



UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y
DISEÑO INDUSTRIAL

Grado en Ingeniería Electrónica y Automática Industrial

TRABAJO FIN DE GRADO

TÍTULO DEL TRABAJO

Autor

Cotutor (si lo hay): nombre y
apellidos

Departamento: departamento

Tutor: nombre y apellidos
Departamento: departamento

Ciudad, Mes año



UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y
DISEÑO INDUSTRIAL

Grado en Ingeniería Electrónica y Automática Industrial

TRABAJO FIN DE GRADO

TÍTULO DEL TRABAJO

Firma Autor

Firma Cotutor (si lo hay)

Firma Tutor

Título: título del trabajo

Autor: nombre del alumno

Tutor: nombre del tutor

Cotutor: nombre del cotutor

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día de de ... en, en la Escuela Técnica Superior de Ingeniería y Diseño Industrial de la Universidad Politécnica de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO

PRESIDENTE

Agradecimientos

Agradezco a

Resumen

Este proyecto se resume en.....

Palabras clave: palabraclave1, palabraclave2, palabraclave3.

Abstract

In this project...

Keywords: keyword1, keyword2, keyword3.

Índice general

Agradecimientos	v
Resumen	vii
Abstract	ix
1. Introducción	1
1.1. Motivación del proyecto	1
1.2. Objetivos	1
1.2.1. Objetivos derivados	2
1.3. Estructura del documento	2
1.4. Software utilizado	3
2. Estado del arte	5
2.1. Soportes articulados sin motorizar	5
2.1.1. Anclaje a la pared: Cotytech MW-M13P	5
2.1.2. Anclaje al techo: Titan Elite T2EQ-C8X5	6
2.1.3. Anclaje a una mesa o superficie de trabajo: Ergotron LX Sit-Stand Desk Arm	7
2.1.4. Consideraciones generales sobre los soportes no motorizados	8
2.2. Soportes articulados motorizados	8
2.2.1. Consideraciones generales sobre los soportes motorizados	9
2.3. Brazos robóticos para asistencia de pacientes	9
2.3.1. JACO 3 fingers, Kinova robotics	10
2.3.2. Brazo Multi-manipulador de la Universidad de Pamplona	10
2.3.3. Brazo robótico para personas con movilidad reducida	11
2.3.4. Consideraciones generales sobre los brazos robóticos	12
3. Diseño del brazo robótico: Punto de inicio	13
4. Mecánica y soporte físico del proyecto	19
4.1. Visión general y materiales estructurales	19
4.2. Filosofía y justificación de diseño	21
4.2.1. Realidad de las formas	22
4.3. Articulación uno. Giro en el eje Z	23
4.4. Articulaciones dos y tres. Posicionamiento en el plano sagital	25
4.5. Grados de libertad de la orientación	29

5. Diseño Electromecánico	31
5.1. Actuadores	31
5.2. Interfaz servos-microcontrolador	33
5.3. Placa controladora	34
5.4. Sensores	36
6. Diseño del software	37
6.1. Librerías (directorio lib)	38
6.1.1. debug	38
6.1.2. rha_types	39
6.1.3. joint_handler	40
6.1.4. joint_rha	41
6.1.5. servo_rha	42
6.1.6. robot_rha	43
6.1.7. chuck_handler	43
6.2. SRC. Fichero de código principal	44
6.3. Interacción entre objetos, flujo de la información y de procedimientos	44
6.4. Test y verificación del software	45
6.4.1. Test librería RHATypes	47
6.4.2. Test librería ServoRHA	48
6.4.3. Test librería JointRHA	48
6.4.4. Test librería JointHandler	48
6.4.5. Test librería ChuckHandler	48
6.4.6. Test librería RobotRHA	48
6.5. Gestión de la complejidad y mantenibilidad:	49
6.5.1. Reglas de codificación del Código	50
7. Estudio Cinemático	53
8. Control	55
8.1. Control de velocidad para los servos G15 cube	55
9. Resultados y discusión	57
9.1. Resultados	57
9.2. Discusión	57
10. Gestión del proyecto	59
10.1. Ciclo de vida	59
10.2. Planificación	59
10.2.1. Planificación inicial	59
10.2.2. Planificación final	59
10.3. Presupuesto	59
10.3.1. Personal	60
10.3.2. Material	60
10.3.3. Resumen de costes	60
11. Conclusiones	61
11.1. Conclusión	61
11.2. Desarrollos futuros	61

Apéndice	62
A. Listado de piezas diseñadas	63
B. Montaje del prototipo	77
C. Reglas de codificación del Software	85
C.1. Aspectos generales	85
C.2. Ficheros de cabecera	85
C.2.1. Inclusión Múltiple	85
C.2.2. Orden de inclusión de ficheros	86
C.3. Ámbitos	86
C.3.1. Espacios de nombres	86
C.3.2. Variables Locales	87
C.4. Clases	87
C.4.1. Constructores y métodos de Inicialización	87
C.4.2. Estructuras o Clases	88
C.4.3. Control de Acceso	88
C.5. Tipos de datos	88
C.6. Nombres	88
C.6.1. Reglas generales	88
C.6.2. Nombre de los ficheros	88
C.6.3. Nombre de los directorios	89
C.6.4. Nombres para objetos	89
C.6.5. Nombres de variables	89
C.6.6. Nombres de atributos de clases	89
C.6.7. Nombres de miembros de estructuras	90
C.6.8. Nombres de funciones	90
C.6.9. Nombres de parámetros funciones	90
C.6.10. Espacios de nombres	90
C.6.11. Nombres de enumeraciones	90
C.6.12. Nombres de macros	91
C.7. Comentarios	91
C.7.1. Comentarios de ficheros	91
C.7.2. Comentarios de Clases	91
C.7.3. Comentarios de funciones	91
C.7.4. Comentarios y aclaraciones	92
C.7.5. TODO y notas	92
C.7.6. Código en desuso	92
C.8. Formato	92
C.8.1. Espacios y tabulaciones	92
C.8.2. Declaración y definición de funciones	93
C.8.3. Condicionales	93
C.8.4. Bucles	94
C.8.5. Valor de retorno de funciones y métodos	94
C.8.6. Formato para clases	94
C.8.7. Espacios de nombre	94
C.9. Espacios en blanco	94

C.9.1. Caso general	94
C.9.2. Bucles, condicionales y estructuras de control	95
C.9.3. Operadores	95
C.10. Espacio vertical	95
D. Documentación del software	97
E. Comunicación con los Servos G15	99
E.1. Distintos tipos de instrucciones	101
E.1.1. Petición del estado del servo	102
E.1.2. Operaciones de lectura	102
E.1.3. Operaciones de escritura	102
E.1.4. Operaciones de escritura con activación desacoplada	103
E.1.5. Resetear la memoria de los servos a los valores de fábrica	103
E.1.6. Operaciones de escritura sobre múltiples servos	103
E.1.7. Aspectos interesantes a tener en cuenta	105
F. Instalación y Configuración del software necesario	109
Bibliografía	111

Índice de figuras

3.1.	Ejemplos de como se acoplan y compensan estructuras basadas en mecanismos de cuatro barras.	14
3.2.	Captura del documento original digitalizado de [13].	14
3.3.	Esquema de la evolución de diferentes versiones del diseño mecánico .	16
4.1.	Modelo de los grados de libertad de posición. Convencionalismos tomados	20
4.2.	Ejemplo de parámetros definidos en Inventor	22
4.3.	Predominancia de formas rectangulares en el entorno de trabajo .	22
4.4.	Montaje de la transmisión del movimiento del servo a la articulación encargada de girar en Z CAMBIAR POR FOTO BUENA	23
4.5.	Apoyo basado en un rodamiento Axial	24
4.6.	Apoyo basado en ruedas	24
4.7.	Comparación de como afecta el punto de apoyo de la carga superior sobre los diferentes apoyos	25
4.8.	Esquema de las articulaciones dos y tres. Estructura de unión triangular	26
4.9.	Esquema de las articulaciones dos y tres. Estructura de unión lineal .	26
4.10.	Imagen del acoplamiento <i>romboide</i>	27
4.11.	Vista genérica de los grados de libertad dos y tres	27
4.12.	Polea Movil	28
4.13.	Transmisión hasta la primera articulación	29
4.14.	Transmisión del movimiento hasta la segunda articulación	29
4.15.	Redirección a la salida de los servomotores	29
4.16.	Redirección superior. Eje de la primera articulación	29
5.1.	Esquema de la conexión de los servos formando un bus serie de servos y la placa <i>Shield</i>	32
5.2.	Conectores de los G15 Cube y uso de cada uno de los cables	32
5.3.	Posibilidades para la alimentación de los servos	34
5.4.	Esquema de la conexión entre la placa Shield y Arduino para utilizar los puertos Hardware Serie de la Arduino Mega	36
6.1.	Estructuras de datos auxiliares	39
6.2.	Esquema de ejecución de bucle de control de joint_handler	40
6.3.	Lazo de control de la velocidad del servo. Cálculos realizados por el objeto <i>ServoRHA</i>	41
6.4.	Atributos y métodos más relevantes del objeto <i>JointHandler</i>	41
6.5.	Atributos y métodos más relevantes del objeto <i>JointRHA</i>	41

6.6.	Lazo de control de la velocidad del servo. Cálculos realizados por el objeto <i>ServoRHA</i>	42
6.7.	Atributos y métodos más relevantes del objeto <i>ServoRHA</i>	43
6.8.	Atributos y métodos más relevantes del objeto <i>ChuckHandler</i>	44
6.9.	Relaciones entre las diferentes Clases HACER IMAGEN CON EL MISMO ESTILO QUE ANTERIORMENTE	44
6.10.	Salida de Platformio Test para cada caso	46
6.11.	Salida de Platformio Test con casos fallidos	47
6.12.	Salida de Platformio Test con el resumen de los casos de test	47
6.13.	Test satisfactorios RHATypes	47
6.14.	Test satisfactorios PIDRegulator (caso dentro de RHATypes)	47
6.15.	Test satisfactorios ServoRHA	48
6.16.	Test satisfactorios JointHandler	48
6.17.	Resumen Test satisfactorios	48
6.18.	Ánálisis del código del proyecto	50
6.19.	Ejemplo de la Salida de cpplint con errores	50
6.20.	Ejemplo de errores con cpplint	51
6.21.	Salida de cpplint una vez eliminados los errores	51
E.1.	Paquete de información genérico para comunicar con los Servos G15 Cube	100
E.2.	Paquete de información genérico de retorno de los Servos G15 Cube .	101

Índice de tablas

5.1. Características relevantes de los Servos G15 de Cytron.	33
5.2. Comparativa entre placas Arduino Uno y Arduino Mega2560	35
5.3. Comparativa entre placas Arduino Uno y Arduino Mega2560	36
10.1. Costes del proyecto	60
A.1. Listado de piezas diseñadas de fabricación propia	64
A.2. Parámetros de las piezas para la estimación de peso	75
E.1. Resumen de las instrucciones aceptadas por los Cytron G15 Cube servo100	
E.2. Codificación del error de los servos G15 Cube en cada bit del byte de	
error	101
E.3. Codificación del error de comunicación en cada bit del segundo byte	
de error	101
E.4. Ejemplo paquete con la instrucción <i>iSYNC_WRITE</i>	105
E.5. Direcciones de memoria de los servos con los diferentes parámetros	
a ajustar. Incluye valores mínimos y máximos así como valores por	
defecto para cada parámetro	107



Capítulo 1

Introducción

En este capítulo se perfila la estructura, organización y contenidos principales del documento así como la motivación y objetivos del proyecto.

1.1. Motivación del proyecto

1.2. Objetivos

El objetivo que este Trabajo de Fin de Grado persigue es el del diseño, construcción y control de brazo robótico previo estudio de las opciones comerciales disponibles y su posible adaptación. Este proyecto está enmarcado bajo el proyecto Robohealth, proyecto financiado por el Ministerio de Economía y Competitividad con el objetivo del diseño de sistemas robóticos y domóticos para entornos hospitalarios que mejoren el sistema de salud actual. (**- COMPLETAR -**<http://robohealth.es/>)

El prototipo debe estar diseñado para una completa adaptación a un entorno hospitalario en el que deberá estar en contacto constante con usuarios a los que deberá respetar.

El objetivo del brazo robótico es el de ubicar ante un paciente una *tablet*. Ésta llevará montado un dispositivo de seguimiento de vista de forma que, a través del movimiento de las pupilas, el paciente podrá interactuar con el resto de dispositivos de la habitación así como el personal médico. Es necesario motorizar el dispositivo para mantener el dispositivo de seguimiento siempre a una distancia y ángulo, respecto a la cara del paciente, mínima para facilitar el funcionamiento del mismo. Está pensado principalmente para pacientes con movilidad reducida o sin movilidad (temporal o permanente), aunque también podría agilizar la interfaz humano-habitación para el resto de pacientes.

Así pues, algunos aspectos claves del prototipo deben ser:

- El Objetivo del prototipo será el de permitir una interacción más cómoda y automatizada entre los pacientes cuya capacidad de interacción se ha visto coartada por la causa que sea.
- Se debe tener en cuenta es que el prototipo estará en constante contacto con

gran variedad de usuarios: pacientes, médicos, familiares, etc. El diseño debe proteger en todo momento la seguridad de dichos usuarios

1.2.1. Objetivos derivados

La realización de dicho prototipo implica el cumplimiento de otros objetivos secundarios o derivados del principal. Se pueden listar algunos como:

- Prueba de diferentes tipos de estructuras y materiales como base física del brazo robótico.
- Adquisición de conocimientos sobre modelado 3D digital así como diferentes métodos de fabricación como son la impresión 3D, el corte láser, fresado CNC así como el uso de otras herramientas de mecanizado más tradicionales.
- Diseño e implementación de un sistema de control en cascada, que permita un control en posición y en velocidad del brazo robótico.

1.3. Estructura del documento

El documento está organizado de tal forma que irá introduciendo al lector progresivamente en los diferentes aspectos del diseño y montaje del prototipo mencionado, desde aspectos más generales hasta los más técnicos.

Los capítulos están a su vez organizados en el orden que se seguiría de cara a montar el prototipo empezando por una base física, añadiendo a posteriori los componentes electromecánicos para finalizar con los aspectos de control. Concretamente los capítulos contienen la siguiente información:

- Como continuación de los requerimientos generales se encuentra descritos en la introducción, en el capítulo 2, un estudio de diferentes modelos y diseños comerciales que podrían adaptarse para cumplir los objetivos presentados.
- El estudio del estado del arte ayuda a definir las ideas más importantes que regirán el diseño del prototipo. Éstas consideraciones se presentan en el capítulo 3.
- El capítulo 4 entra de lleno en los aspectos mecánicos del brazo robótico, pasando por una valoración de distintas posibilidades para llegar al diseño definitivo.
- Continuando con la descripción de soporte físico, el capítulo 7 supone un cambio de perspectiva que guiará al lector desde la parte mecánica y física expuesta anteriormente a la descripción matemática y cinemático del modelo.
- Una vez descrito el soporte físico del brazo robótico se detalla el hardware escogido para su puesta en marcha. En el capítulo 5 presentan los componentes electromecánicos que se han integrado en prototipo; sus características principales así como su ubicación en el montaje.
- El análisis de la estructura software diseñado queda cubierto en el capítulo 6. Este apartado anticipa además algunas ideas sobre el sistema de control diseñado.

- Continuando con las pinceladas aportadas en el apartado anterior, el capítulo 8 expone de forma detallada los distintos aspectos de diseño y desarrollo del control del brazo.
- Una vez alcanzado este punto, habiendo cubierto los aspectos del diseño del brazo, el capítulo 9 recoge los resultados de funcionamiento de prototipo para ser analizados.
- No se dejan de lado aspectos de gestión, costes y viabilidad del prototipo que se detallan en el capítulo 10.
- Finalmente, el capítulo 11 expone las conclusiones finales del trabajo así como posibles desarrollos futuros.

Como complemento a la información que exponen los apartados de esta memoria se añaden al final diferentes anexos:

- Se adjunta un listado de todas las piezas diseñadas con información relevante sobre su uso y fabricación en el anexo A.
- Vistas las piezas que conforman el prototipo es importante localizarlas para entender su uso y diseño. El anexo B detalla unas pautas y orden para el ensamblado del prototipo. Además permite localizar en su contexto las piezas listadas en el anexo ??.
- Volviendo sobre los aspectos del software, en el Anexo C se concretan las reglas de codificación, mencionadas en el capítulo 6, más relevantes que se han aplicado en el desarrollo del código.
- De igual forma se amplia la información sobre el software desarrollado en el anexo D. En él se encuentra la documentación del software generada a través de la herramienta *doxygen*.
- El desarrollo del proyecto conlleva la utilización de diferentes herramientas software cuya instalación y puesta punto se detalla en el anexo F.

1.4. Software utilizado

Aunque en el anexo F se detalla la instalación del software en más detalle no está de más conocer las herramientas a utilizar de antemano, ya que éstas marcan unas pautas en la ideología de diseño y una estructura a la hora de ordenar y desarrollar el proyecto.

- Autodesk Inventor 2016: Es un software de modelado paramétrico 3D de la compañía Autodesk Inc.
- Atom: Se trata de un editor de texto *open source*. Permite la instalación de diferentes extensiones para ampliar sus utilidades, entre otras será necesario instalar *PlatformIO*, que convierte el editor en un *Entorno de desarrollo integrado (IDE)* completo para el desarrollo de software para diferentes placas como Arduino, que será la base de este proyecto.

- Lizard: Software que permite el análisis de la complejidad de código. Se compone de una serie de scripts en python que, al ser ejecutados devuelven un fichero con métricas de complejidad referentes a los ficheros de código sobre los que se invoca: complejidad ciclomática, número de funciones en cada fichero, líneas de código en cada función y fichero, entre otras..
- Cloc: Es una herramienta sencilla que cuenta, de forma más precisa, el número de líneas de código, comentarios y líneas en blanco de los ficheros de código.
- cpplint: Análisis del cumplimiento de las reglas de codificación en el software. Es una herramienta desarrollada en python por Google LLC para asegurar que los proyectos cumplen con sus reglas de codificación, que se han seguido de forma parcial en este proyecto. Se pueden ver los aspectos más relevantes de las reglas de codificación en el Anexo C.
- doxygen: Permite la generación de documentación para código de diferentes lenguajes, c++ en este caso, de forma automática. La herramienta obtiene comentarios del código, escritos con una sintaxis determinada, para documentar los diferentes métodos, objetos y estructura del software.

Además es interesante repasar los términos que se incluyen en el Glosario y que aparecerán referenciados a lo largo del documento. – **COMPLETAR** –



Capítulo 2

Estado del arte

Una vez se han repasado los aspectos generales que se buscan para este proyecto conviene hacer un estudio de la situación actual de modelos comerciales o de investigación con los que se puedan encontrar sinergías.

De esta manera este capítulo hace un repaso de diferentes soluciones destinadas al soporte y posicionamiento de monitores, ordenadores o tablets. Además se hace referencia también a modelos de brazos robóticos específicamente destinados a la asistencia en entornos de usuarios. Dentro de todas las soluciones comerciales se centrará el estudio en las que están específicamente pensadas para su instalación en entornos médicos siempre que sea posible.

Según el apoyo así como los tipos de grados de libertad hay diferentes configuraciones posibles, en este capítulo se verán algunos modelos concretos de cada caso evaluando sus características, ventajas e inconvenientes de los mismos de forma que se pueda generalizar a modelos equivalentes. De igual manera se podrán encontrar modelos motorizados y modelos sin motorizar, clasificación por la que serán agrupados a continuación.

2.1. Soportes articulados sin motorizar

Dentro del grupo de soportes articulados sin motorizar se pueden clasificar según el tipo de anclaje que tienen, ya se enganchen al techo, pared, suelo, etc.

Dentro de cada tipo de anclaje los soportes comerciales disponibles son bastante parecidos por lo que se presentará un modelo concreto que encaje dentro de las dimensiones y capacidad de carga requeridas para la aplicación que se pretende explotar para sacar conclusiones generales sobre cada tipo de soporte.

2.1.1. Anclaje a la pared: Cotytech MW-M13P

Dentro de esta gama (Cotytech MW-M*) se pueden encontrar modelos para soportar diferentes cargas. Concretamente se ha elegido el modelo con menores prestaciones y que soporta menos carga por ser suficiente para la aplicación que se pretende

dar. Otros modelos pueden incluir soporte para teclado, caja y cobertura para cables y enganche a pared, entre otros, suponiendo un incremento sobre el precio de este modelo. Obtenida de [1] tenemos información relevante que se resume a continuación:



- Tipo de anclaje: Anclaje a la pared.
- Tipo de articulación: Articulaciones rotacionales.
- Capacidad de carga: entre $1kg$ y $6kg$.
- Extensión máxima: $185,7cm$.
- Número de grados de libertad: 5.
- Ángulos articulaciones: 370° (brazo posición), 270° (muñeca posición), 180° (pared) ¹.
- Ángulos de orientación: Tilt: $20^\circ/-35^\circ$ (muñeca orientación) and $20^\circ/-60^\circ$ (brazo orientación).
- Peso del soporte: $4,76kg$.
- Precio estándar: 686.98€ . ².

¹En la imagen se pueden ver tres puntos articulados diferenciados, el punto que se fija a la pared con un grado de libertad, el punto central del brazo, que tiene dos grados de libertad (se separarán entre orientación y posición, aunque su efecto no está desacoplado) y la muñeca, que incluye la articulación que se aprecia justo encima de la pantalla como la rotacional sobre la que queda enganchada la misma (con una clasificación análoga al caso intermedio).

²Precio a pasado a Euros según el cambio oficial en el día consultado.

Paralelamente a este modelo la marca Cotytech tiene una versión que, manteniendo el mismo esquema de mecánico, permite un anclaje al techo: el modelo CM-M13 visto en [2] con un coste de 853.06€ .

2.1.2. Anclaje al techo: Titan Elite T2EQ-C8X5

Concretamente se ha tomado el modelo con la montura doblada hacia arriba para un anclaje en el techo. La siguiente información constituye un resumen con los puntos más importantes vistos en [3]:



- Tipo de anclaje: Anclaje al techo.
- Tipo de articulación: Articulaciones rotacionales.
- Capacidad de carga: hasta $12,7\text{kg}$ en diferentes configuraciones.
- Extensión máxima: 106cm (la longitud vertical del anclaje al techo variará según se elija al comprar).
- Número de grados de libertad: 5.
- Ángulos articulaciones: 360° para las tres primeras articulaciones que rotan sobre el eje horizontal.
- Ángulos de orientación: Tilt: 50° .
- Peso del soporte: 9kg .
- Precio estándar: 628.30€ .

Este mismo modelo cuenta con diferentes enganches y longitudes de los tubos que permiten anclarlo al suelo, al techo o a la pared indistintamente.

2.1.3. Anclaje a una mesa o superficie de trabajo: Ergotron LX Sit-Stand Desk Arm

De entre la gama incluida en los modelos de Ergotron LX se ha elegido aquel que tenía unas dimensiones más ajustadas a las necesidades reales. En general el resto de la gama y otros soportes similares tienen unas dimensiones más reducidas. Resumidas de [4] y [5] se encuentran a continuación las principales características del modelo:



- Tipo de anclaje: Anclaje a una mesa, camilla o similar.
- Tipo de articulación: Primera articulación prismática, resto rotacionales.
- Capacidad de carga: hasta 11,3kg.
- Extensión máxima: se puede variar hasta 36cm en altura (articulación prismática, esta es susceptible de ser modificada para ajustarla a otras alturas) y alcanza una extensión de 84cm
- Número de grados de libertad: 6 (una prismática y cinco rotacionales).
- Ángulos articulaciones: 180° la primera articulación, fija al anclaje; 360° a mitad del brazo y 180° en la muñeca.
- Ángulos de orientación: Tilt: (giro sobre el eje medio horizontal de la pantalla) 75°; Pan (eje perpendicular a la pantalla): 360°.
- Peso del soporte: 8,9kg.
- Precio estándar: 247.00€– 299.00€ dependiendo de la tienda y configuración.

Este mismo modelo cuenta con diferentes enganches y longitudes de los tubos que permiten anclarlo al suelo, al techo o a la pared indistintamente.

2.1.4. Consideraciones generales sobre los soportes no motorizados

Los modelos vistos hasta ahora presentan un rango de movimientos muy amplio, están certificados y preparados para su uso en entornos hospitalarios además de estar destinados precisamente al soporte de monitores. En todos los casos la capacidad de carga excede con creces la que se estima necesaria en este proyecto por lo que todas las opciones podrían ser válidas.

Aunque cuentan con puntos bastante favorables se trata de productos cerrados sobre los cuales sería complicado integrar actuadores y sensores de manera adecuada, segura y en última instancia, elegante. Además el rango de precios en el que se encuentran es bastante elevado.

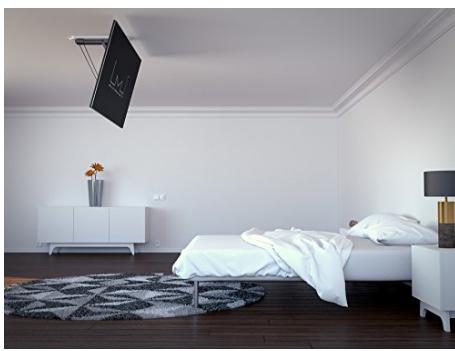
2.2. Soportes articulados motorizados

Se centrará este apartado en los modelos que se pueden ver en [6], como por ejemplo el MaiorFlip 900.



- Tipo de anclaje: Anclaje al techo.
- Tipo de articulación: Primera articulación rotacional, segunda prismática y tercera rotacional.
- Capacidad de carga: máximo de *28kg*.
- Extensión máxima: Decenso de hasta *68cm*
- Número de grados de libertad: 3 (una prismática y dos rotacionales).
- Ángulos articulaciones: Primera rotación giro de hasta 90° ; articulación prismática con un alcance de *68cm* con una capacidad de giro de la articulación final de 360° .
- Peso del soporte: *35kg*.
- Precio estándar: Bajo demanda.

