUINO

Arduino es una plataforma electrónica de código abierto que funciona sobre un IDE (Entorno de Desarrollo Integrado) basado en *Processing* y un lenguaje basado en *Wiring*. La compañía Arduino fabrica las placas y da soporte a la comunidad Arduino desde la web oficial arduino.cc, donde entre otras muchas cosas, se pueden encontrar los planos de las placas que venden. Es por esto por lo que es una plataforma que se basa en la filosofía *open-source* o de código abierto.¹⁴

La mayoría de las placas Arduino llevan como procesador un ATmega donde se carga el código que luego se ejecuta.

Aunque en la página oficial de Arduino pueda encontrarse una bastante información, lo que hace la hace tan popular es la enorme comunidad de usuarios. Cientos de páginas, foros y revistas tratan multitud de temas y constituyen la clave del éxito de Arduino



Ilustración 39. Logo de Arduino

<https://www.arduino.cc/>

Es importante para el proyecto determinar cómo queremos alimentar cada componente. Arduino UNO es capaz de alimentarse de tres formas:

- Mediante el USB del ordenador
- Mediante un conector Jack 5.5 x 2.1mm como el de la **ilustración**

- Mediante los puertos V_{in} y GND representados en la **ilustración**. Opción más compacta por la cual es elegida.



Ilustración 40. Conector Jack macho 5.5 x 2.1mm

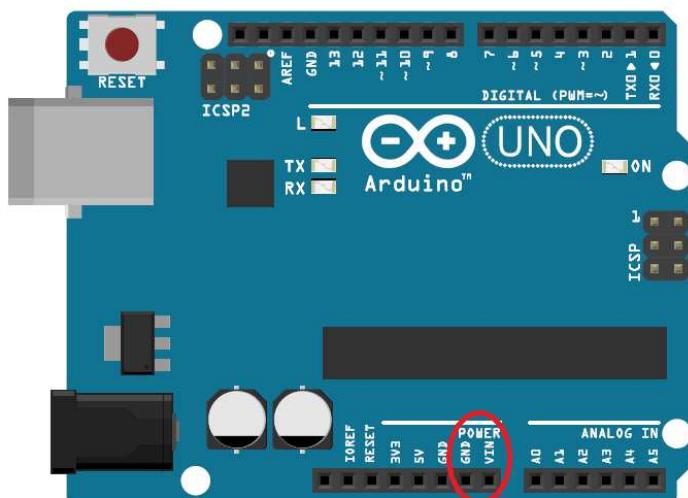


Ilustración 41. Placa Arduino UNO. Rodeado en rojo los pines empleados para alimentación

<https://www.arduino.cc/>

- BATERÍA

Dado que la hoja de especificaciones de Arduino indica que los valores recomendados para la alimentación oscilan entre 7 y 12V y los servos debían alimentarse a 5V se eligió la batería más económica y ligera que suministrara al menos 7V. La batería de litio elegida, propia de aparatos radiocontrol, parecía la más idónea para este proyecto. No obstante, debe leerse con precaución las instrucciones que vienen asociadas. En ellas se indica que deben almacenarse a $20^{\circ}\text{C} \pm 5^{\circ}\text{C}$, no debiendo superar durante el almacenaje los 30°C . Esto es un dato importante para evitar transportes innecesarios, por ejemplo, en coche en agosto. Para el proyecto se compraron un total de tres baterías: una se sobrecalentó perdiendo todas sus características, otra está en uso y la tercera de repuesto.



Ilustración 42. Batería Floureon 7.4V 1000mAh

- SERVOS

Los servos son dispositivos electromecánicos que pueden alcanzar una posición determinada. El funcionamiento es el siguiente:

Un circuito comparador que recibe la señal PWM. La señal, en función de su ancho de onda, determina el número de grados a los que debe girar el servo. Un circuito comparador en el interior del servo compara la posición a la que debe ir con la posición que lee del potenciómetro conectado mecánicamente al eje de salida. De esta forma decide si girar el motor CC hacia un sentido, hacia otro o no moverse. Este motor va ligado a una reductora y trae varios brazos para acoplar al eje de salida.

El servo elegido es el modelo SG90 9g. Es un servo muy pequeño y muy ligero. Tiene una maniobra de 180° y se alimenta a 5V. Es un servo muy testado por la comunidad Arduino. Su sencillez y precio ha conseguido que sean los más utilizados en fines didácticos.



Ilustración 43. Servo SG90 9g

Para controlar el servo hay que mandar una señal PWM (*pulse-width modulation* o modulación por ancho de pulsos). Se trata de modular el pulso indicarle la posición que debe alcanzar. Según la hoja de especificaciones este pulso debe oscilar entre 1 y 2 milisegundos, siendo 1ms

para la posición 0° y 2ms para la posición 180°. Para el resto de posiciones tan solo hay que interpolar linealmente. Esta señal cuadrada hay que emitirla cada 20ms (50Hz).

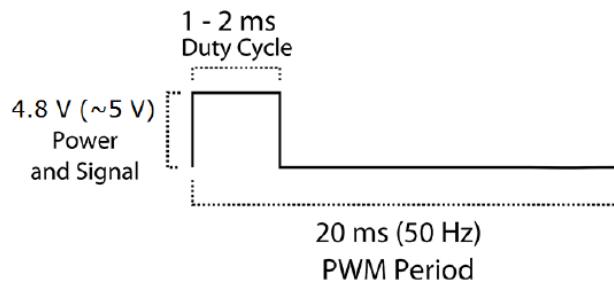


Ilustración 44. Señal PWM extraída de la hoja de especificaciones de los servos empleados

Para hacer todo el proceso y simplificar el código recurrimos a la librería Servo.h.

- INTERRUPTOR

Interruptor con tres patillas: una común, otra normalmente cerrada y otra normalmente abierta. Solo se usa la común y la normalmente abierta. Se conecta entre el positivo de la batería y el positivo del resto de dispositivos

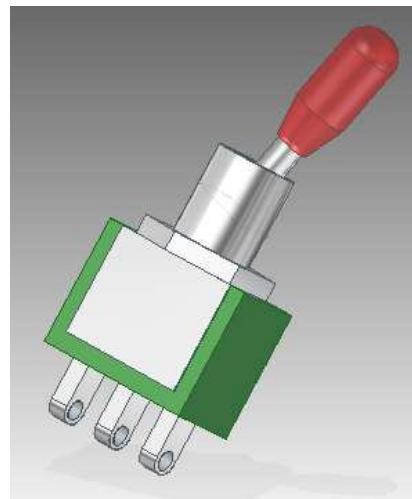


Ilustración 45. Representación CAD del interruptor

- SENSOR DE ULTRASONIDOS

Modelo HC-SR04. Este sensor es capaz de emitir ultrasonidos y medir el tiempo que tardan en volver para calcular la distancia entre el sensor. El rango de medición oscila entre 2cm y 4

metros para una precisión de 3mm. Es muy utilizado en proyectos de robótica. Se debe tener en cuenta que hay varios factores que limitan su uso:

- Tiene un ángulo de medida de 15°
- Cuando el objeto a detectar es un plano con la normal a más de 45° frente al sensor podría no detectarlo. Si el objeto es especialmente liso, como un cristal, este ángulo es menor.
- Los materiales con una gran capacidad de absorber el sonido (como el algodón) pueden resultar invisibles a este sensor



Ilustración 46. HC-SR04. Sensor de distancia por ultrasonidos

Para medir la distancia se multiplica la velocidad del sonido por el tiempo que tarda en llegar el eco y se divide entre dos.

$$\text{Distancia} = \text{tiempo} \cdot \text{velocidad}/2$$

En el código (anexo 5.3):

$$\text{distanciaEnCm} = \text{duración en microsegundos } / 58$$

Donde 58 es un valor que viene en la hoja de especificaciones para obtener la distancia en centímetros. A continuación, se desglosa este valor:

1/58 es la velocidad del sonido en el aire a 20°C con una humedad relativa del 50% en cm/μs entre dos:

Pasando la velocidad del sonido en metros por segundo a cm/μs:

$$343 \frac{\text{m}}{\text{s}} \cdot 100 \frac{\text{cm}}{\text{m}} \cdot \frac{1}{10^6} \frac{\text{s}}{\mu\text{s}} = \frac{1}{29} \frac{\text{cm}}{\mu\text{s}}$$

Qué dividido entre dos corresponde con 1/58

- CIRCUITO DE ALIMENTACIÓN DE LOS SERVOS

Para alimentar los servos se necesitaban más pines de alimentación de lo que Arduino admite.

Se hizo un pequeño circuito con los siguientes componentes:

- 9x2 conectores de cabezal hembra
- x2 conectores de cabezal macho
- Regulador de tensión LM7805
- *Protoboard* de circuito impreso

El circuito se conecta a la batería. Se alimenta mediante dos conectores macho. El LM7805 estabiliza la tensión de la batería a 5V que llega a los cabezales hembra, donde se conectan los cables que alimentan los servos.

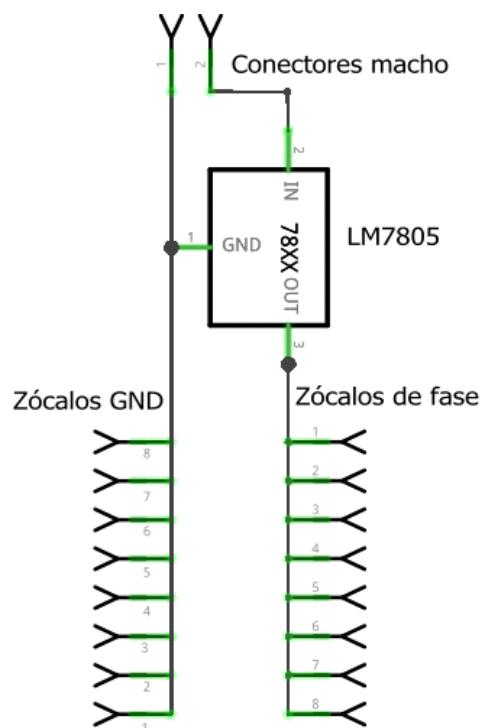


Ilustración 47. Esquema del circuito de alimentación de servos

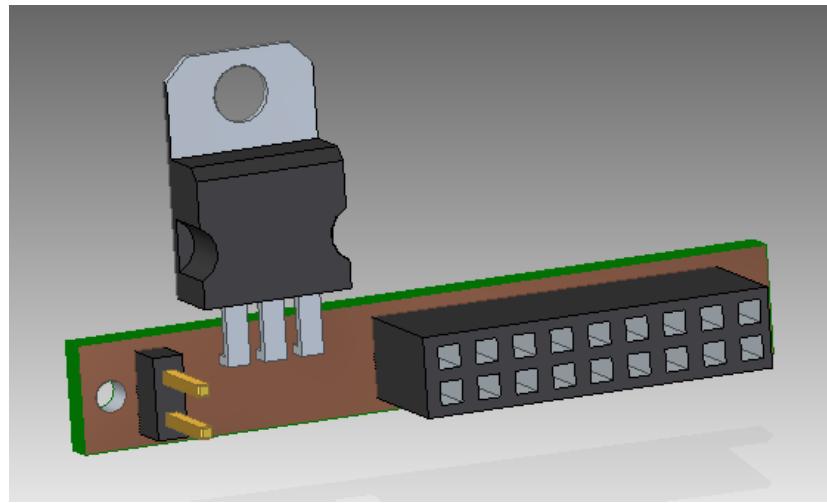


Ilustración 48. Representación CAD del circuito alimentador de servos

El LM7805 es un regulador de tensión a 5V. Desde un inicio, se comprobó la elevada temperatura que cogía, llegando incluso a provocar leves quemaduras en los dedos. En un principio, se insertó un tornillo en el agujero con tuercas y arandelas alternadas, pero ornamentalmente no era una solución satisfactoria, ya que quedaba por el exterior de la cabeza. Se optó por acoplarle una pinza de aluminio para su refrigeración.



Ilustración 49. Fotografía de la pinza acoplada al regulador de tensión

Según la hoja de especificaciones el LM7805 es capaz de soportar una intensidad de 1A. El elevado calentamiento invita a calcular si se está forzando demasiado el circuito. Para comprobarlo, se carga la batería al máximo y hace el robot andar hasta descargarlo por completo para medir la autonomía. Se toman 65 minutos de autonomía. Recordando que la batería tiene

1000mAh y la autonomía es de poco más de una hora, parece que por el regulador de tensión está circula de media poco menos de un amperio, muy cerca del límite. Se calcula cuánto:

Dado que Arduino, el sensor de ultrasonidos y el giroscopio, según sus respectivas hojas de especificaciones, consumen un total de 65mA:

Se calculan cuantos mAh consumen estos dispositivos en 65min:

$$65mA \cdot 1.08h = 70.2mAh$$

Los mAh que consumen los servos serán igual a:

$$1000mAh - 70.2mAh = 929.8mAh$$

La intensidad media que pasa por el regulador de tensión será:

$$\frac{929.8mAh}{1.08h} = 860.9mA$$

Se concluye que el LM7805 funciona al 86% de su capacidad nominal. Es un valor elevado pero que no corre peligro; se debe tener en cuenta que si se desea añadir otro servo o conectar algún otro dispositivo al circuito de alimentación. Se vería muy cerca del límite.

- LED RGB

Los diodos son componentes electrónicos que permiten el paso de la corriente en un solo sentido. Los LEDs (light-emitting diode) son diodos capaces de producir luz cuando pasa corriente. El principio físico del LED es el Efecto Fotoeléctrico que consiste en la emisión de electrones por un material al incidir sobre él una radiación electromagnética. De modo inverso, el paso de electrones a través por un conductor (o semiconductor en este caso) provoca una radiación electromagnética. En función del semiconductor que encontramos dentro del LED la radiación tendrá una longitud de onda u otra.



Ilustración 50. Diodo Led común

Los RGB (Red Green Blue) son simplemente varios diodos LED con diferentes semiconductores embebidos en la misma cápsula. Combinando estos colores es posible obtener cualquier color.

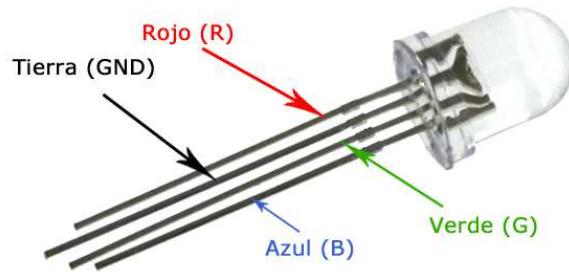


Ilustración 51. Diodo Led RGB

Los diodos Led deben protegerse mediante resistencias para evitar que una intensidad elevada lo queme. Lo común es emplear resistencias de 220Ω o 330Ω .

El elemento comercial que se ha incluido en este proyecto es un módulo Led RGB que consiste en una placa de circuito impreso con un RGB protegido por resistencias. Se ha elegido esta opción por ser la más fácil de manejar.



Ilustración 52. Módulo RGB Led empleado en el proyecto.

A falta de pines, en este proyecto solamente se han conectado el verde y el rojo. El verde para cuando no detecta un obstáculo y el rojo para cuando lo detecta. No obstante, se han incluido rutinas para obtener seis colores si se conectara también el azul (blanco, rojo, verde, azul, morado y cian)

- IMU (INERTIAL MEASUREMENT UNIT)

El aparato con el que se va a conocer la posición del robot es un IMU MPU6050 sobre una GY-521. Expliquemos conceptos:

GY-521 es la *breakboard* desarrollada por Fritzing¹⁵ donde se acopla el IMU. El IMU son el conjunto de sensores capaz de registrar aceleraciones lineales y velocidades angulares, comúnmente conocido como acelerómetro-giróscopo. El MPU 6050 es un IMU desarrollado por InvenSense, consta de un acelerómetro y un giróscopo de tres ejes cada uno. Con este dispositivo podemos conocer el balanceo y el cabeceo, aunque no la guiñada. La guiñada sería posible conocerla con un magnetómetro. El IMU empleado trae la posibilidad de conectarle uno mediante los pines XDA y XCL. Así se podría orientar correctamente la guiñada y enviar todas las señales por el bus I2C, mediante los pines SCL y SDA. También se podría recurrir al hermano mayor, el MPU9150, con acelerómetro, giróscopo y magnetómetro. Los pines restantes ADO e INT se pueden utilizar para establecer una comunicación mediante el protocolo de comunicaciones SPI¹⁶.

En este proyecto solo se utilizan los pines SCL y SDA para la comunicación I2C con Arduino y los pines de alimentación Vcc y GND a 3.3V



Ilustración 53. IMU MPU 5060 sobre breakboard GY521

➤ Protocolo I2C

Elegimos este modelo por su bajo coste y tener plena compatibilidad con Arduino. Este IMU utiliza un protocolo de comunicaciones I2C (*Inter-Integrated Circuit*). Arduino Uno también cuenta con un bus de comunicaciones I2C que coincide con las entradas analógicas A4 y A5. El I2C precisa de dos líneas de señal: reloj (CLK, *Serial Clock*) y la línea de datos (SDA, *Serial Data*). Este protocolo recibe varios nombres: I2C, IIC, I²C, TWI (*Two Wire Interface*)¹⁷

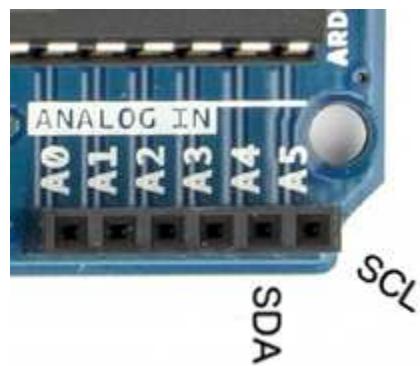


Ilustración 54. El bus I2C de Arduino Uno corresponde con la entrada analógica A4 para trasmisir datos y el A5 para reloj

Controlar el protocolo de comunicaciones I2C mediante código propio es complejo y tedioso. En este proyecto se utiliza la librería que emplea mayoritariamente la comunidad Arduino “Wire.h”.

➤ *Sistemas MEMS (MicroElectroMechanical Systems)*

Este dispositivo ha sido construido gracias a la tecnología MEMS. Los sistemas MEMS logran reducir componentes eléctricos y mecánicos a apenas unos pocos micrómetros. La electrónica de consumo ha disparado su uso en los últimos años especialmente en el mercado de los *smartphones* y *tablets*, con acelerómetros y giróscopos de los cuales se habla a continuación.

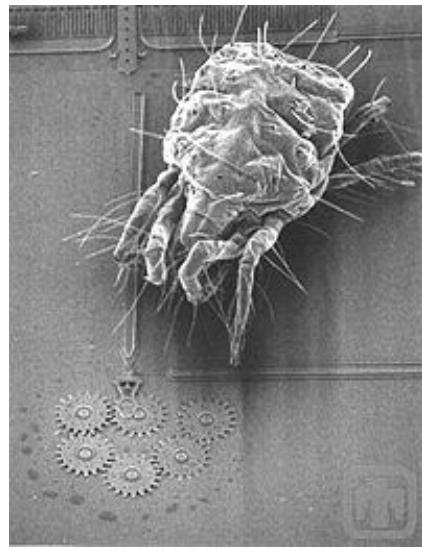


Ilustración 55. Comparativa entre un ácaro y unas piezas MEMS¹⁸

➤ *Acelerómetro*

Un sistema muy común es aquel que se basa en el efecto piezoeléctrico. Los materiales piezoeléctricos, como el cuarzo, tienen la propiedad de poder convertir la tensión mecánica en electricidad.¹⁹ Consta de un material piezoeléctrico de masa conocida que se somete a una aceleración, se comprime y genera una alteración en las cargas. Se genera así una tensión eléctrica proporcional al esfuerzo del material. Conocida la masa y mediante la segunda ley de Newton $F=m \cdot a$ es posible conocer la aceleración a la que ha sido sometido el sistema.

