



Taller de Proyecto I

Poncho para control de motores DC utilizando comunicación WiFi

Alumnos: Ailán, Julián
Bouche, Federico
Liotta, Emiliano
Hourquebie, Lucas

Profesores: Sager, Gerardo Enrique
Juarez, Jose María
Aróztegui, Walter José

Grupo: 1

Año: 2015

Índice

- 1. Introducción y objetivos.**
- 2. Arquitectura de hardware.**
 - 2.1. Componentes a utilizar.**
 - 2.2. Diseño del poncho.**
- 3. Interfaz de usuario.**
- 4. Arquitectura de software.**
 - 4.1. Aplicación de la EDU-CIAA.**
 - 4.2. Aplicación Android.**
- 5. Cuerpo del robot.**
- 6. Conclusiones.**
- 7. Galería de imágenes.**
- 8. Anexos.**

1. Introducción y objetivos.

En el contexto de la materia *Taller de Proyecto I* para Ingeniería en Computación, se realizó un trabajo de aplicación utilizando la placa de desarrollo EDU-CIAA-NXP (*Computadora Industrial Abierta Argentina - Versión educativa*), contemplando el diseño y la realización tanto del hardware como del software necesarios.

Nuestro primer desafío se centró en la elección del proyecto, encontrar una idea que satisfaga a cada uno de los integrantes del grupo y que represente un crecimiento significativo para todos en cuanto a los contenidos académicos, así como también en el trabajo en conjunto.

La idea principal seleccionada se centra en el control de motores de corriente continua utilizando algún tipo de comunicación inalámbrica. Y para ello, se tuvieron que tomar algunas decisiones. La elección del protocolo de comunicaciones fue una de las decisiones más importantes, ya que se pretendió el logro de una abstracción del dispositivo desde el cual se envían los comandos de control. A partir de estas restricciones elegimos realizarlo mediante el uso del protocolo de comunicación inalámbrica IEEE 802.11, comúnmente conocido como WiFi. De esta manera, podría controlarse a través de un smartphone, una tablet, una PC, o cualquier dispositivo con WiFi.

Planteamos como objetivo práctico del proyecto el control de un número de dos motores de corriente continua haciendo uso de una aplicación en un smartphone con sistema operativo Android. Además, se anexaron a los motores cajas reductoras y dos ruedas para poder implementar un vehículo tipo robot a control remoto y así lograr un proyecto de mayor atracción.

La cantidad de tareas a completar se incrementó con la idea complementaria de fabricar un robot controlado mediante WiFi, ya que además del *poncho* para la EDU-CIAA y la requerida aplicación de software para el proyecto, se añadió el diseño y desarrollo de una aplicación para Android con soporte de comunicación inalámbrica y el diseño y fabricación del “chasis” para el robot.

Tanto el modelo del hardware como el software diseñados para este proyecto se encuentran almacenados en la plataforma de desarrollo colaborativo *GitHub*. Por cuestiones de orden, se decidió que los archivos correspondientes a hardware y software se encuentren en repositorios diferentes. De este modo se pudo trabajar en estos dos aspectos de forma independiente. Los repositorios son los siguientes:

- Repositorio de Hardware: github.com/cloud-9-team/motor_control_hardware
- Repositorio de Software: github.com/cloud-9-team/motor_control_firmware_apps

Una vez realizada la descripción del proyecto elegido por el grupo, estamos en condiciones de detallar con mayor profundidad el funcionamiento de los diferentes módulos del sistema, actuando tanto en conjunto como por separado.

2. Arquitectura de hardware.

2.1. Componentes a utilizar.

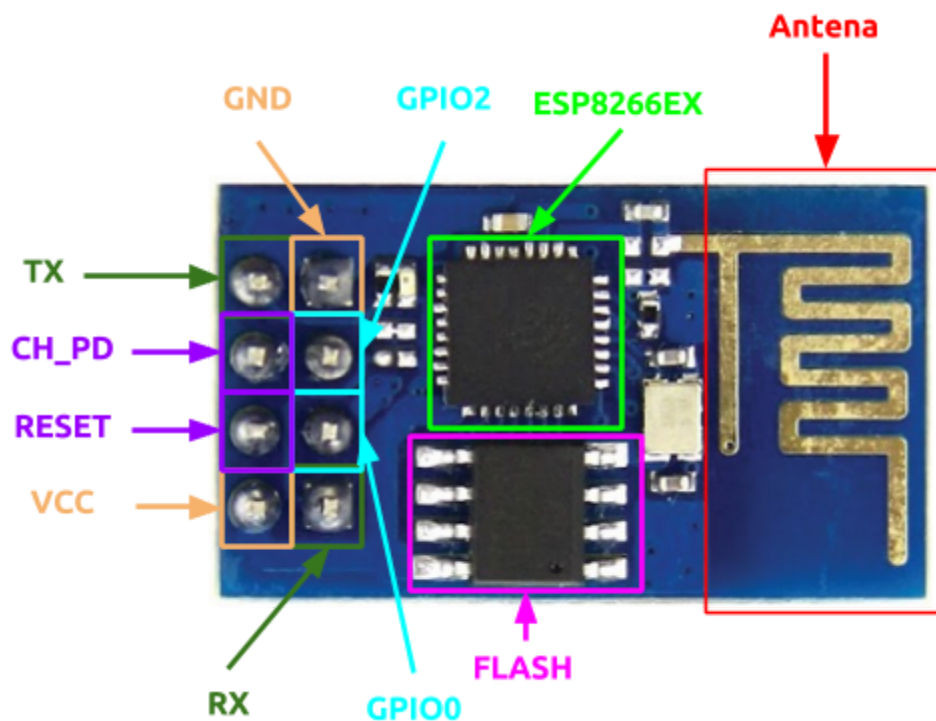
El poncho cuenta con dos tipos de circuitos integrados (ICs), el módulo WiFi ESP8266 y el puente H L293D, ambos ampliamente utilizados en proyectos de robótica y Arduino. De forma externa al poncho, los encoders utilizan el sensor óptico TCST 2103.

Si bien utilizamos estos ICs para el puente H y para los encoders, estos pueden ser armados utilizando los componentes constitutivos de dichos ICs. El puente H puede ser construido con cuatro transistores y cuatro diodos de protección, y el sensor óptico con una serie de resistores, un LED infrarrojo y un fototransistor. Pero el utilizar los ICs anteriormente nombrados nos provee de ventajas como un tamaño reducido en la placa y confiabilidad del resultado que cada una de ellas genere. En el caso del L293D, un IC comúnmente utilizado en robótica para control de motores DC o paso a paso, podemos estar seguros que cada conmutación de los transistores se hará de forma coordinada. Por otro lado, el TCST 2103, además de contar con el fototransistor y el fotodiodo, tiene un encapsulado y un sistema de acondicionamiento de señal, para evitar interferencias que pueda llegar a generar la luz natural.

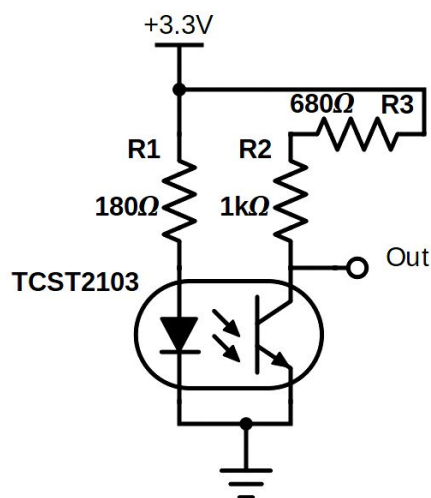
El **ESP8266EX** es el microcontrolador (MCU) que forma parte del **módulo WiFi ESP8266**. Puede funcionar como un MCU independiente con soluciones para redes WiFi, o como adaptador de red para trabajar en conjunto con otro(s) MCUs mediante comunicación serie, SPI, I²C o UART. Las características técnicas del MCU son las siguientes:

- MCU de 32 bits de bajo consumo y frecuencia de trabajo de 80 MHz,
- Memoria RAM de 96 KB,
- 64 KB de RAM de instrucciones,
- Presenta módulos GPIO, I²C, ADC de 10 bits, SPI y PWM,
- Potencia de salida de +20dBm en el modo 802.11b,
- Trabaja bajo el estándar IEEE 802.11 a 2.4GHz, con soporte para WAP y WAP2,
- Contiene la pila del protocolo TCP/IP,
- Rango de operación de -40°C hasta 125°C,
- Wake up y transmisión de paquetes en tiempos menores a 2ms,
- Consumo en Deep Sleep menor a los 10μA.

En el siguiente gráfico se muestra el subconjunto de pines del ESP8266EX que quedan accesibles al desarrollador:

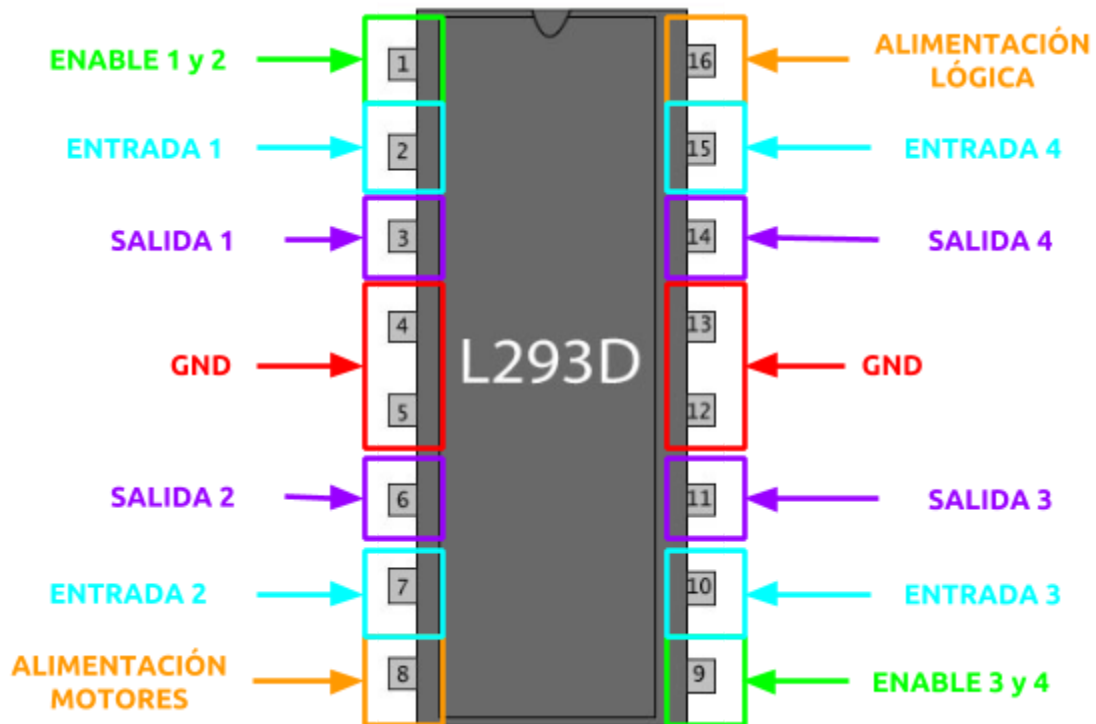


Los **sensores ópticos TCST 2103** están siendo alimentados con +3.3V provistos por la EDU-CIAA. De este modo, la salida del circuito de los encoders puede conectarse de forma directa con un GPIO de la EDU-CIAA, ya que trabajan con los mismos niveles lógicos. El circuito esquemático del encoder utilizado es el siguiente:

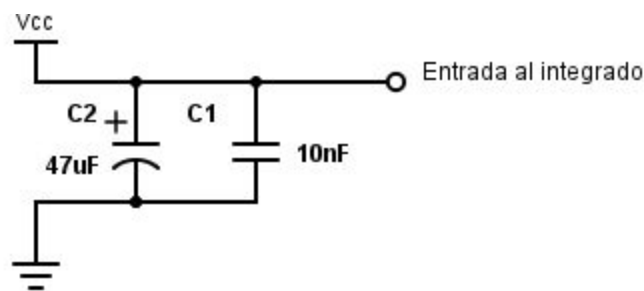


El valor de las resistencias utilizadas fue elegido de forma tal que se respeten los valores de lógica binaria de +3.3V que utiliza la EDU-CIAA. Gracias a la anterior configuración, se logró obtener un valor de tensión de salida del circuito aproximado a +3.28V para el nivel alto y de +0.15V para el nivel bajo.

