

**UNIVERSIDAD POLITÉCNICA DE MADRID**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA**  
**AERONÁUTICA Y DEL ESPACIO**  
**GRADO EN INGENIERÍA AEROESPACIAL**

**TRABAJO FIN DE GRADO**

**Diseño, desarrollo e implementación del *Overhead Panel* de  
un simulador de vuelo del Boeing 737NG con Arduino**

**AUTOR: Santiago GIL LÓPEZ DE PABLO**

**ESPECIALIDAD: Navegación y Sistemas Aeroespaciales**

**TUTOR DEL TRABAJO: Tomás MARTÍN DOMINGO**

**Junio de 2019**



## Prólogo

Desde los orígenes de la aviación la humanidad ha presenciado un sinfín de avances, cada cual más complejo que el anterior. La realidad es que es un campo de conocimiento considerablemente reciente, siendo situado por algunos expertos en los inicios del siglo XX. No obstante, aun solo habiendo pasado aproximadamente 120 años, se han podido ver hazañas increíbles, empezando por el primer vuelo de los hermanos Wright, pasando por la llegada del hombre a la Luna, hasta el vuelo inaugural del Airbus A380, el avión comercial más grande del mundo, entre otras.

En toda esta consecución de avances tecnológicos, la aviónica ha jugado un papel esencial. Aunque es cierto que los primeros prototipos de avión se valían exclusivamente del flujo de aire a través de las alas para volar, pronto surgió la necesidad de navegar a través de él. Con tal de cumplir con este fin, en 1914 Lawrence Sperry creó el primer sistema de control basado en estabilización giroscópica. Desde entonces la aviónica no ha dejado de evolucionar, experimentando durante los años 40 un gran avance con el desarrollo de los sistemas electrónicos de radio navegación (ADF, Transponder, control de tráfico aéreo y Radar), usados aun por miles de aviones cada día. Así pues, surgió esta disciplina de la aeronáutica, la cual recibe su nombre de la contracción de las palabras “aviation-electronics”.

Hoy en día, la tendencia de la aviónica embarcada se dirige hacia una sustitución de aquellos componentes analógicos por una integración digitalizada de los mismos. Es en ese profundo cambio en el que la aviación se encuentra ahora y en el que se invierten grandes cantidades de esfuerzo. Si se comparase una cabina antigua con una actual, se podría ver como multitud de relojes analógicos han cedido el paso a amplias pantallas electrónicas (frecuentemente denominadas “displays” por el argot inglés) donde se puede consultar la misma información en un formato más atractivo y cómodo para la tripulación.

En este sinfín de cambios y desarrollos tecnológicos, se ha hecho esencial el uso de simuladores de vuelo para entrenar a las tripulaciones, dado la enorme complejidad que han alcanzado la mayoría de los sistemas embarcados en un avión. En el mundo de la simulación aérea tiene cabida múltiples herramientas profesionales de instrucción, como son los simuladores de vuelo “clase D”, pero también es la alternativa para aquellos aficionados que no pueden permitirse económicamente volar un avión real.

Sin embargo, gracias al desarrollo de hardware y software económico de código abierto, como es por ejemplo el microcontrolador Arduino, muchos usuarios han podido desarrollar sus propios simuladores, alcanzando en muchos de ellos un gran nivel derealismo. Es por ello, que en este proyecto se lleva a cabo el diseño, desarrollo e implementación en un simulador de vuelo de una réplica realista y funcional del *Overhead Panel* del conocido avión comercial Boeing 737NG, utilizando para ello la tecnología que ofrece Arduino a sus usuarios.



## Índice

Prólogo .....	1
Índice de tablas .....	7
Índice de ilustraciones.....	9
Glosario de términos y abreviaturas.....	13
1. Objetivo .....	15
2. Alcance .....	15
3. Consideraciones iniciales .....	17
3.1 Tarjeta microcontroladora Arduino Mega 2560 .....	18
3.2 Simulador de vuelo Microsoft Flight Simulator X.....	19
3.3 Avión de pago PMDG 737NGX .....	20
3.4 Aplicación Cockpit-X.....	21
3.5 Requisitos estéticos, de calidad y realismo.....	22
3.6 Metodología de diseño, desarrollo e implementación .....	23
4. Listado de elementos .....	25
4.1 Kit “Boeing 737 Overhead v3 with Switches” de Cockpit Sim Parts LTD .....	26
4.2 Material electrónico.....	27
4.3 Software de pago .....	28
4.4 Material adicional y utilaje .....	29
5. Diseño del <i>Overhead Panel</i> .....	31
5.1 Nomenclatura en el diseño .....	32
5.1.1 Nomenclatura de los paneles.....	32
5.1.2 Nomenclatura de los circuitos integrados .....	33
5.2 Pruebas previas con Arduino .....	34
5.3 Bastidor de madera.....	36
5.4 Soporte metálico del <i>Overhead Panel</i> .....	39
5.5 Electrónica.....	41
5.5.1 Microcontrolador Arduino Mega 2560 .....	42
5.5.2 Fuente de alimentación.....	43
5.5.3 IC 74HC595 (SR_OUT).....	44
5.5.4 IC 74HC165 (SR_IN) .....	47
5.5.5 Led de 5 mm.....	49
5.5.6 Display LED 7 segmentos (2, 3 y 5 dígitos) .....	51
5.5.7 Servomotor.....	53

5.5.8 Interruptor (2 y 3 posiciones).....	54
5.5.9 Pulsador tipo botón.....	55
5.5.10 Selector rotatorio (45 y 30 grados) .....	56
5.5.11 Codificador rotatorio.....	57
5.5.12 Potenciómetro .....	59
5.5.13 Transistor IRF1404 MOSFET y sistema de retroiluminación .....	61
5.6 Piezas 3D .....	64
5.6.1 Embellecedores de los interruptores ( <i>toggle switch caps</i> ).....	65
5.6.2 Embellecedores de los selectores rotatorios .....	65
5.6.3 Embellecedores de los codificadores rotatorios .....	66
5.6.4 Embellecedores de las luces de aterrizaje .....	67
5.6.5 Estructura y engranajes multiplicadores de los relojes .....	67
5.5.6 Mecanismo de autorrotación de los selectores de encendido de motor .....	69
6. Construcción del <i>Overhead Panel</i> .....	71
6.1 Bastidor de madera .....	72
6.2 Soporte metálico del <i>Overhead Panel</i> .....	77
6.3 Paneles .....	78
6.4 Piezas 3D .....	82
6.5 Electrónica.....	85
6.6 Sistema de retroiluminación .....	100
6.7 Unión de los bastidores de madera .....	102
7. Integración de los elementos del <i>Overhead Panel</i> con Arduino.....	105
7.1 Estructura del código e interacción con Cockpit-X.....	106
7.2 IC 74HC595 (SR_OUT).....	107
7.3 IC 74HC165 (SR_IN) .....	109
7.4 Leds de 5 mm .....	113
7.5 Display LED 7 segmentos.....	115
7.6 Servomotor.....	121
7.7 Interruptor.....	124
7.8 Pulsador tipo botón.....	128
7.9 Selector rotatorio .....	129
7.10 Codificador rotatorio.....	131
7.11 Potenciómetro .....	133
7.12 Transistor IRF1404 MOSFET y sistema de retroiluminación .....	135

8. Planificación temporal.....	137
9. Conclusiones.....	139
10. Referencias.....	141
10.1 Monografías .....	141
10.2 Recursos web .....	141
10.3 Recursos digitales.....	142
11. Anexos .....	143
11.1 Plano acotado del bastidor inferior de madera .....	144
11.2 Plano acotado del bastidor superior de madera.....	145
11.3 Plano acotado del soporte metálico del <i>Overhead Panel</i> .....	146
11.4 Utilización de los pines del Arduino Mega 2560 .....	147
11.5 Esquema eléctrico del funcionamiento del IC 74HC595 .....	149
11.6 Esquema eléctrico del funcionamiento del IC 74HC165 .....	150
11.7 Dimensiones y multiplexado de un display LED 7 segmentos de 2 dígitos.....	151
11.8 Dimensiones y multiplexado de un display LED 7 segmentos de 3 dígitos.....	152
11.9 Dimensiones y multiplexado de un display LED 7 segmentos de 3 dígitos.....	153
11.10 Esquema eléctrico de la instalación de servomotores.....	154
11.11 Esquema eléctrico de la instalación de codificadores rotatorios .....	155
11.12 Esquema eléctrico de la instalación de potenciómetros .....	156
11.13 Esquema eléctrico de la instalación de retroiluminación .....	157
11.14 Asignación de los elementos de salida a los pines del 74HC595 (SR_OUT).....	158
11.15 Asignación de los elementos de entrada a los pines del 74HC165 (SR_IN) .....	161
11.16 Repositorio web GitHub.....	164



## Índice de tablas

Tabla 1. Kit “Boeing 737 Overhead v3 with Switches” de Cockpit Sim Parts LTD .....	26
Tabla 2. Material electrónico comprado.....	27
Tabla 3. Software de pago comprado .....	28
Tabla 4. Material adicional y utilaje comprado.....	29
Tabla 5. Disposición de pines en el IC 74HC595 (Philips 2003) .....	44
Tabla 6. Disposición de pines en el IC 74HC165 (Philips 1990) .....	47
Tabla 7. Perfil usado en CURA para la impresión de PLA con la impresora Creality Ender 3 .....	82
Tabla 8. Código de colores del cableado.....	89
Tabla 9. Código 7 segmentos cátodo común .....	118
Tabla 10. Utilización de los pines del Arduino Mega 2560 .....	147
Tabla 11. Asignación de los elementos de salida a los pines del 74HC595 (SR_OUT) .....	158
Tabla 12. Asignación de los elementos de entrada a los pines del 74HC165 (SR_IN) .....	161



## Índice de ilustraciones

Ilustración 1. Arduino Mega 2560 Rev. 3 (Arduino 2019).....	18
Ilustración 2. Microsoft Flight Simulator Gold Edition (Microsoft 2008).....	19
Ilustración 3. Cabina del PMDG 737NGX (PMDG 2019) .....	20
Ilustración 4. Diagrama de conexión del software utilizado.....	21
Ilustración 5. Kit Boeing 737 Overhead v3 with Switches de Cockpit Sim Parts LTD .....	22
Ilustración 6. Metodología de diseño, desarrollo e implementación .....	23
Ilustración 7. Ejemplo de nomenclatura de los paneles .....	32
Ilustración 8. Nomenclatura de los paneles.....	32
Ilustración 9. Prototipo v1 del PANEL_C5_4 en modo light test.....	34
Ilustración 10. Prototipo v2 del PANEL_C5_4 mostrando una LAND ALT de 100 ft. ....	35
Ilustración 11. Trasera de un Overhead Panel descolgado (Fly737ng, 2017) .....	36
Ilustración 12. Modelo 3D del bastidor inferior de madera .....	37
Ilustración 13. Modelo 3D del bastidor superior de madera .....	37
Ilustración 14. Modelo 3D del bastidor de madera descolgado .....	38
Ilustración 15. Medidas de una cabina real de B737, vista lateral (Markuspilot 2013).....	39
Ilustración 16. Medidas de una cabina real de B737, vista frontal (Markuspilot 2013) .....	39
Ilustración 17. Modelo 3D del soporte metálico del Overhead Panel .....	40
Ilustración 18. Disposición de los pines del Arduino Mega 2560 (The Engineering Projects 2018) .....	42
Ilustración 19. Especificaciones técnicas de la PSU modelo FSP350-60EMDN .....	43
Ilustración 20. Esquema del IC 74HC595 (Philips 2003).....	44
Ilustración 21. Representación del funcionamiento interno del IC 74HC595 (Last Minute Engineers 2019).....	45
Ilustración 22. Esquema simplificado del funcionamiento del IC 74HC595.....	46
Ilustración 23. Esquema del IC 74HC165 (Philips 1990).....	47
Ilustración 24. Esquema simplificado del funcionamiento del IC 74HC165.....	48
Ilustración 25. Símbolo y polaridad del diodo LED (Ebotics 2019).....	49
Ilustración 26. Anunciadores del Overhead Panel .....	50
Ilustración 27. Conexión en paralelo de los leds de los anunciantes.....	50
Ilustración 28. Nomenclatura del display LED 7 segmentos (Electrontools 2016) .....	51
Ilustración 29. Esquema simplificado de la instalación de servomotores .....	53
Ilustración 30. Interruptor de 3 posiciones tipo ON/MOM - OFF - ON/MOM (Cockpit Sim Parts LTD 2017) .....	54
Ilustración 31. Pulsadores MOM de tipo botón de color negro (Cockpit Sim Parts LTD 2017) ..	55
Ilustración 32. Selector rotatorio de 45 grados (Cockpit Sim Parts 2017) .....	56
Ilustración 33. Esquema eléctrico de un selector rotatorio de 45 grados con tres posiciones ..	56
Ilustración 34. Codificadores rotatorios (Cockpit Sim Parts 2017) .....	57
Ilustración 35. Funcionamiento interno del codificador rotatorio (How to mechatronics 2016) .....	57
Ilustración 36. Señal emitida por el codificador rotatorio en función del sentido de giro (How to mechatronics 2016).....	58
Ilustración 37. Esquema simplificado de la instalación de codificadores rotatorios .....	58
Ilustración 38. Potenciómetro lineal 10 kΩ marca ALPHA (Cockpit Sim Parts 2017).....	59
Ilustración 39. Esquema del interior de un potenciómetro (Build electronic circuits 2016).....	59

Ilustración 40. Esquema simplificado de la instalación de potenciómetros.....	60
Ilustración 41. Esquema eléctrico de la configuración paralelo de la instalación de retroiluminación.....	61
Ilustración 42. Transistor IRF1404 MOSFET canal-N y disipador de calor .....	62
Ilustración 43. Esquema simplificado de la instalación de retroiluminación.....	63
Ilustración 44. Modelo 3D del embellecedor de interruptores .....	65
Ilustración 45. Modelo 3D del embellecedor de selectores rotatorios .....	65
Ilustración 46. Modelo 3D del embellecedor del selector giratorio cross feed.....	66
Ilustración 47. Modelo 3D de los embellecedores de los codificadores rotatorios.....	66
Ilustración 48. Modelo 3D del embellecedor de las luces de aterrizaje .....	67
Ilustración 49. Modelo 3D de los engranajes multiplicadores de los relojes.....	68
Ilustración 50. Modelo 3D de la estructura de los relojes .....	68
Ilustración 51. Modelo 3D de la estructura del reloj manual valve .....	69
Ilustración 52. Modelo 3D del mecanismo de autorrotación de los selectores de encendido de motor.....	69
Ilustración 53. Modelo 3D de los selectores de encendido de motores en posición GRD .....	70
Ilustración 54. Modelo 3D de los selectores de encendido de motores en posición OFF .....	70
Ilustración 55. Regruesadora de carpintería cepillando los tablones de ramín.....	72
Ilustración 56. Tablones del bastidor inferior cortados según medidas tomadas .....	72
Ilustración 57. Bastidor inferior durante el endurecimiento de la cola blanca .....	73
Ilustración 58. Bastidor superior de madera encolado .....	73
Ilustración 59. Bastidor inferior de madera encolado .....	74
Ilustración 60. Unión de ambas partes del bastidor de madera.....	74
Ilustración 61. Capa inferior de paneles superpuestos sobre el bastidor de madera .....	75
Ilustración 62. Imprimación del bastidor de madera previa al lacado.....	75
Ilustración 63. Configuración previa de la fresadora por control numérico.....	76
Ilustración 64. Aspecto del bastidor inferior después de la primera pasada de la fresadora ....	76
Ilustración 65. Bastidor inferior unido al soporte metálico del Overhead Panel.....	77
Ilustración 66. Bastidor superior perforado .....	78
Ilustración 67. Labrado de la primera columna del Overhead Panel.....	78
Ilustración 68. Capa superior del PANEL_C1_4 colocada.....	79
Ilustración 69. Componentes y primera capa de paneles colocados.....	79
Ilustración 70. Delantera del PANEL_C2_1 .....	80
Ilustración 71. Trasera del PANEL_C2_1 .....	80
Ilustración 72. Delantera del Overhead Panel montada.....	81
Ilustración 73. Resultado de la impresión 3D de los engranajes de los relojes .....	83
Ilustración 74. Resultado de la impresión 3D de las estructuras de los relojes.....	83
Ilustración 75. Resultado de la impresión 3D de los embellecedores de los selectores rotatorios .....	84
Ilustración 76. Resultado de la impresión 3D de los embellecedores de los codificadores y las luces de aterrizaje .....	84
Ilustración 77. Embellecedores de las luces de aterrizaje pintados .....	84
Ilustración 78. Instalación de los leds en las cajas de los anunciantes .....	85
Ilustración 79. Placas con las resistencias asociadas a los leds de 5 mm.....	85
Ilustración 80. Primera versión de las placas SR_OUT, parte delantera .....	86
Ilustración 81. Primera versión de las placas SR_OUT, parte trasera .....	86
Ilustración 82. Segunda versión de las placas SR_OUT .....	87

Ilustración 83. Placas SR_IN interconectadas, parte delantera .....	87
Ilustración 84. Placas SR_IN interconectadas, parte trasera .....	88
Ilustración 85. Sistema de sujeción de las placas al bastidor superior de madera.....	88
Ilustración 86. Instalación de los mazos de cables de la primera placa SR_OUT.....	89
Ilustración 87. Mazos de cables de la C1.....	89
Ilustración 88. Instalación del Arduino Mega 2560 y la PSU.....	90
Ilustración 89. Displays LED 7 segmentos del PANEL_C2_1 y su placa SR_OUT .....	90
Ilustración 90. Cubierta de los displays LED del PANEL_C2_1 instalada.....	91
Ilustración 91. Prueba de los displays LED del PANEL_C2_1.....	91
Ilustración 92. Pegamento termofusible negro para sellar fugas de luz de la retroiluminación	92
Ilustración 93. Instalación de las bisagras en el bastidor superior de madera .....	92
Ilustración 94. Prueba del sistema de unión entre bastidores .....	93
Ilustración 95. Placas y mazos de cables de las C2 y C3 instalados .....	93
Ilustración 96. Displays LED del PANEL_C5_4 instalados.....	94
Ilustración 97. Placas y mazos de cables de las C4 y C5 instalados .....	94
Ilustración 98. Montaje de los mecanismos de los relojes .....	95
Ilustración 99. Vista interna del mecanismo de los relojes.....	95
Ilustración 100. Instalación de los relojes del PANEL_C4_6, vista trasera.....	96
Ilustración 101. Instalación de los relojes del PANEL_C4_6, vista delantera.....	96
Ilustración 102. Instalación del mecanismo de autorrotación de los selectores del PANEL_C1_4 .....	97
Ilustración 103. Overhead Panel acabado en modo light test, vista delantera .....	98
Ilustración 104. Overhead Panel terminado en modo light test, vista trasera .....	99
Ilustración 105. Prueba de encendido del sistema de retroiluminación .....	100
Ilustración 106. Prueba del sistema de retroiluminación a través de los paneles.....	100
Ilustración 107. Instalación del sistema de regulación de intensidad lumínica.....	101
Ilustración 108. Unión de la trasera al bastidor inferior de madera.....	101
Ilustración 109. Unión de los bastidores de madera mediante las bisagras.....	102
Ilustración 110. Instalación de la PSU en la trasera del panel .....	103
Ilustración 111. Resultado final del Overhead Panel visto desde el ángulo del piloto .....	103
Ilustración 112. Overhead Panel en modo light test en un entorno oscuro.....	104
Ilustración 113. Diagrama Gantt del proyecto .....	138
Ilustración 114. Plano acotado del bastidor inferior de madera .....	144
Ilustración 115. Plano acotado del bastidor superior de madera.....	145
Ilustración 116. Plano acotado del soporte metálico del Overhead Panel.....	146
Ilustración 117. Esquema eléctrico del funcionamiento del IC 74HC595 .....	149
Ilustración 118. Esquema eléctrico del funcionamiento del IC 74HC165 .....	150
Ilustración 119. Dimensiones y multiplexado de un display LED 7 segmentos de 2 dígitos (Oasistek 2006).....	151
Ilustración 120. Dimensiones y multiplexado de un display LED 7 segmentos de 3 dígitos (Peter parts 2019) .....	152
Ilustración 121. Dimensiones y multiplexado de un display LED 7 segmentos de 5 dígitos (CL Lighting Co 2006).....	153
Ilustración 122. Esquema eléctrico de la instalación de servomotores.....	154
Ilustración 123. Esquema eléctrico de la instalación de codificadores rotatorios.....	155
Ilustración 124. Esquema eléctrico de la instalación de potenciómetros .....	156
Ilustración 125. Esquema eléctrico de la instalación de retroiluminación .....	157



## Glosario de términos, abreviaturas y acrónimos

- **ABS:** Acrilonitrilo Butadieno Estireno (*Acrylonitrile Butadiene Styrene*).
- **AC:** Corriente Alterna (*Alternating Current*).
- **ADF:** Localizador Automático de Dirección (*Automatic Direction Finder*).
- **ALT:** Altitud (*Altitude*).
- **Amperio (A):** Unidad de medida de la corriente eléctrica en el Sistema Internacional de Unidades.
- **Analógico:** Que presenta información, especialmente una medida, mediante una magnitud física continua proporcional al valor de dicha información.
- **Aviónica:** Electrónica utilizada en los vehículos aeroespaciales. Incluye los sistemas de comunicaciones, los sistemas de navegación, el sistema eléctrico, los sistemas de presentación de información y los diversos sistemas de control.
- **Back-end:** En diseño de software, parte que procesa la entrada desde el *front-end*.
- **CAD:** Diseño Asistido por Ordenador (*Computer-Aided Design*).
- **CONT:** Continuo (*Continuous*).
- **DC:** Corriente Continua (*Direct Current*).
- **Digital:** Que crea, presenta, transporta o almacena información mediante la combinación de bits, es decir, de una manera discretizada.
- **Display:** Término anglosajón para referirse a una pantalla.
- **DP:** Punto Decimal (*Decimal Point*).
- **Faradio (F):** Unidad de medida de la capacidad eléctrica en el Sistema Internacional de Unidades.
- **FLT:** Vuelo (*Flight*).
- **Front-end:** En diseño de software, parte del software que interactúa con los usuarios.
- **GRD o GND:** Tierra (*Ground*).
- **HYD:** Hidráulicos (*Hydraulics*).
- **IC:** Circuito Integrado (*Integrated Circuit*).
- **IDE:** Entorno de Desarrollo Integrado (*Integrated Development Environment*).
- **LED:** Diodo Emisor de Luz (*Light-Emitting Diode*).
- **LTD:** Limitada (*Limited*).
- **Microcontrolador:** Circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria.
- **MOSFET:** Transistor de Efecto de Campo Metal-Óxido-Semiconductor (*Metal-oxide-semiconductor Field-effect transistor*).
- **Navegación:** En Ingeniería Aeroespacial se define como el campo de estudio que pretende la realización segura de los desplazamientos de los vehículos aeroespaciales.
- **NG:** Nueva Generación (*Next Generation*).
- **Ohmio ( $\Omega$ ):** Unidad de medida de la resistencia eléctrica en el Sistema Internacional de Unidades.
- **Payware:** Software de pago (*Payment software*).
- **Pb:** Plomo, elemento químico de la tabla periódica.

- **PCB:** Placa de Circuito Impreso (*Printed Circuit Board*).
- **PCS:** Piezas (*Pieces*).
- **PETG:** PolyEthylen Terephthalato de Glicol (*Polyethylene Terephthalate Glycol*).
- **PLA:** Ácido poliláctico (*Polylactic Acid*).
- **PMDG:** Grupo de Desarrollo de Manuales de Precisión (*Precision Manuals Development Group*).
- **Protoboard:** Placa de prototipos (*Prototype board*)
- **PSU:** Fuente de alimentación (*Power Supply Unit*).
- **PWM:** Modulación por Ancho de Pulso (*Pulse-Width Modulation*).
- **PWR:** Potencia (*Power*).
- **Sn:** Estaño, elemento químico de la tabla periódica.
- **SR:** Registro de desplazamiento (*Shift Register*).
- **STL:** (*Standard Triangle Language*) Formato de archivo informático de diseño asistido por computadora (CAD) que define geometría de objetos 3D.
- **TFG:** Trabajo Fin de Grado.
- **Transponder:** Dispositivo utilizado en el ámbito de las telecomunicaciones. Se sus funciones se basan en la recepción de mensajes y la generación y transmisión automática de mensajes de respuesta.
- **UPM:** Universidad Politécnica de Madrid.
- **USB:** Bus Universal Serie (*Universal Serial Bus*).
- **Voltio (V):** Unidad de medida de la diferencia de potencial eléctrico entre dos puntos en el Sistema Internacional de Unidades.

## 1. Objetivo

El presente proyecto tiene como objetivo principal concluir con la construcción de una réplica realista y funcional del *Overhead Panel* correspondiente al avión Boeing 737NG, que será implementada en el simulador de vuelo Microsoft Flight Simulator X para su posterior uso, como nuevo elemento, en un simulador de bajo coste que se lleva desarrollando durante varios años.

De forma adicional al objetivo principal existen dos objetivos secundarios. El primero reside en la importancia de conseguir un bajo coste, permitiendo así que cualquier aficionado a la simulación pueda repetir el proyecto sin necesidad de adquirir un *Overhead Panel* comercial, cuyo coste es muy superior al que se pretende conseguir en este proyecto. Por otro lado, se tiene la intención de mantener el mayor nivel derealismo posible. Por ello, el *Overhead Panel* deberá estar diseñado de tal manera que pueda abrirse una vez instalado, con el fin de poder realizar el mantenimiento oportuno, tal y como se hace en los aviones reales.

## 2. Alcance

Para la consecución de los objetivos de este proyecto se desarrollarán diferentes fases. En una primera fase se realizará el diseño electrónico y de software necesario para hacer el *Overhead Panel* compatible con el microcontrolador Arduino. Durante esta fase del proyecto se hace necesario hacer una planificación inicial de aquellos componentes y elementos que deben ser comprados, y por su condición de bajo coste, exportados desde otros países. A continuación, se realizan los prototipos para verificar la viabilidad del proyecto entero, tanto a nivel de hardware como de software. Una vez se ha comprobado, se continúa con la creación del código que hará funcionar al panel, escrito en lenguaje C++ en el entorno de desarrollo integrado *Arduino IDE*, dividiéndose este, para la facilidad de su desarrollo, en el código de cada subpanel. Finalmente, el código se integra en uno solo y se realizan las correcciones de errores pertinentes, así como su optimización para su funcionamiento adecuado.

Ya es en una segunda fase, aunque simultánea con la primera, donde se procede a la construcción en tamaño real del *Overhead Panel*. Primero se crea el marco de madera y la estructura metálica que permitirán colocar al panel a una distancia y ángulo similar al que se puede encontrar en una cabina real. Realizado esto, se procede a instalar todos los subpaneles comprados, así como sus interruptores y luces LED. Por último, se realiza la instalación eléctrica y electrónica previamente diseñada, comprobando de nuevo su buen funcionamiento.

A lo largo del presente informe se mostrarán más en detalle las dos fases anteriores, además de cómo se han ido solventando los problemas que han surgido durante la consecución de estas.



### 3. Consideraciones iniciales

En este capítulo son expuestos aquellos planteamientos e ideas en los que se apoya el proyecto o aquellas restricciones impuestas desde el inicio. Es importante definirlas en las primeras etapas del proyecto ya que marcará su desarrollo posterior pues, una vez acabado, este debe satisfacer todas ellas. De esta manera, el capítulo se subdivide en las siguientes secciones:

1. Tarjeta microcontroladora Arduino Mega 2560
2. Simulador de vuelo Microsoft Flight Simulator X
3. Avión de pago PMDG 737NGX
4. Aplicación Cockpit-X
5. Requisitos estéticos, de calidad y realismo
6. Metodología de diseño, desarrollo e implementación

### 3.1 Tarjeta microcontroladora Arduino Mega 2560

Una de las restricciones principales del proyecto es su diseño para posterior integración con la tecnología de Arduino. Sus creadores definen su hardware de la siguiente manera:

*"Arduino es una plataforma electrónica de código abierto basada en hardware y software fácil de usar. Las placas Arduino son capaces de leer entradas - luz en un sensor, un dedo en un botón, o un mensaje de Twitter - y convertirlo en una salida - activando un motor, encendiendo un led, publicando algo en línea. Puedes decirle a tu placa qué hacer enviando un conjunto de instrucciones al microcontrolador de la placa. Para ello utilizas el lenguaje de programación Arduino (basado en Wiring), y el Software Arduino (IDE), basado en Processing."*

(Arduino 2019)

Esa facilidad de programación de entradas y salidas hace que resulte sumamente interesante de considerar a la hora de realizar este proyecto, dado que, en términos generales, eso es precisamente de lo que se compone un *Overhead Panel*, entendiendo por entradas interruptores, selectores, botones, potenciómetros y codificadores rotatorios; y por salidas leds de 5 mm, leds 7 segmentos y servomotores.

Sin embargo, el Arduino Mega 2560, el más grande de toda la gama Arduino y el utilizado en este proyecto, solo dispone de 54 pines digitales de entrada / salida (de los cuales 15 se pueden utilizar como salidas PWM) y 16 entradas analógicas. Este número resulta insuficiente para el desarrollo del proyecto, pues de partida ya solo se necesitaría al menos 173 pines de salida para conectar todos los leds del panel. Aunque la solución parecería consistir en añadir más de una placa Arduino, en los siguientes capítulos se detallará la solución adoptada para utilizar un solo Arduino Mega 2560 en todo el proyecto.

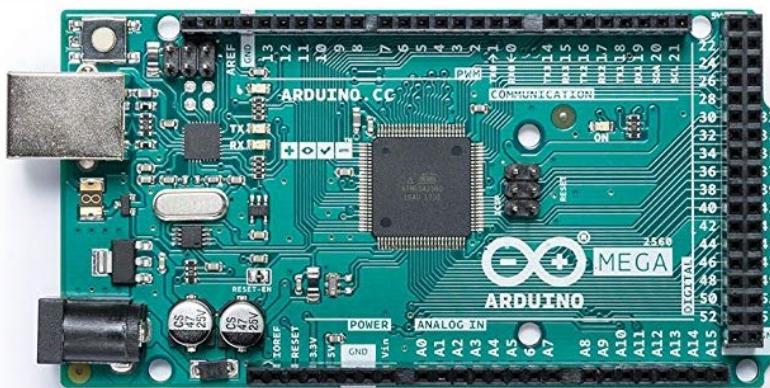


Ilustración 1. Arduino Mega 2560 Rev. 3 (Arduino 2019)

### 3.2 Simulador de vuelo Microsoft Flight Simulator X

Cuando se inició la construcción del simulador casero que pretende continuar el presente proyecto se tuvo que tomar la decisión de que software elegir. Por aquellos años en el mercado estaban disponibles Microsoft Flight Simulator 2004, Microsoft Flight Simulator X y X-Plane 7. Aún no había sido desarrollado Prepar3D. Dado que Microsoft Flight Simulator X ofrecía una gran cantidad de complementos o *add-ons*, entre los que se encontraba el avión de pago Boeing 737NG del desarrollador PMDG, fue el simulador elegido.

A pesar de funcionar de manera correcta con la configuración original, el simulador fue mejorado durante varios meses con diferentes herramientas para funcionar de manera más optimizada con un sistema de triple pantalla y hardware de Saitek. Por ello, aunque a fecha actual existen nuevos simuladores que han superado el rendimiento que puede ofrecer Microsoft Flight Simulator X, se ha decidido continuar con este simulador para aprovechar todo el desarrollo previamente realizado. No obstante, se deja para futuras fases del proyecto, fuera ya de este trabajo fin de grado, la integración con otros simuladores de vuelo más recientes.

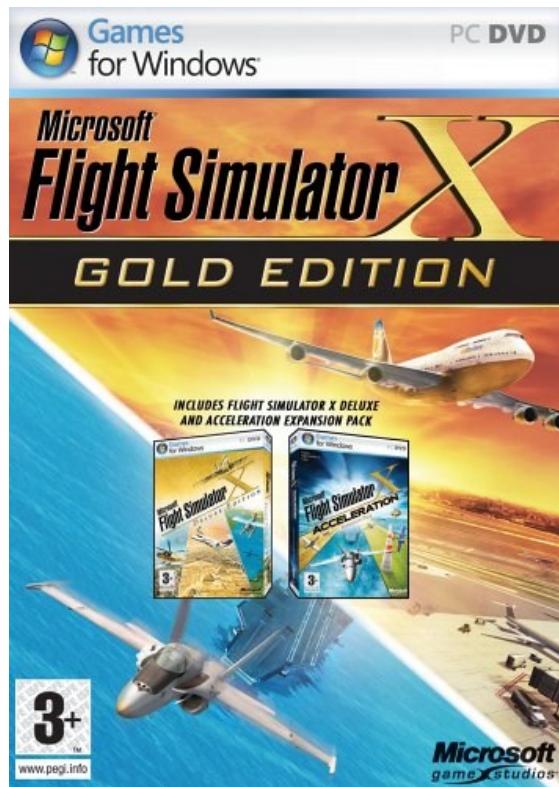


Ilustración 2. Microsoft Flight Simulator Gold Edition (Microsoft 2008)

### 3.3 Avión de pago PMDG 737NGX

Como se ha comentado previamente, tras probar varios desarrolladores, se decidió apostar por el software de pago o *payware* PMDG 737NGX. Este paquete incluye los modelos 800 y 900 del famoso avión de compañía Boeing 737NG, siendo completamente compatible con el simulador de vuelo Microsoft Flight Simulator X.

*"El PMDG 737NGX ofrece un nivel sin precedentes de fidelidad de los sistemas. Desarrollado a lo largo de tres años con la ayuda técnica de Boeing y un equipo de asesores de mantenimiento y tripulación del 737NG de la vida real, hemos modelado minuciosamente casi todos los sistemas de la aeronave real de una manera totalmente dinámica y realista."*

(PMDG 2019)

Ese grado derealismo a nivel de sistemas hace que merezca la pena sustituir el Boeing 737NG que trae por defecto Microsoft Flight Simulator X por este otro, en el que hasta el último interruptor y funcionalidad del *Overhead Panel* está implementado.

Adicionalmente el paquete PMDG 737NGX incluye un software denominado *PMDG SDK* (*Software Development Kit*), que permite a los usuarios desarrollar y conectar este avión con aplicaciones de terceros, pudiendo crear así un entorno real que simule una cabina de vuelo asociando cada elemento de esta con el correspondiente en la cabina del PMDG 737NGX. Gracias al *PMDG SDK* es posible la creación de aplicaciones como Cockpit-X, la cual es utilizada en este proyecto y será comentada en la siguiente sección.



Ilustración 3. Cabina del PMDG 737NGX (PMDG 2019)

### 3.4 Aplicación Cockpit-X

Cockpit-X es una aplicación gratuita desarrollada por H.M. Nayanakantha Kumara Nawarathne, estudiante de la Universidad de Bolton en Inglaterra. Este software proporciona una interfaz fácil de usar para los constructores de simuladores caseros. Permite leer el estado de la mayoría de los elementos de la cabina del PMDG 737NGX e interactuar con ellos a través de un lenguaje sencillo que será comentado más adelante. Además, está diseñado para trabajar con Arduino, permitiendo conectar hasta un total de 10 placas de forma simultánea. Esto aumenta enormemente las posibilidades, haciéndola muy interesante para este proyecto.

Al no ser una aplicación desarrollada por una compañía, sino por una persona de manera desinteresada, se ha ido mejorando progresivamente a lo largo de los meses que se ha realizado este proyecto, gracias al *feedback* que se le ha podido ofrecer a su creador. Así pues, es su última versión, *Cockpit-X v1.5*, la que se utiliza tras haber corregido todos los fallos de su versión anterior, la *v1.4*.

Cockpit-X, hace de intermediario entre la placa Arduino y PMDG 737NGX, dentro de Microsoft Flight Simulator X. Este software está diseñado para operar con FSUIPC (en su versión *v4.939n*), el programa que comunica los periféricos externos con el simulador, siendo el eslabón que cierra esta cadena de programas conectados en serie. El siguiente esquema aclara esta conexión:

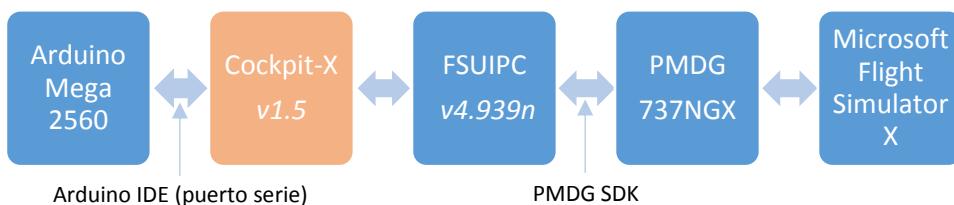


Ilustración 4. Diagrama de conexión del software utilizado

Cockpit-X asigna a cada elemento de la cabina un código de dos letras y una variable numérica. Las letras identifican dicho elemento y el número su estado. Así por ejemplo el código “*BU0*” nos indicaría que el anunciador *STANDBY HYD LOW QUANTITY* estaría apagado, mientras que si recibiéramos por el monitor serial el código “*BU1*” veríamos que ese anunciador está encendido en el *Overhead Panel* del simulador. De la misma manera funcionan los interruptores, siendo este caso el número la posición en la que se encuentra.

De esta manera, a la hora de confeccionar el código del Arduino, será necesario leer los códigos enviados por el programa y gestionar las salidas de acuerdo con ellos. Por otro lado, cuando se actúa sobre una entrada, habrá que escribir en el monitor serial el código correspondiente para que dicha acción se reproduzca en el avión del simulador. Este concepto es, sin duda, la base para la confección del código del Arduino, el cual se expone en el anexo, sección 11.16 de este informe.

En el capítulo 10 “Referencias” se podrá consultar el enlace para descargar Cockpit-X en su última versión disponible.

### 3.5 Requisitos estéticos, de calidad y realismo

Como se comentó previamente en este informe, uno de los aspectos principales del proyecto es el económico, intentando rebajar todo lo posible el coste final del mismo. No obstante, también se pretende ofrecer un producto final con un grado de realismo similar al que se podría encontrar en un *Overhead Panel* comercial. Esto obliga a alcanzar una solución de compromiso entre ambos factores, de manera que se obtenga la mejor calidad y estética posible al precio más bajo.

Es por esto por lo que, a pesar de que se podría haber impreso un poster en tamaño real del *Overhead Panel* reduciendo así más aun su coste, se decidió que esto no cumpliría con el requisito de realismo deseado. De esta manera, se apostó por adquirir un kit en el que se incluyesen todos los paneles realistas del *Overhead* y que además fueran de un material translúcido que permitiese el paso de luz para realizar retroiluminación en los mismos.

Adicionalmente es necesario mantener un nivel alto de calidad tanto en los anunciantes lumínicos, como en los interruptores, consiguiendo que estos tuvieran un aspecto y dureza similar a los reales. Por todo ello, finalmente se tomó la decisión de adquirir el kit “Boeing 737 Overhead v3 with Switches” del fabricante inglés Cockpit Sim Parts LTD. Este es el más económico encontrado en la fecha de realización de este proyecto, entre los que mantienen un realismo a la altura de lo exigido. Este kit, además de incluir los paneles anteriormente mencionados, también incluye las cajas y placas translúcidas de los anunciantes, los leds, los interruptores (con sus guardas), los potenciómetros, los selectores y los codificadores rotatorios. La parte negativa de este fabricante es que no tiene stock, produciendo todo bajo demanda. Por ello los plazos de entrega son extensos (en esta ocasión concretamente de 4 meses), haciendo necesaria una planificación previa muy temprana para asegurar que estuviesen en fecha para la realización de este trabajo fin de grado.

Para abaratar costes no se compró de este fabricante todas las piezas de plástico del panel. Se diseñaron y se realizaron por impresión 3D posteriormente, tal y como será detallado en próximos capítulos de este informe. Tampoco se adquirió la solución de electrónica que ofrecía Cockpit Sim Parts LTD, pasándose a realizar con la tecnología Arduino ya mencionada.



Ilustración 5. Kit Boeing 737 Overhead v3 with Switches de Cockpit Sim Parts LTD

### 3.6 Metodología de diseño, desarrollo e implementación

Como metodología de trabajo para este proyecto se ha elegido aquella que está basada en requisitos. Este tipo de enfoque permite, a la hora de diseñar, desglosar el proyecto en partes más pequeñas, las cuales deben cumplir con una serie de requisitos impuestos de forma anticipada, muchos de ellos ya vistos a lo largo de este capítulo. Una vez se desarrolla y se implementa esa parte del proyecto se comprueba, antes de pasar a la siguiente, si cumple con esos requerimientos impuestos.

De esta manera, la fase de diseño del proyecto sigue una aproximación “*top-down*” en requisitos, comenzando con unas directrices de alto nivel, basadas en disposiciones de tipo funcional, para dar paso posteriormente a requerimientos de bajo nivel, con un enfoque más detallado que las anteriores.

Por otro lado, las fases de construcción e implementación siguen una estructura “*bottom-up*”, donde, a partir del diseño definitivo obtenido, se comienza a crear e integrar cada subpanel verificando que van cumpliendo con los parámetros de diseño.

Esta metodología queda clarificada por el siguiente esquema donde se exponen las aproximaciones “*top-down*” y “*bottom-up*” de manera gráfica, resaltando la importancia que tienen los bucles de verificaciones entre ellas.

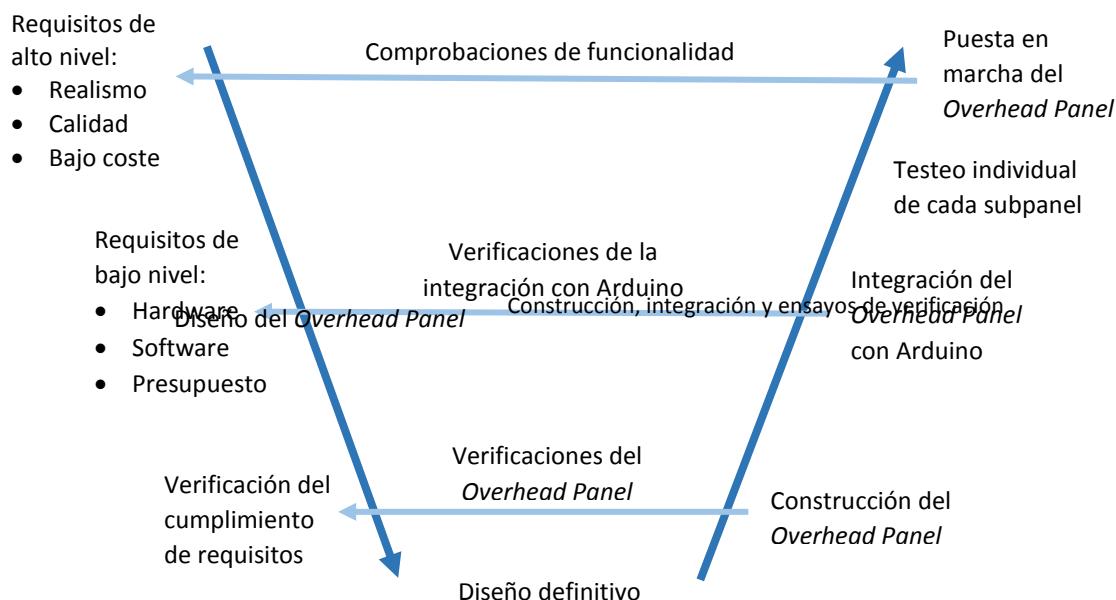


Ilustración 6. Metodología de diseño, desarrollo e implementación



## 4. Listado de elementos

En este capítulo son expuestos los listados de componentes que han sido necesarios para la realización del proyecto. Los precios mostrados son orientativos ya que podrían estar sujetos a cambios, siendo mostrados los que se obtuvieron en el momento de su compra.

Para facilitar la clasificación de estos el capítulo se subdivide en las siguientes secciones:

1. Kit “*Boeing 737 Overhead v3 with Switches*” de Cockpit Sim Parts LTD
2. Material electrónico
3. Software de pago
4. Material adicional y utillaje

#### 4.1 Kit “Boeing 737 Overhead v3 with Switches” de Cockpit Sim Parts LTD

El kit “*Boeing 737 Overhead v3 with Switches*” de Cockpit Sim Parts LTD, al ser un fabricante inglés, tiene un precio de £499, que al cambio en el momento de la compra resultaron ser 634,81€. Como se ha comentado previamente, este kit permite obtener un *front-end* realista y con un gran acabado. En la siguiente tabla se muestran los elementos contenidos en el mismo:

Tabla 1. Kit “Boeing 737 Overhead v3 with Switches” de Cockpit Sim Parts LTD

CONCEPTO	DISTRIBUIDOR	CANTIDAD	PRECIO UD.	TOTAL
Full Boeing 737 Overhead Gauges set	CockpitSimParts	1	£39,99	£ 39,99
Lights panel set with backing panel	CockpitSimParts	1	£25,00	£25,00
Central panel with backing panel	CockpitSimParts	1	£29,00	£29,00
Navigation panel with backing panel	CockpitSimParts	1	£24,99	£24,99
HYD pumps panel (white) with backing panel	CockpitSimParts	1	£34,99	£34,99
Bus panel with backing panel	CockpitSimParts	1	£19,99	£19,99
Fuel panel (white) with backing panel	CockpitSimParts	1	£39,99	£39,99
FLT panel with backing panel	CockpitSimParts	1	£29,99	£29,99
Cabin pressure panel with backing panel	CockpitSimParts	1	£29,99	£29,99
Anti-Ice panel with backing panel	CockpitSimParts	1	£19,99	£19,99
Window heat panel with backing panel	CockpitSimParts	1	£24,99	£24,99
Blanking panels	CockpitSimParts	1	£9,99	£9,99
Air conditioning panel with backing panel	CockpitSimParts	1	£24,99	£24,99
Power source panel (white) with backing panel	CockpitSimParts	1	£39,99	£39,99
Electric control panel with backing panel	CockpitSimParts	1	£29,99	£29,99
Pneumatic system panel (white) with backing panel	CockpitSimParts	1	£39,99	£39,99
100 Annunciator set	CockpitSimParts	1	£49,00	£49,00
100 LED boxes	CockpitSimParts	1	£64,99	£64,99
LED bulb set	CockpitSimParts	1	£19,80	£19,80
Switches set with rotary and switch guards	CockpitSimParts	1	£169,99	£169,99
Dzus screw set	CockpitSimParts	1	£79,92	£79,92
<b>TOTAL</b>				<b>£847,56</b>

Como se aprecia en el total de la tabla, el valor por separado es muy superior al del kit, por lo que resulta realmente interesante comprar todo en conjunto. Como se comentó previamente, esta empresa realiza pedidos bajo demanda, teniendo tiempos de fabricación y entrega elevados. A título informativo, la compra se realizó el 26 de octubre de 2018 y fue enviada el 25 de febrero de 2019, siendo recibida aproximadamente 10-15 días después.

## 4.2 Material electrónico

En esta sección se puede encontrar el desglose de los elementos que fueron necesarios para desarrollar toda la electrónica de la parte trasera, permitiendo así obtener un *back-end* que cumpla con los requisitos de este proyecto. Para tal efecto, se ofrece la siguiente tabla, donde hay que señalar que ciertos elementos se compraron por duplicado para asegurar tener piezas de repuesto para el futuro mantenimiento del *Overhead Panel*.

Tabla 2. Material electrónico comprado

CONCEPTO	DISTRIBUIDOR	CANTIDAD	PRECIO UD.	TOTAL
Kit avanzado Arduino Mega 2560 Elegoo	Amazon	1	59,99 €	59,99 €
Kit 10 pcs servos SG90 KY66 Kuman	Amazon	1	23,99 €	23,99 €
Led 7 segmentos 2 dígitos 0.36" color verde cátodo común	Electrocomponentes	2	1,40 €	2,80 €
Led 7 segmentos 3 dígitos 0.36" color verde cátodo común	Electrocomponentes	5	1,80 €	9,00 €
Led 7 segmentos 5 dígitos 0.36" color blanco cátodo común	Electrocomponentes	3	3,00 €	9,00 €
100 pcs resistencias 220 ohms	Amazon	4	0,87 €	3,48 €
5 pcs IC 74HC595	Amazon	4	0,89 €	3,56 €
10 pcs IC 74HC165	Aliexpress	3	1,44 €	4,31 €
Cable 8m AWG22 6 colores TUOFENG	Amazon	2	18,99 €	37,98 €
Kit 32 PCBs Elegoo	Amazon	2	11,99 €	23,98 €
Carcasa transparente Arduino Mega	Amazon	1	2,99 €	2,99 €
Screw/Terminal Shield Arduino Mega	Amazon	1	19,99 €	19,99 €
60 pcs 5mm 2/3 pines PCB tornillo	Amazon	3	6,29 €	18,87 €
Cable 10m sección 0,5 mm <sup>2</sup> blanco	Electan	3	3,27 €	9,81 €
Cable 10m sección 0,5 mm <sup>2</sup> negro	Electan	1	3,27 €	3,27 €
Cable 10m sección 0,5 mm <sup>2</sup> rojo	Electan	1	3,27 €	3,27 €
Transistor IRF 1404 MOSFET N 40V 162A TO 220	Electrónica Embajadores	2	1,28 €	2,56 €
Disipador de calor para transistor	Electrónica Embajadores	2	0,50 €	1,00 €
Cable 70m unipolar multiflexible sección 0,5 mm <sup>2</sup> blanco	Electrónica Embajadores	1	12,11 €	12,11 €
Cable 10m unipolar multiflexible sección 0,5 mm <sup>2</sup> verde	Electrónica Embajadores	1	2,81 €	2,81 €
IC 74HC595	Electrónica Embajadores	20	0,38 €	7,60 €
IC 74HC165	Electrónica Embajadores	5	0,37 €	1,86 €
Condensador electrolítico tántalo 0,1µF / 35V	Electrónica Embajadores	25	0,17 €	4,25 €
Zócalo IC 7,62 mm 16 pines plano	Electrónica Embajadores	40	0,07 €	2,80 €
Regleta enchufe 3 terminales blanca	Ferretería local	1	4,50 €	4,50 €
Tira LED ALRAI 270mm	Donación	24	0,00 €	0,00 €
PSU FSP350-60EMDN	Donación	1	0,00 €	0,00 €
<b>TOTAL</b>				<b>275,78 €</b>

#### 4.3 Software de pago

Aunque para la realización de este proyecto se han usado algunos programas gratuitos como Arduino IDE para la programación, Fritzing para el diseño de circuitos, FreeCAD para el diseño de las piezas, y Ultimaker CURA para su impresión en 3D, otro software se ha tenido que comprar, exponiendo en la siguiente tabla el precio de las licencias.

