

# **Memoria del proyecto**

para el

## Trabajo de Fin de Grado sobre el manipulador *pArm*

Javier Alonso Silva  
Mihai Octavian Stanescu  
José Alejandro Moya Blanco

*Universidad Politécnica de Madrid  
Ingeniería de Computadores  
Trabajo de Fin de Grado  
Tutor: Norberto Cañas de Paz*

Madrid, 6 de septiembre de 2020

# Índice general

Índice de figuras	v
Índice de cuadros	xii
Historial de versiones	xii
Definiciones, siglas, y abreviatura	xv
<b>1. Motivación y objetivos</b>	<b>1</b>
1.1. [REVISADO] Estado del arte . . . . .	1
1.1.1. Desarrollo de la robótica a lo largo de la historia . . . . .	1
1.1.2. Los brazos robóticos . . . . .	4
1.1.3. La actualidad . . . . .	5
1.2. [J] Motivaciones y objetivos del desarrollo del proyecto . . . . .	11
1.3. [J] Metodología . . . . .	12
<b>2. [REVISADO] Explicación de la estructura del proyecto</b>	<b>13</b>
2.1. Matemáticas . . . . .	14
2.2. <i>Hardware</i> . . . . .	16
2.3. <i>Software</i> . . . . .	16
<b>3. Diagramas, requisitos y diseño</b>	<b>18</b>
3.1. Requisitos . . . . .	18
3.2. Diagramas y diseño . . . . .	18
<b>4. Las matemáticas del proyecto</b>	<b>34</b>
4.1. Cinemática directa . . . . .	34

---

4.2. Cinemática inversa . . . . .	34
4.3. Funciones jacobianas . . . . .	34
4.4. Implementación final realizada . . . . .	34
<b>5. Hardware</b>	<b>35</b>
5.1. [REVISADO] Impresión en 3D . . . . .	35
5.2. [REVISADO] Teoría de la estructura física . . . . .	45
5.3. [REVISADO] Microcontrolador utilizado . . . . .	46
5.4. [REVISADO] Desarrollo y componentes de la PCB . . . . .	47
5.4.1. Objetivos . . . . .	47
5.4.2. Componentes principales . . . . .	48
5.4.3. Diseño lógico y diagrama esquemático . . . . .	49
5.4.4. Conversión del diagrama esquemático a diagrama físico . . . . .	63
5.4.5. Conexionado de los componentes mediante pistas . . . . .	72
5.4.6. Verificaciones realizadas al diseño lógico y físico . . . . .	78
5.4.7. Construcción . . . . .	80
5.4.8. Verificaciones del prototipo construido . . . . .	99
5.4.9. Contratiempos ocurridos y soluciones ingenieras . . . . .	101
5.5. Comunicaciones . . . . .	110
5.6. [REVISADO] Motores empleados (actuadores) . . . . .	110
<b>6. Software</b>	<b>115</b>
6.1. S1 . . . . .	115
6.1.1. UI/UX . . . . .	115
6.1.2. Protocolo de comunicación . . . . .	115
6.1.3. Pseudolenguaje de comunicación . . . . .	115
6.1.4. Logs . . . . .	115
6.1.5. Otros . . . . .	115
6.2. S2 . . . . .	115
6.2.1. Protocolo de comunicación . . . . .	115
6.2.2. Interpretación del pseudo-lenguaje . . . . .	115
6.2.3. Cálculo de movimientos/trayectorias . . . . .	115

---

6.2.4. Control de los componentes . . . . .	115
6.2.5. Otros . . . . .	115
<b>7. Casos de estudio</b>	<b>116</b>
7.1. Decisiones tomadas . . . . .	116
7.2. Desarrollo de las distintas partes del proyecto . . . . .	116
<b>8. Calidad y pruebas</b>	<b>117</b>
8.1. Batería de pruebas . . . . .	117
8.2. Explicación de las pruebas . . . . .	117
8.3. Resultados esperados   resultados obtenidos . . . . .	117
8.4. Reflexión - solución . . . . .	117
<b>9. Demostración</b>	<b>118</b>
<b>10. Planificación, costes y tiempo empleado</b>	<b>119</b>
10.1. Diagramas de Gantt . . . . .	119
10.2. Sueldos propuestos y costes obtenidos . . . . .	119
10.3. Coste de los materiales inicial - coste de los materiales final . . . . .	120
10.4. Evolución del tiempo empleado . . . . .	120
10.5. Contratiempos y tiempo de desarrollo final . . . . .	121
<b>11. Conclusiones</b>	<b>122</b>
11.1. Conclusiones técnicas . . . . .	122
11.2. Experiencia personal en el desarrollo del proyecto . . . . .	122
11.3. Conocimientos adquiridos y nuevas competencias . . . . .	122
<b>12. Futuras mejoras</b>	<b>123</b>
12.1. Desarrollos e implementaciones que no han podido realizarse . . . . .	123
12.2. Ideas propuestas pero no implementadas . . . . .	123
12.3. Otras . . . . .	123
<b>Bibliografía</b>	<b>124</b>
<b>A. Código fuente S2</b>	<b>127</b>

A.1. <i>Header files</i>	127
A.2. <i>Source files</i>	140
<b>B. Especificación de requisitos</b>	<b>152</b>

# Índice de figuras

1.1.	Flautista y tamborilero de Vaucanson [2]. . . . .	1
1.2.	En 1774, “lady musician” por Jaquet-Droz [3]. . . . .	2
1.3.	“The Turk”, creado por von Kempelen en 1769 [4]. . . . .	2
1.4.	Barco a control remoto de Nicola Tesla, en 1898 [5] . . . . .	3
1.5.	“Alpha”, el primer robot diseñado con fines militares y su posterior evolución, “Elektro”. . . . .	3
1.6.	Exposición mundial del 2005 en Japón [6]. . . . .	4
1.7.	Grados de libertad de un brazo robótico y estructura del cuerpo humano [1]. .	4
1.8.	Vista exterior del Canadarm2 [7]. . . . .	5
1.9.	Rover “Spirit”, desarrollado por la NASA y desplegado en 2004 [8]. . . . .	6
1.10.	Modo de funcionamiento del sistema “ <i>rocker–bogie</i> ”, desarrollado por la NASA para sus rover [9]. . . . .	6
1.11.	Rover “Opportunity”, desarrollado por la NASA y desplegado en 2004 [10]. .	7
1.12.	Lugares de aterrizaje de los rovers de la misión espacial a Marte [11]. . . . .	7
1.13.	Robot Roomba en la estación de carga [12]. . . . .	8
1.14.	Lo que ve un Tesla cuando está en conducción autónoma nivel 2 [13]. . . . .	8
1.15.	Robot “ <i>Big-Dog</i> ” de Boston Dynamics [15]. . . . .	9
1.16.	Robot “ <i>Atlas</i> ” de Boston Dynamics [15]. . . . .	9
1.17.	Robot “KR-1000 Titan” de KUKA [16]. . . . .	10
1.18.	Robot $\mu$ Arm de UFACTORY [17]. . . . .	10
2.1.	Ejemplo de cadena cinemática [21]. . . . .	15
2.2.	Diagrama del modelo cinemático [22]. . . . .	15
3.1.	Diagrama de bloques del Sistema 2 – $p$ Arm (S2) . . . . .	19

---

3.2.	Diagrama de estados del método <code>main()</code> del <i>orchestrator</i> . . . . .	20
3.3.	Diagrama de estados del método <code>loop()</code> del <i>orchestrator</i> . . . . .	21
3.4.	Diagrama de estados del método <code>CheckMotorHealthStatus()</code> del <i>orchestrator</i> . . . . .	22
3.5.	Diagrama de estados del método <code>CancelMovement()</code> del <i>orchestrator</i> . . . . .	23
3.6.	Diagrama de estados del método <code>CommunicateError()</code> del <i>orchestrator</i> . . . . .	23
3.7.	Diagrama de estados del método <code>DoHandShake()</code> del <i>orchestrator</i> . . . . .	24
3.8.	Diagrama de estados del método <code>ExecuteMovement()</code> del <i>orchestrator</i> . . . . .	25
3.9.	Diagrama de estados del método <code>ExecuteOrder()</code> del <i>orchestrator</i> . . . . .	26
3.10.	Diagrama de estados del método <code>moveArm()</code> del <i>orchestrator</i> . . . . .	26
3.11.	Diagrama de estados del método <code>uartInit()</code> del <i>UART</i> . . . . .	27
3.12.	Diagrama de estados del método <code>setBaudrate()</code> del <i>UART</i> . . . . .	28
3.13.	Diagrama de estados del método <code>sendMessage()</code> del <i>UART</i> . . . . .	28
3.14.	Diagrama de estados del método <code>receiveBitStream()</code> del <i>UART</i> . . . . .	29
3.15.	Diagrama de estados del método <code>init()</code> del <i>motorHandler</i> . . . . .	29
3.16.	Diagrama de estados del método <code>checkMotorStatus()</code> del <i>motorHandler</i> . . . . .	30
3.17.	Diagrama de estados del método <code>sendMovementToMotors()</code> del <i>motorHandler</i> . . . . .	30
3.18.	Diagrama de estados del método <code>cancelMovement()</code> del <i>motorHandler</i> . . . . .	31
3.19.	Diagrama de estados del método <code>init()</code> del <i>movementComputer</i> . . . . .	32
3.20.	Diagrama de estados del método <code>computeXYZMovement()</code> del <i>movementComputer</i> . . . . .	32
3.21.	Diagrama de estados del método <code>computeAngleMovement()</code> del <i>movementComputer</i> . . . . .	33
5.1.	Concepto inicial del brazo robótico. . . . .	36
5.2.	Logotipos de las herramientas utilizadas. . . . .	37
5.3.	Flujo de trabajo del desarrollo y la impresión 3D. . . . .	37
5.4.	Proyección de la placa de control (naranja) sobre la base original del $\mu$ Arm (gris)	38
5.5.	Placa y motor dentro de la caja original del $\mu$ Arm . . . . .	38
5.6.	Base y paredes tras realizar las modificaciones necesarias . . . . .	39
5.7.	Sistema de raíles de la placa . . . . .	40
5.8.	Pieza de sujeción del motor . . . . .	41
5.9.	Pieza de sujeción del motor (verde) dentro de la caja . . . . .	41
5.10.	Tapa original (derecha) junto a la tapa modificada (izquierda) . . . . .	42

---

5.11. Tapa superior transparente y base giratoria (verde) . . . . .	42
5.12. Pieza de unión del motor y el eje . . . . .	43
5.13. Sistema completo . . . . .	44
5.14. Placa de sujeción de los motores laterales . . . . .	45
5.15. Una estructura pantográfica que transmite el movimiento de un punto al siguiente [25]. . . . .	46
5.16. Diagrama de conexionado del LM317 [26]. . . . .	50
5.17. Diagrama esquemático del circuito de alimentación de los servomotores. . . . .	51
5.18. Diagrama de conexionado del regulador L7805 [27]. . . . .	52
5.19. Diagrama de conexionado del regulador AZ1117H [28]. . . . .	52
5.20. Diagrama esquemático de la etapa de alimentación del microcontrolador. . . . .	53
5.21. Diagrama esquemático del microcontrolador y sus periféricos. . . . .	54
5.22. Conexionado mínimo del microcontrolador [29]. . . . .	55
5.23. Diagrama esquemático del conexionado del generador de señales. . . . .	55
5.24. <i>Pinout</i> del conector de la sonda de programación [29]. . . . .	56
5.25. Conexión del pin $\overline{MCLR}/V_{PP}$ [29]. . . . .	56
5.26. Diagrama esquemático del puerto de programación. . . . .	57
5.27. Circuito lógico para los finales de carrera. . . . .	58
5.28. Diagrama esquemático del puerto TRIS. . . . .	59
5.29. Esquema del generador PWM [29]. . . . .	60
5.30. Diagrama esquemático del puerto PWM. . . . .	61
5.31. Esquema del periférico UART [29]. . . . .	61
5.32. Diagrama esquemático de los puertos UART. . . . .	62
5.33. Diagrama esquemático de los LEDs. . . . .	62
5.34. Diagrama esquemático completo. . . . .	63
5.35. Herramienta de asignación de huellas. . . . .	64
5.36. Ventana de asignación de huellas físicas. . . . .	65
5.37. Huella física de un condensador usado en la <i>Printed Circuit Board</i> (PCB). . . . .	66
5.38. Encapsulado elegido para el microcontrolador [29]. . . . .	67
5.39. Huella física asignada al microcontrolador. . . . .	68
5.40. Herramienta de generado de listado de redes. . . . .	68

---

5.41. Archivo de listado de redes . . . . .	69
5.42. Acceso directo a la herramienta “PCBnew” . . . . .	69
5.43. Herramienta de importado de listado de redes. . . . .	70
5.44. Situación inicial del diseño nada mas importar los componentes físicos. . . . .	70
5.45. Distribución inicial de los componentes. . . . .	71
5.46. Ventana principal de “PCB Calculator” . . . . .	73
5.47. Cálculo del ancho de pistas de alimentación. . . . .	74
5.48. Cálculo del ancho de pistas de comunicación. . . . .	74
5.49. Cálculo inverso del ancho de pistas de comunicación. . . . .	75
5.50. Distribución final de los componentes físicos. . . . .	76
5.51. Diagrama físico final. . . . .	77
5.52. Representación 3D del diseño físico. . . . .	78
5.53. Herramienta de verificación de reglas de diseño. . . . .	80
5.54. Estructura de la placa de prototipado [30]. . . . .	81
5.55. Tratado de las resinas mediante insolado [30]. . . . .	82
5.56. Resultado tras el insolado [30]. . . . .	82
5.57. Proceso de revelado [30]. . . . .	83
5.58. Resultado tras revelado [30]. . . . .	83
5.59. Resultado tras atacado [30]. . . . .	83
5.60. Herramienta de impresión de placas. . . . .	84
5.61. Fotomáscara final de la PCB. . . . .	85
5.62. Proceso de impresión y recorte de las fotomáscaras. . . . .	85
5.63. Fotomáscara final. . . . .	86
5.64. Proceso de insolación de la PCB usando las fotomáscaras. . . . .	87
5.65. Placa de prototipado positiva insolada. . . . .	87
5.66. Revelador universal usado. . . . .	88
5.67. Proceso de revelado de la PCB. . . . .	89
5.68. Proceso de atacado de la PCB. . . . .	90
5.69. Circuito impreso final tras el proceso de atacado. . . . .	90
5.70. Integrante del equipo verificando cortos. . . . .	91
5.71. Algunos de los cortos detectados. . . . .	91

---

5.72. Taladrado de la PCB. . . . .	92
5.73. PCB con orificios taladrados y margenes guillotinados. . . . .	92
5.74. Plano detallado del microcontrolador. . . . .	93
5.75. Aplicado de la pasta de soldadura. . . . .	93
5.76. Colocado del microcontrolador sobre la pasta de soldadura. . . . .	94
5.77. Diferentes etapas del proceso de horneado. . . . .	95
5.78. Imagen a contra luz de la PCB. . . . .	96
5.79. Comprobación de cortos y conductividad de las pistas del microcontrolador. .	96
5.80. Conexionado de las vías de la PCB. . . . .	97
5.81. Comienzo del proceso de soldadura. . . . .	97
5.82. Integrantes del equipo soldando componentes. . . . .	98
5.83. Proceso de soldadura en curso. . . . .	98
5.84. Integrantes del equipo soldando componentes. . . . .	99
5.85. Primer encendido de la PCB usando un código de prueba del PWM. . . . .	100
5.86. Prueba del funcionamiento de la UART. . . . .	100
5.87. Prueba del funcionamiento de los servomotores. . . . .	101
5.88. PCB tras proceso de atacado fallido. . . . .	102
5.89. Proceso de creación de la disolución de atacado. . . . .	102
5.90. Intentos de fabricación de la PCB. . . . .	103
5.91. Fractura en pista de cobre. . . . .	103
5.92. Parcheo de la pista usando hilo de grapinar. . . . .	104
5.93. Reconexión del nuevo módulo PWM al puerto. . . . .	104
5.94. Soldadura del hilo de reconexión con el nuevo pin. . . . .	105
5.95. Esquemático lógico del LM317. . . . .	106
5.96. Huella física del LM317. . . . .	106
5.97. Encapsulado físico del LM317. . . . .	106
5.98. Huella física del LM317 con pines reordenados. . . . .	107
5.99. Modificación del circuito de alimentación de los servomotores, marcada en azul.	108
5.100 Puentes entre pistas de los reguladores de tensión LM317. . . . .	109
5.101 Reflexión [31]. . . . .	110
5.102 Motor de corriente continua . . . . .	110

---

5.103Motor paso a paso . . . . .	111
5.104Servomotor de corriente continua . . . . .	111
5.105Ejemplo genérico de control de posición mediante señal PWM [38]. . . . .	112
5.106Servomotor Parallax utilizado [39] . . . . .	114

# Índice de cuadros

10.1. Tabla completa de presupuestos. . . . .	120
---	-----

# Historial de versiones

Revisión	Fecha	Autor(es)	Descripción
000	27.05.2020	J. Alonso, M. Stanescu, A. Moya	Comienzo del desarrollo de la memoria.
001	16.06.2020	J. Alonso, M. Stanescu, A. Moya	Primera revisión de la memoria – revisados los puntos 1.1 y 2
002	23.06.2020	J. Alonso, M. Stanescu, A. Moya	Revisados los puntos 5.1, 5.2, 5.3
003	30.06.2020	J. Alonso, M. Stanescu, A. Moya	Revisados los puntos 1.2, 1.3 y 3.1
004	06.09.2020	J. Alonso, M. Stanescu, A. Moya	Revisado el punto 5.4

# Definiciones, siglas, y abreviaturas

**SDK** *Software Development Kit*

**ROS** *Robot Operating System*

**SW** *software*

**OS** *Open-Source*

**OH** *Open-Hardware*

**S1** Sistema 1 – ordenador

**S2** Sistema 2 – *p*Arm

**GUI** *Graphical User Interface*

**GTK** *GIMP Toolkit*

**SoC** *System On Chip*

**PWM** *Pulse-Width Modulation*

**GPIO** *General Purpose Input/Output*

**UART** *Universal Asynchronous Receiver-Transmitter*

**RAM** *Random Access Memory*

**PLA** Ácido Poliláctico

**ABS** Acrilonitrilo Butadieno Estireno

**DSP** *Digital Signal Processor*

**PLL** *Phase Loop Lock*

**THT** *Through-Hole Technology*

**SMD** *Surface-Mount Device*

**PCB** *Printed Circuit Board*

- *Software Development Kit* (SDK) – colección de herramientas *software* (SW) disponibles para instalar en un único paquete.
- *hand-shake* – en informática, negociación entre pares para establecer de forma dinámica los parámetros de un canal de comunicaciones.
- *Robot Operating System* (ROS) – conjunto de librerías SW que ayudan a construir aplicaciones para robots.
- *Firmware* – SW programado que especifica el orden de ejecución del sistema.
- *Graphical User Interface* (GUI) – siglas que significan “Interfaz Gráfica de Usuario” (en castellano).
- *GIMP Toolkit* (GTK) – biblioteca de componentes gráficos multiplataforma para desarrollar interfaces gráficas de usuario.
- *System On Chip* (SoC) – tecnología de fabricación que integra todos o gran parte de los módulos en un circuito integrado.
- *Pulse-Width Modulation* (PWM) – Señal cuadrada de periodo habitualmente constante, entre flancos de subida, en la que se modula el tiempo a nivel alto
- *General Purpose Input/Output* (GPIO) – pin genérico cuyo comportamiento puede ser controlado en tiempo de ejecución.
- *Universal Asynchronous Receiver-Transmitter* (UART) – estándar de comunicación dúplex.
- Dúplex – término que define a un sistema que es capaz de mantener una comunicación bidireccional, enviando y recibiendo mensajes de forma simultánea.

- Widget – la parte de una GUI (interfaz gráfica de usuario) que permite al usuario interconectar con la aplicación.
- *Random Access Memory* (RAM) – memoria principal del ordenador, donde se guardan programas y datos, sobre la que se pueden efectuar operaciones de lectura y escritura.
- *Deep-Sleep* – estado de un microcontrolador en el cual consume muy poca cantidad de energía.
- *bit* – unidad mínima de información de un computador digital.
- *Through-Hole Technology* (THT) – tecnología que utiliza agujeros pasantes que se practican en las placas de los circuitos impresos para el montaje de diferentes elementos electrónicos.
- *Surface-Mount Device* (SMD) – tecnología que utiliza componentes de montaje superficial para la inserción de diferentes elementos electrónicos en un circuito impreso.
- PCB – Placa de circuito impreso.

# Capítulo 1

## Motivación y objetivos

### 1.1. [REVISADO] Estado del arte

#### 1.1.1. Desarrollo de la robótica a lo largo de la historia

El mundo de la robótica da acceso a resolver una gran variedad de problemas donde el ser humano estaba limitado físicamente: levantar cargas de gran peso, realizar tareas repetitivas durante tiempos prolongados, etc. Además, como bien se sabe, ha permitido el desarrollo de cadenas de producción en masa para poder desarrollar y crear los productos que usamos diariamente, desde el coche hasta el teléfono móvil.

Desde que se empezó a investigar en este campo, el desarrollo de los brazos robóticos ha sido exponencial: se empezó trabajando con pequeños autómatas hasta el desarrollo de la revolución industrial [1].

Los primeros modelos, como se puede ver en la figura 1.1, empezaron intentando hacer representaciones de las manos humanas. En particular, se crearon un flautista y un tamborilero los cuales eran capaces de tocar los respectivos instrumentos utilizando un complejo sistema de cables y engranajes para poder mover los “dedos” de los músicos.

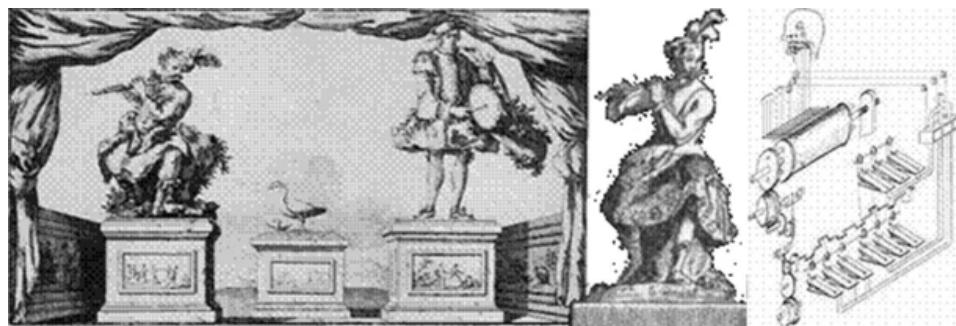


Figura 1.1: Flautista y tamborilero de Vaucanson [2].

Siguiendo con esta idea, se fue mejorando y desarrollando el modelo de imitación de las

articulaciones y los miembros de los humanos, llegando a construir estructuras más complejas y avanzadas, pensadas en su momento para poder tocar el clavicordio mediante un muñeco, como se muestra en la figura 1.2:

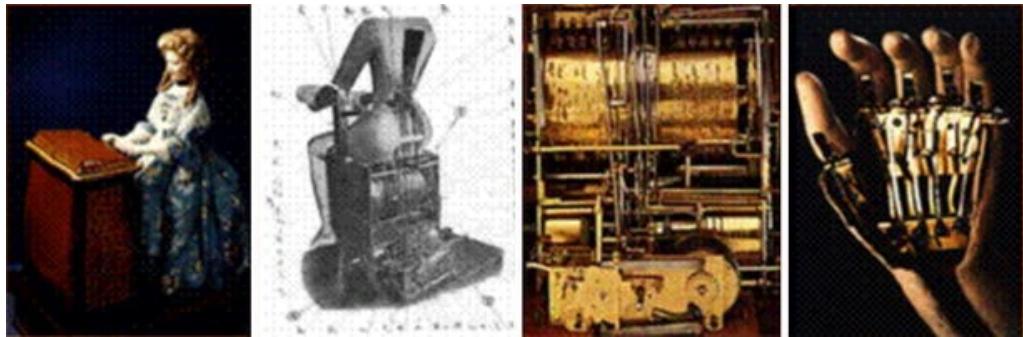


Figura 1.2: En 1774, “lady musician” por Jaquet-Droz [3].

Durante los años siguientes, el proceso se fue refinando hasta el punto de desarrollar un autómata el cual era capaz de jugar al ajedrez, llamado “The Turk” **standage\_tom\_turk\_2002**, construido en 1769. La estructura comprendía un conjunto de mecanismos los cuales eran controlados por un operador, encargado de realizar los movimientos del brazo izquierdo del autómata.

En la figura 1.3 se puede ver cómo está diseñado el sistema para mover un controlador pantográfico sobre el tablero de juego, controlado por el operador externo antes mencionado:

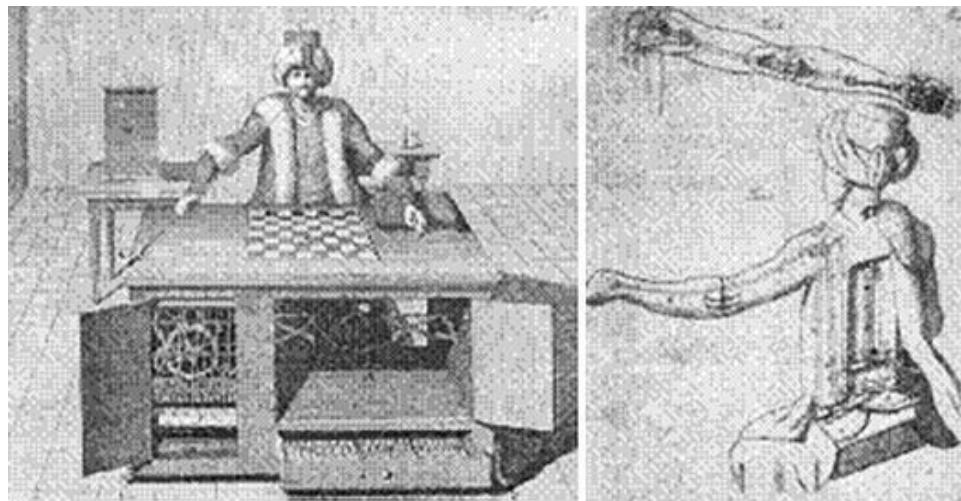


Figura 1.3: “The Turk”, creado por von Kempelen en 1769 [4].

Desde entonces, la robótica ha evolucionado y crecido de manera exponencial. Por una parte, debidas las distintas guerras que han habido en los últimos 200 años, se ha dado un gran impulso a la industria encargada de crear distintos dispositivos con fines de defensa y ataque. En particular, se potenciaron mucho los desarrollos de dispositivos por control remoto, destacando el diseño de NiKola Tesla en 1898 de un barco completamente automatizado, controlado por control remoto y sumergible, como se puede ver en la figura ??:

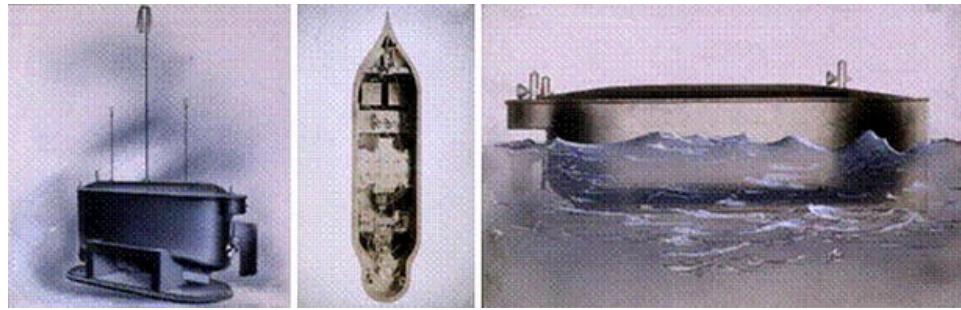


Figura 1.4: Barco a control remoto de Nicola Tesla, en 1898 [5]

Por otro lado, dada la cantidad de bajas de las Primera y Segunda Guerras Mundiales, se empezaron a desarrollar robots que permitieran sustituir a los militares en el campo de batalla, destacando en este campo el robot “Elektro”, creado por la compañía Westinghouse. Dicho robot supuso un gran éxito en la industria de los robots y armamentística, pudiendo moverse completamente, disparar armas, mover elementos faciales para “expresar emociones” e inclusive poder comunicarse.

En la figura 1.5, se puede ver a la izquierda la primera versión “Alpha” y, a la derecha, la versión mejorada “Elektro”:

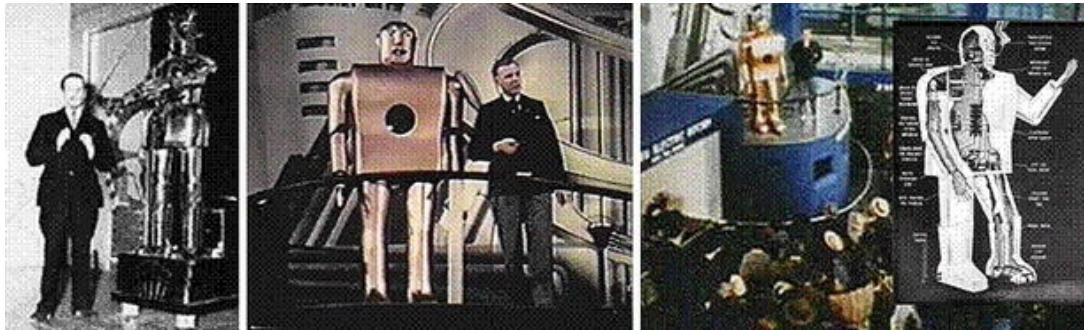


Figura 1.5: “Alpha”, el primer robot diseñado con fines militares y su posterior evolución, “Elektro”.

Toda esta evolución ha desembocado en la robótica moderna, en donde tenemos robots sofisticados y con distintos actuadores, pudiendo interactuar con muchísimos elementos de nuestro entorno y trabajar en distintas fases de producción de cadenas de montaje en serie. Además, se trabaja continuamente para que cada vez los robots puedan realizar más tareas de los humanos, mejorando cada vez más los “*end-effectors*” (controladores del final de los extremos del brazo). En la figura 1.6 se puede ver cómo robots medianamente antiguos (del 2005) ya podían realizar diversas actividades, como interactuar con las personas o tocar un instrumento.

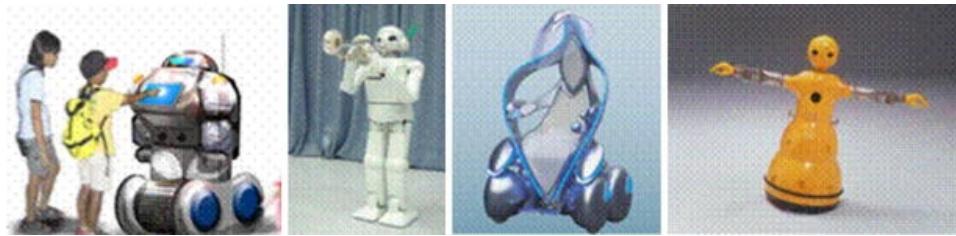


Figura 1.6: Exposición mundial del 2005 en Japón [6].

### 1.1.2. Los brazos robóticos

Con los avances actuales, el mundo de la robótica ha evolucionado a un nuevo nivel: con la inclusión de los transistores en lugar de las válvulas de vacío se han podido desarrollar circuitos integrados que manejan de manera mucho más sofisticada el control del brazo robótico.

En 1962, la empresa “Unimate” introdujo su primer brazo robótico de carácter industrial. Aproximadamente, se vendieron 8500 unidades. Este hito es importante en tanto a que se valoraron por primera vez los grados de libertad que debían de tener los brazos robóticos.

Estos planteamientos derivaron en distintos robots famosos que incluso siguen en activo hoy día. En 1969, Victor Scheinman, de la Universidad de Standford, desarrolló un brazo robótico que funcionaba alimentado por la electricidad y que se podía mover en los seis ejes, el cual se llamó “el brazo de Standford”. De forma paralela, Marvin Minsky, del MIT, desarrolló un brazo robótico para la investigación naval, para exploración submarina. En particular, el brazo tenía veinte grados de libertad ya que funcionaba mediante electricidad impulsando sistemas hidráulicos. Más tarde, Scheinman continuó desarrollando brazos robóticos, creando el “*Programmable Universal Machine for Assembly*”, más conocido como PUMA.

En la actualidad, los brazos robóticos se desarrollan y diseñan para seguir la estructura física del cuerpo humano (ver figura 1.7).

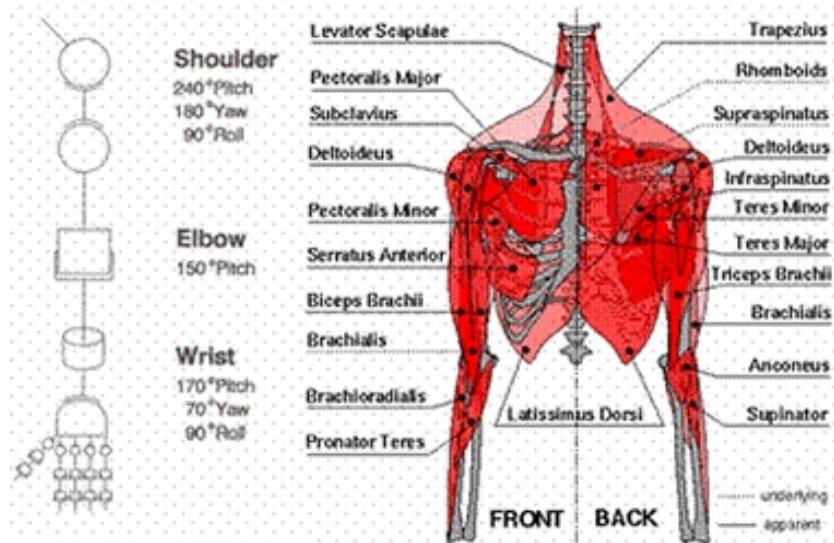


Figura 1.7: Grados de libertad de un brazo robótico y estructura del cuerpo humano [1].

De la estructura anterior, se deducen las siguientes partes:

- Articulación del hombro: dispone de tres grados de libertad que permiten subir y bajar, ir a la izquierda y derecha, y rotar sobre sí mismo.
- Articulación del codo: el codo permite extender, contraer y reorientar tanto la muñeca como la mano. Por lo general, se estima la extensión del codo en unos  $150^\circ$ .
- La muñeca: compone el último elemento del brazo robótico antes de llegar al “*end-effector*”. Es de los elementos más importantes debido a su gran capacidad de movimiento en las tres dimensiones. Sin esta articulación, el brazo robótico se asemejaría en funcionalidad a un robot pantográfico. Cada vez más, las articulaciones de la muñeca se vuelven complejas y sofisticadas. La muñeca humana, por ejemplo, puede moverse  $45^\circ$  desde el centro, pero se reduce mucho la capacidad de rotación de la misma. En la actualidad se está investigando cómo poder mejorar la relación de movimientos para permitir una mayor movilidad, pero las singularidades siguen siendo un gran problema. Por ejemplo, el robot quirúrgico da Vinci, pese a lo avanzado que pueda parecer, tiene problemas de bloqueo de las muñecas cuando se acerca a posiciones singulares.
- La mano: supone un “*end-effector*” diferenciado que define el propósito y la capacidad del brazo robótico. La mano es una herramienta capaz de realizar múltiples acciones muy variadas entre sí. Actualmente, se sigue investigando de forma activa sobre ello para intentar implementar controles sensoriales, de presión y de movimiento en los “*end-effector*” de los robots.

### 1.1.3. La actualidad

Durante los dos últimos decenios la robótica ha evolucionado de manera exponencial. Se ha trabajado de forma activa en mejorar ciertas condiciones industriales, espaciales y en el día a día de las personas. En el 2001 se puso en la ISS el brazo robótico “Canadarm2”, conocido oficialmente como “*Space Station Remote Manipulator System*” (SSRMS) (figura 1.8).



Figura 1.8: Vista exterior del Canadarm2 [7].



Figura 1.9: Rover “Spirit”, desarrollado por la NASA y desplegado en 2004 [8].

Además, en 2005, se desplegaron en Marte los rovers “Spirit” (figura 1.9) y “Opportunity” (figura 1.11).

El primero supuso un gran avance de la ingeniería, ya que crearon un robot teleoperado para enviarlo a un terreno muy hostil. Con las seis ruedas que tenía permitía una movilidad bastante elevada en terrenos muy desiguales, siendo todas ellas motrices e independientes entre sí y, en particular, las cuatro de los extremos direccionales (detalle en la figura 1.9). Además, la fisionomía de las mismas y su elevación permitía que el dispositivo se desplazara por distintos tipos de terreno de una manera óptima, utilizando un sistema de amortiguación conocido como “*rocker–bogie*”. Dicho sistema se caracteriza por no utilizar una suspensión hidráulica sino un diferencial en el centro del vehículo, garantizando así que el cuerpo del mismo siempre se encuentra con una inclinación igual a la mitad que presentan ambos “*rockers*” (ver figura 1.10).

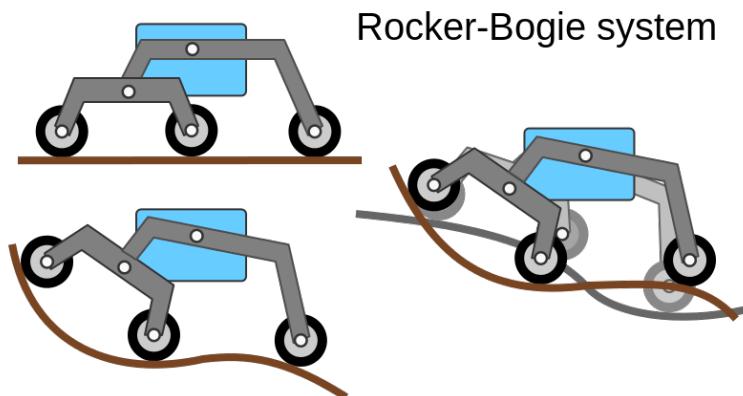


Figura 1.10: Modo de funcionamiento del sistema “*rocker–bogie*”, desarrollado por la NASA para sus rover [9].



Figura 1.11: Rover “Opportunity”, desarrollado por la NASA y desplegado en 2004 [10].

Por otra parte, se puede apreciar cómo el “Opportunity” tenía una estructura bastante similar al “Spirit” pero añadía alguna que otra mejora. La diferencia principal entre ambos era el lugar de aterrizaje (figura 1.12), ya que ambos servían para recorrer distintos puntos de la superficie marciana para recopilar datos y tomar muestras.

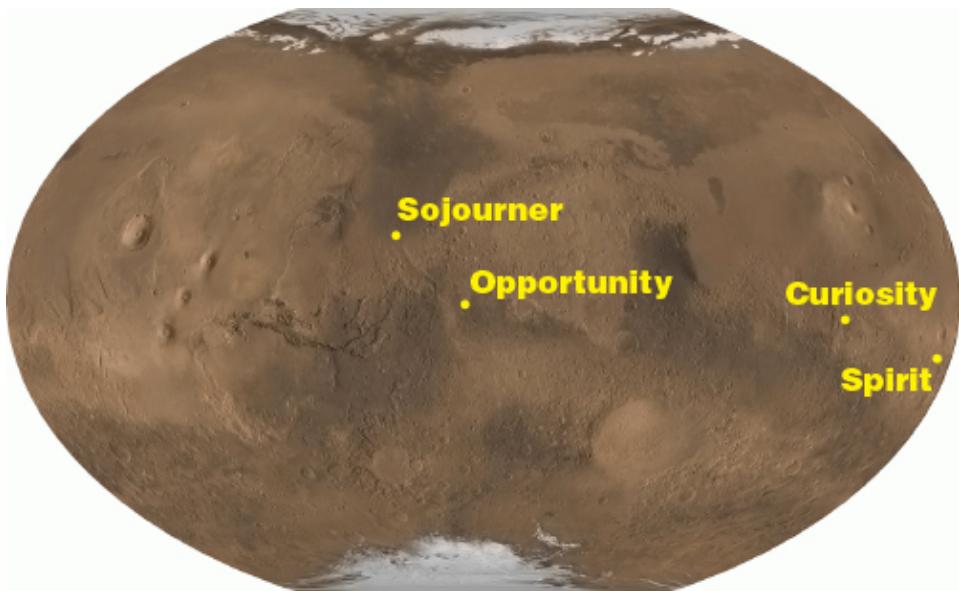


Figura 1.12: Lugares de aterrizaje de los rovers de la misión espacial a Marte [11].

La misión del “Spirit” duró 2623 soles frente a los 90 inicialmente planeados [8], y la misión del “Opportunity” duró 5352 soles frente a los también 90 inicialmente planteados [10]. Esto se traduce en 2695 días terrestres (6 años, 9 meses y 12 días) y 5498 días terrestres (15 años) respectivamente.

Por otra parte, los robots de aplicación doméstica también han ido creciendo cada vez más, naciendo en 2002 el popular Roomba (figura 1.13), de la empresa iRobot, o distintos brazos

articulados para, por ejemplo, ayudar a personas que carezcan de dichos miembros o asistir a personas mayores en sus hogares.



Figura 1.13: Robot Roomba en la estación de carga [12].

Además, la industria de los coches autónomos también ha crecido de forma exponencial, sobre todo con la llegada al mercado en 2003 de Tesla Motors y sus coches eléctricos que disponen del sistema “Autopilot” (figura 1.14), encontrándose actualmente en el nivel 2 de autonomía según la lista del SAE [13][14]. Los avances en esta industria permiten el control mecanizado completo del vehículo, tomando decisiones en tiempo real sobre la suspensión, el giro de las ruedas, el control de tracción, etc. permitiendo además compartir información e ir mejorando el mundo de la robótica y de la industria del automóvil.



Figura 1.14: Lo que ve un Tesla cuando está en conducción autónoma nivel 2 [13].

Por otro lado, se ha avanzado mucho a nivel de robots militares y humanoides. Un ejemplo de ello es la empresa *Boston Dynamics*, la cual ha desarrollado múltiples robots con un grado

de libertad bastante elevado. Dichos robots se caracterizan por una gran estabilidad y la amplia variedad de movimientos que pueden realizar: andar, correr, saltar, subir escaleras, abrir puertas, etc. Actualmente, dos robots son los principales: “*Big-Dog*” (figura 1.15) y “*Atlas*” (figura 1.16).



Figura 1.15: Robot “*Big-Dog*” de Boston Dynamics [15].

El primero tiene un amplio uso militar: debido a su forma, es muy útil para llevar cargas pesadas durante largas distancias. Además, la configuración cuadrúpeda y los avanzados sistemas *software* y *hardware* del que dispone permite que el robot sea estable incluso en condiciones bastante complicadas, como puede ser un suelo helado.



Figura 1.16: Robot “*Atlas*” de Boston Dynamics [15].

“*Atlas*”, por otra parte, es un trabajo en progreso que permitirá, en un futuro, poder utilizarlo con fines militares y domésticos. Puede llevar objetos pesados con sus brazos y moverse con mucha agilidad, haciéndolo un robot muy polivalente según se quiera utilizar.

Finalmente, el desarrollo de los brazos articulados con múltiples finalidades también ha evolucionado mucho. Desde robots industriales tales como los desarrollados por la empresa KUKA, como el “KR-1000 Titan” (figura 1.17), o el brazo “M-2000”, de FANUC hasta brazos más pequeños con motivos educacionales, como el  $\mu$ Arm, de UFACTORY (figura 1.18).



Figura 1.17: Robot “KR-1000 Titan” de KUKA [16].



Figura 1.18: Robot  $\mu$ Arm de UFACTORY [17].

Estos robots tienen múltiples propósitos: el primero, levantar y trasladar piezas muy grandes y pesadas con muchísima precisión. El segundo, disponer de un robot para poder aprender y utilizarlo para tareas como, por ejemplo, impresión 3D. Este último se desarrolló con la intención de ser accesible e intentar introducir en el mundo de la robótica a aquellos que

pudieran estar interesados, pero su alto coste impide el acceso a aquellos con una capacidad adquisitiva más baja.

## 1.2. [J] Motivaciones y objetivos del desarrollo del proyecto

Durante el primer semestre del cuarto curso de Ingeniería de Computadores, hay dos asignaturas las cuales propiciaron el desarrollo de este proyecto: robótica y sistemas empotrados.

Con la primera, se vio la potencia de los brazos robóticos y se desarrolló un estudio sobre un controlador el cual se ha hablado con anterioridad: el  $\mu$ Arm [18]. Con la segunda asignatura, se vio cómo con sistemas de aplicación específica se podían desarrollar circuitos con suficiente potencia como para poder tomar el control de otros dispositivos más grandes y complejos aplicando la lógica estudiada a lo largo de los años.

Se tomaron en cuenta los conocimientos obtenidos de las asignaturas anteriores para empezar un desarrollo que uniera esos dos campos: diseñar un brazo robótico impreso en 3D el cual estuviera gobernado por un microcontrolador en una placa de control de propósito específico. Para ello, se parte de los diseños 3D provistos en la web de UFACTORY [19] para su posterior adaptación y reutilización. En lo referente a la placa de control, el brazo original utiliza una placa Arduino Mega [20], por lo que se decidió (para dar mayor peso a la parte de ingeniería e intentar reducir costes) diseñar e implementar al completo una placa con otro microcontrolador para gobernar dicho brazo robótico.

Principalmente, este trabajo se desarrolla bajo las dos perspectivas siguientes:

- Aplicar en un proyecto de ingeniería real las competencias y técnicas que se han ido aprendiendo a lo largo de los distintos cursos del Grado de Ingeniería de Computadores (61CI).
- Construir una alternativa asequible y accesible, tanto a niveles de *Open-Source* (OS) y *Open-Hardware* (OH), de un brazo robótico de manera que cualquier persona interesada en este ámbito de la ingeniería pueda introducirse y aprender, e incluso montar el brazo por sí mismo.

Para la primera perspectiva, la forma de afrontarla y desarrollarla está detallada en el punto siguiente (1.3). Para la perspectiva de desarrollo de un producto accesible y asequible, se partió desde el abaratamiento de costes: el brazo original  $\mu$ Arm se encuentra disponible en venta por aproximadamente \$749. Dicho precio, pese a no ser especialmente elevado, impide a muchas personas el acceso a la robótica en un brazo que pretende ser educativo y útil. Por este motivo, se desarrolla este proyecto principalmente para resultar barato. Además, siguiendo con la política del brazo original, el proyecto se desarrolla bajo las premisas OS y OH, de manera que inclusive para aquellos que no puedan imprimir el brazo 3D se dispone de forma universal todos los diagramas, planos, esquemas, diseños y código fuente que se ha empleado para acabar desarrollando el brazo  $\mu$ Arm.

### 1.3. [J] Metodología

Dado que se pretende hacer un desarrollo de ingeniería completo, la metodología es un punto muy importante en este proyecto.

Primeramente, antes de hacer ningún tipo de desarrollo o implementación, se hizo un estudio del problema, y de lo que se pretendía obtener. Por una parte, se comprobó hasta qué punto podrían ser reutilizables los diseños provistos por UFACTORY en su página de GitHub. Esto permitió diseñar elementos nuevos, adaptar los recursos a lo que hay disponible, etc. Por otro lado, se estudió qué placa de control se quería utilizar. Debido a la familiaridad de los integrantes del equipo con las placas de la familia “Microchip”, se plantearon distintas alternativas:

- Controladores de gama media de la familia PIC16F.
- Controladores de gama superior de la familia PIC32F.
- Controladores digitales de la señal, de la familia dsPIC.

Se optó por utilizar los últimos mencionados, ya que disponen de un control específico matricial y matemático para poder agilizar las operaciones realizadas, de forma que los cálculos necesarios se podrían realizar íntegramente en el microcontrolador.

Además, se estudió cómo se quería plantear la comunicación con el brazo: de forma completamente autónoma o mediante un equipo auxiliar. Para evitar la complejidad extra que habría surgido de desarrollar un sistema de control completamente autónomo del brazo por sí solo, se decidió conectarlo a un equipo auxiliar externo que lo gobierne, y que el *p*Arm no funcione si no es estando conectado.

Una vez se definieron estos puntos, se pasó al diseño lógico del sistema que deberán tener tanto Sistema 1 – ordenador (S1) como S2, mediante especificación de requisitos, diagramas lógicos, diagramas físicos, diagramas de diseño, etc. Esta parte del proyecto es de las más importantes, ya que sustenta las ideas y las funcionalidades que habrán de estar presentes en el producto final. Mientras tanto, se han ido desarrollando pruebas y mecanismos de control para ir asegurando la correcta calidad del trabajo.

Finalmente, una vez completada esta parte de diseño, se pasa a la implementación real. Dada la situación del COVID-19, esta fase de implementación se ha retrasado sobremanera, impidiendo pues presentar el proyecto en el mes de julio, como estaba previsto, y teniendo que acotar los plazos de implementación a, posiblemente, un mes. En el momento de implementación, se creará la placa diseñada y se empezará la impresión de distintas piezas 3D, para comprobar su funcionamiento en conjunto e ir solucionando los posibles errores que aparezcan.

Durante este proceso, se ha ido desarrollando además de forma paralela la memoria que acompaña el proyecto, permitiendo ir actualizándola con los últimos cambios y mejoras que se han considerado de interés para aparecer descritas.

## Capítulo 2

# [REVISADO] Explicación de la estructura del proyecto

El diseño y construcción de un brazo robótico es un proceso multidisciplinar en el que se deben emplear diversas áreas del conocimiento. Desde un primer momento, este proyecto se postuló como un proyecto completo de ingeniería, y es precisamente por eso que está dividido en varios bloques, los cuales desempeñan una función clave en el desarrollo correcto del mismo.

El proyecto está dividido en tres grandes bloques: modelo matemático, elementos *hardware* y elementos *software*. Cada una de estas partes se encuentra a su vez subdividida en diferentes partes o hitos. Sin embargo, no es necesario describirlos con tanta precisión por el momento para poder comprender la estructura completa del proyecto.

Cabe destacar que, desde un punto de vista de ingeniería, a cada uno de los grandes bloques anteriormente mencionados se le puede asociar a una función dentro del proyecto:

- El modelo matemático es la parte más teórica del proyecto y su función es la de aportar una base formal y lógica que permita predecir y controlar el comportamiento del brazo robótico. Este bloque se encuentra ubicado en el apartado 4 de la memoria.
- Los elementos *hardware* del proyecto constituyen la realidad física del brazo robótico y están estrechamente relacionados con la construcción del mismo, así como con el control de los actuadores y demás componentes físicos presentes en el brazo robótico. Este bloque se encuentra ubicado en el apartado 5 de la memoria.
- Los elementos *software* del proyecto constituyen el principal mecanismo para implementar el modelo matemático y la lógica de funcionamiento del sistema completo mediante la programación de los elementos *hardware* y de los sistemas que necesitan comunicarse con los mismos. Este bloque se encuentra ubicado en el apartado 6 de la memoria.

En cada uno de los bloques de desarrollo anteriores, ya sean *hardware* o *software* y requieran construcción física o implementación mediante programación, se contempla la realización de pruebas de funcionamiento así como las revisiones pertinentes.

Es importante remarcar que, debido a la complejidad del sistema, el mismo está dividido en dos subsistemas que aglutinan funcionalidades vitales para el correcto funcionamiento del manipulador robótico:

- S1: está formado por la interfaz de usuario que se ejecuta sobre un computador de propósito general; esta interfaz es gráfica y le permite controlar los movimientos del brazo robótico, así como visualizar el estado de los parámetros del mismo. Este subsistema es esencialmente un elemento software y está descrito en el apartado 6.1 de la memoria.
- S2: está formado por la estructura física del manipulador, los actuadores y la placa de circuito impreso de control. Este subsistema combina elementos *hardware* y *software*, así como conceptos del modelo matemático. Los elementos *software* se describen en el apartado 6.2 de la memoria, mientras que los elementos *hardware* se describen en el apartado 5 de la memoria.

A continuación, se describen de forma detallada todos los bloques descritos anteriormente y a su vez, se mencionan las principales subdivisiones de cada uno de ellos.

## 2.1. Matemáticas

Los modelos matemáticos aplicados a proyectos de manipuladores robóticos son usados principalmente para realizar cálculos relacionados con los aspectos cinemáticos y dinámicos de los mismos.

Los aspectos cinemáticos de un manipulador robótico describen cómo es el movimiento y las trayectorias del mismo sin tener en cuenta las fuerzas que lo afectan, mientras que los aspectos dinámicos describen cómo se ve afectado dicho movimiento en función de las fuerzas que actúan sobre él.

Ambos aspectos anteriormente mencionados deben de ser descritos mediante un modelo matemático que permita realizar cálculos sobre los movimientos del manipulador.

En este proyecto, se ha llevado a cabo únicamente el modelo cinemático, dado que debido a las características físicas del prototipo a construir, es decir, velocidades de desplazamiento, peso de las articulaciones o masa máxima de carga; se ha concluido que el modelo dinámico no aportaría demasiada información útil para llevar a cabo el control del manipulador. Cabe destacar que el modelo dinámico suele presentar una complejidad mucho más elevada que el modelo cinemático en términos matemáticos y por ello se ha desecharido la posibilidad de llevarlo a cabo.

Desde un punto de vista técnico, el modelo cinemático de un manipulador robótico expresa cuál es la posición del extremo del mismo con respecto al tiempo y en función de la posición de las articulaciones del mismo. Normalmente, los brazos robóticos se pueden describir matemáticamente mediante el concepto de cadena cinemática:

Tal y como se puede apreciar en la figura 2.1, las articulaciones pueden rotar y permiten la movilidad de cada uno de los segmentos del manipulador. Dado que estas articulaciones

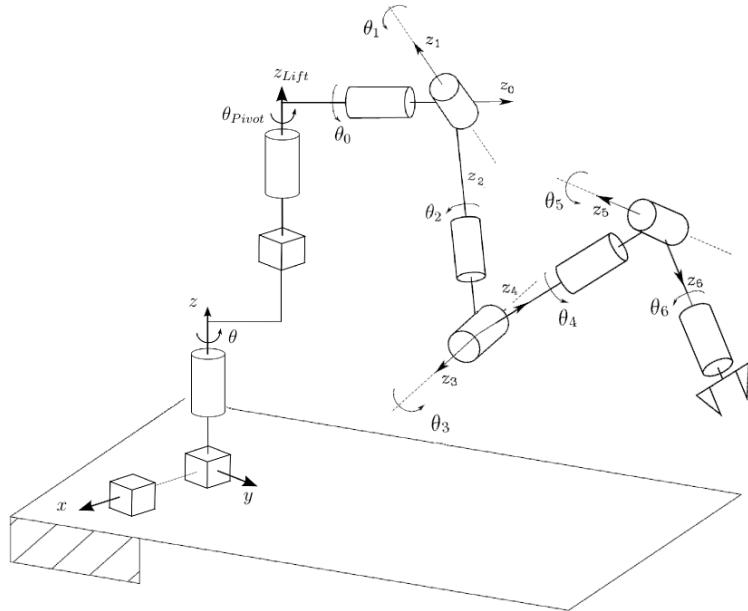


Figura 2.1: Ejemplo de cadena cinemática [21].

rotan, su posición se expresa numéricamente mediante unidades angulares. El concepto de cadena cinemática hace referencia a que, dado que cada una de las articulaciones esta unida a la siguiente mediante un segmento, se genera una cadena de movimientos en la que la posición espacial de cada una de las articulaciones se ve afectada por la posición angular de las anteriores.

Aplicando este principio, el modelo cinemático expresa matemáticamente la posición cartesiana del extremo del robot en función de las coordenadas angulares de las articulaciones. Existen pues dos perspectivas del modelo cinemático:

- El modelo de cinemática directa expresa la posición espacial del extremo del manipulador en función de las coordenadas angulares de las articulaciones.
- El modelo de cinemática inversa expresa las coordenadas angulares de las articulaciones en función de las coordenadas cartesianas del extremo del manipulador.

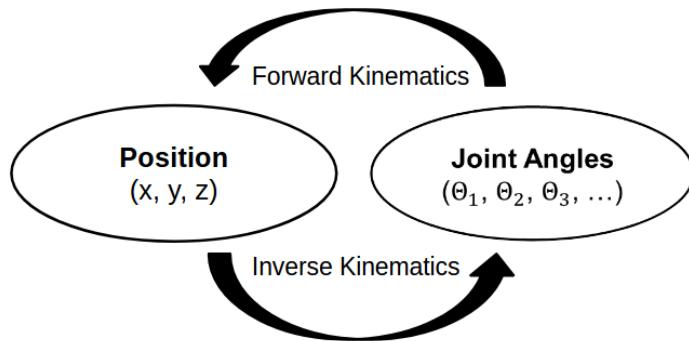


Figura 2.2: Diagrama del modelo cinemático [22].

En conclusión, el modelo matemático conforma la base teórica y formal que permite realizar el estudio de los movimientos del manipulador y es por ello que representa un bloque crucial dentro del proyecto.

## 2.2. *Hardware*

Los elementos *hardware* conforman la implementación física del manipulador y de todos los componentes empleados para controlarlo.

En términos generales, el *hardware* usado en el proyecto se descompone en diferentes elementos:

- Impresión 3D de la estructura física del manipulador.
- Motores empleados en el manipulador.
- Desarrollo de la placa de circuito impreso de control y microcontrolador empleado.
- Comunicaciones entre los diferentes subsistemas.

En primer lugar, la impresión 3D es la tecnología seleccionada para la fabricación de la estructura física del manipulador debido a su bajo coste y accesibilidad. Esta parte del proyecto se centra en llevar a cabo la fabricación y construcción de la estructura física del manipulador, así como su ensamblado y testeo. Este apartado se ubica en el apartado 5.1 de la memoria.

En segundo lugar, la elección de los motores que dotan de movilidad a la estructura es una decisión crucial y que depende principalmente de cuales sean las características físicas del manipulador, así como de las tareas que se quieran realizar con el mismo. Existen numerosas opciones en cuanto a motores, por ejemplo, motores DC, servomotores, motores paso a paso, etc. Este apartado se ubica en el apartado 5.5 de la memoria.

En tercer lugar, el desarrollo de la PCB de control y elección del microcontrolador representan la parte más importante dentro del bloque hardware del proyecto. El objetivo principal de esta parte del proyecto es llevar a cabo el diseño y construcción de una PCB personalizada, adaptada especialmente a los actuadores y microcontrolador usados para llevar a cabo el control del movimiento del manipulador. Se considera que esta PCB representa uno de los elementos hardware esenciales para el correcto desarrollo del proyecto. Este apartado se ubica en el apartado 5.3 de la memoria.

En último lugar, el diseño e implementación de los canales de comunicación y protocolos necesarios para comunicar los dos subsistemas principales requiere desarrollo hardware y software de forma equitativa, además, también representa uno de los elementos cruciales del proyecto. Este apartado se ubica en el apartado 5.4 de la memoria.

## 2.3. *Software*

Los elementos *software* del proyecto abordan los siguientes aspectos:

- Desarrollo de la aplicación de control del brazo robótico, implementada mediante una interfaz gráfica de usuario para garantizar su accesibilidad y facilidad de uso. Esta implementación se lleva a cabo en S1.
- Programación del microcontrolador e implementación del modelo matemático en la práctica con el objetivo de controlar los movimientos del brazo robótico. Esta implementación se lleva a cabo en S2.

En primer lugar, mediante el desarrollo de la aplicación de usuario se busca ofrecer una forma de controlar los movimientos del robot de forma fácil y accesible, para ello se ha desarrollado una interfaz de usuario que se ejecuta en un ordenador auxiliar. Desde esta aplicación el usuario puede controlar los movimientos del robot, además de monitorizar el estado del mismo. Las órdenes dadas por el usuario son enviadas al microcontrolador para su ejecución mediante los canales de comunicación mencionados anteriormente. Este desarrollo se ha llevado a cabo mediante el lenguaje de programación Python. Este apartado se ubica en el apartado 6.1 de la memoria.

En segundo lugar, la programación del microcontrolador representa una parte esencial del proyecto, ya que toda la lógica de funcionamiento y control de los actuadores del brazo robótico se lleva a cabo en el mismo. Es por ello que la labor principal del microcontrolador es orquestar el funcionamiento de los actuadores, así como de realizar el computo necesario para transformar las órdenes del usuario en movimientos consecuentes del brazo robótico. La programación del microcontrolador se ha llevado a cabo mediante el lenguaje C. Este apartado se ubica en el apartado 6.2 de la memoria.