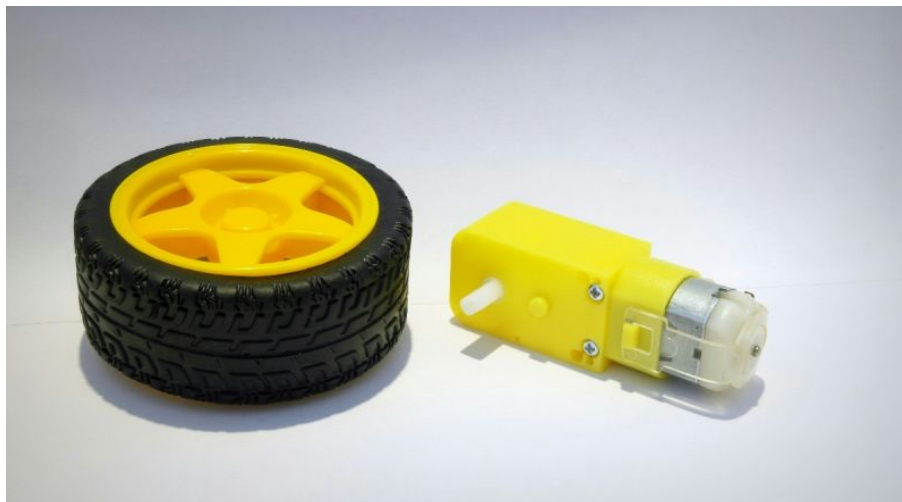
El **punto H L293D** maneja dos tipos de alimentación, una para la lógica propia del IC, y otra para los motores. La alimentación de la lógica debe ser de +5V, mientras que la de los motores puede variar en el rango +4.5V a +36V. Es un integrado que cuenta con alta inmunidad al ruido y protección contra sobretensión. La corriente de salida por cada canal de los motores debe ser a lo sumo 600mA, y de 1.2A para el IC en su totalidad. A diferencia del modelo L293, el L293D ya cuenta con los cuatro diodos de protección integrados. La disposición de los pines del L293D es la siguiente:



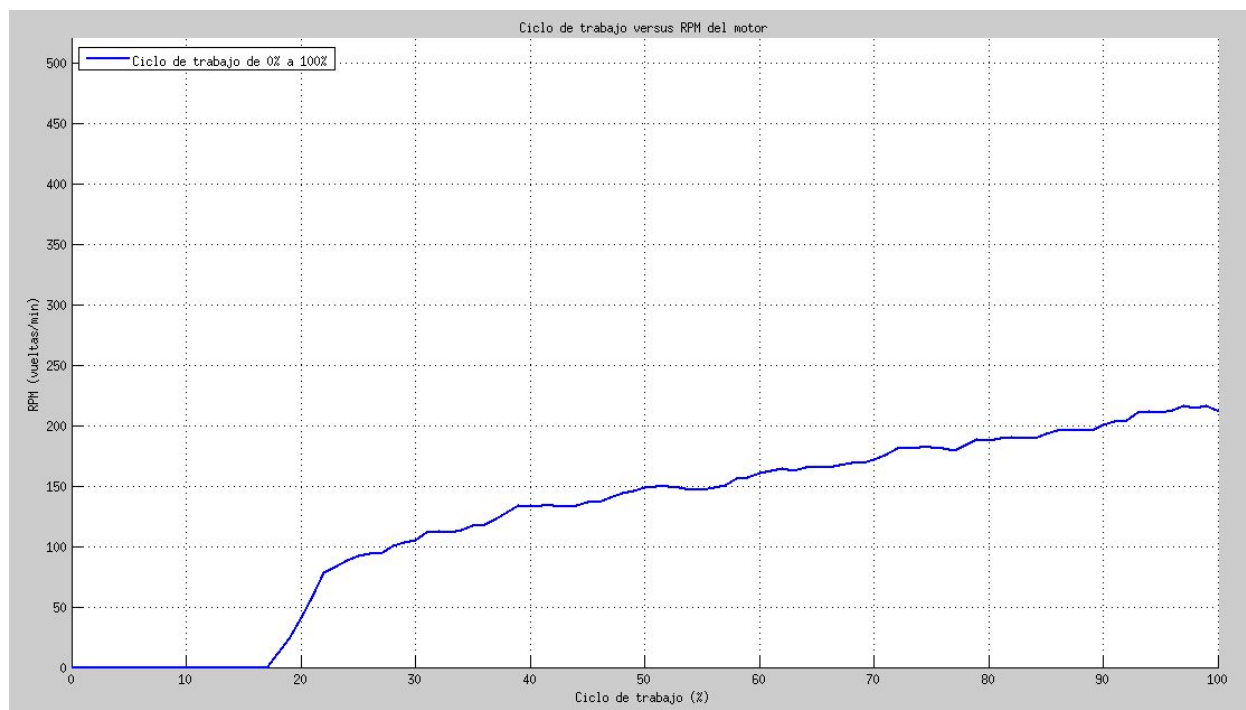
El puente H L293D se alimenta con +5V, y el módulo WiFi ESP8266 con +3.3V. Para mantener dicha tensión constante y filtrar ruidos, se utilizaron dos capacitores de desacople en paralelo, uno cerámico de 10nF y otro electrolítico de 47 μ F, a la entrada de cada uno de estos IC.



Los **motores de corriente continua** utilizados en el proyecto son de similares características entre ellos. Se alimentan con una tensión de 6V y alcanzan una velocidad máxima aproximada de 240 RPM. Esta baja velocidad se debe al anexo de una caja reductora a los mismos como se puede apreciar en la siguiente imagen.



Se realizó un *script* en MATLAB para realizar una comunicación inalámbrica con la EDU-CIAA y poder caracterizar los motores realizando una gráfica con dicho programa que se presentará a continuación. De la información obtenida es posible visualizar la velocidad máxima alcanzada, la forma de la curva durante la aceleración y el valor mínimo de PWM con el cual los motores comienzan a girar.



2.2. Diseño del poncho.

Consideraciones iniciales y primeros prototipos.

Para el diseño del diagrama esquemático y del circuito impreso se utilizó el software *KiCad*, propuesto por la cátedra. KiCad es un entorno de desarrollo de circuitos electrónicos distribuido bajo la licencia pública GNU. Dentro del paquete de software, se utilizaron las funcionalidades de *eeschema* para crear los diagramas esquemáticos, y *pcbnew* para diseñar el circuito impreso.

Desde el inicio del diseño del poncho se sostuvo la necesidad de que el mismo sea expresivo en la información que le va a dar al usuario que la utilice, es decir, que provea información suficiente y clara en lo que refiere a la tensión de alimentación de los componentes del poncho y de si este se encuentra en actividad.

Durante el desarrollo del proyecto trabajamos con diferentes prototipos de poncho. En todos ellos se busco que haya un plano de tierra, con fin de tener tierras accesibles y lograr aislamiento del ruido. El primer prototipo contaba con dos caras, la cara superior era un plano de tierra y la inferior contenía el trazado de las pistas. El problema subyacente a este esquema de diseño es que necesitaba de numerosos puentes para unir pistas lejanas entre sí. Si bien esto no es un inconveniente en un sentido estricto, ya que la funcionalidad era la correcta tanto en hardware como en software, atentaba directamente con la búsqueda de que la placa sea expresiva. Los puentes no mostraban con claridad que estaban uniendo, y no había referencias escritas que indiquen la alimentación necesaria de cada circuito integrado o las polaridades de los capacitores electrolíticos a utilizados.

El segundo prototipo utilizado fue realizado haciendo uso de las dos caras para trazar las pistas, pero aun así manteniendo en la capa superior el plano de tierra. De este modo se logró eliminar los puentes en su totalidad y reemplazarlos por pistas que cambian de capa únicamente de ser necesario. Aun así, este prototipo no contaba con ninguna realimentación visual para el usuario, que le haga entender que el poncho se encontraba en funcionamiento, o que estaba recibiendo datos.

Prototipo final.

Para solventar estos problemas, el tercer y último prototipo incluye dos LEDs que indican si el poncho se encuentra alimentado y si está recibiendo correctamente los comandos enviados desde el dispositivo que lo esté controlando de forma inalámbrica. Además, cuenta con referencias que indican unívocamente cómo debe conectarse la alimentación de los motores y cómo debe conectarse cada componente o circuito integrado.

El tamaño de pista utilizado fue de 1mm para todos los tipos de pistas utilizados. Los valores que si variaron fueron los de los pads. Los pads utilizados en las borneras tienen un diámetro exterior de 3mm y un diámetro interior de 1mm. En cuanto a los pads utilizados en las tiras de pines, se usó un esquema de 2.2mm de diámetro exterior y 0.75mm de diámetro interior. Por último, los pads de los circuitos integrados son de forma elipsoidal, con dimensiones de 2.6mm x 2.3mm y 0.75mm de diámetro interior.

Alimentación de la EDU-CIAA y el poncho.

Para la alimentación de la EDU-CIAA y el poncho se utilizó un *power bank* (cargador portátil de celulares) de 5V y 1A. La elección de esta fuente de alimentación surgió luego de un intento fallido de proveer corriente a través de una batería de 9V con su correspondiente regulación a 5V. Esta configuración no era capaz de proporcionarle a la EDU-CIAA y al poncho la tensión necesaria para que se mantenga en funcionamiento una vez que la misma comenzaba a demandar una intensidad de corriente mayor a los 250mA. Una vez que el módulo WiFi se ponía en funcionamiento, la tensión de alimentación caía drásticamente a aproximadamente unos 3.5V, los cuales resultaban en un continuo reseteo de la placa. A diferencia de la batería, el power bank proporciona de forma constante una tensión de 5V de corriente continua.

De todos modos, se desarrolló un circuito que utiliza un regulador de tensión LM7805, que puede ser utilizado para regular a 5V la alimentación de la EDU-CIAA y el poncho, de dicha fuente capaz de entregar la corriente necesaria. El circuito integrado LM7805 permite tensiones de entrada en el rango de 5V a 18V, y puede manejar corrientes hasta 1A.

Por otro lado, en lo que respecta a los motores, ambos están siendo alimentados con cuatro pilas de 1.5V, ya que la tensión recomendada por el fabricante de los mismos es de 6V.

Consumo de corriente.

En cuanto al consumo de corriente de los motores, cuando tienen que vencer la inercia, o ir en contramarcha, se logran picos de aproximadamente 400mA, que una vez que comienza a moverse se estabiliza alrededor de los 260mA. La EDU-CIAA, junto con el poncho, con la aplicación en funcionamiento, consumen también valores de corriente cercanos a los 260mA.

3. Interfaz de usuario.

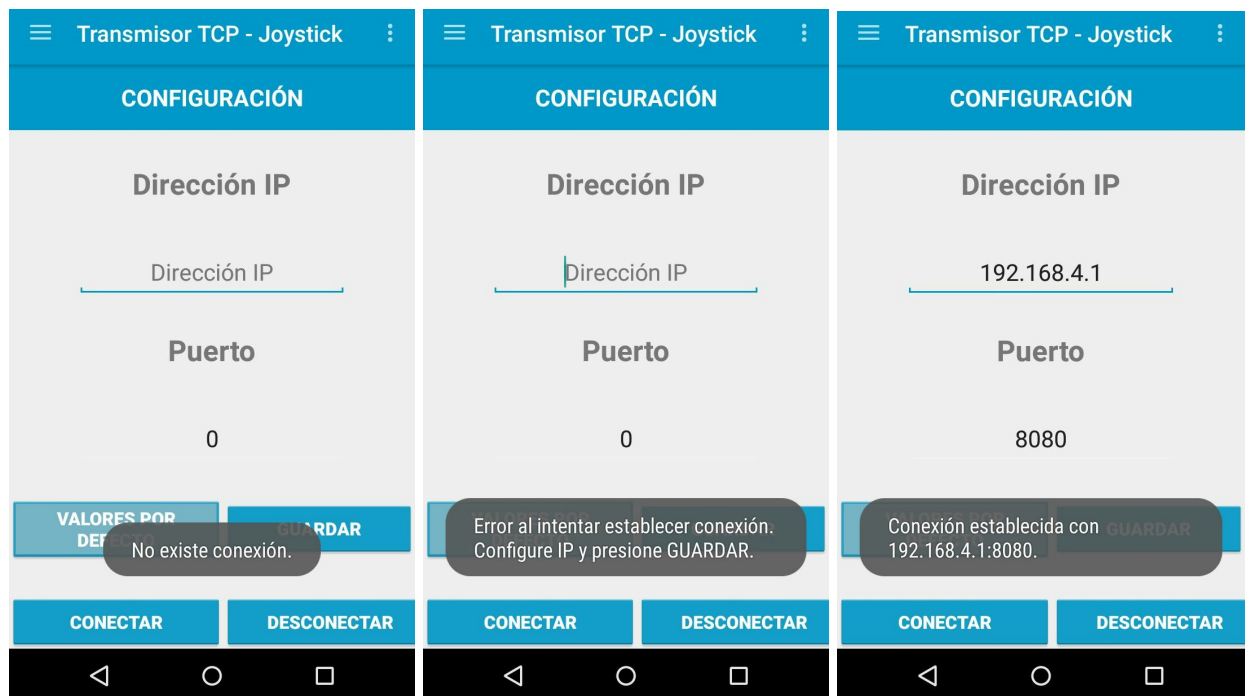
Tal y como se mencionó en la introducción del presente informe, el proyecto tendrá una interfaz de usuario de prueba implementada como aplicación móvil de un smartphone con sistema operativo Android.

La aplicación presenta básicamente tres vistas principales: una para configurar los datos de la conexión TCP de la placa (dirección IP y puerto con el que se comunica), la otra sirve como mando de interacción para el envío de datos y display de la recepción de información, y finalmente la última sirve para la calibración de los motores del robot.