Tabla 3. Software de pago comprado

CONCEPTO	DISTRIBUIDOR	CANTIDAD	PRECIO UD.	TOTAL
Microsoft Flight Simulator X Gold Edition	Microsoft	1	34,99 €	34,99 €
PMDG 737NGX Base Package for FSX	PMDG	1	65,26 €	65,26 €
FSUIPC4 and WIDEFS7 for FSX	SimMarket	1	18,00 €	18,00 €
<b>TOTAL</b>				<b>118,25 €</b>

#### 4.4 Material adicional y utillaje

Adicional a todo lo expuesto anteriormente en este capítulo, ha sido necesario comprar material extra para realizar el proyecto y, por no pertenecer a las anteriores categorías, se exponen en esta nueva lista.

*Tabla 4. Material adicional y utillaje comprado*

CONCEPTO	DISTRIBUIDOR	CANTIDAD	PRECIO UD.	TOTAL
Set de soldadura VICTISING	Amazon	1	20,49 €	20,49 €
Soporte multifunción tercera mano para soldadura con lupa y led Fixpoint	Amazon	1	16,12 €	16,12 €
Filamento PLA 1,75 mm rollo 1 kg color blanco GEEETECH	Amazon	1	19,99 €	19,99 €
Cinta metálica perforada	Ferretería local	1	5,00 €	5,00 €
Pistola termofusible y barras color transparente y negro	Ferretería local	1	6,00 €	6,00 €
Rollo estaño 40% Pb - 60% Sn	Bricor	4	4,50 €	18,00 €
Malla para desoldar 1,5 mm	Electrónica embajadores	1	2,89 €	2,89 €
Tornillería, tuercas y arandelas	Ferretería local	1	10,00 €	10,00 €
Bridas color negro y blanco	Ferretería local	1	5,00v	5,00 €
Tablones de madera de ramín para bastidor	Donación	1	0,00 €	0,00 €
Tubos de metal 40 x 30 x 2 mm para estructura metálica del bastidor	Donación	1	0,00 €	0,00 €
<b>TOTAL</b>				<b>103,49 €</b>



## 5. Diseño del *Overhead Panel*

Este capítulo pretende mostrar el proceso de diseño del *Overhead Panel*. Esta fase es la primera que se lleva a cabo en el proyecto y es de vital importancia, ya que un buen diseño previene de errores en las fases de desarrollo e implementación.

Para facilitar la compresión del lector de todo el proceso, se ha decidido dividir el capítulo en las siguientes secciones, permitiendo profundizar en ellas de forma progresiva sin mezclar los conceptos entre unas y otras:

1. Nomenclatura en el diseño
2. Pruebas previas con Arduino
3. Bastidor de madera
4. Soporte metálico del *Overhead Panel*
5. Electrónica
6. Piezas 3D

## 5.1 Nomenclatura en el diseño

Uno de los aspectos esenciales a la hora de diseñar un producto o un sistema es dejar constancia de la nomenclatura que se va a utilizar. Esta herramienta permite abreviar a la hora de realizar el proyecto y por consiguiente hacerlo más eficiente.

### 5.1.1 Nomenclatura de los paneles

El *Overhead Panel* es el punto de acceso para los pilotos a la mayoría de los sistemas del avión. A través de este, situado encima de las cabezas de la tripulación, se controlan sistemas como el de combustible, eléctrico, aire acondicionado y presurización entre otros. Para la facilidad de localización de estos sistemas, estos se agrupan en subpaneles claramente identificados. Todos ellos están organizados a lo largo de cinco columnas. De esta manera se forma una especie de matriz de filas y columnas. Por tanto, la nomenclatura a utilizar durante este proyecto se basa en este concepto. Se propone el siguiente ejemplo:

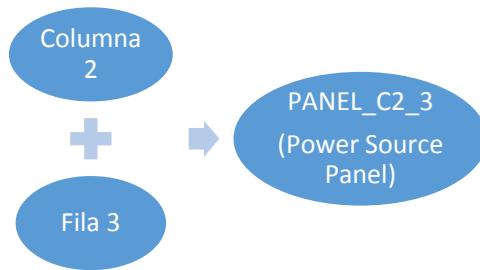


Ilustración 7. Ejemplo de nomenclatura de los paneles

De esta manera se confecciona la nomenclatura de los paneles, que será de gran ayuda para el entendimiento del código de Arduino que se expondrá más adelante.



Ilustración 8. Nomenclatura de los paneles

### 5.1.2 Nomenclatura de los circuitos integrados

Los circuitos integrados (o *integrated circuits*, IC) son chips de 16 pines que permiten reducir el número de cables de entrada al Arduino, permitiendo el uso de una sola placa en todo el proyecto. Aunque se profundizará más en ellos en la sección 5.5 de este capítulo, dado que tienen su propia nomenclatura, han de incluirse también aquí.

En el proyecto se usan dos tipos, el IC 74HC595 y el IC 74HC165. Ambos son registros de desplazamiento (o *shift registers*, SR). El primero es un registro serie-paralelo y el segundo un registro paralelo-serie. Dado que habrá una cantidad importante de cada uno de ellos, es necesario distinguirlos entre sí. Por ello, a los 74HC595, usados para las salidas del panel, se les denominará SR\_OUT\_X, siendo X el número que los identifica. Sus ocho posibles salidas irán de la Q0 a la Q7.

Por el contrario, el IC 74HC165 se utiliza para las entradas del panel. Por ello será denominado SR\_IN\_X. En este caso sus ocho posibles entradas van de la D0 a la D7.

Los circuitos integrados se pueden conectar entre ellos en modo serie, creando largas cadenas y permitiendo más salidas o entradas con el mismo número de cables al Arduino. En el código, las variables asociadas a los grupos de SR\_OUT\_X serán nombradas con el número del último SR de la cadena. Contrariamente, las variables asociadas a los grupos de SR\_IN\_X serán identificadas con el número del primer SR de la cadena. Esta nomenclatura cobrará sentido cuando se explique su funcionamiento detallado en la sección 5.5 de este capítulo.

## 5.2 Pruebas previas con Arduino

A la hora de comenzar este proyecto fue necesario comprobar su viabilidad, para asegurar que era posible integrarlo con la tecnología de Arduino. Así pues, meses antes de pedir el kit de piezas a la página Cockpit Sim Parts LTD se adquirió un kit avanzado del Arduino Mega 2560.

En este se incluía, al menos, una pieza de cada tipo de las que se iban a usar en la construcción del *Overhead Panel*. Entre ellas se encontraban interruptores, selectores, botones, potenciómetros, codificadores rotatorios, leds de 5 mm, displays LED 7 segmentos y servomotores. Además, incluía elementos de electrónica como resistencias y condensadores, además de un IC 74HC595, que serían ampliamente usados posteriormente.

Por ello, se empezaron a construir prototipos de los primeros paneles, haciendo uso de la *protoboard*, una placa que permite montar circuitos sin necesidad de realizar puntos de soldadura. Esto facilitaba la tarea, pues al no tener que soldar, los tiempos de montaje se reducían enormemente.

En la primera versión de los prototipos (*v1*), salidas y entradas se conectaban directamente a los pines digitales de la placa Arduino. Mediante esta configuración se pretendía crear el código básico en el que se apoyaría el resto del proyecto.

Se muestra a continuación una imagen del prototipo *v1* del PANEL\_C5\_4, compuesto de un display LED 7 segmentos de 5 dígitos y un codificador rotatorio (o *rotary encoder*), donde se está comprobando mediante la pulsación de este el modo *light test* del avión.

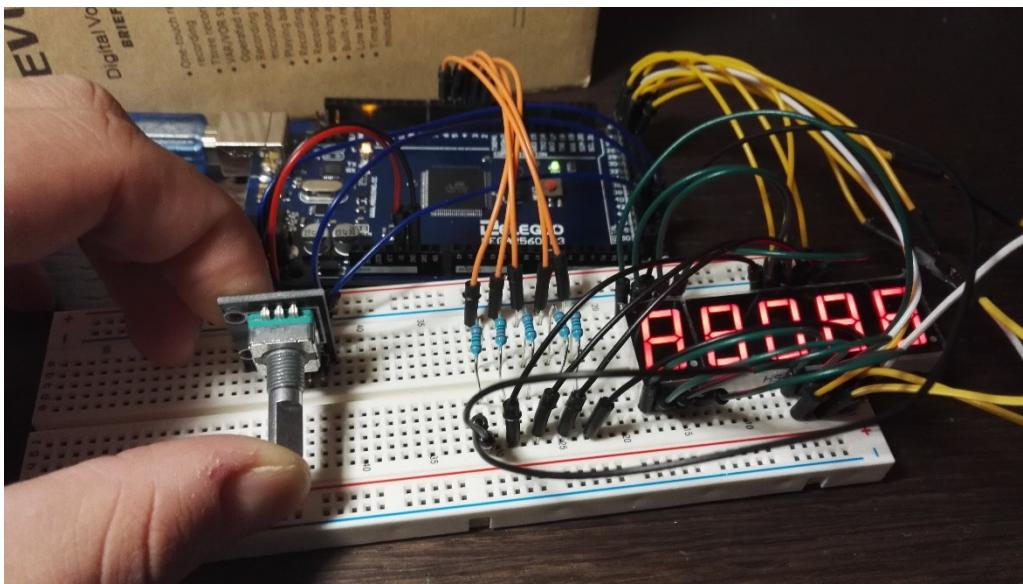


Ilustración 9. Prototipo v1 del PANEL\_C5\_4 en modo light test

Ya es en la siguiente versión de los prototipos (*v2*) donde se empiezan a integrar los registros de desplazamiento 74HC595 (salidas) y 74HC165 (entradas). De esta manera se pretende reducir el número de cables conectados a los pines del Arduino y empezar a contar cuantos ICs sería necesario comprar, además de comprobar si efectivamente son suficientes los 54 pines digitales que tiene el Arduino Mega 2560.

En la siguiente foto se puede comprobar el prototipo *v2* del PANEL\_C5\_4, donde gracias al uso de 2 unidades del IC 74HC595 se reduce el número de cables entrantes al Arduino de 12 (prototipo *v1*) a solo 3. Los anteriores cables que iban directamente al Arduino ahora pasan por los ICs, siendo estos los que gestionan ahora el flujo de información.

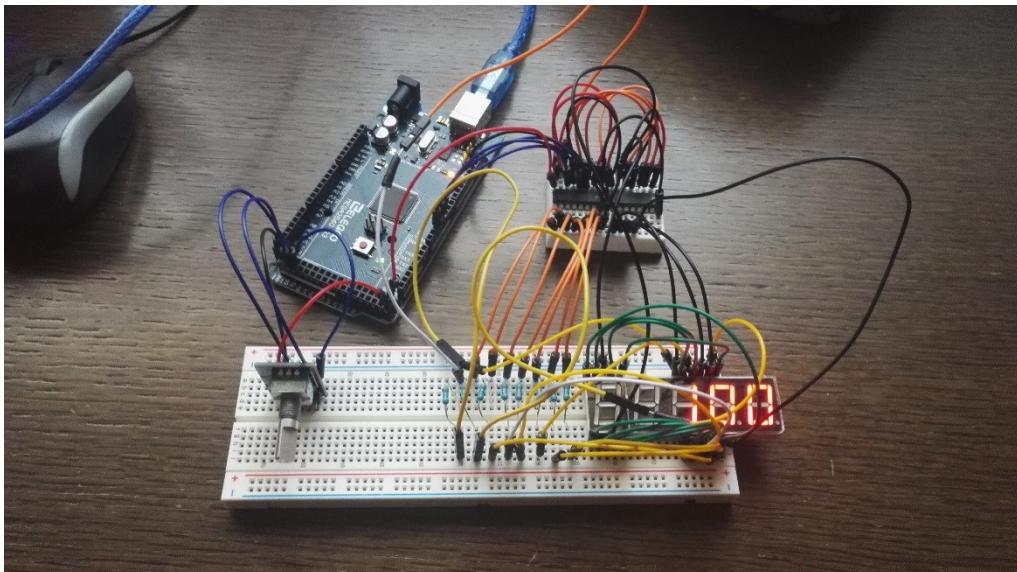


Ilustración 10. Prototipo *v2* del PANEL\_C5\_4 mostrando una LAND ALT de 100 ft.

Para acabar la fase de prototipos se hizo una tercera versión de estos (*v3*) donde se integraron varios paneles de la misma columna para ver la interconexión entre ellos. Tras esta versión, y aprovechando que se tenía el código depurado de cada columna por separado, para la siguiente versión del código (*v4*) solo hubo que juntarlos en uno y optimizarlo para su buen funcionamiento. En cambio, durante las últimas fases del proyecto se detectaron algunos fallos. Estos fueron corregidos en la versión definitiva del código (*v5*), que puede consultarse en el anexo, sección 11.16.

### 5.3 Bastidor de madera

Una de las partes fundamentales de este proyecto es el bastidor de madera, que es la estructura donde apoyan todos los paneles del kit de Cockpit Sim Parts LTD. Como se comentó en el capítulo de consideraciones iniciales, debe estar diseñado de manera que permita el mantenimiento del *Overhead Panel*. En los aviones reales el panel se descuelga al rotar dos tornillos de cuarto de vuelta localizados en las esquinas inferiores del mismo, quedando suspendido de forma vertical mediante unas bisagras en la parte superior. La siguiente fotografía ilustra esta situación.

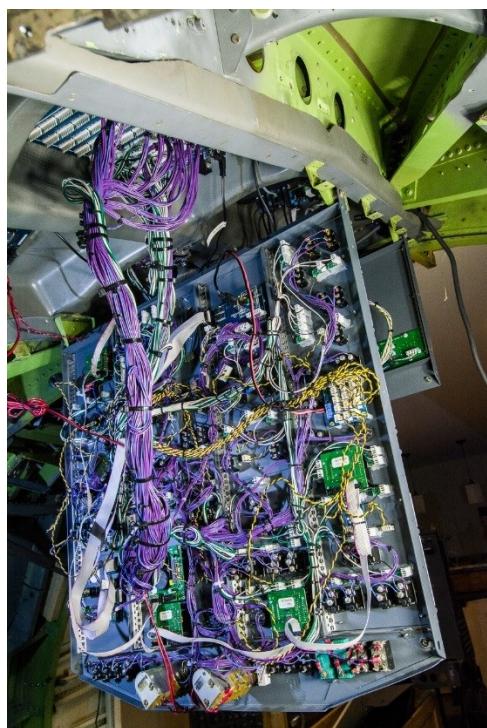


Ilustración 11. Trasera de un Overhead Panel descolgado (Fly737ng, 2017)

Para conseguir esta configuración el bastidor deberá estar compuesto de dos partes. Una primera donde vayan atornillados los paneles y que sea la que pueda descolgarse, y una segunda que sirva a modo de marco de la anterior quedando fija de forma estática a la estructura metálica. Los cables entre ellas deben tener la longitud suficiente para permitir que se descuelgue sin que estén en tensión.

Para abaratar costes, uno de los requisitos de este proyecto, en vez de realizar un bastidor de metal se ha optado por usar madera ya que es más sencilla de trabajar. En concreto se ha utilizado ramín, un tipo de madera de poca veta, facilitando así el trabajo de esta al no generar demasiado astillamiento. Además, es un tipo de madera de gran densidad, lo que le proporciona robustez a la hora de aguantar todo el peso que se va a colgar de ella.

A continuación, se muestra el diseño de ambas partes del bastidor, realizadas con el programa gratuito de modelado 3D FreeCAD. Los planos acotados de estas podrán consultarse en el anexo, en las secciones 11.1 y 11.2 respectivamente.

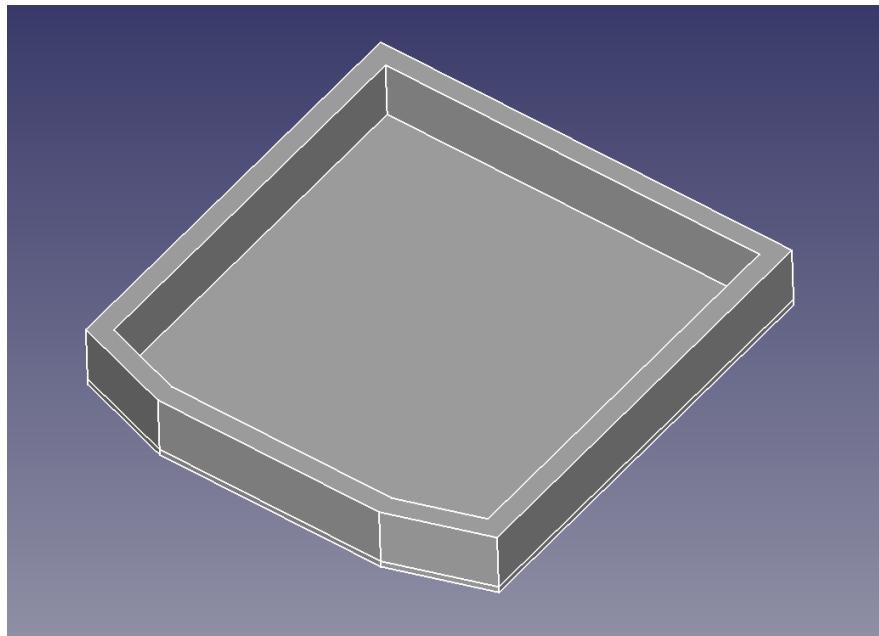


Ilustración 12. Modelo 3D del bastidor inferior de madera

Como se aprecia en esta última foto, para cerrar el bastidor inferior se ha añadido una tapa trasera de 10 mm de grosor. Con ella, la electrónica estará protegida de polvo y agentes externos que la podrían dañar con el paso del tiempo.

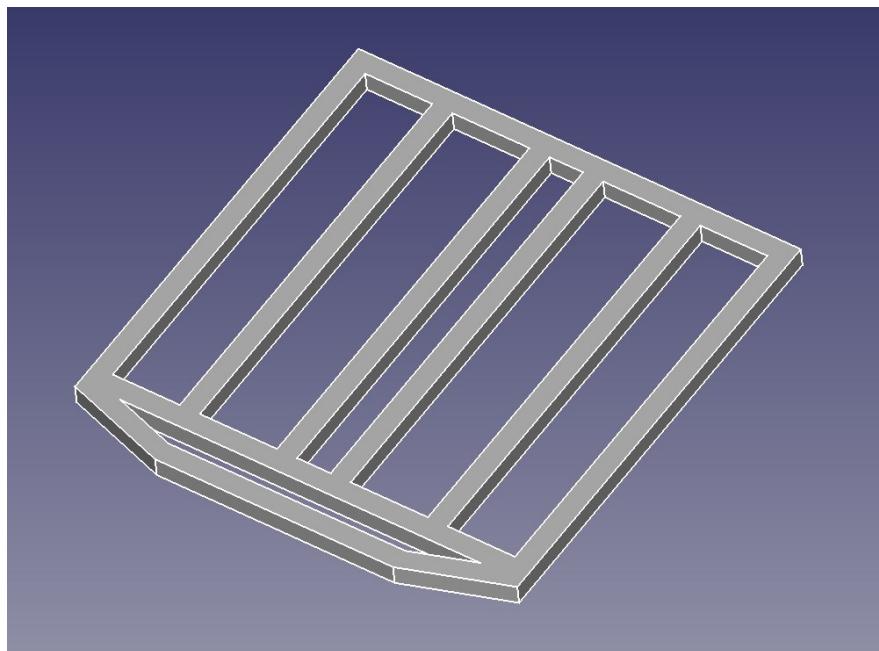
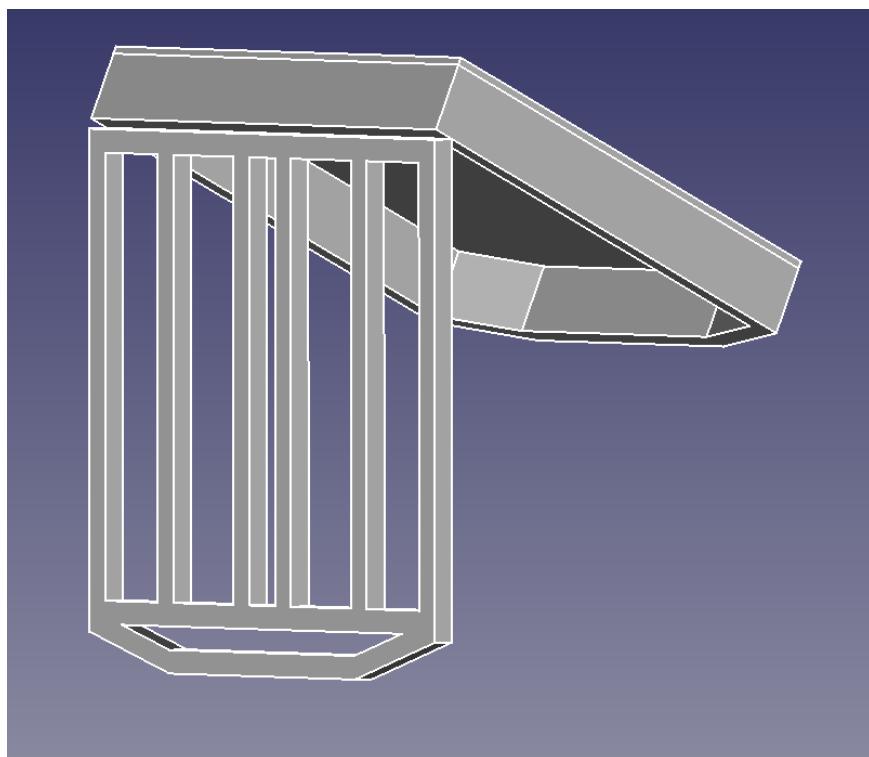


Ilustración 13. Modelo 3D del bastidor superior de madera

El bastidor superior deja delimitadas las cinco columnas que se vieron en la *ilustración 8*, además de la zona inferior para el PANEL\_C1\_4. Los paneles irán colocados sobre los listones laterales y consecuentemente atornillados a ellos. Finalmente, en la parte superior irán acopladas cuatro bisagras que permitirán que se pueda descolgar para el posible mantenimiento futuro.



*Ilustración 14. Modelo 3D del bastidor de madera descolgado*

## 5.4 Soporte metálico del *Overhead Panel*

Tras diseñar el bastidor de madera es necesario crear una estructura metálica que lo mantenga a una altura e inclinación similares a la de una cabina real. Como referencia de medidas se ha tomado el documento realizado por *Markuspilot*, que se podrá consultar a través del enlace web localizado en el capítulo 10. En este documento se han tomado de referencia las medidas de las siguientes imágenes.

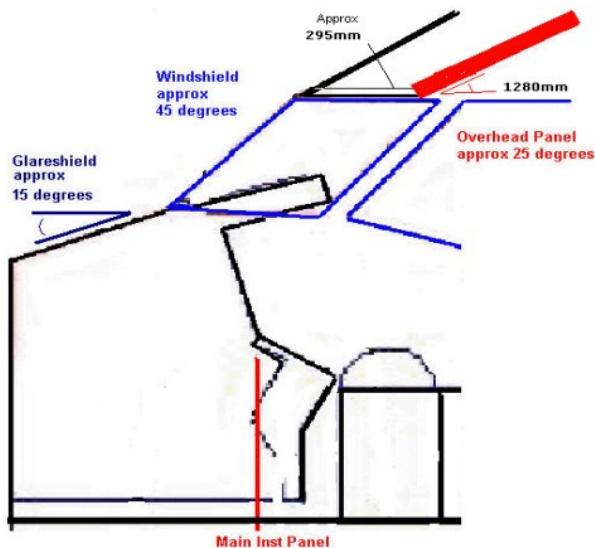


Ilustración 15. Medidas de una cabina real de B737, vista lateral (Markuspilot 2013)

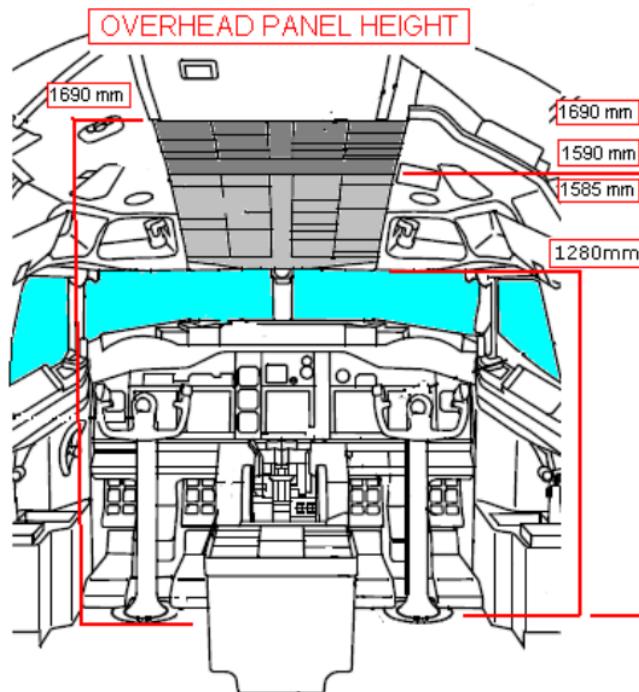


Ilustración 16. Medidas de una cabina real de B737, vista frontal (Markuspilot 2013)

Algunas de estas medidas han sido variadas ligeramente dado que la altura de la mesa en la que se apoya el simulador es algo mayor que la distancia del asiento al suelo de la cabina real. No obstante, otras como el ángulo de inclinación del *Overhead Panel* se han conservado.

La estructura metálica debe cumplir una serie de requisitos. El primero, como se ha comentado, debe mantener el *Overhead Panel* a un ángulo y altura similar al real. Debe ser también lo suficientemente robusto como para aguantar todo su peso, incluyendo el propio del bastidor de madera, pero también el de toda la electrónica que será añadida posteriormente. Por último, debe permitir ser transportado fácilmente, en primer lugar, para el día de la defensa de este proyecto en la universidad, pero también para posibles ferias de simulación que se puedan realizar en el futuro.

Cuando el *Overhead Panel* esté unido al simulador estará atornillado a la trasera de la mesa y contra la pared, no permitiendo la posibilidad de vencer hacia delante. En el resto de las situaciones anteriormente descritas, para evitar construir una base con unas patas demasiado largas, lo que incumpliría el requisito de facilidad de transporte, se utilizará el soporte metálico junto con unos contrapesos colocados en el tubo vertical de este. Es necesario tener en cuenta que cuando el bastidor de madera está descolgado el centro de gravedad se desplaza más hacia el exterior. Por ello, los contrapesos deben ser elegidos de tal forma que puedan contrarrestar la posibilidad de vencer hacia delante en esta configuración al ser la más restrictiva.

Teniendo en cuenta todas estas consideraciones, se muestra a continuación una imagen del diseño adoptado. Los planos acotados pueden consultarse en el anexo, en la sección 11.3.

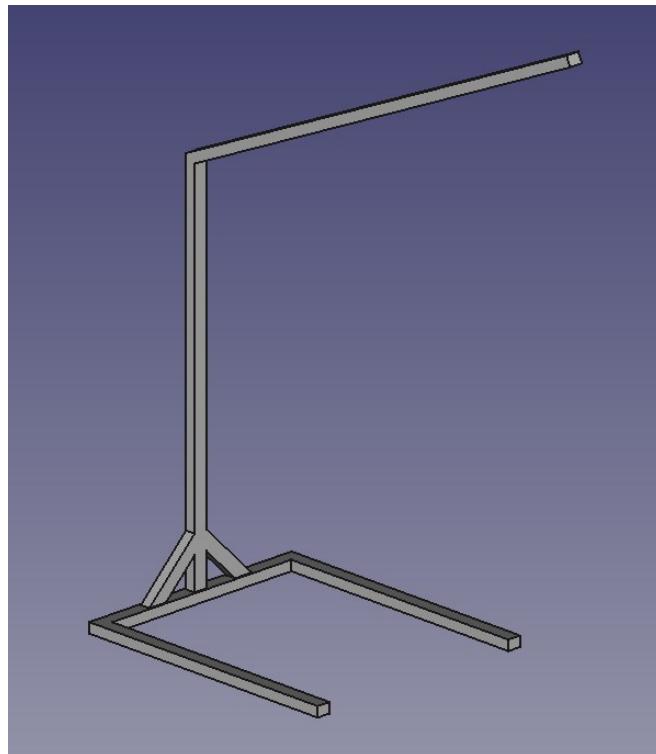


Ilustración 17. Modelo 3D del soporte metálico del *Overhead Panel*

## 5.5 Electrónica

En esta sección se pretende mostrar todo el diseño electrónico del *Overhead Panel*. Este está formado por los siguientes elementos:

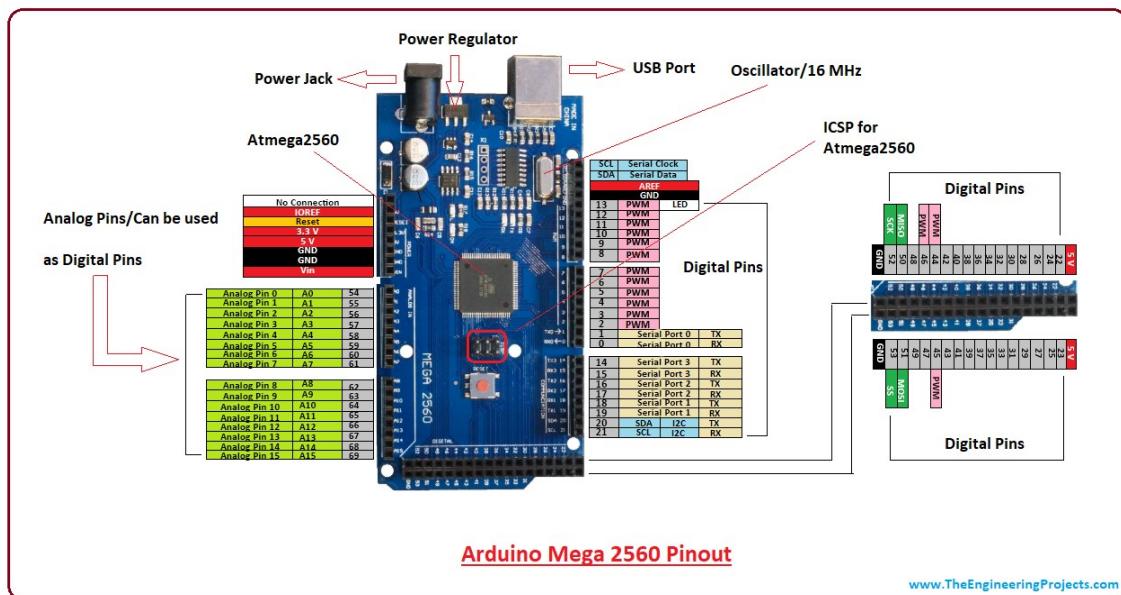
- 200 leds de 5 mm
- 1 display LED 7 segmentos de 2 dígitos
- 4 displays LED 7 segmentos de 3 dígitos
- 2 displays LED 7 segmentos de 5 dígitos
- 24 tiras LED controladas en intensidad por un transistor IRF1404 MOSFET 40V 162A
- 10 servomotores
- 50 interruptores de 2 posiciones
- 23 interruptores de 3 posiciones (algunos de posición momentánea)
- 9 pulsadores tipo botón
- 6 selectores rotatorios de 45 grados (8 posiciones)
- 4 selectores rotatorios de 30 grados (12 posiciones)
- 2 codificadores rotatorios
- 5 potenciómetros 10kΩ
- 22 IC 74HC595 (SR\_OUT)
- 19 IC 74HC165 (SR\_IN)
- 297 resistencias de 220 Ω, 5 resistencias de 10 kΩ y 1 resistencia de 330 Ω
- 22 condensadores electrolíticos de tántalo de 0,1µF / 35V
- 1 Arduino Mega 2560
- 1 fuente de alimentación modelo FSP350-60emdn

Debido a la complejidad electrónica del proyecto, resultaría difícil hacer un esquema electrónico completo del *Overhead Panel*. No obstante, dado que el número de elementos diferentes no es tan extenso, se realizará una explicación individual de cada uno, restando solo hacer la repetición de los circuitos individuales tantas veces como elementos del mismo tipo haya. Por lo tanto, esta sección se dividirá en los siguientes apartados:

1. Microcontrolador Arduino Mega 2560
2. Fuente de alimentación
3. IC 74HC595 (SR\_OUT)
4. IC 74HC165 (SR\_IN)
5. Led de 5 mm
6. Display LED 7 segmentos (2, 3 y 5 dígitos)
7. Servomotor
8. Interruptor (2 y 3 posiciones)
9. Pulsador tipo botón
10. Selector rotatorio (45 y 30 grados)
11. Codificador rotatorio
12. Potenciómetro
13. Transistor IRF1404 MOSFET

### 5.5.1 Microcontrolador Arduino Mega 2560

Como se comentó en el capítulo de consideraciones iniciales, se ha seleccionado como microcontrolador del proyecto al Arduino Mega 2560 por su facilidad de programación y por la gran cantidad de información que hay acerca de él en Internet. Este cuenta con 54 pines digitales de entrada / salida (de los cuales 15 se pueden utilizar como salidas PWM) y 16 entradas analógicas. A continuación, se muestra una imagen de la disposición de sus pines.



*Ilustración 18. Disposición de los pines del Arduino Mega 2560 (The Engineering Projects 2018)*

El número de pines, en primera aproximación, es insuficiente para la consecución del proyecto. No obstante gracias a los circuitos integrados 74HC595 y 74HC165 (que se explicarán en los siguientes apartados) es posible multiplicar los pines del Arduino hasta un número prácticamente ilimitado. De cara a toda esta sección del informe, se expone en el anexo, sección 11.4, la lista de los elementos conectados a los pines del Arduino. A pesar del gran número de componentes, han quedado pines libres, por lo que se podría incluso plantear usar la misma placa para la ampliación futura de la parte superior del *Overhead Panel*.

El Arduino Mega 2560 va conectado al ordenador mediante un USB tipo B y recibe alimentación eléctrica de un adaptador AC/DC que le proporciona 9V a 1A (dentro del intervalo de sus especificaciones técnicas, que va de 7V a 12V). Este adaptador sería incapaz de proporcionar toda la energía que necesita el *Overhead Panel* para funcionar. Por ello, se hace necesario incorporar una fuente de alimentación externa, que se verá en el siguiente apartado.

### 5.5.2 Fuente de alimentación

Como se ha comentado previamente, la carga eléctrica del *Overhead Panel* es elevada. Por ello, es imposible extraer toda la corriente directamente del Arduino. Según especificaciones técnicas de este, de cada pin digital se puede extraer hasta un máximo de 40mA y en total un máximo de 200mA. Estas cifras se quedan muy por detrás de lo que se espera conseguir.

Teniendo en cuenta las características del microcontrolador, es necesario instalar una fuente de alimentación externa (o *power supply unit*, PSU) que alimente de manera independiente al *Overhead Panel*. Para abaratar los costes del proyecto se ha utilizado la PSU de un antiguo ordenador de sobremesa que cumple con los requisitos eléctricos del proyecto.



Ilustración 19. Especificaciones técnicas de la PSU modelo FSP350-60EMDN

De uno de los conectores tipo *molex* disponibles se han obtenido 4 cables. Dos de color negro, que funcionan como negativo, un cable rojo, que proporciona 5V hasta un máximo de 15A, y un cable amarillo, que ofrece 12V hasta un máximo de 16A. El circuito de 5V se utilizará para alimentar la electrónica del *Overhead Panel*, que es la tensión de trabajo del Arduino, mientras que el de 12V alimentará a la retroiluminación. Dado que tanto la PSU como el Arduino comparten circuito, es de vital importancia unir los terminales negativos de ambos componentes. Si esto no se realiza conllevaría a un mal funcionamiento de la instalación, pudiendo incluso dañar la electrónica interna del Arduino.

Cuando la fuente está apagada, a través del cable morado se proporcionan 5V a 2,5A. Para encender la PSU y obtener mayor potencia, en el conector tipo ATX de 20 o 24 pines hay un cable de color verde. Cuando este es conectado a tierra, la PSU se enciende. Dado que la fuente de alimentación tiene un interruptor físico en el exterior, este cable será conectado al negativo de forma permanente. De esta manera cuando se quiera alimentar el *Overhead Panel*, solamente habrá que llevar el interruptor trasero a su posición de encendido.

### 5.5.3 IC 74HC595 (SR\_OUT)

Antes de explicar los componentes de entrada y salida del *Overhead Panel* es conveniente entender el funcionamiento de los circuitos integrados, dado que estos son los intermediarios entre el Arduino y los demás componentes. Una vez explicado el 74HC595, el 74HC165 será fácil de comprender pues su concepto es muy parecido.

La misión de un IC 74HC595 es aumentar los pines digitales de salida del Arduino. Cada chip tiene 16 pines. De ellos, 8 son posibles salidas (Q0 a Q7). A continuación, se muestra la disposición de pines extraída de su ficha técnica.

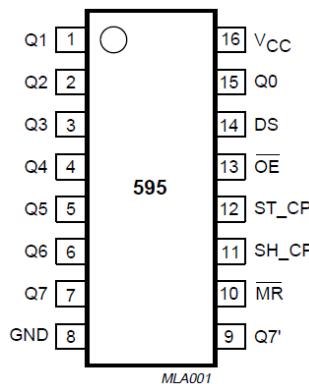


Ilustración 20. Esquema del IC 74HC595 (Philips 2003)

Tabla 5. Disposición de pines en el IC 74HC595 (Philips 2003)

PIN	SÍMBOLO	DESCRIPCIÓN
1	Q1	salida de datos en paralelo
2	Q2	salida de datos en paralelo
3	Q3	salida de datos en paralelo
4	Q4	salida de datos en paralelo
5	Q5	salida de datos en paralelo
6	Q6	salida de datos en paralelo
7	Q7	salida de datos en paralelo
8	GND	tierra (0V)
9	Q7'	salida de datos en serie
10	M̄R	reset maestro o <i>master reset</i> (activo en bajo)
11	SH_CP	entrada de la señal de reloj
12	ST_CP	entrada de la señal de latch
13	OĒ	habilitación de salida u <i>output enable</i> (activo en bajo)
14	DS	entrada de datos en serie
15	Q0	salida de datos en paralelo
16	Vcc	tensión de alimentación positiva

El funcionamiento del circuito integrado viene muy bien representado por la siguiente imagen, realizada por la página de electrónica *Last Minute Engineers*.

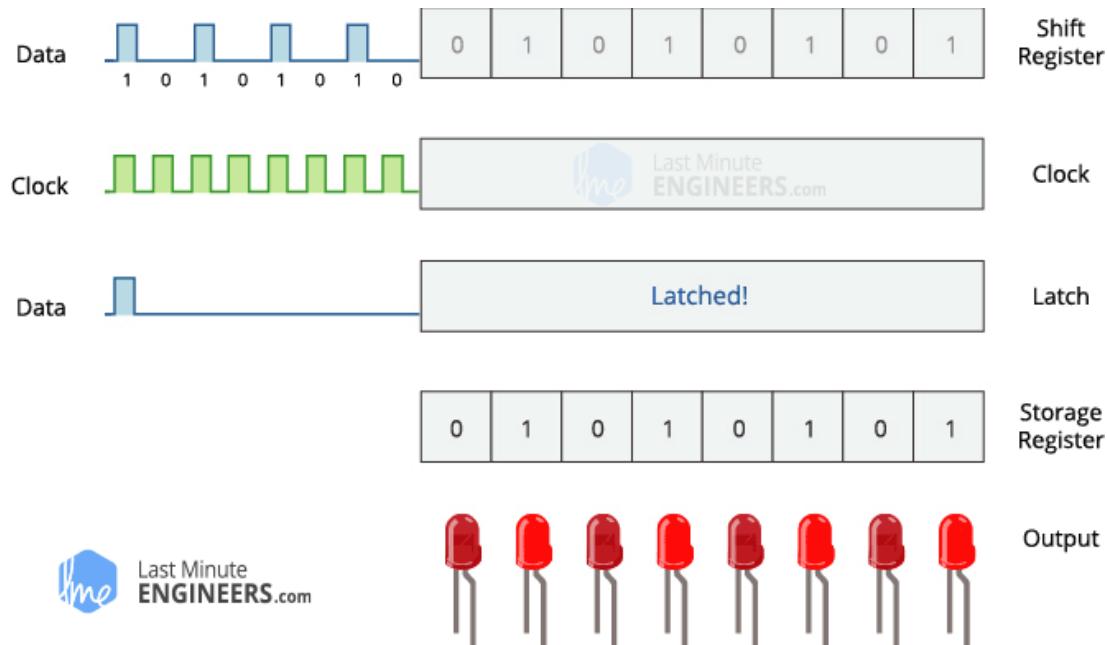


Ilustración 21. Representación del funcionamiento interno del IC 74HC595 (Last Minute Engineers 2019)

La primera línea representa el flujo de bits a través del pin DS, que es el primer cable que se conecta al Arduino. El 1 representa el estado de encendido, mientras que el 0 es el estado de apagado. Los datos avanzan en serie hacia el interior del IC cada vez que se da un pulso de reloj, correspondiente con el segundo cable conectado al Arduino. Nótese que hay 8 espacios, que representan las 8 salidas disponibles (pines Q0 a Q7). Cada 8 pulsos de reloj se emite una señal de latch a través del tercer y último cable conectado al Arduino. Cuando el SR la reconoce, transmite los estados de la primera fila a la de almacenamiento, encendiéndo o apagando los leds. Así pues, los datos van entrando en serie, para después salir en paralelo. Por eso el chip recibe el nombre de registro de desplazamiento serie-paralelo. De esta manera conseguimos 8 salidas con solo 3 cables al Arduino, ahorrando 5.

Como se comentó previamente, estos chips pueden ser conectados en serie entre sí, permitiendo un mayor número de salidas con el mismo número de cables al Arduino. Para ello es necesario conectar el pin Q7' del primer IC, con el pin DS del segundo. De esta manera los datos de la primera línea de la imagen continuarán fluyendo al segundo IC. Lo siguiente sería conectar en común los pines SH\_CP y ST\_CP para que la señal de reloj y latch estén sincronizadas. Por último, en código habría que hacer que en vez de emitir una señal de latch cada 8 pulsos de reloj ahora lo haga cada 16, permitiendo así que la información llegue por completo al segundo IC. Para más ICs en serie se repite este proceso, cambiando consecuentemente cada cuántos pulsos de reloj se envía la señal de latch.

Esta ampliación podría ser prácticamente ilimitada, siendo la única restricción el hecho de que sea una conexión en serie, lo que podría retardar la señal en los últimos circuitos integrados de la cadena si esta es muy larga. Para este proyecto no serán conectados más de nueve 74HC595 en la misma cadena, siendo los tiempos de latencia en esta configuración aceptables.

En el anexo, sección 11.5, puede consultarse un esquema eléctrico de la conexión en serie de dos SR\_OUT y 16 leds. No tiene sentido aumentar la complejidad de este poniendo más ICs en cadena, pues sus conexiones son extrapolables. A continuación, se muestra un esquema más simplificado de la situación para facilitar su entendimiento. En él se hace uso de una *protoboard*, tal y como se realizó en las pruebas iniciales del *Overhead Panel*. El símbolo de las 4 pilas simula la fuente de alimentación externa a 5V, mientras que la de 9V hace referencia al adaptador AC/DC.

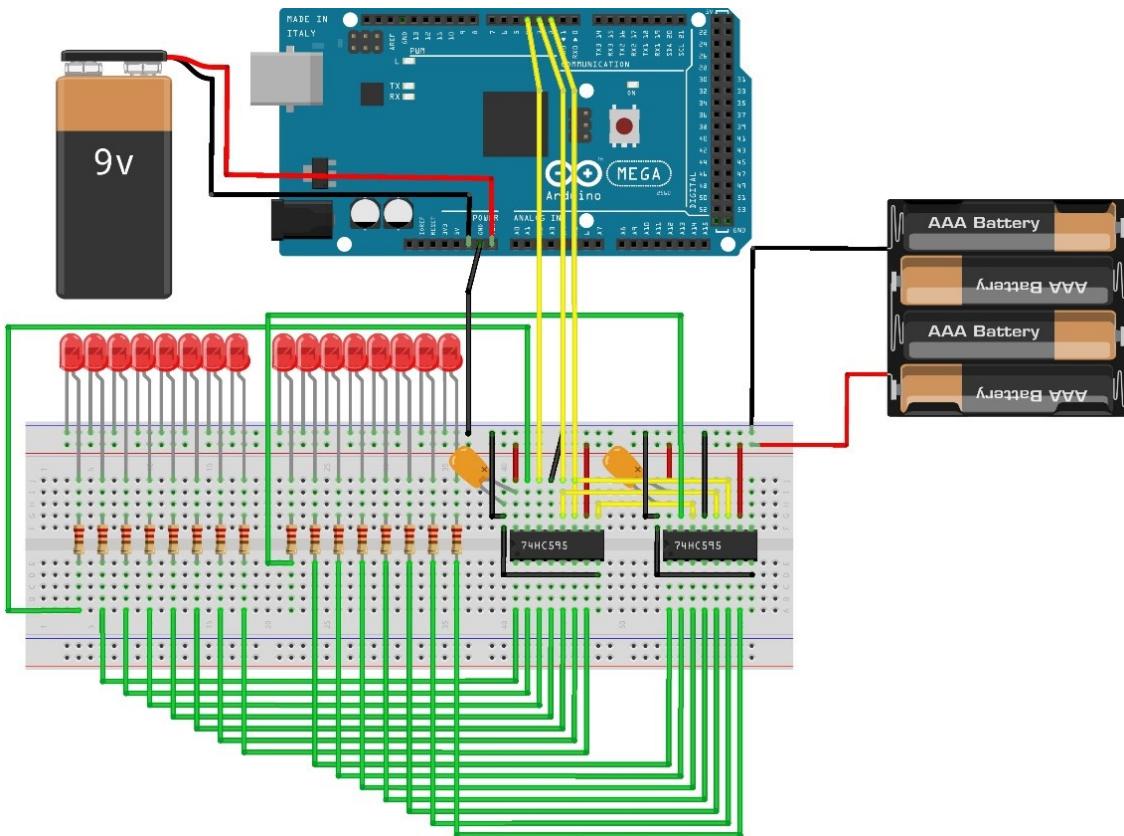


Ilustración 22. Esquema simplificado del funcionamiento del IC 74HC595

Como se puede ver en la imagen, se han introducido dos condensadores electrolíticos de tántalo de valores 0,1 µF / 35V lo más cerca posible del pin Vcc. Este condensador hace de desacoplador, filtrando la componente alterna de la fuente. Esto permite que el circuito reciba 5V constantes y asegure que en ningún momento se generan picos de tensión transitorios, siendo fácil de que ocurra cuando existe una rápida y constante conmutación de los estados lógicos de las salidas, pudiendo llegar a dañar el circuito integrado.

#### 5.5.4 IC 74HC165 (SR\_IN)

Una vez entendido el funcionamiento del IC 74HC595, el del IC 74HC165 resulta sencillo de explicar. El concepto es el mismo, salvo que ahora el chip lee el estado de los interruptores en los pines de entrada, que de nuevo son ocho por circuito integrado (D0 a D7), carga los datos en paralelo, y se los transmite al Arduino en formato serie. Por ello, el 74HC165 recibe el nombre de registro de desplazamiento paralelo-serie. Así pues, se consigue aumentar el número de entradas, donde se conectarán interruptores, pulsadores de tipo botón y selectores rotatorios.

La gran diferencia entre los SR\_OUT y los SR\_IN, es que estos últimos, en vez de 3 cables como tenían los primeros, tienen 4. A continuación se muestra una imagen de la disposición de sus pines.