Otros modelos que se pueden encontrar presentan unas características similares:



2.2.1. Consideraciones generales sobre los soportes motorizados

Este tipo de soportes están principalmente pensados para motorizar televisores de gran tamaño con unos rangos de movimiento bastante limitados. No son aptos para la aplicación que se pretende dar puesto que no permiten su posicionamiento a una distancia adecuada del usuario.

2.3. Brazos robóticos para asistencia de pacientes

En general para el propósito que se plantea en este proyecto se podría adaptar una solución robótica comercial implementando una interfaz entre la tablet y el controlador del brazo. En esta sección se presentan algunos modelos de brazos robóticos especialmente pensados como robots asistenciales, preparados para una interacción directa con pacientes en entornos hospitalarios o caseros **NOT THE WORD....**

2.3.1. JACO 3 fingers, Kinova robotics

Pertenece a la línea de productos de Kinova robotics especialmente diseñados como robots asistenciales. Están pensados para una interacción directa con el paciente o usuario de forma que pueda convertirse en una extensión del mismo proporcionándole una mayor independencia. Entre las características descritas en [7] y [8] podemos recoger las siguientes más relevantes:



- Tipo de anclaje: Adaptativo a una mesa, silla de ruedas, etc
- Tipo de articulación: 6 grados de libertad rotacionales
- Capacidad de carga: entre $1,8kg$ y $1,6kg$ dependiendo de la versión elegida (con tres dedos tiene una capacidad menor).
- Extensión máxima: alcanza $90cm$.
- Número de grados de libertad: 6.
- Ángulos de posición: Rotación continua en las articulaciones.
- Ángulos de orientación: 55° en la muñeca.
- Peso del brazo: $5,7kg$.
- Precio estándar: desde 34900€ .

Este modelo concreto, Jaco, viene en dos formatos pudiendo tener dos o tres dedos en el manipulador de su extremo.

De esta marca se puede adquirir también el modelo MICO con algo menos de alcance ($70cm$) como se ve en [9]. Los grados de libertad ofertados varían entre 4 y 7, se ha optado por tomar una solución lo más parecida a la requerida.

2.3.2. Brazo Multi-manipulador de la Universidad de Pamplona

Dejando de lado el ámbito puramente comercial se encuentran proyectos que es interesante repasar. En [10] se describe un modelo diseñado específicamente como robot asistencial con capacidad de cambiar, de forma autónoma, entre diferentes manipuladores disponibles como pueden ser una cuchara, un tenedor, un cuenco, entre otros.



- Tipo de anclaje: Lleva su propia plataforma sobre la que se monta
- Tipo de articulación: 4 grados de libertad rotacionales
- Capacidad de carga: manipuladores que adjunta.
- Extensión máxima: alcanza Superficie de trabajo de 40cm x 40cm.
- Número de grados de libertad: 4.
- Actuadores: Dynamixel AX-12
- Precio estándar: proyecto no comercial.

Es interesante destacar que aunque no se aportan demasiados datos sobre el modelo (peso del modelo, capacidad de giro de sus articulaciones, etc) si presenta un estudio completo de su aceptación así como facilidad de uso. Un sistema intuitivo es más fácilmente aceptado por los usuarios, a los cuales les será más fácil empezar a hacer uso de las facilidades que ofrece un robot de este tipo.

Los actuadores que utiliza son de la gama de Dynamixel, una serie de servo motores muy versátiles aunque sin demasiada carga útil una vez montados sobre las articulaciones.

2.3.3. Brazo robótico para personas con movilidad reducida

Continuando con modelos de un ámbito más académico se encuentra el modelo descrito en [11]. Se trata de un modelo plegable que puede ser almacenado o transportado dentro de una maleta de forma sencilla. Viene con un manipulador en forma de manopla que permite agarres de una amplia gama de objetos. También pensado para la asistencia de personas con movilidad reducida ha sido principalmente testeado para administrar alimentos y bebidas.



- Tipo de anclaje: Sin determinar.
- Tipo de articulación: 7 grados de libertad rotacionales
- Extensión máxima: alcanza 71cm.
- Número de grados de libertad: 7.
- Peso del brazo: 5kg con dos baterías.
- Precio estándar: proyecto no comercial.

2.3.4. Consideraciones generales sobre los brazos robóticos

Aunque entre los modelos descritos sería fácil encontrar uno válido para la aplicación que se pretende dar, la mayoría de modelos comerciales (en los casos académicos se desconoce el precio que rondaría en caso de convertirse en producto comercial) tienen unos precios elevados que dificultarían su aplicación a gran escala.

Estos modelos, aunque estando diseñados específicamente para entornos de interacción directa con usuarios no justifican las medidas de seguridad que se tienen en cuenta para evitar daños a los mismos. En última instancia en los diseños y prototipos encontrado no se pueden apreciar diferencias en el diseño de los mismos con modelos industriales (más allá de tamaño o peso).



Capítulo 3

Diseño del brazo robótico: Punto de inicio

En este capítulo se continuará perfilando los aspectos de diseño generales del prototipo. Una vez repasado el estado del arte en el capítulo 2 ya se tiene una idea de las soluciones disponibles y sus deficiencias y fortalezas. Este capítulo vuelve sobre los objetivos presentados en el capítulo 1 para ofrecer un marco más completo sobre el que se apoye todo el desarrollo del proyecto así como posteriores capítulos.

Es importante recordar que el prototipo debe estar en constante interacción con usuarios no especializados; es por ello que se han pre establecido unas características básicas que suponen un afianzamiento de la seguridad hacia los usuarios inherentes al diseño.

- Se impone el uso de motores con bajo par de trabajo. De esta manera se vuelve físicamente imposible que los mismos puedan suponer un riesgo para los usuarios ya que la fuerza que son capaces de realizar no es suficiente para suponer un peligro para los mismos.
- El uso de motores de bajo par implica una compensación del peso del propio brazo de forma que los motores deban cargar con el menor peso posible. Eso se consigue a través de mecanismos de cuatro barras acoplados, de igual forma que se hace en diferentes lupas y lámparas comerciales como se puede ver en la figura 3.1. Como demuestra [12] compensar este tipo de estructuras mediante el uso de muelles es bastante sencillo sea cual sea el número de articulaciones que acopladas.
- Además la elección de dichos motores implica una amplificación mecánica a lo largo de la transmisión del movimiento en el diseño del prototipo. Los motores estarán ubicados en la base y el par de los mismos será transmitido a las articulaciones. Se ha optado por una transmisión mediante hilos que, como dice [13] será siempre más silenciosa y suave, permitirá hacer la transmisión solo en un sentido, es decir, para subir el extremo del brazo (figura 3.2). El sentido de bajada se hará soltando dicho hilo dejando que la gravedad y el peso del brazo lo hagan descender. De esta manera en caso de que se encuentre un usuario en la trayectoria del brazo éste no podrá ejercer una fuerza activa contra el mismo. El paciente sentirá como mucho como el peso del brazo, compensado en su mayoría mediante los muelles, se apoya sobre si mismo.

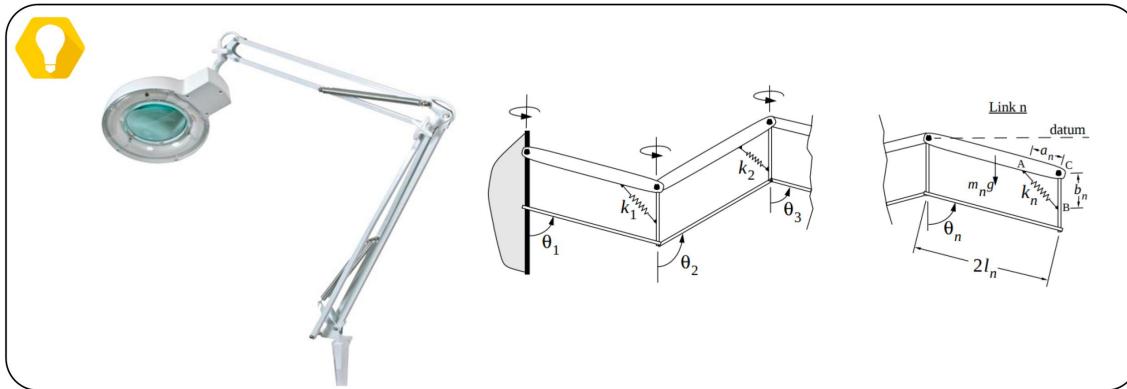


Figura 3.1: Ejemplos de como se acoplan y compensan estructuras basadas en mecanismos de cuatro barras.

Fuente: A la izquierda imagen de internet. A la derecha es una captura de [12]



Figura 3.2: Captura del documento original digitalizado de [13].

Fuente: Captura de [13]

- El uso de hilos para la transmisión permitirá además una reversibilidad en el control efectuado. Otra posible alternativa es la transmisión del movimiento por rozamiento, de forma que el usuario no sufrirá daño alguno en cuanto la fuerza que haga el mismo sobre el robot supere el rozamiento de dicha transmisión.

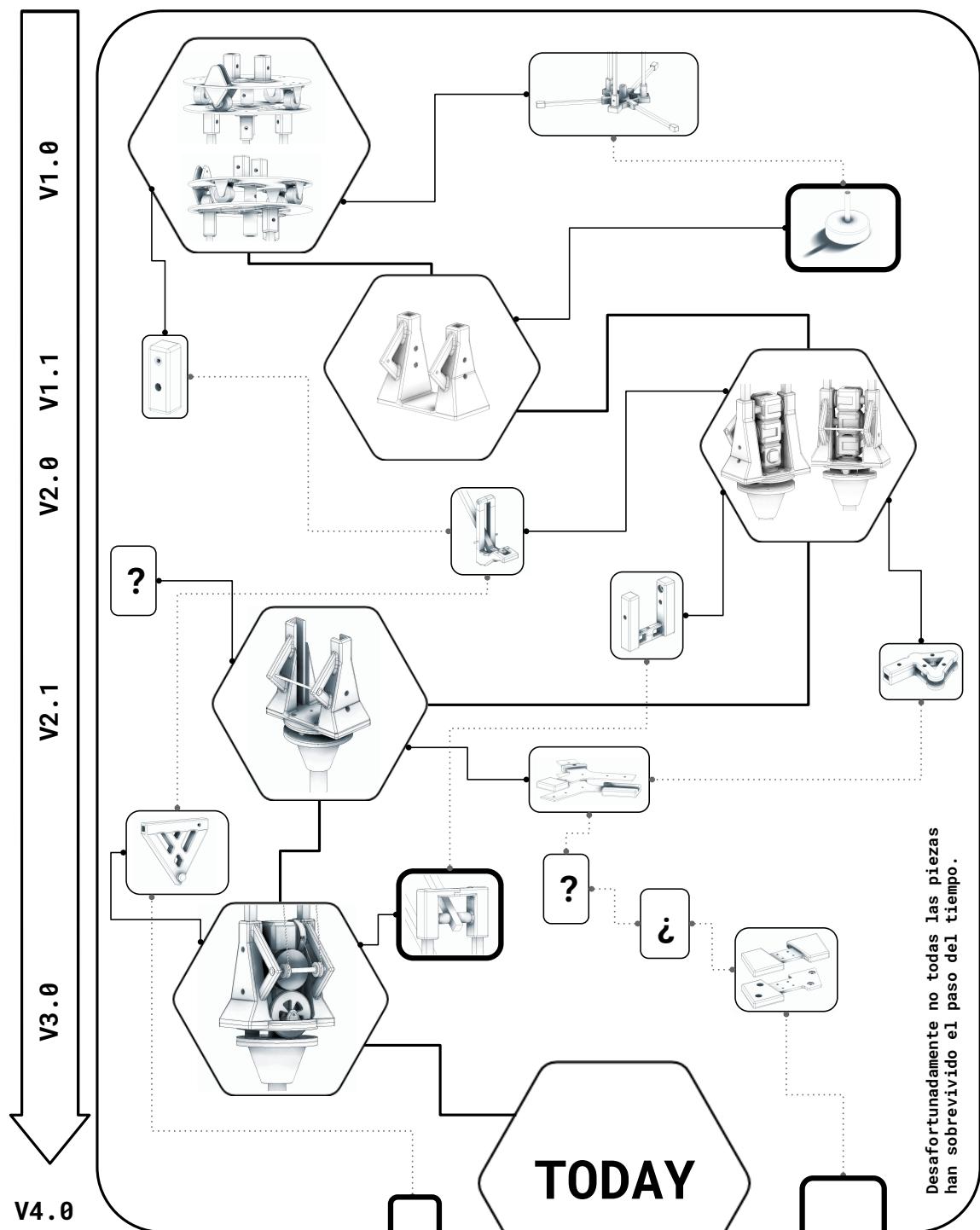
Como pruebas de concepto, puesto que el diseño de una versión completa lleva una cantidad de tiempo y de trabajo nada despreciables se ha hecho uso de diferentes maquetas diseñadas específicamente para probar algunas estructuras mecánicas que formarán parte de la estructura del robot. Algunos métodos sobre los que se puede apoyar la construcción de maquetas, siempre a escala, son la impresión 3D (maquetas en plástico) o el corte láser (maquetas en madera DM). De esta forma se realizan de forma rápida y barata pruebas de concepto para descartar o incluir diferentes ideas en el diseño final. Algunas maquetas realizadas para pruebas de concepto de la estructura mecánica se pueden ver en la figura ??

IMAGEN MAQUETAS

Aun haciendo uso de maquetas, y como en cualquier proyecto de prototipado, ha sido necesario seguir una metodología iterativa. Para los diferentes aspectos del

proyecto se van probando y corrigiendo diferentes aspectos de la mecánica e integración de los componentes de forma que se va refinando y completando un modelo definitivo. Como ejemplo se puede ver en la figura 3.3 como ha ido evolucionando el desarrollo mecánico que se verá a continuación.

Rob-volution



Enrique Heredia Aguado - Proyecto RoboHealtArm

Figura 3.3: Esquema de la evolución de diferentes versiones del diseño mecánico

Fuente: Autor

Como es de esperar, un proceso iterativo de este tipo acaba por generar un *cementerio* de piezas desechadas por modelos más completos y mejorados.

INCLUIR FOTO DEL CEMENTERIO



Capítulo 4

Mecánica y soporte físico del proyecto

Una vez vistas algunas ideas previas generales que deberá tener el prototipo diseñado este capítulo entra de lleno en la descripción de la solución mecánica obtenida así como una comparación con ideas previas.

4.1. Visión general y materiales estructurales

Antes de empezar a repasar los detalles concretos es necesario establecer unos convencionalismos respecto al uso de nomenclatura.

Como se ha descrito en el capítulo 3 el movimiento de las articulaciones encargadas de posicionar el extremo del robot se transmitirá de forma mecánica desde los motores hasta la propia articulación. Aunque no viene impuesto por ningún requisito funcional, se ha optado por fijar los tres primeros grados de libertad de tipo rotacional. En la figura 4.1 se puede ver un diagrama esquematisado de un brazo robótico con tres grados de libertad rotacionales. A partir de ahora se hará referencia a los tres primeros grados como los *GDL* o articulaciones de posición.

En la imagen se puede ver como serán nombradas las primeras tres articulaciones también se debe aclarar la nomenclatura a utilizar sobre los elementos que unen dichas articulaciones. Serán llamados indistintamente eslabones o barras estando numerando desde la barra cero, que conecta el suelo con el primer grado de libertad, en adelante.

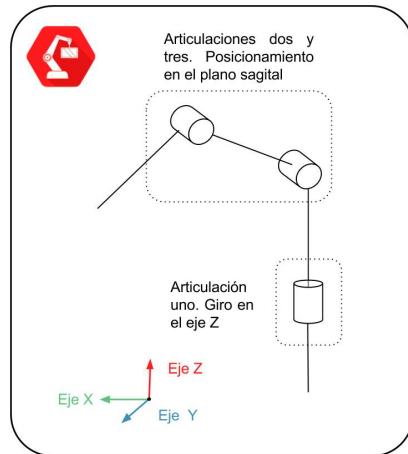


Figura 4.1: Modelo de los grados de libertad de posición. Convencionalismos tomados

Fuente: Autor

Como se ha comentado en secciones anteriores el brazo debe ser capaz de orientar la tablet de forma que ésta sea capaz de *ver* y seguir los ojos del paciente. Aunque el brazo podría anclarse a diferentes estructuras de ahora en adelante se centra la justificación y desarrollo en el caso de uso sobre una camilla. Este caso es el que requiere mayor longitud de trabajo de forma que las dimensiones del brazo deberán permitirle abarcar de un lado a otro de la camilla.

Este espacio de trabajo definido, que deberá abarcar aproximadamente 90cm (la mayoría de camas y camillas tienen esta medida), implica unas dimensiones de los eslabones del brazo considerables. Como se ha visto en capítulos anteriores el peso del prototipo es un factor importante tanto para la seguridad del usuario como para que pueda ser desplazado por unos motores de bajo par.

La transmisión mecánica del movimiento debe optimizarse para ser capaz de mover el brazo; este trabajo se verá simplificado a su vez aligerando, cuando sea posible, la estructura del mismo. Como material estructural base se han elegido perfiles de aluminio de sección cuadrada (se describen en detalle en el capítulo – **COMPLETAR** – junto con el resto de materiales). Estos perfiles presentan una alta resistencia y capacidad de carga a la vez que un peso reducido.

El material estructural definido se complementa con otra serie de piezas diseñadas específicamente para este proyecto (se puede ver el anexo A para ver estas piezas). En este caso, por su versatilidad se utilizarán piezas modeladas digitalmente e Impresas en 3D posteriormente. De igual manera parte del diseño se apoya en piezas de metacrilato cortado con láser.

Aunque el peso del brazo completo se estima será reducido, la transmisión mediante cuerdas debe ser completamente fiable. Debido a las dimensiones con las que se trabaja se ve inviable el uso de cadenas o cuerdas de gran grosor; es por eso que se ha optado por utilizar hilo de *kevlar* de un milímetro de grosor.

Aunque supone anticipar parte del desarrollo que se verá a continuación, a modo de lista se pueden resumir los materiales necesarios a nivel constructivo del proyecto

en:

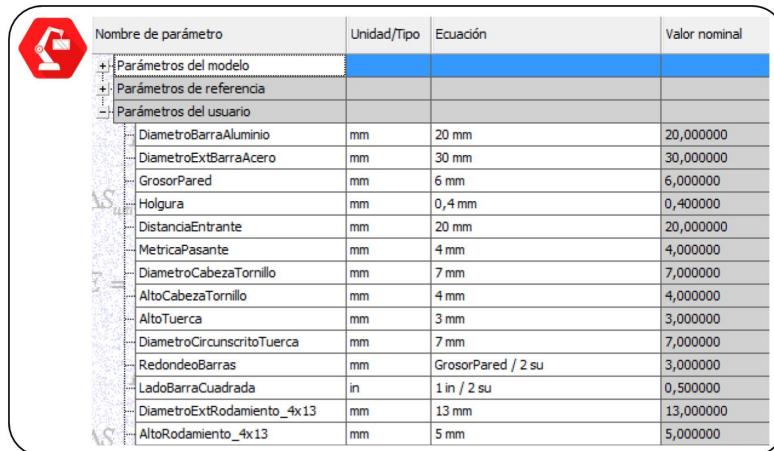
- Barras de aluminio de sección cuadrada de 1/2" x 1m (lado de la sección x longitud).
- Rodamientos de bolas de 4mm x 13mm (diámetro interior X diámetro exterior)
- Rodamientos de bolas de 3mm x 10mm (diámetro interior X diámetro exterior)
- Rodamiento axial de 37.5mm x 52mm (diámetro interior X diámetro exterior)
- Barras de acero macizo cilíndricas de 4mm de diámetro
- Barras de acero macizo cilíndricas de 3mm de diámetro
- Piezas impresas en impresora 3D. (Lista detallada en el Anexo A)
- Piezas de metacrilato cortadas con láser. (Lista detallada en el Anexo A)
- Tornillería: tornillos y tuercas de diferentes métricas.
- Hilo de kevlar.
- Poleas de acetal de 4.9mm x 21.8mm (diámetro interior X diámetro exterior).
- Soporte de sombrilla. La base sobre la que se apoye el brazo debe ser capaz de soportarlo en una posición de trabajo en todo momento. Para no fijar ni imponer ningún tipo de anclaje (a superficie plana, anclaje a una camilla, etc) se ha optado por esta solución para el desarrollo del prototipo.

En el capítulo 10, concretamente en la tabla 10.1, están reflejados los detalles técnicos de estos materiales así como las referencias de compra, cantidades necesarias y precios.

4.2. Filosofía y justificación de diseño

Tal y como se ha adelantado en el capítulo introductorio el software empleado para modelar el brazo robótico es Autodesk Inventor. Este programa es especialmente potente para modelado paramétrico; esto significa que permite definir una serie de parámetros que serán utilizados para definir las medidas de los bocetos. Un diseño bien modelado paramétricamente permite mucha flexibilidad a la hora de modificar ciertas medidas como pueden ser agujeros para tornillería, ejes u holguras.

El modelado de todas las piezas ha seguido esta filosofía de parametrización y dependencia entre las piezas de forma que el diseño sea lo más flexible posible en cuanto a modificación de medidas. En la figura 4.2 se puede ver a modo de ejemplo parte de la lista de parámetros definida.



The screenshot shows a table of parameters defined in the Inventor software. The columns are labeled: Nombre de parámetro (Parameter Name), Unidad/Tipo (Unit/Type), Ecuación (Equation), and Valor nominal (Nominal Value). The table includes rows for model parameters, reference parameters, and user-defined parameters, such as DiametroBarraAluminio (20 mm), DiametroExtBarraAcero (30 mm), GrosorPared (6 mm), Holgura (0,4 mm), DistanciaEntrante (20 mm), MetricaPasante (4 mm), DiametroCabezaTornillo (7 mm), AltoCabezaTornillo (4 mm), AltoTuerca (3 mm), DiametroCircunscritoTuerca (7 mm), RedondeoBarras (GrosorPared / 2 su), LadoBarraCuadrada (1 in / 2 su), DiametroExtRodamiento_4x13 (13 mm), and AltoRodamiento_4x13 (5 mm).

Nombre de parámetro	Unidad/Tipo	Ecuación	Valor nominal
+ Parámetros del modelo			
+ Parámetros de referencia			
+ Parámetros del usuario			
DiametroBarraAluminio	mm	20 mm	20,000000
DiametroExtBarraAcero	mm	30 mm	30,000000
GrosorPared	mm	6 mm	6,000000
Holgura	mm	0,4 mm	0,400000
DistanciaEntrante	mm	20 mm	20,000000
MetricaPasante	mm	4 mm	4,000000
DiametroCabezaTornillo	mm	7 mm	7,000000
AltoCabezaTornillo	mm	4 mm	4,000000
AltoTuerca	mm	3 mm	3,000000
DiametroCircunscritoTuerca	mm	7 mm	7,000000
RedondeoBarras	mm	GrosorPared / 2 su	3,000000
LadoBarraCuadrada	in	1 in / 2 su	0,500000
DiametroExtRodamiento_4x13	mm	13 mm	13,000000
AltoRodamiento_4x13	mm	5 mm	5,000000

Figura 4.2: Ejemplo de parámetros definidos en Inventor

Fuente: Autor

4.2.1. Realidad de las formas

Cualquier diseño de producto, incluso desde un punto de vista más técnico incluye a su vez consideraciones sobre el diseño estético del mismo. Para este proyecto se ha intentado respetar la verdad de las formas predominantes tanto a nivel estructural como en el entorno de trabajo.

Como ya se ha visto, la base estructural del prototipo está conformada por barras de aluminio de sección cuadrada. Unido a la predominancia de formas rectangulares y/o cuadradas en una habitación de hospital (figura 4.3) se ha optado por un diseño basado en prismas rectangulares, con un aspecto más *retro*.

A nivel bibliográfico y del estado del arte estudiado se puede observar una tendencia a formas más suavizadas en muchos de los casos. En este proyecto se busca un contraste entre las formas que caracterizan al ser humano, más suavizadas y con geometrías complejas, en contraposición con las formas que definen el prototipo, formas geométricas y angulares.



Figura 4.3: Predominancia de formas rectangulares en el entorno de trabajo

Fuente: Imagen de consalud.es, editada posteriormente por el Autor

4.3. Articulación uno. Giro en el eje Z

Volviendo sobre el diseño mecánico, la primera articulación que se encuentra subiendo desde la base es la encargada de girar todo el robot en el eje Z.

Esta articulación está actuada por un Servo motor (descrito en detalle en el Capítulo 5. Como se ha anticipado el movimiento de este se transmite a través de un juego de ruedas que, solidarias a la parte superior (parte móvil), y por rozamiento, transmiten el movimiento de giro con respecto a la parte inferior (parte fija o base).

Una de las ruedas gira solidaria al servo motor, haciendo contacto firme con la rueda a continuación (rueda transmisora). Ésta será la encargada de transmitir el par sobre la superficie sobre la que apoya. El peso del brazo sobre dicho apoyo es suficiente como para asegurar que en condiciones normales de funcionamiento la articulación gire con normalidad; en caso de chocar contra un usuario la fuerza será mayor y la rueda transmisora deslizará sobre la superficie inferior evitando daños al usuario.

En la figura 4.4 se puede ver en detalle el montaje de dicha estructura. Las piezas que aparecen en la imagen se pueden encontrar en mayor detalle descritas en el Anexo A.



Figura 4.4: Montaje de la transmisión del movimiento del servo a la articulación encargada de girar en Z **CAMBIAR POR FOTO BUENA**

Fuente: Autor

Una vez vista la transmisión del movimiento queda otro punto importante, el apoyo del peso del brazo sobre la articulación. Referente a este punto se han explorado dos líneas del desarrollo posibles:

- Apoyo sobre ruedas: Además de la rueda transmisora se fijan a la parte móvil de la articulación una serie de ruedas sobre las que apoyará el brazo. Inicialmente puede parecer que el apoyo será más estable cuantos más puntos tenga; desde el punto de vista práctico incluir apoyos redundantes genera más problemas a la larga. Tres puntos son suficientes para definir un plano y es el número de ruedas óptimo a utilizar: dos ruedas de apoyo más la rueda de transmisión. Se puede ver en detalle como queda el montaje en la figura 4.6. Como se puede ver el diseño incluye unas piezas en metacrilato, en concreto la pieza solidaria a la parte fija del robot cuenta con una la superficie ranurada (realizado a modo de grabado con la máquina de corte láser) para aumentar el rozamiento con la rueda de transmisión. Las ruedas de apoyo están ubicadas a la misma distancia del eje de giro que la rueda transmisora.
- Apoyo sobre rodamiento: El apoyo se realiza sobre un rodamiento axial; quedando junto a ambas piezas, superior e inferior, incrustados entre si. Sobre este punto se apoya el peso del brazo y el encaje entre las piezas impide que el eje del brazo se muva de la vertical. El uso de un rodamiento implica reducir la distancia sobre la que se apoya el peso, aunque permite fijar por ambos lados las pizas al rodamiento de forma más firme. Finalmente ésta es la opción adoptada para el brazo robótico por aportar una mayor solidez y estabilidad al aportar punto de apoyo continuo en los 360° al rededor del eje.