Además, la aplicación está preparada para manejar excepciones ante posibles errores en el uso de la misma por parte de los usuarios, tales como el incorrecto o nulo ingreso de información de conexión. Por otro lado, también maneja excepciones que permiten tratar los problemas de una conexión: caída espontánea de un socket TCP, error en el envío de mensajes o en la recepción de datos.

A continuación, se adjuntan capturas del aspecto visual de la aplicación. Por cuestiones de practicidad, en la vista de configuración presenta un botón “Valores por defecto” que permite autocompletar la información de conexión con los valores que se asignaron por defecto al módulo WiFi del poncho de la CIAA.

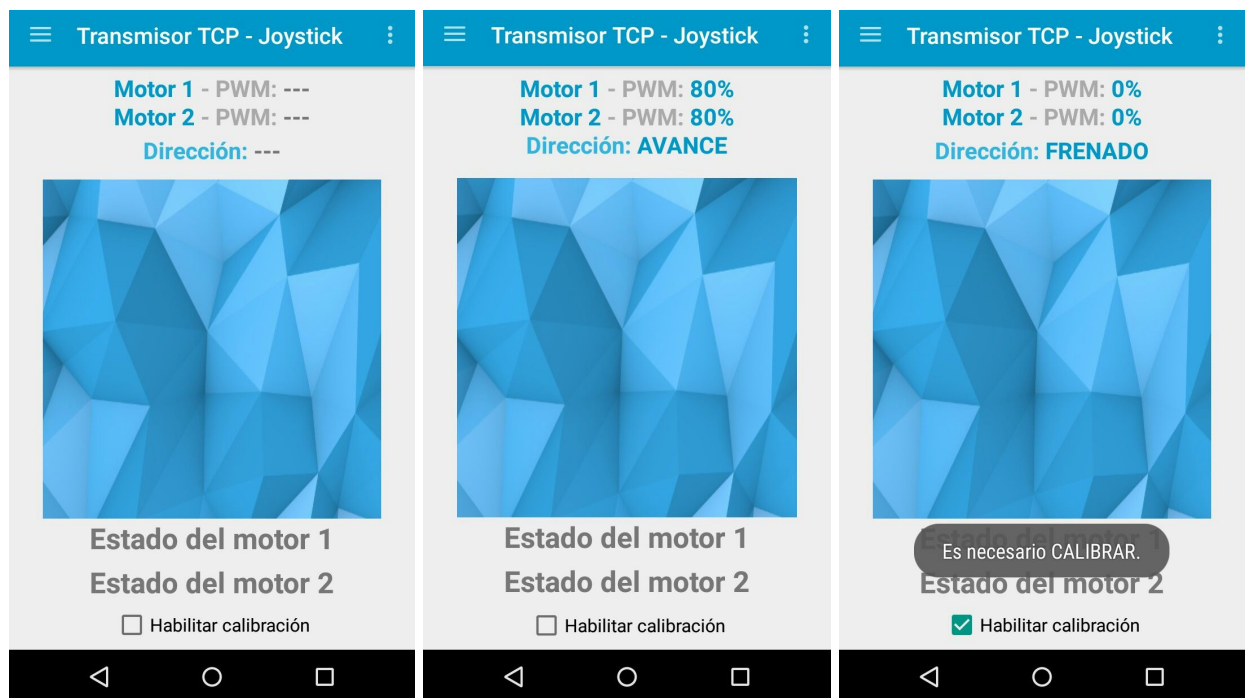
→ Pantalla de configuración:



En las capturas de pantallas anteriores, se puede observar la configuración de conexión con el servidor, gracias a la cual la aplicación se conecta con la EDU-CIAA y se controla el robot.

Para ello es necesaria la introducción de dos datos: la dirección IP del servidor y el puerto del mismo. Por defecto, el módulo WiFi trabaja en la dirección *192.168.4.1* y el puerto *8080*: la función del botón “Valores por Defecto” es autocompletar el formulario con estos datos.

→ Pantalla de interacción:



Una vez realizada la conexión, para poder controlar el robot, se implementó una especie de *joystick*. La dirección de movimiento está dada de acuerdo a la posición en la cual se apoya el dedo en el gráfico rectangular en el centro de la pantalla.

Por sobre el *joystick* se observa la información de los motores y la dirección de movimiento del robot. También se presenta en la pantalla la velocidad real en RPM (revoluciones por minuto) de cada uno de los motores.

En el hipotético caso de que los motores no funcionen de igual manera, es decir, que con la misma tensión de entrada giren a distinta velocidad, puede habilitarse la calibración de los mismos. Dicha calibración limita la velocidad máxima del motor más veloz a la velocidad máxima del motor más lento.

→ Pantalla de calibración:



Para realizar la calibración de los motores nombrada previamente, se necesitan dos parámetros. El primero de ellos es el tiempo en milisegundos durante el cual el programa recibirá información de los encoders para cada ciclo de trabajo del PWM (de 0% a 100%). Cuanto mayor sea este tiempo, más exacta será la calibración.

El número de ranuras del disco hace referencia a las ranuras del disco perteneciente al encoder. Dicho número es necesario para los cálculos de RPM. Un número mayor de ranuras genera una resolución mayor en el cálculo final.

4. Arquitectura de software.

2.1. Aplicación de la EDU-CIAA.

El software desarrollado se basa en el empleo del firmware desarrollado por los participantes del proyecto CIAA. Este firmware, a su vez, hace uso del sistema operativo de tiempo real OSEK-OS implementado de acuerdo a su estándar.

La metodología de desarrollo utilizada es la programación modular, utilizando el lenguaje de programación C.

La planificación de tareas consiste en tareas periódicas, cada una caracterizada por su período y además por su prioridad de acuerdo a la necesidad de ejecución que posea. El sistema operativo tiene la capacidad de apropiarse de tareas de baja prioridad en pos de la ejecución de tareas de prioridad mayor. Una tarea será ejecutada hasta que ésta explícitamente finalice su ejecución ó hasta que una tarea de mayor prioridad pase a estado *ready*.

Nuestro proyecto, como anteriormente se ha detallado, tiene como objetivo principal el control de motores a través de WiFi. También, se prevé la posibilidad de caracterizar cada uno de los motores mediante curvas de ciclo de trabajo versus velocidad (en unidades de RPM por ejemplo), así, con previo análisis de los datos, lograr optimizar su control y brindar una mejor experiencia de manejo.

Para llevar a cabo la tarea, primero es necesario delimitar qué tareas realizará el software de la EDU-CIAA, y cuáles otras serán delegadas al sistema con el que interactúa el usuario para controlar los motores a su antojo, siendo este software la aplicación en Android. En la siguiente tabla se pone de manifiesto la delimitación de las tareas.

Temática	EDU-CIAA	Aplicación Android
Encoders	Obtención de mediciones (interrupciones generadas durante un tiempo determinado). Envío de estos datos a la aplicación.	Recepción de datos (cantidad de interrupciones por período de tiempo). Conversión de los datos a unidades más significativas para el usuario.
PWM	Generación de las señales que ingresan al puente H encargado de controlar los motores. Esto implica, dado un número de motor, un valor de ciclo de trabajo y un sentido de movimiento, habilitar y/o deshabilitar las salidas PWM que correspondan.	Interpretar el movimiento que busca realizar el usuario, por ejemplo mover los motores hacia adelante con determinada velocidad, y a partir de ello, informar qué motor, con qué ciclo de trabajo y sentido, debe controlarse.

Módulo WiFi	Administrar la interfaz serie con el módulo, procesando los mensajes que emite y enviando comandos de acuerdo a la interfaz provista por el fabricante.	La interacción con el módulo WiFi es transparente para la aplicación. Le basta con conocer los parámetros para conectarse al <i>access point</i> iniciado por el módulo, y los datos para iniciar la conexión TCP (IP, puerto) con el servidor.
Caracterización de motores	Ante expresa solicitud, se generan los datos que caracterizan al motor. Estos datos son la cantidad de interrupciones medidas durante un período de tiempo, para cada uno de los valores de ciclo de trabajo. Los datos generados son enviados a la aplicación, no se realiza ningún procesamiento adicional.	Solicita la caracterización, y recibe los datos medidos, almacenándolos para futuro uso.
Calibración	Respecto a este tópico, no se realiza ninguna actividad.	Este tema ha sido delegado en su totalidad a la aplicación en Android. Básicamente se busca caracterizar los dos motores empleados, y a partir de los resultados obtenidos, realizar los ajustes que sean necesarios para asemejar ambas curvas.

Nótese que algunas de las tareas delegadas a la aplicación de Android podían realizarse directamente en la EDU-CIAA. La implementación para un microcontrolador podía resultar dificultosa o carente de utilidad, por lo que fueron asignadas a la aplicación Android debido a las facilidades que esta presenta para su implementación. Por ejemplo, la conversión de cantidad de interrupciones en un período de tiempo a RPM involucra varios productos y cocientes, algo costoso para el MCU, además que no aporta información significativa para el programa. Pero el usuario difícilmente pueda entender los valores sin convertir, por lo cual la aplicación sí debería convertirlos.

Una vez asignadas las tareas, hay que definir una interfaz común para que conjuntamente estos dispositivos lleven a cabo los objetivos.

La comunicación entre la EDU-CIAA y la aplicación en Android se realiza de la siguiente forma:

1. EDU-CIAA configura al módulo WiFi para que monte un *access point* proporcionándole un SSID, una contraseña, el tipo de encriptación y el canal a emplear.

2. EDU-CIAA configura al módulo WiFi para que inicie un servidor TCP en un puerto determinado.
3. El dispositivo con Android se conecta al WiFi, y luego la aplicación se conecta como cliente al servidor TCP.

Ahora, ambos dispositivos son libres para intercambiar datos, por lo cual se requiere definir cómo será este intercambio. Para ello, hacemos uso de comandos representados por cadenas de caracteres.

A continuación, dichos comandos. Tener en cuenta que si hay que enviar un número, se envía su representación como cadena de caracteres.

Nombre	Formato		Enviado por
Control de motor			
DUTYCYCLE	%[LONGITUD][IDMOTOR][DUTYCYCLE]		Android
	Permite manipular los motores.		
	LONGITUD	Un solo carácter, que indica la cantidad de caracteres que se suceden luego de este campo.	
	IDMOTOR	Número de motor, si hay dos motores, 0 o 1.	
	DUTYCYCLE	Ciclo de trabajo a establecer en el PWM que controla el motor especificado. Toma valores de 0 a 200, porque indica el sentido también. Valores de 0 a 100 mueven al motor en un sentido, siendo 0 la mayor velocidad y 100 la menor. Valores de 100 a 200 controlar al motor en el otro sentido, siendo 100 el menor valor de ciclo de trabajo y 200 el máximo.	
SPEED	\$SPEED[IDMOTOR][TIPO][VALOR]\$		EDU-CIAA
	Información brindada por los encoders a quien controle los motores.		
	IDMOTOR	Número de motor, un solo carácter.	

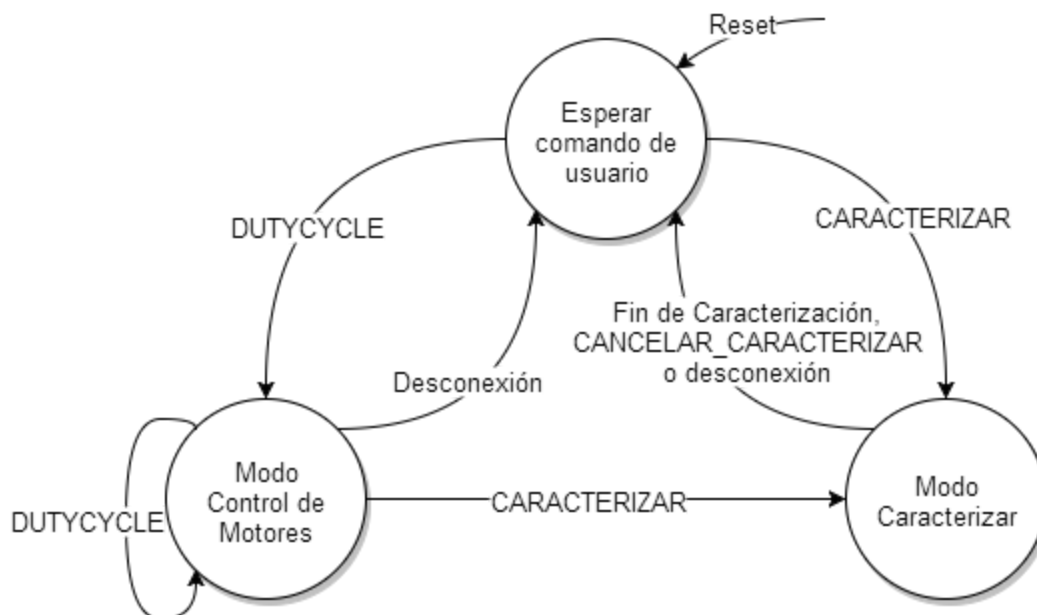
	TIPO	Tipo de valor devuelto. Un solo carácter. 0: RPM, 1: interrupciones por segundo.	
	VALOR	Valor medido.	
Caracterización de motor			
CARACTERIZAR	\$CARACTERIZAR=[IDMOTOR],[TIEMPO]\$		Android
	Comienza el proceso de caracterización de un motor.		
	IDMOTOR	Número de motor a caracterizar.	
	TIEMPO	Tiempo en milisegundos durante el cual se contarán interrupciones del motor IDMOTOR, para cada uno de los distintos valores de ciclo de trabajo. Valor máximo: 10000.	
MOTOR	\$MOTOR=[IDMOTOR], [DUTYCYCLE], [CANT_INT]\$		EDU-CIAA
	Cuando se está caracterizando, se envía este mensaje que va indicando el progreso del proceso, además de dar la información pertinente. Es enviado cada TIEMPO milisegundos, indicados en el comando CARACTERIZAR.		
	IDMOTOR	Número de motor, igual al IDMOTOR del comando CARACTERIZAR.	
	DUTYCYCLE	Ciclo de trabajo del PWM usado para controlar el motor.	
	CANT_INT	Total de interrupciones computadas durante un período de tiempo bajo el mismo ciclo de trabajo.	
FIN_CARACTERIZAR	\$FIN_CARACTERIZAR\$		EDU-CIAA
	En cualquier caso de error, esta cadena es enviada. Si MENSAJE contiene el carácter \$ deberá ser reemplazado por \$\$.		
ERROR	\$ERROR=[MENSAJE]\$		EDU-CIAA