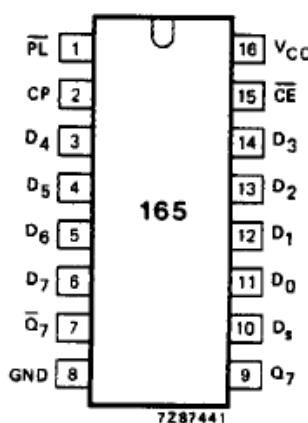


Ilustración 23. Esquema del IC 74HC165 (Philips 1990)

Tabla 6. Disposición de pines en el IC 74HC165 (Philips 1990)

PIN	SÍMBOLO	DESCRIPCIÓN
1	$\overline{PL}$	entrada de carga paralela asíncrona (activa en bajo)
2	CP	entrada de la señal de reloj
3	D4	entrada de datos en paralelo
4	D5	entrada de datos en paralelo
5	D6	entrada de datos en paralelo
6	D7	entrada de datos en paralelo
7	$\overline{Q7}$	salida complementaria del último estado
8	GND	tierra (0V)
9	Q7	salida de datos en serie del último estado
10	DS	entrada de datos en serie
11	D0	entrada de datos en paralelo
12	D1	entrada de datos en paralelo
13	D2	entrada de datos en paralelo
14	D3	entrada de datos en paralelo
15	$\overline{CE}$	entrada de activación de reloj o <i>clock enable</i> (activa en bajo)
16	Vcc	tensión de alimentación positiva

Cuando el pin  $\overline{PL}$  (primer cable al Arduino) está en bajo, se leen y se cargan los estados de los pines de entrada (D0 a D7) de forma asíncrona. Cuando el pin  $\overline{PL}$  está en alto, los datos entran en serie por el pin DS, provenientes de otros SR\_IN, a la vez que la información va saliendo progresivamente por el pin Q7 del último SR\_IN, que es el que lee el Arduino (segundo cable). Estas acciones van sucediendo como se vio previamente con los pulsos enviados a través de la señal de reloj del pin CP (tercer cable). El cuarto y último cable es el del pin  $\overline{CE}$ , cuya función es mostrarse en alto durante la adquisición de datos y en bajo durante el proceso de trabajo.

Al igual que ocurría con los 74HC595, los 74HC165 también se pueden conectar entre ellos. Para hacerlo es necesario conectar de forma común los pines  $\overline{PL}$ ,  $\overline{CE}$  y CP. Adicionalmente debe conectarse el Q7 del primer SR\_IN con el DS del segundo. Tal y como se hizo la otra vez, será necesario modificar el código acorde con el número total de SR\_IN en la cadena. Por último, hay que tener en cuenta que el fabricante recomienda que si algún pin de entrada no se utiliza se ponga en bajo, ya que podría causar falsos activos.

En el anexo, sección 11.6, puede consultarse un esquema eléctrico de la conexión en serie de dos SR\_IN, 4 interruptores de dos posiciones (parte superior de la *protoboard*), 4 de tres posiciones (parte inferior) y 4 pulsadores de tipo botón (parte izquierda). A continuación, se muestra un esquema más simplificado de la situación para facilitar su entendimiento.

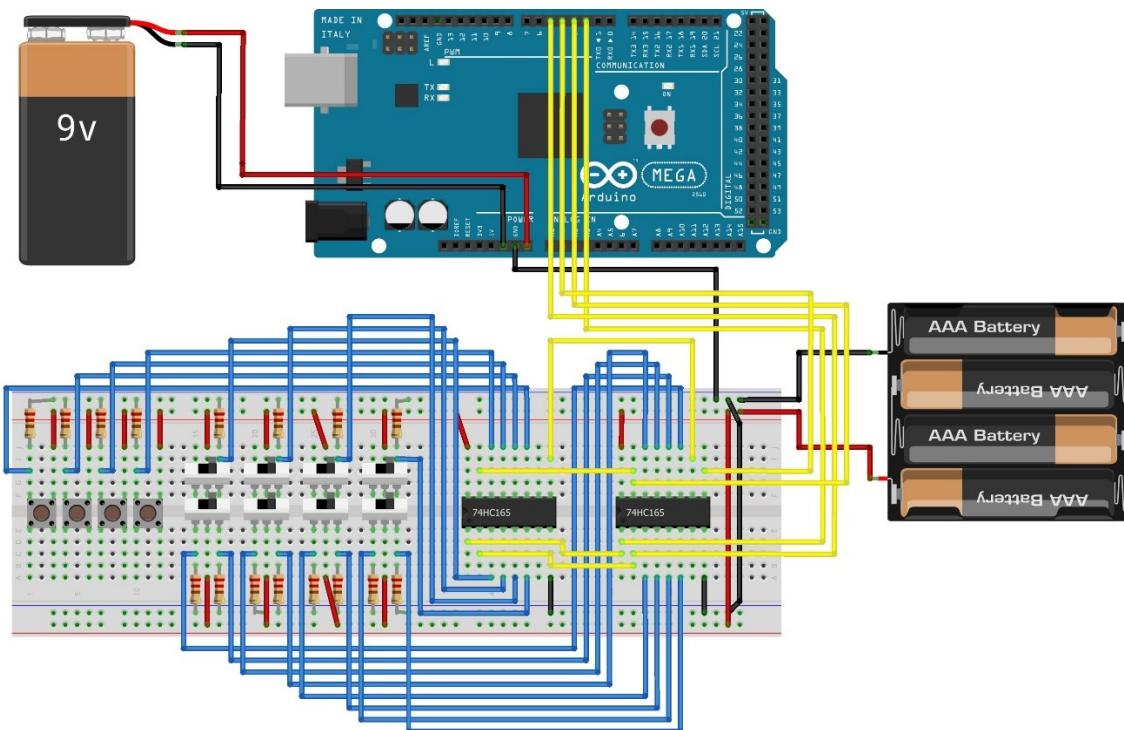


Ilustración 24. Esquema simplificado del funcionamiento del IC 74HC165

### 5.5.5 Led de 5 mm

Un diodo emisor de luz o LED (*Light-Emitting Diode*) es una fuente de luz constituida por un material semiconductor dotado de dos terminales, el ánodo (+) y el cátodo (-). A la hora de usarlos es necesario unirlos con una resistencia, ya que una corriente por encima de su valor máximo podría quemarlo. Normalmente, la corriente máxima que puede soportar un led está cerca de los 20mA. En este proyecto el valor de las resistencias será de  $220\Omega$  ( $\pm 5\%$ ).

Al ser un diodo, solo conduce la electricidad en un sentido, por lo que será necesario prestar atención en la identificación del ánodo y el cátodo. Normalmente, la patilla del ánodo (+) es más larga. También se puede identificar mirando en el interior del led, siendo su pletina la más estrecha.

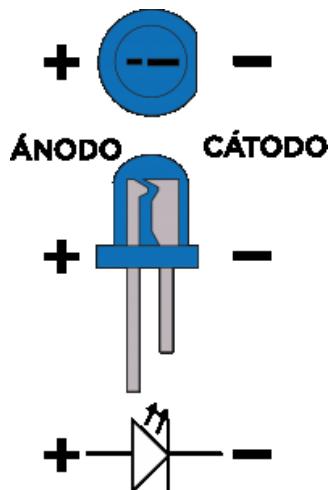


Ilustración 25. Símbolo y polaridad del diodo LED (Ebotics 2019)

Su funcionamiento es bastante simple. Para ello, se considerará de nuevo la *ilustración 22*, donde se usó dos IC 74HC595 para controlar 16 leds, además del esquema eléctrico del anexo, sección 11.5. El ánodo del led debe ser conectado a una de las salidas del SR\_OUT (pines Q0 a Q7) a través de una resistencia de valor  $220\Omega$  ( $\pm 5\%$ ). Valores de resistencia más pequeños aumentarían la luminosidad, y valores más altos la disminuirían, al circular menos corriente a través de él. Por último, el cátodo se conecta a tierra. Cuando a través del pin se emita un 0 lógico el led estará apagado. Cuando lo que se emite es un 1 lógico, el led pasará a emitir luz.

En el *Overhead Panel* se usarán leds de 5 mm transparentes de colores ámbar, verde y azul. El hecho de ser transparente permite que, cuando no estén encendidos, pase la luz blanca de la retroiluminación, posibilitando distinguir las letras del anunciador.

Los diodos LED estarán colocados en las cajas de los anunciadores, evitando que la luz que emiten salga hacia otra dirección que no sea hacia adelante. Estas cajas, permiten colocar dos leds en cada una, como se aprecia en la siguiente imagen.



Ilustración 26. Anunciadores del Overhead Panel

Dado que los leds de un mismo anunciador emitirán luz a la vez, es poco eficiente destinar una salida del SR\_OUT para cada uno. Así pues, queda decidir si entre ellos estarán conectados en serie o en paralelo. En la primera configuración la intensidad a través de ellos sería la misma. En la configuración en paralelo esta se repartiría. Esta última configuración tiene una ventaja frente a la primera y es que aumenta la redundancia de la instalación. Si uno de los dos leds se fundiera en medio de un vuelo el otro seguiría emitiendo, permitiendo acabar el vuelo antes de reponer la pieza. En la configuración serie si un led se funde el otro deja de emitir. Por ello, la configuración elegida será la conexión en paralelo. En la siguiente imagen se aprecia esta configuración, siendo el resto del circuito el que se ve en el anexo, sección 11.5.

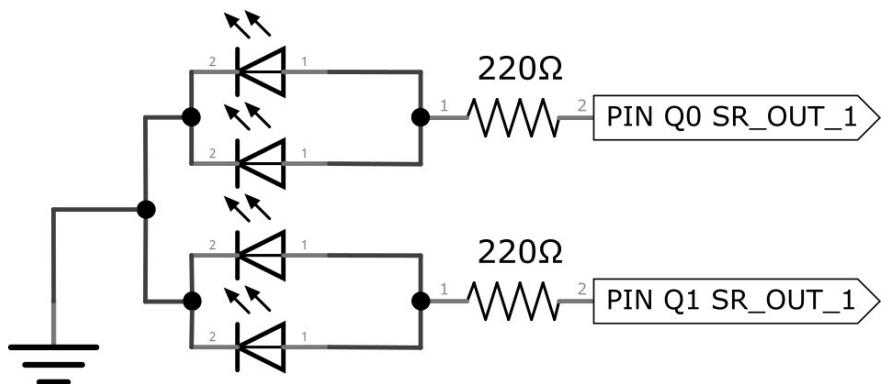


Ilustración 27. Conexión en paralelo de los leds de los anunciadores

### 5.5.6 Display LED 7 segmentos (2, 3 y 5 dígitos)

Un display LED 7 segmentos no deja de ser un componente electrónico con 7 leds internos. Estos diodos emiten luz a través de una estructura con forma, que permite generar números del 0 al 9 y algunas letras. Algunos incluyen un punto en la parte inferior derecha para marcar decimales. Cada diodo o segmento recibe una letra que lo identifica (de la A a la G).

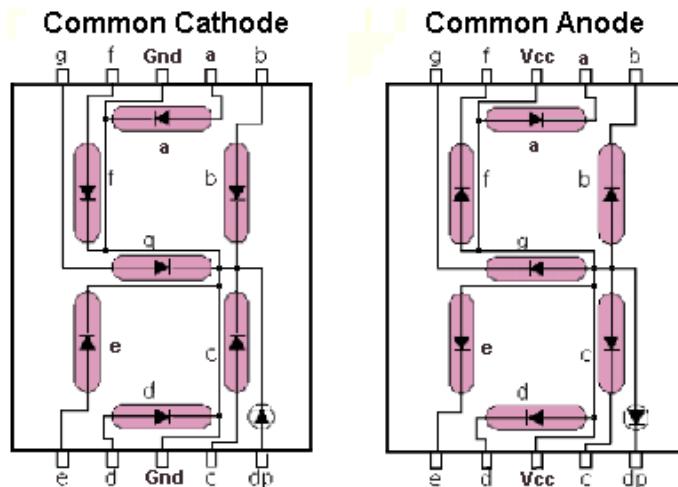


Ilustración 28. Nomenclatura del display LED 7 segmentos (Electrontools 2016)

Existen dos tipos de display, cátodo común y ánodo común. Como se aprecia en la imagen, la diferencia eléctrica entre ellos es el terminal del diodo LED que es común. Aunque se profundizará más en el capítulo 7 de este informe, a la hora de programar se diferencian en que en el display de cátodo común el led se enciende con un 1 lógico (activos en alto), mientras que en el de ánodo común esto ocurre con un 0 lógico (activos en bajo). Para este proyecto todos los displays serán de cátodo común. No se debe olvidar que siguen siendo diodos LED, por lo que es necesario usar resistencias de  $220\Omega$  en todos ellos.

Una vez entendido el concepto básico, ahora es necesario ver como se consiguen displays de más de un dígito. Aunque se podrían usar displays individuales, resulta poco eficiente ya que el número de 74HC595 aumentaría considerablemente. Para evitar esto se utiliza la técnica de multiplexado.

El multiplexado de los displays 7 segmentos se aprovecha de la característica del ser humano de ser incapaz de reconocer variaciones en el parpadeo de un led cuando se trabaja en la escala del milisegundo o el microsegundo. Así pues, si se encendiera y apagara cada dígito consecutivamente, a vista del ojo humano parecería que todos los dígitos del display están encendidos, cuando en realidad solo uno estaría emitiendo luz en cada instante de tiempo.

El esquema del multiplexado de los displays de 2, 3 y 5 dígitos pueden consultarse en el anexo, en las secciones 11.7, 11.8 y 11.9 respectivamente. En estos se podrán encontrar la disposición de pines de los displays.

Lo primero que podemos ver es que en la parte inferior del esquema de multiplexado aparecen los pines que ya se vieron en la *ilustración 28* y que controlan el encendido y apagado de cada segmento individual. En este proyecto el pin del punto (DP) no será utilizado, teniendo por tanto 7 pines asociados a los segmentos.

Por otro lado, en la parte superior tenemos los pines asociados al control de cada dígito. El número de estos pines es igual al número de dígitos del display. Así, por ejemplo, el display de 2 dígitos tiene dos pines, el 10 y el 5. De esta manera, en el display de dos dígitos se usarán 9 pines (7 segmentos y 2 dígitos), en el de tres dígitos 10 pines (7 segmentos y 3 dígitos) y en el de cinco dígitos 12 pines (7 segmentos y 5 dígitos). Esto se tendrá en cuenta para el cálculo de cuántos SR\_OUT serán necesarios, sabiendo que en cada chip hay 8 salidas disponibles. En los pines de dígitos no es necesario usar una resistencia, de manera que por cada display se usarán 7 resistencias de  $220\Omega$  conectadas a los pines de los segmentos.

En los displays de cátodo común los pines asociados a los dígitos son activos en bajo (en los de ánodo común son activos en alto). Así pues, considerando el esquema del display de 2 dígitos, si se quisiera encender por ejemplo el segmento A en el primer dígito, habría que emitir un 1 lógico por el pin 3 (enciende el segmento A) y un 0 lógico por el pin 10 (enciende el dígito 1).

En este proyecto se ha intentado mantener el realismo con respecto al *Overhead Panel* real. Por ello los displays de 2 y 3 dígitos serán de color verde y los de 5 dígitos de color blanco. Para implementar los dígitos del 0 al 9 en todos ellos, en el capítulo 7 se expondrá la tabla con el código 7 segmentos para displays de cátodo común que debe ser considerado a la hora de programar el Arduino.

### 5.5.7 Servomotor

El servomotor es el último tipo de componente de salida del *Overhead Panel*. Este está formado por un motor que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación, que en el caso de los servomotores adquiridos para este proyecto es de 0 a 180 grados, y mantenerse estable en esta posición.

Para conseguir esto se usa la modulación por ancho de pulsos (PWM), por lo tanto, deberán estar conectados directamente a los pines que tengan esta función en el Arduino (ver *ilustración 18*). Su programación es sencilla dado que se utiliza una librería incorporada en el Arduino IDE. Esta genera pulsos de una duración determinada para que el servomotor rote en sentido horario, antihorario o se mantenga neutro en una posición. Para que el giro sea horario los pulsos deben tener una duración de entre 1 a 1,4 milisegundos. Si duran 1,5 milisegundos el servomotor no girará. Por último, si se reciben pulsos de entre 1,6 a 2 milisegundos el servomotor rotará en sentido antihorario.

En este proyecto se utiliza un total de 10 servomotores para controlar nueve relojes. Adicionalmente se usa uno para el sistema de autorrotación de los selectores de encendido de motores (PANEL\_C1\_4). En algunos relojes el rango de giro requerido es de 270 grados, mientras que en otros se requiere un giro completo de 360 grados. Tal y como se verá en la sección 5.6, donde se explica el diseño de piezas 3D, se han impreso engranajes multiplicadores para transformar los giros de 180 grados en los anteriormente citados.

Los servomotores tienen tres cables. El rojo es el que se conecta a 5V, el negro o marrón a tierra y el naranja o amarillo el que va al pin PWM del Arduino. En el anexo, sección 11.10, puede consultarse el esquema eléctrico de la instalación (que utiliza del pin 2 al 11 del microcontrolador). A continuación, se expone el esquema simplificado.

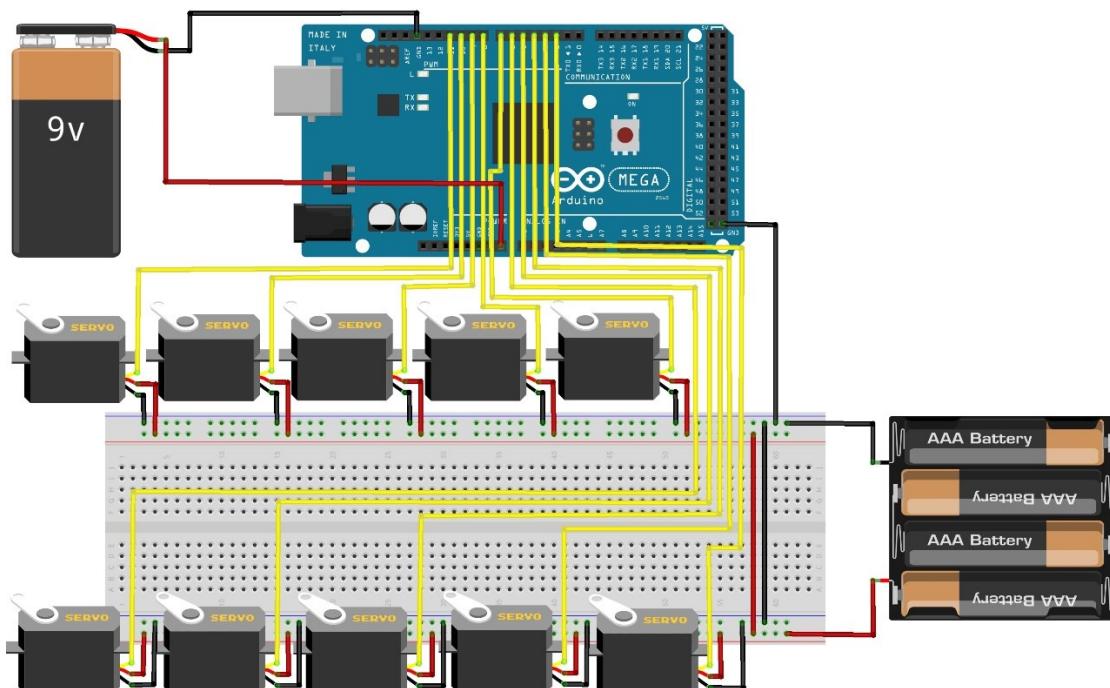


Ilustración 29. Esquema simplificado de la instalación de servomotores

### 5.5.8 Interruptor (2 y 3 posiciones)

Un interruptor es un mecanismo destinado a abrir o cerrar un circuito eléctrico. En el caso del *Overhead Panel* se trabaja con interruptores de dos y tres posiciones. Además, los de tres posiciones pueden tener una o dos posiciones momentáneas (MOM), lo que significa que cuando no se accionan vuelven por sí solos a la posición central.

Para explicar su uso se va a volver a recurrir a la *Ilustración 24* y la sección 11.6 del anexo. La instalación es bastante sencilla. Todos los interruptores estarán conectados al circuito de 5V de la PSU mediante una resistencia de  $220\ \Omega$ . Como se comentó previamente, lo que hace el IC 74HC165 es leer el estado del circuito a través de los pines de entrada (D0 a D7), generando un 0 lógico cuando el circuito está abierto y un 1 lógico cuando el circuito está cerrado.



Ilustración 30. Interruptor de 3 posiciones tipo ON/MOM - OFF - ON/MOM (Cockpit Sim Parts LTD 2017)

Tanto los interruptores de dos posiciones como los de tres tienen tres patillas. En el caso de los de tres posiciones, la tercera posición se configura mediante código. Cuando el interruptor está en la posición central ambos circuitos están abiertos, leyendo por tanto dos 0 lógicos. En las posiciones laterales se obtendrán un 0 lógico en un lado y un 1 lógico en el otro y viceversa. Como se verá en el capítulo 7, mediante una sentencia *if* se configuran los tres estados.

### 5.5.9 Pulsador tipo botón

Un pulsador de tipo botón no deja de ser un tipo de interruptor momentáneo, que cierra el circuito cuando es pulsado y lo abre cuando se suelta. Por tanto, su explicación de uso se puede extraer de la explicación del anterior apartado. Su conexión es muy similar a la del interruptor de dos posiciones y puede ser consultada en la *ilustración 24* y en la sección 11.6 del anexo.

Aunque no afecta a su configuración eléctrica, en el *Overhead Panel* para mantener el realismo se han usado botones de color negro y adicionalmente uno de color verde y otro rojo.



*Ilustración 31. Pulsadores MOM de tipo botón de color negro (Cockpit Sim Parts LTD 2017)*

### 5.5.10 Selector rotatorio (45 y 30 grados)

Un selector rotatorio es un interruptor cuyas posiciones, normalmente más de dos o tres, se consiguen mediante la rotación de su eje central. El número total de posiciones del selector rotatorio viene definido por el ángulo que haya entre cada posición consecutiva. En este proyecto se usarán selectores rotatorios de 45 y 30 grados, que ofrecen 8 y 12 posiciones consecutivamente.



Ilustración 32. Selector rotatorio de 45 grados (Cockpit Sim Parts 2017)

En la mayoría de ellos no se necesitan todas sus posiciones. Estas pueden ser bloqueadas mediante la arandela inferior de la anterior imagen. Esta se coloca en la base del eje y mediante esa pequeña pletina se bloquea el giro a partir de la posición donde es introducida. Aunque la conexión es muy similar a la de los interruptores se ofrece a continuación un esquema eléctrico a modo de ejemplo. Se considerará un selector de 30 grados (12 posiciones), donde solo se usan sus primeras tres posiciones, permaneciendo bloqueadas el resto. Al igual que se hizo con los interruptores de tres posiciones, se ahorrará el cable de un estado, al implementarlo posteriormente mediante código.

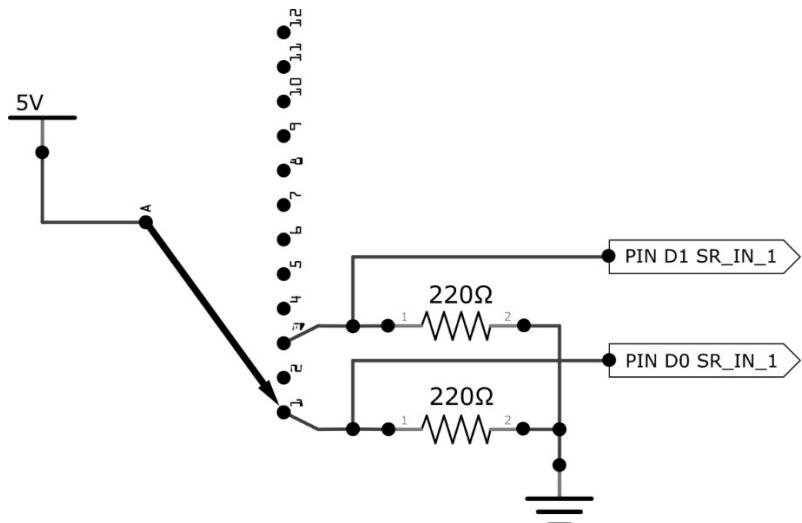


Ilustración 33. Esquema eléctrico de un selector rotatorio de 45 grados con tres posiciones

### 5.5.11 Codificador rotatorio

Un codificador rotatorio o *rotary encoder* es un tipo de sensor que se utiliza para determinar la posición angular de un eje giratorio. En el *Overhead Panel* se usan dos unidades para aumentar o disminuir las cifras que aparecen en los displays del PANEL\_C5\_4 (FLT ALT y LAND ALT). Su giro en sentido horario aumenta el número, mientras que un giro antihorario lo disminuye.



Ilustración 34. Codificadores rotatorios (Cockpit Sim Parts 2017)

Para explicar su funcionamiento se considerará la siguiente imagen, facilitada en la página web *How to mechatronics*.

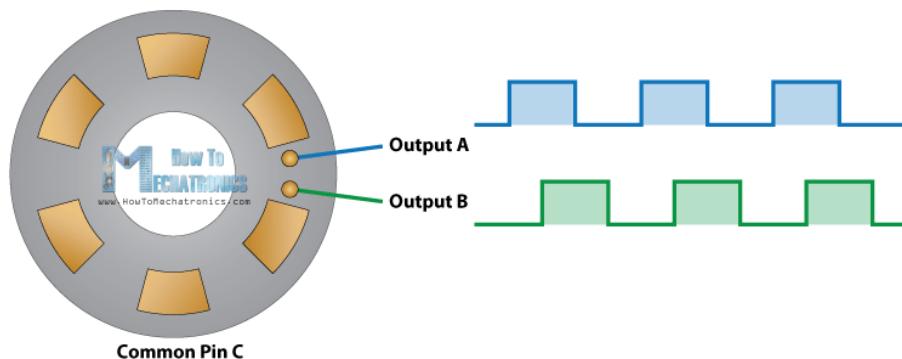


Ilustración 35. Funcionamiento interno del codificador rotatorio (How to mechatronics 2016)

El codificador rotatorio tiene una corona con partes conductoras (pin común C) y otras aislantes. En contacto con ella están dos pines, A y B. Cada vez que uno de los pines entra en contacto con la parte conductora se emite un pulso, como se ve en la parte derecha de la imagen. El conteo del número de pulsos nos indica cuantos grados ha girado el eje. Para identificar en qué sentido lo ha hecho se contemplará la siguiente imagen.

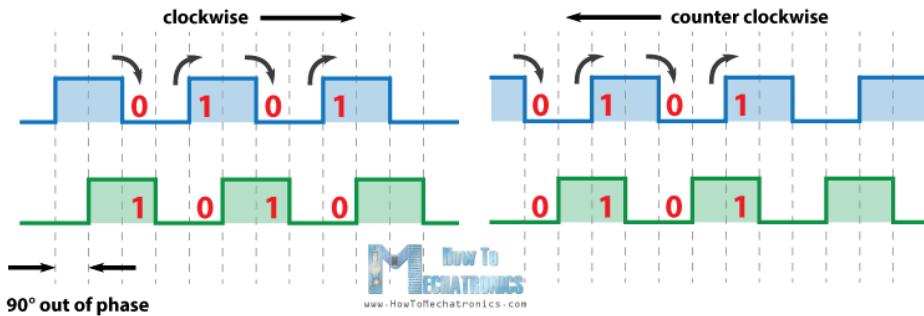


Ilustración 36. Señal emitida por el codificador rotatorio en función del sentido de giro (How to mechatronics 2016)

Las señales producidas entre los pines A y B están desfasadas entre ellas 90°. Cuando el codificador gira en sentido horario los valores lógicos consecutivos emitidos son opuestos. Por el contrario, cuando gira en sentido antihorario se emiten valores consecutivos iguales. La lectura y comparación de esos dos valores permite saber el sentido de la rotación. De esta manera se tiene definido los dos parámetros fundamentales de un codificador rotatorio, el ángulo girado y su sentido.

El codificador rotatorio tiene 5 pines. Uno de ellos (*SW*) se utiliza como pulsador momentáneo, ya que el eje puede ser presionado, haciendo de botón. En el *Overhead Panel* esta función no se utilizará, por tanto, se usarán los otros 4 pines. Para su correcto funcionamiento, y si no se dispone de un *rotary encoder* montado sobre una PCB prefabricada, como suelen proporcionarse en los kits de Arduino, es necesario realizar un pequeño circuito soldando dos resistencias de valor 10kΩ a dos de sus pines. En el anexo, sección 11.11, se puede consultar el circuito eléctrico. En este caso los pines CLK y DT (o pines A y B como aparecían en la ilustración 35), van directamente conectados a dos pines digitales del Arduino. Como se ha realizado en otras ocasiones, a continuación, se muestra un esquema simplificado.

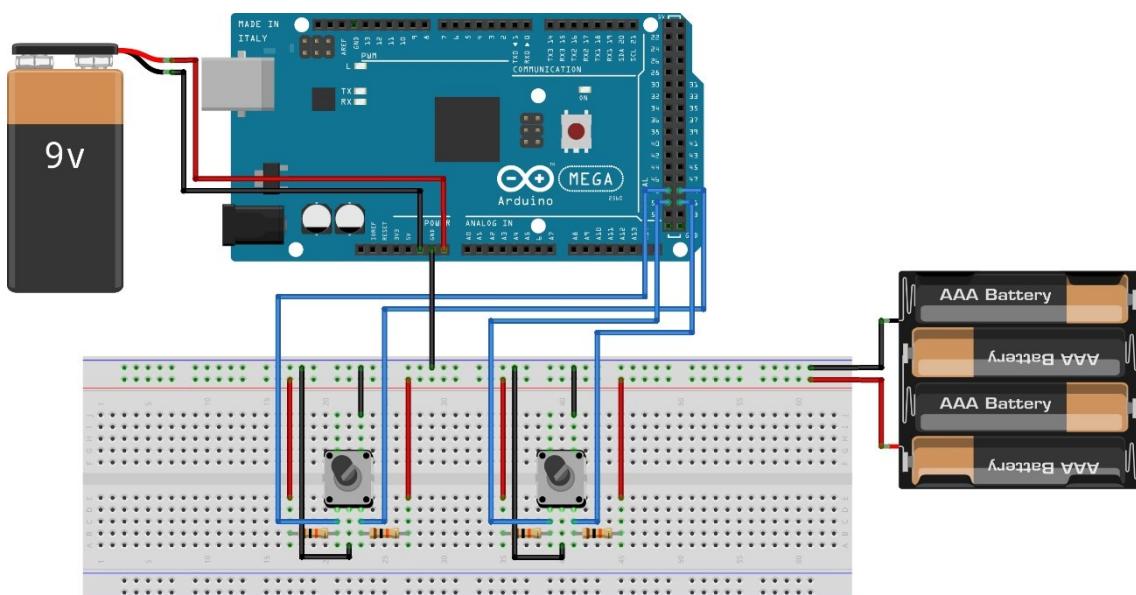


Ilustración 37. Esquema simplificado de la instalación de codificadores rotatorios

### 5.5.12 Potenciómetro

Un potenciómetro es una resistencia variable cuyo valor es controlado mediante el giro de su eje. En el *Overhead Panel* se utilizan cinco unidades de tipo lineal de resistencia máxima 10 kΩ, dos para el control de la retroiluminación y tres para el ajuste de la temperatura de la cabina.



Ilustración 38. Potenciómetro lineal 10 kΩ marca ALPHA (Cockpit Sim Parts 2017)

Un potenciómetro tiene tres pines. Los dos laterales son para conectar a 5V y a tierra. Este admite dos posibilidades de conexión, en función de si interesa que el valor máximo de resistencia se alcance con un giro horario o antihorario. En el caso del *Overhead Panel*, y apoyando esta decisión en los potenciómetros del PANEL\_C3\_1, es necesario que cuando el eje haya rotado al máximo de su recorrido en sentido antihorario ofrezca la máxima resistencia. De esta manera, la intensidad que circulará a través del pin central será mínima y por tanto se podrá simular la posición OFF de la retroiluminación. Para ello, el pin izquierdo estará conectado a tierra y el derecho a 5V.

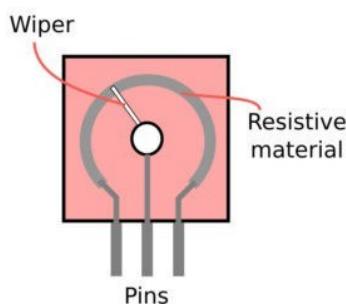


Ilustración 39. Esquema del interior de un potenciómetro (Build electronic circuits 2016)

El pin central irá conectado a uno de los puertos analógicos del Arduino, que convierte la lectura analógica en una señal digital. Estos puertos tienen una resolución de 10 bits y retornan un valor entero de 0 a 1023 en función de la intensidad que circula a través de ellos.

Para el uso de potenciómetros de este estilo es necesario tener una serie de precauciones. Como se comentó previamente, debido a la imposibilidad del Arduino de proporcionar toda la energía requerida por la instalación del *Overhead Panel*, se ha usado una fuente de alimentación externa capaz de proporcionar hasta un máximo de 5V a 15A. Este valor es demasiado alto para el potenciómetro y para el Arduino a través de sus pines analógicos. Por ello, los potenciómetros son los únicos elementos de todo el proyecto conectados a los 5V proporcionados por el microcontrolador, siendo su amperaje del orden de los miliamperios.

En el anexo, sección 11.12, puede consultarse el esquema eléctrico de la instalación de los potenciómetros. A continuación, se ofrece el esquema simplificado del anterior.

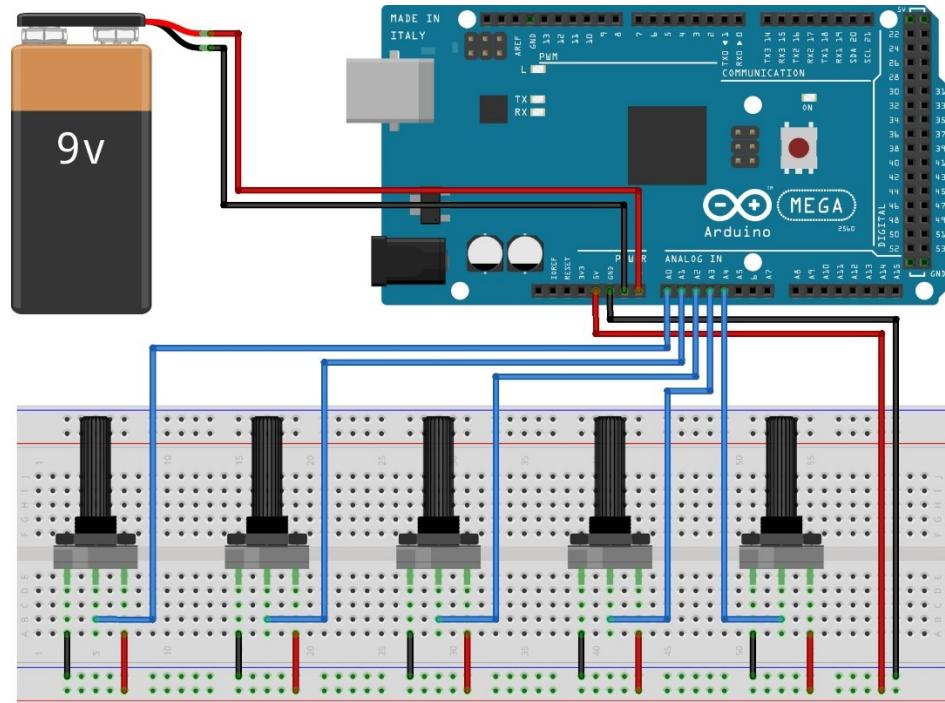


Ilustración 40. Esquema simplificado de la instalación de potenciómetros

### 5.5.13 Transistor IRF1404 MOSFET y sistema de retroiluminación

En este último apartado de la sección de electrónica se va a explicar cómo se ha diseñado el sistema de retroiluminación. Este está unido a la trasera del bastidor inferior de madera como se vio en la sección 5.3 y está compuesto por 24 tiras LED de luz blanca. Para su funcionamiento, las tiras deben estar conectadas a 12V. Por ello, se aprovecha el circuito de 12V a 16A de la PSU.

Por cada columna del panel se colocan dos tiras LED para así maximizar la luz que pasa a través de los paneles translúcidos. Como se comentó previamente, por cuestiones de redundancia, tanto las tiras de cada columna, como dentro de la misma columna, están conectadas entre ellas en configuración paralelo. Además, dentro de una tira LED, los diodos están también en paralelo, de manera que, si se funde uno, todos los demás siguen emitiendo luz. A continuación, se ofrece un esquema eléctrico de la instalación a modo de ejemplo. Para simplificarlo solo se han representado las dos primeras columnas del *Overhead Panel*.

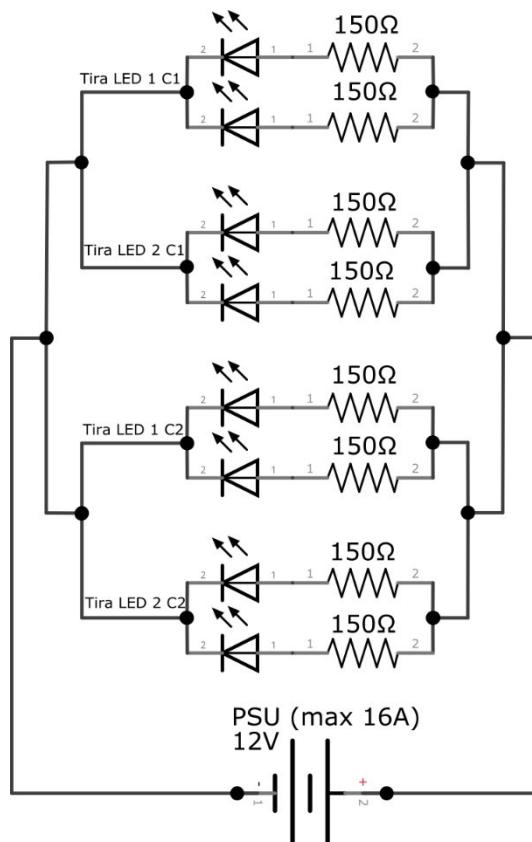


Ilustración 41. Esquema eléctrico de la configuración paralelo de la instalación de retroiluminación

Si el esquema eléctrico final fuera el de la anterior imagen, no se tendría control sobre la intensidad lumínica de las tiras LED, como si se tiene en el *Overhead Panel* real. Aquí es donde entra uno de los dos potenciómetros del PANEL\_C3\_1, encargado de ajustar la cantidad de luz emitida. El problema que surge a continuación es la imposibilidad de que este elemento aguante los 16A que circulan por el circuito de 12V. Es necesario, por tanto, hacer uso de un transistor IRF1404 MOSFET canal-N.

El cometido del transistor MOSFET canal-N consiste en regular la intensidad que pasa a las tiras LED a partir de una señal PWM proveniente de uno de los pines digitales del Arduino. Esta señal, ahora sí, está controlada por el potenciómetro previamente mencionado, solucionando así el problema del exceso de corriente a través de él.

Un transistor MOSFET tiene tres patillas, de izquierda a derecha, compuerta (G), drenaje (D) y surtidor (S). De forma general y simplificada, cuando en la compuerta se le aplica un voltaje positivo (voltaje negativo en el caso de los MOSFET canal-P), el transistor deja pasar corriente entre el drenaje y el surtidor. Cuando no recibe voltaje, este flujo se interrumpe. La cantidad de intensidad que circula se controla con el valor de voltaje, comprendido entre 0V y 5V en el caso de Arduino.

Por ello, el modelo IRF1404 es el elegido, ya que, según sus especificaciones técnicas, la compuerta se activa a partir de 4V, permitiendo un flujo máximo entre el drenaje y el surtidor de 40V a 162A, siendo superior al que realmente será aplicado (12V a 16A).

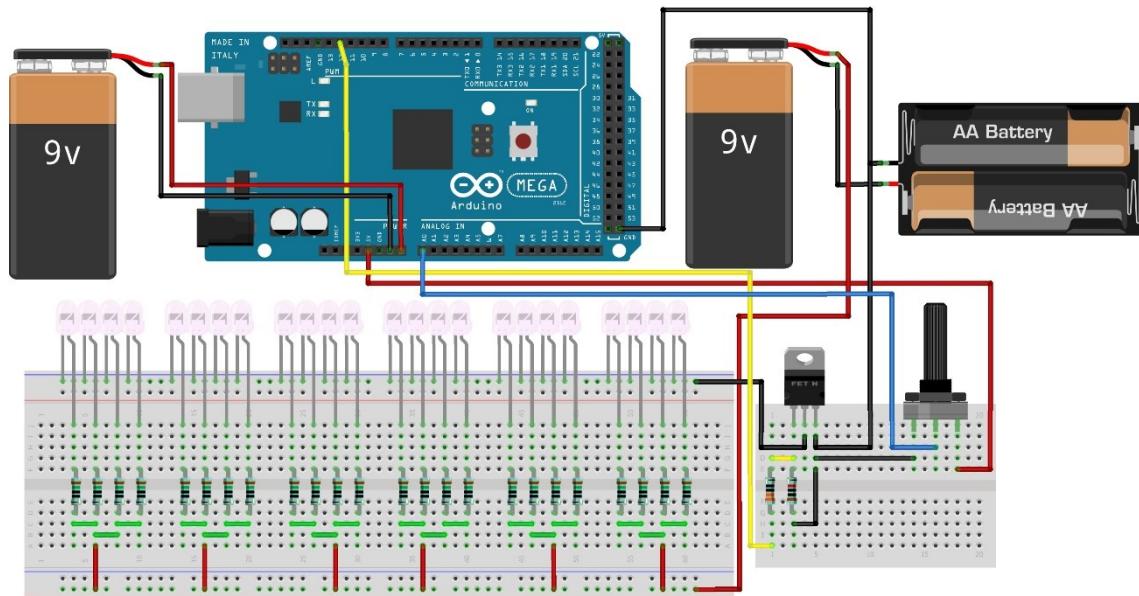
Dado que el transistor estará soportando una intensidad máxima de 16A, puede alcanzar una alta temperatura, por ello se une a un disipador de calor.



Ilustración 42. Transistor IRF1404 MOSFET canal-N y disipador de calor

Adicionalmente, con el transistor MOSFET es recomendable usar dos resistencias. La primera, de  $330\Omega$ , se conecta entre el pin del Arduino y la compuerta y previene de una sobreoscilación cuando se pasa de apagado a encendido. De esta manera el efecto de ir subiendo la intensidad lumínica es más progresivo, además de proteger la compuerta del transistor. Por otro lado, también se usa una resistencia de  $10k\Omega$  conectada entre la compuerta y tierra. La función de esta es evitar la activación de la compuerta por posibles cargas estáticas.

En el anexo, sección 11.13, puede consultarse la instalación eléctrica final de la retroiluminación del *Overhead Panel*, incluyendo el transistor MOSFET, el potenciómetro y las tiras LED. Para reducir el tamaño de la representación, las tiras LED aparecerán representadas por un solo led y una resistencia. A continuación, se expone el esquema simplificado, donde las dos pilas de 9V y 3V conectadas en serie simulan en circuito a 12V de la PSU.



*Ilustración 43. Esquema simplificado de la instalación de retroiluminación*

## 5.6 Piezas 3D

Uno de los requisitos del proyecto consiste en intentar conseguir un producto final lo más económico posible. Por ello, se ha usado la impresión 3D para evitar comprar todos los embellecedores de interruptores, selectores y codificadores rotatorios, además de todos los engranajes y estructuras traseras de los relojes. Algunos modelos se encuentran disponibles en Internet, otros, aunque también lo están, tuvieron que ser modificados para encajar con las dimensiones de las piezas del fabricante Cockpit Sim Parts LTD. No obstante, en la mayoría de los casos tuvieron que diseñarse desde cero, haciendo uso del programa gratuito FreeCAD. Todos ellos se podrán descargar del repositorio localizado en el anexo, sección 11.16.

Como es entendible, aunque existen modelos económicos en el mercado, la adquisición de una impresora 3D encarecería el coste del proyecto. Por ello, exclusivamente se ha comprado el rollo de 1 kg de filamento PLA blanco. Para la impresión de las piezas se ha usado la impresora modelo *Creality Ender 3* del Club de Vuelo UPM (Akaflieg Madrid), una asociación de estudiantes de la Universidad Politécnica de Madrid que dedica su tiempo al vuelo a vela y a la realización de proyectos de ingeniería aeroespacial, siendo su gran apuesta a futuro la construcción de un planeador propio.

Para imprimir piezas en 3D se deben diseñar con FreeCAD (o cualquier otro software equivalente) y exportarlas en formato STL (Standard Triangle Language), un lenguaje de diseño asistido por computadora. Una vez hecho esto, con el programa Ultimaker CURA, se genera un archivo con formato GCODE, un lenguaje de programación de control numérico que incluye los ajustes de impresión y la trayectoria que debe seguir el cabezal. Este es el archivo que lee la impresora y que permite la impresión de la pieza en 3D.

Como se comentó previamente, el material elegido es PLA por su facilidad a la hora de imprimir, puesto que otros materiales, como el ABS o el PETG, necesitan tener un entorno de impresión más controlado, como por ejemplo el que se consigue en impresoras 3D con una zona de impresión cerrada. El color blanco es elegido para evitar tener que pintar después la mayoría de las piezas.

En esta sección se mostrarán imágenes del diseño de las piezas que serán posteriormente impresas. En las ilustraciones se omitirán las medidas, ya que se proporcionan directamente los archivos de diseño en el repositorio del anexo. Se considerarán las siguientes piezas:

1. Embellecedores de los interruptores (*toggle switch caps*)
2. Embellecedores de los selectores rotatorios
3. Embellecedores de los codificadores rotatorios
4. Embellecedores de las luces de aterrizaje
5. Estructura y engranajes multiplicadores de los relojes
6. Mecanismo de autorrotación de los selectores de encendido de motor

### 5.6.1 Embellecedores de los interruptores (*toggle switch caps*)

Algunos interruptores, en concreto aquellos de los cuales se debe tirar hacia atrás para cambiar su posición, tienen unas piezas de goma de color blanco en las puntas que sirven para agarrarlos mejor. Aunque esta característica no está implementada en este *Overead Panel* por el elevado precio que tienen este tipo de interruptores, se han diseñado e impreso estos embellecedores para mantener la mayor similitud visual posible con el real.

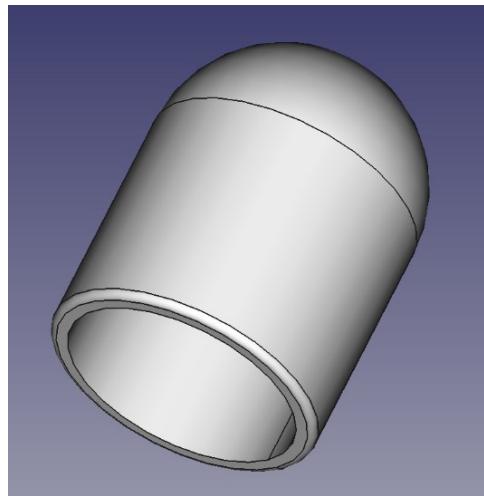


Ilustración 44. Modelo 3D del embellecedor de interruptores

### 5.6.2 Embellecedores de los selectores rotatorios

Estos embellecedores cumplen la función de permitir un buen agarre a la hora de rotar los selectores. En este caso, se ha utilizado el archivo creado por un usuario de la página web de modelos STL *Thingiverse*. El enlace de descarga se podrá consultar en el repositorio del anexo. Estas piezas serán posteriormente decoradas con pegatinas, también subidas al repositorio.

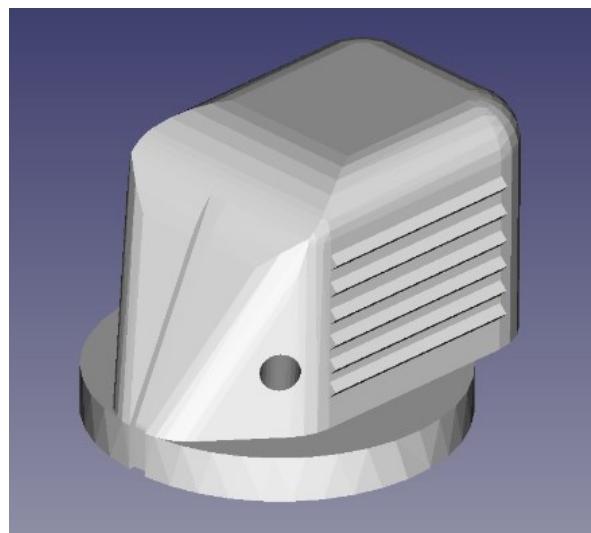


Ilustración 45. Modelo 3D del embellecedor de selectores rotatorios

Adicionalmente a los anteriores, hay un embellecedor que ha sido necesario diseñar. Se trata del selector *cross feed* situado en el PANEL\_C1\_3. El surco horizontal que se encuentra en la parte superior permite colocar posteriormente la pegatina decorativa.