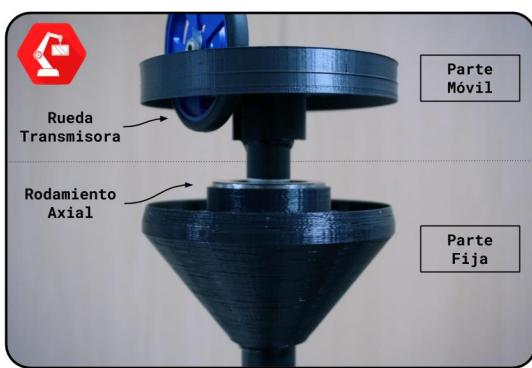


Figura 4.5: Apoyo basado en un rodamiento Axial

Fuente: Autor

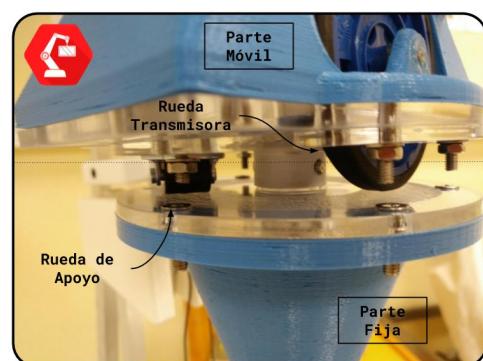


Figura 4.6: Apoyo basado en ruedas

Fuente: Autor

Además del tipo de apoyo se debe valorar en qué lado se sitúa la rueda de transmisión. Anticipando siguientes apartados la estructura del brazo sobresale por uno de los lados. En caso de ubicar la rueda motriz en el lado opuesto la carga dificulta el apoyo de la misma. En el caso de apoyo con ruedas además se apoya todo el peso sobre un voladizo, la distancia entre las dos ruedas de apoyo contrarias a la rueda de transmisión. Aunque en el caso del apoyo mediante rodamiento axial este efecto se suaviza, ya no apoya sobre un voladizo, el diseño final aprovecha la carga del brazo para asegurar el apoyo de la rueda de transmisión. En caso de que la explicación textual no sea suficientemente explicativa, se puede ver una representación gráfica de las diferentes situaciones en la figura 4.7.

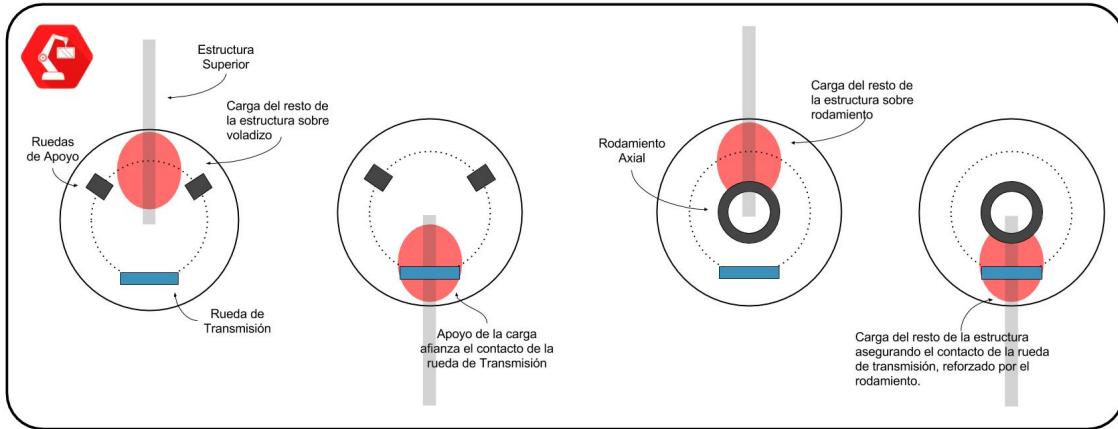


Figura 4.7: Comparación de como afecta el punto de apoyo de la carga superior sobre los diferentes apoyos

Fuente: Autor

Toda esta estructura va montada sobre una base que ejerce de contrapeso para evitar que el eje central del brazo (eje Z) se incline. Como se ha visto anteriormente para esta base se ha adaptado un contrapeso de sombrilla sobre el que encaja el brazo. Este apoyo podría modificarse e incluso adaptar una fijación a otra superficie en caso de montar el brazo en otro entorno.

¿DEBERÍA AÑADIR UNA IMAGEN CON EL PIE Y ESTA ARTICULACIÓN O YA ES PASARSE DE IMÁGENES? LA TENGO HECHA REALMENTE ASÍ QUE...

4.4. Articulaciones dos y tres. Posicionamiento en el plano sagital

Las articulaciones dos y tres, representadas de forma esquemática en la figura 4.1 son las encargadas de posicionar el robot en el plano sagital del robot. Este plano girará al rededor del eje Z gracias a la articulación primera, vista en la sección anterior. Como se ha anticipado en el capítulo 3 estas dos articulaciones están conformadas por dos mecanismos de cuatro barras acopladas en serie.

La figura ?? del capítulo 3 ya anticipaba diferentes métodos de acople entre los mecanismos , cada uno presentando sus virtudes y desventajas.

La primera forma de acoplar ambos mecanismos es mediante una unión triangular (representada en la figura 4.8). En este caso se plantea fijar el segundo mecanismo, situado en el extremo a modo de estructura fija trasladando la responsabilidad de manejar ambos grados de libertad desde la primera articulación. El movimiento del extremo vendrá determinado por la combinación del giro de ambas articulaciones. LLevar ambos grados de libertad a un mismo punto de actuación supone una gran ventaja al evitar alargar la transmisión del movimiento del motor. Según se varie la relación entre el tamaño de las barras así como los ángulos del acople se podrán conseguir diferentes relaciones de movimientos.

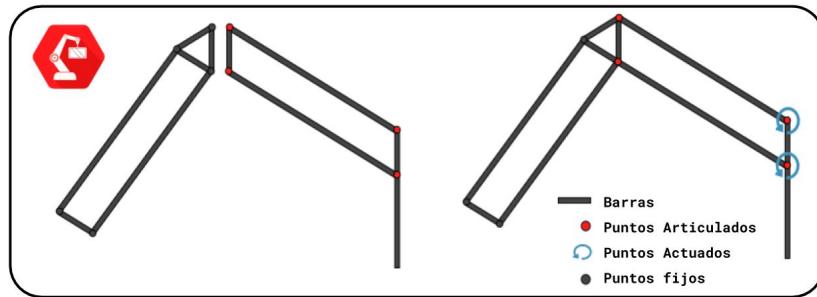


Figura 4.8: Esquema de las articulaciones dos y tres. Estructura de unión triangular

Fuente: Autor

El segundo tipo de acople, lineal, libera los puntos articulares del segundo mecanismo. En este caso ambos mecanismos presentan las mismas dimensiones siendo sus barras iguales dos a dos. Una cadena de eslabones configurada de esta manera, aunque alarga la necesidad de transmitir el movimiento hasta cada eje de giro, tiene la particularidad de que mantiene la orientación del extremo del brazo siempre paralela a la de la barra de acople y en este caso, al eje Z. En la figura 4.9 se pueden ver los puntos sobre los que se actúan ambas articulaciones.

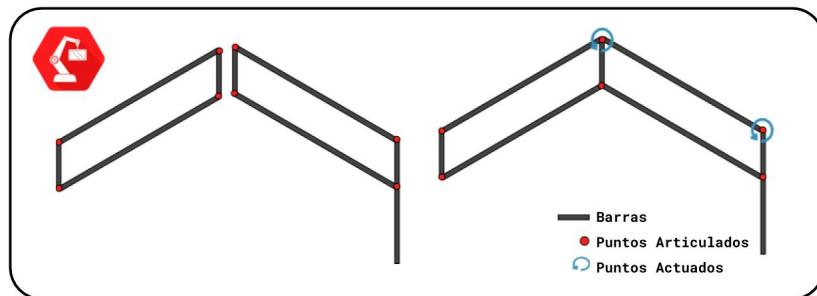
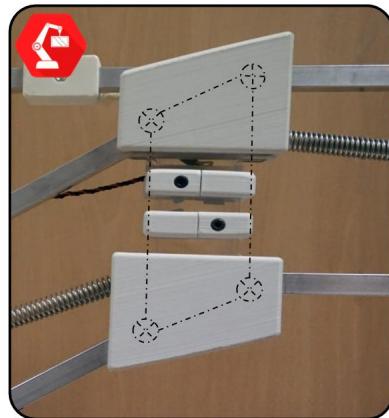


Figura 4.9: Esquema de las articulaciones dos y tres. Estructura de unión lineal

Fuente: Autor

Se ha adelantado en el capítulo 3 la importancia de compensar el peso. En [12] se puede encontrar una demostración matemática de como se puede compensar, mediante el uso de muelles, el peso de los eslabones. Ambas estructuras expuestas permiten este tipo de compensación de la carga. Finalmente se ha optado por la segunda opción (acoplamiento lineal) ya que mantiene la orientación del extremo siempre constante y perpendicular al plano del suelo. Esta característica resulta muy útil ya que simplifica el análisis matemático y posterior control del brazo robótico en gran medida.

Aunque sobre el esquema sobre el papel cuadre bien un acoplamiento lineal, en la realidad esto supone el apilamiento de muchos componentes sobre la misma barra. En ese caso la estructura resultante va engrosando y alejándose del plano sagital, lo que a futuro puede provocar fuerzas transversales indeseadas. Para el montaje este acoplamiento lineal se ha separado formando una geometría de tipo paralelogramo, concretamente romboide. Está conformado por una base de metacrilato con piezas de apoyo impresas en plástico que atrapan, a modo de *sandwich*, las barras estructurales de aluminio.

Figura 4.10: Imagen del acoplamiento *romboide**Fuente: Autor*

Volviendo sobre la compensación del peso del robot, se ha seguido una estrategia similar aunque simplificada a la planteada por [12]. El propósito de los muelles, uno para cada mecanismo, es el de compensar el peso de toda la estructura del brazo de forma que, en vacío, el brazo tenderá a elevarse. Será el peso de la *tablet* una vez se acople el culpable del funcionamiento del brazo tal y como se ha planteado: el manipulador asciende gracias al par aportado por los servomotores; y desciende por gravedad.

En la figura ?? se puede ver el montaje de las barras con los muelles que, de forma cruzada, se oponen al peso de la estructura. En esta imagen se anticipa algunos conceptos y servirá como mapa general para localizar diferentes componentes del brazo robótico sobre los que se focalizará a continuación.

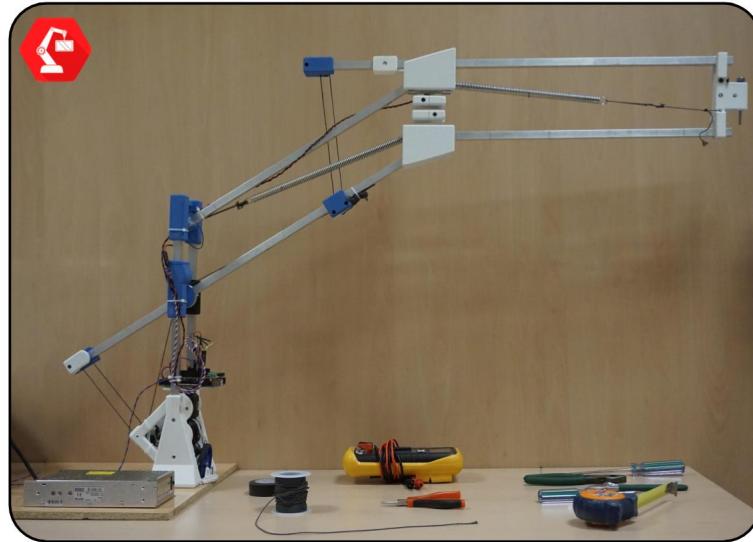


Figura 4.11: Vista genérica de los grados de libertad dos y tres

Fuente: Autor

La transmisión del movimiento para estos dos grados de libertad se ha realizado a través de hilos que van desde los servomotores hasta la articulación. Esos hilos están configurados de forma que no retornan hasta la polea de salida, polea acoplada al

giro del motor. La transmisión está preparada para tirar de la carga de forma activa cuando se quiere subir la muñeca del robot, soltando cuerda en el caso contrario para dejarlo descender poco a poco.

En el caso de ambas articulaciones la carga recibida por el motor ha sido reducida a través de elementos mecánicos sencillos. El enganche para ambas articulaciones hace uso de la misma filosofía. Se ha alejado el punto de actuación del eje de giro para conseguir una palanca sobre la que se actua maximizando así la fuerza realizada por el servo.

A su vez se ha aprovechado el concepto de *polipasto* ampliamente utilizado en actividades de trabajo vertical en las que sea necesario elevar cargas demasiado pesadas para ser manipuladas por una persona. Descritas en detalle por [14] basan su funcionamiento en combinar el uso de poleas fijas y móviles de forma que, el cable que las recorre, queda conectado a un punto fijo. De esta manera se reduce la fuerza necesaria para desplazar la carga, que estará acoplada a una de las poleas móviles.

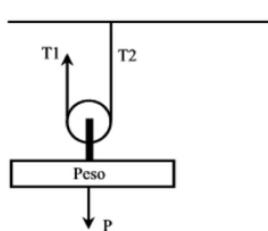


Figura 4.12: Polea Movil
Fuente: [14]

La figura 4.12 representa una polea móvil de forma similar a como se ha aplicado en el proyecto. Como puntualiza [14] siempre y cuando las cuerdas se mantengan paralelas se obtiene una reducción de la fuerza necesaria para mover la carga P del 50 %. Esta sencilla demostración se puede ver en 4.1, que se deduce de un punto de equilibrio estático:

$$P = T_1 + T_2; T_1 = T_2; T_1 = \frac{1}{2}P \quad (4.1)$$

Es importante remarcar que ésta relación es una relación matemática; en la práctica, como bien apunta [14], el rozamiento supondrá pérdida del rendimiento. En función de la calidad de los rodamientos de la polea se obtendrán rendimientos de entre el 70 % y el 97 %.

Volviendo sobre el diseño del brazo se ha diseñado de forma que en todo momento ambas cuerdas se mantendrán paralelas; además las poleas están montadas sobre dos rodamientos que aseguran un mayor rendimiento de la amplificación mecánica. Se pueden ver ambos montajes (palanca y polipasto) en las figuras 4.13 y 4.14 el montaje de estos *columpios* para ambas articulaciones.

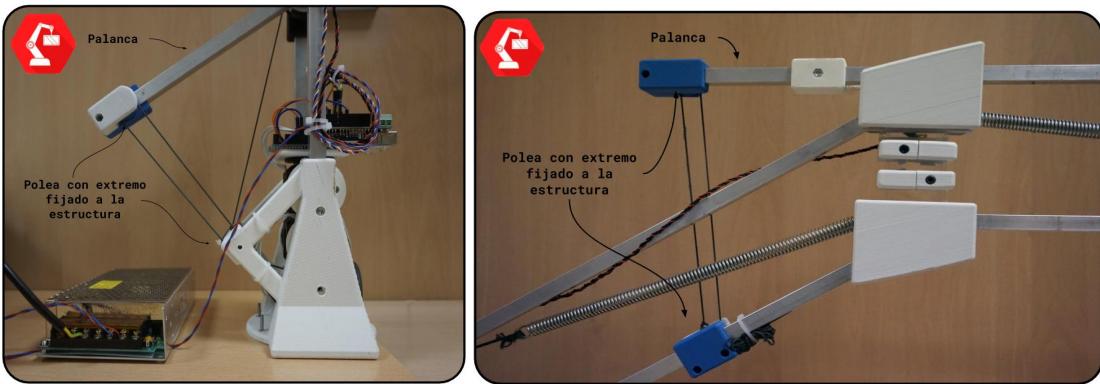


Figura 4.13: Transmisión hasta la primera articulación Figura 4.14: Transmisión del movimiento hasta la segunda articulación

Fuente: Autor

Fuente: Autor

Para conducir los cables de forma controlada hasta los puntos de actuación se ha hecho uso de poleas de redirección, cuyo ensamblaje puede verse en las figuras 4.15 y 4.16.

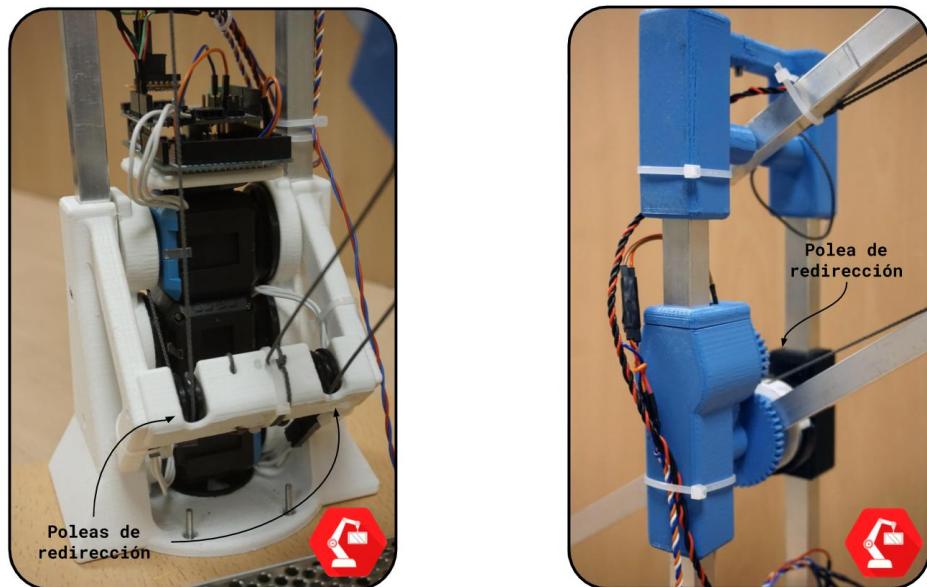


Figura 4.15: Redirección a la salida de los servomotores Figura 4.16: Redirección superior. Eje de la primera articulación

Fuente: Autor

Fuente: Autor

4.5. Grados de libertad de la orientación

¡TBD!



Capítulo 5

Diseño Electromecánico

Aunque se han anticipado tanto textualmente como a través de imágenes algunas características de los componentes electrónicos utilizados aún no se han descrito en detalle. Este capítulo se mete de lleno en los aspectos electromecánicos del brazo robótico, haciendo una descripción de los componentes empleados, algunas pautas para su correcto uso así como su integración dentro de la estructura descrita en el capítulo 4

5.1. Actuadores

La imposición de utilizar unos motores de par reducido puede interpretarse como una desventaja, pero en este caso al quedar descartados los motores de corriente continua y motores paso a paso convencionales queda la alternativa de uso de *smart servos* con todas las ventajas que ofrecen.

Los *smart servos* elegidos para el proyecto son, concretamente, los G15 Cube de la marca Cytron. Estos servos vienen acompañados de una gran variedad de funcionalidades que facilitarán el control y manejo del brazo robótico. Estos servos, como la mayoría de *smart servos*, tienen implementado un sistema de comunicación bidireccional con la placa controladora a través de un tipo de comunicación conocida como *Half Serial Duplex Communication*. Como se cuenta en [15], este tipo de comunicación utiliza un solo cable que podrá operar en una u otra dirección cada vez. Puede darse la situación en que varios componentes intenten comunicar al mismo tiempo en ambas direcciones pudiendo ocasionar graves problemas en la electrónica de los mismos. El uso de este tipo de servos implica el uso de electrónica adicional, no solo a modo de etapa de potencia, si no para la gestión de la comunicación entre los mismos y el microcontrolador.

Utilizar un protocolo de comunicación más complejo permite conectar varios servos a un mismo puerto de comunicación, conectando cada servo al anterior, también conocido como conexión *daisy chain*. En la figura 5.1 se puede ver representada este tipo de cadena y donde se puede ver el aspecto del modelo de servos seleccionado.

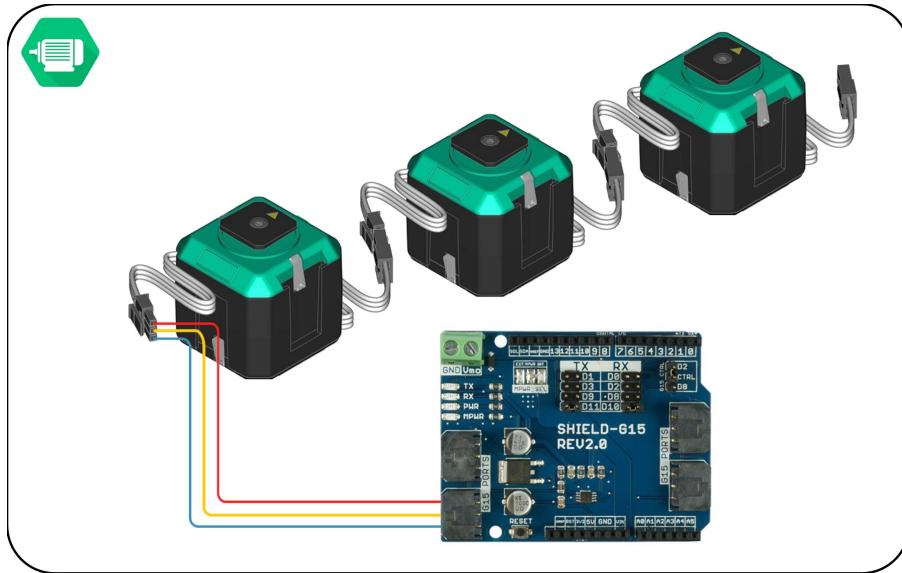


Figura 5.1: Esquema de la conexión de los servos formando un bus serie de servos y la placa *Shield*
Fuente: Montaje del Autor a partir de imágenes del fabricante

Para hacer efectiva la comunicación los servos G15 Cube cuentan con dos puertos de tres cables cada uno. Cada uno cuenta con un conector de aspecto y forma diferente que fuerzan las conexiones en un mismo sentido siempre de forma inequívoca. Estos tres cables, como se puede ver en la imagen 5.2, son utilizados para alimentación, referencia a tierra y canal de información.

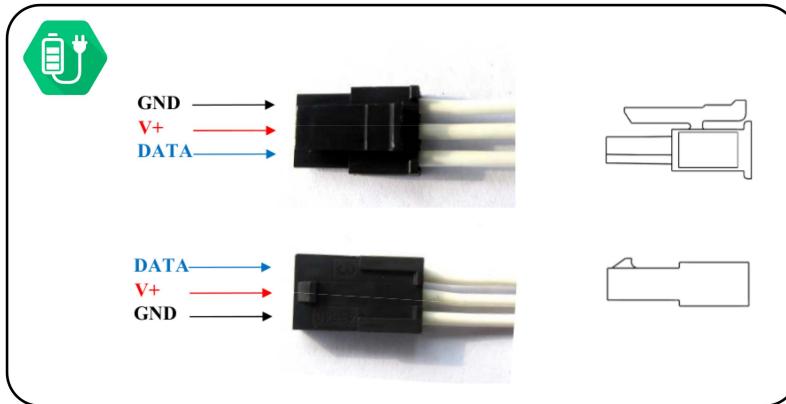


Figura 5.2: Conectores de los G15 Cube y uso de cada uno de los cables
Fuente: Captura obtenida de [16]

Entre las ventajas de utilizar *smart servos* es que se cuenta con información relevante que se podrá *preguntar* al servo cuando sea necesaria. Estos servos tienen diferentes modos de funcionamiento, aunque se utilizará principalmente el modo de **giro continuo**. Para este caso los servos ofrecen un control en par de manera que se podrán enviar comandos del par que deberá ejercer. Como información relevante a consultar ofrece datos de posición del servo, par realizado y sentido del mismo, temperatura, voltaje de alimentación, velocidad y sentido de movimiento, entre otros.

Cabe destacar algunas características importantes de los mismos, que se encuen-

tran resumidas en la tabla 5.1. Concretamente es de destacar los $12kg \cdot cm$ de par efectivo; un par relativamente bajo que se aprovechará como medida extra de protección a usuarios del brazo robótico tal y como se ha anticipado en capítulos anteriores.

Tabla 5.1: Características relevantes de los Servos G15 de Cytron.

Fuente: Tabla traducida y resumida de [16]

Características eléctricas			
Parámetro	Valor Mínimo	Valor Típico	Valor Máximo
Voltaje	6,5V	12V	17,8V
Consumo de corriente (12V)			1,5A
Temperatura de funcionamiento	0°C		80°C

Especificaciones técnicas	
Peso	63g
Par capaz de realizar (a 12V)	12kg · cm
Par capaz de soportar	15km · cm
Margen angular de operación	360° en giro continuo
Máxima velocidad (en vacío a 12V)	63RPM
Comunicaciōn	Half duplex asynchronous serial communication (7812,5bps – 500kbps)

5.2. Interfaz servos-microcontrolador

Pasada la descripción de los actuadores, los G15 Cube Servo, se hace patente la necesidad de una etapa intermedia entre la placa controladora y los servos que gestione la comunicación entre ambos de forma segura y que desacople la alimentación del controlador de la alimentación de los servos, que requieren un voltaje de 12V (ver 5.1).

Es la propia marca que fabrica los servos, Cytron Technologies, la que suministra una placa auxiliar o *shield* con este propósito. Concretamente se utilizará la segunda generación de dicha placa, vista en la figura 5.1 de la sección anterior.