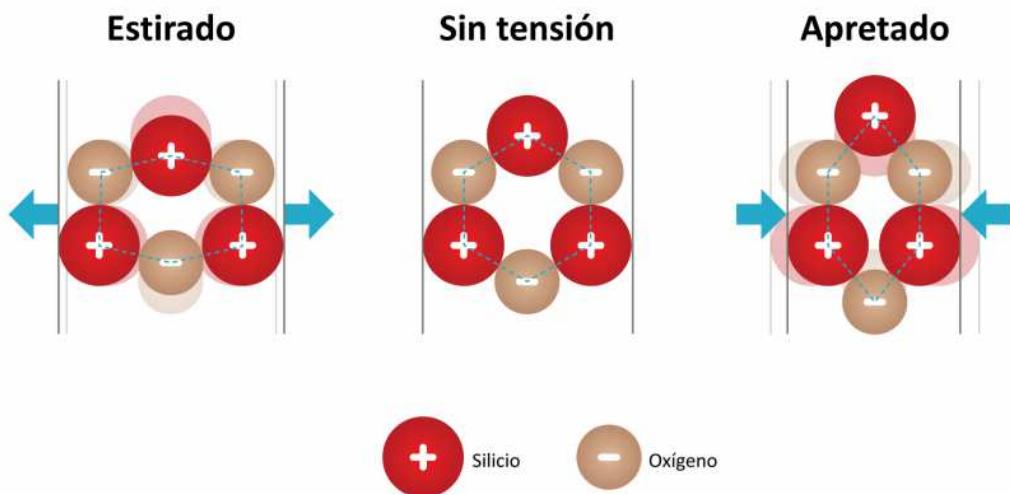


Ilustración 56. Efecto piezoelectrónico en el cuarzo. Principio físico en el que se basan los acelerómetros
http://www.nisenet.org/sites/default/files/catalog/uploads/spanish/12194/electricsqueeze_images_13nov13_sp.pdf

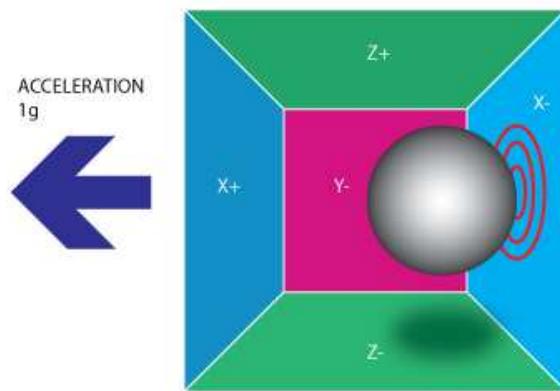


Ilustración 57. Esquema de funcionamiento de un acelerómetro basado en el efecto piezoelectrónico
<http://diyhacking.com/arduino-mpu-6050-imu-sensor-tutorial>

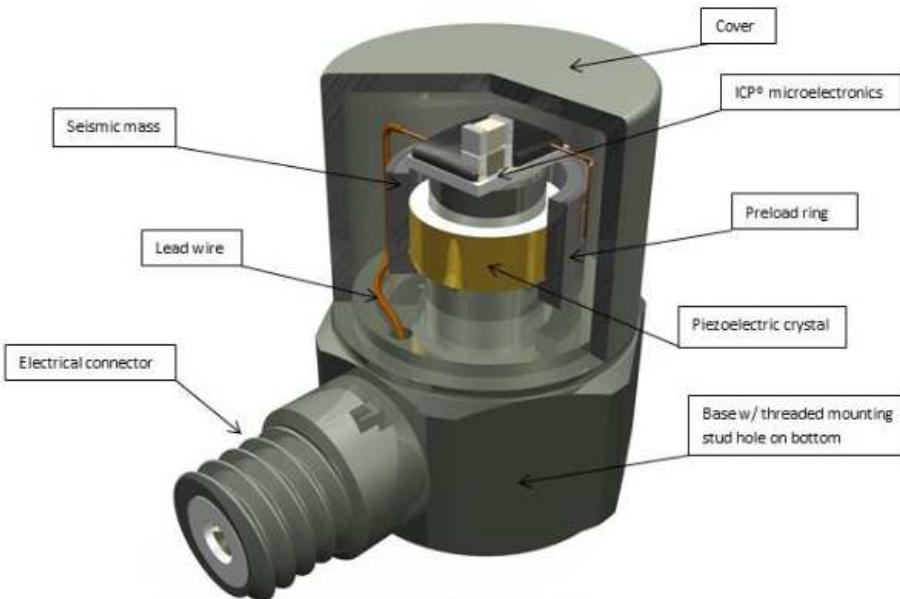


Ilustración 58. Acelerómetro piezoeléctrico desarrollado por PCBpiezotronics²⁰

El IMU empleado en el robot consta de un acelerómetro capacitivo²¹. La aceleración se obtiene midiendo la capacitancia entre dos planos, la cual depende de la distancia entre estos, el área y la permitividad del medio en el que estén. Al desplazarse uno de los planos, varía la distancia a la que se encuentra y, en consecuencia, la capacitancia.²²

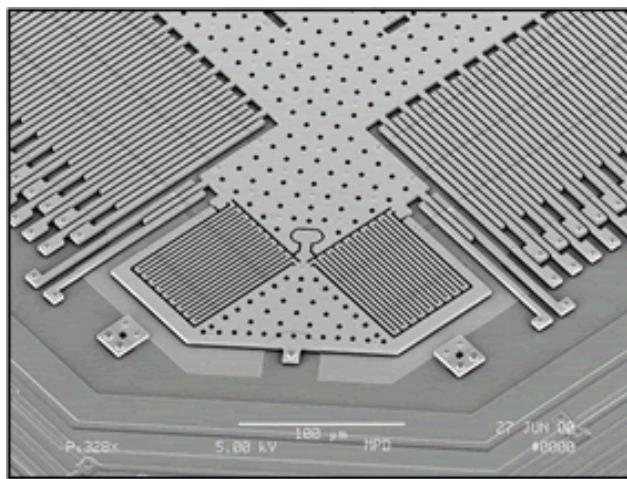


Ilustración 59. Acelerómetro MEMS capacitivo real

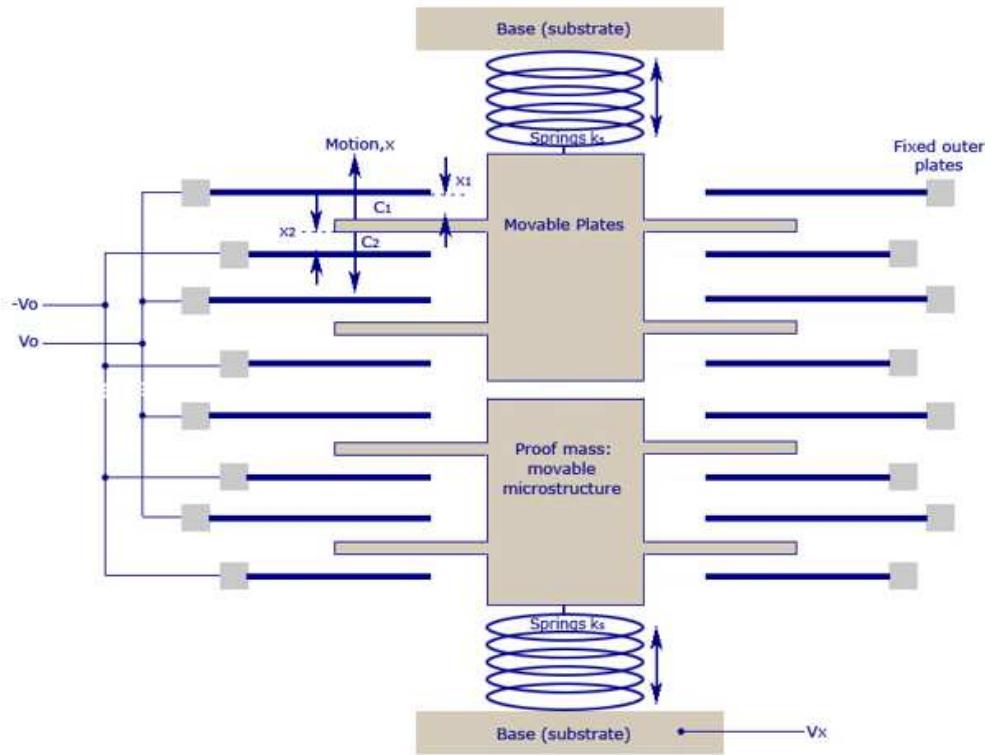


Ilustración 60. Esquema de funcionamiento de un acelerómetro MEMS capacitivo

La IMU empleada en este proyecto, el MPU6050, envía a Arduino valores en bruto procedentes del acelerómetro entre -16384 y 16384. Tiene la posibilidad de modificar la escala del acelerómetro a $\pm 2g$, $\pm 4g$, $\pm 8g$, y $\pm 16g$, pero como no se esperan registrar fuertes aceleraciones, se deja la aceleración por defecto en $\pm 2g$. El objetivo que tiene el acelerómetro en este código es conocer la inclinación de la cabeza del robot tomando como referencia la gravedad. A continuación, se explica cómo utilizar los valores brutos del acelerómetro para calcular la inclinación del robot.

Trabajando directamente con los datos en bruto proporcionados por el MPU sobre las siguientes formulas se obtienen los distintos ángulos para el balanceo (θ_y) y el cabeceo (θ_z)

$$\theta_y = \text{atan} \frac{A_z}{\sqrt{A_y^2 + A_x^2}}$$

$$\theta_z = \text{atan} \frac{A_y}{\sqrt{A_z^2 + A_x^2}}$$

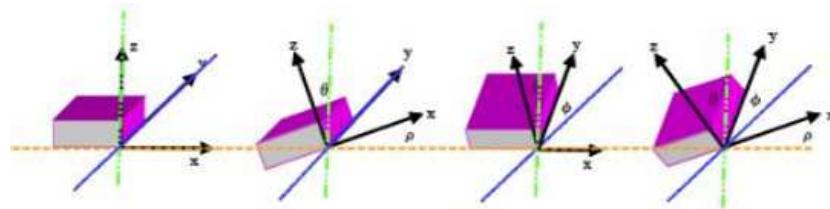


Ilustración 61. Representación de la cabeza respecto tres ejes de coordenadas

<http://husstechlabs.com/projects/atb1/using-the-accelerometer/>

Dado que nos ayudamos de la gravedad para calcular la inclinación, no es posible conocer la giñada, giro correspondiente al eje X. Para ello, se podría recurrir a un futuro a implementar al robot un magnetómetro, para que tome como referencia para el eje X el campo magnético terrestre. Esto será una razón por la que le cueste caminar recto.

Hay que destacar que calcular la inclinación de nuestro dispositivo solamente mediante el acelerómetro es altamente impreciso debido a dos razones. La precisión del acelerómetro de los MPU es bastante baja (comparada con la del giroscopio) debido al ruido, al cambio de la capacitancia interna debido a la temperatura y a que las medidas se ven afectadas por acceleraciones ajenas a las de la gravedad. De esta forma, el acelerómetro no será capaz de conocer la inclinación del robot por si solo cuando se mueva de lado a lado. Hace falta emplear el giroscopio.

➤ *Giróscopo*

Los giróscopos, o giroscopios, son instrumentos capaces de medir y mantener la orientación a la que viaja un cuerpo. Fue desarrollado como tal por Foucault en 1852 gracias a las ideas que fueron apareciendo desde 1813 de mano de Johann Bohnenberger, descubridor del efecto giroscópico.

Las primeras aplicaciones de esta tecnología fueron orientadas a la industria de la guerra especialmente para la navegación de torpedos y misiles. En aquellos tiempos, los giróscopos eran pesadas y voluminosas máquinas que nada tenían que ver con lo que son hoy en día. Básicamente lo formaba una pieza pesada en rotación en el núcleo sujeto por un soporte de Cardano. La rotación se mantenía mediante la proyección constante de aire y la desviación del soporte respecto al núcleo la detectaban unos sensores que indicaban cuánto tenía que virar el torpedo o misil para restablecer la orientación.

Con el desarrollo de la electrónica y la mecánica aparecen los giróscopos MEMS. Se reducen en peso, volumen y precio llegando al mercado civil. Se basan en el efecto Coriolis para tomar los valores.

El efecto Coriolis es la distinta percepción del movimiento al observar, desde un sistema de referencia no inercial, un cuerpo presente en un sistema inercial. Recordamos que los sistemas inerciales son aquellos sistemas que obedecen a las leyes de Newton por lo que los sistemas rotatorios, como la tierra, no lo son. Por ello, si lanzamos un proyectil desde un sistema rotatorio, el proyectil parecerá estar sometido a una fuerza a medida que se aleja del eje de rotación. Esta fuerza es la fuerza de Coriolis.

La fuerza de Coriolis, que no es una fuerza como tal, hay que tenerla en cuenta cuando se desea trabajar con las leyes de Newton en sistemas no inerciales. El problema por el que se hace esto es que, técnicamente, los sistemas inerciales no existen. La Tierra no lo es ya que, al rotar sobre el sol, deja de ser un sistema inercial y el sol tampoco ya que gira a su vez sobre la Vía Láctea. No obstante, se aceptan ciertas consideraciones para simplificar los cálculos y en la mayoría de ocasiones la tierra se toma como un sistema inercial.

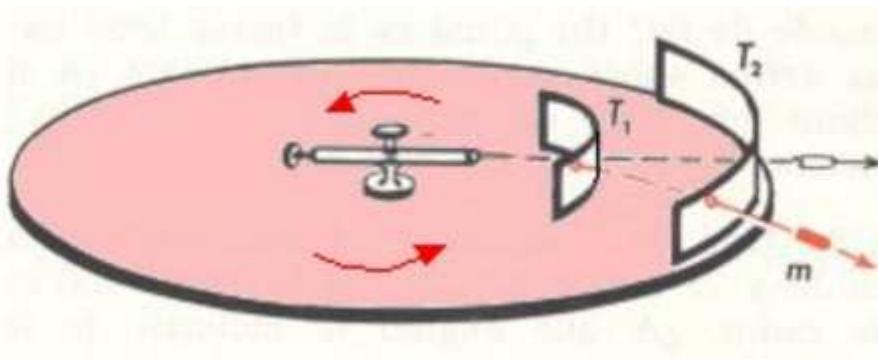


Ilustración 62 Ejemplo visual del efecto Coriolis.

https://en.wikipedia.org/wiki/Coriolis_force

El giróscopo MEMS consta de una masa sujeta por materiales flexibles, a modo de resortes, permitiéndola tener cierta movilidad. La masa se hace vibrar mediante campos electromagnéticos para que, cuando el giróscopo gire, se vea sometida al efecto Coriolis y empuje “lateralmente” el marco interior (véase la ilustración 64). El marco interior pasa a vibrar lateralmente, vibración es detectada por sensores capacitivos y amplificada enviando una señal con la velocidad angular al procesador del IMU. El hecho de vibrar con campos

electromagnéticos es la causa de que el giróscopo consuma 3.6mA frente 500 μ A del acelerómetro.²¹

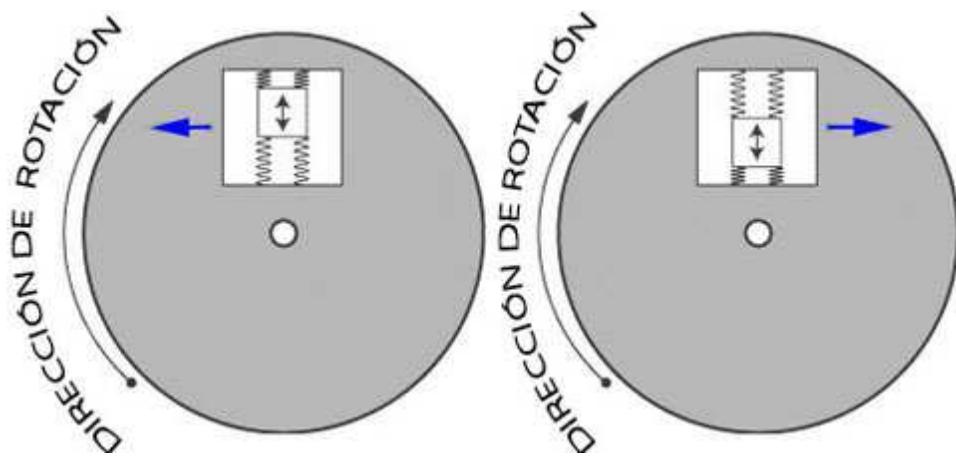


Ilustración 63. La masa vibra a causa de los campos electromagnéticos. Cuando el giróscopo gira, esta sufre una desviación lateral detectada por los sensores capacitivos

<http://5hertz.com/tutoriales/?p=431>

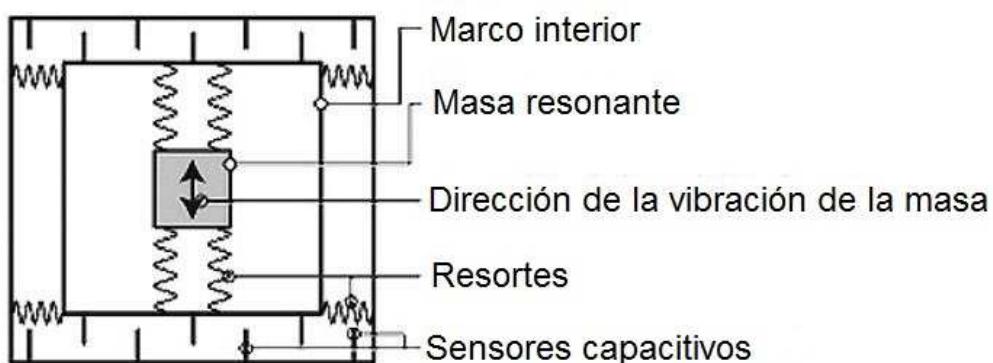


Ilustración 64. Esquema del funcionamiento de un giróscopo MEMS.
http://www.sapiensman.com/tecnoficio/electricidad/electricidad_del_automotor19.php

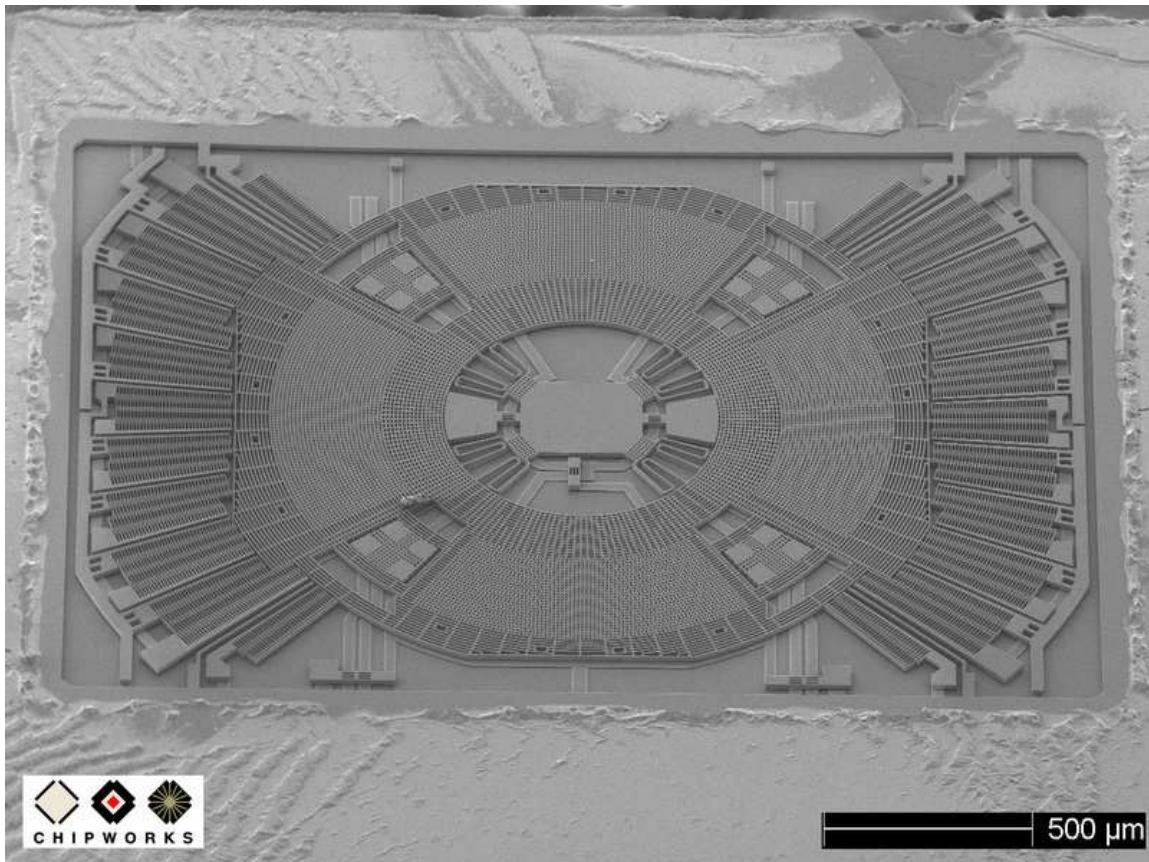


Ilustración 65. Giróscopo MEMS real

<https://es.ifixit.com/Teardown/iPhone+4+Gyroscope+Teardown/3156>

Al igual que con el acelerómetro, el giróscopo tiene la posibilidad de escalar la medida en ± 250 , ± 500 , ± 1000 y ± 2000 °/segundo. En este proyecto se ha dejado la escala por defecto ± 250 °/s.

Dado que los valores en bruto del giróscopo representan la velocidad angular, para conocer el ángulo en el que se encuentra nuestro robot, se debe integrar:

$$\omega = \frac{d\theta}{dt} \rightarrow \theta = \int \omega \cdot dt$$

Que implementado en el código queda

$$\theta = \theta_{anterior} + \omega \cdot \Delta t$$

Siendo Δt una variable discreta que marcará la frecuencia con la que se tomará la medida. Se entiende que cuanto menor sea este valor mayor será la precisión. Para este caso el valor 5ms funciona a la perfección.

Al tratar valores que deberían ser continuos con valores discretos se crea un pequeño error comúnmente denominado “*drift*”. Este error se acumula a lo largo del tiempo al sumarlo una y otra vez. A largo plazo se obtienen unos valores que difieren mucho de la realidad, por eso se necesitan los valores del acelerómetro para tomar la dirección de la gravedad como referencia.

La combinación de los datos proporcionados por el acelerómetro y los del giróscopo se hace mediante un filtro. El tipo de filtro que se emplea en este proyecto es un filtro complementario.

➤ *Filtro*

Los errores en las medidas son algo intrínseco al MPU. Ruido, desviaciones en el tiempo de procesado, acumulación del error (*drift*)... El código necesita filtro para obtener medidas adecuadas.

El filtro Kálman es actualmente el filtro con el que se guían misiles balísticos y demás aparatos que necesitan seguir una orientación extremadamente precisa. Es un conjunto de algoritmos que autogeneran el error de predicción aprendiendo recursivamente el error que deben compensar.²³ Este filtro presenta dos problemas para implementarlo en Arduino, el primero es que es muy complejo y el segundo que requiere una gran capacidad de procesamiento con la que Arduino puede tener problemas.

Se decide adoptar el filtro complementario. Es sencillo, fácil de implementar, requiere poco procesador y es lo suficientemente preciso para guardar el equilibrio de nuestro robot.

Consiste en valorar los datos del giróscopo por encima de los datos proporcionados por el acelerómetro. Aparte de que los datos del acelerómetro pueden hallar la inclinación con bastante menor precisión que con el giróscopo, también valora todas las aceleraciones que actúan sobre él. El giróscopo, aunque sea más preciso que el acelerómetro, no tiene una referencia de la que partir, por eso se necesita el acelerómetro.

A groso modo, el filtro complementario utiliza el acelerómetro tomando la gravedad como referencia y el giróscopo para reducir al mínimo los errores del acelerómetro. Este proceso valora los valores proporcionados por el acelerómetro y el giróscopo en función de un

parámetro denominado ganancia. La ganancia tiene un valor entre 0 y 1 y hace una ponderación entre los datos del acelerómetro y del giróscopo. La fórmula implementada es:

$$\text{ángulo} = \text{ganancia} \cdot \text{ánguloGiróscopo} + (1 - \text{ganancia}) \cdot \text{ánguloAcelerómetro}$$

Estos algoritmos han sido implementados en la rutina tomarDatosMPU.. Ver anexo 5.4

A los ejes se les denomina balanceo, cabeceo y guiñada en analogía a la jerga utilizada en aviación.

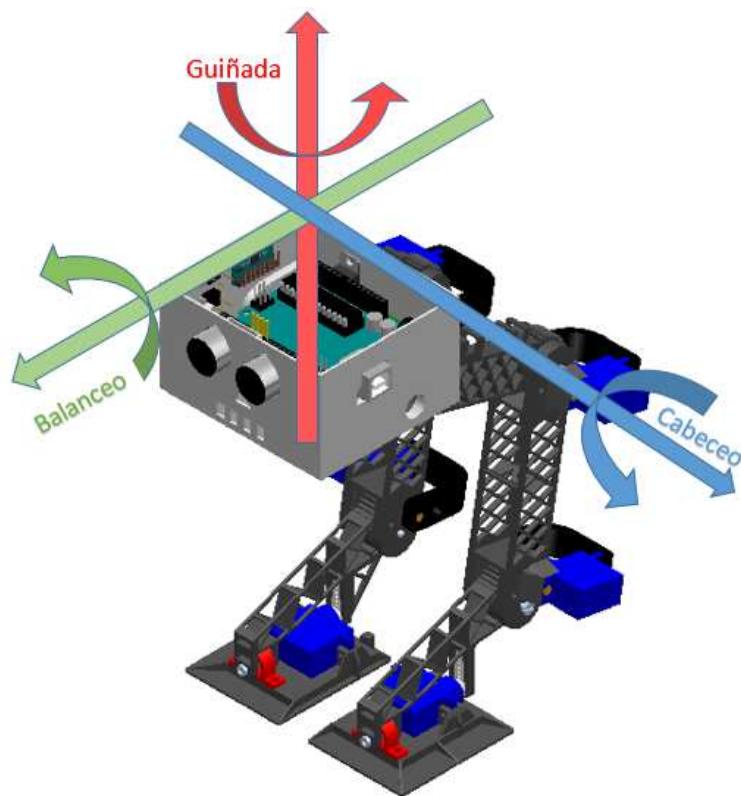


Ilustración 66. Dirección y denominación de los ejes

CAPÍTULO 3. PROGRAMACIÓN Y CÓDIGO

Para caminar, la línea general consiste en asignarle al robot una rutina con todos los movimientos y posiciones que debe adoptar. Un problema de esto era colocar cada servo en la misma posición de un montaje a otro. Si hacemos un código asignando unos valores directamente a los servos y deseamos desmontar y volver a montar el servo, los valores dados en el primer montaje no se corresponden con los del segundo, y si así fuera, sería mera casualidad.