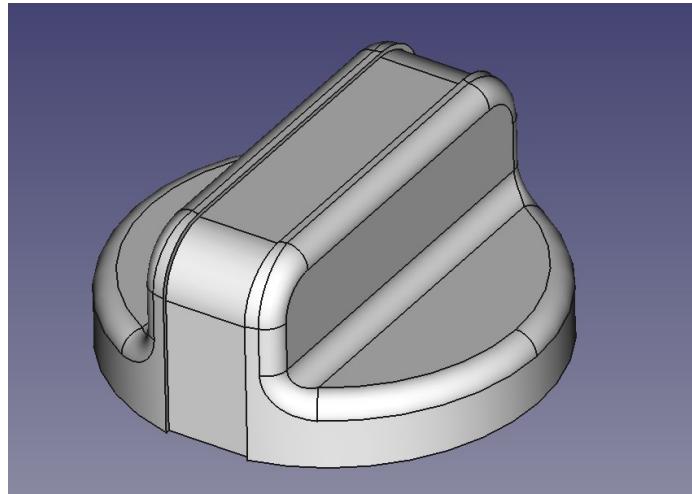


Ilustración 46. Modelo 3D del embellecedor del selector giratorio cross feed

### 5.6.3 Embellecedores de los codificadores rotatorios

Los embellecedores de los codificadores han sido creados por otro usuario de *Thingiverse*. Aunque son parecidos, el embellecedor del codificador rotatorio LAND ALT del PANEL\_C5\_4 tiene un saliente vertical que lo diferencia del anterior, el cual ha sido necesario añadir, pues el original no lo incluía.

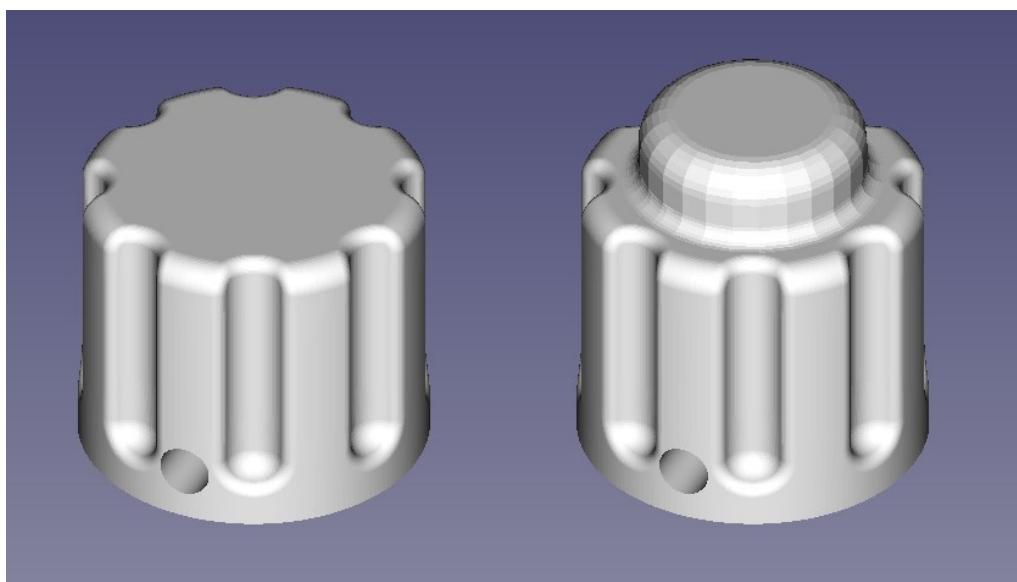


Ilustración 47. Modelo 3D de los embellecedores de los codificadores rotatorios

#### 5.6.4 Embellecedores de las luces de aterrizaje

Los embellecedores de las luces de aterrizaje de Boeing tienen una forma bastante característica. Estos intentan representar visualmente los faros del avión. Las piezas han sido replicadas a partir de geometrías simples como cilindros, esferas y prismas.

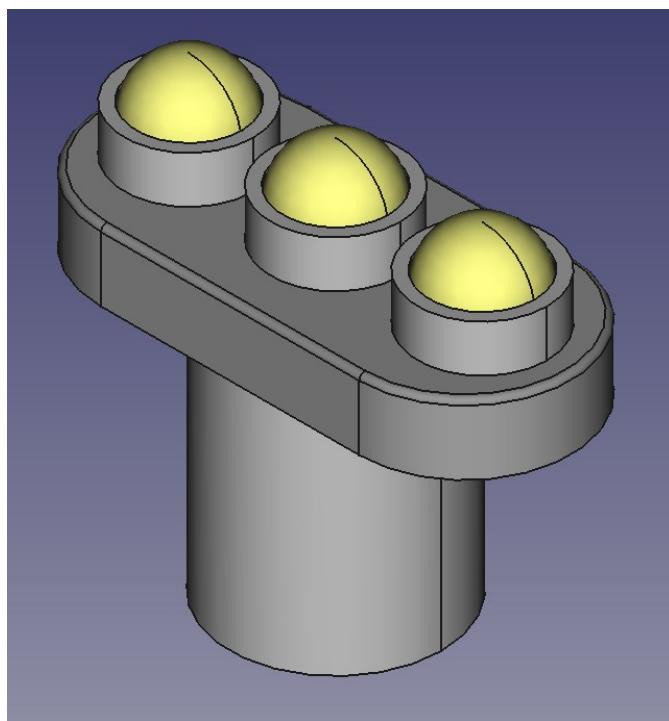


Ilustración 48. Modelo 3D del embellecedor de las luces de aterrizaje

#### 5.6.5 Estructura y engranajes multiplicadores de los relojes

Como se comentó en anteriores apartados, los servomotores utilizados solo tienen un intervalo de giro de 0 a 180 grados. Sin embargo, la mayoría de los relojes necesitan un rango de 270 o 360 grados. Por ello, será necesario usar engranajes multiplicadores.

Un usuario de la página web *Thingiverse* diseñó una estructura para el reloj del PANEL\_C5\_3 (*duct pressure gauge*), así como los engranajes para permitir un giro de 270 grados. No obstante, dado que el fabricante es diferente, la estructura no se ajusta correctamente, de manera que ha sido ligeramente modificada. La versión original, así como todos los diseños personalizados para cada reloj se podrán consultar en el repositorio del anexo.

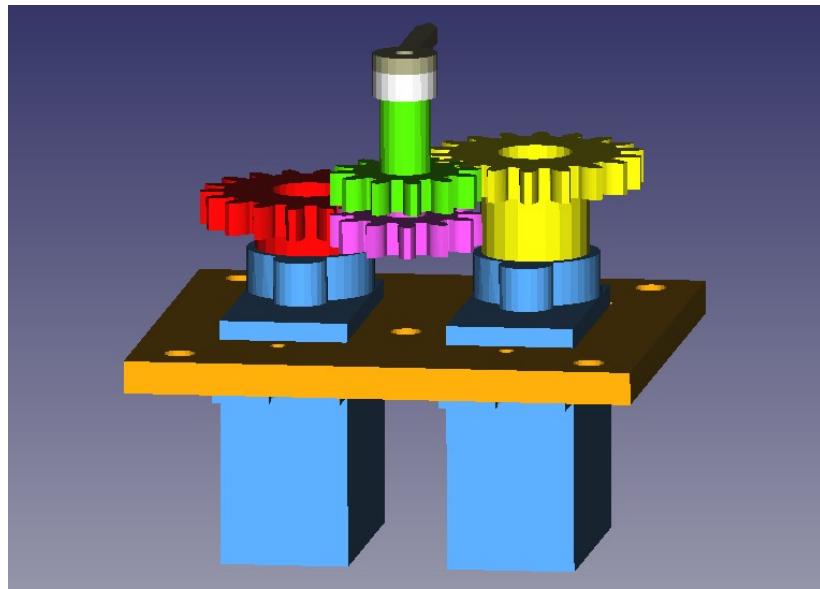


Ilustración 49. Modelo 3D de los engranajes multiplicadores de los relojes

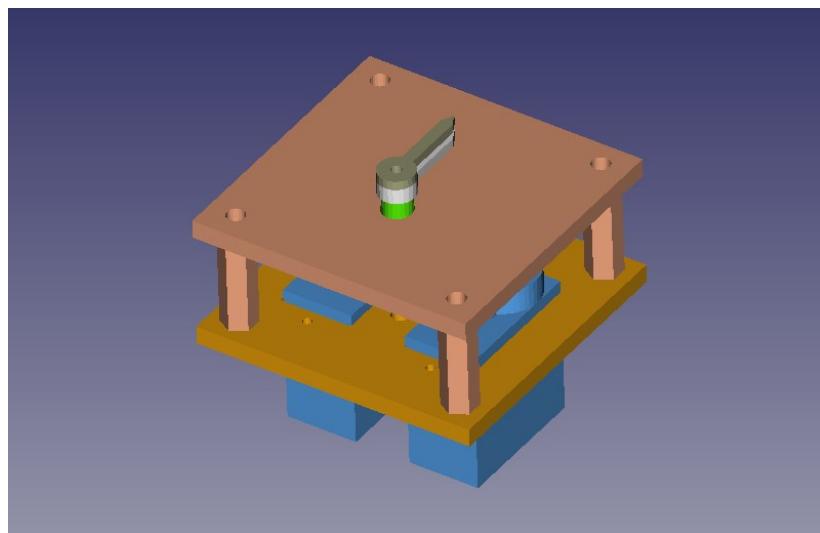


Ilustración 50. Modelo 3D de la estructura de los relojes

A la hora de diseñar engranajes en FreeCAD hay que definir un parámetro denominado módulo del engranaje. Este es la relación entre el diámetro primitivo del engranaje en milímetros y el número de dientes, de manera que, para que dos piezas engranen, deben tener el mismo módulo.

El número de dientes viene definido por la relación de ángulos que se quiere conseguir. El servomotor estará conectado a un engranaje de 18 dientes. De esta manera, al ser engranajes multiplicadores, los relojes que tienen 270 grados tendrán engranajes de 12 dientes y los de 360 grados 9 dientes. En aquellos relojes donde haya dos agujas, los engranajes serán concéntricos, como se vio en la *ilustración 49*.

Adicionalmente, para el reloj del PANEL\_C5\_4 (*manual valve gauge*), que tiene un giro menor a 180 grados, no es necesario el uso de engranajes multiplicadores. Por tanto, se ha diseñado el siguiente mecanismo que transmite el giro de forma directa.

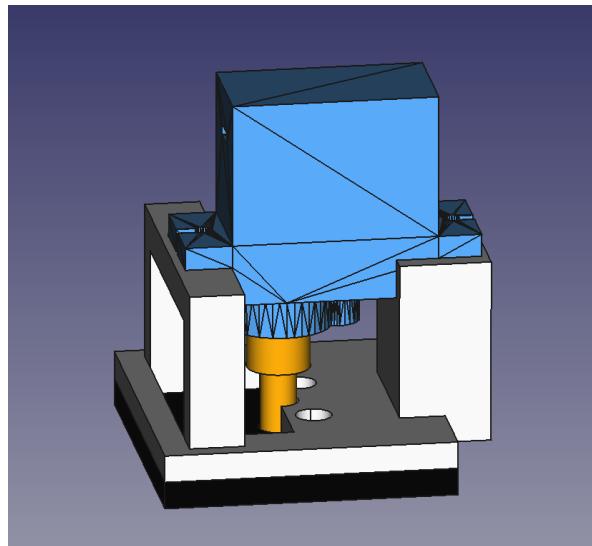


Ilustración 51. Modelo 3D de la estructura del reloj manual valve

#### 5.5.6 Mecanismo de autorrotación de los selectores de encendido de motor

Los selectores del PANEL\_C1\_4 tienen cuatro posiciones. Para iniciar el procedimiento de encendido de los motores los selectores se deben de llevar a la posición GRD (primera posición por la izquierda). Una vez encendido, el selector vuelve automáticamente a la posición OFF. Para implementar esta característica se ha tenido que diseñar un mecanismo que hace uso de un servomotor.

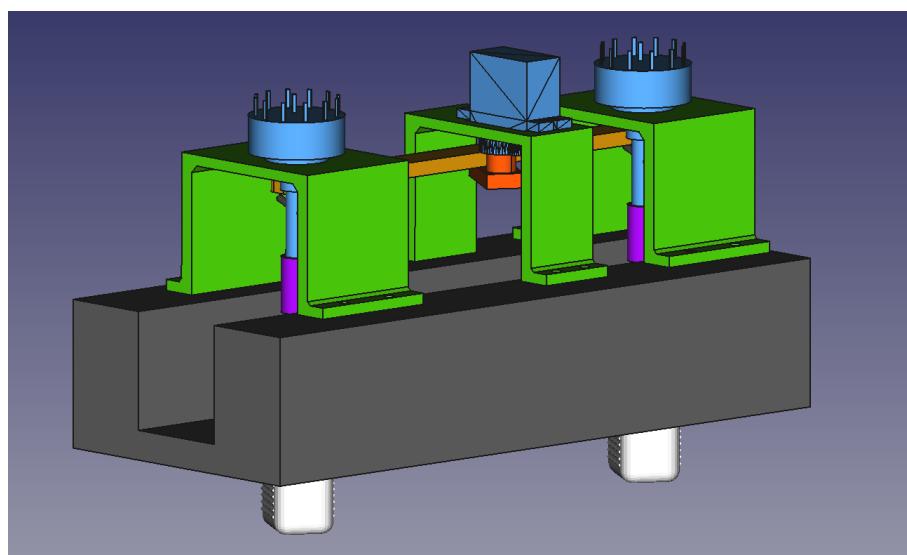


Ilustración 52. Modelo 3D del mecanismo de autorrotación de los selectores de encendido de motor

El servomotor, localizado en el centro del mecanismo y apoyado en el bastidor superior de madera mediante una estructura impresa en 3D, se acopla a un sistema de levas que empuja a una pequeña varilla metálica que atraviesa cada selector. De esta manera, se consigue devolver el selector a su posición OFF cuando el servomotor es accionado. Aunque el servomotor adquirido no es muy potente, no pudiendo mover ambos selectores a la vez, en los aviones reales no se enciende el segundo motor hasta que el primero ya ha encendido. Por ello, el servomotor elegido es apto para este proyecto. A continuación, se muestran dos imágenes ocultando el bastidor, de manera que se pueda apreciar mejor el mecanismo en sus dos posiciones.

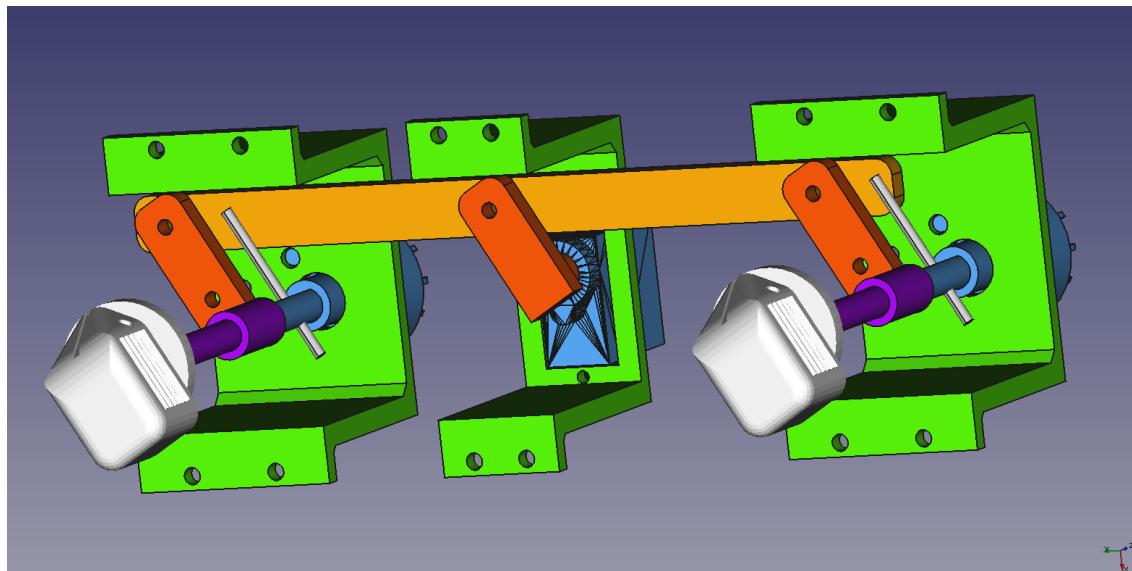


Ilustración 53. Modelo 3D de los selectores de encendido de motores en posición GRD

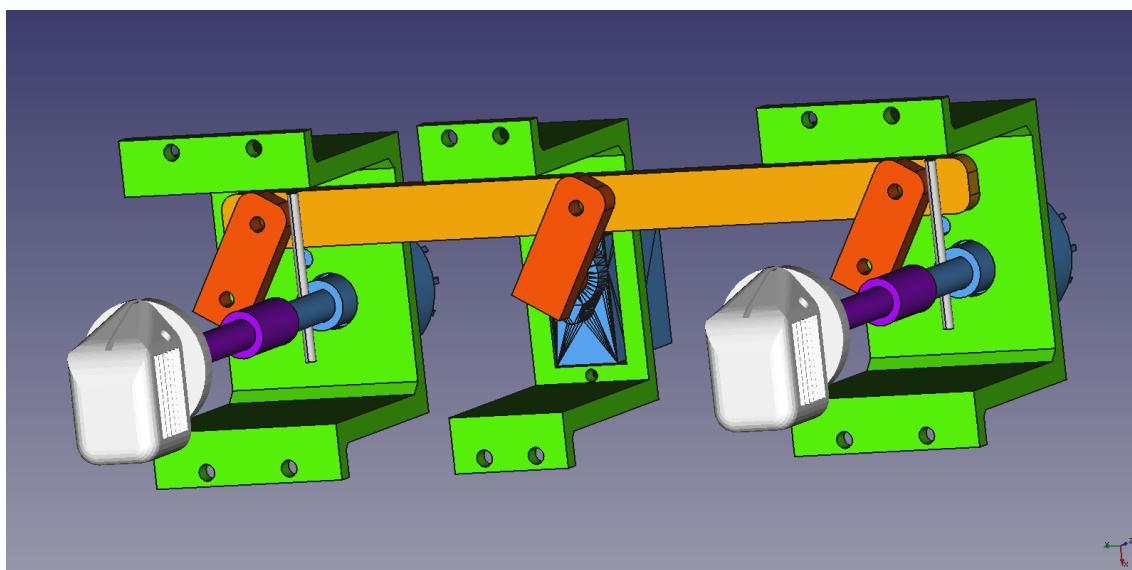


Ilustración 54. Modelo 3D de los selectores de encendido de motores en posición OFF

## 6. Construcción del *Overhead Panel*

Una vez concluida la fase de diseño del proyecto, cumpliendo con todos los requisitos impuestos en las consideraciones iniciales, se procede a la construcción del *Overhead Panel*. Es necesario, durante esta fase, ir verificando que se satisfacen los parámetros del diseño definitivo adoptado.

Este capítulo se divide en las siguientes secciones, dispuestas en el orden cronológico en el que se fueron realizando:

1. Bastidor de madera
2. Soporte metálico del *Overhead Panel*
3. Paneles
4. Piezas 3D
5. Electrónica
6. Sistema de retroiluminación
7. Unión de los bastidores de madera

## 6.1 Bastidor de madera

El proceso de construcción empieza con el bastidor de madera, como también se empezó en la fase de diseño. El primer paso consiste en coger los tablones de madera de ramín y pasarlo a grueso. Con esto se busca cepillar todas las caras, eliminando la corteza exterior, saneando la pieza, además de escuadrarlas y reducirlas al espesor buscado según el diseño.



Ilustración 55. Regruesadora de carpintería cepillando los tablones de ramín

A continuación, se toman medidas de los paneles del *Overhead Panel* y, haciendo uso de una circular ingletadora, se cortan los tablones según las dimensiones adquiridas. Es importante que, de cara al siguiente paso, las zonas que se vayan a encolar y formen entre ellas un ángulo recto se corten a 45 grados para una mejor unión.



Ilustración 56. Tablones del bastidor inferior cortados según medidas tomadas

Una vez aplicada la cola blanca, se presionan las uniones con tornillos de apriete y se deja un tiempo de secado no inferior a 12 horas para que endurezca.



Ilustración 57. Bastidor inferior durante el endurecimiento de la cola blanca

Mientras tanto, tomando las medias exteriores del bastidor inferior y la distancia entre los paneles de diferentes columnas, se cortan los tablones para el bastidor superior y se encolan. A continuación, se muestra el resultado final de ambas piezas tras el proceso de endurecimiento.



Ilustración 58. Bastidor superior de madera encolado



*Ilustración 59. Bastidor inferior de madera encolado*

Es conveniente, una vez listas ambas partes, comprobar que casan bien en su posición, superponiéndolas una sobre otra. En el caso de no coincidir, se repasan las piezas.



*Ilustración 60. Unión de ambas partes del bastidor de madera*



Ilustración 61. Capa inferior de paneles superpuestos sobre el bastidor de madera

En el siguiente paso, se procede al lacado del bastidor en el color gris de Boeing, que se corresponde con el RAL 7011. Previo a aplicar la laca, es necesario administrar una capa de imprimación para cerrar los posibles poros que tiene la madera y posteriormente lijárla.

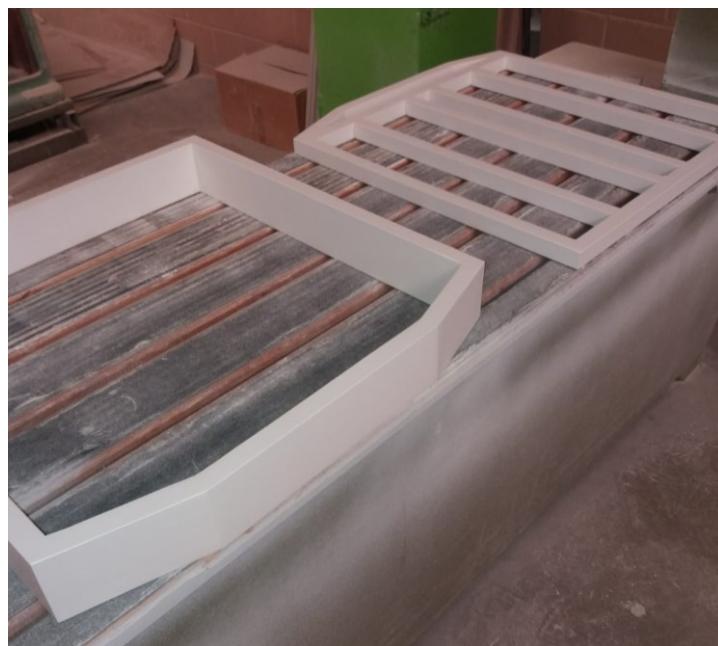


Ilustración 62. Imprimación del bastidor de madera previa al lacado

Una vez terminado el lacado del bastidor, se procede al fresado por control numérico de los cantos interiores del bastidor inferior. Con ello se busca facilitar el cierre cuando la electrónica esté instalada, siendo este el último paso de la construcción del bastidor de madera.



Ilustración 63. Configuración previa de la fresadora por control numérico



Ilustración 64. Aspecto del bastidor inferior después de la primera pasada de la fresadora

## 6.2 Soporte metálico del *Overhead Panel*

El primer paso para construir el soporte metálico consiste en cortar con una ingletadora metálica los tubos previamente adquiridos según las medidas expuestas en el plano del anexo, sección 11.3.

Posteriormente se procede a unirlos mediante soldadura por arco, usando para ello un grupo electrógeno, compuesto por una fuente de alimentación y un electrodo. Este tipo de soldadura proporciona una unión robusta, capaz de soportar las fuerzas transmitidas una vez el panel ha sido colgado.

El resultado final se expone en la siguiente imagen, en la que se está comprobando que las medidas de diseño satisfacen la altura final con respecto a la silla que se utiliza en el simulador. Para unir el bastidor inferior con el soporte metálico se han usado tornillos rosca madera de 4.5x70 mm.



Ilustración 65. Bastidor inferior unido al soporte metálico del *Overhead Panel*

### 6.3 Paneles

Para comenzar con la instalación del kit de paneles de Cockpit Sim Parts LTD, la primera capa de estos es unida de forma provisional al bastidor superior con cinta adhesiva. Este kit viene con dos tipos de tornillos. Unos de rosca madera embellecidos con una pequeña pieza cilíndrica típica de los *Overhead Panel* de Boeing (*dzus*) y unos de rosca metal. Estos últimos se usan con las tuercas y arandelas provistas en el kit. Para su instalación es necesario perforar el bastidor superior tal y como se ve en la siguiente imagen.



Ilustración 66. Bastidor superior perforado

A continuación, es necesario ir labrando manualmente la madera con un formón en aquellos puntos donde se vea que los orificios de los paneles de la primera capa están tapados por el bastidor. La función de estos es la de dejar pasar la luz de la retroiluminación. También con ello se busca hacer hueco para la electrónica que está pegada al marco. En la siguiente imagen se puede comprobar cómo se está realizando esta tarea en toda la primera columna.



Ilustración 67. Labrado de la primera columna del Overhead Panel

Adicionalmente se puede ir comprobando a la vez si la segunda capa de paneles va también encajando.



Ilustración 68. Capa superior del PANEL\_C1\_4 colocada

Cuando se ha labrado todo el bastidor superior, la primera capa de paneles, así como los interruptores, selectores y demás componentes pueden ser perfectamente encajados. Son estos paneles los que se fijan con los tornillos de rosca madera. A continuación, se aprecia cómo quedan una vez colocados.



Ilustración 69. Componentes y primera capa de paneles colocados

Finalmente, tras acabar este proceso se puede colocar la segunda fila de paneles, más estéticos que los anteriores, haciendo uso de los tornillos de rosca metal. Estos paneles son de un material translúcido que deja pasar la luz solamente por las zonas que no han sido pintadas. En las siguientes dos ilustraciones se aprecia esta característica.

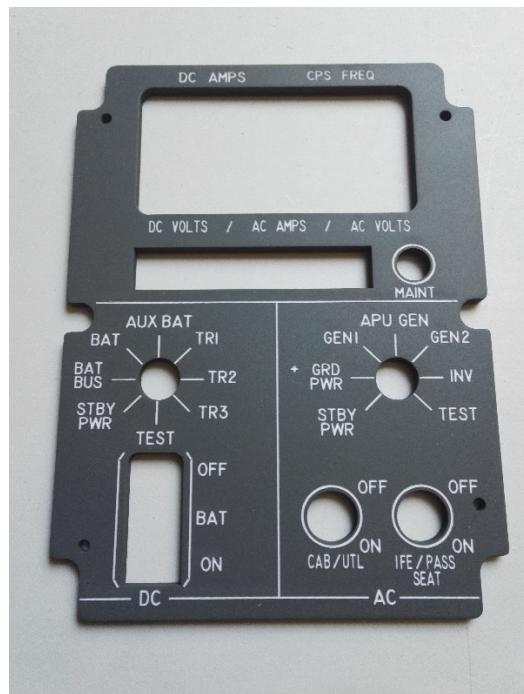


Ilustración 70. Delantera del PANEL\_C2\_1



Ilustración 71. Trasera del PANEL\_C2\_1

Tras el montaje de la segunda fila de paneles, los relojes, las cajas de anunciantes (cuyos letreros son fijados con adhesivo líquido), algunas de las piezas impresas en 3D (unidas mediante pegamento termofusible) y el resto de los elementos del kit de Cockpit Sim Parts LTD, el resultado final de esta sección es el mostrado en la siguiente imagen.



Ilustración 72. Delantera del Overhead Panel montada

## 6.4 Piezas 3D

Una de las tareas más complicadas a la hora de imprimir en 3D es conseguir en el programa Ultimaker CURA un perfil con un buen ajuste de los parámetros de impresión. Estos dependen del material que se imprime y la marca de la impresora. En este caso se va a imprimir PLA con la impresora *Creatlity Ender 3*. Tras realizar varias pruebas previas, se ha llegado a los siguientes ajustes, los cuales se han utilizado durante todas las impresiones.

Tabla 7. Perfil usado en CURA para la impresión de PLA con la impresora Creatlity Ender 3

CATEGORÍA	PARÁMETRO	VALOR
Calidad	Altura de capa	0.15 – 0.2 mm
	Altura de capa inicial	0.2 mm
	Ancho de línea	0.4 mm
Perímetro	Grosor de la pared	1 mm
	Recuento de líneas de pared	2
	Grosor superior/inferior	0.8 mm
	Patrón superior/inferior	Líneas
	Expansión horizontal	0 mm
Relleno	Densidad de relleno	30 – 100 %
	Patrón de relleno	Rejilla
	Relleno antes que las paredes	✓
Material	Temperatura de impresión	205 °C
	Temperatura de la cama de impresión	60 °C
	Flujo	100 %
	Habilitar la retracción	✓
	Retracción en el cambio de capa	✗
	Distancia de retracción	7 mm
	Velocidad de retracción	20 mm/s
Velocidad	Velocidad de impresión	60 mm/s
	Velocidad de relleno	60 mm/s
	Velocidad de pared	30 mm/s
	Velocidad de desplazamientos	70 mm/s
	Velocidad de capa inicial	30 mm/s
Soportes (si se requieren)	Generar soportes	✓
	Colocación del soporte	Tocando la base
	Ángulo de voladizo del soporte	60°
	Patrón del soporte	Triángulos
	Densidad del soporte	5 %
	Distancia en Z del soporte	0.1 mm
	Distancia X/Y de soporte	0.7 mm
	Habilitar interfaz de soporte	✓
Adherencia a la placa de impresión	Tipo adherencia de la placa de impresión	Falda
	Recuento de líneas de falda	3

Como se ve en la categoría de relleno, la densidad varía entre los valores 30% y 100%. Esto se debe a que no todas las piezas requieren ser macizas. Solamente se imprimirán con una densidad de relleno del 100% aquellas piezas que no vayan a ser estáticas y, por tanto, deben tener unas propiedades mecánicas diferentes. Esto es el caso, por ejemplo, de los engranajes y las levas. Otro parámetro variable es el de la altura de capa, en el apartado de calidad. El ajuste de 0.15 mm ha sido usado en elementos de carácter estético, como aquellos que van en la parte frontal, mientras que todas las piezas 3D del interior se han impreso a 0.2 mm.

A continuación, se ofrecen imágenes del resultado final de algunas de las impresiones. A pesar de que la *Creality Ender 3* es una impresora de bajo coste, la calidad que ofrece es bastante elevada.

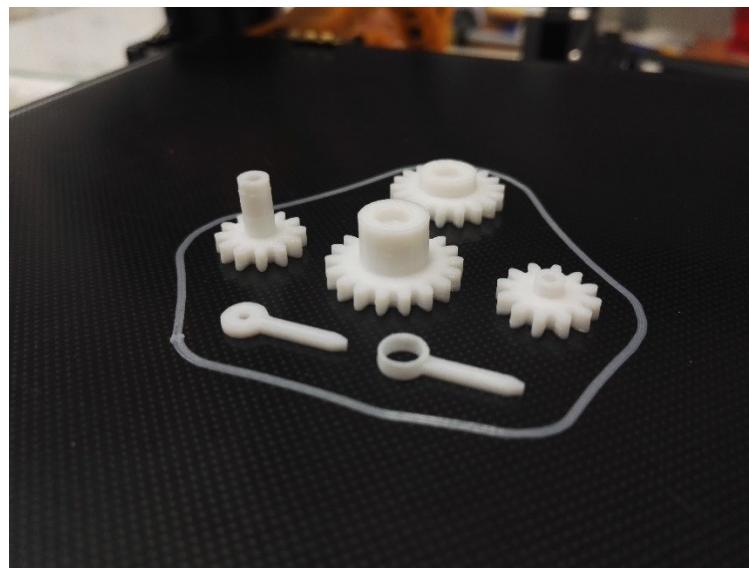


Ilustración 73. Resultado de la impresión 3D de los engranajes de los relojes

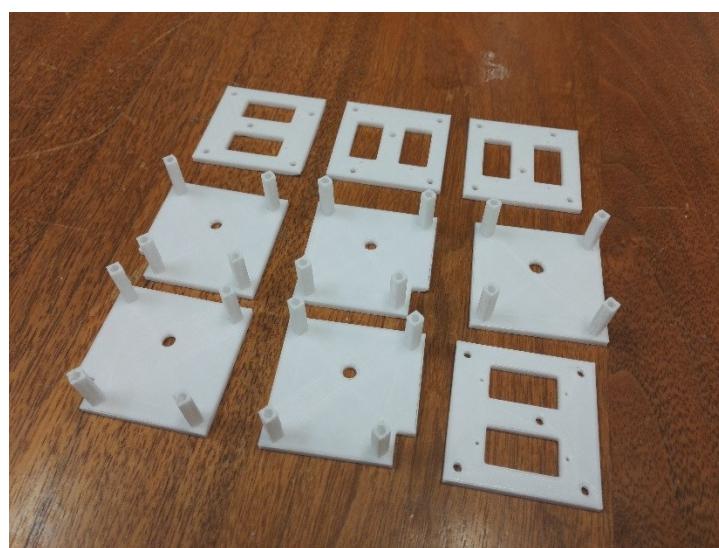


Ilustración 74. Resultado de la impresión 3D de las estructuras de los relojes



Ilustración 75. Resultado de la impresión 3D de los embellecedores de los selectores rotatorios

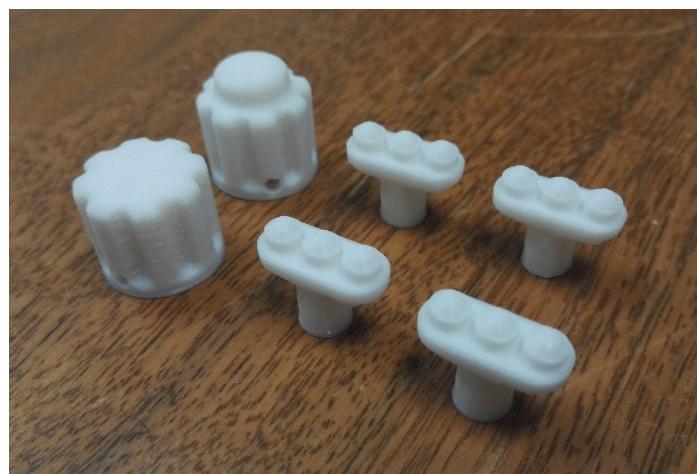


Ilustración 76. Resultado de la impresión 3D de los embellecedores de los codificadores y las luces de aterrizaje



Ilustración 77. Embellecedores de las luces de aterrizaje pintados

## 6.5 Electrónica

La fase de construcción de la electrónica trasera del *Overhead Panel* puede que sea una de las más exigentes de todo el proyecto. Esto se debe a la gran cantidad de cables y componentes electrónicos que se deben soldar. Por ello, se debe ser muy meticuloso a la hora de instalar todo.

El proceso comienza con la instalación de los leds en las cajas de los anunciadores. Cada una tiene un color determinado, variando entre ámbar, verde y azul. Los diodos son fijados usando pegamento termofusible transparente. Hay que tener en cuenta que los leds tienen polaridad, de manera que se coloca siempre a la izquierda el ánodo (patilla larga) para estandarizar la instalación.



Ilustración 78. Instalación de los leds en las cajas de los anunciadores

A continuación, se preparan las placas que contienen las resistencias de  $220\Omega$  que protegerán a los leds de 5 mm colocados en las cajas de los anunciadores.



Ilustración 79. Placas con las resistencias asociadas a los leds de 5 mm

A la hora de integrar los IC 74HC595 se construye la siguiente placa, en la cual los chips van en el centro y las clemas en los laterales, donde irán conectados todos los cables. Todas ellas se numeran indicando el SR\_OUT al que pertenecen y con qué pin de este se corresponden.

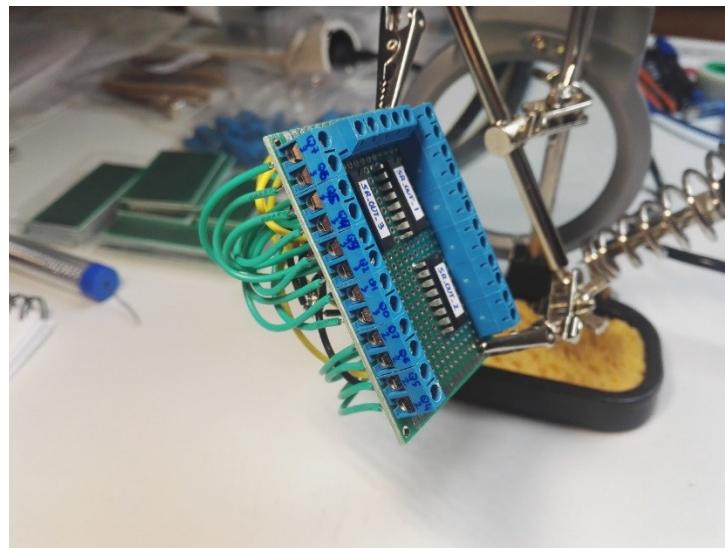


Ilustración 80. Primera versión de las placas SR\_OUT, parte delantera

Por la parte trasera se realizan las conexiones oportunas, tal y como se vio en los esquemas eléctricos del anexo.

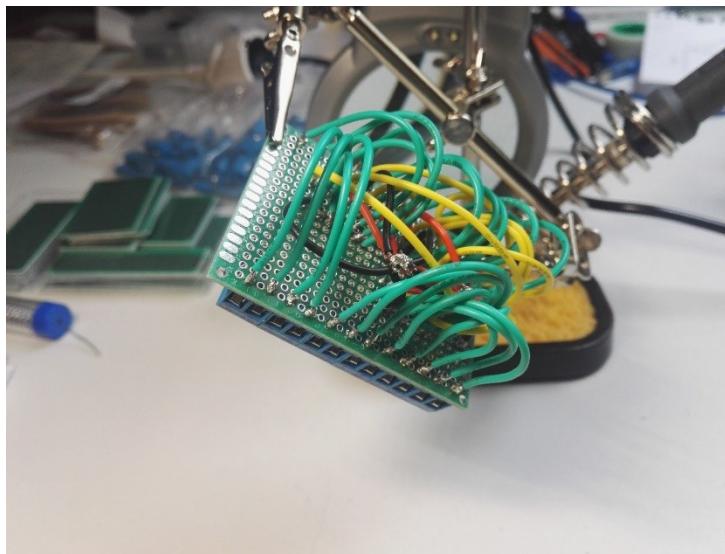


Ilustración 81. Primera versión de las placas SR\_OUT, parte trasera

Durante las primeras pruebas de las placas se identificaron una serie de problemas que dieron lugar a considerar un nuevo diseño.

El primero, ya comentado, fue la necesidad de incluir condensadores electrolíticos de tántalo que filtraran la componente alterna, protegiendo al circuito integrado de posibles picos de tensión transitorios.

Por otro lado, bien por las altas temperaturas aplicadas a los pines a la hora de soldar cables en ellos, bien por picos de tensión al no tener instalados previamente los condensadores, o por el simple hecho de que vinieran defectuosos de fábrica, hubo que cambiar varios circuitos integrados que no funcionaban correctamente. Algunos mostraban una temperatura elevada al ser utilizados u ofrecían medidas anormales en sus pines al ser comprobadas con un polímetro. La tarea de desoldar un IC una vez está montado es complicada, ya que para que salga hay que remover completamente el estaño en sus 16 pines. Para facilitar esta labor de cara al mantenimiento, se introdujo el uso de zócalos. Esto permite soldar los cables directamente a sus pines, y una vez enfriado, colocar el circuito integrado encima de él. A continuación, se muestra una imagen de la segunda versión de estas placas, siendo esta la definitiva.

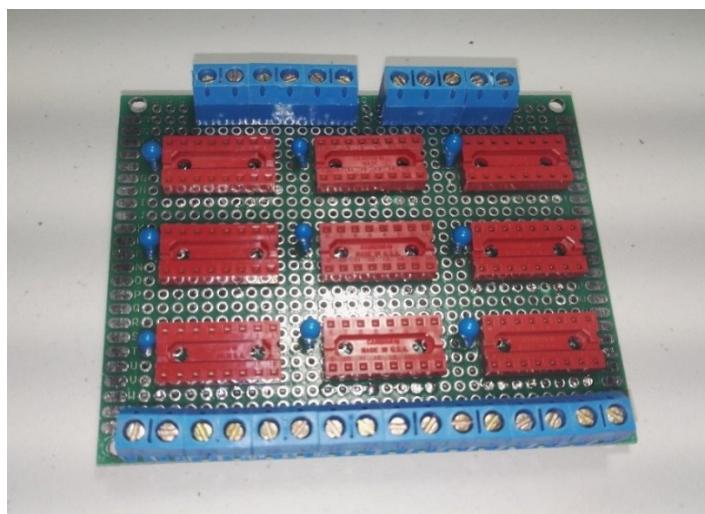


Ilustración 82. Segunda versión de las placas SR\_OUT

Con el diseño mejorado, se pasa ahora a crear las placas de los 74HC165. Estas son muy parecidas a las anteriores, con la diferencia que en la mayoría de los casos se necesita más de una para instalar todos los ICs de una misma cadena. Por ello, se interconectan entre ellas.

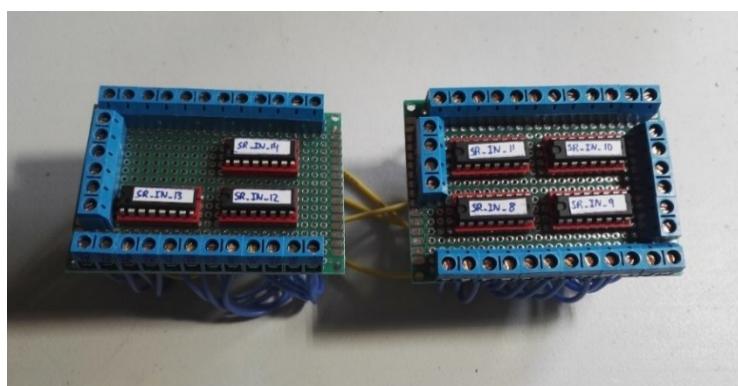


Ilustración 83. Placas SR\_IN interconectadas, parte delantera

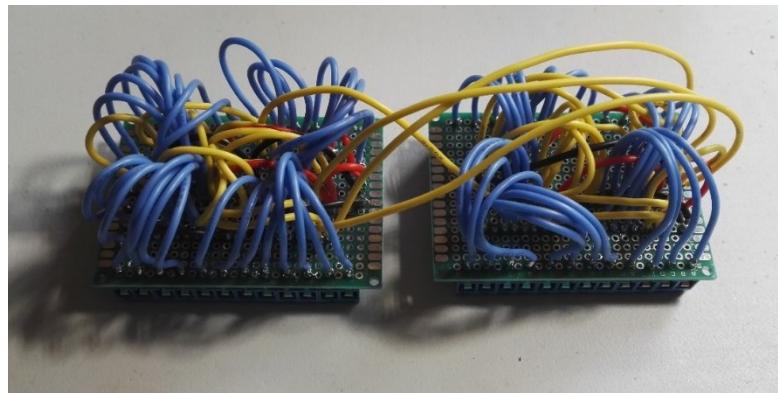


Ilustración 84. Placas SR\_IN interconectadas, parte trasera

Para la fijación de todas las placas se ha utilizado cinta metálica perforada. Esta, al ser considerablemente maleable, permite instalarlas de forma sencilla. Se busca que estén en una posición elevada, en el centro de la columna, de manera que puedan pasar por debajo los mazos de cables. Por ello, se cortan porciones de cinta y se disponen como aparece en la siguiente imagen, utilizando tornillos de rosca madera en el bastidor y tornillos de rosca chapa 2.5x15 mm en las placas.



Ilustración 85. Sistema de sujeción de las placas al bastidor superior de madera

Una vez fijadas, se procede a conectar los cables. Para facilitar el orden, todos ellos irán organizados en mazos y con un código de colores, expuesto a continuación.

Tabla 8. Código de colores del cableado

COLOR	FUNCIÓN
<b>Rojo</b>	Voltaje positivo de 5V
<b>Negro / Marrón</b>	Tierra
<b>Amarillo</b>	Conexión entre ICs y entre estos y Arduino
<b>Verde</b>	Pines de salida de los 74HC595
<b>Azul</b>	Pines de entrada de los 74HC165
<b>Blanco</b>	Cables de datos de entrada / salida a las placas

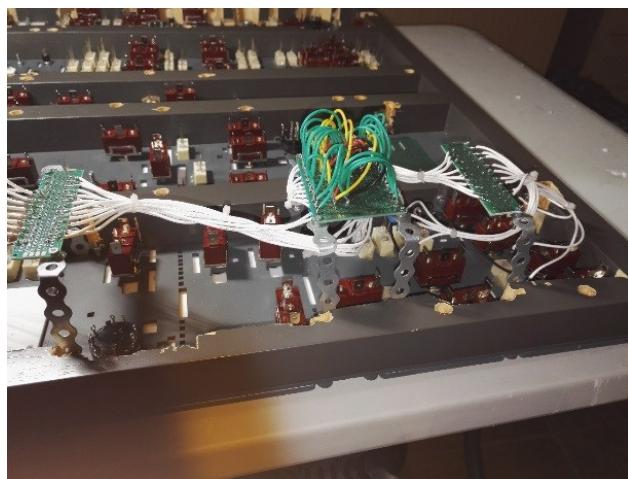


Ilustración 86. Instalación de los mazos de cables de la primera placa SR\_OUT

Adicional al código de colores anterior, los mazos de cables asociados a las salidas irán recogidos con bridgas blancas y los de las entradas con bridgas negras. De esta manera, se hace más sencillo identificar un cable en caso de que haya que realizar algún tipo de mantenimiento.



Ilustración 87. Mazos de cables de la C1

Fue durante la prueba de la primera columna cuando se identificó la necesidad de añadir una fuente de alimentación externa. Cuando todos los leds estaban encendidos estos parpadeaban de forma anormal debido a la falta de energía. Por ello, se recuperó una antigua PSU de un ordenador de sobremesa y se realizó la instalación que se ve en la imagen, solucionando el problema. Apréciense también que el Arduino Mega 2560, junto con la *screw shield*, fueron colocados en la parte superior central del *Overhead Panel*.

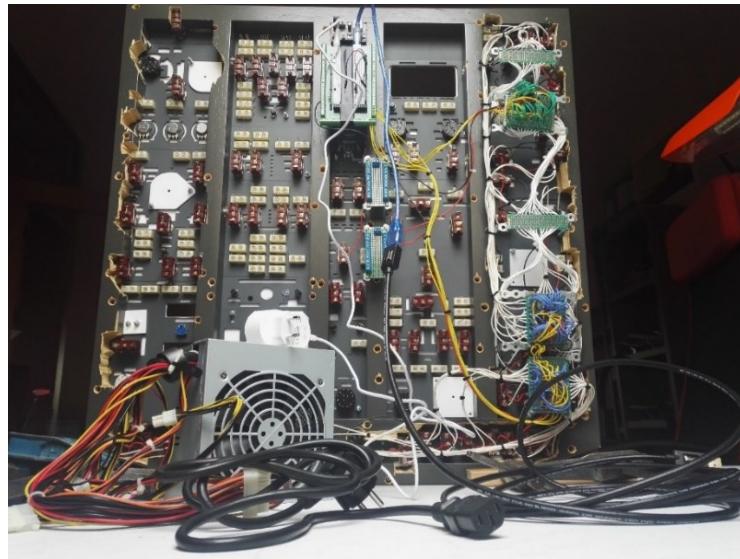


Ilustración 88. Instalación del Arduino Mega 2560 y la PSU

Una vez comprobado que la C1 funciona correctamente, se pasa a la C2. El primer elemento que hay que implementar es la pantalla con los displays LED 7 segmentos del PANEL\_C2\_1. Para ello, se crea una placa de 9 circuitos integrados 74HC595 que la gestione (además de dar servicio a varios anunciantes). Adicionalmente, se instalan las resistencias que protegen los displays directamente en la placa. El resultado se muestra en la siguiente imagen.

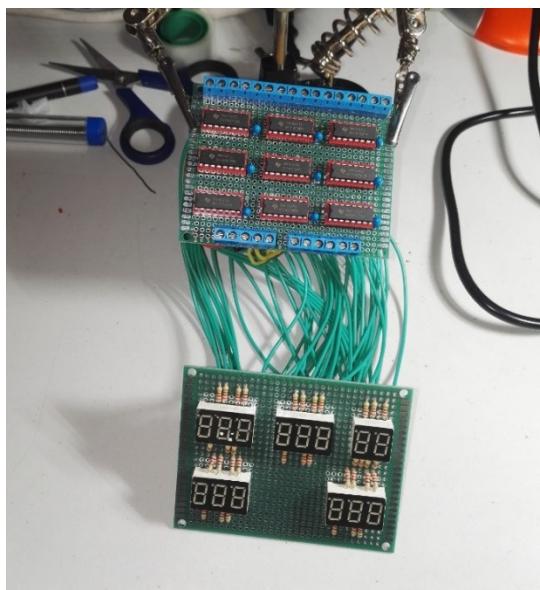


Ilustración 89. Displays LED 7 segmentos del PANEL\_C2\_1 y su placa SR\_OUT

Aprovechando el sobrante del corte de la primera capa de paneles, que se envía junto con el kit, se realizan unos agujeros de manera que los displays encajen en ellos. Los posibles huecos se tapan con cinta aislante de color negro para evitar las fugas de luz desde la parte trasera.