Para la alimentación de los servos se ofrecen dos posibles entradas remarcadas en la figura 5.3 con los colores azul y rojo. Para alternar de una a otra habrá que, mediante el uso de un soldador, modificar la conexión recuadrada en amarillo para habilitar la opción deseada deshabilitando la contraria.

- Alimentación externa (recuadrada en azul): en este caso se conecta la fuente de alimentación directamente a los conectores pasando el voltaje a los cables de alimentación de los servos.
- Alimentación mixta shield-controlador (recuadrada en rojo): en este caso la alimentación se comparte con la placa controladora (que deberá rectificar el voltaje de entrada a valores aceptables para la misma). Por defecto esta es la entrada que viene habilitada; se ha mantenido ya que permite la alimentación simultánea de los servos y de la placa controladora a partir de la misma fuente de alimentación (se verá en secciones posteriores la elección del controlador y otros aspectos).

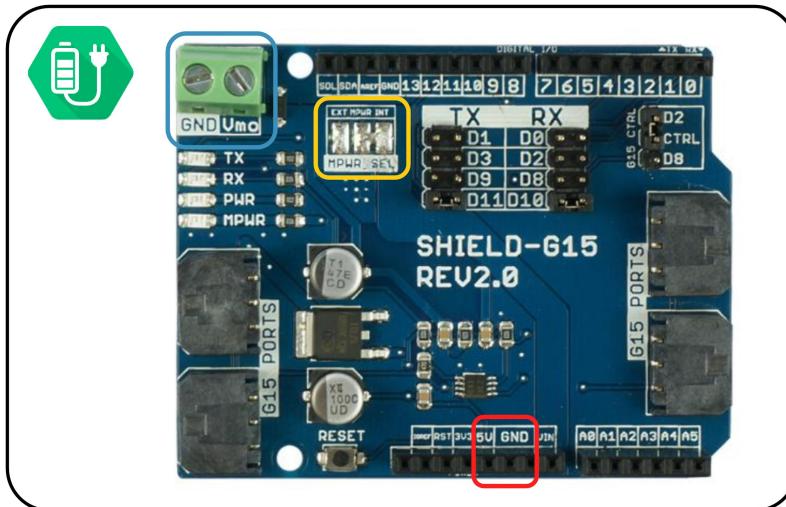


Figura 5.3: Posibilidades para la alimentación de los servos

Fuente: Captura obtenida de [16] y editada por el Autor del proyecto

En la figura 5.3 pueden distinguirse una serie de pines de conexión con las letras RX, TX y CTRL. Esta placa está pensada para funcionar a modo de interfaz entre un puerto serie común (con un cable de emisión y otro de recepción de datos) y un puerto serie de tipo *Half Duplex* como el empleado por los servos.

Como se describe en [16] la *shield* incluye integrada un circuito integrado (concretamente el 74HC126 IC) que resuelve los problemas de comunicación inherentes a la comunicación bidireccional por un solo hilo. A través de un pin de control (CTRL en la shield) se gestiona la conexión entre el hilo del *Half Duplex* y los hilos del puerto serie alternando de uno a otro en función del estado de la señal de control (alto nivel o bajo nivel).

5.3. Placa controladora

La placa *shield* descrita en el apartado anterior está especialmente diseñada para encajar en placas tipo Arduino, concretamente el modelo Arduino Uno. En el marco de este proyecto se ha realizado una fase del desarrollo utilizando como base una placa Arduino Uno, aunque posteriormente se ha cambiado a un Arduino Mega. Más adelante se explicará la motivación de dicho cambio, pero merece volver sobre el aspecto de la alimentación de la placa descrito en el apartado anterior. Según se especifica en [17] y en [18] el voltaje de entrada recomendado abarca desde los 7 a los 12V teniendo como limitación inferior un mínimo de 6V y un máximo de 20V. En este caso se aplicarán 12V, que quedan incluidos dentro del rango recomendado por el fabricante.

En el caso de la placa Arduino Uno la *shield* viene preparada para encajar sobre la misma. La comunicación con los servos está pensada para efectuarse de dos formas:

- A través de un puerto serie UART hardware: en el caso de la placa Arduino Uno solo dispone del puerto conectado a los pines 0 y 1, que también es el usado para la carga de software y comunicación con el ordenador, por lo que

queda descartado.

- Emulando un puerto serie mediante software en otros pines de la placa. La *shield* trae una serie de *jumpers* **¿DEFINIR O NO ES NECESARIO?** que permiten cambiar entre una selección de pines para cada caso (RX, TX o CTRL).

Como se puede ver, utilizando una placa Arduino Uno la comunicación con los servos queda relegada a un puerto emulado por software. Esta es la razón principal por la cual se decide cambiar y utilizar una Placa Arduino Mega2560, que además presenta mayores prestaciones respecto a memoria (ver tabla 5.3). Para el control del brazo robótico y el diseño y testeo del software es necesario optimizar la velocidad de comunicación entre los dispositivos al máximo. La placa Arduino Mega incluye tres puertos serie hardware adicionales que se podrán puentear a la placa *shield* para ser utilizados. Esta conexión se puede ver en la figura 5.4. De esta forma se podrá aprovechar todo el potencial de la comunicación a través de un puerto serie hardware.

Para hacerse una idea de la importancia que tiene este cambio se ha forzado la comunicación en ambos casos para obtener los máximos en los cuales sería viable trabajar. Los datos presentados en la tabla 5.2 se han obtenido de forma experimental bajo el marco de este proyecto. Entre los mismos se puede apreciar la gran diferencia existente entre las diferentes formas de comunicación. Las velocidades se han ido duplicando (a modo de convencionalismo las velocidades de comunicación estándar para Placas Arduino suelen obtenerse de esta manera) hasta llegar al máximo que permite una comunicación satisfactoria.

Tabla 5.2: Comparativa entre placas Arduino Uno y Arduino Mega2560

Fuente: Tabla con información resumida de [17] y [18]

Tipo de comunicación	Velocidad máxima en baudios	Velocidad máxima en bytes/milisegundo
Puerto Serie Sowtware (AUno): Shield-controlador	57600 bauds	7.2 bytes/ms
Puerto Serie Hardware (AMega): Shield-controlador	460800 bauds	57.6 bytes/ms
Puerto Serie Hardware (Ambas): controlador-ordenador	921600 bauds	115.2 bytes/ms

En este proyecto concreto se enlazaran diferentes lazos de control a diferentes frecuencias de refresco (ver capítulo 8) que exigirán el máximo de la capacidad comunicativa entre los dispositivos. Los datos máximos obtenidos para la placa Arduino Mega son los utilizados para el funcionamiento del robot.

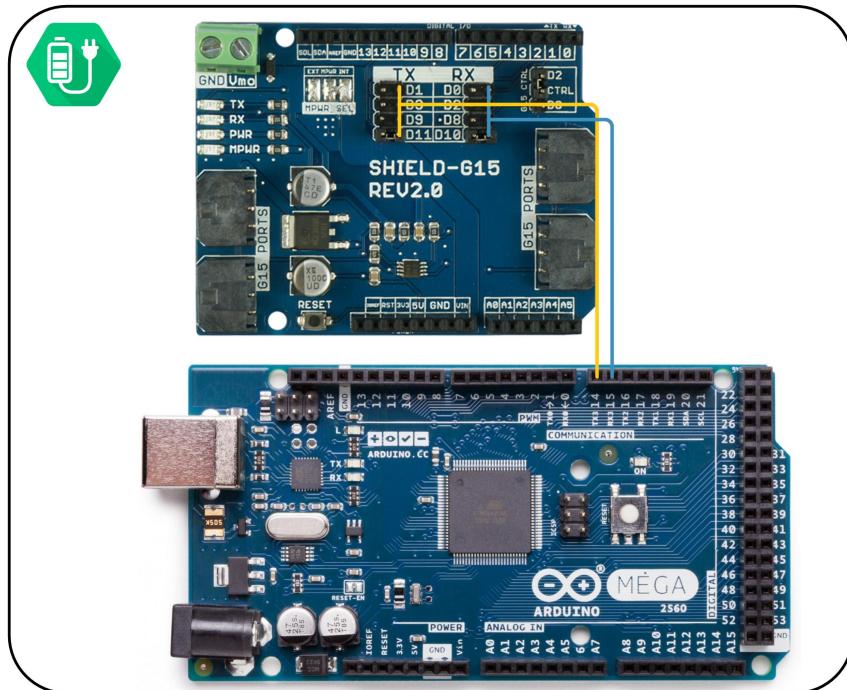


Figura 5.4: Esquema de la conexión entre la placa Shield y Arduino para utilizar los puertos Hardware Serie de la Arduino Mega

Fuente: Montaje del Autor a partir de imágenes del fabricante

Tabla 5.3: Comparativa entre placas Arduino Uno y Arduino Mega2560

Fuente: Tabla con información resumida de [17] y [18]

	Arduino Uno	Arduino Mega2560
Número de pines entrada/salida	14	54
Memoria flash	32KB	256 KB
SRAM	2KB	8KB
EEPROM	1KB	4KB
Velocidad de reloj	16MHz	16Mhz

5.4. Sensores



Capítulo 6

Diseño del software

En este capítulo se hace una descripción detallada del *software* involucrado en este proyecto. Primeramente se hace un repaso de la estructura de directorios, que es característica y viene previamente marcada por el software utilizado. Una vez explicado se pasará a describir las diferentes librerías que se han desarrollado (sección 6.1) y del directorio de fuentes principal SRC (sección 6.2). Se deja para una sección posterior la interacción e integración de estos objetos así como una explicación más detallada del flujo de la información, funcionamiento del sistema como ejemplos de uso (sección 6.3). Finalmente se exponen los test realizados sobre el software (sección 6.4) así como la gestión de la complejidad (sección 6.5) del proyecto.

Para el desarrollo y test del software se ha utilizado el editor *Atom* ampliando su funcionalidad con el paquete *PlatformIO*, que expande las capacidades del editor base para permitir trabajar con diferentes placas – **COMPLETAR** –, entre ellas las de la gama de *Arduino*.

La elección de esta herramienta conlleva un formato en el árbol de directorios en los que se separa el código debido a la forma que tiene de compilar y lincar los diferentes ficheros. De esta manera y para mantener el orden los ficheros de código se separan en distintos directorios:

- lib: directorio donde se introducen, en carpetas, las librerías que se van a utilizar.
- src: directorio donde se introduce el fichero o ficheros de código principales.
- test: directorio donde se introducen los ficheros donde se codifican los test. Estos irán metidos dentro de directorios con el nombre de cada test.

Concretamente, en el caso de este proyecto, queda el siguiente árbol de directorios en el cual se han expandido hasta el nivel de ficheros en algunos casos de modo que sirvan de ejemplo:

```

-- Sw
|-- lib
|   |-- debug
|   |   |-- debug.cpp
|   |   |-- debug.h
|   |-- joint_handler
|       |-- joint_handler.cpp

```

```

|   | + joint_handler.h
|--- joint_rha
|--- rha_types
|--- robot_rha
|--- servo_rha
|--- chuck_handler
|--- readme.txt
|-- src
|--- main.cpp
|--- utilities.cpp
|--- utilities.h
|-- test
|--- a_test_servo_rha
|   | + test_servo_rha.cpp
|--- b_ttest_servo_realest_joint_rha
|--- c_ttest_joint_handler_mock
|-- platformio.ini

|-- code_analysis
|-- makeAnalysis.sh

```

En los siguientes apartados se hará un análisis detallado del contenido de estos directorios. Se desarrollan primero los ficheros con información auxiliar y posteriormente en una jerarquía desde más afuera <— *WTF?* detallando luego los componentes internos.

6.1. Librerías (directorio lib)

6.1.1. debug

Dentro de este directorio se encuentra el fichero *debug.h* donde se han definido macros para *debug* de los diferentes espacios. A continuación se muestra un ejemplo de como se codifican dichas macros para hacer un seguimiento de la clase *servo_rha* (Ver apartado 6.1.5).

```

#ifndef DEBUG_SERVO_RHA
#define DebugSerialSRHALn(a) { Serial.print("[DC] --ServoRHA::"); Serial.
    println(a); }
#define DebugSerialSRHALn2(a, b) { Serial.print("[DC] --ServoRHA::");
    Serial.print(a); Serial.println(b); }
#define DebugSerialSRHALn4(a, b, c, d) { Serial.print("[DC] --ServoRHA::");
    Serial.print(a); Serial.print(b); Serial.print(c); Serial.println(d);
}
#else
#define DebugSerialSRHALn(a)
#define DebugSerialSRHALn2(a, b)
#define DebugSerialSRHALn4(a, b, c, d)
#endif

```

Como se puede apreciar estas macros utilizan la comunicación serial de *Arduino*, concretamente la función *print* y *println* de la librería *Serial* para mostrar los mensajes en un formato determinado. Además se definen de tal forma que se pueden activar o desactivar (en este mismo fichero *debug.h*) de forma que se enviarán o no los mensajes de *debug*. A través de este tipo de mensajes se puede hacer un seguimiento de la ejecución de los diferentes métodos o funciones de la librería afectada para ubicar fallos en los mismos.

Para activar la opción de *debug* bastará con descomentar la línea correspondiente:

```
// #define DEBUG_SERVO_RHA
```

```
// #define DEBUG_TEST_SERVO_RHA_MOCK
#define DEBUG_TEST_SERVO_RHA_REAL
// #define DEBUG_CYTRON_G15_SERVO
// #define DEBUG_TEST_CYTRON_G15_SERVO
```

6.1.2. rha_types

Dentro de este directorio se encuentra el fichero *rha_types.h* donde se definen algunos tipos de datos y estructuras que se usan en el proyecto.

Estas estructuras de datos se listan a continuación, pudiendo ver sus respectivos diagramas de clases en la figura 6.1:

- *SpeedGoal* condensa en un solo objeto toda la información necesaria para codificar un objetivo de velocidad.
- *Regulator* encapsula el funcionamiento de un *Regulador PID* estandar. Guarda los valores de las constantes así como de la integral del error para luego, pasándole error, derivada del error e integral del error en un intervalo poder devolver la salida del regulador.
- *Timer* codifica un temporizador (en milisegundos) de forma que se le podrá preguntar al objeto si el tiempo ya ha pasado, bloqueando o no la ejecución del programa hasta el final del tiempo.
- *TimerMicroseconds* hereda las características del objeto *Timer* funcionando en microsegundos.

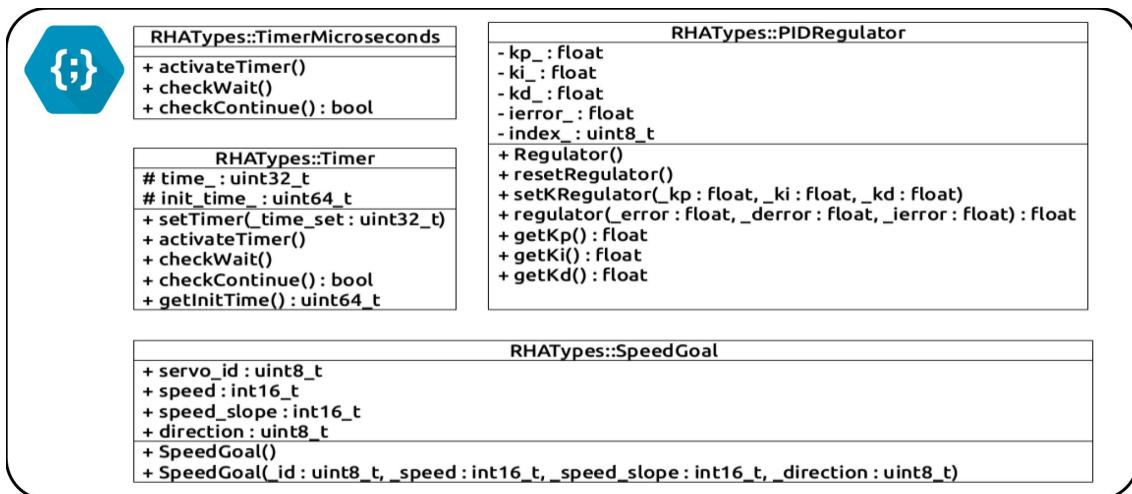


Figura 6.1: Estructuras de datos auxiliares

Fuente: Autor

Se puede consultar información de más bajo nivel referente a estos tipos de datos en el Anexo D sección – **COMPLETAR** –.

6.1.3. joint_handler

La librería *joint_handler* se hace cargo de generar un objeto de la clase *JointHandler* que será en encargado de gestionar la sincronización de todas las articulaciones. Este objeto es propietario de las articulaciones e implementa un método que cíclicamente sincroniza el funcionamiento de todas las articulaciones. En el objeto *JointHandler* se implementa a su vez la comunicación con los *servos*, es decir, el encapsulamiento de los paquetes de datos con la información genérica de forma que la información que se envía cumpla con el protocolo de comunicación que comparten los *servos*. Además implementa las funciones que gestionan el envío y recepción de dichos paquetes de datos.

El bucle de control implementado se encarga de ir llamando una a una a todas las articulaciones para que hagan las siguientes operaciones (se puede ver representado el funcionamiento de dicho ciclo en la Figura 6.2):

1. Asegurar que cada articulación actualice la información propia, compuesta por la posición recibida de la realimentación así como toda la información proveniente del servo (datos de posición, velocidad, par soportado y dirección en que se aplican, voltaje y temperatura).
2. Llamar a cada articulación para que se realicen los cálculos correspondientes del *torque* que se enviará calculado a partir del error entre la velocidad real y deseada. Este valor queda guardado en cada servo para ser posteriormente empaquetado.
3. Invocar a cada servo para que se adhieran al paquete de información que se va a enviar.
4. Se enpaqueta la información a enviar a los diferentes *servos* en un mismo paquete, añadiendo posteriormente el correspondiente encabezado así como el comprobante de que el paquete se ha recibido correctamente (checksum). Una vez preparado el paquete se envía por el puerto serie.

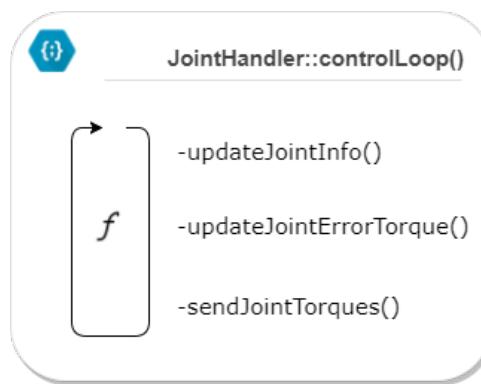


Figura 6.2: Esquema de ejecución de bucle de control de joint_handler

Fuente: Autor

Como se puede intuir es en este objeto donde realmente se implementa el lazo de control de velocidad para todos los *servos* conectados al bus. Esta serie de operaciones descrita anteriormente constituye, de forma discretizada, el lazo de control

representado en la Figura 6.3 para cada servo y asegura su correcto funcionamiento y sincronismo.

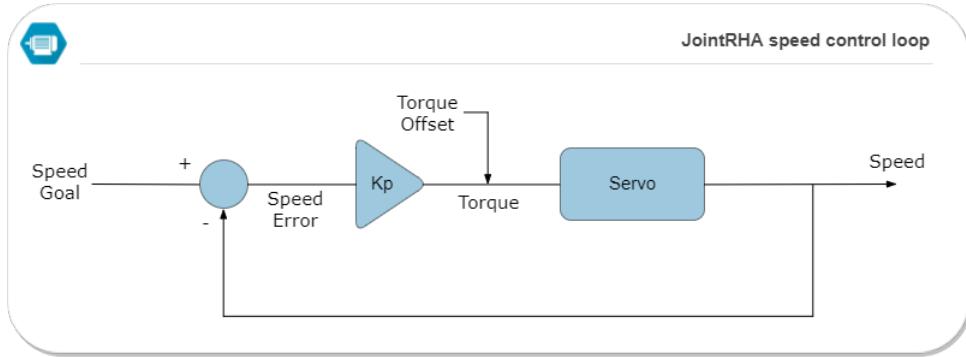


Figura 6.3: Lazo de control de la velocidad del servo. Cálculos realizados por el objeto *ServoRHA*.



Figura 6.4: Atributos y métodos más relevantes del objeto *JointHandler*

Fuente: Autor

Se puede consultar información de más bajo nivel referente al objeto *JointHandler* así como a sus atributos y métodos en el Anexo D sección – **COMPLETAR** –.

6.1.4. joint_rha

La librería *joint_rha* implementa un objeto de tipo *JointRHA* que aúna en un mismo objeto las lecturas y capacidades del objeto *ServoRHA* (explicado en la sección 6.1.5) junto con la realimentación de posición de la articulación en cuestión.



Figura 6.5: Atributos y métodos más relevantes del objeto *JointRHA*

Fuente: Autor

Se puede consultar información de más bajo nivel referente al objeto *JointRHA* así como a sus atributos y métodos en el Anexo D sección – **COMPLETAR** –.

6.1.5. servo_rha

La librería *servo_rha* implementa un objeto del tipo *ServoRHA* que está encargado de gestionar toda la información referente al servo. Se encarga de encapsular la información específica a un servo en paquetes de datos a petición del objeto *JointHandler* que luego serán procesados por el mismo previo a ser enviados a través del bus. Estos paquetes se forman a partir de la información contenida en el objeto *ServoRHA*, que además de formar los paquetes a enviar almacena la información referente al servo que se recibe de los mismos.

Gestiona además una parte importante referente al lazo de control de velocidad del servo vista en el apartado 6.1.3. El objeto *ServoRHA* contiene los datos del regulador utilizado así como el offset a aplicar. De esta forma, recibiendo el error realiza las operaciones para calcular y empaquetar el *torque* deseado. En la Figura 6.6 se representa, recuadrado, la parte correspondiente del lazo de control de velocidad (representado anteriormente en la Figura 6.3) que efectúa el objeto en cuestión.

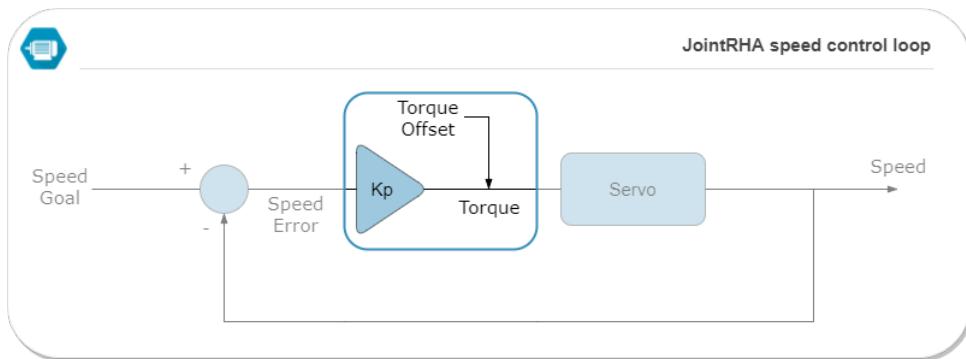
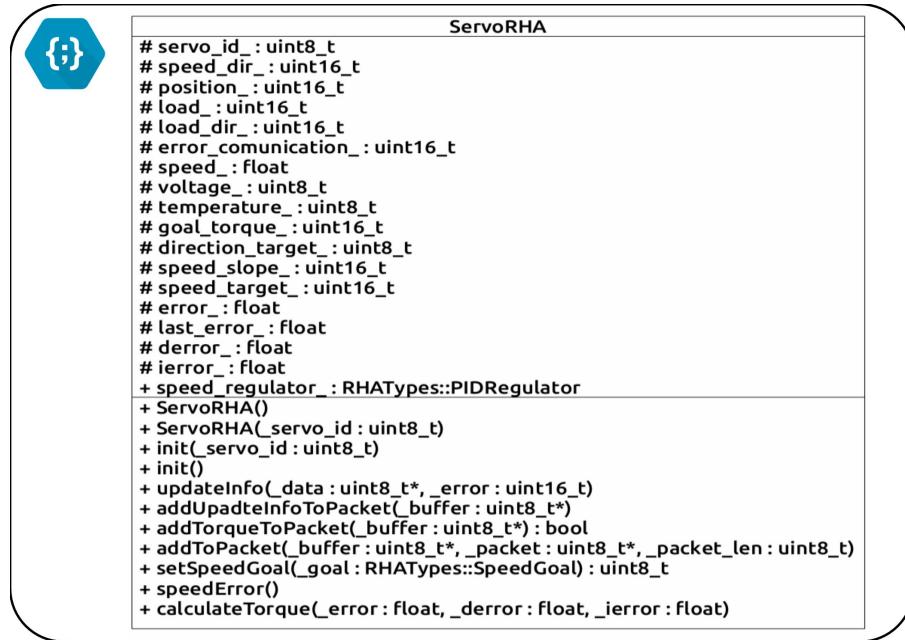


Figura 6.6: Lazo de control de la velocidad del servo. Cálculos realizados por el objeto *ServoRHA*.

Figura 6.7: Atributos y métodos más relevantes del objeto *ServoRHA**Fuente: Autor*

Se puede consultar información de más bajo nivel referente al objeto *ServoRHA* así como a sus atributos y métodos en el Anexo D sección – **COMPLETAR** –.

6.1.6. robot_rha

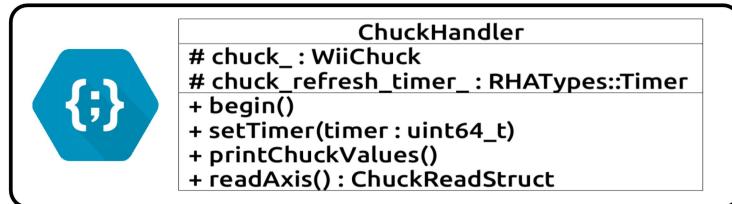
En la librería *robot_rha* se implementa el objeto de tipo *RobotRHA* encargado de coordinar el funcionamiento del robot. Implementa el ciclo de más alto nivel donde se actualizan los objetivos ya sean de posicion o velocidad para llamar a las funciones correspondientes que lo traducen a valores articulares que luego se ejecutan.

Tiene diferentes modos de funcionamiento según de donde provengan los comandos a seguir:

- Control a través de un *Nunchuk* de la consola *Wii*. Se controla la velocidad del robot en sus diferentes ejes a través del joystick y los botones del mando.