Por ejemplo, si se quiere que la articulación del pie esté a 90°, quizá el valor que deba enviarse al servo sea de 82, pero puede que para el próximo montaje ese valor sea de 95.

Por mucho que se intente ajustar en el montaje, acabará siendo aleatorio. Se pensó despreciar esto y utilizar tan solo incrementos y decrementos directamente sobre los servos, pero de esta forma, no sería capaz de ir a la misma posición inicial cada vez que se montara. La solución tomada está implementada en la rutina IrA:

- **IRA**

Al robot se le suministran una serie de posiciones introduciendo el valor de todas y cada una de las articulaciones.

```
IrAP2(103, 100, 121, 118, 114, 102, 175, 175, velDef2);
IrAP2(100, 99, 118, 124, 114, 98, 185, 173, velDef2);
IrAP2(99, 99, 116, 126, 114, 102, 187, 171, velDef2);
IrAP2(99, 99, 114, 128, 114, 102, 180, 172, velDef2);
IrAP2(99, 99, 112, 130, 114, 102, 175, 174, velDef2);
IrAP2(97, 96, 112, 130, 114, 102, 175, 175, velDef2);
IrAP2(95, 93, 112, 130, 114, 103, 175, 175, velDef2);
IrAP2(80, 80, 112, 130, 114, 104, 175, 175, velDef2);
IrAP2(80, 78, 116, 126, 116, 108, 173, 178, velDef2);
IrAP2(80, 78, 116, 126, 110, 108, 173, 178, velDef2);
IrAP2(80, 78, 116, 126, 105, 108, 173, 178, velDef2);
IrAP2(80, 78, 125, 126, 92, 108, 170, 180, velDef2);
IrAP2(80, 78, 125, 126, 91, 108, 168, 180, velDef2);
IrAP2(80, 78, 125, 126, 90, 108, 167, 180, velDef2);
IrAP2(80, 78, 130, 126, 89, 108, 164, 180, velDef2);
IrAP2(80, 78, 135, 126, 88, 108, 160, 180, velDef2);
```

Ilustración 67. Extracto del programa 2, Se introducen los valores brutos de las articulaciones para cada posición acompañado de un tiempo se espera “velDef”. Es el código más primitivo.

IrA no se implementa directamente sobre las rutinas principales, sino que va dentro de las rutinas “posicionInicio”, “irAPaso” e “IrAP2”. La rutina “IrAP2” hace lo mismo, pero incluye

un *delay* para controlar la velocidad del movimiento. La fórmula para mandar al servo el valor de la articulación es:

$$\text{ValorServo} = \text{redu} \cdot \text{sentido} \cdot (\text{ánguloArticulación} - \text{angIni}) + \text{valIni}$$

Donde:

ValorServo: es el resultado. Es el valor que se manda al servo para un ángulo dado.

Redu: es la reducción de la articulación.

Sentido: es una variable para cambiar el sentido de giro para que, al aumentar este valor, la articulación abra y no cierre.

ánguloArticulación: es el ángulo que se quiere para la articulación. Es el dato de entrada. (ver Ilustración 62)

angIni: es el ángulo que se quiere para cada articulación en la posición inicial, Independientemente de cómo se haya montado el robot

valIni: es el valor que recibe cada servo para que la abertura de la articulación corresponda con “angIni”. Estos valores deben actualizarse en cada montaje con ayuda de la rutina “Calibración”. Esta rutina manda directamente los valores de “valIni” a los servos.

Para cada montaje, se debe desnombrar el *delay* que hay debajo de la rutina “Calibración” para añadir un tiempo a la posición inicial, donde se ajustarán los valores “valIni” hasta que la abertura de la articulación corresponda con la abertura que debe tener (angIni).

La ejecución del código del robot puede seleccionar varios tipos de caminar a través del ordenador (ver anexo 5.5). A continuación, se explicarán las rutinas de la rutina principal Caminar1.

- POSICIÓN DE INICIO

Va a la posición de equilibrio. Para ello, se coloca inicialmente en una posición estable que le será cómoda para iniciar la primera zancada. Mediante los datos proporcionados por el MPU se incrementa o decrementa el ángulo de las articulaciones de los pies y de la cabeza, actuando sobre los ejes Z e Y hasta que la cabeza queda paralela a la superficie. Esta rutina se pensó para que el robot fuera capaz de caminar sobre planos inclinados. Posibilidad que se vio frustrada al comprobar que en pocas zancadas el robot giraba notablemente hacia arriba o hacia abajo de la cuesta deshaciendo el equilibrio adquirido en la posición de inicio y perdiendo el equilibrio.

Llevó mucho más tiempo del esperado conseguir que caminara recto sobre un plano horizontal razón por la que no se ha profundizado en caminar por planos inclinados. No obstante, el código se pensó en llevar a cabo esta posibilidad.

La posición de equilibrio necesita de tres condiciones.

- Que hayan pasado 2 segundos
- Que corrobore 30 veces que el ángulo del cabeceo se encuentra entre ± 1
- Que corrobore 30 veces que el ángulo del balanceo se encuentra entre ± 1

Cumplidas estas tres condiciones se guardan los valores donde se encontraban las articulaciones para utilizarlas como referencia durante las zancadas. Ver anexo 5.6

- **IR HASTA**

Después de tener el robot perfectamente posicionado, la rutina irHasta se ejecuta inclinando lateralmente el robot hasta que el MPU detecta que ha llegado a la inclinación que se le ha ordenado. Se ha comprobado que con 8° funciona correctamente. También se ha comprobado que puede llegar a los 17° sin caerse lateralmente.

Cuando se termina de inclinar el robot, se guarda la inclinación de los pies para utilizarla en las rutinas zancada izquierda/derecha. (Ver anexo 5.7)

- **ZANCADA DERECHA/IZQUIERDA**

Estas rutinas, como su nombre indica, ejecutan una zancada con la pierna derecha o con la izquierda. Es una rutina que funciona mezclando valores brutos de las articulaciones con valores proporcionados por el MPU para mantener el equilibrio. Se determinaron sobretodo empíricamente y, en alguna ocasión, se empleó Solid Edge para hacer alguna pequeña simulación.

```
//----- Desplaza la pierna derecha adelante -----
void zancadaDerecha() {
    ledOff();
    irAPaso(incliA1, incliA2 - 1, 121, 118, 114, 110, 175, 175, velDef);
    irAPaso(incliA1, incliA2 - 2, 121, 115, 114, 106, 175, 177, velDef);
    irAPaso(incliA1, incliA2 - 2, 121, 118, 114, 102, 175, 180, velDef);
    irAPaso(incliA1 - 3, incliA2 - 2, 118, 120, 114, 98, 185, 183, velDef);
    irAPaso(incliA1 - 4, incliA2 - 2, 116, 125, 114, 98, 187, 183, velDef);
    irAPaso(incliA1 - 4, incliA2 - 2, 114, 125, 114, 98, 180, 183, velDef);
    irAPaso(incliA1 - 4, incliA2 - 2, 112, 125, 114, 98, 175, 180, velDef);
    irAPaso(incliA1 - 7, incliA2 - 2, 112, 125, 114, 98, 175, 177, velDef);
    irAPaso(incliA1 - 8, incliA2 - 10, 112, 125, 114, 100, 175, 175, velDef);
    irAPaso(inicioA1, inicioA2, 112, 126, 114, 103, 175, 175, velDef);
}
}
```

Ilustración 68. Captura de pantalla del código. Rutina zancadaDerecha. El inicio, ledOFF, es para apagar el led si estuviera encendido por una rutina anterior. IncliAX son los valores guardados para permanecer inclinado 8°

- IR A PASO

irAPaso es una variante de la rutina ya comentada irA. Además de hacer lo mismo que esta rutina, chequea los valores del MPU y detecta si se ha caído mediante la rutina testCaida (Ver anexo 5.8). La rutina testCaida evalúa si el cabeceo o el balanceo han excedido unos determinados valores y si ha sido así, parpadea el Led rojo y se reinicia el Arduino.

La rutina irAPaso termina con un algoritmo que detecta si el valor velDef, es mayor que el tiempo de procesar la inclinación mediante el MPU. Si es mayor, la espera total es velDef y si es menor, la espera total es el tiempo que tarda en tomar los datos del MPU.

- EVITAR DESDE DERECHA/IZQUIERDA

Después de cada zancada se ejecuta evitarDesdeDerecha/Izquierda. Esta rutina detecta si el sensor de ultrasonidos detecta algún obstáculo en frente del robot. Si no lo detecta parpadea brevemente la luz verde y sigue con otra zancada sino, parpadea la luz roja y ejecuta la maniobra de girar.

- MANIOBRA DE GIRAR

Esta maniobra está dentro de la rutina evitarDesdeIzquierda/Derecha.

```

444 //Realiza el movimiento de giro a la izquierda 8 veces
445 for (int i = 0; i < 8; i++) {
446     giroDesdeDerecha();
447     irHasta(-10);
448     zancadaDerecha();
449     ledOff();
450 }
  
```

Ilustración 69. Extracto de la rutina evitarDesdeDerecha. El numero de veces que se ejecuta el bucle determina el ángulo de giro

Si se detecta el obstáculo desde, por ejemplo, la zancada derecha el robot gira a la izquierda y viceversa.

El principio por el que gira el robot a falta de libertad en un eje se debe a que roza más por una de las plantas de los pies que por la otra. Las fuerzas de rozamiento en direcciones opuestas provocan un momento que lo obliga a torcer. “Giro desde Izquierda/Derecha” es la rutina que hace rozar las plantas para rotar el robot. También acerca el robot a una posición cercana a la de inicio, para conseguir el enlace entre movimientos. Se repiten “irHasta”, “zancadaDerecha/Izquierda” y de nuevo “giroDesdeDerecha/Izquierda”. Cuando termina el giro, el robot queda estratégicamente colocado. Por ejemplo, al final de zancadaDerecha le resultaría fácil empezar con la zancadaIzquierda y así sucesivamente.

El motivo por el cual la dirección de giro depende de la zancada que se esté llevando a cabo reside en la dificultad de conseguir que las acciones enlacen entre sí sin hacerlo rotar involuntariamente ni llevarlo a una posición en la que corra peligro su estabilidad. Este punto consumió gran parte del tiempo del proyecto.

- PROGRAMA CAMINAR1

Después de calibrar e ir a la posición de inicio con ayuda del MPU, entra en el siguiente bucle.

```

253 while (true) {
254     irHasta(-8);      // - es inclinar a la izquierda y + es inclinar a la derecha
255     zancadaDerecha();
256     evitarDesdeDerecha();
257     irHasta(8);
258     zancadaIzquierda();
259     evitarDesdeIzquierda();
260 }
  
```

Ilustración 70. Bucle del programa principal.

Puede verse el diagrama de flujo y el resto del programa en el anexo 5.9.

Dado que se querían incluir varios programas que entraran en bucle individualmente se decidió no utilizar el *loop* de la manera convencional, en su lugar se utilizan varios bucles *while (true)* tal y como se ve en la ilustración 70. El *loop* solo se utiliza para la selección de la rutina caminar.

- PROGRAMA CAMINAR2

El programa Caminar2 es de exhibición. Es el primer programa con el que el robot pudo andar. El código es extremadamente simple. Se caracteriza por dar zancadas largas considerablemente más rápidas que el programa Caminar1. No hace uso del MPU ni del sensor de ultrasonidos, tan solo camina.

Fue durante la elaboración de este programa cuando se vio la necesidad de añadir alguna luz con la que poder saber qué línea de código estaba fallando. Se intercalaban los colores de los leds para saber la línea en la que se encontraba la ejecución.

```
323 IrAP2(103, 102, 121, 118, 114, 110, 175, 175, velDef2);  
324 IrAP2(103, 101, 121, 115, 114, 106, 175, 175, velDef2);  
325 IrAP2(103, 100, 121, 118, 114, 102, 175, 175, velDef2);  
326 IrAP2(100, 99, 118, 124, 114, 98, 185, 173, velDef2);  
327 IrAP2( 99, 99, 116, 126, 114, 102, 187, 171, velDef2);  
328 IrAP2( 99, 99, 114, 128, 114, 102, 180, 172, velDef2);
```

Ilustración 71. Extracto del código de la rutina "Caminar2"

CAPÍTULO 4. BIPEDESTACIÓN Y CARACTERÍSTICAS FINALES

- BIPEDESTACIÓN

El avance del robot es simple: se inclina, se levanta una pierna, se la hace avanzar, se asienta en una nueva posición y se inclina hacia la nueva posición. Al final de cada zancada utiliza el sensor de ultrasonidos para saber si hay algún obstáculo, y si lo hay, lo rodea en la dirección opuesta a la zancada (si es la zancada izquierda, lo rodea hacia la derecha, y viceversa). Tal y como se ha explicado en el capítulo anterior, el código indica al robot posición a posición todos los movimientos que debe hacer ayudándose en algunas ocasiones del MPU.

Durante la elaboración de los códigos de movimiento se encontraron algunas reglas y particularidades a las que cabe hacer referencia. A continuación, se van a enumerar y explicar.

➤ *Sumatorio de ángulos*

Uno de los problemas encontrados fue conocer cómo se tenían que colocar las articulaciones para que el pie aterrizara paralelo al suelo al terminar una zancada. Se descubrió que para que el pie siguiera paralelo a la base de la cabeza, el sumatorio de los ángulos de las articulaciones de cada pierna debía ser el mismo que para la posición de inicio.

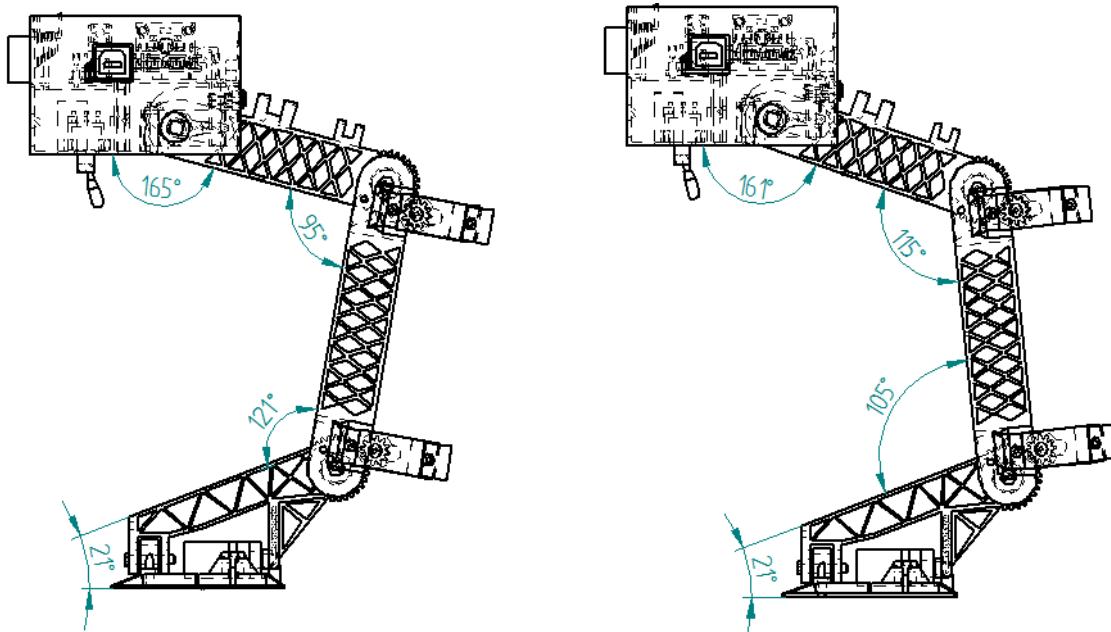


Ilustración 72. La suma de ángulos menos el ángulo del empeine es igual a 360°

Al hacer la simulación con Solid Edge se descubre que

$$\text{Izquierda} \rightarrow 165+95+121-21=360$$

$$\text{Derecha} \rightarrow 161+115+105-21=360$$

No obstante, cuando se intenta implementar esto en el código se experimenta que no funciona tan bien como gustaría. El motivo es que debido a las pequeñas holguras y a los desajustes al asignar la variable “*angIni*” solo se cumple la regla en la simulación.

Sí que se tuvo en cuenta que la suma de los ángulos debe ser igual o parecida a la de la posición de inicio, condición sí se respeta en la práctica.

➤ *Enlace entre zancadas*

Hay una diferencia notable en la forma de caminar entre el “Caminar1” y el “Caminar2”. El “Caminar2” trata de respetar la forma natural de andar en cuanto al cambio de pesos de lado a lado se refiere.

Una vez se asienta el pie que ha dado la zancada, se empieza a hacer una redistribución de pesos en diagonal a la dirección en la que se camina, tal y como hacemos los humanos.

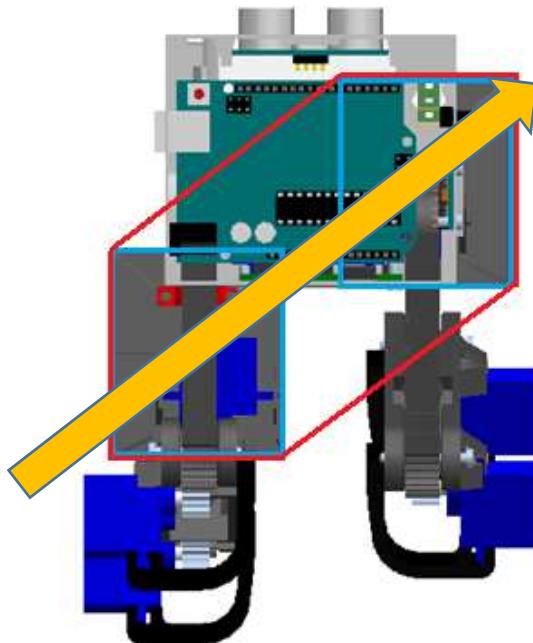


Ilustración 73. Cambio de pesos en diagonal propio del programa 2. Los rectángulos azules representan los pies, el polígono rojo los lugares donde puede estar el centro de gravedad sin poner en peligro la estabilidad. La flecha naranja la dirección del centro de gravedad a lo largo del movimiento.

El problema que entraña nuestro robot reside en que esa redistribución no se reparte correctamente a lo largo de la planta del pie y rota considerablemente. Puede verse en la práctica se apoya primero la esquina frontal exterior del pie y rota.

Para que no rote, la rutina “Caminar2” evita trasladar el centro de gravedad en diagonal, inclinando el robot lateralmente en un solo eje.

Esta forma de caminar requiere de pasos más cortos. En la siguiente ilustración puede verse el recorrido por el que puede pasar el centro de gravedad para ser estable si solo se mueve en un eje. El hecho de que no avance mientras se inclina repercute negativamente en la velocidad. Esta es una de las razones por la que “Caminar1” es bastante más lento que “Caminar2”

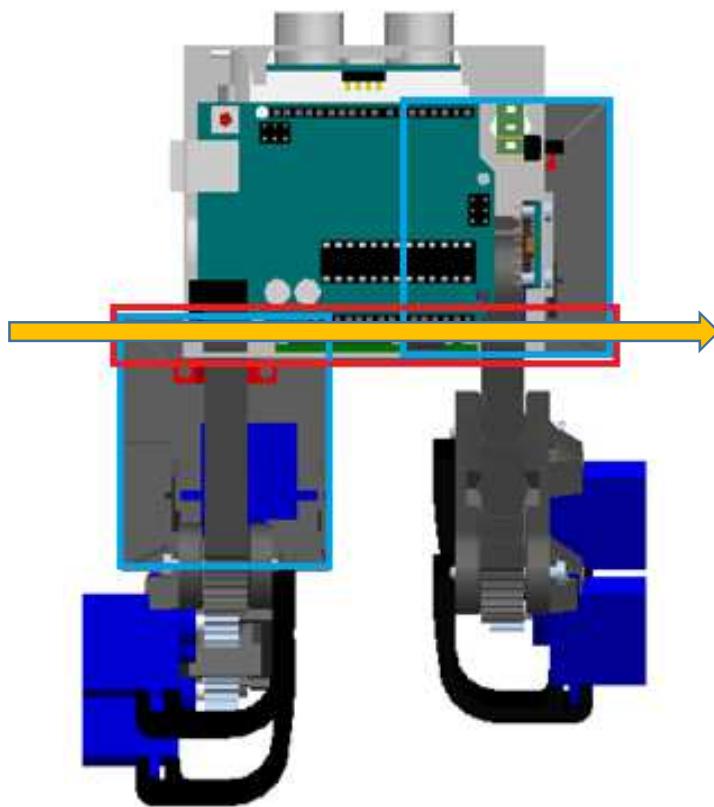


Ilustración 74. Cambio de pesos sobre un eje tal y como hace la rutina irHasta. Propio del programa 1. Nótese que el centro de gravedad no debe salir del recuadro rojo. Las magnitudes del dibujo no se corresponden con la realidad

➤ Caminar recto

Se ha comentado anteriormente las complicaciones relativas a caminar derecho. A modo de resumen, resulta extremadamente complicado establecer unas rutinas para la zancada derecha

e izquierda que enlacen entre sí sin rotar en ningún momento, aunque esté prácticamente conseguido en el programa principal. Se hizo a costa de hacer un programa lento de zancadas cortas y con un cambio de pesos únicamente lateral. Ver imagen 74.

Durante la realización del código se barajó la posibilidad de utilizar los datos proporcionados por el giróscopo para saber la deriva que iba teniendo al caminar y emplear las rutinas de girar para compensarlo. Aunque no tenga magnetómetro si se puede saber la velocidad angular del eje de la guñada. Esta opción presentaba varios problemas.

En un principio, para probar esta idea, se hizo un código simple en el que se visualizaba en la consola de Arduino el ángulo de la guñada. El MPU era capaz de detectar cuándo y cuánto rotaba. La fórmula era la misma que se emplea en los otros ejes de los giróscopos:

$$\theta = \theta_{anterior} + \omega \cdot \Delta t$$

Para un $\theta_{anterior}$ inicial igual a 0

Siendo ω la velocidad angular detectada e Δt el intervalo de tiempo de la medición, valor que corresponde con un delay de 5ms.

La prueba no fue satisfactoria por dos razones distintas. La primera se debía al error producido por el *drift*. Las medidas recibidas tienen ruido y el eje carece de una referencia que reoriente la guñada, las medidas recibidas al cabo de un tiempo divergen considerablemente de la realidad. La segunda fue que cuando el sensor se movía a una velocidad algo elevada las medias se desviaban rápidamente. Por estas razones se desecha la idea de controlar la guñada.

Volviendo al programa 1, se consigue empíricamente que el robot camine bastante recto sobre una tarima de madera lisa. No obstante, se experimenta que este pequeño giro varía en función de la superficie por donde camine. Para determinar si este giro era proporcional al coeficiente de rozamiento entre el pie y la superficie se toman muestras de radios de giro.

