

UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
CARRERA DE ESPECIALIZACIÓN EN SISTEMAS
EMBEBIDOS



MEMORIA DEL TRABAJO FINAL

**CIAABOT: Robótica educativa abierta
basada en la CIAA**

Autor:
Ing. Leandro Lanzieri Rodríguez

Director:
Ing. Eric Pernía

Jurados:
Dr. Ing. Pablo Gómez (FIUBA)
Esp. Ing. Ernesto Gigliotti (UTN FRA)
Esp. Ing. Patricio Bos (FIUBA)

Este trabajo fue realizado en las Ciudad Autónoma de Buenos Aires entre febrero de 2017 y diciembre de 2017.

Resumen

En la presente memoria se describe el desarrollo de CIAABOT, una plataforma libre de robótica educativa que funciona sobre la CIAA. El objetivo es facilitar la enseñanza de materias básicas y programación en escuelas, utilizando la robótica como herramienta. Para esto se desarrolló una interfaz de programación gráfica, para reducir la dificultad la sintaxis.

Para cumplir con los objetivos planteados se aplicaron técnicas de programación como división de responsabilidades por capas, modularización y servicios.

También se utilizaron herramientas de desarrollo, entre ellas control de versiones, debug y testing. Se aplicaron a su vez conocimientos adquiridos en la especialización de diseño de PCB y gestión de proyectos.

Agradecimientos

En primer lugar agradezco a mis padres Silvano y Nelly, por apoyarme en todos mis objetivos y hacerme pensar.

A mis hermanas Camila y Sol, por estar siempre conmigo.

A Deborah, por su paciencia y apoyo en todas mis locuras, aunque implique menos tiempo juntos.

A Daniel Acerbi, director del Laboratorio Abierto de la UTN Facultad Regional Avellaneda, por su confianza y la beca que me permitió cursar esta especialización.

Finalmente, a mi director Eric Pernía, por su apoyo y entusiasmo constantes que hicieron posible este proyecto.

A todos ellos, muchas gracias.

Índice general

Resumen	III
1. Introducción General	1
1.1. Motivación	1
1.1.1. Robótica en la educación	1
1.1.2. Introducción a la programación de sistemas embebidos	2
1.2. CIAABOT: Lineamientos principales	2
1.2.1. Software y Hardware libres	2
1.2.2. Orientación a la enseñanza	3
1.2.3. Amplia documentación	3
1.2.4. Escalabilidad	3
1.3. Alcance y Objetivos planteados	4
2. Introducción Específica	5
2.1. CIAABOT: Partes componentes	5
2.1.1. Entorno de desarrollo integrado	6
2.1.2. CIAABOTS	6
2.1.3. Firmware	7
2.2. Plataforma CIAA	8
2.2.1. EDU-CIAA-NXP	8
2.3. Requerimientos	9
2.3.1. Requerimientos del sistema	9
2.3.2. Componentes del Sistema	9
2.3.3. Firmware	9
2.3.4. Procesos Finales	9
2.4. Planificación	10
2.4.1. Desglose en tareas	10
2.4.2. Activity On-node	10
2.4.3. Diagrama de Gantt	11
3. Diseño e Implementación	13
3.1. Estructura del software	13
3.1.1. Estructura de un proyecto base	13
3.1.2. Editor de código en bloques	15
3.1.3. Definición del lenguaje específico	19
3.1.4. Servicios	21
3.1.5. Compilación	23
3.2. Diseño del poncho CIAABOT G1	25
3.3. Herramientas	28
3.3.1. Git	28
3.3.2. OpenOCD	29
4. Ensayos y Resultados	31

4.1. Pruebas de sensores y actuadores	31
4.2. Workshop en el SASE	33
4.3. Experiencias en cursos CAPSE y de INET	34
4.4. Uso en un proyecto de brazo robótico	35
Bibliografía	37

Índice de figuras

2.1.	Diagrama de las partes que componen CIAABOT.	5
2.2.	Editor gráfico de CIAABOT-IDE.	7
2.3.	Placa EDU-CIAA-NXP	8
2.4.	Diagrama <i>Activity on-node</i>	11
2.5.	Diagrama de Gantt - Parte 1.	12
2.6.	Diagrama de Gantt - Parte 2.	12
3.1.	Estructura simplificada de un proyecto base.	15
3.2.	Secciones del editor de bloques.	17
3.3.	Detalle de la barra de acciones.	18
3.4.	Bloque Esperar x segundos.	18
3.5.	Bloque Leer conversor analógico a digital.	18
3.6.	Bloque Condicional.	18
3.7.	Ejemplo de código en bloques para el barrido de un servo.	19
3.8.	Distribución en capas de las tecnologías utilizadas.	21
3.9.	Interacción entre componente, plantilla y servicios en Angular.	22
3.10.	Ejemplo de rutas de búsqueda adicionales en el IDE.	25
3.11.	Recorte del esquemático que muestra los conectores disponibles.	26
3.12.	Driver A4988 de motor paso a paso.	26
3.13.	Recorte del esquemático que muestra la fuente de alimentación.	27
3.14.	Diseño del circuito impreso del poncho G1.	28
3.15.	Diagrama de un repositorio distribuido. Cada computadora posee todas las versiones.	29
4.1.	Prueba de servomotor sobre dedo 3D.	31
4.2.	Anemómetro utilizado en las pruebas.	32
4.3.	Rosa de los vientos y conexión de los leds.	33
4.4.	Maqueta vehicular utilizada en el workshop.	34
4.5.	Prótesis impresa en 3D.	35

Índice de Tablas

3.1. Definición del lenguaje CIAABOT v1.0.0	20
3.1. Definición del lenguaje CIAABOT v1.0.0	21

A mi abuela Neli.

Capítulo 1

Introducción General

En este capítulo se exponen las problemáticas observadas que originaron el desarrollo del presente trabajo, junto con una breve descripción de la plataforma propuesta para solucionarlas.

1.1. Motivación

En la constante búsqueda de mejorar la experiencia de los alumnos, docentes de los Cursos Abiertos de Programación de Sistemas Embebidos (CAPSE) realizaron evaluaciones sobre los procesos de enseñanza y sus resultados. A raíz de esto se detectó que un gran obstáculo común entre los alumnos es aprender en conjunto la lógica para resolución de problemas utilizando algoritmos, y la sintaxis de un lenguaje de programación. En función de esto se propuso una plataforma que redujera la problemática de la sintaxis.

1.1.1. Robótica en la educación

En el último tiempo se ha extendido cada vez más el uso de la robótica educativa como una herramienta particularmente útil en ámbitos escolares. Es utilizada como un sistema de enseñanza multidisciplinaria, que potencia el desarrollo de habilidades en las áreas de ciencias, tecnología, ingeniería y matemáticas. Correctamente estructurada, la robótica educativa permite incentivar el trabajo en equipo, el liderazgo, el emprendimiento y el aprendizaje a partir los errores.

La robótica en las escuelas se presenta entonces como una opción tecnológica e innovadora para afrontar la problemática de capturar el interés de los alumnos en los temas propuestos en la currícula. Esto se debe a la necesidad de involucrarse y trabajar en conjunto con sus pares para la resolución de problemas donde deben aplicar los conocimientos de las diferentes asignaturas. Al realizarse de manera entretenida los contenidos se fijan de manera más simple y natural.

Sin embargo, a la hora de aplicar estas técnicas de enseñanza se presenta una problemática. En el ambiente de las escuelas primarias el público son estudiantes jóvenes en proceso de formación, que en general no poseen conocimientos técnicos en detalle como electrónica o manejo de lenguajes de programación. Esto se traduce en una limitación y dificultad a la hora de acercarse a este tipo de tecnologías, aunque el objetivo final no sea el aprendizaje de la robótica en sí misma sino utilizarla *como herramienta*.

Las opciones actuales se presentan como plataformas que son, en su mayoría, diseños privativos, cerrados, con pocas posibilidades de modificación y en general desarrollos extranjeros importados a nuestro país. Se observa por lo tanto la necesidad de que la solución alternativa debe ser abierta y desarrollada localmente.

1.1.2. Introducción a la programación de sistemas embebidos

Los primeros pasos para la inserción en el ámbito de la programación de sistemas embebidos pueden ser difíciles para algunas personas. Esto se debe, en parte, al hecho de tener que trabajar sobre plataformas diferentes a una PC y utilizando lenguajes de programación de bajo nivel (generalmente C o assembler).

Una de las mayores dificultades a la hora de iniciarse en la programación, de sistemas embebidos o en general, es la sintaxis específica del lenguaje que debe ser utilizado. Esto se evidencia a la hora de expresar en código la idea que se tiene, es decir la lógica del programa que se está desarrollando.

Una opción que presenta una curva de aprendizaje más plana es abstraerse, por lo menos en una primera etapa, del lenguaje que se utilizará para programar más adelante. De esta manera el alumno puede concentrarse en definir la lógica de su solución propuesta al problema planteado, sin preocuparse en cómo se expresaría en código (dicha tarea queda para etapas posteriores del proceso de aprendizaje). Esta última tarea queda para etapas posteriores del proceso de aprendizaje.

1.2. CIABOT: Lineamientos principales

CIABOT se presenta como una plataforma abierta orientada a la robótica educativa, es decir utilizada como una herramienta para la educación, con el fin de suplir las necesidades identificadas en la sección 1.1. A la hora de darle forma a este proyecto se plantearon lineamientos generales que debía cumplir, más allá de los alcances a nivel técnico. A continuación se da una breve descripción y se justifica cada uno de ellos.

1.2.1. Software y Hardware libres

La *cultura libre* es una corriente o movimiento que, en contraposición a las medidas restrictivas de los derechos de autor, promueve la libertad para distribuir y modificar trabajos.

Esta ideología aplicada al software se traduce en código que se libera bajo alguna de las licencias libres (por ejemplo BSD [9], GPL [6] o MIT [8]), lo que implica ciertas libertades a la hora de utilizarlo, modificarlo y distribuirlo. En general, aunque depende de la licencia elegida, se libera el código fuente de manera gratuita (aunque esto último no es condición obligatoria) y los usuarios pueden verlo o modificarlo.

Cuando se trata de hardware, la idea de un diseño libre se traduce en la liberación bajo alguna licencia de especificaciones, diagramas esquemáticos, diseños de circuitos impresos, memorias de cálculo y cualquier otro documento accesorio de interés para el funcionamiento del sistema. Que el hardware sea libre no implica

necesariamente que se pueda adquirir físicamente de manera gratuita, ya que el la fabricación y armado suponen un costo.

La ventaja principal por la que se optó que el proyecto sea libre, tanto en hardware como en software, es que los interesados pueden descargar esquemáticos o código fuente respectivamente y armar sus circuitos o compilar sus programas, junto con cualquier modificación que crean conveniente. Particularmente el software de CIAABOT se liberó con la licencia GNU GPLv3. Esto implica, entre otras cosas, que cualquier modificación o utilización del mismo debe liberarse bajo la misma licencia.

1.2.2. Orientación a la enseñanza

Como se describió en la sección 1.1 la robótica como herramienta para promover el aprendizaje de materias básicas, lógica y programación está ganando terreno. Es por esto que CIAABOT se desarrolló con el objetivo de ser utilizado para introducirse a la programación de sistemas embebidos de manera fácil y didáctica, permitiendo de esta manera a docentes, alumnos de escuelas o entusiastas que deseen iniciarse en la programación de embebidos, un posibilidad simplificada sin la complicación de encontrarse con una complicada sintaxis.

Teniendo en mente este lineamiento, se analizó el armado del DSL (*Domain-Specific Language*, Lenguaje de Dominio Específico) para que fuera natural su uso, como si es estuviera escribiendo el algoritmo con palabras.