6.1.7. chuck_handler

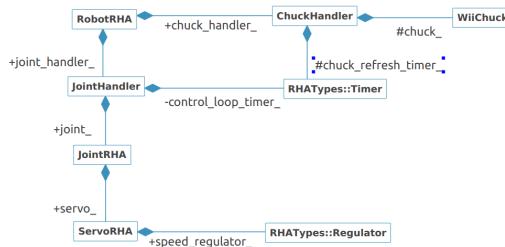
La librería *chuck_handler* codifica el objeto de tipo *ChuckHandler* que se encarga de implementar todo lo necesario para realizar lecturas del mando *Nunchuck* de la *Wii* y devolver comandos de velocidad con un periodo establecido. Se llamará al método correspondiente de forma continuada devolviendo este los valores de velocidad cuando se cumpla el tiempo mínimo establecido.

Figura 6.8: Atributos y métodos más relevantes del objeto *ChuckHandler**Fuente: Autor*

6.2. SRC. Fichero de código principal

6.3. Interacción entre objetos, flujo de la información y de procedimientos

Como se ha visto en los apartados anteriores (sección 6.1) el control de la información está separado entre los diferentes objetos. Primeramente se va a detallar como se distribuye y gestiona la información para el control de velocidad, revisando como se construyen los paquetes necesarios para satisfacer los requerimientos del protocolo de comunicación de los servos.

Figura 6.9: Relaciones entre las diferentes Clases **HACER IMAGEN CON EL MISMO ESTILO QUE ANTERIORMENTE***Fuente: Autor*

Como se ha visto es el objeto de tipo *JointRHA* el encargado de almacenar la información concreta de cada articulación. Aúna la información de la realimentación de posición del potenciómetro con la contenida en el objeto *ServoRHA* que contiene. Este último almacena la información referente al servo, tanto el estado como la acción de control de velocidad así como la próxima consigna de velocidad (traducida a par) que se va a enviar.

La información fluye entre los diferentes objetos en forma de paquetes (vectores o arrays de *bytes*) que se van completando por el responsable de cada operación.

El objeto JoingHandler es el encargado de adecuar la información al protocolo de comunicación concreto de los servos. Para ello pide a los servos la información concreta que se va a enviar a cada en cada caso. Se exponen a continuación dos ejemplos:

1. Un primer caso de lectura de información (actualizar la información de los servos)
2. Un ejemplo de una operación de escritura (enviar una consigna de velocidad a los servos).

En el primer caso será el objeto *JointHandler* quien pase un paquete vacío al objeto *JointRHA* (en este caso se pretende actualizar la información del objeto *ServoRHA* por lo que se accederá al mismo directamente) con una petición de que se almacene en dicho paquete la información referente a la lectura: cuantos bytes se quieren leer y a partir de que dirección de memoria en los registros de los servos (se puede ver una tabla con todas las direcciones de memoria disponibles en los servos en el Anexo E).

De esta forma el Servo recibe un paquete vacío que rellena con la información correspondiente y lo devuelve al objeto *JointHandler*, que, a partir de esa información monta un paquete acorde con el protocolo de los servos añadiendo la cabecera correspondiente, la comprobación del error y la orden para que se realice una operación de lectura, entre otros datos.

Una vez enviado, y tratándose de una operación de lectura se espera la respuesta por parte del servo correspondiente, en caso de ser satisfactoria (no se haya producido ningún error durante la operación) se filtra nuevamente la cabecera y demás información propia del protocolo de comunicación para volver a enviar, ahora lleno, el paquete de información con los bytes pedidos al objeto *ServoRHA*.

INCLUIR DIAGRAMA CON EL FLUJO DE INFORMACIÓN Y EJEMPLO

El segundo caso será equivalente. El servo rellena en el *buffer* la información propia como es el ID del servo, la posición sobre la que se quiere escribir, el número de bytes a escribir y la información que se escribirá.

El objeto *JointHandler* se encarga de llenar los datos propios del protocolo añadiendo en este caso una orden de escritura en el paquete.

INCLUIR DIAGRAMA CON EL FLUJO DE LA INFORMACIÓN Y EJEMPLO

COMENTAR EL RESTO DEL SOFTWARE

6.4. Test y verificación del software

Como parte del proyecto se han desarrollado una serie de test para verificar el correcto funcionamiento de las diferentes librerías. El *testing* del *software* se ha utilizado como herramienta implementándose solo en aquellos casos en que facilita la verificación y comprobación del correcto funcionamiento del código implementado. El *testing* del código no forma parte del núcleo del proyecto por lo que no se han establecido porcentajes mínimos de cobertura (cantidad de código que se está evaluando a través de las pruebas) ni objetivos mínimos. Los test son puramente funcionales desarrollándose los necesarios y no forzando el desarrollo de los distintos niveles de *test*: test unitarios, test de integración y test de sistemas. **ALGUNA CITA QUE HABLE DE LOS NIVELES DE TEST**

Para el desarrollo y ejecución de dichos test se ha utilizado la funcionalidad de test que viene integrada en *PlatformIO*: *PlatformIO Test*. Esta herramienta, permite

definir una serie de test que pueden ser ejecutados en la propia placa. De esta forma se puede automatizar el proceso de test.

Una completa definición de test (desde test unitarios hasta test de integración) permite controlar de forma continuada el correcto funcionamiento del sistema frente a modificaciones en el código. De forma genérica estos test se han agrupado bajos los siguientes nombres:

- test_rha_types
- test_pid_regulator
- test_servo_rha
- test_joint_rha
- test_joint_handler_mock

En el directorio SW/test se encuentran definidos los diferentes test que se realizan. Cada fichero de test, destinado a testear de forma parcial o completa una librería, tiene diferentes funciones de test definidas.

Para realizar un test sobre un método o grupo de métodos se define una función de test en la que se define la ejecución que se va a realizar, con las entradas predefinidas de forma que se puede comprobar como ciertas salidas o parámetros internos satisfacen las necesidades impuestas. Para la definición de estas condiciones así como de los test se sigue el formato propuesto desde *PlatformIO Test* y la API que adjuntan.

Una vez codificados los ficheros para testing es necesario especificar su uso en el fichero de código principal **src/main.cpp** mediante las directivas al preprocesador que se muestran a continuación. De esta forma mientras se realizan los test no se ejecutará el programa principal.

```
#ifndef UNIT_TEST // disable program main loop while unit testing in progress
...
...
...
#endif
```



```
=====
[test::b_test_pid_regulator] Building... (1/3)
=====
Please wait...
test/b test_pid_regulator/test_pid_regulator.cpp: In function 'void test_class_regulator_function_regulator_simple_acumulative_overload()':
test/b test_pid_regulator/test_pid_regulator.cpp:96:11: warning: unused variable 'ierror_accumulated2' [-Wunused-variable]
float ierror_accumulated2 = 0;
^

=====
[test::b_test_pid_regulator] Uploading... (2/3)
=====
Please wait...
Warning! Please install '99-platformio-udev.rules' and check that your board's PID and VID are listed in the rules.
https://raw.githubusercontent.com/platformio/platformio/develop/scripts/99-platformio-udev.rules
Reading | #####| 100% 0.01s
Writing | #####| 100% 1.39s
Reading | #####| 100% 1.05s

=====
[test::b_test_pid_regulator] Testing... (3/3)
=====
If you don't see any output for the first 10 secs, please reset board (press reset button)

test/b test_pid_regulator/test_pid_regulator.cpp:122:test_class_regulator_function_setRegulator [PASSED]
test/b test_pid_regulator/test_pid_regulator.cpp:123:test_class_regulator_function_resetRegulator [PASSED]
test/b test_pid_regulator/test_pid_regulator.cpp:124:test_class_regulator_function_regulator_simple [PASSED]
test/b test_pid_regulator/test_pid_regulator.cpp:125:test_class_regulator_function_regulator_simple_acumulative_1 [PASSED]
test/b test_pid_regulator/test_pid_regulator.cpp:126:test_class_regulator_function_regulator_simple_acumulative_2 [PASSED]
test/b test_pid_regulator/test_pid_regulator.cpp:127:test_class_regulator_function_regulator_simple_acumulative_overload [PASSED]
-----
6 Tests 0 Failures 0 Ignored
```

Figura 6.10: Salida de Platformio Test para cada caso

Fuente: Autor



```
===== [test::d_test_servo_rha] Testing... (3/3) =====
If you don't see any output for the first 10 secs, please reset board (press reset button)

test/d_test_servo_rha/test_servo_rha.cpp:289:test_function_compareSpeed [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:290:test_function_compareAngles [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:291:test_function_addToPacket [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:64:test_function_updateInfoToPacket: Element 3 Expected 8 Was 6 [FAILED]
test/d_test_servo_rha/test_servo_rha.cpp:293:test_function_addReturnOptionToPacket [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:294:test_function_setTorqueOnOffToPacket [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:295:test_function_set_exit_WheelModeToPacket [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:136:test_function_updateInfo:FAIL: Expected 50 Was 5. Speed [FAILED]
test/d_test_servo_rha/test_servo_rha.cpp:157:test_function_calculateTorque:FAIL: Expected 0 Was 101. Torque test 1 [FAILED]
test/d_test_servo_rha/test_servo_rha.cpp:196:test_function_addTorqueToPacket:FAIL: Element 3 Expected 150 Was 121 [FAILED]
test/d_test_servo_rha/test_servo_rha.cpp:299:test_function_addPingToPacket [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:300:test_function_setSpeedGoal [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:249:test_function_speedError:FAIL: Values Not Within Delta . Test 1 [FAILED]
-----
13 Tests 5 Failures 0 Ignored
```

Figura 6.11: Salida de Platformio Test con casos fallidos

Fuente: Autor


```
===== [TEST SUMMARY] =====
test:a_test_rha_types/env:megaatmega2560 [PASSED]
test:b_test_pid_regulator/env:megaatmega2560 [PASSED]
test:e_test_joint_rha/env:megaatmega2560 [PASSED]
test:f_test_joint_handler_mock/env:megaatmega2560 [PASSED]
test:log/env:megaatmega2560 [IGNORED]
test:d_test_servo_rha/env:megaatmega2560 [FAILED]
=====
[FAILED] Took 71.92 seconds =====
```

Figura 6.12: Salida de Platformio Test con el resumen de los casos de test

Fuente: Autor

6.4.1. Test librería RHATypes



```
===== [test::a_test_rha_types] Testing... (3/3) =====
If you don't see any output for the first 10 secs, please reset board (press reset button)

test/a_test_rha_types/test_rha_types.cpp:65:test_class_timer_function_checkwait [PASSED]
test/a_test_rha_types/test_rha_types.cpp:66:test_class_timer_function_checkcontinue [PASSED]
test/a_test_rha_types/test_rha_types.cpp:67:test_class_timer_micros_function_checkwait [PASSED]
test/a_test_rha_types/test_rha_types.cpp:68:test_class_timer_micros_function_checkcontinue [PASSED]
-----
4 Tests 0 Failures 0 Ignored
```

Figura 6.13: Test satisfactorios RHATypes

Fuente: Autor


```
===== [test::b_test_pid_regulator] Testing... (3/3) =====
If you don't see any output for the first 10 secs, please reset board (press reset button)

test/b_test_pid_regulator/test_pid_regulator.cpp:122:test_class_regulator_function_setRegulator [PASSED]
test/b_test_pid_regulator/test_pid_regulator.cpp:123:test_class_regulator_function_resetRegulator [PASSED]
test/b_test_pid_regulator/test_pid_regulator.cpp:124:test_class_regulator_function_Regulator_simple [PASSED]
test/b_test_pid_regulator/test_pid_regulator.cpp:125:test_class_regulator_function_Regulator_simple_acumulative_1 [PASSED]
test/b_test_pid_regulator/test_pid_regulator.cpp:126:test_class_regulator_function_Regulator_simple_acumulative_2 [PASSED]
test/b_test_pid_regulator/test_pid_regulator.cpp:127:test_class_regulator_function_Regulator_simple_acumulative_overload [PASSED]
-----
6 Tests 0 Failures 0 Ignored
```

Figura 6.14: Test satisfactorios PIDRegulator (caso dentro de RHATypes)

Fuente: Autor

6.4.2. Test librería ServoRHA



```
===== [test::d_test_servo_rha] Testing... (3/3) =====
If you don't see any output for the first 10 secs, please reset board (press reset button)

test/d_test_servo_rha/test_servo_rha.cpp:289:test_function_compareSpeed [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:290:test_function_compareAngles [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:291:test_function_addToPacket [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:292:test_function_updateInfoToPacket:FAIL: Element 3 Expected 8 Was 6 [FAILED]
test/d_test_servo_rha/test_servo_rha.cpp:293:test_function_addReturnOptionToPacket [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:294:test_function_setTorqueOnOffToPacket [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:295:test_function_set_exitWheelModeToPacket [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:136:test_function_updateInfo:FAIL: Expected 50 Was 5. Speed [FAILED]
test/d_test_servo_rha/test_servo_rha.cpp:157:test_function_calculateTorque:FAIL: Expected 0 Was 101. Torque test 1 [FAILED]
test/d_test_servo_rha/test_servo_rha.cpp:196:test_function_addTorqueToPacket:FAIL: Element 3 Expected 150 Was 121 [FAILED]
test/d_test_servo_rha/test_servo_rha.cpp:299:test_function_addPingToPacket [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:300:test_function_setSpeedGoal [PASSED]
test/d_test_servo_rha/test_servo_rha.cpp:249:test_function_speedError:FAIL: Values Not Within Delta . Test 1 [FAILED]
-----
13 Tests 5 Failures 0 Ignored
```

Figura 6.15: Test satisfactorios ServoRHA

Fuente: Autor

6.4.3. Test librería JointRHA

6.4.4. Test librería JointHandler

Test para comprobar el correcto funcionamiento de los métodos encargados de gestionar la construcción en interpretación de los paquetes de datos. No hay comunicación real con los servos en esta serie de test.



```
===== [test::f_test_joint_handler_mock] Testing... (3/3) =====
If you don't see any output for the first 10 secs, please reset board (press reset button)

test/f_test_joint_handler_mock/test_joint_handler.cpp:212:test_function_warpSinglePacket [PASSED]
test/f_test_joint_handler_mock/test_joint_handler.cpp:213:test_function_addToSyncPacket [PASSED]
test/f_test_joint_handler_mock/test_joint_handler.cpp:214:test_function_warpSyncPacket [PASSED]
test/f_test_joint_handler_mock/test_joint_handler.cpp:215:test_function_checkConnection_mock [PASSED]
test/f_test_joint_handler_mock/test_joint_handler.cpp:216:test_function_sendSetWheelModeAll_mock [PASSED]
test/f_test_joint_handler_mock/test_joint_handler.cpp:217:test_function_controlLoop_oneJoint [PASSED]
-----
6 Tests 0 Failures 0 Ignored
```

Figura 6.16: Test satisfactorios JointHandler

Fuente: Autor

6.4.5. Test librería ChuckHandler

6.4.6. Test librería RobotRHA

Resumen de los test:



```
===== [TEST SUMMARY] =====
test:a_test_rha_types/env:megaatmega2560 [PASSED]
test:b_test_pid_regulator/env:megaatmega2560 [PASSED]
test:d_test_servo_rha/env:megaatmega2560 [PASSED]
test:e_test_joint_rha/env:megaatmega2560 [PASSED]
test:f_test_joint_handler_mock/env:megaatmega2560 [PASSED]
test:log/env:megaatmega2560 [IGNORED]
===== [PASSED] Took 52.65 seconds =====
```

Figura 6.17: Resumen Test satisfactorios

Fuente: Autor

6.5. Gestión de la complejidad y mantenibilidad:

Para controlar el desarrollo del proyecto de forma paralela en todas sus partes asegurando así un control de la complejidad y mantenibilidad se han ido controlando diferentes métricas referentes al software del proyecto.

Además de dichas métricas se han ido haciendo revisiones periódicas del cumplimiento las reglas de codificación en el software (ver Anexo C). Para ello se ha utilizado un *script* *cpplint* que automatiza la revisión del código.

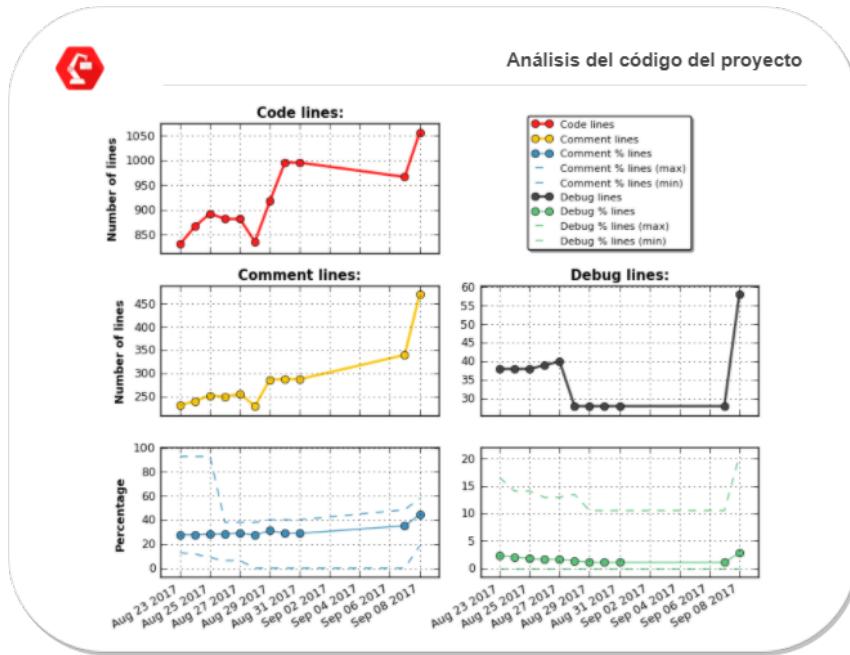
Para obtener la información referente a la complejidad y desarrollo del software se han utilizado dos herramientas (*lizard* y *Cloc*) junto con una serie de *scripts* que se han desarrollado para automatizar la obtención y visualización de la información. Las métricas que se han obtenido y valorado son:

1. Número de líneas de código.
2. Número de líneas de comentarios.
3. Número de líneas de mensajes de *debug*.
4. Porcentaje de líneas de comentarios (media de todos los ficheros así como máximos y mínimos).
5. Porcentaje de líneas de mensajes de *debug* (media de todos los ficheros así como máximos y mínimos).
6. Número de ficheros.
7. Número de funciones.
8. Media de métodos por fichero.
9. Complejidad ciclomática **COMENTAR QUE ES LA COMPLEJIDAD CICLOMÁTICA...; GLOSARIO O DIRECTAMENTE AQUÍ?** (media entre todos los métodos así como valores extremos).

Las métricas de la uno a la cinco de la lista anterior permiten controlar un desarrollo paralelo y equilibrado entre el código así como la documentación del mismo. De igual forma, aunque menos importante, permiten ver el desarrollo paralelo de métodos de *debugging*. Este se considera menos importante ya que en su mayoría se ha implementado, más que como metodología de desarrollo, cuando las pruebas del software lo requieren.

En la Figura 6.18 se puede una serie de gráficas donde se puede ver la relación entre el código, la documentación (comentarios) y el *debugging* (líneas de *debug*).

En la imagen se muestra el desarrollo temporal de dichos parámetros de forma que se puede apreciar un desarrollo paralelo tanto del código como de la documentación. Además, se puede apreciar como la relación entre documentación y el *debugging* con el total de líneas se mantiene relativamente constante gracias a las gráficas porcentuales. Esto concuerda con la metodología de desarrollo adoptada para el software del proyecto, y su control periódico a lo largo del tiempo ha permitido corregir desvíos en cualquiera de las partes.



Fuente: Autor

6.5.1. Reglas de codificación del Código

```

    Category 'readability/namespace' errors found: 8
    Category 'whitespace/braces' errors found: 17
    Category 'whitespace:semicolon' errors found: 3
    Category 'whitespace/tab' errors found: 4
    Category 'readability/alt_tokens' errors found: 2
    Category 'whitespace/blank_line' errors found: 30
    Category 'whitespace/newline' errors found: 18
    Category 'readability/todo' errors found: 9
    Category 'readability/braces' errors found: 10
    Category 'whitespace/comma' errors found: 37
    Category 'whitespace/operators' errors found: 8
    Category 'readability/casting' errors found: 15
    Category 'whitespace/parens' errors found: 31
    Category 'runtime/references' errors found: 4
    Category 'runtime/indentation_namespace' errors found: 7
    Category 'runtime/int' errors found: 9
    Category 'whitespace/comments' errors found: 86
    Category 'whitespace/indent' errors found: 20
    Category 'runtime/explicit' errors found: 4
    Total errors found: 322
  
```

Figura 6.19: Ejemplo de la Salida de cpplint con errores

Fuente: Autor



```

/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:37: Missing space after , [whitespace/comma] [3]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:77: Using deprecated casting style. Use static cast<float>(...) instead [readability/casting] [4]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:115: If an else has a brace on one side, it should have it on both [readability/braces] [5]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:115: Else clause should never be on same line as else (use 2 lines) [whitespace/newline] [4]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:124: Missing username in TODO; it should look like "-// TODO(my username): Stuff." [readability/todo] [2]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:131: Missing username in TODO; it should look like "-// TODO(my username): Stuff." [readability/todo] [2]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:132: Using C-style cast. Use static_cast<float>(...) instead [readability/casting] [4]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:133: Extra space after ( [whitespace/paren] [2]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:133: Extra space before ) [whitespace/paren] [2]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:152: If/else bodies with multiple statements require braces [readability/braces] [4]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:152: Using deprecated casting style. Use static cast<float>(...) instead [readability/casting] [4]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:248: Redundant blank line at the end of a code block should be deleted. [whitespace/blank_line] [3]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:307: Redundant blank line at the end of a code block should be deleted. [whitespace/blank_line] [3]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:366: Else clause should never be on same line as else (use 2 lines) [whitespace/newline] [4]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp:379: Else clause should never be on same line as else (use 2 lines) [whitespace/newline] [4]
Done processing /home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.h:119: Do not indent within a namespace [runtime/indentation_namespace] [4]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.h:120: At least two spaces is best between code and comments [whitespace/comments] [2]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.h:120: Should have a space between // and comment [whitespace/comments] [4]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.h:124: Namespace should be terminated with "// namespace ServoRHAConstants" [readability/namespace] [5]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.h:149: Single-parameter constructors should be marked explicit. [runtime/explicit] [5]
/home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.h:196: Redundant blank line at the end of a code block should be deleted. [whitespace/blank_line] [3]
Done processing /home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp

```

Figura 6.20: Ejemplo de errores con cpplint

Fuente: Autor


```

Done processing /home/quique/Documentos/RHA/SW/lib/chuck_handler/chuck_handler.h
Ignoring /home/quique/Documentos/RHA/SW/lib/cytron_g15_servo/cytron_g15_servo.cpp.txt; not a valid file name (cpp, h)
Done processing /home/quique/Documentos/RHA/SW/lib/cytron_g15_servo/cytron_g15_servo.cpp.txt
Ignoring /home/quique/Documentos/RHA/SW/lib/cytron_g15_servo/cytron_g15_servo.h.txt; not a valid file name (cpp, h)
Done processing /home/quique/Documentos/RHA/SW/lib/cytron_g15_servo/cytron_g15_servo.h.txt
Done processing /home/quique/Documentos/RHA/SW/lib/debug/debug.cpp
Done processing /home/quique/Documentos/RHA/SW/lib/debug/debug.h
Ignoring /home/quique/Documentos/RHA/SW/lib/debug/debug.h.gch; not a valid file name (cpp, h)
Done processing /home/quique/Documentos/RHA/SW/lib/debug/debug.h.gch
Done processing /home/quique/Documentos/RHA/SW/lib/joint_handler/joint_handler.cpp
Done processing /home/quique/Documentos/RHA/SW/lib/joint_handler/joint_handler.h
Ignoring /home/quique/Documentos/RHA/SW/lib/joint_handler/joint_handler.h.gch; not a valid file name (cpp, h)
Done processing /home/quique/Documentos/RHA/SW/lib/joint_handler/joint_handler.h.gch
Done processing /home/quique/Documentos/RHA/SW/lib/joint_rha/joint_rha.cpp
Done processing /home/quique/Documentos/RHA/SW/lib/joint_rha/joint_rha.h
Ignoring /home/quique/Documentos/RHA/SW/lib/joint_rha/joint_rha.h.gch; not a valid file name (cpp, h)
Done processing /home/quique/Documentos/RHA/SW/lib/joint_rha/joint_rha.h.gch
Done processing /home/quique/Documentos/RHA/SW/lib/rha_types/fuzzy_regulator.h
Done processing /home/quique/Documentos/RHA/SW/lib/rha_types/pid_regulator.h
Done processing /home/quique/Documentos/RHA/SW/lib/rha_types/rha_types.h
Ignoring /home/quique/Documentos/RHA/SW/lib/rha_types/rha_types.h.gch; not a valid file name (cpp, h)
Done processing /home/quique/Documentos/RHA/SW/lib/rha_types/rha_types.h.gch
Done processing /home/quique/Documentos/RHA/SW/lib/robot_rha/robot_rha.cpp
Done processing /home/quique/Documentos/RHA/SW/lib/robot_rha/robot_rha.h
Ignoring /home/quique/Documentos/RHA/SW/lib/robot_rha/robot_rha.h; not a valid file name (cpp, h)
Done processing /home/quique/Documentos/RHA/SW/lib/robot_rha/robot_rha.h
Ignoring /home/quique/Documentos/RHA/SW/lib/robot_rha/robot_rha.h.gch; not a valid file name (cpp, h)
Done processing /home/quique/Documentos/RHA/SW/lib/robot_rha/robot_rha.h.gch
Done processing /home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.cpp
Done processing /home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.h
Ignoring /home/quique/Documentos/RHA/SW/lib/servo_rha/servo_rha.h.gch; not a valid file name (cpp, h)
Done processing /home/quique/Documentos/RHA/SW/lib/utilities/utilities.cpp.old; not a valid file name (cpp, h)
Done processing /home/quique/Documentos/RHA/SW/lib/utilities/utilities.cpp.old
Done processing /home/quique/Documentos/RHA/SW/lib/utilities/utilities.h
Done processing /home/quique/Documentos/RHA/SW/src/main.cpp
Ignoring /home/quique/Documentos/RHA/SW/src/main_utilities.cpp; not a valid file name (cpp, h)
Done processing /home/quique/Documentos/RHA/SW/src/main_utilities.cpp_
Done processing /home/quique/Documentos/RHA/SW/test/a_test_rha_types/test_rha_types.cpp
Done processing /home/quique/Documentos/RHA/SW/test/b_test_pid_regulator/test_pid_regulator.cpp
Done processing /home/quique/Documentos/RHA/SW/test/c_test_fuzzy_regulator/test_fuzzy_regulator.cpp
Done processing /home/quique/Documentos/RHA/SW/test/d_test_servo_rha/test_servo_rha.cpp
Done processing /home/quique/Documentos/RHA/SW/test/e_test_joint_rha/test_joint_rha.cpp
Done processing /home/quique/Documentos/RHA/SW/test/f_test_joint_handler_mock/test_joint_handler.cpp
Total errors found: 0

```

Figura 6.21: Salida de cpplint una vez eliminados los errores

Fuente: Autor



Capítulo 7

Estudio Cinemático



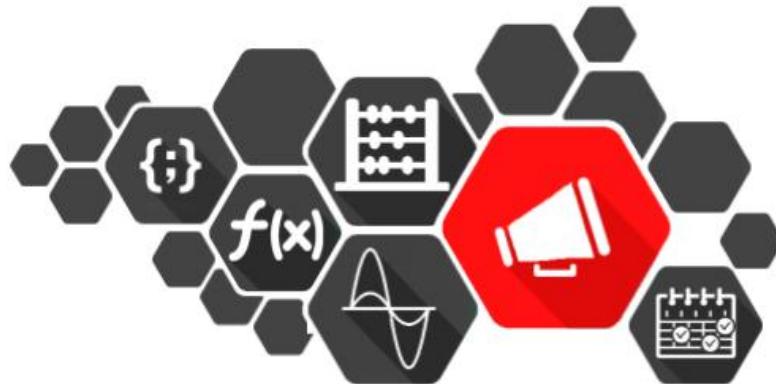
Capítulo 8

Control

En este capítulo se detalla el trabajo realizado dentro del ámbito de la ingeniería de control aplicada a este proyecto.