En la siguiente tabla se pueden ver los radios de giro obtenidos para diferentes superficies. Las superficies están ordenadas de mayor a menor rugosidad según la percepción al tacto.

Superficie	Sentido	Radio (cm)
Tarima de madera rugosa	derecha	70
Tarima de madera lisa	derecha	>200
Aglomerado de madera pintada	Izquierda	>200

Baldosas granuladas	Izquierda	130
Baldosas pulidas	Izquierda	180
Cristal	derecha	25

Tabla 1. Radio de giro del robot en función de la superficie

Se considera que el robot se desplaza recto cuando no se puede medir con precisión el radio de giro. Para estos casos se han tomado unos 200cm.

Sorprende la variedad de valores diferentes y de sentidos. En vista del trabajo invertido en que caminara recto por una tarima, se demuestra que resulta extremadamente complicado definir una rutina para la que el robot viaje recto en cualquier superficie.

Dado que el único parámetro del que depende el giro es la superficie tiene que ser necesariamente el coeficiente de rozamiento lo que lo determine. Parece intuitivo que la percepción personal de la rugosidad de una superficie no es un parámetro del que pueda partir un análisis más detallado, no obstante, una tarima de madera rugosa tendrá un coeficiente de rozamiento considerablemente mayor que un cristal por lo que no tiene aparentemente ningún sentido que sean los radios más próximos entre sí, teniendo además mismo sentido.

No se consigue dar una explicación plausible a este fenómeno. Simplemente parece que el radio depende del rozamiento, pero no proporcionalmente. Parece responder a una función muy irregular.

- COSTE GLOBAL

A continuación, se expone un desglose del precio de los artículos, materiales, herramientas y mano de obra necesario para construir el robot:

Lista de artículos	Unidades	Precio unidad	Precio total
Electrónica			

Arduino Uno	1	24,50 €	24,50 €
Batería Floureon Lipo 2S 7.4V 1000mAh	1	14,99 €	14,99 €
Servo SG90 9g	8	1,42 €	11,36 €
Interruptor de 3 pines	1	0,80 €	0,80 €
Sensor de ultrasonidos HC-SR04	1	1,49 €	1,49 €
2x9 zócalo hembra	1	0,20 €	0,20 €
x2 zócalo macho	1	0,05 €	0,05 €
Regulador de tensión LM7805	1	0,14 €	0,14 €
<i>Protoboard</i> de circuito impreso	1	1,00 €	1,00 €
Módulo Led RGB	1	1,75 €	1,75 €
Módulo GY-521 MPU 6050	1	2,33 €	2,33 €
<i>Jumpers</i> varios	1	2,50 €	2,50 €
Pinza de refrigeración	1	0,40 €	0,40 €
		Total electrónica	61,51 €
Ferretería			
Terrajante 2x10	14	0,05 €	0,70 €
M3x16 Allen	4	0,20 €	0,80 €
M3x6 Allen	7	0,20 €	1,40 €
M3x25 Ranurado	4	0,20 €	0,80 €
Rodamiento 623ZZ	6	0,29 €	1,74 €
Arandela 3mm	1	0,01 €	0,01 €
		Total ferretería	5,45 €
Coste impresión 3D			
PLA azul cielo 1,75 (en kg)	0,125	17,60 €	2,20 €
Electricidad, laca fijadora y desgaste de la impresora (+15%)			0,33 €
		Total impresión 3D	2,53 €
		Total en materiales	69,49€

Tabla 2. Lista y coste de los materiales

Lista de herramientas necesarias	Unidades	Coste
Impresora 3D genérica	1	270,00 €
Cable macho USB-A<-->macho USB-B	1	2,60 €
Tijeras	1	1,00 €
Juego de destornilladores	1	2,75 €
Transportador de ángulos	1	0,80 €
Alicates	1	2,80 €
	Total herramientas	279,95 €

Tabla 3. Lista y coste de herramientas

Mano de obra de diseño y fabricación	Tiempo (h)	Coste por hora	Coste total
12 créditos de TFG	300	8,00 €	2.400,00 €

Tabla 4. Coste de la mano de obra

Con las tablas anteriores se halla el coste final del proyecto

Coste total del proyecto	Coste
Materiales	70,00 €
Herramientas necesarias	279,95 €
Mano de obra	2.400,00 €
	Total
	2.749,95 €

Tabla 5. Desglose del coste total

- ESPECIFICACIONES TÉCNICAS

Velocidad lineal	40cm/min	Baldosa lisa. Caminar1
Velocidad de giro	0,3rpm	Baldosa lisa. Caminar1
Autonomía	65min	
Grados de libertad	8	
Inclinación máxima lateral	17º	
Peso total	360g	
Peso del plástico	125g	Anexo
Tiempo de impresión	28h:02'	Anexo
Boquillas necesarias	0.4 y/o 0.3mm	Se recomienda 0.3 para todo el conjunto, pero solo es estrictamente necesaria para la impresión de los piñones
Área mínima de impresión	120x85x48	Dadas por las piezas más grandes, "1.05 espinilla" y "1.07 caja cabeza"
Temperatura máxima de almacenaje	30ºC	Dado de la hoja de especificaciones de la batería
Tiempo aprox. de montaje	2h	
Tiempo aprox. de calibración	45min	
Coste de los materiales	69,49 €	

Tabla 6. Especificaciones técnicas finales

- LÍNEAS DE FUTURO

Con perspectiva de continuar este proyecto se enumeran algunas posibilidades para llevar a cabo en el futuro:

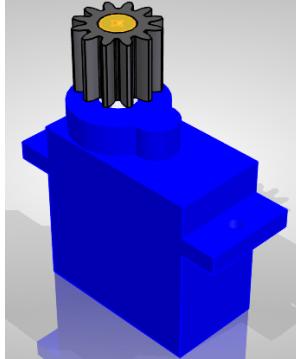
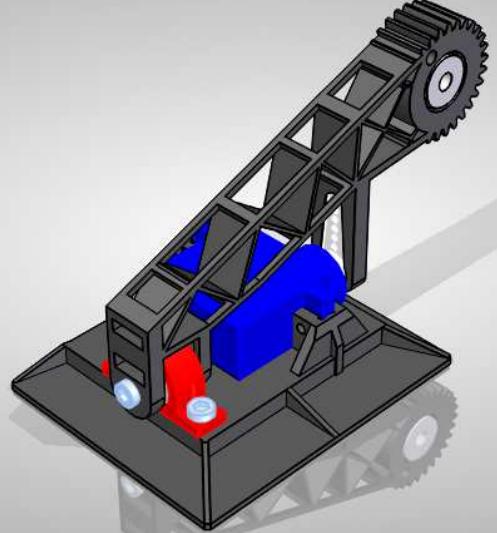
1. Implementación de un IMU con magnetómetro para conocer el norte terrestre y así corregir la deriva al caminar recto. También se podrían realizar los giros con precisión.
2. Añadir dos pulsadores (tal y como parece en el diseño CAD) para seleccionar el tipo de caminar. Un botón para alternar y otro para seleccionar.
3. Al añadir los pulsadores del apartado anterior y conectarlos a las entradas analógicas, se liberan los pines digitales 0 y 1, a los que se podrían conectar el led azul y un zumbador
4. Hacer una rutina de caminar en zigzag para tener más información por el sensor de ultrasonidos y poder tomar decisiones de giro más adecuadas acorde a las posiciones de los obstáculos
5. Establecer una rutina que incline la cabeza hacia abajo para detectar escalones
6. Maximizar la eficiencia de los algoritmos para andar para hacer zancadas más rápidas.
7. Imprimir en ABS y observar los cambios en la resistencia frente a golpes.
8. Diseñar una tapa para la cabeza que no desentone con el resto de la estructura.
9. Introducción de un indicador de batería disponible
10. Implementar un módulo *bluetooth* y utilizar un teléfono móvil como mando radiocontrol.

ANEXOS

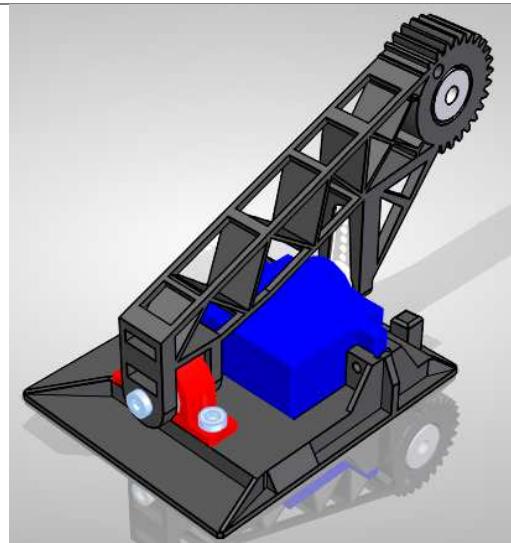
- ANEXO 1. DISEÑO MECÁNICO

Anexo 1.1 Tabla de conjuntos

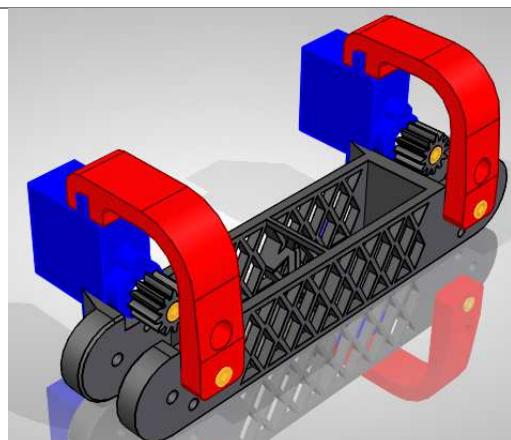
El sufijo_mir significa que es la copia simétrica de la anterior.

Nombre del conjunto	Piezas que contiene	Ilustración
9.6 Servo con rueda R2.asm	3.01 Servo.par 1.03 Piñón.par 2.05 Terrajante 2x10.par	
4.1 Pie.asm	1.01a Pie.par 3.01 Servo.par 2.02 Brazo Servo.par 1.02 Soporte Pie.par 2.04 Cojinete3x10x4.par 1.04 Empeine.par 2.06 M3x16 Allen.par 2.07 M3x6 Allen.par	

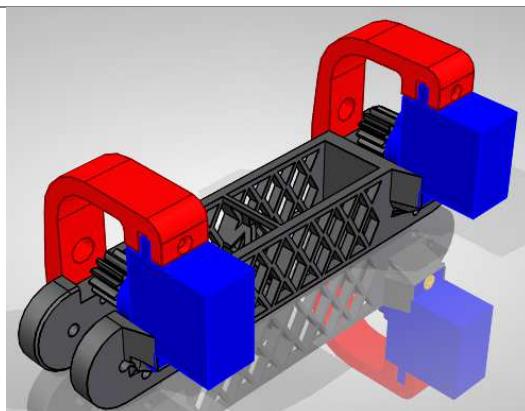
4.1 Pie_mir.asm	1.01 Pie_mir.par
Simétrico del	3.01 Servo.par
anterior	2.02 Brazo Servo.par
	1.02 Soporte Pie.par
	2.04 Cojinete3x10x4.par
	1.04 Empeine_mir.par
	2.06 M3x16 Allen.par
	2.07 M3x6 Allen.par



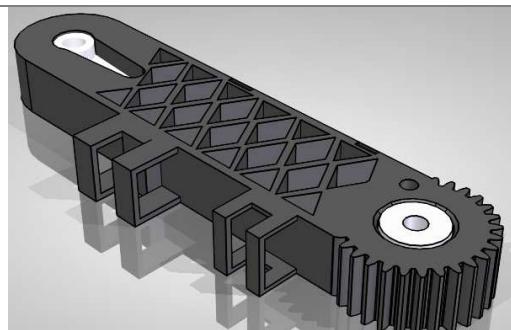
4.2 Espinilla.asm	1.05 Espinilla.par
	9.6 Servo con rueda M2.asm
	2.05 Terrajante 2x10.par
	1.08 Apoyos servos.par



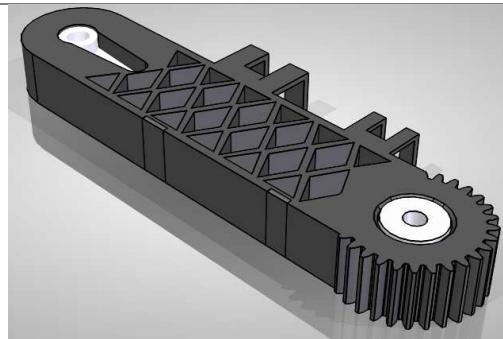
4.2 Espinilla_mir.asm	1.05 Espinilla_mir.par
	9.6 Servo con rueda M2.asm
Simétrico del	2.05 Terrajante 2x10.par
anterior	1.08 Apoyos servos.par



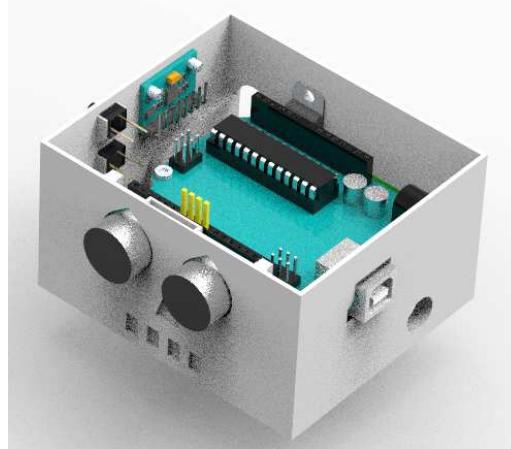
4.3 Femur.asm	1.06 Femur.par
	2.04 Cojinete 3x10x4.par
	2.02 Brazo servo.par



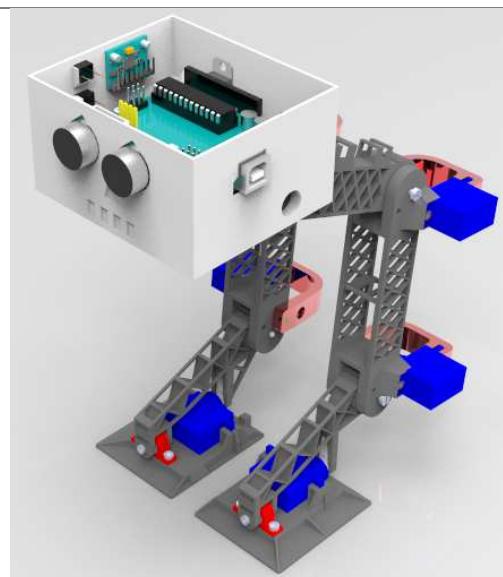
4.3	1.06 Femur_mir.par
Femur_mir.asm	2.04 Cojinete 3x10x4.par
Simétrico del anterior	2.02 Brazo servo.par



4.4 Cabeza+ electrónica.asm	1.07 Caja Cabeza.par
	3.02 Bateria.par
	3.06 Arduino.par
	3.03 Circuito Alimentacion Servos.par
	2.07 M3x6 Allen.par
	3.05 Ultrasonidos.par
	3.08 Pulsador.par
	3.04 Interruptor.par
	3.07 Giróscopo.par
	2.01 Arandela 3mm.par
	2.06 M3x16 Allen.par
	3.01 Servo.par

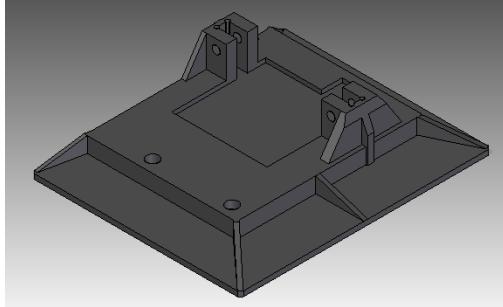
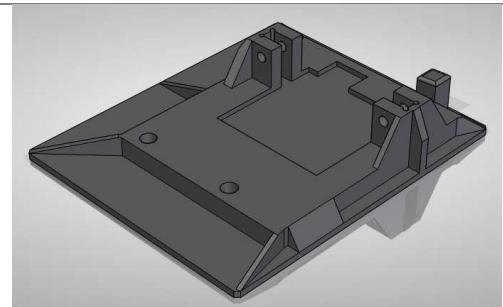
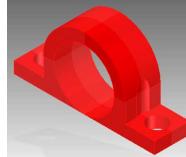
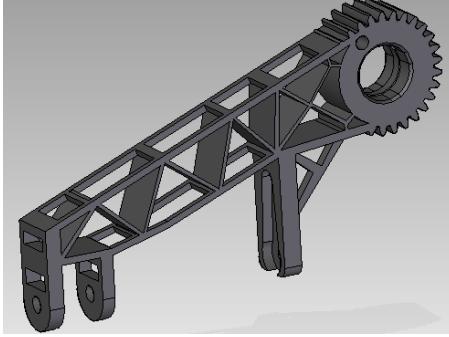


5.0 Robot.asm	4.4 Cabeza+ Electrónica.asm
	4.3 Femur.asm
	4.3 Femur_mir.asm
	4.2 Espinilla.asm
	4.2 Espinilla_mir.asm
	4.1 Pie.asm
	4.1 Pie_mir.asm
	2.08 M3x25.par



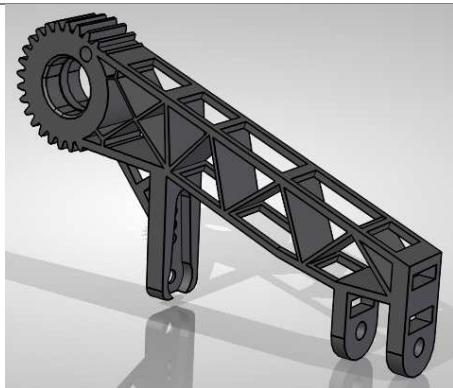
Anexo 1.2. Tabla de Piezas.

El sufijo_mir significa que es la copia simétrica de la anterior.

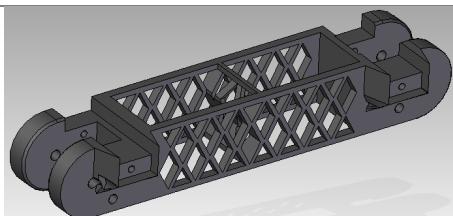
Nombre de la pieza	Conjunto al que pertenece	Ilustración
1.01 PIE.par	4.1 Pie.asm	
1.01 PIE_mir.par	4.1 Pie_mir.asm	
1.02 SOPORTE PIE.par	4.1 Pie.asm 4.1 Pie_mir.asm	
1.03 Piñón.par	9.6 Servo con rueda M2.asm	
1.04 EMPEINE.par	4.2 Pie.asm	

1.04

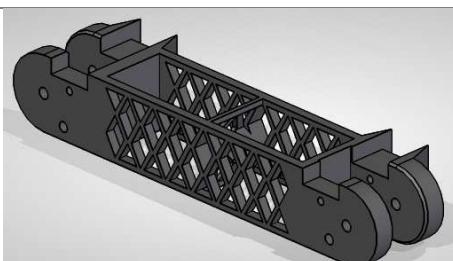
4.2 Pie_mir.asm

EMPEINE_mir.par**1.05 ESPINILLA.par**

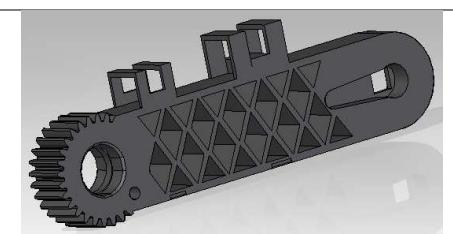
1.02 Espinilla.asm

**1.05**

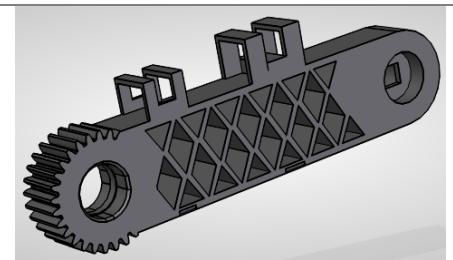
1.02 Espinilla_mir.asm

ESPINILLA_mir.par**1.06 FEMUR.par**

1.03 Femur.asm

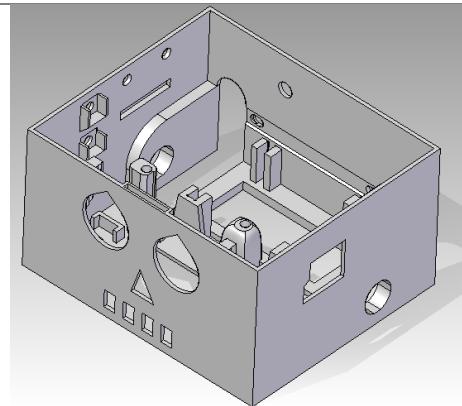
**1.06**

1.03 Femur_mir.asm

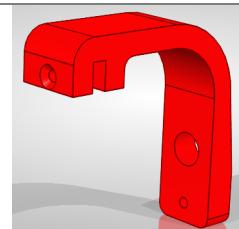
FEMUR_mir.par



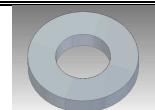
1.07 CAJA 4.4 cabeza +
CABEZA.par electrónica.asm



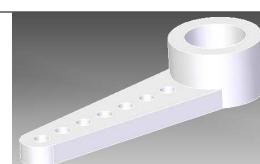
1.08 APOYOS 1.02 Espinilla.asm
SERVOS.par 1.02 Espinilla_mir.asm



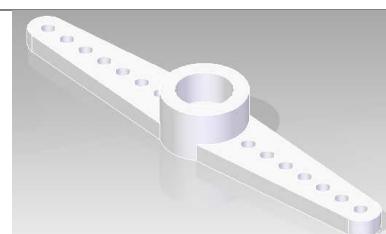
2.01 Arandela 4.4 cabeza +
3mm.par electrónica.asm



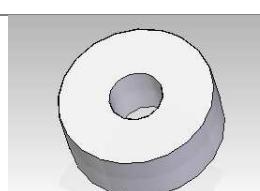
2.02 Brazo servo.par 4.1 Pie.asm
 4.1 Pie_mir.asm
 1.03 Femur.asm
 1.03 Femur_mir.asm



2.03 Brazo servo No se usa
doble.par



2.04 Cojinete 4.1 Pie.asm
3x10x4.par 4.1 Pie_mir.asm
 1.03 Femur.asm
 1.03 Femur_mir.asm



2.05 Terrajate 2x10 1.02 Espinilla.asm
.par 1.02 Espinilla_mir.asm
 9.6 Servo con rueda
 M2.asm



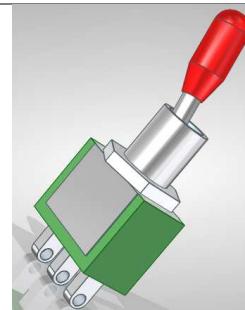


2.06 M3x16 Allen.par	4.1 Pie.asm 4.1 Pie_mir.asm 4.4 cabeza + electrónica.asm	
2.07 M3x6 Allen.par	4.1 Pie.asm 4.1 Pie_mir.asm 4.4 cabeza + electrónica.asm	
2.08 M3x25 Ranurado.par	5.0 Robot.asm	
2.09 M3x30 Ranurado.par	No se usa	
3.01 SERVO.par	4.1 Pie.asm 4.1 Pie_mir.asm 9.6 Servo con rueda M2.asm 4.4 cabeza + electrónica.asm	
3.02 BATERIA.par	4.4 cabeza + electrónica.asm	
3.03 CIRCUITO ALIMENTACIÓN SERVOS.par	4.4 cabeza + electrónica.asm	