1.2.3. Amplia documentación

La documentación a la hora de usar una plataforma, ya sea de software o hardware es esencial. Es una información fehaciente que tienen los usuarios para poder aprender a utilizarla o resolver los posibles problemas que se presenten. CIAABOT apuesta para ampliar la red de personas que la utilizan, a la formación de una comunidad de usuarios que aporte experiencias, conocimiento e incluso mejoras a la plataforma.

Las plataformas abiertas más conocidas lo son porque son fáciles de utilizar y se pueden conseguir resultados visibles de manera inmediata. Todo esto sería difícil de conseguir si la documentación fuera escasa o inexistente, como ocurre con algunos proyectos.

Al seguir esta idea de poner al usuario en el centro, CIAABOT posee documentación de instalación, ejemplos y preguntas frecuentes que crece con cada versión [5]. Además de basarse en otros proyectos que presentan a su vez extensa documentación y ejemplos de uso (CIAA Firmware v2 y sAPI).

1.2.4. Escalabilidad

CIAABOT está basada en la plataforma de CIAA (Computadora Industrial Abierta Argentina) [11]. Actualmente funciona sobre la placa EDU-CIAA-NXP, una versión educativa y reducida de la CIAA-NXP pensada para la educación y la

formación en sistemas embebidos. El proyecto CIAA presenta una gran dinámica, con la liberación de herramientas nuevas, actualizaciones y hasta placas nuevas.

Un ejemplo de esto es la reciente placa CIAA Z3R0. Esta placa utiliza el microcontrolador EFM32HG de Silicon Labs, y tiene como objetivo un muy bajo consumo de energía, simplicidad y pequeño tamaño. Está pensada para aplicaciones en robótica e IoT (*Internet of Things*, Internet de las cosas).

Debido a esto, se pensó CIAABOT como una plataforma que pueda adaptarse a estos cambios. Internamente las placas soportadas del ecosistema CIAA tienen una definición de capacidades y pines disponibles, donde se pueden adicionar placas nuevas a medida que aparezcan.

Además, como se explica en la sección 2.1, CIAABOT pretende tener varios modelos de robots que funcionen con la plataforma, y que puedan basarse en diferentes placas del proyecto CIAA.

1.3. Alcance y Objetivos planteados

El objetivo principal fue desarrollar el entorno de desarrollo y programación para CIAABOT, junto con firmware específico que se requiera, basándose en bibliotecas existentes para la CIAA (por ejemplo sAPI). Además, se planteó la implementación y uso de lo anterior en un robot o maqueta existente adaptado para utilizar la EDU-CIAA-NXP. Esto se hizo para demostrar las funcionalidades que se proveen. Para esto también se consideró el armado de todo el hardware adicional necesario.

Se buscó que el proyecto sea utilizado para la enseñanza de programación de sistemas embebidos, apuntando principalmente a la robótica.

En el contexto del trabajo de especialización, se restringió el alcance por una cuestión de tiempo. Se resolvió que el desarrollo incluya la aplicación de escritorio, que permita la programación gráfica de los algoritmos, utilizando bloques. Además debe traducir estos bloques a lenguaje C. Por último, se planteó que se utilizará el protocolo Firmata para un monitoreo en tiempo real del estado de la placa cuando se desarrolla conectado a la PC.

No se incluyó en el desarrollo de un modelo específico de CIAABOT.

Capítulo 2

Introducción Específica

En este capítulo se presenta CIAABOT con más detalle, incluyendo especificaciones técnicas sobre la plataforma utilizada, se establecen los requerimientos planteados y la planificación para el desarrollo del trabajo.

2.1. CIAABOT: Partes componentes

Se planteó que para lograr los objetivos propuestos, la plataforma debería estar formada por tres partes fundamentales, que funcionen complementadas para armar un ecosistema CIAABOT, como se esquematiza en la figura 2.1. A continuación se describirá cada una de estas partes.

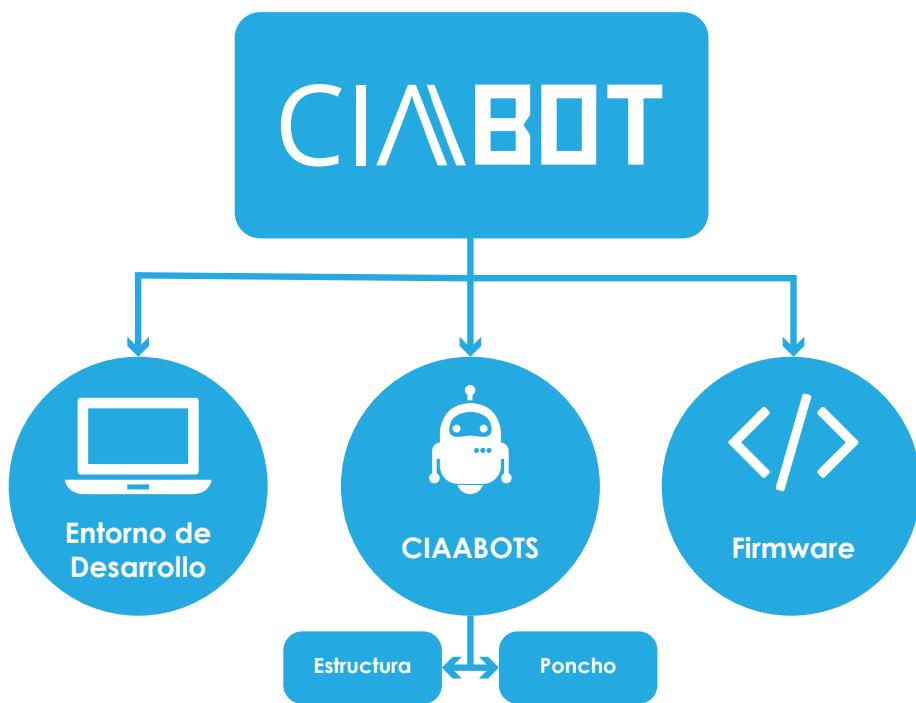


FIGURA 2.1: Diagrama de las partes que componen CIAABOT.

2.1.1. Entorno de desarrollo integrado

Un IDE (*Integrated Development Environment*, Entorno de Desarrollo Integrado) es un software que incluye varias herramientas para asistir a los desarrolladores con su trabajo. Generalmente incluye algún editor para el código, herramientas de construcción o compilación automáticas, programación de la plataforma y depuración.

El CIAABOT-IDE es el centro del trabajo de los usuarios, donde plasman sus ideas y soluciones a problemas. Esto lo hacen por medio de un editor de código en el que utilizan bloques predefinidos y no texto, como se observa en la figura 2.2. Arrastrando e interconectando los mismos logran definir la lógica para su programa.

Una opción interesante para remarcar es la salida de código en C. Cada vez que se modifica el diagrama de bloques, el código en C producido se modifica en tiempo real. Esto es muy útil para aquellos que pretenden aprender a programar en C, ya que ven de manera directa cómo afecta cada bloque a las líneas de código y pueden apreciar las equivalencias.

La plataforma también permite compilar y luego descargar el código generado en la placa solamente conectándola por USB.

Una vez que se finalizó el trabajo se puede guardar el proyecto en un archivo *.cbp* que contiene la estructura de bloques en la que se estuvo trabajando, junto con las opciones adicionales del proyecto. Este archivo puede ser compartido y abierto en otra PC para ser reutilizado o modificado y guardado nuevamente.

Otra de las opciones que fue publicada en la última versión de CIAABOT-IDE (v0.0.8) es la de seleccionar los directorios de búsqueda para el toolchain. Las aplicaciones necesarias para todos los sistemas operativos son dos: El compilador arm-none-eabi-gcc y el debugger openOCD. Adicionalmente, en Windows se requieren otros ejecutables como make, para poder compilar y descargar el programa.

2.1.2. CIAABOTS

Los robots que funcionen bajo esta plataforma serán denominados CIAABOTS. Siguiendo la idea original de que CIAABOT sea utilizada en escuelas o por entusiastas, se planteó que estos robots estén diseñados estructuralmente para poder ser impresos con impresoras 3D.

De esta manera es posible enviar los archivos para imprimirlas en cualquier escuela que cuente con este sistema de impresión para replicarlo. Sólo sería necesario tener la placa CIAA correspondiente al modelo, armar el poncho (de diseño abierto) y adquirir en algún lugar de conveniencia los actuadores o sensores particulares del CIAABOT (motores DC, sensores IR, ultrasonido, etc.).

El diseño del poncho para el primer modelo de CIAABOT, el G1 se puede encontrar en github [2], realizado con KiCad, como todos los diseños relacionados al proyecto CIAA.

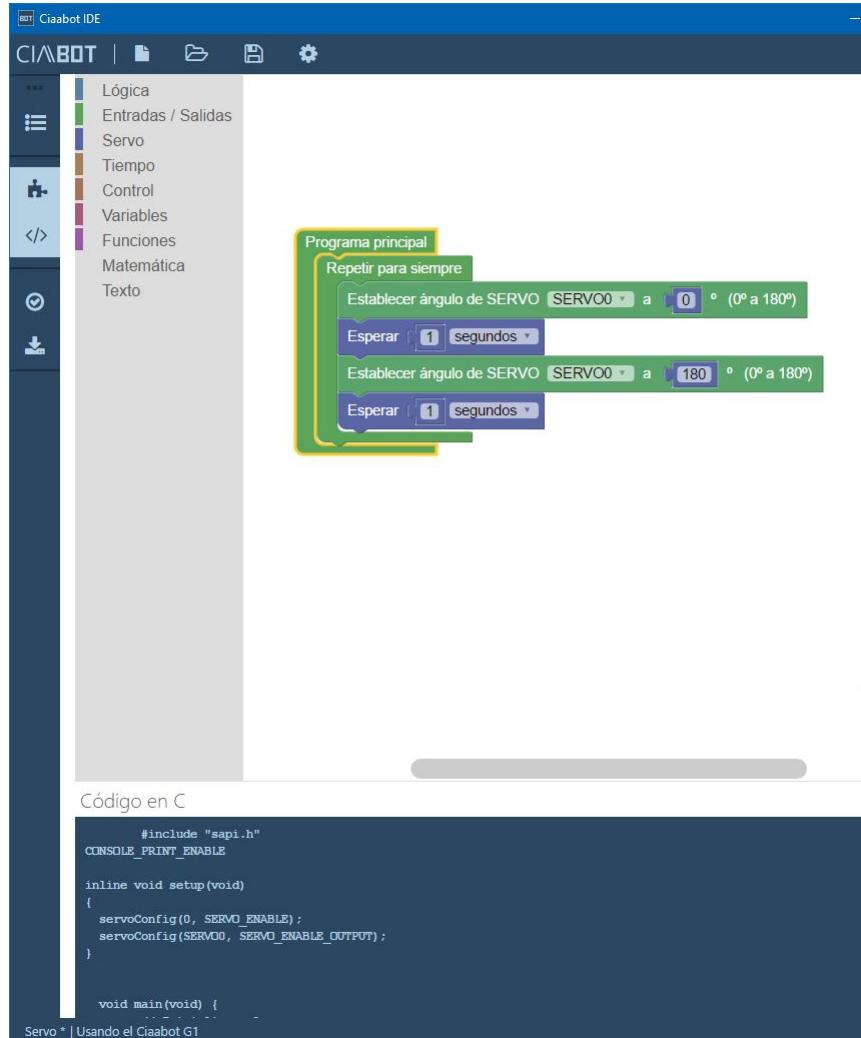


FIGURA 2.2: Editor gráfico de CIAABOT-IDE.

2.1.3. Firmware

CIAABOT utiliza como base para su firmware dos proyectos principales del ecosistema CIAA: Firmware v2 [1] y sAPI [13]. En conjunto proveen una forma simplificada de programar las placas CIAA.

La sAPI es una HAL (*Hardware Abstraction Layer*, Capa de Abstracción de Hardware) para los microcontroladores, que permite acceder de manera simple a varios de los diferentes periféricos.

Juntando varias funciones de la sAPI es posible armar funciones más complejas de mayor nivel para manejar características más específicas de cada modelo de CIAABOT.

