Facultad de Ingeniería, Universidad Nacional de La Plata

ObservAR – Informe Final

Garay Francisco	Lambre Jerónimo	Bruno Laureano	Zeballos Matías
02714/4	02628/7	02585/3	02255/7

Grupo 5

Taller de Proyecto I – Ingeniería en Computación

La Plata

6/2/24

Índice

2.1. Introducción

A lo largo del tiempo, la tecnología ha sido utilizada para la compleja labor de solucionar problemas del mundo real, así pues, el proyecto planteado en este plan propone resolver una cuestión del orden de la seguridad. En particular, el desafío sobre el que se fundamenta el desarrollo de este trabajo es el del reconocimiento y seguimiento de personas mediante una estructura compuesta por una cámara y un brazo robótico que mueva el dispositivo en pos de que la imagen continue el recorrido de la persona en cuestión.

Resulta determinante destacar que, si bien el problema que busca resolverse es el del seguimiento automático de personas mediante el procesamiento de imágenes en cualquier ambiente (sistema que podría ser utilizado en lugares con un gran tráfico de personas que requieran vigilancia, como lo es un aeropuerto), el proyecto aquí planteado se ocupará de sentar las bases para un futuro desarrollo de tales magnitudes. Es por esto por lo que el sistema a desarrollar se plantea para un ambiente controlado (misma iluminación, contraste, fondo, y demás factores que afecten al reconocimiento) y un objeto en específico que no modifique su forma y para el cual se tendrá un modelo de datos bien entrenado (la elección del objeto variará según el modelo de datos que se escoja durante el proceso de desarrollo del sistema).

Con respecto a la implementación del proyecto se evaluaron distintas formas de hacerlo hasta que se llegó a la conclusión de que la mejor forma de capturar imágenes para el reconocimiento es el uso de una placa de desarrollo como lo es el ESP32 junto con un módulo de cámara. La elección de esta placa radica en que la misma, aparte de ser de bajo costo y consumo, y de programación simple (tipo Arduino), cuenta con dos núcleos que se programan y trabajan por separado. A su vez, esta placa se comunicará con una placa de desarrollo EDU-CIAA mediante el protocolo serie UART. La CIAA será la encargada de generar las señales para mover al servo que controla la rotación sobre el eje x de la estructura. En adición, el sistema cuenta con una serie de botones físicos que permitirán al usuario realizar tareas básicas de configuración sobre el sistema: START, STOP, RESET.

2.2. Objetivos

El objetivo principal de este proyecto es el diseño e implementación de un brazo robótico para el seguimiento de objetos mediante reconocimiento de imágenes en tiempo real. Este sistema podría implementarse en líneas de ensamblaje automatizadas, mejorando la eficiencia y precisión en la selección y colocación de componentes. En el sector de la salud, podría asistir en intervenciones quirúrgicas, ofreciendo movimientos precisos y estables

guiados por imágenes médicas. En el ámbito de la seguridad, el brazo podría usarse para la vigilancia y el monitoreo, identificando y siguiendo objetos o individuos automáticamente.

El proyecto se divide en los siguientes módulos:

- Detección: con un algoritmo que determine la presencia del objeto y su posición.
- Seguimiento: con un algoritmo de control que siga al objeto en base a la posición del mismo.
- Alimentación: con circuitos específicos para energizar todos los componentes de manera adecuada.
- Comunicación: se definirá un protocolo para la comunicación cableada entre las placas EDU-CIAA y ESP32-CAM.

En la Figura 1 se presenta el diagrama en bloques del sistema a desarrollar.

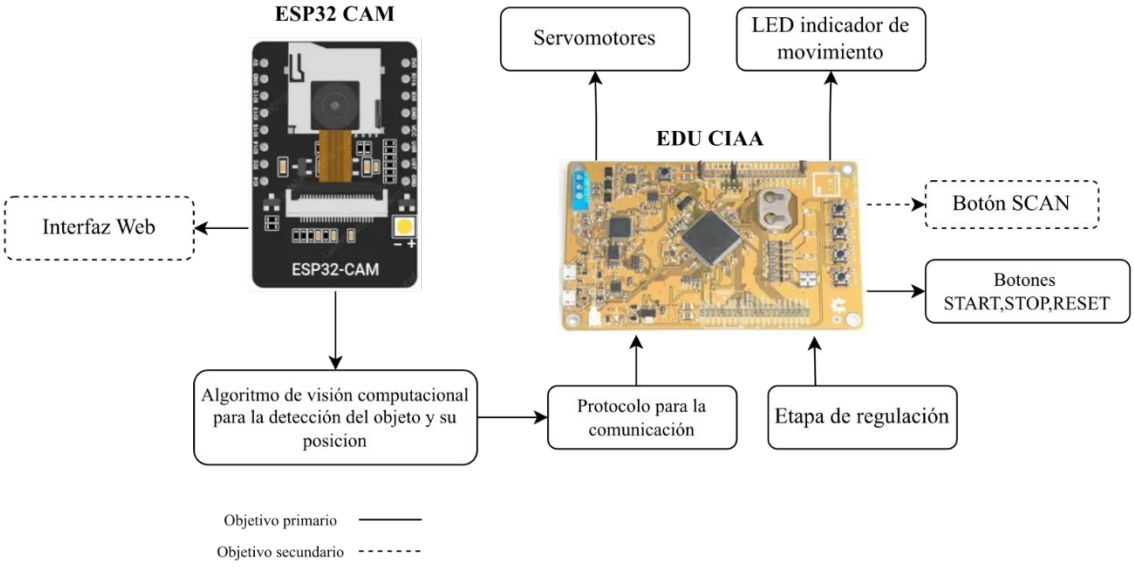


Figura 1 - Diagrama en bloques del sistema del brazo robótico

2.3. Objetivos Primarios

2.3.1. Movimiento del brazo robótico

Se busca crear un brazo robótico con un eje de libertad, utilizando un servomotor controlado por la placa EDU-CIAA. Será necesario mover el servomotor tantos grados como corresponda según dónde se encuentre ubicado el objeto en la imagen. Se deberá entonces calcular una distancia de posición entre el centro del eje de referencia y la posición detectada del objeto a seguir, y a partir de esta magnitud realizar una conversión a grados de rotación, pudiéndose considerar algún margen de error para no sobrecargar al servo de movimientos innecesarios y teniendo en cuenta una posible histéresis en cada movimiento.

Por otro lado, el movimiento estará condicionado por el accionar de tres botones por parte de los usuarios: START, STOP y RESET. El primero comenzará el movimiento del brazo, el segundo lo detendrá, y el tercero lo reiniciará a su posición por defecto (centrado en el origen y detenido).

2.3.2. Visión del brazo

Se deberá integrar una cámara ESP32-CAM que, mediante el procesamiento de imágenes utilizando algoritmos de inteligencia artificial optimizados para MCUs (Área de la computación denominada TinyML), determine la presencia de un objeto específico y su ubicación, debiendo ser este cálculo lo suficientemente rápido para que se note una cierta correlación entre el movimiento del objeto y la reacción del brazo.

2.3.3. Comunicación entre MCUs

Establecer una comunicación efectiva entre la cámara ESP32-CAM y la placa EDU-CIAA de modo que la primera le informe a la segunda, a partir de alguna interfaz de comunicación como SPI o USART, si se ha detectado el objeto y en qué posición. Esto se deberá llevar a cabo mediante la implementación de algún protocolo para la comunicación cableada entre MCUs, lo suficientemente veloz para asegurar la fluidez del sistema.

2.3.4. Diseño físico

Realizar el conexionado físico de los componentes electrónicos mediante el diseño y construcción de un "poncho" que se conecte a la placa EDU-CIAA para administrar de manera segura la alimentación externa del sistema, los servomotores, un botón para reiniciar el sistema y la interfaz de comunicación entre los MCUs.

2.4. Objetivos Secundarios

2.4.1. Segundo eje de movimiento

Una vez implementado y validado el movimiento con un grado de libertad del sistema (Con respecto al eje Vertical), se desea poder ampliar su funcionalidad agregando un segundo grado de libertad (Con respecto al eje Horizontal) para así poder seguir con mayor exactitud y en una mayor área de detección el objeto a detectar. Esto se realizaría bajo la asunción de que la suma del procesamiento y rotación de cada eje individual da como resultado la rotación requerida en dos grados de libertad.

2.4.2. Interfaz web

Desplegar una interfaz web en una WLAN donde se pueda visualizar en tiempo real lo que captura la cámara ubicada en el brazo robótico. Esto le permitirá al usuario poder visualizar lo que se está procesando en cada momento por la ESP-CAM en tiempo real.

2.4.3. Dashboard web

Desplegar una interfaz web en una WLAN donde se puedan visualizar estadísticas del brazo como historial de movimientos, cantidad de veces que reconoció al objeto, cantidad de veces que se reinició o que escaneó el entorno, etc.

3. Análisis de requerimientos

El análisis de requerimientos es esencial para garantizar que el sistema cumpla con las expectativas y necesidades del usuario final, así como para asegurar su funcionamiento eficiente y confiable. Los mismos se detallan a continuación:

3.1. Requerimientos funcionales

- Detectar presencia del objeto
- Detectar ubicación del objeto
- Seguir el objeto dentro del rango de 180 grados
- Si el objeto desaparece de imagen, el brazo debe mantener la última posición donde lo detectó.
- Implementar de una interfaz web para ver la cámara.
- Botón START para comenzar el movimiento del brazo.
- Botón STOP para detener el movimiento del brazo.
- Botón RESET para reiniciar el brazo a su estado por defecto (centrado en el origen y detenido).

3.2. Requerimientos no funcionales

- El movimiento debe ser “Fluido”
- Evitar falsas detecciones.
- Utilizar una fuente de alimentación alterna 220V 50Hz
- Visualizar la interfaz desde dispositivos móviles

3.3. Hardware a utilizar

- ESP32-CAM

- 2 servomotores
- 3x Pulsadores
- Fuente AC/DC
- Reguladores de tensión lineales (LM317)
- Capacitores cerámicos y electrolíticos
- Resistencias

4. Diseño de hardware

4.1. Lista de Materiales

A continuación, se presenta la lista final de componentes con los datos de los encapsulados y el footprint correspondiente en la PCB diseñada en Proteus

Tabla 1 – Lista completa de materiales con su encapsulado y footprint

	Componente	Encapsulado	Footprint	Cantidad	Valor Unitario	Valor Total	Fecha	Link compra
Cámara	ESP32-CAM	-	-	1	$ 9,777.99	$ 9,777.99	02-oct-23	MercadoLibre
	Programador ESP32-CAM	-	-	1	$ 3,532.00	$ 3,532.00	02-oct-23	MercadoLibre
Movimiento de la Torreta	Botón pulsador	6mm×6mm Top Push	BUTTON PCK	3	$ 100.00	$ 300.00	02-oct-23	MercadoLibre
	Servomotor SG90	-	-	2	$ 2,250.00	$ 4,500.00	02-oct-23	MercadoLibre
Etapa de regulación de Tensión	Regulador De Tensión LM317	KCS (TO-220)	TO220	2	$ 695.00	$ 1,390.00	02-oct-23	MercadoLibre
	Capacitor electrolítico de 10µF 16V	Radial Electrolítico	CAPPRD200W50D500H1250	2	$ 91.90	$ 183.80	02-oct-23	MercadoLibre
	Capacitor cerámico de 100 nF	Disco cerámico	CAP20	2	$ 138.20	$ 276.40	02-oct-23	MercadoLibre
	Resistencia de 4k7Ω	1/4w	RES40	1	$ 75.00	$ 75.00	09-nov-23	MercadoLibre
	Resistencia de 1k2Ω	1/4w	RES40	1	$ 75.00	$ 75.00	09-nov-23	MercadoLibre
	Resistencia de 600Ω	1/4w	RES40	1	$ 75.00	$ 75.00	09-nov-23	MercadoLibre

	Resistencia de 220Ω	1/4w	RES40	1	$ 75.00	$ 75.00	09-nov-23	MercadoLibre
	Mini Jumpers	-	-	2	$ 40.00	$40.00	12-dic-23	MercadoLibre
	Jack DC	Hembra 5.5x2.1 Mm	JACK DC PACK	1	$ 367.00	$ 367.00	09-nov-23	MercadoLibre

4.2. Etapa de regulación de tensión

El sistema deberá conectarse a 220V AC. Para ello, se dispone de una fuente de PC con salidas de 5V y 12V de continua. El poncho a diseñar debe admitir estas tensiones en la entrada y en caso de que se alimente con 12V, se debe implementar una etapa de regulación de tensión para energizar el resto de los componentes con las tensiones adecuadas. Para ello, es importante entender el consumo de corriente de cada componente involucrado. A continuación, se presentan los consumos máximos de los elementos del proyecto:

- **Servos SG90**: Corriente máxima por servomotor (Arduino Forum, 2023): 680 mA, tomando el valor más alto para tener un margen de seguridad.

$$I_{servos} = 680mA$$

- **Botones**: Para este proyecto, su consumo es insignificante.
- **ESP32-CAM**: El consumo de máximo puede llegar a los $300mA$.
- **Placa EDU-CIAA**: El consumo de máximo puede llegar a los $200mA$.

Teniendo en cuenta estos datos, el consumo de corriente máximo teórico es de

$$680mA + 300mA + 200mA = 1.18A$$

Con el objetivo de que no se disipe tanta potencia en solo 1 regulador, se determinó el uso de 2 etapas reguladoras. Una alimentará la EDU-CIAA y el ESP-32, mientras que la otra alimentará los servomotores. El consumo de corriente máximo teórico para cada etapa será:

$$300mA + 200mA = 500mA \Rightarrow Placas$$

$$680mA \Rightarrow Servomotor$$

Antes de llevar a cabo el diseño de la regulación, se investigaron múltiples modelos de reguladores lineales como el LM7805, LM317 y el UCC283. (las fuentes switching fueron descartadas para poder colocar más componentes en el poncho).

En cuanto al LM7805, si bien las tensiones de entrada y salida se encuentran en rango, la corriente de salida máxima[1] es de 1A y no alcanza para la aplicación buscada dado que los picos de corrientes de la inicialización de la cámara y wifi del ESP32 pueden llegar a superar este valor. Por otro lado, el UCC283 solo permite hasta 9V de entrada[2], condición que limita su uso en esta aplicación.