# Capítulo 3

## Diagramas, requisitos y diseño

Una parte importante de un proyecto integral de ingeniería es la elicitation de requisitos y la creación de diagramas que representen el sistema de manera abstracta en base a dichos requisitos.

El sistema de gobierno del p-Arm esta compuesto de dos subsistema al ser necesaria tanto una placa de control como un ordenador auxiliar desde el cual un operario humano pueda interactuar con el brazo. El software del sistema de control del ordenador será representado mediante un diagrama de clases mientras que el software que ira cargado en la placa de control será representado por un diagrama de bloques general y varios diagramas de estados que detallarán el comportamiento del sistema.

Para realizar dichos diagramas se ha empleado Papyrus, una herramienta de edición gráfica para realización de diagramas. Para modelizar los diagramas del sistema de control que ira en el ordenador auxiliar se ha empleado el estándar UML2 definido por la OMG, por otro lado, para realizar los diagramas del software que será cargado en la placa de control se ha empleado el estándar SysUML 1.4 ya que permite mejor representación del sistema empotrado.

A continuación se pueden observar los requisitos que el equipo de desarrollo ha elicited además de una explicación de los diagramas que representan los dos subsistemas que componen el proyecto,

### 3.1. Requisitos

Los requisitos se incluyen como anexo dada su extensión (ver anexo B).

### 3.2. Diagramas y diseño

En base a los anteriores requisitos el grupo de desarrollo ha generado los siguientes diagramas para el software de la placa de control.

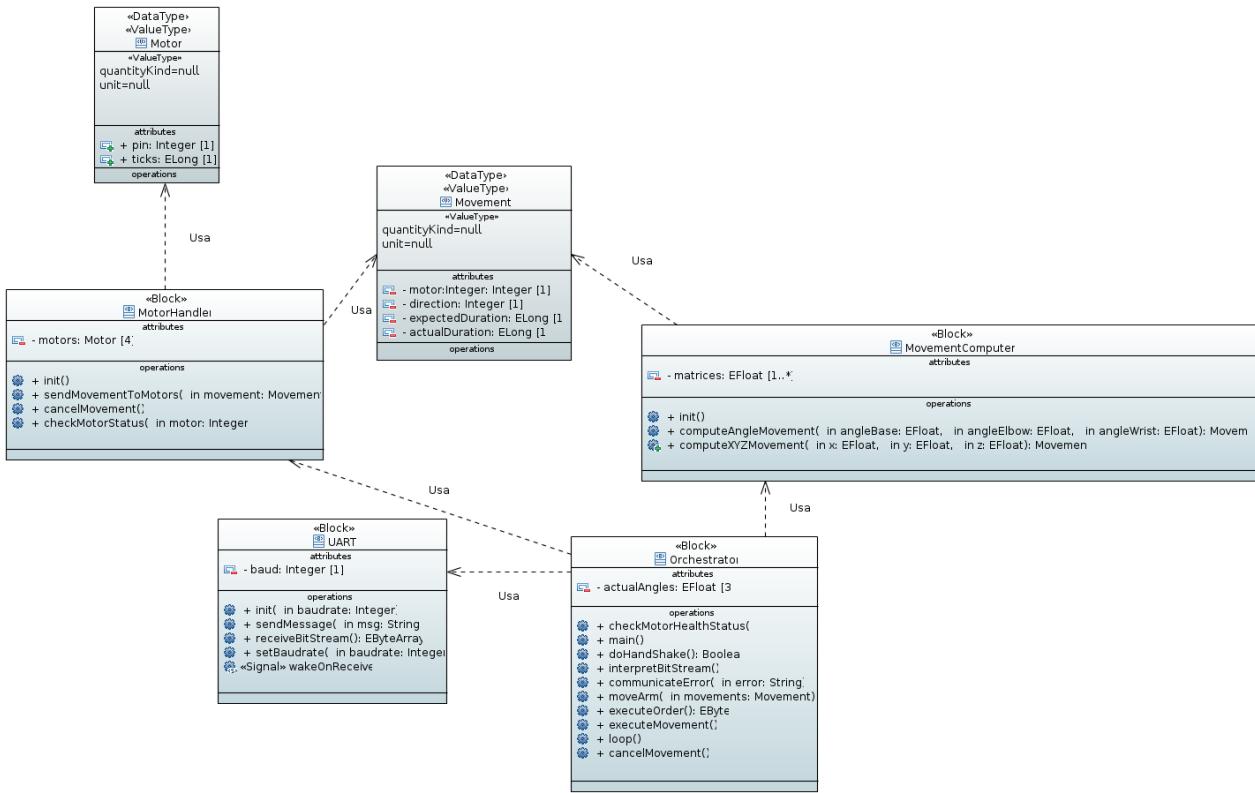


Figura 3.1: Diagrama de bloques del S2

En el diagrama 3.1 se pueden observar los bloques que componen el S2 además de dos tipos de datos los cuales han sido creados para facilitar el control de los motores del brazo.

A continuación se explican cada uno de los bloques:

- **MotorHandler**: este bloque es capaz de controlar los motores de manera directa empleando el tipo de dato “Movement” enviando la señal necesaria para realizar el movimiento requerido. Además permite verificar el estado de los motores y cancelar los movimientos si esto fuese necesario.
- **UART**: este bloque es el encargado de la comunicación asíncrona entre el S1 y el S2. Controla el ratio de baudios de la comunicación y realiza la transmisión y la recepción de información hasta y desde el S1. A través de este bloque se reciben las órdenes procedentes del S1 y se envían los errores y la posición del brazo al S1 desde el S2.
- **Orchestrator**: encargado de coordinar los demás bloques. En él se encuentra la lógica principal del S2. Algunas de sus funciones mas importantes son interpretar el flujo de bits que llega desde el S1 para obtener una orden concreta; ordenar el movimiento del brazo empleando los demás bloques o hacer el “handshake” inicial entre el S1 y el S2. Posteriormente se entrará mas en detalle en el comportamiento de este bloque al analizar los diagramas de estados.
- **MovementComputer**: se encarga de computar el movimiento que se tendrá que comunicar a los motores. Para ello deberá obtener las posiciones deseadas gracias al bloque UART

y al “Orchestrator”.

A continuación se explican las dos estructuras de datos que se aprecian en el diagrama 3.1

- Motor: Este tipo de dato es empleado por el “MotorHandler” para saber a que pin debe mandar la señal “PWM” que gobierna los motores y durante cuantos ticks deberá estar activa dicha señal
- Movement: El “MovementComputer” genera un array de 3 posiciones de este tipo de dato, uno por cada motor de giro del brazo. El atributo “moto” guarda un integer que representa uno de los motores del brazo; “direction” sirve para conocer la dirección de giro de dicho motor; “expectedDuration” guarda la duración

A continuación se explican los diagramas de estados de cada uno de los métodos que aparecen en el diagrama de bloques general.

En el caso del “Orchestrator” tenemos los siguientes diagramas.

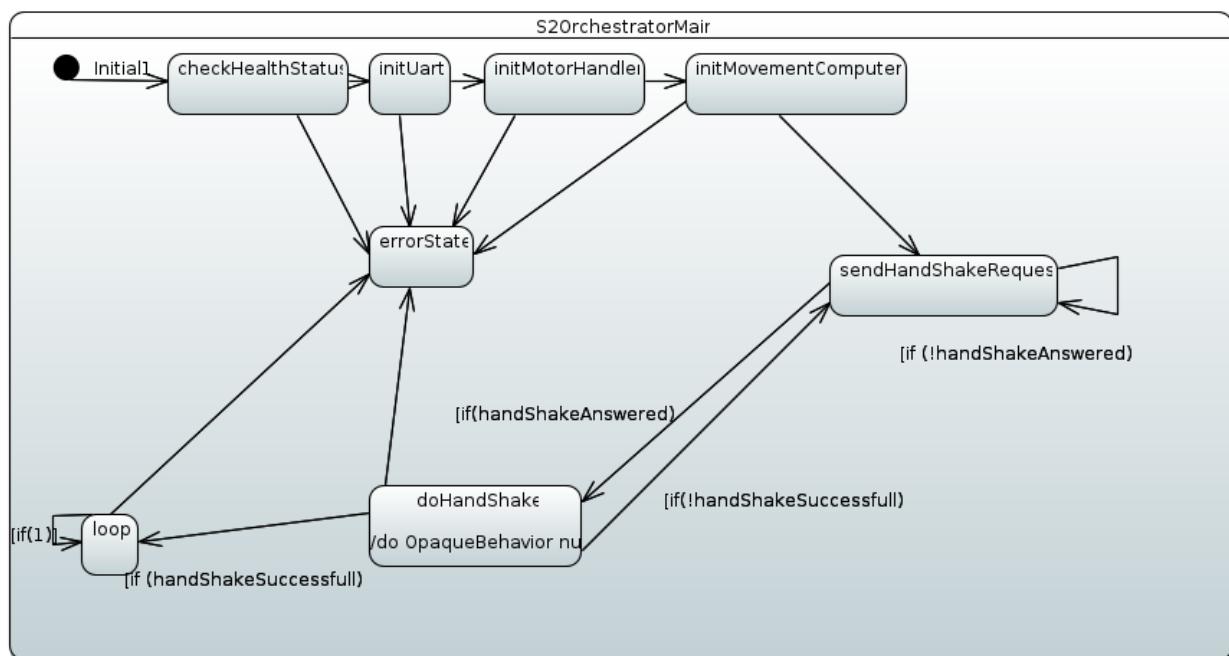


Figura 3.2: Diagrama de estados del método `main()` del *orchestrator*.

Este método solo se ejecutará una vez, en cuanto el sistema se ponga en marcha.

- **checkHealthStatus:** Se verifica la situación de los componentes del brazo robótico para confirmar que todos están en un estado adecuado para el funcionamiento.
- **initUart:** Se inicializa la UART definiendo un ratio de baudios concreto.
- **initMotorHandler:** Se inicializa el controlador de los motores.

- **initMovementComputer**: Se inicializa el computador de movimientos.
- **sendHandShakeRequest** : Se manda una petición de handshake para verificar si hay algún ordenador conectado. Si se detecta alguno se pasa al siguiente estado, si no, se mantiene en ese estado mandando requests.
- **doHandShake**: Si en el estado anterior se detecta un ordenador se pasa a este estado. Se realiza una serie de intercambios de información para verificar que el ordenador conectado es adecuado para el control del brazo.
- **loop**: Se pasa al bucle de funcionamiento si el handshake ha sido correcto.
- **errorState**: Estado de error al que se llega si en alguno de los estados ocurre algún problema inesperado.

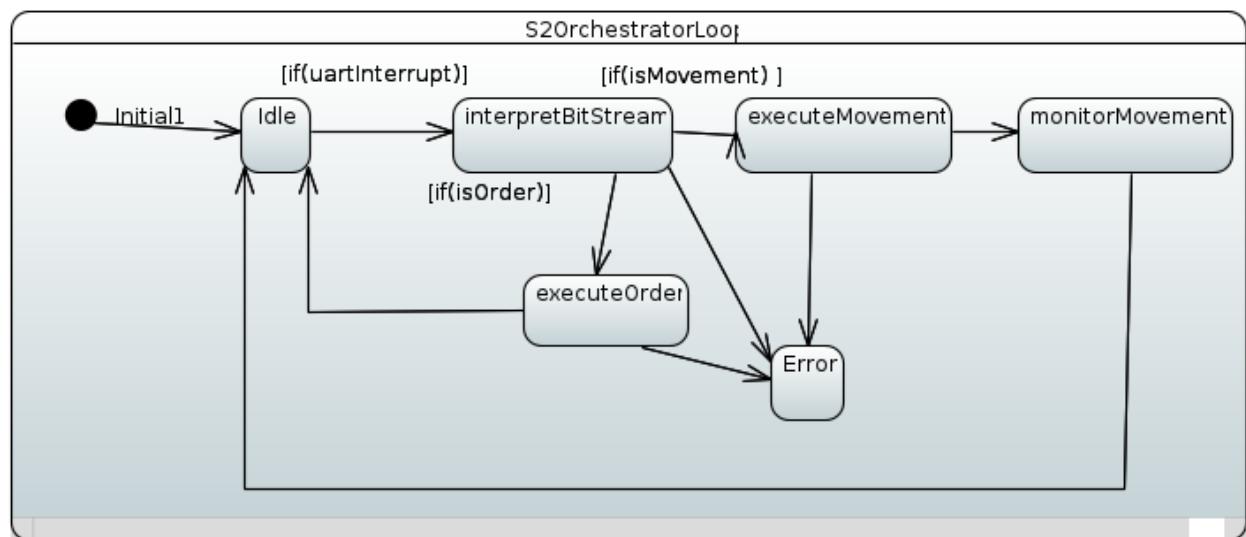


Figura 3.3: Diagrama de estados del método `loop()` del *orchestrator*.

Este método es el bucle principal del brazo robotico. Tras ejecutar `main()` el sistema entrará en este bucle y no saldrá hasta que se apaga.

- **Idle**: el brazo se encuentra ocioso y a la espera de una orden desde el S1
- **interpretBitStream**: tras una interrupción de la UART el S2 entiende que hay una orden o movimiento procedentes del S1 y se avanza a este estado. El bitStream es interpretado para saber si es una orden o un movimiento.
- **executeMovement**: si tras interpretar el bitStream resulta que es un movimiento, el sistema avanza a este estado y se ponen en marcha los demás bloques para poder generar un movimiento en los motores en base a la posición recibida desde S1
- **executeOrder**: si tras interpretar el bitStream resulta que es una orden, el sistema avanza a este estado y se ponen en marcha los bloques necesarios para ejecutar dicha orden.

- **monitorMovement**: tras empezar a ejecutar un movimiento el brazo empieza a monitorearlo para poder determinar cuando se ha terminado o, si es cancelado, actualizar la posición en la que se ha quedado el brazo.
- **errorState**: Estado de error al que se llega si en alguno de los estados ocurre algún problema inesperado.

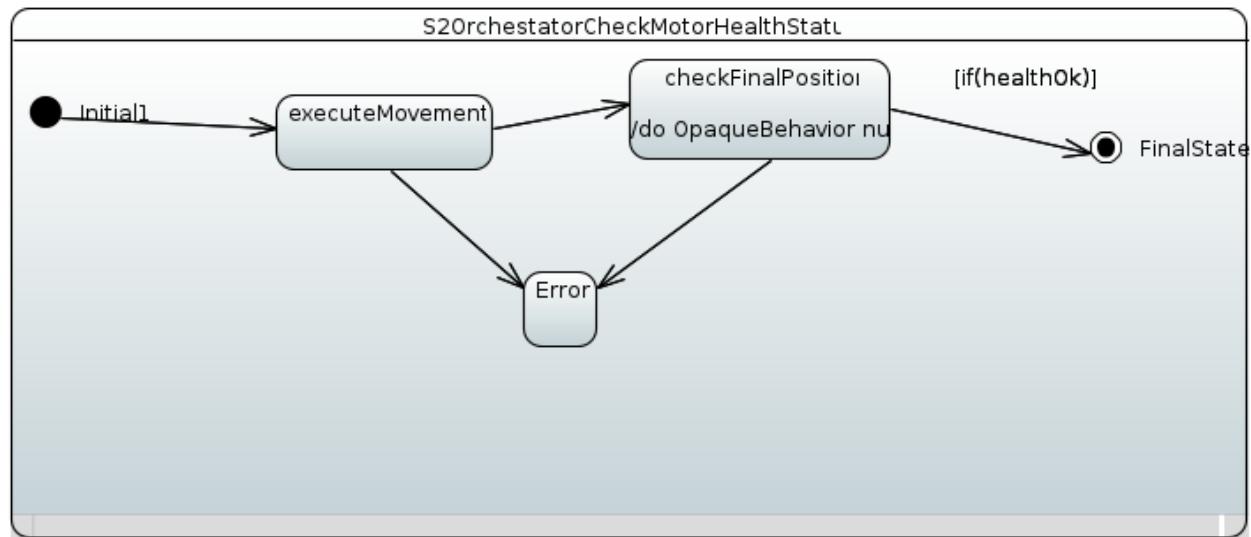


Figura 3.4: Diagrama de estados del método `CheckMotorHealthStatus()` del *orchestrator*.

Este método comprueba el estado de los motores para asegurar que estos tienen un funcionamiento correcto antes de recibir cualquier orden de movimiento.

- **executeMovement**: se ejecuta un movimiento a una posición en la que todos los fines de carrera sean activados.
- **interpretBitStream**: se verifica que todos los fines de carrera han sido alcanzados pudiendo concluir que el brazo es capaz de mover todos sus motores.

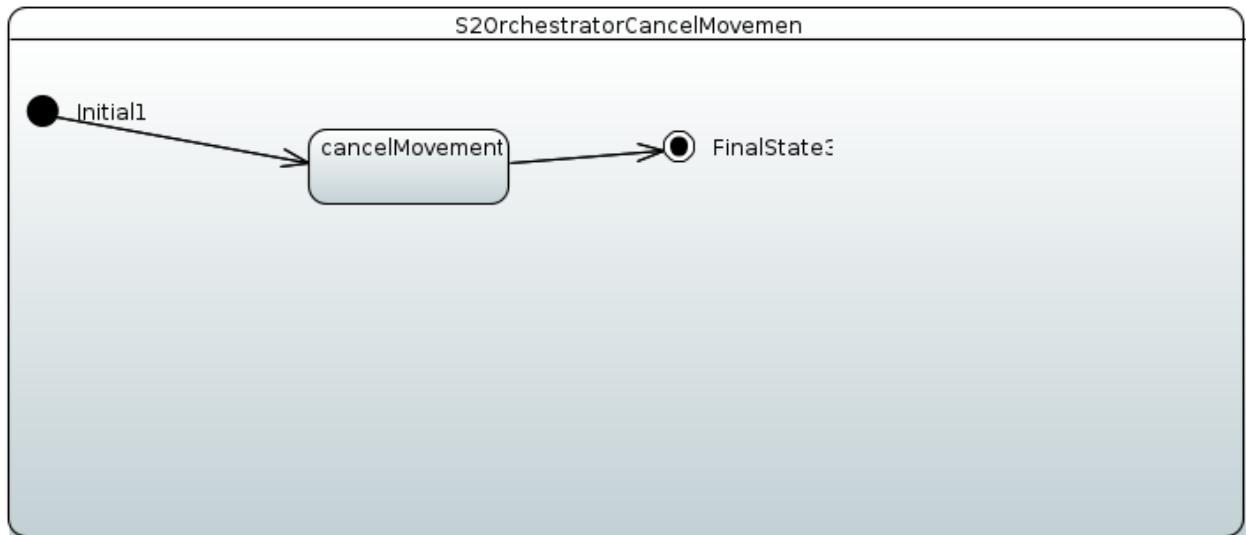


Figura 3.5: Diagrama de estados del método `CancelMovement()` del *orchestrator*.

Este método finaliza un movimiento que se este realizando.

- `cancelMovement`: se cancela el movimiento y se guarda la posición actual del brazo..

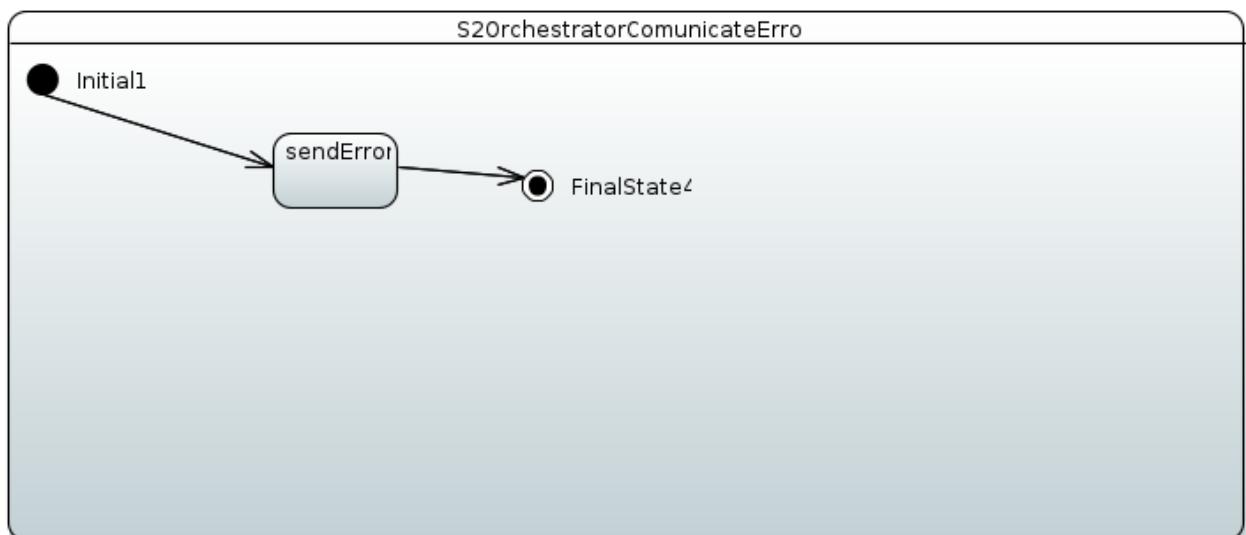


Figura 3.6: Diagrama de estados del método `CommunicateError()` del *orchestrator*.

Este método comunica un error al S1.

- `cancelMovement`: se envía un bitStream que representa un error ocurrido en el S2

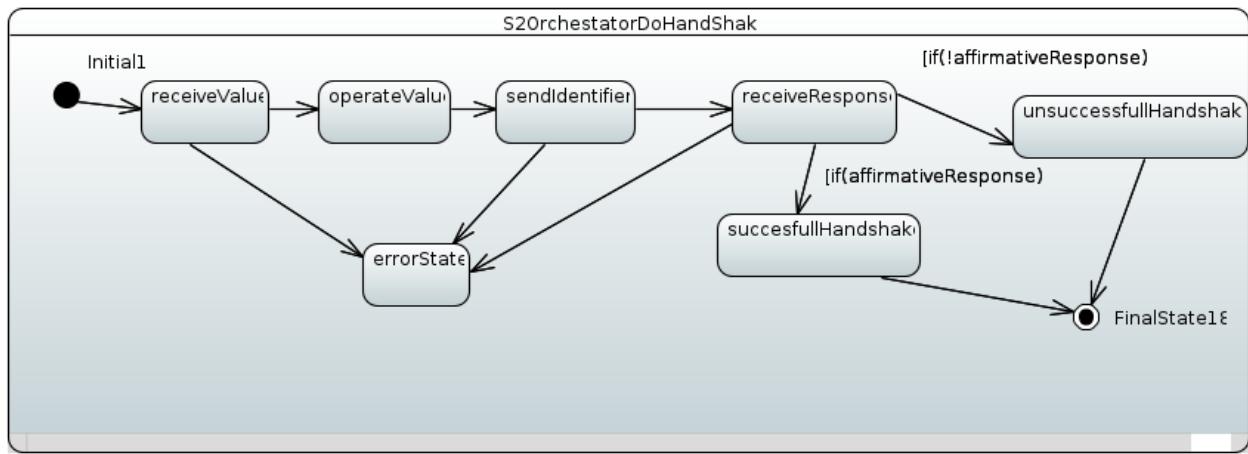


Figura 3.7: Diagrama de estados del método `DoHandShake()` del *orchestrator*.

Este método es el encargado de autenticar a los dispositivos entre si y configurar un canal para su posterior comunicación

- **receiveValue:** se realizan los procedimientos necesarios para recibir un valor desde S1 a través de la UART.
- **operateValue:** se realiza una operación matemática con el valor recibido para generar de esta manera un identificador.
- **sendIdentifier:** se envía dicho identificador de vuelta a S1.
- **receiveResponse:** se recibe la respuesta de acS1 para saber si el **hand-shake!** (**hand-shake!**) ha sido realizado con éxito.
- **successfulHandshake:** en caso de que en el estado `receiveResponse` se haya recibido una respuesta afirmativa se pasa a este estado que representa que el dispositivo que conforma S1 y S2 han conseguido autenticarse entre si.
- **unsuccessfulHandshake:** en caso de que en el estado `receiveResponse` se haya recibido una respuesta negativa se pasa a este estado que representa que el dispositivo que conforma S1 y S2 no han conseguido autenticarse entre si.
- **errorState:** Estado de error al que se llega si en alguno de los estados ocurre algún problema inesperado.

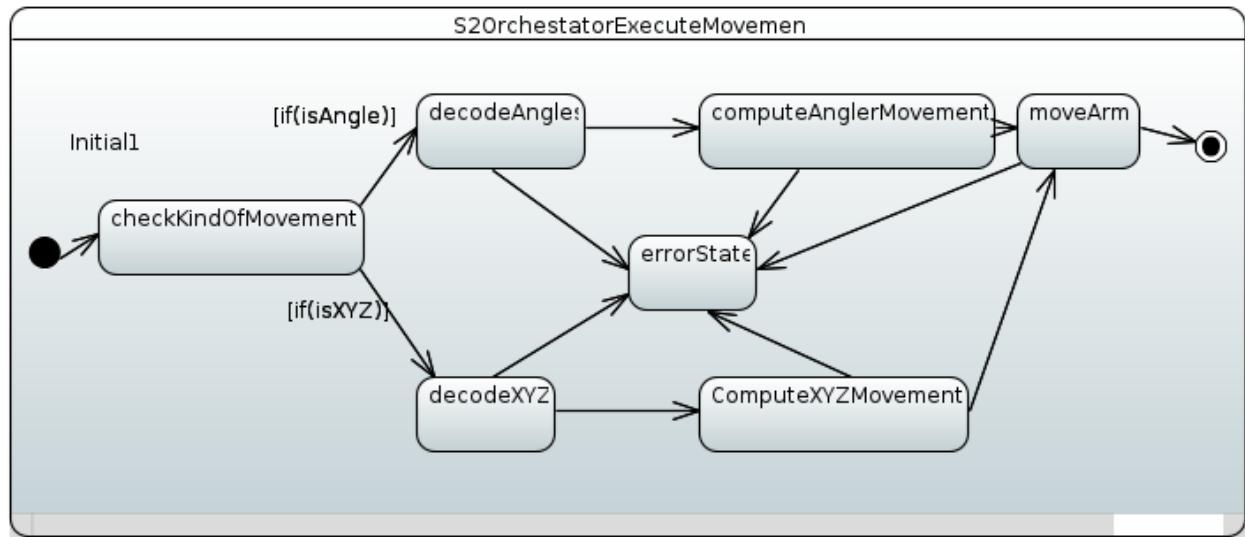


Figura 3.8: Diagrama de estados del método `ExecuteMovement()` del *orchestrator*.

Este método es el encargado de, una vez recibida la trama de bits que representa movimiento desde el S1, decidir si el movimiento ha sido representado como ángulos o posiciones cartesianas y posteriormente ordenar las operaciones necesarias para que se generen las señales PWM que moveran los motores.

- **checkKindOfMovement:** Se verifica si el movimiento ha sido transmitido como una posición cartesiana o como unos ángulos destino para los motores y se procede en consecuencia.
- **decodeAngles:** En caso de que fueran ángulos, se transita a este estado. Se interpreta la trama de bits y se obtiene el valor numérico de los ángulos.
- **computeAngleMovement:** Se realizan comprobaciones para verificar que los ángulos están dentro de los límites del brazo y se procede a generar el array de movimientos que se necesitan hacer para conseguir llegar desde la posición actual a la posición destino.
- **decodeXYZ:** En caso de que fueran posiciones cartesianas se transita a este estado. Se interpreta la trama de bits y se obtienen las coordenadas en centímetros.
- **computeXYZMovements:** Para simplificar los cálculos matemáticos posteriores las posiciones cartesianas se convierten en ángulos. Se verifica si los ángulos están dentro de los límites del brazo y se procede a generar el *array* de movimientos que se necesitan hacer para conseguir llegar desde la posición actual a la posición destino.
- **moveArm:** Se envían los movimientos a los motores.
- **errorState:** Estado de error al que se llega si en alguno de los estados ocurre algún problema inesperado.

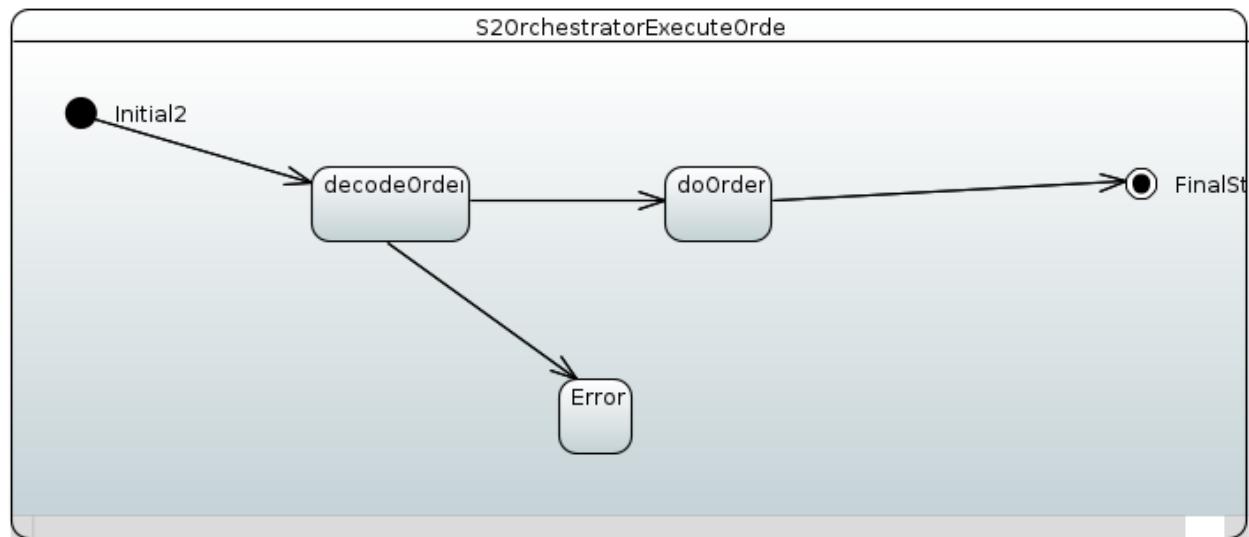


Figura 3.9: Diagrama de estados del método `ExecuteOrder()` del *orchestrator*.

Este método es el encargado de, una vez recibida la trama de bits que representa una orden distinta de realizar un movimiento desde el S1, decodificar dicha orden y realizarla.

- **decodeOrder:** Se interpreta la trama de bits para obtener la orden proveniente desde S1.
- **doOrder:** Se ejecuta la orden obtenida en el estado anterior.
- **Error:** Estado de error al que se llega si en alguno de los estados ocurre algún problema inesperado.

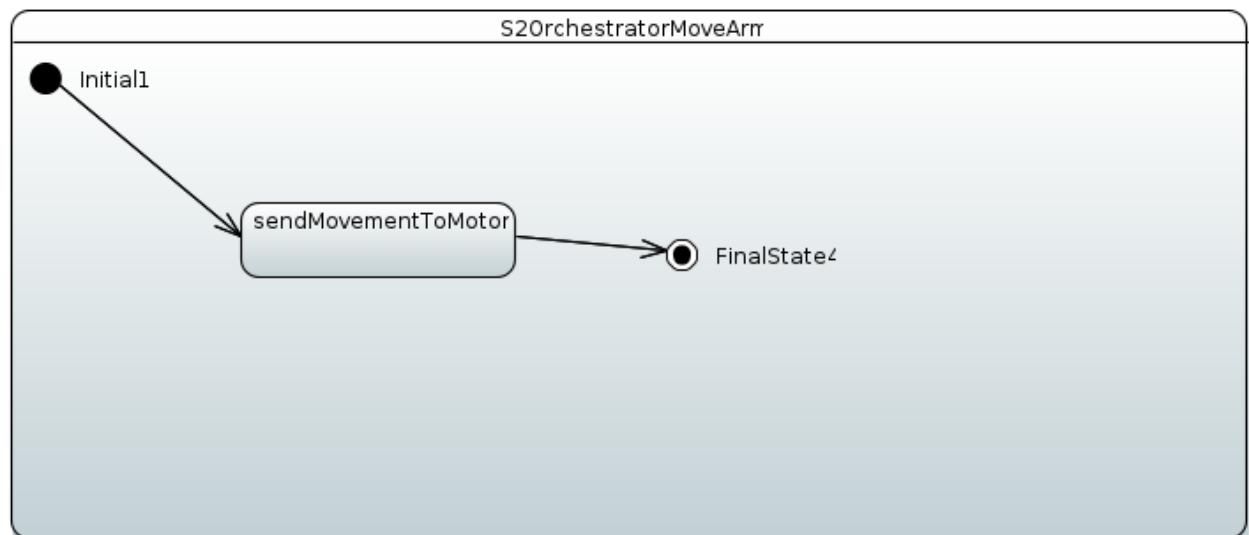


Figura 3.10: Diagrama de estados del método `moveArm()` del *orchestrator*.

Este método es el encargado de mandar los movimientos a los motores una vez estos se hayan computado.

- **sendMovementToMotors:** se mandan los movimientos necesarios a los motores.

En el caso del bloque “UART” tenemos los siguientes diagramas.

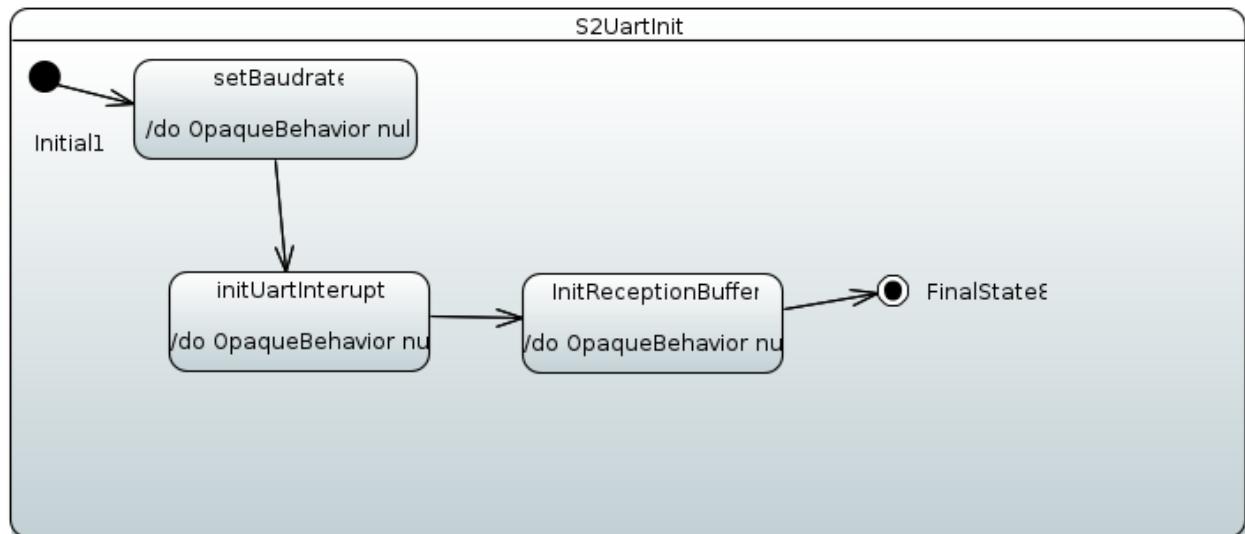


Figura 3.11: Diagrama de estados del método `uartInit()` del *UART*.

Se ejecuta este método al inicio de la comunicación a través de la UART para configurar la transmisión de datos.

- **setBaudrate:** Se establece el ratio de baudios para la transmisión asíncrona
- **initUartInterrupt:** Se configura el registro de interrupciones de tal manera que la UART sea capaz de generar una interrupción en el sistema con el objetivo de poder saber cuando se ha recibido una nueva trama de bits.
- **initReceptionBuffer:** Se inicializa el *buffer* de recepción.

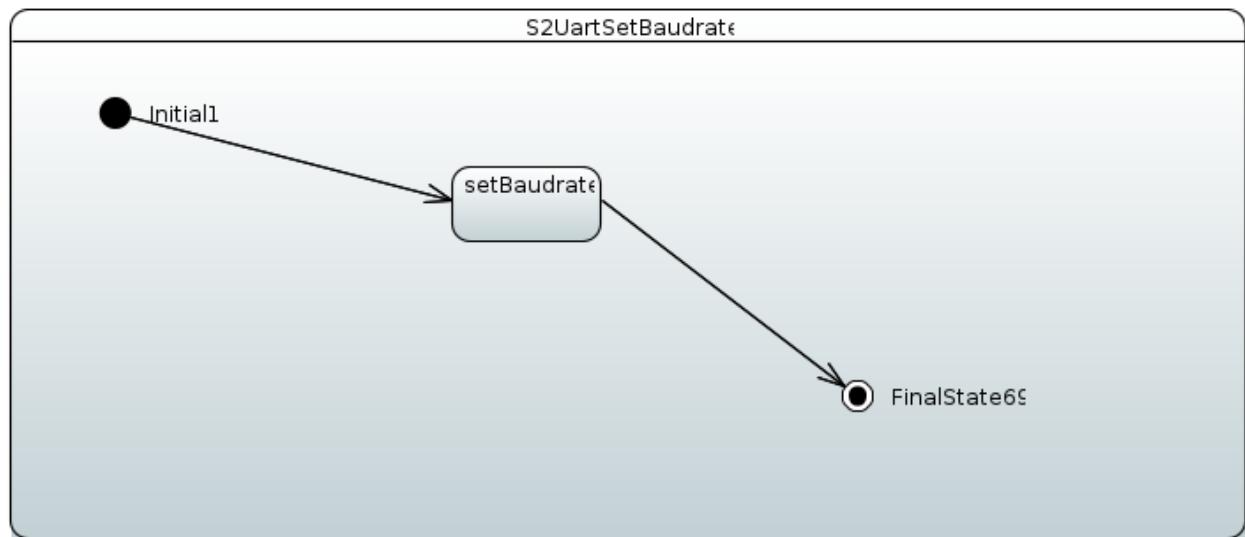


Figura 3.12: Diagrama de estados del método `setBaudrate()` del *UART*.

Se configuran los registros necesarios para obtener un ratio de baudios adecuados para la comunicación

- `setBaudrate`: Se realiza la configuración

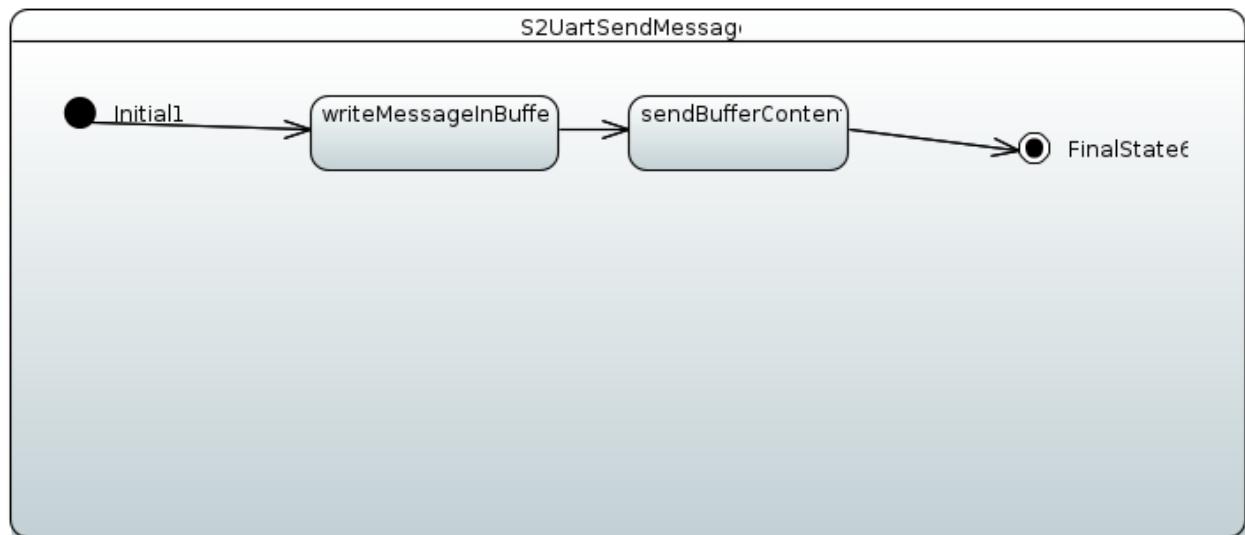


Figura 3.13: Diagrama de estados del método `sendMessage()` del *UART*.

Se escribe un mensaje en el *buffer* de envío y este es posteriormente enviado.

- `writeMessageInBuffer`: Se escribe el mensaje en el *buffer* de salida de la UART.
- `sendBufferContent`: Se envía el contenido del buffer al S1

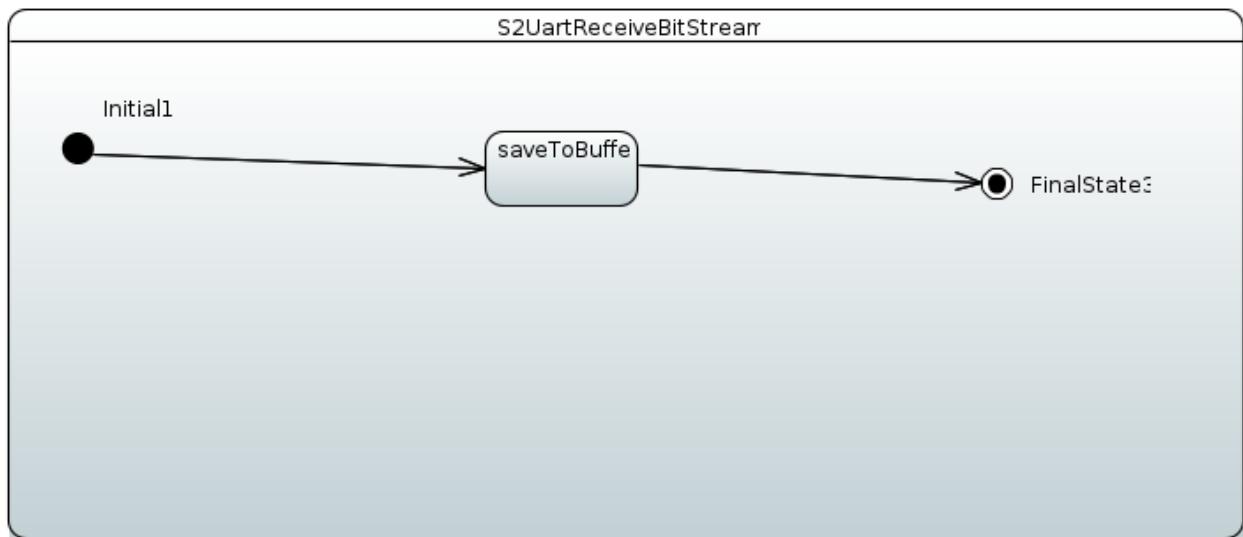


Figura 3.14: Diagrama de estados del método `receiveBitStream()` del *UART*.

Se guarda un mensaje en el *buffer* de recepción.

- **saveToBuffer:** Se escribe el mensaje en el *buffer* de entrada de la UART.

En el caso del bloque “MotorHandler” tenemos los siguientes diagramas.

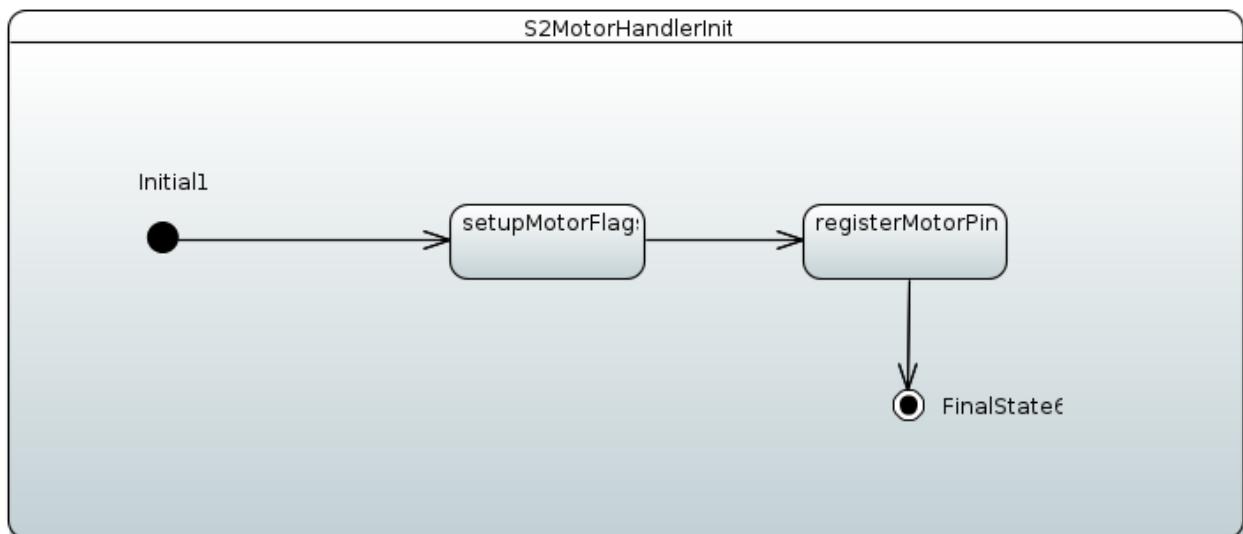


Figura 3.15: Diagrama de estados del método `init()` del *motorHandler*.

Se inicializan los *flags* y se registran los pines a los que están conectados los motores

- **setupMotorFlags:** Se establecen los flags de los motores.

- **registerMotorPin:** Se registran los pines físicos a los que están conectados los motores con el objetivo de saber donde se deben enviar las señales PWM.

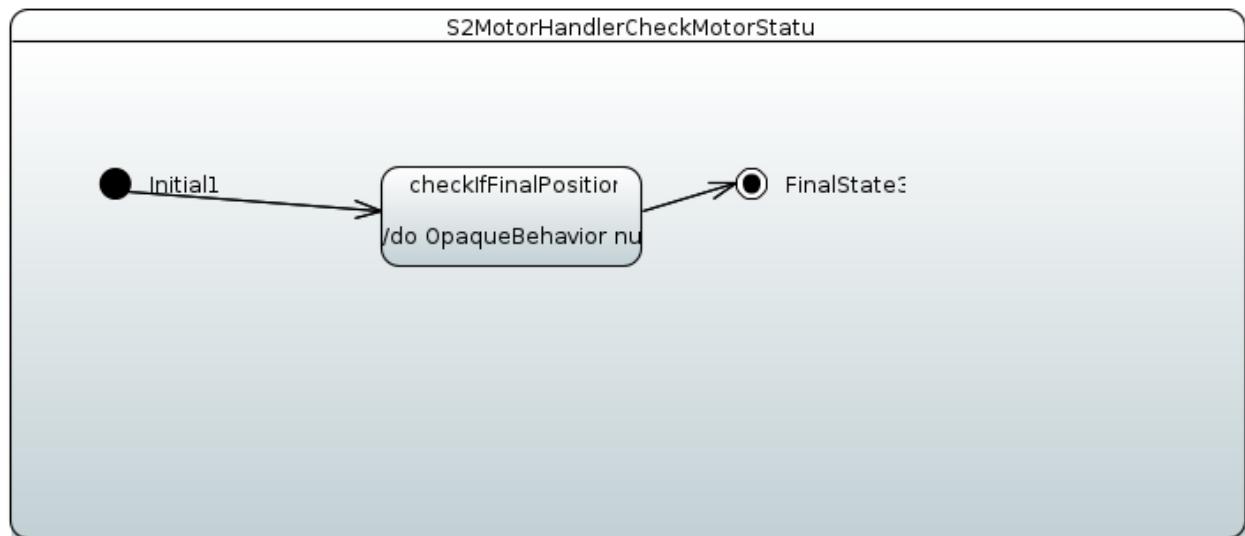


Figura 3.16: Diagrama de estados del método `checkMotorStatus()` del *motorHandler*.

Este método sirve para asegurar que los motores están en buenas condiciones de funcionamiento

- **checkIfFinalPosition:** Se envía a los motores a una posición en la que se sabe que debería estar en contacto con algún fin de carrera y posteriormente se verifica que dichos fines de carrera están activados. De esta manera se asegura que los motores pueden girar.

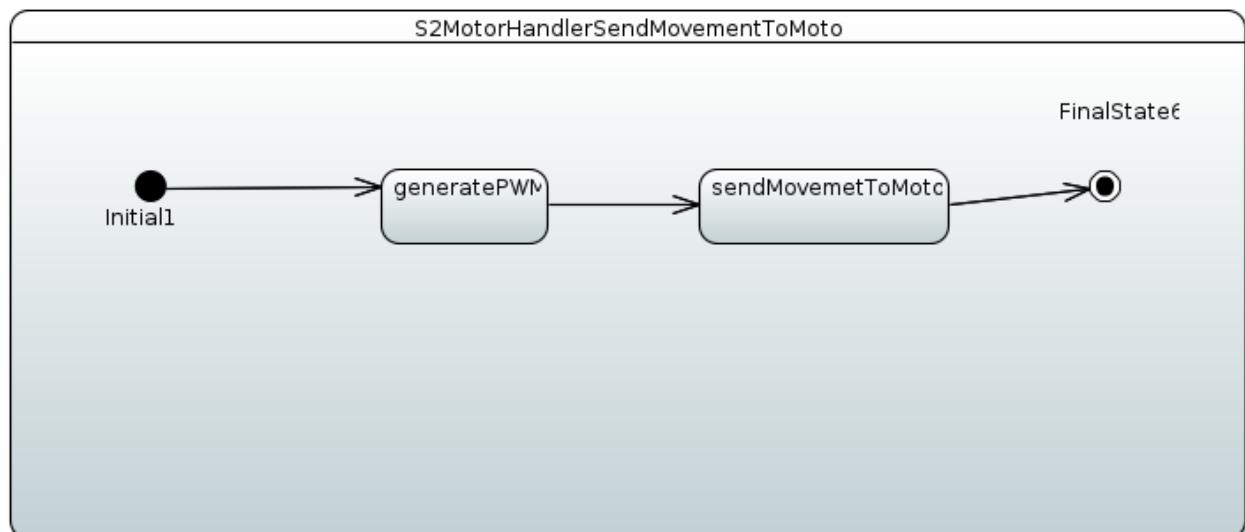


Figura 3.17: Diagrama de estados del método `sendMovementToMotors()` del *motorHandler*.

Se genera una señal PWM y se envia al motor correspondiente

- **generatePWM:** En base al *array* de movimientos que se obtiene a través del *orchestator* y del *movementComputer* se generan las señales PWM necesarias para poder realizarlos.
- **sendMovementToMotor:** Se envía la señal PWM al motor.

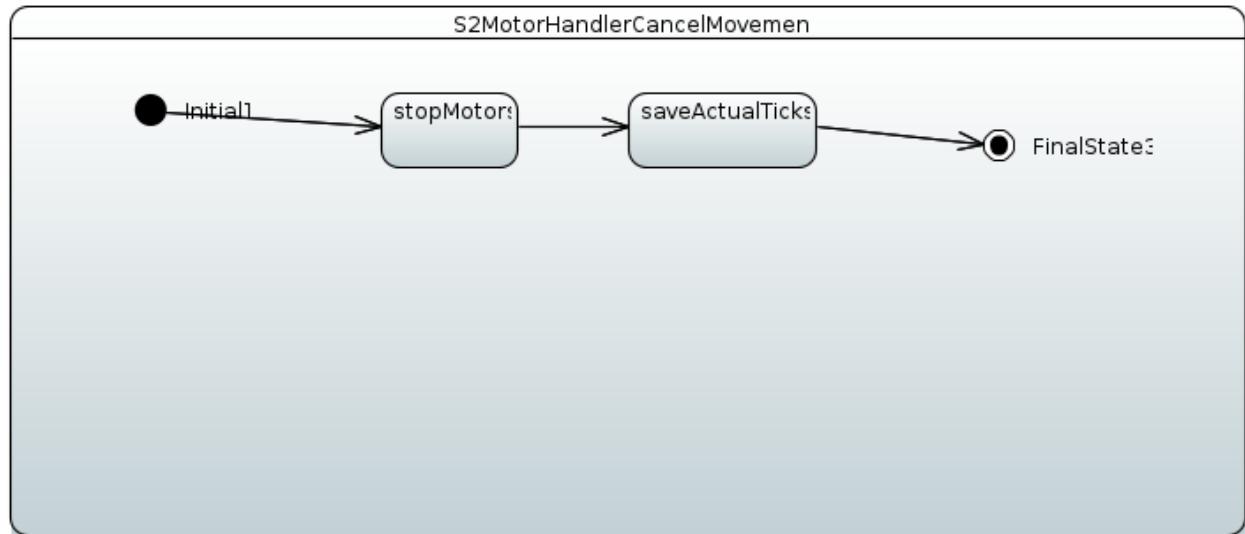


Figura 3.18: Diagrama de estados del método `cancelMovement()` del *motorHandler*.

Se cancela un movimiento que se esté ejecutando actualmente

- **stopMotor:** Se para el motor.
- **saveActualTicks:** Se guardan los ticks actuales que el motor ha recorrido.

En el caso del bloque “MovementComputer” tenemos los siguientes diagramas.

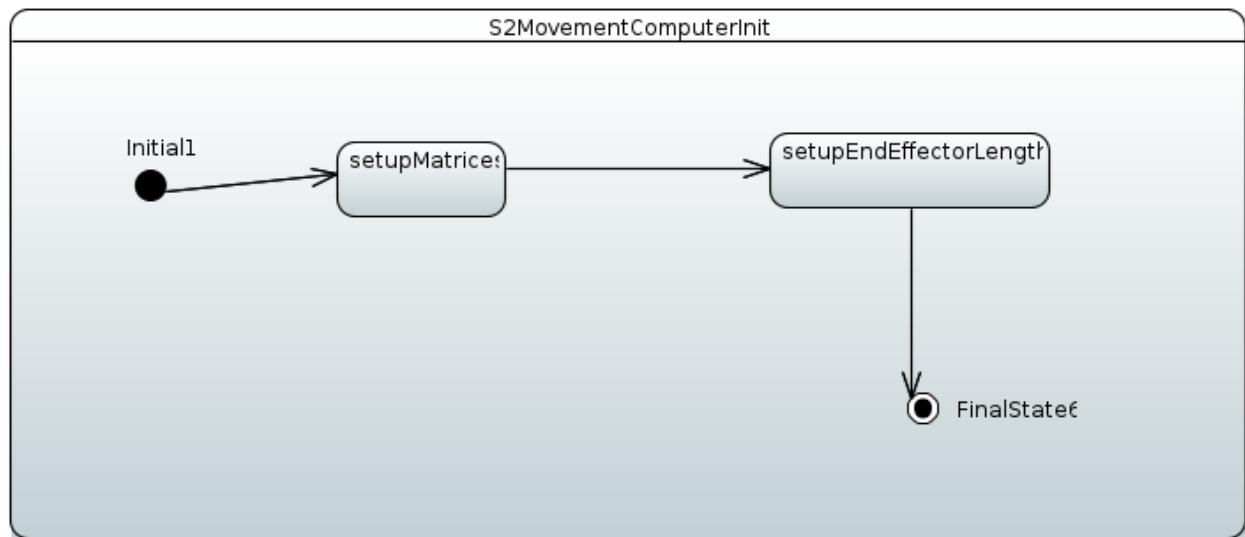


Figura 3.19: Diagrama de estados del método `init()` del *movementComputer*.

Se inicializan las matrices de la cinemática directa e inversa y se define la distancia del end-effector desde la base del brazo.

- `setupMatrices`: Se inicializan las matrices de la cinemática directa e inversa.
- `setupEndEffectorLength`: Se definen la distancia desde la base hasta el **end–effector!** (**end–effector!**).

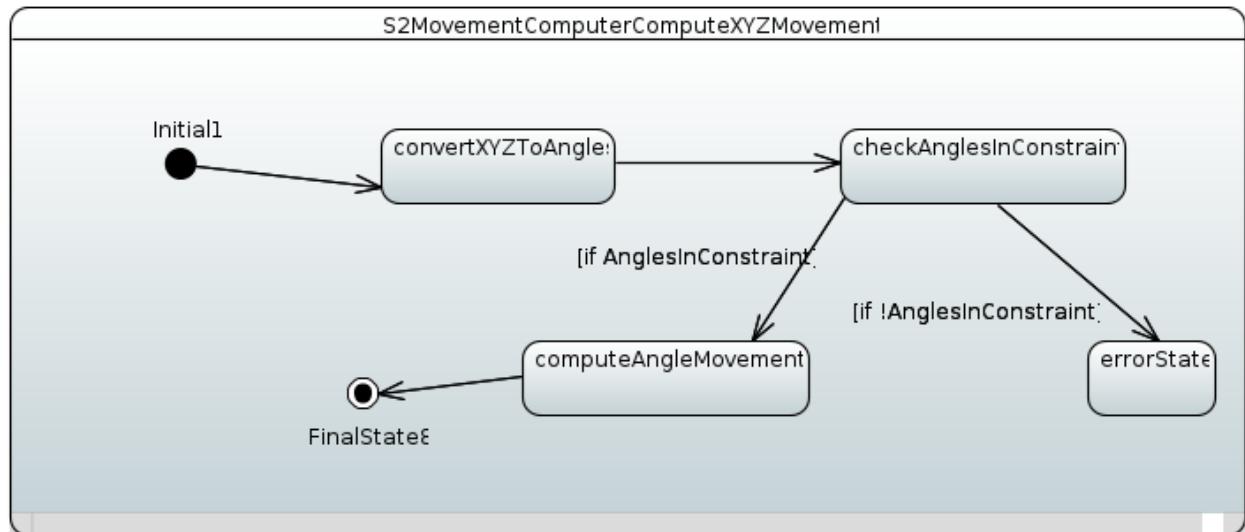


Figura 3.20: Diagrama de estados del método `computeXYZMovement()` del *movementComputer*.

- `convertXYZToAngles`: Se emplea la cinemática inversa para obtener la posición de los motores a partir de la posición del **end–effector!**

- **checkAnglesInConstraint:** Se verifica si los ángulos están dentro de las limitaciones de los motores.
- **computeAngleMovement:** Se calcula el movimiento que se debe realizar para mover el **end-effector!** desde la posición actual a la deseada.
- **errorState:** Estado de error al que se llega si en alguno de los estados ocurre algún problema inesperado.

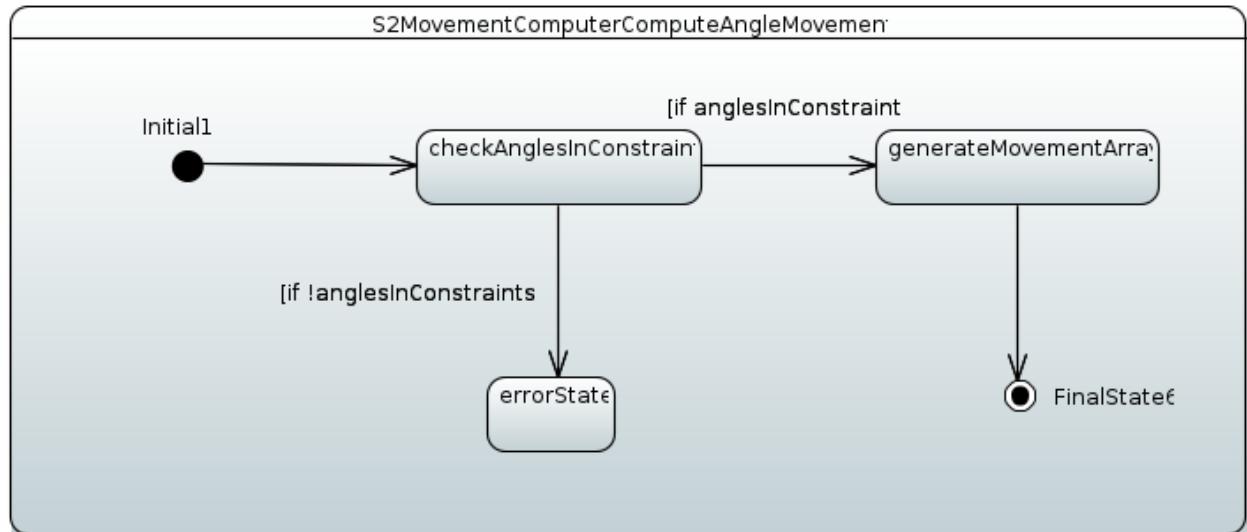


Figura 3.21: Diagrama de estados del método `computeAngleMovement()` del *movementComputer*.