Se prevé que con el avance del desarrollo de la plataforma cada modelo tendrá bloques específicos que estarán disponibles en el editor, cuando se lo seleccione a la hora de armar el proyecto en el IDE. Así, podrían armarse bloques que realicen por ejemplo una lectura de un sensor, que implicaría el manejo de pines y tiempos de espera.

2.2. Plataforma CIAA

Como base para el proyecto se optó por utilizar CIAA. Es un proyecto centrado en el trabajo colaborativo que busca la innovación para crear, diseñar y desarrollar soluciones electrónicas en la industria. Dentro de sus objetivos se encuentran [16]:

- Impulsar el desarrollo tecnológico nacional, a partir de sumar valor agregado al trabajo y a los productos y servicios, mediante el uso de sistemas electrónicos, en el marco de la vinculación de las instituciones educativas y el sistema científico-tecnológico con la industria.
- Darle visibilidad positiva a la electrónica argentina.
- Generar cambios estructurales en la forma en la que se desarrollan y utilizan en nuestro país los conocimientos en el ámbito de la electrónica y de las instituciones y empresas que hacen uso de ella.

Se utiliza esta plataforma por compartir la misma idea de ser un proyecto libre, poseer una creciente comunidad e incentivar la industria nacional. Se consideró que es un proyecto suficientemente maduro y con actualizaciones y desarrollos constantes como para ser utilizado de base.

2.2.1. EDU-CIAA-NXP

La EDU-CIAA-NXP es la versión educativa y de bajo costo de la CIAA-NXP. Utiliza, al igual que la CIAA-NXP, el microcontrolador LPC4337 de NXP, un doble núcleo ARM Cortex-M4F y Cortex M0. Como se observa en la figura 2.3, la placa posee 4 leds y 4 pulsadores de hardware adicional. Para ampliar sus capacidades se utilizan los conectores laterales, para adicionar un *Poncho*, o placa expansora.



FIGURA 2.3: Placa EDU-CIAA-NXP

La EDU-CIAA-NXP permite realizar programación y debugging, así como también acceder a una de sus UARTs, por medio de USB. Además posee otro puerto para que la placa funcione en modo OTG (*On The Go*).

2.3. Requerimientos

Se plantearon requerimientos que el proyecto debe cumplir a la hora de ser entregado. Se evaluaron sus posibilidades y se clasificaron en categorías.

2.3.1. Requerimientos del sistema

Se estableció que el IDE debe poder ejecutarse como mínimo en un entorno Linux o Windows. Por la orientación didáctica del proyecto su uso debe ser simple, amigable y didáctico. Además, la programación debe poder realizarse íntegramente de manera gráfica por medio de bloques, donde cada uno represente acciones simples.

El IDE permite un manejo de proyectos en forma de archivos, que posibilitan el guardado y distribución de los mismos. Dentro de las opciones del proyecto se puede seleccionar el modelo de CIAABOT utilizado. Actualmente la opción existe, pero hay un solo modelo disponible.

El algoritmo que se genera a partir de los bloques está disponible en lenguaje C, para poder compilar y programar desde la aplicación, por un puerto USB.

2.3.2. Componentes del Sistema

Como ya se mencionó en 2.2.1, se utilizará en principio la placa EDU-CIAA-NXP, y un poncho si es necesario para controlar la maqueta o robot implementado para la muestra. En el diseño de este poncho todos los periféricos deben funcionar independientes al resto.

2.3.3. Firmware

Para el desarrollo del software se implementó un sistema de control de versiones. Para la programación de los robots se utilizaron dos bibliotecas: En C la mencionada sAPI, y una de Firmata para un monitoreo del mismo por medio del puerto serie, con un cliente corriendo en la placa.

2.3.4. Procesos Finales

Se requiere un manual de uso e instalación, junto con ejemplos básicos y funcionales, que sean representativos de las características de la plataforma. Se implementó sobre una maqueta adaptada a la EDU-CIAA, y con ello se evaluaron los resultados del proyecto y su aplicación en ámbitos de enseñanza reales.

2.4. Planificación

2.4.1. Desglose en tareas

En primer lugar, para definir objetivos concretos, se plantearon una serie de entregables para el proyecto:

- IDE en funcionamiento.
- Poncho CIAABOT G1 diseñado enteramente.
- Manual de uso, instalación y ejemplos funcionales.
- El presente informe final.

Siguiendo los lineamientos de las estrategias vistas en la materia Gestión de Proyectos, se procedió al diseño de una lista de desglose del trabajo en tareas. Se estimó un tiempo aproximado de 615 horas, distribuidas en grupos de tareas de la siguiente manera:

- Planificación del proyecto (40 hs.).
- Investigación preliminar (45 hs.).
- Selección de bibliotecas y plataformas (30 hs.).
- Desarrollo de la aplicación de escritorio (45 hs.).
- Desarrollo del editor gráfico (55 hs.).
- Programación por USB y monitoreo (60 hs.).
- Desarrollo del hardware (100 hs.).
- Desarrollo del firmware (70 hs.).
- Integración del sistema (50 hs.).
- Procesos finales (120 hs.).

2.4.2. Activity On-node

Si se toman las tareas antes descriptas y se evalúan sus dependencias, es decir las tareas que deben estar finalizadas antes de poder realizarlas, se puede armar un diagrama denominado *Activity on-node*. En la figura 2.4 se pueden observar las tareas propuestas junto con el tiempo estimado en días para cada tarea. Todas las flechas entrantes a un nodo o tarea son las dependencias de la misma.

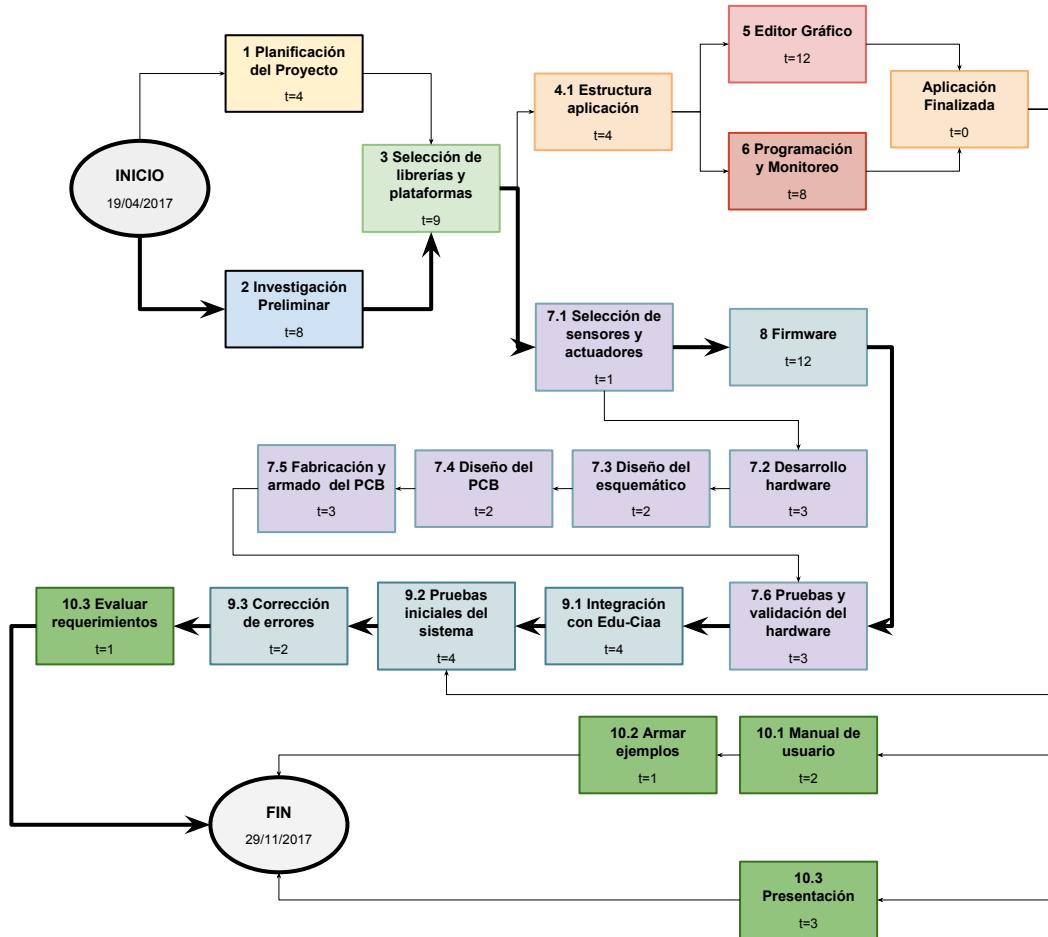


FIGURA 2.4: Diagrama Activity on-node

Los días están expresados en días laborales de aproximadamente 3 horas. Evaluando las duraciones y precedencias de las tareas, la sucesión que presente la mayor cantidad de horas es llamada *camino crítico*.

En la planificación de un proyecto es importante su identificación, ya que cualquier tarea que se encuentre sobre él tiene lo que se llama flotación cero. Esto significa que un incremento en su duración llevará indefectiblemente a una dilatación del plazo de finalización del proyecto. En el diagrama este camino está resaltado con flechas más gruesas.

2.4.3. Diagrama de Gantt

Si se establecen fechas concretas para los inicios de las tareas, teniendo en cuenta su duración y precedencias, se puede armar el llamado diagrama de Gantt. Esto da una referencia rápida de dónde se debería encontrar el desarrollo del proyecto según la planificación inicial en cada momento dado. En las figuras 2.5 y 2.6 se puede observar el diagrama para este proyecto.



FIGURA 2.5: Diagrama de Gantt - Parte 1.

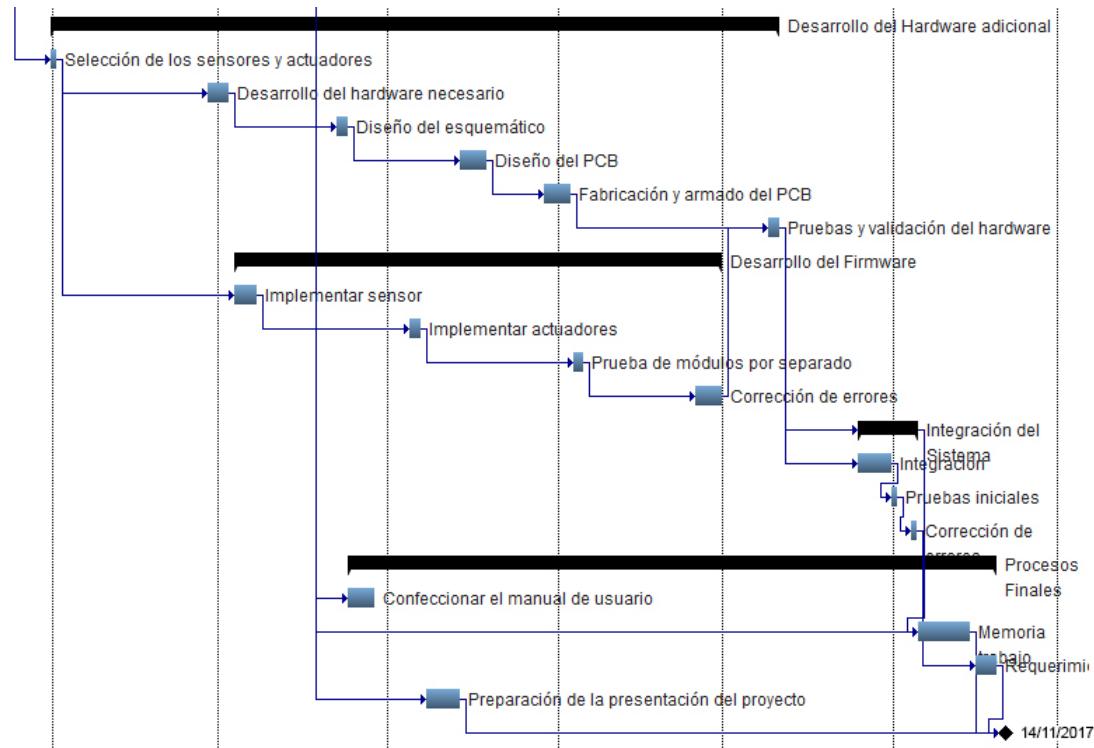


FIGURA 2.6: Diagrama de Gantt - Parte 2.