Finalmente, el regulador lineal de tensión elegido es el LM317 ya que admite hasta 1.5A de salida, tensiones variables de salida de 1.2V a 37V y tensión de entrada de hasta 40V[3]

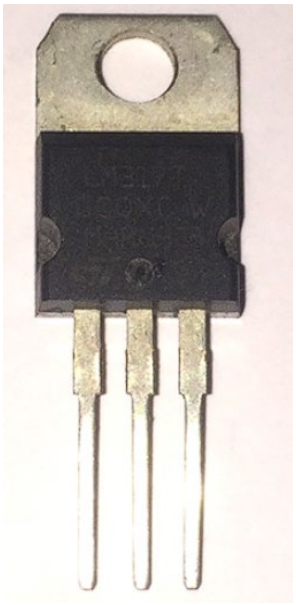


Figura 2 – LM317 – Encapsulado T0220

El circuito físico para para la conexión del componente sigue el siguiente esquema definido en la hoja de datos

[1] Datasheet LM7805
[2] Datasheet UCC283
[3] Datasheet LM317

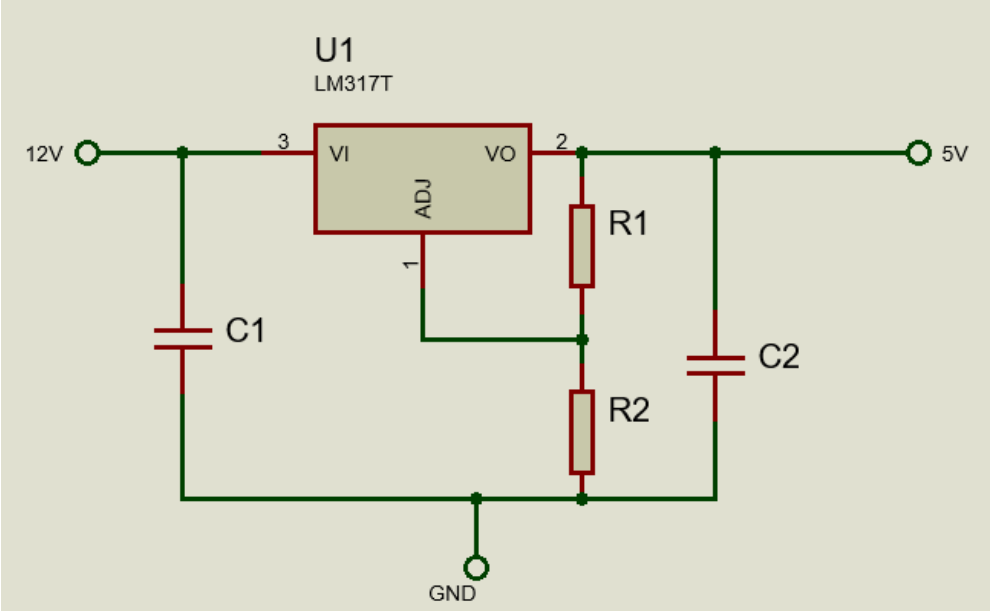


Figura 3 – Circuito regulador de tensión utilizando un LM317

Cada regulador debe estar acompañado de dos capacitores, uno cerámico en la entrada (C1) y uno a la salida (C2) de tipo electrolítico. Los valores de capacitancia recomendados por el fabricante para cumplir estos objetivos son de 100nF para el cerámico y de 1uF a 10uF 16V para el electrolítico.

Por otra parte, se deben elegir las resistencias correctas para que la salida sea aproximadamente 5V en caso de la alimentación de las placas y 6.5V para la alimentación del servomotor. La ecuación que define la tensión de salida[4] es la siguiente:

$$V_{out} = V_{ref} * \left(1 + \frac{R2}{R1}\right) + I_{adj} * R2$$

Donde V_{ref} es la tensión de referencia del LM317 (1.25V) e I_{adj}es una corriente pequeña que fluye a través de R2 (despreciable). Teniendo esto en cuenta estos parámetros, la ecuación final es:

$$V_{out} = 1.25 * \left(1 + \frac{R2}{R1}\right)$$

Por lo que para $V_{out} = 5V$ (para las placas)

$$\frac{R2}{R1} = 3$$

Y para $V_{out} = 6.5V$ (para el servomotor)

$$\frac{R2}{R1} = 4{,}2$$

[4] Extraído de la detasheet del LM317

Utilizando resistencias comerciales de 220Ω para R1 y 680Ω para R2 se obtiene una tensión de salida de:

$$1.25 * \left(1 + \frac{680\Omega}{220\Omega}\right) = 5.11V$$

Dado que estas dan como resultado un valor cercano a 5V sin superarlo significativamente, fueron las que se eligieron a la hora de la construcción de la placa. Luego, al medir con un multímetro las resistencias y tensión de salida con el sistema funcionando, se encontró que los valores reales difieren ligeramente de los teóricos, tal como se puede ver a continuación

$$1.25 * \left(1 + \frac{654\Omega}{215\Omega}\right) = 5.05V$$

La cual es aceptable para alimentar tanto la CIAA como la ESP dado que ambas placas regulan sus entradas de 5V a 3.3V para su debido funcionamiento.

Por otra parte, para la alimentación del servomotor, se utilizaron resistencias comerciales de 1.2kΩ para R1 y 4.7kΩ para R2 se obtiene una tensión de salida de:

$$1.25 * \left(1 + \frac{4.7k\Omega}{1.2k\Omega}\right) = 6.14V$$

Nuevamente, al medir con un multímetro las resistencias, se eligieron aquellas que favorecen la proporción para que se encuentre lo más cercano a 6.5V posible. Al momento de la construcción de la placa, se utilizaron resistencias lo más cercanas posibles a los valores calculados para que no exceda la tensión buscada.

$$1.25 * \left(1 + \frac{4.7k\Omega}{1.13k\Omega}\right) = 6.45V$$

Otra consideración que se debe tomar a la hora de utilizar este regulador son las características térmicas del mismo. La potencia disipada en cada uno de los reguladores, con 12V en la entrada y 5V a la salida es de

$$(12V - 5V) * 0.5A = 7V * 0.5A = 3.5W \; para \; el \; regulador \; que \; alimenta \; las \; placas$$

$$(12V - 6.5V) * 0.68A = 7V * 0.68A = 3.74W \; para \; el \; regulador \; que \; alimenta \; el \; servo$$

Según la hoja de datos del componente, el valor Rθ(JA) para el encapsulado TO-220 (más común entre los LM317 comerciales) es de 23.5 °C/W. Este valor se denomina "Junction-to-ambient thermal resistance" y es una medida de la capacidad de un dispositivo semiconductor para disipar calor en el ambiente circundante sin el uso de disipadores de calor adicionales. Se mide en grados Celsius por watt (°C/W) y representa el aumento de temperatura de la unión del semiconductor (generalmente la parte más caliente del dispositivo, donde se

generan los portadores de carga) por cada watt de potencia eléctrica disipada. Esto indica que, disipando 3.5W y 3.74W, los semiconductores se calentarán

$$3.5W * 23.5° \frac{C}{W} = 82.25°C$$

$$3.74W * 23.5° \frac{C}{W} = 87.89°C$$

respectivamente por encima de la temperatura ambiente.

Para solucionar las altas temperaturas, se utilizará un disipador térmico unido a cada uno de los reguladores. Para ver cómo afecta este cambio a la temperatura del componente, necesitamos calcular la resistencia térmica del total del sistema. Suponiendo que:

- La resistencia térmica del disipador de calor (Rθ(CA)): 10 °C/W.
- La resistencia térmica de la interfaz con pasta térmica (Rθ(interfaz)): 0.5 °C/W.

Entonces, la resistencia térmica efectiva de la unión al ambiente (Rθ(JA) efectiva) con el disipador de calor y la pasta térmica aplicados sería:

$$R\theta(JA)_{efectiva} = \frac{1}{\frac{1}{R\theta(JA)} + \frac{1}{R\theta(CA) + R\theta(interfaz)}} = \frac{1}{\frac{1}{23.5} + \frac{1}{10.5}} = 7.25\ °C/W$$

Por lo que cada uno de los reguladores lineales se calentarán $3.5W * 7.25\ °C/W = 25.37°C$ y $3.74W * 7.25\ °C/W = 27.11°C$ por encima de la temperatura ambiente. Estos valores son bajos respecto al rango de temperaturas admisibles manejables por los semiconductores dado que la temperatura máxima de junta de los mismos es de 150°C.

4.3. Brazo con dos grados de libertad

En las etapas iniciales del desarrollo del proyecto, se prestó una meticulosa atención al diseño y construcción del que proporciona dos grados de libertad y actúa como una plataforma móvil para la cámara del ESP32-CAM. La estructura de esta torreta fue fabricada mediante impresión 3D, utilizando plástico como material principal debido a su adecuada relación entre peso, resistencia y precio. El archivo de impresión puede encontrarse en el siguiente link[5].

Posterior a la impresión, se procedió con la integración de los servomotores, los cuales permiten los movimientos de "pan" y "tilt" (rotación horizontal y vertical, respectivamente) de la torreta.

[5] Link al archivo 3D de la torreta: https://www.thingiverse.com/thing:2038433/files

Una vez colocados, para poder mover la torreta correctamente, fue necesario determinar ciertos parámetros de los servomotores SG90 como, por ejemplo, el rango de ancho de pulso, el ángulo de rotación y la frecuencia a la que trabaja. Todos estos datos fueron extraídos de la hoja de datos[6].

A lo largo del proceso de desarrollo, se identificó una limitación física respecto al rango de movimiento vertical de la torreta debido al diseño de la impresión. Esto derivó en la necesidad de implementar ajustes en el software de control para no romper la estructura. Para resolver este inconveniente, se desarrolló un código específico destinado a controlar el máximo ángulo de rotación vertical del servomotor.

Esta implementación reveló que el movimiento vertical de la torreta tendrá un rango operativo de 0 a 130 grados, estableciendo una limitación para los movimientos en este eje.

A continuación, se observan imágenes del brazo impreso y montado:

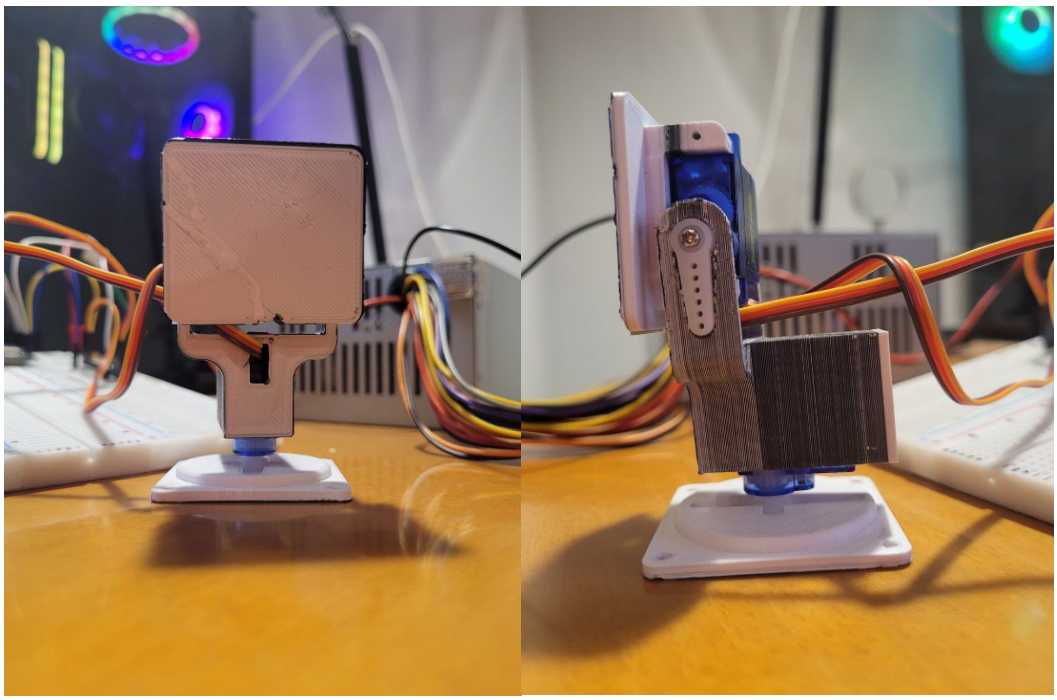


Figura 4 – Imágenes del brazo impreso en 3D montado

4.4. ESP32-CAM

El ESP-32 CAM es un MCU de 32 bits con dos cores de bajo consumo, que trae integrada la cámara OV2640. La funcionalidad de este bloque es fundamental para el desarrollo de este proyecto dado que es el encargado de la recolección y el procesamiento de imágenes para determinar una tupla (x, y) con la posición en el plano en la cual se encuentra el objeto siendo reconocido, tal como se puede ver en la siguiente figura.

[6] Link a las datasheets del servomotor: https://datasheetspdf.com/pdf-file/791970/TowerPro/SG90/1

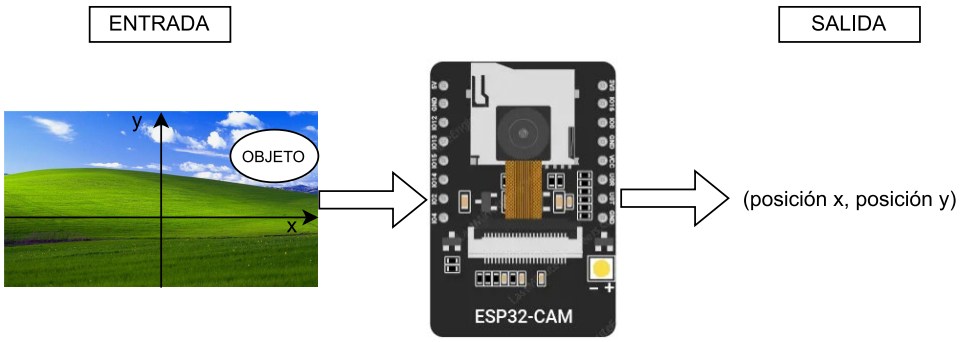


Figura 5 - Entradas y salidas del bloque ESP32-CAM

Los componentes de hardware necesarios dicho bloque son: una ESP32-CAM y un programador externo para la misma.

4.5. EDU CIAA y Botones

El bloque de la EDU CIAA será la encargada de recibir la posición x, y calculado por el bloque de la ESP32-CAM y, en base a esta, determinar el ángulo que deberá mover el brazo robótico (el servo) para hacer efectivo el seguimiento del objeto a detectar. A su vez, tendrá conectados tres botones: START, STOP y RESET. El primero será el encargado de comenzar la detección del objeto (START), el segundo detendrá la detección y el movimiento (STOP), y el tercero reiniciará el brazo a su posición inicial y detendrá la detección (RESET). Se puede profundizar acerca del comportamiento en la sección “Interfaz de usuario”.

4.6. Comunicación EDU-CIAA – ESP32 CAM

Dado que, tanto la ESP como la CIAA, se encuentran en una misma estructura, se decidió optar por el protocolo de comunicación UART. Este brinda una comunicación rápida y muy sencilla con una mínima implementación de cables, lo que implica simplicidad en el circuito impreso final. Además, el protocolo UART es ampliamente utilizado en sistemas de adquisición de datos y ofrece una comunicación sincrónica universal que puede ser fácilmente implementada en ambas placas.

La elección del protocolo se llevó a cabo analizando ventajas y desventajas de distintos tipos de comunicación cableadas: SPI, I^2C y UART. Con respecto al primero, si bien ofrece una alta velocidad de transmisión, requiere de varios cables para su implementación, lo que, aparte de incrementar la complejidad del diseño final, también aumentaría la probabilidad de errores de la conexión. Por el lado de la comunicación con I^2C, esta aporta confiabilidad a la comunicación (ya que incluye detección de errores incorporada), pero trae aparejadas algunas limitaciones con respecto a la velocidad de transmisión de datos. Estos dos protocolos

mencionados anteriormente implican complejidad extra a la hora de la configuración y programación, obstáculo que no tendría sentido afrontar teniendo en cuenta que las ventajas que brindan por sobre la comunicación UART no significaban un gran aporte a la versión final del proyecto.