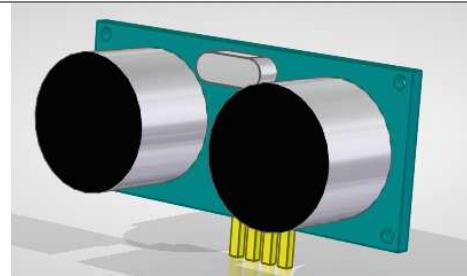
3.04

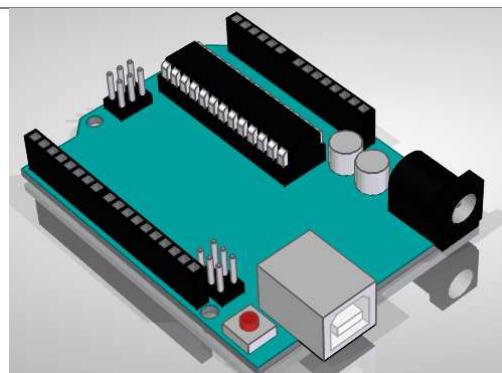
4.4 cabeza +

INTERRUPTOR.par electrónica.asm

3.05

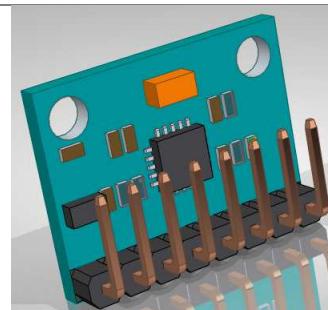
4.4 cabeza +

ULTRASONIDOS.par electrónica.asm
ar

3.06 ARDUINO.par4.4 cabeza +
electrónica.asm

3.07

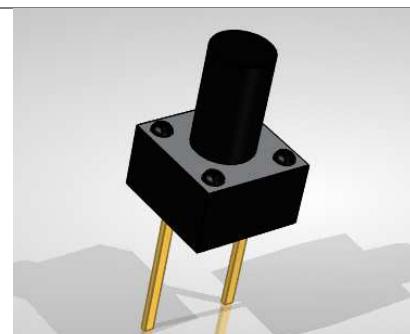
4.4 cabeza +

GIROSCOPO.par electrónica.asm

3.08 PULSADOR.par

4.4 cabeza +

electrónica.asm



- **ANEXO 2. ESTUDIO DEL COMPORTAMIENTO DE LOS ÁNGULOS**

Anexo 2.1 Tabla ángulos para el primer programa

ang0	ang1	ang2	ang3	ang4	ang5	ang6	ang7	tiempo (acumulada)	tiempo
90	90	121	121	114	114	175	175	0,3	300
93	93	121	121	114	114	175	175	0,6	300
96	96	121	121	114	114	175	175	0,9	300
99	99	121	121	114	114	175	175	1,2	300
101	101	121	121	114	114	175	175	1,5	300
103	103	121	121	114	114	175	175	1,8	300
103	102	121	118	114	110	175	175	2,1	300
103	101	121	115	114	106	175	175	2,4	300
103	100	121	118	114	102	175	175	2,7	300
100	99	118	124	114	98	165	173	3,2	def
99	99	116	126	114	102	167	171	3,7	def
99	99	114	128	114	102	170	172	4,2	def
99	99	112	130	114	102	175	174	4,7	def
97	96	112	130	114	102	175	175	5,2	def
95	93	112	130	114	103	175	175	5,7	def
80	80	112	130	114	104	175	175	6,2	def
80	78	116	126	116	108	178	178	6,7	def
80	78	125	126	90	108	170	180	7,2	def
80	78	125	126	95	110	170	180	7,7	def
82	80	125	126	100	115	170	180	8,2	def
85	85	125	126	110	115	170	180	8,7	def

Anexo 2.2 Datos globales de ángulos para el primer programa

	Ángulos máximos	Ángulos mínimos	Incremento
ang0	103	80	23
ang1	103	78	25
ang2	125	112	13
ang3	130	115	15
ang4	116	90	26
ang5	115	98	17
ang6	178	165	13
ang7	180	171	9

- ANEXO 3. DATOS DE ENTRADA PARA EL CÁLCULO DE LOS ENGRANAJES

Solid Edge Engineering Reference

PARÁMETROS DE ENTRADA DE DISEÑO**PARÁMETROS DE ENGRANAJE**

Relación de transmisión deseada: 2,417

(al) Ángulo de presión: 20,00 °

(bt) Ángulo de hélice: 0,00 °

Módulo: 0,75 mm

Dirección de ángulo de hélice: Izquierdo

(r) Acuerdo de raíz: 0,20 mm

Corrección total de la unidad: 1

PARÁMETROS DE CÁLCULO

Eficiencia: 0,8

Piñón:

No. de dientes: 12

(DM) Diámetro de agujero de montaje: 2,00 mm

(b) Anchura de cara: 8,00 mm

Velocidad: 600,000 grados/s

Par: 1,000 N·m

Engranaje:

No. de dientes: 29

(DM) Diámetro de agujero de montaje: 2,00 mm

(b) Anchura de cara: 10,00 mm

OPCIONES DE ENTRADA DE DISEÑO

Tipo de tren de engranajes: Tren de engranajes externo

Parámetros geométricos de salida: Buscar distancia al centro

Método de cálculo de resistencia: Según tensión de plegado simple

Cálculo de carga: Buscar potencia

Tipo de cálculo de resistencia: Buscar resist mat en factor seguridad mínimo

Valores de anchura de cara: Especificada por el usuario

Colocar con Relación de conjunto completa: No

COTAS BÁSICAS

Piñón:

(db) Diámetro base: 8,46 mm

(da) Diámetro exterior: 10,50 mm

(d) Diámetro primitivo: 9,00 mm

(dr) Diámetro de raíz: 7,13 mm

Diámetro primitivo de trabajo: 9,00 mm

(T) Espesor de cuerda: 1,04 mm

(ht) Altura de espesor de cuerda: 0,56 mm

(M) Cota de la cuerda: 5,54 mm

(S) Espesor del diente: 1,41 mm

Engranaje:

(db) Diámetro base: 20,44 mm

(da) Diámetro exterior: 23,25 mm

(d) Diámetro primitivo: 21,75 mm

(dr) Diámetro de raíz: 19,88 mm

Diámetro primitivo de trabajo: 21,75 mm

(T) Espesor de cuerda: 1,04 mm

(ht) Altura de espesor de cuerda: 0,56 mm

(M) Cota de la cuerda: 5,54 mm

(S) Espesor del diente: 1,41 mm

**- ANEXO 4. PARÁMETROS RECOMENDADOS DE IMPRESIÓN**

Basic		Advanced	Plugins	Start/End-GCode
Quality				
Layer height (mm)	0.10			
Shell thickness (mm)	0.9			
Enable retraction	<input checked="" type="checkbox"/>	...		
Fill				
Bottom/Top thickness (mm)	0.12			
Fill Density (%)	40	...		
Speed and Temperature				
Print speed (mm/s)	50			
Printing temperature (C)	210			
Bed temperature (C)	60			
Support				
Support type	None	...		
Platform adhesion type	None	...		
Filament				
Diameter (mm)	1,736			
Flow (%)	96			
Machine				
Nozzle size (mm)	0.3			
Retraction				
Speed (mm/s)	40.0			
Distance (mm)	4.5			
Quality				
Initial layer thickness (mm)	0.12			
Initial layer line width (%)	96			
Cut off object bottom (mm)	0.0			
Dual extrusion overlap (mm)	0.15			
Speed				
Travel speed (mm/s)	120.0			
Bottom layer speed (mm/s)	40			
Infill speed (mm/s)	70			
Top/bottom speed (mm/s)	40			
Outer shell speed (mm/s)	35			
Inner shell speed (mm/s)	60			
Cool				
Minimal layer time (sec)	5			
Enable cooling fan	<input checked="" type="checkbox"/>	...		