	Cuando se procesaron todos los ciclos de trabajo, finaliza la caracterización, y se envía este mensaje así indicándolo.	
CANCELAR_CARACTERIZAR	\$CANCELAR_CARACTERIZAR\$	Android
	Como su nombre indica, cancela la caracterización	

Una consideración relevante es el uso de delimitadores para los comandos, tales como los caracteres '%' y '\$'. Esto se debe a cuestiones del protocolo TCP, asociadas al hecho de que la transmisión de datos es en forma de *stream*, no provee un método para diferenciar mensajes individuales. El emisor tampoco está obligado a transmitir los datos apenas le llegan, ya que si son datos de muy poco tamaño se reduce el rendimiento de la transferencia porque el overhead del paquete TCP es considerable. Una mejora consiste en demorar la transmisión para poder recibir suficientes datos y aumentar la eficiencia. Esto, a muy grandes rasgos, es el algoritmo de Nagle. Por lo tanto, es muy probable que al enviar dos comandos separados por una cantidad de tiempo pequeña, lleguen juntos al receptor como un solo mensaje.

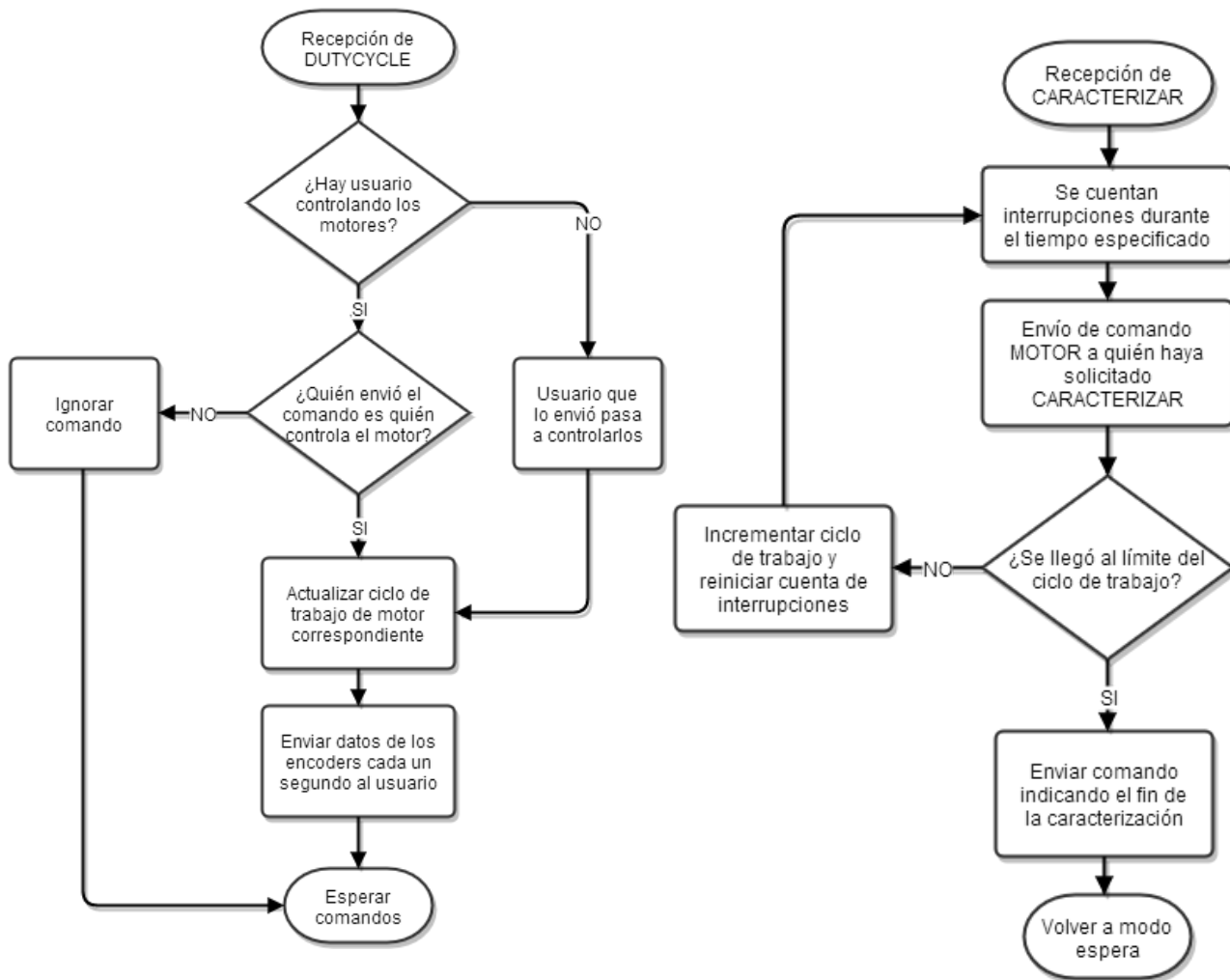
Viendo los comandos, es fácil apreciar la existencia de dos modos de funcionamiento, uno en el cual se controlan los motores a gusto del usuario, y otro para caracterizar los motores siendo el fin de esto poder calibrarlos. También se puede concebir otro modo, el cual se encuentra a la espera de comandos.

Para el desarrollo en la EDU-CIAA, podemos explicitar el siguiente comportamiento frente a los comandos:



Sólo los comandos especificados en la gráfica pueden realizar el cambio de un modo a otro. Desconexión es la desconexión (cierre de la conexión TCP) del usuario que envió el comando que provocó el cambio de modo. Se puede apreciar, por ejemplo, que si se está caracterizando, no van a ser aceptados los comandos de DUTYCYCLE.

Los dos modos de funcionamiento se pueden describir empleando diagramas de flujo:



Una vez presentada de manera general el funcionamiento de la solución implementada, podemos proceder a introducir los módulos que conforman el software, que materializan todo lo anteriormente expresado.

Módulo PWM

Administra, empleando la implementación como dispositivo provista por el firmware, los distintos dispositivos PWM. En total se utilizan cuatro, dos por motor, cada uno de éstos para desplazarse en un sentido distinto. Los pines utilizados se pueden observar del esquemático del poncho.

El módulo al ser inicializado establece el valor del ciclo de trabajo para todos los PWM en 0%.

Expone la posibilidad de modificar el ciclo de trabajo de las señales, a partir del número de motor, la dirección del movimiento y el valor del ciclo de trabajo en sí.

Módulo ENCODER

Internamente se configuran los pines utilizados como GPIO en los cuales se encuentra conectada la salida de los encoders. Tales pines pasan a ser entradas, y se habilita un canal de interrupción GPIO para cada uno de estos. La interrupción ocurrirá en el flanco descendente de la señal de entrada.

Los manejadores de las interrupciones simplemente incrementan una cuenta de la cantidad de interrupciones ocurridas para cada uno de los encoders.

Este módulo tiene la capacidad de contar periódicamente las interrupciones:

1. Cuenta interrupciones durante un tiempo especificado.
2. Al transcurrir ese tiempo, notifica a quien hace uso del módulo, y le permite obtener la cantidad de interrupciones contabilizadas en el período.
3. Luego, se reinicia la cuenta, y vuelve a comenzar el período, regresando a 1.

La notificación del tiempo transcurrido se hace a través de un *callback*, que es el llamado a una función previamente definida por quien usa el módulo. Esta función, por lo general, se encuentra en otro módulo del programa, permitiendo así que el módulo ENCODER se desligue de lo que se hace con la información provista.

En este módulo entra en juego una tarea, llamada **EncoderTask**. Ésta se activa periódicamente de acuerdo al período establecido y es quien se encarga de finalizar las cuentas y realizar las notificaciones al sucederse el tiempo.

Módulo TIMER

Configura el periférico Repetitive Interrupt Timer (RIT), para que genere una interrupción cada milisegundo. Su utilidad reside en la capacidad de generar retardos (*delay*) por una cantidad de milisegundos dada. Estos retardos son utilizados en otros módulos para generar *timeouts*, o esperar a que un dispositivo se inicialice, como lo es en el caso del módulo WiFi.

Los retardos también se pueden generar “quemando” ciclos de ejecución, conociendo de antemano el tiempo que tarda en ejecutarse una instrucción, y a partir de ello ejecutarla todas las veces que sea necesario para que transcurra el tiempo deseado. Esto presenta la desventaja que, ante cualquier modificación del entorno tal como el cambio de la frecuencia de reloj, u optimizaciones del compilador, produce retardos incorrectos. Esto no sucede al emplear un timer.

Módulo STRINGUTILS

Provee funciones de manipulación de cadenas de caracteres, conversión de enteros a cadena de caracteres, y viceversa. Al trabajar con comandos en que varios de sus parámetros son enteros convertidos, este módulo adquiere gran utilidad.

Módulo DEBUG_LOGGER

La EDU-CIAA nos proporciona a través de la conexión USB el acceso a una UART. Ésta, para propósitos de debugging, es realmente útil, pero una vez que el desarrollo está finalizado, y no se posee acceso a una computadora para leer su salida, es conveniente deshabilitarla para ahorrar aunque sea un poco en el consumo.

Este módulo provee funciones para escribir y leer de dicha UART. Cuando una macro está definida (LOGGING_ALLOWED), su funcionamiento es normal. Si dicha macro no está definida, entonces la UART será deshabilitada, y además el cuerpo de todas las funciones que expone no será compilado, por lo cual un compilador inteligente optimizará los llamados a esas funciones, eliminándolos.

Módulo PARSE

Este módulo es el eje del procesamiento de los caracteres recibidos por parte del módulo WiFi. Provee una interfaz común y declara ciertos tipos utilizados para crear y manipular parsers. Un parser, en el contexto de este programa, es un módulo de software que cumple con ciertas especificaciones del módulo PARSE y que tiene como finalidad recibir caracteres, uno a uno, y decidir si cumplen con algún patrón o regla dados. A su vez, existe la posibilidad que al detectar el patrón, también se extraigan datos de éste y sean devueltos al usuario como resultados del parseo.

Un ejemplo de patrón puede ser: un carácter ‘%’, seguido de un carácter numérico, cuyo valor numérico debe ser guardado, y además éste indica la cantidad de caracteres siguientes que corresponden al mismo patrón, que también deben ser guardados.

Entonces si se realiza un parser para ese patrón, al recibir los caracteres “%2AB”, de izquierda a derecha, puede decir que encontró el patrón, y devolverá como resultado 2 y “AB”. Si recibe “%2A”, informa que todavía no se encontró el patrón pero existe la

posibilidad de encontrarlo próximamente, y lo hará cuando reciba un carácter más. Si recibe “%2AB%1C”, al recibir el carácter ‘B’ informa que encontró el patrón, y hará lo mismo cuando reciba ‘C’.

Entonces, cada vez que se provea un carácter, se verifica si este carácter forma parte del patrón. Si parece válido pero aún restan más caracteres para formarlo, se dice que el parseo está **incompleto**. Si el patrón fue hallado exitosamente, el parseo se **completó**, y si el carácter de ninguna manera forma parte del patrón, se dice que **falló**. Se observa que para cada carácter introducido al parser, éste informa el estado actual de la operación.

Una posible forma de implementar un parser, y como se los ha implementado en este proyecto, es con una máquina de estados finitos, y en función del estado actual se decide si el carácter es válido o no. Para el ejemplo propuesto, el estado inicial sería esperar por un ‘%’, una vez recibido este carácter se pasa al siguiente estado donde sólo será válido aquel carácter que represente un número. Si en este estado se recibe algo distinto, se pierde el progreso logrado y se vuelve al estado inicial. El último estado consistirá en guardar caracteres hasta llegar a la cantidad indicada en el segundo estado.