4.6.1. Compatibilidad de comunicación

Se analizó la compatibilidad de la transmisión de datos entre dispositivos electrónicos, pudiéndose observar los siguientes aspectos:

El ESP32 cuenta con 3 controladores UART ($UART_0$, $UART_1$ y $UART_2$), cada cual tiene un baudRate programable, pudiéndose configurar 5/6/7/8 bits de datos y 1/1.5/2 bits de STOP. Cuentan con soporte de bit de paridad y 1kByte de memoria RAM compartida para los buffers de recepción y transmisión.

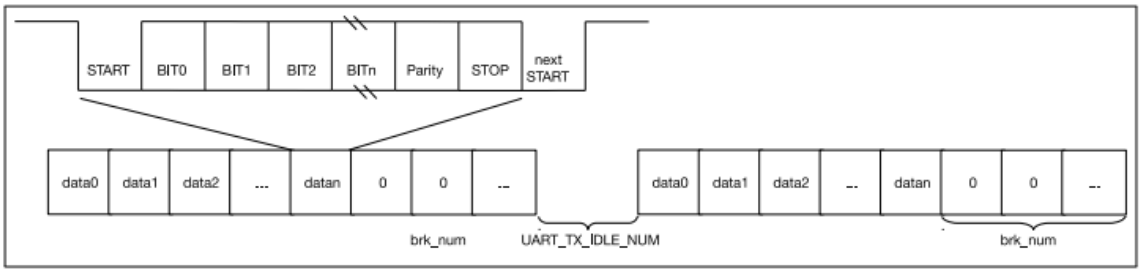


Figura 6 – Secuencia de datos enviada por UART

La configuración de su baudRate se realiza utilizando un timer de alta resolución denominado APB_CLK el cual tiene una frecuencia de 80MHz y una desviación de menos de ±10 ppm, esta fuente es dividida por un registro denominado UART_CLKDIV_REG el cual está conformado por una parte de entera y una parte decimal, permitiendo que la selección de su valor para elegir un baudRate deseado sea lo más precisa posible.

Con respecto al Cortex ARM4 de la EDU-CIAA, cuenta con 4 controladores UART ($UART_0$, $UART_1$, $UART_2$ y $UART_3$), cada cual con bloque generador de baudRate programable, siendo la $UART_1$ la única que no puede ser configurada de forma sincrónica. Se pueden configurar 5/6/7/8 bits de datos, 1/1.5/2 bits de STOP y se puede elegir si tener bit de paridad.

La configuración de su baudRate se realiza utilizando el timer APB_CLOCK (Trabajando a 204MHz) el cual es pre-escalado por un registro denominado Divisor Latch de 16 bits (DLM y DLL); Además puede pre-escalarse por un número fraccionario usando el USART Fractional Divider Register, el cual está compuesto por 4 bits de divisor y 4 bits de numerador, obteniendo un baudRate deseado con alta precisión según la siguiente fórmula:

$$Un\_UCLK = \frac{BASE\_UARTn\_CLK}{2 \times (256 \times DLM + DLL) \times \left(1 + \frac{DIVADDVAL}{MULVAL}\right)}$$

Figura7 – Cálculo de baudRate EDU-CIAA

Además, se validaron por hoja de datos que los niveles de tensión utilizados para las señales lógicas transmitidas y recibidas por cada MCU sean compatibles.

V_{IH}	HIGH-level input voltage			0.7 × $V_{DD(IO)}$	-	5.5	V
V_{IL}	LOW-level input voltage			–0.5	-	0.3 × $V_{DD(IO)}$	V

Symbol	Parameter	Conditions		Min	Typ[1]	Max	Unit
V_{OH}	HIGH-level output voltage	I_{OH} = –8 mA		$V_{DD(IO)}$ – 0.4	-	-	V
V_{OL}	LOW-level output voltage	I_{OL} = 8 mA		-	-	0.4	V

Figura 8 – Valores de Tensión de señales lógicas EDU-CIAA

V_{IH}	High-level input voltage	0.75×VDD[1]	—	VDD[1]+0.3	V
V_{IL}	Low-level input voltage	–0.3	—	0.25×VDD[1]	V
V_{OH}	High-level output voltage	0.8×VDD[1]	—	—	V
V_{OL}	Low-level output voltage	—	—	0.1×VDD[1]	V

Figura 9 – Valores de Tensión de señales lógicas ESP32

Al ser en ambos MCU el valor de V_{DD} = 3.3V, se comprueban los siguientes parámetros bajo condiciones normales de Temperatura [-40ºC, 105 Cº]:

- EDU-CIAA
  - Tensión de entrada valor lógico ALTO ϵ [2.31V, 5.5V]
  - Tensión de entrada valor lógico BAJO ϵ [-0.5, 1V]
  - Tensión de salida valor lógico ALTO > 1.32V
  - Tensión de salida valor lógico BAJO < 0.4V
- ESP32
  - Tensión de entrada valor lógico ALTO ϵ [2.475V, 3.3V]
  - Tensión de entrada valor lógico BAJO ϵ [-0.4, 0.825V]
  - Tensión de salida valor lógico ALTO > 2.64V
  - Tensión de salida valor lógico BAJO < 0.33

Se valida la compatibilidad lógica eléctrica debido a que:

$$V_{OH\_ESP32} > V_{IH\_EDUCIAA}^{Min}$$
$$V_{OL\_ESP32} < V_{IL\_EDUCIAA}^{Max}$$
$$V_{OL\_EDUCIAA} < V_{IL\_ESP32}^{Max}$$

Y aunque por hoja de datos no se pueda validar que:

$$V_{OH\_EDUCIAA} > V_{IH\_ESP32}^{Min}$$

Cuando se realizaron pruebas empíricas de la comunicación entre módulos se verificó el correcto funcionamiento.

4.7. Diseño de la PCB

Antes de comenzar con el diseño de la PCB se realizó el modelo esquemático del proyecto. El mismo puede observarse en el anexo B. Para poder representar todas las partes, se crearon nuevos componentes dentro del esquemático de manera manual como son los botones y la entrada Jack.

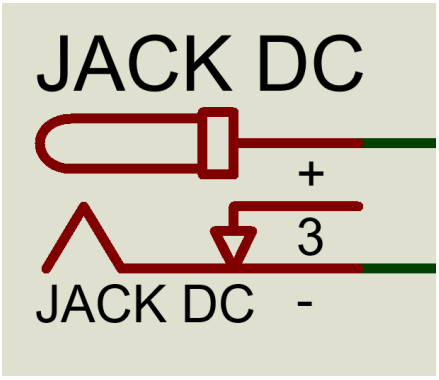


Figura 10 – Esquemático del conector JACK de entrada

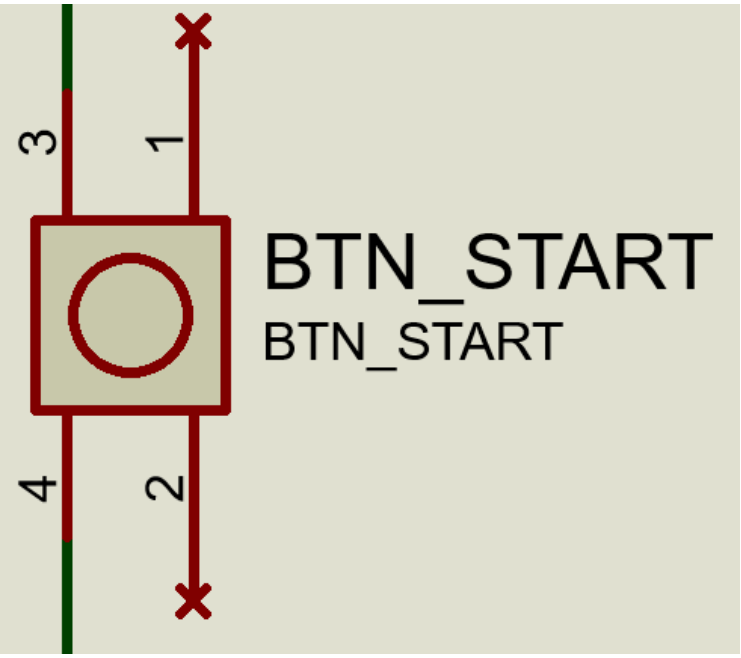


Figura 11 – Esquemático del botón STOP

Una vez definidos todos los componentes y creados, se procedió a la interconexion de los mismos. El hecho que el poncho pueda ser alimentado por 2 tensiones diferentes (5V y 12V) implico un gran desafio al momento del diseño de las conexiones. Hay 2 escenarios posibles

I. La tension de entrada son 5V, por lo que se debe alimentar la EDU-CIAA, el ESP-32 y los servomotores desde la entrada Jack.

II. La tension de entrada son 12V, por lo que se deben habilitar las 2 etapas de regulacion de tensión. Con 1 se debe alimentar SOLO las placas EDU-CIAA y ESP32, y con la otra SOLO los servomotores

Esta variacion entre 2 modos de funcionamiento llevo al agregado de 2 nuevas tiras de 2 y 3 pines.

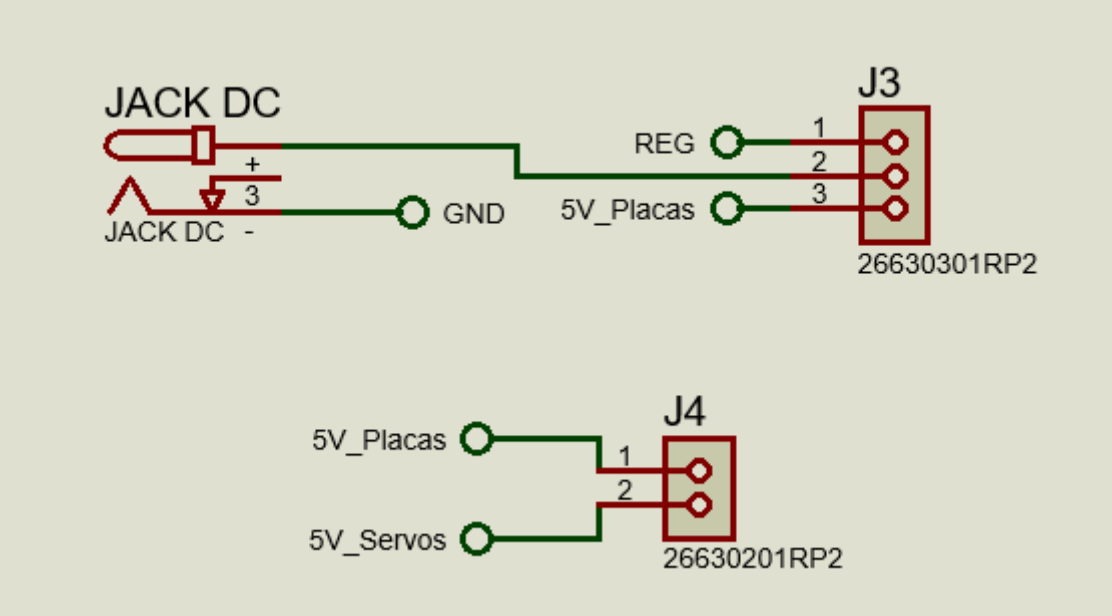


Figura 12 – Conectores J3, J4 y la entrada jack y su interconección con las distintas partes del sistema

La lógica es la siguiente:

- Si la tension de entrada es 5V, se debe conectar un jumper entre el pin 2 y 3 del conector J3 para que el mismo alimente las placas. Además, se debe agregar un jumper entre los pines 1 y 2 del conector J4 para que tambien se alimenten los servomotores.
- Si la tension de entrada es 12V, se debe conectar un jumper entre los pines 1 y 2 del conector J3 para habilitar las etapas de regulación. En cuanto al conector J4, es sumamente importante que NO tenga un jumper conectado entre sus pines dado que se cortocircuitaria la salida de las etapas de regulacion.

Por otra parte, se definieron los pines a utilizar para conectar el poncho con la EDU-CIAA. Se agregaron pines extras que no se utilizan para una mayor firmeza y sujeción al momento de la conexión de ambas placas.

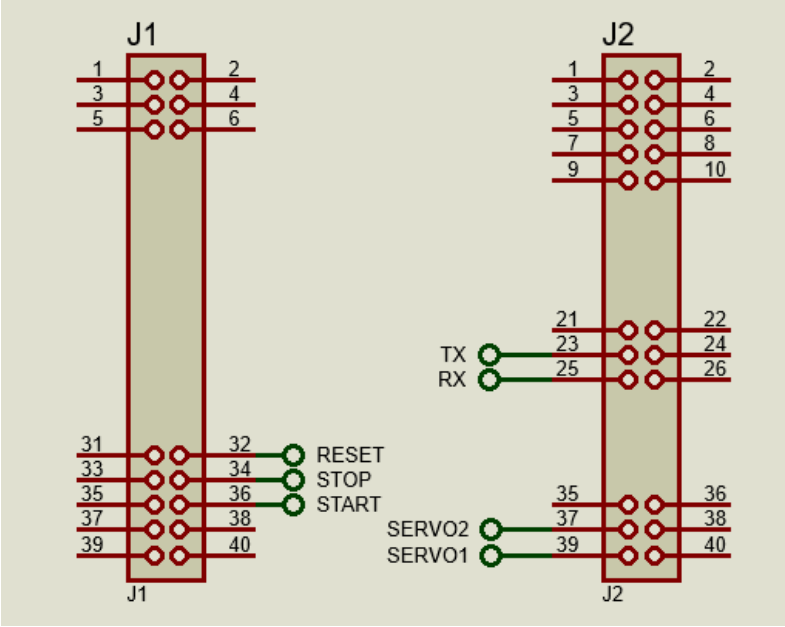


Figura 13 – Tiras de pines J1 y J2 que se conenectaran en la EDU-CIAA

Una vez definidas estas cuestiones, se inició con la etapa de diseño de la PCB. En primer lugar, se crearon los footprints de aquellos componentes creados manualmente, como los botones y la entrada jack

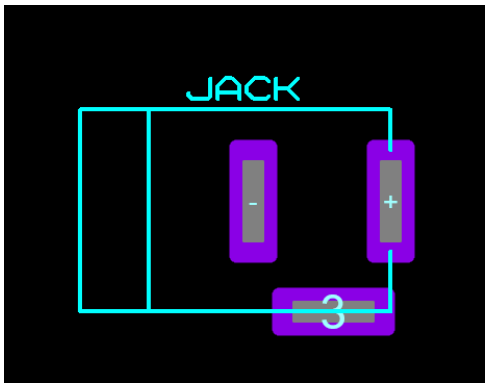


Figura 14 – Footprint hecha a medida para el conector jack hembra

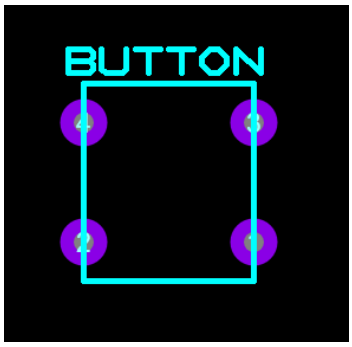


Figura 15 – Footprint hacha a medida para los botones

Una vez finalizada la creación, se mapearon todos los componentes del esquematico con sus respectivos footprints y se ubicaron los componentes en la placa. Para ello, se tuvo en cuenta:

- Las dimensiones de la CIAA para que el poncho “encastre” sobre la placa
- Los reguladores lineales deben estar en uno de los bordes de la placa para tener mayor circulacion de aire y para que el disipador no estorbe a otros componentes
- Se debe definir una zona de regulacion y otra zona donde el usuario interactuará con los botones
- La base del brazo debe ir atornillada al poncho por lo que se deben realizar los agujeros correspondeintes
- La orientacion del brazo define en que lugar se encontraran las conecciones para los servos y para el ESP32 y las distancias de los mismos para evitar enredos de cables
- El jack para la conexión de entrada debe estar en uno de los bordes de la placa
- Al alimentar a la CIAA desde el poncho, se debe realizar un corte en la zona superior izquierda del poncho para que el conector de la alimentacion entre correctamente.