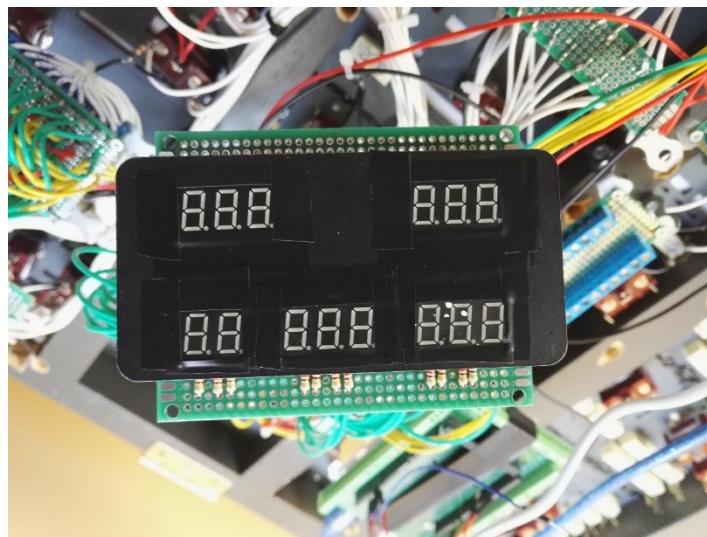


Ilustración 90. Cubierta de los displays LED del PANEL\_C2\_1 instalada

Encima de esta plancha negra, que cubre las resistencias y da estructura a la pantalla, se coloca el cristal tintado proporcionado en el kit. El resultado final del PANEL\_C2\_1 se puede observar en la siguiente imagen.



Ilustración 91. Prueba de los displays LED del PANEL\_C2\_1

A lo largo de toda la instalación, se irán sellando con termofusible negro las distintas uniones entre paneles para reducir las fugas de luz de la retroiluminación a través de ellos.

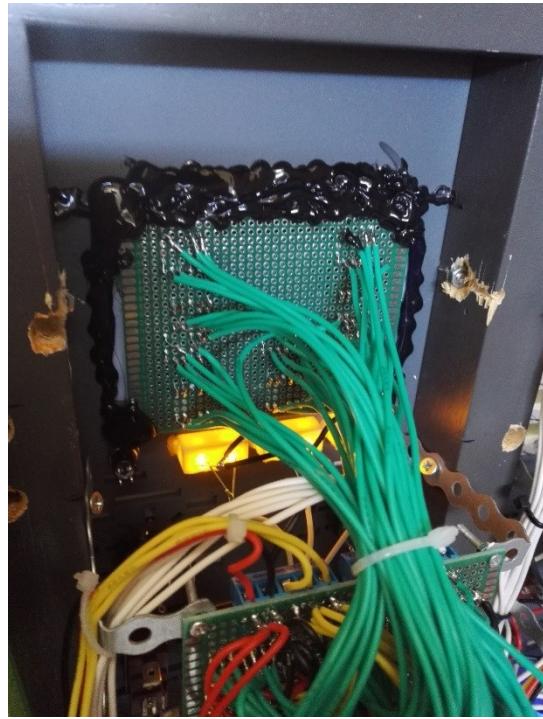


Ilustración 92. Pegamento termofusible negro para sellar fugas de luz de la retroiluminación

Se aprovecha este momento de la instalación, dado que no hay todavía demasiados cables, para instalar las bisagras que unirán ambos bastidores y realizar una prueba de cierre. Nótese que en la *ilustración 94* aun el bastidor inferior no había pasado por la fresadora, siendo en el momento de intentar cerrar cuando se comprobó que era necesario aumentar las dimensiones del interior unos milímetros para permitir el cierre.

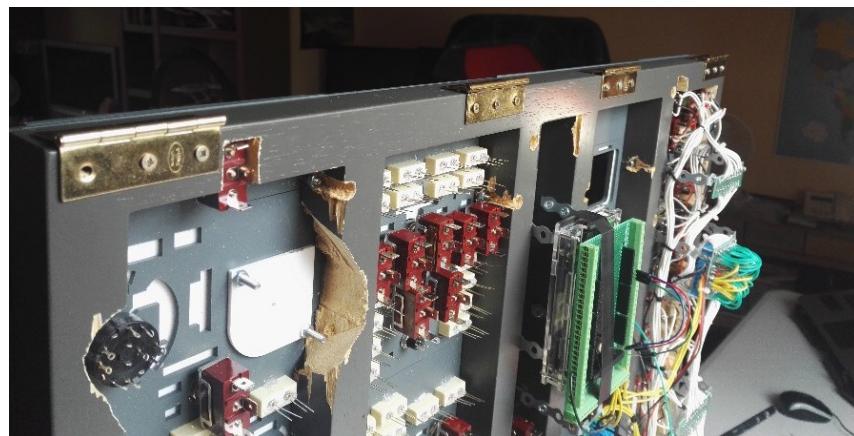


Ilustración 93. Instalación de las bisagras en el bastidor superior de madera

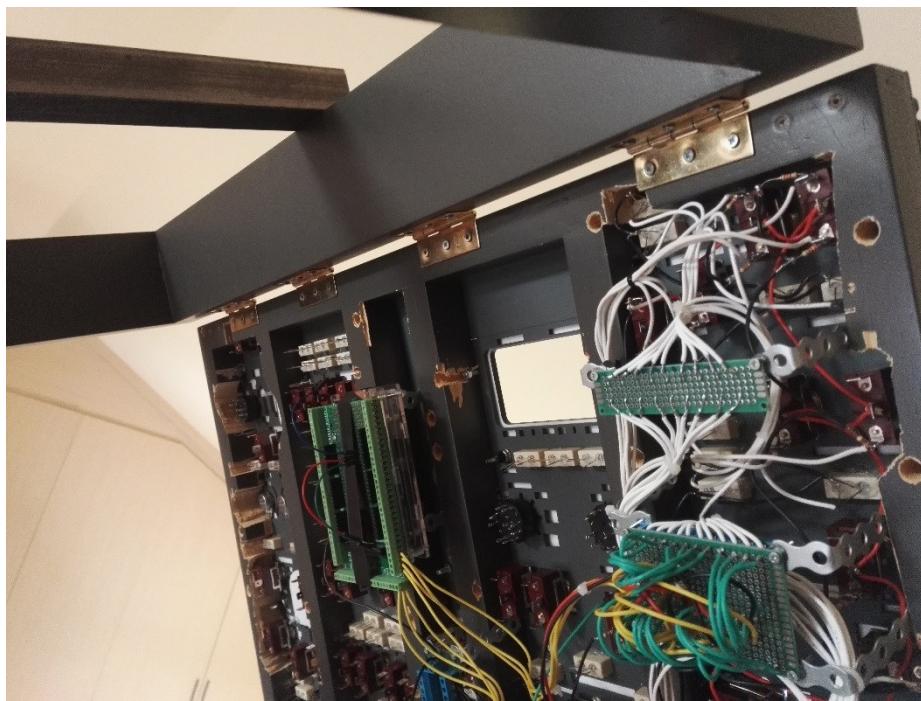


Ilustración 94. Prueba del sistema de unión entre bastidores

De manera equivalente a lo explicado hasta ahora, se procede con las C2 y C3. El progreso de la instalación se puede ver en la siguiente imagen.



Ilustración 95. Placas y mazos de cables de las C2 y C3 instalados

En lo sucesivo, los componentes son similares a las anteriores tres columnas. Destaca el PANEL\_C5\_4, que de nuevo vuelve a tener displays LED 7 segmentos, esta vez de color blanco en vez de verde. El proceso es el mismo que se describió para el PANEL\_C2\_1.



Ilustración 96. Displays LED del PANEL\_C5\_4 instalados

Una vez finalizado el cableado e instalación de las placas de SR\_OUT y SR\_IN, la trasera del *Overhead Panel* tiene el siguiente aspecto.

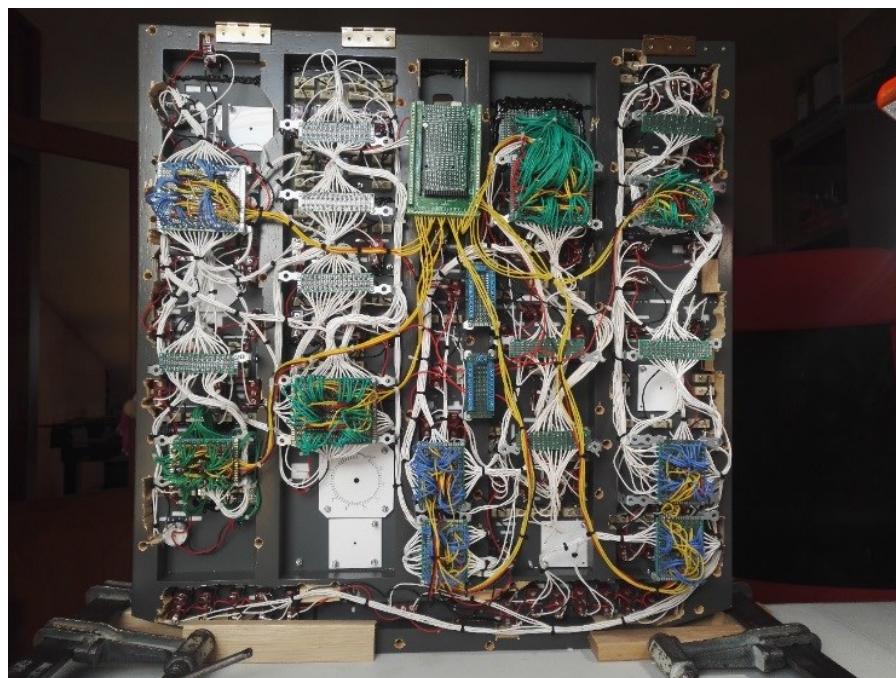


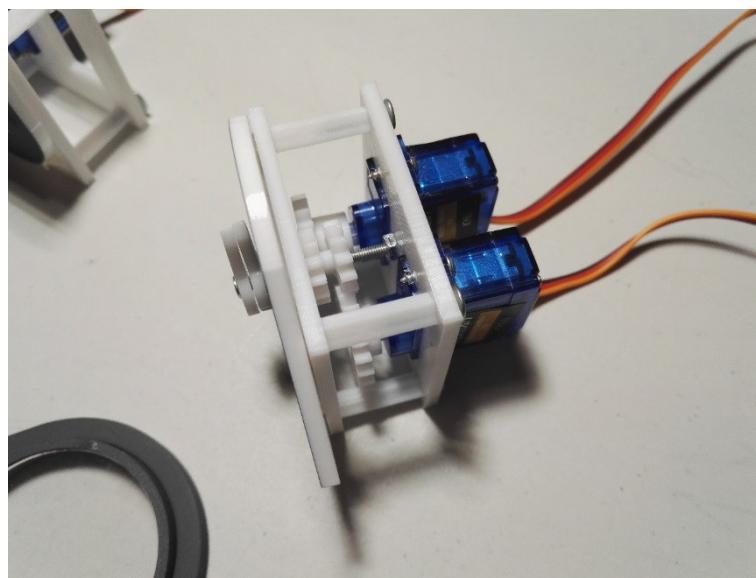
Ilustración 97. Placas y mazos de cables de las C4 y C5 instalados

El siguiente paso consiste en añadir los relojes. Estos se deben montar conforme a lo que se vio en el apartado de diseño. Todos ellos tienen una estructura personalizada, ya que, debido a las dimensiones de la zona de apoyo, fue necesario editar la pieza original para permitir pasar a los tornillos de cada reloj.



*Ilustración 98. Montaje de los mecanismos de los relojes*

Una vez montado, pegando la estructura de los servomotores y engranajes al panel del reloj con termofusible transparente, el interior del mecanismo tiene el siguiente aspecto.



*Ilustración 99. Vista interna del mecanismo de los relojes*

Cuando todos los relojes están ensamblados se procede a su instalación. Haciendo uso de los tornillos de rosca chapa del kit, estos quedan fijados a la primera capa de paneles, tal y como se aprecia en la siguiente ilustración.

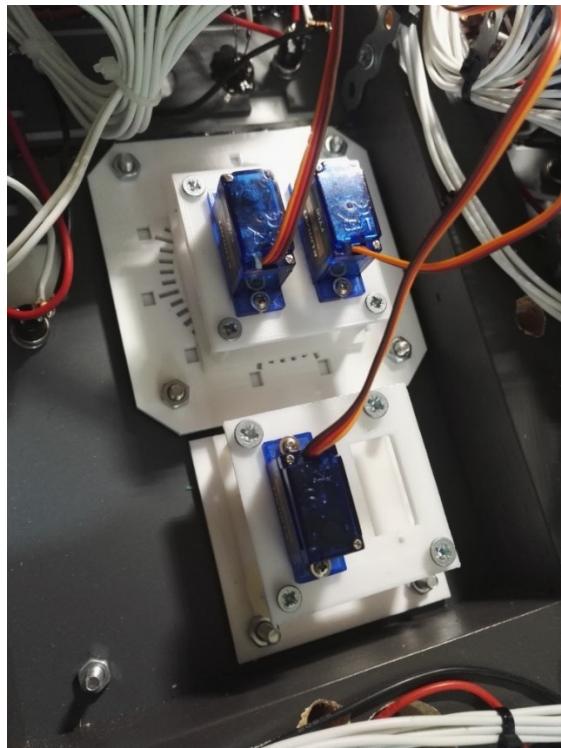


Ilustración 100. Instalación de los relojes del PANEL\_C4\_6, vista trasera

Por la parte delantera se colocan las agujas (incluidas en el kit) y la pegatina circular negra que tapa al tornillo que las une con los engranajes. Por último, se añade el cristal del reloj.



Ilustración 101. Instalación de los relojes del PANEL\_C4\_6, vista delantera

Terminada la instalación de los relojes, para finalizar la construcción de la electrónica se coloca el sistema de autorrotación de los selectores del PANEL\_C1\_4. Este comprende dos piezas laterales, donde se instalan los selectores, uno central para el servomotor y el sistema de transmisión de movimiento que se vio en el apartado 5.5.6 de este informe. Para la unión de las primeras piezas con el bastidor se usan tornillos de rosca madera. En cambio, para la unión de las levas se utilizan tornillos rosca chapa con doble tuerca, de manera que queden fijadas, pero con cierta holgura que las permita rotar. Es imprescindible, una vez instalado, calibrar el mecanismo de tal forma que no gire más de lo necesario, pues de lo contrario podría devolver los selectores en vez de a la posición OFF, a la siguiente (CONT).



Ilustración 102. Instalación del mecanismo de autorrotación de los selectores del PANEL\_C1\_4

Este es el último paso de la construcción de la electrónica trasera. Una vez terminado, se conecta el Arduino al ordenador y se comprueba que todo funciona correctamente. El *Overhead Panel*, una vez acabado, tiene el siguiente aspecto.



Ilustración 103. Overhead Panel acabado en modo light test, vista delantera

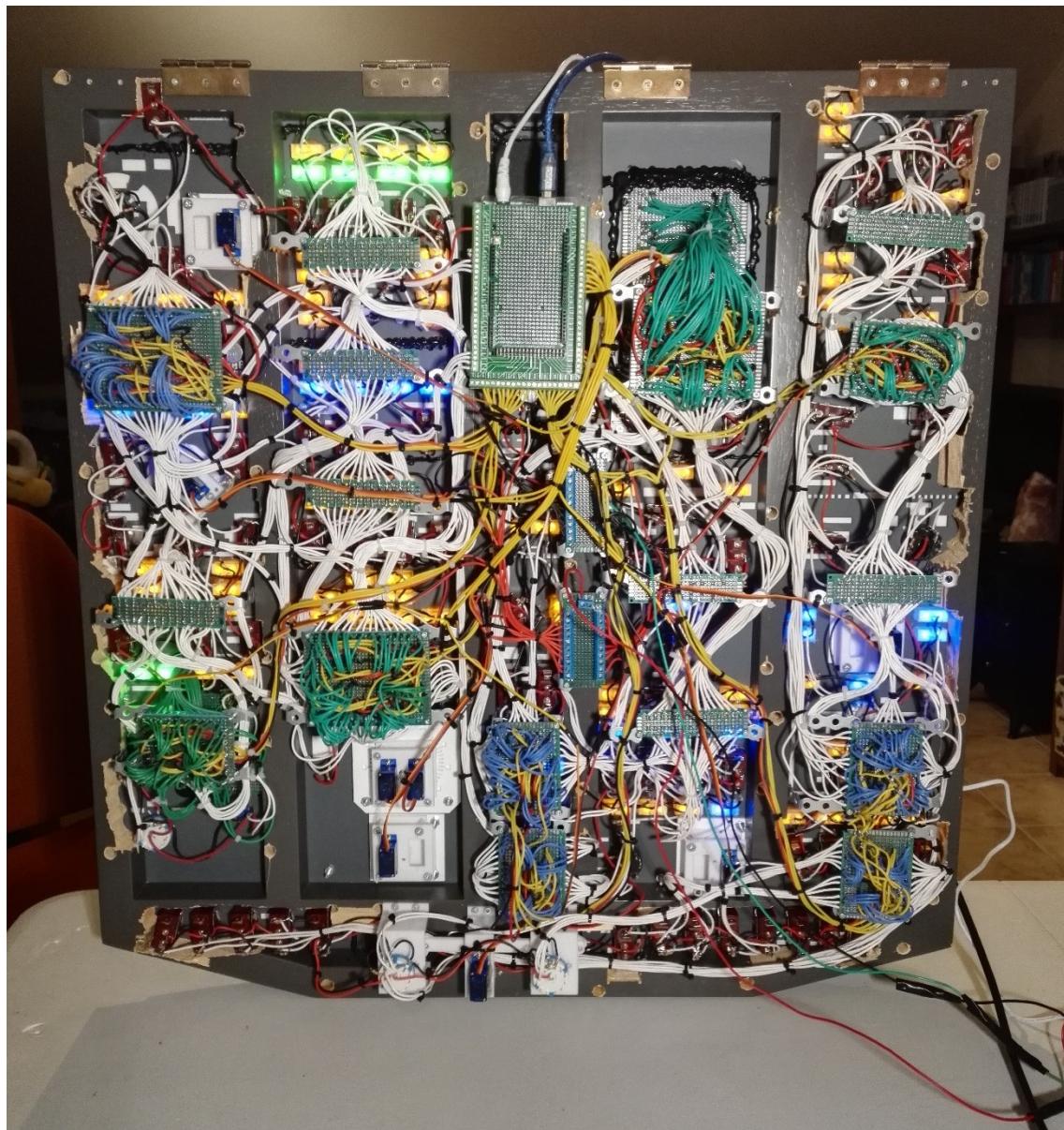


Ilustración 104. Overhead Panel terminado en modo light test, vista trasera

## 6.6 Sistema de retroiluminación

El sistema de retroiluminación, como se comentó previamente, está compuesto por 24 tiras LED, todas ellas conectadas en paralelo entre sí. El circuito que las alimenta es diferente al del resto del panel (12V a 16A). Las tiras LED son fijadas a la trasera del bastidor del panel con pegamento termofusible y todas sus conexiones son protegidas con cinta aislante negra. Adicionalmente, los cables también son fijados para evitar que queden colgando. Una vez realizada la instalación se prueba para confirmar que no haya ningún led fundido. También se verifica como quedaría una vez superpuesto con los paneles.



Ilustración 105. Prueba de encendido del sistema de retroiluminación



Ilustración 106. Prueba del sistema de retroiluminación a través de los paneles

Realizadas las anteriores comprobaciones, se procede a instalar el sistema de control de intensidad lumínica. Como ya se explicó, este incluye el transistor MOSFET, el disipador de calor y las dos resistencias, además del potenciómetro. El sistema es fijado en la parte superior de la trasera del bastidor.

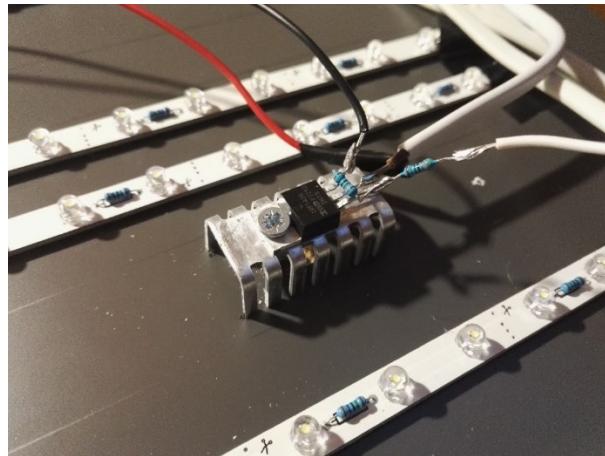


Ilustración 107. Instalación del sistema de regulación de intensidad lumínica

Finalmente se une la trasera con el bastidor inferior y se realiza un orificio que permita la salida de cables hacia el exterior del *Overhead Panel*.



Ilustración 108. Unión de la trasera al bastidor inferior de madera

## 6.7 Unión de los bastidores de madera

Realizada la instalación de la electrónica y del sistema de retroiluminación, solo resta unir ambos bastidores mediante el sistema de bisagras. En la siguiente imagen se puede comprobar cómo se vería el *Overhead Panel* desplegado para poder realizar mantenimiento.

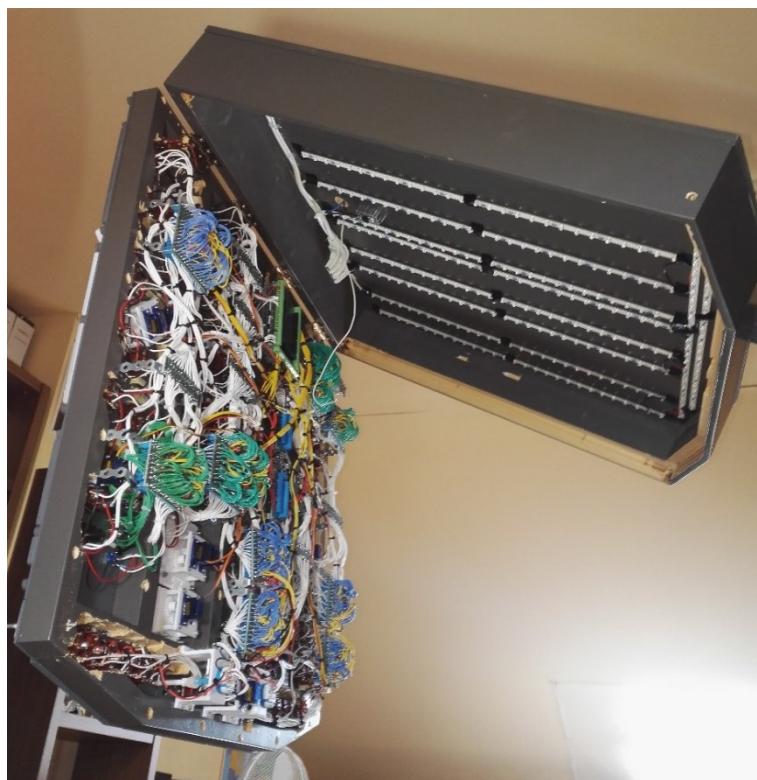


Ilustración 109. Unión de los bastidores de madera mediante las bisagras

A continuación, se cierra mediante dos tornillos de rosca madera de 4.5x100 mm situados en las esquinas inferiores del bastidor superior. Dado que estos tornillos son muy largos y atraviesan una gran cantidad de madera maciza, es recomendable impregnarlos de jabón sólido o cera para facilitar su entrada a la hora de ser roscados.

Con el bastidor cerrado, se instala en la parte superior de la trasera la fuente de alimentación, así como una regleta eléctrica para conectar tanto la PSU, como el Arduino. Para este cometido, se vuelve a usar de nuevo la cinta metálica perforada. Por último, se organizan todos los cables y se atan con bridales a la estructura metálica, dejando una holgura suficiente para que el *Overhead Panel* pueda ser desplegado.



Ilustración 110. Instalación de la PSU en la trasera del panel

El *Overhead Panel* terminado, contemplado desde el ángulo que lo vería el piloto, se puede observar en la siguiente ilustración.



Ilustración 111. Resultado final del Overhead Panel visto desde el ángulo del piloto

Para finalizar la fase de construcción se realiza una última prueba, esta vez en un entorno más oscuro, para comprobar que los leds, los displays y la retroiluminación, una vez se ha cerrado el bastidor y no existen fugas de luz, funcionan correctamente y de acuerdo con los parámetros de diseño establecidos.

En la siguiente imagen se ve el modo *light test*, durante el cual la intensidad de la retroiluminación se reduce para suministrar más corriente a los anunciantes y maximizar su brillo, con el objetivo de poder detectar, si existe, alguno con un led fundido de los dos que hay en su interior.



Ilustración 112. Overhead Panel en modo light test en un entorno oscuro

## 7. Integración de los elementos del *Overhead Panel* con Arduino

La última fase del proyecto consiste en la integración de los elementos del *Overhead Panel* con Arduino. A pesar de ser la última expuesta en este informe, como se comentó previamente, no por ello es la que se realiza al final, pues es necesario tenerse en cuenta a la vez que el resto de las fases se van desarrollando.

Este capítulo intentará explicar de forma resumida el código que configurará el *Overhead Panel*. Este se puede consultar en el Anexo, sección 11.16. Para facilitar la labor, dado que es bastante extenso, este se desglosará en el código asociado a la integración de cada elemento. Por ello, el capítulo se dividirá en las siguientes secciones:

1. Estructura del código e interacción con Cockpit-X
2. IC 74HC595 (SR\_OUT)
3. IC 74HC165 (SR\_IN)
4. Leds de 5 mm
5. Display LED 7 segmentos
6. Servomotor
7. Interruptor
8. Pulsador tipo botón
9. Selector rotatorio
10. Codificador rotatorio
11. Potenciómetro
12. Transistor IRF1404 MOSFET y sistema de retroiluminación

## 7.1 Estructura del código e interacción con Cockpit-X

El código de Arduino comienza con una etiqueta donde se identifica el nombre del proyecto, el de la universidad, el autor y la versión del código. En el momento de la realización de este trabajo fin de grado se considera la versión *v1.5*. Es posible que, con el desarrollo de nuevos simuladores y funcionalidades, se vaya actualizando en el futuro.

El código se divide en seis secciones:

1. Libraries section: Se incluyen todas las librerías que se utilizan a lo largo del código.
2. Classes section: Se definen todas las clases que se utilizarán a lo largo del código.
3. Variables section: Se definen todas las variables que se utilizarán a lo largo del código y sus valores iniciales. Estas se organizan por paneles y dentro de estos por tipo de elemento.
4. Setup section: Es aquel código que se ejecutará una sola vez cuando el Arduino enciende. En él se establece la conexión serial, se realiza la definición de los pines y se inicializan los IC y los servomotores. También se deja el *Overhead Panel* en modo “*Cold & Dark*”, es decir, completamente apagado.
5. Loop section: Es aquel código que se ejecutará en bucle mientras el Arduino esté encendido. Es, por tanto, el código que permite el refresco continuo del panel. En él se incluye, en términos generales, la lectura y escritura en el monitor serial, así como todo lo que hay detrás para que esta comunicación se haga según el lenguaje estandarizado de Cockpit-X.
6. Function section: Se incluyen todas aquellas funciones que son llamadas desde otras partes del código.

A la hora de comunicarse con Cockpit-X, Arduino utiliza el monitor serial. Como se vio previamente, Cockpit-X asigna a cada elemento de la cabina un código de dos letras y una variable numérica. Las letras identifican dicho elemento y el número su estado. De esta manera, a la hora de confeccionar el código del Arduino, será necesario leer los códigos enviados por el programa y gestionar las salidas de acuerdo con ellos. En siguientes secciones se verá cómo se realiza para cada elemento por individual.

## 7.2 IC 74HC595 (SR\_OUT)

Para explicar el código del 74HC595 se va a considerar la primera placa (SR\_OUT\_1 a SR\_OUT\_3) que da servicio a los anunciantes de la C1.

Lo primero que se realiza es crear las tres variables asociadas a los pines de control del 74HC595 (data, latch y clock), asociándolas con el número del pin digital del Arduino al que están conectadas. A continuación, se definen las tres variables de tipo *byte* (cada una contiene 8 bits de información) que almacenarán los estados de las 24 salidas.

```
72 // **SHIFT REGISTERS (OUT)**
73
74     int dataPin_OUT_1 = 22; //Pin connected to DS of SR_OUT_1
75     int latchPin_OUT_1 = 23; //Pin connected to ST_CP of SR_OUT_1
76     int clockPin_OUT_1 = 24; //Pin connected to SH_CP of SR_OUT_1
77
78     byte SR_OUT_1; //Control LED state
79     byte SR_OUT_2; //Control LED state
80     byte SR_OUT_3; //Control LED state
```

En la sección *setup* se inicializa la conexión serial a 115200 baudios y se asignan las variables de los pines del IC a los del Arduino mediante la sentencia *pinMode()*.

```
477 // --- SETUP SECTION --- //
478
479 void setup() {
480
481     // *SERIAL CONNECTION*
482
483     Serial.begin(115200);
484
485
486     // *PINS*
487
488     // **PANEL C1-1, C1-2, C1-3, C1-4 PINS**
489
490     pinMode(dataPin_OUT_1, OUTPUT);
491     pinMode(latchPin_OUT_1, OUTPUT);
492     pinMode(clockPin_OUT_1, OUTPUT);
```

Dentro también del *setup*, se configura el estado “*Cold & Dark*” mediante un bucle que escribe 0 lógicos en las tres variables con la sentencia *bitClear()* (si se quisiera escribir 1 lógicos se usaría *bitSet()*). Una vez escritos, es necesario actualizar los SR\_OUT. Para ello se llama a la función *update\_SR\_OUT\_1()*. Cuando se termina de actualizar se ponen en bajo los tres pines de control del 74HC595, dejándolo así en *stand-by*.

```

535 // *COLD AND DARK SETUP*
536
537 // **PANEL C1-1 COLD AND DARK**
538
539 // ***SHIFT REGISTERS (OUT) ***
540
541 for (int i=0; i<8; i++){
542     bitClear(SR_OUT_1,i);
543     bitClear(SR_OUT_2,i);
544     bitClear(SR_OUT_3,i);
545 }
546
547 update_SR_OUT_1();
548
549 digitalWrite(dataPin_OUT_1, LOW);
550 digitalWrite(latchPin_OUT_1, LOW);
551 digitalWrite(clockPin_OUT_1, LOW);

```

La función *update\_SR\_OUT\_1()*; contiene las siguientes sentencias.

```

2564 // *UPDATE SR_OUT_1 FUNCTION*
2565
2566 void update_SR_OUT_1() { // Sends the data to SR_OUT_1
2567
2568 if ((millis() - lastUpdateLED) > 0.01){
2569
2570     digitalWrite(latchPin_OUT_1,LOW);
2571     shiftOut(dataPin_OUT_1, clockPin_OUT_1, MSBFIRST, SR_OUT_3);
2572     shiftOut(dataPin_OUT_1, clockPin_OUT_1, MSBFIRST, SR_OUT_2);
2573     shiftOut(dataPin_OUT_1, clockPin_OUT_1, MSBFIRST, SR_OUT_1);
2574     digitalWrite(latchPin_OUT_1, HIGH);
2575     lastUpdateLED = millis();
2576
2577 }
2578
2579 } //End update_SR_OUT_1()

```

Para evitar el uso de la función *delay()* que deja al Arduino inoperativo durante esos segundos, se utiliza como alternativa *millis()*, que almacena los milisegundos que pasan desde el encendido del Arduino. Esta se compara con la variable *lastUpdateLED*. Cuando han pasado 0.01 ms desde la última actualización se refresca el SR\_OUT. Para ello, el pin de latch se pone en bajo, se carga la información de las variables *byte* mediante la sentencia *shiftOut* y se pone de nuevo el latch en alto. Cuando se ha terminado se actualizan los milisegundos almacenados en la variable *lastUpdateLED*. Nótese que se carga primero los datos del SR\_OUT\_3 (último IC de la cadena), pues al principio entrarán por el primer IC, pero después pasarán al segundo IC y por último quedarán alojados en el tercer IC, como se vio en la *ilustración 21*. Cuantos más SR\_OUT tenga la cadena, más líneas tendrá esta función. En el anexo, sección 11.14, se expone la asignación de los elementos de salida del *Overhead Panel* a los pines del 74HC595.

### 7.3 IC 74HC165 (SR\_IN)

Para explicar el código del 74HC165 se va a considerar la primera placa (SR\_IN\_1 a SR\_IN\_7) que da servicio a los interruptores y selectores de la C1.

Como se hizo con los SR\_OUT, lo primero que se realiza es crear las cuatro variables asociadas a los pines de control del 74HC165 (parallel load, clock enable, data y clock), asociándolas con el número del pin digital del Arduino al que están conectadas. A continuación, se definen los parámetros de la cadena, entre los que se incluyen el número de SR\_IN totales en la cadena, el tamaño del array de datos, el retardo entre lectura y latch y el tipo de variable en el que se almacenaran los datos (*unsigned long*).

En el caso de los SR\_IN es necesario tener dos variables de datos. La primera es donde se almacenan los nuevos datos y la segunda donde están los antiguos. De esta manera, si se detecta que algún estado ha cambiado al comparar ambas variables, se refresca el SR\_IN.

Dado que la variable de almacenamiento de estados es de tipo *unsigned long*, esta puede almacenar hasta 32 bits de información (4 bytes), es decir, los datos de 4 SR\_IN. Por ello, dado que esta placa tiene 7 SR\_IN, será necesario crear dos variables. Se decide utilizar la de tipo *unsigned* ya que las variables de tipo *long* reservan un bit para el signo negativo, imposibilitando almacenar en ese caso 4 SR\_IN en una sola variable. Como solo se almacenarán los valores lógicos 0 ó 1, el signo negativo no es necesario.

```
82 // **SHIFT REGISTERS (IN)**
83
84     int ploadPin_IN_1 = 25; // Pin connected to PL of SR_IN_7
85     int clockEnablePin_IN_1 = 26; // Pin connected to CE of SR_IN_7
86     int dataPin_IN_1 = 27; // Pin connected to Q7 of SR_IN_7
87     int clockPin_IN_1 = 28; // Pin connected to CP of SR_IN_7
88
89 #define NUMBER_OF_SHIFT_CHIPS_1    7 // How many shift register chips are daisy-chained.
90 #define DATA_WIDTH_1    NUMBER_OF_SHIFT_CHIPS_1 * 8 // Width of data (how many ext lines).
91 #define PULSE_WIDTH_USEC   0.001 // Width of pulse to trigger the shift register to read and latch.
92 #define SR_IN_1_T unsigned long
93
94     SR_IN_1_T SR_IN_1; //Control SWITCH state
95     SR_IN_1_T OLD_SR_IN_1; //Control previous SWITCH state
96     SR_IN_1_T SR_IN_5; //Control SWITCH state
97     SR_IN_1_T OLD_SR_IN_5; //Control previous SWITCH state
```

En la sección setup se asignan las variables de los pines del IC a los del Arduino mediante la sentencia *pinMode()*, como se hizo con los SR\_OUT.

```
494     pinMode(ploadPin_IN_1, OUTPUT);
495     pinMode(clockEnablePin_IN_1, OUTPUT);
496     pinMode(clockPin_IN_1, OUTPUT);
497     pinMode(dataPin_IN_1, INPUT);
```

Dentro también del setup, se configura el estado “*Cold & Dark*”. Para sincronizar el *Overhead Panel* con el del simulador, los interruptores deberán estar tal y como se encontrarían al iniciar el simulador en este modo. Asegurando esto, se realiza una lectura completa del *Overhead Panel* y se actualizan los SR\_IN con los estados de cada uno de los elementos de entrada. Para ello se recurre a las funciones *update\_SR\_IN\_5()*, *update\_SR\_1()* y *modify\_switch\_values\_1()*. Esta última será explicada en la sección 7.7 de este capítulo. Por último, se actualizan las variables de estados antiguos y se ponen los SR\_IN en modo *stand-by*.

```
554 // ***SHIFT REGISTERS (IN) ***
555
556     SR_IN_5 = update_SR_IN_5();
557     SR_IN_1 = update_SR_IN_1();
558     modify_switch_values_1();
559     OLD_SR_IN_5 = SR_IN_5;
560     OLD_SR_IN_1 = SR_IN_1;
561
562     digitalWrite(clockPin_IN_1, LOW);
563     digitalWrite(ploadPin_IN_1, HIGH);
```

Las dos primeras funciones son similares, aunque incluyen diferencias. En la primera se leen los estados de los SR\_IN\_7, SR\_IN\_6 y SR\_IN\_5 (en ese orden) y se guardan en la variable *SR\_IN\_5*, mientras que en la segunda se leen los estados del SR\_IN\_4 al SR\_IN\_1 y se guardan en la variable *SR\_IN\_1*. Esto se hace de acuerdo con la explicación de funcionamiento de los 74HC165 que se dio en el apartado 5.5.4.

La función comienza con la definición de la variable *bitVal*, donde será almacenado de forma temporal el valor lógico de un pin, hasta volcarse en la variable *bytesVal*, que se inicializa a cero y será la que se devuelva al finalizar la función mediante la sentencia *return(bytesVal)*.

A continuación, se leen los estados de los pines. Para ello el pin *clock enable* se lleva a alto y el *parallel load* a bajo. Se dejan pasar los microsegundos que se fijaron en la sección de variables y se devuelven los pines a bajo y alto respectivamente. Con esto el SR\_IN ha leído los valores lógicos en sus pines de entrada y los tiene cargados.

Para realizar ahora la lectura de esos estados se confecciona un bucle. La condición de este es repetirse 24 veces, es decir el número de pines de entrada que se deben leer en los tres primeros SR\_IN. Para ello se usa la sentencia *i < (DATA\_WIDTH\_1-32)*. Los valores lógicos de los estados van saliendo del SR\_IN\_7 por su pin *Q7* (o *data pin*) y se almacenan temporalmente en el primer bit de la variable *bitVal*. Posteriormente, este valor se vuela en la variable *bytesVal* mediante la sentencia de la línea 2679. En ella es interesante prestar atención a cómo se define la posición del array en la que se debe escribir.

Durante la primera iteración del bucle la *i* toma de valor *i=0*, por tanto, el resultado de la operación *((DATA\_WIDTH-33) - i)* es 23. De esta manera, el estado lógico, que había sido alojado en el primer bit de la variable *bitVal*, es desplazado 23 posiciones a la izquierda mediante el operador “*<<*”. La escritura en la variable *bytesVal* se realiza con el operador “*=/*”, que actúa según la lógica de una puerta OR.

Teniendo en cuenta que en Arduino (C++) la primera posición de un array empieza en 0, el valor 23 representa la vigesimocuarta posición del array, que se corresponde con el vigesimocuarto pin, el D7 del SR\_IN\_7, el primero en leerse a través del pin Q7. De esta manera se van almacenando todos hasta llegar al D0 del SR\_IN\_5, que ocuparía la posición cero del array de datos.

Como en la cadena no está el SR\_OUT\_8, los 8 bits libres de la variable *bytesVal* (del total de 32 bits que tiene), quedan sin sobrescribir y por tanto con valor lógico 0.

Por último, todavía dentro del bucle, se genera un pulso de reloj que desplaza el siguiente estado al pin Q7 del SR\_IN\_7 para que sea leído en la siguiente iteración.

```
2654 // *UPDATE_SR_IN_5 FUNCTION*
2655
2656 SR_IN_1_T update_SR_IN_5() { // Receive the data from SR_IN_5 to SR_IN_7
2657
2658     long bitVal;
2659     SR_IN_1_T bytesVal= 0;
2660
2661
2662     /* Trigger a parallel Load to latch the state of the data lines,
2663     */
2664     digitalWrite(clockEnablePin_IN_1, HIGH);
2665     digitalWrite(ploadPin_IN_1, LOW);
2666     delayMicroseconds(PULSE_WIDTH_USEC);
2667     digitalWrite(ploadPin_IN_1, HIGH);
2668     digitalWrite(clockEnablePin_IN_1, LOW);
2669
2670     /* Loop to read each bit value from the serial out line
2671     * of the SN74HC165N.
2672     */
2673     for(int i = 0; i < (DATA_WIDTH_1-32); i++)
2674     {
2675         bitVal = digitalRead(dataPin_IN_1);
2676
2677         /* Set the corresponding bit in bytesVal.
2678         */
2679         bytesVal |= (bitVal << ((DATA_WIDTH_1-33) - i));
2680
2681         /* Pulse the Clock (rising edge shifts the next bit).
2682         */
2683         digitalWrite(clockPin_IN_1, HIGH);
2684         delayMicroseconds(PULSE_WIDTH_USEC);
2685         digitalWrite(clockPin_IN_1, LOW);
2686     }
2687
2688     return(bytesVal);
2689
2690 } //End update_SR_IN_5()
```

Cuando el bucle ha terminado las 24 iteraciones, el estado del pin D7 del SR\_IN\_4 está esperando a ser leído en el pin Q7 del SR\_IN\_7 mediante la función *update\_SR\_1()*.

La diferencia entre esta función que se acaba explicar y la nueva reside únicamente en el número de iteraciones del bucle, que ahora pasan a ser 32 al tenerse que leer cuatro SR\_IN, y la sentencia de la línea 2673 que se redefine para ajustarse al nuevo número de iteraciones. Al haber sido ya leídos y cargados los estados mediante la función *update\_SR\_IN\_5*, en esta función no se repiten las sentencias de la línea 2664 a la 2668, pasando directamente a leer los estados mediante el bucle ya citado.

```
2694 // *UPDATE SR_IN_1 FUNCTION*
2695
2696 SR_IN_1_T update_SR_IN_1() { // Receive the data from SR_IN_1 to SR_IN_4
2697
2698     long bitVal;
2699     SR_IN_1_T bytesVal= 0;
2700
2701
2702     /* Loop to read each bit value from the serial out line
2703      * of the SN74HC165N.
2704      */
2705     for(int i = 24; i < DATA_WIDTH_1; i++)
2706     {
2707         bitVal = digitalRead(dataPin_IN_1);
2708
2709         /* Set the corresponding bit in bytesVal.
2710         */
2711         bytesVal |= (bitVal << ((DATA_WIDTH_1-1) - i));
2712
2713         /* Pulse the Clock (rising edge shifts the next bit).
2714         */
2715         digitalWrite(clockPin_IN_1, HIGH);
2716         delayMicroseconds(PULSE_WIDTH_USEC);
2717         digitalWrite(clockPin_IN_1, LOW);
2718     }
2719
2720     return(bytesVal);
2721
2722 } //End update_SR_IN_1()
```

Si las nuevas cadenas tienen diferente número de SR\_IN, será necesario definir nuevas funciones, pero siempre respetando los principios de lectura y escritura que se han expuesto en esta sección. En el anexo, sección 11.15, se expone la asignación de los elementos de entrada del *Overhead Panel* a los pines del 74HC165.

## 7.4 Leds de 5 mm

Como en casos anteriores, lo primero que se realiza es definir las variables con las que se va a trabajar. En el caso de los leds, estos se identifican por el código de dos letras que tienen asociados según el protocolo de comunicación establecido por Cockpit-X. Adicionalmente, se les asigna un valor en función del estado en el que se encontraran cuando se encienda el *Overhead Panel* mediante el interruptor *SWITCH\_MAIN\_BATTERY*.

```
54 // --- VARIABLES SECTION --- //
55
56
57 // *PANEL C1-1 VARIABLES*
58
59 // **LEDS**
60
61     int valueBU='0', valueBV='0', valueBW='0', valueBX='1', valueBY='1', valueBZ='0'; //Control LED state
62     int valueCA='0', valueCB='0', valueCC='0', valueCE='1'; //Control LED state
```

Para encender o apagar leds se ha de leer constantemente la información enviada por Cockpit-X a través del monitor serial. Esto se realiza con la función *getChar()*, la cual devuelve un carácter que es almacenado en la variable *CodeIn*. Mediante las sentencias *if* se llama a las funciones que determinan que led ha sido modificado y en qué medida.

```
678 // --- LOOP SECTION --- //
679
680 void loop() {
681
682     // *LEDS*
683
684     // **SERIAL DATA ACQUISITION**
685
686     if (Serial.available()){
687
688         CodeIn = getChar();
689
690         if (CodeIn == 'B') {B();} //First identifier is B
691         if (CodeIn == 'C') {C();} //First identifier is C
692         if (CodeIn == 'D') {D();} //First identifier is D
693         if (CodeIn == 'E') {E();} //First identifier is E
694         if (CodeIn == 'F') {FOX();} //First identifier is F
695         if (CodeIn == 'G') {G();} //First identifier is G
696         if (CodeIn == 'Z') {Z();} //First identifier is Z
697         if (CodeIn == '$') {DOLLAR();} //First identifier is $
698
699     }
```

La función *getChar()* tiene el siguiente aspecto:

```
1652 // --- FUNCTION SECTION --- //
1653
1654 // *GETCHAR FUNCTION*
1655
1656 void getChar() { // Get one character from serial buffer
1657     while(Serial.available() == 0); //Wait for the data
1658     return((char)Serial.read());
1659 }
```

Para explicar las funciones que están dentro de las sentencias *if* se va a considerar la función *B()*, pues el resto son similares cambiando solo la letra del primer identificador.

Imagínese que el primer carácter recibido a través del monitor serial es *B*. Esto llama a la función *B()*, donde se lee cuál es el segundo carácter. Si este fuera *U*, se tendría el código *BU*, que es el identificador del led *LED\_STANDBY\_HYD\_LOW\_QUANTITY*, primer anunciador del *PANEL\_C1\_1*. Consecuentemente, la sentencia *case "U"* se ejecuta. Dentro de ella existen dos opciones. Si el siguiente carácter leído es un 1, se enciende el anunciador usando la sentencia *bitSet()*. Por el contrario, si lo que se recibe es un 0, el led se apaga usando *bitClear()*. Dentro de estas dos sentencias se debe escribir el *SR\_OUT* y el pin al que está conectado el led. En este caso, se corresponde con el pin *Q0* del *SR\_OUT\_1*, como se puede ver también en la *tabla 11* del anexo, sección 11.14. Por último, se refresca el *SR* haciendo uso de la ya explicada función *update\_SR\_OUT\_1()*.

```
1663 // *CHARACTER B FUNCTION*
1664
1665 void B() { // First identifier has been B
1666
1667     CodeIn = getChar(); // Get the second identifier
1668
1669 switch(CodeIn) {
1670
1671     case 'U': // CODE BU; LED STANDBY_HYD_LOW_QUANTITY
1672         valueBU = getChar();
1673         if (valueBU == '1'){ bitSet(SR_OUT_1,0); }
1674         else{ bitClear(SR_OUT_1,0); }
1675         update_SR_OUT_1();
1676         break;
1677
1678     case 'V': // CODE BV; LED STANDBY_HYD_LOW_PRESSURE
1679         valueBV = getChar();
1680         if (valueBV == '1'){ bitSet(SR_OUT_1,1); }
1681         else{ bitClear(SR_OUT_1,1); }
1682         update_SR_OUT_1();
1683         break;
```

## 7.5 Display LED 7 segmentos

Como se vio previamente, un display LED no deja de ser un conjunto de leds. Por ello, la forma de trabajar con ellos es similar a los leds de 5 mm. Para la explicación se considerarán los displays del PANEL\_C2\_1. Cada uno tiene asociado, de nuevo, un identificador. De izquierda a derecha y de arriba abajo, estos son CS, CU, CV, CW y CX.

A la hora de definir las variables, en este caso se hace necesario definir varias. En primer lugar, se crean variables de tipo *string* cuya función será almacenar los caracteres numéricos del valor a mostrar. A continuación, se definen variables de tipo *int*, donde se convertirán esos caracteres en números enteros. Posteriormente, se establecen variables de tipo *long*, que son las que se utilizan dentro de la función que configura los displays. Por último, se definen otras variables que serán necesarias en esta función, entre las que se encuentran aquellas que posibilitan definir valores negativos o eliminar los ceros a la izquierda de los displays.

```

157 // *PANEL C2-1 VARIABLES*
158
159 // **LEDS**
160
161 int valueCY = '0', valueCZ = '0', valueDA = '0'; //Control LED state
162 String valueCS, valueCU, valueCV, valueCW, valueCX; //Control LED numbers on the display, string value
163 int valueCSI = 0, valueCUi = 0, valueCVi = 0, valueCWi = 0, valueCXi = 0; //Control LED numbers on the display, integer
164
165 long display_num_1, display_num_2, display_num_3, display_num_4, display_num_5; //number shown in each digit
166 int display_pos; //shifted position value
167 int shift_number_1 [3]={}, shift_number_2 [3]={}, shift_number_3 [2]={}, shift_number_4 [3]={}, shift_number_5 [3]={};
168 int neg_num_1=0, neg_num_4=0;

```

Al igual que se hacía con los leds de 5 mm, las actualizaciones del estado de los displays llegan a través del monitor serial. Esta vez, en vez de ser un 0 ó 1, es el valor que se debe mostrar en el display. Por tanto, dentro de la función *C()*, en el *case "S"*, se lee el valor del display DC\_AMPS y se almacena en la variable *valueCS*. Como no se sabe la longitud de ese número y, por tanto, el número de caracteres a leer, se utiliza la sentencia *Serial.readStringUntil('\n')* que continúa leyendo valores hasta que detecta un salto de línea, donde se genera ese código. Los valores al salir del monitor serial vienen en formato texto y deben pasarse a números para poder operar con ellos. Por ello se utiliza la sentencia *valueCS.toInt()* que transforma los anteriores caracteres en un número entero.

```

1845 case 'S': // CODE CS; DISPLAY DC_AMPS
1846     valueCS = "";
1847     valueCS = Serial.readStringUntil('\n');
1848     valueCSI = valueCS.toInt();
1849     break;

```

A lo largo de la sección *loop*, se utiliza la función *refreshDisplay()* que es la que gestiona la actualización de valores en los displays. Esta es bastante extensa, por lo que se comentará poco a poco.