Hasta ahora, se puede pensar que todo lo anteriormente visto se puede suplir con alguna función del tipo *sscanf*. En principio, es una opción válida y sencilla de manejar. Una dificultad a superar es que generalmente no se encuentra implementada para sistemas embebidos debido a su complejidad inherente y lo poco eficiente que puede llegar a ser su ejecución.

La principal ventaja que presenta el uso de parsers reside en el hecho de que no es necesario tener la cadena de caracteres completa para comenzar a procesar. Si se usa para buscar determinados patrones en los caracteres recibidos por serie, se los puede ir procesando a medida que son recibidos, sin necesidad de almacenarlos en su totalidad. Otra ventaja es la posibilidad de buscar patrones más complejos que los que se pueden representar con las cadenas de formato de *sscanf*.

Múltiples parsers son utilizados en esta solución para procesar mensajes del módulo WiFi, tales como la conexión o desconexión de un usuario, detectar el reinicio del módulo por alguna causa externa, identificar mensajes que indican la recepción de datos enviados por un cliente conectado al servidor TCP. También son usados para detectar todos los comandos que pueden ser recibidos de la aplicación de Android, detallados anteriormente.

Módulo ESP8266

Este módulo tiene el objetivo de brindar una interfaz sencilla para el programador que desea hacer uso del módulo WiFi. Se encarga de manejar la entrada y salida correspondiente al módulo WiFi.

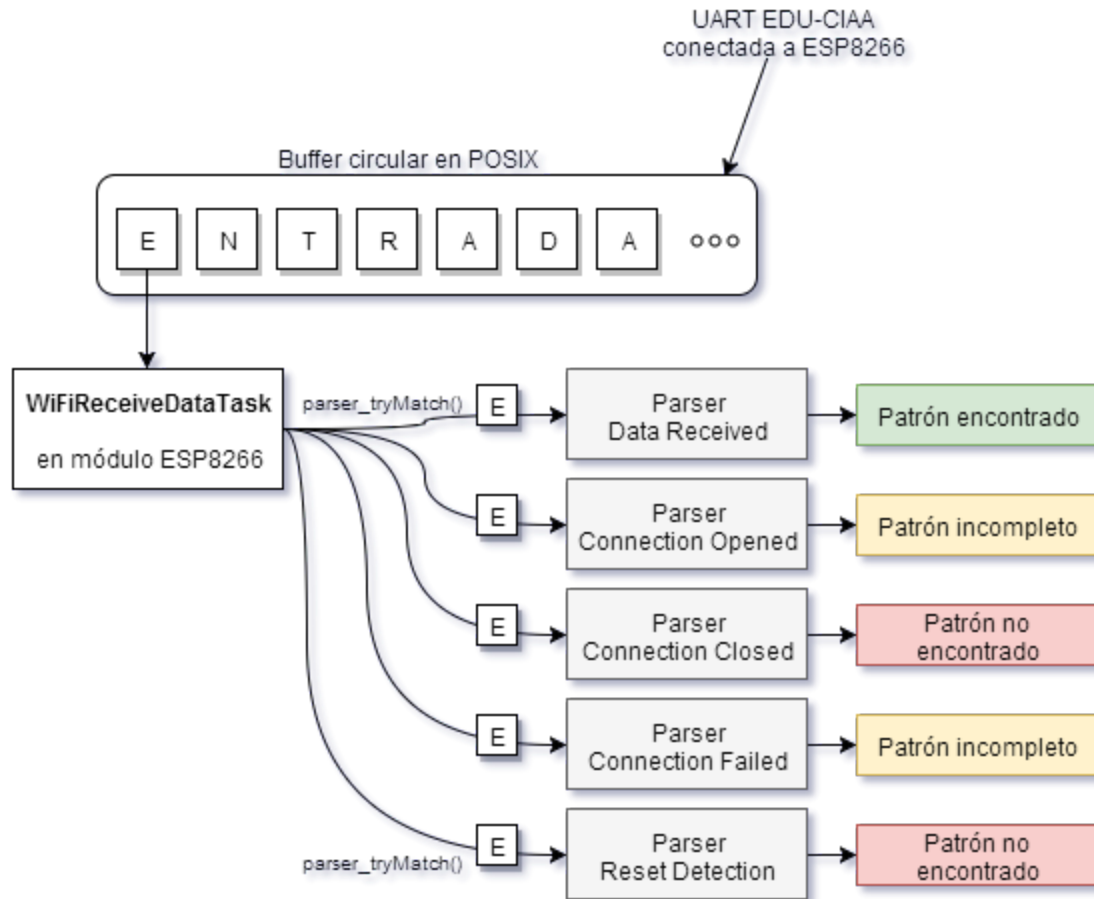
Por la entrada se entiende a los caracteres que envía hacia la EDU-CIAA. Aquí entra en juego otra tarea definida, **WiFiDataReceiveTask**, que se ejecuta cada 20ms y lee los

datos recibidos por la UART en un buffer temporal de 255 caracteres. La selección de estos valores no es al azar, sino que garantizan que no se pierda ningún dato en el buffer de recepción del dispositivo implementado bajo POSIX a causa de un overrun.

Dicho buffer tiene una capacidad para 255 caracteres. La transmisión de un carácter implica transmitir 10 bits (1 bit de comienzo, 8 de datos y 1 de parada). La velocidad de la transmisión es 115200 baudios (bits/segundo en este caso). Para que se llene el buffer, deben transcurrir $10 \cdot 255 / 115200 = 22,13\text{ms}$. Por lo cual si se leen cada 20ms, con un buffer de la misma capacidad, se evita cualquier *overrun*. También es necesario que la tarea **WiFiDataReceiveTask** tenga la mayor prioridad, para que no sea retrasada por ninguna otra tarea.

Una vez leídos los datos del buffer, se los somete a la búsqueda de varios patrones, como conexión o desconexión de usuario, datos recibidos o reseteo del dispositivo. En función de los resultados, se actualiza el estado interno del módulo, por ejemplo se tiene información para saber qué conexión está activa y cuál no, y se llaman a los *callbacks* correspondientes.

El módulo presenta *callbacks* para indicar el cambio de estado de una conexión, informar la ocurrencia de un reset en el módulo WiFi y para notificar la recepción de datos en la sesión TCP. El funcionamiento de la recepción, a grandes rasgos, se puede graficar así:



Cualquier cosa que se desee comunicar al módulo WiFi debe realizarse a través de la interfaz de comandos especificada por el fabricante. Estos comandos son comandos AT y la lista de estos con sus respectivos parámetros se encuentran en la página del fabricante, espressif (<http://bbs.espressif.com/viewtopic.php?f=51&t=1022>).

Algunos comandos, por ejemplo, son:

- *AT+CWSAP="wifi","12345678",11,4* para configurar un access point,
- *AT+CIPSERVER=1,8080* crea un servidor TCP en el puerto 8080.

El módulo de software añade la capacidad de encolar comandos para enviar al módulo WiFi. De esta manera, se puede controlar para cada comando si su envío fue exitoso, si falló o si el receptor estaba ocupado y no lo pudo procesar. En función de la respuesta obtenida, automáticamente el software puede iniciar el reenvío del comando, realizando esto hasta que el comando pueda ser enviado exitosamente o hasta que se exceda un límite preestablecido de reintentos.

Al enviarse un comando exitosamente, existe la posibilidad de notificar al programador de dicho evento mediante otro *callback*.

También se pueden enviar comandos sin encolar, directamente al módulo WiFi, pero de esta manera se pierden las ventajas antes enumeradas.

Para el adecuado funcionamiento de este módulo, una tarea de *background* debe llamar a una función expuesta por el módulo, cada vez que pueda. Dicha función se encarga del envío de los comandos encolados, y tiene la necesidad de esperar por la respuesta del módulo WiFi para cada uno de ellos. Estas esperas pueden demorar varios milisegundos, por lo cual deberían realizarse bajo el contexto de una tarea NO periódica (es difícil establecer un período lo suficientemente grande que abarque todas las esperas posibles de los comandos enviados) y de la más baja prioridad (hay tareas que no pueden ser retrasadas).

Tareas

De acuerdo a las descripciones provistas para cada módulo, son necesarias las siguientes tareas, con la configuración mostrada:

- **WiFiDataReceiveTask** (módulo ESP8266): periódica con 20ms de período, prioridad más alta.
- **EncoderTask** (módulo ENCODER): periódica con período de 1 segundo en modo de control de motores, o un período especificado por el usuario en caso de encontrarse en modo de caracterización. Prioridad intermedia.
- **BackgroundTask** (main): no periódica y de más baja prioridad que todas las demás.

En el caso de que se deseen asignar valores de prioridad, la prioridad más alta de las tres puede ser 20, la intermedia 10, y la menor 5, ya que la especificación del RTOS indica que mientras más alto sea el valor de prioridad, mayor será la prioridad de la tarea.

También en el programa principal se llama automáticamente a la tarea InitTask que se encarga de inicializar todos los módulos y activar la BackgroundTask.

Programa principal

El programa principal contiene la InitTask que cumple con las funciones antes nombradas. Además, registra los *callbacks* (asocia funciones suyas para que sean llamadas por otros módulos ante la ocurrencia de eventos) necesarios para cumplir con su cometido.

Del módulo ENCODER registra el callback que notifica el tiempo transcurrido, entonces, si se encuentra en modo de control de motores, esta notificación será cada un segundo, y cada vez que se realice se debe mandar al usuario los datos medidos por los encoders. Si el modo es el de caracterización, la notificación se realizará cada vez que transcurra la cantidad de milisegundos especificados en el comando CARACTERIZAR.

Del módulo ESP8266, el principal callback registrado corresponde al que notifica la recepción de datos en la sesión TCP. En la función que atiende la notificación se hace uso de parsers para detectar los comandos de usuario, y realizar las acciones acordes.

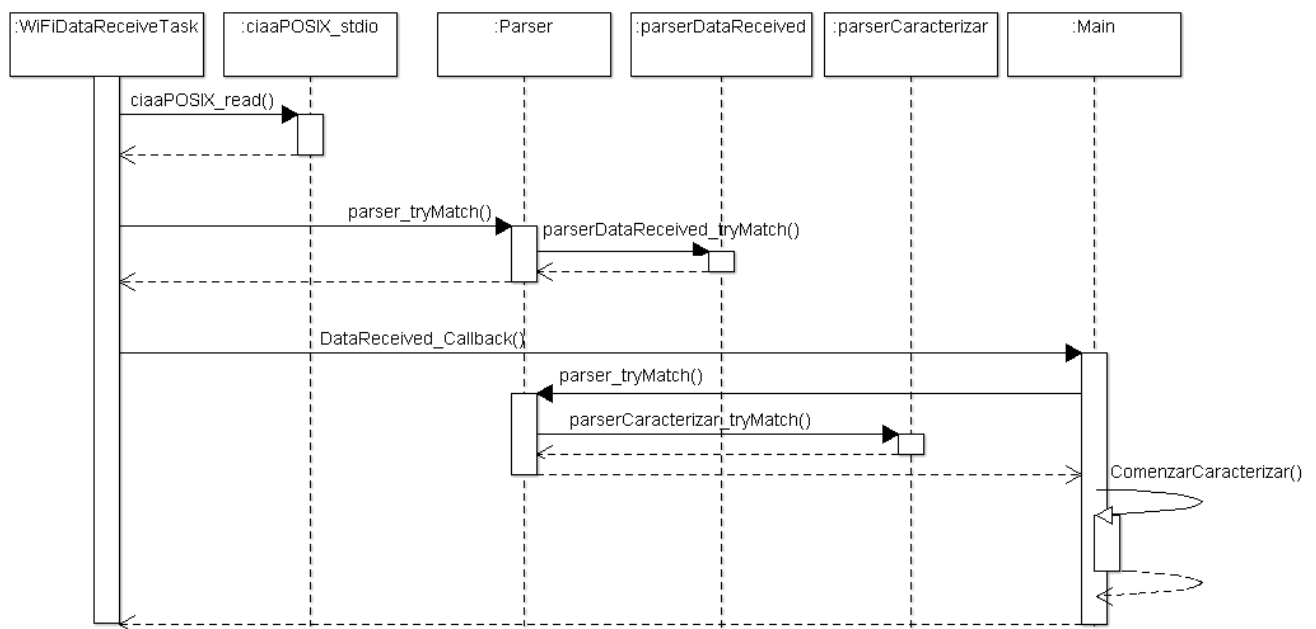
Otro callback registrado de dicho módulo es aquel que notifica el cambio de estado de alguna conexión. Aquí se puede detectar si el usuario que se encuentra controlando o caracterizando los motores se desconecta. Si esto ocurre, es necesario apagar los motores y volver a un estado de espera de comandos.