A medida que se fueron colocando los componentes, se fueron definiendo los caminos de las pistas para conectarlos. En 2 casos fue necesario crear una conexión del lado que no tiene cobre para poder conectar ciertos componentes a tierra y para alimentar la CIAA. Esto se resolverá en la etapa de construcción fisica soldando 2 cables.

Por el tipo de técnica de impresión que utilizaremos, las pistas deben tener un ancho mayor a 0.7mm, y los agujeros deben tener un diametro mínimo de 0.7mm y una corona de 1.88mm, por lo que se actualizaroin todas las pistas y los footprints para cumplir con esta condición. El diseño final del PCB puede observarse en el Anexo C o, para mayor detalle, en el archivo “PCB_Proyecto.pdspej” adjunto con esta entrega.

Una vez finalizado el desarrollo de la PCB, se procedió a cargar y ajustar los modelos 3D de los objetos. Especialmente para los botones, y para el conector jack.

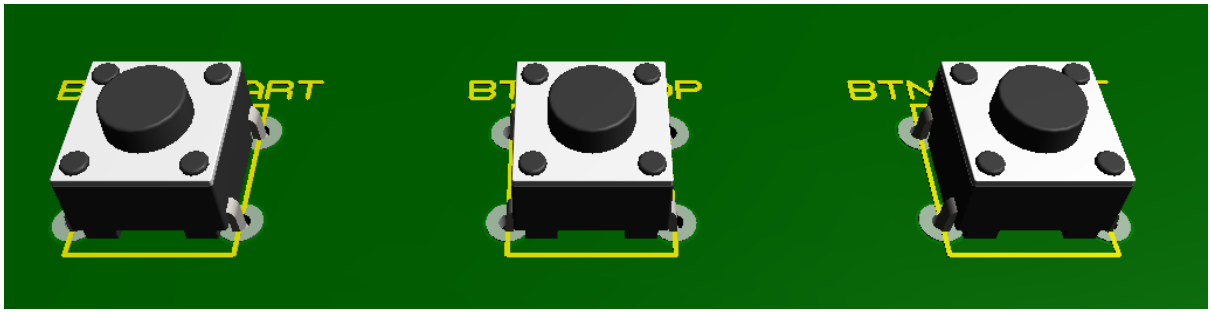


Figura 16 – Modelo 3D de los botones y su encastre exacto en el footprint diseñado

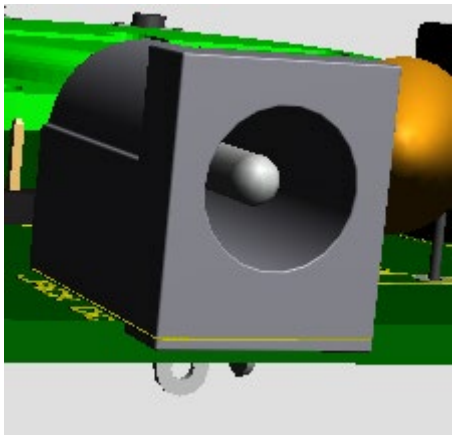


Figura 17 – modelo 3D de la entrada jack y su encastre en la PCB

Ademas, se cargo el modelo 3d de la base del brazo para ser atornillada a la placa. Al momento fijar este componente se utilizaran aislantes para que ningun metal entre en contacto con la parte de cobre del poncho evitando posibles problemas. Los agujeros en la placa fueron medidos de manera precisa para que coincidan con los agujeros de la base. A continuacion se observa la pieza sobre la placa:

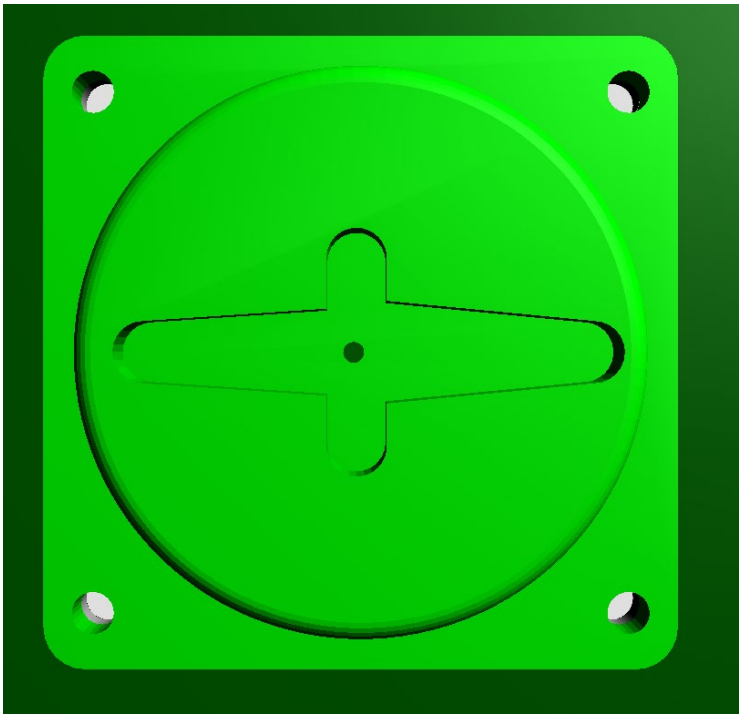


Figura 18 – Base del brazo robótico en 3D sobre la PCB con los agujeros para fijarla a la placa

La vista final 3D de la placa puede observarse en el Anexo D o en el archivo .pdspej adjunto con esta entrega.

5. Diseño de firmware, simulación y depuración

5.1. Pruebas de la comunicación UART

Como se mencionó anteriormente, el firmware que se utilizó para la programación de la CIAA fue la sAPI, la cual brinda variedad de herramientas para el desarrollo, entre ellas dos librerías que son de especial ayuda para el programa de prueba: *sapi_servo* y *sapi_uart*, incluidas en la librería *sapi*. Gracias a las mismas es que podemos abstraernos del funcionamiento de ambos componentes con unas pocas órdenes, en caso del servo basta con inicializarlo y enviarle un número de tipo entero para hacerlo rotar, y por parte de la comunicación UART solo es necesario indicar la tasa de transferencia en baudios y ya se puede enviar y recibir datos, tal como se puede ver en la figura 4.

```
#include "sapi.h"
#define SERVO_N   SERVO0  /*cada número de servo se
                          corresponde con un pin específico*/
void main(void)
{
    boardConfig(); /* Funcion provista por la sAPI
                      para inicializar la placa*/

    /*se configura una tasa de transferencia para
    la comunicación USB y otra para la cableada*/
    uartConfig(UART_USB, 9600);
    uartConfig(UART_232, 115200);

    /*se habilita el servo*/
    servoConfig(0, SERVO_ENABLE);
    servoConfig(SERVO_N, SERVO_ENABLE_OUTPUT);
    servoWrite(SERVO_N, servoAngle);
}
```

Figura 19 - Configuración de los periféricos en la CIAA

Cabe mencionar que la sAPI cuenta con un código de ejemplo de control de un servo mediante estas instrucciones, que fue probado en la CIAA y el servo con los que cuenta este equipo de desarrollo y no se ha observado ningún fallo en el funcionamiento del programa.

Por parte de la ESP, para poder realizar la comunicación UART basta con importar la librería *HardwareSerial*, incluida en la librería *Arduino*, que permite el uso de las instrucciones para establecer la tasa de transferencia y enviar datos mediante los puertos serial. En la siguiente figura se enseña un programa básico con el uso de estas instrucciones.

```arduino
#include <Arduino.h>

void setup()
{
  Serial.begin(115200); // Inicializa la comunicación serial
}

void loop()
{
  Serial.write('A'); // Envía el carácter 'A' a través de UART
}
```

Figura 20 - Configuración de UART ESP32

La transmisión y recepción exitosa de información entre ambas placas demuestra la practicidad de este protocolo y la sencillez a nivel código para enviar y recibir datos. En la sección anexos podrán consultarse el código de ambas placas, así como también un video demostrativo de la correcta comunicación por UART.

5.1.1. Movimiento del servo

En un principio, el servomotor producía movimientos muy lentos dado que se alimentaba con la EDU CIAA. Como se mencionó anteriormente, este dispositivo puede requerir picos de corriente de hasta 680mA y la salida de la placa está limitada a 300mA. Para solucionar este problema, se alimentó de manera externa el servomotor y se comprobó la correcta funcionalidad del servo al recibir las señales de control desde la EDU-CIAA.

Una vez solucionado esto, se procedió a la integración de este programa con el modelo de reconocimiento de objetos, alojado en la ESP-CAM, la cual fue exitosa. Mediante la conexión por UART de las placas, la EDU-CIAA recibía correctamente las coordenadas del objeto proporcionadas por la ESP32-CAM y las traducía a grados para enviarle al servo el comando de posición correspondiente.

En la sección de "Anexos" se encuentra un enlace al video de funcionamiento del programa previamente enseñado corroborando la correcta interacción entre las placas y el servo en movimiento.

5.1.2. Lectura por interrupciones

Luego de lograr que el servo se moviera en función de lo que recibía la EDU-CIAA a partir de la ESP32-CAM, el trabajo fue enfocado en lograr que la comunicación se realizara mediante interrupciones del periférico UART, tal como se había planteado en el pasado

informe. Es así que se procedió al estudio de las librerías que la sAPI proveía para realizar esta tarea.

Gracias al programa de ejemplo que esta biblioteca brindaba, observamos que existía una manera sencilla de habilitar las interrupciones de recepción y transmisión del periférico UART y asignarle una función como manejador de las mismas.

```
    // Seteo un callback al evento de recepcion y habilito su interrupcion
    uartCallbackSet(UART_232, UART_RECEIVE, onRx, NULL);
    // Habilito todas las interrupciones de UART_USB
    uartInterrupt(UART_232, true);
```

Figura 21 – recepción mediante interrupciones

En la figura mostrada previamente se pueden observar las instrucciones necesarias realizar lo anteriormente mencionado, cabe destacar que, en este caso, la rutina de interrupción es la función denominada "onRx".

En cuanto a la recepción, se precisó de una estructura de datos eficiente y confiable para almacenar los datos leídos, es así que se utilizó un buffer circular, mediante una cola FIFO de tamaño fijo y dos índices, el de lectura y el de escritura, con tamaño suficiente para almacenar 3 mensajes a la vez. Todo esto fue implementado en una librería llamada "Buffer", la cual realizamos para asegurar un manejo ordenado de la estructura. A continuación, podrá observarse una figura que muestra las instrucciones realizadas en la rutina de interrupción.

```
void onRx(void *noUsado) {
    char c = uartRxRead(UART_232);
    if (c == '\n') {
        stringReceived = 1; // Flag de dato recibido por UART
    }
    bufferPush(c);
}
```

Figura 22 – Rutina de manejo de interrupciones

Como se puede observar en el código previo, cada vez que se recibe un dato por UART, éste es almacenado en el buffer y una vez que se recibe un carácter "\n" se establece en 1 una flag que le indica al bucle principal del programa que es momento de procesar estos datos.

5.2. Interacción del usuario

Como se mencionó anteriormente, el sistema podrá ser controlado mediante tres botones principales: START, STOP, y RESET. En el siguiente diagrama de flujo (Figura 7) se

detalla el funcionamiento de estos botones y como deberá responder el software al ser presionados:

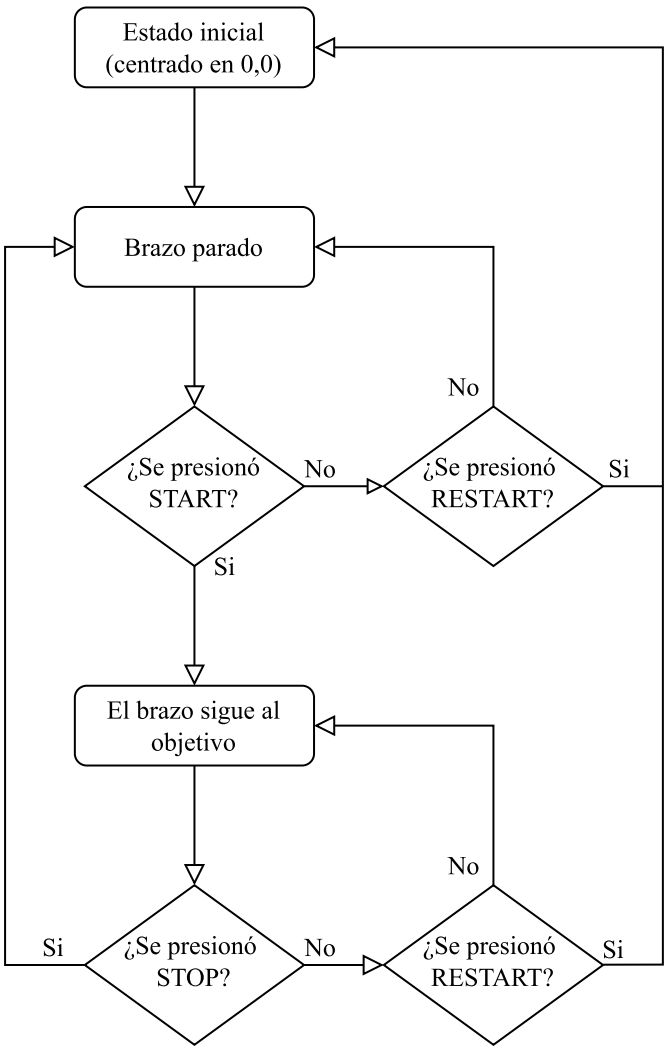


Figura 23 - Diagrama de flujo del sistema al interactuar con los botones

5.3. Reconocimiento de Objetos

Para la implementación del algoritmo capaz de reconocer un objeto a partir de una imagen captada por la ESP-32 CAM se investigaron varios trabajos previos realizados en esta área. Para seleccionar el más adecuado se tuvieron en cuenta parámetros como: Almacenamiento en memoria del modelo de detección, tiempo de inferencia de la detección, precisión y exactitud al devolver coordenadas espaciales de la detección.

Luego de este análisis, se optó por elegir el modelo de detección de objetos FOMO (MobileNet V2 0.1), el cual es un modelo de aprendizaje profundo diseñado para implementar detección de objetos en dispositivos con recursos limitados, como los MCU. Según sus creadores, permite el seguimiento de objetos en tiempo real usando hasta 30 veces menos poder de procesamiento y memoria que MobileNetSSD o YoloV5.

Este modelo combina un extractor de propiedades de imágenes MobileNet V2 con un clasificador completamente convolusional para enmarcar la “Detección de Objetos” como una “Clasificación de objetos” dentro de una grilla, permitiendo en caso de tener varias unidades de procesamiento la clasificación de imágenes de forma paralela.