- **ANEXO 5. CÓDIGO**

Anexo 5.1 Código completo

Este anexo muestra el código en su conjunto. Los anexos consecutivos son extractos del código precedidos por un organigrama indicando su funcionamiento.

```
1 /*----- Septiembre de 2016
2 RoboJop-----
3 **** -----
4 Autor: Jose Garcia Manzanaro
5 Tutora: Beatriz Garcia Vasallo
6 Proyecto: Robot Bipedo Microcontrolado
7 Trabajo Fin de Grado
8 Universidad de Salamanca
9 Contacto: josegm@usal.es
10
11 English
12 The code below is able to control autonomously a biped robot with 8 degrees of freedom, specially
13 desing to be generated by 3D printing. This code control the movement step by step by a gyroscope
14 MPU5060 and can avoid obstacles with an ultrasonic sensor HC-SR04.
15
16 Español
17 El siguiente código es capaz de controlar de forma autónoma un robot bipedo de 8 grados de libertad
18 diseñado para ser generado por una impresora 3D. Este código controla el movimiento posición a posición
19 con ayuda de un giroscopio MPU5060 y evita obstáculos con ayuda de un sensor de ultrasonidos HC-SR04.
20 **** -----
21 */
22
23 //***** -----
24 // Comienzo de la definición de variables y constantes **** -----
25 //***** -----
26
27
28 ///////////////
29 // Definición de includes
30 ///////////////
31
32 #include <Wire.h>
33 #include <MPU6050.h>
34 #include <Servo.h>
35
36 ///////////////
37 // Declaración de constantes para el MPU
38 ///////////////
39 const float pi = 3.1415926535897932384626433832795;
40 const float GraRad = 180 / pi;
41
42 //El acelerómetro devuelve los valores que registra en un rango de números entre -16384 y +16384. Para
43 //conocer la aceleración exacta hay que dividir el valor devuelto por el acelerómetro entre la sensibilidad
44 //establecida en 2g.
45 //La siguiente constante se utiliza para obtener los valores de la aceleración y obtener la inclinación
46 //con resultado en radianes.
47 const float AaR = 16384.00; //de Acelerómetro A Radianes
48
49 //El giroscopio devuelve los valores registrados en el mismo rango que el acelerómetro. La sensibilidad
50 //esta establecida en +/-250°/s.
51 //Para determinar la velocidad angular se multiplica el valor registrado por la siguiente constante:
52 //+/-250°/s = 32768/250= 131.072 datos*s°
53 const float GaG = 131.072; //de Giroscopio A Grados
54
```

```
55 //////////////////////////////////////////////////////////////////
56 // Declaracion de variables del MPU////////////////////////////////////////////////////////////////
57 //////////////////////////////////////////////////////////////////
58 MPU6050 mpu; // Definicion de la estructura de datos tipo MPU6050
59
60 int16_t ax, ay, az; //int16 define un integer con signo
61 int16_t gx, gy, gz;
62 float balanceoA, cabeceoA, balanceoG, cabeceoG, guinadaG, balanceo, cabeceo, guinada;
63 float ganancia = 0.9; //expresa el nivel de aceptacion entre el giroscopio y el acelerometro
64
65
66 //Valores que ajustan la inclinacion del MPU para que se corresponda con la cabeza
67 float calibracionCabeceo = -1, calibracionBalanceo = 2.8;
68
69 int tiempoChequeo = 250; //en milisegundos. Tiempo para el que se considera estable la señal del MPU
70 int dt = 5; //intervalo de tiempo por chequeo en milisegundos
71
72 //-----Declaracion de variables del Ultrasonidos-----
73 const int pinEco = 3;
74 const int pinDisparo = 2;
75 const int distanciaSeguridad = 15; //distancia en cm a la que se detecta el obstaculo
76
77 //-----Declaracion de variables de los LEDs -----
78 const int pinRojo13 = 13;
79 const int pinVerde12 = 12;
80
81 //-----Relacion entre las definiciones de xxx0 y la rotula correspondiente
82 /**
83 /*
84 Las referencias relativas a los numeros se corresponden con articulaciones de la siguiente forma:
85 xxx0 => pie izquierdo
86 xxx1 => pie derecho
87 xxx2 => rotula del tobillo izquierdo
88 xxx3 => rotula del tobillo derecho
89 xxx4 => rotula de la rodilla izquierda
90 xxx5 => rotula de la rodilla derecha
91 xxx6 => rotula de la cabeza izquierda
92 xxx7 => rotula de la cabeza derecha
93
94 Los angulos de las articulaciones tambien se denominan de la siguiente forma:
95 Alfas son las articulaciones de los pies, Betas los tobillos, Gamma las rodillas
96 y Delta los de la cabeza los enumerados como 1 son las articulaciones de la
97 parte izquierda y los enumerados como 2 son las articulaciones de la derecha.
98 P.Ej beta2 -> tobillo derecho
99 */
100
101 //-----TODOS LOS ANGULOS SON EN GRADOS
102 //-----
```

```
105 //-----Declaracion de pines de los servos-----
106 const int pinSer0 = 9; //Pin del servo 0 (pie izq)
107 const int pinSer1 = 6; //Pin del servo 1
108 const int pinSer2 = 8; //Pin del servo 2
109 const int pinSer3 = 7; //Pin del servo 3
110 const int pinSer4 = 10; //Pin del servo 4
111 const int pinSer5 = 5; //Pin del servo 5
112 const int pinSer6 = 11; //Pin del servo 6
113 const int pinSer7 = 4; //Pin del servo 7
114
115 //-----Declaracion de los servos -----
116 Servo ser0; // pie izq
117 Servo ser1; // pie der
118 Servo ser2; // r1 izq. r1 es la rotula del tobillo
119 Servo ser3; // r1 der
120 Servo ser4; // r2 izq. r2 es la rotula de la rodilla
121 Servo ser5; // r2 der
122 Servo ser6; // r3 izq. r3 es la rotula de la cabeza
123 Servo ser7; // r3 der
124
125 //----- Angulos entre las articulaciones -----
126 int ang0; // ang= angulo de la articulacion
127 int ang1;
128 int ang2;
129 int ang3;
130 int ang4;
131 int ang5;
132 int ang6;
133 int ang7;
134
135 //-----
136 // Ajustes iniciales
137 //-----
138 /*
139 Calibramos el robot cada vez que lo montamos modificando el valor de ValIni.
140 angIni es el angulo que tiene la articulacion para la posicion inicial.
141 Por ejemplo, con ayuda de un transportador se coloca la articulacion en el angulo angIni y se
142 cambia el valor de valIni para que este valor del servo coincida con el valor de la articulacion.
143 */
144 //----- Valores iniciales de los servos -----
145 //Valores para la posicion estatica. Son valores caracteristicos del montaje
146 int valIni0 = 110; // Pie izquierdo. Aumentar es a la derecha
147 int valIni1 = 112; // Pie derecho. Aumentar es a la derecha
148 int valIni2 = 90; // R1 izq Aumentar el valor es abrir la articulacion
149 int valIni3 = 77; // R1 der Aumentar el valor cerrar la articulacion
150 int valIni4 = 72; // R2 izq Aumentar el valor es cerrar la articulacion
151 int valIni5 = 121; // R2 der Aumentar el valor es abrir la articulacion
```

```
152 int valIni6 = 71; // R3 izq Aumentar el valor es abrir la articulacion
153 int valIni7 = 131; // R3 der Aumentar el valor es cerrar la articulacion
154
155 //----- Valores propios de las articulaciones -----
156 /*
157   Fijamos el angulo que queramos que tengan las articulaciones para la posicion de equilibrio,
158   esto es independientemente de como hallamos montado el robot
159 */
160 int angIni0 = 90;
161 int angIni1 = 90;
162 int angIni2 = 121;
163 int angIni3 = 121;
164 int angIni4 = 114;
165 int angIni5 = 114;
166 int angIni6 = 175;
167 int angIni7 = 175;
168
169 //Se establece el sentido de giro para que abra o cierre como debe.
170 int sentido0 = +1;
171 int sentido1 = +1;
172 int sentido2 = +1;
173 int sentido3 = -1;
174 int sentido4 = -1;
175 int sentido5 = +1;
176 int sentido6 = +1;
177 int sentido7 = -1;
178
179 //valor de la reduccion para cada articulacion dada
180 float redu0 = 1.0;
181 float redu1 = 1.0;
182 float redu2 = 2.4;
183 float redu3 = 2.4;
184 float redu4 = 2.4;
185 float redu5 = 2.4;
186 float redu6 = 1.0;
187 float redu7 = 1.0;
188
189 //----- Variables globales de datos del giroscopo -----
190 int inicioA1; //giroscopoAlfa para la posicion de inicio. Pie. Referencia del giroscopo
191 int inicioA2;
192 int inicioD1; //giroscopioDelta para la posicion de inicio. Cabeza
193 int inicioD2;
194
195 //referencia para los pies tomada cuando se termina de inclinar hacia un lado con irHasta()
196 int incliA1;
197 int incliA2;
198
199 //----- Variables para el control de velocidad de ejecucion -----
200 int velDef = 200; //velocidad por defecto del programa principal (Caminar1)
201 int velDef2 = 200; //velocidad pro defecto del programa 2 (Caminar2)
202
```

```
203 //----- Funcion para reiniciar el sistema -----
204 void(* reset) (void) = 0; //resetea. Regresa a la posicion 0 de memoria

205
206 //----- Variables del selector de ejecucion -----
207 //Selecciona la forma de caminar
208 char estado = 0;
209 boolean preguntarCaminarPorDefecto = 1; //1 --> por defecto en Caminar 1. 0 --> espera a recibir el
210 //numero del Caminar a ejecutar
211

212 //*****
213 // Fin de la definicion de variables y constantes. ****
214 //*****
215
216
217
218 //*****
219 // Comienzo del establecimiento de definiciones e inicializaciones para la ejecucion ****
220 //*****
221
222 void setup() {
223     //----- Deficion de pines de los LEDs -----
224     pinMode(pinRojo13, OUTPUT);
225     pinMode(pinVerde12, OUTPUT);
226     //pinMode(pinAzul3, OUTPUT);
227
228     // ----- Definicion de pines del SENSOR DE ULTRASONIDOS HC-SR04 -----
229     pinMode(pinDisparo, OUTPUT);
230     pinMode(pinEco, INPUT);
231
232     //----- Iniciar canal de comunicacion -----
233     Serial.begin(9600); //para detectar fallos en el programa y la seleccion de ejecucion
234     Serial.println("Conectado");
235
236     // ----- Inicializacion del giroscopo-----
237     mpu.initialize();
238     // Solo para depuracion
239     //Serial.println(mpu.testConnection() ? "Connected" : "Connection failed");
240
241     //----- Asignacion de pines de los servos -----
242     ser0.attach(pinSer0); //Cable negro
243     ser1.attach(pinSer1); //Cable amarillo
244     ser2.attach(pinSer2); //Cable verde
245     ser3.attach(pinSer3); //Cable rojo
246     ser4.attach(pinSer4); //Cable verde
247     ser5.attach(pinSer5); //Cable verde
248     ser6.attach(pinSer6); //Cable verde
249     ser7.attach(pinSer7); //Cable rojo
250 }
```



```
251 //*****  
252 // Fin del establecimiento de definiciones e inicializaciones para la ejecucion. *****  
253 //*****  
254  
255 //*****  
256 // Comienzo de la ejecucion *****  
257 //*****  
258 //*****  
259 //*****  
  
260 void loop() {  
261     //----- Seleccion de Caminar -----  
262     if (preguntarCaminarPorDefecto == 1) {//para entrar por defecto al caminar1 si se activa el boolean  
263         Caminar1();  
264     }  
265     Serial.println("Introduzca el numero del programa que desea ejecutar");  
266     do {  
267         estado = Serial.read(); // El bucle permanece hasta que le llega uno de los caracteres 1, 2 o 3.  
268     } while (estado != '1' and estado != '2' and estado != '3'); //cuando estado no es igual a 1, 2 ni 3  
269             //repite  
270  
271     //pasa al programa de caminar elegido  
272     if (estado == '1') {  
273         Caminar1();  
274     }  
275     else if (estado == '2') {  
276         Caminar2();  
277     }  
278     else if (estado == '3') {  
279         Caminar3();  
280     }  
281 }  
282 //*****  
283 // Fin de la ejecucion (loop) *****  
284 //*****
```

```
285 //////////////////////////////////////////////////////////////////
286 // Camina con giroscopo y evita obstaculos ****//*****
287 //////////////////////////////////////////////////////////////////
288 //////////////////////////////////////////////////////////////////
289 void Caminar1() {
290     Calibracion(); //va a la posicion de inicio.
291     delay (500); // este valor se aumenta cuando se quieren calibrar los servos
292     posicionInicio(90, 90, 121, 121, 114, 114, 175, 175, 2000); //la posicion de prueba
293     //Bucle infinito: se inclina, da una zancada a la derecha, se inclina, zancada a la
294     //izquierda, se inclina, y así, sucesivamente
295     //Al final de cada zancada chequea si hay algún obstaculo y lo rodea
296     while (true) {
297         irHasta(-8);      // - es inclinar a la izquierda y + es inclinar a la derecha
298         zancadaDerecha();
299         evitarDesdeDerecha();
300         irHasta(8);
301         zancadaIzquierda();
302         evitarDesdeIzquierda();
303     }
304 }
305
306 //////////////////////////////////////////////////////////////////
307 // Camina rápido para demostracion ****//*****
308 //////////////////////////////////////////////////////////////////
309 void Caminar2() {
310     Calibracion(); //va a la posicion de inicio
311     delay(1000);
312 }
```

```
313 //da media zancada hasta situarse en una posición que pueda enlazar con la siguiente
314 while (true) {
315     IrAP2( 90,  90, 121, 121, 114, 114, 175, 175, velDef2);
316     IrAP2( 93,  93, 121, 121, 114, 114, 175, 175, velDef2);
317     IrAP2( 96,  96, 121, 121, 114, 114, 175, 175, velDef2);
318     IrAP2( 99,  99, 121, 121, 114, 114, 175, 175, velDef2);
319     IrAP2(101, 101, 121, 121, 114, 114, 175, 175, velDef2);
320     IrAP2(103, 103, 121, 121, 114, 114, 175, 175, velDef2);
321 }
322
323 IrAP2(103, 102, 121, 118, 114, 110, 175, 175, velDef2);
324 IrAP2(103, 101, 121, 115, 114, 106, 175, 175, velDef2);
325 IrAP2(103, 100, 121, 118, 114, 102, 175, 175, velDef2);
326 IrAP2(100,  99, 118, 124, 114,  98, 185, 173, velDef2);
327 IrAP2( 99,  99, 116, 126, 114, 102, 187, 171, velDef2);
328 IrAP2( 99,  99, 114, 128, 114, 102, 180, 172, velDef2);
329 IrAP2( 99,  99, 112, 130, 114, 102, 175, 174, velDef2);
330 IrAP2( 97,  96, 112, 130, 114, 102, 175, 175, velDef2);
331 IrAP2( 95,  93, 112, 130, 114, 103, 175, 175, velDef2);
332 IrAP2( 80,  80, 112, 130, 114, 104, 175, 175, velDef2);
333 IrAP2( 80,  78, 116, 126, 116, 108, 173, 178, velDef2);
334 IrAP2( 80,  78, 116, 126, 110, 108, 173, 178, velDef2);
335 IrAP2( 80,  78, 116, 126, 105, 108, 173, 178, velDef2);
336 IrAP2( 80,  78, 125, 126,  92, 108, 170, 180, velDef2);
337 IrAP2( 80,  78, 125, 126,  91, 108, 168, 180, velDef2);

337 IrAP2( 80,  78, 125, 126,  91, 108, 168, 180, velDef2);
338 IrAP2( 80,  78, 125, 126,  90, 108, 167, 180, velDef2);
339 IrAP2( 80,  78, 130, 126,  89, 108, 164, 180, velDef2);
340 IrAP2( 80,  78, 135, 126,  88, 108, 160, 180, velDef2);
341 IrAP2( 81,  79, 140, 125,  87, 108, 160, 180, velDef2);
342 IrAP2( 82,  81, 140, 123,  88, 108, 160, 180, velDef2);
343 IrAP2( 82,  82, 140, 122,  89, 108, 160, 180, velDef2);
344 IrAP2( 82,  82, 140, 121,  91, 108, 160, 178, velDef2);
345 IrAP2( 85,  85, 140, 118,  91, 108, 165, 175, velDef2);
346 IrAP2( 88,  88, 140, 115,  91, 108, 170, 172, velDef2);
347 IrAP2( 90,  90, 139, 113,  94, 111, 175, 169, velDef2);
348 IrAP2( 90,  90, 137, 111,  97, 115, 175, 169, velDef2);
349 IrAP2( 90,  90, 135, 108, 100, 118, 175, 169, velDef2);
350 IrAP2( 90,  90, 132, 105, 105, 121, 175, 169, velDef2);
351 IrAP2( 92,  92, 130, 105, 108, 125, 175, 172, velDef2);
352 IrAP2( 94,  94, 127, 105, 110, 127, 175, 174, velDef2);
353 IrAP2( 97,  97, 120, 105, 112, 129, 175, 174, velDef2);
354 IrAP2(100, 100, 120, 115, 115, 125, 175, 175, velDef2);
355 IrAP2(103, 103, 121, 121, 115, 120, 175, 175, velDef2);
356 IrAP2(103, 103, 121, 121, 114, 114, 175, 175, velDef2);
357 }
358
359 //----- programa vacante -----
360 void Caminar3() {
361     arcoIris();
362 }
```

```
363 //////////////////////////////////////////////////////////////////
364 // Fuciones auxiliares ****
365 // -----
366 //-----Ayuda a calibrar el robot-----
367 void Calibracion() {
368     ser0.write(valIni0);
369     ser1.write(valIni1);
370     ser2.write(valIni2);
371     ser3.write(valIni3);
372     ser4.write(valIni4);
373     ser5.write(valIni5);
374     ser6.write(valIni6);
375     ser7.write(valIni7);
376 }
377

378 //-----Posicion de inicio con datos proporcionados por el giroscopo-----
379 //-----El siguiente código posiciona la cabeza y los pies paralelos al suelo-----
380 void posicionInicio(int ang0, int ang1, int ang2,
381                     int ang3, int ang4, int ang5,
382                     int ang6, int ang7, int tiempoInicio) {
383
384     boolean i;
385     int contadorParaPasarBalanceo = 0;
386     int contadorParaPasarCabeceo = 0;
387     byte necesariosParaPasar = 30; //con 30 va bien, con 50 muy bien, pero tarda demasiado
388     byte margenError = 1;          //+- grados de margen de error en la colocacion
389     float giroAlfa1 = ang0 - balanceo,
390           giroAlfa2 = ang1 - balanceo,
391           giroDelta1 = ang6,
392           giroDelta2 = ang7;
393
394 //*****
395 //do while, inicia un contador de balanceo y otro de cabeceo. El siguiente codigo hace que se necesiten
396 //tres condiciones para pasar:
397 //    -que pasen 2 segundos
398 //    -que se halla llegado a la posicion de equilibrio en balanceo (+-1°) 30 veces
399 //    -que se halla llegado a la posicion de equilibrio en cabeceo (+-1°) 30 veces
400 //*****
401
402 do {
403     //tiempoInicio=2000ms/dt=5ms hace un minimo de 400 calculos.
404     for (int n = 0; n < tiempoInicio / dt; n++) {
405         mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
406         //Balanceo y cabeceo obtenidos de la aceleracion
407         balanceoA = atan((az / AaR) / sqrt(pow((ay / AaR), 2) + pow((ax / AaR), 2))) * GraRad
408             + calibracionBalanceo;
409         cabeceoA = atan((ay / AaR) / sqrt(pow((ax / AaR), 2) + pow((az / AaR), 2))) * GraRad
410             + calibracionCabeceo;
411         //Con este if hacemos que la primera señal del balanceo sea la del acelerometro y el resto
412         //provengan del resultado del filtro. Se consigue un proceso algo mas rapido.
413         if (i == 0) {
414             balanceoG = balanceoA + (gz / GaG) * dt / 1000;
415             cabeceoG = cabeceoA + (gy / GaG) * dt / 1000;
416             i++;
417         }
418     }
```

```
419 //angulo=anguloAnterior+ (velocidadAngular/131.072)*dt dt corresponde con el delay
420 else {
421     balanceoG = balanceo + (gz / GaG) * dt / 1000; //gz*sensibilidad. GaR (de giroscopo a Grados) [GaR=1
422     cabeceoG = cabeceo + (gy / GaG) * dt / 1000; //sensibilidad] es la inversa de la sensibilidad
423 }
424 balanceo = (ganancia * balanceoG + (1 - ganancia) * balanceoA);
425 cabeceo = (ganancia * cabeceoG + (1 - ganancia) * cabeceoA);
426 delay(dt);
427

428 -----
429 // Estabilizacion -----
430 -----
431 /*
432     Cambia la posición de la cabeza 1 de cada 10 veces que se toma la posicion
433     La toma de datos va mucho mas rapido que la respuesta de los servos. Estos tienen una
434     velocidad de 6°/10ms pero un arranque lento, si cambiaron de posicion cada 5 ms, se
435     produciría un cabeceo indeseado.
436     Cada 10 veces se estabiliza bien.
437 */
438 if (n % 10 == 0) {
439     // ----- Gestion del balanceo -----
440     if (balanceo < -margenError) {

441         giroAlfa1--;
442         giroAlfa2--;
443     }
444     else if (balanceo > margenError) {
445         giroAlfa1++;
446         giroAlfa2++;
447     }
448     else {
449         contadorParaPasarBalanceo++;
450     }
451     // ----- Gestion del cabeceo -----
452     if (cabeceo > margenError) {
453         giroDelta1--;
454         giroDelta2--;
455     }
456     else if (cabeceo < -margenError) {
457         giroDelta1++;
458         giroDelta2++;
459     }
460     else {
461         contadorParaPasarCabeceo++;
462     }
463
464     IrA(giroAlfa1, giroAlfa2, ang2, ang3, ang4, ang5, giroDelta1, giroDelta2);
465 }
466 }
467 ledOff();
```

```
468     // Deja pasar la ejecucion cuando se alcanza el minimo requerido en los contadores
469 } while (contadorParaPasarBalanceo < necesariosParaPasar
470     and contadorParaPasarCabeceo < necesariosParaPasar);
471
472 //-----
473 // inicioAlfa e inicioDelta se usaran en la siguiente funcion.
474 // El 0.5 es para redondear. Sin el 0.5 simplemente se truncan los decimales (6.9->6)
475 // pero con +0.5 se redondea => 6.9+0.5=7.4->7.
476 //-----
477
478 inicioA1 = giroAlfa1 + 0.5;
479 inicioA2 = giroAlfa2 + 0.5;
480 inicioD1 = giroDelta1 + 0.5;
481 inicioD2 = giroDelta2 + 0.5;
482 }
483
484 //----- Desplaza la pierna derecha adelante -----
485 void zancadaDerecha() {
486     ledOff();
487     irAPaso(incliA1,      incliA2 - 1, 121, 118, 114, 110, 175, 175, velDef);
488     irAPaso(incliA1,      incliA2 - 2, 121, 115, 114, 106, 175, 177, velDef);
489     irAPaso(incliA1,      incliA2 - 2, 121, 118, 114, 102, 175, 180, velDef);
490     irAPaso(incliA1 - 3, incliA2 - 2, 118, 120, 114, 98, 185, 183, velDef);
491     irAPaso(incliA1 - 4, incliA2 - 2, 116, 125, 114, 98, 187, 183, velDef);
492     irAPaso(incliA1 - 4, incliA2 - 2, 114, 125, 114, 98, 180, 183, velDef);
493     irAPaso(incliA1 - 4, incliA2 - 2, 112, 125, 114, 98, 175, 180, velDef);
494     irAPaso(incliA1 - 7, incliA2 - 2, 112, 125, 114, 98, 175, 177, velDef);
495
496     irAPaso(incliA1 - 8, incliA2 - 10, 112, 125, 114, 100, 175, 175, velDef);
497     irAPaso(inicioA1,      inicioA2 , 112, 126, 114, 103, 175, 175, velDef);
498 }
499 //----- Detecta un obstaculo y gira a la izquierda-----
500 void evitarDesdeDerecha() {
501     long cm = ping();
502     if (cm > 15 or cm == 0) {
503         verde();
504         delay(400);
505         ledOff();
506     }
507     else {
508         rojo();
509         delay(400);
510         ledOff();
511         delay(400);
512         rojo();
513
514     //Realiza el movimiento de giro a la izquierda 8 veces
515     for (int i = 0; i < 8; i++) {
516         giroDesdeDerecha();
517         irHasta(-10);
518         zancadaDerecha();
519         ledOff();
520     }
521 }
522 }
```

```
524 //----- Gira desde zancadaDerecha -----
525 //Esta posicion hace rotar el robot y enlaza con una posicion desde la que zancada derecha
526 //puede avanzar.
527 void giroDesdeDerecha() {
528     irAPaso(inicioA1, inicioA2 , 120, 120, 114, 114, 175, 175, velDef);
529 }
530
531 //----- Desplaza la pierna izquierda adelante -----
532 void zancadaIzquierda() {
533     ledOff();
534     irAPaso(incliA1,      incliA2,      120, 130, 114, 104, 175, 175, velDef);
535     irAPaso(incliA1,      incliA2,      120, 126, 116, 104, 174, 178, velDef);
536     irAPaso(incliA1,      incliA2,      120, 126, 110, 104, 173, 178, velDef);
537     irAPaso(incliA1,      incliA2,      120, 126, 105, 104, 172, 178, velDef);
538     irAPaso(incliA1,      incliA2,      125, 126, 100, 104, 171, 180, velDef);
539     irAPaso(incliA1,      incliA2,      125, 126, 100, 104, 170, 180, velDef);
540     irAPaso(incliA1,      incliA2,      130, 126, 100, 104, 170, 180, velDef);
541     irAPaso(incliA1,      incliA2,      130, 126, 97, 104, 170, 180, velDef);
542     irAPaso(incliA1,      incliA2,      135, 126, 97, 104, 170, 180, velDef);
543     irAPaso(incliA1 + 1, incliA2,      135, 125, 97, 104, 171, 179, velDef);
544     irAPaso(incliA1 + 2, incliA2 + 1, 135, 125, 97, 104, 172, 178, velDef);
545     irAPaso(incliA1 + 2, incliA2 + 2, 132, 121, 100, 104, 173, 177, velDef);
546     irAPaso(incliA1 + 4, incliA2 + 4, 129, 119, 100, 104, 174, 176, velDef);
547     irAPaso(incliA1 + 6, incliA2 + 6, 126, 118, 100, 104, 175, 175, velDef);
548 }
549
550 //Si detecta un obstaculo a menos de 15 cm lo evita
551 void evitarDesdeIzquierda() {
552     long cm = ping();
553     if (cm > 15 or cm == 0) {
554         verde();
555         delay(400);
556         ledOff();
557     }
558     else {
559         rojo();
560         delay(400);
561         ledOff();
562         delay(400);
563         rojo();
564
565         //realiza el movimiento de giro a la derecha 8 veces
566         for (int i = 0; i < 8; i++) {
567             giroDesdeIzquierda();
568             irHasta(10);
569             zancadaIzquierda();
570             ledOff();
571         }
572     }
573 }
574 //----- Desplaza la pierna izquierda adelante -----
575 void giroDesdeIzquierda() {
576     irAPaso(inicioA1, inicioA1, 118, 126, 105, 100, 175, 175, velDef);
577     irAPaso(inicioA1, inicioA1, 120, 125, 105, 100, 175, 178, velDef);
578 }
579
```

```
580 //////////////////////////////////////////////////////////////////
581 // Funciones de movimiento //////////////////////////////////////////////////////////////////
582 //////////////////////////////////////////////////////////////////
583 /*
584     Las siguientes funciones reciben los angulos de las articulaciones. Alfas son los servos de los pies,
585     Betas los tobillos, Gamma las rodillas y Delta los de la cabeza. Los enumerados como 1 son las
586     articulaciones de la parte izquierda y los enumerados como 2 son las articulaciones de la derecha.
587 */
588 //----IrAPaso() se le proporcionan los angulos entre las articulaciones y coloca los servos en posicion--
589 void irAPaso(int alfa1, int alfa2,
590             int beta1, int beta2,
591             int gamma1, int gamma2,
592             int delta1, int delta2,
593             int velocidad) {
594     testCaida(); //detecta si se ha caido y se reinicia
595     tomarDatosMPU(); //devuelve balanceo y cabecero
596     IrA( alfa1, alfa2, beta1, beta2, gamma1, gamma2, delta1, delta2);
597     //el testeо de la inclinaciоn lleva un tiempo (tiempoChequeo). El siguiente if sirve para comparar el
598     //tiempo de espera que debe tener con el tiempo que tarda en hacer la toma de datos del MPU.
599     //Si la toma de datos tarda mas que el tiempo de espera no hace delay, y si tarda menos, se
600     //calcula la diferencia entre ambos
601     if (velocidad - tiempoChequeo > 0) {
602         delay(velocidad - tiempoChequeo);
603     }
604 }
605

606 //-----IrA() pasa el angulo de las articulaciones a los servos-----
607 void IrA(int alfa1, int alfa2,
608           int beta1, int beta2,
609           int gamma1, int gamma2,
610           int delta1, int delta2) {
611
612     //Coloca cada servo en el angulo al que se le ha mandado
613     ser0.write(red0 * sentido0 * (alfa1 - angIni0) + valIni0);
614     ser1.write(red1 * sentido1 * (alfa2 - angIni1) + valIni1);
615     ser2.write(red2 * sentido2 * (beta1 - angIni2) + valIni2);
616     ser3.write(red3 * sentido3 * (beta2 - angIni3) + valIni3);
617     ser4.write(red4 * sentido4 * (gamma1 - angIni4) + valIni4);
618     ser5.write(red5 * sentido5 * (gamma2 - angIni5) + valIni5);
619     ser6.write(red6 * sentido6 * (delta1 - angIni6) + valIni6);
620     ser7.write(red7 * sentido7 * (delta2 - angIni7) + valIni7);
621 }

622 //-----IrAP2 solo se utiliza en el Caminar2-----
623
624
625 //Coloca cada servo en el angulo al que se le ha mandado y espera un tiempo "velocidad"
626 void IrAP2(int alfa1, int alfa2,
627             int beta1, int beta2,
628             int gamma1, int gamma2,
629             int delta1, int delta2,
630             int velocidad) {
631     IrA( alfa1, alfa2, beta1, beta2, gamma1, gamma2, delta1, delta2);
632     delay(velocidad);
633 }
```

```
635 //-----irHasta-----  
636 //Es una rutina para ir hasta un angulo en el eje Y, en el balanceo, controlado solo por los  
637 //servos de los pies  
638  
639 void irHasta(int angHasta) {  
640     int alfa1 = inicioA1;  
641     int alfa2 = inicioA2;  
642     if (angHasta < 0) {  
643         do {  
644             tomarDatosMPU();  
645             // se modifica balanceo  
646             alfa1++;  
647             alfa2++;  
648             ser0.write(red0 * sentido0 * (alfa1 - angIni0) + valIni0);  
649             ser1.write(red1 * sentido1 * (alfa2 - angIni1) + valIni1);  
650         } while (balanceo > angHasta);  
651     }  
652     else if (angHasta > 0) {  
653         do {  
654             tomarDatosMPU();  
655             alfa1--;  
656             alfa2--;  
657             ser0.write(red0 * sentido0 * (alfa1 - angIni0) + valIni0);  
658             ser1.write(red1 * sentido1 * (alfa2 - angIni1) + valIni1);  
659         } while (balanceo < angHasta);  
660     }  
661     incliA1 = alfa1;  
662     incliA2 = alfa2;  
663 }  
664  
665 //////////////////////////////// Otras funciones ///////////////////////////////  
666 ////////////////////////////////  
667 ////////////////////////////////  
668  
669 //-----Obtencion datos del giroscopo-----  
670 void tomarDatosMPU() {  
671  
672     //Codigo basado en los siguientes blogs: http://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/  
673     //y http://www.circuitmagic.com/arduino/control-dc-motor-cwccw-with-mpu-6050-gyroaccelerometer-arduino/  
674  
675     int i;  
676     for (int n = 0; n < tiempoChequeo / dt; n++) { //tiempoChequeo=300ms/dt=5 hacen son 60 calculos.  
677         //Con estos valores se obtiene una buena precision en un periodo de tiempo corto  
678         mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);  
679         //Balanceo y cabeceo obtenidos de la aceleracion  
680         balanceoA = atan((az / AaR) / sqrt(pow((ay / AaR), 2) + pow((ax / AaR), 2))) * GraRad + calibracionBalanceo;  
681         cabeceoA = atan((ay / AaR) / sqrt(pow((ax / AaR), 2) + pow((az / AaR), 2))) * GraRad + calibracionCabeceo;  
682 }
```

```
683 //Con este if hacemos que la primera señal del balanceo sea la del acelerometro y el resto provengan
684 //del resultado del filtro, asi hacemos que sea algo mas rapido
685 if (i == 0) {
686     balanceoG = balanceoA + (gz / GaG) * dt / 1000; //angulo=anguloAnterior+ (velocidadAngular/131.072)*dt
687     cabeceoG = cabeceoA + (gy / GaG) * dt / 1000; //angulo=anguloAnterior+ (velocidadAngular/131.072)*dt
688     i++;
689 }
690 else {
691     balanceoG = balanceo + (gz / GaG) * dt / 1000; //gz*sensibilidad. GaR (de giroscopio a Grados)
692     cabeceoG = cabeceo + (gy / GaG) * dt / 1000; ///[GaR=1/sensibilidad] es la inversa de la sensibilidad
693 }
694 //Filtro complementario. Funciona bien para una ganancia de 0.9
695 balanceo = (ganancia * balanceoG + (1 - ganancia) * balanceoA);
696 cabeceo = (ganancia * cabeceoG + (1 - ganancia) * cabeceoA);
697 delay(dt);
698 }
699 }

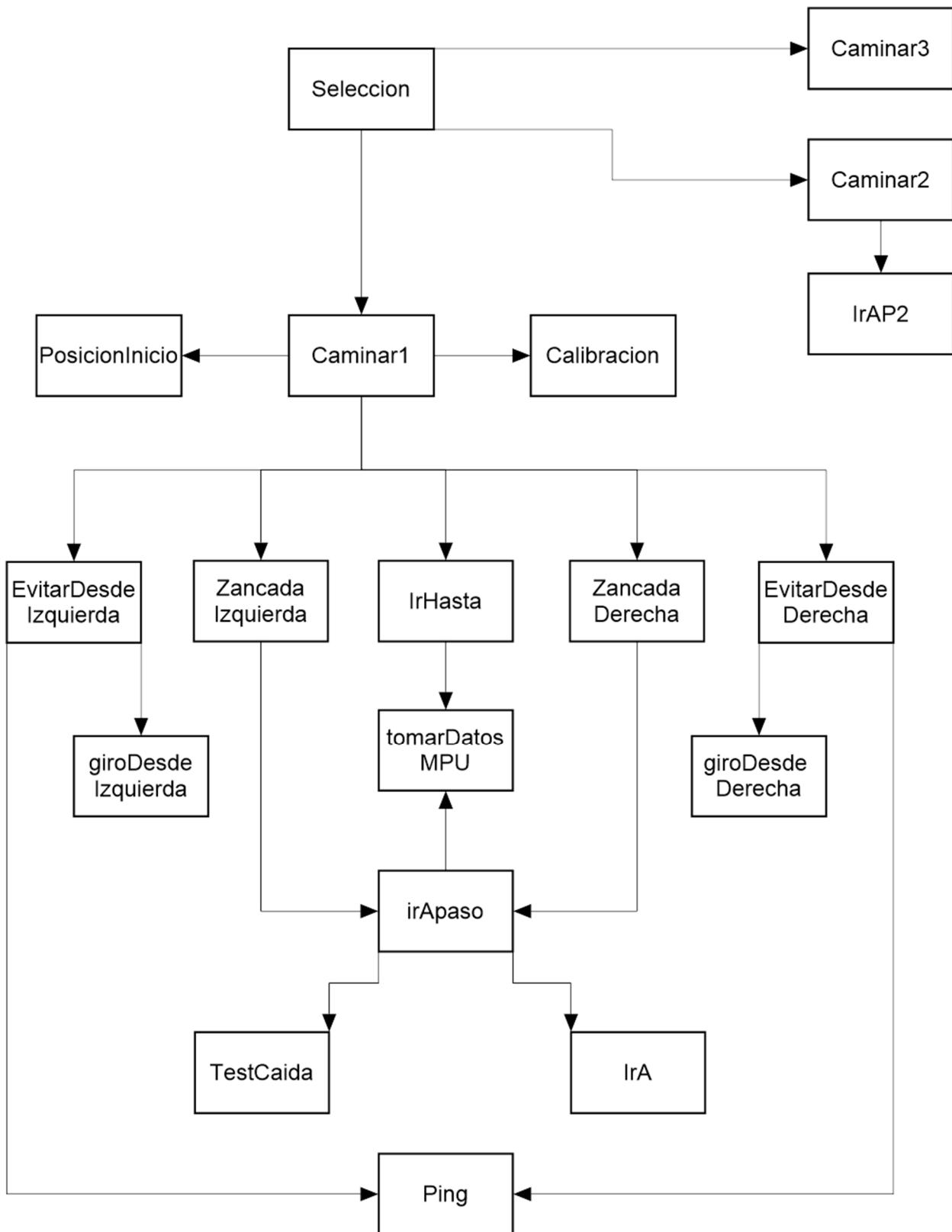
701 //-----Codigo del sensor de ultrasonidos. Devuelve la distancia que hay con el objeto de delante-----
702 long ping() {
703     long duracion, distanciaEnCm;
704
705     digitalWrite(pinDisparo, LOW); //para generar un pulso limpio ponemos a LOW 4us
706     delayMicroseconds(4);
707     digitalWrite(pinDisparo, HIGH); //generamos el disparo de 10us
708     delayMicroseconds(10);
709     digitalWrite(pinDisparo, LOW);
710
711     duracion = pulseIn(pinEco, HIGH); //medimos el tiempo entre el pulso enviado y el recibido,
712     //en microsegundos
713     distanciaEnCm = duracion / 58; //29*2=58. *donde 29 es la velocidad del sonido medido en
714     //cm/microsegundos.
715
716     return distanciaEnCm;
717 }

719 // ----- Detecta la perdida de estabilidad (angulo de inclinacion > 45°) -----
720 void testCaida() { //detecta que se ha caido, luce todo, parpadea y se reinicia
721     if (balanceo < -45 or 45 < balanceo or cabeceo < -45 or cabeceo > 45) {
722         rojo();
723         delay(500);
724         ledOff();
725         delay(500);
726         rojo();
727         delay(500);
728         reset(); //se reinicia
729     }
730 }
```