8.1. Control de velocidad para los servos G15 cube

Los *servos* de Cytron aún teniendo diferentes modos de control no poseen un control efectivo de la velocidad de rotación cuando se utilizan en el modo de giro continuo o *wheel mode*, parte importante del esfuerzo destinado al control del brazo se ha destinado a conseguir implementar un control de velocidad funcional para las articulaciones dos y tres.



Capítulo 9

Resultados y discusión

En este capítulo...

9.1. Resultados

9.2. Discusión



Capítulo 10

Gestión del proyecto

En este capítulo se describe la gestión del proyecto: ciclo de vida, planificación, presupuesto, etc.

10.1. Ciclo de vida

Explicación de las fases del proyecto: definición, análisis, diseño, construcción, pruebas, implementación, validación, documentación. Ejemplo: diagrama de Pert.

10.2. Planificación

Se puede indicar mediante un diagrama de Gantt.

10.2.1. Planificación inicial

10.2.2. Planificación final

10.3. Presupuesto

Coste de los materiales en la Tabla 10.1:

– COMPLETAR –

Tabla 10.1: Costes del proyecto

Artículo	Coste Unitario ¹	Nº de unidades	Total
Arduino Uno	17.00 € ²	1	17.00 €
Cytron G15 Cube Servo	23.23 € ³	3	total
Cytron G15 Shield	6.64 € ⁴	1	6.64 €
Potenciómetro Serie TW	9.29 € ⁵	2	18.58€
Barras Aluminio sección cuadrada	5,626 € ⁶	5	28,13 €
Rodamiento 13x4	1.83€ ⁷		€
Rodamiento 10x3	2.02€ ⁸		€
Hilo Kevlar	23.23 € ⁹	1	23.23 €
GM Series Plastic Wheel	2.70 € ¹⁰	1	2.70 €

¹En los casos en qué ha sido necesario se ha aplicado el cambio a Euros oficial propuesto por el Banco de España en el día en que se han consultado los precios: <https://www.bde.es/bde/es/>

²Precios consultados en la página oficial de Arduino a 07 de Septiembre 2017 (<https://store.arduino.cc/arduino-uno-rev3>).

³Precios consultados en la página oficial de Cytron a 07 de Septiembre 2017 (<http://www.cytron.com.my/p-g15>).

⁴Precio consultados en la página oficial de Cytron a 07 de Septiembre 2017 (<https://www.cytron.com.my/p-shield-g15>).

⁵Precio consultados en la página oficial de RS a 07 de Septiembre 2017 (<http://uk.rs-online.com/web/p/potentiometers/5028586/>)

⁶Precio consultados en la página oficial de RS a 07 de Septiembre 2017 (<http://es.rs-online.com/web/p/tubos-de-aluminio/3047894/>)

⁷Precio consultados en la página oficial de RS a 20 de Enero 2018 (<https://es.rs-online.com/web/p/rodamientos-de-bola/6189890/>)

⁸Precio consultados en la página oficial de RS a 20 de Enero 2018 (<https://es.rs-online.com/web/p/rodamientos-de-bola/6189856/>)

⁹Precio consultados en la página de compra a 07 de Septiembre 2017 (http://www.emmakites.com/index.php?main_page=product_info&cPath=336_365&products_id=1199)

¹⁰Precio consultados en la página oficial de Solarbotics a 11 de Septiembre 2017 (<https://solarbotics.com/product/gmpw/>)

Poleas de acetal - <https://es.rs-online.com/web/p/poleas-trapezoidales-con-perfil-en-v-de-transmision-mecanica/3520636/> Soporte de sombrilla

10.3.1. Personal

10.3.2. Material

10.3.3. Resumen de costes



Capítulo 11

Conclusiones

Se presentan a continuación las conclusiones...

11.1. Conclusión

Una vez finalizado el proyecto...

11.2. Desarrollos futuros

Un posible desarrollo...

Apéndice A

Listado de piezas diseñadas

- COMPLETAR -

En este anexo se presenta una tabla listando las piezas que se han diseñado y fabricado para el proyecto. En la tabla A1 se puede ver una miniatura de la pieza en cuestión, la cantidad necesaria de cada tipo, una estimación del peso (material consumido en su fabricación) así como una breve descripción de la pieza y/o proceso de fabricación de la misma. A su vez llevan asociado una referencia alfanumérica que se corresponde con los ficheros en formato digital entregados, asignada de la siguiente forma:

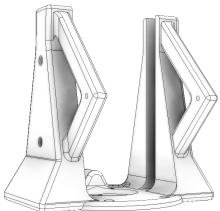
RHA + ubicación + número_de_pieza

- *ubicación*: A\$ (articulación), B\$ (barra). La letra va acompañada de un número (sustituyendo al carácter \$) que variará dependiendo de donde se encuentre la pieza en el montaje. A1 - articulación uno, B2 - barra 2 y así sucesivamente.
- *número_de_pieza*: valor numérico que diferencia las piezas en la misma ubicación.

Nota: En caso de que la pieza se utilice en varias partes diferenciadas la referencia se tomará para la primera vez que aparece la pieza en orden ascendente (desde la barra 0 en adelante e igual desde la articulación 1).

Nota 2: la miniatura de las piezas no sigue ninguna escala concreta. Es una representación que permite un reconocimiento visual de la pieza, aunque no de su tamaño.

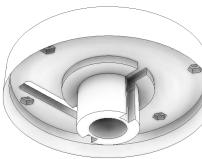
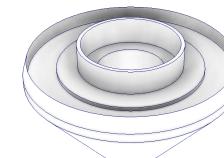
Tabla A.1: Listado de piezas diseñadas de fabricación propia

Num	Esquema Pieza	Referencia	Cantidad	Descripción	Peso Estimado ¹
1		RHAB1001	1	Es la base sobre la que se colocan los servos de los tres primeros grados de libertad. También fija la rueda para efectuar el giro en Z	CORREGIR CON PESO AL 90% 193g
2		RHAB1002	1	blah	24g
3		RHAB1003	1	blah	24g

¹ El peso estimado se obtiene con el programa Cura — **COMPLETAR** — aplicando los parámetros de la tabla A.2. Este peso incluye el de los soportes necesarios para su impresión. Excepto en los casos donde se especifique lo contrario el juego de Parámetros utilizados es el 1.

Continua en la página siguiente

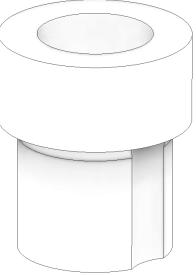
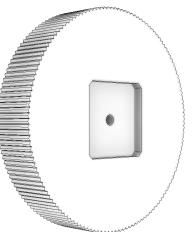
Tabla A.1 – Continuación de la página anterior

Num	Esquema Pieza	Referencia	Cantidad	Descripción	Peso Estimado ¹
4		RHAB1004	2	blah	3g
5		RHAA1005	1	PESO AL 90 %	54g
6		RHAA1006	1	blah	128g

¹ El peso estimado se obtiene con el programa Cura – **COMPLETAR** – aplicando los parámetros de la tabla A.2. Este peso incluye el de los soportes necesarios para su impresión. Excepto en los casos donde se especifique lo contrario el juego de Parámetros utilizados es el 1.

Continua en la página siguiente

Tabla A.1 – Continuación de la página anterior

Num	Esquema Pieza	Referencia	Cantidad	Descripción	Peso Estimado ¹
7		RHAB0001	1	blah	blah
8		RHAB0002	1	blah	blah
10		RHAA1001	1	blah	18g

¹ El peso estimado se obtiene con el programa Cura – **COMPLETAR** – aplicando los parámetros de la tabla A.2. Este peso incluye el de los soportes necesarios para su impresión. Excepto en los casos donde se especifique lo contrario el juego de Parámetros utilizados es el 1.

Continua en la página siguiente

Tabla A.1 – Continuación de la página anterior

Num	Esquema Pieza	Referencia	Cantidad	Descripción	Peso Estimado ¹
19		RHAB1007	2	blah	1g
19		RHAB1008	2	blah	20g
9		RHAB1009	1	blah	10g

¹ El peso estimado se obtiene con el programa Cura – **COMPLETAR** – aplicando los parámetros de la tabla A.2. Este peso incluye el de los soportes necesarios para su impresión. Excepto en los casos donde se especifique lo contrario el juego de Parámetros utilizados es el 1.

Continua en la página siguiente

Tabla A.1 – Continuación de la página anterior

Num	Esquema Pieza	Referencia	Cantidad	Descripción	Peso Estimado ¹
11		RHAA2001	1	blah	blah
12		RHAA2002	1	blah	blah
13		RHAA2003	1	blah	8g

¹ El peso estimado se obtiene con el programa Cura – **COMPLETAR** – aplicando los parámetros de la tabla A.2. Este peso incluye el de los soportes necesarios para su impresión. Excepto en los casos donde se especifique lo contrario el juego de Parámetros utilizados es el 1.

Continua en la página siguiente

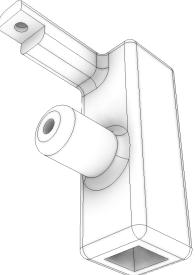
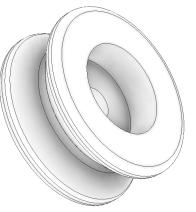
Tabla A.1 – Continuación de la página anterior

Num	Esquema Pieza	Referencia	Cantidad	Descripción	Peso Estimado ¹
14		RHAA2004	1	blah	blah
15		RHAA2005	1	blah	blah
16		RHAA2006	1	blah	blah

¹ El peso estimado se obtiene con el programa Cura – **COMPLETAR** – aplicando los parámetros de la tabla A.2. Este peso incluye el de los soportes necesarios para su impresión. Excepto en los casos donde se especifique lo contrario el juego de Parámetros utilizados es el 1.

Continua en la página siguiente

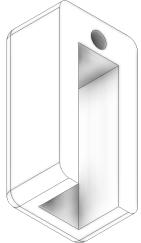
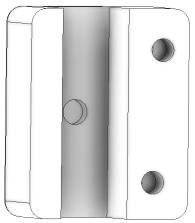
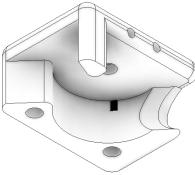
Tabla A.1 – Continuación de la página anterior

Num	Esquema Pieza	Referencia	Cantidad	Descripción	Peso Estimado ¹
17		RHAB2001	1	blah	blah
18		RHAB2002	3	blah	blah
19		RHAB2003	2	blah	6g

¹ El peso estimado se obtiene con el programa Cura – **COMPLETAR** – aplicando los parámetros de la tabla A.2. Este peso incluye el de los soportes necesarios para su impresión. Excepto en los casos donde se especifique lo contrario el juego de Parámetros utilizados es el 1.

Continua en la página siguiente

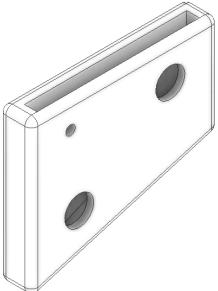
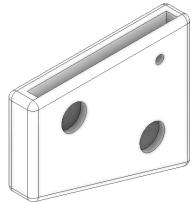
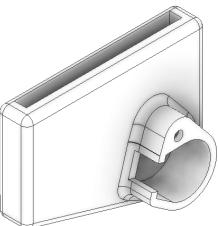
Tabla A.1 – Continuación de la página anterior

Num	Esquema Pieza	Referencia	Cantidad	Descripción	Peso Estimado ¹
20		RHAB2004	2	blah	blah
21		RHAB3001	1	blah	blah
22		RHAB3002	1	blah	blah

¹ El peso estimado se obtiene con el programa Cura – **COMPLETAR** – aplicando los parámetros de la tabla A.2. Este peso incluye el de los soportes necesarios para su impresión. Excepto en los casos donde se especifique lo contrario el juego de Parámetros utilizados es el 1.

Continua en la página siguiente

Tabla A.1 – Continuación de la página anterior

Num	Esquema Pieza	Referencia	Cantidad	Descripción	Peso Estimado ¹
23		RHAA3001	2	blah	39g
24		RHAA3002	1	blah	39g
25		RHAA3003	1	blah	44g

¹ El peso estimado se obtiene con el programa Cura – **COMPLETAR** – aplicando los parámetros de la tabla A.2. Este peso incluye el de los soportes necesarios para su impresión. Excepto en los casos donde se especifique lo contrario el juego de Parámetros utilizados es el 1.

Continua en la página siguiente

Tabla A.1 – Continuación de la página anterior

Num	Esquema Pieza	Referencia	Cantidad	Descripción	Peso Estimado ¹
26		RHAA3004	1	1	2g
27		RHAA3005	2	blah	-
28		RHAA3006	4	blah	9g

¹ El peso estimado se obtiene con el programa Cura – **COMPLETAR** – aplicando los parámetros de la tabla A.2. Este peso incluye el de los soportes necesarios para su impresión. Excepto en los casos donde se especifique lo contrario el juego de Parámetros utilizados es el 1.

Continua en la página siguiente

Tabla A.1 – Continuación de la página anterior

Num	Esquema Pieza	Referencia	Cantidad	Descripción	Peso Estimado ¹
29		RHAA3007	1	blah	24g
30		RHAA3008	1	blah	blah
31	PIEZAS DEL EXTREMO PROVISIONALES	blah	blah	blah	blah

Se han utilizado diferentes juegos de parámetros según las piezas, se pueden ver los parámetros principales que influyen en el peso final de la pieza en la tabla A.2 a continuación:

Tabla A.2: Parámetros de las piezas para la estimación de peso

	Parámetros 1	Parámetros 2
Grosor capa inferior y superior	1.2 mm	1.2mm
Grosor de pared	1.2 mm	1.2mm
Densidad interna	30 %	90 %

Apéndice B

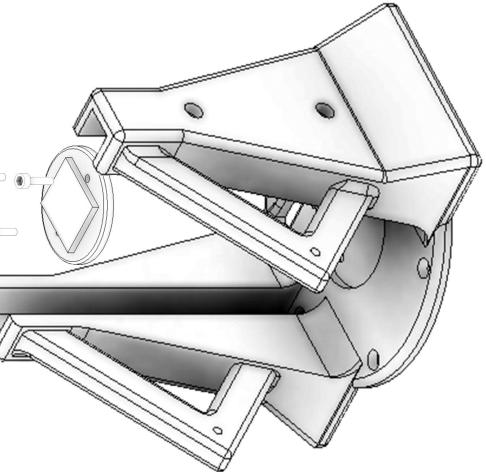
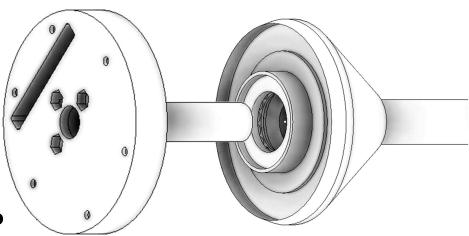
Montaje del prototipo

ASEGURAR QUE LA ORIENTACION AL IMPRIMIR SEA CORRECTA. ¿O LO DEJO UNA DIPOSITIVA POR HOJA Y LISTO?

Al unir los pasos 2 y 3, si las tuercas no han quedado encajadas por exceso de holgura, se pueden fijar con cualquier tipo de adhesivo.

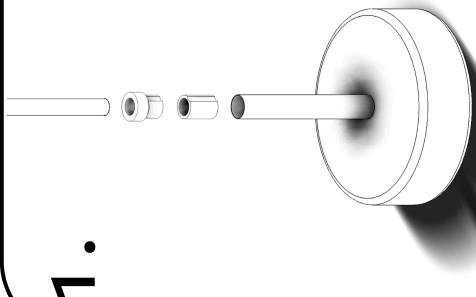
También se podrán sujetar a mano cuando en el momento de ensamblar, sobre dicha estructura, el montaje del paso 5.

4.

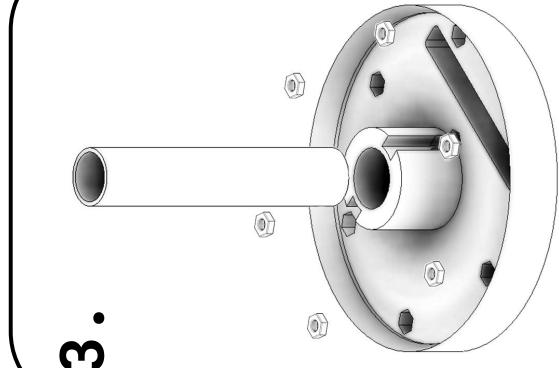


5.

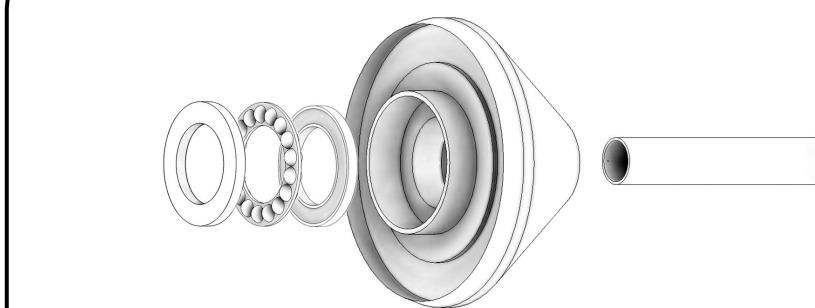
1.



3.



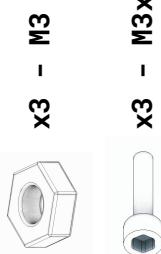
2.



El paso 3 requiere
para su posterior
ensamblaje:



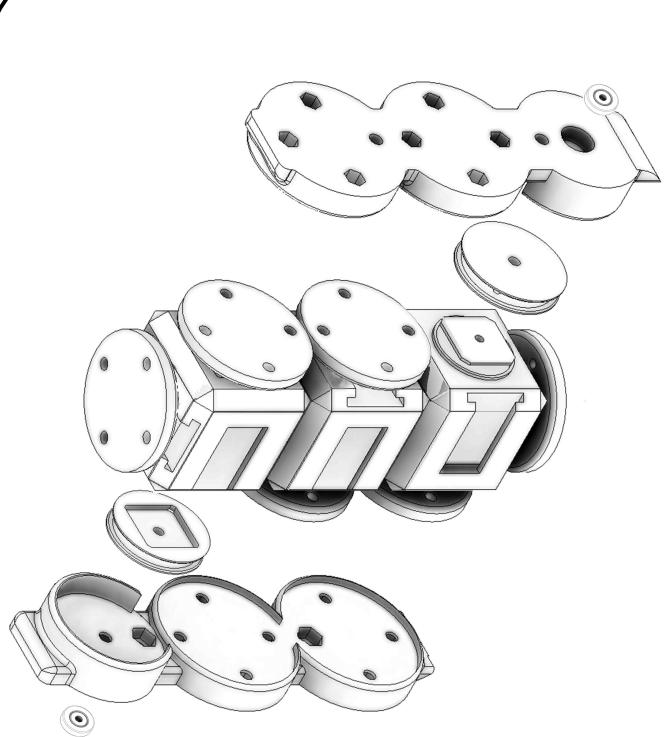
El paso 5 requiere
para fijar el
soporte de los
motores:



6.

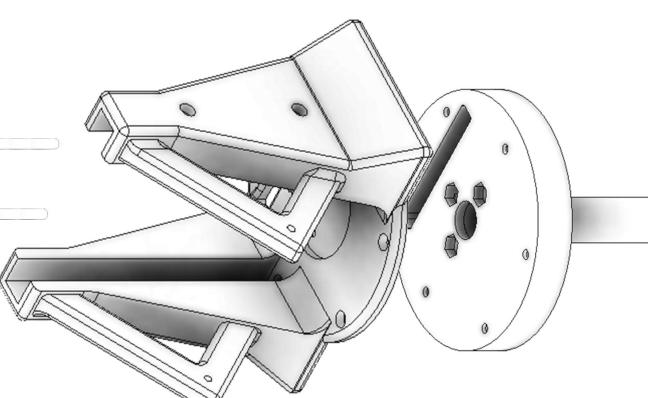
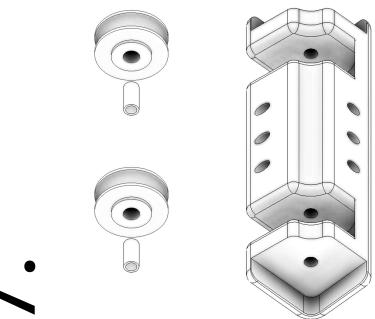
7.

8.



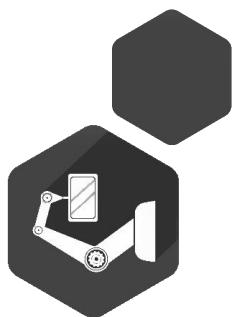
Las poleas llevan un adaptador para reducir el diámetro interior de las mismas para evitar movimiento respecto al eje utilizado.

Esta pieza se reserva para ser ensamblada posteriormente.



E1 paso 6 requiere:

x6 - M4x20



E1 paso 8 requiere:

x12 - M3 - M3x10

x2 - 3x10



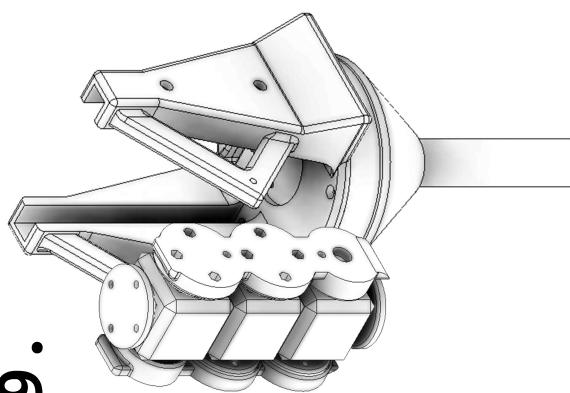
Es recomendable que se ate el hilo de kevlar a las poleas antes de taparlas.



x4 - M4

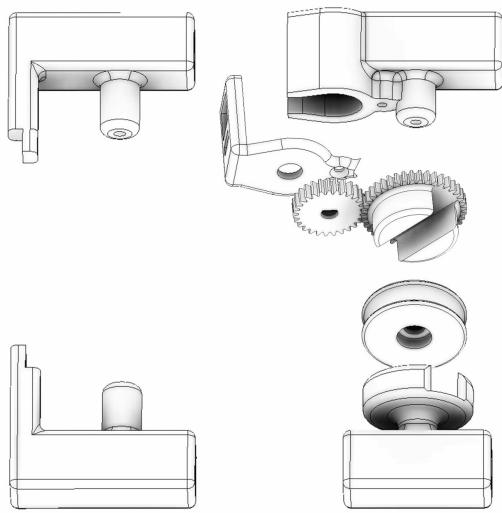


- 9.**
- Paso 9: El motor inferior se encaja sobre el soporte que se ha fijado en el paso 5 sobre la base.

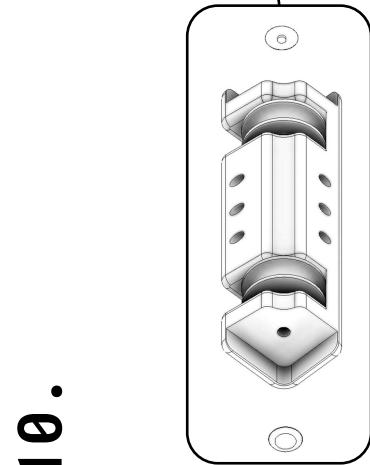
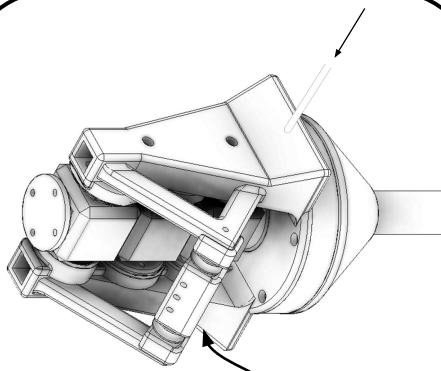


Paso 10: se recupera el montaje del paso 7 para encajarlo con el eje (acero 3mm de diámetro) en la pieza montada en el paso 9. Opcionalmente se pueden añadir las tapas para bloquear las salidas del eje.