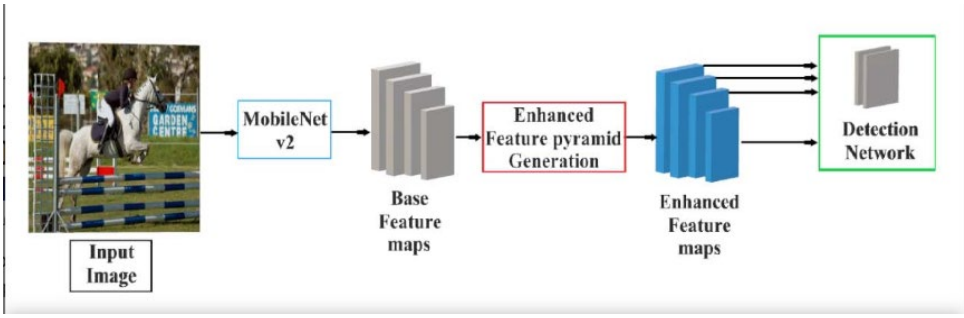


Figura 24 - Diagrama de Bloques FOMO

Para poder utilizar este modelo para que detecte algún objeto que nosotros queramos, tuvimos que entrenarlo con un conjunto de datos personalizado. Este proceso se conoce como “Etiquetado” de imágenes, en donde cargamos un conjunto de imágenes y le asignamos de forma manual a cada una las etiquetas correspondientes a los objetos a detectar presentes en cada imagen.

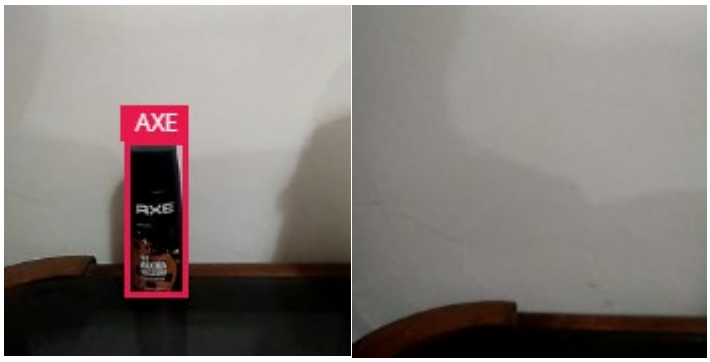


Figura 25 - Imágenes etiquetadas.

En nuestro caso, decidimos que el objeto a detectar sea un desodorante, para su simplicidad y distinción con un fondo claro. Gracias a la plataforma EdgeImpulse en su versión gratuita, fuimos capaces de entrenar este modelo con nuestro dataset en la nube. Una vez entrenado pudimos exportarlo como una librería Arduino para su conveniencia de uso.

Al probar su ejecución en un solo hilo dentro de la ESP32CAM, pudimos observar que el tiempo de inferencia es el esperado (Alrededor de 500 ms) y que su precisión es consistente (Aunque se podría mejorar para futuras entregas).

```
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
Predictions (DSP: 7 ms., Classification: 543 ms., Anomaly: 0 ms.):
```

Figura 26 - Cálculo de tiempo de inferencia de detección

Además, este modelo nos permite enviar por el puerto serie las coordenadas del objeto detectado:

```
Predictions (DSP:
(x: 28, y: 68)
Predictions (DSP:
(x: 28, y: 60)
(x: 28, y: 80)
Predictions (DSP:
(x: 28, y: 76)
Predictions (DSP:
(x: 36, y: 68)
Predictions (DSP:
(x: 36, y: 72)
Predictions (DSP:
(x: 36, y: 72)
Predictions (DSP:
(x: 44, y: 68)
```

Figura 27 - Tupla enviada por puerto serie al detectar el objeto

Las cuáles serán procesadas posteriormente por la EDU-CIAA. Es por esto que se optó porque cuando se detecte un objeto se envíen sus coordenadas y si no se detecta ningún objeto no se envíe nada.

5.4. Arquitectura productor-consumidor

Una vez decidido que la comunicación se realiza a través de UART entre los dos MCU, se decidió utilizar una arquitectura productor-consumidor, en la cual el ESP32-Cam produce coordenadas del objeto a trackear (Productor), y la EDU-CIAA las recibe y procesa en el orden en el cual llegan (Consumidor) según se ve en el diagrama de la Figura 15:

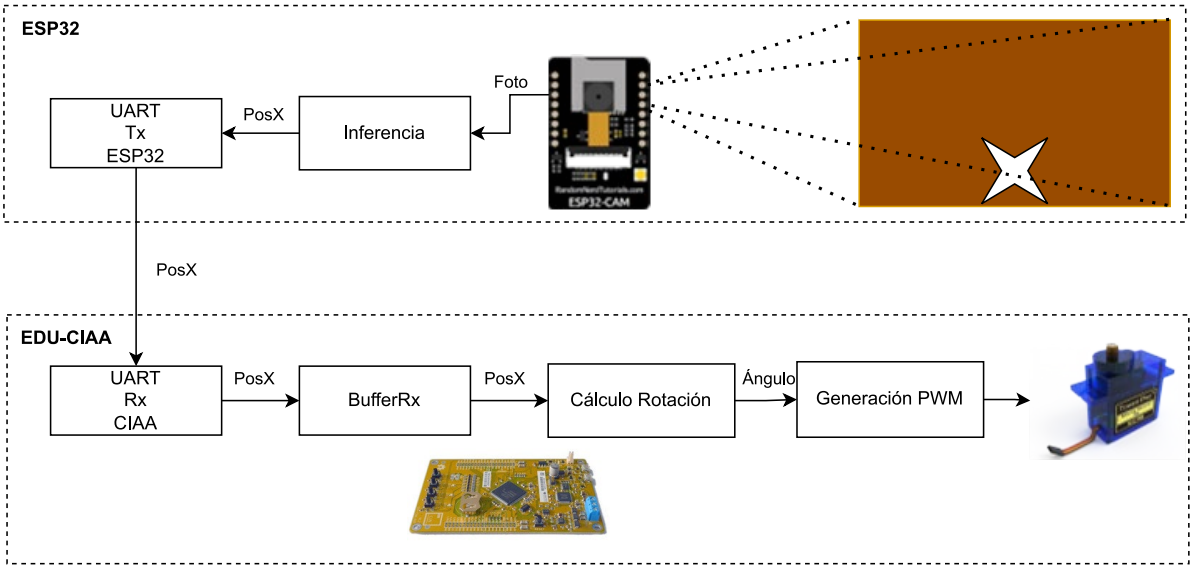


Figura 28 – Diagrama de comunicación entre MCUs

Al ser el tiempo de inferencia alrededor de 260ms, podemos asegurar que el buffer de recepción nunca se desbordará ya que la velocidad de procesamiento de las coordenadas para la generación de señales PWM es mucho mayor.

La recepción de coordenadas se realizará dentro de una interrupción para que sean almacenadas en el buffer lo antes posible, tal como se describe en el punto 5.4.1.

El cálculo de la rotación a partir de la coordenada de la imagen se describe en el inciso próximo.

5.5. Cálculo de ángulo de rotación

Para poder asegurar un trackeo óptimo, se asumirá el siguiente escenario acondicionado para la detección del objeto:

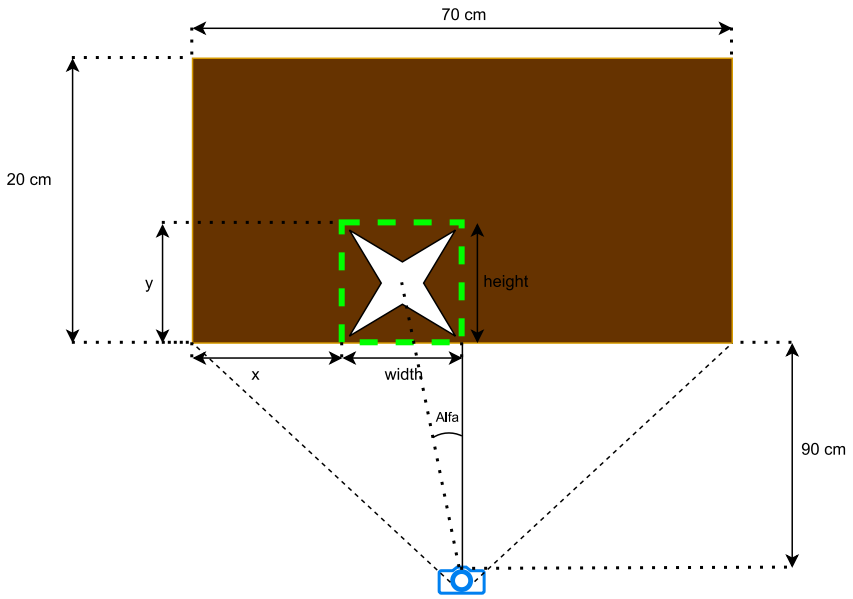


Figura 29 – Escenario controlado de detección

Se pueden observar los siguientes aspectos:

1. El fondo debe ser obscuro
2. El objeto a detectar será una estrella de cuatro puntas blanca
3. La cámara se encontrará a 30cm de distancia del objeto a detectar
4. La cámara saca capturas de 96x96, por ende, todas las medidas con respecto a la inferencia se realizan utilizando estas medidas. Es a partir de esta resolución y de haber medido el largo en cm de detección que podemos observar la relación de 70cm/96pixeles ➔ 0.73cm/pixel.

Es a partir de esta información que se pueden calcular los siguientes datos:

- La posición del objeto detectado: Como lo que nos interesa es el centro del objeto, lo calcularemos como:

$$X = x + \frac{width}{2} \wedge Y = y - \frac{height}{2}$$

  Este valor, según valores experimentales, se encuentra entre [0,64] por lo cual tomaremos como medio el valor 64/2 = 32.

- El ángulo de rotación: Debido a que la distancia del sensor al objeto y el rango de visión los asumimos constantes, el ángulo de rotación será:

$$\alpha = \arctan\left(\frac{|X - 32|px * \frac{0.73cm}{px}}{30cm}\right) = \arctan\left(|X - 32| * 0.024\right)$$

Este será el ángulo de desplazamiento con respecto al ángulo actual, teniendo en cuenta que $\alpha = \alpha$ si $X \geq 32 \wedge \alpha = -\alpha$ si $X < 32$.

Ya con estos datos se puede generar la señal PWM usando la librería que recibe un parámetro en ángulos, siguiendo el siguiente algoritmo y asumiendo un ángulo de rotación inicial de 90°:

```
Alfa = CalcularAlfa(PosX)
Si (PosX < 32)
        Alfa = -Alfa
Si ( ((AnguloActual + Alfa) no es menor a 0) y ( (AnguloActual + Alfa) no es mayor a 180) )
        AnguloActual +=Alfa
GirarServo(AnguloActual)
```

Pseudocódigo 1: Generación de PWM Invertido por SOFTWARE.

5.6. Complejidad del modelo de detección

Para mejorar los tiempos de inferencia, decidimos disminuir la cantidad de épocas en las que se entrenará al modelo, así como tomamos un conjunto de imágenes más estandarizadas ya que fueron tomadas con la misma ESP-CAM que se encargará de la detección, dentro del entorno controlado descripto en la sección anterior, siendo el objeto para detectar simple y contrastante con el fondo, también reduciendo la cantidad de muestras.

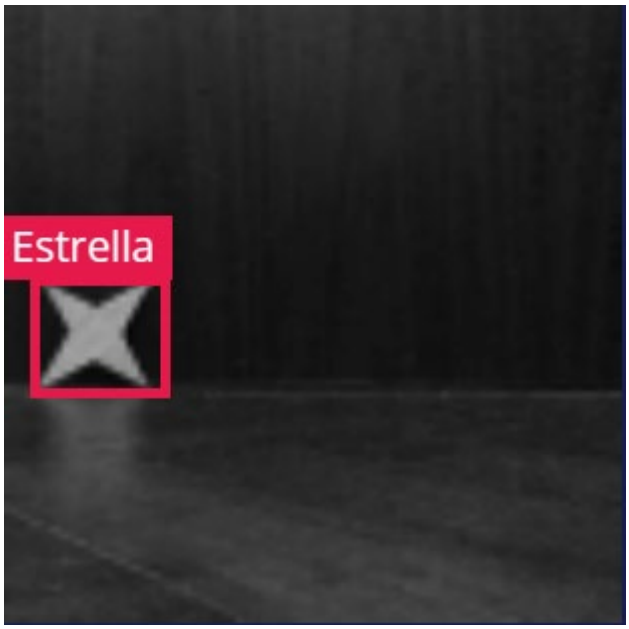


Figura 30 – Ejemplo de muestra utilizada para entrenar el modelo

Como resultado de estas medidas, logramos que el tiempo de inferencia de nuestro modelo se reduzca de 500ms a 200ms. Esta decisión tiene como consecuencia una disminución

de la precisión en ambientes distintos al ambiente controlado, esto se debe a que al ser entrenado en pocas épocas el modelo no llega a entender los patrones más complejos del objeto a detectar (Como forma, vértices, detalles, etc) y se queda con las características más simples (Como color, iluminación y tamaño) para la detección, pudiendo generar mayores falsas detecciones. Sin embargo, fue un precio que estuvimos dispuestos a pagar con tal de mejorar la fluidez del movimiento del servo.

6. Ensayos y mediciones

Un video con pruebas realizadas con el brazo y el ESP32-Cam puede observarse en el siguiente link

Al momento de la construcción de la PCB uno de los principales problemas fue que el papel no protegió el cobre del plano de tierra correctamente dejando una parte de este aislada. Para solucionarlo, se soldó un cable para conectar estos 2 sectores.