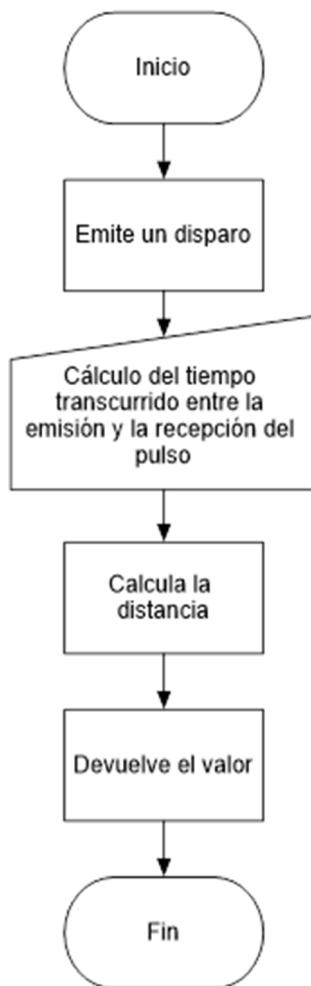
```
731 //////////////////////////////////////////////////////////////////
732 ////////////// Funciones de colores /////////////////////
733 //////////////// Se definen y configuran colores, así como rutinas para apagarlos
734 //////////////////////////////////////////////////////////////////
735 void ledOn() {
736   color(255, 255, 255);
737 }
738 void rojo() {
739   color(255, 0, 0);
740 }
741 void verde() {
742   color (0, 255, 0);
743 }
744 void azul() {
745   color (0, 0, 255);
746 }
747 void morado() {
748   color(255, 0, 200);
749 }
750 void cian() {
751   color (0, 255, 250);
752 }
753
754 void ledOff() {
755   color(0, 0, 0);
756 }
757 }

758 void rojoOff() {
759   analogWrite(pinRojo13, 0);
760 }
761 void verdeOff() {
762   analogWrite(pinVerde12, 0);
763 }
764 //void azulOff() {
765 //analogWrite(3, 0); } // Pin 3 ocupado
766
767 void arcoIris() {
768   int tiempoArcoiris = 3000; //fue empleado para chequear los contactos de los Leds al iniciar el programa
769   color(255, 0, 0);
770   delay(tiempoArcoiris / 3);
771   color(0, 255, 0);
772   delay(tiempoArcoiris / 3);
773   color(0, 0, 255);
774   delay(tiempoArcoiris / 3);
775   ledOff();
776 }
777
778 //Se definen los pines a los que van conectados los LEDs
779 void color(byte red, byte green, byte blue) {
780   analogWrite(pinRojo13, red);
781   analogWrite(pinVerde12, green);
782   //analogWrite(3, blue); //desnombrar y asociar si se conecta.
783 }
```

Anexo 5.2. Diagrama de contexto

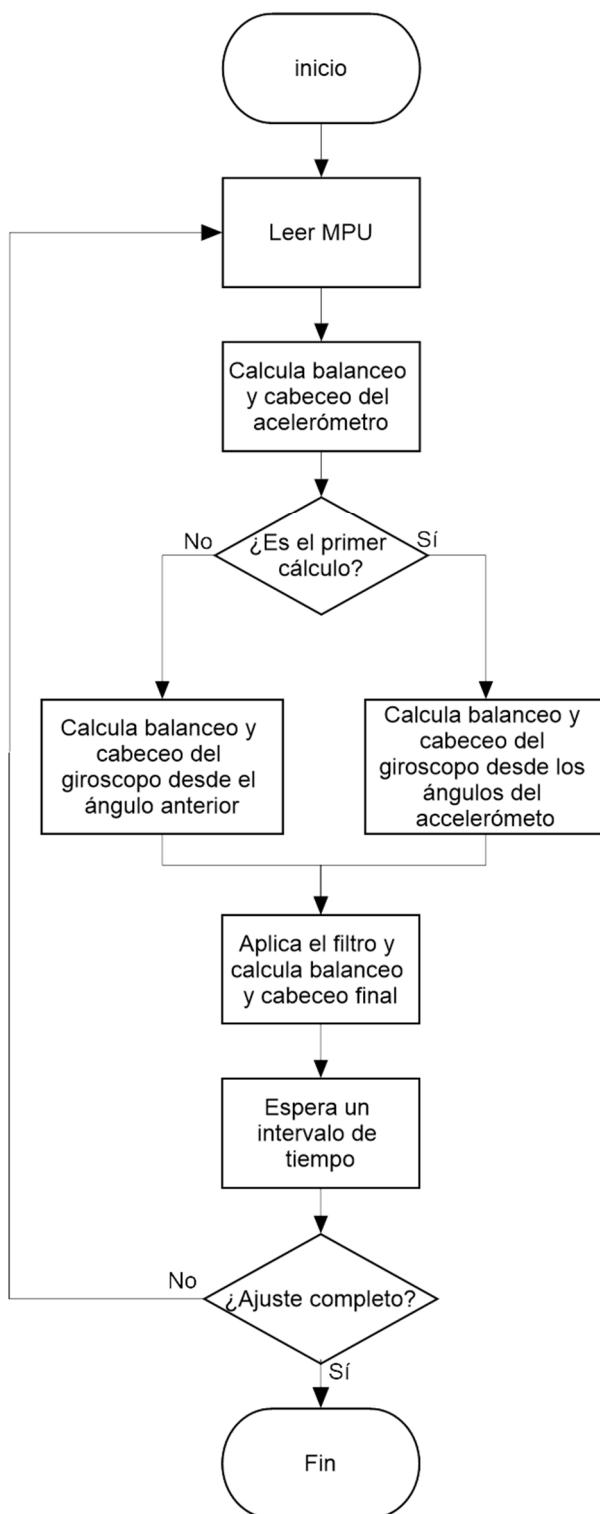


Anexo 5.3. Organigrama y código del ultrasonidos



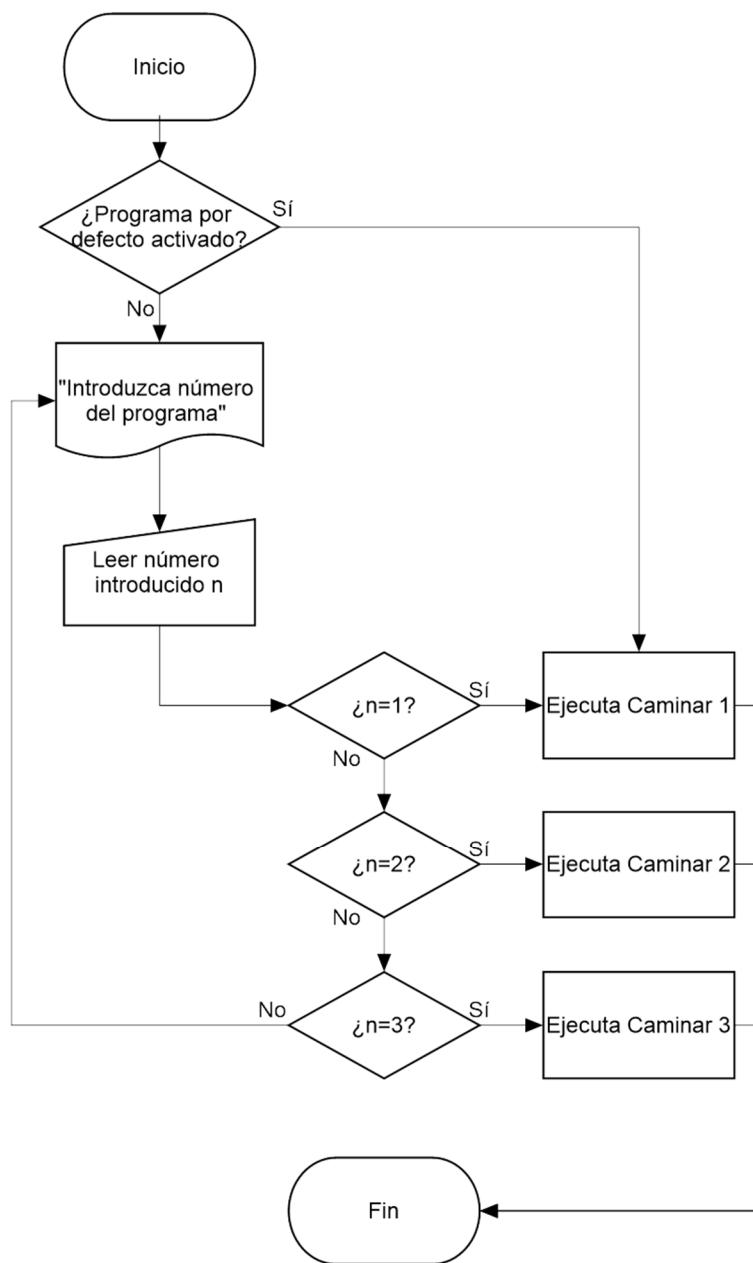
```
701 //-----Codigo del sensor de ultrasonidos. Devuelve la distancia que hay con el objeto de delante-----
702 long ping() {
703     long duracion, distanciaEnCm;
704
705     digitalWrite(pinDisparo, LOW); //para generar un pulso limpio ponemos a LOW 4us
706     delayMicroseconds(4);
707     digitalWrite(pinDisparo, HIGH); //generamos el disparo de 10us
708     delayMicroseconds(10);
709     digitalWrite(pinDisparo, LOW);
710
711     duracion = pulseIn(pinEco, HIGH); //medimos el tiempo entre el pulso enviado y el recibido,
712     //en microsegundos
713     distanciaEnCm = duracion / 58; //29*2=58. *donde 1/29 es la velocidad del sonido medida en
714     //cm/microsegundos.
715
716     return distanciaEnCm;
717 }
718 }
```

Anexo 5.4. Organigrama y código de la rutina “tomarDatosMPU”



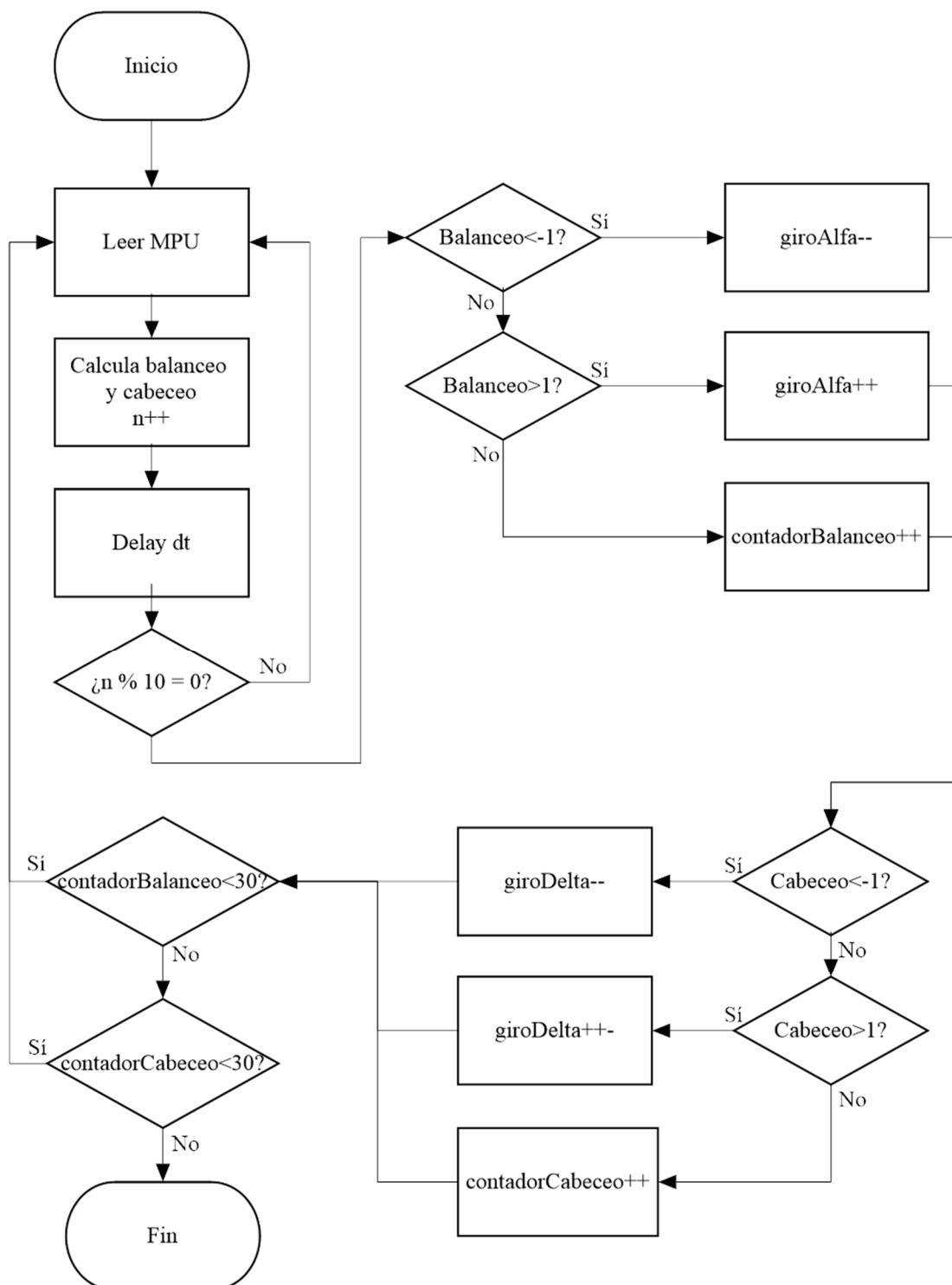
```
669 //-----Obtencion datos del giroscopo-----
670 void tomarDatosMPU() {
671
672 //Codigo basado en los siguientes blogs: http://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/
673 //y http://www.circuitmagic.com/arduino/control-dc-motor-cwccw-with-mpu-6050-gyroaccelerometer-arduino/
674
675 int i;
676 for (int n = 0; n < tiempoChequeo / dt; n++) { //tiempoChequeo=250ms/dt=5 hacen son 50 calculos.
677 //Con estos valores se obtiene una buena precision en un periodo de tiempo corto
678 mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
679 //Balanceo y cabeceo obtenidos de la aceleracion
680 balanceoA = atan((az / AaR) / sqrt(pow((ay / AaR), 2) + pow((ax / AaR), 2))) * GraRad + calibracionBalanceo;
681 cabeceoA = atan((ay / AaR) / sqrt(pow((ax / AaR), 2) + pow((az / AaR), 2))) * GraRad + calibracionCabeceo;
682
683 //Con este if hacemos que la primera señal del balanceo sea la del acelerometro y el resto provengan
684 //del resultado del filtro, asi hacemos que sea algo mas rapido
685 if (i == 0) {
686     balanceoG = balanceoA + (gz / GaG) * dt / 1000; //angulo=anguloAnterior+ (velocidadAngular/131.072)*dt
687     cabeceoG = cabeceoA + (gy / GaG) * dt / 1000; //angulo=anguloAnterior+ (velocidadAngular/131.072)*dt
688     i++;
689 }
690 else {
691     balanceoG = balanceo + (gz / GaG) * dt / 1000; //gz*sensibilidad. GaR (de giroscopio a Grados)
692     cabeceoG = cabeceo + (gy / GaG) * dt / 1000; //#[GaR=1/sensibilidad] es la inversa de la sensibilidad
693 }
694 //Filtro complementario. Funciona bien para una ganancia de 0.9
695 balanceo = (ganancia * balanceoG + (1 - ganancia) * balanceoA);
696 cabeceo = (ganancia * cabeceoG + (1 - ganancia) * cabeceoA);
697 delay(dt);
698 }
699 }
```

Anexo 5.5. Organigrama y código de la rutina “Selector”



```
261 //----- Seleccion de Caminar -----
262 if (preguntarCaminarPorDefecto == 1) {//para entrar por defecto al caminar1 si se activa el boolean
263   Caminar1();
264 }
265 Serial.println("Introduzca el numero del programa que desea ejecutar");
266 do {
267   estado = Serial.read(); // El bucle permanece hasta que le llega uno de los caracteres 1, 2 o 3.
268 } while (estado != '1' and estado != '2' and estado != '3'); //cuando estado no es igual a 1, 2 ni 3
269 //repite
270
271 //pasa al programa de caminar elegido
272 if (estado == '1') {
273   Caminar1();
274 }
275 else if (estado == '2') {
276   Caminar2();
277 }
278 else if (estado == '3') {
279   Caminar3();
280 }
281 }
```

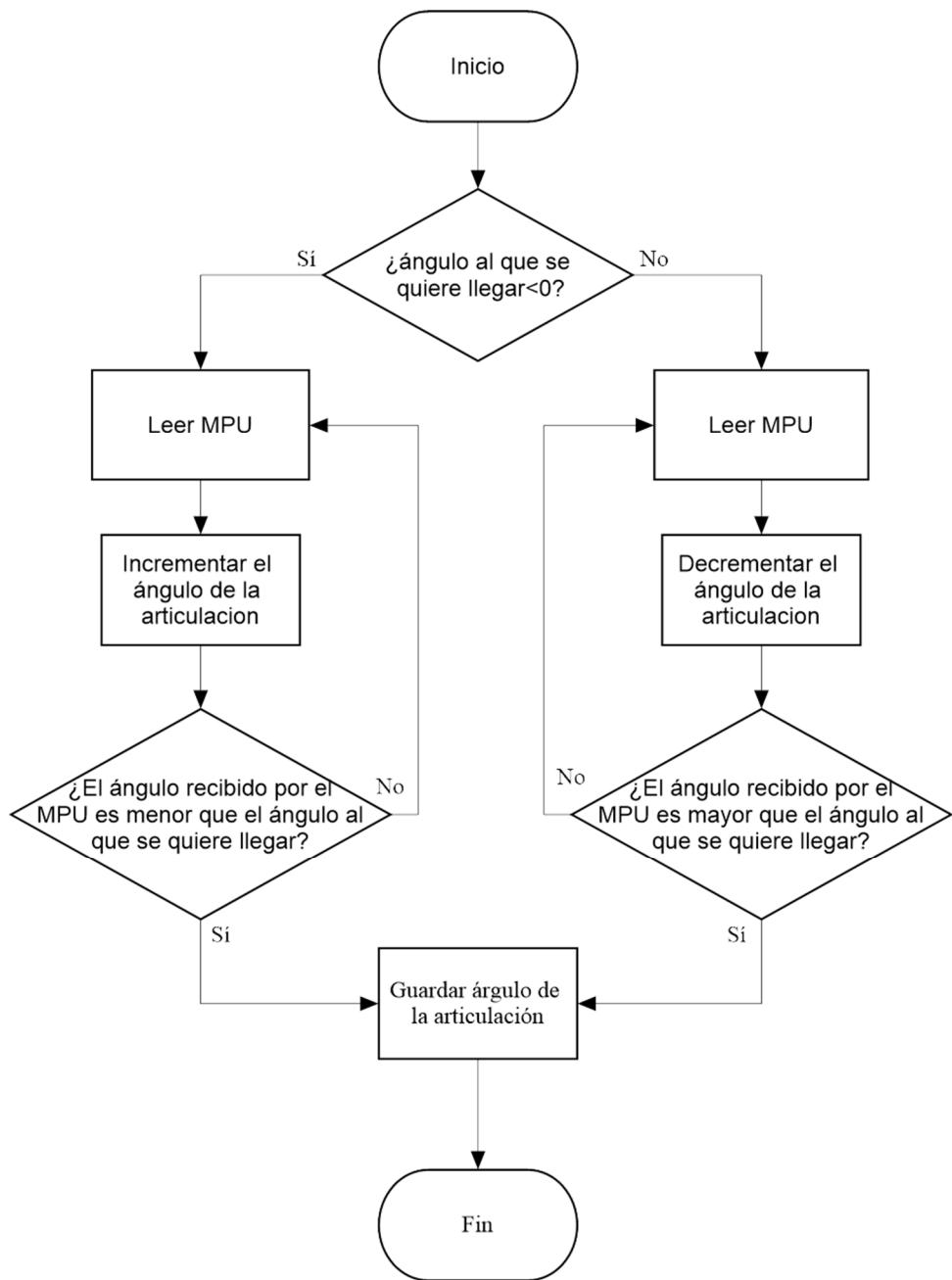
Anexo 5.6. Organigrama y código de la rutina “posicionInicio”



```
378
379 //-----Posicion de inicio con datos proporcionados por el giroscopo-----
380 //-----El siguiente código posiciona la cabeza y los pies paralelos al suelo-----
381 void posicionInicio(int ang0, int ang1, int ang2,
382                     int ang3, int ang4, int ang5,
383                     int ang6, int ang7, int tiempoInicio) {
384
385     boolean i;
386     int contadorParaPasarBalanceo = 0;
387     int contadorParaPasarCabeceo = 0;
388     byte necesariosParaPasar = 30; //con 30 va bien, con 50 muy bien, pero tarda demasiado
389     byte margenError = 1;           //+- grados de margen de error en la colocacion
390     float giroAlfa1 = ang0 - balanceo,
391         giroAlfa2 = ang1 - balanceo,
392         giroDelta1 = ang6,
393         giroDelta2 = ang7;
394 //*****
395 /////////////////
396 //do while, inicia un contador de balanceo y otro de cabeceo. El siguiente codigo hace que se necesiten
397 //tres condiciones para pasar:
398 //      -que pasen 2 segundos
399 //      -que se halla llegado a la posicion de equilibrio en balanceo (+-1°) 30 veces
400 //      -que se halla llegado a la posicion de equilibrio en cabeceo (+-1°) 30 veces
401 ///////////////////
402
403 do {
404     //tiempoInicio=2000ms/dt=5ms hace un minimo de 400 calculos.
405     for (int n = 0; n < tiempoInicio / dt; n++) {
406         mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
407         //Balanceo y cabeceo obtenidos de la aceleracion
408         balanceoA = atan((az / AaR) / sqrt(pow((ay / AaR), 2) + pow((ax / AaR), 2))) * GraRad
409             + calibracionBalanceo;
410         cabeceoA = atan((ay / AaR) / sqrt(pow((ax / AaR), 2) + pow((az / AaR), 2))) * GraRad
411             + calibracionCabeceo;
412         //Con este if hacemos que la primera señal del balanceo sea la del acelerometro y el resto
413         //provengan del resultado del filtro. Se consigue un proceso algo mas rapido.
414         if (i == 0) {
415             balanceoG = balanceoA + (gz / GaG) * dt / 1000;
416             cabeceoG = cabeceoA + (gy / GaG) * dt / 1000;
417             i++;
418         }
419         //angulo=anguloAnterior+ (velocidadAngular/131.072)*dt  dt corresponde con el delay
420         else {
421             balanceoG = balanceo + (gz / GaG) * dt / 1000; //gz*sensibilidad. GaR (de giroscopo a Grados) [GaR=1
422             cabeceoG = cabeceo + (gy / GaG) * dt / 1000; //sensibilidad] es la inversa de la sensibilidad
423         }
424         balanceo = (ganancia * balanceoG + (1 - ganancia) * balanceoA);
425         cabeceo = (ganancia * cabeceoG + (1 - ganancia) * cabeceoA);
426         delay(dt);
427     }
```