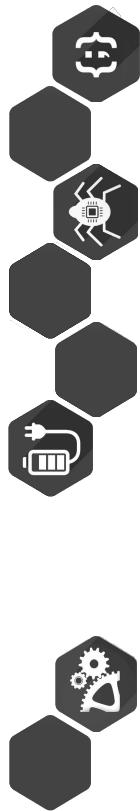
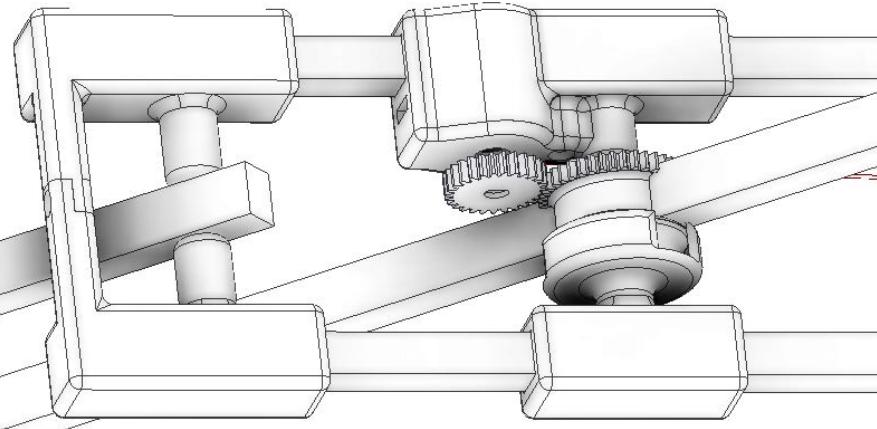
11.



10.



11. Montaje final

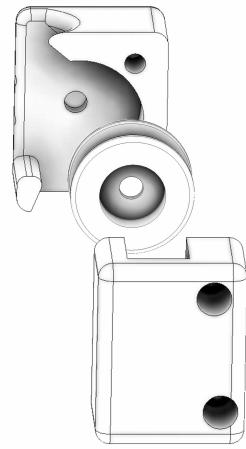
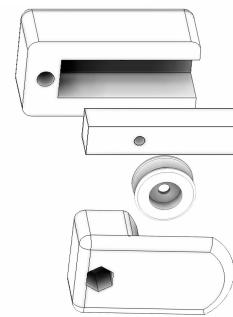


El paso 11 requerirá:

- x2 Ejes de acero de 4mm de diámetro
- x1 - M4x12 Los ejes atraviesan todas las piezas fijando las mismas a las barras de aluminio, que tienen agujeros pasantes donde quedan encajados los ejes y sobre los que giran.
- x1 - M4x10
- x1 - M4
- x2 - 4x13

- Potenciómetro: habrá que insertarlo en el hueco reservado

12.

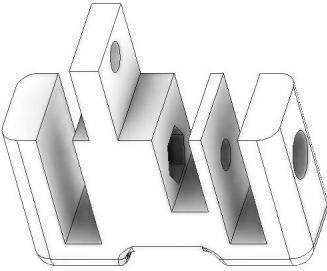


Nota: Una vez encajadas las piezas expuestas en el paso 13, se fijan encastrados en los huecos preparados para ello. Serán necesarios:

x8 - 4x13



Nota:



13.



El paso 12 requerirá:

- x2 Ejes de acero de 4mm de diámetro

x3 - M4x10mm



x4 - 4x13 (dos en cada polea)



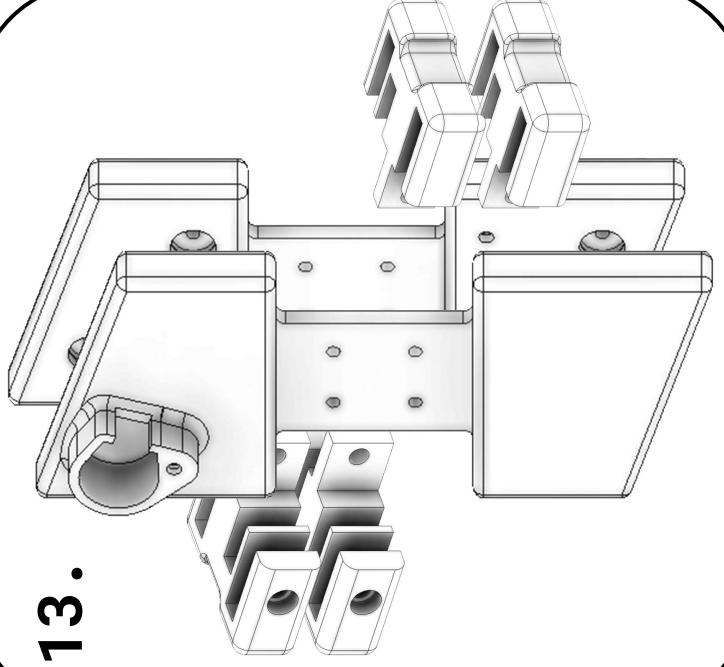
Ambos montajes abrazan una barra de aluminio. En el caso de la derecha la pieza redirige el cableado; el caso de la izquierda reduce el par soportado por el servo.

El montaje representado en el lado izquierdo se debe ensamblar por duplicado ya que se utiliza para las articulaciones dos y tres.



Para el siguiente paso
será necesario encajar
tuercas en cada pieza para
fijarlas posteriormente.

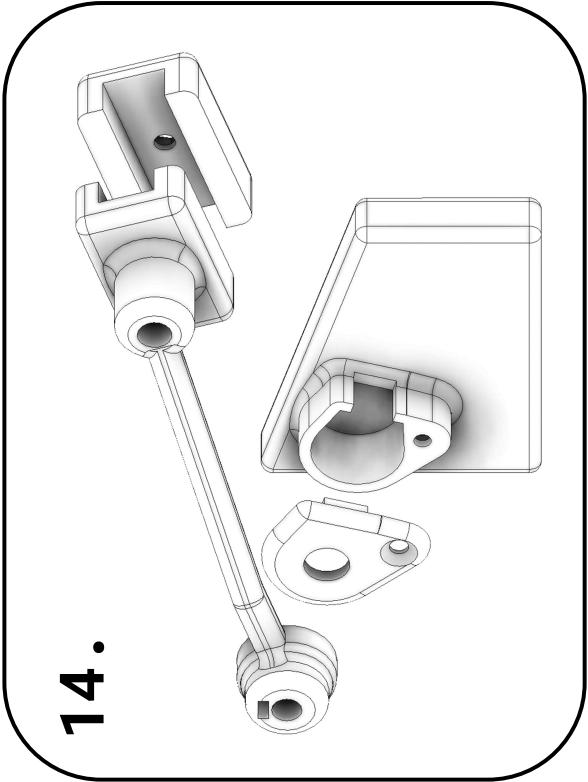
13.



Notas: En cada pareja de rodamientos queda encerrado un eje de acero de 4mm de diámetro, que atraviesa las barras de aluminio. Es recomendable añadir arandelas a cada lado de la barra. Habrá que añadir un quinto eje en el par de agujeros que corresponde, este será el punto de enganche del muelle. Una vez encajadas las piezas se solapan las pinzas dos a dos y se fijan con un tornillo a cada lado. Será necesario para este ensamblaje:

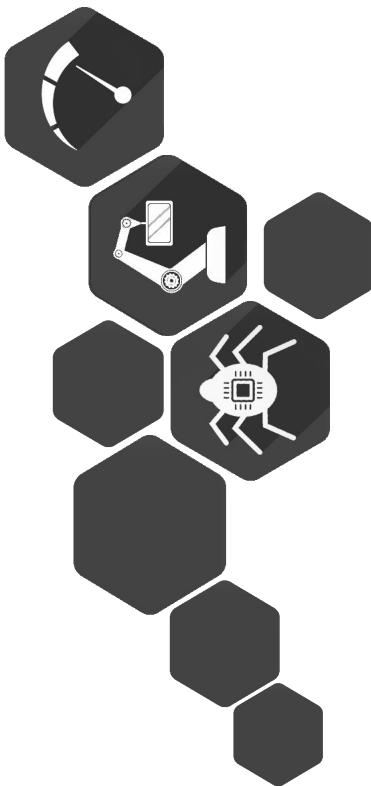
 x4 - M4
 x4 - M4x20

14.



Nota: Se debe introducir el potenciómetro en el hueco correspondiente. El acople del potenciómetro abraza una de las barras de aluminio para transmitir el giro de la articulación al potenciómetro.

 x1 - M4x12
 x2 - M4
 x1 - M4x20



Apéndice C

Reglas de codificación del Software

Las reglas de codificación aplicadas al software del proyecto se han obtenido, por utilizar una referencia, de las reglas aplicadas por Google en sus proyectos libres. Esta guía está ampliamente documentada e incluye su propia herramienta para comprobar su correcta aplicación – **COMPLETAR** – lo que facilita la revisión del código así como corrección de desviaciones de estilo.

En este anexo se traducen y resumen los aspectos más importantes de dicha guía. En algunos casos se han adaptado las reglas al caso concreto de este proyecto.

Establecer unas reglas de codificación, unificando un estilo en la notación y uso de la sintaxis, es interesante de cara a posibilitar una mayor facilidad de lectura en futuros desarrollos aumentando así la mantenibilidad del código.

Estas reglas aplican al código C++ del proyecto, no a los *scripts* auxiliares.

C.1. Aspectos generales

C.2. Ficheros de cabecera

En general todos los ficheros con extensión .cpp correspondientes a las librerías deberán ir acompañados del fichero de cabecera .h correspondiente.

Quedan exentos de cumplir esta regla los ficheros correspondientes a Test (unitarios, de integración, etc) así como ficheros que contengan únicamente una función main().

C.2.1. Inclusión Múltiple

Para evitar problemas de inclusión múltiple todos los ficheros de cabecera con extensión .h deberán incluir guardas con el siguiente formato y escrito en mayúsculas: <nombre_del_fichero>_<extensión>.

Ejemplo :

```
#ifndef SERVO_RHA_H  
#define SERVO_RHA_H
```

```

...
...
...
#endif
```

C.2.2. Orden de inclusión de ficheros

Para evitar problemas en las dependencias de las distintas librerías se incluirán las mismas dejando para el final las librerías propias del proyecto e incluyendo el resto de la más general a la más particular. **REVISAR EL CÓDIGO!**

Ejemplo orden al incluir cabeceras:

```
#include <stdint.h>      // lib estandar de c++
#include <Arduino.h>      // lib de Arduino
#include <SoftwareSerial.h>    // lib para controlar el puerto serie. Basado
en Arduino

#include "debug.h"        // def y control de las funciones de debug
#include "rha_types.h"    // tipos de datos
#include "joint_rha.h"    // clase a incluir
```

¿DEF COMO ABREVIATURA?

Se deben incluir todos los ficheros que definan los símbolos utilizados en el fichero sobre el que se incluyen. Las declaraciones anticipadas de objetos no están permitidas salvo excepciones justificadas.

C.3. Ámbitos

C.3.1. Espacios de nombres

Como norma general las constantes, variables o funciones que no estén contenidas en ningún objeto se incluirán dentro de un espacio de nombres o *namespace* que haga referencia a la utilidad de las mismas.

No está permitido usar directivas del tipo `<using namespace ____;>`.

Los espacios de nombres se escriben con la primera letra de cada palabra, en caso de haber más de una, en mayúscula y sin separación de ningún tipo.

Ejemplo: Constantes referentes al test de comportamiento ante una entrada tipo rampa

```
namespace SlopeTest {
#define SAMPLE_SLOPE 110
#define SAMPLE_TEST_SLOPE 20
#define SLOPE_SPEED 0.1
}
```

Prohibido el uso de:

```
using namespace StepTest;
```

¿DEF COMO ABREVIATURA?

C.3.2. Variables Locales

Las variables se definirán preferiblemente en el ámbito más bajo **¡SE DICE ASÍ?** en que se vayan a utilizar. Preferiblemente la inicialización de las mismas se hará junto a la declaración.

Se pueden dar excepciones, como pueden ser vectores sobre los que se iterará dentro de un bucle u otros casos similares. En estos casos se de declarará el objeto fuera del propio ámbito para evitar recursivas llamadas a constructor y destructor de los mismos.

Ejemplo:

```
// Siempre que la variable sobre la que se itera no se vaya a utilizar para
// posteriores operaciones:
for(int i = 0, i < 10; i++) {
}

//mejor que el caso siguiente, que adicionalmente incumple la regla preferente
//de inicializar la variable cuando se declara:
int i;
for(i = 0, i < 10; i++) {

}

//quedá permitido declarar vectores u otros objetos similares antes si se va a
//iterar o trabajar sobre los mismos
int vector[5] = {1,2,3,4,5};
for(int i = 0, i < 10; i++) {
    Serial.print(vector[i]);
}
```

¡DEF COMO ABREVIATURA?

C.4. Clases

C.4.1. Constructores y métodos de Inicialización

Para todos los objetos debe haber constructores por defecto sin parámetros de entrada. Aunque se pueden añadir constructores que inicialicen los diferentes parámetros será obligatorio generar métodos que los inicialicen una vez construido el objeto. Arduino, aún estando basado en el lenguaje C++ no permite un uso completo de memoria dinámica. Es especialmente importante para declarar estos objetos como miembros de otros. **NO SE SI ESTÁ MUY CLARO** :—

Ejemplo:

```
//NO se permite:
- joint_rha.h -
ServoRHA servo_.*;
- joint_rha.cpp -
servo_ = new ServoRHA(1, 10, 5);

//Se llama al constructor del objeto para luego inicializarlo:
- joint_rha.h -
ServoRHA servo_;
- joint_rha.cpp -
servo.init(... params ...);
```

Para evitar funciones con muchos parámetros que reduzcan la legibilidad del código se permite generar diferentes inicializadores para los distintos parámetros. En la documentación del objeto deberá quedar bien claro que inicializadores deben invocarse para el correcto funcionamiento del mismo.

C.4.2. Estructuras o Clases

Por norma general las estructuras se utilizarán exclusivamente para objetos pasivos, objetos que contienen información. Todo lo demás se codificará dentro una clase.

En el caso de estructuras se permiten únicamente métodos para el manejo de los datos sin añadir ninguno tipo de comportamiento, están permitidos los constructores, destructores, métodos de reset, validación, etc. El acceso a los miembros de la estructura se hará directamente sobre los propios parámetros y no mediante métodos específicos. Los parámetros serán siempre públicos para ser consistente con este punto.

Para mayores funcionalidades se generará una clase.

C.4.3. Control de Acceso

Como norma general se declararan como privados todos los atributos de las clases exceptuando aquellos objetos que a su vez tengan, internamente, control de acceso definido (otras clases). De cara a generar Test con clases propias se permite la declaración de atributos como `protected`.

C.5. Tipos de datos

Los tipos de datos usados irán acordes con la librería `stdint.h`. Estos son del tipo `int16_t`, `uint32_t`, etc. Este tipo de datos garantiza el control del tamaño del dato declarado.

Se utilizarán los nombres `float` y `double` convencionales para declarar datos en coma flotante.

C.6. Nombres

C.6.1. Reglas generales

Los nombres deberán ser descriptivos. Por norma general no se utilizarán abreviaciones que no estén comúnmente aceptadas.

C.6.2. Nombre de los ficheros

Los nombres de los ficheros de código C++ se nombran en minúsculas separando, en caso de haber varias palabras, con un guión bajo. Los ficheros correspondientes a los test llevarán, precediendo al nombre la palabra "test".

Ejemplos :

```
joint_handler.h
joint_rha.cpp
test_servo_rha.cpp
```

C.6.3. Nombre de los directorios

Los ficheros de código irán contenidos en diferentes directorios para cada librería o conjunto de test. Estos directorios llevarán el nombre de la librería que contienen, en el mismo formato que la misma, en este caso sin extensión. Los test se ejecutan en el orden en que se ordenan los directorios. En este caso se añadirá un carácter para ordenar los mismos de manera adecuada.

Están exentos de esta regla los ficheros principales (que contienen la función `main()`, ó `setup()` y `loop()` en caso de ser ficheros con extensión `.ino`).

Ejemplos :

```
/lib/
  joint_handler/
  joint_rha/
/test/
  a_test_servo_rha/
  b_test_joint_rha/
```

C.6.4. Nombres para objetos

Los nombres llevarán mayúscula al comienzo así como al inicio de cada palabra, sin guion bajo como separación.

Ejemplo :

```
class ServoRHA{ ... };
class JointHandler{ ... };
struct SpeedGoal { ... };
```

C.6.5. Nombres de variables

Por norma general las variables se nombrarán en minúsculas, separando, cuando fuera necesario, las diferentes palabras mediante un guión bajo.

C.6.6. Nombres de atributos de clases

La norma para nombrar atributos de clases será igual que en el caso general acabando, en este caso, con un guión bajo.

Ejemplo :

```
class Regulator {
  float kp_, ki_, kd_;
  float ierror_[INTEGER_INTERVAL];
  uint8_t index_;
```

<pre> ... } ;</pre>

C.6.7. Nombres de miembros de estructuras

Las variables miembro de estructuras serán nombradas de igual forma que en el caso general.

<p>Ejemplo:</p>

```
struct SpeedGoal {
    uint8_t servo_id;
    int16_t speed;
    int16_t speed_slope;
    uint8_t direction;
} ;
```

C.6.8. Nombres de funciones

Las funciones comenzarán en minúscula marcando con mayúscula cada nueva palabra que aparezca. Los acrónimos irán en mayúscula. Esta regla afecta a métodos de clases a de igual manera a excepción de constructores y destructores.

<p>Ejemplo:</p>

```
class ServoRHA {
    ...
public:
    ServoRHA() { time_last_error_ = 0; time_last_ = 0; last_error_ = 0;
                  error_ = 0; derror_ = 0; ierror_ = 0; }
    ServoRHA(uint8_t servo_id);
    void init(uint8_t servo_id);
    void addUpadteInfoToPacket(uint8_t *buffer);
    bool addTorqueToPacket(uint8_t *buffer);
    void setTorqueOnOffToPacket(uint8_t *buffer, uint8_t onOff);
} ;
```

C.6.9. Nombres de parámetros funciones

Los parámetros de métodos y funciones se nombran siguiendo el caso general para nombrar variables.

C.6.10. Espacios de nombres

Como se ha visto en la sección C.3 los espacios de nombres se definen de manera equivalente a las clases.

C.6.11. Nombres de enumeraciones

En el caso de enumeraciones se seguirá la misma norma que para las clases y espacios de nombres. En este caso cabe la excepción de poder ser declaradas sin nombre.

C.6.12. Nombres de macros

Todo nombre precedido por una instrucción `#define` se nombrará en mayúsculas, separando las palabras, si las hubiera, mediante el uso del guión bajo. Esto aplica tanto a macros como constantes.

C.7. Comentarios

Es necesario el uso de comentarios para documentar el código y aumentar la legibilidad del mismo. En este caso se seguirá el estilo utilizado por *doxygen*, que será la herramienta utilizada para, posteriormente generar la documentación.

C.7.1. Comentarios de ficheros

Todos los ficheros deberán llevar comentarios en su cabecera. Estos comentarios tendrán el siguiente aspecto:

Ejemplo:

```
/**  
 * @file  
 * @brief Implements ServoRHA class. This object inherits from CytronG15Servo  
 * object to enhance its capabilities  
 * @Author: Enrique Heredia Aguado <enheragu>  
 * @Date: 2017-Sep-08  
 * @Project: RHA  
 * @Filename: servo_rha.h  
 * @Last modified by: quique  
 * @Last modified time: 30-Sep-2017  
 */
```

C.7.2. Comentarios de Clases

TDB!

C.7.3. Comentarios de funciones

Todas las funciones y métodos deberán llevar un comentario describiendo su funcionamiento así como los parámetros de entrada y salida. Estos comentarios tendrán el siguiente aspecto y se situarán encima de la definición de la función o método:

Ejemplo:

```
/** @brief Saves in buffer the package return level of servo (error information  
for each command sent)  
* @method ServoRHA::addReturnOptionToPacket  
* @param { uint8_t* } buffer array in which add the information  
* @param { uint8_t } option RETURN_PACKET_ALL -> servo returns packet for all  
commands sent; RETURN_PACKET_NONE -> servo never returns state packet;  
RETURN_PACKET_READ_INSTRUCTIONS -> servo answer packet state when a READ  
command is sent (to read position, temperature, etc)  
* @see addToPacket()  
*/
```

C.7.4. Comentarios y aclaraciones

Cuando sea necesario hacer aclaraciones, a nivel de código se harán utilizando el estilo de comentario con doble barra `//`. Por lo general los nombres de variables y funciones deberán ser de por si descriptivas por lo que este tipo de comentarios se reservan para partes del código especialmente enrevesadas.

Los comentarios, cuando vayan en línea con el código, se situarán a dos espacios del mismo, dejando un espacio entre el comentario en sí y la doble barra.

C.7.5. TODO y notas

En algunos casos se podrán dejar cosas para hacer en futuro (TODO) o notas aclaratorias (NOTE). En ambos casos se pondrá en mayúsculas y seguido de dos puntos. Quedando comentados mediante doble barra.

Ejemplo :

```
// TODO: complete CW and CCW selection
// NOTE: important the use of mascares to obtain direction os movement
```

C.7.6. Código en desuso

En algunas situaciones hay fragmentos de código que ya no se utilizan o están temporalmente deshabilitados. Estos fragmentos serán comentados mediante barra y asterisco :

Ejemplo :

```
/* ...
... some code ...
... */
```

C.8. Formato

C.8.1. Espacios y tabulaciones

Por norma general se utilizará cuatro espacios como indentación para distintos ámbitos.

Ejemplo :

```
void ServoRHA::setWheelSpeedToPacket( ... ) {
    ...
    if ( ... ) {
        ...
    }
    ...
}
```

C.8.2. Declaración y definición de funciones

El valor de retorno así como los parámetros de una función deberán ir en la misma línea. En caso de no caber o para mayor claridad se pondrán a la misma altura que los anteriores.

Ejemplo:

```
void ServoRHA :: setWheelSpeedToPacket( uint8_t *buffer , uint16_t speed , uint8_t
                                         direction ) {
    ...
}

void ServoRHA :: setWheelSpeedToPacket( uint8_t *buffer , uint16_t speed ,
                                         uint8_t direction ) {
    ...
}
```

C.8.3. Condicionales

Como norma general no se dejarán espacios entre los paréntesis. Si se dejará un espacio entre la sentencia **if** y el condicional, así como entre este último y la llave que abre el ámbito condicional.

Ejemplo:

```
//Forma correcta:
if (direction == CW) {
    speed = speed | 0x0400;
}

//Ejemplos incorrectos:
if(direction == CW) { // Falta un espacio tras la sentencia if
if(direction == CW){ // Falta un espacio entre el condicional y la llave
if(direction == CW){ // Combina los casos anteriores
```

En caso de que el condicional afecte solo a una sentencia esta se pondrá, como norma general, sin llaves y en la misma línea que el condicional. De igual forma se hará tras sentencias de tipo **else** o combinando **else if**. En caso de utilizar llaves se seguirá la norma que aplica a dicho caso.

Ejemplo:

```
if (speed1 < speed2-speed_margin) return ServoRHAConstants :: LESS_THAN;
else if (speed1 > speed2+speed_margin) return ServoRHAConstants :: GREATER_THAN;
else return ServoRHAConstants :: EQUAL;
```

Cuando si afecta a diferentes líneas y hay sentencias de tipo **else**, estas irán en la misma línea de cierre de llave del condicional (siempre que no afecte a la legibilidad del código ya sea por presencia de comentarios u otras causas similares).

Ejemplo:

```
if (....) {
    ...
} else {
    ...
}
```

C.8.4. Bucles

El formato será equivalente al caso de los condicionales:

Ejemplo:

```
for (...) {
    ...
}
for (...) oneLineStatement;
while (condition) {
    ...
}
```

C.8.5. Valor de retorno de funciones y métodos

No es necesario utilizar paréntesis para rodear la expresión a retornar. Solo se utilizarán en los casos en que se utilizarían si se fuera a asignar dicha expresión a una variable.

C.8.6. Formato para clases

Las directivas `public`, `protected` y `private` irán indentados un espacio respecto a la definición de la clase. Por norma general irán precedidos por una línea en blanco (excepto cuando las preceda la definición de la propia clase).

Ejemplo:

```
class JointHandler {
    private: // un espacio
    ...
    public:
    ...
```

C.8.7. Espacios de nombre

Los espacios de nombres siguen la norma general para indentar diferentes ámbitos.

C.9. Espacios en blanco

Los espacios horizontales dependerán de cada caso. En ningún caso se finalizará una línea con un espacio en blanco.

C.9.1. Caso general

Ejemplo:

```
void JointHandler::setTimer(uint64_t timer) { // Un espacio entre el cierre
    del parentesis y la apertura de llaves
    class TimerMicroseconds : public Timer { // Espacio entre los dos puntos en
        casos de herencia o inicializadores dentro de constructores. Se pone un
        espacio a cada lado.
    void checkWait() // No se deja espacio entre el nombre y los parentesis.
    Tampoco entre parentesis vacios.
```

```
float getError() { return error_; } // Se deja espacio entre llaves e
// implementacion, a ambos lados.
```

C.9.2. Bucles, condicionales y estructuras de control

Ejemplo:

```
if (b) { // Espacio entre la sentencia if y la condicion, asi como esta misma
    con la apertura de llaves
} else { // Espacios al rededor de la sentencia else
}
switch (i) {
    case 1: // No se deja espacio antes de los dos puntos
    ...
    case 2: break; // Si se deja despues de los mismos
for (int i = 0 ; i < 5 ; i++) { // En caso de bucles for, ademas de los
    espacios al rededor de los parentesis se dejara un espacio tras cada punto
    y coma.
```

C.9.3. Operadores

Ejemplo:

```
// En general se deja un espacio al rededor de los distintos tipos de
operadores
x = 0;
v = w * x + y / z;
v = w*x + y/z;
v = w * (x + z);

// No se separan operadores unarios de sus argumentos:
x = -5;
x++;
if (x && !y)
```

¿ESPAZOS VERTICALES Y HORIZONTALES DEBERIAN IR DENTRO DE LA SECCION DE FORMATO NO?