- **checkAnglesInConstraint:** Se verifica si los ángulos están dentro de las limitaciones de los motores.
- **generateMovementArray:** Se generan los array de movimientos.
- **errorState:** Estado de error al que se llega si en alguno de los estados ocurre algún problema inesperado.

## Capítulo 4

### Las matemáticas del proyecto

4.1. Cinemática directa

4.2. Cinemática inversa

4.3. Funciones jacobianas

4.4. Implementación final realizada

# Capítulo 5

## *Hardware*

### 5.1. [REVISADO] Impresión en 3D

Las razones por las cuales se toma la decisión de fabricar la estructura del brazo mediante impresión 3D se detallan a continuación:

- Cumplir con el requisito de replicabilidad y asequibilidad: una de las bases del proyecto es que pueda ser reproducible a bajo coste tanto de recursos como de tiempo. Se decide por tanto construir la estructura física del brazo mediante técnicas de impresión 3D, ya que están altamente extendidas y son cada vez más asequibles.
- Características físicas del material: los plásticos utilizado en impresión 3D suelen ser ligeros y suficientemente resistente para soportar las cargas para las que está pensado el manipulador.
- Disponibilidad de impresora 3D: dado que la Universidad es capaz de proveer al equipo con una impresora 3D, los costes del proyecto se abaratan si la estructura es realizada con los medios de los que la ya se disponen.
- Simplificar el proceso de mejora y personalización: debido a la naturaleza OS y OH del proyecto, se espera que las personas interesadas puedan contribuir a él, mejorándolo y/o personalizándolo. Además, la impresión 3D facilita estas acciones.

En particular, la impresora que la Universidad pone a disposición del equipo de trabajo es la “*Ultimaker 3 Extended*”, la cual es capaz de imprimir en una alta variedad de materiales, de los cuales destacan los siguientes:

- Ácido Poliláctico (PLA)[23]: este material permite imprimir de manera segura con alta precisión dimensional y una resistencia a la tracción excepcional, que además soporta grandes velocidades de impresión y es biodegradable, ya que se obtienen a partir de almidón de maíz, de yuca, mandioca o de caña de azúcar.

- Acrilonitrilo Butadieno Estireno (ABS)[24]: material que presenta buena adhesión entre capas y una resistencia a temperaturas de hasta 85ºC. Permite obtener buenos detalles estéticos.
- Ultimaker Nylon: este material es un tipo de poliamida basada en los polímeros plásticos PA6/66. Presenta una absorción de humedad reducida así como una capacidad considerable de resistencia ante tensiones mecánicas junto con un bajo coeficiente de fricción, haciéndolo un material ideal para construcciones mecánicas.
- CPE y CPE+: este material presenta una alta estabilidad dimensional, con buena resistencia al impacto y a la temperatura. Debido a su alta solidez y su estabilidad dimensional ofrece un buen rendimiento mecánico y gran resistencia al desgaste.

Debido a la naturaleza mecánica del proyecto, el equipo ha decidido emplear materiales con alta resistencia mecánica para las piezas móviles. El Ultimaker Nylon junto con el CPE cumplen con dicha característica.

Por otro lado, los componentes que no sean móviles como carcasa o piezas protectoras se imprimirán en PLA ya que tras realizar pruebas, el equipo de desarrollo ha concluido que el material es lo suficientemente resistente para soportar los pesos a los que será sometido.

Aprovechando la licencia original GPL 3.0 del  $\mu$ Arm, se ha recuperado el modelo 3D proporcionado por UFACTORY como punto de partida. A partir de este modelo se han impreso las piezas que hemos decidido conservar para nuestro proyecto, a saber, la estructura general del brazo, exceptuando los soportes de los motores, los cuales tendrán que ser adaptados a los motores que se han decidido utilizar para este proyecto y la base la cual tendrá que adaptarse para poder albergar la placa de control y uno de los motores.

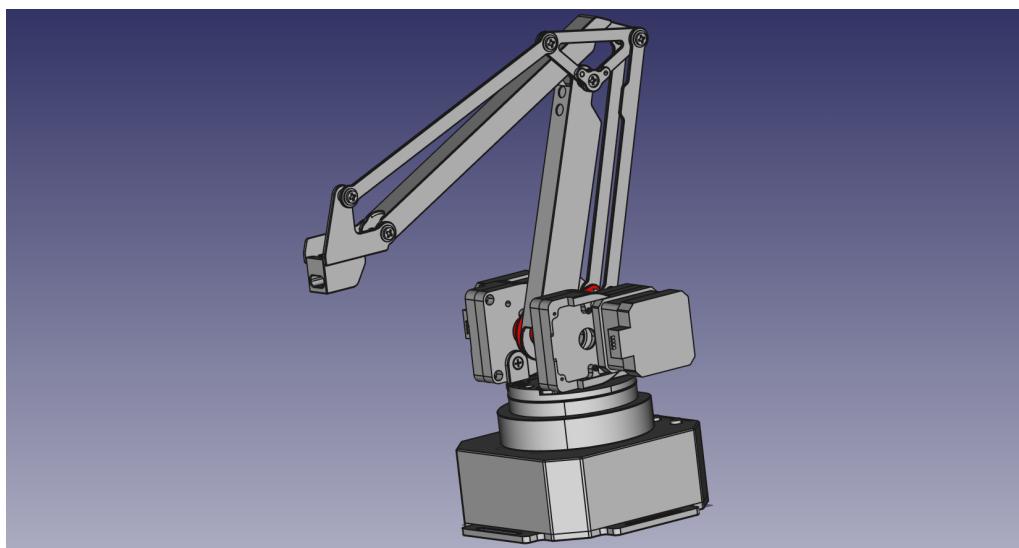


Figura 5.1: Concepto inicial del brazo robótico.

Las herramientas que han sido empleadas para visualizar y modificar el modelo y posteriormente imprimir las piezas han sido respectivamente FreeCAD y Ultimake Cura.



Figura 5.2: Logotipos de las herramientas utilizadas.

El flujo de trabajo que se ha seguido desde el modelo 3D hasta la impresión de una pieza ha sido el mostrado en la figura 5.3:

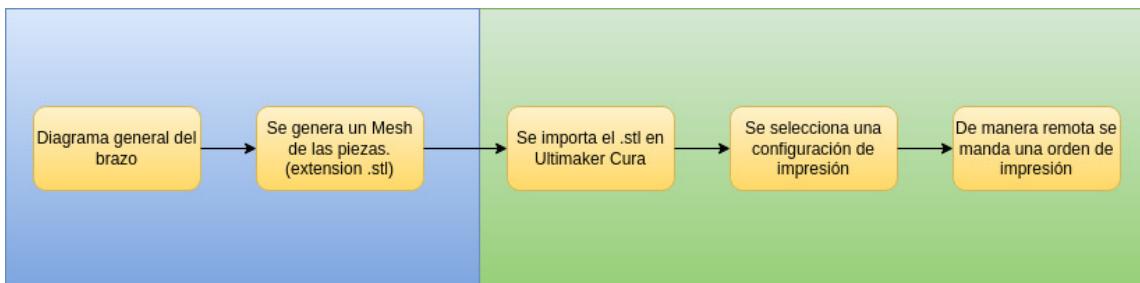


Figura 5.3: Flujo de trabajo del desarrollo y la impresión 3D.

A continuación se procederá a explicar la necesidad de remodelar ciertas piezas y los inconvenientes y contratiempos que han surgido durante el modelado y la impresión de estas.

En primer lugar se explicará la caja que alberga la placa de control y uno de los motores.

La placa de control del brazo robótico no es la misma que en el caso del  $\mu$ Arm de UFACTORY. Además, los motores que se han empleado en este proyecto son servomotores con carcasa y sistema de sujeción distinto que los motores paso a paso del  $\mu$ Arm. Debido a estos dos factores se ha tenido que diseñar nuevas partes para la base del brazo robótico..

Mas concretamente, la necesidad de rediseñar esta parte es debida a que la base original era demasiado pequeña en superficie para permitir introducir la placa de control. Además, el servomotores no podrían haber cabido junto con la placa ya que la altura era insuficiente. Por otro lado los sistemas de sujeción presentes en la caja existente no podían ser empleados para la placa de control desarrollada en este proyecto.

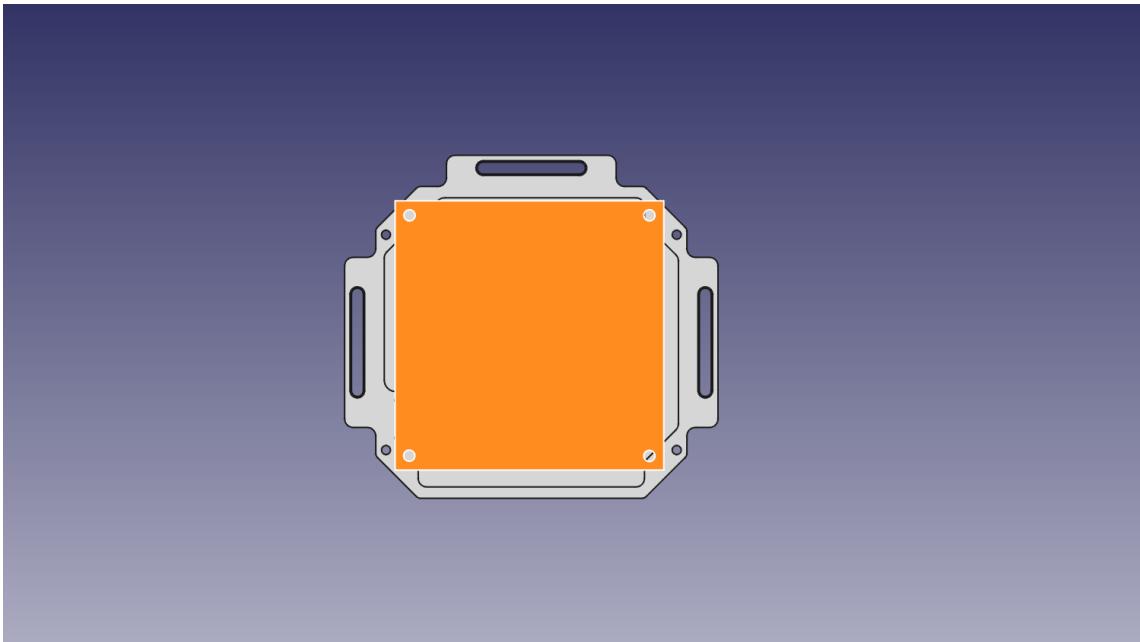


Figura 5.4: Proyección de la placa de control (naranja) sobre la base original del  $\mu$ Arm (gris)

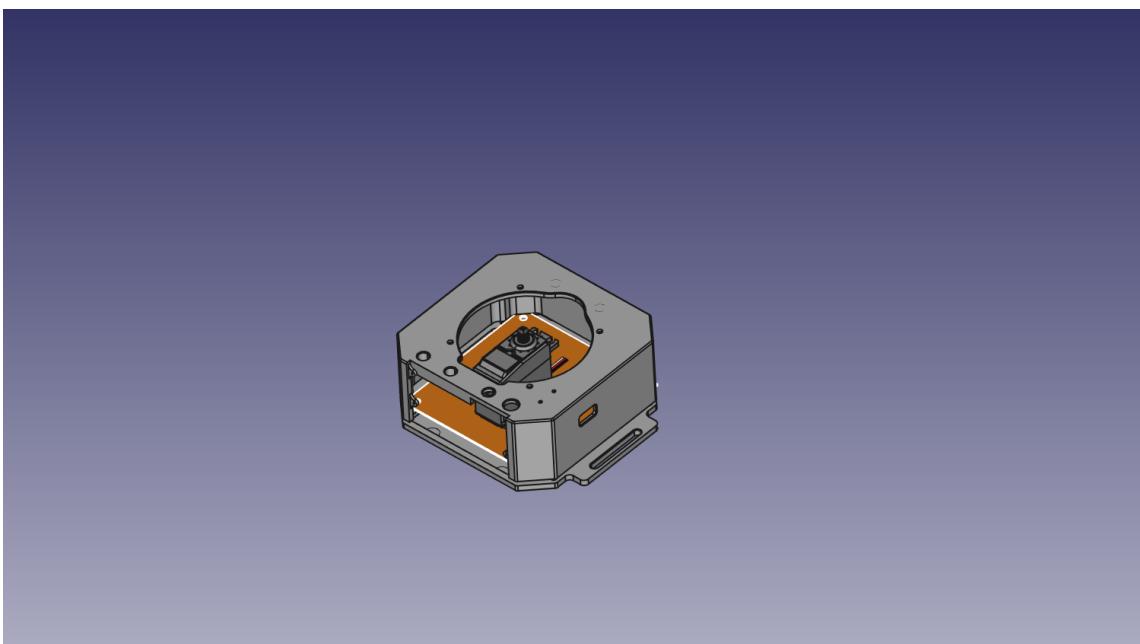


Figura 5.5: Placa y motor dentro de la caja original del  $\mu$ Arm

Como se observa en la figura 5.5 el motor sobresale por encima de las paredes y no hay ninguna manera de sujetarlo a estas o a la base.

Para solucionar los anteriores problemas se diseña una nueva base en la que se pueda encajar la placa, además de unas paredes lo suficientemente altas para poder introducir el motor junto con la placa.

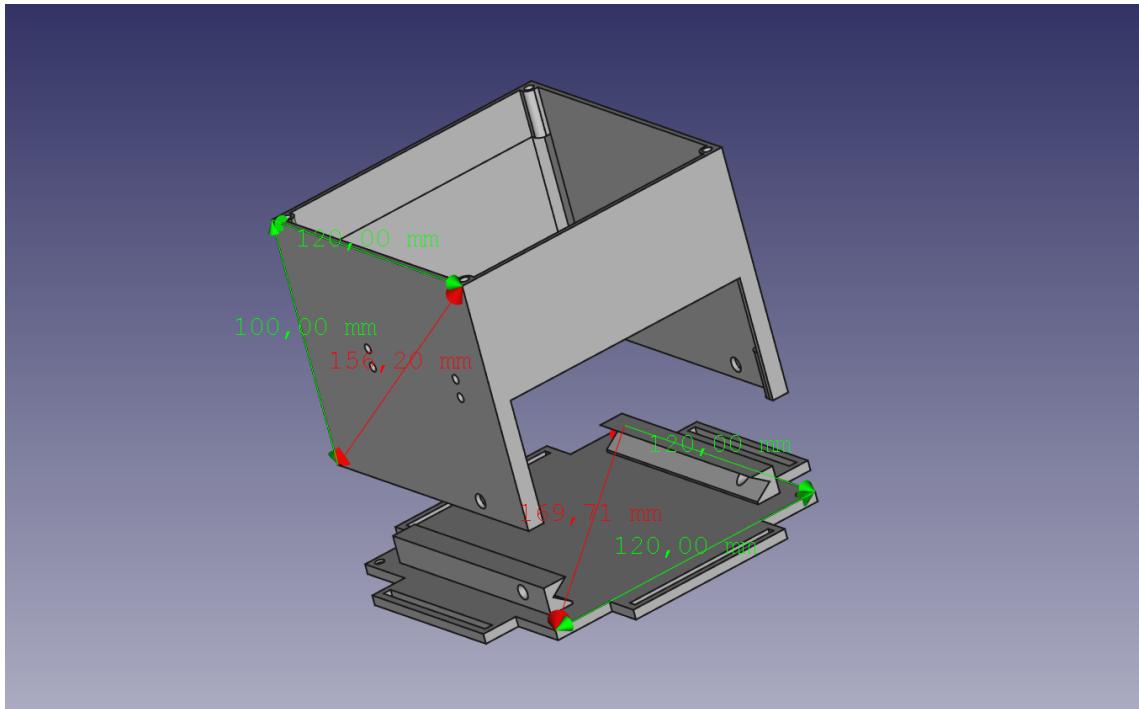


Figura 5.6: Base y paredes tras realizar las modificaciones necesarias

En la base se pueden observar unos carriles cuyo cometido es sujetar la placa. Después de que la placa sea insertada en estos carriles, se asegura su posición mediante los agujeros laterales que pueden observarse en la figura 5.6.

Cabe destacar que para poder introducir la placa en las ranuras de los carriles, se han diseñado las siguientes piezas.

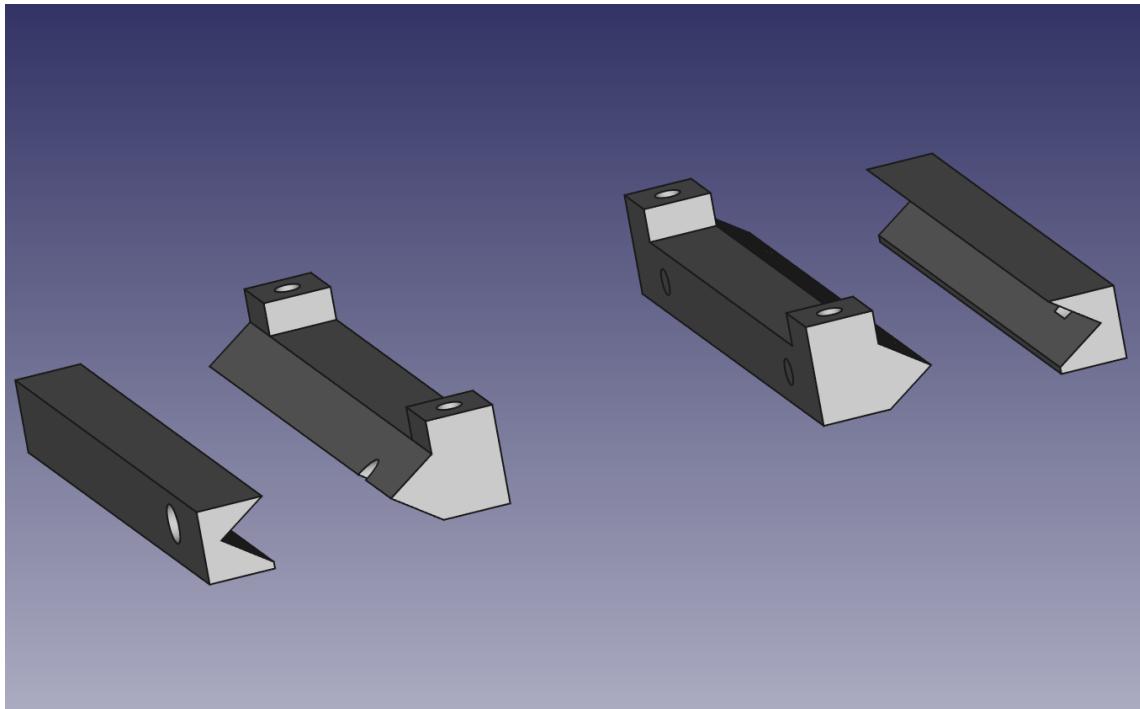


Figura 5.7: Sistema de raíles de la placa

En la figura 5.7 se pueden observar los raíles que serán añadidos a la placa de control. En el exterior de la imagen aparecen los carriles presentes en la base de la caja, donde se puede ver que, al estar la placa completamente introducida en los carriles, los agujeros del carril y del raíl se posicionan de tal manera que se permite introducir un pasador que asegura la posición de la placa.

Por otro lado, para poder sujetar el motor que moverá el brazo alrededor del eje vertical se ha tenido que diseñar la siguiente pieza.

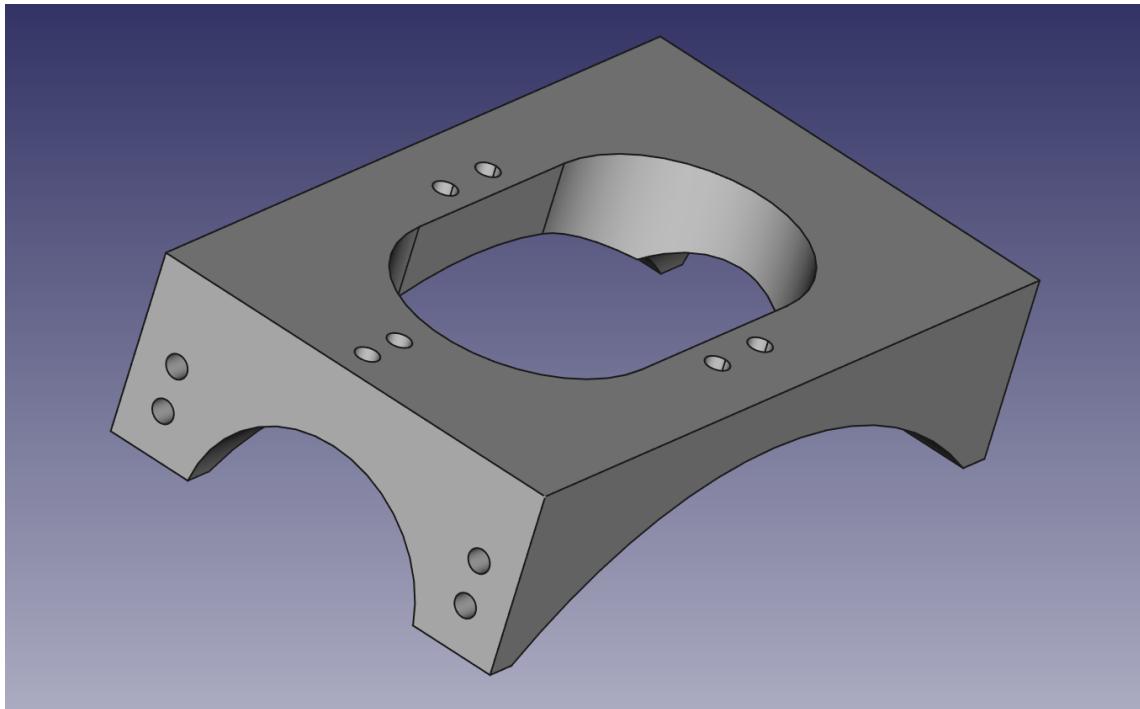


Figura 5.8: Pieza de sujeción del motor

Esta pieza se atornilla a las paredes de la caja y empleando las solapas del motor, este se atornilla en el centro de la pieza como se puede ver en la figura 5.9.

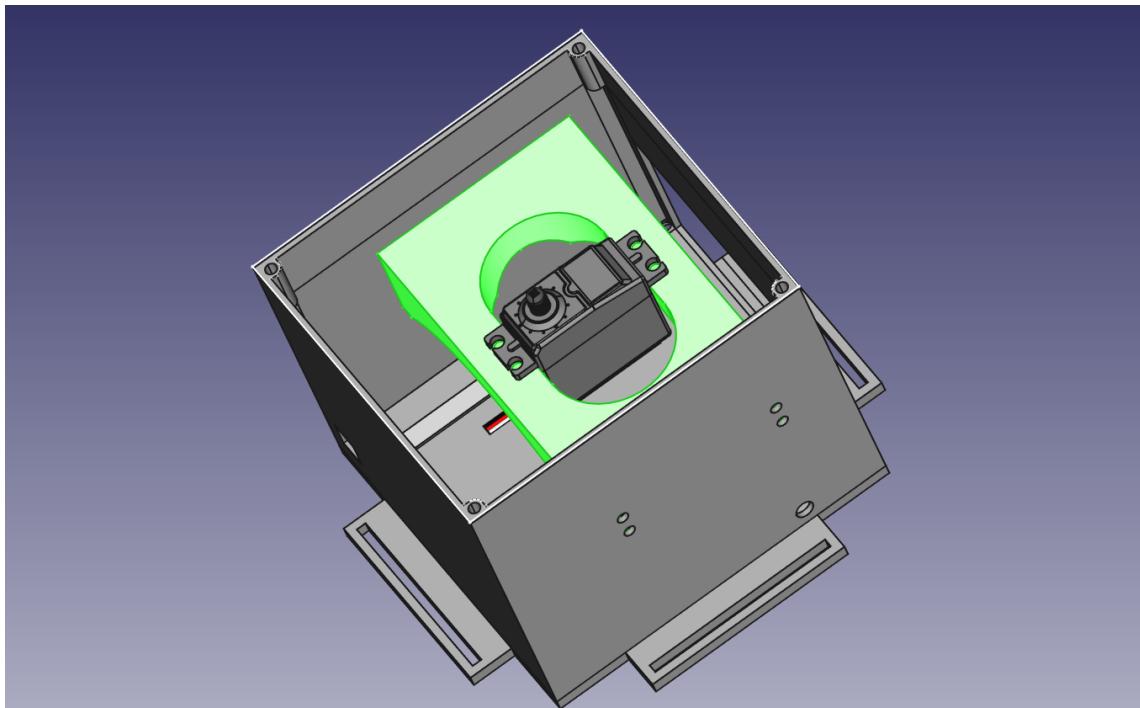


Figura 5.9: Pieza de sujeción del motor (verde) dentro de la caja

Dado que el tamaño de la base y de las paredes ha cambiado, la tapa superior debe también

ser modificada para adaptar su tamaño y su sistema de sujeción a los nuevos diseños

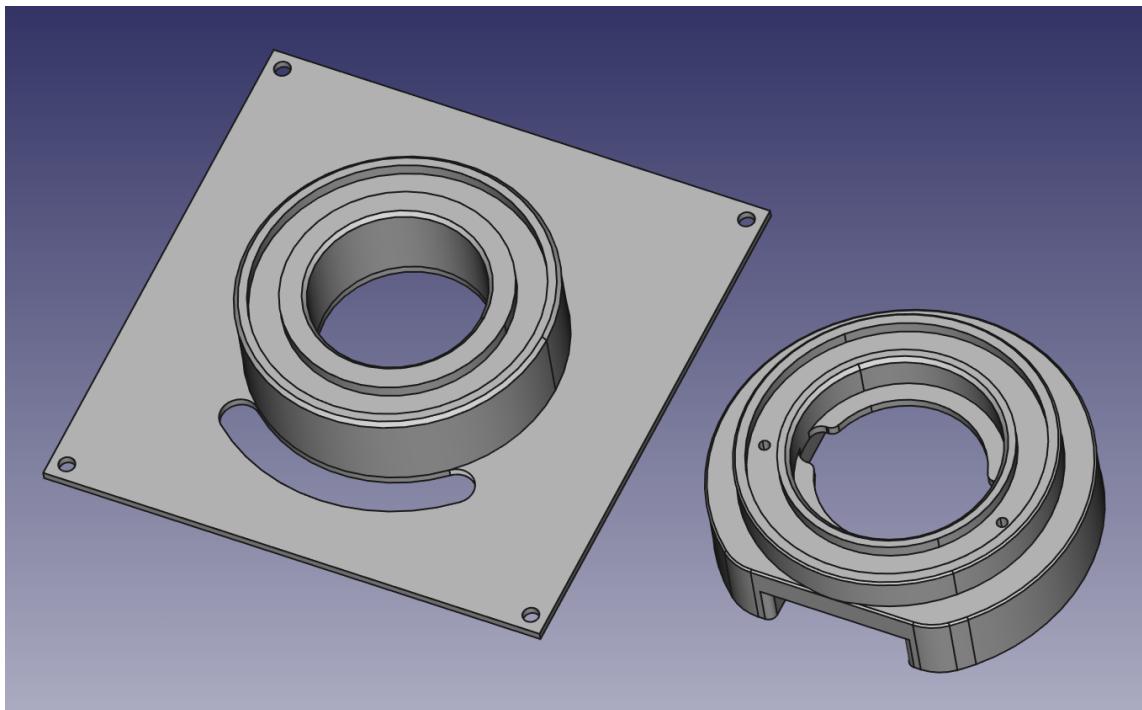


Figura 5.10: Tapa original (derecha) junto a la tapa modificada (izquierda)

En la figura 5.10 se observa que en la parte superior las diferencias entre ambas piezas son mínimas respetándose los diámetros del disco exterior e interior sobre los que descansará la base giratoria.

En la figura 5.11 podemos observar como la base giratoria descansa sobre los anillos superiores y se inserta dentro de la tapa.

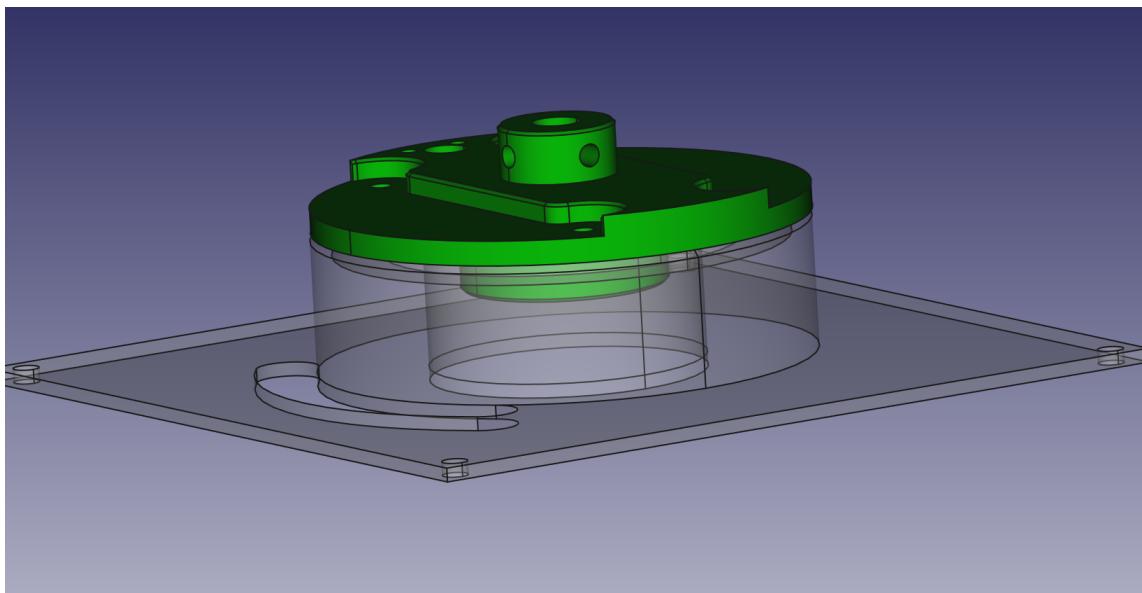


Figura 5.11: Tapa superior transparente y base giratoria (verde)

De esta manera se consigue que el peso de la parte móvil del brazo descansen sobre la tapa y no sobre el eje de rotación. Por otro lado, el hecho de que la base giratoria protruya hacia el interior de la tapa permite que la pieza este mas cerca del motor permitiendo acortar el eje.

A en la figura 5.12 se muestra la pieza que conecta el motor a la base giratoria a través de un eje. Esta pieza tiene en su base un desgaste tal que permite que el engranaje del motor pueda ser introducido dentro, mientras que en la parte de arriba tiene unos agujeros que permiten la sujeción de un eje metálico el cual llega hasta la base giratoria y transmite el movimiento del motor a esta.

La pieza plana que sale hacia un lateral sirve para que el cilindro pueda alcanzar un fin de carrera que indique cuando el motor ha llegado a una posición extrema.

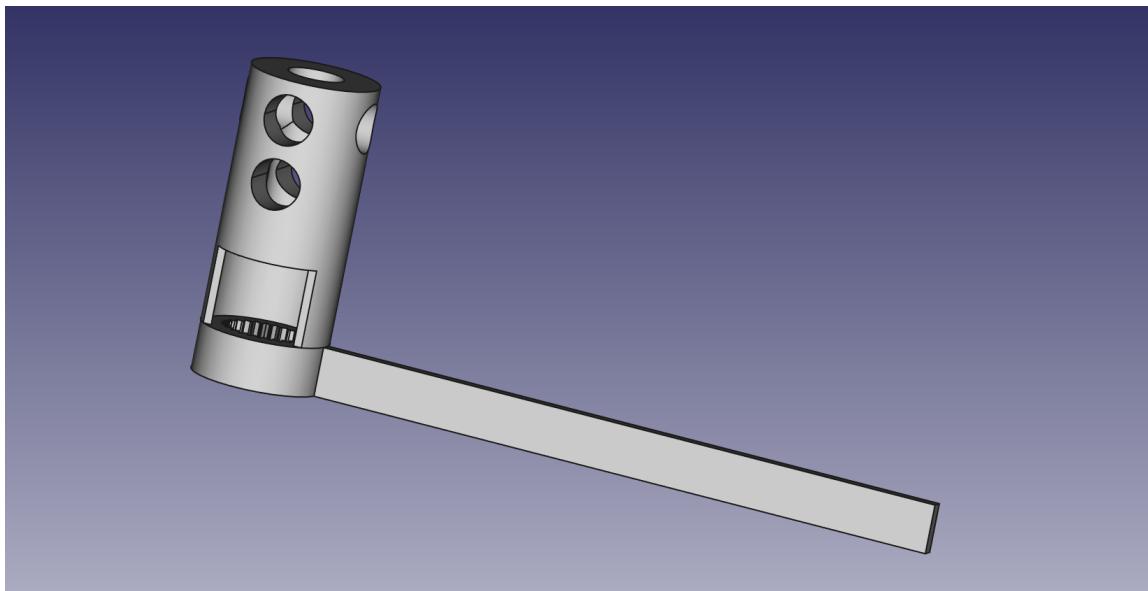


Figura 5.12: Pieza de unión del motor y el eje

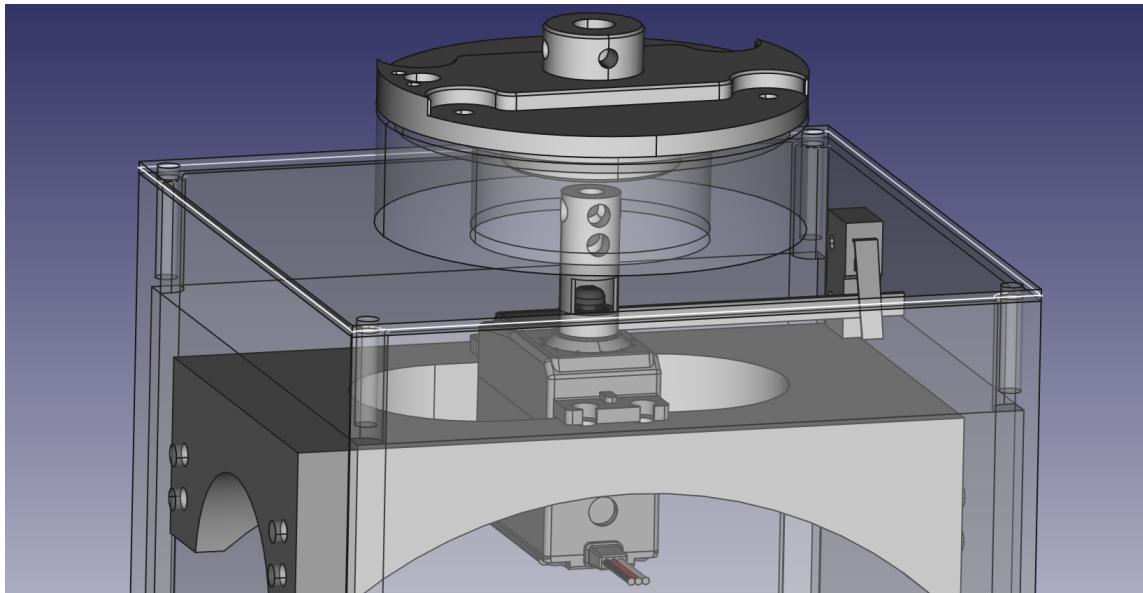


Figura 5.13: Sistema completo

En la figura 5.13 se observa la construcción completa de la base. En esta imagen se dejan las paredes y la tapa superior con cierta transparencia para poder ver el interior de la caja. Entre el cilindro montado encima del motor y la base giratoria habrá un eje metálico que comunicara el movimiento.

Por otro lado, para que sea posible ubicar los motores que se encargaran del movimiento vertical del brazo se han de crear nuevas piezas en las que puedan ser atornillados. Esta pieza puede observarse en la figura 5.14

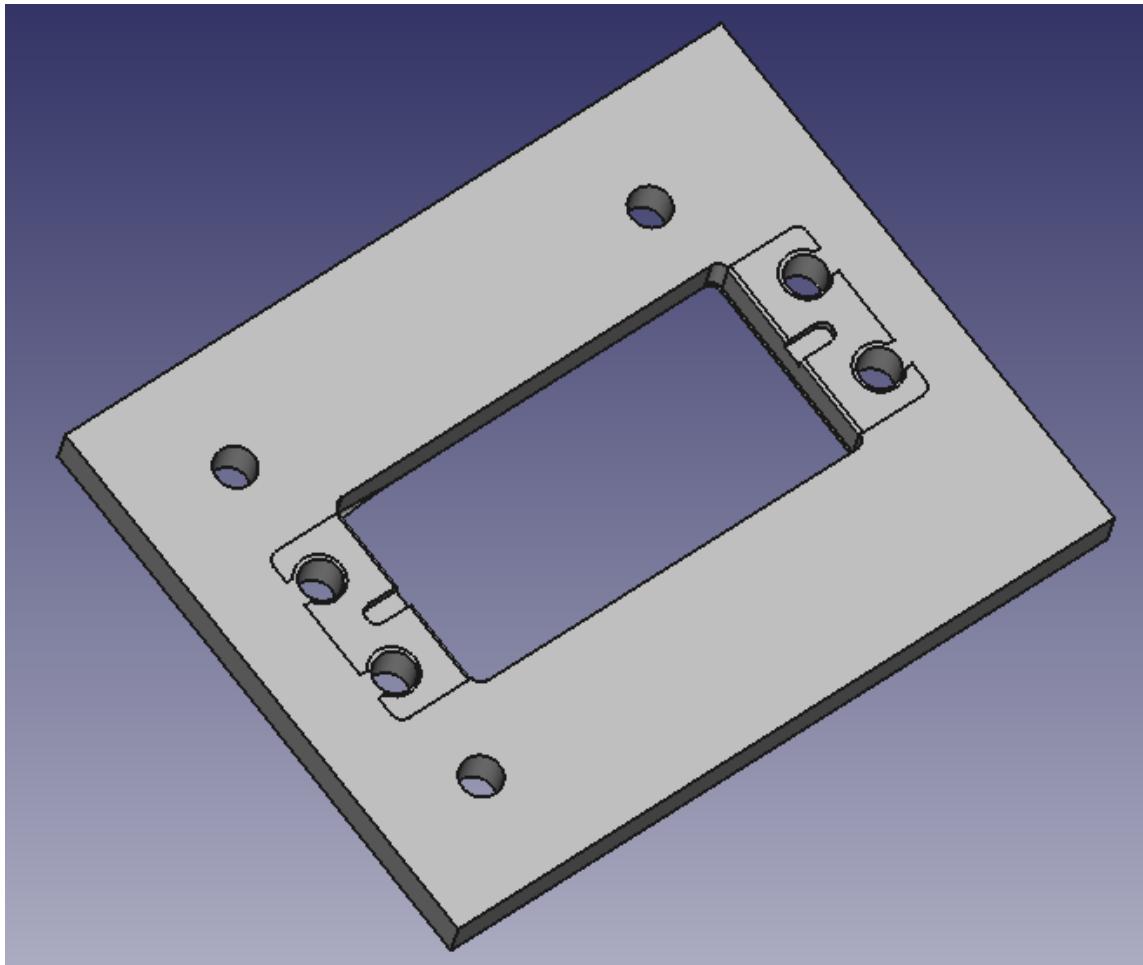


Figura 5.14: Placa de sujeción de los motores laterales

## 5.2. [REVISADO] Teoría de la estructura física

La estructura del brazo es pantográfica, lo cual significa que es un sistema de enlaces mecánicos que reproduce el movimiento de un punto de una articulación en un segundo punto, normalmente a un tamaño o bien más pequeño o bien más grande, como se puede apreciar en la figura 5.15. Se originó en el siglo XVII y su aplicación más conocida es como instrumento de dibujo.

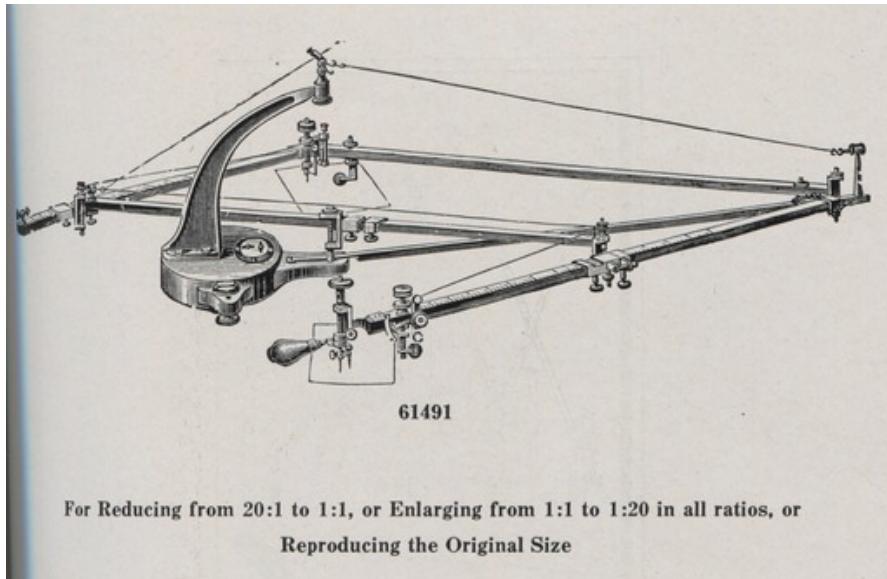


Figura 5.15: Una estructura pantográfica que transmite el movimiento de un punto al siguiente [25].

El hecho de que la estructura del brazo sea pantográfica permite controlar todas las articulaciones mediante motores ubicados en la base. Esto es de especial importancia ya que hace posible que las articulaciones finales no carguen con el peso de los motores, permitiendo emplear materiales como el plástico para la construcción de la estructura y, además, dando capacidad al brazo para levantar cargas más pesadas.

Otro beneficio derivado de la ubicación de los motores en la base es la alta estabilidad del brazo, ya que siendo estos los componentes más pesados y estando ubicados en la base se consigue un centro de gravedad muy cercano a la superficie de apoyo, permitiendo así un amplio rango de movimientos.

Otra característica a destacar es que debido a la estructura pantográfica, el *end-effector* mantiene siempre un ángulo perpendicular con la superficie de apoyo del brazo. Esto supone, por un lado, perder ciertos grados de libertad pero por otro, simplifica la estructura y abarata el coste de producción.

### 5.3. [REVISADO] Microcontrolador utilizado

Para el desarrollo de la placa que conforma el S2 se ha empleado un microcontrolador “dsPIC33EP512GM604”. Los motivos por los cuales se ha usado este modelo son los siguientes:

- En primera instancia, la cantidad y la precisión de los canales PWM que este ofrece, ya que son suficientes para poder controlar todos los motores y su precisión permite generar una señal adecuada para controlar a cada uno de ellos.
- Por otro lado, debido a la naturaleza de los cálculos que se deben realizar para convertir posiciones cartesianas a ángulos y viceversa, el *Digital Signal Processor* (DSP) facilita

la obtención de los valores de las diferentes operaciones matriciales que permiten esta conversión.

- Otro aspecto importante es la posibilidad de almacenar hasta 512KB de memoria de programa.
- Por último, se ha elegido un DSP de Microchip debido a que todos los integrantes del grupo de desarrollo tiene experiencia previa con este fabricante. Además, dicho fabricante proporciona documentación extensa sobre sus productos.

Una parte crítica del proyecto es la precisión y el control de los motores. En este aspecto, los canales PWM del DSP permiten generar ciclos de trabajo a partir de un registro de 16 bits. Para una frecuencia de  $50Hz$  (período de  $20ms$ ), un giro de  $360^\circ$  supondría una duración del nivel alto del PWM de  $5ms$ . Es decir, se podrá controlar la rotación entre  $0^\circ$  y  $360^\circ$  con  $2^{16}/4 = 16384$  bits de control. Esto supone que obtendremos una precisión máxima de  $360/16384 = 0'02197^\circ$ , lo cual es suficiente para el proyecto.

Cabe destacar, que si bien los motores son capaces de girar  $360^\circ$  hexadecimales, debido a las limitaciones geométricas del brazo, los ángulos de giro se ven limitados en la aplicación concreta que se les dará.

En cuanto a las operaciones matemáticas, el DSP cuenta con un circuito *Phase Loop Lock* (PLL), el cual permite incrementar la frecuencia del oscilador para conseguir de esta manera una mayor cantidad de instrucciones por segundo y, por tanto, un mayor volumen de operaciones.

En el momento de comenzar el proyecto se podía suponer que el código no podría albergarse en la memoria que normalmente suministran microcontroladores de menores prestaciones.

Este microcontrolador, gracias a que dispone de varios puertos UART permite recibir las órdenes y los movimientos necesarios desde el S1. Desde S2, se realizará las operaciones matemáticas necesarias y después se encargará de generar las señales PWM para mover los motores de tal manera que el brazo robótico quede en la posición deseada.

## 5.4. [REVISADO] Desarrollo y componentes de la PCB

### 5.4.1. Objetivos

El desarrollo de una PCB se considera una de las partes esenciales de este proyecto y por lo tanto su importancia es máxima.

El objetivo principal que persigue el diseñar y construir una PCB en este proyecto es el de dotar al sistema de un centro de cómputo principal, el cual es utilizado para procesar las órdenes de S1, realizar los cálculos pertinentes y ejecutar las acciones necesarias sobre la estructura del brazo robótico. La PCB por lo tanto se encarga de alojar el microcontrolador dsPIC, así como todos los periféricos necesarios para establecer la comunicación con S1, realizar el control de los actuadores y monitorizar el estado del manipulador.

### 5.4.2. Componentes principales

En general, se podría decir que los componentes de la PCB se clasifican en tres categorías principales:

- Componentes de alimentación eléctrica.
- Microcontrolador y componentes auxiliares para su correcto funcionamiento.
- Periféricos destinados a control de actuadores y canales de comunicación.

En primer lugar, los componentes de alimentación eléctrica son aquellos que forman el circuito de alimentación del microcontrolador así como de los actuadores. El circuito eléctrico de alimentación de la PCB se ha diseñado para poder alimentar de forma simultánea al microcontrolador y a cada uno de los cuatro servomotores y está formado por dos etapas:

- La PCB recibe una tensión de entrada de 9V y una corriente de entre 1,8A–2A mediante una clema. Posteriormente, esta tensión de alimentación será reducida y adaptada para alimentar a cada una de las etapas de la PCB, es decir, al microcontrolador y servomotores.
- En la primera etapa se reduce el voltaje de alimentación principal a 6V y 0,4A aproximadamente para cada uno de los servomotores, utilizando para ello un regulador LM317 por cada uno. Esta alimentación se realiza mediante clemas, las cuales se usan para conectar la alimentación de los motores.
- En la segunda etapa se reduce el voltaje de alimentación principal a 3,3V y 0,15A aproximadamente con el objetivo de alimentar el microcontrolador, utilizando en conjunto los reguladores L7805CV y A1117H. Esta alimentación se realiza mediante pistas únicamente.

En segundo lugar, el microcontrolador y sus componentes auxiliares representan el núcleo de la PCB:

- El dsPIC se encuentra localizado en el centro de la PCB y de él surgen todas las conexiones necesarias hacia los periféricos.
- Los componentes auxiliares del microcontrolador son componentes eléctricos que aseguran el correcto funcionamiento del dsPIC, así como su seguridad. En el caso específico de este microcontrolador, es necesario incluir varios condensadores en sus pines de alimentación.

En último lugar, se han utilizado los siguientes periféricos:

- Cristal de cuarzo, el cual genera una señal de reloj precisa y de buena calidad. Su frecuencia es de 7MHz y será recibida por el microcontrolador para ser usada como la señal de reloj principal del sistema.

- Puerto de programación, donde se puede conectar la sonda de programación del microcontrolador y por lo tanto es un elemento con especial importancia y relevancia dentro del sistema.
- Puerto TRIS para recibir señales digitales y analógicas, las cuales son procesadas por el microcontrolador. En este proyecto, estos puertos se utilizan para monitorizar los finales de carrera de la estructura del brazo robótico.
- Puerto PWM: capaz de generar señales PWM, es decir, señales digitales cuadradas y con un ancho de pulso variable, las cuales son completamente necesarias para controlar los servomotores.
- UART, donde se establece un canal de comunicación *hardware* con S1, el cual se usa para recibir órdenes, movimientos y realimentar su resultado de vuelta a S1.
- LEDs de estado, los cuales muestran información del sistema usando tres diodos LED.

Mediante esta descripción general de la PCB y sus componentes se pretende brindar una idea global de la misma, así como de cuál es su papel dentro del proyecto. En los apartados siguientes se describe en términos técnicos los elementos de esta PCB, así como el proceso de diseño y fabricación llevado a cabo.

#### 5.4.3. Diseño lógico y diagrama esquemático

El primer paso llevado a cabo durante el diseño de la PCB ha sido realizar un diseño lógico de alto nivel, en el cual se muestran las conexiones lógicas que existen entre los componentes; se trata, por lo tanto, del diseño con mayor nivel de abstracción y que tiene como objetivo establecer la primera toma de contacto con el plano de la PCB.

El diseño lógico se lleva a cabo mediante un diagrama esquemático que contiene dos tipos de elementos: huella lógica de cada uno de los componentes y las conexiones entre ellos. Este diagrama se ha llevado a cabo utilizando la herramienta “*Schematic Layout Editor*”, incluida en KiCad.

El diagrama esquemático está dividido en dos partes principales, las cuales facilitan la compresión del mismo:

- Diagrama esquemático del circuito de alimentación.
- Diagrama esquemático del microcontrolador y sus periféricos.

En primer lugar, se procede a describir detalladamente el diagrama esquemático del circuito de alimentación, el cual contiene las dos etapas de alimentación mencionadas anteriormente en el punto 5.4.2, siendo los principales componentes usados los reguladores de voltaje LM317, L7805CV y AZ1117H.

La clema principal de alimentación recibe 9V y 1,8A aproximadamente. Esta alimentación debe ser provista externamente mediante una fuente de alimentación regulable o similares.

Conectados directamente a la clema principal, se encuentran los reguladores de tensión correspondientes a las dos etapas de alimentación:

- La primera está formada por cuatro reguladores LM317, los cuales alimentan cada uno de los servomotores. En dicha etapa se reduce el voltaje de 9V a 5,5V.
- La segunda está formada por un regulador de tensión L7805CV y un AZ1117H. En dicha etapa de alimentación se reduce el voltaje de 9V a 5V usando el primer regulador, y de 5V a 3,3V usando el segundo.

En particular, el LM317 es un regulador convencional que recibe una tensión continua de entrada de entre 3V – 40V y provee una tensión continua salida de entre 1,25V – 37V. Además, la relación entre la tensión de entrada y salida depende del dimensionado de dos resistencias auxiliares. El conexionado sugerido por el fabricante es el siguiente (ver imagen 5.16) y se ha obtenido del *datasheet* del regulador:

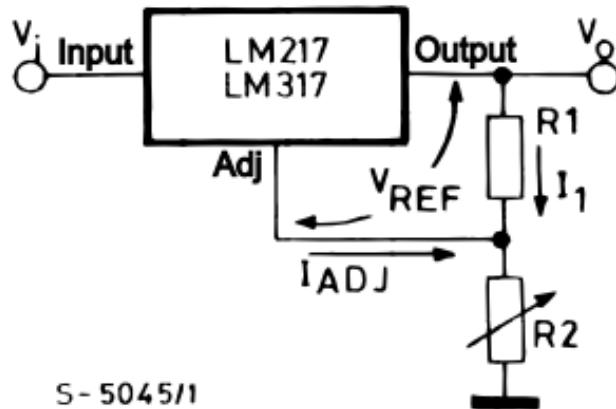


Figura 5.16: Diagrama de conexionado del LM317 [26].

La ecuación de funcionamiento que ofrece el fabricante para el regulador LM317 es la siguiente:

$$V_{OUT} = V_{REF} \cdot \left( 1 + \frac{R_2}{R_1} \right) + I_{ADJ} \cdot R_2 \quad (5.1)$$

La ecuación anterior (ver ecuación 5.1) debe ser usada para realizar los cálculos pertinentes sobre el valor de las resistencias  $R_2$  y  $R_1$ . Se tienen en cuenta, además, dos observaciones necesarias para la correcta aplicación de la ecuación anterior:

- Por definición, el fabricante establece el valor  $V_{REF} = 1,25V$ .
- Por motivos de construcción del regulador, la corriente  $I_{ADJ}$  tiene un valor máximo de  $100\mu A$  y, por lo tanto, el término de la ecuación que la involucra puede ser despreciado.

Se obtiene entonces que la ecuación funcional (ecuación 5.2) a utilizar en el cálculo de  $R_1$  y  $R_2$  es:

$$V_{OUT} = 1,25V \cdot \left( 1 + \frac{R_2}{240\Omega} \right) \quad (5.2)$$

Teniendo en cuenta los voltajes de alimentación requeridos para las dos etapas de alimentación de la PCB ( $\sim 5,5V$  para la primera,  $\sim 3,3V$  para la segunda), se han realizado los siguientes cálculos para los valores de la resistencia  $R_1$  y  $R_2$ :

- Primera etapa, alimentación de servomotores a  $5,5V$  y  $0,4A$  requeridos:

$$5,5V = 1,25V \cdot \left( 1 + \frac{R_2}{R_1} \right) \quad (5.3)$$

Por disponibilidad de materiales, se ha decidido escoger los valores  $150\Omega$  y  $500\Omega$  para las resistencias  $R_1$  y  $R_2$  respectivamente. Tomando en cuenta dichos valores y utilizando la ecuación anterior, se obtiene una reducción de voltaje de  $9V$  a  $5,41V$ , voltaje suficiente para alimentar los servomotores.

Aplicando el conexionado recomendado por el fabricante y los cálculos para el valor de las resistencias se obtiene finalmente el diagrama esquemático del circuito de alimentación de los servomotores (imagen 5.17):

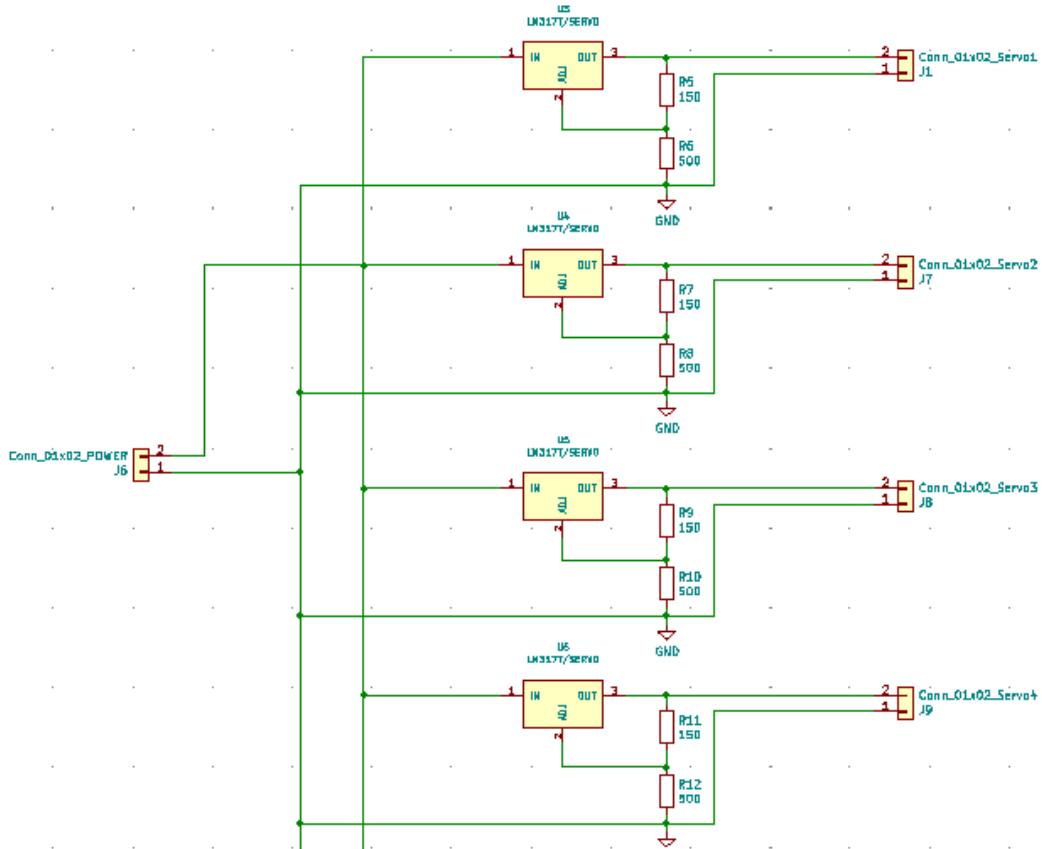


Figura 5.17: Diagrama esquemático del circuito de alimentación de los servomotores.

El funcionamiento de los reguladores L7805CV y AZ1117H es simple:

- El regulador L7805CV recibe un voltaje de entre  $7V - 35V$  y provee un voltaje de salida fijo de  $5V$ . Su conexionado se realiza utilizando dos condensadores auxiliares, tal y como se describe en el *datasheet* del componente:

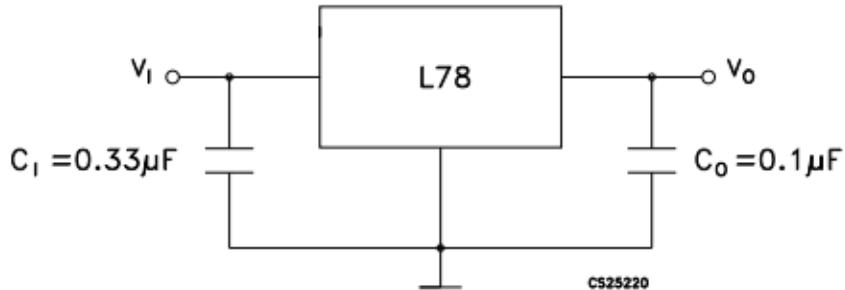


Figura 5.18: Diagrama de conexionado del regulador L7805 [27].

- El regulador AZ1117H recibe un voltaje de hasta  $15V$  y provee un voltaje de salida fijo de  $3,3V$ . Su conexionado se realiza de la misma forma que para el regulador anterior, tal y como se describe en el *datasheet* del componente:

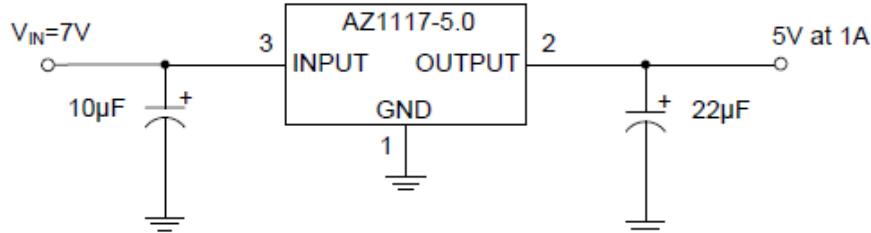


Figura 5.19: Diagrama de conexionado del regulador AZ1117H [28].

En el diagrama anterior (imagen 5.19) se utiliza el modelo que ofrece regulación de voltaje a  $5V$ . Sin embargo, el modelo usado en el proyecto es el que ofrece regulación de voltaje a  $3,3V$ , siendo su conexionado exactamente igual.

Teniendo en cuenta la información expuesta anteriormente, se ha decidido conectar en serie ambos reguladores, consiguiendo de esta forma una regulación de  $9V$  a  $5V$  y posteriormente una regulación de  $5V$  a  $3,3V$ . El diagrama esquemático final es el siguiente (ver imagen 5.20):

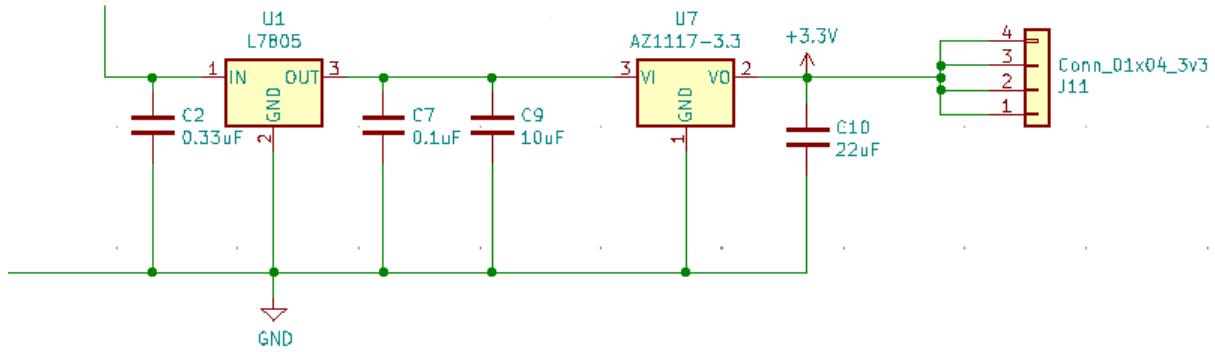


Figura 5.20: Diagrama esquemático de la etapa de alimentación del microcontrolador.