Capítulo 3

Diseño e Implementación

En este capítulo se describe la implementación y la estructura de software adoptada para el diseño de CIAABOT.

3.1. Estructura del software

En esta sección se detallan y justifican las decisiones de diseño tomadas. También se desarrollará en profundidad el diseño del software del IDE.

3.1.1. Estructura de un proyecto base

Cuando se crea un proyecto nuevo en CIAABOT IDE, se crea una carpeta con varias subcarpetas y archivos. En la figura 3.1.1 se observa esta estructura simplificada. Está basada en la estructura de archivos que se utiliza en el Firmware V2 de la CIAA.

Firmware V2 es un proyecto, basado en makefile para el desarrollo de software en C, especialmente enfocado en sistemas embebidos, particularmente microcontroladores. Actualmente provee soporte para:

- LPC11U68 (Cortex M0+).
- LPC1769 (Cortex M3).
- LPC4337 (Cortex M4 y M0).
- LPC54102 (Cortex M4 y M0+).

El Firmware V2 se basa principalmente en el workspace del Esp. Ing. Pablo Ridolfi y la sAPI del Ing. Eric Pernía, que es el director del presente trabajo final.

En la estructura del proyecto se encuentra el archivo con extensión .cbp. Éste contiene la información del proyecto, su nombre, el modelo de CIAABOT utilizado y el diagrama en bloques. Los archivos config.mk y Makefile se utilizan para la compilación del código C y la descarga del mismo a la placa.

En el directorio `libs` se encuentran las diferentes bibliotecas que incluye el proyecto. En `lpc_chip_43xx` están las bibliotecas de LPCOpen para el microcontrolador LPC4337, provistas por el fabricante. En `lpc_board_ciaa_edu_4337` está la biblioteca a nivel de placa puntualmente para manejar la placa EDU-CIAA-NXP.

En la carpeta `sapi` se encuentra la biblioteca sAPI. Esta biblioteca permite manejar varios periféricos del microcontrolador de manera simplificada. Estos son:

- Interrupciones.
- SysTick.
- GPIO.
- UART.
- ADC.
- DAC.
- I^2C .
- SPI.
- RTC.
- SCT.
- Timers.

Además incluye funciones avanzadas de conversión de datos, impresión formateada por UART, manejo de displays 7 segmentos, teclados matriciales, servos y PWM entre otras.

Finalmente, la carpeta `app` contiene la aplicación de usuario. Puntualmente el archivo `main.c` es el que se actualiza al realizar modificaciones en el editor de bloques del IDE.

```

Estructura de un proyecto de CIAABOT IDE
| config.mk
| Makefile
| mi-proyecto.cbp
|
|---app
|   |---inc
|   |   programa.h
|   |
|   |---src
|   |   main.c
|
|---libs
|   |---lpc_board_ciaa_edu_4337
|   |   |---inc
|   |   |---src
|   |
|   |---lpc_chip_43xx
|   |   |---inc
|   |   |---src
|   |---sapi
|   |   |---inc
|   |   |---src
|
|---scripts

```

FIGURA 3.1: Estructura simplificada de un proyecto base.

3.1.2. Editor de código en bloques

El editor de bloques es la sección central de CIAABOT IDE. Aquí el usuario toma e interconecta los diferentes bloques que permiten realizar los programas. Se utilizó la biblioteca de JavaScript Blockly de Google [4], realizada por Neil Fraser entre otros.

Blockly es una biblioteca que utiliza bloques gráficos encastrables para representar los conceptos de código como variables, expresiones lógicas, bucles y más. Permite que los usuarios apliquen conocimientos de programación sin preocuparse por una sintaxis específica.

Durante el proceso de selección de la biblioteca para armar el editor gráfico, se seleccionó Blockly por las siguientes ventajas que presenta:

- Permite exportar el código simplemente.
- Es un proyecto de software libre.
- Es ampliable, ya que se pueden agregar bloques personalizados.
- Ha sido traducido a más de 40 idiomas, y permite una traducción de los bloques propios.

Esta biblioteca permite, por un lado, definir bloques con diferentes tipos de conexiones y parámetros, y por otro armar generadores que reciban los bloques y

generen código. Nativamente Blockly soporta varios lenguajes, pero no C. Para eso, se utilizó parte del desarrollo de Fred Lin, BlocklyDuino [10], que resolvió varios de las estructuras típicas de C.

Blockly permite definir los bloques independientemente de los generadores de código. La definición implica la forma que tendrá el bloque para el usuario (colores, secciones, textos, tipos de bloques admitidos en cada sección y configuraciones especiales).

Para definir los bloques se utilizan archivos JavaScript y una sintaxis especial. Por ejemplo, así se define el bloque de retardo bloqueante:

```

1 Blockly.Blocks['ciaa_sapi_blocking_delay'] = {
2     init: function () {
3         this.appendValueInput("delay_time")
4             .setCheck("Number")
5             .appendField("Esperar ticks");
6         this.setPreviousStatement(true, null);
7         this.setNextStatement(true, null);
8         this.setColour(230);
9         this.setTooltip('Retardo bloqueante');
10        this.setHelpUrl('');
11    }
12};
13

```

CÓDIGO 3.1: Definición del bloque de retardo bloqueante.

Es posible definir varios generadores de código para los mismos bloques. Esto significa que un mismo programa podría ser traducido a varios lenguajes según la necesidad. A modo de ejemplo se muestra cómo se puede definir el generador de código para lenguaje C para el bloque de retardo bloqueante:

```

1 Blockly.CiaaSapi.ciaa_sapi_blocking_delay = function() {
2     var delay_time = Blockly.CiaaSapi.valueToCode(this, 'delay_time',
3         Blockly.CiaaSapi.ORDER_ATOMIC) || '2000';
4     var code = 'delay(' + delay_time + ');\\n';
5     return code;
6 };

```

CÓDIGO 3.2: Generador de código en lenguaje C para el retardo bloqueante.

Si se observa la definición del bloque, se está agregando una entrada del tipo *value* llamada *delay_time*, que luego en el generador es tomada y utilizada como argumento de la función *delay* de la sAPI.

Internamente en el programa, cuando el usuario modifica el esquema de bloques, se llama a la biblioteca. Esta recibe como parámetro el nuevo diagrama en formato XML (*eXtensible Markup Language*, Lenguaje de Marcado Extensible), y el generador de código que se desea aplicar. Como resultado se obtiene una cadena de caracteres que es el código generado. Este código puede suponerse sintácticamente correcto, y en la mayoría de los casos compilará sin problemas.

Como se aprecia en la figura 3.2, el editor tiene tres secciones principales: Una barra de acciones, una caja de herramientas y el área de edición.

En la figura 3.3 se presenta una descripción detallada de las acciones disponibles en la Barra de Acciones. La caja de herramientas divide los bloques en categorías:

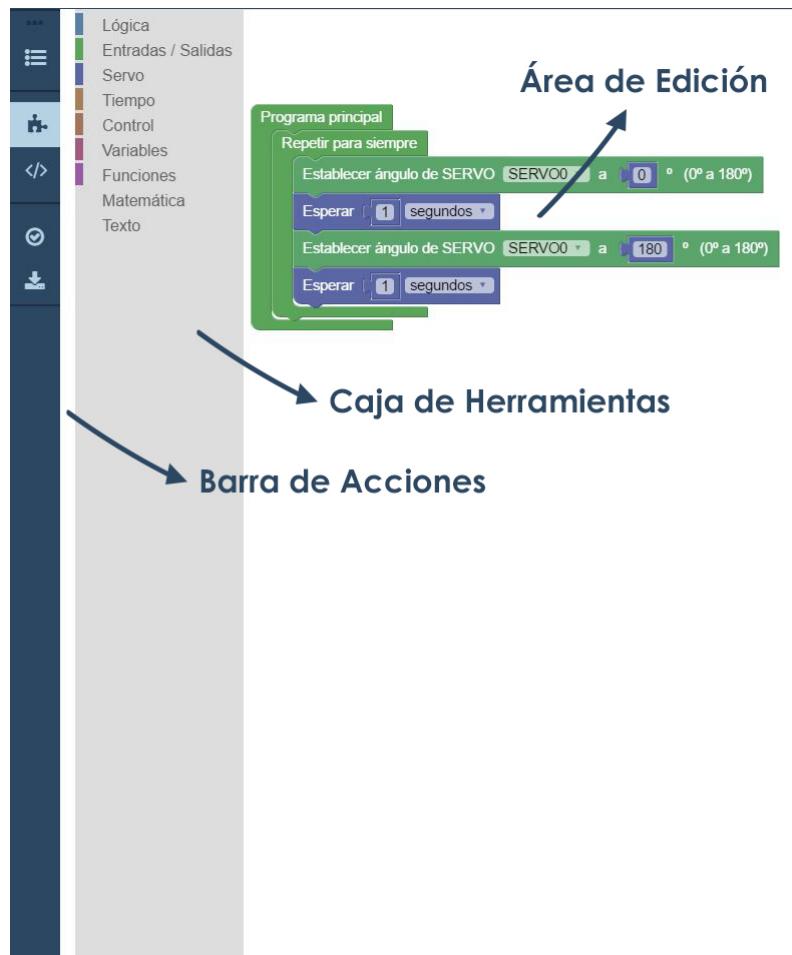


FIGURA 3.2: Secciones del editor de bloques.

- Lógica
- Entradas / Salidas.
- Servo.
- Tiempo.
- Control.
- Variables.
- Funciones.
- Matemática.
- Texto

Los bloques se arrastran al área de edición y se interconectan con los demás. Hay diferentes tipos de bloques, según su tipo de conexión. Hay bloques que admiten solamente conexiones superior e inferior (figura 3.4), como el de *Esperar x segundos*. Estos bloques no retornan ningún valor, ni contienen otros bloques internamente. En el caso de este bloque, admite dos configuraciones: Unidad de tiempo en la que se esperará y cantidad de esas unidades. La unidad se selecciona con una lista de valores predeterminados, y la cantidad admite un bloque numérico (puede ser un número inmediato o el resultado de alguna operación).



FIGURA 3.3: Detalle de la barra de acciones.



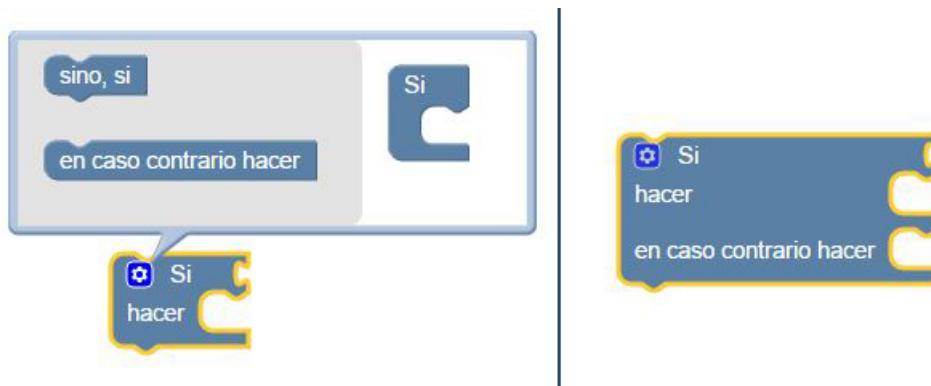
FIGURA 3.4: Bloque Esperar x segundos.