C.10. Espacio vertical

Por lo general se dejaran espacios verticales para una mayor claridad del código sin abusar de los mismos. Aunque separar diferentes partes puede ayudar demasiados espacios verticales pueden dificultar la lectura de código.

HABLAR DE COMO SE DEFINEN LAS VARIABLES, CUANDO SE PUEDEN PONER VARIAS EN LA MISMA LINEA Y DEMAS

<https://google.github.io/styleguide/cppguide.html>

Apéndice D

Documentación del software

Apéndice E

Comunicación con los Servos G15

En este anexo se describe como se trabaja con el protocolo de comunicación a bajo nivel para codificar el paso de mensajes entre los servos G15 Cube y el microcontrolador.

Antes de describir el formato de la información cabe destacar que en todo momento la información enviada irá codificada en formato hexadecimal, para los paquetes enviados como recibidos desde el microcontrolador. Todos los paquetes, tanto los enviados a los servos como la respuesta por parte de los mismos tendrán en común la siguiente información:

- Encabezado: Los primeros dos bytes del mensaje estarán compuestos por encabezado que será el que marque el inicio del mensaje. Estos bytes serán: 0xFF 0xFF
- Un fin de mensaje: El último byte del mensaje estará marcado por un valor llamado *CheckSum* que será el encargado de verificar que todo el paquete ha llegado correctamente. El *CheckSum* es el inverso del valor binario de la suma de todos los bytes enviados a excepción del encabezado y el propio *CheckSum*. En la figura – **COMPLETAR** – se puede ver un ejemplo de como se calcula dicho valor.

EJEMPLO DE COMO SE CALCULA EL CHECKSUM

En los paquetes que se envíen a los servos la información se codificará de la siguiente manera concreta:

- Bytes 0 y 1: reservados para el encabezado.
- Byte 2: codifica el ID del servo al que se quiere enviar la acción. De forma general se puede utilizar la dirección 0xFE (en hexadecimal) para enviar el mensaje a todos los servos conectados.
- Byte 3: codifica la longitud del mensaje. Contando a partir del encabezado y el ID (excluyendo ambos) el número de bytes que se envían. De esta forma, al recibirse el paquete se podrá identificar el inicio del mismo, a que servo va dirigido (el resto ignorarán el paquete) y cuantos bytes tendrá que leer el aludido. Por supuesto el último byte de la cadena será el ya mencionado *CheckSum* cuyo valor tendrá que coincidir con el esperado al analizar la cadena.

- Byte 4: codifica la instrucción que se desea realizar. Sobre la memoria de los servos se podrán hacer operaciones de lectura y escritura de distinta manera. Se pueden ver las diferentes instrucciones posibles en la tabla E.1. Serán explicadas posteriormente más en detalle.
- Bytes del 5 al N: Parámetros que se quieran enviar al servo.
- Byte N+1: *CheckSum*

En la figura E.1 se puede ver representado, a modo de resumen gráfico, este esquema de información genérico.

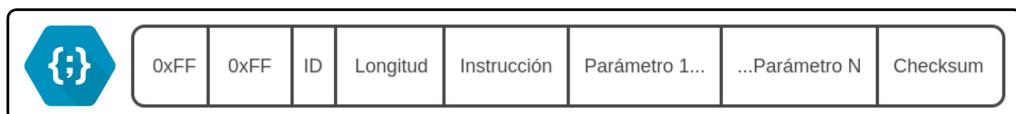


Figura E.1: Paquete de información genérico para comunicar con los Servos G15 Cube

Tabla E.1: Resumen de las instrucciones aceptadas por los Cytron G15 Cube servo

Fuente: Información obtenida de [16]

Instrucción	Valor Hex.	Comentarios
iPING	0x01	Solicita un paquete con el estado del servo
iREAD_DATA	0x02	Lee información de la memoria del servo
iWRITE_DATA	0x03	Escribe información en la memoria del servo
iREG_WRITE	0x04	Escribe sobre la memoria y hasta que llega la acción <i>ACTION</i> para ejecutar dichos cambios
iACTION	0x05	Activa la acción codificada con la instrucción <i>REG_WRITE</i>
iRESET	0x06	Resetea la memoria a los valores por defecto
iSYNC_WRITE	0x83	Para escribir simultáneamente información sobre varios servos

La respuesta por parte de los servos tiene también una estructura general que se detalla a continuación byte a byte:

- Bytes 0 y 1 de encabezado. Igual que en el caso anterior.
- Byte 2: codifica el ID del servo que responde.
- Byte 3: codifica la longitud a leer.
- Byte 4: sirve para informar de posibles errores en el servo. Cada bit del byte codifica un tipo de error, estando todos a 0 cuando la comunicación y el servo se encuentran buen estado. Estos errores están detallados en la tabla E.2, junto a la máscara en binario que se aplicará a dicho byte para comprobar cada error.
- Bytes del 5 al N: Parámetros que envía el servo.
- Byte N+1: *CheckSum*.

En la figura E.2 se puede ver representado, a modo de resumen gráfico, este esquema de información genérico que se ha expuesto previamente.

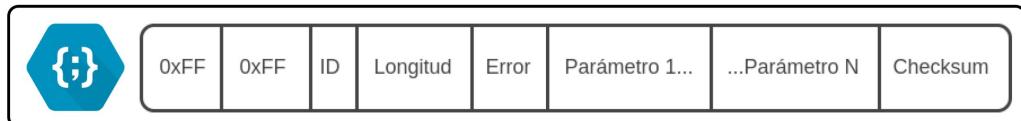


Figura E.2: Paquete de información genérico de retorno de los Servos G15 Cube

Tabla E.2: Codificación del error de los servos G15 Cube en cada bit del byte de error.

Fuente: Información obtenida de [16]

Bit	Error	Máscara a aplicar
0	Error en voltaje de entrada	0X0001
1	Límite de ángulo	0X0002
2	Sobrecalentamiento	0X0004
3	Error en el rango pedido	0X0008
4	Error en el CheckSum	0X0010
5	Sobrecarga	0X0020
6	Instrucción incorrecta	0X0040
7	-	-

Aunque como se ha visto anteriormente y se ha detallado en la tabla E.2 el servo devuelve un solo byte de error, al leer la información recibida el error, tanto en la librería de Cytron como en la estructura desarrollada para este proyecto este byte se amplia a dos bytes para añadir la posibilidad de nuevos errores en la recepción del paquete de datos. Se pueden ver de forma detallada en la tabla E.3, nuevamente junto a la máscara que se aplicará a dicho byte para cada caso. En el byte más bajo queda la información que devuelve el servo y en el más alto la información añadida.

Tabla E.3: Codificación del error de comunicación en cada bit del segundo byte de error.

Fuente: Información obtenida de [16]

Bit	Error	Máscara a aplicar
8	Paquete perdido o tiempo de espera superado	0X0100
9	Encabezado incorrecto	0X0200
10	ID incorrecto	0X0400
11	Error en el CheckSum	0X0800
12	-	-
13	-	-
14	-	-
15	-	-

E.1. Distintos tipos de instrucciones

A continuación se detallan las distintas instrucciones que aceptan los Servos, presentadas en la tabla E.1.

E.1.1. Petición del estado del servo

Se hace una petición del estado o una operación de tipo *PING* cuando se quiere conocer la existencia y estado de un servo con un ID específico. Adicionalmente si se envía utilizando el ID comodín (0xFE, para todos los servos) se podrá recoger el ID del servo conectado (cuando solo haya uno).

Un paquete para una operación *PING* podría tener el siguiente aspecto (*iPING* al servo con ID 1). El mensaje enviado tendrá una longitud de dos bytes a leer. Las diferentes instrucciones se han visto en la tabla E.1):

0xFF	0xFF	0x01	0x02	0x01	0xFB
------	------	------	------	------	------

Como se puede ver este mensaje no lleva parámetros.

El mensaje de retorno podría ser, por ejemplo el siguiente. Respondiendo el servo con ID 1 dos bytes a leer, el error que es 0 (todo correcto) y el *CheckSum*:

0xFF	0xFF	0x01	0x02	0x00	0xFC
------	------	------	------	------	------

E.1.2. Operaciones de lectura

Las operaciones de lectura, con la instrucción *iREAD_DATA* vista en la tabla E.1 están pensadas para leer la memoria interna de los servos. Los parámetros enviados serán la dirección de memoria a partir de la cual se quiere leer y el número de bytes a leer. Se puede ver en la tabla E.5 las posibles direcciones de memoria a las que acceder, el tipo de acceso que tienen (lectura/escritura o ambos) así como los valores típicos (por defecto, máximos y mínimos).

Un ejemplo de este tipo de mensajes puede ser una petición de lectura de la temperatura (0x2B en la tabla E.5). En este caso sobre el servo con ID 1. La temperatura ocupa un solo byte.

0xFF	0xFF	0x01	0x04	0x02	0x2B	0x01	0xCC
------	------	------	------	------	------	------	------

Como en el caso anterior el paquete de retorno devolverá el ID del servo que responde, la longitud, que en este caso será de 3 bytes, y el error, el parámetro leido (un byte con la temperatura) y el *CheckSum*:

0xFF	0xFF	0x01	0x03	0x00	0x20	0xDB
------	------	------	------	------	------	------

E.1.3. Operaciones de escritura

Las operaciones de escritura tienen un funcionamiento análogo al de lectura. La instrucción *iWRITE_DATA* vista en la tabla E.1 envía una cadena de bytes a escribir a partir de una dirección de memoria dada. Nuevamente se pueden ver las direcciones de memoria en la tabla E.5.

Como ejemplo se escribirá, sobre el servo con ID 1 un ángulo objetivo (empieza en la dirección 0x1E). Como se puede ver ocupa dos bytes, empezando por el de más bajo y a continuación el más alto. A la hora de enviar el mensaje hay que tener en cuenta estos aspectos para que los parámetros vayan ordenados de igual manera:

0xFF	0xFF	0x01	0x05	0x03	0x1E	0xAA	0x00	0x2E
------	------	------	------	------	------	------	------	------

El paquete de retorno en este caso devolverá el posible error en la comunicación, sin parámetros.

0xFF	0xFF	0x01	0x02	0x00	0xFC
------	------	------	------	------	------

E.1.4. Operaciones de escritura con activación desacoplada

La idea de este tipo de operación es la de enviar a distintos servos operaciones de escritura pero que queden pendientes de ejecutarse. De esta forma, utilizando la instrucción *iACTION* se podrán activar todos a la vez.

Para la operación de escritura en memoria se seguirá un formáto análogo al descrito en el apartado E.1.3. El único cambio es que la instrucción utilizada, en vez de ser *iWRITE_DATA* como se ha descrito en dicho apartado, se utilizará la instrucción *iREG_WRITE* (se puede consultar en la tabla E.1).

Una vez enviada la información a escribir a todos los servos, se procederá a enviar el mensaje de activación. En este caso se enviará a todos los servos conectados (ID genérico de 0xFE):

0xFF	0xFF	0xFE	0x02	0x05	0xFA
------	------	------	------	------	------

Por norma general, en los casos en los que se utiliza el ID genérico (menos en el caso de la instrucción *iPING* en los que hay un servo conectado) no se recogerá mensaje de retorno alguno.

E.1.5. Resetear la memoria de los servos a los valores de fábrica

La instrucción *iRESET* devolverá la memoria al estado por defecto. Se pueden ver los valores por defecto en la tabla E.5.

Para efectuarla se enviará dicha instrucción al servo correspondiente, en este caso al servo cuyo ID es 0:

0xFF	0xFF	0x00	0x02	0x06	0xF7
------	------	------	------	------	------

Nuevamente el mensaje de error contendrá únicamente el posible error en la comunicación.

0xFF	0xFF	0x00	0x02	0x00	0xFD
------	------	------	------	------	------

Se debe tener en cuenta, que aunque el mensaje de retorno responda con el valor inicial, una vez ejecutado el reset de la memoria el ID del servo valdrá 1 y el *baud rate* será de 19,2kb/s

E.1.6. Operaciones de escritura sobre múltiples servos

De igual forma que se puede escribir sobre varios servos para activar la acción al mismo tiempo, se puede encapsular la información de todos los servos en un solo paquete de información. Este caso presenta la limitación de que la dirección de

memoria a la que se quiere acceder y la longitud de parámetros a escribir son iguales para todos los servos.

La comunicación será parecida a los casos anteriores. En este caso se utilizará siempre el ID genérico (0xFE). La instrucción a utilizar será la *iSYNC_WRITE* (0x83, de la tabla E.1). La longitud del paquete será:

$$\text{Longitudtotal} = (L + 1) * N + 4 \quad (\text{E.1})$$

Siendo L la longitud del mensaje a cada servo y N el número de servos en la ecuación E.1.

Los parámetros necesarios serán:

- Parámetro 1: dirección de memoria a partir de la cual se quiere escribir.
- Parámetro 2: longitud de los datos a escribir, L en la ecuación E.1 (distinto de la longitud del paquete).
- Parámetro 3: ID del servo al que se quiere escribir.
- Parámetro 4: primer byte de los datos a enviar al servo con el ID marcado en el parámetro 3.
- Parámetro 5: segundo byte de los datos a enviar al servo con el ID marcado en el parámetro 3.
- ...
- Parámetro L+3: ID del segundo servo sobre el que se quiere escribir.
- Parámetro L+4: primer byte a escribir sobre el mismo.
- ...

Se puede ver en la tabla E.4 un ejemplo de como se construye un paquete con este propósito; en este caso concreto se comunicará con cuatro servos para enviarle objetivos de posición y velocidad (orientarse a un cierto ángulo con una velocidad dada):

- G15 con ID 0 a la posición 0x010 con velocidad 0x150
- G15 con ID 1 a la posición 0x220 con velocidad 0x360
- G15 con ID 2 a la posición 0x030 con velocidad 0x170
- G15 con ID 3 a la posición 0x220 con velocidad 0x380

Aun siguiendo las indicaciones del manual de usuario no se ha conseguido efectuar este tipo de comunicación de forma efectiva. Se ha desarrollado una estructura de funciones para aplicarlo (en la librería original no estaba dicha funcionalidad implementada) que están a la espera de completarse.

Tabla E.4: Ejemplo paquete con la instrucción *iSYNC_WRITE*

Fuente: Ejemplo obtenido de [16]

Byte	Contenido	Exp	Byte	Contenido	Exp
1	0xFF	Encabezado	16	0x60	Param 2
2	0xFF	Encabezado	17	0x03	Param 2
3	0xFE	ID genérico	18	0x02	ID tercer servo
4	0x18	Longitud total	19	0x30	Param 1
5	0x83	Instrucción	20	0x00	Param 1
6	0x1E	Dirección	21	0x70	Param 2
7	0x04	Longitud para cada servo	22	0x01	Param 2
8	0x00	ID primer servo	23	0x03	ID cuarto servo
9	0x10	Param 1	24	0x20	Param 1
10	0x00	Param 1	25	0x02	Param 1
11	0x50	Param 2	26	0x80	Param 2
12	0x01	Param 2	27	0x03	Param 2
13	0x01	ID segundo servo	28	0x12	CheckSum
14	0x20	Param 1			
15	0x02	Param 1			

E.1.7. Aspectos interesantes a tener en cuenta

Los servos aceptan dos modos de funcionamiento: un modo para controlarlo en posición y otro modo de rotación continua. Se darán mas detalles sobre el modo de giro continuo ya que es sobre el que se vasa el funcionamiento del prototipo.

Para cambiar de un modo a otro se deberá modificar los cuatro bytes correspondientes a los límites de giro (cuatro bytes a partir de la dirección 0x06, CW Angle Limit (L), se puede ver la información más detallada en la tabla E.5). Es necesario recordar que el valor debe escribirse en hexadecimal respetando el byte inferior y superior tal y como se muestra en la tabla mencionada.

- Para activar el modo de giro continuo se pondrán ambos valores a 0.
- Para salir del modo de giro continuo se pondrá el *CW Angle Limit* a 0 y el *CCW Angle Limit* a 1087.

Es importante tener en cuenta que cuando se controla en modo rueda, aunque el registro sobre el que se escribe hace referencia a la velocidad del servo (dirección 0x020, Moving Speed (L)) e realidad lo que se está enviando es el par que ejercerá el servo, con valores discretizados entre 0 y 1023. En vacío los servos son capaces de alcanzar una velocidad de *65rpm*. A un mismo valor de par enviado la velocidad variará en función de la carga que soporte el servo.

Aunque el control se deba efectuar respecto al par, el valor contenido en la dirección 0x24 y 0x25 (Present Speed) contiene el dato de velocidad. Esta información se almacena de la siguiente manera: el 10 bit codifica el sentido de giro mientras que los 9 anteriores la velocidad. Esta codificación es equivalente a la dirección de memoria correspondiente a Present Load (0x28 y 0x29).

Estos datos se pueden obtener de la siguiente forma, expresada mediante pseudocódigo c++. Se obtiene la información de ambos bytes, en este caso concreto de los datos de velocidad: **DEFINIR EN ALGÚN SITIO QUE ES EL PSEUDOCÓDIGO**

```
velocidad = byte(L) \\
velocidad |= (byte(H) << 8)
velocidad_direction = ((velocidad & 0x0400) >> 10)
velocidad = velocidad & inverso(0x0400)
```

Además de la velocidad, a través de las posiciones de memoria adecuadas se podrá leer el torque aplicado (Present Load), la posición en la que se encuentra el servo (Present Position), la velocidad (Present Speed) así como el Voltaje, la temperatura o se podrá consultar si el servo se está moviendo o no. Para interpretar algunos de estos valores habrá que tener algunas consideraciones en cuenta:

- Posición: El dato de posición viene almacenado en un rango de 0 a 1087, para pasarlo a grados habrá que aplicar la ecuación E.2

$$Pos(grados) = \frac{(PosReg - 0) * (360 - 0)}{(1087 - 0) + 0} \quad (E.2)$$

- Velocidad: El dato de velocidad almacenado en los registros correspondientes viene en una escala de 0 a 1023. Para pasar a revoluciones por minuto (RPM) se deberá aplicar la ecuación E.3

$$Vel(RPM) = \frac{VelReg * 112,83}{1023} \quad (E.3)$$

- Torque aplicado: Es el torque efectuado por el servo en el momento actual (valor entre 0 y 1023). En el caso de control por par el valor es el mismo que se ha enviado.
- Detectar movimiento: Este registro se pone a uno cuando el servo se está moviendo a un objetivo de posición y a cero una vez llega. En caso de controlar en modo rueda este byte permanece inalterable.

Tabla E.5: Direcciones de memoria de los servos con los diferentes parámetros a ajustar. Incluye valores mínimos y máximos así como valores por defecto para cada parámetro

Fuente: Tabla obtenida de [16]

Area	Address (Hex)	Parameter	Read only /Read Write	Factory default value (Hex)	Minimum value (Hex)	Maximum value (Hex)
EEPROM	0 (0x00)	Model (L)	R	'G' (0x0F)	-	-
	1 (0x01)	Model(H)	R	15 (0x47)	-	-
	2 (0x02)	Firmware Revision	R		-	-
	3 (0x03)	ID	RW	1 (0x01)	0 (0x00)	253 (0xFD)
	4 (0x04)	Baud Rate	RW	103 (0x67)	3 (0x03)	255 (0xFF)
	5 (0x05)	Return Delay	RW	250 (0xFA)	1 (0x01)	255 (0xFF)
	6 (0x06)	CW Angle Limit (L)	RW	0 (0x0000)	0 (0x0000)	1087 (0x043F)
	7 (0x07)	CW Angle Limit (H)	RW			
	8 (0x08)	CCW Angle Limit (L)	RW	1087 (0x043F)	0 (0x0000)	1087 (0x043F)
	9 (0x09)	CCW Angle Limit (H)	RW			
	10 (0x0A)	Reserved	-	-	-	-
	11 (0x0B)	Temperature Limit	RW	70 (0x46)	0 (0x00)	
	12 (0x0C)	Lowest Voltage Limit	RW	65 (0x41)	65 (0x41)	178 (0xB2)
	13 (0x0D)	Highest Voltage Limit	RW	150 (0x96)		
	14 (0x0E)	Max Torque (L)	RW	1023 (0x03FF)	0 (0x0000)	1023 (0x03FF)
	15 (0x0F)	Max Torque (H)	RW			
	16 (0x10)	Return Packet Enable	RW	2 (0x02)	0 (0x00)	2 (0x02)
	17 (0x11)	Alarm LED	RW	36 (0x24)	0 (0x00)	127 (0x7F)
	18 (0x12)	Alarm Shutdown	RW	36 (0x24)	0 (0x00)	127 (0x7F)
	19 (0x13)	Reserved	-	-	-	-
	20 (0x14)	Down Calibration (L)	R			
	21 (0x15)	Down Calibration (H)	R			
	22 (0x16)	Up Calibration (L)	R			
	23 (0x17)	Up Calibration (H)	R			
RAM	24 (0x18)	Torque Enable	RW	0 (0x00)	0 (0x00)	1 (0x01)
	25 (0x19)	LED	RW	0 (0x00)	0 (0x00)	1 (0x01)
	26 (0x1A)	CW Compliance Margin	RW	1 (0x01)	0 (0x00)	254(0xFE)
	27 (0x1B)	CCW Compliance	RW	1 (0x01)	0 (0x00)	254(0xFE)
	28 (0x1C)	CW Compliance Slope	RW	32 (0x0020)	1 (0x01)	254(0xFE)
	29 (0x1D)	CCW Compliance Slope	RW	32 (0x0020)	1 (0x01)	254(0xFE)
	30 (0x1E)	Goal Position (L)	RW	Address 36	0 (0x0000)	1087 (0x043F)
	31 (0x1F)	Goal Position (H)	RW	Address 37		
	32 (0x20)	Moving Speed (L)	RW	0 (0x0000)	0 (0x0000)	1023 (0x03FF)
	33 (0x21)	Moving Speed (H)	RW			
	34 (0x22)	Torque Limit (L)	RW	Address 14	0 (0x0000)	1023 (0x03FF)
	35 (0x23)	Torque Limit (H)	RW	Address 15		
	36 (0x24)	Present Position (L)	R			
	37 (0x25)	Present Position (H)	R			
	38 (0x26)	Present Speed (L)	R			
	39 (0x27)	Present Speed (H)	R			
	40 (0x28)	Present Load (L)	R			
	41 (0x29)	Present Load (H)	R			
	42 (0x2A)	Present Voltage	R			
	43 (0x2B)	Present Temperature	R			
	44 (0x2C)	Registered	R	0 (0x00)	0 (0x00)	1 (0x01)
	45 (0x2D)	Reserved	-	-	-	-
	46 (0x2E)	Moving	R	0 (0x00)	0 (0x00)	1 (0x01)
	47 (0x2F)	Lock	RW	0 (0x00)	1 (0x01)	1 (0x01)
	48 (0x30)	Punch (L)	RW	32 (0x0020)	0 (0x0000)	1023 (0x03FF)
	49 (0x31)	Punch (H)	RW			

Apéndice F

Instalación y Configuración del software necesario

Bibliografía

- [1] ErgoDirect, “Características del modelo de soporte cotoytech mw-m13p.” http://www.ergodirect.com/product_info.php?products_id=16881, (s.d). [Consulta Online del 02-01-2018].
- [2] ErgoDirect, “Características del modelo de soporte cotoytech cm-m13.” http://www.ergodirect.com/product_info.php?products_id=16891, (s.d). [Consulta Online del 02-01-2018].
- [3] ICW, *Titan Elite Ceiling Mount Product Sheet Document (model T2EQ-C8X5)*, 2016. [Consulta Online del 02-01-2018].
- [4] Ergotron, *LX Sit-Stand Monitor Arms*, 2017. [Consulta Online del 02-01-2018].
- [5] Ergotron, “Características del modelo de soporte lx-stand.” <https://www.ergotron.com/en-us/products/product-details/45-360#/>, (s.d). [Consulta Online del 02-01-2018].
- [6] Maior, *Maior Flip 900 technical specification*, -. [Consulta Online del 28-01-2018].
- [7] K. Robotics, *Jaco User Guide Assistive Robotics*, 2017. [Consulta Online del 20-01-2018].
- [8] K. Robotics, “Características del modelo de soporte jaco3.” <http://www.kinovarobotics.com/assistive-robotics/products/robot-arms/>, (s.d). [Consulta Online del 20-01-2018].
- [9] K. Robotics, *Mico2 technical specification*, 2017. [Consulta Online del 20-01-2018].
- [10] P. C. Márquez P, Alfredo J., “Avances en el desarrollo de un prototipo de robot asistencial para personas con limitaciones de movilidad,” *Ingenio Magno*, vol. 4, pp. 53–56, 2013.
- [11] T. S. Hideyuki Uehara, Hiroki Higa, “A mobile robotic arm for people with severe disabilities,” in *3rd IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*, University of Tokyo, September 2010.
- [12] R. R. Tariq Rahman, “A simple technique to passively gravity-balance articulated mechanisms.” PDF, published in The College of Information Sciences and Technology - CiteSeerX. [Consulta Online en Noviembre de 2017].

- [13] L. da Vinci, *Codex Madrid I: Tratado de estática y mecánica*, vol. 1, pp. 59–60. 1493. [recurso ONLINE, documentos digitalizados en año 2017 por La Biblioteca Nacional de España].
- [14] H. del Campo, “Polipastos: teoría y aplicaciones.” <http://www.granvertical.com/2015/05/23/98/>, 2015. [Consulta Online en Enero de 2018].
- [15] D. H. Summerville, *Embedded Systems Interfacing for Engineers Using the Freescale HCS08 Microcontroller*, vol. II of *Digital and analog hardware interfacing*. Morgan and Claypool Publishers, 2009. [Consulta Online].
- [16] Cytron Technologies Sdn. Bhd., *G15 Cube Servo User’s Manual ROBOT . HEAD to TOE Product User’s Manual –G15 Cube Servo*, 2012.
- [17] Arduino, “Arduino uno tech specs.” <https://store.arduino.cc/arduino-uno-rev3>, -. [Consulta Online en Enero de 2018].
- [18] RobotShop, *Arduino Mega 2560 Datasheet*, -.