Es importante destacar tres aspectos:

- La primera etapa de alimentación incluye clemas para su conexiónado con los servomotores.
- La segunda etapa de alimentación incluye un puerto de cuatro pines, los cuales se usan para alimentar los micro-interruptores y el microcontrolador.
- El conexionado de todos los reguladores se ha realizado en paralelo, dedicando un regulador para cada servomotor así como para el microcontrolador. El objetivo de esta configuración es garantizar una vía de alimentación independiente para cada componente, reduciendo las interferencias de alimentación entre los reguladores y diferenciando la etapa de alimentación de los servomotores y microcontrolador.

En segundo lugar se procede a describir detalladamente el diagrama esquemático del microcontrolador y sus periféricos. A continuación se muestra el diagrama esquemático final y posteriormente se detallará cada uno de los periféricos:

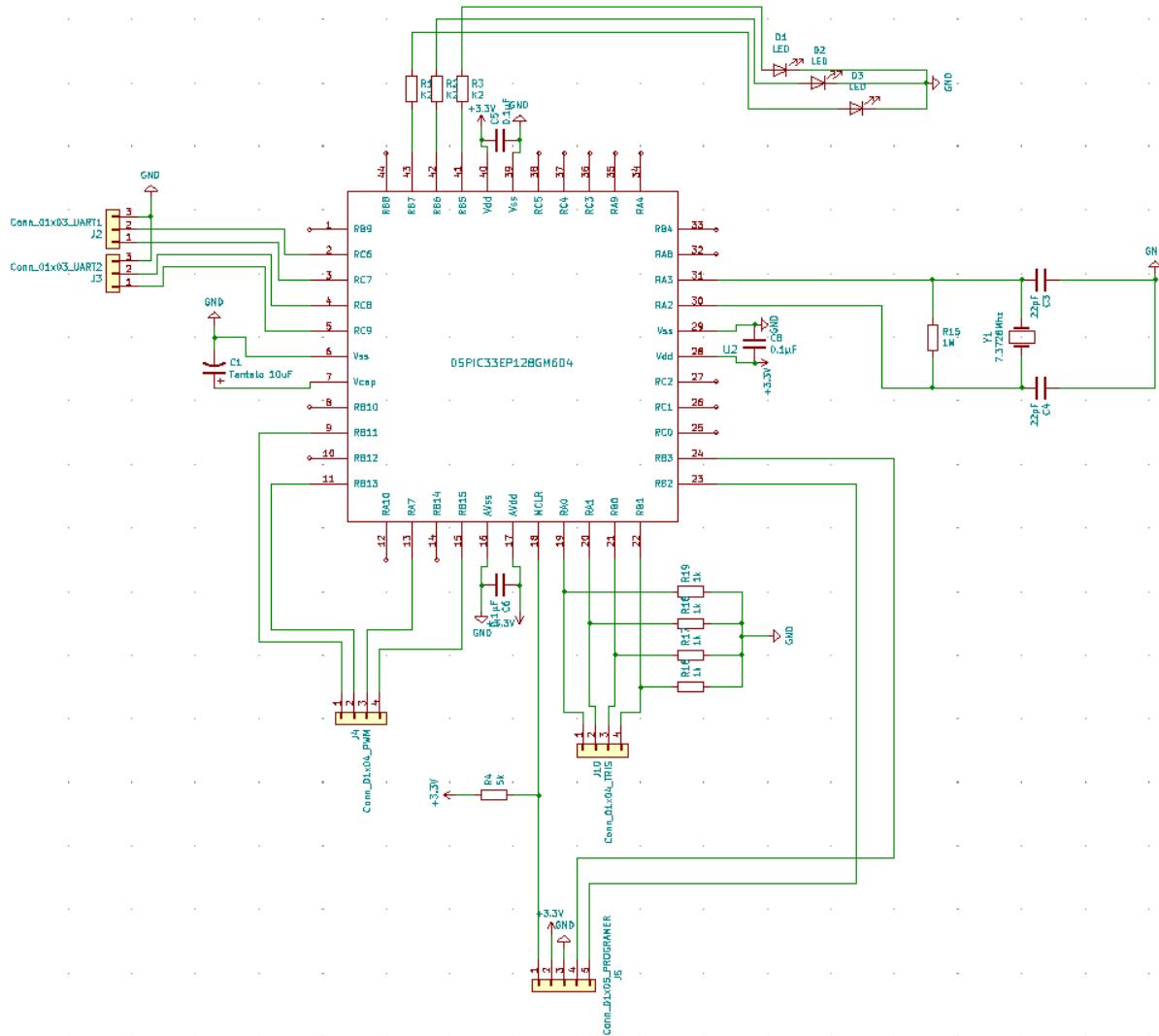


Figura 5.21: Diagrama esquemático del microcontrolador y sus periféricos.

Primeramente, es importante describir los condensadores auxiliares necesarios para el correcto funcionamiento del microcontrolador, los cuales se encuentran conectados en los pines de alimentación del mismo. Su conexionado es sugerido por el fabricante en el *datasheet* según el siguiente esquema (ver imagen 5.22):

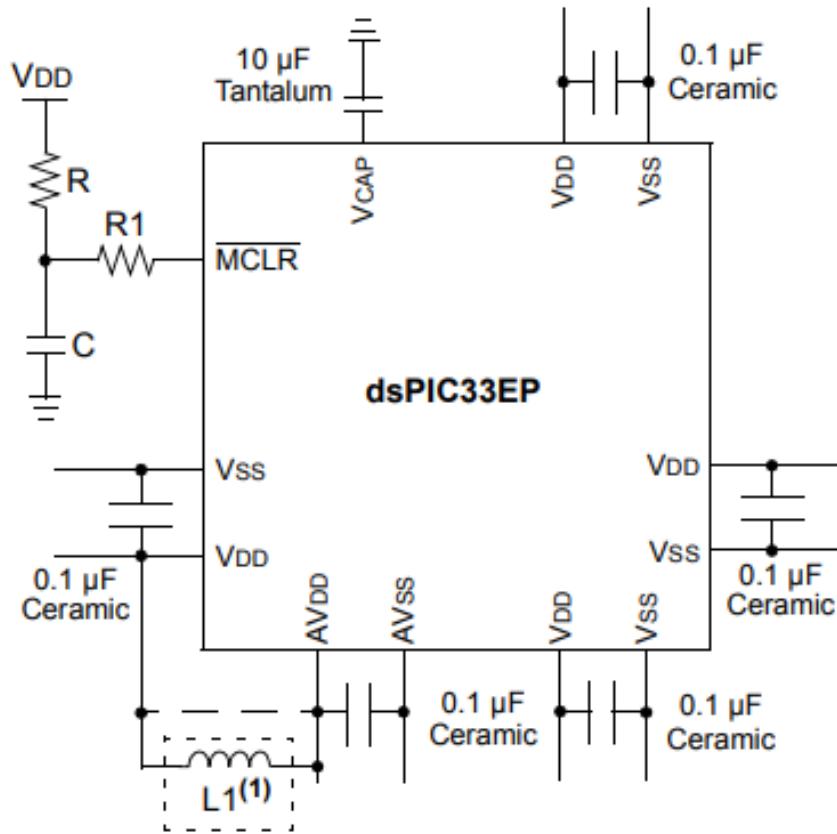


Figura 5.22: Conexionado mínimo del microcontrolador [29].

Todos los condensadores han sido conectados a los pines descritos por el fabricante y escogidos teniendo en cuenta las características técnicas de los mismos, también descritas en el *datasheet*.

A continuación, se describe el conexionado del resto de puertos y periféricos:

- Cristal de cuarzo, en términos técnicos, oscila a  $7,32\text{MHz}$ , frecuencia perfectamente válida para el microcontrolador usado en el proyecto:

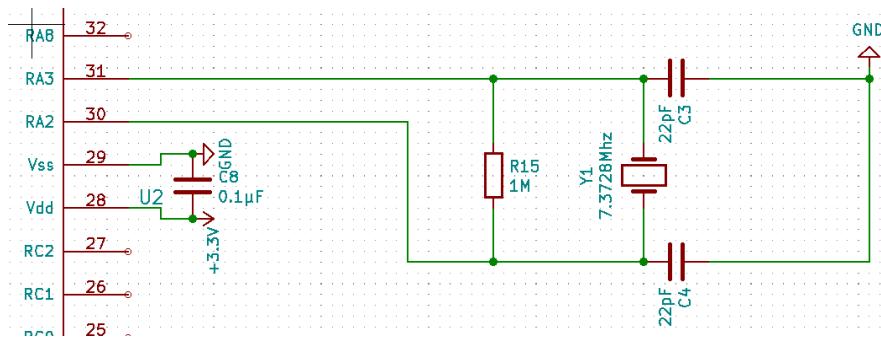


Figura 5.23: Diagrama esquemático del conexionado del generador de señales.

Su conexionado se realiza con los pines 32 y 31 del microcontrolador, siguiendo la estructura de la imagen anterior (ver imagen 5.23), empleando también una resistencia de

$1M\Omega$  y dos condensadores de  $22pF$ .

- Puerto de programación mediante sonda, el cual debe tener una estructura específica y se describe en el *datasheet* de la misma (ver imagen 5.24):

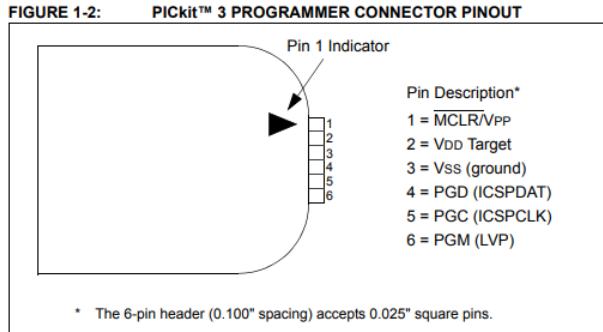
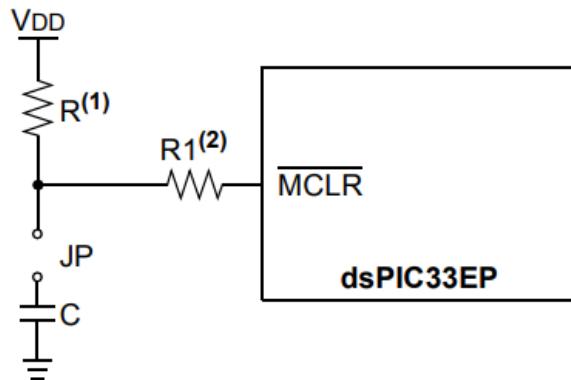


Figura 5.24: *Pinout* del conector de la sonda de programación [29].

Cabe destacar que el pin 18 o  $\overline{MCLR}/V_{PP}$  debe tener un conexionado específico en el que se emplea una resistencia *pull-up*; esta estructura de conexión se muestra en el *datasheet* del microcontrolador (ver imagen 5.25):



- Note 1:**  $R \leq 10 \text{ k}\Omega$  is recommended. A suggested starting value is  $10 \text{ k}\Omega$ . Ensure that the MCLR pin  $V_{IH}$  and  $V_{IL}$  specifications are met.
- 2:**  $R1 \leq 470\Omega$  will limit any current flowing into MCLR from the external capacitor, C, in the event of MCLR pin breakdown due to Electrostatic Discharge (ESD) or Electrical Overstress (EOS). Ensure that the MCLR pin  $V_{IH}$  and  $V_{IL}$  specifications are met.

Figura 5.25: Conexión del pin  $\overline{MCLR}/V_{PP}$  [29].

En este proyecto se ha decidido no incluir el *jumper* sugerido para conexión del pin  $MCLR/V_{PP}$  y por lo tanto  $R_1$  no se añade en el diagrama esquemático.

Teniendo en cuenta lo anteriormente mencionado, el conexionado final del puerto de programación mediante sonda es el siguiente (ver imagen 5.26):

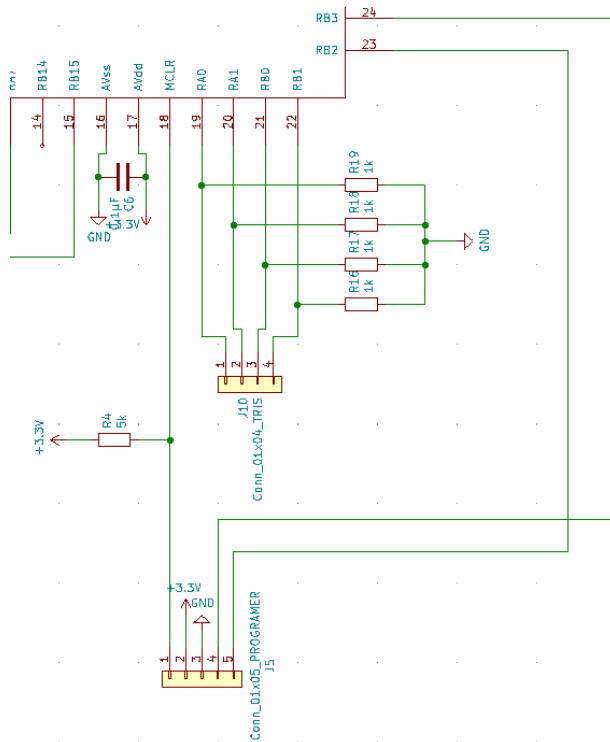


Figura 5.26: Diagrama esquemático del puerto de programación.

- Puerto TRIS, usado para detectar si el brazo robótico se encuentra en uno de sus finales de carrera o zonas límite de movimiento se utilizan unos micro-interruptores que, al ser presionados por el manipulador, realizan un cortocircuito. Mediante dicha señal y dado que estos micro-interruptores se encuentran conectados a los pines 19, 20, 21 y 22 del microcontrolador, se puede realizar la lectura de los mismos y detectar el estado de micro-interruptor. De esta manera, se consigue saber cuándo el manipulador ha alcanzado o no un final de carrera.

A continuación se muestra un esquema de lo mencionado anteriormente:

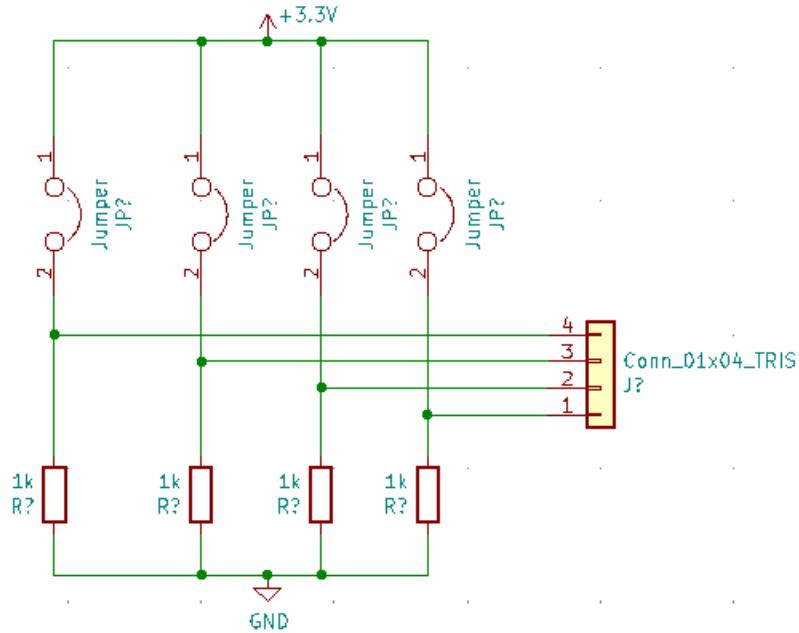


Figura 5.27: Circuito lógico para los finales de carrera.

Mediante el conexionado anterior (ver imagen 5.27), si el micro-interruptor está abierto se recibe un nivel bajo por el pin del microcontrolador, mientras que si el micro-interruptor está cerrado se recibe un nivel alto por el pin del microcontrolador.

Cabe destacar que tanto la resistencia *pull-down* como la conexión a tierra se incluyen en la PCB. Sin embargo, los micro-interruptores y su conexión a  $V_{DD}$  se encuentran localizados en la estructura del brazo robótico.

El diagrama esquemático que implementa esta funcionalidad es el siguiente (imagen 5.28):

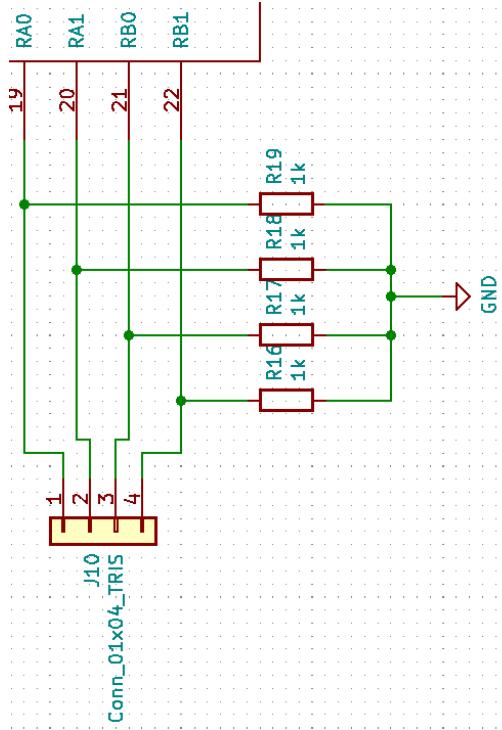


Figura 5.28: Diagrama esquemático del puerto TRIS.

- Puerto PWM, para el control de servomotores controlando así la posición angular de los mismos.

Los generadores de señal PWM de este microcontrolador poseen las siguientes características técnicas relevantes:

- El microcontrolador ofrece 6 generadores de señal PWM de alta precisión y velocidad.
- Cada uno de los generadores posee un registro de 16 bits para la selección de la duración del ciclo de trabajo; este registro está dividido en parte alta y parte baja.
- Es posible utilizar de forma independiente la parte alta y parte baja de cada uno de los generadores, consiguiendo de esta forma dos subgeneradores que producen señales PWM independientes. Sin embargo, su precisión se reduce a 8 bits.

El esquema mostrado por el fabricante para los generadores PWM es el siguiente (ver imagen 5.29):

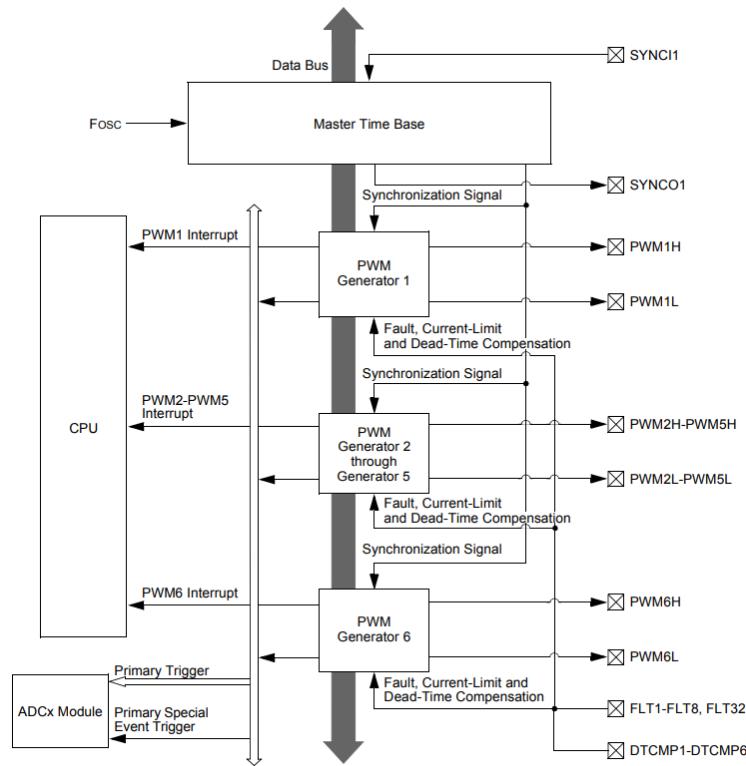


Figura 5.29: Esquema del generador PWM [29].

Puesto que para este proyecto únicamente se necesitan cuatro generadores PWM, se ha decidido utilizar cuatro módulos con precisión de 16 bits. En el caso de utilizar la precisión completa de cada generador, la señal de salida se proporciona por el pin correspondiente a la parte baja del registro. Mediante la información obtenida del *pinout* del microcontrolador, se tiene que los generadores de PWM 1, 4, 2 y 3 tienen asignados los pines 15, 13, 11 y 9, respectivamente.

El conexionado final del puerto PWM en el diagrama esquemático es el siguiente (imagen 5.30):

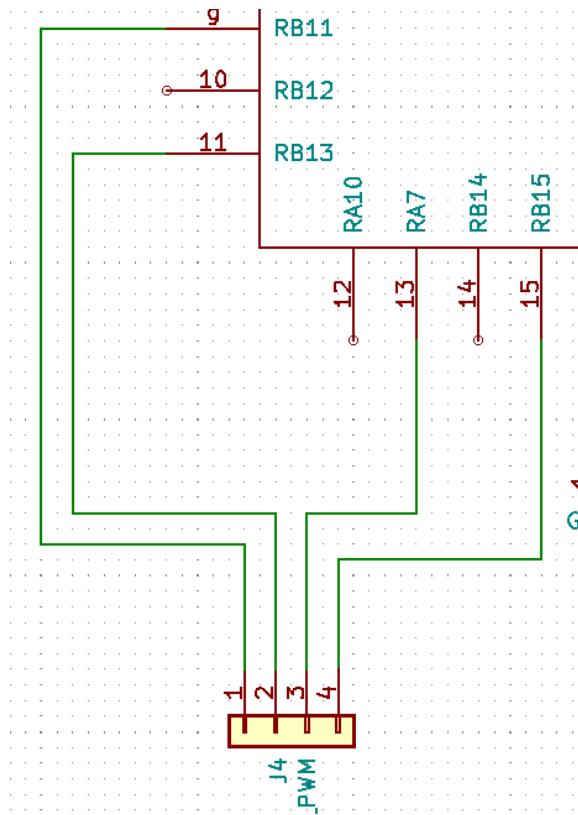


Figura 5.30: Diagrama esquemático del puerto PWM.

- Puertos UART, donde se establece un canal de comunicación *hardware* asíncrono en el cual existen diversas configuraciones en cuanto a formato de transmisión de bits y velocidades de comunicación. Suele ser un método de comunicación muy usado en microcontroladores y dispositivos *hardware* en general. El protocolo UART utiliza un puerto con tres conexiones: emisor ( $T_X$ ), receptor ( $R_X$ ) y tierra ( $GND$ ).

El esquema simplificado de este periférico en el *datasheet* es el siguiente (imagen 5.31):

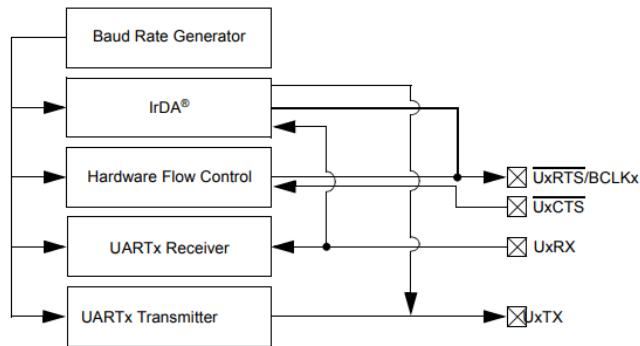


Figura 5.31: Esquema del periférico UART [29].

Se ha tomado la decisión de incluir dos puertos UART independientes en la PCB que se ha desarrollado, con el objetivo principalmente de dedicar uno de los canales a envío y

recepción de instrucciones; mientras que el otro se usa para realizar labores de depuración y pruebas.

El conexionado de los puertos UART se realiza mediante pines reconfigurables del microcontrolador, en este caso se han utilizado los pines 2 y 3 para el primer canal, además de los pines 3 y 4 para el segundo canal. El diagrama esquemático final es el siguiente (imagen 5.32):

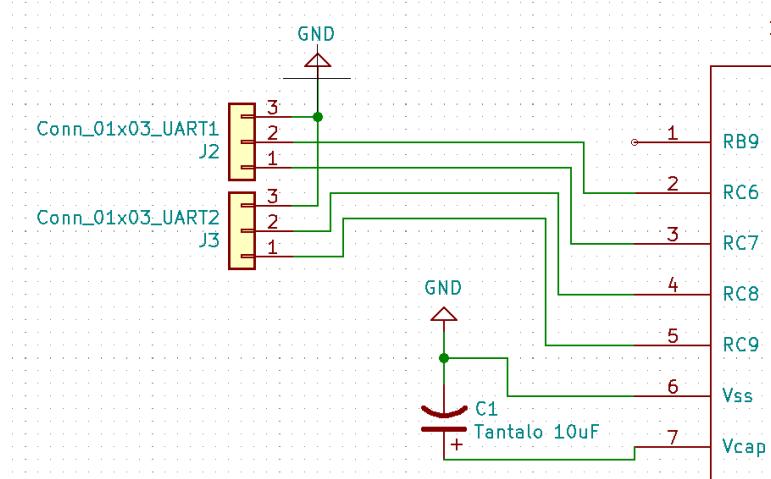


Figura 5.32: Diagrama esquemático de los puertos UART.

- LEDs de estado que muestran el estado del brazo robótico y demás aspectos del sistema. Su conexionado es realizado utilizando los pines reconfigurables 41, 42 y 43, los cuales pueden ser usados para habilitar una salida digital. Se evalúa un nivel alto para encender el LED mientras que un nivel bajo para apagarlo.

A continuación se muestra el diagrama esquemático del circuito (imagen 5.33):

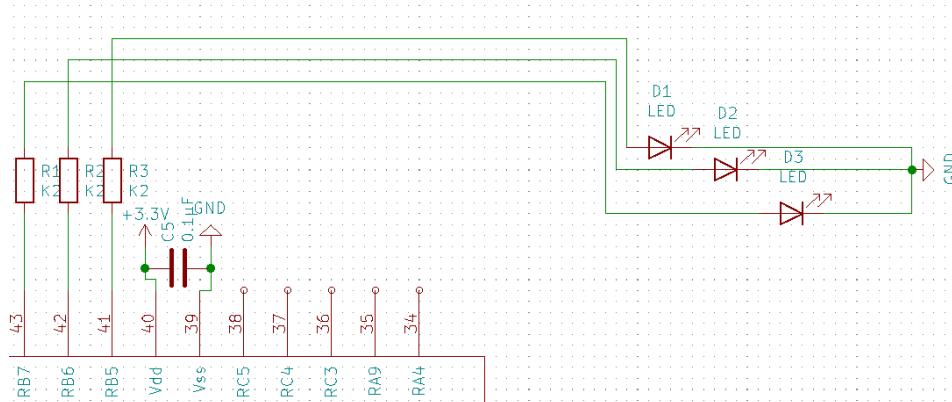


Figura 5.33: Diagrama esquemático de los LEDs.

Teniendo en cuenta que la corriente máxima suministrada por el microcontrolador es de  $18mA$  para salida digital y el que voltaje elegido ha sido  $3,3V$ , el pin suministraría como mucho una potencia de  $0,0594W$ . Se ha decidido que una potencia adecuada a suministrar

sería un 80% de la máxima, es decir  $0,0475W$ . Para cumplir esta restricción, se ha calculado el valor ideal de la resistencia del esquema anterior y su valor recomendado es de entre  $180\Omega$  y  $200\Omega$ .

En conclusión, una vista completa sobre el diagrama esquemático del proyecto es la siguiente (ver imagen 5.34):

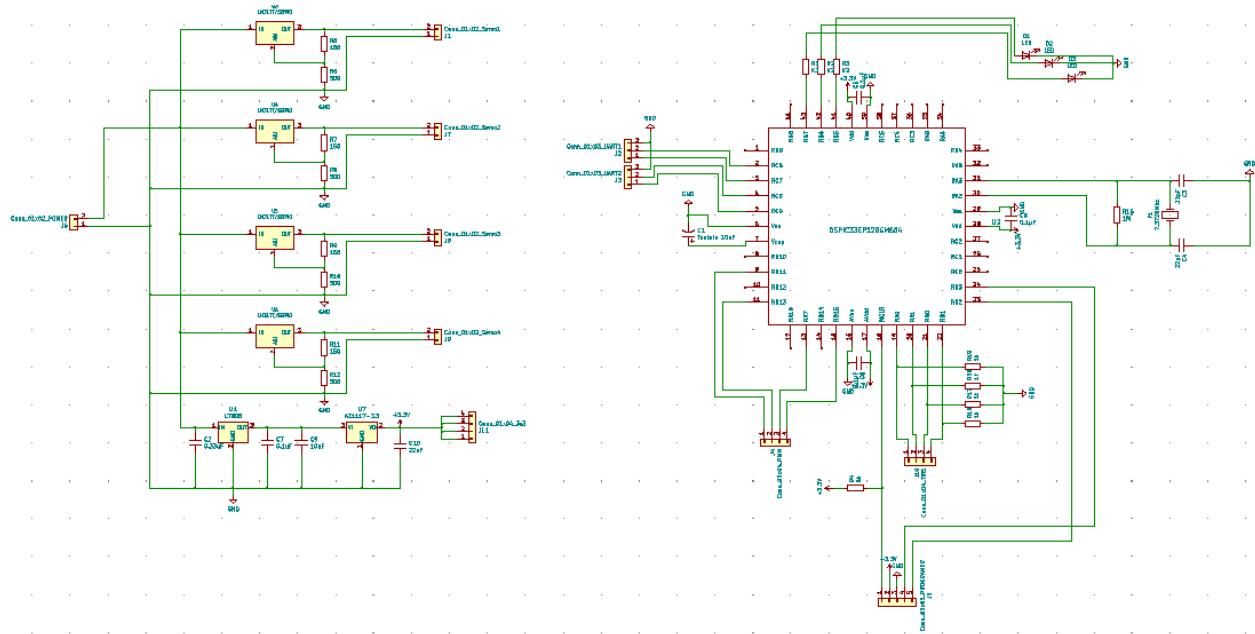


Figura 5.34: Diagrama esquemático completo.

#### 5.4.4. Conversión del diagrama esquemático a diagrama físico

Tal y como se ha descrito en el apartado anterior, el diseño lógico de la PCB se implementa mediante el diagrama esquemático, y su objetivo es el de describir las huellas y conexiones lógicas de los componentes; sin embargo, este diseño es de alto nivel y no es implementable directamente en términos físicos.

El siguiente paso tras completar el diagrama esquemático es transformar este diseño lógico en un diseño físico más cercano a la implementación real.

El diseño físico de una PCB debe ser obtenido directamente de la información establecida en el diseño lógico y, por lo tanto, se tiene que transformar el diagrama esquemático en un diagrama físico, en el cual se deben contemplar los aspectos físicos de los componentes y sus conexiones, además de sus aspectos lógicos.

El objetivo principal del diseño y diagrama físico es el de plasmar la realidad física de los componentes y sus conexiones a partir de un diagrama esquemático, en el cual no se contemplan los aspectos físicos para simplificar el diseño inicial. En general, el diseño y diagrama físico es más complejo y difícil de comprender. Por ello, normalmente la primera etapa del diseño comienza con el diseño lógico y diagrama esquemático.

En general, el proceso que se debe llevar a cabo para obtener el diagrama físico a partir de un diagrama esquemático consta de varios pasos:

- Asignación de huellas físicas a cada uno de los componentes lógicos.
- Generar un listado de redes en el cual se especifiquen las conexiones que existen entre todos los componentes.
- Importar ambos elementos anteriores a la herramienta de creación del diagrama físico y comenzar el diseño.

Tal y como se ha mencionado anteriormente, en primer lugar se debe asignar una huella física a cada uno de los componentes lógicos del diagrama esquemático. Este proceso se realiza en la herramienta “*Schematic Layout Editor*” incluida en KiCad, utilizando la opción de “asignar huellas a símbolos del sistema” disponible en la barra de herramientas situada en la parte superior de la ventana:

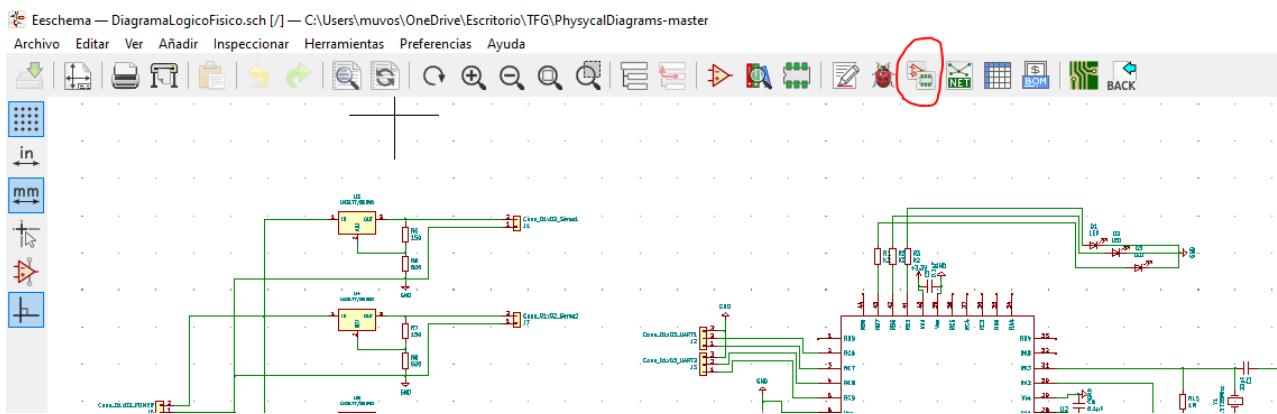


Figura 5.35: Herramienta de asignación de huellas.

Accediendo al menú de dicha herramienta (ver imagen 5.35), se encuentra una lista de los componentes del diagrama esquemático a los cuales se les debe asignar una huella física. Las huellas físicas que se deben asignar a cada componente lógico pueden ser seleccionadas de las extensa librerías que ofrece KiCad, o bien, ser diseñada y personalizada por el usuario.

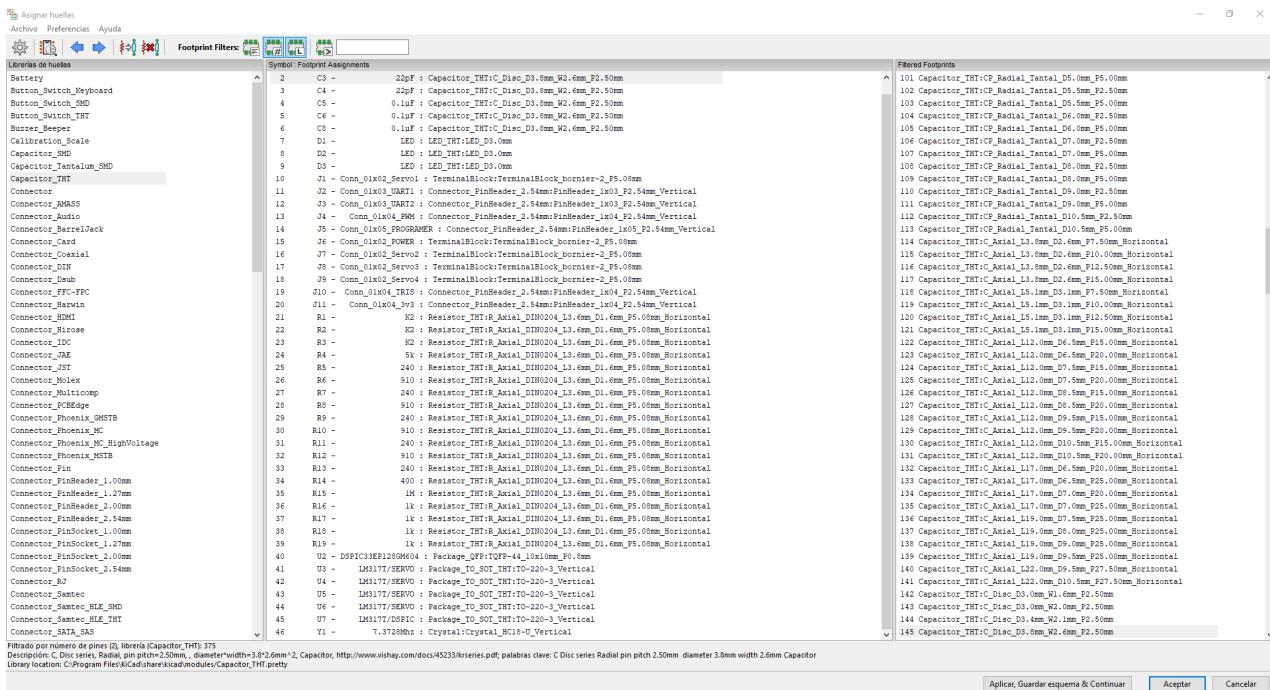


Figura 5.36: Ventana de asignación de huellas físicas.

En la imagen anterior (imagen 5.36) se muestra la ventana de asignación de huellas físicas, la cual está dividida en tres secciones:

- La sección izquierda muestra las librerías de componentes de KiCad.
- La sección central muestra los componentes lógicos del diagrama esquemático, seguidos de la huella física que tienen asignados.
- La sección derecha muestra las huellas físicas que cumplen los filtros establecidos dependiendo del componente lógico. Estos filtros suelen ser: nombre, número de pines y librería.

Como ejemplo, en la imagen anterior (imagen 5.36) se ha realizado una búsqueda dentro de la librería “capacitor THT”, ya que se quieren utilizar condensadores de agujero pasante en el diseño físico y, por lo tanto, en la sección derecha de la ventana se muestran las huellas susceptibles de ser usadas, filtradas por número de pines y librería.

Una vez se ha elegido la huella que se quiere utilizar en el diseño físico, esta se asigna al componente esquemático y puede ser visualizada (imagen 5.37):

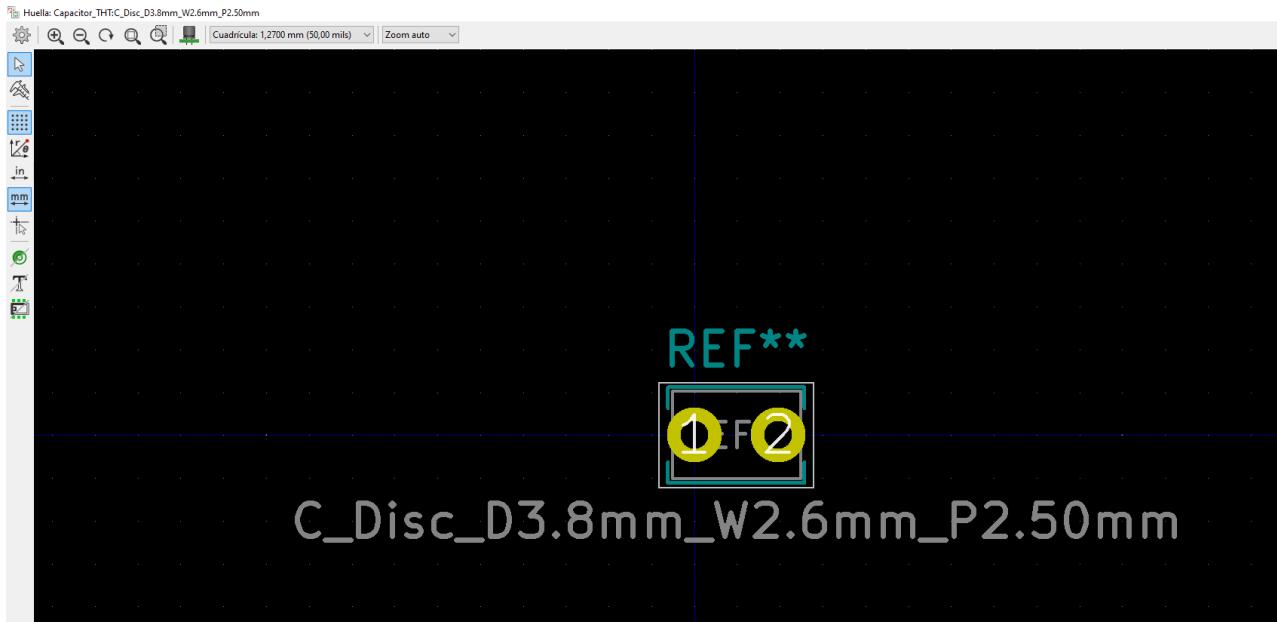


Figura 5.37: Huella física de un condensador usado en la PCB.

Existen numerosos aspectos que afectan a la decisión de qué huella física asignarle a cada componente lógico y principalmente depende del tipo de implementación que se vaya a realizar en la PCB. Normalmente, estas decisiones se deben tomar a través de la información técnica suministrada por el fabricante en los diversos *datasheets*.

Un ejemplo a destacar de lo recién mencionado es la huella que se ha asignado al microcontrolador dsPIC utilizado, el cual dispone de diversos encapsulados. Dichos encapsulados se muestran de forma detallada en el *datasheet*.

En este proyecto, se ha decidido utilizar el encapsulado de 44 pines de soldadura superficial y de tipo “*Thin Quad Flat Package*” (TQFP), cuya descripción en el *datasheet* es la siguiente (ver imagen 5.38):

### Pin Diagrams

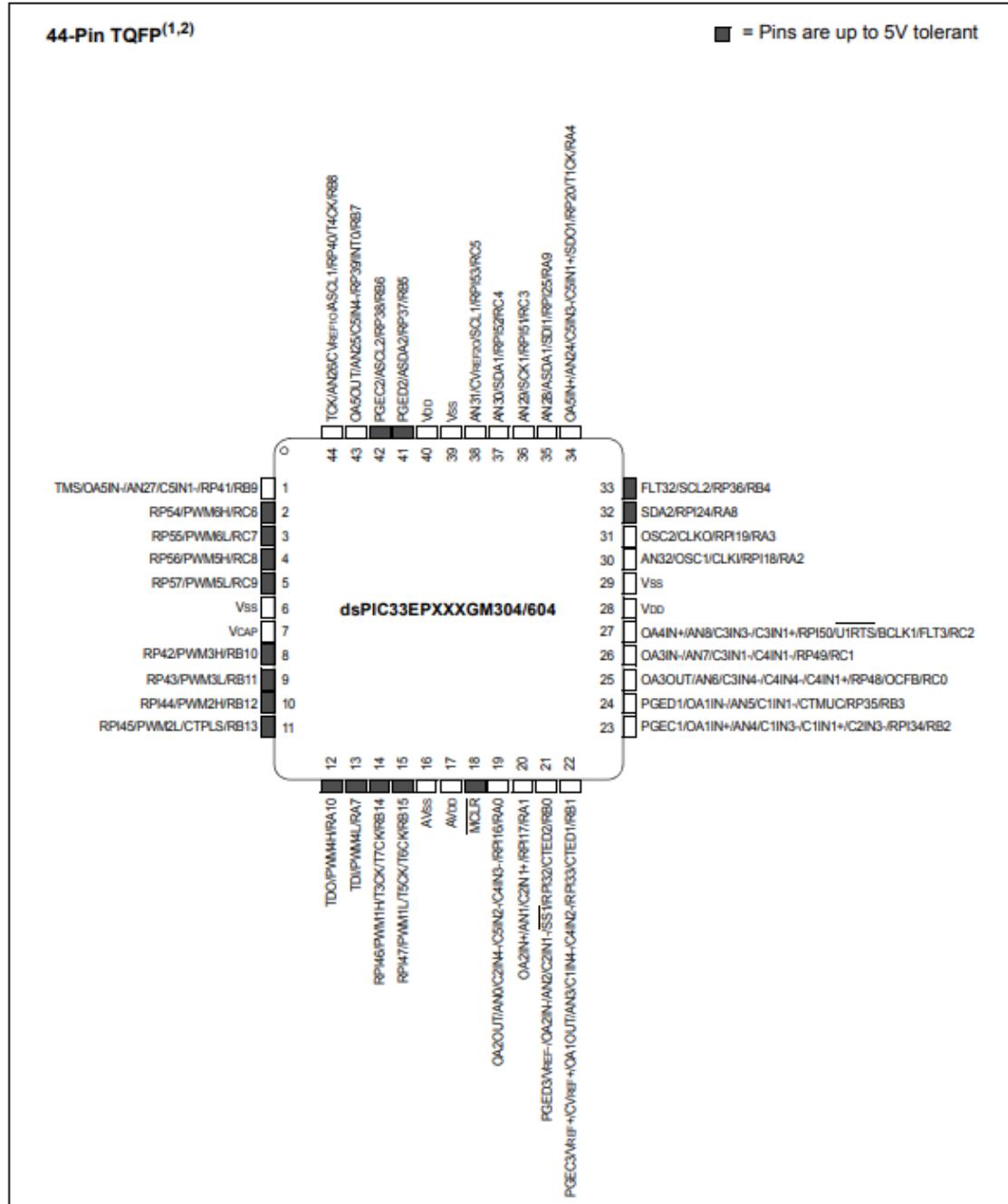


Figura 5.38: Encapsulado elegido para el microcontrolador [29].

La huella asignada al componente lógico es la siguiente (ver imagen 5.39) y se encuentra en la librería QFP incluida por defecto en KiCad:

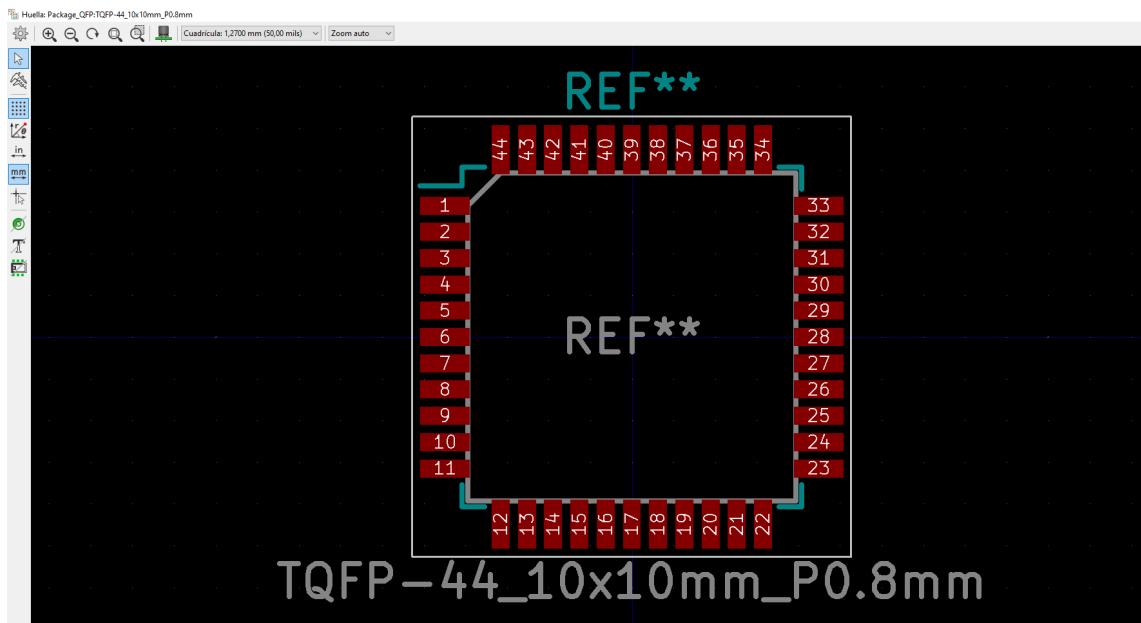


Figura 5.39: Huella física asignada al microcontrolador.

Cabe destacar que muchas de las huellas pueden ser usadas para distintos componentes de distintos fabricantes, ya que los encapsulados suelen ser similares. Sin embargo, para evitar errores, siempre se deben realizar comprobaciones con respecto a las dimensiones, número de pines, etc.

Una vez se ha realizado el primer paso, se debe generar un listado de redes de los componentes lógicos usados, sus huellas físicas asignadas y las conexiones existentes entre todos ellos. Este listado de redes se genera usando la herramienta “Generar listado de redes” disponible en la barra de herramientas de la parte superior de la ventana:

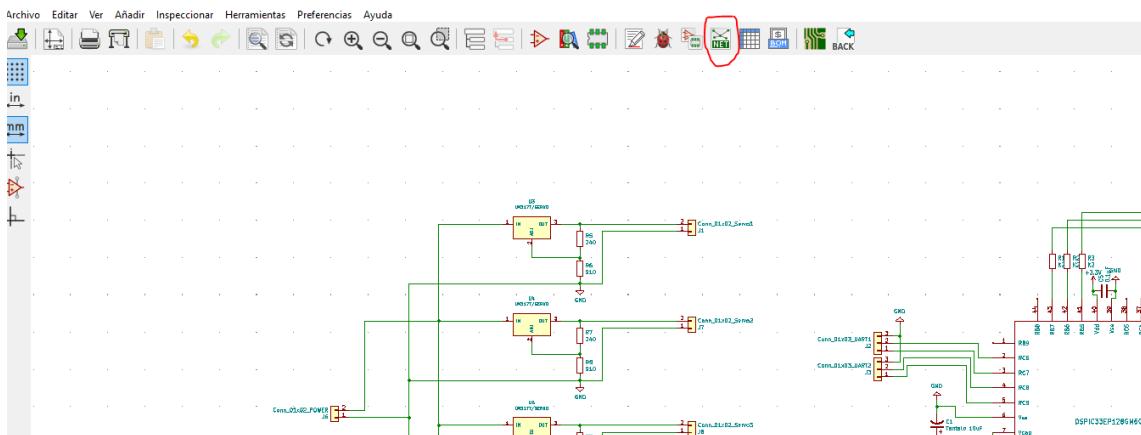


Figura 5.40: Herramienta de generado de listado de redes.

Utilizando la herramienta señalizada anteriormente (ver imagen 5.40) se puede generar un archivo con extensión “.net” que almacena el listado de componentes y conexiones del diagrama esquemático. Su contenido es de la siguiente forma (ver imagen 5.41):

```
(components
  (comp (ref R15)
    (value 1M)
    (footprint Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P5.08mm_Horizontal)
    (datasheet ~)
    (libsource (lib Device) (part R) (description Resistor))
    (sheetpath (names /) (tstamps /))
    (tstamp 5EA0CFA9))
  (comp (ref R1)
    (value K2)
    (footprint Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P5.08mm_Horizontal)
    (datasheet ~)
    (libsource (lib Device) (part R) (description Resistor))
    (sheetpath (names /) (tstamps /))
    (tstamp 5E722A02))
  (comp (ref C1)
    (value "Tantalum 10uF")
    (footprint Capacitor_THT:C_Disc_D3.8mm_W2.6mm_P2.50mm)
    (datasheet ~)
    (libsource (lib Device) (part CP1) (description "Polarized capacitor, US symbol"))
    (sheetpath (names /) (tstamps /))
    (tstamp 5E88CB39)))

```

Figura 5.41: Archivo de listado de redes.

Como último paso necesario para poder transformar el diagrama esquemático a diagrama físico se debe importar el listado de redes generado en el paso anterior a la herramienta de diseño físico. En este caso, la herramienta elegida ha sido “PCBnew” y se encuentra incluida dentro de KiCad. Esta herramienta se puede ejecutar usando la barra de herramientas de la parte superior (ver imagen 5.42):

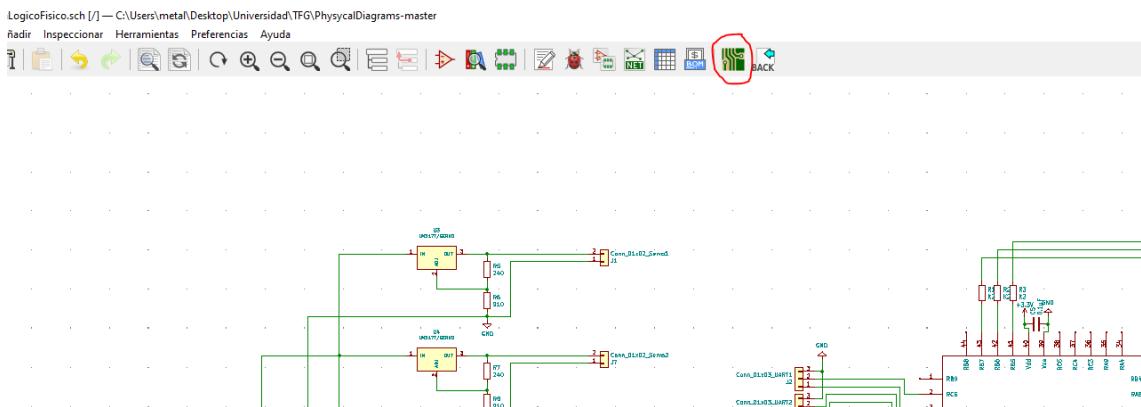


Figura 5.42: Acceso directo a la herramienta “PCBnew”.

Una vez se ha accedido a “PCBnew”, se debe importar el listado de redes generado anteriormente, usando el ícono designado para ello en la barra de herramientas superior (ver imagen 5.43):

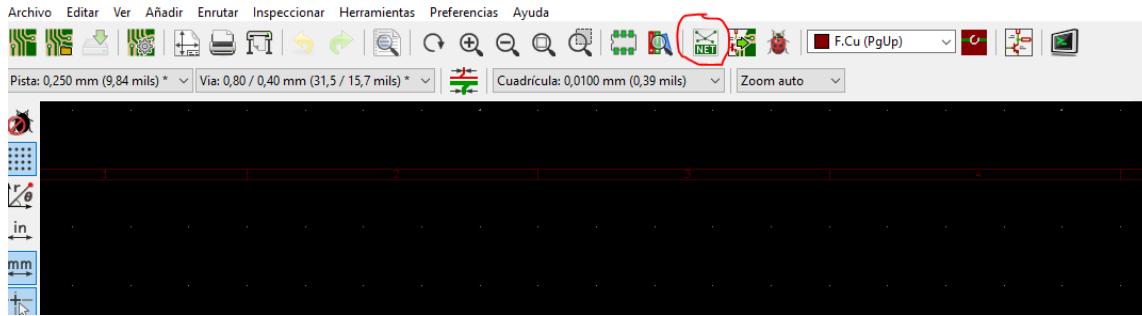


Figura 5.43: Herramienta de importado de listado de redes.

Para importar el listado de redes generado anteriormente, basta con buscar la ubicación del archivo, seleccionarlo y hacer clic en “actualizar PCB”. Al hacer esto, todos los componentes del diagrama esquemático son importados a “PCBnew”, y su apariencia es la huella física asignada anteriormente:

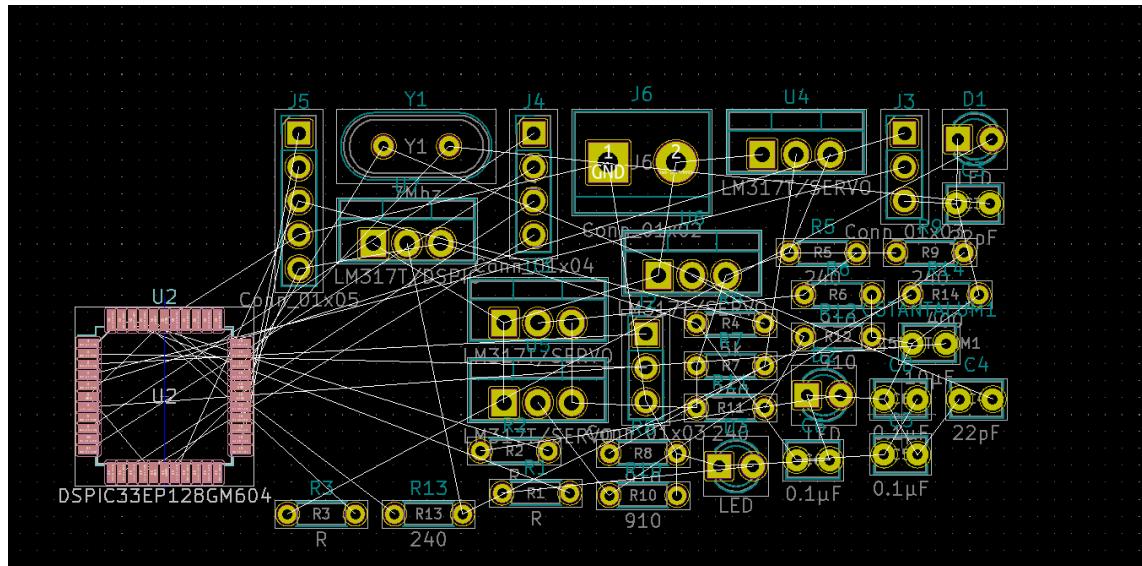


Figura 5.44: Situación inicial del diseño nada mas importar los componentes físicos.

Inicialmente, se muestra la huella física de los componentes y sus conexiones lógica pero se encuentran desordenados. Es recomendable reorganizarlos para verificar si todos ellos han sido importados correctamente.

El primer paso para comenzar el diseño físico de la PCB es distribuir los componentes por el plano, tratando de visualizar cuál va a ser la ubicación futura de los componentes en la placa de circuito impreso y cuál van a ser las dimensiones de la misma.

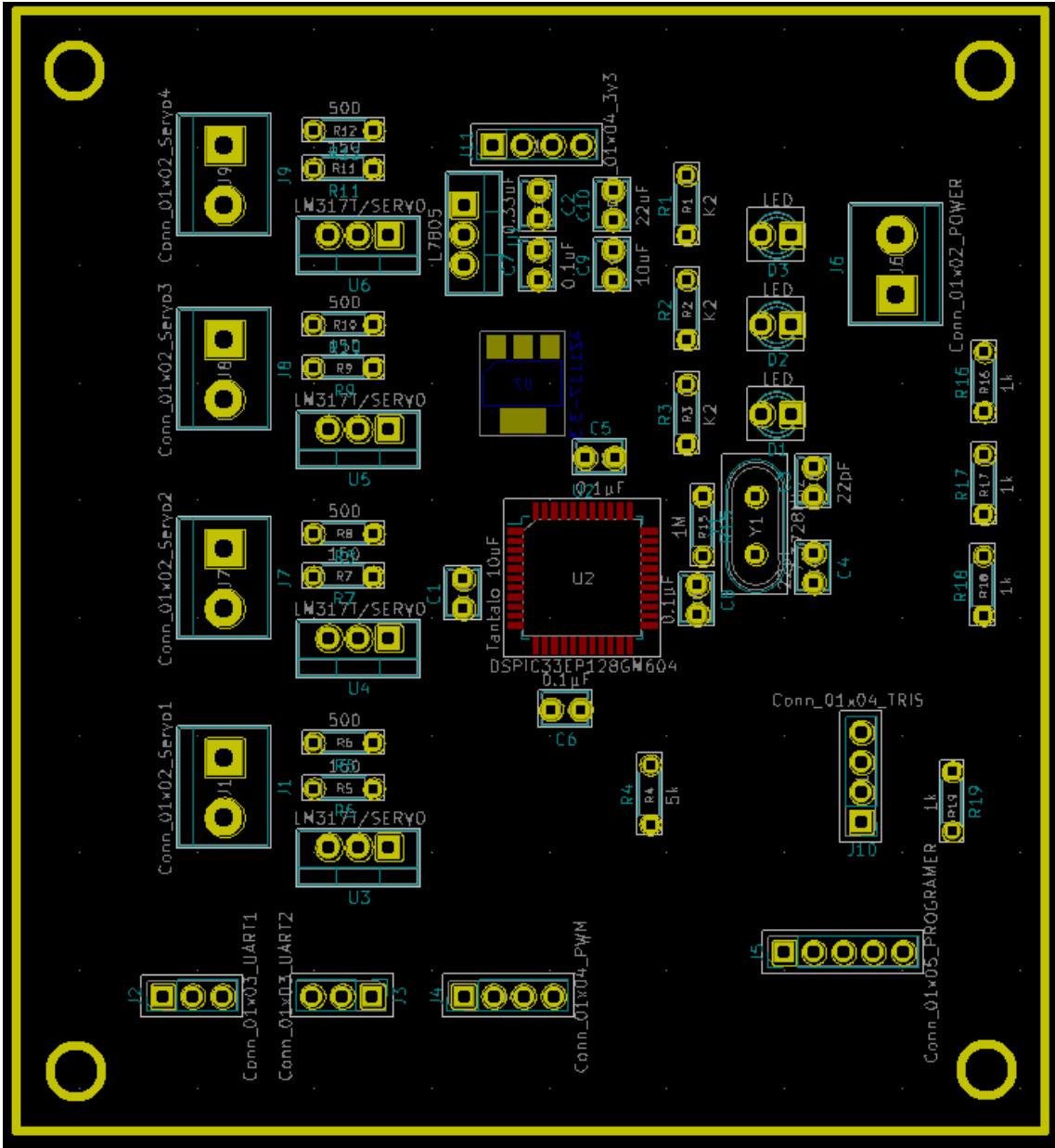


Figura 5.45: Distribución inicial de los componentes.