Otros bloques, como la lectura del periférico conversor analógico a digital, retornan un valor que es tomado por algún bloque como argumento (por ejemplo una comparación). Este tipo de bloque se muestra en la figura 3.5. No admite conexiones superior ni inferior.



FIGURA 3.5: Bloque Leer conversor analógico a digital.

Hay algunos bloques más complejos como el *Si...hacer* que permiten una configuración avanzada de la forma que presentan y los bloques que admiten. En la figura 3.6 se observa cómo se configura este bloque para que admita una segunda sección de bloques que se ejecutan de no cumplirse la condición estipulada en *Si....*



A modo de ejemplo, en la figura 3.7 cómo se armaría en bloques el código para el barrido angular de un servomotor.



FIGURA 3.7: Ejemplo de código en bloques para el barrido de un servo.

El código de salida de la figura 3.7 se muestra a continuación:

```

1 #include "sapi.h"
2 CONSOLE_PRINT_ENABLE
3
4 inline void setup(void) {
5     servoConfig(0, SERVO_ENABLE);
6     servoConfig(SERVO0, SERVO_ENABLE_OUTPUT);
7 }
8
9
10 void main(void) {
11     // Inicializa placa
12     boardConfig();
13
14     // Habilita cuenta de tick cada 1ms
15     tickConfig(1, 0);
16
17     // Inicializaciones del usuario
18     setup();
19
20     while (TRUE) {
21         servoWrite(SERVO0, 0);
22         delay(1000 * 1);
23         servoWrite(SERVO0, 180);
24         delay(1000 * 1);
25     }
26 }
```

CÓDIGO 3.3: Salida del código en bloques de la figura 3.7.

3.1.3. Definición del lenguaje específico

Un DSL (*Domain-specific language*, Lenguaje de dominio específico) es un lenguaje de programación o especificación dedicado a resolver un problema en particular, representar un problema específico y proveer una técnica para solucionar esa situación particular. Ejemplos de esto incluyen Verilog (descripción de hardware), R y S para estadísticas, SQL para consultas a bases de datos, entre otros. [7].

A la hora de diseñar los bloques que se incluirían en CIAABOT IDE se puso como foco principal su uso didáctico, que fuera de fácil comprensión y que el esquema de bloques resultante pudiese leerse con una fluidez similar al habla. Para esto se debió hacer una definición del lenguaje CIAABOT.

La versión actual se acotó a una serie de limitaciones, por cuestiones de tiempo. Estas son:

- Pines con funcionalidades fijas.
- Periféricos disponibles: GPIO, ADC, DAC, UART, I^2C , PWM, Servo.
- Todas las variables del usuario son globales.
- Las funciones (bloques de usuario) no retornan valores (son procedimientos).
- El tipo de dato *string* se utiliza únicamente para crear literales para enviar por UART.

En la tabla 3.1 se lista la versión 1.0.0 de la definición del lenguaje CIAABOT. Los bloques se agruparon por categoría.

TABLA 3.1: Definición del lenguaje CIAABOT v1.0.0

	API	
Nombre/categoría		Traducción
Control de ejecución		
if	Si <condicion booleana>hacer []	
else	Si no hacer []	
switch, case, default	Si <var>es igual a <literal>hacer [] si es distinto a los anteriores hacer []	
Bucles de repetición		
do	Hacer [] y repetir <mientras/hasta><condicion booleana>	
while	Repetir [] <mientras/hasta><condicion booleana>	
for	Iterar <var int>desde <literal int>hasta <literal int> incrementando <literal int>y hacer []	
	Repetir [] <literal int>veces	
while(TRUE)	Repetir para siempre []	
Manejo de tiempos		
tickRead	Leer base de tiempo (ms)	
tickWrite	Escribir base de tiempo (ms)	
delay	Esperar durante <int><unidad>	
delay_t <var>;	Crear temporizador con nombre <var>	
delayConfig	Iniciar <temporizador>con <int><unidad>	
delayRead	Leer <temporizador>	
delayWrite	Escribir <temporizador>con <int><unidad>	
GPIO		
gpioRead	Leer estado del GPIO <GPIOi, TECi, LEDi>	
gpioWrite	Establecer estado del GPIO <GPIOi , LEDi> en <encender/apagar>	
gpioToggle	Invertir el estado del pin <GPIOi, LEDi>	
ADC		
adcRead	Leer ADC <ADCi>	

TABLA 3.1: Definición del lenguaje CIAABOT v1.0.0

API	
DAC	
dacWtite	Establecer DAC <DACi>al valor <literal int>
UART	
uartReadByte	Recibir byte desde UART <UARTi>
uartWriteByte	Enviar byte <var/literal>por UART <UARTi>
uartWriteString	Enviar texto <Texto>por UART <UARTi>
PWM	
pwmWrite	Establecer ciclo de trabajo de PWM <PWMi>en <literal int> %
SERVO	
servoWrite	Establecer ángulo de SERVO <SERVOi>a <literal int>
servoRead	Leer ángulo de SERVO <SERVOi>

3.1.4. Servicios

Para el desarrollo de CIAABOT IDE se utilizó *Electron*[14], una biblioteca de código abierto desarrollada por GitHub para armar aplicaciones de escritorio multiplataforma utilizando tecnologías web como HTML, CSS y JavaScript (o TypeScript). De esta manera, se puede simplificar y reutilizar el diseño de una aplicación, ya que con mínimas consideraciones se puede desarrollar una aplicación de escritorio una vez que compile para varios sistemas operativos. Esto lo logra combinando el navegador Chromium de Google con *NodeJS* [15]. Se puede observar en la figura 3.1.4 las tecnologías utilizadas para el desarrollo de CIAABOT IDE, en un formato de capas, que finaliza con la interacción de la aplicación con el sistema operativo.

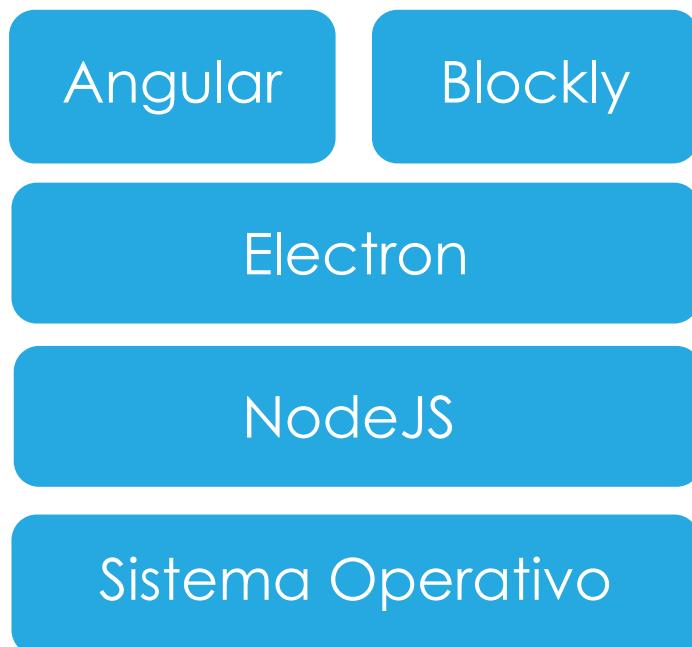


FIGURA 3.8: Distribución en capas de las tecnologías utilizadas.

NodeJS es un entorno de ejecución de JavaScript que utiliza el motor V8 de Google. Utiliza un modelo de entradas y salidas no bloqueante y orientado a eventos. Esto y la posibilidad de utilizar múltiples hilos de ejecución, lo hacen ideal para el desarrollo de aplicaciones que requieran escalar.

Sobre las tecnologías arriba descritas se utilizó *Angular*[3], una plataforma para desarrollar aplicaciones web en HTML y JavaScript, o algún lenguaje que compile en JavaScript como TypeScript. La biblioteca consiste en varias bibliotecas modularizadas.

La arquitectura de Angular consiste en *plantillas (templates)* HTML con una sintaxis especial, clases de *componentes* que manejan estas plantillas y lo que muestran, *servicios* que manejan la lógica de la aplicación y *módulos* que engloban estos dos últimos. En general las aplicaciones tienen varios módulos que se incluyen en un módulo principal que inicia la aplicación.

Los componentes manejan en general pequeñas porciones de *vistas* dentro de la aplicación (como ser navegaciones, listas dinámicas entre otras). Las plantillas, utilizando directivas, permiten mostrar propiedades del componente y actualizarlo a medida que éstas lo hagan. A su vez, informa al componente de los eventos ocurridos para que pueda reaccionar ante ellos. Esta dinámica se esquematiza en la figura 3.9.

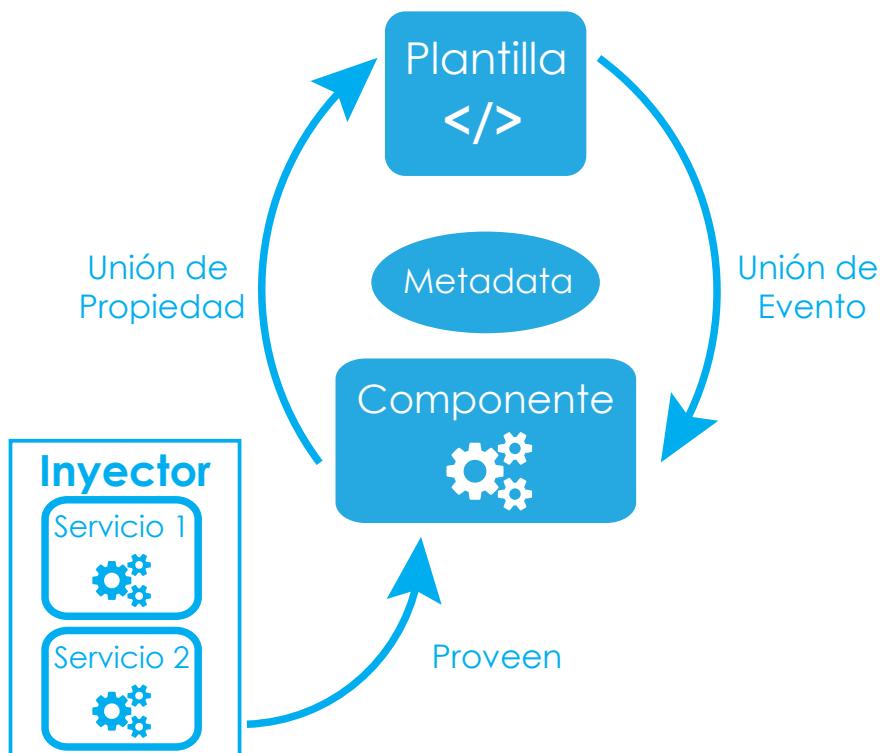


FIGURA 3.9: Interacción entre componente, plantilla y servicios en Angular.

Los servicios intentan crearse con funciones puntuales y bien definidas. Los servicios son generalmente consumidos por componentes o incluso por otros servicios. Ejemplos de posibles usos son:

- Manejo de datos.

- Bus de mensajes.
- Configuración de la aplicación.
- Acceso a repositorio de datos.

Angular utiliza inyección de dependencias para proveer a las clases de las instancias de clases de las que depende. En general estas clases son servicios, se crea una instancia de ellos y se utiliza en los componentes en los que se requiera.

Este modo de estructurar el código sigue el lineamiento de división de responsabilidades, donde se arman pequeñas clases (componentes y servicios) que cumplen una tarea determinada, puntual y bien acotada.

Siguiendo con la idea de separación de tareas, se crearon (entre otros) dos servicios: *ProjectService* encargado de tareas inherentes al manejo del proyecto abierto, y *CompilingService* que permite compilar el código C y programarlo en la placa. Estos archivos pueden encontrarse en la ruta `/src/app/providers` dentro del código fuente del proyecto, ya que proveen a componentes de servicios o funcionalidades.