La función comienza con el traspaso del valor a mostrar desde la variable *valueCSI* a la variable *display\_num\_1*, que como se vio anteriormente es de tipo *long*. A continuación, se identifica si ese número es negativo, y si lo es, se le quita el signo y se activa la variable *neg\_num\_1*, que luego hará que se muestre en el display un signo negativo. Las únicas cifras que en principio pueden ser negativas son las del display 1 y 4, al ser ambos medidores de amperaje (*DC AMPS* y *AC AMPS*).

```
3654 // *REFRESH DISPLAY FUNCTION*
3655
3656 void refreshDisplay(){
3657
3658     display_num_1 = valueCSI;
3659     display_num_2 = valueCUI;
3660     display_num_3 = valueCVi;
3661     display_num_4 = valueCWi;
3662     display_num_5 = valueCXi;
3663
3664 if(display_num_1 < 0){
3665     display_num_1 = abs(display_num_1);
3666     neg_num_1 = 1;
3667 }
3668 else{
3669     neg_num_1 = 0;
3670 }
3671
3672 if(display_num_4 < 0){
3673     display_num_4 = abs(display_num_4);
3674     neg_num_4 = 1;
3675 }
3676 else{
3677     neg_num_4 = 0;
3678 }
```

Como el mayor número de dígitos que tienen estos displays es 3, el bucle que se define a continuación toma ese número de iteraciones mediante la variable *display\_pos*. El propósito de este bucle es coger las cifras a mostrar, separarlas en dígitos individuales y alojarlas en una posición concreta de un array de datos, dimensionalizado en función del número de dígitos del display en la sección de variables.

```
3680 for (display_pos=0; display_pos<3; display_pos++){
3681
3682     shift_number_1[2-display_pos]= display_num_1%10;
3683     shift_number_2[2-display_pos]= display_num_2%10;
3684     shift_number_4[2-display_pos]= display_num_4%10;
3685     shift_number_5[2-display_pos]= display_num_5%10;
3686
3687     display_num_1 = display_num_1/10;
3688     display_num_2 = display_num_2/10;
3689     display_num_4 = display_num_4/10;
3690     display_num_5 = display_num_5/10;
3691
3692 }
```

El display 3, al ser el único que tiene dos dígitos, tiene un bucle separado, donde el número de iteraciones se reduce a dos.

```
3694     for (display_pos=0; display_pos<2; display_pos++) {
3695
3696         shift_number_3[1-display_pos]= display_num_3%10;
3697         display_num_3 = display_num_3/10;
3698
3699     }
```

El siguiente paso consiste en eliminar de los displays los posibles ceros a la izquierda, ya que en el *Overhead Panel* real estos no se muestran. Para ello, se utilizan las siguientes sentencias, donde si se identifica que el valor es un cero, se sustituye por un 10, dado que, como se verá más adelante, se establecerá un *case “10”* que contendrá los ajustes para dejar ese dígito del display apagado. Si el número a mostrar es exclusivamente 0, este no se elimina. Por ello, los displays de 3 dígitos tienen dos sentencias *if* encadenadas, y el display de 2 dígitos solo una.

```
3701     if (shift_number_1[0] == 0 || shift_number_1[0] == 10){ //Eliminate 0 at the left
3702         shift_number_1[0]=10;
3703     if(shift_number_1[1] == 0 || shift_number_1[1] == 10){
3704         shift_number_1[1]=10;
3705     }
3706 }
3707
3708     if (shift_number_2[0] == 0 || shift_number_2[0] == 10){ //Eliminate 0 at the left
3709         shift_number_2[0]=10;
3710     if(shift_number_2[1] == 0 || shift_number_2[1] == 10){
3711         shift_number_2[1]=10;
3712     }
3713 }
3714
3715     if (shift_number_3[0] == 0 || shift_number_3[0] == 10){ //Eliminate 0 at the left
3716         shift_number_3[0]=10;
3717     }
3718
3719     if (shift_number_4[0] == 0 || shift_number_4[0] == 10){ //Eliminate 0 at the left
3720         shift_number_4[0]=10;
3721     if(shift_number_4[1] == 0 || shift_number_4[1] == 10){
3722         shift_number_4[1]=10;
3723     }
3724 }
3725
3726     if (shift_number_5[0] == 0 || shift_number_5[0] == 10){ //Eliminate 0 at the left
3727         shift_number_5[0]=10;
3728     if(shift_number_5[1] == 0 || shift_number_5[1] == 10){
3729         shift_number_5[1]=10;
3730     }
3731 }
```

La siguiente parte de la función gestiona la escritura de los valores alojados en las variables *shift\_number* en los displays LED. Como se comentó en el apartado 5.5.6, se utilizará el multiplexado. Dado que el número máximo de dígitos es 3, habrá tres iteraciones. Por ello, en la primera iteración el display de dos dígitos permanecerá apagado, mostrando valores exclusivamente en la segunda y la tercera.

Recuérdese que para mostrar un dígito el pin correspondiente tiene que estar en bajo, mediante la sentencia *bitClear()*, mientras que para apagarlo debe estar en alto, sentencia *bitSet()*. Si por el contrario se gestionan los diodos de un dígito en concreto, estos se encienden cuando están en alto y se apagan cuando están en bajo. Esta lógica es la propia de los displays de cátodo común, siendo la inversa para los de ánodo común.

```

3733 ⊞    for (display_pos=0; display_pos<3; display_pos++){
3734
3735 ⊞      if (display_pos == 0){bitClear(SR_OUT_4,7), bitSet(SR_OUT_5,0), bitSet(SR_OUT_5,1);
3736          bitClear(SR_OUT_6,1), bitSet(SR_OUT_6,2), bitSet(SR_OUT_6,3);
3737          bitSet(SR_OUT_7,3), bitSet(SR_OUT_7,4);
3738          bitClear(SR_OUT_8,4), bitSet(SR_OUT_8,5), bitSet(SR_OUT_8,6);
3739          bitClear(SR_OUT_9,6), bitSet(SR_OUT_9,7), bitSet(SR_OUT_10,0);}
3740 ⊞      if (display_pos == 1){bitSet(SR_OUT_4,7), bitClear(SR_OUT_5,0), bitSet(SR_OUT_5,1);
3741          bitSet(SR_OUT_6,1), bitClear(SR_OUT_6,2), bitSet(SR_OUT_6,3);
3742          bitClear(SR_OUT_7,3), bitSet(SR_OUT_7,4);
3743          bitSet(SR_OUT_8,4), bitClear(SR_OUT_8,5), bitSet(SR_OUT_8,6);
3744          bitSet(SR_OUT_9,6), bitClear(SR_OUT_9,7), bitSet(SR_OUT_10,0);}
3745 ⊞      if (display_pos == 2){bitSet(SR_OUT_4,7), bitSet(SR_OUT_5,0), bitClear(SR_OUT_5,1);
3746          bitSet(SR_OUT_6,1), bitSet(SR_OUT_6,2), bitClear(SR_OUT_6,3);
3747          bitSet(SR_OUT_7,3), bitClear(SR_OUT_7,4);
3748          bitSet(SR_OUT_8,4), bitSet(SR_OUT_8,5), bitClear(SR_OUT_8,6);
3749          bitSet(SR_OUT_9,6), bitSet(SR_OUT_9,7), bitClear(SR_OUT_10,0);}

```

A continuación, se implementan los *cases* que contienen las configuraciones para mostrar las cifras del 0 al 9, dejar el dígito apagado y mostrar el símbolo negativo. Recordando la nomenclatura que seguían los diodos LED del display, se ofrece la siguiente tabla donde se muestra qué valores tienen que tomar los segmentos para mostrar los anteriores valores.

Tabla 9. Código 7 segmentos cátodo común

VALOR	A	B	C	D	E	F	G
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1
	0	0	0	0	0	0	0
-	0	0	0	0	0	0	1

A modo de ejemplo se ofrece el código que se usaría para configurar el *case "0"* y el *case "1"* del primer display, siguiendo el resto de *los cases* la misma estructura.

```

3751     switch( shift_number_1[display_pos] ){
3752
3753         case 0:
3754             bitSet(SR_OUT_4,0); // A segment
3755             bitSet(SR_OUT_4,1); // B segment
3756             bitSet(SR_OUT_4,2); // C segment
3757             bitSet(SR_OUT_4,3); // D segment
3758             bitSet(SR_OUT_4,4); // E segment
3759             bitSet(SR_OUT_4,5); // F segment
3760             bitClear(SR_OUT_4,6); // G segment
3761         break;
3762
3763         case 1:
3764             bitClear(SR_OUT_4,0); // A segment
3765             bitSet(SR_OUT_4,1); // B segment
3766             bitSet(SR_OUT_4,2); // C segment
3767             bitClear(SR_OUT_4,3); // D segment
3768             bitClear(SR_OUT_4,4); // E segment
3769             bitClear(SR_OUT_4,5); // F segment
3770             bitClear(SR_OUT_4,6); // G segment
3771         break;

```

En caso de que el número sea negativo se utiliza la siguiente sentencia para mostrar el signo negativo a la izquierda de la cifra.

```

4318     if ( (neg_num_1 == 1) && (display_pos == 0) ){ //print a dash (-) on the first digit
4319
4320         bitClear(SR_OUT_4,0); // A segment
4321         bitClear(SR_OUT_4,1); // B segment
4322         bitClear(SR_OUT_4,2); // C segment
4323         bitClear(SR_OUT_4,3); // D segment
4324         bitClear(SR_OUT_4,4); // E segment
4325         bitClear(SR_OUT_4,5); // F segment
4326         bitSet(SR_OUT_4,6); // G segment
4327
4328     }
4329
4330     if ( (neg_num_4 == 1) && (display_pos == 0) ){ //print a dash (-) on the first digit
4331
4332         bitClear(SR_OUT_7,5); // A segment
4333         bitClear(SR_OUT_7,6); // B segment
4334         bitClear(SR_OUT_7,7); // C segment
4335         bitClear(SR_OUT_8,0); // D segment
4336         bitClear(SR_OUT_8,1); // E segment
4337         bitClear(SR_OUT_8,2); // F segment
4338         bitSet(SR_OUT_8,3); // G segment
4339
4340     }

```

La última parte de la función incorpora las sentencias necesarias para simular correctamente el PANEL\_C2\_1, ya que, en función de las posiciones de los selectores de DC y AC, algunos displays mostrarán su valor o no. Adicionalmente se refresca la pantalla mediante la función *update\_SR\_OUT\_4()* y se dejan pasar unos milisegundos para que el ojo humano pueda ver la cifra como si estuvieran todos los dígitos encendidos a la vez.

```
4342 if (PRE_SELECTOR_DC_SOURCE ==0){bitSet(SR_OUT_4,7); bitSet(SR_OUT_5,0); bitSet(SR_OUT_5,1);
4343                                         bitSet(SR_OUT_7,3); bitSet(SR_OUT_7,4);}
4344 if (PRE_SELECTOR_DC_SOURCE ==1){bitSet(SR_OUT_4,7); bitSet(SR_OUT_5,0); bitSet(SR_OUT_5,1);}
4345 if (PRE_SELECTOR_DC_SOURCE ==2){bitSet(SR_OUT_4,7); bitSet(SR_OUT_5,0); bitSet(SR_OUT_5,1);}
4346
4347 if (PRE_SELECTOR_AC_SOURCE ==0){bitSet(SR_OUT_8,4); bitSet(SR_OUT_8,5); bitSet(SR_OUT_8,6);}
4348 if (PRE_SELECTOR_AC_SOURCE ==2){bitSet(SR_OUT_6,1); bitSet(SR_OUT_6,2); bitSet(SR_OUT_6,3);}
4349 if (PRE_SELECTOR_AC_SOURCE ==3){bitSet(SR_OUT_6,1); bitSet(SR_OUT_6,2); bitSet(SR_OUT_6,3);}
4350 if (PRE_SELECTOR_AC_SOURCE ==4){bitSet(SR_OUT_6,1); bitSet(SR_OUT_6,2); bitSet(SR_OUT_6,3);}
4351 if (PRE_SELECTOR_AC_SOURCE ==5){bitSet(SR_OUT_8,4); bitSet(SR_OUT_8,5); bitSet(SR_OUT_8,6);}
4352 if (PRE_SELECTOR_AC_SOURCE ==6){bitSet(SR_OUT_6,1); bitSet(SR_OUT_6,2); bitSet(SR_OUT_6,3);
4353                                         bitSet(SR_OUT_8,4); bitSet(SR_OUT_8,5); bitSet(SR_OUT_8,6);
4354                                         bitSet(SR_OUT_9,6); bitSet(SR_OUT_9,7); bitSet(SR_OUT_10,0);}
4355
4356     delay(2); //delay as long as necessary to get time to watch on display
4357     update_SR_OUT_4();
4358
4359 }
4360
4361 }//End refreshDisplay()
```

## 7.6 Servomotor

El servomotor es el último elemento de tipo salida que se va a explicar. Lo primero que se hace es incluir la librería de Arduino *Servo.h* y crear una clase para estandarizar el uso de los servomotores y no tener que repetir líneas de forma innecesaria.

Esta clase define las variables que utiliza y a continuación establece las sentencias necesarias para asociar el servo a un pin de Arduino, activarlo o desactivarlo y para actualizar la posición de este.

```
11 // --- LIBRARIES SECTION --- //
12
13 #include <Servo.h>
14
15
16
17 // --- CLASSES SECTION --- //
18
19 // *UPDATE SERVO POSITION CLASS*
20
21 class SERVO {
22
23     Servo servo; // the servo
24     int updatePos; // desired new position
25     int updateInterval; // Interval between updates
26     int servo_pin; // Pin the servo must be attached to
27     unsigned long lastUpdateServo = millis(); // Last update of position
28
29 public:
30     SERVO(int pin){
31         servo_pin = pin;
32     }
33
34     void Attach(){
35         servo.attach(servo_pin);
36     }
37
38     void Detach(){
39         servo.detach();
40     }
41
42     void Update(int interval, int pos){
43         updateInterval = interval;
44         updatePos = pos;
45         if( (millis() - lastUpdateServo) > updateInterval ){ //time to update
46             servo.write(updatePos);
47             lastUpdateServo = millis();
48         }
49     }
50 };
```

Para la explicación se considerará el servo asociado al reloj *EGT\_APU* en el *PANEL\_C2\_4*. En la sección de variables se definen aquellas que se van a usar para integrarlo. Estas incluyen el pin de Arduino al que está conectado el servo, la posición inicial del servo, la variable que recoge la posición del monitor serial en formato texto y aquella donde se transformara a formato *int*. Estas reciben el nombre de *valueDOLLARC*, pues el identificador de este servo es el código “\$C”

```
254 // **SERVOS**  
255  
256 SERVO SERVO_EGT_APU(4); // Attached to pin D4  
257 int pos_servo_egt_apu = 10; // Servo initial position after cool and dark  
258 String valueDOLLARC; // Entrance to servo position  
259 int valueDOLLARCi = 0; // valueDOLLARC string converted to integer (initial before cool and dark)
```

En la sección *setup* se inicializa en su posición inicial. Para ello se conecta mediante la primera sentencia y se actualiza su posición. La primera cifra dentro del paréntesis indica un retardo opcional hasta ejecutar el giro y la segunda la posición a la que se debe mover.

```
617 // **PANEL C2-4 COLD AND DARK**  
618  
619 // ***SERVOS***  
620  
621 SERVO_EGT_APU.Attach();  
622 delay(100);  
623 SERVO_EGT_APU.Update(0, 10);
```

Dentro de la sección *loop*, está como se vio previamente, la función *DOLLAR()*, que es similar a la que se explicó, *B()*. Dentro de esta, y en el *case “C”*, se encuentran las sentencias de lectura del monitor serial y la actualización de posición para este reloj.

Se comienza con la lectura de la posición del reloj en el simulador y se almacena en la variable *valueDOLLARC*. A continuación, se transforma a número entero y se almacena en *valueDOLLARCi*. Ahora es necesario transformar esa posición, que en el caso de este reloj tiene un rango de 0 a 1000, en un giro de 0 a 270 grados (aunque mediante el mecanismo de engranajes multiplicadores el servomotor lo logrará con un giro de 180 grados). Para conseguirlo se utiliza la sentencia *map*, que coge el valor de *valueDOLLARCi* y, mediante la definición de los intervalos que están a continuación, genera el valor equivalente, almacenado en la variable *pos\_servo\_egt\_apu*. Después de una calibración, se ha identificado que la posición 0 se corresponde con un giro de 10 grados y la 1000 con 162 grados. Por ello, dentro de la sentencia *map* se configuran estos intervalos. Esta calibración se deberá hacer para cada uno de los relojes.

Por último, si la variable *servo\_activated* vale 1, que esto sucede cuando el avión está energizado, haciendo uso de la clase creada al inicio del código, se activa el servomotor y se actualiza su posición.

```
2541     case 'C': // CODE $C; SERVO_EGT_APU
2542
2543         valueDOLLARC = "";
2544         valueDOLLARC = Serial.readStringUntil("\n");
2545         valueDOLLARCi = valueDOLLARC.toInt();
2546         pos_servo_egt_apu = map(valueDOLLARCi, 0, 1000, 10, 162);
2547
2548     if (servo_activated == 1){
2549         SERVO_EGT_APU.Attach();
2550         SERVO_EGT_APU.Update(10, pos_servo_egt_apu);
2551     }
2552     else if (servo_activated == 0){
2553         break;
2554     }
2555     break;
```

## 7.7 Interruptor

El interruptor es el primer elemento de entrada y el básico para entender los que vendrán a continuación. Este usa dos variables (o tres si tiene 3 posiciones), una que almacena la posición previa y otra que almacena el valor leído por los SR\_IN (74HC165). Para la explicación se considerarán los interruptores del PANEL\_C1\_1, que tiene tanto de 2, como de 3 posiciones.

```
66     int PRE_SWITCH_YAW_DAMPER = 0, PRE_SWITCH_SPOILER_A = 1, PRE_SWITCH_SPOILER_B = 1;
67     int PRE_SWITCH_ALTERNATE_FLAPS = 0, PRE_SWITCH_STBY_RUD_A = 2;
68     int PRE_SWITCH_STBY_RUD_B = 2, PRE_SWITCH_FLAPS_POS = 1;
69
70     int SWITCH_YAW_DAMPER, SWITCH_SPOILER_A, SWITCH_SPOILER_B, SWITCH_ALTERNATE_FLAPS;
71     int SWITCH_STBY_RUD_A1, SWITCH_STBY_RUD_A2, SWITCH_STBY_RUD_B1, SWITCH_STBY_RUD_B2;
72     int SWITCH_FLAPS_POS_1, SWITCH_FLAPS_POS_2;
```

Pasando directamente a la sección *loop*, pues en la de *setup* no se necesita escribir nada, lo primero que se ve son las sentencias de refresco de los estados de los SR\_IN. Lo que hacen estas es leer los pines de entrada y, mediante la comparación de las variables nuevas y antiguas, identificar si ha habido alguna modificación respecto a la última iteración. Si esto ha sucedido, se llama a la función *modify\_switch\_values\_1()* y se actualizan las variables antiguas.

```
1198 // *SWITCHES*
1199
1200 // **PANEL C1-1 SWITCHES**
1201
1202 // ***SHIFT REGISTERS (IN) ***
1203
1204     SR_IN_5 = update_SR_IN_5();
1205     SR_IN_1 = update_SR_IN_1();
1206
1207 if( (SR_IN_1 != OLD_SR_IN_1) || (SR_IN_5 != OLD_SR_IN_5) ){
1208
1209     modify_switch_values_1();
1210     OLD_SR_IN_5 = SR_IN_5;
1211     OLD_SR_IN_1 = SR_IN_1;
1212
1213 }
```

La función *modify\_switch\_values\_1()* es la encargada de leer los estados de los interruptores a través de los valores de las variables *SR\_IN\_1* y *SR\_IN\_5* y, de acuerdo ellos, actualizar el valor de las variables definidas en la parte superior de esta página.

Dentro de la función, lo primero que se aprecia es un bucle con el mismo número de iteraciones que pines de entrada tiene la cadena, mediante la variable *DATA\_WIDTH\_1* (56 iteraciones en este caso).

A continuación, se define un *switch* con 54 *cases* (consecuente con la *tabla 12* del anexo, sección 11.15). Dentro de cada uno de ellos se encuentran las sentencias que leen el estado de los pines de entrada a través de las variables *SR\_IN\_1* y *SR\_IN\_5*. Si el valor leído es un 1 (tras haber sido seleccionada la posición en la que se encuentra dentro de la variable mediante la sentencia *SR\_IN\_1 >> i*), se le asigna a la variable de estado del interruptor un 1. Por el contrario, si es un 0, la variable toma ese valor. Los interruptores con tres posiciones, como el caso mostrado abajo, tienen dos variables asociadas, en este caso *SWITCH\_STBY\_RUD\_A1* y *SWITCH\_STBY\_RUD\_A2*. Esto servirá para identificar en qué posición se encuentra el interruptor más adelante.

```

2868 // *MODIFY SWITCH VALUES 1 FUNCTION*
2869
2870 void modify_switch_values_1(){ //Modifies the SWITCH values associated with SR_IN_1 to SR_IN_7
2871
2872 for(int i = 0; i < DATA_WIDTH_1; i++){
2873
2874 switch(i){
2875
2876     case 0: // PIN D0_1; SWITCH_STBY_RUD_A1; CODE HS
2877         if((SR_IN_1 >> i) & 1){SWITCH_STBY_RUD_A1 = 1;}
2878         else{SWITCH_STBY_RUD_A1 = 0;}
2879     break;
2880
2881     case 1: // PIN D1_1; SWITCH_STBY_RUD_A2; CODE HS
2882         if((SR_IN_1 >> i) & 1){SWITCH_STBY_RUD_A2 = 1;}
2883         else{SWITCH_STBY_RUD_A2 = 0;}
2884     break;
2885
2886     case 2: // PIN D2_1; SWITCH_STBY_RUD_B1; CODE HT
2887         if((SR_IN_1 >> i) & 1){SWITCH_STBY_RUD_B1 = 1;}
2888         else{SWITCH_STBY_RUD_B1 = 0;}
2889     break;

```

Nótese que cuando se cambia a leer la variable *SR\_IN\_5* es necesario cambiar la sentencia de selección de posición, de manera que esa lectura se realice de forma correcta.

```

3021     case 31: // PIN D7_4; SWITCH_FIXED_LDG_LIGHTS_L; CODE Ad
3022         if((SR_IN_1 >> i) & 1){SWITCH_FIXED_LDG_LIGHTS_L = 1;}
3023         else{SWITCH_FIXED_LDG_LIGHTS_L = 0;}
3024     break;
3025
3026     case 32: // PIN D0_5; SWITCH_FIXED_LDG_LIGHTS_R; CODE Ae
3027         if((SR_IN_5 >> (i-32)) & 1){SWITCH_FIXED_LDG_LIGHTS_R = 1;}
3028         else{SWITCH_FIXED_LDG_LIGHTS_R = 0;}
3029     break;
3030
3031     case 33: // PIN D1_5; SWITCH_RWY_LIGHT_L; CODE Af
3032         if((SR_IN_5 >> (i-32)) & 1){SWITCH_RWY_LIGHT_L = 1;}
3033         else{SWITCH_RWY_LIGHT_L = 0;}
3034     break;

```

Una vez entendida la función *modify\_switch\_values\_1()* solo falta explicar cómo se hace la escritura en el monitor serial del identificador y el estado de los interruptores para que se transmita a través de Cockpit-X al simulador.

Se va a explicar primero el caso de los interruptores de dos posiciones. Recuérdese de los esquemas eléctricos que, si un interruptor está en la posición 0, es decir, la palanca del interruptor está en la parte superior o en la izquierda, por el cable se lee un 1 lógico (5V). Por el contrario, si el interruptor está en la posición 1, es decir, la palanca está en la parte inferior o a la derecha, por el cable se lee un 0 (tierra). Teniendo esto en cuenta, se comienza definiendo dos sentencias *if*. Estas detectan el estado actual del interruptor y, si es diferente al que tenían antes, se ejecutan. Dentro de ellas, si han pasado más de 10 ms desde la última actualización (esto se usa para eliminar el efecto rebote mecánico de los interruptores), escribe en el monitor serial el identificador y el número del estado, actualiza el estado anterior y refresca el conteo de milisegundos. Cockpit-X, al recibir el identificador "HY1", lleva en el simulador el interruptor *SWITCH\_YAW\_DAMPER* (identificador "HY") a la posición 1.

```
1237     // ***SWITCHES LOGIC***  
1238  
1239     if( ((SWITCH_YAW_DAMPER) == 0) && (PRE_SWITCH_YAW_DAMPER == 0) ){  
1240         if( (millis()-lastUpdateSwitch) > 10){  
1241             Serial.println("HY1");  
1242             PRE_SWITCH_YAW_DAMPER = 1;  
1243             lastUpdateSwitch = millis();  
1244         }  
1245     }  
1246  
1247     if( ((SWITCH_YAW_DAMPER) == 1) && (PRE_SWITCH_YAW_DAMPER == 1) ){  
1248         if( (millis()-lastUpdateSwitch) > 10){  
1249             Serial.println("HY0");  
1250             PRE_SWITCH_YAW_DAMPER = 0;  
1251             lastUpdateSwitch = millis();  
1252         }  
1253     }
```

En el caso de los interruptores de 3 posiciones, si el interruptor está en la posición 0, es decir, la palanca del interruptor está en la parte superior o en la izquierda, por el primer cable se lee un 0 y por el segundo un 1. Si el interruptor está en la posición 1, es decir, la palanca está en el centro, por ambos cables se lee 0. Por último, si esta se encuentra en la posición 2, es decir, la palanca está en la parte inferior o a la derecha, por el primer cable se lee un 1 y por el segundo un 0. Teniendo en cuenta esta lógica, en el caso de los interruptores de tres posiciones se necesitan tres sentencias *if*. El contenido de estas es equivalente al que se vio para los interruptores de 2 posiciones.

```
1251 if( ((SWITCH_STBY_RUD_A1) == 0) && (SWITCH_STBY_RUD_A2) == 1 && (PRE_SWITCH_STBY_RUD_A == 1) ){
1252     if( (millis()-lastUpdateSwitch) > 10){
1253         Serial.println("HS0");
1254         PRE_SWITCH_STBY_RUD_A = 0;
1255         lastUpdateSwitch = millis();
1256     }
1257 }
1258
1259 if( ((SWITCH_STBY_RUD_A1) == 0) && (SWITCH_STBY_RUD_A2) == 0 && ( (PRE_SWITCH_STBY_RUD_A == 0) || (PRE_SWITCH_STBY_RUD_A == 2) ) ){
1260     if( (millis()-lastUpdateSwitch) > 10){
1261         Serial.println("HS1");
1262         PRE_SWITCH_STBY_RUD_A = 1;
1263         lastUpdateSwitch = millis();
1264     }
1265 }
1266
1267 if( ((SWITCH_STBY_RUD_A1) == 1) && (SWITCH_STBY_RUD_A2) == 0 && (PRE_SWITCH_STBY_RUD_A == 1) ){
1268     if( (millis()-lastUpdateSwitch) > 10){
1269         Serial.println("HS2");
1270         PRE_SWITCH_STBY_RUD_A = 2;
1271         lastUpdateSwitch = millis();
1272     }
1273 }
```

## 7.8 Pulsador tipo botón

Los botones se implementan igual que los interruptores de dos posiciones, siendo en este caso una de ellas, la de pulsación, momentánea. Esta es la definición de variables del botón `BUTTON_MAINT` situado en el `PANEL_C2_1`.

```
170 // **SWITCHES**
171
172     int PRE_BUTTON_MAINT = 0;
173     int PRE_SELECTOR_DC_SOURCE = 2, PRE_SELECTOR_AC_SOURCE = 3;
174     int PRE_SWITCH_MAIN_BATTERY = 0, PRE_SWITCH CAB_UTIL = 1, PRE_SWITCH_IKE_PASS_SEAT = 1;
175
176     int BUTTON_MAINT;
```

La función `modify_switch_value_8()` es equivalente a la `modify_switch_value_1()`, pero en este caso se utiliza otra cadena de 74HC165, que va desde el `SR_IN_8` al `SR_IN_14`.

```
3156 // *MODIFY SWITCH VALUES 8 FUNCTION*
3157
3158 void modify_switch_values_8(){ //Modifies the SWITCH values associated with SR_IN_8 to SR_IN_14
3159
3160     for(int i = 0; i < DATA_WIDTH_8; i++){
3161
3162         switch(i){
3163
3164             case 0: // PIN D0_8; BUTTON_MAINT; CODE IM
3165                 if((SR_IN_8 >> i) & 1){BUTTON_MAINT = 1;}
3166                 else{BUTTON_MAINT = 0;}
3167             break;
3168     }
3169 }
```

Cuando el botón se pulsa (posición 1) por el cable se detecta un 1 lógico y cuando se suelta (posición 0) un 0 lógico.

```
1375 // **PANEL C2-1 SWITCHES**
1376
1377     // ***SWITCHES LOGIC***
1378
1379     if( ((BUTTON_MAINT) == 0) && (PRE_BUTTON_MAINT == 1) ) {
1380         if( (millis()-lastUpdateSwitch) > 10){
1381             Serial.println("IM0");
1382             PRE_BUTTON_MAINT = 0;
1383             lastUpdateSwitch = millis();
1384         }
1385     }
1386
1387     if( ((BUTTON_MAINT) == 1) && (PRE_BUTTON_MAINT == 0) ) {
1388         if( (millis()-lastUpdateSwitch) > 10){
1389             Serial.println("IM1");
1390             PRE_BUTTON_MAINT = 1;
1391             lastUpdateSwitch = millis();
1392         }
1393     }
1394 }
```

## 7.9 Selector rotatorio

Lo mismo sucede con los selectores rotatorios, que son como los interruptores, pero con múltiples posiciones, seleccionadas mediante el giro de su eje. Como se comentó, en el caso de selectores con tres posiciones, se puede ahorrar el cable de la posición central. El código es exactamente el mismo que el de los interruptores de 3 posiciones. Por ello, se va a explicar los selectores de más de tres posiciones para ver las posibles diferencias.

Como se ha hecho otras veces, se comienza definiendo sus variables. Al igual que con los interruptores, se tendrá una variable de estado por cada posición y una por cada selector. Para la explicación se considerará el selector de cuatro posiciones *SELECTOR\_WIPER\_L*, del *PANEL\_C2\_4*.

```
248     // **SWITCHES**
249
250     int PRE_SELECTOR_WIPER_L = 0;
251
252     int SELECTOR_WIPER_L_0, SELECTOR_WIPER_L_1, SELECTOR_WIPER_L_2, SELECTOR_WIPER_L_3;
```

El uso de la función *modify\_switch\_value\_8()* es el mismo que el de los interruptores y botones.

```
3334         case 34: // PIN D2_12; SELECTOR_WIPER_L_0; CODE JB
3335             if((SR_IN_12 >> (i-32)) & 1){SELECTOR_WIPER_L_0 = 1;}
3336             else{SELECTOR_WIPER_L_0 = 0;}
3337             break;
3338
3339         case 35: // PIN D3_12; SELECTOR_WIPER_L_1; CODE JB
3340             if((SR_IN_12 >> (i-32)) & 1){SELECTOR_WIPER_L_1 = 1;}
3341             else{SELECTOR_WIPER_L_1 = 0;}
3342             break;
3343
3344         case 36: // PIN D4_12; SELECTOR_WIPER_L_2; CODE JB
3345             if((SR_IN_12 >> (i-32)) & 1){SELECTOR_WIPER_L_2 = 1;}
3346             else{SELECTOR_WIPER_L_2 = 0;}
3347             break;
3348
3349         case 37: // PIN D5_12; SELECTOR_WIPER_L_3; CODE JB
3350             if((SR_IN_12 >> (i-32)) & 1){SELECTOR_WIPER_L_3 = 1;}
3351             else{SELECTOR_WIPER_L_3 = 0;}
3352             break;
```

A la hora de definir las sentencias *if* que escriben en el monitor serial se sigue la misma estructura que las de los interruptores, añadiendo más líneas en función del número total de posiciones del selector.

```

1463 // **PANEL C2-4 SWITCHES**
1464
1465 // ***SWITCHES LOGIC***
1466
1467 if( ((SELECTOR_WIPER_L_0) == 1) && (PRE_SELECTOR_WIPER_L == 1) ){
1468     if( (millis()-lastUpdateSwitch) > 10){
1469         Serial.println("JB0");
1470         PRE_SELECTOR_WIPER_L = 0;
1471         lastUpdateSwitch = millis();
1472     }
1473 }
1474
1475 if( ((SELECTOR_WIPER_L_1) == 1) && ((PRE_SELECTOR_WIPER_L == 2) || (PRE_SELECTOR_WIPER_L == 0)) ){
1476     if( (millis()-lastUpdateSwitch) > 10){
1477         Serial.println("JB1");
1478         PRE_SELECTOR_WIPER_L = 1;
1479         lastUpdateSwitch = millis();
1480     }
1481 }
1482
1483 if( ((SELECTOR_WIPER_L_2) == 1) && ((PRE_SELECTOR_WIPER_L == 3) || (PRE_SELECTOR_WIPER_L == 1)) ){
1484     if( (millis()-lastUpdateSwitch) > 10){
1485         Serial.println("JB2");
1486         PRE_SELECTOR_WIPER_L = 2;
1487         lastUpdateSwitch = millis();
1488     }
1489 }
1490
1491 if( ((SELECTOR_WIPER_L_3) == 1) && (PRE_SELECTOR_WIPER_L == 2) ){
1492     if( (millis()-lastUpdateSwitch) > 10){
1493         Serial.println("JB3");
1494         PRE_SELECTOR_WIPER_L = 3;
1495         lastUpdateSwitch = millis();
1496     }
1497 }

```

Por último, para implementar los selectores rotatorios automáticos del PANEL\_C1\_4 se recurre a la opción de Cockpit-X de generar un código por el monitor serial, como si de una salida se tratase, mostrando la posición en la que se encuentra el selector del simulador. Para ello se usan los códigos “Zb” y “Zc” seguidos de un valor del 0 al 3 que indica la posición en la que se encuentran. Por ello, lo único que hay que hacer es que, si se lee en el monitor serial “Zb0” o “Zc0”, cuando se lea a continuación “Zb1” o “Zc1” activar el servomotor asociado al mecanismo, como se vio en la sección 7.6, para que el mecanismo lleve los selectores físicos a esa posición.

```

2519 // *CHARACTER Z FUNCTION*
2520
2521 void Z(){ // First identifier has been Z
2522
2523     CodeIn = getChar(); // Get the second identifier
2524
2525     switch(CodeIn) {
2526
2527         case 'b': // CODE Zb; SERVO SELECTOR_ENGINE_STATOR
2528             prevalueZb = valueZb;
2529             valueZb = getChar();
2530             if ( (valueZb == '1') && (prevalueZb == '0') ){
2531
2532                 SERVO_SELECTOR_ENGINE_STARTOR.Attach();
2533                 SERVO_SELECTOR_ENGINE_STARTOR.Update(150, 70);
2534                 delay(500);
2535                 SERVO_SELECTOR_ENGINE_STARTOR.Update(0, 105);
2536
2537             }
2538             break;

```

## 7.10 Codificador rotatorio

En el *Overhead Panel* hay dos codificadores rotatorios, encargados de controlar las cifras de los displays *FLT ALT* y *LAND ALT*. El giro del primero modifica la cifra en 500 pies, mientras que la segunda lo hace en 50. Lo primero que se realiza es definir las variables que se van a usar. Estas incluyen las variables que asocian los pines A y B de cada codificador con los pines del Arduino al que están conectados, una variable que recoge el estado actual y otra que almacena el anterior.

```
452     const int ROTARY_FLT_ALT_A=48, ROTARY_FLT_ALT_B=49;  
453     const int ROTARY_LAND_ALT_A=50, ROTARY_LAND_ALT_B=51;  
454  
455     int rotary_flt_alt_State, rotary_flt_alt_LastState;  
456     int rotary_land_alt_State, rotary_land_alt_LastState;
```

Dentro de la sección *setup* se configuran los pines del Arduino a los que están conectados los pines de los codificadores.

```
527     pinMode (ROTARY_FLT_ALT_A, INPUT);  
528     pinMode (ROTARY_FLT_ALT_B, INPUT);  
529     pinMode (ROTARY_LAND_ALT_A, INPUT);  
530     pinMode (ROTARY_LAND_ALT_B, INPUT);
```

Adicionalmente, dentro de la misma sección, es necesario hacer una primera lectura del pin A de cada codificador y almacenarlo en la variable de estado anterior.

```
673 // **PANEL C5-4 COLD AND DARK**  
674  
675 // ***ROTARY ENCODER***  
676  
677 rotary_flt_alt_LastState = digitalRead(ROTARY_FLT_ALT_A);  
678 rotary_land_alt_LastState = digitalRead(ROTARY_LAND_ALT_A);
```

Por último, en la sección *loop*, se configuran las sentencias que escriben los identificadores en el monitor serial según corresponda. En ellas se empieza haciendo una nueva lectura del pin A del codificador rotatorio y se almacena el valor en la variable del estado actual. Esta se compara con la del estado anterior y si es diferente significa que ha ocurrido un pulso y, por consiguiente, el codificador ha girado una posición. Para determinar el sentido del giro, como se vio en la *ilustración 36*, se utiliza la lectura del pin B. Si las medidas son diferentes significa que está girando en sentido horario, escribiendo el identificador más un 1. Por el contrario, si son iguales, significa que está girando en sentido rotatorio. En este caso se escribe el identificador y un 0. Por último, se actualiza la variable de último estado.

```
1644 // ***ROTARY ENCODER LOGIC***  
1645  
1646     rotary_flt_alt_State = digitalRead(ROTARY_FLT_ALT_A);  
1647     if (rotary_flt_alt_State != rotary_flt_alt_LastState){  
1648         if (digitalRead(ROTARY_FLT_ALT_B) != rotary_flt_alt_State) {  
1649             Serial.println("KN1"); //Rotating clockwise  
1650         }  
1651     else {  
1652         Serial.println("KN0"); //Rotating antti-clockwise  
1653     }  
1654 }  
1655     rotary_flt_alt_LastState = rotary_flt_alt_State;  
1656  
1657     rotary_land_alt_State = digitalRead(ROTARY_LAND_ALT_A);  
1658     if (rotary_land_alt_State != rotary_land_alt_LastState){  
1659         if (digitalRead(ROTARY_LAND_ALT_B) != rotary_land_alt_State) {  
1660             Serial.println("KO1"); //Rotating clockwise  
1661         }  
1662     else {  
1663         Serial.println("KO0"); //Rotating antti-clockwise  
1664     }  
1665 }  
1666     rotary_land_alt_LastState = rotary_land_alt_State;
```

## 7.11 Potenciómetro

En el *Overhead Panel* hay cinco potenciómetros, de los cuales dos se usan para la retroiluminación de los paneles y tres para el control de la temperatura de la cabina. De los dos primeros solo será implementado el *POT\_PANEL*, que controla la intensidad de la retroiluminación del *Overhead Panel*. El otro controla la retroiluminación de los *circuit breakers*, localizados detrás de los asientos de los pilotos, que aún no han sido añadidos.

Tanto el potenciómetro de la retroiluminación, como los del control de temperatura, tienen un código similar, solo diferenciándose en que estos últimos escriben en el monitor serial para mandar los datos al simulador. En esta sección se van a explicar los de temperatura, dejando el otro para la siguiente sección.

A continuación, se muestran las variables que se necesitan definir. En estas se incluyen aquellas que contienen el pin de Arduino al que están conectados los potenciómetros, las que contienen el valor actual y anterior de la lectura analógica y las que contienen tanto el valor actual, como el anterior, del resultado de mapear la lectura analógica en un intervalo determinado.

```
406     const int POT_CONT_CAB = A2, POT_FWD_CAB = A3, POT_AFT_CAB = A4;
407     int pot_cont_cab_Aread, pot_fwd_cab_Aread, pot_aft_cab_Aread;
408     int pot_cont_cab_Lastread, pot_fwd_cab_Lastread, pot_aft_cab_Lastread;
409     int pot_cont_cab_Avalue, pot_fwd_cab_Avalue, pot_aft_cab_Avalue;
410     int pot_cont_cab_Lastvalue, pot_fwd_cab_Lastvalue, pot_aft_cab_Lastvalue;
```

En la sección *setup*, al igual que se hizo con los codificadores rotatorios, se hace una primera lectura de la posición del potenciómetro y se actualizan las variables que almacenan ambos estados, el actual y el anterior. Los intervalos de la sentencia *map* son el resultado de una calibración previa.

```
660 // **PANEL C5-2 COLD AND DARK**
661
662 // ***POTENCIOMETERS***
663
664     pot_cont_cab_Aread = analogRead(POT_CONT_CAB);
665     pot_cont_cab_Lastread = pot_cont_cab_Aread;
666     pot_cont_cab_Avalue = map(pot_cont_cab_Aread, 87, 1023, 0, 106);
667     pot_cont_cab_Lastvalue = pot_cont_cab_Avalue;
668
669     pot_fwd_cab_Aread = analogRead(POT_FWD_CAB);
670     pot_fwd_cab_Lastread = pot_fwd_cab_Aread;
671     pot_fwd_cab_Avalue = map(pot_fwd_cab_Aread, 87, 1023, 0, 106);
672     pot_fwd_cab_Lastvalue = pot_fwd_cab_Avalue;
673
674     pot_aft_cab_Aread = analogRead(POT_AFT_CAB);
675     pot_aft_cab_Lastread = pot_aft_cab_Aread;
676     pot_aft_cab_Avalue = map(pot_aft_cab_Aread, 87, 1023, 0, 106);
677     pot_aft_cab_Lastvalue = pot_aft_cab_Avalue;
```

Por último, en la sección *loop* se encuentran las sentencias que permiten escribir en el monitor serial. En estas se vuelve a leer la posición del potenciómetro y, para evitar falsas lecturas de movimiento, pues el potenciómetro es muy sensible y puede variar en la lectura alguna unidad, se compara con la lectura anterior. Si la diferencia en valor absoluto entre estas es de 2 o más unidades se continua con la sentencia.

Tras un mapeo de la variable analógica, se guarda el valor en la variable de estado actual. Mediante la comparación de esta con la del estado anterior se sabe el sentido de giro y, mediante su resta, cuanto ha girado. Por cada vez que se escribe el identificador “KB1” el potenciómetro *POT\_CONT\_CAB* gira en el simulador una posición en sentido horario. Por eso, se define un bucle con un número de iteraciones igual al número de veces que ha girado el potenciómetro, escribiendo consecuentemente tantas veces el identificador. En caso de girar en sentido antihorario, el identificador es “KB0”.

```
1621      // ***POTENCIOMETERS LOGIC***  
1622  
1623      pot_cont_cab_Aread = analogRead(POT_CONT_CAB);  
1624      if ( abs(pot_cont_cab_Aread - pot_cont_cab_Lastread) >= 2){  
1625          pot_cont_cab_Avalue = map(pot_cont_cab_Aread, 87, 1023, 0, 106);  
1626          if (pot_cont_cab_Avalue > pot_cont_cab_Lastvalue){  
1627              int dif = (pot_cont_cab_Avalue - pot_cont_cab_Lastvalue);  
1628              for (int i=0; i<dif; i++){  
1629                  Serial.println("KB1");  
1630              }  
1631          }  
1632          else if (pot_cont_cab_Avalue < pot_cont_cab_Lastvalue){  
1633              int dif = (pot_cont_cab_Lastvalue - pot_cont_cab_Avalue);  
1634              for (int i=0; i<dif; i++){  
1635                  Serial.println("KB0");  
1636              }  
1637          }  
1638          pot_cont_cab_Lastread = pot_cont_cab_Aread;  
1639      }  
1640      pot_cont_cab_Lastvalue = pot_cont_cab_Avalue;
```

Al final de todo el código se actualizan las variables de lectura anterior con el valor de las nuevas.

## 7.12 Transistor IRF1404 MOSFET y sistema de retroiluminación

En esta sección, la última del capítulo, se va a explicar cómo se ha integrado el sistema de retroiluminación haciendo uso del potenciómetro y el transistor MOSFET. Aunque podría parecer difícil desde el punto de vista eléctrico, desde el de programación es bastante sencillo. Se comienzan definiendo las variables del potenciómetro, así como las del transistor.

```
261 // *PANEL C3-1 VARIABLES*
262
263 // **SWITCHES**
264
265 const int POT_CIRCUIT_BREAKER = A0, POT_PANEL = A1;
266 const int LED_BACKLIGHT = 12;
267
268 int pot_circuit_breaker_value = 0, pot_panel_value = 0;
269 int led_backlight_value = 0; // value output to the PWM (analog out)
```

Solamente se usará la sección *loop* en este caso. En ella, en primer lugar, se lee la posición del potenciómetro y se mapea como se vio en la sección 7.11. Los intervalos van, en el caso del potenciómetro, de 0 a 1023, y los valores PWM que se deberán aplicar al transistor, de 0 a 255, es decir, en el primero se tiene una resolución de  $2^{10}$  y en el segundo de  $2^8$ . El resultado se almacena en la variable *led\_backlight\_value*.

El retroiluminado del *Overhead Panel* es un sistema no esencial, por ello cuando el avión está apagado o energizado solo con las baterías, este no funciona. A lo largo de todo el código se han definido diferentes estados de energía, siendo controlados mediante la variable *aircraft\_energy*. Cuando el avión está apagado esta toma valor 0, cuando solamente está activa la batería vale 1 y cuando está conectado el GRD PWR equivale a 2. Este es el primer estado de energía a partir del cual el sistema de retroiluminación comienza a funcionar. Por ello, mediante una sentencia *if* se restringe su uso en los dos primeros estados de energía. Dentro de ella, se encuentra la sentencia de escritura, que envía el valor obtenido del mapeo, comprendido entre 0 y 255 (mínima y máxima luz), al transistor mediante la técnica PWM a través del pin asociado a la variable *LED\_BACKLIGHT*.

Por último, se configura un retraso de 2 ms para que el conversor analógico-digital se asiente después de la última lectura del potenciómetro.

```
1213 // **PANEL BACKLIT**
1214
1215     pot_panel_value = analogRead(POT_PANEL);
1216     led_backlight_value = map(pot_panel_value, 0, 1023, 0, 255);
1217
1218 if(aircraft_energy >= 2){
1219     analogWrite(LED_BACKLIGHT, led_backlight_value);
1220 }
1221
1222 delay(2);
```



## 8. Planificación temporal

Uno de los grandes retos del proyecto ha sido conseguir acabar todas las fases en tiempo y forma. Aunque la fecha oficial de inicio del trabajo fin de grado se corresponde con la del semestre, 1 de febrero de 2019, la realidad es que se empezó meses antes. Concretamente el 1 de julio de 2018, cuando se empezó a aprender a usar Arduino, así como su lenguaje de programación, C++.

Una vez se adquirieron los conocimientos necesarios para emprender este proyecto, y sabiendo los plazos de fabricación que tenía la empresa Cockpit Sim Parts LTD, se empezó a hacer una estimación del material que se iba a usar y a pedirlo. Según iban llegando los componentes se iban realizando pruebas iniciales para comprobar la viabilidad del proyecto.

Un mes antes de la recepción del kit del *Overhead Panel* se comenzó a diseñar los prototipos de cada panel, como se veía en la *ilustración 9*, y a realizar una primera versión del código asociado a cada uno.

Cuando el kit fue entregado, se empezaron a tomar medidas de los paneles y comenzaron las tareas de diseño y construcción del bastidor y el soporte metálico. Finalizadas estas, se procedió con el diseño y construcción de la electrónica. Durante estas fases se iba trabajando en paralelo en la integración de todos los elementos electrónicos y del código final que sería usado en el Arduino.

Tras montar los componentes electrónicos a los paneles del bastidor superior de madera, se pudieron empezar a diseñar e imprimir las piezas 3D que se han visto a lo largo del proyecto.

Con el *Overhead Panel* acabado y el código optimizado en su última versión se comenzó a preparar la documentación de este proyecto, donde se han recogido en detalle todas las fases, de cara a que otras personas en el futuro puedan repetirlo o, incluso, continuarlo y mejorarlo.

Finalmente, y sin saber la fecha exacta de la defensa de este trabajo fin de grado en el momento de su entrega, se dispone de aproximadamente una semana para prepararla, la cual está prevista que ocurra en la semana del 15 a 19 de julio, según el calendario publicado en la normativa de la Universidad Politécnica de Madrid.

Por tanto, este proyecto ha tenido una duración total de 1 año y 18 días. La extensión temporal de cada tarea, así como la relación entre ellas, se expone en el diagrama Gantt de la siguiente página, donde a la izquierda se ve la información de cada tarea y a la derecha el citado diagrama Gantt.