Es conveniente definir un contorno a la placa, el cual puede ser incluido en el diagrama usando la herramienta de dibujado de líneas y la herramienta de medición de distancias.

Por último, es importante remarcar que la distribución de los componentes queda a elección del diseñador de la PCB. Sin embargo, se debe de tratar de tener en cuenta factores como el tamaño deseado para la PCB, la localización de los componentes con respecto a sus conexiones cercanas, el proceso de fabricación a usar, etc.

### 5.4.5. Conexionado de los componentes mediante pistas

Una vez se ha creado el contorno de la PCB y se han distribuido los componentes de la forma deseada, se deben realizar las conexiones físicas entre los componentes.

Dado que se trata de una placa de circuito impreso, las conexiones lógicas entre los componentes se corresponden con pistas de cobre en el diagrama físico.

El proceso de conexionado mediante pistas se denomina “enrutado” y puede ser realizado de manera automática o manual. La complejidad del proceso de enrutado puede ser más o menos elevada en función del numero de componentes, capas que se utilicen para pistas, ubicación de los componentes, dimensiones de la PCB, etc. Se considera que el proceso de enrutado ha sido completado con éxito cuando todas las conexiones lógicas han sido realizadas y no existen conflictos o choques entre las pistas, así como soldaduras no realizablemente.

Durante el proceso de enrutado de las pistas de una PCB es habitual combinar herramientas de enrutado automático con enrutado manual, dado que, normalmente, las herramientas de enrutado automático suelen encontrar soluciones exitosas al proceso de enrutado, sin embargo no suelen ser óptimas y pueden requerir alguna modificación manual por parte del diseñador.

Para este proyecto se ha decidido realizar un proceso de enrutado íntegramente manual, debido a que, a pesar de haber intentado utilizar la herramienta “FreeRouting” de enrutado automático, no se ha obtenido un resultado apropiado y era bastante complejo.

El proceso de enrutado se ha realizado utilizando ambas capas de la PCB, esto quiere decir que se han trazado pistas en la capa de soldadura y en la capa de componentes. Esta decisión se ha tomado principalmente debido a que en esta PCB se usan componentes de tipo “THT” y “SMD”, por lo tanto es completamente necesario incluir pistas en la capa de componentes y capa de soldadura.

En relación con lo anterior, las dos capas mencionadas anteriormente cumplen una función específica:

- La capa superior o de componentes contiene las pistas de comunicación del microcontrolador, ya que este componente es de tipo “SMD”.
- La capa inferior o de soldadura contiene las pistas de alimentación de la PCB, en la cual se han realizado las soldaduras de los componentes “THT”.
- Para comunicar las pistas de ambas capas se han realizado vías de conexión en los casos necesarios. Un ejemplo de su uso está en el conexionado de los pines del microcontrolador en la capa superior con las pistas de los conectores, los cuales se encuentran soldados en la capa inferior de soldadura.

Otro de los aspectos claves a la hora de enrutar una PCB es escoger un ancho de pista adecuado. Algunos de los factores que afectan a esta decisión son los siguientes:

- Intensidad de corriente y voltaje que conducirá la pista. Distinción entre pistas de alimentación y de comunicación de señales digitales.

- Requerimientos de tamaño de la PCB, ya sea por número de pistas, espacio disponible, tamaño de los pines de conexión de componentes, etc.
- Limitaciones físicas y de precisión del proceso de fabricación de la PCB.
- Otros factores como la resistividad del material de la pista, el aumento de temperatura máximo tolerado por la misma o su longitud aproximada, son factores determinantes para el ancho de la pista.

KiCad incluye una herramienta denominada “PCB Calculator” la cual permite realizar cálculos sobre diversos aspectos relacionados con la PCB, entre ellos, el ancho de pista. Esta herramienta realiza el cálculo del ancho de pistas en función de los factores mencionados anteriormente y su interfaz es la siguiente (imagen 5.46):

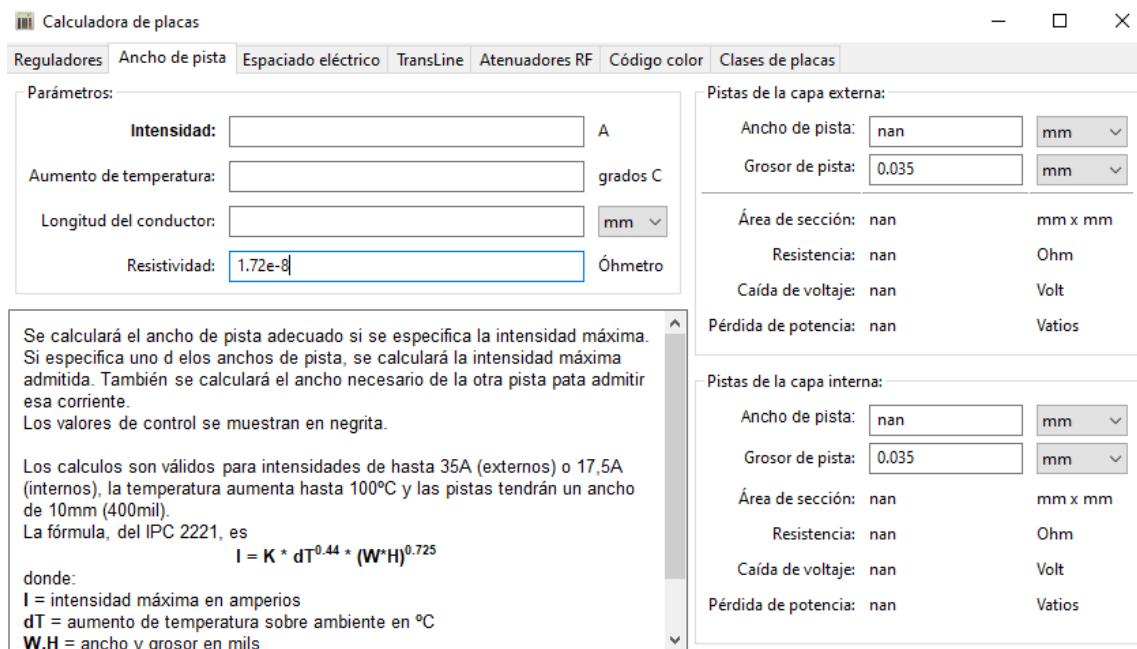


Figura 5.46: Ventana principal de “PCB Calculator”.

Mediante la herramienta anterior (imagen 5.46) se puede realizar el cálculo para los dos tipos de pistas usadas en esta PCB: pistas de alimentación y pistas de comunicación. Para el cálculo de ambos tipos, se asume un aumento de temperatura máximo de 10 °C, una longitud del conductor de 20cm y un valor de resistividad de  $1,72 \cdot 10^{-8} \rho$ .

En primer lugar, se considera que las pistas de alimentación de la PCB conducen 2A de intensidad y un voltaje de 9V máximo. Teniendo en cuenta dichos datos, el ancho de pista obtenido es el siguiente (imagen 5.47):

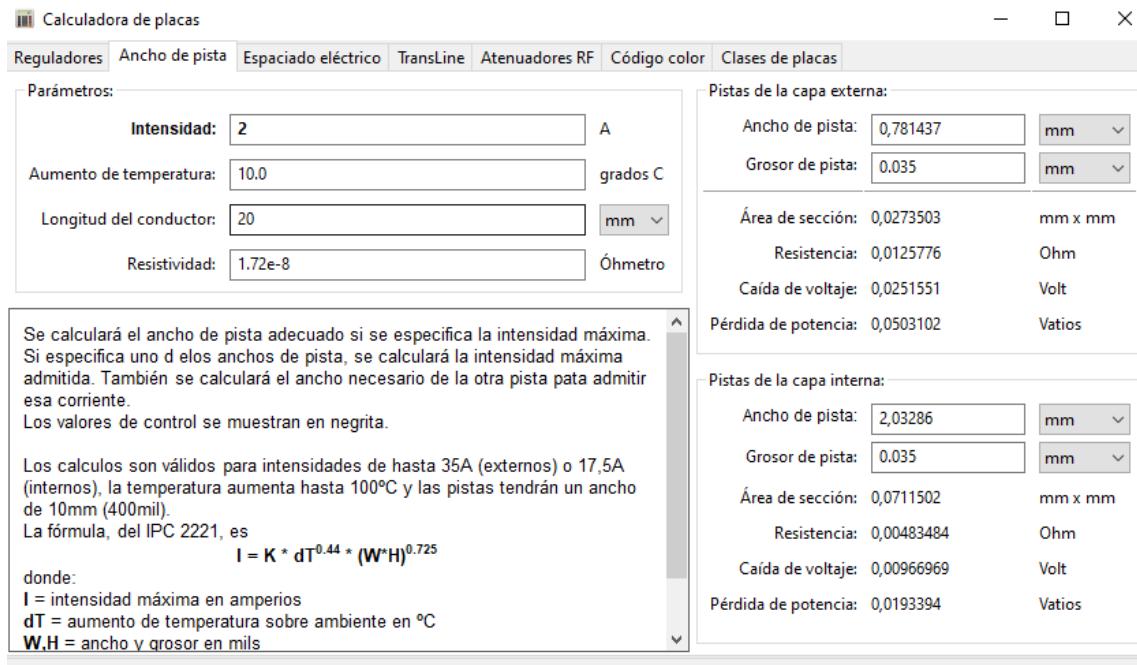


Figura 5.47: Cálculo del ancho de pistas de alimentación.

Se obtiene una ancho de pista de  $0,78\text{mm}$  para las pistas de alimentación, por comodidad de approxima este valor de ancho a  $0,8\text{mm}$ .

En segundo lugar, se considera que las pistas de comunicación del microcontrolador, conducen  $0,25\text{A}$  y  $3,3\text{V}$  máximo. Teniendo en cuenta dichos datos, el ancho de pista obtenido es el siguiente (imagen 5.48):

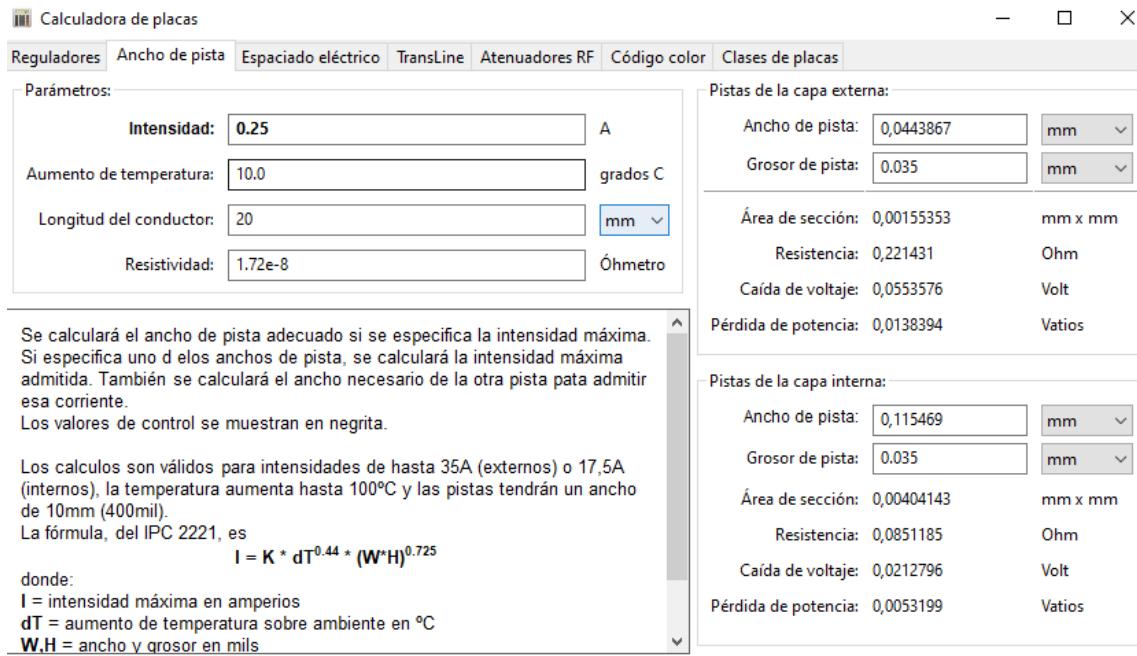


Figura 5.48: Cálculo del ancho de pistas de comunicación.

Se obtiene un ancho de pista de  $0,044\text{mm}$  para las pistas de comunicación, sin embargo, se ha decidido no trazar pistas con un ancho menor a  $0,4\text{mm}$ , debido principalmente a que se pueden producir errores en el proceso de fabricación.

En este momento, cabe destacar que el proceso de fabricación llevado a cabo para construir la placa, es de carácter artesanal, no industrial.

Asumiendo un ancho mínimo de pista de  $0,4\text{mm}$ , las pistas de comunicación están sobredimensionadas por motivos justificados y por lo tanto se obtiene el siguiente cálculo (imagen 5.49) :

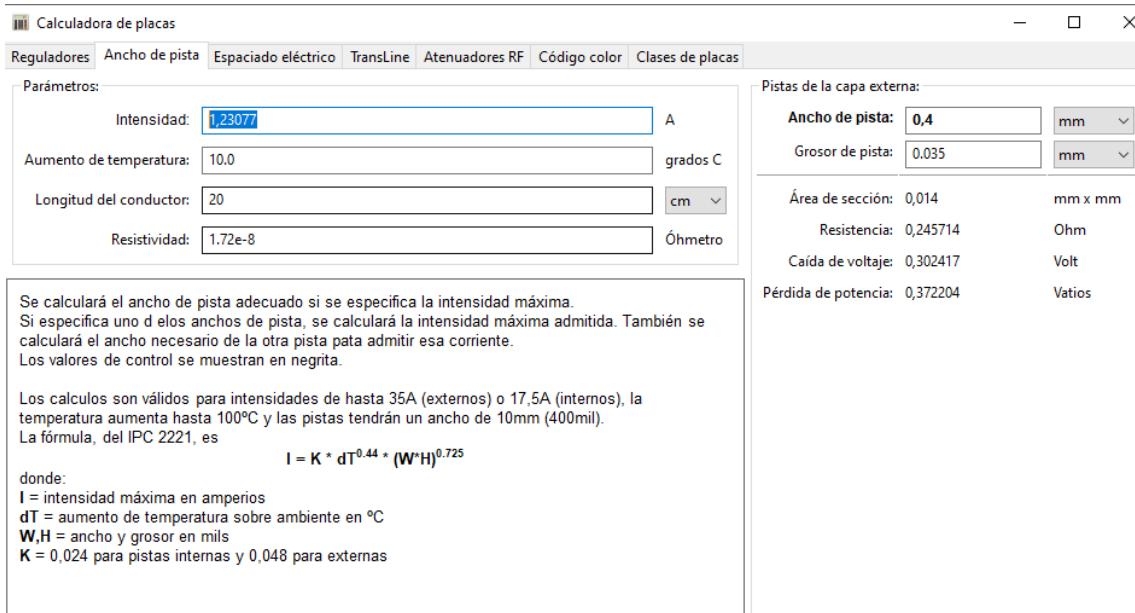


Figura 5.49: Cálculo inverso del ancho de pistas de comunicación.

Las pistas de comunicación tienen finalmente un ancho de  $0,4\text{mm}$  y debido a su sobredimensionado, soportan una corriente de  $1,23\text{A}$ , la cual se sitúa muy por encima de la corriente que circulará por las mismas ( $0,25\text{A}$ ).

A continuación se muestra la distribución final de los componentes físicos dentro del contorno de la PCB, aún sin haber trazado las pistas de conexión:

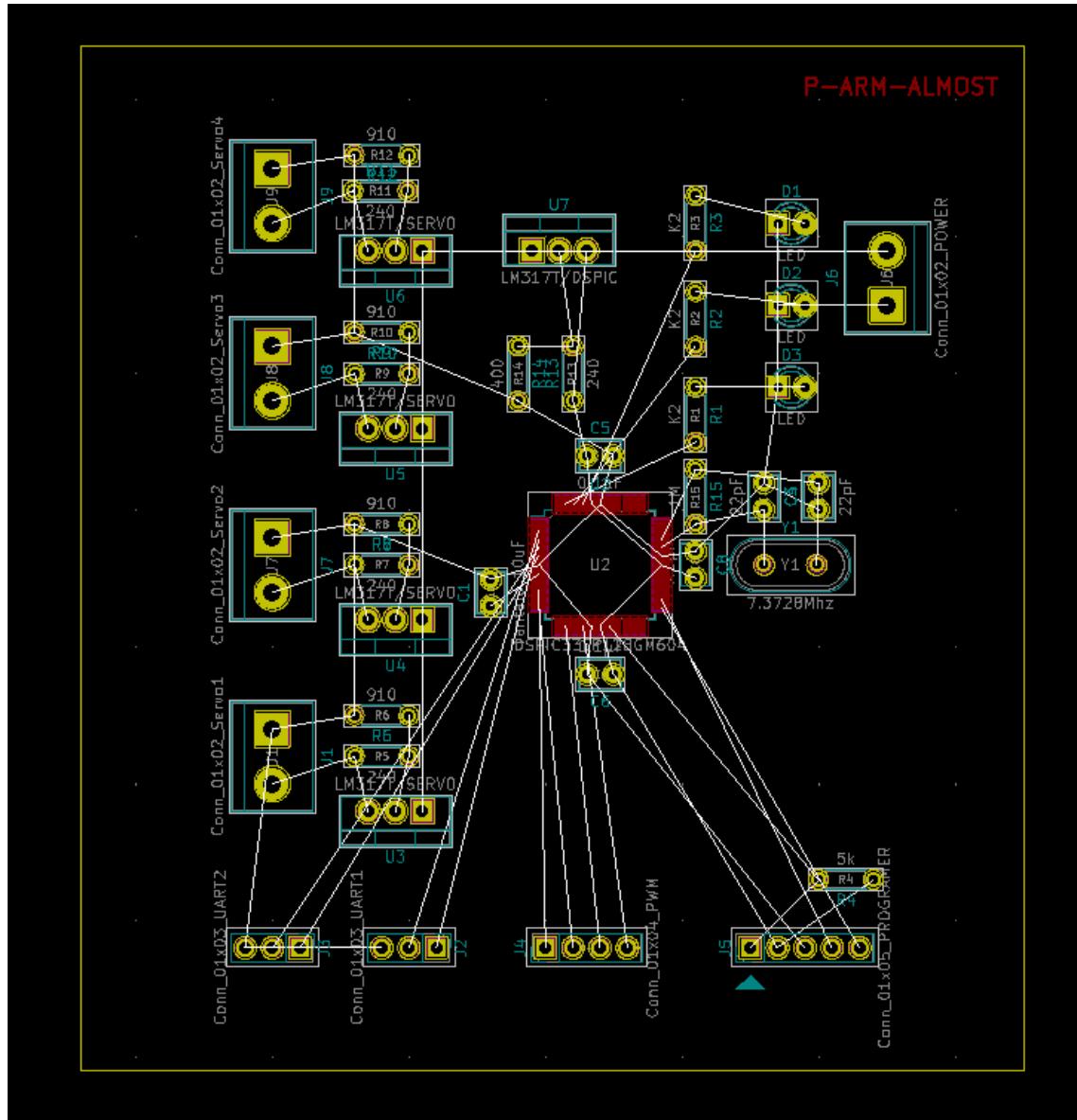


Figura 5.50: Distribución final de los componentes físicos.

Tal y como puede verse en la imagen anterior (imagen 5.50), las líneas blancas representan las conexiones físicas entre los componentes y por lo tanto, deben ser sustituidas por pistas de cobre. En este diagrama se han generado conexiones inexistentes en el diagrama esquemático, ya que se contemplan conexiones físicas que a nivel lógico no son necesarias.

Tras realizar el proceso de enrutado en ambas caras, el diagrama físico final obtenido es el siguiente (ver imagen 5.51):

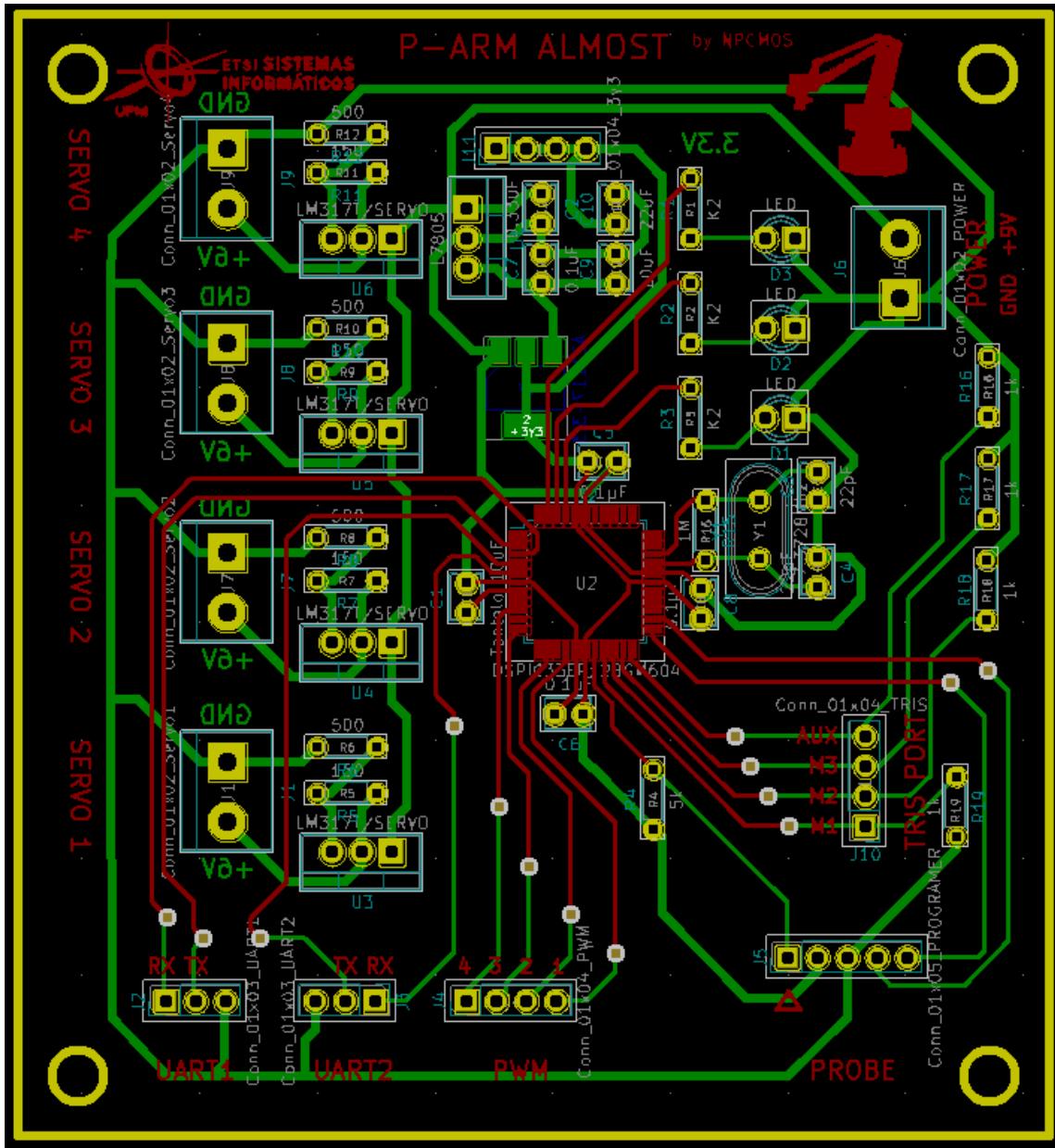


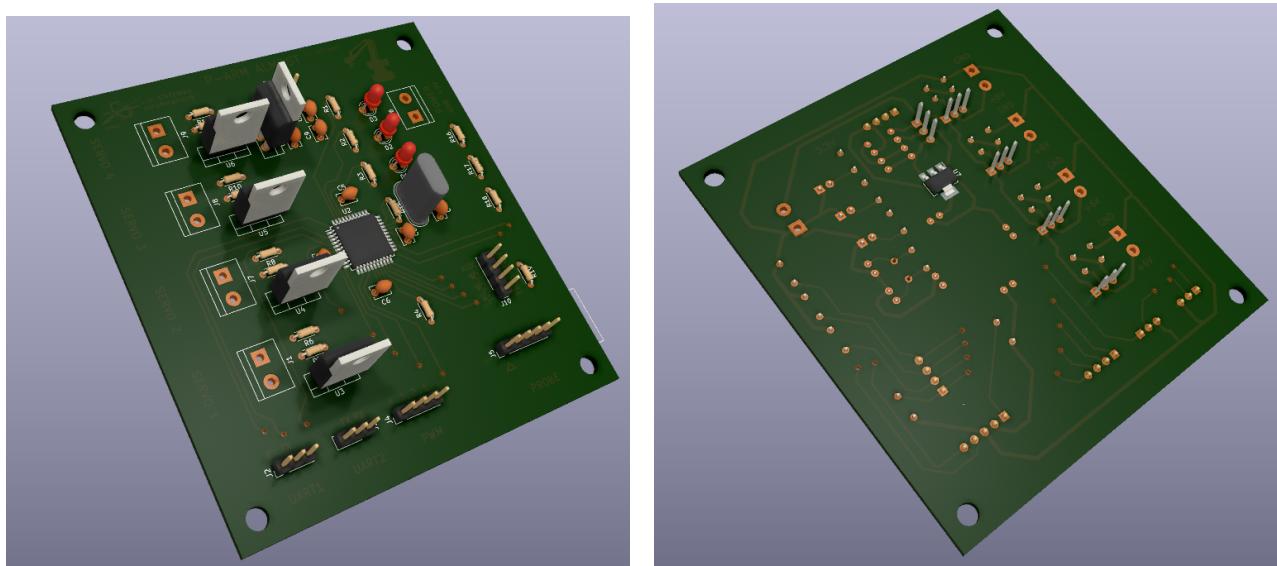
Figura 5.51: Diagrama físico final.

Cabe destacar varios aspectos:

- Las pistas de color rojo se corresponden con la cara frontal o de componentes.
- Las pistas de color verde se corresponden con la cara trasera o de soldadura.
- Se han incluido marcas de serigrafiado adecuadas para identificar correctamente la PCB y su interfaz.
- Se han incluido orificios de mecanizado para la futura sujeción de la PCB.
- En la parte superior izquierda y derecha de la capa frontal se han añadido los logos de la ETSISI y el *pArm*.

- En la parte central de la capa frontal se ha incluido el nombre de la placa (P-ARM ALMOST) y el nombre del grupo de ingenieros (NPCMOS).

Utilizando el visualizador 3D de KiCad se puede obtener un representación cercana a la realidad de como será la PCB al fabricarse (ver imagen 5.52) :



(a) Capa frontal

(b) Capa trasera

Figura 5.52: Representación 3D del diseño físico.

#### 5.4.6. Verificaciones realizadas al diseño lógico y físico

Dado que el proceso de diseño de la PCB es uno de los elementos críticos dentro del proyecto, es necesario llevar a cabo una serie de verificaciones durante dicho proceso para minimizar el riesgo de fallos.

En primer lugar, se deben realizar las verificaciones del diseño lógico de la PCB, ya que es el primer diagrama que se realiza. Como aporta información de las conexiones lógicas entre los distintos componentes de la PCB, las verificaciones a realizar irán destinadas a comprobar la corrección de dichas conexiones y de los componentes empleados.

A continuación, se muestra una lista de las principales verificaciones realizadas en el diagrama lógico:

- Verificar si se han incluido en la PCB todos los componentes deseados.
- Verificar si se ha incluido el circuito de alimentación de la PCB, y por lo tanto, comprobar si todos los componentes están alimentados correctamente.
- Verificar si se ha incluido el conexionado mínimo recomendado por el fabricante para asegurar un correcto funcionamiento del microcontrolador y sus periféricos.

- Verificar si el conexionado de los dispositivos periféricos, puertos de conexión y demás componentes es correcto y, además, se ha realizado a los pines adecuados del microcontrolador.
- Verificar si se ha asignado una huella física correcta a cada uno de los componentes lógicos, la cual se utilizará posteriormente en el diagrama físico.

Tras realizar las revisiones anteriores se considera que el diagrama lógico está en un estado de madurez suficiente como para poder crear el diagrama físico a partir de él y, posteriormente, realizar también las revisiones del mismo.

Dichas verificaciones a realizar van destinadas a comprobar los aspectos físicos de los componentes, conexiones mediante pistas, requerimientos estructurales, dimensiones, etc., donde se comprueba:

- Verificar que todos los componentes del diagrama lógico aparecen en el diagrama físico al importar la lista de redes.
- Verificar que todas las conexiones lógicas entre componentes del diagrama lógico aparecen representadas en el diagrama físico.
- Verificar que todas las conexiones que aparecen en el diseño físico se corresponden con conexiones establecidas en el diagrama lógico.
- Verificar que todas las huellas físicas de los componentes lógicos son correctas y su distribución de pines es la deseada. Se debe contrastar esta información usando el *datasheet* de cada uno de los componentes.
- Verificar que todas las conexiones se han realizado, y por lo tanto, tienen una pista asignada.
- Verificar que todos los componentes están alimentados y que las pistas de alimentación tienen el tamaño adecuado.
- Verificar que todas las pistas de comunicación tienen el origen y destino adecuado, y que su ancho es el adecuado.
- Verificar que no existen pistas con un ancho menor de  $0,3mm$ , ya que el proceso de fabricación podría fallar por debajo de este tamaño.
- Verificar que la separación mínima entre pistas es como mínimo de  $0,3mm$  para pistas de alimentación y  $0,25mm$  para pistas de comunicación digital.
- Verificar que no existen pistas que hagan contacto no deseado con componentes de soldadura superficial bajo su superficie.
- Verificar que se han incluido marcas de serigrafía que permitan identificar a la PCB y su interfaz de forma clara.
- Verificar que se ha incluido el mecanizado de sujeción adecuado.

- Verificar que no existan pistas que contacten con partes metálicas del mecanizado de sujeción.
- Verificar que todas las soldaduras de los componentes SMD y THT sean realizables físicamente.
- Verificar que el conector de programación dispone de espacio suficiente para realizar la conexión de la sonda.
- Verificar que las vías y *pads* tienen un diámetro mínimo de 1mm con agujero de 0,5mm.

KiCad ofrece una herramienta llamada “Comprobar reglas de diseño” (ver imagen 5.53), la cual puede ser utilizada para facilitar el proceso de verificación del diseño físico. Utilizando esta herramienta se pueden verificar automáticamente factores como el ancho de pista mínimo permitido, separación mínima entre pistas, conexiones no realizadas, etc. A pesar de poder realizar estas verificaciones de forma automática, se han realizado todas las verificaciones de forma manual para también reducir el riesgo de fallos.

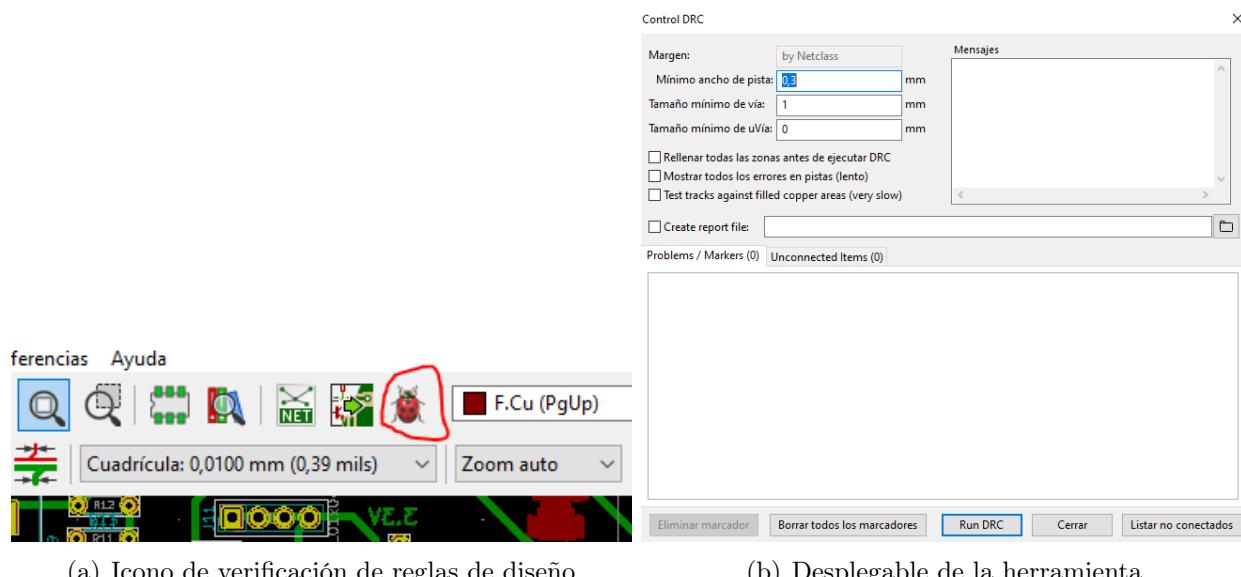


Figura 5.53: Herramienta de verificación de reglas de diseño.

Tras realizar todas las verificaciones anteriormente mencionadas en el diagrama físico, se han subsanado los errores encontrados previamente a realizar la fabricación de la PCB; se considera por lo tanto que el proceso de verificación del diseño ha sido exitoso.

#### 5.4.7. Construcción

Una vez se ha finalizado el proceso de diseño de la PCB, se comienza el proceso de fabricación de la misma. Dicho proceso consta de diversas etapas, las cuales van desde la preparación de los materiales hasta la obtención del prototipo final; todas estas etapas se detallan a continuación una por una.

En primer lugar, cabe destacar que el proceso de fabricación seleccionado para construir la PCB ha sido la fotolitografía. Dicho proceso consiste en transferir un patrón de un circuito desde una fotomáscara a una placa de prototipado positiva mediante una serie de procesos lumínicos y químicos. Posteriormente, una vez se tiene impreso el circuito en la placa de prototipado, se procede al taladrado de orificios y soldado de los componentes del circuito.

A continuación, se presentan los conceptos básicos empleados durante el proceso de fotolitografía y cuyo entendimiento es fundamental para comprender el proceso de fabricación:

- El proceso de fotolitografía parte necesariamente de una placa de prototipado de fibra de vidrio, la cual posee las siguientes capas de materiales:

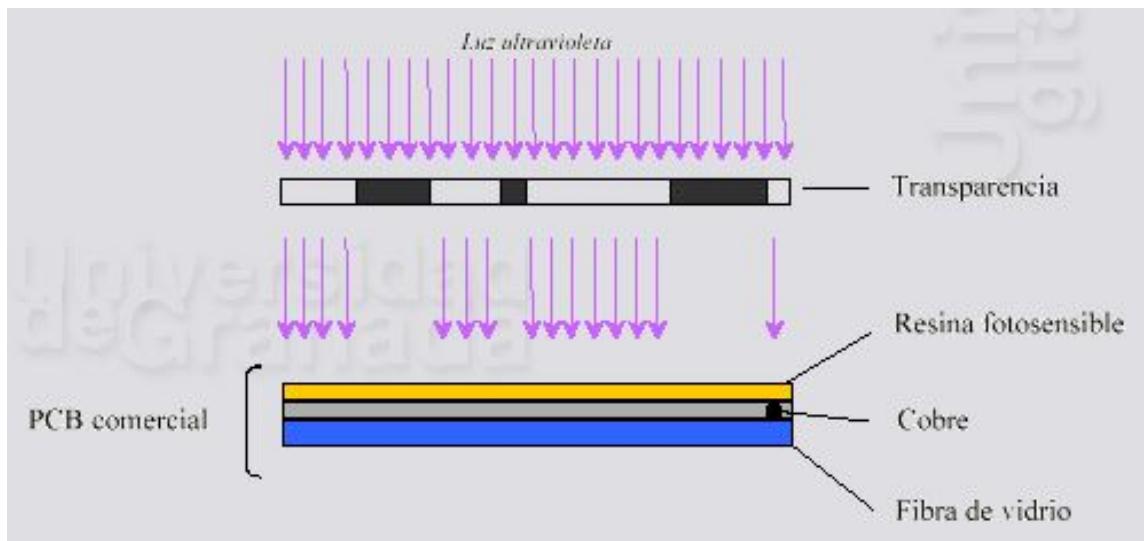


Figura 5.54: Estructura de la placa de prototipado [30].

Tal y como se puede ver en la imagen anterior (imagen 5.54), la placa de prototipado de fibra de vidrio posee tres capas: fibra de vidrio como material base encima de la cual hay una capa de cobre y, protegiendo la anterior, una capa de resina fotosensible a la luz ultravioleta.

El aspecto clave de esta estructura en capas recae en la capa de resina fotosensible, la cual tiene como objetivo capturar el patrón del circuito de la fotomáscara o transparencia. El funcionamiento de esta resina depende de si la placa de prototipado es positiva o negativa:

- En las placas de prototipado positivas, la resina fotosensible que es insolada con luz ultravioleta reaccionará correctamente con el revelador, y por lo tanto desaparecerá; mientras que la resina que no es insolada permanecerá tras el revelado.
- En las placas de prototipado negativas, la resina fotosensible que es insolada con luz ultravioleta se convierte en resistente al revelador y por tanto permanecerá tras el proceso de revelado; mientras que la resina que no es insolada con luz ultravioleta, reaccionará correctamente con el revelador, y por lo tanto desaparecerá.

En este proyecto se han utilizado placas de prototipado positivas de fibra de vidrio.

- Mediante el proceso de insolado con luz ultravioleta, la fotomáscara plasma el patrón del circuito a imprimir en la resina fotosensible:

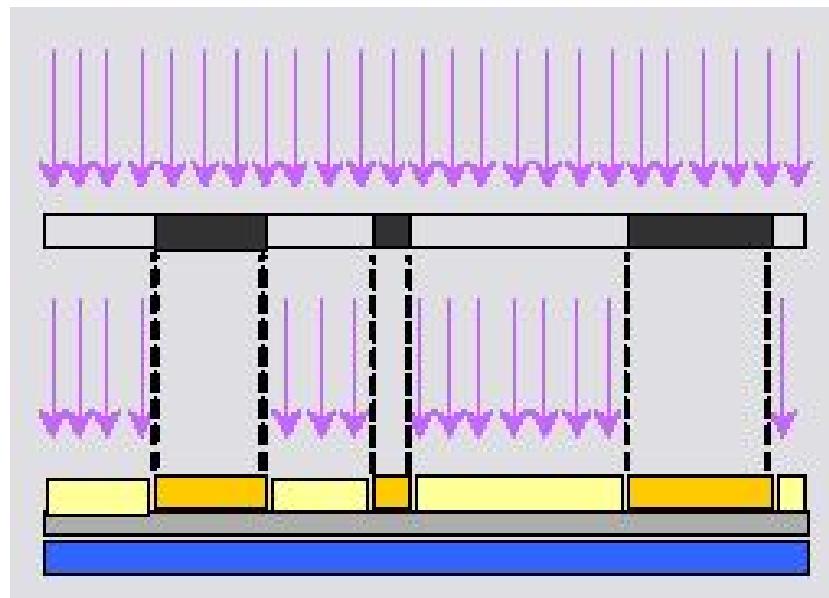


Figura 5.55: Tratado de las resinas mediante insolado [30].

Tal y como se puede ver en la imagen anterior (imagen 5.55), las superficies de la resina que han sido insoladas (amarillo claro), se convierten en reactivas al revelador, ya que en esa zona de la fotomáscara existe una transparencia. En las zonas en las cuales la transparencia es opaca e impide el paso de la luz ultravioleta la resina no se ve afectada y se mantiene no reactiva con el revelador.

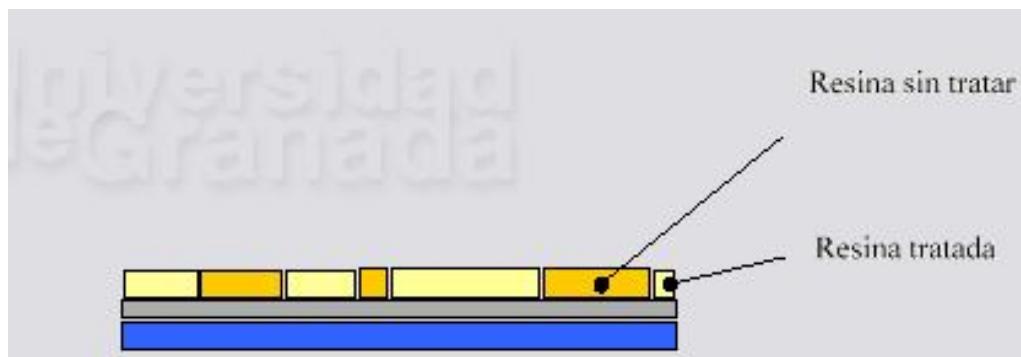


Figura 5.56: Resultado tras el insolado [30].

- Mediante el proceso de revelado, se elimina la resina que fue tratada en el proceso de insolación (ver imagen 5.57):

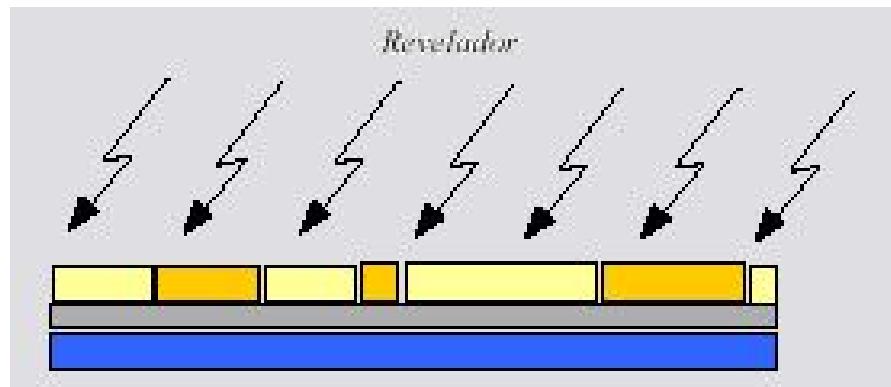


Figura 5.57: Proceso de revelado [30].

Tras someter la PCB al proceso de revelado, se obtiene el siguiente resultado (imagen 5.58):

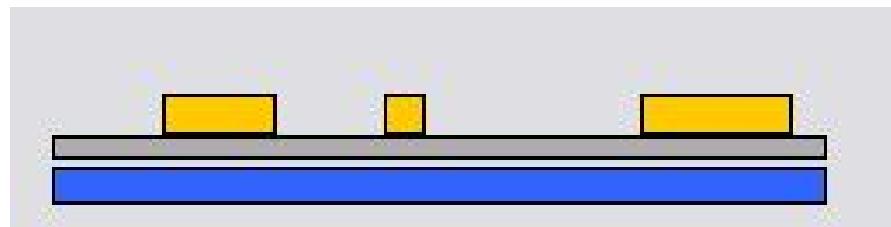


Figura 5.58: Resultado tras revelado [30].

El proceso de revelado elimina la resina que había sido insolada, ya que se trata de una placa positiva. En estas zonas, la capa de cobre queda expuesta y sin ninguna protección. Por el contrario, en las zonas que no fueron insoladas, el revelado no actúa y por lo tanto la resina inicial permanece protegiendo la capa de cobre.

- Mediante el proceso de atacado, se eliminan las superficies de la capa de cobre sobrantes, es decir, aquellas que no están protegidas por resina:

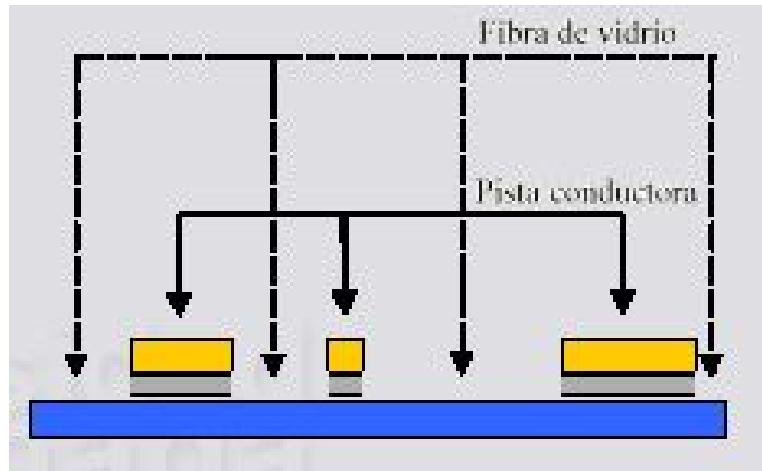


Figura 5.59: Resultado tras atacado [30].

Tal y como se puede observar en la imagen anterior (imagen 5.59), la capa de cobre solo permanece en los lugares que estaban protegidos por la resina inicial; este patrón coincide con el circuito de la fotomáscara. Para retirar la resina sobrante, basta con aclarar la PCB con alcohol, quedando las pistas del circuito perfectamente delimitadas e impresas en la placa de fibra de vidrio.

Existen otros procesos de fabricación para PCBs que ofrecen un resultado muy parecido al deseado en este proyecto, por ejemplo fabricación mediante máquina CNC; sin embargo, por motivos de disponibilidad de materiales y maquinaria, se ha decidido utilizar la fotolitografía.

A continuación se enumeran las diversas etapas del proceso de fabricación, ordenadas cronológicamente desde el comienzo del proceso hasta su finalización:

1. Generación de las fotomáscaras a partir del diagrama físico de la PCB.

El diagrama físico representa de forma precisa cual es la apariencia física de la PCB, las huellas de sus componentes, pistas de conexión, marcas de mecanizado, serigrafía, orificios y límites. Es precisamente toda esta información la que se quiere plasmar mediante fotolitografía en la placa de prototipado positiva que contendrá el circuito impreso.

Utilizando la herramienta “imprimir placa” disponible en KiCad, se puede generar las máscaras asociadas al diagrama físico:

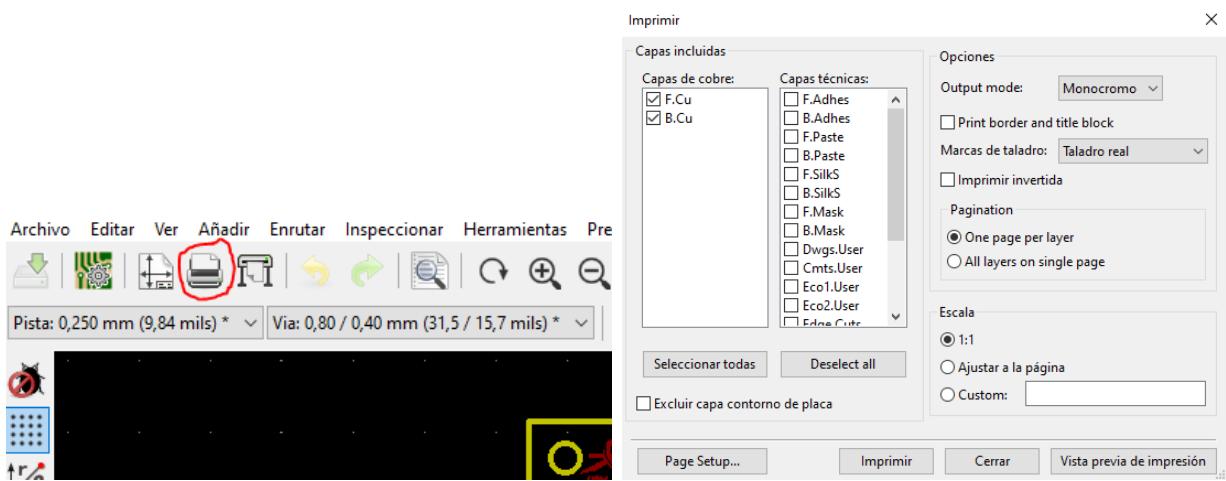
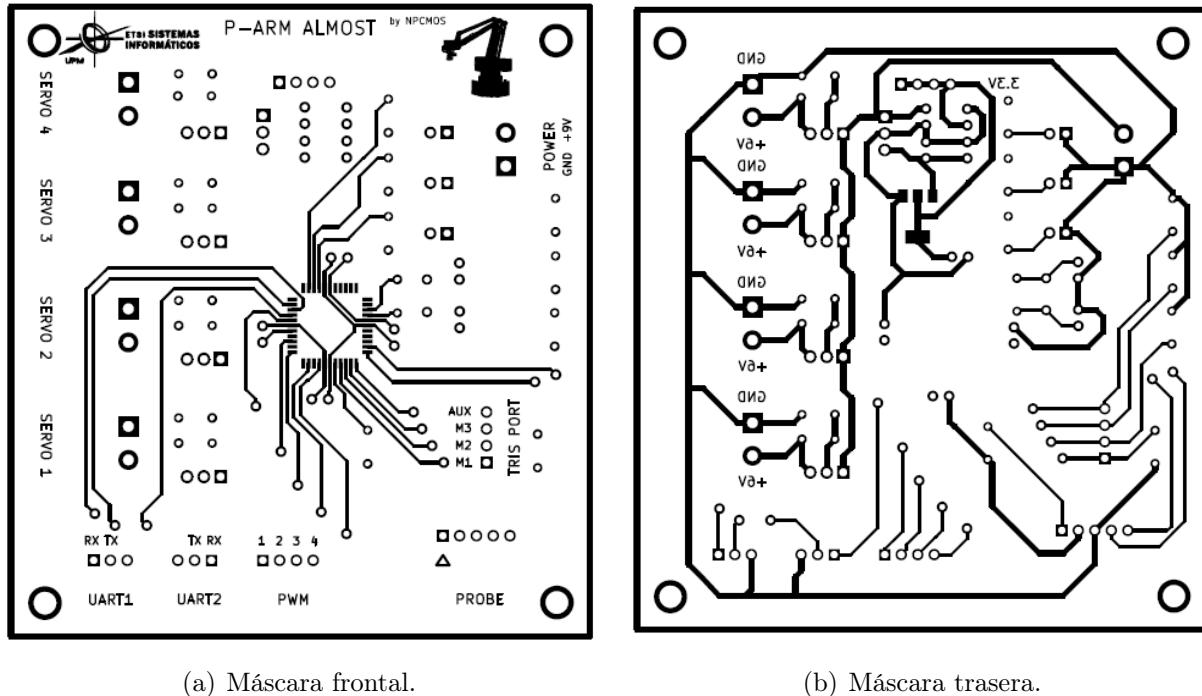


Figura 5.60: Herramienta de impresión de placas.

Las fotomáscaras obtenidas tras el uso de dicha herramienta son las siguientes (ver imágenes 5.61):



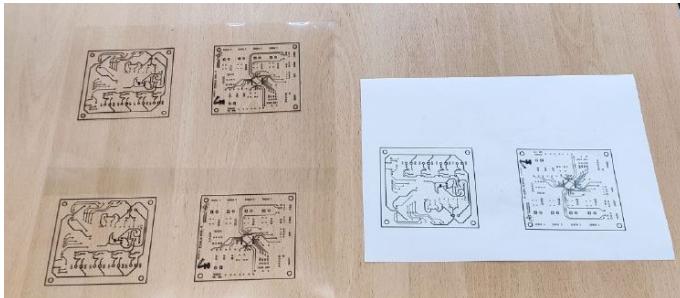
(a) Máscara frontal.

(b) Máscara trasera.

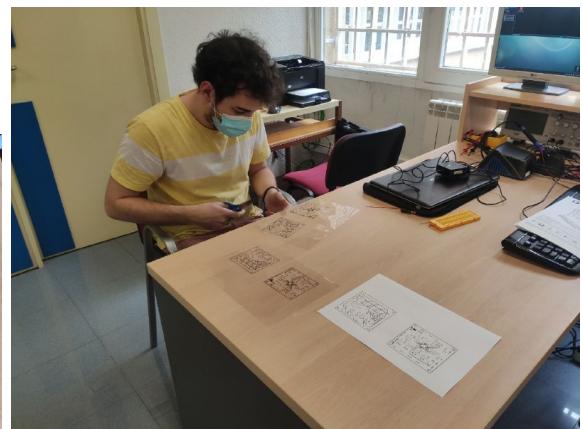
Figura 5.61: Fotomáscara final de la PCB.

Con el fin de poder utilizar dichas máscaras en el proceso de fotolitografía, es necesario imprimirlas en papel de transparencia, puesto que deben dejar pasar la luz en los lugares que no contienen el patrón del circuito a fotolitografiar.

Para imprimir dichas máscaras en papel de transparencia, primeramente se imprimen en papel convencional y posteriormente se fotocopian a un papel de transparencia, obteniendo el resultado siguiente (ver imagen 5.62):



(a) Máscaras de papel y sus transparencias.



(b) Recorte de las máscaras.

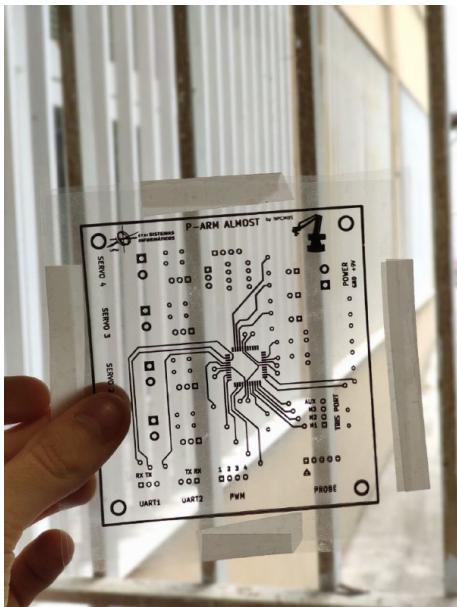
Figura 5.62: Proceso de impresión y recorte de las fotomáscaras.

Otro de los aspectos a destacar y que es vital en relación a la impresión de las transparencias es que, dado que todas las impresoras producen cierta distorsión en los documentos

que imprimen, se debe incluir una corrección de escala *XY*, en particular para este proyecto de 1,037 para calibrar correctamente la impresión. En caso contrario, se puede producir algún pequeño desajuste en el encaje de los componentes una vez esté fabricado el circuito.

Dado que el objetivo principal de dichas fotomáscaras es el de transferir el patrón del circuito de la PCB a la placa de prototipado positiva, se deben ensamblar de tal forma que permitan introducir en su interior la misma, para que posteriormente durante el proceso de insolación, se consiga fotolitografiar el circuito en ambas caras de la PCB.

A continuación se muestra el ensamblado final de la funda construida con las fotomáscaras:



(a) Funda de transparencias.



(b) Demostración de su capacidad de contener a la placa.

Figura 5.63: Fotomáscara final.

Cabe destacar que se han incluido dos transparencias superpuestas por cada cara de la fotomáscara para aumentar la opacidad que se genera al fotolitografiar la placa de prototipado positiva. Estas transparencias deben estar superpuestas con una exactitud muy elevada para impedir desfases en la impresión del circuito.

2. Una vez se ha completado el generado de la fotomáscara, se debe realizar el proceso de insolación de la placa de prototipado positiva. Durante este proceso, se realiza la transferencia del patrón del circuito desde la fotomáscara a la placa de prototipado mediante la exposición de la misma a luz ultravioleta.

A continuación se muestra el proceso de insolación de la placa de prototipado positiva, el cual se ha realizado usando una maquina insoladora:



(a) Placa de prototipado colocada dentro de la fotomáscara.

(b) Ajuste de la máquina de insolación.

Figura 5.64: Proceso de insolación de la PCB usando las fotomáscaras.

Tras el proceso de insolación, la placa de prototipado positiva habrá sido expuesta a la luz ultravioleta durante un minuto y, por lo tanto, el patrón del circuito habrá sido transferido a la superficie de resina de la placa de prototipado:

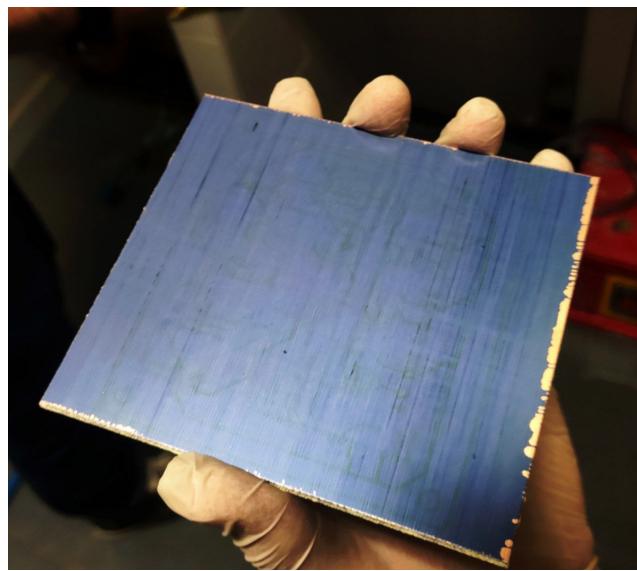


Figura 5.65: Placa de prototipado positiva insolada.

3. Cuando se ha completado el proceso de insolación de la placa de prototipado se debe llevar a cabo el proceso de revelado de la PCB.

El proceso de revelado es un proceso químico mediante el cual se consigue eliminar la resina protectora de la placa de prototipado que ha sido expuesta a la luz ultravioleta durante el proceso de insolación; esta resina que es eliminada coincide con la superficie de la PCB que no contiene pistas del circuito impreso, gracias a la fotomáscara.

Este proceso puede llevarse a cabo con un revelador universal indicado para realizar este tipo de labores, a continuación se muestra el usado en este proyecto:



Figura 5.66: Revelador universal usado.

El proceso de revelado de la PCB se lleva a cabo disolviendo el revelador anteriormente mencionado en agua y, posteriormente, sumergiendo la PCB en esta disolución:

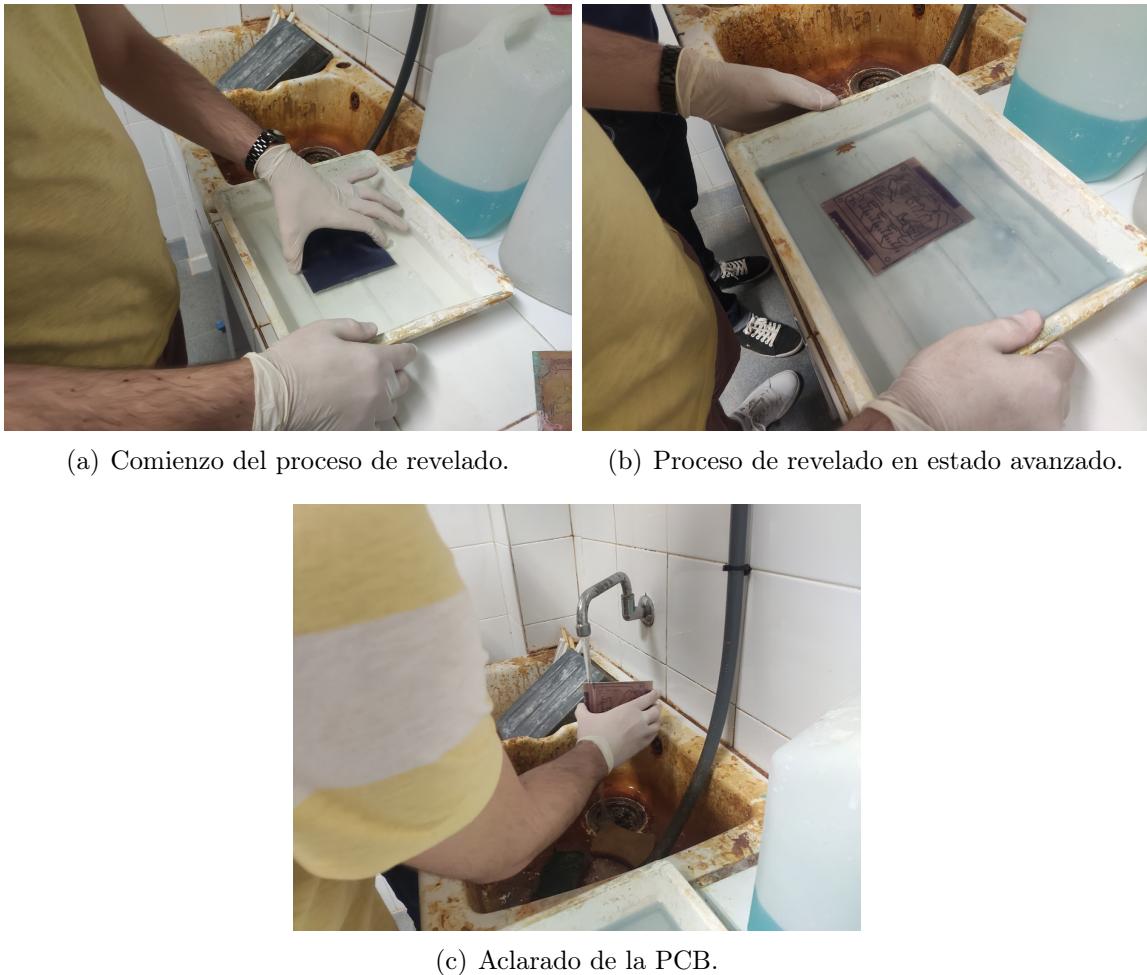


Figura 5.67: Proceso de revelado de la PCB.