ProjectService posee el método `createProject` que crea la estructura básica de carpetas descrita en la subsección 3.1.1. Para esto recibe el directorio que el usuario seleccionó para guardar su proyecto y utiliza la API `fs (file system)` de NodeJS para crear los archivos y carpetas requeridos. Además, agrega este proyecto nuevo a la lista de proyectos recientes, que se guarda con la configuración persistente del usuario.

Otro método importante que posee este servicio es `openService`, que recibe la ruta de un proyecto existente y lo abre. De él toma los datos básicos y la estructura de bloques guardada en XML. Si el usuario aplica cambios sobre el proyecto abierto y están sin guardar, este servicio se encarga de registrar esa situación, por ejemplo para emitir una alerta al momento de intentar cerrar la aplicación sin haber guardado previamente.

Finalmente, este servicio también se encarga del guardado de los proyectos en archivos con extensión `.cbp`.

3.1.5. Compilación

Como se mencionó en la subsección 3.1.1, el proyecto base utiliza Firmware V2 que es un proyecto en base a makefile. Esto significa que se utiliza la herramienta *make* para realizar las diferentes tareas. Make sirve para la gestión de dependencias, como las que existen entre los archivos que componen el código fuente de un programa, para manejar su generación automática.

La función básica de make es determinar qué partes de un programa requieren ser recompiladas, y se encarga de ejecutar los comandos necesarios para lograrlo. Los makefiles son los archivos que utiliza esta herramienta para saber cómo compilar los programas y de qué depende cada uno. Podría hacerse una analogía con una receta, donde se encuentran los pasos para obtener lo requerido.

Los makefiles tienen una forma de reglas, donde se especifica lo que se debe hacer para obtener un módulo. Por ejemplo, a continuación se muestra cómo se define

un módulo *juego*, que requiere que se creen tres archivos de tipo objeto (.o), y luego especifica los comandos que deben ejecutarse para poder obtenerlos:

```

1 juego : ventana.o motor.o bd.o
2     gcc -O2 -c juego.c -o juego.o
3     gcc -O2 juego.o ventana.o motor.o bd.o -o juego

```

CÓDIGO 3.4: Ejemplo de un objetivo en makefile

Luego, si se ejecutara `make juego`, se evaluarían los archivos con cambios desde la última compilación, y si hubiese cambios se recompilarían. De manera similar, el makefile de Firmware V2 tiene objetivos para programar el archivo binario compilado directamente a la placa, invocando a la herramienta de debug OpenOCD.

Contando con este simple manejo que provee un proyecto basado en makefile, se armó el servicio `compilingService`. Posee tres métodos que valen la pena destacar por su importancia para la aplicación.

El método `createMainFile` es el encargado de reescribir el archivo `main.c` con los últimos cambios que el usuario introdujo en el diagrama de bloques en el editor. Básicamente toma el archivo y solicita a `blockly` el código generado por el diagrama actual, luego lo reescribe y guarda el archivo.

Luego, el `compileProgram` actualiza el archivo `main.c` llamando al método recién descrito, y ejecuta un proceso hijo. Este proceso hijo corre en paralelo al hilo principal de la aplicación y actúa de manera no bloqueante. Esta es una gran ventaja que proporciona trabajar con NodeJS, ya que muchas operaciones que interactúan con el sistema operativo son bloqueantes y si no se pudiese lanzar un hilo en paralelo la aplicación no respondería hasta que finalizaran. Este proceso hijo que se lanza ejecuta la herramienta `make`, invocando al objetivo de compilación presente en el Makefile.

Algo importante de destacar es que la variable de ambiente *Path*, es decir las rutas donde se buscarán los ejecutables, es modificada para los procesos hijos de este servicio. En las opciones de la aplicación, el usuario puede seleccionar rutas adicionales de búsqueda para indicar dónde se deberían buscar las herramientas necesarias (compilador y debugger), como se puede apreciar en la figura 3.10.

El proceso hijo recibe una función de *callback*, que será llamada una vez finalizado. Recibirá como argumentos el posible error si existiese, y la salida estándar del proceso. En función del resultado se notifica al usuario con el servicio de notificaciones.

Por último, el método `downloadProgram` actúa de manera similar a `compileProgram`, solo que esta vez se invoca el objetivo de grabado en flash del binario ya compilado.



FIGURA 3.10: Ejemplo de rutas de búsqueda adicionales en el IDE.

3.2. Diseño del poncho CIAABOT G1

Como se mencionó en la sección 1.3, formó parte del alcance el diseño de un poncho para adaptar a una maqueta o robot existente a la plataforma. Para esto se diseñó un poncho, una placa que se puede conectar a la EDU-CIAA-NXP para ampliar las funcionalidades.

En este caso, el primer modelo se llamó *G1*. Se pensó en un robot que tuviera sensores infrarrojos, o algún otro que deba ser medido de manera proporcional por un ADC o que pudiese conectarse directamente a una entrada digital. Además, se colocaron tres conectores que corresponden a las salidas de servomotores (SERVO0 a SERVO2) de la placa. Esto puede servir para movimiento de actuadores o para apuntar sensores. La sección de conectores puede apreciarse en el recorte del esquemático de la figura 3.11.

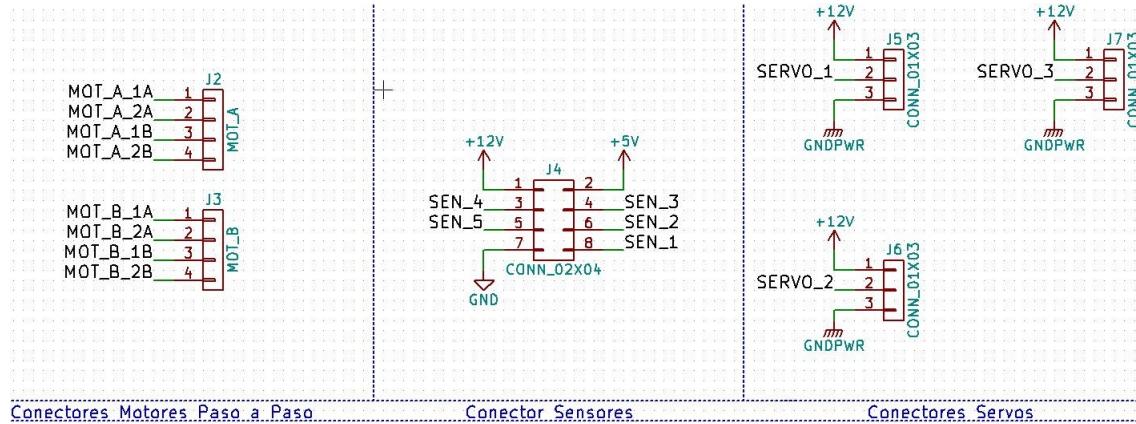


FIGURA 3.11: Recorte del esquemático que muestra los conectores disponibles.

Para el movimiento se optó por colocar dos drivers A4988 de motores paso a paso, como los de la figura 3.12. Cada uno permite manejar un motor, con tensiones de hasta 35V y corrientes de hasta 2A. Además, permiten realizar micro-pasos sobre el motor, con una resolución de hasta un dieciseisavo de paso.

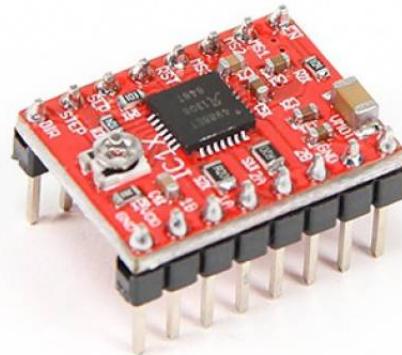


FIGURA 3.12: Driver A4988 de motor paso a paso.

Para la alimentación del robot se consideró el uso de baterías. Se diseño un circuito comutador con el integrado LM2577adj (figura 3.13), para elevar la tensión a los 12V utilizados sobre los motores paso a paso. Además, se agregó un regulador lineal LM7805 para la alimentación de la placa EDU-CIAA-NXP, que a su vez cuenta con un regulador para obtener los 3,3V necesarios para la lógica.

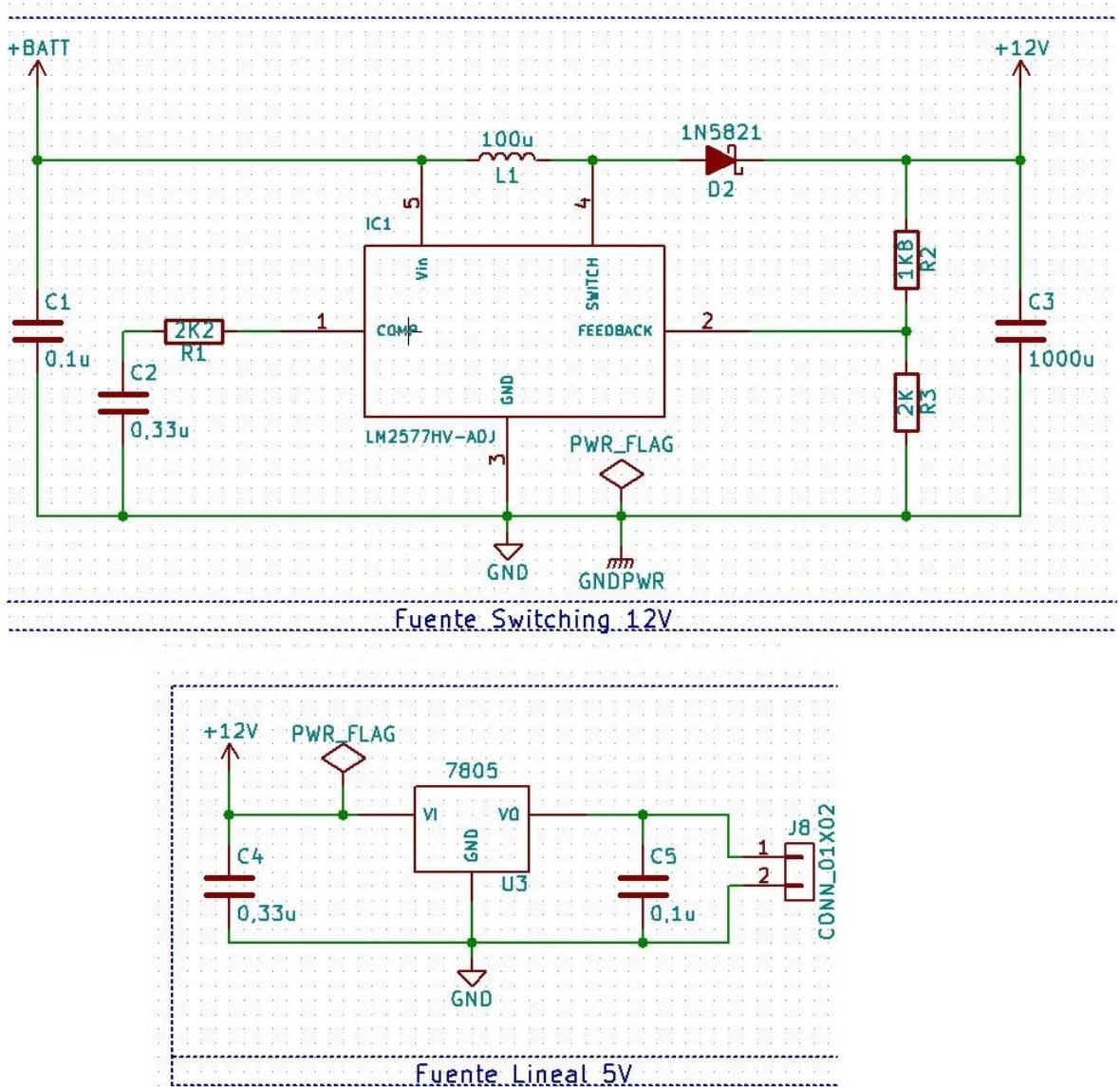


FIGURA 3.13: Recorte del esquemático que muestra la fuente de alimentación.