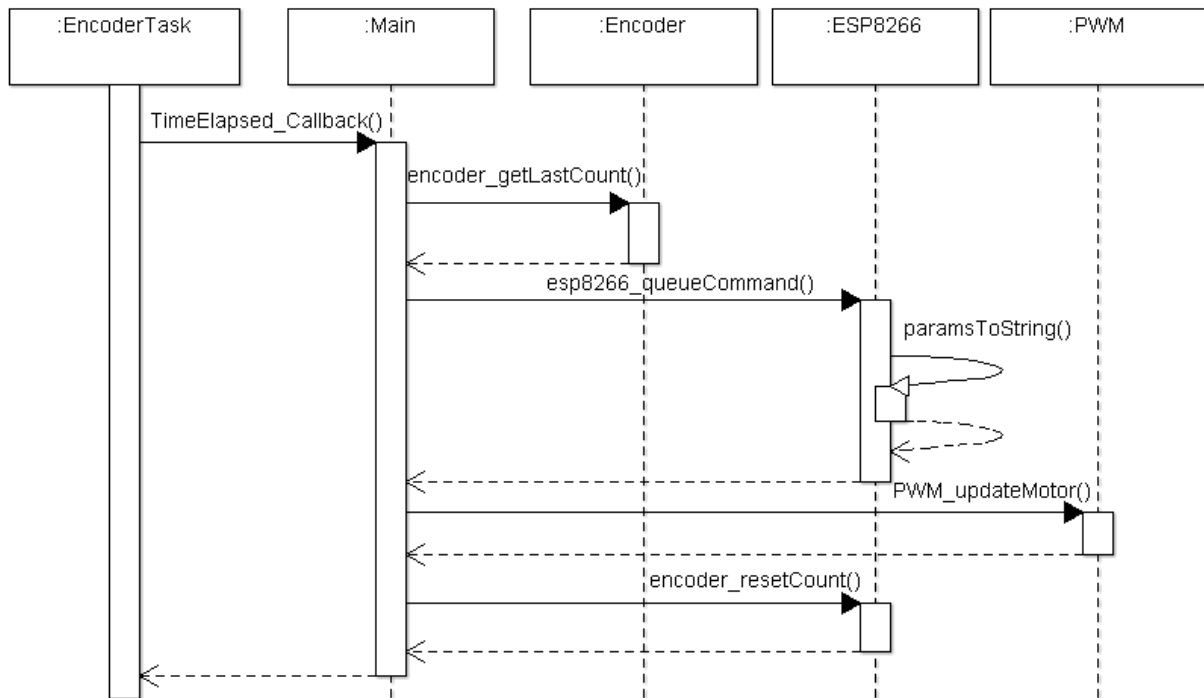
Para ejemplificar un poco el funcionamiento del software, los siguientes diagramas de secuencia muestran la secuencia de llamados para determinadas acciones a llevar a cabo.

El primero, muestra el accionar de la tarea **WiFiDataReceiveTask**, suponiendo que se leyeron caracteres que indican la recepción de un mensaje por TCP, específicamente el comando CARACTERIZAR. El módulo PARSER presenta una interfaz unificada para todos los parsers, e internamente, llamará a la función definida para el parser específico:

`parser_tryMatch()` es la interfaz unificada, y `parserDataReceived_tryMatch()` es la función asociada a un parser particular, que será llamada por la interfaz.

El segundo diagrama, muestra el funcionamiento de la tarea **EncoderTask**. El modo actual del programa es el Modo Caracterizar, por lo cual se ha registrado un callback para `TimeElapsed` acorde a ese modo, ya que, cuando pase el tiempo de conteo de interrupciones (especificado en el comando CARACTERIZAR) hay que enviar el comando MOTOR, y además incrementar el ciclo de trabajo del motor objetivo de la caracterización.





2.2. Aplicación Android.

El aplicativo se construyó utilizando el entorno de desarrollo integrado oficial de Google®, el Android Studio IDE®, con el soporte del lenguaje de programación Java y con la utilización de bibliotecas de funciones específicas como la Net Library para la utilización de sockets y conexiones TCP, y la I/O Library para el manejo de entradas y salidas y el procesamiento de los buffers de información, producto de la interacción de la aplicación con la EDU-CIAA.

La aplicación, construida bajo el paradigma de programación orientado a objetos, presenta clases o módulos bien definidos según su función, donde básicamente existen dos tipos: funciones visuales y funciones de comportamiento.

En primer instancia, la aplicación se divide visualmente en tres partes fundamentales: la vista de configuración y gestión de conexión, la vista de interacción con la cual se envían y reciben datos en comunicación con la EDU-CIAA, y la vista de calibración, con la cual se establece una comunicación con la placa para la generación de las curvas de RPM acordes a las características de cada motor. Para su implementación, se definieron tres tipos de clases, una para cada una de las vistas, donde se definen las características gráficas de cada una, además de configurarse los eventos que dispara la aplicación para cada interacción que tiene con el usuario. Cabe mencionar que aquellas características generales a las tres vistas se extrajeron y modularizaron en un componente general.

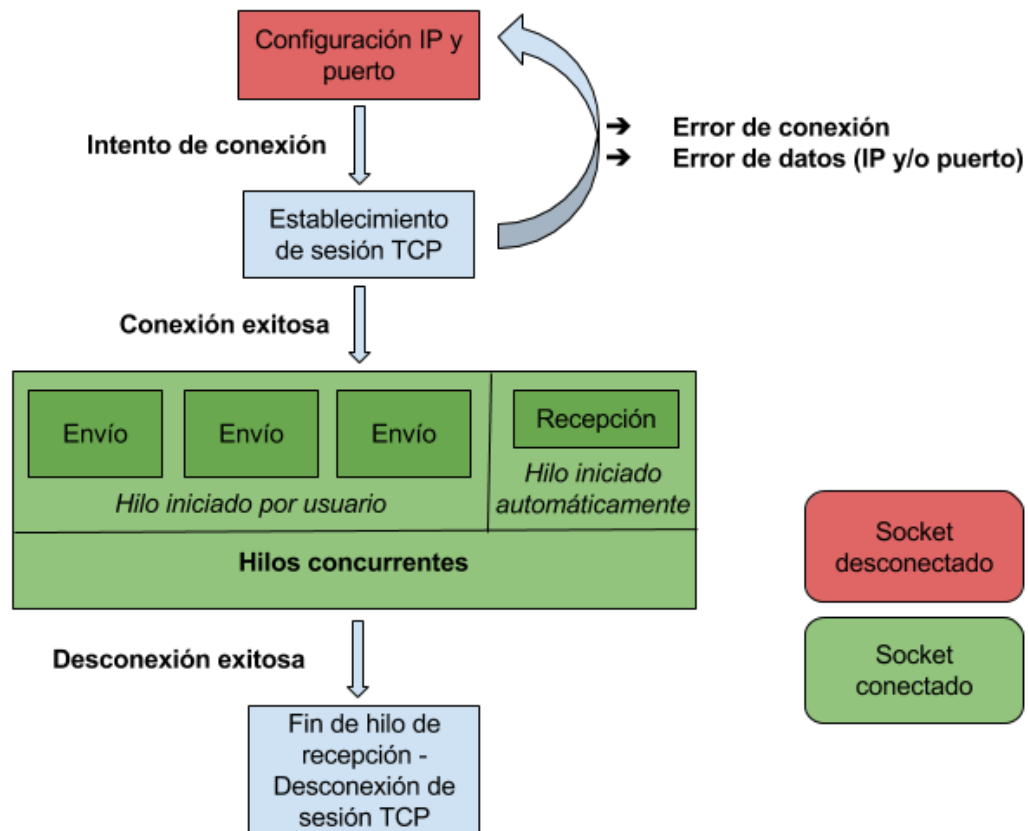
Por otro lado, la aplicación se divide en cuanto al comportamiento en una serie de clases de funcionalidades y estructuras generales de almacenamiento de información. Destacamos las responsabilidades y alcances de las clases:

- TcpAsyncSend: Clase cuya responsabilidad es la de efectuar, controlar y regular el envío de datos desde la aplicación hacia la EDU-CIAA. Define el comportamiento del flujo de datos y el manejo de excepciones, y trabaja concurrentemente con el proceso principal de la aplicación, de forma tal de que se genera un hilo por cada mensaje que se envía. Su funcionalidad se dispara desde la vista de interacción y su ciclo de vida comprende desde el inicio del envío de los datos hasta la confirmación de correcta emisión o disparo de excepción de error de transmisión.
- TcpAsyncReceive: Clase cuya responsabilidad es la de efectuar, controlar y regular la recepción de datos desde la EDU-CIAA hacia la aplicación. Funciona concurrentemente con el proceso de la aplicación, como un único hilo que se dispara una vez establecida la conexión del socket y cuyo ciclo de vida se extiende hasta la desconexión del mismo. Básicamente, procesa todos los datos recibidos decodificados como caracteres e interpreta los mensajes

según una estructura estandarizada gracias a la clase `TcpMessagesManager`. Se utiliza para la retroalimentación con la EDU-CIAA, a través de la recepción de los valores de velocidad de los motores en RPM, información que se muestra en la vista de interacción de la aplicación. Además, es usada para la decodificación de los mensajes de calibración, construyendo las curvas de comportamiento PWM-RPM de los motores, almacenándolas en la clase `CalibrationData`.






























- `TcpMessagesManager`: Clase cuya responsabilidad es la de decodificar los mensajes recibidos en la conexión TCP. Es utilizada únicamente por la clase `TcpAsyncReceive`, para desligarse de la responsabilidad del procesamiento de strings.
- `CalibrationData`: Clase cuya responsabilidad es la de almacenar toda la información relacionada a la calibración de los motores: valores de cantidad de ranuras de disco de los encoders y de tiempo (en milisegundos) durante el cual se contarán interrupciones. Además, almacena en una matriz las dos curvas de relación PWM-RPM de los motores, las cuales sirven para determinar el correcto funcionamiento del joystick de la vista de interacción.
- `TcpSocketData` y `TcpSocketManager`: Dos clases íntimamente relacionadas que tiene como responsabilidad la gestión del establecimiento, flujo y desconexión de las sesiones TCP. *`TcpSocketData`* define una estructura global para el almacenamiento y la utilización de la información del socket de conexión: el objeto `Socket` de Java, los valores de IP y puerto para la conexión. Por otro lado, *`TcpSocketManager`* define los métodos a utilizar para la utilización del socket y su configuración provista por *`TcpSocketData`*. Entre los métodos encontramos la conexión con el socket, la desconexión del socket y la implementación a bajo nivel del envío y recepción de datos a través del socket, métodos de los que se valen *`TcpAsyncSend`* y *`TcpAsyncReceive`*, respectivamente.

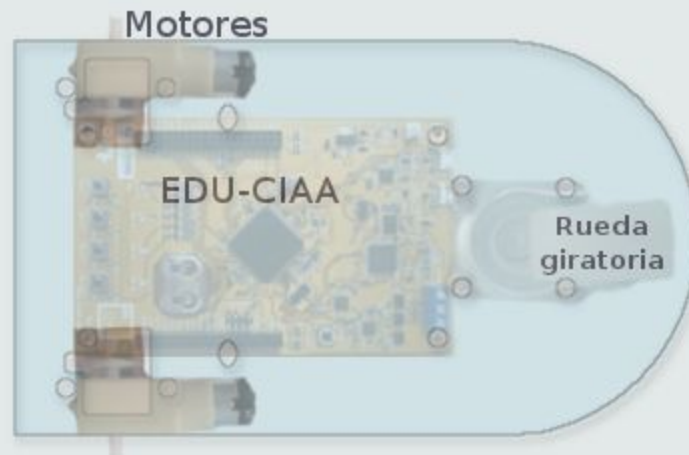
A continuación, se muestra un diagrama del ciclo de vida de una sesión TCP establecida entre la aplicación y la EDU-CIAA:



Por otra parte, el código se organizó en paquetes específicos para cada función de las clases que contiene. A continuación, se muestra un resumen de su jerarquía:

- Calibration: Contiene a la clase pertinente a la calibración de los motores y al almacenamiento de su información.
- Fragments: Contiene a las clases que describen cada apartado visual de la aplicación, denominados "fragments".
- GeneralGUI: Contiene a las clases que describen el comportamiento visual general a todos los fragments de la aplicación. Indica, además, la forma en la que se debe migrar de un fragment a otro correctamente.
- TCP: Contiene a las clases propias de la administración de las conexiones TCP, así como también de aquellas que gestionan el envío y recepción de mensajes.
- TcpMessagesManagement: Contiene a la clase pertinente a la decodificación y procesamiento de los mensajes recibidos por la aplicación vía TCP.

- ▼  Calibration
 -   CalibrationData
- ▼  Fragments
 -   BehaviourFragment
 -   CalibrationFragment
 -   ConfigurationFragment
- ▼  GeneralGUI
 -   MainActivity
 -   MyAdapter
 -   ToastManager
- ▼  TCP
 -   TcpAsyncReceive
 -   TcpAsyncSend
 -   TcpSocketData
 -   TcpSocketManager
- ▼  TcpMessagesManagement
 -   TcpMessagesManager



Para asegurar los motores al chasis fue necesaria la construcción de dos agarraderas metálicas ya que los mismos tienen perforaciones de agarre a los lados. Dichas plaquitas metálicas se realizaron de aluminio debido a la maleabilidad de este metal.