Durante el proceso de revelado se tienen que generar movimientos delicados de la disolución reveladora para que esta haga un efecto adecuado y consiga eliminar la resina sobrante. Cuando el proceso ha finalizado, basta con aclarar la PCB con agua sin tocar el patrón del circuito que ha quedado revelado.

4. Tras revelar el patrón del circuito de la PCB, se debe realizar el proceso de atacado.

En la etapa anterior del proceso de fabricación, las zonas de la PCB que no fueron insoladas con luz ultravioleta quedan cubiertas de resina protectora; mientras que en las parte de la PCB que fueron expuestas a luz ultravioleta, la capa de cobre queda expuesta y sin cubrir por ningún tipo de resina ya que el proceso de revelado ha eliminado dicha resina sobrante.

La etapa de atacado del proceso de fabricación consiste en someter a la PCB a un proceso químico, mediante el cual se elimina el cobre de la placa de prototipado a través del atacado mediante una sal persulfato de amonio, el cual actúa de forma similar a un ácido. Las zonas de la PCB en las cuales el cobre está expuesto resultarán atacadas por el ácido y, por lo tanto, el cobre desaparecerá. Por el contrario, las zonas de la PCB en las cuales el cobre está protegido por una capa de resina, no serán atacadas tan

agresivamente por el ácido y el cobre permanecerá.

Al igual que en la etapa anterior, el atacado de la PCB se realiza a través de una disolución, en este caso de agua y persulfato de amonio.

A continuación, se muestran algunas imágenes (imágenes 5.68) del proceso de atacado mediante ácido:

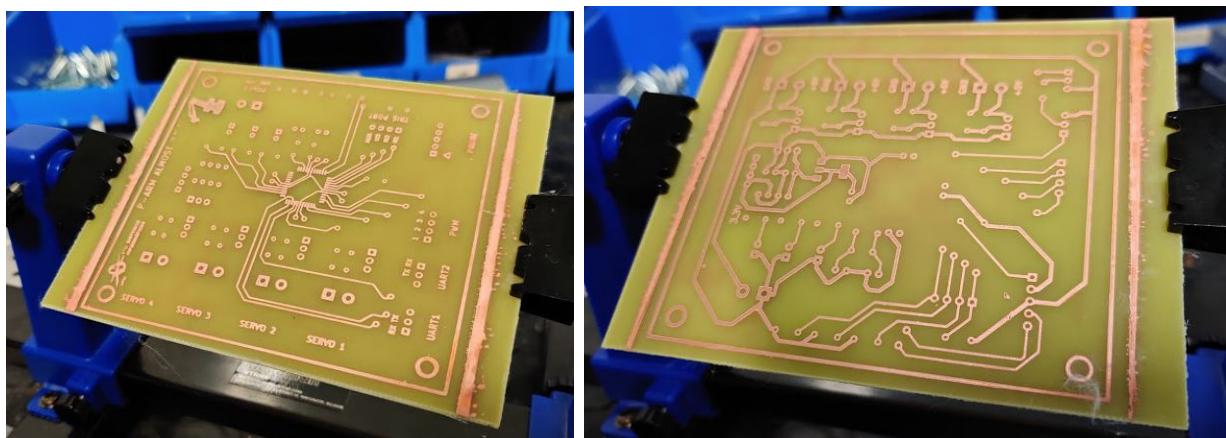


(a) Recipiente contenedor de la disolución de atacado. (b) PCB siendo sometida al proceso de atacado.

Figura 5.68: Proceso de atacado de la PCB.

Tras el proceso de atacado, el cual lleva entre una y dos horas, se debe aclarar y limpiar el circuito impreso con alcohol para eliminar los restos de resina sobrantes, así como las impurezas producidas por la disolución de atacado.

Al finalizar esta etapa del proceso de fabricación, se obtiene una PCB en la cual aparece el patrón del circuito de las fotomáscaras impreso en cobre:



(a) Parte frontal de la PCB.

(b) Parte trasera de la PCB.

Figura 5.69: Circuito impreso final tras el proceso de atacado.

5. Es recomendable realizar una revisión general de la PCB en busca de posibles imperfecciones que se hayan podido producir durante el proceso de impresión del circuito.

Normalmente, durante el proceso de atacado, se pueden producir imperfecciones en el trazado de las pistas debido a que la disolución de atacado puede producir una corrosión irregular de la capa de cobre. Este tipo de imperfecciones suelen ser cortos entre pistas, fracturas de pista, etc.

Para realizar el proceso de comprobación de imperfecciones es necesario utilizar un polímetro que pueda funcionar en modo detección de cortos para que, de esta forma, se pueda determinar cuándo la conductividad de las pistas es correcta. También es recomendable usar una lupa de aumento o similar para mejorar la visibilidad de las imperfecciones:



Figura 5.70: Integrante del equipo verificando cortos.

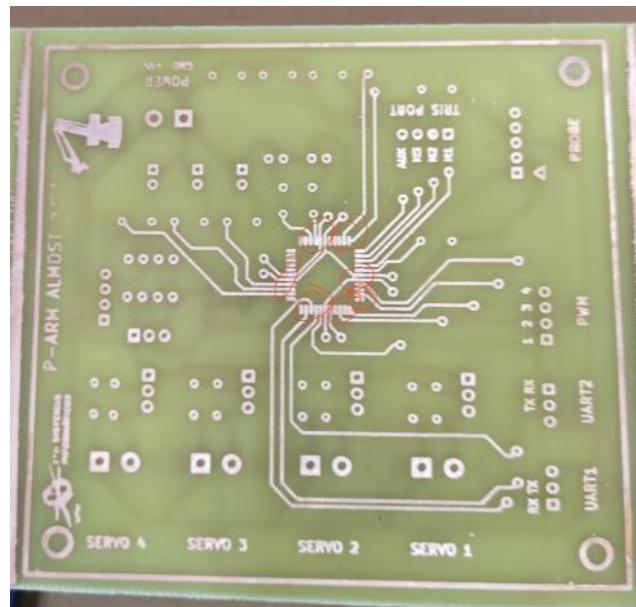


Figura 5.71: Algunos de los cortos detectados.

En caso de encontrar algún corto entre dos pistas que no deberían estar conectadas,

basta con utilizar un útil con punta para rasgar la superficie del punto de unión entre ambas.

En caso de encontrar alguna fractura en una de las pistas, basta con realizar una soldadura de empalme utilizando un hilo de grapinar o similar en el punto de fractura.

Ambas aproximaciones propuestas anteriormente son soluciones artesanales y no puede aplicarse cuando las imperfecciones son demasiado grandes o complejas de resolver, siendo mejor en este caso repetir el proceso de fotolitografía.

6. Se debe realizar taladrado de los orificios de la PCB y el guillotinado de los márgenes de la misma.

El taladrado de todos los orificios de la PCB, es decir, *pads* de componentes, vías y mecanizado de sujeción, se debe realizar con un taladro que cuente con una broca del tamaño adecuado, además de un soporte que aporte la estabilidad suficiente:



(a) Taladro y soporte usado.

(b) Taladrado de los distintos orificios para componentes.

Figura 5.72: Taladrado de la PCB.

El recorte de la PCB se realiza con una herramienta de guillotinado que tenga precisión y permita realizarlos de forma limpia de la longitud necesaria. Estos cortes se realizan siguiendo los límites establecidos en el diagrama físico de la PCB.

Tras el proceso de guillotinado y taladrado se obtiene el siguiente resultado:

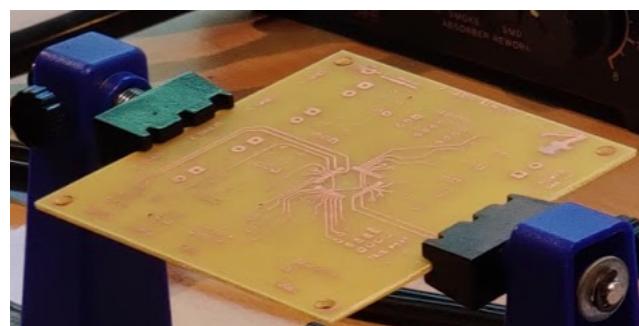


Figura 5.73: PCB con orificios taladrados y margenes guillotinados.

7. Soldar el componente principal de la PCB, es decir, el microcontrolador:

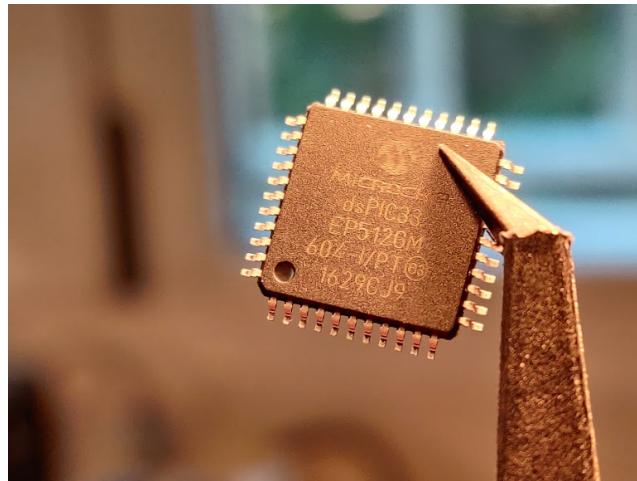


Figura 5.74: Plano detallado del microcontrolador.

Tal y como se puede ver en la imagen 5.74, el dsPIC utilizado tiene un encapsulado destinado a soldadura superficial (SMD). Normalmente, este tipo de componentes deben ser colocados y soldados los primeros, ya que el proceso de colocación y soldado de los mismos es delicado y requiere máxima precisión.

El soldado de este tipo de componentes se realiza mediante el aplicado de una pasta de soldadura, la cual se debe colocar en el punto de contacto de los pines del componente con las pistas de cobre:

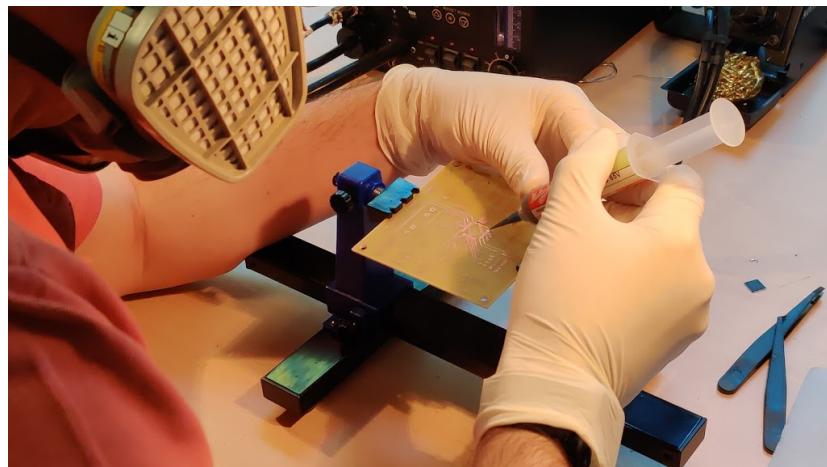


Figura 5.75: Aplicado de la pasta de soldadura.

El aplicado de la pasta de soldadura debe llevarse a cabo con la protección adecuada, ya que es altamente tóxica.

Tras aplicarla, se debe colocar el microcontrolador de forma precisa y cautelosa, cerciorándose de que cada uno de los pines realiza contacto correctamente con las pistas de la PCB:

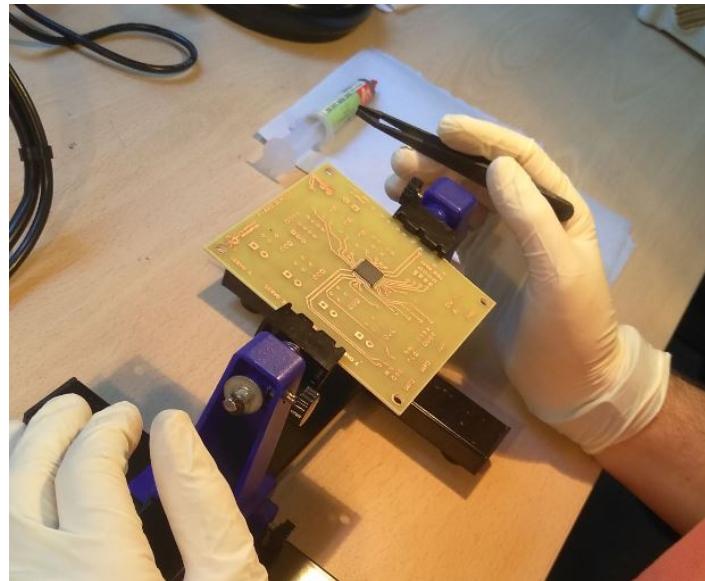
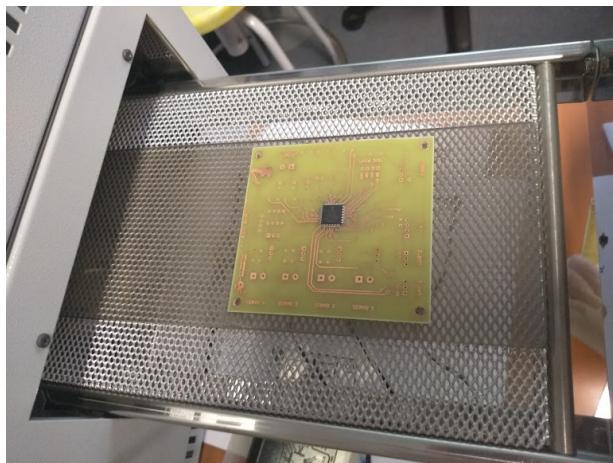
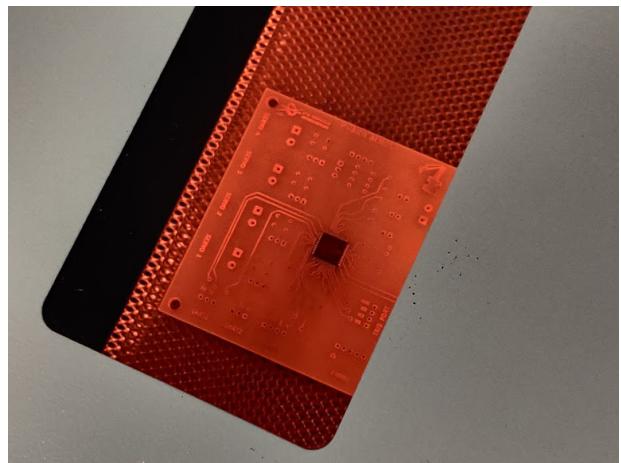


Figura 5.76: Colocado del microcontrolador sobre la pasta de soldadura.

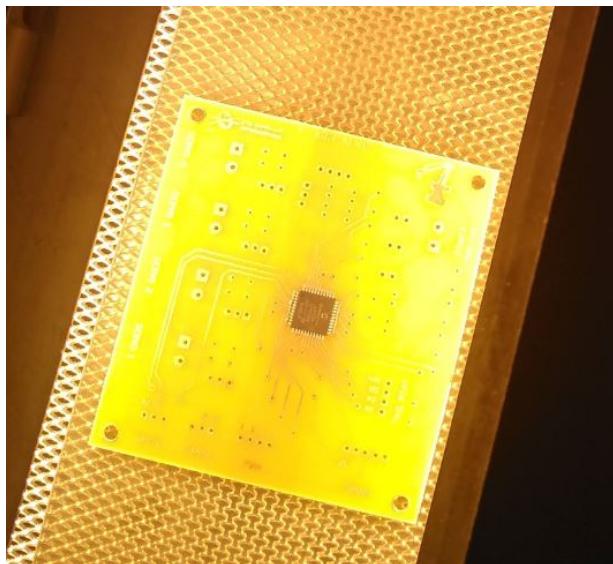
Tras colocar la pasta de soldadura, se debe introducir la PCB en un horno para que la pasta de soldadura se endurezca y selle por completo:



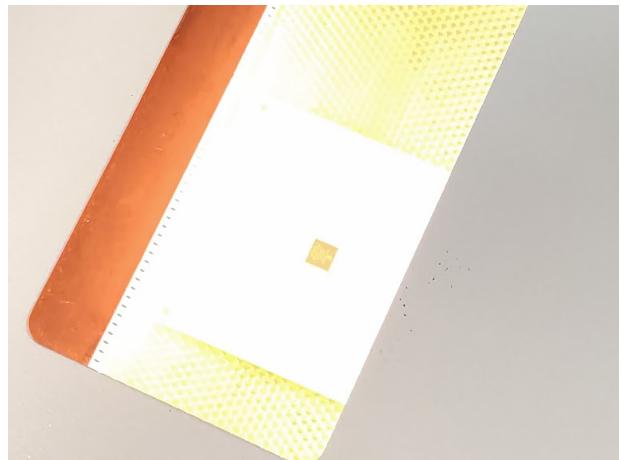
(a) Preparado de la placa en el horno.



(b) Precalentado del horno a 260 °C.



(c) Proceso de endurecimiento (1) - se deja reposar la pasta para que se solidifique.



(d) Proceso de endurecimiento (2) - se aplican golpes de calor para fijar la pasta.

Figura 5.77: Diferentes etapas del proceso de horneado.

Este proceso se debe realizar durante unos quince minutos a una temperatura de entre 250 °C y 260 °C.

Tras el proceso de horneado, se debe dejar enfriar la PCB. El resultado final tras el horneado es el siguiente:

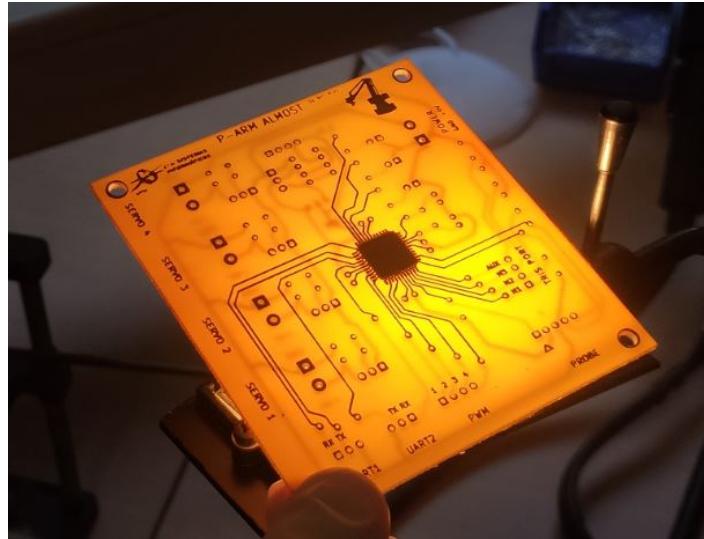


Figura 5.78: Imagen a contra luz de la PCB.

El microcontrolador se encuentra totalmente fijado y soldado superficialmente con las pistas de conexión gracias a la pasta de soldadura. En este momento, es recomendable verificar si existen cortos entre los pines del microcontrolador así como si la conductividad de los mismos es correcta a lo largo de las pistas:

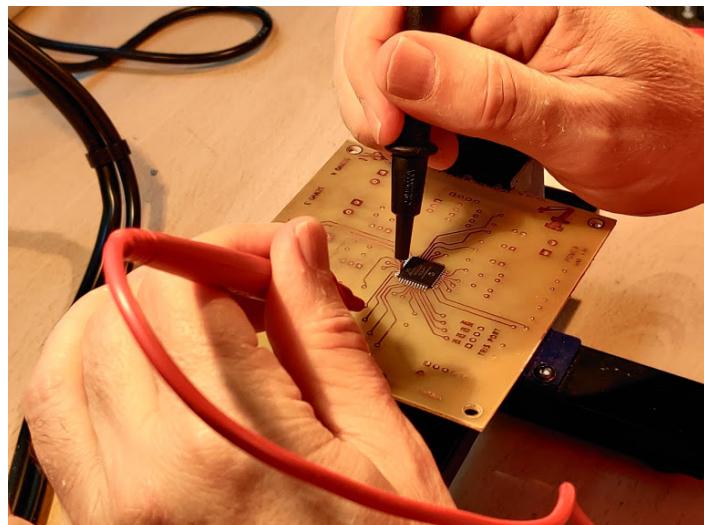


Figura 5.79: Comprobación de cortos y conductividad de las pistas del microcontrolador.

#### 8. Soldado de las vías de conexión entre las pistas de la capa superior e inferior.

Para realizar esta conexión, basta con introducir un hilo de grapinar por el orificio de la vía, realizar una pequeña soldadura con ambas pistas superior e inferior y, por último, cortar el hilo excedente:

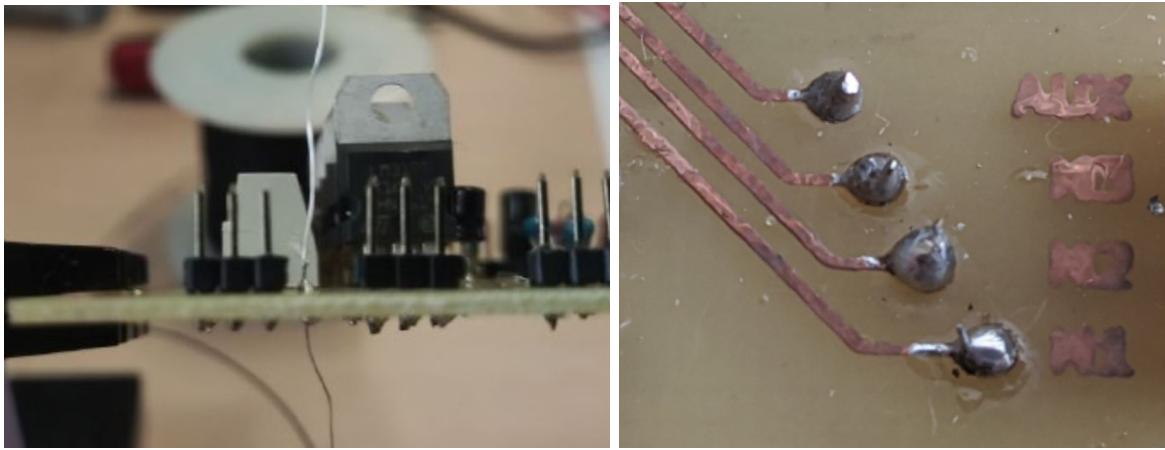


Figura 5.80: Conexiónado de las vías de la PCB.

9. En último lugar, se deben soldar el resto de componentes de la PCB.

Los componentes restantes tienen un encapsulado de agujero pasante (THT), a excepción del regulador A1117Z, el cual es de tipo SMD. Sin embargo, sus pines son de gran tamaño y no es necesario utilizar pasta de soldadura.

Debido a lo anterior, todos estos componentes se pueden soldar de forma convencional, utilizando un soldador y estaño:

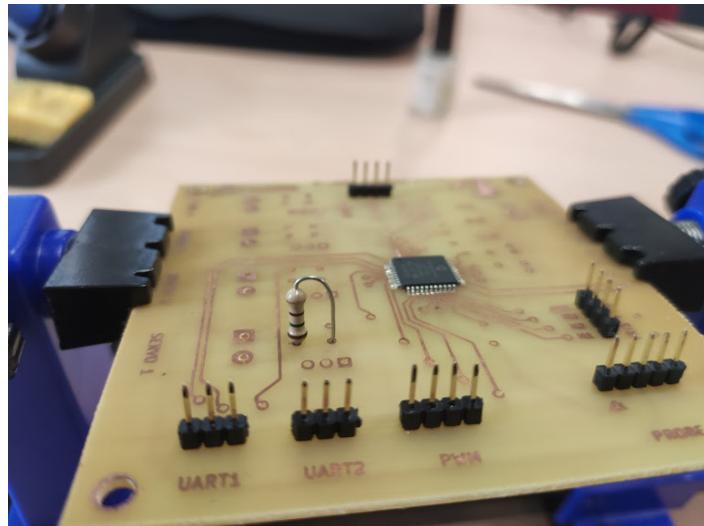


Figura 5.81: Comienzo del proceso de soldadura.

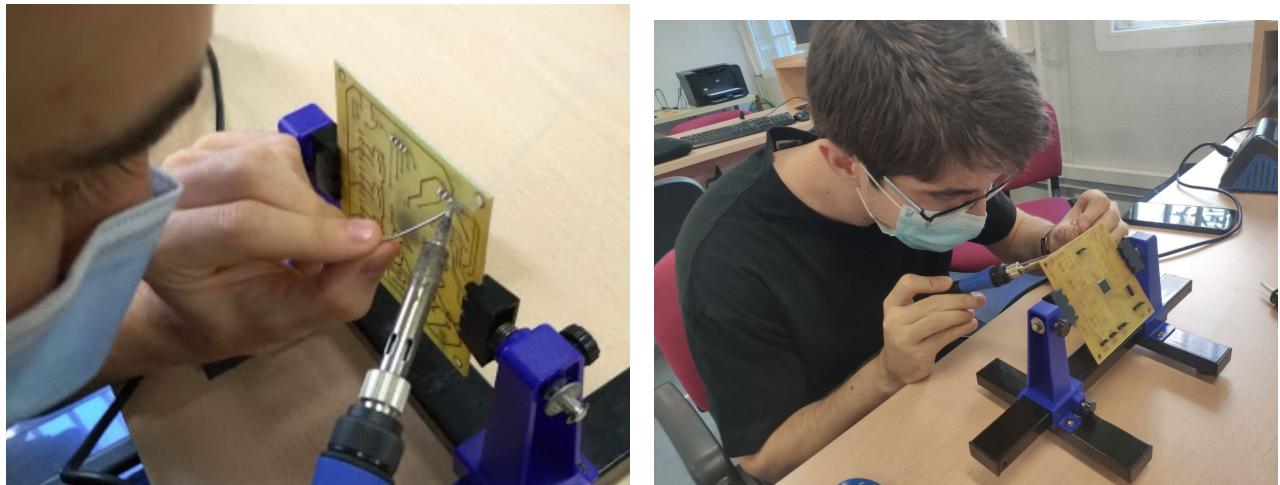


Figura 5.82: Integrantes del equipo soldando componentes.

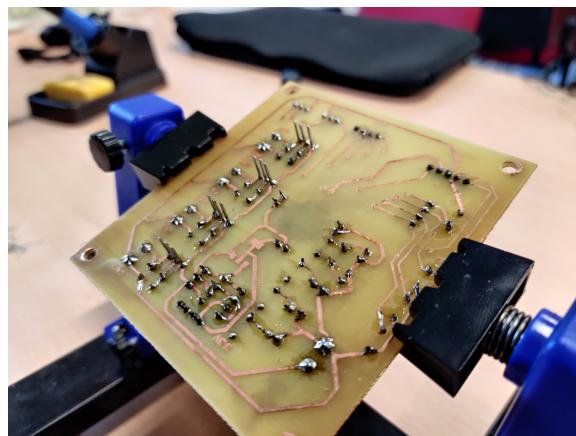
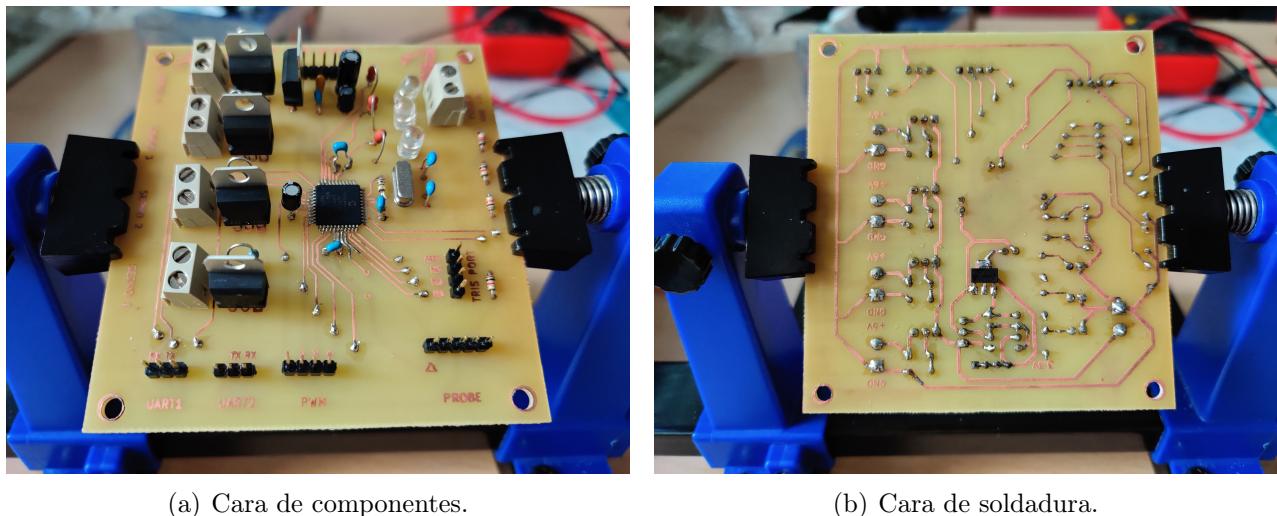


Figura 5.83: Proceso de soldadura en curso.

Tras finalizar el proceso de soldadura de los componentes y de haber recortado el exceso de sus pines, se obtiene el resultado final:



(a) Cara de componentes.

(b) Cara de soldadura.

Figura 5.84: Integrantes del equipo soldando componentes.

Cabe destacar que, tras el proceso de soldadura, se deben revisarlas en cuanto a los siguientes aspectos:

- Verificar si las soldaduras son limpias y brillantes.
- Verificar si las soldaduras conducen correctamente
- Eliminar excesos de estaño que puedan producir cortos.

En este punto se considera que el proceso de fabricación de la PCB ha terminado, a falta de realizar el proceso de verificación pendiente y que se muestra en el siguiente apartado.

#### 5.4.8. Verificaciones del prototipo construido

Tras completar la fabricación de la PCB, es recomendable realizar algunas comprobaciones mínimas para asegurarse de que el prototipo construido es funcional:

- Al completar la impresión del circuito y antes de soldar los componentes se debe verificar la correcta conductividad de las pistas de cobre, así como la búsqueda de cortocircuitos o fracturas. Para esta labor se debe utilizar un voltímetro.
- Posteriormente, pero antes de soldar los componentes, se debe verificar la correcta conducción de las distintas redes según lo descrito en el diagrama lógico. Para esta labor se debe utilizar un voltímetro.
- En el momento de soldar los componentes, se debe verificar su correcta inserción en los orificios de los *pads*, verificando también que su orientación y colocación de pines sea la correcta.

- Tras realizar todos las verificaciones anteriores y una vez se ha completado el proceso de soldadura de los componentes, se debe realizar el primer encendido del microcontrolador y el testeo de todos los componentes de la PCB.

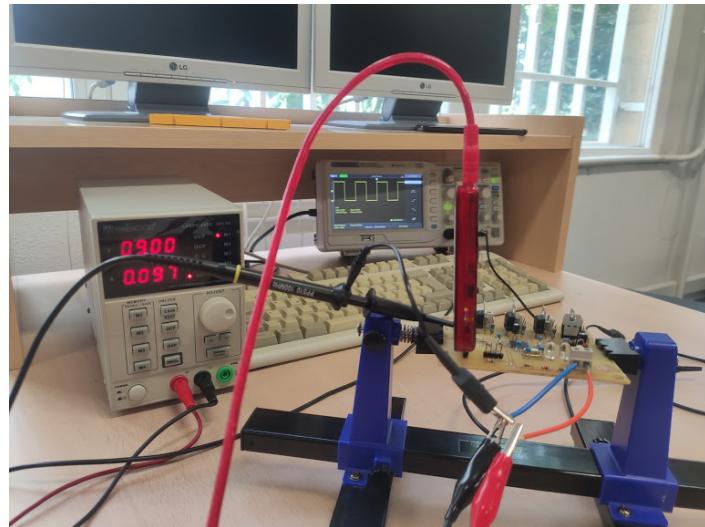


Figura 5.85: Primer encendido de la PCB usando un código de prueba del PWM.

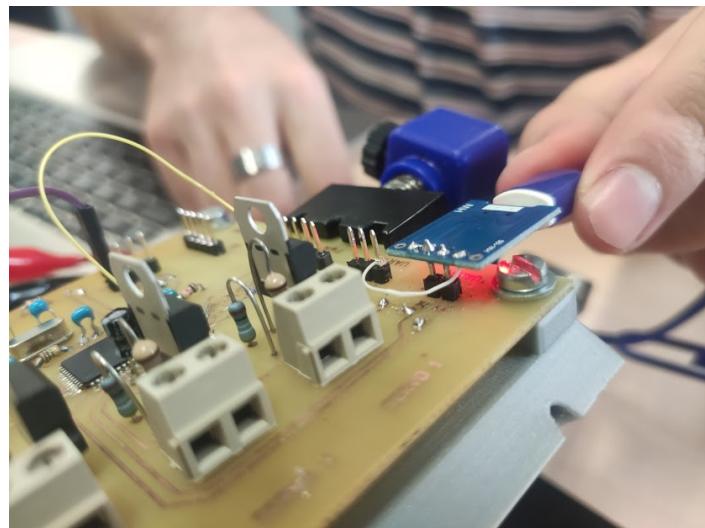


Figura 5.86: Prueba del funcionamiento de la UART.



Figura 5.87: Prueba del funcionamiento de los servomotores.

Todas las pruebas realizadas dieron buen resultado y gracias a ellas se descubrieron alguno de los contratiempos descritos en el apartado siguiente.

#### 5.4.9. Contratiempos ocurridos y soluciones ingenieras

El proceso de fabricación de la PCB ha sido uno de las etapas más críticas dentro del proyecto y se considera importante comentar algunos de los contratiempos que han ocurrido, puesto que estos han tenido un impacto considerable en el desarrollo del proyecto.

En general, estos contratiempos han ralentizado el proceso de fabricación. Sin embargo, mediante su detección y correcta solución, se ha obtenido una PCB mucho más robusta y segura, ya que se han revelado aspectos negativos que de otra forma habrían pasado desapercibidos.

A continuación se muestran los principales contratiempos ocurridos y las soluciones que se han llevado a cabo para enmendarlos:

- El primer contratiempo sucedido durante el proceso de fabricación está relacionado con el proceso de atacado de la PCB.

La efectividad de esta disolución de atacado sobre la PCB verse afectada en función de la temperatura a la que se realiza el proceso, saturación de la disolución y otros factores específicos.

Debido a lo expuesto anteriormente, se realizaron dos intentos fallidos de atacado de la PCB, en los cuales la disolución no consiguió eliminar la capa de cobre de los lugares que no estaban protegidas por resina. El resultado obtenido fue el siguiente:

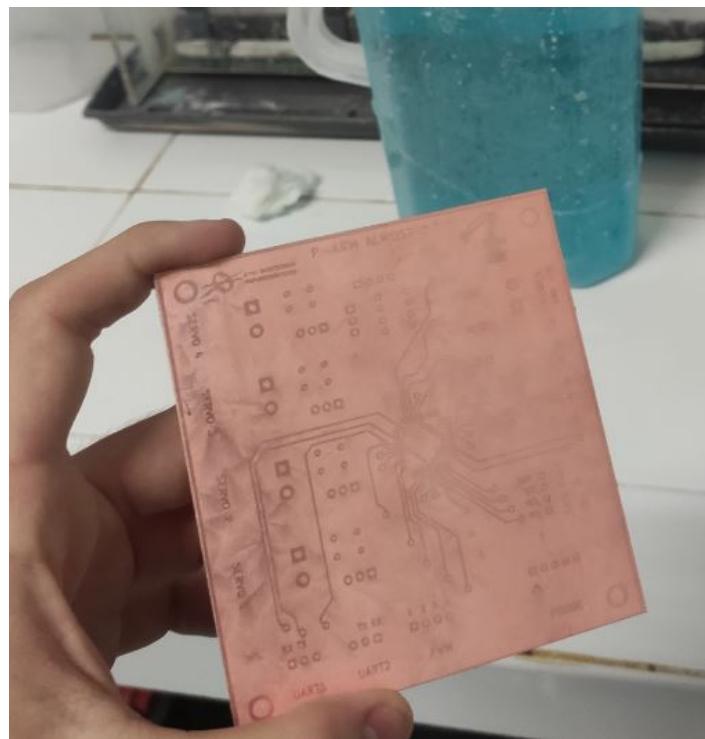


Figura 5.88: PCB tras proceso de atacado fallido.

Tras cada uno de los intentos fallidos, se tuvo que repetir el proceso de insolación de las fotomáscaras sobre una nueva placa positiva de prototipado.

Después de dos intentos fallidos, se decidió desechar la disolución de atacado, ya que esta llevaba bastante tiempo fabricada y podría estar en mal estado; posteriormente, se creó una nueva disolución de atacado:



Figura 5.89: Proceso de creación de la disolución de atacado.

Tras el atacado con la nueva disolución, el proceso tuvo éxito y el circuito se imprimió correctamente en la placa de prototipado.

A continuación se muestran las PCB obtenidas durante esta etapa:

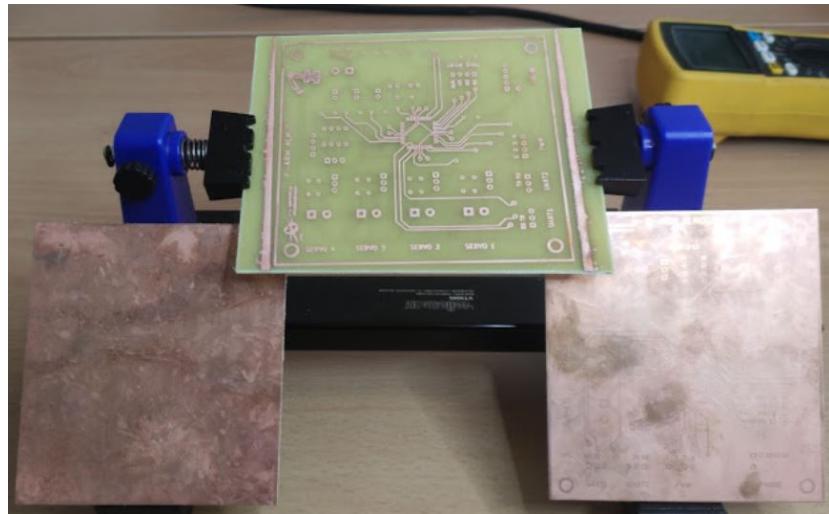


Figura 5.90: Intentos de fabricación de la PCB.

Tal y como se puede observar, en el centro de la imagen se encuentra la PCB exitosa y en los laterales los intentos fallidos, en los cuales el cobre no se eliminó adecuadamente.

- Otro de los contratiempos sucedidos fue detectado tras el proceso de soldado de los componentes, inmediatamente después de realizar el primer encendido del microcontrolador. Este contratiempo afecta al periférico PWM por partida doble, ya que se detectó una fractura en una de las pistas de comunicación del puerto PWM y, a su vez, se comprobó que el módulo PWM interno que usaba esta pista también estaba defectuoso y era inservible.

En la siguiente imagen (ver imagen 5.91) se puede apreciar la fractura en la pista de cobre:

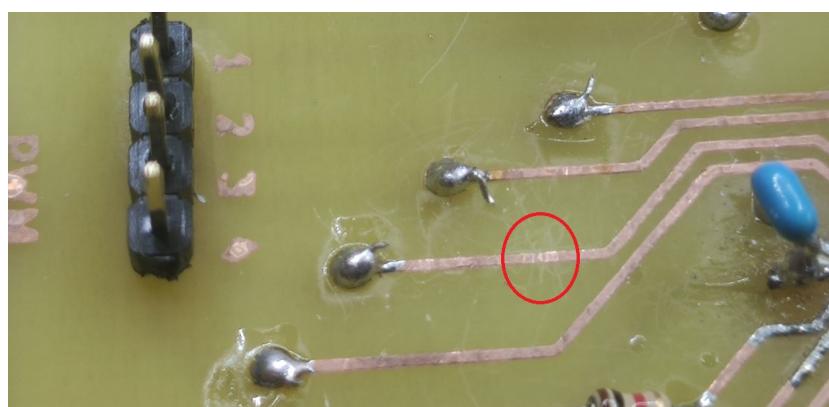


Figura 5.91: Fractura en pista de cobre.

En primer lugar, se procedió a realizar una nueva conexión de la pista usando hilo de grapinar:



Figura 5.92: Parcheo de la pista usando hilo de grapinar.

Tras verificar que la conductividad del nuevo hilo era correcta, se detectó que el módulo PWM seguía sin funcionar a pesar de que la fractura se había solucionado. Posteriormente, tras realizar algunas pruebas al respecto usando un osciloscopio, se concluyó que el módulo estaba defectuoso y no funcionaba. Esto además fue comprobado y corroborado por el tutor, por lo que se optó por utilizar un hilo de grapinar para reconectar el puerto PWM inservible a otro de los pines del microcontrolador, el cual usaba un módulo PWM que funcionaba correctamente y que no estaba siendo usado. Se obtuvo el siguiente resultado:

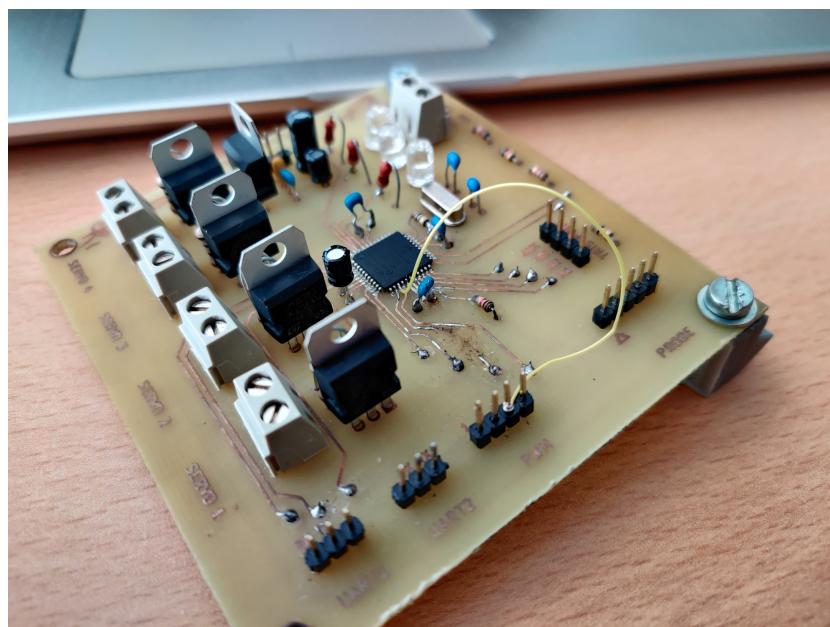


Figura 5.93: Reconexión del nuevo módulo PWM al puerto.

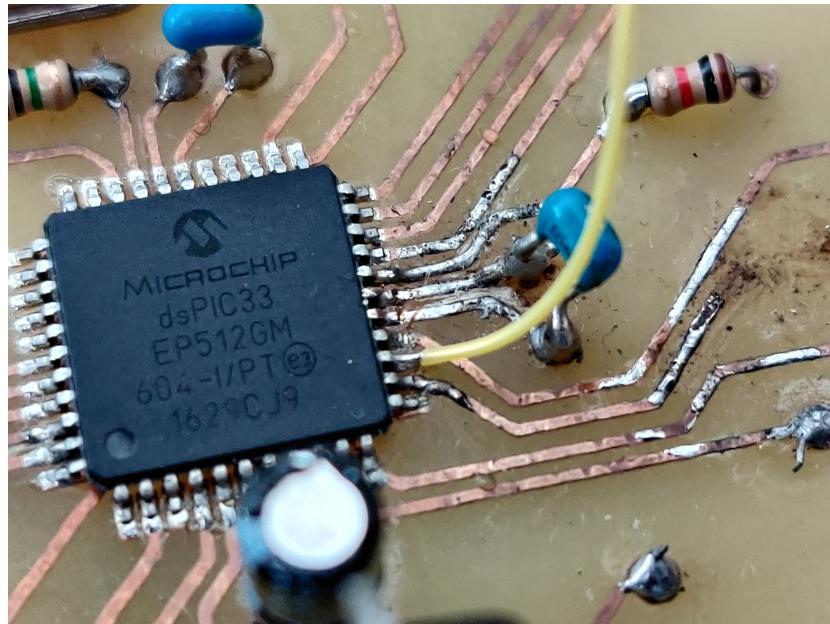


Figura 5.94: Soldadura del hilo de reconexión con el nuevo pin.

Tal y como se puede observar en las figuras anteriores (imágenes 5.93 y 5.94), el reducido tamaño de la soldadura requirió que este proceso fuese realizado con máxima precisión. Además, por precaución, se eliminó parte de la pista original para evitar errores en la señal.

A pesar de que esta solución es de carácter artesanal y lo recomendable hubiese sido fabricar de nuevo la PCB, se tomó la decisión de realizar este arreglo para reducir el impacto del contratiempo en el proyecto, ya que además se comprobó un buen resultado y fue suficientemente segura.

- El último de los contratiempos fue encontrado durante el proceso de soldado de la PCB y afecta al circuito de alimentación de los motores.

Tal y como se puede ver en otros apartados de este documento, los servomotores son alimentados mediante una clema, la cual obtiene su voltaje de los reguladores LM317 que adaptan el voltaje de alimentación de entrada.

Tras el soldado de la clema y los reguladores, se alimentó a la PCB para verificar que el voltaje de salida por las clemas fuese el requerido. Al realizar la medición del voltaje que había en los contactos de las clemas usando un voltímetro se detectó que este era muy superior al que debería ser.

El voltaje que deberían ofrecer las clemas es de  $5,41V$  y, sin embargo al realizar la medición, se obtuvieron  $7,1V$ , un valor intolerable ya que los servomotores no pueden soportarlo. Tras revisar la zona del circuito de alimentación, es decir, clemas y reguladores de tensión, se detectó que los reguladores de tensión LM317 no estaban funcionando correctamente.

Posteriormente, se procedió a realizar una revisión en profundidad de la zona, pistas cercanas, las soldaduras de las clemas y reguladores; sin embargo, no se encontró ningún fallo de fabricación o ensamblaje.

Dado que el fallo no parecía deberse al proceso de fabricación, se procedió a revisar de nuevo los diseños lógico y físico de la PCB. Tras esta revisión, se concluyó que el conexionado del diagrama lógico era correcto pero la huella física elegida para el regulador de tensión LM317 en el diagrama físico presentaba un orden en los pines diferente al del utilizado en el prototipo. Se muestran las imágenes en las que se ve claramente esta diferencia:

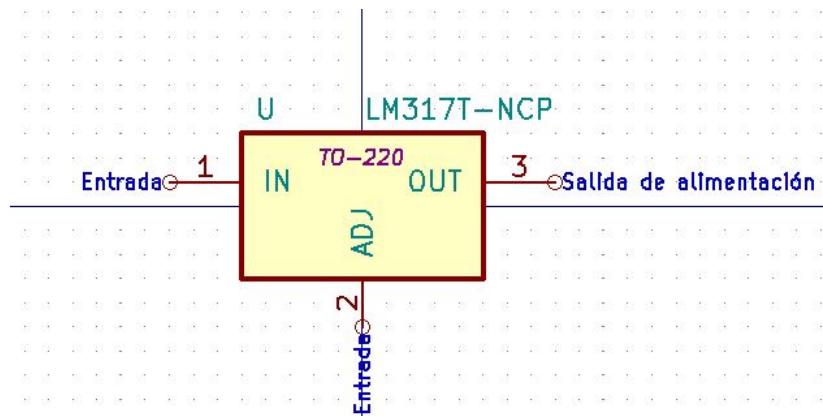


Figura 5.95: Esquemático lógico del LM317.

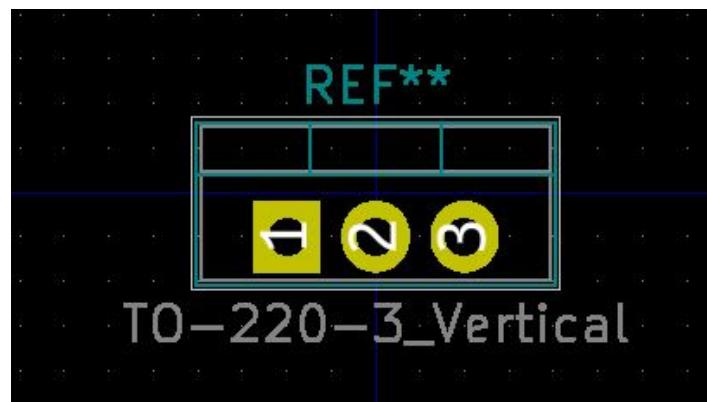


Figura 5.96: Huella física del LM317.

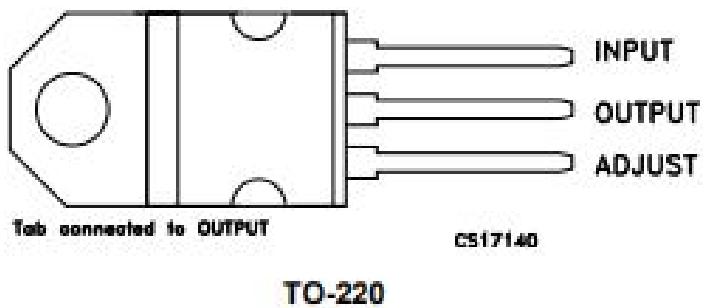


Figura 5.97: Encapsulado físico del LM317.

Tal y como se puede observar en las imágenes anteriores (imágenes 5.95, 5.96 y 5.97):

- El esquemático lógico del LM317 (ver imagen 5.95) consta de 3 pines, al igual que su encapsulado y huella física.
- El encapsulado del LM317 (ver imagen 5.97) tiene como referencia TO-220, con lo cual se debe seleccionar dicha huella física (ver imagen 5.96).
- A pesar de que la referencia de la huella física es TO-220, se puede afirmar que la distribución de pines de la misma no es igual al del encapsulado físico del componente.
- Los *pads* 2 y 3 de la huella física, correspondientes a los pines **ADJUST** y **OUTPUT** del encapsulado del LM317, están intercambiados y, en consecuencia, la colocación de los pines es errónea.

Debido a lo descrito anteriormente, se comprueba que el diagrama físico de la PCB contenía un error en el conexionado de los reguladores LM317 que alimentan los servomotores y, por ello, el voltaje de salida de las clemas era de 7,1V y no de 5,41V, tal y como se había diseñado.

Como primer paso para solucionar dicho problema, se procedió a editar la huella física del componente y se intercambiaron los pines:

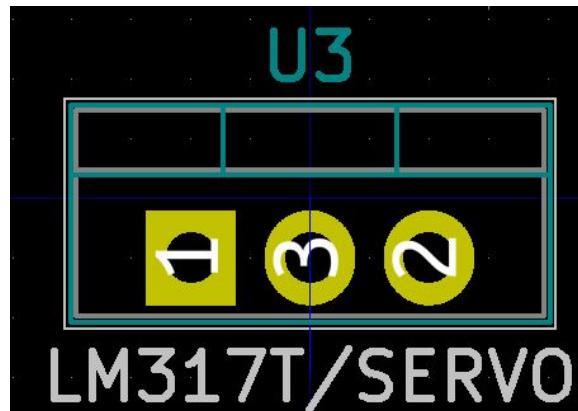


Figura 5.98: Huella física del LM317 con pines reordenados.

Puesto que se habían intercambiado dos pines de la huella física, las pistas de conexión de los mismos debían ser reconectadas para arreglar el problema:

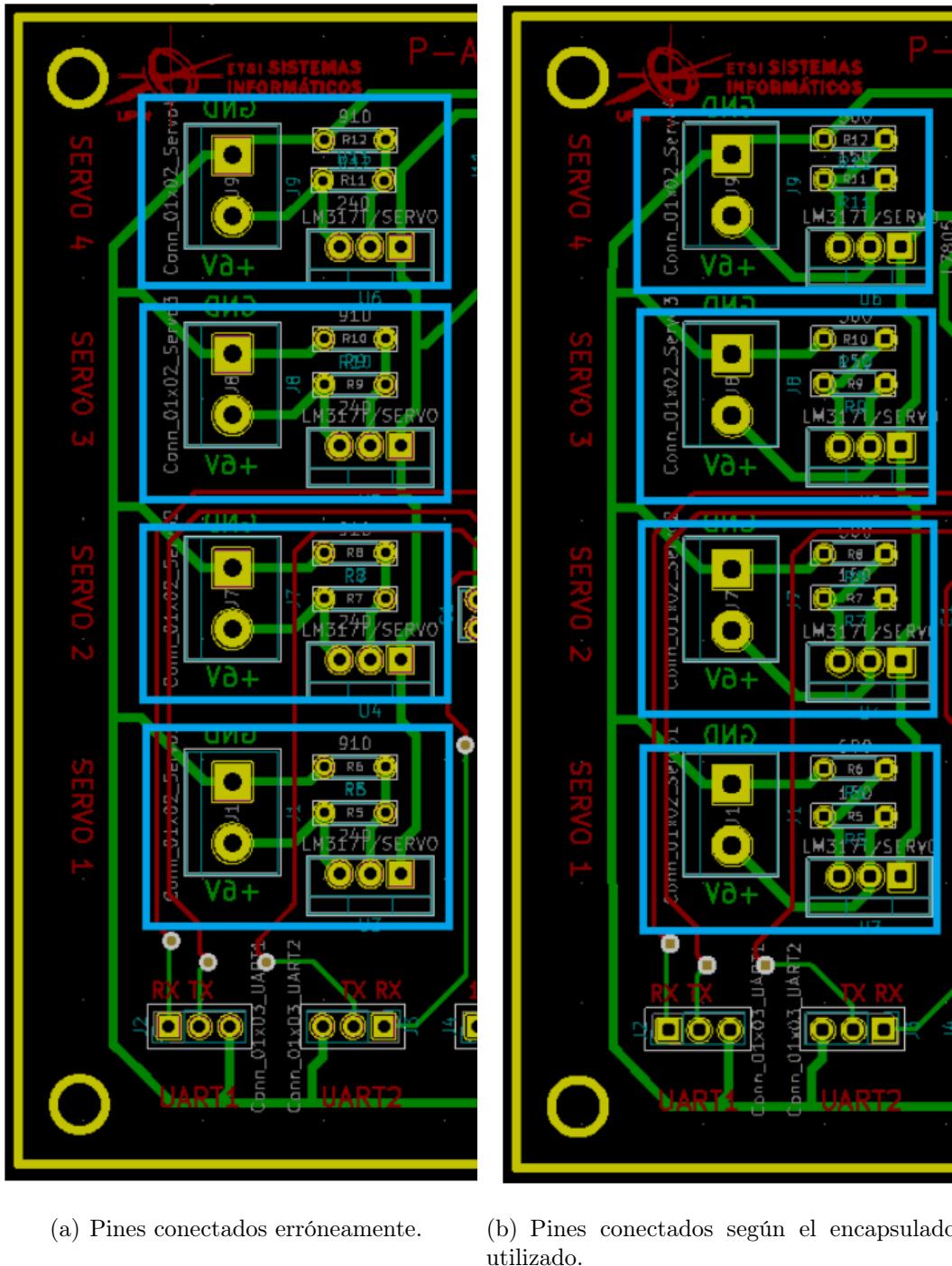


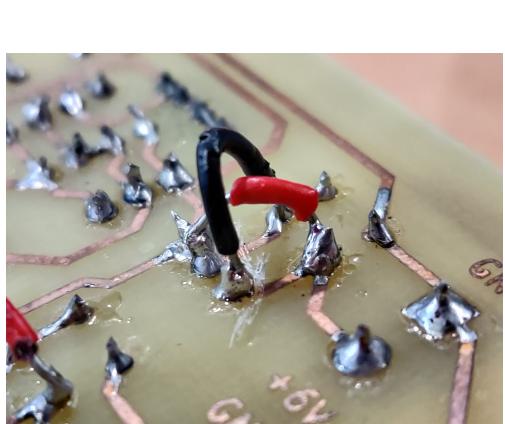
Figura 5.99: Modificación del circuito de alimentación de los servomotores, marcada en azul.

Esta modificación del diagrama físico de la PCB hecha a posteriori ya se contempla en el apartado 5.4.5 de la memoria, dado que en dicho apartado se muestra la versión final del diagrama físico.

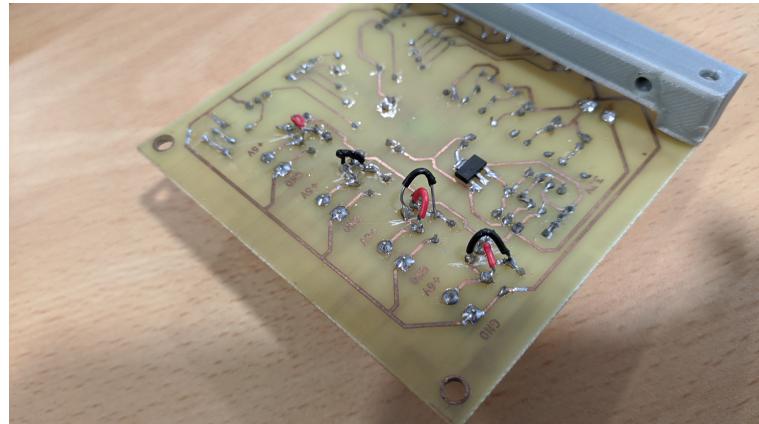
Como segundo paso para solucionar el problema y puesto que la PCB ya había sido fabricada siguiendo el diagrama físico que contenía el fallo se debía de ingeniar alguna solución para el prototipo construido. A pesar de que se planteó la posibilidad de fa-

bricar de nuevo la PCB, se consideró que este proceso causaría demasiado retraso en el desarrollo del proyecto y por lo tanto se optó por ingeniar otra solución directamente sobre el prototipo que estaba fabricado y ensamblado.

Puesto que el fallo consistía en dos pines que deberían estar intercambiados, bastaría con intercambiar las pistas de comunicación que conectaban con ellos. Se decidió, de esta forma, fracturar las pistas existentes para después colocar dos puentes, intercambiando de esta forma el conexionado de los pines con sus pistas (ver imágenes 5.100).



(a) Vista detallada del puente.



(b) Vista de los cuatro puentes realizados

Figura 5.100: Puentes entre pistas de los reguladores de tensión LM317.

Para la realización se emplearon cables de tamaño reducido que actuaban como puente entre las pistas. Por su tamaño y dado que el soldado de los mismos se realizó a mano, se considera que el resultado podría ser mejorable; sin embargo, esta solución dió buenos resultados y corrigió correctamente el voltaje de salida de los reguladores de tensión LM317.

A pesar de todos los errores cometidos durante el proceso de fabricación y diseño de la PCB, los integrantes del equipo han sido capaces de:

- Ingeniar soluciones apropiadas que generasen el mínimo impacto en el desarrollo del proyecto.
- Implementar dichas soluciones en el prototipo ya fabricado.
- Verificar que la corrección es apropiada y que daba el resultado buscado y esperado en un principio.