En la figura 3.14 se puede observar el diseño preliminar del circuito impreso del poncho, realizado en dos capas.

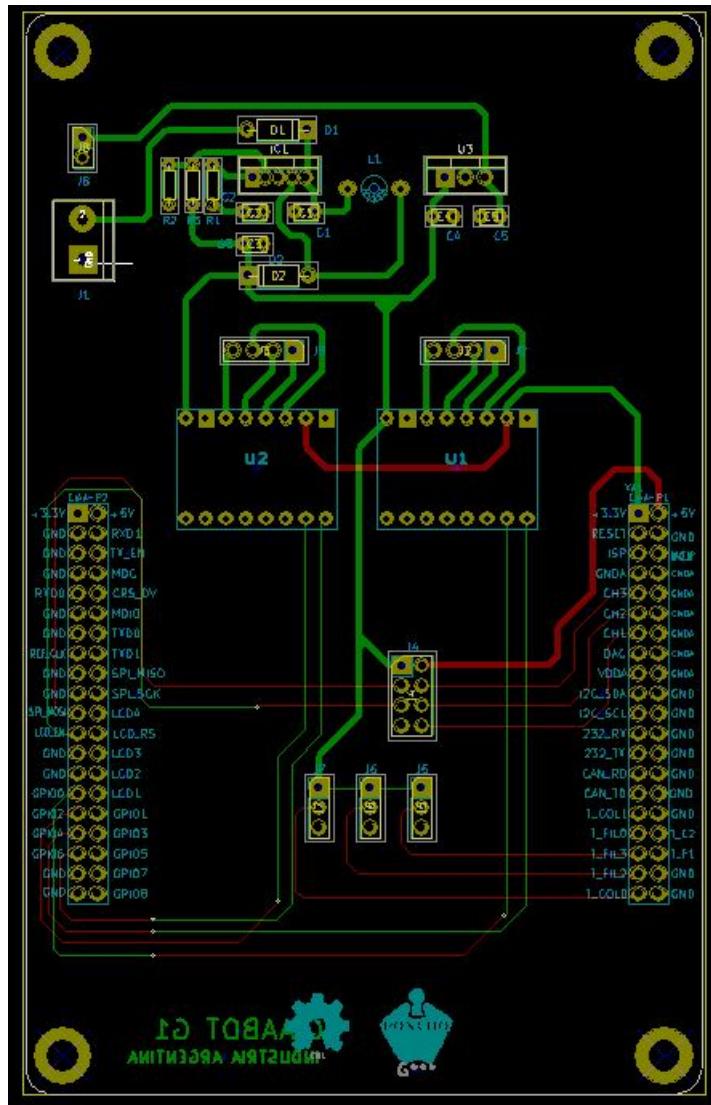


FIGURA 3.14: Diseño del circuito impreso del poncho G1.

3.3. Herramientas

3.3.1. Git

Al momento de realizar la planificación del presente trabajo, se evaluaron posibles riesgos, cuál sería su impacto y cómo deberían poder mitigarse, ya sea reduciendo su probabilidad de ocurrencia o su severidad.

Dentro de los riesgos planteados se consideró la pérdida de archivos importantes para el desarrollo de la aplicación. Se evaluó que su consecuencia sería muy severa ya que implicaría una pérdida muy grande de tiempo, que no podría recuperarse en el cronograma de trabajo. La estrategia de mitigación para este riesgo fue el uso constante y riguroso del sistema de control de versiones *Git*.

Un sistema de control de versiones registra los cambios a uno o más archivos a lo largo del tiempo, para luego poder recuperar versiones específicas. Hay diferentes sistemas, hoy en día el más utilizado es *Git*.

Git es un sistema del tipo distribuido, esto quiere decir que los usuarios toman la historia completa del repositorio cada vez que lo clavan del servidor, como se observa en la figura 3.15. De esta manera cada computadora es una copia de seguridad completa del proyecto, y en caso de que el servidor sufriera problemas, las pérdidas serán casi nulas.

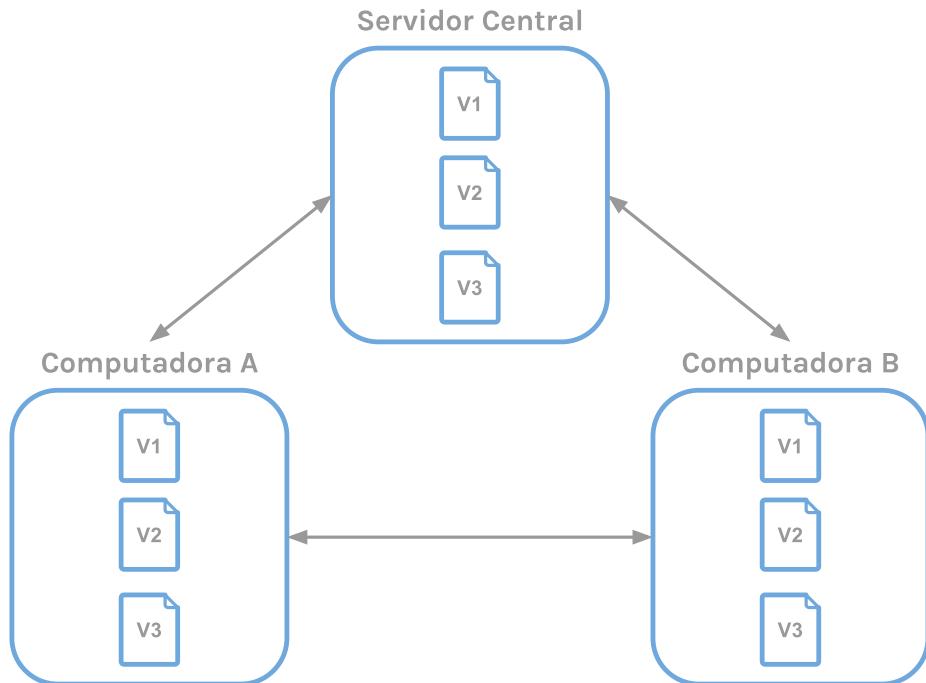


FIGURA 3.15: Diagrama de un repositorio distribuido. Cada computadora posee todas las versiones.

El proyecto CIAABOT es abierto, y el código fuente para compilar o modificar el IDE se encuentra en GitHub [12]. El desarrollo se separó en ramas en el repositorio, para un trabajo más puntual sobre diferentes características (pantalla de carga, rutas adicionales de búsqueda, variables tipadas, etc), y el código que debe considerarse estable se mantiene en la rama Master.

3.3.2. OpenOCD

Para el grabado del firmware en las placas y su posterior depuración, se utilizó la herramienta OpenOCD. Es un depurador libre *on-chip*, creado por Dominic Rath. Utilizando entornos de desarrollo como Eclipse o Visual Studio Code, es posible depurar diferentes placas conectadas por puerto serie a la PC.

OpenOCD permite utilizar archivos de configuración para seleccionar la placa que se desea utilizar, lo cual es muy útil para CIAABOT ya que se espera que se puedan ampliar las placas soportadas del ecosistema CIAA.

Capítulo 4

Ensayos y Resultados

Este capítulo describe las pruebas que se realizaron sobre la plataforma, así como también su utilización en una situación aúlica real para dictado de cursos.

4.1. Pruebas de sensores y actuadores

Una vez desarrollados todos los bloques propuestos para la primera versión de la definición del lenguaje de CIAABOT, se pasó a una etapa de pruebas, para verificar que el manejo de los diferentes periféricos fuera correcta.

La primera prueba fue sobre un modelo impreso en 3D de un dedo índice, que es flexionado por un mecanismo accionado por un servomotor. En la figura 4.1 se observa el ensayo. Adicionalmente, la prueba también evaluó el funcionamiento de la lectura del conversor analógico-digital. Para esto se conectó un potenciómetro al canal 1 de conversión, que era leído constantemente.

El valor obtenido por el conversor (rango de 0 a 1023) era mapeado luego al rango admitido por la función de posicionamiento del servomotor (0 a 180), y luego era enviado al bloque de manejo del ángulo.

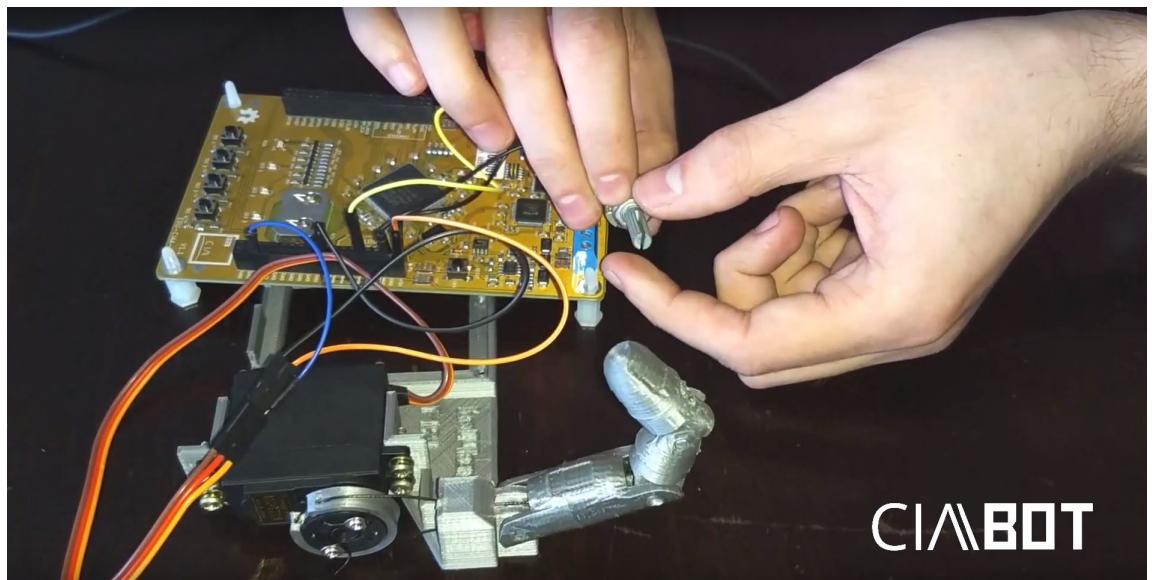


FIGURA 4.1: Prueba de servomotor sobre dedo 3D.

Teniendo en mente la necesidad de desarrollar ejemplos didácticos para utilizar CIAABOT, se utilizaron materiales desarrollados y provistos por la Universidad Nacional de Quilmes, en el marco del proyecto "La universidad y la escuela secundaria", dirigido por Cristina Wainmaier (Vicedirectora del Dto. de CyT, UNQ). Estos materiales incluían una maqueta de manejo de barreras y semáforos e instrumentos meteorológicos.

Para la verificación del correcto manejo del tiempo por parte del programa, se utilizó un anemómetro. Este dispositivo, que puede apreciarse en la figura 4.2, gira ante la presencia de viento y genera pulsos a intervalos angulares fijos. Conociendo la frecuencia de estos pulsos es posible determinar la velocidad del viento en cada momento.

Además, en este ensayo se probó la UART, proveyendo de una salida de consola por puerto serie, que podía observarse desde la PC. De esta manera el programa podía informar las mediciones realizadas, y guardarse en un archivo de log.



FIGURA 4.2: Anemómetro utilizado en las pruebas.

De los instrumentos meteorológicos también se utilizó una rosa de los vientos formada por una matriz de leds. Con esto se validó el funcionamiento de las salidas digitales funcionando a la vez. En la figura 4.3, se observa un dibujo de la rosa de los vientos utilizada, así como la conexión de la matriz de leds que posee para encender los diferentes puntos cardinales.