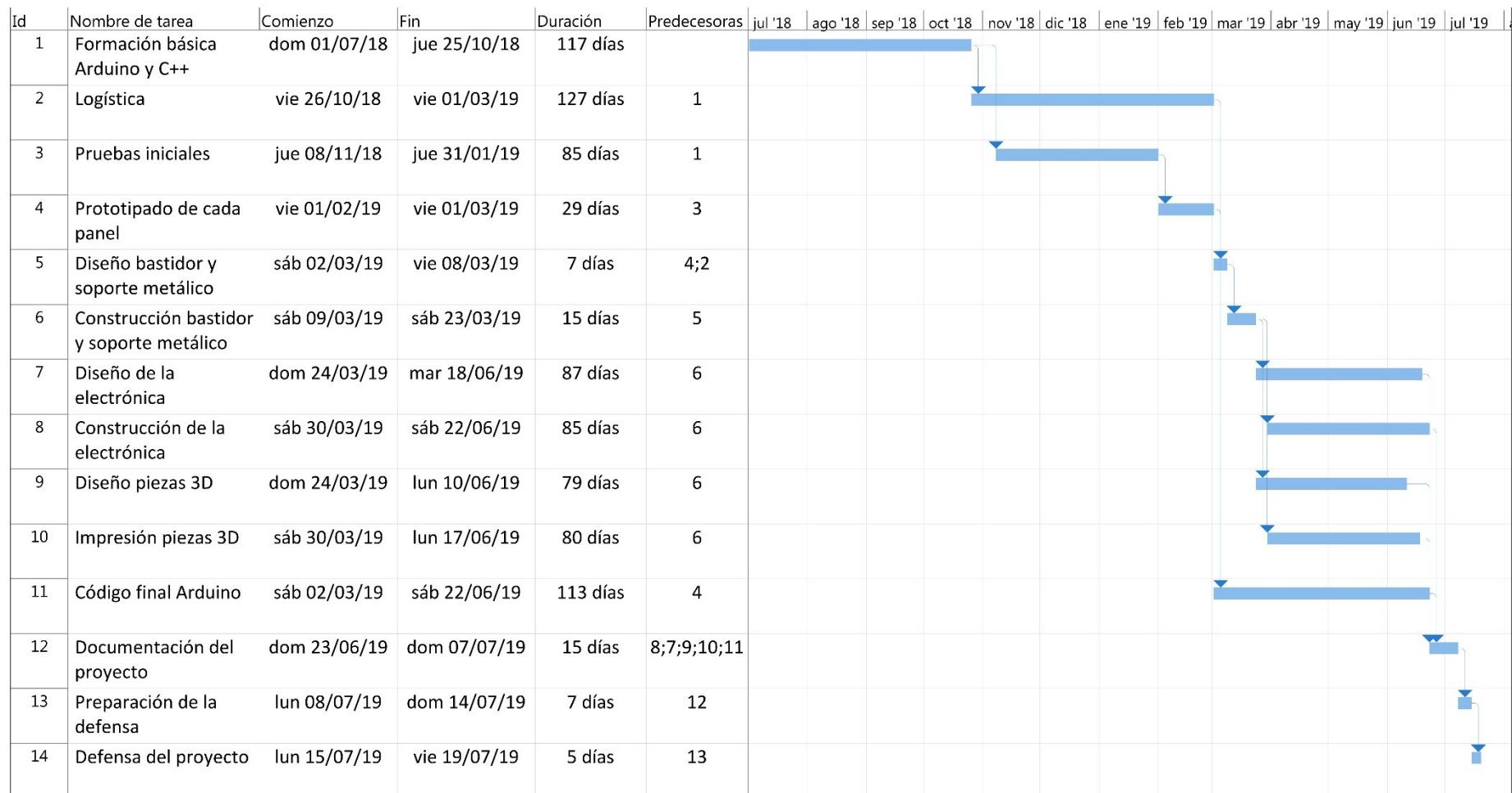


Ilustración 113. Diagrama Gantt del proyecto

## 9. Conclusiones

El presente trabajo fin de grado muestra el proceso llevado a cabo para **diseñar, desarrollar e implementar el *Overhead Panel* del avión **Boeing 737NG** con la tecnología de Arduino**. Aunque a la hora de hablar de este microcontrolador se podría pensar en proyectos sencillos como el parpadeo de un led o hacer girar un motor, la realidad es que **sus posibilidades son infinitas**, llegando incluso a permitir la construcción de un elemento complejo como es el que se realiza en este proyecto. Por ello, no se debe subestimar a este tipo de tecnologías pues, aunque no son las totalmente óptimas en productos comerciales, si bien son más que **suficientes para la creación de simuladores caseros**, y en especial, aquellos de **bajo coste**.

En la fase de **diseño** hubo que enfrentarse al reto de transformar un elemento de aviación real en una **réplica realista y funcional** para un simulador. Por ello, tuvo que investigarse cada elemento por individual y ver cómo podía ser implementado. Para esta tarea fueron de gran ayuda las **herramientas de diseño 3D y creación de circuitos electrónicos**, pues aportan una visión gráfica de lo que se quiere diseñar y permiten modificarlo fácilmente de cara a ir generando versiones mejoradas y optimizadas.

La fase de **construcción** aportó una visión realista de lo que se había esquematizado en la fase anterior. En la mayoría de los casos, los **diseños eran óptimos** y no se encontraron contratiempos. Sin embargo, otros tuvieron que **rediseñarse para cumplir con los requisitos** impuestos en el proyecto, generando el diseño definitivo que se ha expuesto a lo largo de este informe. Para la construcción del *Overhead Panel* se ha utilizado una gran variedad de **máquinas herramienta**. Por ello, se puede concluir que durante esta fase se ha demostrado el manejo de estas y la capacidad de seguir las directrices del diseño propuesto.

Finalmente, en la fase de **integración** se ha conseguido cumplir con el requisito inicial de usar exclusivamente **un solo Arduino Mega 2560** mediante el uso de los **registros de desplazamiento**, un componente electrónico barato y sencillo, pero a la vez de gran utilidad. En esta fase ha sido de gran ayuda la inmensa cantidad de **documentación online** aportada por la comunidad de usuarios que existe detrás de este microcontrolador. Esta, sin duda, es una gran **filosofía** que comparte este proyecto, pues se ofrecen todas las pautas para poder replicarlo, de manera que aficionados a la simulación puedan tener acceso a un producto de alto realismo, por un precio mucho inferior respecto a los *Overhead Panel* vendidos en tiendas.

Para concluir, este proyecto ha sido posible gracias a la utilización de múltiples **conocimientos adquiridos** en varias **disciplinas** estudiadas a lo largo de la carrera. Entre ellas destacan el **tratamiento digital de la información**, de gran utilidad a la hora de entender el comportamiento interno de los registros de desplazamiento, uno de los elementos electrónicos más importantes de este proyecto. También fueron necesarios conocimientos de la **instalación eléctrica del avión y su aviónica**, para comprender el funcionamiento real de los componentes y poder generar el modelo realista de estos a través de la programación del código. Para esta tarea, asignaturas como **Informática**, han sentado las bases del aprendizaje para poder entender el nuevo lenguaje de programación usado en este proyecto. Por último, de cara a la fase de construcción, se hizo uso de los conocimientos de **fabricación aeroespacial**, además de los de **gestión de proyectos** a la hora de planificar las diferentes fases y actividades.

De cara al **futuro**, se espera que este trabajo fin de grado sea el **inicio** de la construcción de un **simulador realista completo** del avión Boeing 737NG. El *Overhead Panel*, creado en este proyecto, será añadido al simulador en el que se lleva trabajando varios años. Así mismo, el objetivo es que muy pronto puedan seguir siendo añadidos **nuevos elementos** de la cabina creados de **forma casera** y siempre con la característica de **bajo coste**, no implicando por ello un bajo nivel derealismo, como se ha visto a lo largo de este informe.

## 10. Referencias

### 10.1 Monografías

MARTÍN, Agustín, 2016. *Tratamiento Digital de la Información*. Madrid: Sección de Publicaciones ETSIAE.

### 10.2 Recursos web

ARDUINO, 2019. Arduino Mega 2560 Rev. 3. En: *Arduino* [en línea]. Disponible en: <https://store.arduino.cc/mega-2560-r3> [consulta: 23 junio 2019].

ARDUINO, 2019. What is Arduino? En: *Arduino* [en línea]. Disponible en: <https://www.arduino.cc/en/Guide/Introduction> [consulta: 23 junio 2019].

BUILD ELECTRONIC CIRCUITS, 2016. The Potentiometer and Wiring Guide En: *Build electronic circuits* [en línea]. Disponible en: <https://www.build-electronic-circuits.com/potentiometer/> [consulta: 01 julio 2019].

COCKPIT SIM PARTS LTD, 2017. Boeing 737 Overhead v3 with Switches. En: *CockpitSimParts Flight Simulator Manufacturer* [en línea]. Disponible en: <https://www.cockpitsimparts.co.uk/forward-overhead-panel-/128-boeing-737-overhead-v3-with-switches.html> [consulta: 26 octubre 2018]

CL LIGHTING CO, 2006. 3651A datasheet. En: *ElectroComponentes* [en línea]. Disponible en: [https://www.electrocomponentes.es/buscar?controller=search&orderby=position&orderway=desc&search\\_query=display+7+segmentos&submit\\_search=](https://www.electrocomponentes.es/buscar?controller=search&orderby=position&orderway=desc&search_query=display+7+segmentos&submit_search=) [consulta: 19 diciembre 2018]

COCKPIT-X, 2019. En: *Cockpit-X* [en línea]. Disponible en: <http://www.cockpitx.space/> [consulta: 24 junio 2019].

EBOTICS, 2019. Proyecto N.º 2: controlar el brillo del led. En: *Ebotics* [en línea]. Disponible en: <https://www.ebotics.com/es/actividad/proyecto-no-2-controlar-el-brillo-del-led/> [consulta: 30 junio 2019].

ELECTRONTOLS, 2016. Display 7 segmentos. En: *Electrontools* [en línea]. Disponible en: <https://www.electrontools.com/Home/WP/2016/03/09/display-7-segmentos/> [consulta: 30 junio 2019].

FLY737NG, 2017. The 737NG experience. En: *Fly737ng* [en línea]. Disponible en: <http://www.fly737ng.com/> [consulta: 27 junio 2019]

HOW TO MECHATRONICS, 2016. How Rotary Encoder Works and How To Use It with Arduino. En: *How to mechatronics* [en línea]. Disponible en: <https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/> [consulta: 17 septiembre 2018]

MARKUSPILOT, 2013. Boeing 737 measurements. En: *Markuspilot, home for flight simulator*.

Disponible en:

<https://www.markuspilot.com/download/BOEING%20737%20MEASUREMENTS.pdf> [consulta: 15 abril 2019]

LAST MINUTE ENGINEERS, 2019. How 74HC595 Shift Register Works & Interface it with Arduino. En: *Last Minute Engineers* [en línea]. Disponible en:

<https://lastminuteengineers.com/74hc595-shift-register-arduino-tutorial/> [consulta: 29 junio 2019]

OASISTEK, 2006. TOD-3261AG-B datasheet. En: *Component search engine* [en línea].

Disponible en: <https://componentsearchengine.com/TOD-3261AG-B/Oasistek> [consulta: 30 junio 2019]

PETER PARTS, 2019. NET-3631/32AG-11 datasheet. En: *Peter parts* [en línea]. Disponible en: <https://www.peterparts.com/CatalogPages/57/973.pdf> [consulta: 30 junio 2019]

PHILIPS, 1990. IC 74HC165 datasheet. En: *Datasheet catalog* [en línea]. Disponible en:

[http://www.datasheetcatalog.com/datasheets\\_pdf/7/4/H/C/74HC165.shtml](http://www.datasheetcatalog.com/datasheets_pdf/7/4/H/C/74HC165.shtml) [consulta: 29 junio 2019]

PHILIPS, 2003. IC 74HC595 datasheet. En: *Octopart* [en línea]. Disponible en:

<https://datasheet.octopart.com/74HC595N-Philips-datasheet-7085704.pdf> [consulta: 29 junio 2019]

PMDG, 2011. PMDG 737-800/900 Base Package for FSX. En: *PMDG Simulations, LLC.* [en línea]. Disponible en: <https://www.precisionmanuals.com/pages/products/FSX/NGX8900.html> [consulta: 24 junio 2019].

THE ENGINEERING PROJECTS, 2018. Introduction to Arduino Mega 2560. En: *The Engineering Projects* [en línea]. Disponible en:

<https://www.theengineeringprojects.com/2018/06/introduction-to-arduino-mega-2560.html> [consulta: 3 agosto 2018].

### 10.3 Recursos digitales

MICROSOFT, 2008. En: *Microsoft Flight Simulator X Gold Edition* [DVD]

## 11. Anexos

En este capítulo pueden encontrarse los planos acotados y esquemas eléctricos que han sido referenciados a lo largo de todo el informe, así como las tablas de asignación de pines y el repositorio web con todos los archivos del proyecto. Los elementos incluidos en el anexo se enumeran en la siguiente lista:

1. Plano acotado del bastidor inferior de madera
2. Plano acotado del bastidor superior de madera
3. Plano acotado del soporte metálico del *Overhead Panel*
4. Utilización de los pines del Arduino Mega 2560
5. Esquema eléctrico del funcionamiento del IC 74HC595
6. Esquema eléctrico del funcionamiento del IC 74HC165
7. Dimensiones y multiplexado de un display LED 7 segmentos de 2 dígitos
8. Dimensiones y multiplexado de un display LED 7 segmentos de 3 dígitos
9. Dimensiones y multiplexado de un display LED 7 segmentos de 5 dígitos
10. Esquema eléctrico de la instalación de servomotores
11. Esquema eléctrico de la instalación de codificadores rotatorios
12. Esquema eléctrico de la instalación de potenciómetros
13. Esquema eléctrico de la instalación de retroiluminación
14. Asignación de los elementos de salida a los pines del 74HC595 (SR\_OUT)
15. Asignación de los elementos de entrada a los pines del 74HC165 (SR\_IN)
16. Repositorio web GitHub

## 11.1 Plano acotado del bastidor inferior de madera

Dimensiones referenciadas en milímetros.

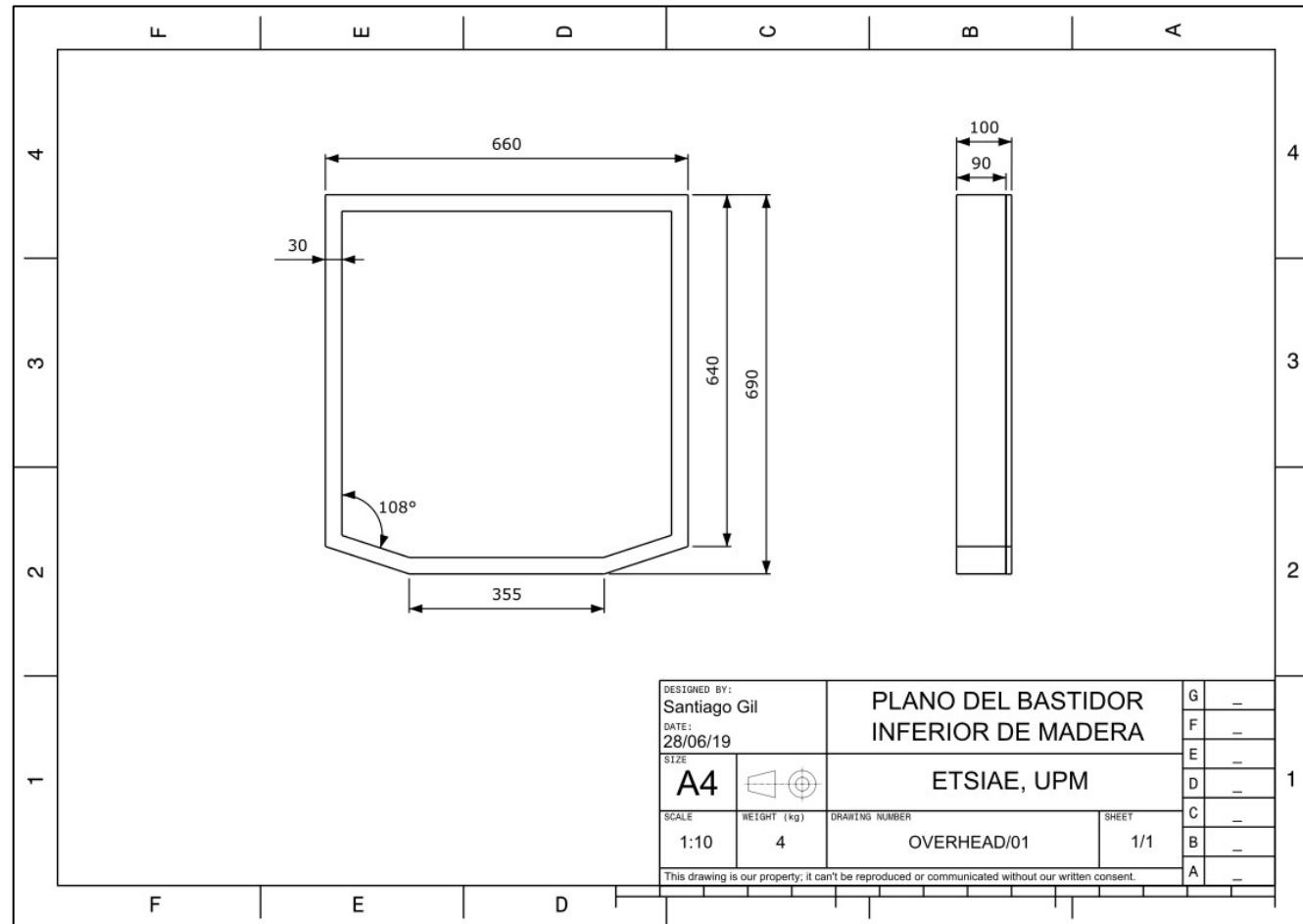


Ilustración 114. Plano acotado del bastidor inferior de madera

## 11.2 Plano acotado del bastidor superior de madera

Dimensiones referenciadas en milímetros.

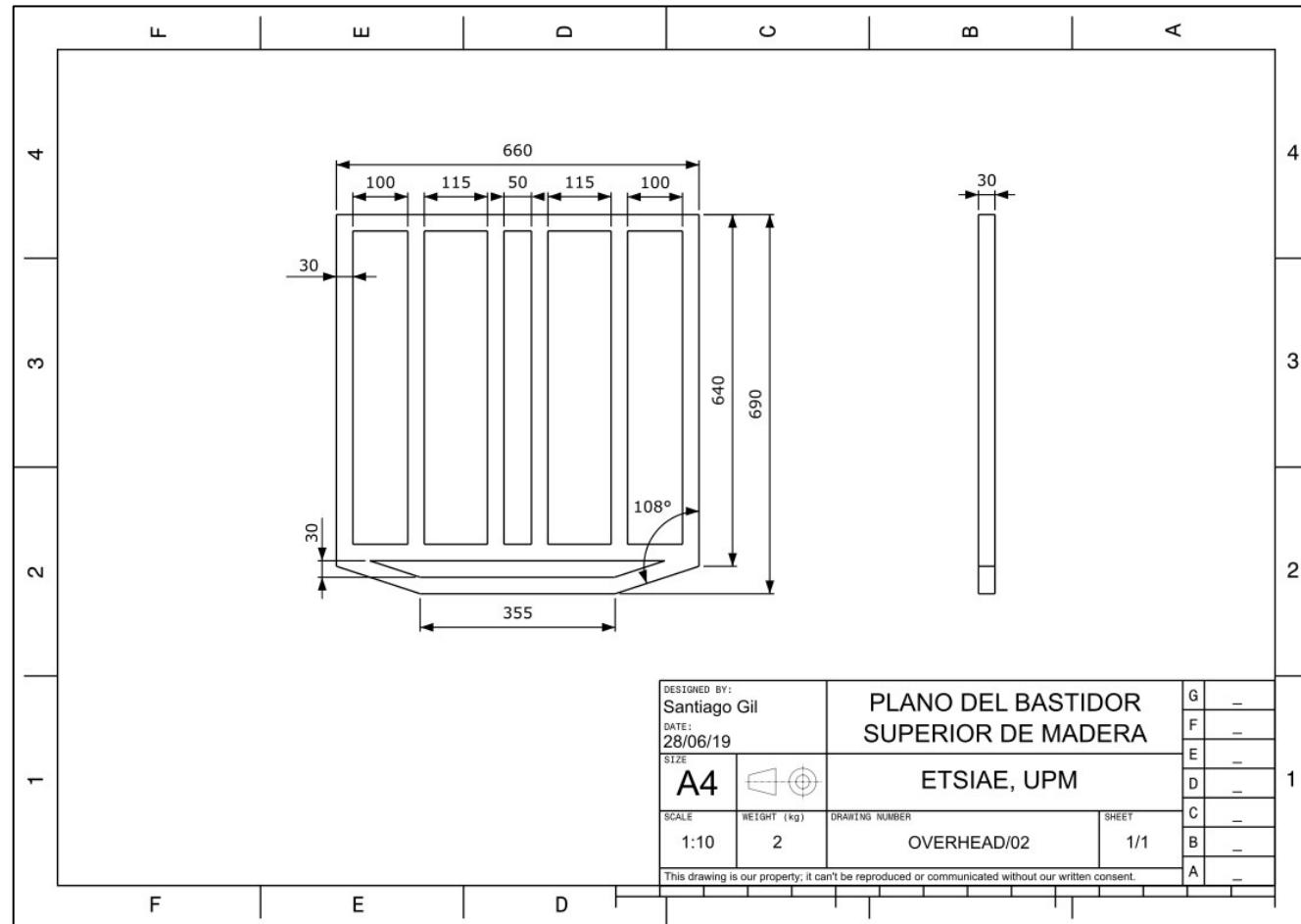


Ilustración 115. Plano acotado del bastidor superior de madera

### 11.3 Plano acotado del soporte metálico del *Overhead Panel*

Dimensiones referenciadas en milímetros.

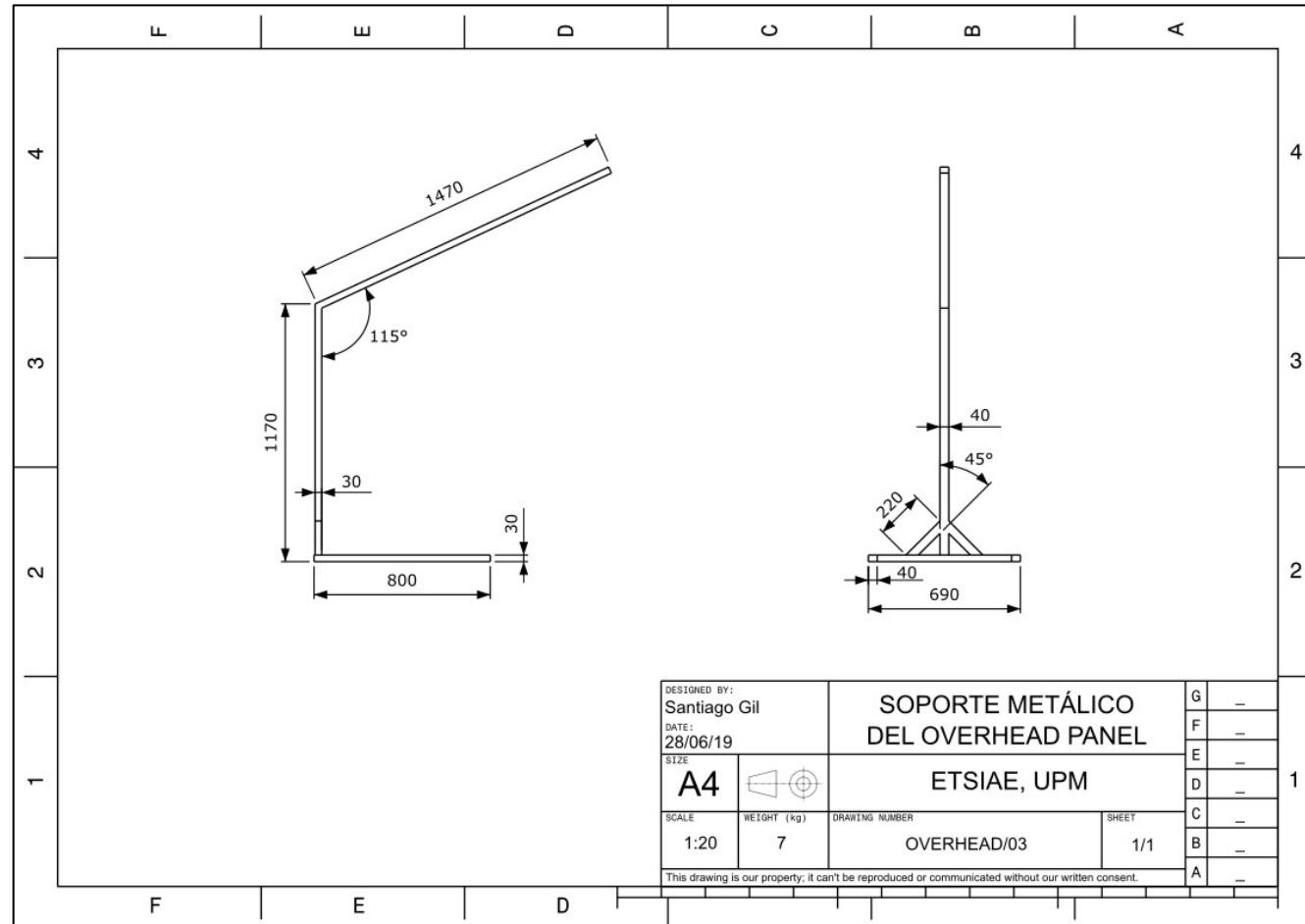


Ilustración 116. Plano acotado del soporte metálico del *Overhead Panel*

## 11.4 Utilización de los pines del Arduino Mega 2560

A continuación, se muestra la lista de la utilización que se ha hecho de los pines del Arduino Mega 2560. Los pines marcados con la letra *D* hacen referencia a los digitales y con la *A* los analógicos.

Tabla 10. Utilización de los pines del Arduino Mega 2560

PIN	CONEXIÓN	FUNCIÓN
<b>D2</b>	SERVO_FUEL_TEMP	SERVO_FUEL_TEMP data pin PANEL_C1_3
<b>D3</b>	SERVO_SELECTOR_ENGINE_STATOR	SERVO_SELECTOR_ENGINE_STATOR data pin PANEL_C1_4
<b>D4</b>	SERVO_EGT_APU	SERVO_EGT_APU data pin PANEL_C2_4
<b>D5</b>	SERVO_CABIN_ALT_EXT	SERVO_CABIN_ALT_EXT data pin PANEL_C4_6
<b>D6</b>	SERVO_CABIN_ALT_INT	SERVO_CABIN_ALT_INT data pin PANEL_C4_6
<b>D7</b>	SERVO_CABIN_CLIMB	SERVO_CABIN_CLIMB data pin PANEL_C4_6
<b>D8</b>	SERVO_CAB_TEMP	SERVO_CAB_TEMP data pin PANEL_C5_2
<b>D9</b>	SERVO_DUCT_PRESS_L	SERVO_DUCT_PRESS_L data pin PANEL_C5_3
<b>D10</b>	SERVO_DUCT_PRESS_R	SERVO_DUCT_PRESS_R data pin PANEL_C5_3
<b>D11</b>	SERVO_MANUAL_VALVE	SERVO_MANUAL_VALVE data pin PANEL_C5_4
<b>D12</b>	LED_BACKLIGHT (PWM)	Retroiluminación del panel controlado por el transistor MOSFET y el potenciómetro POT_PANEL
<b>D13</b>	LIBRE	-
<b>D22</b>	SR_OUT_1_DS	Data pin PANEL_C1_1, 2, 3 y 4
<b>D23</b>	SR_OUT_1_ST_CP	Latch pin PANEL_C1_1, 2, 3 y 4
<b>D24</b>	SR_OUT_1_SH_CP	Clock pin PANEL_C1_1, 2, 3 y 4
<b>D25</b>	SR_IN_7_PL	Parallel load input PANEL_C1_1, 2, 3 y 4
<b>D26</b>	SR_IN_7_CE	Clock enable input PANEL_C1_1, 2, 3 y 4
<b>D27</b>	SR_IN_7_Q7	Data pin input PANEL_C1_1, 2, 3 y 4
<b>D28</b>	SR_IN_7_CP	Clock input PANEL_C1_1, 2, 3 y 4
<b>D29</b>	SR_OUT_4_DS	Data pin PANEL_C2 y C3
<b>D30</b>	SR_OUT_4_ST_CP	Latch pin PANEL_C2 y C3
<b>D31</b>	SR_OUT_4_SH_CP	Clock pin PANEL_C2 y C3
<b>D32</b>	SR_IN_14_PL	Parallel load input PANEL_C2_1, 2, 3, 4, C3_1, 2
<b>D33</b>	SR_IN_14_CE	Clock enable input PANEL_C2_1, 2, 3, 4, C3_1, 2
<b>D34</b>	SR_IN_14_Q7	Data pin input PANEL_C2_1, 2, 3, 4, C3_1, 2
<b>D35</b>	SR_IN_14_CP	Clock input PANEL_C2_1, 2, 3, 4, C3_1, 2
<b>D36</b>	SR_OUT_13_DS	Data pin PANEL_C4, C5_2, C5_3
<b>D37</b>	SR_OUT_13_ST_CP	Latch pin PANEL_C4, C5_2, C5_3
<b>D38</b>	SR_OUT_13_SH_CP	Clock pin PANEL_C4, C5_2, C5_3
<b>D39</b>	SR_IN_19_PL	Parallel load input PANEL_C4 y C5
<b>D40</b>	SR_IN_19_CE	Clock enable input PANEL_C4 y C5
<b>D41</b>	SR_IN_19_Q7	Data pin input PANEL_C4 y C5
<b>D42</b>	SR_IN_19_CP	Clock input PANEL_C4 y C5
<b>D43</b>	SR_OUT_19_DS	Data pin PANEL_C5_3, C5_4
<b>D44</b>	SR_OUT_19_ST_CP	Latch pin PANEL_C5_3, C5_4
<b>D45</b>	SR_OUT_19_SH_CP	Clock pin PANEL_C5_3, C5_4
<b>D46</b>	LIBRE	-
<b>D47</b>	LIBRE	-
<b>D48</b>	ROTARY_FLT_ALT_A	DATA A PIN rotary encoder FLT_ALT PANEL_C5_4
<b>D49</b>	ROTARY_FLT_ALT_B	DATA B PIN rotary encoder FLT_ALT PANEL_C5_4

<b>D50</b>	ROTARY_LAND_ALT_A	DATA A PIN rotary encoder LAND_ALT PANEL_C5_4
<b>D51</b>	ROTARY_LAND_ALT_B	DATA B PIN rotary encoder LAND_ALT PANEL_C5_4
<b>D52</b>	LIBRE	-
<b>D53</b>	LIBRE	-
<b>A0</b>	POT_CIRCUIT_BREAKER	Potenciómetro de 10kΩ que controla la intensidad de la retroiluminación
<b>A1</b>	POT_PANEL	Potenciómetro de 10 kΩ que controla la intensidad de la retroiluminación
<b>A2</b>	POT_CONT_CAB	Potenciómetro de 10 kΩ que controla la temperatura de CONT CAB
<b>A3</b>	POT_FWD_CAB	Potenciómetro de 10 kΩ que controla la temperatura de FWD CAB
<b>A4</b>	POT_AFT_CAB	Potenciómetro de 10 kΩ que controla la temperatura de AFT CAB
<b>A5</b>	LIBRE	-
<b>A6</b>	LIBRE	-
<b>A7</b>	LIBRE	-
<b>A8</b>	LIBRE	-
<b>A9</b>	LIBRE	-
<b>A10</b>	LIBRE	-
<b>A11</b>	LIBRE	-
<b>A12</b>	LIBRE	-
<b>A13</b>	LIBRE	-
<b>A14</b>	LIBRE	-
<b>A15</b>	LIBRE	-

## 11.5 Esquema eléctrico del funcionamiento del IC 74HC595

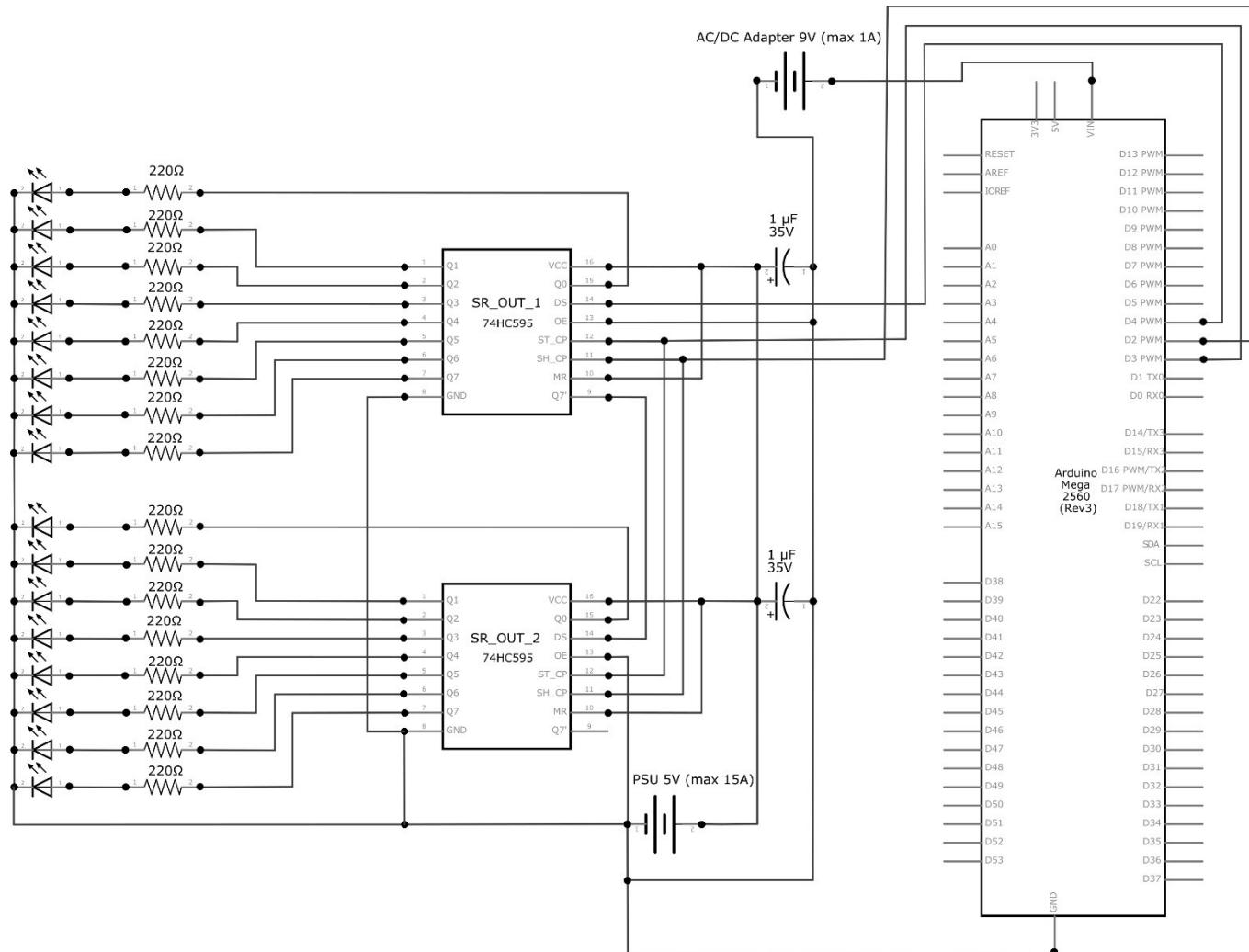


Ilustración 117. Esquema eléctrico del funcionamiento del IC 74HC595

## 11.6 Esquema eléctrico del funcionamiento del IC 74HC165

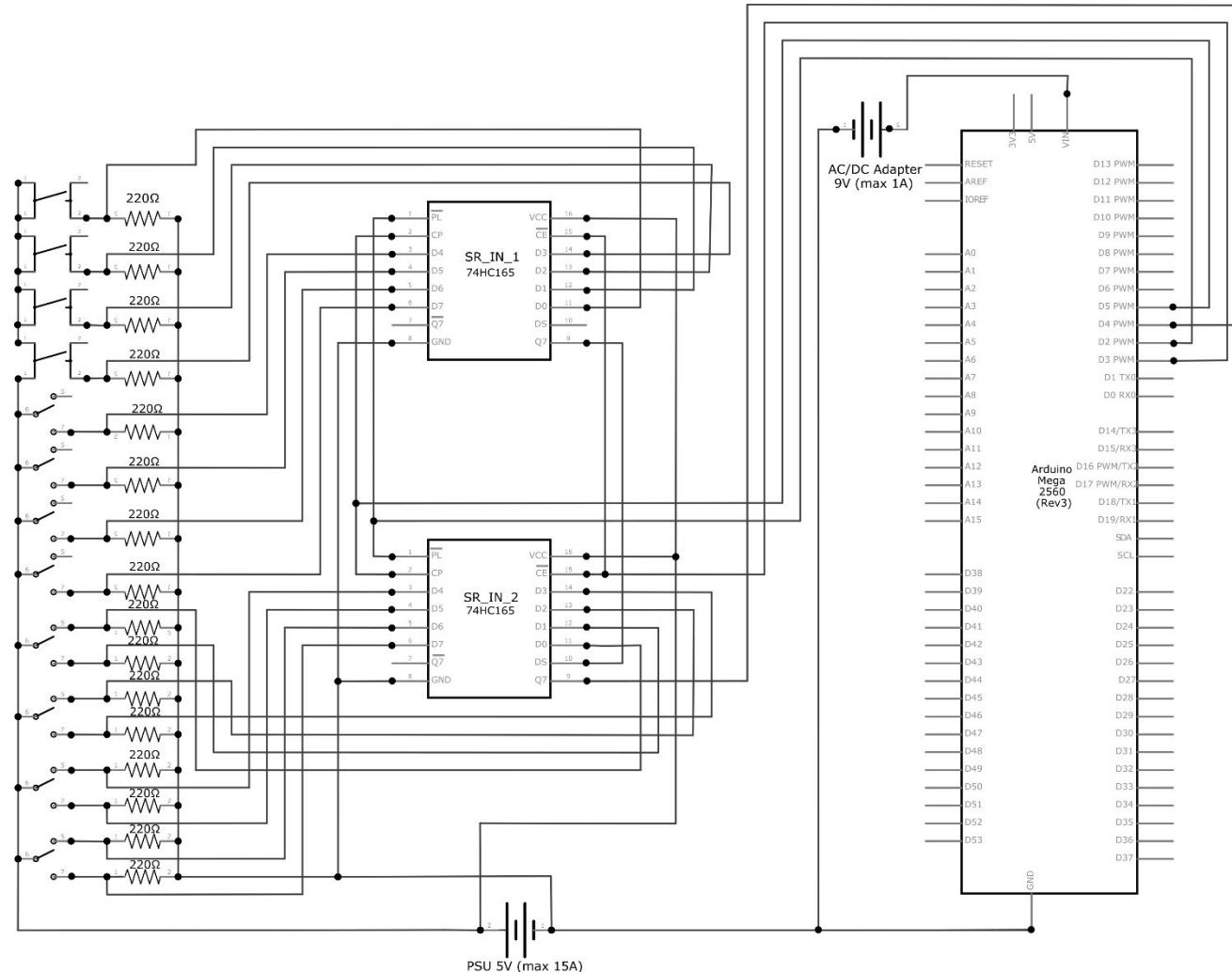


Ilustración 118. Esquema eléctrico del funcionamiento del IC 74HC165

## 11.7 Dimensiones y multiplexado de un display LED 7 segmentos de 2 dígitos

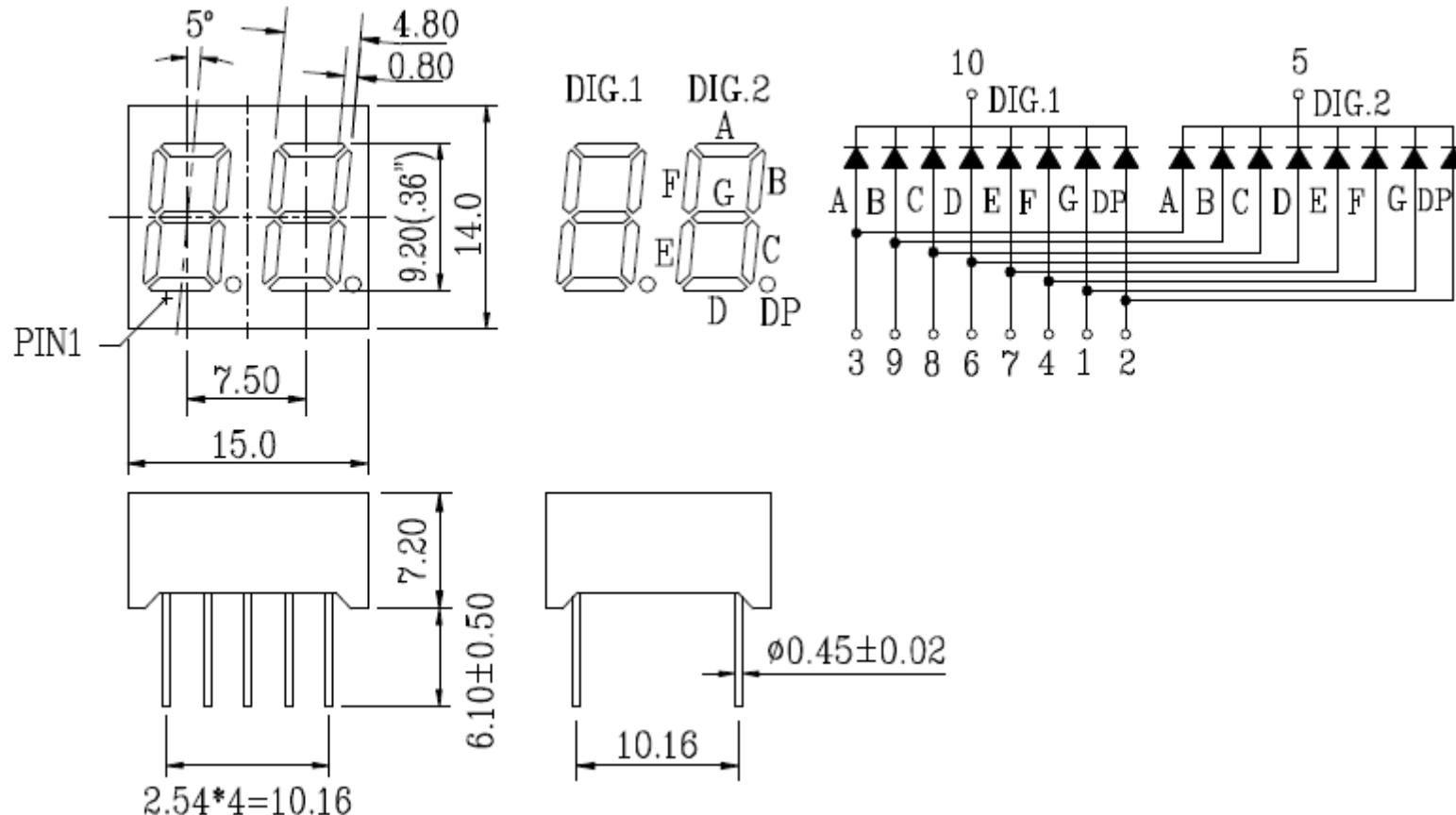
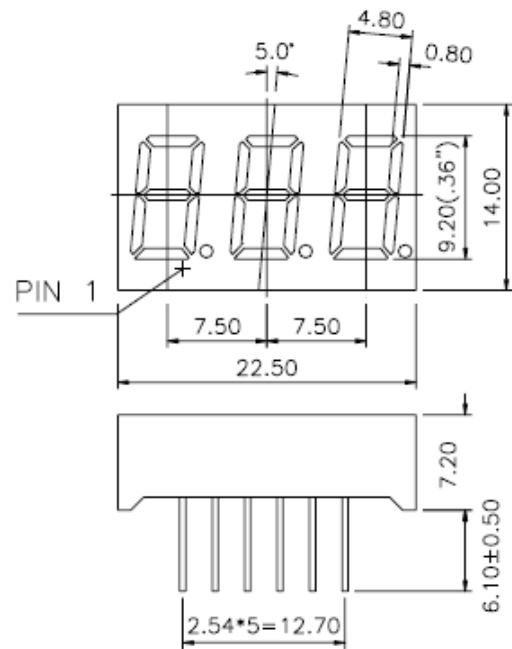
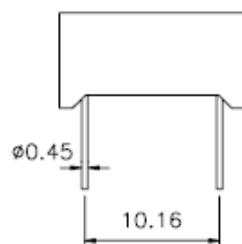
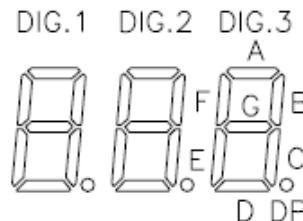
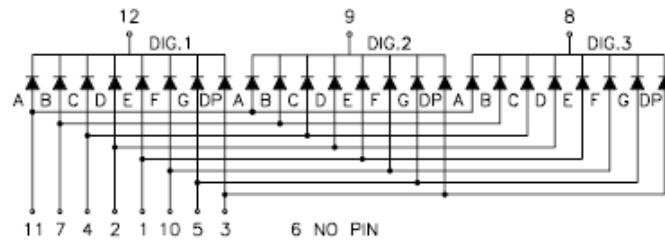


Ilustración 119. Dimensiones y multiplexado de un display LED 7 segmentos de 2 dígitos (Oasistek 2006)

## 11.8 Dimensiones y multiplexado de un display LED 7 segmentos de 3 dígitos



**NET-3631AG**



**NET-3632AG**

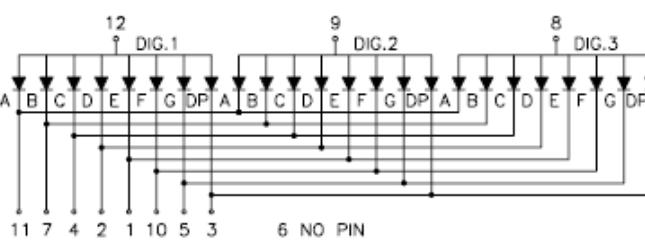


Ilustración 120. Dimensiones y multiplexado de un display LED 7 segmentos de 3 dígitos (Peter parts 2019)

## 11.9 Dimensiones y multiplexado de un display LED 7 segmentos de 3 dígitos

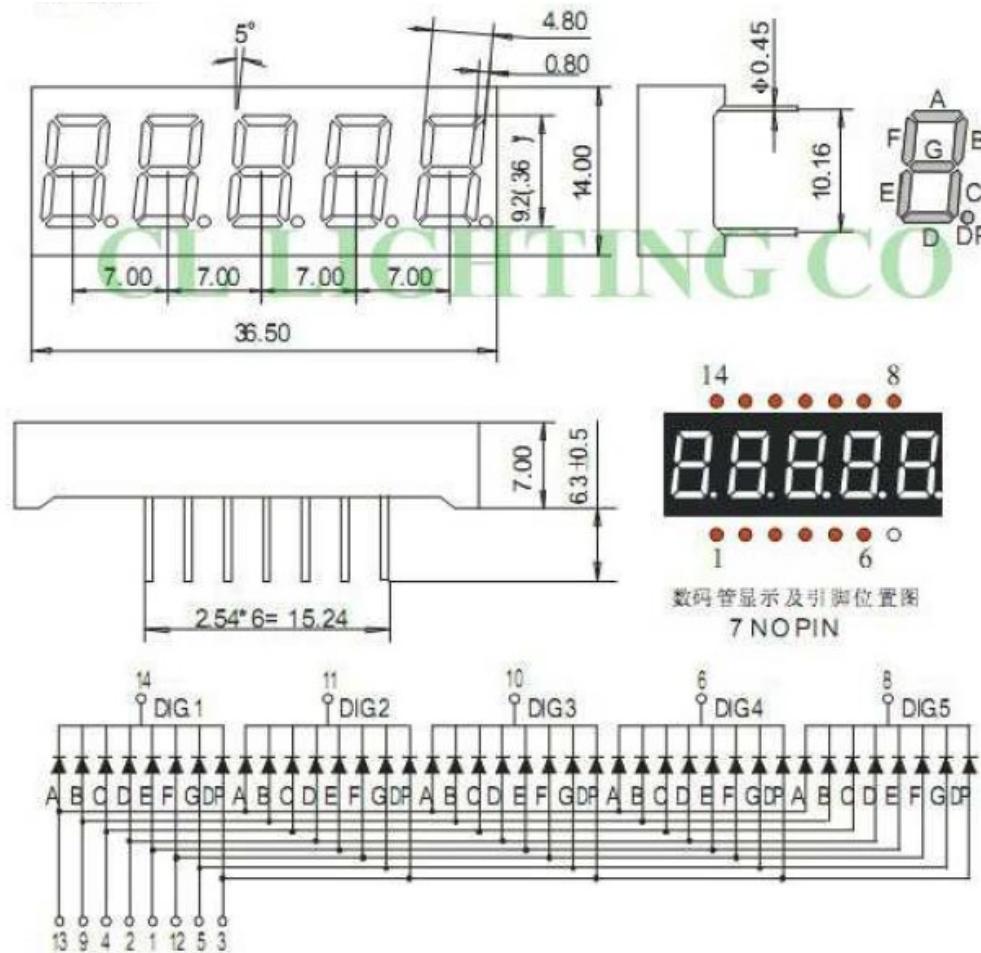


Ilustración 121. Dimensiones y multiplexado de un display LED 7 segmentos de 5 dígitos (CL Lighting Co 2006)