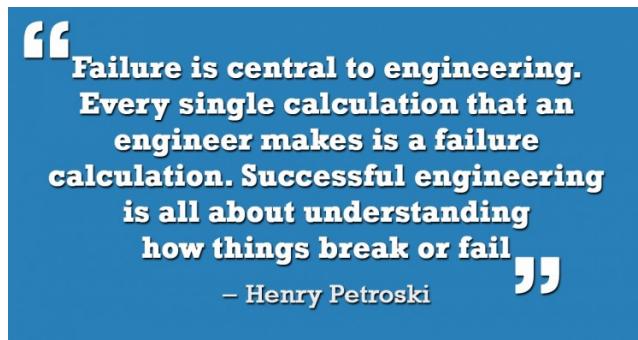


Figura 5.101: Reflexión [31].

## 5.5. Comunicaciones

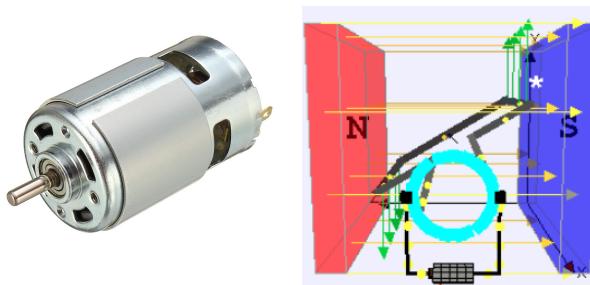
## 5.6. [REVISADO] Motores empleados (actuadores)

Dado que en este proyecto se ha planteado la construcción de un manipulador o brazo robótico, los únicos actuadores empleados han sido motores, los cuales dotan de movilidad a la estructura física del brazo.

Cabe destacar que, debido a la forma de la estructura física del brazo y dado a que el mismo tiene tres articulaciones móviles, se han empleado tres motores principales en cada una de ellas y un motor auxiliar en el extremo del manipulador.

Existen varios tipos de motores eléctricos que pueden ser usados para dotar de movilidad a proyectos de robótica de pequeña escala. Sin embargo los principales tipos se pueden agrupar en las siguientes tres categorías:

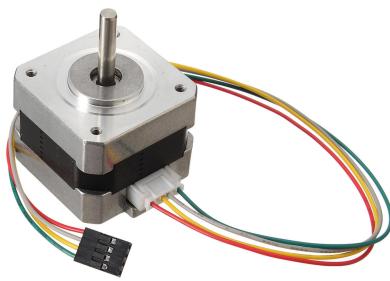
- Motores de corriente continua: son los motores eléctricos más sencillos y básicos. Debido a esto, realizar el control de la posición angular del eje y su velocidad de rotación es complicado y requiere aplicar técnicas de control de lazo cerrado. Además, el control físico de este tipo de motores se lleva a cabo mediante una señal PWM actuando sobre un puente H.



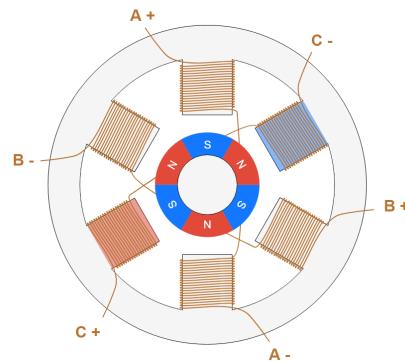
(a) Motor DC real [32]      (b) Funcionamiento [33]

Figura 5.102: Motor de corriente continua

- Motores paso a paso: se trata de motores eléctricos más complejos que ofrecen una precisión muy alta en cuanto a control de posición y velocidad, ya que descomponen su movimiento en pasos de longitud constante. En este tipo de motores se puede realizar control de velocidad y posición del eje del motor mediante técnicas de control de lazo abierto, dado que en este tipo de motores se controla el número de pasos que da el motor, así como cada cuánto tiempo se produce un paso. Este tipo de motores suelen necesitar un *driver* específico para ser controlados.



(a) Motor real [34]



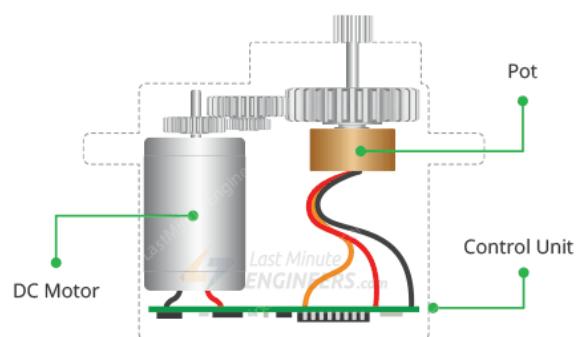
(b) Funcionamiento [35]

Figura 5.103: Motor paso a paso

- Servomotores: se trata de motores de corriente continua que incorporan un sistema de control de posición de lazo cerrado. Por ello, este tipo de motores ofrecen un control muy simple de la posición angular del eje del motor. A través de una señal PWM enviada al motor, se puede establecer una posición consigna que el eje del motor debe cumplir. Estos motores incluyen un sensor de posición (encoder, potenciómetro solidario al eje o similar) que determina la posición angular del eje del motor y una unidad de control que verifica la posición actual del eje en comparación con la posición de consigna establecida, realizando las correcciones necesarias hasta alcanzar dicha posición angular.



(a) Servomotor real [36]



(b) Funcionamiento [37]

Figura 5.104: Servomotor de corriente continua

Tras analizar los diferentes tipos de motores anteriormente expuestos, se ha decidido utilizar servomotores para dotar de movilidad al brazo robótico. Esta decisión se fundamenta en los siguientes motivos:

- A diferencia de los motores paso a paso o motores de corriente continua, no se suele necesitar ningún tipo de circuito externo, driver o puente H para controlar un servomotor; únicamente se debe alimentar el motor y proporcionar una señal de control.
- Este tipo de motores ofrece un control de posición preciso y simple mediante una señal PWM. A pesar de que dicho control de posición se realiza mediante lazo cerrado internamente dentro del motor, desde un punto de vista externo, no se necesita ningún tipo de realimentación externa.

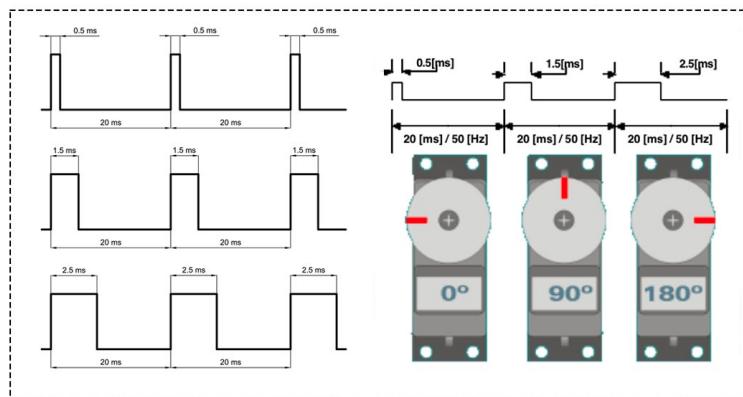


Figura 5.105: Ejemplo genérico de control de posición mediante señal PWM [38].

- Se trata de motores que se adaptan muy bien para proyectos de robótica de pequeña escala, debido a su bajo coste y sencillez de uso.
- Este tipo de motores está muy extendido en el mercado y existen numerosos modelos con diferentes potencias, tamaños, etc.

Es importante destacar que existen dos tipos de servomotores:

- Servomotores de giro limitado: son aquellos servomotores que tienen un rango de rotación limitado, el cual suele ser normalmente de  $180^\circ$ . Son el tipo de servomotor más sencillo.
- Servomotores de giro continuo: son aquellos servomotores que tienen rango completo de giro, es decir, pueden realizar giros sin limitación de recorrido.

Dado que ninguna de las articulaciones del motores está diseñada para realizar giros de más de  $180^\circ$ , se han empleado servomotores de giro limitado.

Otro de los datos que es importante clarificar antes de tomar la decisión de que motores van a ser usados en un proyecto de robótica, es la carga máxima que va a tener que desplazar el manipulador robótico. Este dato afecta principalmente al diseño de la estructura física del brazo y a la potencia de los motores escogidos, en especial, el torque que ejercen.

Finalmente, el modelo de servomotor elegido para las articulaciones ha sido el *Parallax 900-00005 Standard Servo* el cual tiene las siguientes características técnicas:

- Servomotor de rango limitado de  $180^\circ$ .
- Control mediante señal PWM de 50Hz.
- Alimentación de entre  $4V$  y  $6V$ , utilizando entre  $15mA$  y  $200mA$ . Potencia nominal de  $140mA$ .
- Torque máximo ejercido de  $27N \cdot cm$ , es decir aproximadamente  $2,75Kgf \cdot cm$ .
- Conociendo el torque ejercido por los servomotores y área de trabajo del manipulador, mostrada en la figura ??, se pueden realizar algunos cálculos para deducir cuál será la carga máxima que podrá soportar el brazo robótico:
  - En la zona de trabajo en la cual el brazo robótico está menos extendido, y por lo tanto situación en la que el esfuerzo es mínimo sobre la estructura del manipulador, los motores aplican su fuerza a 8.1 cm del extremo del robot. Dado que el torque es generado es de aproximadamente  $2,75Kgf \cdot cm$ , se podría levantar una masa de aproximadamente 300g.
  - En la zona de trabajo en la cual el brazo robótico está más extendido, y por lo tanto situación en la que el esfuerzo es máximo sobre la estructura del robot, los motores aplican su fuerza a 34.6 cm del extremo del robot. Dado que el torque es generado es de aproximadamente  $2,75Kgf \cdot cm$ , se podría levantar una masa de aproximadamente 80g.
  - Teniendo en cuenta los cálculos anteriores, se recomienda que la carga máxima del manipulador sea de entre 150g y 60g, siempre teniendo en cuenta las zonas de trabajo en las que se vaya a desplazar la carga para tener garantías de que el desplazamiento es seguro.
- Peso de 44g.
- Dimensiones 406 x 55,8 x 19 mm



Figura 5.106: Servomotor Parallax utilizado [39]

Teniendo en cuenta los datos técnicos anteriores, este modelo de servomotor se adapta perfectamente a las características del brazo robótico que se ha desarrollado, cumpliendo todas la cualidades deseadas para que el funcionamiento del brazo robótico sea correcto.

# Capítulo 6

## *Software*

### 6.1. S1

#### 6.1.1. UI/UX

#### 6.1.2. Protocolo de comunicación

#### 6.1.3. Pseudolenguaje de comunicación

#### 6.1.4. Logs

#### 6.1.5. Otros

### 6.2. S2

#### 6.2.1. Protocolo de comunicación

#### 6.2.2. Interpretación del pseudo-lenguaje

#### 6.2.3. Cálculo de movimientos/trayectorias

#### 6.2.4. Control de los componentes

#### 6.2.5. Otros

# Capítulo 7

## Casos de estudio

7.1. Decisiones tomadas

7.2. Desarrollo de las distintas partes del proyecto

# Capítulo 8

## Calidad y pruebas

8.1. Batería de pruebas

8.2. Explicación de las pruebas

8.3. Resultados esperados | resultados obtenidos

8.4. Reflexión - solución

# Capítulo 9

## Demostración

## Capítulo 10

# Planificación, costes y tiempo empleado

10.1. Diagramas de Gantt

10.2. Sueldos propuestos y costes obtenidos

### 10.3. Coste de los materiales inicial - coste de los materiales final

Elemento	Descripción	Cantidad	Precio	Código RS
Servomotor Parallax Inc.	Los servomotores permitirán controlar los distintos movimientos que puede realizar el robot. Será necesario disponer de varios para poder controlar los tres grados de libertad de los que dispondrá el <i>pArm</i> : uno para la base, otro para el primer segmento y un último para el segundo segmento.	4	16,01 €	781-3058
Rodamiento de bolas NMB, Radial.	La unión entre ejes del brazo robótico se realiza mediante rodamientos, permitiendo así un movimiento fluido y duradero del brazo.	16	5,035 €	612-6013
Pasador cilíndrico de 20 mm. de largo y 4 mm. de diámetro	El pasador que permitirá unir dos ejes separados, pasando por medio de los rodamientos anteriores.	1 bolsa	7,05 €	270-439
Microinterruptor de hasta 125V@5A	Fin de carrera para controlar la posición del brazo robótico.	4	1,709 €	682-0866
Resistencia de $240\Omega$	Resistencia necesaria para la construcción de la placa de control del brazo robótico.	10	0,116 €	148-354
Resistencia de $5,1K\Omega$	Resistencia necesaria para la construcción de la placa de control del brazo robótico.	100	0,033 €	199-7887
<b>Total:</b>			<b>162,949 €</b>	

Tabla 10.1: Tabla completa de presupuestos.

### 10.4. Evolución del tiempo empleado

## 10.5. Contratiempos y tiempo de desarrollo final

# Capítulo 11

## Conclusiones

11.1. Conclusiones técnicas

11.2. Experiencia personal en el desarrollo del proyecto

11.3. Conocimientos adquiridos y nuevas competencias

# Capítulo 12

## Futuras mejoras

- 12.1. Desarrollos e implementaciones que no han podido realizarse
- 12.2. Ideas propuestas pero no implementadas
- 12.3. Otras

# Bibliografía

- [1] M. E. Moran, «Evolution of robotic arms,» *Journal of Robotic Surgery*, vol. 1, n.º 2, págs. 103-111, jul. de 2007, ISSN: 1863-2491. DOI: [10.1007/s11701-006-0002-x](https://doi.org/10.1007/s11701-006-0002-x).
- [2] J. de Vaucanson, *Le mécanisme du flûteur automate: présenté à Messieurs de l'Académie royale des Sciences : avec la description d'un canard artificiel et aussi celle d'une autre figure également merveilleuse, jouant du tambourin et de la flûte.* chez Jacques Guerin, imprimeur-libraire, 1738, Google-Books-ID: UNw6AAAAcAAJ.
- [3] Chapuis, Alfred and Droz, Edmond, *Automata: A Historical and Technological Study.* L'Editions du Griffon, 1958.
- [4] Standage, Tom, *The Turk: The Life and Times of the Famous Eighteenth-Century Chess-Playing Machine.* Walker company, 2002.
- [5] Belarmino, J and Moran, ME and Firoozi, F and Capello, S and Kolios, E and Perrotti, M, «Tesla's Robot and the Dawn of the Current Era. Society of Urology and Engineering,» 7.<sup>a</sup> ép., vol. 19, A214, J Endourol 2005.
- [6] ——, «An Oriental Culture of Robotics—the Coming Maelstrom. Society of Urology and Engineering,» 7.<sup>a</sup> ép., vol. 19, A119, J Endourol 2005.
- [7] *Mobile Servicing System.* feb. de 2020, Page Version ID: 942154747  
Publication Title: Wikipedia. dirección: [https://en.wikipedia.org/w/index.php?title=Mobile\\_Servicing\\_System&oldid=942154747](https://en.wikipedia.org/w/index.php?title=Mobile_Servicing_System&oldid=942154747) (visitado 14-06-2020).
- [8] *Spirit (Rover).* jun. de 2020, Page Version ID: 960186651  
Publication Title: Wikipedia. dirección: [https://en.wikipedia.org/w/index.php?title=Spirit\\_\(rover\)&oldid=960186651](https://en.wikipedia.org/w/index.php?title=Spirit_(rover)&oldid=960186651) (visitado 15-06-2020).
- [9] *Rocker-bogie*, en Wikipedia, la enciclopedia libre, 27 de mar. de 2020. dirección: <https://es.wikipedia.org/w/index.php?title=Rocker-bogie&oldid=124613397> (visitado 24-08-2020).
- [10] *Opportunity (Rover)*. mayo de 2020, Page Version ID: 960035335  
Publication Title: Wikipedia. dirección: [https://en.wikipedia.org/w/index.php?title=Opportunity\\_\(rover\)&oldid=960035335](https://en.wikipedia.org/w/index.php?title=Opportunity_(rover)&oldid=960035335) (visitado 15-06-2020).
- [11] *The Mars Rovers: Spirit and Opportunity* |textbackslashtextbar NASA Space Place – NASA Science for Kids. dirección: <https://spaceplace.nasa.gov/mars-spirit-opportunity/en/> (visitado 15-06-2020).
- [12] *History of Robots.* jun. de 2020, Page Version ID: 961981460  
Publication Title: Wikipedia. dirección: [https://en.wikipedia.org/w/index.php?title=History\\_of\\_robots&oldid=961981460](https://en.wikipedia.org/w/index.php?title=History_of_robots&oldid=961981460) (visitado 15-06-2020).

- [13] R. Baldwin, *Tesla's Video Shows What Its Autopilot System Sees*. feb. de 2020, Library Catalog: www.caranddriver.com Section: News  
Publication Title: Car and Driver. dirección: <https://www.caranddriver.com/news/a30733506/tesla-autopilot-full-self-driving-video/> (visitado 15-06-2020).
- [14] *Tesla's Full Self-Driving Computer Is Now in All New Cars and a next-Gen Chip Is Already 'Halfway Done'*, Library Catalog: techcrunch.com  
Publication Title: TechCrunch. dirección: <https://social.techcrunch.com/2019/04/22/teslas-computer-is-now-in-all-new-cars-and-a-next-gen-chip-is-already-halfway-done/> (visitado 15-06-2020).
- [15] *Boston Dynamics*. mar. de 2020, Page Version ID: 123964032  
Publication Title: Wikipedia, la enciclopedia libre. dirección: [https://es.wikipedia.org/w/index.php?title=Boston\\_Dynamics&oldid=123964032](https://es.wikipedia.org/w/index.php?title=Boston_Dynamics&oldid=123964032) (visitado 15-06-2020).
- [16] *KR 1000 titan*, Library Catalog: www.kuka.com  
Publication Title: KUKA AG. dirección: <https://www.kuka.com/es-es/productos-servicios/sistemas-de-robot/robot-industrial/kr-1000-titan> (visitado 15-06-2020).
- [17] *UFACTORY Official Website*, Library Catalog: store.ufactory.cc  
Publication Title: store.ufactory.cc. dirección: <https://store.ufactory.cc/> (visitado 15-06-2020).
- [18] *UPM-Robotics/Uarm*, Robotics UPM, 19 de nov. de 2019. dirección: <https://github.com/UPM-Robotics/uarm> (visitado 28-06-2020).
- [19] *UFACTORY xArm|textbackslashtextbaruArm*. dirección: <https://www.ufactory.cc/#/en/uarmswift> (visitado 28-01-2020).
- [20] (). «Arduino Mega 2560 Rev3 | Arduino Official Store,» dirección: <https://store.arduino.cc/arduino-mega-2560-rev3> (visitado 28-06-2020).
- [21] S. Sharma y C. Scheurer, «Generalized Unified Closed Form Inverse Kinematics for Mobile Manipulators With Reusable Redundancy Parameters,» 6 de ago. de 2017. DOI: 10.1115/DETC2017-68104.
- [22] (). «Roboy 2.0 - Inverse Kinematics,» dirección: <https://ik-test.readthedocs.io/en/latest/> (visitado 28-06-2020).
- [23] *Ácido poliláctico*. mayo de 2020, Page Version ID: 126434778. dirección: [https://es.wikipedia.org/w/index.php?title=%C3%81cido\\_polil%C3%A1ctico&oldid=126434778](https://es.wikipedia.org/w/index.php?title=%C3%81cido_polil%C3%A1ctico&oldid=126434778) (visitado 16-06-2020).
- [24] *Acrilonitrilo butadieno estireno*. jun. de 2020, Page Version ID: 126716603. dirección: [https://es.wikipedia.org/w/index.php?title=Acrilonitrilo\\_butadieno\\_estireno&oldid=126716603](https://es.wikipedia.org/w/index.php?title=Acrilonitrilo_butadieno_estireno&oldid=126716603) (visitado 16-06-2020).
- [25] (). «The Pantograph in Context,» dirección: <https://www.circuitousroot.com/artifice/letters/press/typemaking/making-matrices/pantograph-in-context/index.html#:~:text=A%20pantograph%20is%20a%20system,an%20increased%20or%20decreased%20size.&text=The%20so%2Dcalled%20%22parallel%20motion,transmission%20of%20power%20and%20motion> (visitado 23-06-2020).
- [26] *Regulador LM317 Datasheet*. dirección: <https://www.st.com/resource/en/datasheet/lm217.pdf> (visitado 04-09-2020).

- [27] *Regulador L7805CV Datasheet.* dirección: <https://www.st.com/resource/en/datasheet/178.pdf> (visitado 04-09-2020).
- [28] *Regulador AZ1117H Datasheet.* dirección: [https://www.diodes.com/assets/Datasheets/products\\_inactive\\_data/AZ1117.pdf](https://www.diodes.com/assets/Datasheets/products_inactive_data/AZ1117.pdf) (visitado 04-09-2020).
- [29] *dsPIC33EP512GM604 - 16-Bit - Microcontrollers and Digital Signal Controllers.* dirección: <https://www.microchip.com/wwwproducts/en/dsPIC33EP512GM604> (visitado 16-06-2020).
- [30] *Construcción de una PCB.* dirección: [http://www.ugr.es/~amroldan/cursos/pcb\\_uhu\\_98/construccionpcb.html](http://www.ugr.es/~amroldan/cursos/pcb_uhu_98/construccionpcb.html) (visitado 04-09-2020).
- [31] *10 of the Best Engineering Quotes Ever,* en-US, feb. de 2018. dirección: <https://interestingengineering.com/10-of-the-best-engineering-quotes-ever> (visitado 05-09-2020).
- [32] (). «775-9009F-C-CC Datasheet ( Hoja de Datos) - DC Motor,» dirección: <http://www.datasheet.es/PDF/917192/775-9009F-C-CC-pdf.html> (visitado 28-06-2020).
- [33] *Motor de corriente continua,* en Wikipedia, la enciclopedia libre, 26 de jun. de 2020. dirección: [https://es.wikipedia.org/w/index.php?title=Motor\\_de\\_corriente\\_continua&oldid=127266375](https://es.wikipedia.org/w/index.php?title=Motor_de_corriente_continua&oldid=127266375) (visitado 28-06-2020).
- [34] Banggood.com. (). «3pcs 42mm 12V Nema 17 Stepper de dos fases motor para impresora 3D Office Equipment from Ordenador & Reds on banggood.com,» dirección: <https://www.banggood.com/es/3pcs-42mm-12V-Nema-17-Two-Phase-Stepper-Motor-For-3D-Printer-p-1556469.html> (visitado 28-06-2020).
- [35] (). «Stepper Motors: Types, Uses and Working Principle | Article | MPS,» dirección: <https://www.monolithicpower.com/en/stepper-motors-basics-types-uses> (visitado 28-06-2020).
- [36] (). «Mini Servomotor - Ebotics,» dirección: <https://www.ebotics.com/es/producto/mini-servomotor/> (visitado 28-06-2020).
- [37] (20 de oct. de 2019). «How Servo Motor Works & Interface It With Arduino,» dirección: <https://lastminuteengineers.com/servo-motor-arduino-tutorial/> (visitado 28-06-2020).
- [38] *Zona Maker - Servo-Motores.* dirección: <https://www.zonamaker.com/electronica/intro-electronica/componentes/motores/servo-motores> (visitado 16-06-2020).
- [39] (). «900-00005 | Servomotor Parallax Inc 140 mA, 4 → 6 V | RS Components,» dirección: <https://es.rs-online.com/web/p/servomotores/7813058/> (visitado 28-06-2020).

# Anexo A

## Código fuente S2

### A.1. *Header files*

```

1  /* Microchip Technology Inc. and its subsidiaries. You may use this software
2   * and any derivatives exclusively with Microchip products.
3   *
4   * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
5   * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
6   * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
7   * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
8   * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
9   *
10  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
11  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
12  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
13  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
14  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS
15  * IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF
16  * ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
17  *
18  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
19  * TERMS.
20  */
21
22 /*
23  * File:
24  * Author:
25  * Comments:
26  * Revision history:
27  */
28
29 // This is a guard condition so that contents of this file are not included
30 // more than once.
31 #ifndef INIT_H
32 #define INIT_H
33
34 #include <xc.h> // include processor files - each processor file is guarded.
35 #include "system_types.h"

```

```
36 void initBoard(void);
37 void initUART(void);
38 void initPWM(void);
39 void TMR1_Initialize(void);
40 void TMR2_Initialize(void);
41 void initUnusedIOPorts(void);
42 void initDigitalPorts(void);
43
44 void init_pins(void);
45 void init_clock(void);
46 void init_interrupts(void);
47
48 /**
49 * @Summary
50 *     Enables global interrupts of the dsPIC33EP512GM604
51 *
52 * @Description
53 *     This routine enables the global interrupt bit for the dsPIC33EP512GM604
54 *
55 * @Preconditions
56 *     None.
57 *
58 * @Returns
59 *     None.
60 *
61 * @Param
62 *     None.
63 *
64 * @Example
65 *     <code>
66 *         void SYSTEM_Initialize(void)
67 *         {
68 *             // Other initializers are called from this function
69 *             INTERRUPT_GlobalEnable ();
70 *         }
71 *     </code>
72 *
73 */
74
75 inline static void INTERRUPT_GlobalEnable(void)
76 {
77     __builtin_enable_interrupts();
78 }
79
80 /**
81 * Sets the CPU core control register operating mode to a value that is decided by the
82 * SYSTEM_CORCON_MODES argument.
83 * @param modeValue SYSTEM_CORCON_MODES initialization mode specifier
84 * @example
85 * <code>
86 * SYSTEM_CorconModeOperatingSet(CORCON_MODE_ENABLEALLSATNORMAL_ROUNDUNBIASED);
87 * </code>
88 */
89 inline static void SYSTEM_CorconModeOperatingSet(SYSTEM_CORCON_MODES modeValue)
90 {
91     CORCON = (CORCON & 0x00F2) | modeValue;
```

```

92 }
93
94 void system_initialize(void);
95
96 #endif /* INIT_H */

```

Listing A.1: pArm-S2/pArm.X/init.h

```

1  /* Microchip Technology Inc. and its subsidiaries. You may use this software
2   * and any derivatives exclusively with Microchip products.
3   *
4   * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
5   * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
6   * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
7   * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
8   * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
9   *
10  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
11  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
12  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
13  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
14  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS
15  * IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF
16  * ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
17  *
18  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
19  * TERMS.
20 */
21
22 /*
23  * File: interrupts.h
24  * Author: Javinator9889
25  * Comments: base interrupts handler with different handlers
26  * Revision history: 1.0
27 */
28
29 // This is a guard condition so that contents of this file are not included
30 // more than once.
31 #ifndef INTERRUPTS_H
32 #define INTERRUPTS_H
33
34 #include <xc.h>
35
36 // Global variable used to notify Input Change on I/O Ports
37 extern volatile int _ICNFLAG;
38
39 // Define Timer interrupts
40 void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void);
41 void __attribute__((__interrupt__, no_auto_psv)) _T2Interrupt(void);
42 void __attribute__((__interrupt__, no_auto_psv)) _U1TXInterrupt(void);
43 void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void);
44 void __attribute__((__interrupt__, no_auto_psv)) _CNIInterrupt(void);
45 void __attribute__((__interrupt__, no_auto_psv)) _U1ErrInterrupt(void);
46
47 #endif /* INTERRUPTS_H */

```

Listing A.2: pArm-S2/pArm.X/interrupts.h

```
1 /* Microchip Technology Inc. and its subsidiaries. You may use this software
2 * and any derivatives exclusively with Microchip products.
3 *
4 * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
5 * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
6 * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
7 * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
8 * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
9 *
10 * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
11 * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
12 * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
13 * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
14 * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS
15 * IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF
16 * ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
17 *
18 * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
19 * TERMS.
20 */
21
22 /*
23 * File: pragmas.h
24 * Author: Javinator9889
25 * Comments: A collection of configurations used in dsPIC33EP
26 * Revision history: 1.0
27 */
28
29 // This is a guard condition so that contents of this file are not included
30 // more than once.
31 #ifndef PRAGMAS_H
32 #define PRAGMAS_H
33
34 #ifndef CONFIG_SIMULATOR
35
36 // FICD
37 #pragma config ICS = PGD1      //ICD Communication Channel Select bits->Communicate on PGEC1 and
38 // PGED1
39 #pragma config JTAGEN = OFF    //JTAG Enable bit->JTAG is disabled
40
41 // FPOR
42 #pragma config BOREN = ON     //=>BOR is enabled
43 #pragma config ALTI2C1 = OFF   //Alternate I2C1 pins->I2C1 mapped to SDA1/SCL1 pins
44 #pragma config ALTI2C2 = OFF   //Alternate I2C2 pins->I2C2 mapped to SDA2/SCL2 pins
45 #pragma config WDTWIN = WIN25  //Watchdog Window Select bits->WDT Window is 25% of WDT period
46
47 // FWDT
48 #pragma config WDTPS = PS32768 //Watchdog Timer Postscaler bits->1:32768
49 #pragma config WDTPRE = PR128   //Watchdog Timer Prescaler bit->1:128
50 #pragma config PLLKEN = ON     //PLL Lock Enable bit->Clock switch to PLL source will wait until
51 // the PLL lock signal is valid.
52 #pragma config WNDIS = OFF     //Watchdog Timer Window Enable bit->Watchdog Timer in Non-Window
53 // mode
54 #pragma config FWDTEN = OFF    //Watchdog Timer Enable bit->Watchdog timer enabled/disabled by
55 // user software
```

```

52 // FOSC
53 #pragma config POSCMD = NONE      //Primary Oscillator Mode Select bits->Primary Oscillator
54             disabled
55 #pragma config OSCIOFNC = OFF     //OSC2 Pin Function bit->OSC2 is clock output
56 #pragma config IOL1WAY = ON       //Peripheral pin select configuration->Allow only one reconfig
57             uration
58 #pragma config FCKSM = CSECME    //Clock Switching Mode bits->Both Clock switching and Fail-
59             safe Clock Monitor are enabled
60
61 // FOSCSEL
62 #pragma config FNOSC = FRC       //Oscillator Source Selection->FRC
63 #pragma config PWMLOCK = ON      //PWM Lock Enable bit->Certain PWM registers may only be written
64             after key sequence
65 #pragma config IESO = OFF        //Two-speed Oscillator Start-up Enable bit->Start up with user-
66             selected oscillator source
67
68 // FGS
69 #pragma config GWRP = OFF        //General Segment Write-Protect bit->General Segment may be
70             written
71 #pragma config GCP = OFF         //General Segment Code-Protect bit->General Segment Code protect is
72             Disabled
73
74 #endif /* CONFIG_SIMULATOR */
75
76 #endif /* PRAGMAS_H */

```

Listing A.3: pArm-S2/pArm.X/pragmas.h

```

1 /**
2  *Generated PIC24 / dsPIC33 / PIC32MM MCUs Source File
3
4  *@Company:
5   Microchip Technology Inc.
6
7  *@File Name:
8   system_types.h
9
10 @Summary:
11   This is the system_types.h file generated using PIC24 / dsPIC33 / PIC32MM MCUs
12
13 @Description:
14   This header file provides implementations for driver APIs for all modules selected in the
15   GUI.
16   Generation Information :
17     Product Revision : PIC24 / dsPIC33 / PIC32MM MCUs - 1.167.0
18     Device          : dsPIC33EP512GM604
19   The generated drivers are tested against the following:
20     Compiler        : XC16 v1.50
21     MPLAB          : MPLAB X v5.35
22
23 */
24
25 /* (c) 2020 Microchip Technology Inc. and its subsidiaries. You may use this
26  software and any derivatives exclusively with Microchip products.

```

```
27 THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
28 EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
29 WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
30 PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
31 WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
32
33 IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
34 INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
35 WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
36 BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
37 FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
38 ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
39 THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
40
41 MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
42 TERMS.
43 */
44
45 #ifndef SYSTEM_TYPES_H
46 #define SYSTEM_TYPES_H
47
48 /**
49  Section: Type defines
50 */
51
52 /**
53 * CORCON initialization type enumerator. Supported types:
54 * CORCON_MODE_PORVALUES
55 * CORCON_MODE_ENABLEALLSATNORMAL_ROUNDBIASED
56 * CORCON_MODE_ENABLEALLSATNORMAL_ROUNDUNBIASED
57 * CORCON_MODE_DISABLEALLSAT_ROUNDBIASED
58 * CORCON_MODE_DISABLEALLSAT_ROUNDUNBIASED
59 * CORCON_MODE_ENABLEALLSATSUPER_ROUNDBIASED
60 * CORCON_MODE_ENABLEALLSATSUPER_ROUNDUNBIASED
61 */
62 typedef enum tagCORCON_MODE_TYPE
63 {
64     CORCON_MODE_PORVALUES      = 0x0020,           /* Use POR values of CORCON */
65     CORCON_MODE_ENABLEALLSATNORMAL_ROUNDBIASED = 0x00E2, /* Enable saturation for ACCA,
66     ACCB                         * and Dataspace write, enable
67
68     normal                      * ACCA/ACCB saturation mode and
69
70     set                         * rounding to Biased (
71
72     conventional                * mode. Rest of CORCON settings
73
74     are                         * set to the default POR values.
75
76     CORCON_MODE_ENABLEALLSATNORMAL_ROUNDUNBIASED = 0x00E0, /* Enable saturation for ACCA,
77     ACCB                         * and Dataspace write, enable
78
79     normal                      * ACCA/ACCB saturation mode and
80
81     set                         *
```

```

75           * rounding to Unbiased (
76           * mode. Rest of CORCON settings
77           * set to the default POR values.
78           */
79           /** Disable saturation for ACCA,
80               ACCB
81           * and Dataspace write and set
82           * rounding to Biased (
83           * mode. Rest of CORCON settings
84           * set to the default POR values.
85           */
86           /** Disable saturation for ACCA,
87               ACCB
88           * and Dataspace write and set
89           * rounding to Unbiased (
90           * mode. Rest of CORCON settings
91           * set to the default POR values.
92           */
93           /** Enable saturation for ACCA, ACCB
94               * and Dataspace write, enable
95               * ACCA/ACCB saturation mode and
96               * rounding to Biased (
97               * mode. Rest of CORCON settings
98               * set to the default POR values.
99               */
100          /** Enable saturation for ACCA, ACCB
101              * and Dataspace write, enable
102              * ACCA/ACCB saturation mode and
103              * rounding to Unbiased (
104              * mode. Rest of CORCON settings
105              * set to the default POR values.
106              */
107 } SYSTEM_CORCON_MODES;
108
109 #endif /* SYSTEM_TYPES_H */
110 /**
111 End of File
112 */

```

Listing A.4: pArm-S2/pArm.X/system\_types.h

```

2 * and any derivatives exclusively with Microchip products.
3 *
4 * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
5 * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
6 * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
7 * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
8 * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
9 *
10 * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
11 * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
12 * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
13 * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
14 * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS
15 * IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF
16 * ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
17 *
18 * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
19 * TERMS.
20 */
21
22 /*
23 * File: defs.h
24 * Author: Javinator9889
25 * Comments: Includes common definitions that may be used in the entire project
26 * Revision history: 1.0
27 */
28
29 #ifndef DEFS_H
30 #define DEFS_H
31 #include <stdint.h>
32
33 // CLK definitions
34 #define FOSC      119808000UL
35 #define FCY       59904000UL
36 #define CLK_SPEED 7.3728
37 #define FCLK_SPEED 73728000UL
38 #define PRESCALE ((uint8_t) FOSC / FCLK_SPEED)
39
40 // Servo definitions
41 #define MIN_PULSE_MS 0.75
42 #define MAX_PULSE_MS 2.25
43
44 // UART delay
45 #define DELAY_105uS asm volatile ("REPEAT, #4201"); Nop(); // 105uS delay
46
47 // Support for external printf implementation
48 // #define USE_CUSTOM_PRINTF
49
50 #endif /* DEFS_H */

```

Listing A.5: pArm-S2/pArm.X/utils/defs.h

```

1 /* Microchip Technology Inc. and its subsidiaries. You may use this software
2 * and any derivatives exclusively with Microchip products.
3 *
4 * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

```

```

5  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
6  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
7  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
8  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
9  *
10 * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
11 * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
12 * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
13 * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
14 * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS
15 * IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF
16 * ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
17 *
18 * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
19 * TERMS.
20 */
21
22 /*
23 * File: time.h
24 * Author: Javinator9889
25 * Comments: the time management library
26 * Revision history: 1.0
27 */
28
29 // This is a guard condition so that contents of this file are not included
30 // more than once.
31 #ifndef TIME_H
32 #define TIME_H
33
34 #include "types.h"
35
36 extern volatile time_t _now_ms;
37 extern volatile time_t _now_us;
38
39 void _updateMs(void);
40 time_t now(void);
41 time_t now_us(void);
42 void increment_us(void);
43 void set_time(time_t value_us);
44
45 #endif /* TIME_H */

```

Listing A.6: pArm-S2/pArm.X/utils/time.h

```

1  /* Microchip Technology Inc. and its subsidiaries. You may use this software
2  * and any derivatives exclusively with Microchip products.
3  *
4  * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
5  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
6  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
7  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
8  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
9  *
10 * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
11 * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
12 * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

```

```

13 * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
14 * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS
15 * IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF
16 * ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
17 *
18 * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
19 * TERMS.
20 */
21
22 /*
23 * File: uart.h
24 * Author: Javinator9889
25 * Comments: UART general I/O file handler
26 * Revision history: 1.0
27 */
28
29 // This is a guard condition so that contents of this file are not included
30 // more than once.
31 #ifndef UART_H
32 #define UART_H
33
34 #include <stdint.h>
35
36 void putch(char data);
37
38 uint8_t getch(void);
39
40#endif /* UART_H */

```

Listing A.7: pArm-S2/pArm.X/utils/uart.h

```

1 /* Microchip Technology Inc. and its subsidiaries. You may use this software
2 * and any derivatives exclusively with Microchip products.
3 *
4 * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
5 * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
6 * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
7 * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
8 * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
9 *
10 * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
11 * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
12 * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
13 * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
14 * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS
15 * IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF
16 * ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
17 *
18 * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
19 * TERMS.
20 */
21
22 /*
23 * File: utils.h
24 * Author: Javinator9889
25 * Comments: Standard utils for using them along the project

```

```
26 * Revision history: 1.0
27 */
28
29 // This is a guard condition so that contents of this file are not included
30 // more than once.
31 #ifndef UTILS_H
32 #define UTILS_H
33
34 #include <math.h>
35 #include <stddef.h>
36 #include "defs.h"
37
38 #define arrsize(array) (sizeof (array) / sizeof *(array))
39 #define foreach(idxtyp, item, array) \
40     idxtyp* item; \
41     size_t size = arrsize(array); \
42     for (item = array; item < (array + size); ++item)
43 #define clockCyclesPerMicrosecond() ( FCY / 1000000UL )
44
45 /**
46 * Maps a value in between the output range by the given input range
47 * @param x the value to map.
48 * @param in_min the minimum acceptable value.
49 * @param in_max the maximum acceptable value.
50 * @param out_min the minimum output value.
51 * @param out_max the maximum output value.
52 * @return the mapped 'x' value in between [out_min, out_max]
53 * @see https://www.arduino.cc/reference/en/language/functions/math/map/
54 */
55 inline long map(long x, long in_min, long in_max, long out_min, long out_max)
56 {
57     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
58 }
59
60 inline double roundp(double value) {
61     return floor(value + .5);
62 }
63
64 inline double preciseMap(
65     double value,
66     double in_min,
67     double in_max,
68     double out_min,
69     double out_max)
70 {
71     double slope = 1.0 * (out_max - out_min) / (in_max - in_min);
72     return out_min + roundp(slope * (value - in_min));
73 }
74
75 inline double mapf(double x, double in_min, double in_max, double out_min, double out_max)
76 {
77     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
78 }
79
80#endif /* UTILS_H */
```

Listing A.8: pArm-S2/pArm.X/utils/utils.h

```

1  /* Microchip Technology Inc. and its subsidiaries. You may use this software
2   * and any derivatives exclusively with Microchip products.
3   *
4   * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
5   * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
6   * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
7   * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
8   * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
9   *
10  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
11  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
12  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
13  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
14  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS
15  * IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF
16  * ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
17  *
18  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
19  * TERMS.
20  */
21
22 /*
23  * File: types.h
24  * Author: Javinator9889
25  * Comments: A header file that contains all custom data types used in this project.
26  * Revision history: 1.0
27  */
28
29 #ifndef TYPES_H
30 #define TYPES_H
31
32 #include <stdint.h>
33
34 // Time definitions
35 #ifndef time_t
36 typedef uint64_t time_t;
37 #define time_t time_t
38 #endif
39
40#endif /* TYPES_H */

```

Listing A.9: pArm-S2/pArm.X/utils/types.h

```

1  /* Microchip Technology Inc. and its subsidiaries. You may use this software
2   * and any derivatives exclusively with Microchip products.
3   *
4   * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
5   * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
6   * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
7   * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
8   * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
9   *

```

```

10 * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
11 * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
12 * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
13 * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
14 * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS
15 * IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF
16 * ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
17 *
18 * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
19 * TERMS.
20 */
21
22 /*
23 * File: servo.h
24 * Author: Javinator9889
25 * Comments: Servo controller header file
26 * Revision history: 1.0
27 */
28
29 #ifndef SERVO_H
30 #define SERVO_H
31
32 #include <stdint.h>
33 #include "../utils/defs.h"
34 #include "../utils/types.h"
35
36 #define usToTicks(_us)      ( (clockCyclesPerMicrosecond() * _us) / PRESCALE )
37 #define ticksToUs(_ticks)   ( ((unsigned)_ticks * PRESCALE) / clockCyclesPerMicrosecond() )
38 #define MIN_PULSE_WIDTH    usToTicks((unsigned long) MIN_PULSE_MS * 1000)
39 #define MAX_PULSE_WIDTH    usToTicks((unsigned long) MAX_PULSE_MS * 1000)
40
41 // Servo definition
42 typedef struct {
43     uint16_t *dutyCycleRegister;
44 } Servo;
45
46 void writeAngle(const Servo *servo, uint16_t angle);
47 void writeMilliseconds(const Servo *servo, double ms);
48 void writeValue(const Servo *servo, uint16_t dutyCycleValue);
49
50 #endif /* SERVO_H */

```

Listing A.10: pArm-S2/pArm.X/motor/servo.h

```

1 /* Microchip Technology Inc. and its subsidiaries. You may use this software
2 * and any derivatives exclusively with Microchip products.
3 *
4 * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
5 * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
6 * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
7 * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
8 * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
9 *
10 * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
11 * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
12 * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

```

```

13 * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
14 * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS
15 * IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF
16 * ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
17 *
18 * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
19 * TERMS.
20 */
21
22 /*
23 * File: motor.h
24 * Author: Javinator9889
25 * Comments: The motor handler header file definition
26 * Revision history: 1.0
27 */
28
29 #ifndef MOTOR_H
30 #define MOTOR_H
31
32 #include <stdint.h>
33 #include "servo.h"
34 #include "../utils/types.h"
35 #include "../utils/utils.h"
36
37 #define MAX_MOTORS 4U
38
39 typedef struct {
40     const Servo *servoHandler;
41     volatile time_t ticks;
42     const uint8_t id;
43 } Motor;
44
45
46 static void handleInterrupt(void);
47 void move(Motor *motor, uint16_t angle);
48 void home(Motor motor[MAX_MOTORS]);
49 void freeze(Motor *motor);
50 double positionMs(Motor *motor);
51 double position(Motor *motor);
52
53#endif /* MOTOR_H */

```

Listing A.11: pArm-S2/pArm.X/motor/motor.h

## A.2. *Source files*

```

1 /*
2 * File: init.c
3 * Author: javinator9889
4 *
5 * Created on 3 de julio de 2020, 13:07
6 */
7

```

```
8 #include "init.h"
9 #include "utils/utils.h"
10 #include "utils/defs.h"
11 #include "system_types.h"
12
13
14 void init_pins(void) {
15     // Unlock the Peripheral Pin Selector (PPS)
16     // for allowing changes on TRIS ports without
17     // affecting expected device behavior.
18     // 0xBF is a shortcut for ~(1 << 6) == 191
19 #ifndef CONFIG_SIMULATOR
20     __builtin_write_OSCCONL(OSCCON & 0xBF); // unlock PPS
21 #endif
22
23     RPOR6bits.RP54R = 0x0001; //RC6->UART1:U1TX
24     RPINR18bits.U1RXR = 0x0037; //RC7->UART1:U1RX
25
26     TRISCbits.TRISC7 = 1;
27     TRISCbits.TRISC6 = 0;
28
29     // Lock again the PPS as we are done
30     // configuring the remappable ports.
31     // 0x40 is a shortcut for (1 << 6) == 64
32 #ifndef CONFIG_SIMULATOR
33     __builtin_write_OSCCONL(OSCCON | 0x40); // lock PPS
34 #endif
35 }
36
37
38 void initBoard(void) {
39     // Disable watchdog timer
40     RCONbits.SWDTEN = 0;
41
42 #ifndef CONFIG_SIMULATOR
43     // Setup de PLL for reaching 40 MHz with a 7.3728 clock.
44     // Maximum speed is of 140 MHz as the maximum temperature
45     // of 85 °C implies 70 MIPS.
46     //
47     // For working at ~120 MHz:
48     // F_osc = F_in * M / (N1 * N2)
49     // F_cy = F_osc / 2
50     // F_osc ~= 120 MHz -> F_osc = 7.3728 * 65 / (2 * 2) = 119.808 MHz
51     // F_cy = F_osc / 2 = 59.904 MHz
52     //
53     // Then, setup the PLL's prescaler, postcaler and divisor
54     PLLFBDbits.PLLDIV = 63; // M = PLLDIV + 2 -> PLLDIV = 65 - 2 = 63
55     CLKDIVbits.PLLPOST = 0; // N2 = 2 * (PLLPOST + 1) -> PLLPOST = (N2 / 2) - 1 = 0
56     CLKDIVbits.PLLPRE = 0; // N1 = PLLPRE + 2; -> PLLPRE = N1 - 2 = 0
57
58     // Notify clock to use PLL
59     // Start clock switching to primary
60     __builtin_write_OSCCONH(0x03);
61     __builtin_write_OSCCONL(0x01);
62
63     // And wait for clock switching to happen
```

```
64 // First, wait for clock switch to occur
65 // and thenm wait the PLL to lock
66 while (OSCCONbits.COSC != 0b011);
67 while (OSCCONbits.LOCK != 1);
68 #endif
69 }
70
71 void init_clock(void) {
72 #ifndef CONFIG_SIMULATOR
73     // FRCDIV FRC/1; PLLPRE 2; DOZE 1:8; PLLPOST 1:2; DOZEN disabled; ROI disabled;
74     CLKDIV = 0x3000;
75     // TUN Center frequency;
76     OSCTUN = 0x00;
77     // ROON disabled; ROSEL FOSC; RODIV 0; ROSSLP disabled;
78     REFOCON = 0x00;
79     // Setup de PLL for reaching 40 MHz with a 7.3728 clock.
80     // Maximum speed is of 140 MHz as the maximum temperature
81     // of 85 °C implies 70 MIPS.
82     //
83     // For working at ~120 MHz:
84     // F_osc = F_in * M / (N1 * N2)
85     // F_cy = F_osc / 2
86     // F_osc ~≈ 120 MHz -> F_osc = 7.3728 * 65 / (2 * 2) = 119.808 MHz
87     // F_cy = F_osc / 2 = 59.904 MHz
88     //
89     // Then, setup the PLL's prescaler, postcaler and divisor
90     PLLFBDbits.PLLDIV = 63; // M = PLLDIV + 2 -> PLLDIV = 65 - 2 = 63
91     CLKDIVbits.PLLPOST = 0; // N2 = 2 * (PLLPOST + 1) -> PLLPOST = (N2 / 2) - 1 = 0
92     CLKDIVbits.PLLPRE = 0; // N1 = PLLPRE + 2; -> PLLPRE = N1 - 2 = 0
93     // AD1MD enabled; PWMMMD enabled; T3MD enabled; T4MD enabled; T1MD enabled; U2MD enabled;
94     // T2MD enabled; U1MD enabled; QEI1MD enabled; SPI2MD enabled; SPI1MD enabled; C2MD enabled;
95     // C1MD enabled; DCIMD enabled; T5MD enabled; I2C1MD enabled;
96     PMD1 = 0x00;
97     // OC5MD enabled; OC6MD enabled; OC7MD enabled; OC8MD enabled; OC1MD enabled; IC2MD enabled
98     ; OC2MD enabled; IC1MD enabled; OC3MD enabled; OC4MD enabled; IC6MD enabled; IC7MD enabled;
99     // IC5MD enabled; IC8MD enabled; IC4MD enabled; IC3MD enabled;
100    PMD2 = 0x00;
101    // ADC2MD enabled; PMPMD enabled; U3MD enabled; QEI2MD enabled; RTCCMD enabled; CMPMD
102    enabled; T9MD enabled; T8MD enabled; CRCMD enabled; T7MD enabled; I2C2MD enabled; T6MD
103    enabled;
104    PMD3 = 0x00;
105    // U4MD enabled; CTMUMD enabled; REFOMD enabled;
106    PMD4 = 0x00;
107    // PWM2MD enabled; PWM1MD enabled; PWM4MD enabled; SPI3MD enabled; PWM3MD enabled; PWM6MD
108    enabled; PWM5MD enabled;
109    PMD6 = 0x00;
110    // PTGMD enabled; DMA0MD enabled;
111    PMD7 = 0x00;
112    // CF no clock failure; NOSC FRCPLL; CLKLOCK unlocked; OSWEN Switch is Complete; IOLOCK not
113    -active;
114    __builtin_write_OSCCONH((uint8_t) (0x01));
115    __builtin_write_OSCCONL((uint8_t) (0x01));
116
117    // Wait for Clock switch to occur
118    while (OSCCONbits.OSWEN != 0);
119    // And wait for clock switching to happen
```

```
112 // First, wait for clock switch to occur
113 // and thenm wait the PLL to lock
114 while (OSCCONbits.COSC != 0b011);
115 while (OSCCONbits.LOCK != 1);
116 #endif
117 }
118
119 void init_interrupts(void) {
120     // TI: Timer 2
121     // Priority: 1
122     IPC1bits.T2IP = 1;
123     // TI: Timer 1
124     // Priority: 1
125     IPC0bits.T1IP = 1;
126     // UERI: UART1 Error
127     // Priority: 1
128     IPC16bits.U1EIP = 1;
129 }
130
131 void initUART(void) {
132     IEC0bits.U1TXIE = 0;
133     IEC0bits.U1RXIE = 0;
134
135     // Setup UART
136     // Stop on idle
137     U1MODEbits.USIDL = 1;
138     // Disable IrDA
139     U1MODEbits.IREN = 0;
140     // Use only TX and RX pins
141     // ignoring CTS, RTS and BCLK
142     U1MODEbits.UEN = 0;
143     // Do not wake-up with UART
144     U1MODEbits.WAKE = 0;
145     // Disable loopback mode
146     U1MODEbits.LPBACK = 0;
147     // Do not use automatic baudrate when receiving
148     U1MODEbits.ABAUD = 0;
149     // Disable polarity inversion. Idle state is '1'
150     U1MODEbits.URXINV = 0;
151     // Do not use high speed baudrate
152     U1MODEbits.BRGH = 0;
153     // 8 data bits without parity
154     U1MODEbits.PDSEL = 0;
155     // One stop bit
156     U1MODEbits.STSEL = 0;
157
158     // Calculate the baudrate using the following equation
159     // UxBRG = ((FCY / Desired Baud rate) / 16) - 1
160     // For 9600 bauds and FCY = 59.904E6, the obtained BRG is
161     // -> 389, and the obtained baudrate is: 9600, with an error
162     // of 0%
163     U1BRG = 389;
164
165     // Interrupt after one RX character is received;
166     // UTXISEL0 TX_ONE_CHAR; UTXINV disabled; OERR NO_ERROR_cleared; URXISEL RX_ONE_CHAR;
167     // UTXBRK COMPLETED; UTXEN enabled; ADDEN disabled;
```

```
167 U1STA = 0x400;
168
169 // Enable UART TX Interrupt
170 IEC0bits.U1TXIE = 1;
171 IEC0bits.U1RXIE = 1;
172 IFS0bits.U1RXIF = 0;
173 IFS0bits.U1TXIF = 0;
174 IPC2bits.U1RXIP = 0b110;
175
176 //Make sure to set LAT bit corresponding to TxPin as high before UART initialization
177 U1MODEbits.UARTEN = 1; // enabling UART ON bit
178 U1STAbits.UTXEN = 1;
179
180 // Wait 105 uS (when baudrate is 9600) for a first
181 // transmission bit to be sent and detected, so then
182 // the UART can be used
183 DELAY_105uS;
184 }
185
186 void initPWM(void) {
187 TRISBbits.TRISB11 = 0; // PWM3L
188 TRISBbits.TRISB13 = 0; // PWM2L
189 TRISBbits.TRISB15 = 0; // PWM1L
190 TRISBbits.TRISB14 = 0; // PWM1H
191
192 PTCON2bits.PCLKDIV = 0b110; // Prescaler 1:32
193
194 // Setup PWM period - the motors have a
195 // minimum time in between pulses of 20ms,
196 // so the frequency must be of 50 Hz.
197 //
198 // F_osc = 119.808 MHz
199 // F_PWM = 50 Hz
200 // PWM_Prescaler = 64
201 // PTPER = F_osc / (F_PWM * PWM_Prescaler) --> PTPER = 119.808 MHz / (50 Hz * 32)
202 // = 37440 = PTPER --> F_PWM = 50.000 Hz
203 PTPER = 37440;
204
205 // Initialize independent time base to zero.
206 // As we are using PWMxL, we only use
207 // SPHASEx ports. If using PWMxH, just change
208 // SPHASEx to PHASEx ones.
209 SPHASE3 = 0;
210 SPHASE2 = 0;
211 SPHASE1 = 0;
212 PHASE1 = 0;
213
214 // By default, set no duty cycle of programmed signals
215 SDC3 = 0;
216 SDC2 = 0;
217 SDC1 = 0;
218 PDC1 = 0;
219
220 // Disable Dead Time values
221 ALTDTR4 = 0;
222 ALTDTR3 = 0;
```

```
223 ALTDTR2 = 0;
224 ALTDTR1 = 0;
225
226 DTR4 = 0;
227 DTR3 = 0;
228 DTR2 = 0;
229 DTR1 = 0;
230
231 // True independent work mode, so then both PWMxH and
232 // PWMxL can be used independently
233 IOCON4bits.PMOD = 0b11;
234 IOCON3bits.PMOD = 0b11;
235 IOCON2bits.PMOD = 0b11;
236 IOCON1bits.PMOD = 0b11;
237
238 // Disable PWM fault input
239 FCLCON4bits.FLTMOD = 0b11;
240 FCLCON3bits.FLTMOD = 0b11;
241 FCLCON2bits.FLTMOD = 0b11;
242 FCLCON1bits.FLTMOD = 0b11;
243
244 // Do not swap LOW/HIGH values
245 IOCON4bits.SWAP = 0;
246 IOCON3bits.SWAP = 0;
247 IOCON2bits.SWAP = 0;
248 IOCON1bits.SWAP = 0;
249
250 // Set pins as PWM ones
251 IOCON4bits.PENL = 1;
252 IOCON3bits.PENL = 1;
253 IOCON2bits.PENL = 1;
254 IOCON1bits.PENL = 1;
255 IOCON1bits.PENH = 1;
256 // Disable high output as we are not using it
257 IOCON4bits.PENH = 0;
258 IOCON3bits.PENH = 0;
259 IOCON2bits.PENH = 0;
260
261 // Set PWM configurations to zero by default
262 PWMCON4 = 0;
263 PWMCON3 = 0;
264 PWMCON2 = 0;
265 PWMCON1 = 0;
266
267 // Disable dead time in-between output switches
268 PWMCON4bits.DTC = 0b10;
269 PWMCON3bits.DTC = 0b10;
270 PWMCON2bits.DTC = 0b10;
271 PWMCON1bits.DTC = 0b10;
272
273 // and enable the PWM module
274 PTCONbits.PTEN = 1;
275 }
276
277 void TMR1_Initialize(void) {
278     TMR1 = 0x00;
```

```
279 // Period = 1 us;
280 // Frequency = 59904000 Hz;
281 // PR1 = 59 == an interrupt ~= 1.0016 us
282 PR1 = 0x3B;
283
284 // TON enabled; -> 1
285 // Empty bit -> 0
286 // TSIDL disabled; -> 0
287
288 // Empty bit -> 0
289 // Empty bit -> 0
290 // Empty bit -> 0
291 // Empty bit -> 0
292
293 // Empty bit -> 0
294 // Empty bit -> 0
295 // TGATE disabled; -> 0
296 // TCKPS 1:1; -> 0
297
298 // Empty bit -> 0
299 // TSYNC enabled -> 0
300 // TCS FOSC/2; -> 0
301 // Empty bit -> 0
302 // 1000 0000 0000 0000 == 0x8000
303 T1CON = 0x8000;
304
305 // Clear interrupt flag,
306 IFS0bits.T1IF = 0;
307 // set priority to maximum
308 IPC0bits.T1IP = 6;
309 // and enable interrupts
310 IEC0bits.T1IE = 1;
311 }
312
313 void TMR2_Initialize(void) {
314 // Reset TMR2 to zero
315 TMR2 = 0x00;
316
317 // Period = 1 ms;
318 // Frequency = 59904000 Hz;
319 // PR2 59903 == an interrupt each millisecond.
320 PR2 = 0xE9FF;
321 // TON enabled; -> 1
322 // Empty bit -> 0
323 // TSIDL disabled; -> 0
324
325 // Empty bit -> 0
326 // Empty bit -> 0
327 // Empty bit -> 0
328 // Empty bit -> 0
329
330 // Empty bit -> 0
331 // Empty bit -> 0
332 // TGATE disabled; -> 0
333 // TCKPS 1:1; -> 0
334 }
```

```
335 // T32 16 Bit; -> 0
336 // Empty bit -> 0
337 // TCS FOSC/2; -> 0
338 // Empty bit -> 0
339 // 1000 0000 0000 0000 == 0x8000
340 T2CON = 0x8000;
341
342 // Clear interrupt flag...
343 IFS0bits.T2IF = 0;
344 // and enable TMR2 interruptions
345 IEC0bits.T2IE = 1;
346 }
347
348 void initDigitalPorts(void)
349 {
350 //Digital Ports for micro-interruptors, set as input
351 TRISAbits.TRISA0 = 1;
352 TRISAbits.TRISA1 = 1;
353 TRISBbits.TRISB0 = 1;
354 TRISBbits.TRISB1 = 1;
355
356 //Input Change Notification Interrupt configuration
357 _CNIP = 5; // priority (7 = highest)
358 _CNIE = 1; // Enable CN interrupts
359 _CNIF = 0; // Interrupt flag cleared
360 CNENBbits.CNIEB0 = 1;
361 CNENBbits.CNIEB1 = 1;
362 CNENAbits.CNIEA0 = 1;
363 CNENAbits.CNIEA1 = 1;
364
365
366 //Digital Ports for LED lights, set as output.
367 TRISBbits.TRISB5 = 0;
368 TRISBbits.TRISB6 = 0;
369 TRISBbits.TRISB7 = 0;
370 TRISBbits.TRISB8 = 0;
371
372 //Set I/O ports to digital, clear the analogic enable bit.
373 ANSELAbits.ANSA0 = 0;
374 ANSELAbits.ANSA1 = 0;
375 ANSELBbits.ANSB0 = 0;
376 ANSELBbits.ANSB1 = 0;
377 ANSELBbits.ANSB7 = 0;
378 ANSELBbits.ANSB8 = 0;
379 }
380
381 inline void system_initialize(void) {
382 init_pins();
383 init_clock();
384 initUART();
385 TMR1_Initialize();
386 TMR2_Initialize();
387 initPWM();
388 INTERRUPT_GlobalEnable();
389 SYSTEM_CORCONModeOperatingSet(CORCON_MODE_PORVALUES);
390 // Init interrupts only after the hole initialization process is finished
```

```

391     init_interrupts();
392 }
```

Listing A.12: pArm-S2/pArm.X/init.c

```

1 #include "interrupts.h"
2 #include "utils/time.h"
3
4 extern char receivedValue;
5 volatile int _ICNFLAG = 0; // Auxiliar Flag defined in interrupts.h
6
7 void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void) {
8     _now_us += 1ULL;
9     // Clear Timer1 interrupt
10    IFS0bits.T1IF = 0;
11 }
12
13 void __attribute__((__interrupt__, no_auto_psv)) _T2Interrupt(void) {
14     _now_ms += 1ULL;
15     // Clear Timer2 interrupt
16     IFS0bits.T2IF = 0;
17 }
18
19 void __attribute__((__interrupt__, no_auto_psv)) _U1TXInterrupt(void) {
20     IFS0bits.U1TXIF = 0; // Clear TX Interrupt flag
21 }
22
23 void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void) {
24     IFS0bits.U1RXIF = 0;
25     printf("!\n");
26     if (U1STAbits.FERR == 1)
27         return;
28     if (U1STAbits.URXDA == 1) {
29         receivedValue = U1RXREG;
30     }
31 }
32
33 /*void __attribute__((__interrupt__, no_auto_psv)) _U2RXInterrupt(void) {
34     printf("RXI2 received");
35     if (U2STAbits.FERR == 1);
36     if (U2STAbits.OERR == 1)
37         U2STAbits.OERR = 0;
38     if (U2STAbits.URXDA == 1) {
39         receivedValue = U2RXREG;
40     }
41     IFS1bits.U2RXIF = 0;
42 }*/
43
44 void __attribute__((__interrupt__, no_auto_psv)) _CNInterrupt(void) {
45     _ICNFLAG = 1; // Notify the input change using the auxiliar flag
46     _CNIF = 0; // Clear the interruption flag
47 }
48
49 void __attribute__((__interrupt__, no_auto_psv)) _U1ErrInterrupt(void) {
50     if ((U1STAbits.OERR == 1)) {
51         U1STAbits.OERR = 0;
```

```

52     }
53
54     IFS4bits.U1EIF = 0;
55 }
```

Listing A.13: pArm-S2/pArm.X/interrupts.c

```

1 #include "time.h"
2 #include "types.h"
3
4 volatile time_t _now_us = 0ULL;
5 volatile time_t _now_ms = 0ULL;
6
7 inline void _updateMs(void) {
8     _now_ms = (time_t) (_now_us / 1000ULL);
9 }
10
11 inline time_t now(void) {
12     return _now_ms;
13 }
14
15 inline time_t now_us(void) {
16     return _now_us;
17 }
18
19 inline void increment_us(void) {
20     _now_us += 1ULL;
21     _updateMs();
22 }
23
24 inline void set_time(time_t value_us) {
25     _now_us = value_us;
26     _updateMs();
27 }
```

Listing A.14: pArm-S2/pArm.X/utils/time.c

```

1 #include <xc.h>
2 #include "uart.h"
3
4 void putch(char data) {
5     while (!IFS0bits.U1TXIF);
6     U1TXREG = data;
7 //     while (!IFS1bits.U2TXIF);
8 //     U2TXREG = data;
9 }
10
11 uint8_t getch(void) {
12     while (!IFS0bits.U1RXIF);
13     return U1RXREG;
14 }
```

Listing A.15: pArm-S2/pArm.X/utils/uart.c

```

1 #include "servo.h"
2 #include "../utils/defs.h"
```

```

3 #include "../utils/utils.h"
4
5
6 void writeAngle(const Servo *servo, uint16_t angle) {
7     double time = mapf(angle * 1.0, .0, 180.0, .75, 2.25);
8     writeMilliseconds(servo, time);
9 }
10
11 inline void writeMilliseconds(const Servo *servo, double ms) {
12     *servo->dutyCycleRegister = (uint16_t) (FOSC / ((1 / ms) * 1000 * 64));
13 }
14
15 inline void writeValue(const Servo *servo, uint16_t dutyCycleValue) {
16     *servo->dutyCycleRegister = dutyCycleValue;
17 }
```

Listing A.16: pArm-S2/pArm.X/motor/servo.c

```

1 #include "motor.h"
2 #include "../utils/utils.h"
3 #include "../utils/defs.h"
4
5 static inline void handleInterrupt(void) {
6     // TODO
7 }
8
9 inline void move(Motor *motor, uint16_t angle) {
10     writeAngle(motor->servoHandler, angle);
11     // TODO - setup expected ticks and actual ticks
12 }
13
14 inline void home(Motor motors[MAX_MOTORS]) {
15     foreach(Motor, motor, motors) {
16         // TODO - Define "home" position for each motor
17         switch (motor->id) {
18             case 0:
19                 break;
20             case 1:
21                 break;
22             case 2:
23                 break;
24             case 3:
25                 break;
26         }
27     }
28     // And setup interruptors for detecting the end
29 }
30
31 inline void freeze(Motor *motor) {
32     // TODO - Disable motor {id} interrupts / ticks so stop counting
33     const volatile time_t ticks = motor->ticks;
34     const double motorActualMillis = (double) (ticksToUs(ticks) * 1000);
35     writeMilliseconds(motor->servoHandler, motorActualMillis);
36 }
37
38 inline double positionMs(Motor *motor) {
```

```
39     const time_t ticks = motor->ticks;
40     return (double) (ticksToUs(ticks) * 1000);
41 }
42
43 inline double position(Motor *motor) {
44     const time_t ticks = motor->ticks;
45     const double timeMillis = (double) (ticksToUs(ticks) * 1000);
46     return mapf(timeMillis, MIN_PULSE_MS, MAX_PULSE_MS, .0, 180.);
47 }
```

Listing A.17: pArm-S2/pArm.X/motor/motor.c

## Anexo B

### Especificación de requisitos

# Especificación de Requisitos

## para el

### Trabajo de Fin de Grado sobre el manipulador *pArm*

Javier Alonso Silva  
Mihai Octavian Stanescu  
José Alejandro Moya Blanco

*Universidad Politécnica de Madrid*  
*Ingeniería de Computadores*  
*Trabajo de Fin de Grado*  
*Tutor: Norberto Cañas de Paz*

Madrid, 27 de enero 2020

# Índice general

<b>Historial de versiones</b>	<b>1</b>
<b>1. Introducción</b>	<b>2</b>
1.1. Propósito . . . . .	3
1.2. Alcance . . . . .	3
1.3. Definiciones, siglas, y abreviaturas . . . . .	3
1.4. Visión global . . . . .	5
<b>2. Descripción general</b>	<b>6</b>
2.1. Perspectiva del producto . . . . .	6
2.1.1. Interfaz del sistema . . . . .	7
2.1.2. Interfaz de usuario . . . . .	9
2.1.3. Interfaz <i>hardware</i> . . . . .	9
2.1.4. Interfaz de comunicaciones . . . . .	10
2.1.5. Memoria . . . . .	10
2.1.6. Operaciones . . . . .	10
2.2. Funciones del producto . . . . .	10
2.3. Características del usuario . . . . .	11
2.4. Restricciones . . . . .	11
2.5. Supuestos y dependencias . . . . .	12
2.6. Requisitos pospuestos . . . . .	14
<b>3. Requisitos específicos</b>	<b>15</b>
3.1. Requisitos de la interfaz externa . . . . .	15
3.1.1. Interfaz con el usuario . . . . .	15

3.1.2. Interfaz <i>hardware</i> . . . . .	15
3.1.3. Interfaz de comunicaciones . . . . .	16
3.2. Casos de uso . . . . .	17
3.3. Requisitos funcionales . . . . .	21
3.3.1. Requisitos <i>software</i> . . . . .	21
3.3.2. Requisitos <i>hardware</i> . . . . .	30
3.3.3. Requisitos de rendimiento . . . . .	32
3.3.4. Restricciones del diseño . . . . .	32
3.3.5. Atributos del sistema <i>software</i> y <i>hardware</i> . . . . .	33
3.4. Requisitos no funcionales . . . . .	33
<b>A. Enlaces útiles</b>	<b>34</b>

# Historial de versiones

Revisión	Fecha	Autor(es)	Descripción
1.0	27.01.2020	J. Alonso, M. Stanescu, A. Moya	Primera especificación de los requisitos del proyecto.
1.1	04.02.2020	J. Alonso, M. Stanescu, A. Moya	Nuevos requisitos, incluidos casos de uso – eliminado de la especificación el cómo se realizan ciertos procedimientos.