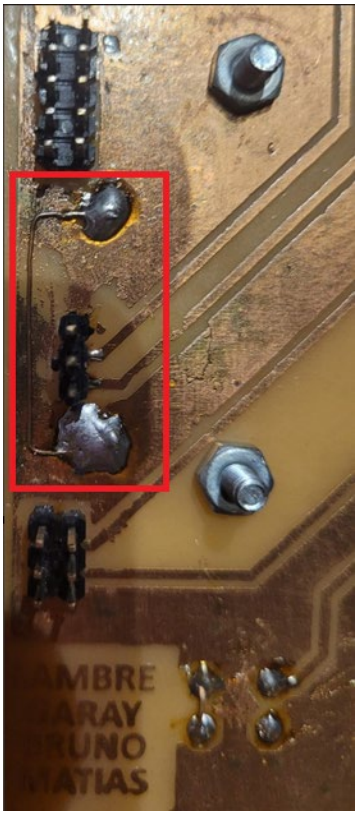


Figura 31 – Medición de tensión suministrada a los servomotores conectando el sistema a 12V y habilitando la etapa de regulación

Se realizaron diversas mediciones para testear el voltaje de alimentación para los distintos componentes. Varias de estas mediciones se observan a continuación:

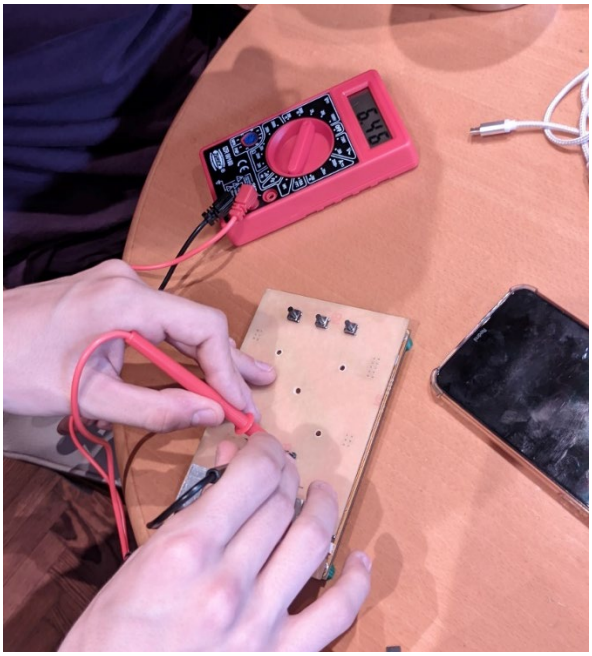


Figura 32 – Medición de tensión suministrada a los servomotores conectando el sistema a 12V y habilitando la etapa de regulación

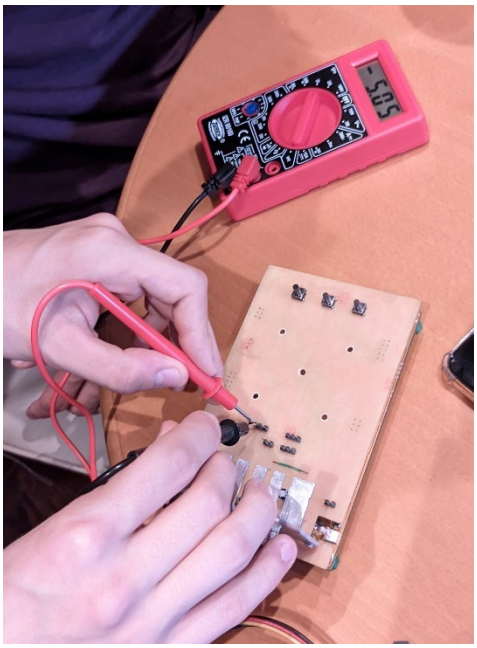


Figura 33 – Medición de tensión suministrada a las placas ESP32 y EDU-CIAA conectando el sistema a 12V y habilitando la etapa de regulación

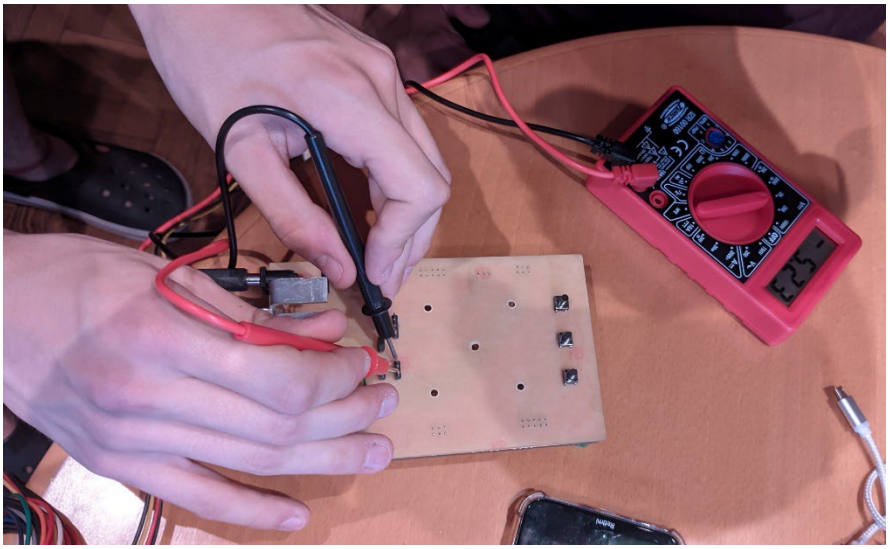


Figura 34 – Medición de tensión suministrada a los servomotores conectando el sistema a 5V sin habilitar la etapa de regulación

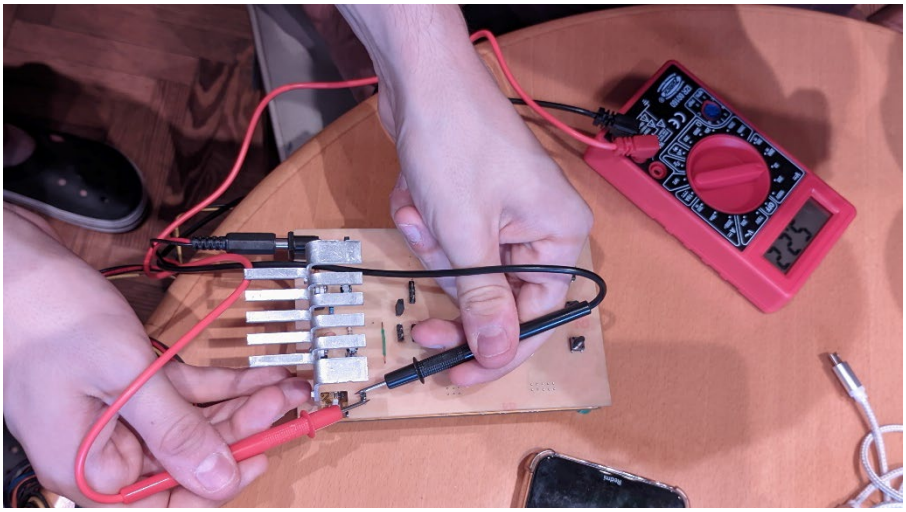


Figura 35 – Medición de tensión suministrada a las placas ESP32 y EDU-CIAA conectando el sistema a 5V sin habilitar la etapa de regulación

En las siguientes imágenes se observa la construcción final del proyecto:

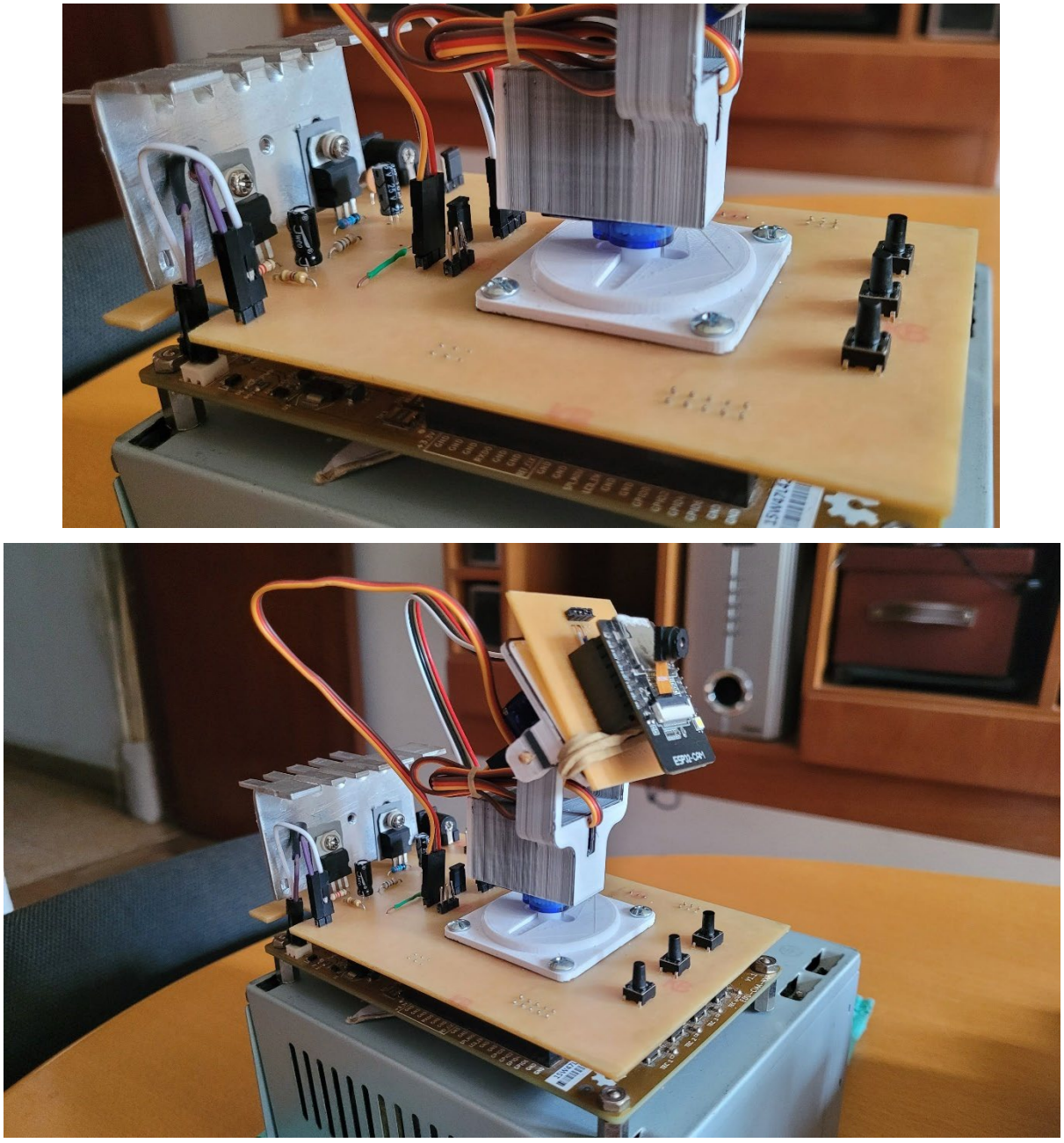


Figura 36 – Proyecto final junto a su fuente de alimentación 220AC – 12V/5V

Por último, podemos observar el sistema en funcionamiento dentro del ambiente controlado el día de la demostración en clase en el siguiente link.

El firmware utilizado en este proyecto se encuentra subido en el gestor de versiones GitHub en el siguiente repositorio público.

7. Conclusiones

En la realización del proyecto logramos cumplir en su totalidad los objetivos primarios establecidos, lo que representó un éxito significativo y la base sobre la cual se construyó todo el trabajo. Estos objetivos, fundamentales para la dirección y propósito del proyecto, fueron alcanzados gracias a una planificación meticulosa y un esfuerzo conjunto del equipo, asegurando así que el núcleo del proyecto se desarrollara eficientemente y de acuerdo con las expectativas. Sin embargo, en cuanto a los objetivos secundarios, enfrentamos diversas dificultades que impidieron su total cumplimiento. Estos se vieron afectados por limitaciones de tiempo, recursos, y desafíos inesperados que surgieron durante la ejecución. A pesar de este contratiempo, el aprendizaje obtenido en el proceso y la adaptabilidad demostrada ante estos obstáculos han sido de gran valor para el equipo, ofreciendo lecciones importantes para futuras iniciativas.

Realizar el proyecto fue una gran oportunidad para ganar experiencia práctica y ampliar nuestros conocimientos respecto a los sistemas embebidos. La aplicación directa de teorías y conceptos aprendidos en clases (principalmente del primer semestre en circuitos digitales) a situaciones reales nos permitió entender mejor su utilidad y alcance, además de identificar áreas donde necesitábamos profundizar nuestro aprendizaje. En conjunto, esta experiencia no solo enriqueció nuestro conocimiento académico, sino que también nos preparó mejor para futuros desafíos profesionales, dotándonos de herramientas y habilidades altamente valoradas en el mundo laboral.

Finalmente, el hecho de trabajar en grupo para el desarrollo del proyecto resultó ser una experiencia enriquecedora y altamente productiva. La diversidad de habilidades y perspectivas de cada miembro del equipo contribuyó a una lluvia de ideas más amplia y a soluciones creativas que difícilmente habrían surgido de manera individual. La colaboración permitió repartir las tareas de acuerdo con las fortalezas de cada persona, optimizando así el tiempo y los recursos disponibles. Esta experiencia no solo facilitó el logro de los objetivos propuestos con éxito, sino que también mejoró nuestras habilidades de comunicación y trabajo en equipo, herramientas valiosas para nuestro desarrollo profesional y personal.

8. Cronograma

SEMANA	4-sep	11-sep	18-sep	25-sep	2-oct	9-oct	16-oct	23-oct	30-oct	6-nov
	1	2	3	4	5	6	7	8	9	10
PROYECTO: ETAPA DE INVESTIGACIÓN										
Investigar Arquitectura ESP32	✔									
Investigación Arquitectura Cortex-M4	✔	✔								
Investigar Funcionamiento CAM-ESP32	✔	✔								
Búsqueda de Componentes		✔	✔							
Investigación TinyML en ESP32		✔	✔	✔	✔	✔				
PROYECTO: ETAPA DE DISEÑO										
Diseño de Servo-Cámara				✔	✔					
Diseño del Circuito Esquemático					✔	✔				
Implementación del Código					✔	✔	✔	✔	✔	
Implementación Comunicación de MCUs					✔	✔				
Diseño del Circuito Impreso (PCB)						✔	✔	✔	✔	
PROYECTO: ETAPA FINAL										
Fabricación										✔
Ensamblaje										
Pruebas de Validación										
Muestra Grupal										
ENTREGAS FORMALES										
Informe Inicial	X									
Informe Avance 1						X				
Informe Avance 2										X
Informe Final										
Hora sem. De clase	4	2	4	4	4	4	4	4	4	4
Total Hs de Clase	4	6	10	14	18	22	26	30	34	38

13-nov	20-nov	27-nov	4-dic	11-dic	18-dic	5-feb	12-feb
11	12	13	14	15	16	17	18
✔							
	✔	✔					
			✔	✔			
				✔			
						X	
4	2	4	4	4	4	0	0
42	44	48	52	56	60	60	60

Tabla 2 - Cronograma de trabajo

9. División de tareas

Tarea	Responsable	Colaborador	Horas	Completada
~~Especificación de Requerimientos~~	~~Bruno, Laureano~~	~~Zeballos, Matías Manuel~~	~~8~~	☑
~~Adquisición de Componentes~~	~~Garay, Francisco~~	~~Lambre, Jerónimo~~	~~4~~	☑
~~Implementar y Entrenar modelo de reconocimiento~~	~~Zeballos, Matías Manuel~~	~~Garay, Francisco~~	~~12~~	☑
~~Algoritmo de movimiento del Servo~~	~~Lambre, Jerónimo~~	~~Garay, Francisco~~	~~12~~	☑
~~Interfaz de Comunicación entre MCUs~~	~~Garay, Francisco~~	~~Bruno, Laureano~~	~~8~~	☑
~~Pre-procesamiento de imágenes~~	~~Zeballos, Matías Manuel~~	~~Lambre, Jerónimo~~	~~2~~	☑
~~Post-Procesamiento de imágenes~~	~~Zeballos, Matías Manuel~~	~~Lambre, Jerónimo~~	~~2~~	☑
~~Servidor local con cámara en tiempo real~~	~~Bruno, Laureano~~	~~Zeballos, Matías Manuel~~	~~10~~	☑
~~Análisis y solución de necesidades eléctricas~~	~~Lambre, Jerónimo~~	~~Garay, Francisco~~	~~8~~	☑
~~Diseño del Esquemático~~	~~Garay, Francisco~~	~~Lambre, Jerónimo~~	~~12~~	☑
~~Diseño del PCB~~	~~Bruno, Laureano~~ ~~Lambre, Jerónimo~~ ~~Garay, Francisco~~ ~~Zeballos, Matías Manuel~~	-	~~12~~	☑
~~Confección del Poncho~~	~~Lambre, Jerónimo~~	~~Bruno, Laureano~~	~~12~~	☑
~~Testeo de Reconocimiento~~	~~Garay, Francisco~~	~~Zeballos, Matías Manuel~~	~~4~~	☑
~~Testeo de Trackeo~~	~~Zeballos, Matías Manuel~~	~~Bruno, Laureano~~	~~8~~	☑
~~Testeo de Interfaz Web~~	~~Bruno, Laureano~~	-	~~4~~	☑
~~Armado de informes de avance~~	~~Bruno, Laureano~~ ~~Lambre, Jerónimo~~ ~~Garay, Francisco~~ ~~Zeballos, Matías Manuel~~	-	~~8~~	☑
~~Armado de Informe final~~	~~Bruno, Laureano~~ ~~Lambre, Jerónimo~~ ~~Garay, Francisco~~ ~~Zeballos, Matías Manuel~~	-	~~12~~	☑

Tabla 3 – Tareas asignadas por integrante

Integrante	Horas invertidas	Horas asignadas	Horas a invertir
Bruno, Laureano	54	54	0
Zeballos, Matías Manuel	60	60	0
Garay, Francisco	60	60	0
Lambre, Jerónimo	64	64	0

Tabla 4 – Horas asignadas por integrante

10. Bibliografía

- FOMO (Faster Objects, More Objects) Guide [portal web]. Disponible: 10/2023. URL: https://edgeimpulse.com/blog/fomo-self-attention
- MobileNetV2 Description [Artículo Académico], Disponible: 10/2023. URL: https://doi.org/10.48550/arXiv.1801.04381
- FOMO Object Detection for Constrained Devices [portal web]. Disponible 9/2023. URL: docs.edgeimpulse.com
- Arduino Forum [portal web]. Disponible: 10/2023). How much power do I need for controlling 5 SG90 microservos?