## 11.10 Esquema eléctrico de la instalación de servomotores

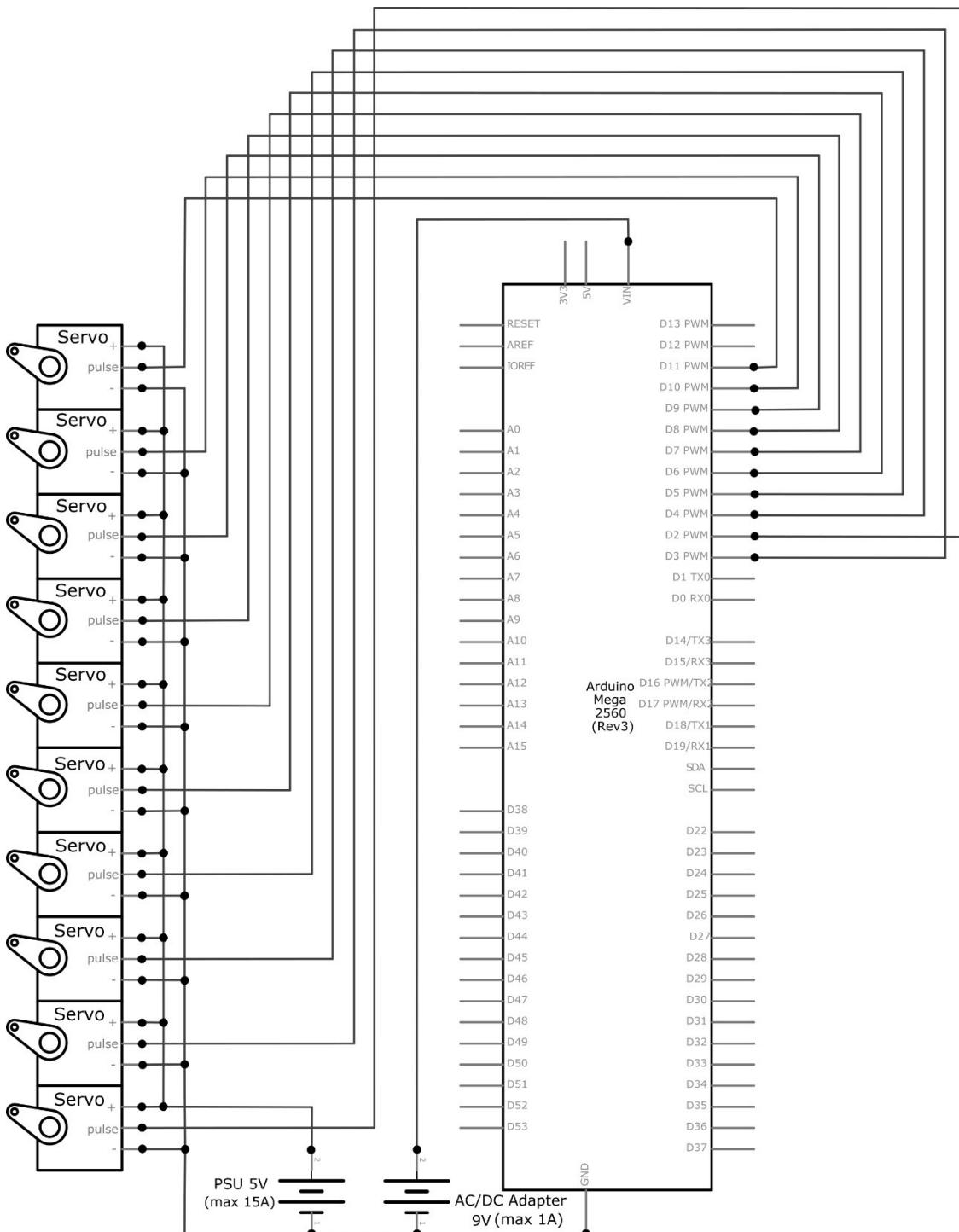


Ilustración 122. Esquema eléctrico de la instalación de servomotores

### 11.11 Esquema eléctrico de la instalación de codificadores rotatorios

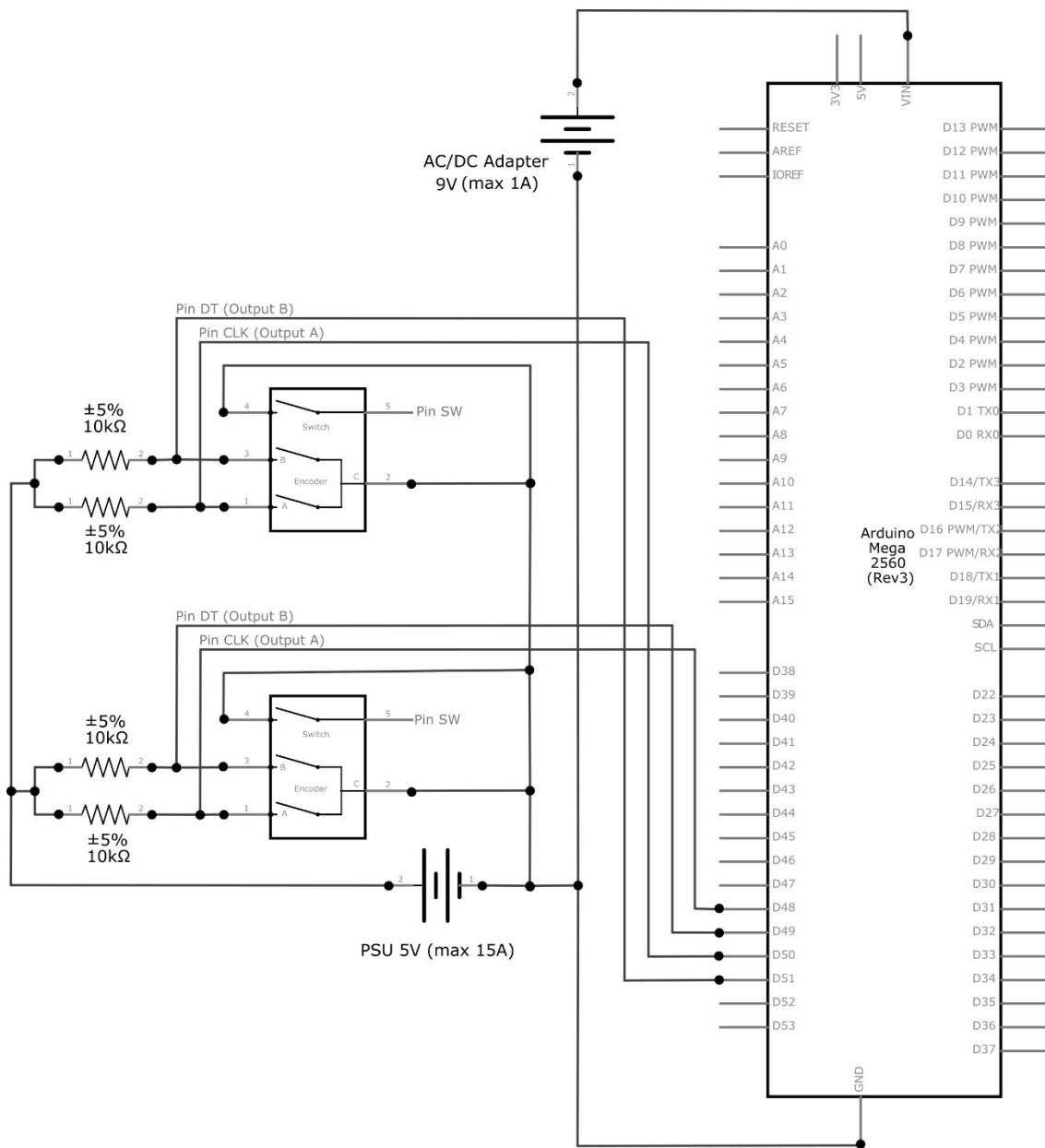


Ilustración 123. Esquema eléctrico de la instalación de codificadores rotatorios

### 11.12 Esquema eléctrico de la instalación de potenciómetros

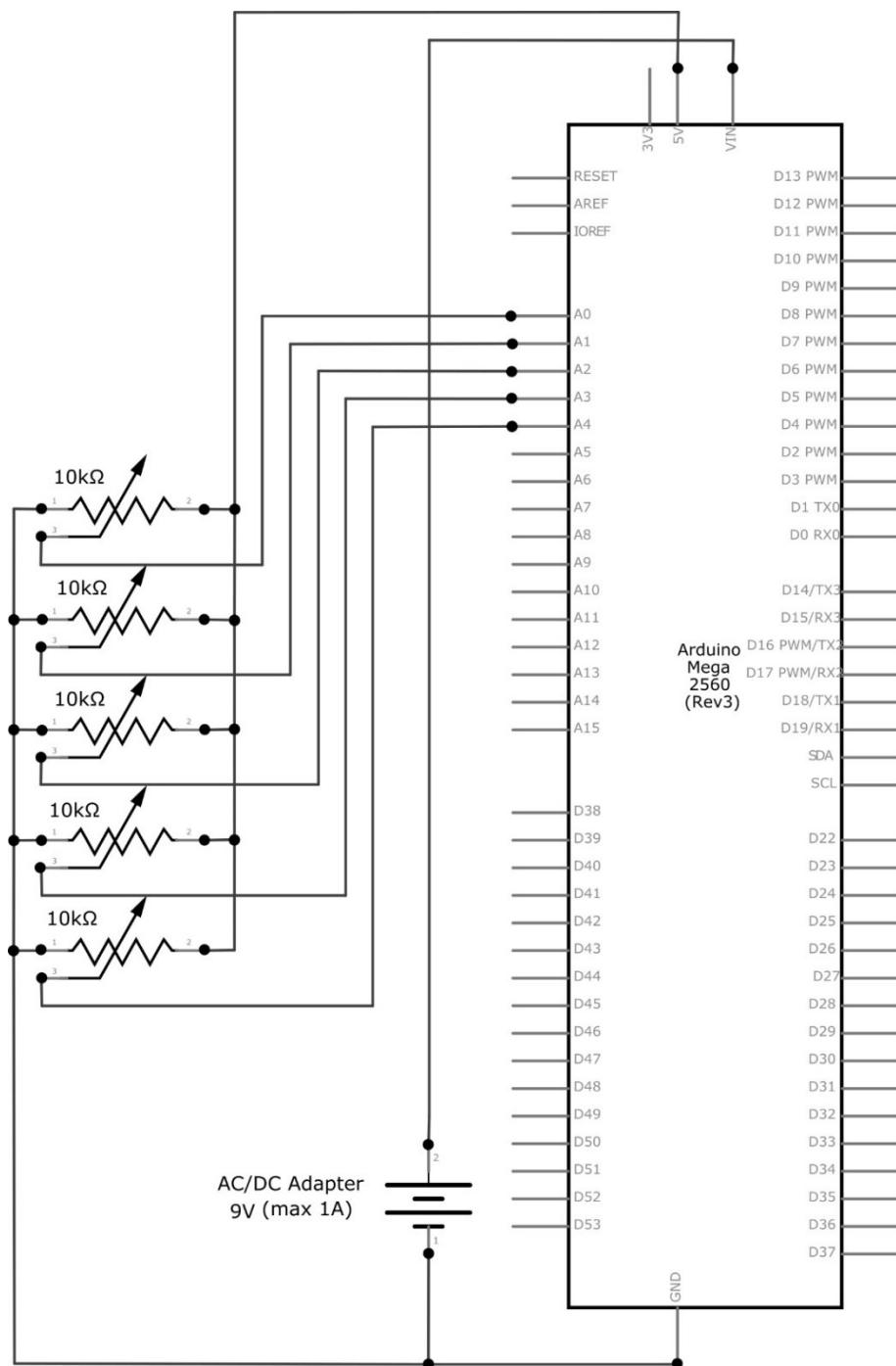


Ilustración 124. Esquema eléctrico de la instalación de potenciómetros

### 11.13 Esquema eléctrico de la instalación de retroiluminación

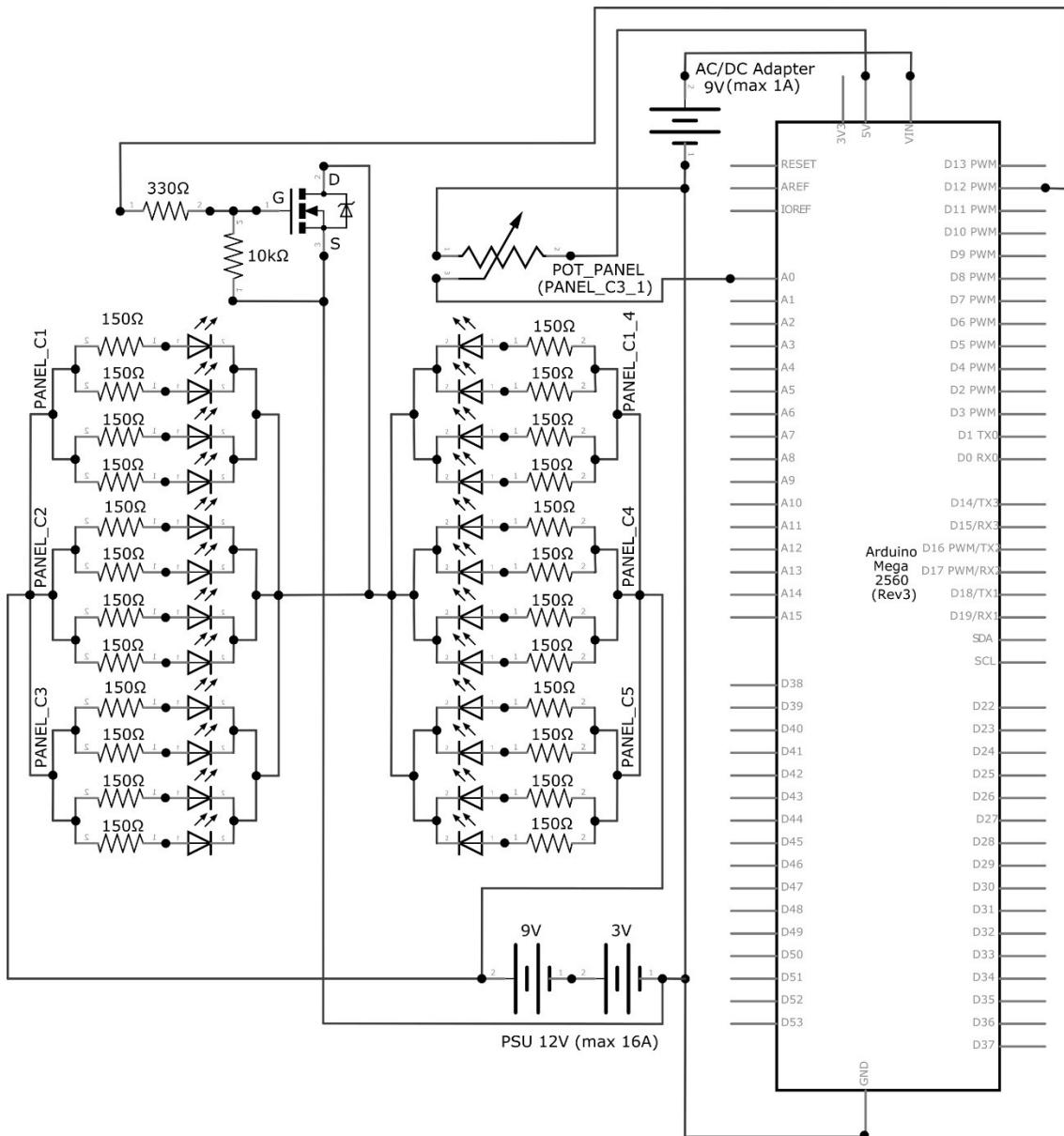


Ilustración 125. Esquema eléctrico de la instalación de retroiluminación

## 11.14 Asignación de los elementos de salida a los pines del 74HC595 (SR\_OUT)

Tabla 11. Asignación de los elementos de salida a los pines del 74HC595 (SR\_OUT)

SHIIFT REGISTER	PIN	ELEMENTO	PANEL	CÓDIGO	ID
SR_OUT_1	Q0	LED_STANDBY_HYD_LOW_QUANTITY	PANEL_C1_1	BU	42
	Q1	LED_STANDBY_HYD_LOW_PRESSURE	PANEL_C1_1	BV	43
	Q2	LIBRE	-	-	-
	Q3	LED_FLT_CONTROL_A_LOW_PRESSURE	PANEL_C1_1	BX	45
	Q4	LED_FLT_CONTROL_B_LOW_PRESSURE	PANEL_C1_1	BY	46
	Q5	LED_FEEL_DIFF_PRESS	PANEL_C1_1	BZ	47
	Q6	LED_SPEED_TRIM_FAIL	PANEL_C1_1	CA	48
	Q7	LED_MACH_TRIM_FAIL	PANEL_C1_1	CB	49
SR_OUT_2	Q0	LED_AUTO_SLAT_FAIL	PANEL_C1_1	CC	50
	Q1	LED_YAW_DAMPER	PANEL_C1_1	CE	51
	Q2	LED_ENGINE_VALVE_CLOSED_1	PANEL_C1_3	CF	54
	Q3	LED_ENGINE_VALVE_CLOSED_2	PANEL_C1_3	CG	55
	Q4	LED_SPAR_VALVE_CLOSED_1	PANEL_C1_3	CH	56
	Q5	LED_SPAR_VALVE_CLOSED_2	PANEL_C1_3	CI	57
	Q6	LED_FILTER_BYPASS_1	PANEL_C1_3	CJ	58
	Q7	LED_FILTER_BYPASS_2	PANEL_C1_3	CK	59
SR_OUT_3	Q0	LED_VALVE_OPEN	PANEL_C1_3	CL	60
	Q1	LED_FUEL_PUMP_LOW_PRESSURE_1	PANEL_C1_3	CM	61
	Q2	LED_FUEL_PUMP_LOW_PRESSURE_2	PANEL_C1_3	CN	62
	Q3	LED_AFT_LOW_PRESSURE_1	PANEL_C1_3	CO	63
	Q4	LED_AFT_LOW_PRESSURE_2	PANEL_C1_3	CP	64
	Q5	LED_FWD_LOW_PRESSURE_1	PANEL_C1_3	CQ	65
	Q6	LED_FWD_LOW_PRESSURE_2	PANEL_C1_3	CR	66
	Q7	LIBRE	-	-	-
SR_OUT_4	Q0	PIN 11 DC AMPS (SEGMENTO A)	PANEL_C2_1	CS	6
	Q1	PIN 7 DC AMPS (SEGMENTO B)	PANEL_C2_1	CS	6
	Q2	PIN 4 DC AMPS (SEGMENTO C)	PANEL_C2_1	CS	6
	Q3	PIN 2 DC AMPS (SEGMENTO D)	PANEL_C2_1	CS	6
	Q4	PIN 1 DC AMPS (SEGMENTO E)	PANEL_C2_1	CS	6
	Q5	PIN 10 DC AMPS (SEGMENTO F)	PANEL_C2_1	CS	6
	Q6	PIN 5 DC AMPS (SEGMENTO G)	PANEL_C2_1	CS	6
	Q7	PIN 12 DC AMPS (DÍGITO 1)	PANEL_C2_1	CS	6
SR_OUT_5	Q0	PIN 9 DC AMPS (DÍGITO 2)	PANEL_C2_1	CS	6
	Q1	PIN 8 DC AMPS (DÍGITO 3)	PANEL_C2_1	CS	6
	Q2	PIN 11 CPS FREQ (SEGMENTO A)	PANEL_C2_1	CU	8
	Q3	PIN 7 CPS FREQ (SEGMENTO B)	PANEL_C2_1	CU	8
	Q4	PIN 4 CPS FREQ (SEGMENTO C)	PANEL_C2_1	CU	8
	Q5	PIN 2 CPS FREQ (SEGMENTO D)	PANEL_C2_1	CU	8
	Q6	PIN 1 CPS FREQ (SEGMENTO E)	PANEL_C2_1	CU	8
	Q7	PIN 10 CPS FREQ (SEGMENTO F)	PANEL_C2_1	CU	8
SR_OUT_6	Q0	PIN 5 CPS FREQ (SEGMENTO G)	PANEL_C2_1	CU	8
	Q1	PIN 12 CPS FREQ (DÍGITO 1)	PANEL_C2_1	CU	8
	Q2	PIN 9 CPS FREQ (DÍGITO 2)	PANEL_C2_1	CU	8
	Q3	PIN 8 CPS FREQ (DÍGITO 3)	PANEL_C2_1	CU	8
	Q4	PIN 3 DC VOLTS (SEGMENTO A)	PANEL_C2_1	CV	9
	Q5	PIN 9 DC VOLTS (SEGMENTO B)	PANEL_C2_1	CV	9
	Q6	PIN 8 DC VOLTS (SEGMENTO C)	PANEL_C2_1	CV	9
	Q7	PIN 6 DC VOLTS (SEGMENTO D)	PANEL_C2_1	CV	9
SR_OUT_7	Q0	PIN 7 DC VOLTS (SEGMENTO E)	PANEL_C2_1	CV	9
	Q1	PIN 4 DC VOLTS (SEGMENTO F)	PANEL_C2_1	CV	9
	Q2	PIN 1 DC VOLTS (SEGMENTO G)	PANEL_C2_1	CV	9
	Q3	PIN 10 DC VOLTS (DÍGITO 1)	PANEL_C2_1	CV	9
	Q4	PIN 5 DC VOLTS (DÍGITO 2)	PANEL_C2_1	CV	9
	Q5	PIN 11 AC AMPS (SEGMENTO A)	PANEL_C2_1	CW	10
	Q6	PIN 7 AC AMPS (SEGMENTO B)	PANEL_C2_1	CW	10

	Q7	PIN 4 AC AMPS (SEGMENTO C)	PANEL_C2_1	CW	10
SR_OUT_8	Q0	PIN 2 AC AMPS (SEGMENTO D)	PANEL_C2_1	CW	10
	Q1	PIN 1 AC AMPS (SEGMENTO E)	PANEL_C2_1	CW	10
	Q2	PIN 10 AC AMPS (SEGMENTO F)	PANEL_C2_1	CW	10
	Q3	PIN 5 AC AMPS (SEGMENTO G)	PANEL_C2_1	CW	10
	Q4	PIN 12 AC AMPS (DÍGITO 1)	PANEL_C2_1	CW	10
	Q5	PIN 9 AC AMPS (DÍGITO 2)	PANEL_C2_1	CW	10
	Q6	PIN 8 AC AMPS (DÍGITO 3)	PANEL_C2_1	CW	10
	Q7	PIN 11 AC VOLTS (SEGMENTO A)	PANEL_C2_1	CX	11
SR_OUT_9	Q0	PIN 7 AC VOLTS (SEGMENTO B)	PANEL_C2_1	CX	11
	Q1	PIN 4 AC VOLTS (SEGMENTO C)	PANEL_C2_1	CX	11
	Q2	PIN 2 AC VOLTS (SEGMENTO D)	PANEL_C2_1	CX	11
	Q3	PIN 1 AC VOLTS (SEGMENTO E)	PANEL_C2_1	CX	11
	Q4	PIN 10 AC VOLTS (SEGMENTO F)	PANEL_C2_1	CX	11
	Q5	PIN 5 AC VOLTS (SEGMENTO G)	PANEL_C2_1	CX	11
	Q6	PIN 12 AC VOLTS (DÍGITO 1)	PANEL_C2_1	CX	11
	Q7	PIN 9 AC VOLTS (DÍGITO 2)	PANEL_C2_1	CX	11
SR_OUT_10	Q0	PIN 8 AC VOLTS (DÍGITO 3)	PANEL_C2_1	CX	11
	Q1	LED_BAT_DISCHARGE	PANEL_C2_1	CY	69
	Q2	LED_TR_UNIT	PANEL_C2_1	CZ	70
	Q3	LED_ELEC	PANEL_C2_1	DA	71
	Q4	LED_STANDBY_POWER_OFF	PANEL_C2_2	DB	72
	Q5	LED_DRIVE_1	PANEL_C2_2	DC	73
	Q6	LED_DRIVE_2	PANEL_C2_2	DD	74
	Q7	LED_GROUND_POWER_AVAILABLE	PANEL_C2_3	DE	75
SR_OUT_11	Q0	LED_TRANSFER_BUS_OFF_1	PANEL_C2_3	DF	76
	Q1	LED_TRANSFER_BUS_OFF_2	PANEL_C2_3	DG	77
	Q2	LED_SOURCE_OFF_1	PANEL_C2_3	DH	78
	Q3	LED_SOURCE_OFF_2	PANEL_C2_3	DI	79
	Q4	LED_GEN_OFF_BUS_1	PANEL_C2_3	DJ	80
	Q5	LED_GEN_OFF_BUS_2	PANEL_C2_3	DK	81
	Q6	LED_APU_GEN_OFF_BUS	PANEL_C2_3	DL	82
	Q7	LED_APU_MAINT	PANEL_C2_4	DM	85
SR_OUT_12	Q0	LED_LOW_OIL_PRESSURE	PANEL_C2_4	DN	86
	Q1	LED_FAULT	PANEL_C2_4	DO	87
	Q2	LED_OVER_SPEED	PANEL_C2_4	DP	88
	Q3	LED_EQP_COOLING_SUPPLY	PANEL_C3_2	DR	91
	Q4	LED_COOLING_EXHAUST	PANEL_C3_2	DS	92
	Q5	LED_EMERGENCY_EXIT_NOT_ARMED	PANEL_C3_2	DT	93
	Q6	LED_CALL	PANEL_C3_2	DU	94
	Q7	LIBRE	-	-	-
SR_OUT_13	Q0	LED_OVER_HEAT_1	PANEL_C4_1	DW	98
	Q1	LED_OVER_HEAT_2	PANEL_C4_1	DX	99
	Q2	LED_OVER_HEAT_3	PANEL_C4_1	DY	100
	Q3	LED_OVER_HEAT_4	PANEL_C4_1	DZ	101
	Q4	LED_WH_ON_1	PANEL_C4_1	EA	102
	Q5	LED_WH_ON_2	PANEL_C4_1	EB	103
	Q6	LED_WH_ON_3	PANEL_C4_1	EC	104
	Q7	LED_WH_ON_4	PANEL_C4_1	ED	105
SR_OUT_14	Q0	LED_CAPT_PITOT	PANEL_C4_1	EE	106
	Q1	LED_L_ELEV_PITOT	PANEL_C4_1	EF	107
	Q2	LED_L_ALPHA_VANE	PANEL_C4_1	EG	108
	Q3	LED_TEMP_PROBE	PANEL_C4_1	EH	109
	Q4	LED_FO_PITOT	PANEL_C4_1	EI	110
	Q5	LED_R_ELEV_PITOT	PANEL_C4_1	EJ	111
	Q6	LED_R_ALPHA_VANE	PANEL_C4_1	EK	112
	Q7	LED_AUX_PITOT	PANEL_C4_1	EL	113
SR_OUT_15	Q0	LED_COWL_ANIT_ICE_1	PANEL_C4_2	EQ	116
	Q1	LED_COWL_ANIT_ICE_2	PANEL_C4_2	ER	117
	Q2	LED_L_VALVE_OPEN	PANEL_C4_2	EM	118
	Q3	LED_R_VALVE_OPEN	PANEL_C4_2	EN	119
	Q4	LED_COWL_VALVE_OPEN_1	PANEL_C4_2	EO	120
	Q5	LED_COWL_VALVE_OPEN_2	PANEL_C4_2	EP	121

	Q6	LED_HYD_OVER_HEAT_1	PANEL_C4_3	ES	124
	Q7	LED_HYD_OVER_HEAT_2	PANEL_C4_3	ET	125
SR_OUT_16	Q0	LED_LOW_PRESSURE_ENG_1	PANEL_C4_3	EU	126
	Q1	LED_LOW_PRESSURE_ENG_2	PANEL_C4_3	EV	127
	Q2	LED_LOW_PRESSURE_ELEC_1	PANEL_C4_3	EW	128
	Q3	LED_LOW_PRESSURE_ELEC_2	PANEL_C4_3	EX	129
	Q4	LED_FWD_ENTRY	PANEL_C4_4	EY	132
	Q5	LED_FWD_SERVICE	PANEL_C4_4	EZ	133
	Q6	LED_LEFT_FWD_OVERWING	PANEL_C4_4	FB	135
	Q7	LED_RIGHT_FWD_OVERWING	PANEL_C4_4	FC	136
SR_OUT_17	Q0	LED_FWD_CARGO	PANEL_C4_4	FD	137
	Q1	LED_EQUIP	PANEL_C4_4	FE	138
	Q2	LED_LEFT_AFT_OVERWING	PANEL_C4_4	FF	139
	Q3	LED_RIGHT_AFT_OVERWING	PANEL_C4_4	FG	140
	Q4	LED_AFT_CARGO	PANEL_C4_4	FH	141
	Q5	LED_AFT_ENTRY	PANEL_C4_4	FI	142
	Q6	LED_AFT_SERVICE	PANEL_C4_4	FJ	143
	Q7	LED_TEST	PANEL_C4_5	ZZ	-
SR_OUT_18	Q0	LED_ZONE_TEMP_1	PANEL_C5_2	FO	146
	Q1	LED_ZONE_TEMP_2	PANEL_C5_2	FP	147
	Q2	LED_ZONE_TEMP_3	PANEL_C5_2	FQ	148
	Q3	LED_DUAL_BLEED	PANEL_C5_3	FR	149
	Q4	LED_RAM_DOOR_FULL_OPEN_1	PANEL_C5_3	FS	150
	Q5	LED_RAM_DOOR_FULL_OPEN_2	PANEL_C5_3	FT	151
	Q6	LED_PACK_TRIP_OFF_1	PANEL_C5_3	FU	152
	Q7	LED_PACK_TRIP_OFF_2	PANEL_C5_3	FV	153
SR_OUT_19	Q0	LED_WING_BODY_OVER_HEAT_1	PANEL_C5_3	FW	154
	Q1	LED_WING_BODY_OVER_HEAT_2	PANEL_C5_3	FX	155
	Q2	LED_BLEED_TRIP_OFF_1	PANEL_C5_3	FY	156
	Q3	LED_BLEED_TRIP_OFF_2	PANEL_C5_3	FZ	157
	Q4	LED_AUTO_FAIL	PANEL_C5_3	GC	160
	Q5	LED_OFF_SCHED_DESCENT	PANEL_C5_3	GD	161
	Q6	LED_ALTN	PANEL_C5_3	GE	162
	Q7	LED_MANUAL	PANEL_C5_3	GF	163
SR_OUT_20	Q0	PIN 13 AUTO FLT ALT (SEGMENTO A)	PANEL_C5_4	GG	14
	Q1	PIN 9 AUTO FLT ALT (SEGMENTO B)	PANEL_C5_4	GG	14
	Q2	PIN 4 AUTO FLT ALT (SEGMENTO C)	PANEL_C5_4	GG	14
	Q3	PIN 2 AUTO FLT ALT (SEGMENTO D)	PANEL_C5_4	GG	14
	Q4	PIN 1 AUTO FLT ALT (SEGMENTO E)	PANEL_C5_4	GG	14
	Q5	PIN 12 AUTO FLT ALT (SEGMENTO F)	PANEL_C5_4	GG	14
	Q6	PIN 5 AUTO FLT ALT (SEGMENTO G)	PANEL_C5_4	GG	14
	Q7	PIN 14 AUTO FLT ALT (DÍGITO 1)	PANEL_C5_4	GG	14
SR_OUT_21	Q0	PIN 11 AUTO FLT ALT (DÍGITO 2)	PANEL_C5_4	GG	14
	Q1	PIN 10 AUTO FLT ALT (DÍGITO 3)	PANEL_C5_4	GG	14
	Q2	PIN 6 AUTO FLT ALT (DÍGITO 4)	PANEL_C5_4	GG	14
	Q3	PIN 8 AUTO FLT ALT (DÍGITO 5)	PANEL_C5_4	GG	14
	Q4	PIN 13 AUTO LAND ALT (SEGMENTO A)	PANEL_C5_4	GH	15
	Q5	PIN 9 AUTO LAND ALT (SEGMENTO B)	PANEL_C5_4	GH	15
	Q6	PIN 4 AUTO LAND ALT (SEGMENTO C)	PANEL_C5_4	GH	15
	Q7	PIN 2 AUTO LAND ALT (SEGMENTO D)	PANEL_C5_4	GH	15
SR_OUT_22	Q0	PIN 1 AUTO LAND ALT (SEGMENTO E)	PANEL_C5_4	GH	15
	Q1	PIN 12 AUTO LAND ALT (SEGMENTO F)	PANEL_C5_4	GH	15
	Q2	PIN 5 AUTO LAND ALT (SEGMENTO G)	PANEL_C5_4	GH	15
	Q3	PIN 14 AUTO LAND ALT (DÍGITO 1)	PANEL_C5_4	GH	15
	Q4	PIN 11 AUTO LAND ALT (DÍGITO 2)	PANEL_C5_4	GH	15
	Q5	PIN 10 AUTO LAND ALT (DÍGITO 3)	PANEL_C5_4	GH	15
	Q6	PIN 6 AUTO LAND ALT (DÍGITO 4)	PANEL_C5_4	GH	15
	Q7	PIN 8 AUTO LAND ALT (DÍGITO 5)	PANEL_C5_4	GH	15

## 11.15 Asignación de los elementos de entrada a los pines del 74HC165 (SR\_IN)

Tabla 12. Asignación de los elementos de entrada a los pines del 74HC165 (SR\_IN)

SHIFT REGISTER	PIN	ELEMENTO	PANEL	CÓDIGO	ID	CASE
SR_IN_1	D0	SWITCH_STBY_RUD_A1	PANEL_C1_1	HS	52	0
	D1	SWITCH_STBY_RUD_A2	PANEL_C1_1	HS	52	1
	D2	SWITCH_STBY_RUD_B1	PANEL_C1_1	HT	53	2
	D3	SWITCH_STBY_RUD_B2	PANEL_C1_1	HT	53	3
	D4	SWITCH_ALTERNATE_FLAPS	PANEL_C1_1	HU	54	4
	D5	SWITCH_FLAPS_POS_1	PANEL_C1_1	HV	55	5
	D6	SWITCH_FLAPS_POS_2	PANEL_C1_1	HV	55	6
	D7	SWITCH_SPOILER_A	PANEL_C1_1	HW	56	7
SR_IN_2	D0	SWITCH_SPOILER_B	PANEL_C1_1	HX	57	8
	D1	SWITCH_YAW_DAMPER	PANEL_C1_1	HY	58	9
	D2	SWITCH_VHF_NAVIGATION_1	PANEL_C1_2	HZ	61	10
	D3	SWITCH_VHF_NAVIGATION_2	PANEL_C1_2	HZ	61	11
	D4	SWITCH_IRS_SELECTOR_1	PANEL_C1_2	IA	62	12
	D5	SWITCH_IRS_SELECTOR_2	PANEL_C1_2	IA	62	13
	D6	SWITCH_FMC_1	PANEL_C1_2	IB	63	14
	D7	SWITCH_FMC_2	PANEL_C1_2	IB	63	15
SR_IN_3	D0	SELECTOR_DISPLAY_SOURCE_1	PANEL_C1_2	IC	64	16
	D1	SELECTOR_DISPLAY_SOURCE_2	PANEL_C1_2	IC	64	17
	D2	SWITCH_CONTROL_PANEL_1	PANEL_C1_2	ID	65	18
	D3	SWITCH_CONTROL_PANEL_2	PANEL_C1_2	ID	65	19
	D4	SWITCH_CROSS_FEED	PANEL_C1_3	IE	68	20
	D5	SWITCH_FUEL_PUMPS_A	PANEL_C1_3	IF	69	21
	D6	SWITCH_FUEL_PUMPS_B	PANEL_C1_3	IG	70	22
	D7	SWITCH_AFT_FUEL_PUMP_1	PANEL_C1_3	IH	71	23
SR_IN_4	D0	SWITCH_AFT_FUEL_PUMP_2	PANEL_C1_3	II	72	24
	D1	SWITCH_FWD_FUEL_PUMP_1	PANEL_C1_3	IJ	73	25
	D2	SWITCH_FWD_FUEL_PUMP_2	PANEL_C1_3	IK	74	26
	D3	SWITCH_RETRACTABLE_LDG_LIGHTS_L_1	PANEL_C1_4	Ab	154	27
	D4	LIBRE (GND)	-	-	-	-
	D5	SWITCH_RETRACTABLE_LDG_LIGHTS_R_1	PANEL_C1_4	Ac	155	29
	D6	LIBRE (GND)	-	-	-	-
	D7	SWITCH_FIXED_LDG_LIGHTS_L	PANEL_C1_4	Ad	156	31
SR_IN_5	D0	SWITCH_FIXED_LDG_LIGHTS_R	PANEL_C1_4	Ae	157	32
	D1	SWITCH_RWY_LIGHT_L	PANEL_C1_4	Af	158	33
	D2	SWITCH_RWY_LIGHT_R	PANEL_C1_4	Ag	159	34
	D3	SWITCH_TAXI_LIGHT	PANEL_C1_4	Ah	160	35
	D4	SWITCH_APU_1	PANEL_C1_4	Ai	161	36
	D5	SWITCH_APU_2	PANEL_C1_4	Ai	161	37
	D6	SELECTOR_ENGINE_STATOR_L_1	PANEL_C1_4	Aj	162	38
	D7	SELECTOR_ENGINE_STATOR_L_2	PANEL_C1_4	Aj	162	39
SR_IN_6	D0	SELECTOR_ENGINE_STATOR_L_3	PANEL_C1_4	Aj	162	40
	D1	SELECTOR_ENGINE_STATOR_L_4	PANEL_C1_4	Aj	162	41
	D2	SELECTOR_ENGINE_STATOR_R_1	PANEL_C1_4	Ak	163	42
	D3	SELECTOR_ENGINE_STATOR_R_2	PANEL_C1_4	Ak	163	43
	D4	SELECTOR_ENGINE_STATOR_R_3	PANEL_C1_4	Ak	163	44
	D5	SELECTOR_ENGINE_STATOR_R_4	PANEL_C1_4	Ak	163	45
	D6	SWITCH_IGNITION_1	PANEL_C1_4	Al	164	46
	D7	SWITCH_IGNITION_2	PANEL_C1_4	Al	164	47
SR_IN_7	D0	SWITCH_LOGO_LIGHT	PANEL_C1_4	Am	165	48
	D1	SWITCH_STROBE_LIGHT_1	PANEL_C1_4	An	166	49
	D2	SWITCH_STROBE_LIGHT_2	PANEL_C1_4	An	166	50
	D3	SWITCH_ANTI_COLLISION_LIGHT	PANEL_C1_4	Ao	167	51
	D4	SWITCH_WING_LIGHT	PANEL_C1_4	Ap	168	52
	D5	SWITCH_WHEEL_WELL_LIGHT	PANEL_C1_4	Aq	169	53
	D6	LIBRE (GND)	-	-	-	-
	D7	LIBRE (GND)	-	-	-	-

SR_IN_8	D0	BUTTON_MAINT	PANEL_C2_1	IM	78	0
	D1	SELECTOR_DC_SOURCE_0	PANEL_C2_1	IN	79	1
	D2	SELECTOR_DC_SOURCE_1	PANEL_C2_1	IN	79	2
	D3	SELECTOR_DC_SOURCE_2	PANEL_C2_1	IN	79	3
	D4	SELECTOR_DC_SOURCE_3	PANEL_C2_1	IN	79	4
	D5	SELECTOR_DC_SOURCE_4	PANEL_C2_1	IN	79	5
	D6	SELECTOR_DC_SOURCE_5	PANEL_C2_1	IN	79	6
	D7	SELECTOR_DC_SOURCE_6	PANEL_C2_1	IN	79	7
SR_IN_9	D0	SELECTOR_DC_SOURCE_7	PANEL_C2_1	IN	79	8
	D1	SELECTOR_AC_SOURCE_0	PANEL_C2_1	IO	80	9
	D2	SELECTOR_AC_SOURCE_1	PANEL_C2_1	IO	80	10
	D3	SELECTOR_AC_SOURCE_2	PANEL_C2_1	IO	80	11
	D4	SELECTOR_AC_SOURCE_3	PANEL_C2_1	IO	80	12
	D5	SELECTOR_AC_SOURCE_4	PANEL_C2_1	IO	80	13
	D6	SELECTOR_AC_SOURCE_5	PANEL_C2_1	IO	80	14
	D7	SELECTOR_AC_SOURCE_6	PANEL_C2_1	IO	80	15
SR_IN_10	D0	SWITCH_MAIN_BATTERY	PANEL_C2_1	IP	81	16
	D1	SWITCH_CAB_UTIL	PANEL_C2_1	IQ	82	17
	D2	SWITCH_IFE_PASS_SEAT	PANEL_C2_1	IR	83	18
	D3	SWITCH_STANDBY_POWER_1	PANEL_C2_2	IS	84	19
	D4	SWITCH_STANDBY_POWER_2	PANEL_C2_2	IS	84	20
	D5	SWITCH_DRIVE_1	PANEL_C2_2	IT	85	21
	D6	SWITCH_DRIVE_2	PANEL_C2_2	IU	86	22
	D7	SWITCH_GROUND_POWER_ON	PANEL_C2_3	IV	87	23
SR_IN_11	D0	SWITCH_GROUND_POWER_OFF	PANEL_C2_3	IV	87	24
	D1	SWITCH_BUS_TRANSFER	PANEL_C2_3	IW	88	25
	D2	SWITCH_MAIN_GEN_1_ON	PANEL_C2_3	IX	89	26
	D3	SWITCH_MAIN_GEN_1_OFF	PANEL_C2_3	IX	89	27
	D4	SWITCH_MAIN_GEN_2_ON	PANEL_C2_3	IY	90	28
	D5	SWITCH_MAIN_GEN_2_OFF	PANEL_C2_3	IY	90	29
	D6	SWITCH_APU_GEN_1_ON	PANEL_C2_3	IZ	91	30
	D7	SWITCH_APU_GEN_1_OFF	PANEL_C2_3	IZ	91	31
SR_IN_12	D0	SWITCH_APU_GEN_2_ON	PANEL_C2_3	JA	92	32
	D1	SWITCH_APU_GEN_2_OFF	PANEL_C2_3	JA	92	33
	D2	SELECTOR_WIPER_L_0	PANEL_C2_4	JB	93	34
	D3	SELECTOR_WIPER_L_1	PANEL_C2_4	JB	93	35
	D4	SELECTOR_WIPER_L_2	PANEL_C2_4	JB	93	36
	D5	SELECTOR_WIPER_L_3	PANEL_C2_4	JB	93	37
	D6	SWITCH_EQUIP_SUPPLY	PANEL_C3_2	JD	100	38
	D7	SWITCH_EQUIP_EXHAUST	PANEL_C3_2	JE	101	39
SR_IN_13	D0	SWITCH_EMER_EXIT_LIGHT_1	PANEL_C3_2	JF	102	40
	D1	SWITCH_EMER_EXIT_LIGHT_2	PANEL_C3_2	JF	102	41
	D2	SWITCH_CHIME	PANEL_C3_2	JG	103	42
	D3	SWITCH_FASTEN BELTS_1	PANEL_C3_2	JH	104	43
	D4	SWITCH_FASTEN BELTS_2	PANEL_C3_2	JH	104	44
	D5	BUTTON_ATTEND	PANEL_C3_2	JI	105	45
	D6	BUTTON_GRD_CALL	PANEL_C3_2	JJ	106	46
	D7	SELECTOR_WIPER_R_0	PANEL_C3_2	JC	94	47
SR_IN_14	D0	SELECTOR_WIPER_R_1	PANEL_C3_2	JC	94	48
	D1	SELECTOR_WIPER_R_2	PANEL_C3_2	JC	94	49
	D2	SELECTOR_WIPER_R_3	PANEL_C3_2	JC	94	50
	D3	SWITCH_WH_SIDE_1	PANEL_C4_1	JK	109	51
	D4	SWITCH_WH_SIDE_2	PANEL_C4_1	JL	110	52
	D5	SWITCH_WH_FWD_1	PANEL_C4_1	JM	111	53
	D6	SWITCH_WH_FWD_2	PANEL_C4_1	JN	112	54
	D7	SWITCH_PROBE_HEAT_A	PANEL_C4_1	JO	113	55
SR_IN_15	D0	SWITCH_PROBE_HEAT_B	PANEL_C4_1	JP	114	0
	D1	SWITCH_OVERHEAT_1	PANEL_C4_1	JQ	115	1
	D2	SWITCH_OVERHEAT_2	PANEL_C4_1	JQ	115	2
	D3	BUTTON_TAT_TEST	PANEL_C4_1	-	-	3
	D4	SWITCH_WING_ANTI_ICE	PANEL_C4_2	JR	118	4
	D5	SWITCH_ENGINE_ANTI_ICE_1	PANEL_C4_2	JS	119	5
	D6	SWITCH_ENGINE_ANTI_ICE_2	PANEL_C4_2	JT	120	6

	D7	SWITCH_ENGINE_HYD_1	PANEL_C4_3	JU	123	7
SR_IN_16	D0	SWITCH_ENGINE_HYD_2	PANEL_C4_3	JV	124	8
	D1	SWITCH_ELECTRIC_HYD_1	PANEL_C4_3	JW	125	9
	D2	SWITCH_ELECTRIC_HYD_2	PANEL_C4_3	JX	126	10
	D3	BUTTON_COCKPIT_VOICE_TEST	PANEL_C4_5	JY	150	11
	D4	BUTTON_COCKPIT_VOICE_ERASE	PANEL_C4_5	JZ	151	12
	D5	BUTTON_ALT_HORN_CUTOFF	PANEL_C4_6	As	444	13
	D6	SWITCH_VOICE_RECORDER	PANEL_C5_1	IL	77	14
	D7	SELECTOR_AIR_TEMP_0	PANEL_C5_2	KA	129	15
SR_IN_17	D0	SELECTOR_AIR_TEMP_1	PANEL_C5_2	KA	129	16
	D1	SELECTOR_AIR_TEMP_2	PANEL_C5_2	KA	129	17
	D2	SELECTOR_AIR_TEMP_3	PANEL_C5_2	KA	129	18
	D3	SELECTOR_AIR_TEMP_4	PANEL_C5_2	KA	129	19
	D4	SELECTOR_AIR_TEMP_5	PANEL_C5_2	KA	129	20
	D5	SELECTOR_AIR_TEMP_6	PANEL_C5_2	KA	129	21
	D6	SWITCH_TRIM_AIR	PANEL_C5_2	At	445	22
	D7	SWITCH_RECIRC_FAN_L	PANEL_C5_3	KE	133	23
SR_IN_18	D0	SWITCH_RECIRC_FAN_R	PANEL_C5_3	KF	134	24
	D1	BUTTON_OVHT_TEST	PANEL_C5_3	Au	446	25
	D2	SWITCH_ISOLATION_VALVE_1	PANEL_C5_3	KG	135	26
	D3	SWITCH_ISOLATION_VALVE_2	PANEL_C5_3	KG	135	27
	D4	BUTTON_TRIP_RESET	PANEL_C5_3	KH	136	28
	D5	SWITCH_PACK_L_1	PANEL_C5_3	KI	137	29
	D6	SWITCH_PACK_L_2	PANEL_C5_3	KI	137	30
	D7	SWITCH_PACK_R_1	PANEL_C5_3	KJ	138	31
SR_IN_19	D0	SWITCH_PACK_R_2	PANEL_C5_3	KJ	138	32
	D1	SWITCH_BLEED_AIR_1	PANEL_C5_3	KK	139	33
	D2	SWITCH_BLEED_AIR_2	PANEL_C5_3	KL	140	34
	D3	SWITCH_APU_BLEED_AIR	PANEL_C5_3	KM	141	35
	D4	SELECTOR_PRESSURIZATION_1	PANEL_C5_4	KP	146	36
	D5	SELECTOR_PRESSURIZATION_2	PANEL_C5_4	KP	146	37
	D6	SWITCH_VALVE_1	PANEL_C5_4	KQ	147	38
	D7	SWITCH_VALVE_2	PANEL_C5_4	KQ	147	39

## 11.16 Repositorio web GitHub

A lo largo de este proyecto se han ido generando multitud de archivos que, por el formato de este informe, son imposibles de compartir. Estos son, por ejemplo, los archivos de diseño de las piezas para impresión 3D. Otros, como el código de Arduino, harían que este informe tuviera excesivas páginas. Además, al ser un código en evolución, este se quedaría rápidamente obsoleto, siendo ineficiente añadirlo en un anexo. Por ello, se utiliza la plataforma GitHub, un repositorio web donde todos los archivos del proyecto están alojados. En él se podrá encontrar la última versión del código Arduino, las piezas para impresión 3D (tanto los archivos STL, como los de FreeCAD y las pegatinas) y una copia digital de este documento.

El enlace al repositorio de GitHub llamado *Overhead-Panel-B737NG* es el siguiente:

<https://github.com/PilotoF-22/Overhead-Panel-B737NG>