# Capítulo 1

## Introducción

El  $\mu$ Arm es un brazo robótico creado por la compañía UFACTORY<sup>1</sup> el cual se ha diseñado con propósito didáctico y, a su vez, creacional.

En la actualidad, se puede obtener uno a través de su página web o de proveedores externos, pero no está previsto fabricar más, por lo que en un tiempo estará fuera de existencias.

Debido a su propósito didáctico, todos los recursos sobre el manipulador son de código libre, por lo que resultan accesibles a cualquiera que los necesite. Entre otros, se encuentran<sup>2</sup>:

- *Firmware* del  $\mu$ Arm Swift Pro.
- *Software Development Kit (SDK)* de Python para el  $\mu$ Arm Swift Pro.
- *Firmware* que maneja el controlador del brazo.
- *Robot Operating System (ROS)* para el  $\mu$ Arm Swift Pro.
- Distintos ejemplos para toda la gama de brazos robóticos.
- $\mu$ Arm Creator Studio.
- Visión esquemática de las conexiones de la placa Arduino.
- Modelos 3D del brazo robótico.
- Guías de usuario, desarrollador y especificaciones técnicas.

Aprovechando dichos recursos, se pretende desarrollar un brazo robótico basado en el  $\mu$ Arm que esté impreso en 3D y sea controlado por un microcontrolador en conjunción con un ordenador cualquiera. Aprovechando los recursos provistos por UFACTORY, se busca que el brazo desarrollado sea más barato de construir (frente a los casi 800€ que cuesta el original) y que pueda ser desarrollado por cualquiera con acceso a Internet y a los recursos necesarios, a saber, una impresora en 3D y un *software (SW)* de impresión en 3D.

---

<sup>1</sup><https://www.ufactory.cc/#/en/uarmswift>

<sup>2</sup>todos los elementos descritos se encuentran disponibles tanto en GitHub como en la web de UFACTORY

## 1.1. Propósito

El propósito de este documento es establecer un punto de partida claro y conciso que permita empezar el desarrollo del brazo robótico sabiendo los puntos primordiales del mismo. A su vez, también pretende establecer ciertos puntos que se consideran importantes e incluso necesarios para poder continuar el desarrollo del sistema en un futuro, implementando nuevas funciones o arreglando errores que pudieran existir.

Este documento está dirigido a ingenieros que quieran llevar a cabo su propia implementación del brazo robótico o que quieran conocer la estructura en la que se basa el proyecto, así como las necesidades del mismo y las adiciones extraordinarias que se han incluido. A su vez, se pretende que sea accesible a cualquiera que pretenda iniciarse en el mundo de la robótica y que busque estudiar y aprender sobre el brazo robótico.

## 1.2. Alcance

El objetivo principal de este proyecto fin de grado es construir una brazo robótico similar al manipulador  $\mu$ Arm, al cual se le ha asignado el nombre *Printed – Arm* (*pArm*).

Este brazo robótico debe ser capaz de moverse libremente dentro de su campo de movimiento, el cual está limitado por su estructura física. Además, el *pArm* debe ser capaz de coger, transportar y depositar objetos de poco peso y, en consecuencia, debe ser capaz de describir trayectorias previamente planificadas o calculadas en el momento.

Es importante destacar que, dado que el brazo robótico *pArm* no está sensorizado, este no será capaz de moverse de forma completamente autónoma ni de imitar movimientos realizados por el usuario.

Cabe destacar que el brazo robótico está controlado mediante un microcontrolador. Sin embargo, las instrucciones de movimiento y trayectorias no se computan, en principio, en el mismo sino en un ordenador auxiliar.

Debido a la estructura física, tamaño y materiales de fabricación, el *pArm* no es un brazo robótico pensado para la realización de tareas industriales ni para el transporte de cargas pesadas.

En relación a lo anteriormente mencionado, la aplicación principal del *pArm* es didáctica, dado que se busca construir un brazo robótico económico y sencillo que facilite la introducción de los usuarios a este tipo de tecnologías.

## 1.3. Definiciones, siglas, y abreviaturas

**SDK** *Software Development Kit*

**ROS** *Robot Operating System*

**SW** *software*

**HW** *hardware*

**pArm** *Printed – Arm*

**USB** *Universal Serial Bus*

**ODS** Objetivos de Desarrollo Sostenible

**OS** *Open-Source*

**OH** *Open-Hardware*

**S1** Sistema 1 – ordenador

**S2** Sistema 2 – pArm

**GUI** *Graphical User Interface*

**GTK** *GIMP Toolkit*

**SoC** *System On Chip*

**PWM** *Pulse-Width Modulation*

**GPIO** *General Purpose Input/Output*

**UART** *Universal Asynchronous Receiver–Transmitter*

**RAM** *Random Access Memory*

**ADC** *Analog–Digital Conversor*

- **SDK** – colección de herramientas SW disponibles para instalar en un único paquete.
- **ROS** – conjunto de librerías SW que ayudan a construir aplicaciones para robots.
- **Firmware** – SW programado que especifica el orden de ejecución del sistema.
- **Graphical User Interface (GUI)** – siglas que significan “Interfaz Gráfica de Usuario” (en castellano).
- **GIMP Toolkit (GTK)** – biblioteca de componentes gráficos multiplataforma para desarrollar interfaces gráficas de usuario.
- **System On Chip (SoC)** – tecnología de fabricación que integra todos o gran parte de los módulos en un circuito integrado. Un ejemplo muy común es la placa base de un teléfono móvil, la cual es un SoC que integra todos los componentes (antenas, conversores, sensores, etc.).
- **Pulse–Width Modulation (PWM)** – señal en la cual se modifica el ciclo de trabajo de una señal periódica, para transmitir información por un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

- *General Purpose Input/Output* (GPIO) – pin genérico cuyo comportamiento puede ser controlado en tiempo de ejecución.
- *Universal Asynchronous Receiver–Transmitter* (UART) – estándar de comunicación dúplex simultáneos.
- Dúplex – término que define a un sistema que es capaz de mantener una comunicación bidireccional, enviando y recibiendo mensajes de forma simultánea.
- Widget – la parte de una GUI (interfaz gráfica de usuario) que permite al usuario interconectar con la aplicación y el sistema operativo.
- *Random Access Memory* (RAM) – memoria principal del ordenador, donde se guardan programas y datos, sobre la que se pueden efectuar operaciones de lectura y escritura.
- *Deep-Sleep* – estado de un microcontrolador por el cual consume muy poca cantidad de energía.
- *bit* – unidad mínima de información de una computadora.

## 1.4. Visión global

En las siguientes páginas se pasa a explicar los distintos detalles del sistema que debe construirse, estructurando:

- Perspectiva del producto.
- Funciones del producto.
- Características del producto.
- Restricciones.
- Supuestos y dependencias.
- Requisitos propuestos.

En el punto “Requisitos específicos” (3) se detallan los requisitos específicos del sistema.

# Capítulo 2

## Descripción general

### 2.1. Perspectiva del producto

El *pArm* se basa en el trabajo inicial del *μArm*, no utilizando directamente lo desarrollado por la empresa UFACTORY sino aprovechando el trabajo ya realizado y los recursos disponibles para estudiarlos. Si bien es cierto que el *μArm* ya es un sistema avanzado y capaz, como se explicó en la Introducción (1), se pretende estudiar y desarrollar un sistema propio el cual pueda servir para ayudar y facilitar la entrada a este tipo de tecnologías a otras personas, haciéndolo comprensible y, aprovechando la tecnología de la impresión en 3D, fabricable por uno mismo.

Además, en relación a los Objetivos de Desarrollo Sostenible (ODS), con el desarrollo de este sistema se pretende trabajar en:

4 - Educación de Calidad<sup>1</sup>.

7 - Energía Asequible y No Contaminante<sup>2</sup>.

10 - Reducción de las desigualdades<sup>3</sup>.

Para el primero, se tiene en cuenta que el producto se desarrollará siguiendo las iniciativas *Open-Source* (OS) y *Open-Hardware* (OH), las cuales facilitan el acceso a la información a cualquiera que la requiera. Además, se facilitará el desarrollo al completo detallado y explicado, con la resolución de los problemas pertinentes y el porqué de ella.

Para el segundo, el *pArm* utilizará la electricidad como fuente de energía, evitando así otras más contaminantes como las producidas por combustibles fósiles. En añadido, se trabajará para que el consumo de energía sea el menor posible, permitiendo así un mayor tiempo de uso con la misma fuente de alimentación y no abusando de los recursos de los que se disponen.

---

<sup>1</sup><https://www.un.org/sustainabledevelopment/es/education/>

<sup>2</sup><https://www.un.org/sustainabledevelopment/es/energy/>

<sup>3</sup><https://www.un.org/sustainabledevelopment/es/inequality/>

Finalmente, se pretende hacer que el *pArm* tenga un coste bajo, permitiendo así el acceso a los recursos y los procesos de fabricación a todo el mundo que pudiera estar interesado y que disponga de la cantidad mínima necesaria para poder poner en funcionamiento el brazo robótico.

Por otra parte, el *pArm* es dependiente de otro sistema que lo controle, ya que no se plantea como sistema autónomo. Por consiguiente, se proponen diversos métodos de conexión entre el brazo y dicho sistema. Por ejemplo, se puede utilizar el puerto serie (*Universal Serial Bus (USB)*) o bien comunicaciones inalámbricas, como *Bluetooth* y *WiFi*. Además, debido a su disponibilidad multiplataforma, se propone el uso de Python como aplicación de control.

De ahora en adelante, se denominará “Sistema 1 – ordenador (S1)” al equipo que controla al *pArm*; y “Sistema 2 – *pArm* (S2)” al brazo robótico en sí.

Para este proyecto, se ha de desarrollar el SW que se ejecutará en el S1 y tanto el SW como el *hardware* (HW) que irán en el S2, así como la estructura del mismo.

### 2.1.1. Interfaz del sistema

En un principio, el sistema estará dividido en dos módulos:

#### S1

El S1 consiste en un equipo el cual controlará el brazo robótico (S2). Para ello, tal y como se planteó anteriormente, se propone como lenguaje de programación Python, el cual soporta la ejecución con GUI.

En lo referente al sistema operativo, al ser una aplicación en Python la cual es multiplataforma, no se define ninguna restricción respecto al mismo.

Finalmente, se plantea la conexión con el S2 utilizando el puerto serie USB, por lo que también será necesario que el equipo anfitrión S1 disponga de una conexión de ese estilo.

En resumen (ver la tabla 2.1):

Componente	Función	Restricciones
Sistema Operativo	Hospedar y ejecutar la aplicación Python que controlará el brazo robótico.	Debe poder ejecutar aplicaciones Python con GUI, por ejemplo, GTK.
<i>Conexión con S2</i>	Permitir la comunicación con el sistema S2 en modo dúplex.	Velocidad adaptable ( <i>baud-rate</i> ) y capacidad para gran ancho de banda.
Python	Control del sistema S2 y monitoreo del estado del mismo.	Versión Python $\geq 3.6.*$

Tabla 2.1: Requisitos del sistema S1.

En principio, no será necesaria la conexión a Internet, pero tampoco se descarta el uso de la misma en el proyecto a la hora de poder recibir actualizaciones o en lo referente a futuras mejoras.

En añadido, pese a que no se restringe, se sugiere que el equipo que hospede la aplicación disponga de un procesador multinúcleo así como suficiente memoria RAM para poder manejarlas.

## S2

Para el sistema S2 no se ha pensado en ningún microprocesador ni SoC en particular, pero se han contemplado algunos que cumplen con las características requeridas (ver la tabla 2.2).

Será necesario que el circuito escogido disponga de algún tipo de entrada de las propuestas para la comunicación con el S1. Debido a la característica descrita en la tabla 2.1 sobre la interfaz de comunicación, no será estrictamente necesario que la velocidad sea adaptable (ya que se asume que se adaptará en el S1); sin embargo, sí será requisito fundamental que la conexión sea dúplex y que soporte gran cantidad de datos con las menores pérdidas posibles.

Por otra parte, el microcontrolador deberá poder modular señales PWM para controlar los distintos motores de los que dispondrá el brazo. Sin embargo, en caso de que finalmente el *chip* escogido no disponga de dicha modularización, se podrán usar motores los cuales cuenten con un *driver* que permitan controlarlos usando señales digitales y/o analógicas.

Finalmente, a raíz del punto anterior, será imprescindible que el sistema escogido tenga la capacidad de controlar señales digitales y analógicas, permitiendo así mayor versatilidad en el desarrollo del producto final.

Teniendo en cuenta lo anterior, se plantea el uso de los siguientes dispositivos (ver tabla 2.2):

Placa	Ventajas	Desventajas	ID <i>RS-Online</i> y precio
ESP8266	SoC bastante barato (5€) con conexión WiFi y modo de bajo consumo	Señal PWM generada por SW; poca cantidad de GPIO (6).	124-5505 – 19,29€
ESP32	SoC con procesador de dos núcleos que permite comunicaciones WiFi y Bluetooth	No cuenta con GPIO pero permite la comunicación mediante el protocolo I <sup>2</sup> C.	188-5441 – 25,29€
PIC16F18326-I/P	Microcontrolador de 8 bits de baja potencia de consumo y bajo precio con capacidad de modularizar hasta dos señales PWM y con más memoria RAM que otros componentes de su familia. Finalmente, cuenta con bastantes salidas GPIO, suficientes como para añadir más componentes al sistema.	No está integrada en una placa (SoC) por lo que habría que hacer toda la lógica del diseño HW. No dispone de conexiones de red (aunque no son necesarias) y la capacidad de cómputo, en comparación con las otras propuestas, es menor.	124-1554 – 1,375€

Tabla 2.2: Posibles *chips* que se han planteado para el proyecto.

### 2.1.2. Interfaz de usuario

El usuario final del producto solamente interactuara de manera directa con el S1. Para que esta interacción sea posible, se desarrollara un panel de control que permita al usuario definir movimientos que el robot deberá realizar. El panel de control se mostrará en una sola pantalla y permitirá al usuario, mediante una interfaz gráfica sencilla, mover de manera independiente cada uno de los motores del robot, o bien mediante el uso del ratón, describir trayectorias que el robot realizara en tiempo real replicando el movimiento del ratón.

### 2.1.3. Interfaz hardware

El S2 deberá de tener una interfaz con el HW tal que se puedan rotar los motores que controlan el movimiento del robot. Por tanto, el microcontrolador que sea elegido para el proyecto final deberá permitir dicho control en base a las características de los motores. Por otro lado, si los motores devuelven su posición, el microcontrolador deberá ser capaz de recibir este dato para poder enviarlo al S1.

### 2.1.4. Interfaz de comunicaciones

Debido a la naturaleza del proyecto deberá existir una interfaz de comunicaciones que permita el envío y recepción de datos desde el S1 al S2 y viceversa. Según lo indicado en el apartado Descripción General (2), esta comunicación se propone hacer mediante *UART*, *Bluetooth*, *WiFi*, o algún estándar de comunicación que permita una conexión bidireccional simultanea y por tanto permite controlar el robot a la vez que se recibe información referente al estado de este.

Además con este estándar la comunicación es serie y cumple con las limitaciones de diseño que se han impuesto en la recepción de datos en el S2.

### 2.1.5. Memoria

- En cuanto a la memoria de programa se refiere, el microcontrolador del sistema S2 debe tener una memoria suficientemente grande como para poder albergar el SW que recibirá datos del S1 y que actuará sobre los motores en base a estos.
- Por otro lado, la memoria RAM deberá ser tal que permita realizar las operaciones necesarias para ejecutar los movimientos. En caso de que se considere posible la completa implementación de la lógica en el sistema S2, el microcontrolador deberá tener memoria principal suficiente como para conseguir realizar los cálculos matriciales relacionados con los distintos movimientos.

Por otro lado, la memoria principal deberá ser tal que permita las operaciones necesarias para realizar los movimientos en tiempo de ejecución. Cabe destacar que una de las prioridades principales del equipo es ser efectivos en cuanto a la ocupación de memoria del código que desarrolla. Por ello, se buscará reducir el tamaño de este al mínimo manteniendo las funcionalidades necesarias para cumplir los objetivos establecidos.

### 2.1.6. Operaciones

Los usuarios deberán desempeñar acciones tales que generen los movimientos deseados en el brazo. Estas acciones pueden implicar interactuar con los *widgets* presentes en el panel de control o bien efectuar movimientos con el ratón para que el robot los desempeñe directamente.

## 2.2. Funciones del producto

Las funcionalidades principales del brazo robótico han sido descritas de forma introductoria en apartados anteriores de este documento.

En general, existen dos funcionalidades principales que caracterizan tanto al *pArm* como al sistema de control del mismo:

- La funcionalidad principal del brazo robótico S2 es la de realizar movimientos dentro de su campo de movimiento y describir trayectorias previamente planificadas o calculadas en el momento. Mediante este movimiento, se pretende transportar objetos de poco peso. Cabe destacar que el brazo robótico S2 recibe órdenes del S1 y procesa las mismas utilizando el microcontrolador que posee, por lo tanto el computo principal se realiza en S1.
- El sistema de control en del brazo robótico S2 ofrece la funcionalidad principal de planificar trayectorias y controlar el movimiento del brazo. Este sistema se muestra al usuario mediante una interfaz gráfica en S1, la cual permite al usuario controlar el movimiento del brazo mediante la modificación de sus parámetros.

## 2.3. Características del usuario

El sistema de control ejecutado en S1 ofrecerá una interfaz gráfica que permitirá al usuario interactuar con los parámetros del brazo robótico S2 y, por lo tanto, permitirá al mismo controlar el movimiento del robot así como la establecer la descripción de ciertas trayectorias.

Dado que el objetivo del proyecto es ofrecer un sistema didáctico, amigable y fácil de usar, no se imponen requerimientos específicos sobre el usuario en cuanto a conocimientos técnicos sobre programación, HW, electrónica o matemáticos.

El usuario debe estar familiarizado con la interacción y el uso básico de aplicaciones de escritorio para poder interactuar de forma correcta con el sistema de control del brazo.

A pesar de no ser completamente necesario, es recomendable que el usuario esté familiarizado con la estructura física del robot, los movimientos que este puede realizar y los parámetros que se usan para controlar al mismo, ya que de esta forma el control del brazo robótico será más eficaz y seguro.

## 2.4. Restricciones

Dado que actualmente no se presenta ninguna limitación de presupuesto, se busca en el proyecto intentar reducir los costes todo lo posible, para permitir a cualquiera que pueda acceder a los recursos que se necesitan para desarrollar el proyecto. Por ello, se propone usar si es posible el PIC16F18326-I/P y montar la placa con los componentes necesarios.

En cualquier caso, como se ha mencionado anteriormente, es necesario que:

- Se provea de una interfaz para la comunicación que permita comunicarse con el sistema S1 de manera simultánea y con alta capacidad.
- El sistema ha de consumir la menor energía posible, entrando en el modo de *deep-sleep* cuando fuera posible.

- La estructura de S2 ha de ser imprimible en 3D, permitiendo así replicarlo.
- El sistema S1 ha de poder ejecutar aplicaciones Python según lo propuesto anteriormente, en particular, la versión de este superior a la 3.6. En otro caso, el sistema habrá de poder ejecutar la aplicación diseñada sin problemas e indiferentemente del sistema operativo.
- Además, el sistema S1 ha de tener capacidad de cómputo suficiente para realizar los cálculos previstos de forma efectiva. Para ello, se sugiere que el equipo disponga al menos de un procesador con dos núcleos y 512 MB de memoria RAM.
- Todo lo realizado en el proyecto ha de ser OS y OH, permitiendo así que cualquiera pueda acceder y estudiar el proyecto.

## 2.5. Supuestos y dependencias

Indiferentemente de la placa que finalmente se use, el sistema ha de tener tres motores: uno para la base, otro para el primer segmento del brazo robótico y el último para el segundo segmento. Además, para controlar el *end-effector* hará falta una conexión con el extremo del brazo que permita, por ejemplo, añadir un pequeño motor que permita la rotación del mismo (ver el manual de desarrollador de UFACTORY para más información).

Para ello, en la tabla 2.3 se muestran distintas propuestas de motores que podrían ser viables para el proyecto. Intentando cubrir las necesidades, se tienen en cuenta para este proyecto:

- Motor paso a paso: dispositivo electromecánico que convierte una serie de pulsos eléctricos en desplazamientos angulares. Esto permite realizar movimientos muy precisos, los cuales pueden variar de 1,8° hasta 90°. Además, presentan la ventaja de poder quedarse en una posición de forma estática.
- Servomotor: dispositivos de accionamiento para el control de la velocidad, par motor y posición. En su interior suelen tener un decodificador el cual convierte el giro mecánico en pulsos digitales. Además, suelen disponer de un *driver* el cual permite comandar los distintos controles mencionados al principio.

Nombre	Tipo	Características	Código RS y precio
Servomotor Parallax Inc.	Servomotor	<ul style="list-style-type: none"> <li>■ Voltaje entrada: 4 V a 6 V.</li> <li>■ Conector de tres contactos.</li> <li>■ PWM a 50 Hz.</li> </ul>	781-3058 – 16,01 €

Servomotor Faulhaber 9 W	Servomotor	<ul style="list-style-type: none"> <li>■ Par máximo: 9,5 mNm.</li> <li>■ Voltaje entrada: 6 V.</li> <li>■ Potencia nominal: 9 W.</li> <li>■ Conector MOLEX Microfit 3.0.</li> </ul>	184-6932 – 186,73 €
Motor paso a paso bobinado unipolar	Motor paso a paso	<ul style="list-style-type: none"> <li>■ Precisión de 1,8°.</li> <li>■ Par de sujeción: 70 mNm.</li> <li>■ Voltaje entrada: 6 V.</li> <li>■ Conexión de 6 cables.</li> </ul>	440-420 – 30,29 €
Motor paso a paso híbrido	Motor paso a paso	<ul style="list-style-type: none"> <li>■ Precisión de 1,8°.</li> <li>■ Par de sujeción: 1,26 Nm.</li> <li>■ Voltaje entrada: 2,5 V.</li> <li>■ Conexión de 4 cables.</li> </ul>	535-0439 – 108,69 €

Motor paso a paso híbrido	Motor paso a paso	<ul style="list-style-type: none"> <li>■ Precisión de <math>0,9^\circ</math>.</li> <li>■ Par de sujeción: <math>0,44 \text{ Nm}</math>.</li> <li>■ Voltaje entrada: <math>2,8 \text{ V}</math>.</li> <li>■ Conexión de 4 cables.</li> </ul>	535-0401 – 66,72 €
------------------------------------	-------------------------	---	--------------------

Tabla 2.3: Lista de motores propuestos para el sistema S2.

## 2.6. Requisitos pospuestos

En esta sección se describen algunos requisitos del sistema que se postergan a futuras implementaciones o versiones del proyecto.

En el comienzo del proyecto se plantearon algunas funcionalidades y requisitos que, finalmente, se han decidido postergar a futuras implementaciones del proyecto, principalmente debido a su complejidad. En la siguiente se lista se presentan las mas relevantes, las cuales representan posibles mejoras futuras del *pArm*:

- Implementación del sistema de control y planificación de trayectorias en el microcontrolador del *pArm*, de esta forma se busca centralizar el computo en S2.
- Implementación de un sistema de descripción de trayectorias mediante imitación de movimientos realizados por el usuario, es decir, el usuario podría mover físicamente el *pArm* y memorizaría dicha trayectoria para posteriormente describirla.
- Construcción e implementación de diversos tipos de *end-effector* para el *pArm*, los cuales le dotarían de nuevas funcionalidades en cuanto a manejar objetos.
- Implementación de las estructura física del *pArm* utilizando materiales metálicos para mejorar su resistencia y estabilidad. Junto con esta mejora, se podrían utilizar nuevos rotores para dotar al *pArm* de una mayor capacidad de carga.

# Capítulo 3

## Requisitos específicos

### 3.1. Requisitos de la interfaz externa

#### 3.1.1. Interfaz con el usuario

El S1 dispondrá de una interfaz que permitirá al usuario tener un control total sobre los movimientos del robot. Dicha interfaz dispondrá de 6 *sliders*. Tres de ellos que permitirán mover el *end-effector* en coordenadas cartesianas y los tres restantes variarán las coordenadas angulares de los motores generando cambios en la posición del *end-effector*. De esta manera, se consigue mover el *end-effector* con la máxima precisión que ofrecen los motores. Por otro lado, se permitirá al usuario cambiar del modo de funcionamiento en precisión al modo de funcionamiento en movimiento libre mediante un botón presente en la interfaz. Por último, la interfaz dispondrá de un botón que permita interactuar con el *end-effector*.

#### 3.1.2. Interfaz hardware

El S2 esta formado por el brazo robótico *pArm* y el microcontrolador que computa las instrucciones recibidas desde S1. Mediante dicho microcontrolador, S2 interactúa directamente con el HW. El microcontrolador realiza las labores de comunicación con S1, así como las labores de recepción y procesamiento de las instrucciones que controlan el movimiento del *pArm*.

Tras la recepción y procesamiento de las diferentes secuencias de bits, las cuales son instrucciones, el microcontrolador genera señales de salida mediante sus pines, las cuales controlan el movimiento de cada uno de los motores, así como del *end-effector*. Cabe destacar que, en el caso de utilizar motores que proporcionen información sobre su posición angular actual, el microcontrolador debe recibir dicha señal y procesarla, enviando dicha información a S1.

Dependiendo del tipo de motores que se utilicen finalmente, el microcontrolador debe ser capaz de generar señales analógicas PWM, así como señales digitales de control.

### 3.1.3. Interfaz de comunicaciones

Las comunicaciones que se realicen entre el S1 y S2 están planteadas para utilizar UART como método de comunicación. Además, se mencionó como futura implementación poder hacer las comunicaciones entre ambos sistemas utilizando protocolos de red inalámbricos.

No se restringe la velocidad de transmisión (*baud-rate*), ya que se asume que S1 tendrá la posibilidad de adaptar su velocidad. Se escoge el USB como método para intercambiar la información debido a:

- Universalidad: los dispositivos cuentan con al menos una conexión USB.
- Energía: el USB provee 5 V al circuito que se conecta en el otro extremo. Además, la versión 2.0 del estándar, que es lo generalizado en microcontroladores, puede proveer hasta 500 mA al componente conectado.
- Simplicidad: no es necesario entender cómo se conectan los cables sino directamente conectar los extremos.

Para un correcto funcionamiento, la comunicación ha de ser bidireccional, en particular *full duplex*. De esta forma, se podrán recibir y enviar datos simultáneamente, pudiendo así conocer el estado del brazo robótico y actuar en consecuencia en caso de que se encuentre algún tipo de error o problema. Al utilizar el USB como método de comunicación este problema está subsanado, ya que va implícito en la definición del estándar.

### 3.2. Casos de uso

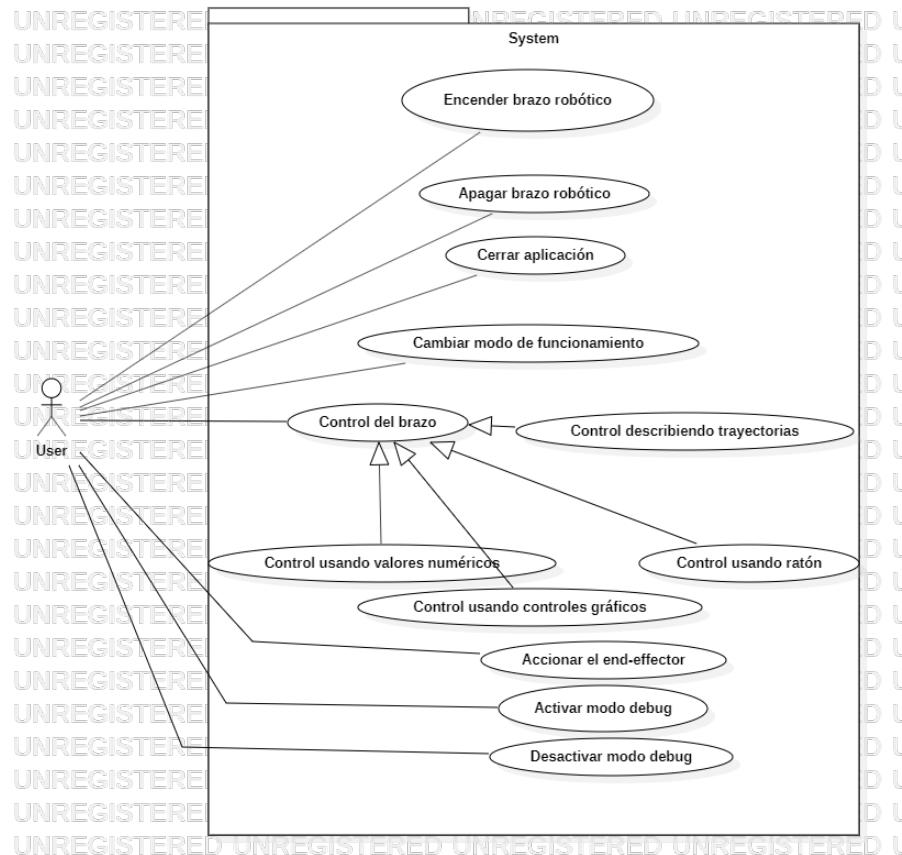


Figura 3.1: Diagrama de casos de uso

0001	Encender brazo robótico (S2)	
<b>Descripción</b>	El usuario deberá ser capaz de encender el sistema del brazo robótico de manera independiente de la aplicación de control.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario interactúa con el sistema para encenderlo.
	2	El sistema comprueba que los componentes están correctamente conectados.
<b>Excepciones</b>	3	Si las comprobaciones son satisfactorias, el sistema continúa con su normal ejecución.
	<b>Paso</b>	<b>Acción</b>
	3	Si las comprobaciones no son satisfactorias el sistema activará un indicador luminoso y se informará del error a S1, si está conectado.
<b>Importancia</b>	1	
<b>Comentarios</b>	Sin comentarios	

Tabla 3.1: Caso de uso 0001 - Encender brazo robótico (S2)

0002	Apagar brazo robótico(S2)	
<b>Descripción</b>	El usuario deberá ser capaz de apagar el brazo robótico de manera independiente a la aplicación de control.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario interactúa con el sistema para apagarlo.
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
		No existen
<b>Importancia</b>	1	
<b>Comentarios</b>	Sin comentarios	

Tabla 3.2: Caso de uso 0002 - Apagar brazo robótico (S2)

0003	Cerrar aplicación	
<b>Descripción</b>	El usuario deberá ser capaz de cerrar la aplicación de control de manera independiente al brazo robótico.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario interactúa con la aplicación para cerrarla
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
	2	Se comprueba que la aplicación se puede cerrar de manera segura.
<b>Importancia</b>	1	
	Sin comentarios	

Tabla 3.3: Caso de uso 0003 - Cerrar aplicación

0004	Cambiar modo de funcionamiento (S1)	
<b>Descripción</b>	El usuario deberá ser capaz de seleccionar el modo de control del brazo robótico, pudiendo escoger entre control mediante ratón o control mediante parámetros.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario interactúa con la aplicación y selecciona el modo de control del robot.
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
		No existen
<b>Importancia</b>	1	
<b>Comentarios</b>	Sin comentarios	

Tabla 3.4: Caso de uso 0004 - Cambiar modo de funcionamiento

<b>0005</b>	Control usando valores numéricos (S1)	
<b>Descripción</b>	El usuario deberá ser capaz de cambiar el valor numérico de cada uno de los parámetros de control del brazo robótico	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario interactúa con la aplicación y cambia el valor de los parámetros de control usando el teclado.
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
	2	Se comprueba si el valor es correcto y se confirma el cambio del valor numérico.
<b>Importancia</b>	1	
<b>Comentarios</b>	Sin comentarios	

Tabla 3.5: Caso de uso 0005 - Control usando valores numéricos (S1)

<b>0006</b>	Control describiendo trayectorias	
<b>Descripción</b>	Se permitirá al usuario escoger una trayectoria predefinida que el brazo robótico podrá realizar.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario selecciona una trayectoria a realizar.
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
		No existen
<b>Importancia</b>	1	
<b>Comentarios</b>	Sin comentarios	

Tabla 3.6: Caso de uso 0006 - Control describiendo trayectorias

<b>0007</b>	Control usando controles gráficos	
<b>Descripción</b>	La interfaz gráfica de la aplicación debe ofrecer control sobre los parámetros del brazo robótico mediante <i>sliders</i> .	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario interactúa con la aplicación y mueve los <i>sliders</i> para variar los parámetros del brazo robótico.
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
	2	Se verifica si se puede realizar dicho movimiento y se ejecuta el cambio en los parámetros.
<b>Importancia</b>	1	
<b>Comentarios</b>	Sin comentarios	

Tabla 3.7: Caso de uso 0007 - Control usando controles gráficos

<b>0008</b>	Control usando ratón	
<b>Descripción</b>	Se permitirá al usuario controlar el brazo robótico de manera directa con el movimiento del ratón	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario mueve el ratón realizando movimientos libres.
	2	Se comprueba que el movimiento no se sale de los margenes permitidos
<b>Excepciones</b>	3	Se realiza el movimiento
	<b>Paso</b>	<b>Acción</b>
<b>Excepciones</b>	2	Si los movimientos se salen de los margenes permitidos no se realizan.
<b>Importancia</b>	1	
<b>Comentarios</b>	Sin comentarios	

Tabla 3.8: Caso de uso 0008 - Control usando ratón

<b>0009</b>	Accionar el <i>end-effector</i>	
<b>Descripción</b>	Se permite al usuario abrir y cerrar la pinza	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario interactúa con la aplicación para abrir y cerrar el <i>end-effector</i>
	2	Se cambia el estado del <i>end-effector</i> según sea necesario.
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
		No existe
<b>Importancia</b>	1	
<b>Comentarios</b>	Sin comentarios	

Tabla 3.9: Caso de uso 0009 - Accionar el *end-effector*

<b>0010</b>	Activar modo debug	
<b>Descripción</b>	Se permite al usuario activar un modo tal que se pueda mandar al S2 el código de control del brazo robótico	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario interactúa con S2 para ponerlo en modo debug.
	2	El sistema comprueba que el cambio de modo se puede hacer de manera segura.
<b>Excepciones</b>	3	El sistema cambia de modo.
	<b>Paso</b>	<b>Acción</b>
<b>Excepciones</b>	2	El sistema detecta que el cambio de modo no se puede hacer de manera segura e impide que este se realice. Se informará del error a S1, si está conectado.
<b>Importancia</b>	1	
<b>Comentarios</b>	Sin comentarios	

Tabla 3.10: Caso de uso 0009 - Accionar el *end-effector*

0011	Desactivar modo debug	
<b>Descripción</b>	Se permite al usuario desactivar el modo debug tal que sea posible emplear el sistema de manera normal	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario interactúa con S2 para desactivar el modo debug.
	2	El sistema comprueba que el cambio de modo se puede hacer de manera segura.
<b>Excepciones</b>	3	El sistema cambia de modo.
	<b>Paso</b>	<b>Acción</b>
<b>Importancia</b>	2	El sistema detecta que el cambio de modo no se puede hacer de manera segura e impide que este se realice. Se informara del error a S1, si esta conectado.
		1
<b>Comentarios</b>	Sin comentarios	

Tabla 3.11: Caso de uso 0009 - Accionar el *end-effector*

### 3.3. Requisitos funcionales

#### 3.3.1. Requisitos *software*

**S1**

##### Mostrar pantalla de control

- ID: 1
- Prioridad: 3.
- Descripción: se muestra una pantalla donde serán situados los *sliders*.
- Entradas: ninguna.
- Salidas: ninguna.
- Errores: no se espera ningún error.

##### Mostrar pantalla informativa

- ID: 2
- Prioridad: 2
- Descripción: se mostrarán distintos datos informativos sobre el estado del sistema S2, en caso de que este se haya conectado correctamente.
- Entradas: los datos recibidos por el sistema S2, si estuviera conectado.

- Salidas: los datos recibidos debidamente interpretados por el sistema.
- Errores: se mostrará un aviso en caso de que el sistema S2 no se detecte o presente algún problema.

### Mostrar botones de edición de pantalla

- ID: 3
- Prioridad: 2
- Descripción: se muestran *widgets* de tipo botones clicables de minimizar, maximizar y cerrar pantalla.
- Entradas: ninguna.
- Salidas: ninguna.
- Errores: en caso de que algún proceso pudiera quedar bloqueado, los *widgets* permitirían el cierre inmediato de la aplicación.

### Interactuar con botones de edición de pantalla

- ID: 4
- Prioridad: 3.
- Descripción: se permite interactuar con los botones que aparecen en la GUI para controlar la pantalla S2.
- Entradas: interacción del usuario con los botones de edición de pantalla.
- Salidas: cambios lógicos tales que se realicen los cometidos de cada botón.
- Errores: no se espera ningún error.

### Mostrar *sliders*

- ID: 5
- Prioridad: 3.
- Descripción: se muestra por pantalla una serie de *widgets* de tipo *sliders*.
- Entradas: coordenadas angulares  $(\theta_1, \theta_2, \theta_3)$  y coordenadas cartesianas  $(X, Y, Z)$ .
- Salidas: mostrar por pantalla de manera gráfica el valor de las variables.
- Errores: no se espera ningún error.

### Editar la posición de los *sliders*

- ID: 6
- Prioridad: 3.
- Descripción: se permite al usuario interactuar de manera directa e independiente con cada uno de los motores del brazo robótico, así como con las componentes de la posición cartesiana del *end-effector*. Este requisito permite al usuario tener una mayor precisión en cuanto a la posición que desea obtener para el brazo robótico.
- Entradas: variación por parte del usuario de la posición de los *sliders*.
- Salidas: modificar el valor numérico de las variables.
- Errores: no se espera ningún error.

### Mostrar botón cambio de modo de funcionamiento del S1

- ID: 7
- Prioridad: 3.
- Descripción: se muestra por pantalla un *widget* de tipo botón clicable que permite cambiar el modo de funcionamiento.
- Entradas: al hacer clic se permite el cambio de modo.
- Salidas: mostrar el nuevo modo de funcionamiento.
- Errores: no se espera ningún error.

### Interactuar con botón de cambio de modo de funcionamiento del S1

- ID: 8
- Descripción: se permite al usuario interactuar con el botón para cambiar el modo de funcionamiento.
- Entradas: interacción del usuario con el botón.
- Salidas: modificación del modo de funcionamiento.
- Errores: no se espera ningún error.

### Mostrar variables

- ID: 9
- Prioridad: 2.
- Descripción: se muestran por pantalla las variables que definen las posiciones cartesianas del *end-effector* y las angulares de los motores.
- Entradas: valores numéricicos de las variables.
- Salidas: mostrar por pantalla dichos valores.
- Errores: no se espera ningún error.

### Mostrar botón de control del *end-effector*

- ID: 10
- Prioridad: 2.
- Descripción: se muestra por pantalla un *widget* de tipo botón clicable que permite cambiar el estado del *end-effector*.
- Entradas: estado del *end-effector*
- Salidas: mostrar el estado de *end-effector*.
- Errores: no se espera ningún error.

### Interactuar con botón de control del *end-effector*

- ID: 11
- Prioridad: 2.
- Descripción: se permite al usuario interactuar con el botón para cambiar el estado de la pinza
- Entradas: interacción con el botón por parte del usuario.
- Salidas: modificar el estado del *end-effector*.
- Errores: no se espera ningún error.

## Editar variables

- ID: 12
- Prioridad: 3.
- Descripción: se permite al usuario editar las variables numéricas de manera directa interactuando con el campo y escribiendo los valores numéricos deseados.
- Entradas: cambio por parte del usuario del valor numérico mostrado.
- Salidas: modificar el valor numérico de la variable.
- Errores: no se espera ningún error.

## Comprobar variables

- ID: 13
- Prioridad: 3.
- Descripción: El sistema, al detectar cambios en alguna de las coordenadas, ya sean cartesianas o angulares se encarga de verificar que dichas coordenadas están en el rango de trabajo del robot. De no ser así se impide el movimiento del brazo para prevenir daños en su estructura o en los motores.
- Entradas: valor de las coordenadas angulares y cartesianas deseadas.
- Salidas: validación de dichas coordenadas.
- Errores: si alguno de los valores introducidos no es válido, se notificará al usuario de dicho error y se evitará que el S2 realice dichos movimientos.

## Comunicación con el sistema S2

- ID: 14
- Prioridad: 1.
- Descripción: utilizando un lenguaje binario, se comunicarán las secuencias de órdenes desde el sistema S1 al sistema S2.
- Entradas: secuencia de movimientos representada como movimientos en puntos cartesianos o como rotaciones de las articulaciones.
- Salidas: secuencia binaria que especifica, en el sistema S2, los movimientos que se han de realizar.
- Errores: como se han comprobado los elementos con anterioridad, no se esperan errores.

## Cálculo de coordenadas articulares

- ID: 15
- Prioridad: 1.
- Descripción: dadas unas coordenadas en forma cartesiana, el sistema S1 debe poder calcular las coordenadas articulares de cada una de las articulaciones del robot.
- Entradas: conjunto de tres puntos cartesianos ( $X, Y, Z$ ).
- Salidas: conjunto de tres puntos articulares ( $\theta_1, \theta_2, \theta_3$ ).
- Errores: dada la configuración geométrica del robot, no se esperan errores.

## Cálculo de coordenadas cartesianas

- ID: 16
- Prioridad: 1.
- Descripción: dadas unas coordenadas articulares, el sistema S1 debe poder obtener las coordenadas cartesianas en las que se encuentra en *end-effector*.
- Entradas: conjunto de tres puntos articulares ( $\theta_1, \theta_2, \theta_3$ ).
- Salidas: conjunto de tres puntos cartesianos ( $X, Y, Z$ ).
- Errores: no se esperan errores.

## Interpretación de los datos

- ID: 17
- Prioridad: 1.
- Descripción: el sistema S1 debe de poder entender e interpretar los datos recibidos desde S2.
- Entradas: cadena binaria con información provista por S2.
- Salidas: mostrar, utilizando la interfaz de usuario, la información pertinente.
- Errores: no se esperan errores.

## Protocolo de intercambio de información

- ID: 18
- Prioridad: 1.
- Descripción: debe existir un protocolo de intercambio de información que defina la longitud, significado y estructura de las instrucciones o secuencias de bits que se transmiten mediante *UART*.
- Entradas: Ninguna.
- Salidas: Ninguna.

## Encendido del sistema

- ID: 19
- Prioridad: 0.
- Descripción: el S1, al encenderse, debe comprobar si está conectado el S2 e inicializar aquellos recursos que serán necesarios.
- Entradas: ninguna.
- Salidas: ninguna.
- Errores: se esperan errores si hubiera algún tipo de corrupción de datos en los ficheros del programa o en los contenedores de datos.

## Apagado del sistema

- ID: 20
- Prioridad: 1.
- Descripción: cuando el usuario cierra la interfaz, el sistema S1 debe desconectarse del todo y cesar cualquier comunicación que pudiera existir con S2. Además, deberá eliminar cualquier tipo de dato residual resultante.
- Entradas: el usuario cierra la aplicación.
- Salidas: ninguna.
- Errores: se esperan errores si no fuese posible cesar la comunicación debido a alguna política del sistema operativo. Se notificará al usuario al respecto.

## S2

### Encendido del sistema

- ID: 21
- Prioridad: 0.
- Descripción: cuando se inicie el sistema HW, debe iniciarse también el SW.
- Entradas: encendido del sistema HW.
- Salidas: activación del sistema SW.
- Errores: no se esperan errores.

### Apagado del sistema

- ID: 22
- Prioridad: 0.
- Descripción: cuando se reciba la orden de apagado desde el S1, el sistema debe cortar toda comunicación con el mismo y apagarse lo antes posible.
- Entradas: orden de apagado desde S1.
- Salidas: ninguna.
- Errores: no se espera ningún error.

### Interpretación de los valores binarios

- ID: 23
- Prioridad: 1.
- Descripción: tras recibir el HW una cantidad de bits que represente el tamaño designado para un determinado comando, los bits se interpretarán y se definirá de qué comando se trata.
- Entradas: bits de control
- Salidas: comando para el sistema físico
- Errores: no se espera ningún error.

## Comprobación de los dispositivos

- ID: 24
- Prioridad: 0.
- Descripción: el sistema deberá comprobar que detecta adecuadamente los dispositivos que están conectados al mismo.
- Entradas: conexiones con cada uno de los dispositivos.
- Salidas: ninguna.
- Errores: si no se detecta algún dispositivo se notificará al sistema S1 sobre dicha falta. Además, se actuará sobre un indicador luminoso para mostrar dicha falla.

## Comunicación con S1

- ID: 25
- Prioridad: 1.
- Descripción: utilizando los protocolos de comunicación, el sistema debe poder comunicarse con S1 correctamente.
- Entradas: valores recibidos por S1.
- Salidas: valores enviados hacia S1.
- Errores: no se esperan errores en la comunicación. En caso de existir, se reenviarían las tramas hasta que se recibieran por el S1.

## Comprobación de la conexión

- ID: 26
- Prioridad: 1.
- Descripción: como el S2 es dependiente del S1, este necesitará comprobar que se encuentra activado para empezar a funcionar.
- Entradas: valor acordado por el sistema S1.
- Salidas: valor acordado con el sistema S2.
- Errores: en caso de no encontrar al sistema S1, el microcontrolador emitiría algún tipo de señal visual o acústica.

### 3.3.2. Requisitos *hardware*

#### Recepción y envío de secuencias de bits entre S1 y S2

- Prioridad: 1.
- Descripción: debe existir un medio de comunicación basado en UART entre S1 y S2 que permita el intercambio de secuencias de bits.
- Entradas: secuencia de bits o instrucción a enviar.
- Salidas: recepción correcta por parte del destinatario.
- Errores: no se espera ningún error.

#### Generación de señales PWM

- Prioridad: 0.
- Descripción: el microcontrolador situado en S2 debe ser capaz de generar señales eléctricas analógicas PWM, las cuales serán usadas para controlar los motores.
- Entradas: instrucciones del sistema de control en S1
- Salidas: señal de control PWM que se corresponde con la respuesta a dicha instrucción.
- Errores: no se espera ningún error.

#### Generación de señales digitales

- Prioridad: 0.
- Descripción: el microcontrolador situado en S2 debe ser capaz de generar señales digitales.
- Entradas: instrucciones del sistema de control en S1.
- Salidas: señal de control digital.
- Errores: no se espera ningún error.

#### Recepción y procesamiento de señales analógicas

- Prioridad: 1.
- Descripción: El microcontrolador debe ser capaz de recibir mediante sus pines y procesar las señales analógicas provenientes de los motores, en caso de que estos informen sobre su posición angular. Estas señales deben ser recibidas y procesadas mediante el *Analog-Digital Conversor* (ADC) para poder ser tratadas a nivel de software.

- Entradas: señal analógica
- Salidas: señal procesada y convertida a datos tratables por el SW.
- Errores: no se espera ningún error.

### Recepción y procesamiento de señales digitales

- Prioridad: 1.
- Descripción: El microcontrolador debe ser capaz de recibir mediante sus pines y procesar las señales digitales. Estas señales deben ser recibidas y tratadas nivel de *software*.
- Entradas: señal digital
- Salidas: ninguna.
- Errores: no se espera ningún error.

### Modo *Deep-Sleep*

- Prioridad: 2.
- Descripción: el microcontrolador debe ser capaz de entrar en modo *Deep-Sleep*
- Entradas: ninguna.
- Salidas: ninguna.
- Errores: no se espera ningún error.

### Encendido

- Prioridad: 0.
- Descripción: se requiere un periodo de inicialización cuando se produce el encendido del sistema. Durante este periodo se realiza la inicialización de la señal de reloj, así como de los periféricos del microcontrolador.
- Entradas: ninguna.
- Salidas: ninguna.
- Errores: no se espera ningún error.

## Apagado

- Prioridad: 0.
- Descripción: S1 puede enviar la señal de apagado del sistema a S2. Cuando esto suceda, el microcontrolador debe de apagarse por completo y debe interrumpir la recepción de instrucciones, así como la generación de señales de control hacia los motores.
- Entradas: instrucción de apagado.
- Salidas: apagado del sistema.
- Errores: no se espera ningún error.

### 3.3.3. Requisitos de rendimiento

Pese a que no se ha restringido en particular, interesa que el sistema propuesto tanto en S1 como en S2 utilice los menos recursos posibles. Por una parte, en S1 la cantidad mínima de RAM que se recomienda es 512 MB, junto con un procesador que permita la ejecución de aplicaciones de forma concurrente.

Además, los cálculos matemáticos, que en principio se harán sobre ese sistema, han de poder ejecutarse, a ser posible, de forma asíncrona y estar optimizados para permitir un cómputo mínimo de un millón de operaciones cada segundo.

S2, por su parte, presenta más limitaciones en lo que a memoria y capacidad de cómputo se refiere. En particular para este proyecto, interesa que S2 bloquee el menor tiempo posible a S1, por lo que se intentará optimizar en tiempo de ejecución intentando además usar la menor cantidad de memoria posible. De esta forma:

1. El uso de la memoria RAM se buscará que sea el menor posible, sin sacrificar en rendimiento.
2. El uso de la memoria *flash* no se buscará reducirlo necesariamente, ya que eso puede afectar directamente al rendimiento.
3. Se trabajará en que el tiempo que el microcontrolador esté haciendo ejecuciones sea el menor posible, permitiendo así un ahorro de energía junto con estar menos tiempo bloqueando al S1.

### 3.3.4. Restricciones del diseño

En esta sección se describen algunas limitaciones existentes debido a distintos motivos, principalmente al HW y estructura física del *pArm*.

En primer lugar, existe una limitación en cuanto a los materiales de fabricación de la estructura física del brazo, ya que se quiere construir íntegramente mediante un material plástico

denominado *PLA*. Este material se utiliza para impresión en 3D y, dado que el *pArm* se quiere imprimir por piezas utilizando una impresora de este tipo, el *PLA* es un material adecuado.

Por otro lado, para simplificar los cálculos en el modelo dinámico, se ha optado por usar un manipulador robótico pantográfico. Este tipo de manipuladores tienen una estructura similar a un flexo y la principal ventaja es que los motores se encuentran muy cercanos a la base. De esta forma, el peso de los mismos no debe ser desplazado al realizar movimientos en las articulaciones del brazo. En el caso de este tipo de brazos, los motores no se encuentran incluidos en los ejes de giro, si no que estos se encuentran en la base del mismo y transmiten su movimiento gracias a la estructura pantográfica formada por una serie de juntas.

### 3.3.5. Atributos del sistema *software* y *hardware*

Tanto para el SW como para el HW, se busca que ambos cumplan las siguientes premisas:

1. El sistema al completo ha de ser fiable. Esto es, no se permitirá al S2 realizar movimientos que puedan perjudicar la estructura del mismo de forma irremediable. A su vez, el sistema S2 deberá tener en cuenta posibles fallos en las órdenes de S1 y comprobar así que la secuencia de órdenes es segura.
2. Relacionado con el punto anterior, también se busca que el sistema sea seguro. En particular, se coordina junto con la fiabilidad para evitar que se puedan hacer movimientos que dañen al robot y, además, se harán diversas comprobaciones relativas al S1, en las cuales se habrá de verificar que está conectado, que es el sistema que dice ser y que se reciben instrucciones coherentes con la programación del sistema.
3. Teniendo en cuenta lo desarrollado en el punto de “Descripción general” (2) y lo mencionado en la “Introducción” (1), es importante que el sistema sea mantenible. Esto se traduce en que, por una parte, se pueda actualizar para corregir problemas que se han encontrado una vez se ha desplegado el sistema; y que la sustitución de piezas o elementos del mismo resulte accesible y barato.
4. Finalmente, dado que el *pArm* está impreso en 3D, se busca que sea portable en lo referente a que pueda ser fácilmente transportado de un lugar a otro. Esto se traducirá en un bajo peso y que el área ocupada por el mismo sea también baja.

## 3.4. Requisitos no funcionales

*Por motivos de tiempo, se dejan los requisitos no funcionales para una futura especificación.*

## Anexo A

### Enlaces útiles

- GitHub de UFACTORY: <https://github.com/uArm-Developer>.
- Web de UFACTORY: <https://www.ufactory.cc/#/en/support/download/pro>.
- Estudio del manipulador  $\mu$ Arm: <https://github.com/UPM-Robotics/uarm>.
- Funcionamiento del  $\mu$ Arm: <https://www.youtube.com/watch?v=VeZ0i11NQRA>.