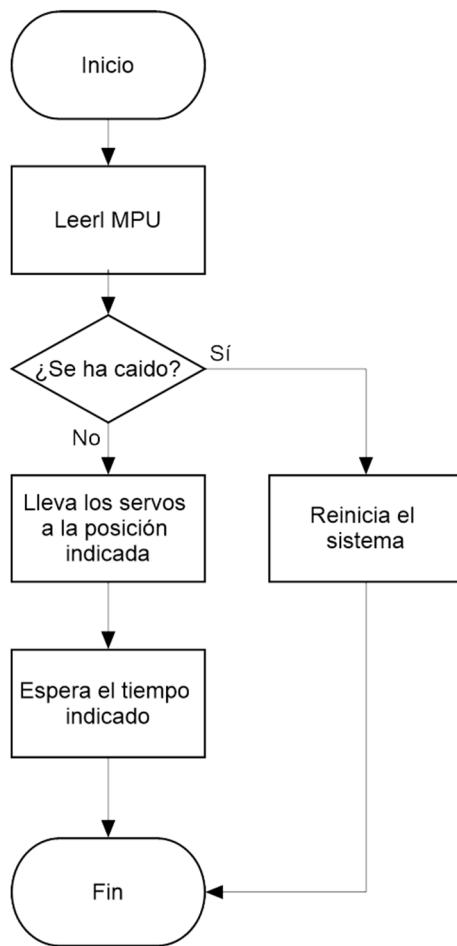
```
428 //-----
429 // Estabilizacion -----
430 //-----
431 /*
432     Cambia la posición de la cabeza 1 de cada 10 veces que se toma la posición
433     La toma de datos va mucho más rápido que la respuesta de los servos. Estos tienen una
434     velocidad de 6°/10ms pero un arranque lento, si cambiaron de posición cada 5 ms, se
435     produciría un cabeceo indeseado.
436     Cada 10 veces se estabiliza bien.
437 */
438 if (n % 10 == 0) {
439     // ----- Gestión del balanceo -----
440     if (balanceo < -margenError) {
441         giroAlfa1--;
442         giroAlfa2--;
443     }
444     else if (balanceo > margenError) {
445         giroAlfa1++;
446         giroAlfa2++;
447     }
448     else {
449         contadorParaPasarBalanceo++;
450     }
451     // ----- Gestión del cabeceo -----
452     if (cabeceo > margenError) {
453         giroDelta1--;
454         giroDelta2--;
455     }
456     else if (cabeceo < -margenError) {
457         giroDelta1++;
458         giroDelta2++;
459     }
460     else {
461         contadorParaPasarCabeceo++;
462     }
463
464     IrA(giroAlfa1, giroAlfa2, ang2, ang3, ang4, ang5, giroDelta1, giroDelta2);
465 }
466 }
467 ledOff();
468
469 // Deja pasar la ejecución cuando se alcanza el mínimo requerido en los contadores
470 } while (contadorParaPasarBalanceo < necesariosParaPasar
471 and contadorParaPasarCabeceo < necesariosParaPasar);
472
473 //-----
474 // inicioAlfa e inicioDelta se usarán en la siguiente función.
475 // El 0.5 es para redondear. Sin él 0.5 simplemente se truncan los decimales (6.9->6)
476 // pero con +0.5 se redondea => 6.9+0.5=7.4->7.
477 //-----
478 inicioA1 = giroAlfa1 + 0.5;
479 inicioA2 = giroAlfa2 + 0.5;
480 inicioD1 = giroDelta1 + 0.5;
481 inicioD2 = giroDelta2 + 0.5;
482 }
483 }
```

Anexo 5.7. Organigrama y código de la rutina “irHasta”



```
635 //-----irHasta-----  
636 //Es una rutina para ir hasta un angulo en el eje Y, en el balanceo, controlado solo por los  
637 //servos de los pies  
638  
639 void irHasta(int angHasta) {  
640     int alfa1 = inicioA1;  
641     int alfa2 = inicioA2;  
642     if (angHasta < 0) {  
643         do {  
644             tomarDatosMPU();  
645             // se modifica balanceo  
646             alfa1++;  
647             alfa2++;  
648             ser0.write(red0 * sentido0 * (alfa1 - angIni0) + valIni0);  
649             ser1.write(red1 * sentido1 * (alfa2 - angIni1) + valIni1);  
650         } while (balanceo > angHasta);  
651     }  
652     else if (angHasta > 0) {  
653         do {  
654             tomarDatosMPU();  
655             alfa1--;  
656             alfa2--;  
657             ser0.write(red0 * sentido0 * (alfa1 - angIni0) + valIni0);  
658             ser1.write(red1 * sentido1 * (alfa2 - angIni1) + valIni1);  
659         } while (balanceo < angHasta);  
660     }  
661     incliA1 = alfa1;  
662     incliA2 = alfa2;  
663 }
```

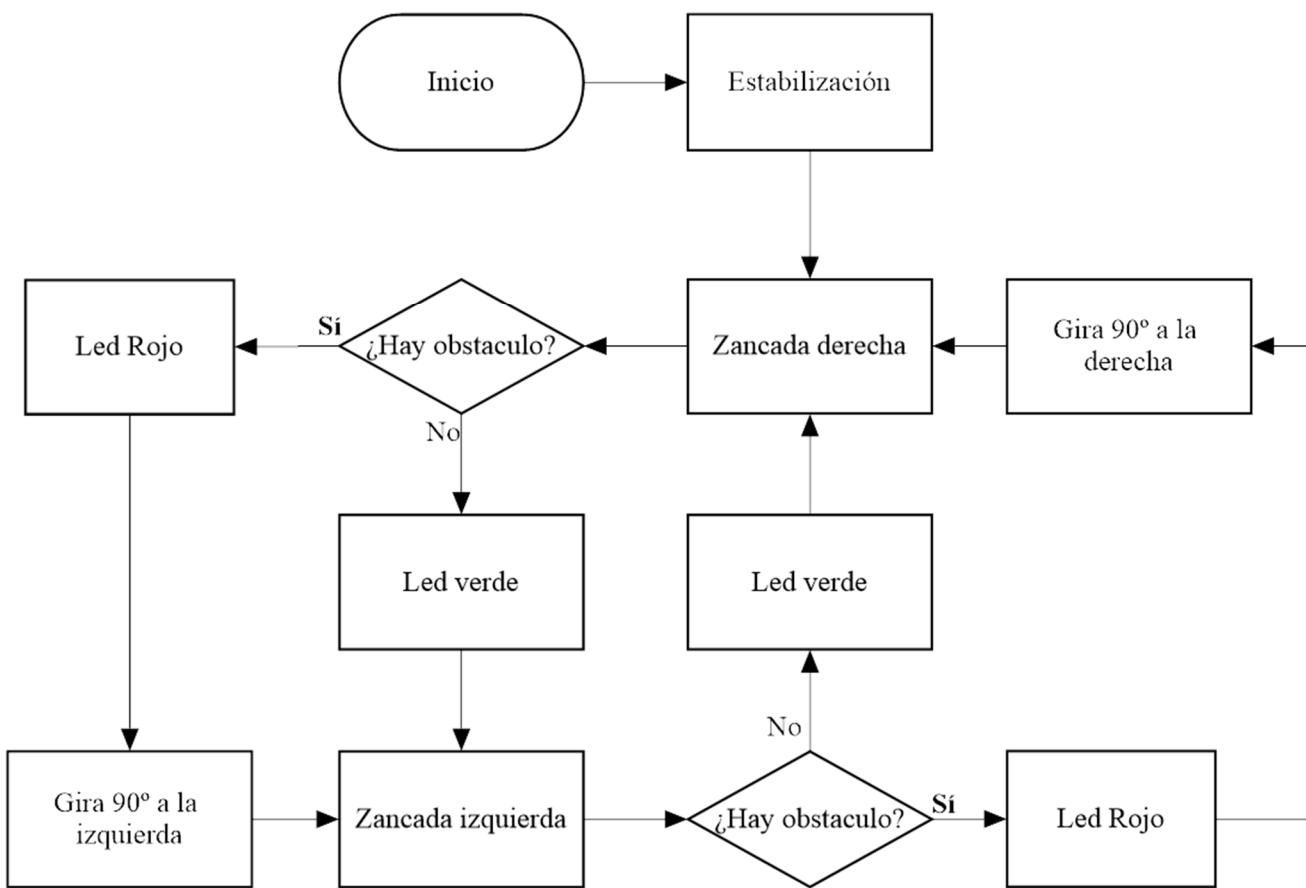
Anexo 5.8. Organigrama y código de la rutina “irAPaso”



```

588 //----IrAPaso() se le proporcionan los angulos entre las articulaciones y coloca los servos en posicion--
589 void irAPaso(int alfa1, int alfa2,
590               int beta1, int beta2,
591               int gamma1, int gamma2,
592               int delta1, int delta2,
593               int velocidad) {
594   testCaida(); //detecta si se ha caido y se reinicia
595   tomarDatosMPU(); //devuelve balanceo y cabeceo
596   IrA( alfa1, alfa2, beta1, beta2, gamma1, gamma2, delta1, delta2);
597   //el testeo de la inclinacion lleva un tiempo (tiempoChequeo). El siguiente if sirve para comparar el
598   //tiempo de espera que debe tener con el tiempo que tarda en hacer la toma de datos del MPU.
599   //Si la toma de datos tarda mas que el tiempo de espera no hace delay, y si tarda menos, se
600   //calcula la diferencia entre ambos
601   if (velocidad - tiempoChequeo > 0) {
602     delay(velocidad - tiempoChequeo);
603   }
604 }
  
```

Anexo 5.9 Organigrama y código de la rutina “Caminar1”



```

287 // Camina con giroscopo y evita obstaculos ****
288 ///////////////////////////////////////////////////
289 void Caminar1() {
290   Calibracion(); //va a la posicion de inicio.
291   delay (500); // este valor se aumenta cuando se quieren calibrar los servos
292   posicionInicio(90, 90, 121, 121, 114, 114, 175, 175, 2000); //la posicion de prueba
293   //Bucle infinito: se inclina, da una zancada a la derecha, se inclina, zancada a la
294   //izquierda, se inclina, y así, sucesivamente
295   //Al final de cada zancada chequea si hay algún obstáculo y lo rodea
296   while (true) {
297     irHasta(-8); // - es inclinar a la izquierda y + es inclinar a la derecha
298     zancadaDerecha();
299     evitarDesdeDerecha();
300     irHasta(8);
301     zancadaIzquierda();
302     evitarDesdeIzquierda();
303   }
304 }
  
```

- **ANEXO 6. TABLA DE TIEMPOS Y PESOS DE IMPRESIÓN.**

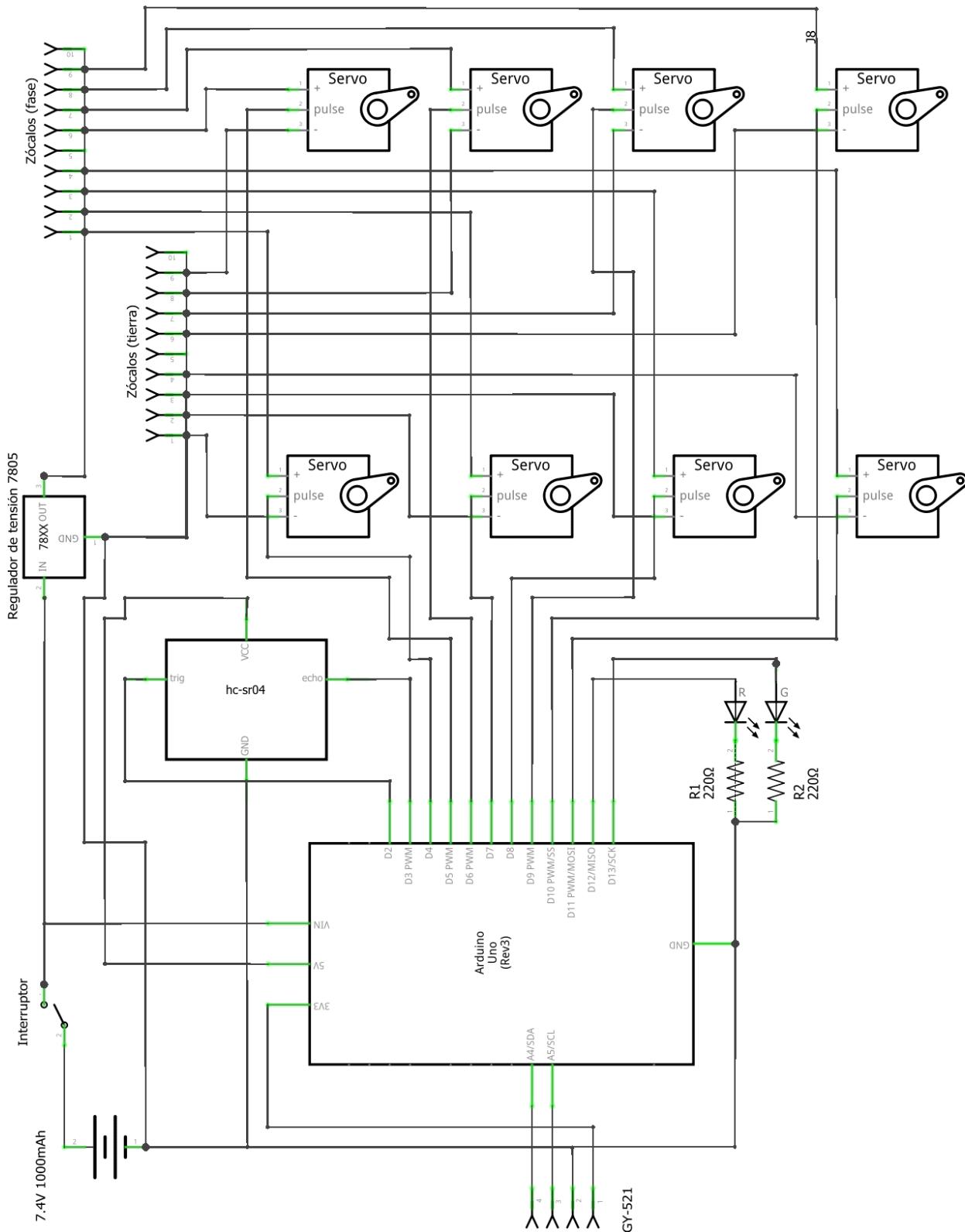
Datos proporcionados por el programa “Cura”

Denominación de la pieza	Tiempo de impresión		Peso	Cantidad	Algunos datos de entrada	
	h	min	g		Relleno	40%
1.01a PIE.par	1	17	7	1	Boquilla	0,3mm
1.01b PIE.par	1	17	7	1	Altura de capa	0,1mm
1.02 SOPORTE PIE.par	0	13	1	2	Revestimiento	0,9mm
1.03 Piñón.par	0	10	1	4		
1.04 EMPEINE.par	1	35	6	1		
1.04 EMPEINE_mir.par	1	35	6	1		
1.05 ESPINILLA.par	3	23	14	1		
1.05 ESPINILLA_mir.par	3	23	14	1		
1.06 FEMUR.par	2	32	11	1		
1.06 FEMUR_mir.par	2	32	11	1		
1.07 CAJA CABEZA.par	6	46	31	1		
1.08 APOYOS SERVOS.par	0	39	3	4		
Tiempo total (h)		Peso total (g)				
28,03333333		125				

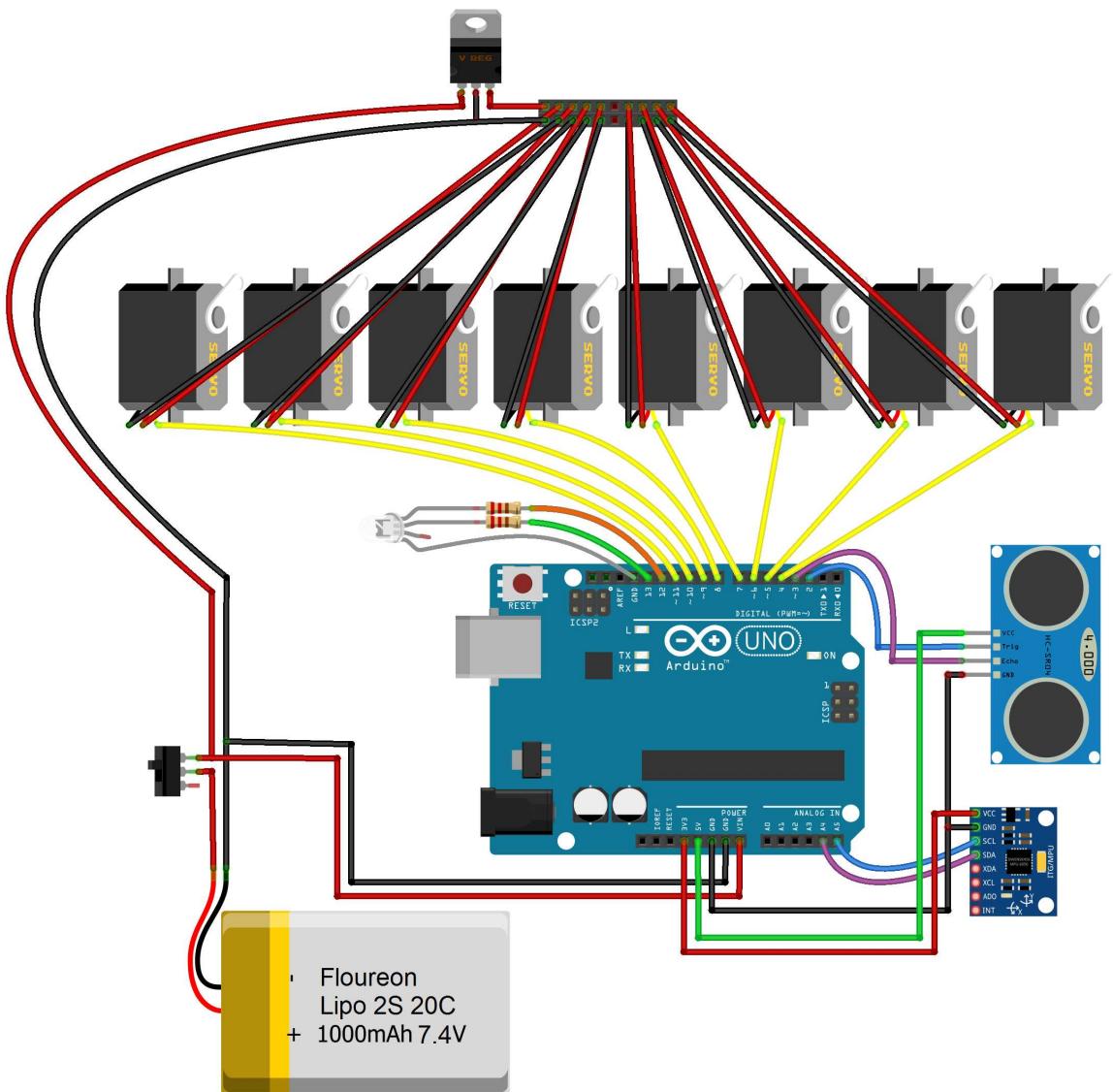
- ANEXO 7. ESQUEMAS

Esquemas generados con el programa “*Fritzing*”

Anexo 7.1 Esquema eléctrico



Anexo 7.2. Esquema gráfico



REFERENCIAS

-
- [1] Esteban Viso *El origen de la palabra robot.* Enero 2016
<http://m.xatakaciencia.com/robotica/el-origen-de-la-palabra-robot>
 - [2] Real Academia Española
<http://dle.rae.es/?id=WYRlhzm>
 - [3] Escuela técnica Superior de Ingeniería de Sistemas Informáticos. Universidad Politécnica de Madrid
https://www.etsisi.upm.es/museo_virtual/origenes/jmjacquard
 - [4] Comunicación en visita (2010) y Centro de Automática y Robótica (CAR)
CSIC-UPM.
 - [5] Honda Motor Co.
<http://world.honda.com/ASIMO/technology/2000/index.html>
 - [6] Distribuidor Oficial de Nao
<http://aliverobots.com/nao>
 - [7] E. Ackerman and E. Guizzo Spectrum *Boston Dynamics Now Belongs to Google*
IEEE 14 Dec 2013
<http://spectrum.ieee.org/automaton/robotics/military-robots/boston-dynamics-now-belongs-to-google>
 - [8] E. Ackerman, *ATLAS DRC Robot Is 75 Percent New, Completely Unplugged.*
Spectrum IEEE, 20 junio 2015:
<http://spectrum.ieee.org/automaton/robotics/military-robots/atlas-drc-robot-is-75-percent-new-completely-unplugged>
 - [9] Página oficial de *Boston Dynamics*.
www.bostondynamics.com
 - [10] Página oficial del proyecto *ICub*
<http://www.icub.org/>
 - [11] E. Guizzo *Researchers Build Fast Running Robot Inspired by Velociraptor*
Spectrum IEEE. 29 May 2014
<http://spectrum.ieee.org/automaton/robotics/robotics-hardware/fast-running-biped-robot-based-on-velociraptor>
 - [12] Comunicación en conferencia en la Escuela Politécnica Superior de Zamora. 2014

- [13] Especificaciones de los plásticos. hxx, proovedor de sistemas de impresión 3D
 - <http://hxx.es/2015/03/12/materiales-de-impresion-3d-i-pla-acido-polilactico/>
 - <http://hxx.es/2015/03/23/materiales-de-impresion-3d-ii-abs-acrilonitrilo-butadieno-estireno/>
- [14] Página oficial de Arduino.
<https://www.arduino.cc/en/Reference/Comparison>
- [15] Página oficial de Fritzing
<http://fritzing.org/projects/mpu-6050-board-gy-521-acelerometro-y-giroscopio>
- [16] Página oficial del fabricante (*invensense*) del MPU6050
<https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/>
- [17] Página especializada en tutoriales de Arduino
<http://www.prometec.net/bus-i2c/>
- [18] Página de los laboratorios SANDIA
http://www.sandia.gov/mstc/mems_info/movie_gallery.html
- [19] National Informal STEM Education Network (NISE Network)
http://www.nisenet.org/sites/default/files/catalog/uploads/spanish/12194/electric_squeeze_images_13nov13_sp.pdf
- [20] Página oficial del fabricante PCB
http://wwwpcb.com/techsupport/tech_accel.aspx
- [21] Hoja de especificaciones MPU-6050
<https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/>
- [22] Instrumentation-Electronics
<http://www.instrumentationtoday.com/?s=accelerometer>
- [23] Á. Solera Ramírez. *El filtro de Kalman*. Banco central de costa rica división económica departamento de investigaciones económicas. 2003