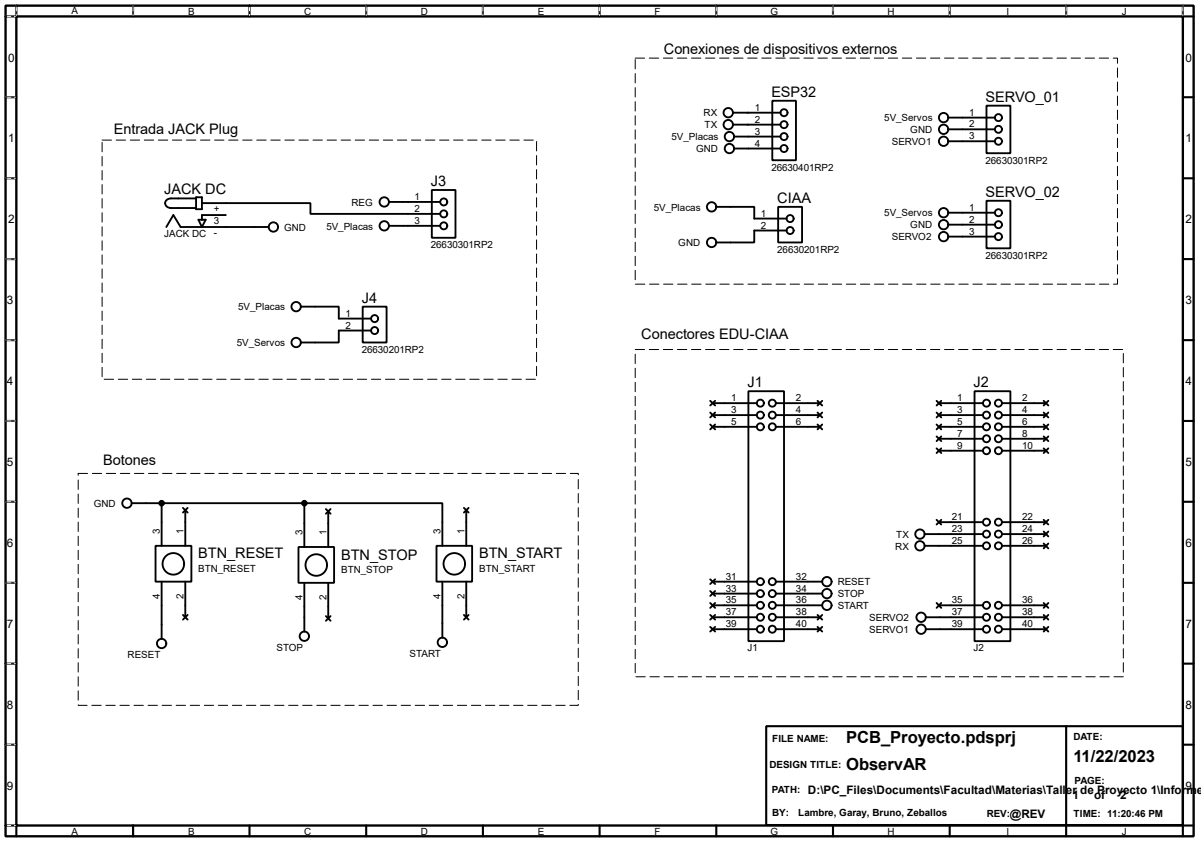
- ESP32 datasheet [pdf]. Disponible 11/2023. URL: esp32_datasheet_en.pdf (espressif.com)

11. Anexo A – Comunicación

Comunicación UART: En el siguiente enlace se podrá acceder a una carpeta en la nube que contiene, tanto el código de ambas placas como un video mostrando el correcto funcionamiento de la comunicación UART, correspondiente a la sección **5.1.** de este mismo documento: drive.google.com.

En el siguiente enlace se podrá acceder a una carpeta en la nube que contiene, tanto el código de ambas placas como un video mostrando el correcto funcionamiento de la comunicación UART, correspondiente a la sección **5.3.** de este mismo documento: drive.google.com.