La versión final del trabajo se puede observar en las fotografías presentadas en la última sección del presente informe.

6. Conclusiones.

Consideramos que el desarrollo de este proyecto fue de gran importancia, ya que es la primera vez que nos vimos enfrentados a realizar un diseño conjunto de hardware y software con este nivel de complejidad. Fue una práctica que no podíamos dejar de realizar en nuestra formación académica. En algún punto, se trató de la integración de gran parte de los conceptos aprendidos y trabajados hasta el momento en la carrera.

Al encarar el desarrollo del proyecto se buscó que cada uno de los integrantes del grupo trabajasen en todos los aspectos constitutivos del mismo. Para eso se pensó un cronograma de trabajo inicial, el cual se encuentra dividido en tres partes principales: *desarrollo de hardware y mecánica de las ruedas, desarrollo de software y revisión integral.*

A medida que el proyecto fue evolucionando, la línea temporal que dividía a cada una de estas partes constitutivas del trabajo se tornó difusa, ya que se comenzó a diseñar el hardware a medida del software y viceversa. Ciertos aspectos de realimentación visual al usuario, que corresponden al poncho, fueron primero pensadas mientras se desarrollaba el software, y por lo tanto se hizo necesario modificar el diseño original.

Este desarrollo en paralelo del hardware y software resultó en que los integrantes del grupo nos comencemos a centrar en aspectos particulares del proyecto. Al mismo tiempo, generó un cierto orden en el modo de trabajo, ya que nos logramos interiorizar en metodologías y buenas prácticas de desarrollo, tanto de hardware como de software.

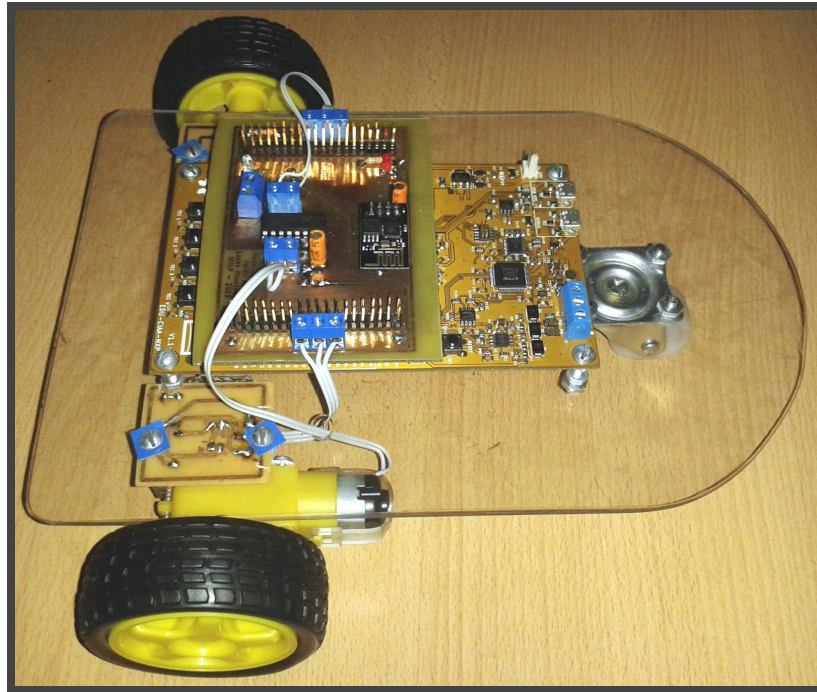
Otro aspecto que contribuyó a que el desarrollo del hardware se retrasara fue la elección de los componentes a utilizar. Por ejemplo, en un primer momento, se armaron los encoders utilizando un fototransistor y un fotodiodo. Si bien la funcionalidad no cambió, fue necesario rediseñar el PCB para que este utilice el TCST2103 por el cual se reemplazó al fotodiodo y fototransistor.

Por su parte, el chasis, que se encontraba dentro de las primeras tareas a realizar, fue uno de los desarrollos más tardíos del proyecto, ya que en un principio no contábamos con dos motores de DC del mismo tipo, ni las cajas reductoras correspondientes. Tampoco se pensó que se terminaría usando un power bank para la alimentación de la EDU-CIAA y el poncho.

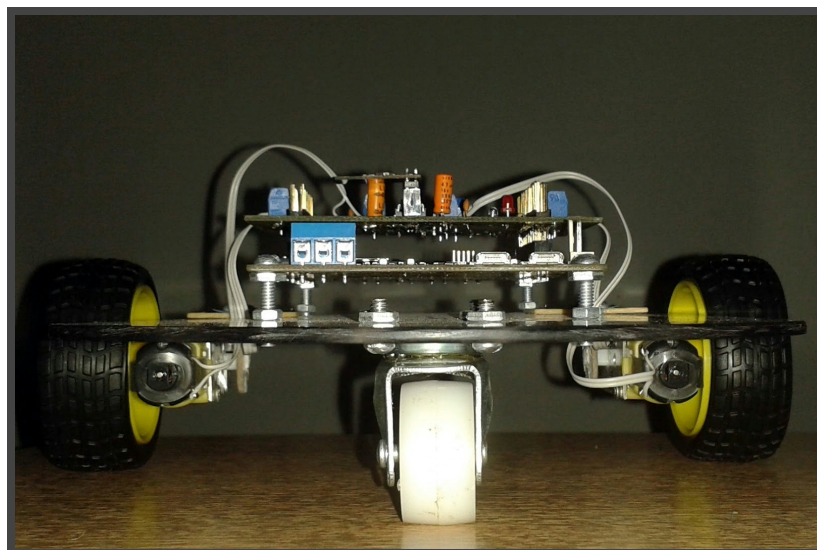
En resumen, fue un proyecto que nos permitió aplicar la mayoría de los conceptos vistos en estos años de carrera y apreciar los tipos de productos que vamos a poder desarrollar en nuestra futura vida profesional. Sirvió también para trabajar en equipo y para aprender a organizar el diseño y desarrollo de un proyecto de esta complejidad.

7. Galería.

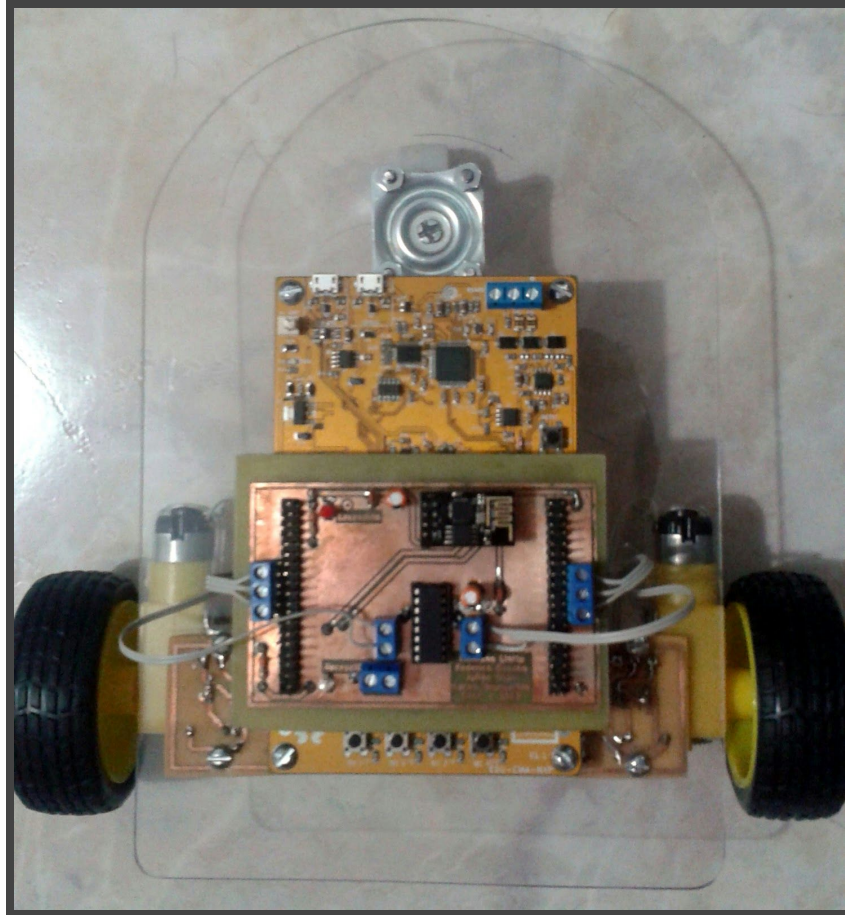
Se adjuntan a continuación imágenes del resultado final del cuerpo del robot:



Vista auxiliar del robot.



Vista frontal del robot.



Vista superior del robot.

8. Anexos.

1. División de tareas:

A lo largo del proyecto, no existió una división de tareas estricta. La mayor parte del trabajo se realizó en conjunto y todos los participantes del grupo trabajamos en prácticamente todas las tareas.

Sin embargo, cada uno se enfocó de manera más directa a uno de cuatro tópicos generales: software de EDU-CIAA, software de Android, esquemático y PCB, chasis y encoders.

Software EDU-CIAA	Federico Bouche
Software de Android	Lucas Hourquebie
Esquemático y PCB	Julián Ailán
Chasis y Encoders	Emiliano Liotta

2. Código OIL de OSEK-OS con las tareas empleadas

```
OSEK OSEK {

OS  MotorControlOS {
    STATUS = EXTENDED;
    ERRORHOOK = TRUE;
    PRETASKHOOK = FALSE;
    POSTTASKHOOK = FALSE;
    STARTUPHOOK = FALSE;
    SHUTDOWNHOOK = FALSE;
    USERESSCHEDULER = FALSE;
    MEMMAP = FALSE;
};

TASK InitTask {
    PRIORITY = 1;
    ACTIVATION = 1;
    AUTOSTART = TRUE {
        APPMODE = AppModel1;
    }
    STACK = 512;
    TYPE = BASIC;
    SCHEDULE = NON;
```

```

    RESOURCE = POSIXR;
    EVENT = POSIXE;
}

TASK BackgroundTask {
    PRIORITY = 5;
    ACTIVATION = 1;
    STACK = 512;
    TYPE = EXTENDED;
    SCHEDULE = FULL;
    EVENT = POSIXE;
    RESOURCE = POSIXR;
}

TASK EncoderTask {
    PRIORITY = 10;
    ACTIVATION = 1;
    STACK = 1024;
    TYPE = BASIC;
    SCHEDULE = FULL;
    RESOURCE = POSIXR;
}

TASK WiFiDataReceiveTask {
    PRIORITY = 20;
    ACTIVATION = 1;
    STACK = 1024;
    TYPE = BASIC;
    SCHEDULE = FULL;
    RESOURCE = POSIXR;
}

ALARM ActivateEncoderTask {
    COUNTER = SoftwareCounter;
    ACTION = ACTIVATETASK {
        TASK = EncoderTask;
    }
}

ALARM ActivateWiFiDataReceiveTask {
    COUNTER = SoftwareCounter;
    ACTION = ACTIVATETASK {
        TASK = WiFiDataReceiveTask;
    }
}

RESOURCE = POSIXR;

```

```

EVENT = POSIXE;

APPMODE = AppModel1;

COUNTER HardwareCounter {
    MAXALLOWEDVALUE = 100;
    TICKSPERBASE = 1;
    MINCYCLE = 1;
    TYPE = HARDWARE;
    COUNTER = HWCOUNTER0;
};

COUNTER SoftwareCounter {
    MAXALLOWEDVALUE = 10000;
    TICKSPERBASE = 1;
    MINCYCLE = 1;
    TYPE = SOFTWARE;
};

ALARM IncrementsSWCounter {
    COUNTER = HardwareCounter;
    ACTION = INCREMENT {
        COUNTER = SoftwareCounter;
    };
    AUTOSTART = TRUE {
        APPMODE = AppModel1;
        ALARMTIME = 1;
        CYCLETIME = 1;
    };
};

ISR UART0_IRQHandler {
    INTERRUPT = UART0;
    CATEGORY = 2;
    PRIORITY = 0;
};

ISR UART2_IRQHandler {
    INTERRUPT = UART2;
    CATEGORY = 2;
    PRIORITY = 0;
};

ISR UART3_IRQHandler {
    INTERRUPT = UART3;
    CATEGORY = 2;
    PRIORITY = 0;
};

```

```

ISR GPIO0_IRQHandler {
    INTERRUPT = GPIO0;
    CATEGORY = 1;
    PRIORITY = 0;
};

ISR GPIO1_IRQHandler {
    INTERRUPT = GPIO1;
    CATEGORY = 1;
    PRIORITY = 0;
};

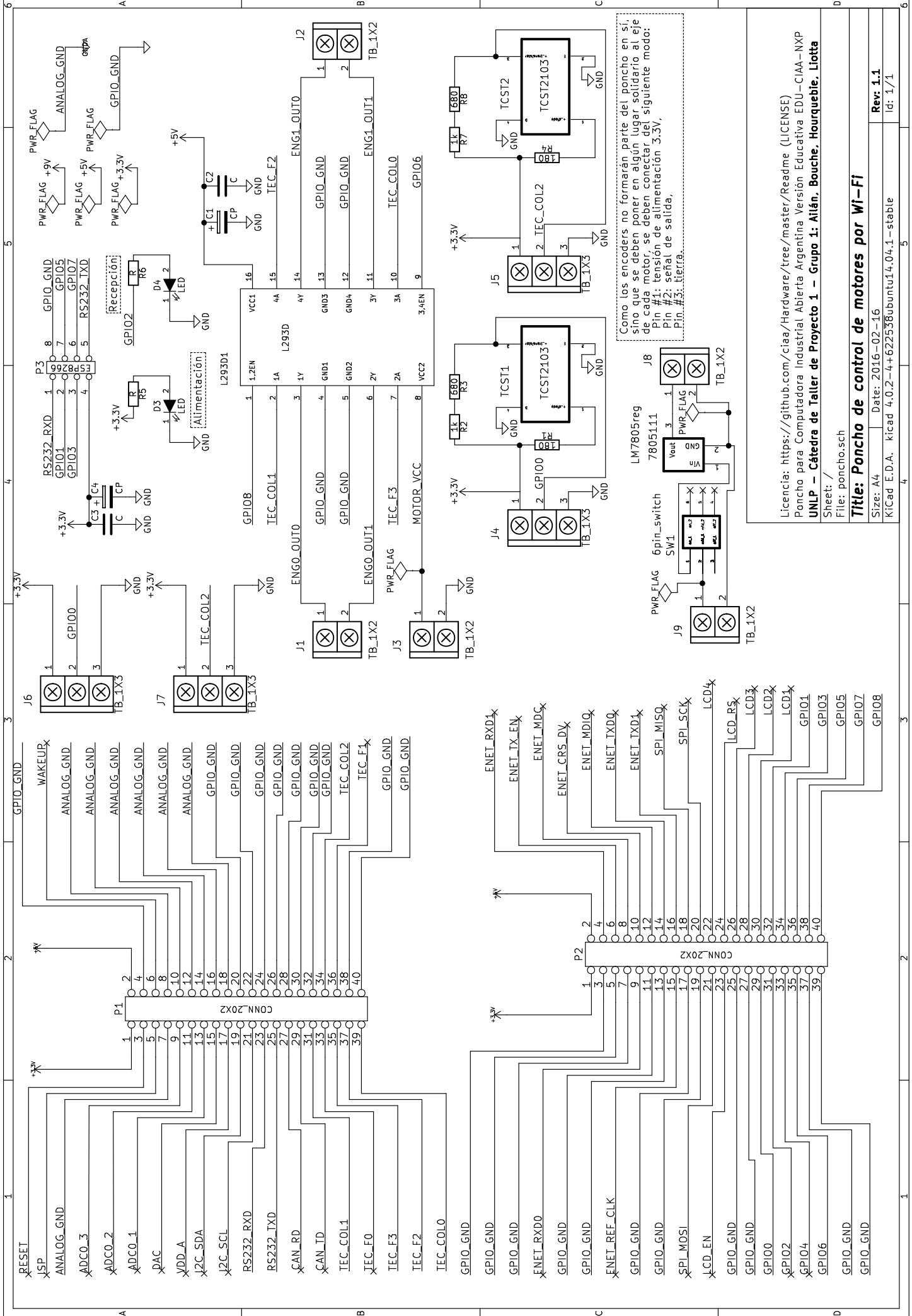
ISR RIT_IRQHandler {
    INTERRUPT = RIT;
    CATEGORY = 1;
    PRIORITY = 0;
};

};

```

3. Esquemático y PCB.

A continuación se muestran el diagrama esquemático y el PCB que describen al poncho desarrollado. Del PCB se incluyen las caras superior e inferior correspondientes a las dos caras del poncho:

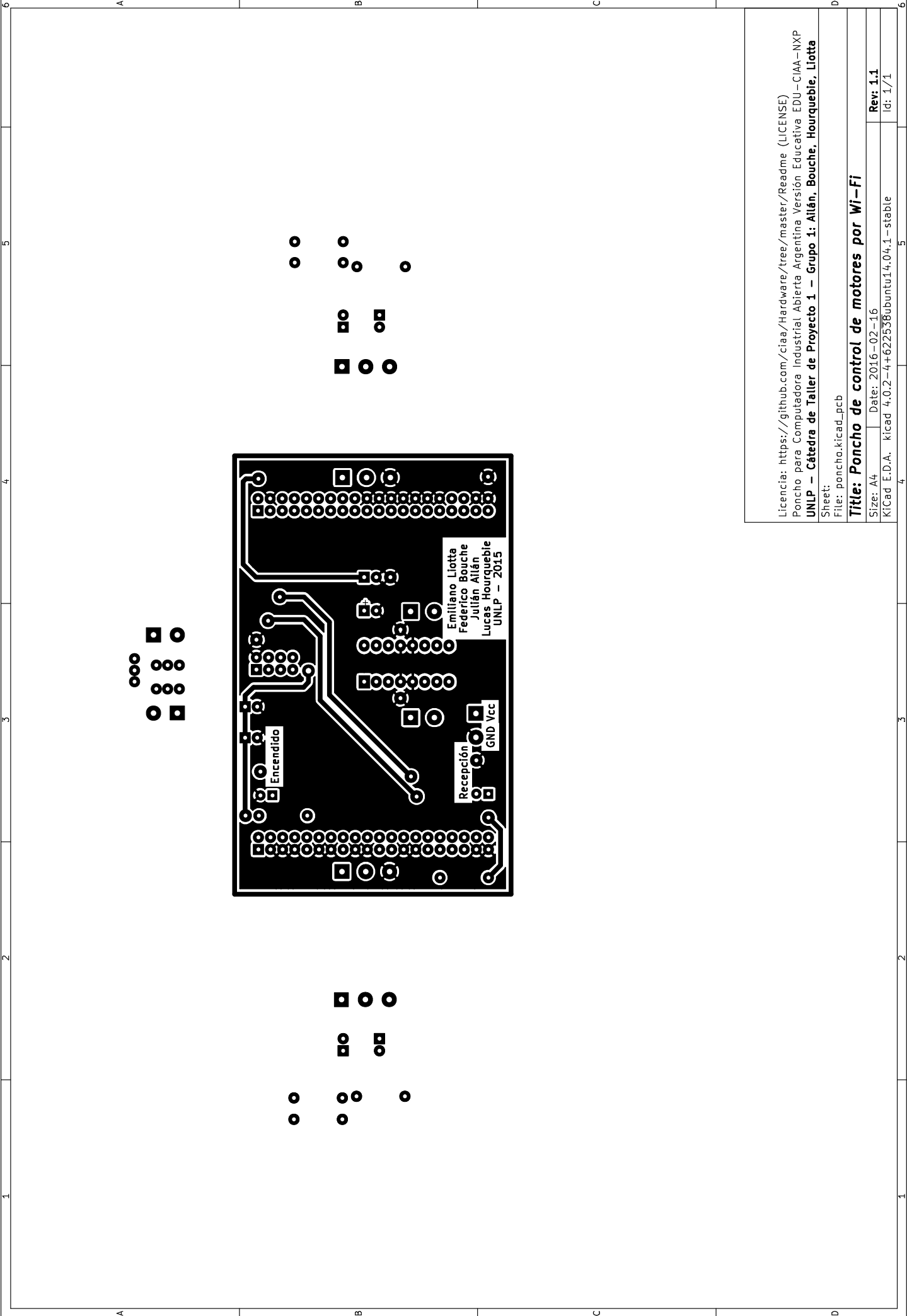


Licencia: <https://github.com/ciaa/hardware/tree/master/Readme> (LICENSE)
PONCHO para Computadora Industrial Abierta Argentina Versión Educativa EDU-CIAA-NXP
UNLP – Cátedra de Taller de Proyecto 1 – Grupo 1: Allán, Bouche, Hourquebie, Liotta

Sheet: /
File: poncho.sch

Title: Poncho de control de motores por Wi-Fi

Size: A4
Date: 2016-02-16
KiCad E.D.A. kicad 4.0.2-4+622538ubuntu14.04.1-stable
Rev: 1.1
Id: 1/1



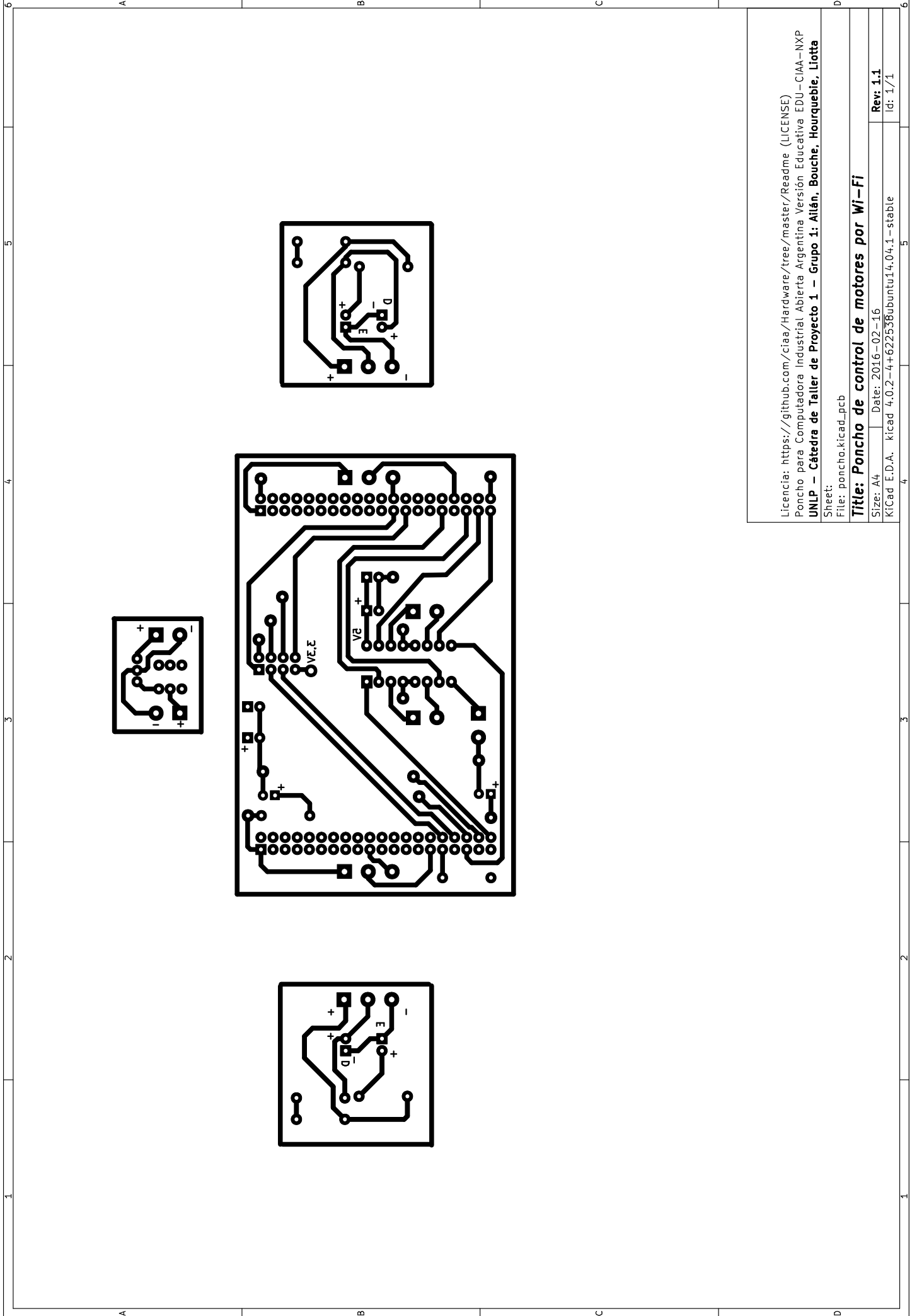
Licencia: <https://github.com/ciaa/Hardware/tree/master/Readme> (LICENSE)
Poncho para Computadora Industrial Abierta Versión Educativa EDU - CIAA - NXP
UNLP - Cátedra de Taller de Proyecto 1 - Grupo 1: Allán, Bouche, Hourquebie, Liotta

Sheet:
File: poncho.kicad_pcb

Title: Poncho de control de motores por Wi-Fi

Size: A4 | Date: 2016-02-16 | **Rev: 1.1**

KiCad E.D.A. | kicad 4.0.2-4+622538ubuntu14.04.1 - stable | Id: 1/1



Licencia: <https://github.com/ciaa/Hardware/tree/master/Readme> (LICENSE)
Poncho para Computadora Industrial Abierta Versión Educativa EDU – CIAA – NXP
UNLP – Cátedra de Taller de Proyecto 1 – Grupo 1: Allán, Bouche, Hourquebie, Liotta

Sheet:
File: poncho.kicad_pcb

Title: Poncho de control de motores por Wi-Fi

Size: A4 | Date: 2016-02-16 | Rev: 1.1

KiCad E.D.A. kicad 4.0.2-4+622538ubuntu14.04.1 -stable | Id: 1/1