12. Anexo B – Esquemático



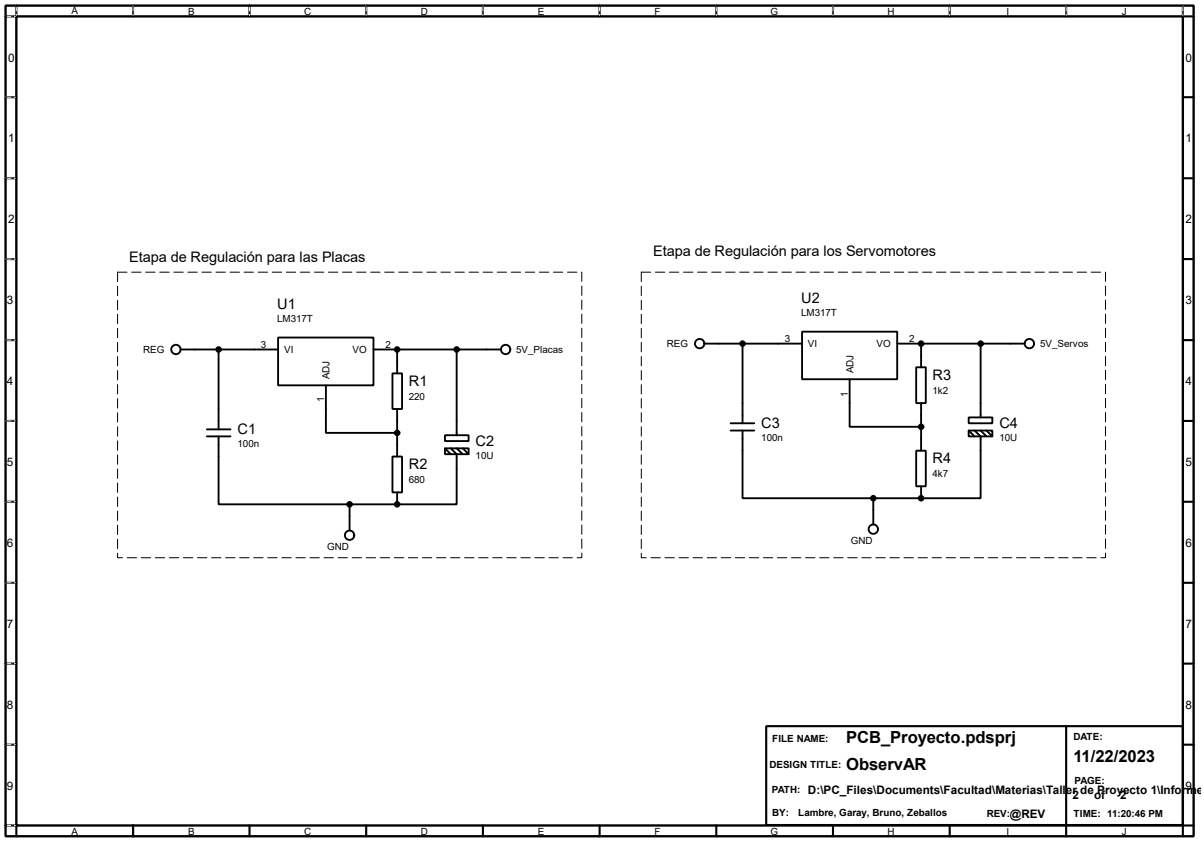


Figura A1 – Diagrama esquemático del proyecto

13. Anexo C – PCB

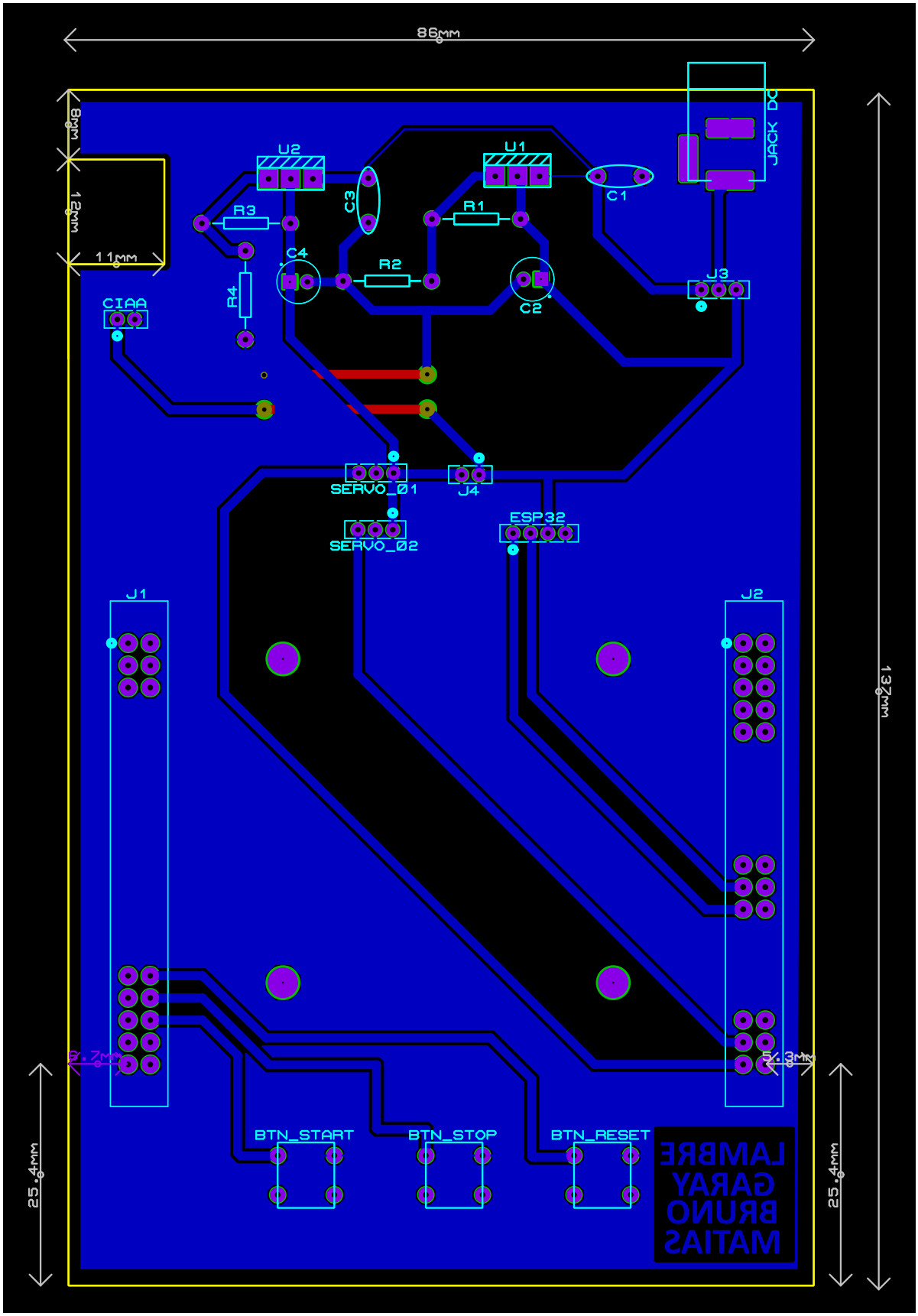


Figura B1 – PCB

14. Anexo D – Modelo 3D del PCB

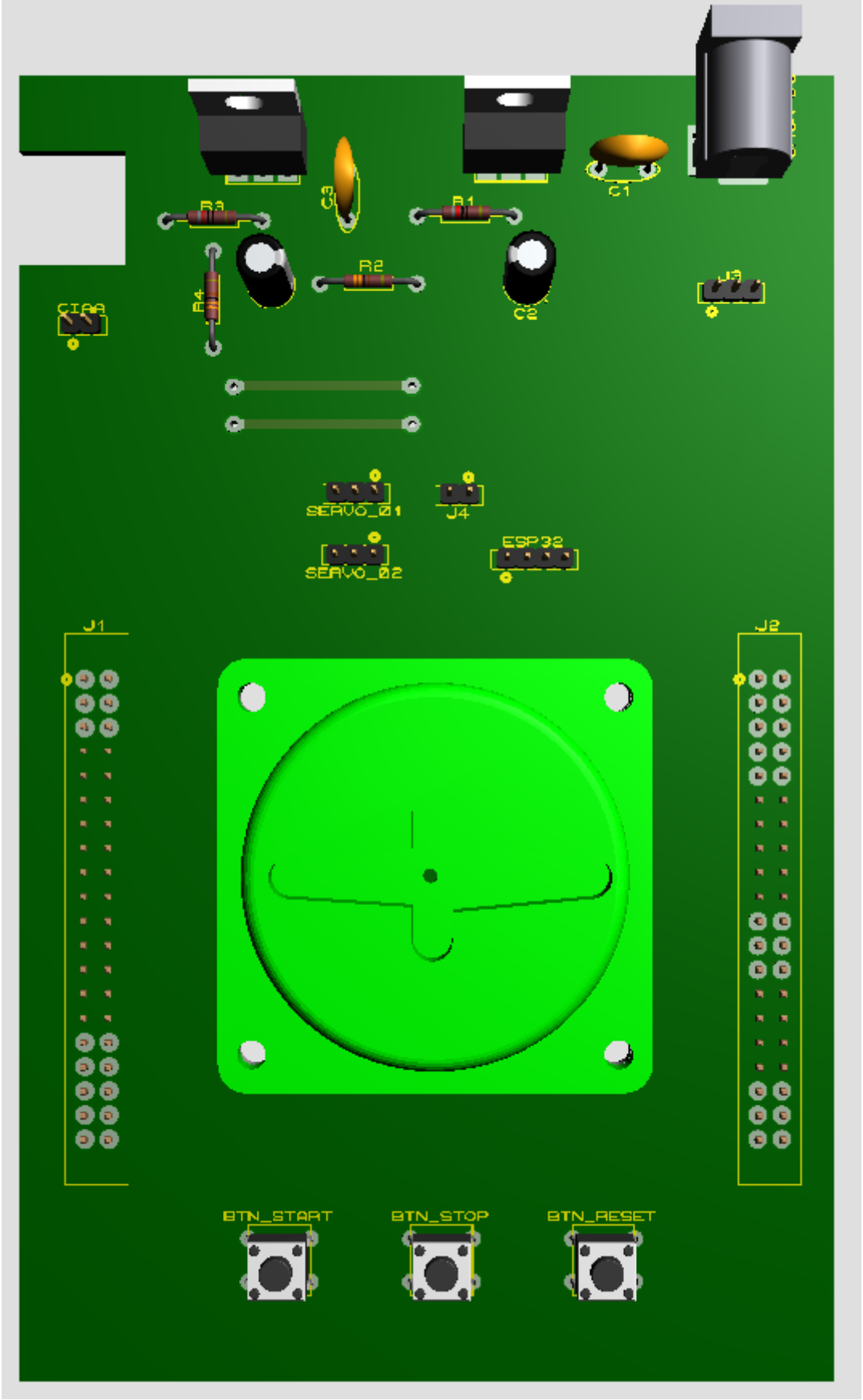


Figura D1 – Vista superior 3D de la PCB

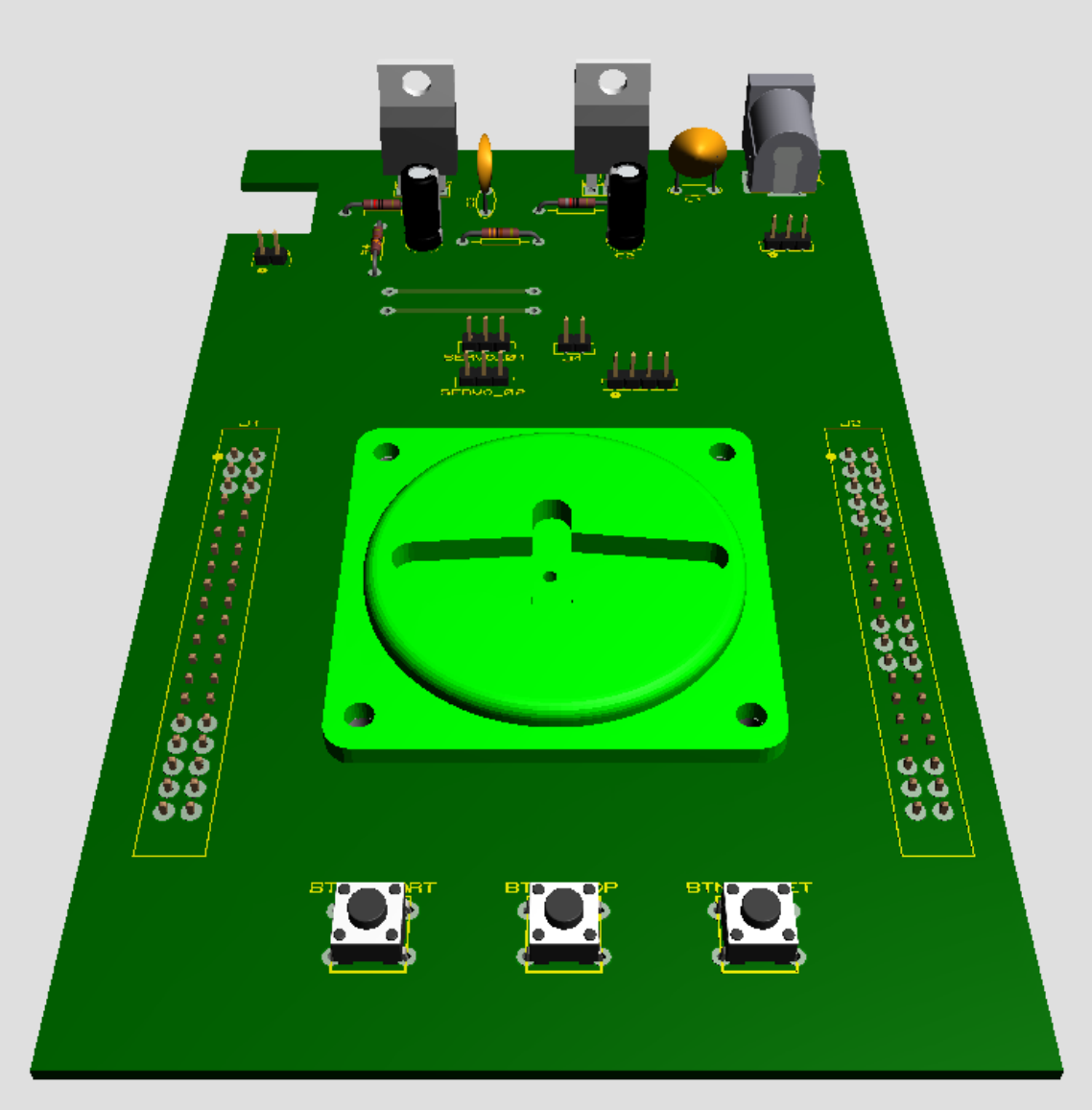


Figura D2 – Vista frontal 3D de la PCB

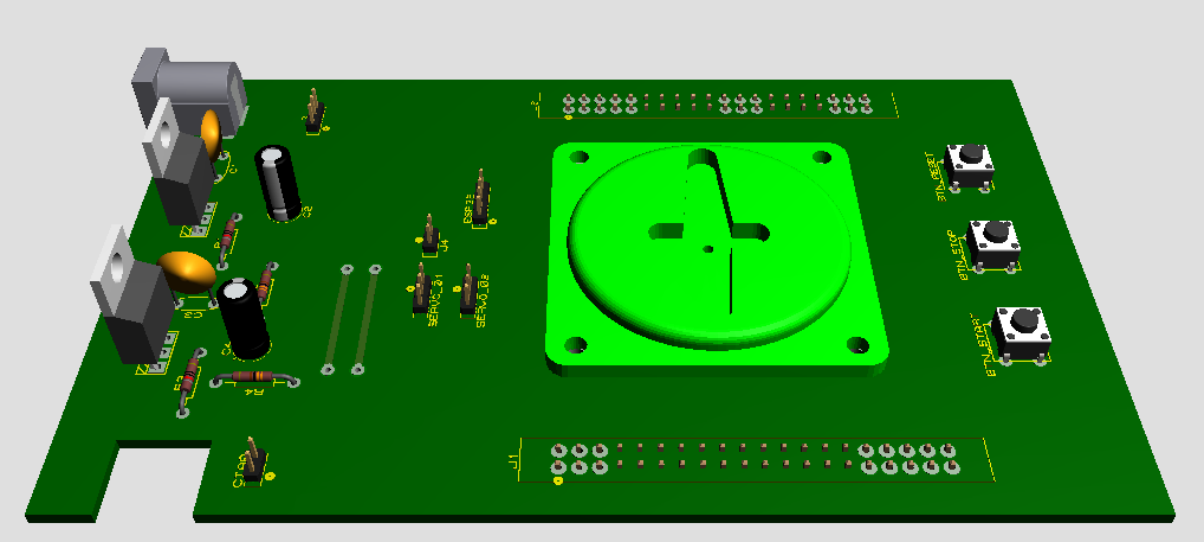


Figura D3 – Vista izquierda 3D de la PCB

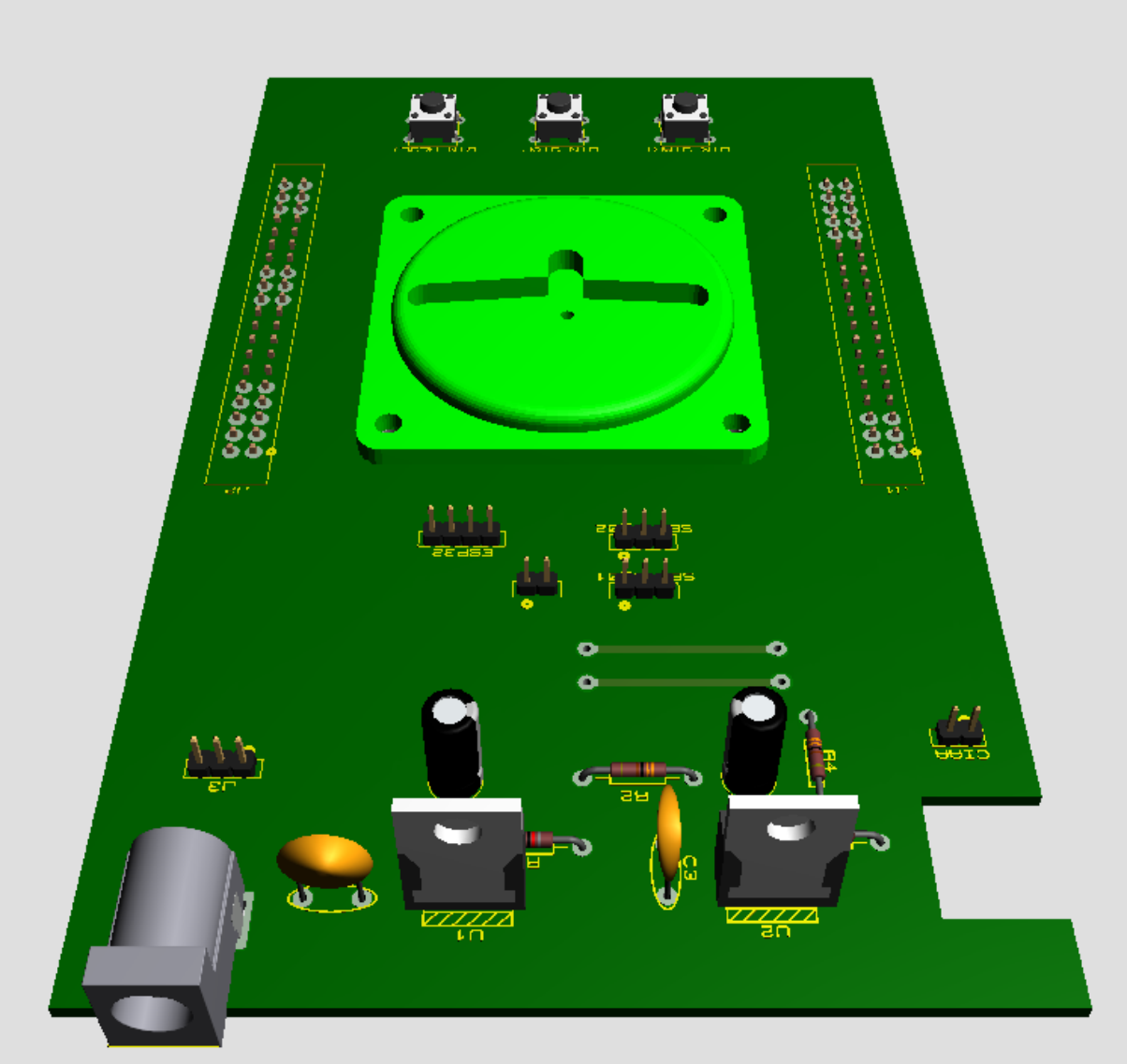


Figura D4 – Vista trasera 3D de la PCB

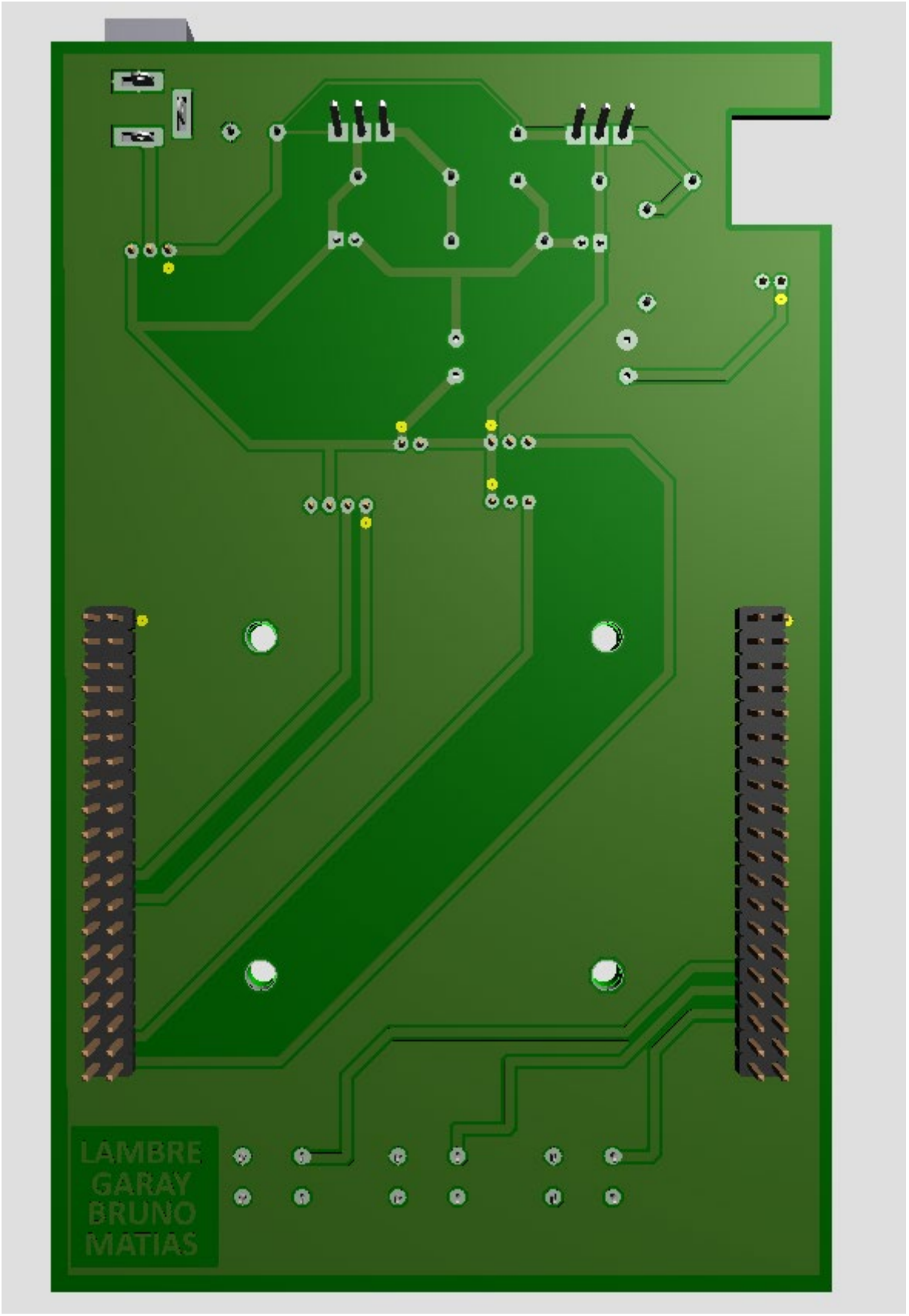


Figura D5 – Vista posterior 3D de la PCB con sus pistas y nombres