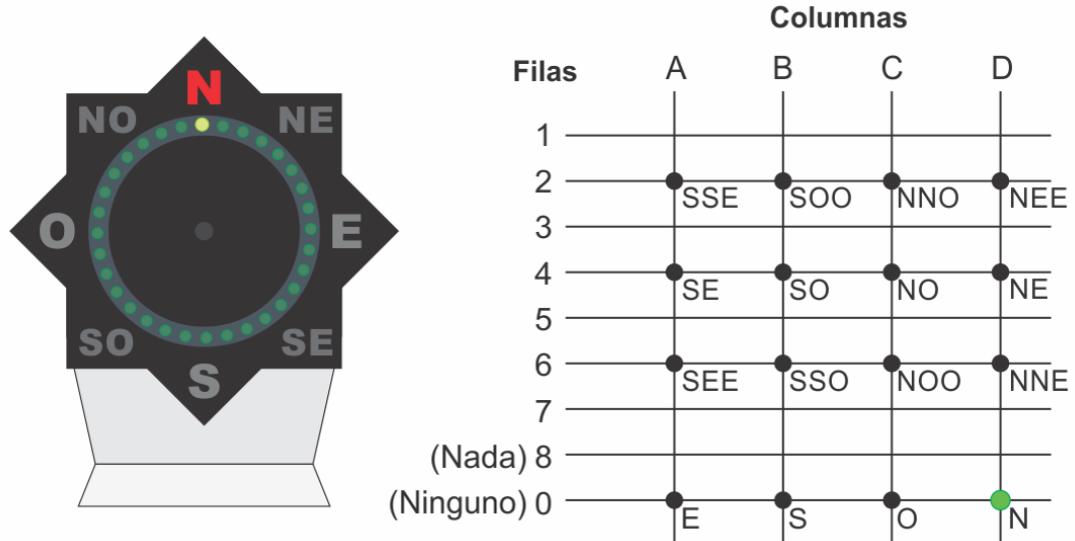


FIGURA 4.3: Rosa de los vientos y conexión de los leds.

4.2. Workshop en el SASE

Desde el 2010 se viene realizando año a año el Simposio Argentino de Sistemas Embebidos. Este evento es organizado por ACSE (Asociación Civil para la Investigación Promoción y Desarrollo de los Sistemas Electrónicos Embebidos). Es un evento anual que reúne a la comunidad académica y a la industria en torno a esta temática, donde se realizan Tutoriales, Workshops y el Congreso Argentino de Sistemas Embebidos.

En la edición de este año se agregó el workshop "Introducción a la programación de sistemas embebidos utilizando CIAABOT", que tuve la posibilidad de dictar y organizar en conjunto con el Ing. Eric Pernía. Estuvo orientado principalmente a alumnos de escuelas secundarias o entusiastas que buscaran una introducción a la programación de embebidos pero que no tuvieran conocimientos de ningún lenguaje para hacerlo.

Con una buena concurrencia se desarrolló la primera experiencia real en campo de la utilización de la plataforma CIAABOT para la enseñanza de programación. Se abarcaron temas básicos como el funcionamiento de sentencias condicionales, estructuras de control, lógica booleana y manejo básico de periféricos.

El workshop se realizó utilizando solamente CIAABOT IDE como medio para programar las placas. Se hizo uso de los elementos meteorológicos y la maqueta vehicular descrita en la sección 4.1, que puede observarse en la figura 4.4. Fue una buena oportunidad para obtener devoluciones por parte de los primeros usuarios de prueba, que fueron los asistentes al curso. Encontraron la aplicación intuitiva y se desenvolvieron de manera natural. Hubo dos casos puntuales de personas que no habían programado nunca antes de asistir, y ambos pudieron completar satisfactoriamente los ejercicios propuestos.

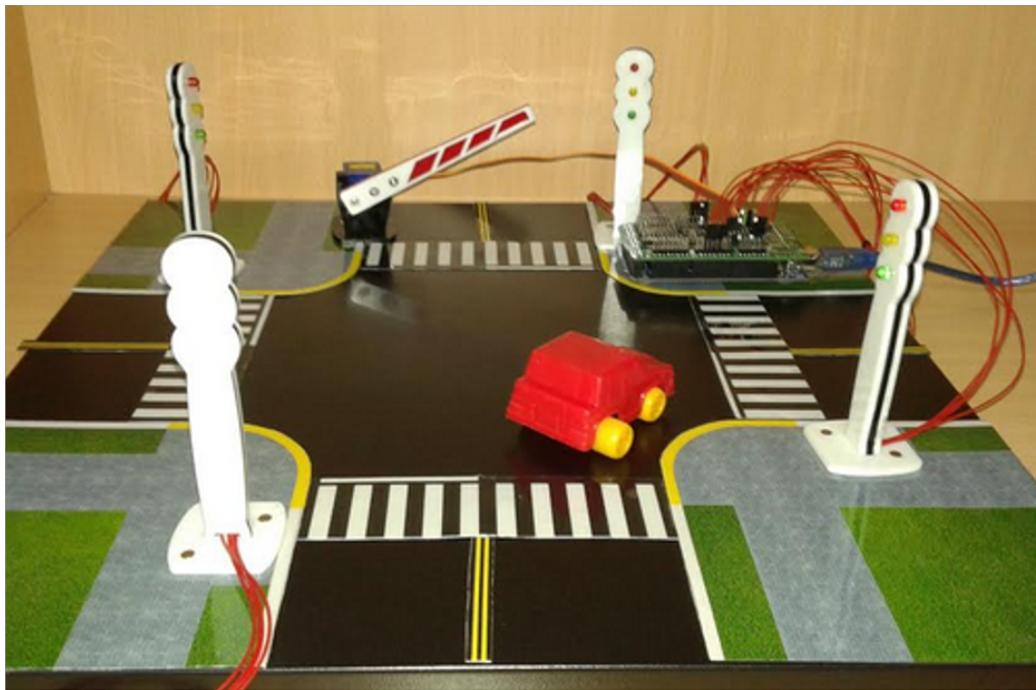


FIGURA 4.4: Maqueta vehicular utilizada en el workshop.

4.3. Experiencias en cursos CAPSE y de INET

Los Cursos Abiertos de Programación de Sistemas Embebidos están orientados a personas con o sin experiencia previa, como docentes secundarios o universitarios, personal de empresas o estudiantes, que estén interesados en aprender a trabajar con sistemas embebidos. El objetivo de los cursos es aprender a realizar aplicaciones electrónicoas basadas en microcontroladores ARM de 32 bits, abordando desde problemas simples hasta problemas de mayor complejidad. Se utilizan kits compuestos por sensores y actuadores, y placas EDU-CIAA-NXP.

Además, cursos similares, y con objetivos alineados a los de los CAPSE, se brindaron a pedido del Instituto Nacional de Educación Tecnológica (INET).

En la última edición de estos cursos se utilizó CIAABOT IDE para las primeras clases, como manera de introducir a los asistentes a los conceptos básicos de la programación, así como también para obtener resultados rápidos.

Esta segunda experiencia de uso de CIAABOT en un entorno real de enseñanza fue realmente útil. Se obtuvieron recomendaciones y se registraron errores descubiertos a raíz de casos de uso que no se tuvieron en cuenta a la hora del desarrollo. Todas estas mejoras y errores fueron registrados en la sección de *issues* del repositorio de CIAABOT.

Por ejemplo, gran parte de los asistentes poseían netbooks con pantallas de resolución limitada, lo que hacía algo dificultoso el manejo de la vista partida que presenta el editor gráfico del IDE en conjunto con la salida del código C. Además, la sección de proyectos recientes y el formulario de 'Nuevo Proyecto' no se mostraban de manera cómoda. A raíz de esto, se comenzó un proceso de mejora de la

adaptación de la aplicación ante cambios de resoluciones de pantalla, a lo que se refiere en inglés como *responsiveness*.

4.4. Uso en un proyecto de brazo robótico

El Laboratorio Abierto de la Universidad Tecnológica Nacional Facultad Regional Avellaneda, comenzó este año el desarrollo de una prótesis de brazo y mano robótica, impresas en 3D. La mano cuenta con movilidad en la muñeca y las falanges de los dedos, logrando así una funcionalidad similar a la de la mano humana. El brazo puede observarse en la figura 4.5.

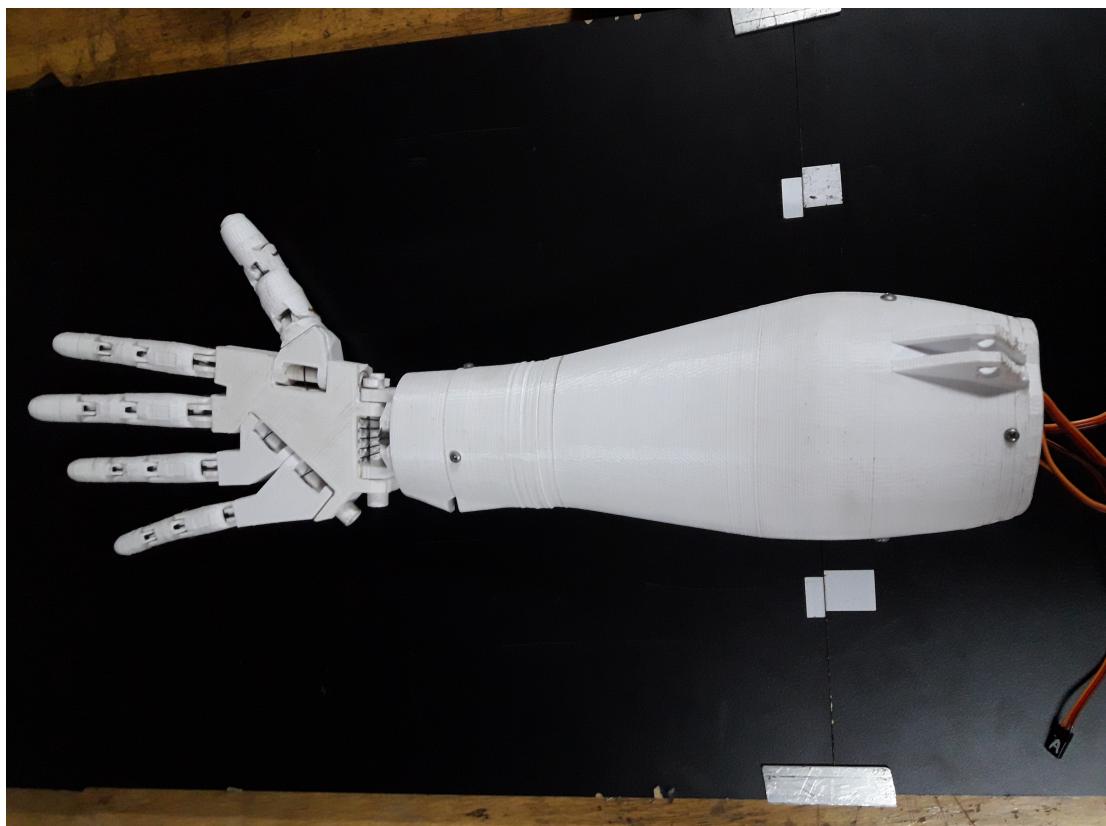


FIGURA 4.5: Prótesis impresa en 3D.

El objetivo principal del mismo, es poder realizar un aporte a la mejora de la calidad de vida de personas con amputaciones parciales de brazo y mano. Se utilizan las señales bioeléctricas obtenidas superficialmente desde el músculo del usuario para luego transformarla en una señal eléctrica capaz de manipularse, con el fin de controlarla. Por tal motivo, el faltante de miembro del paciente no debe ser total, para permitir una colocación de 3 electrodos.

Actualmente el proyecto busca migrarse a microcontroladores de 32 bits, y se optó utilizar la placa EDU-CIAA-NXP. Esto es debido a que las mediciones de los electrodos sirven para realizar un control rudimentario y detectar si el músculo está contraído o no, pero no alcanza para obtener una señal proporcional a la fuerza ejercida para emular este movimiento en la prótesis.

La idea es mejorar algoritmo de detección de fuerza en el músculo, realizando un análisis en el dominio de la frecuencia con el objetivo de eliminar ruidos propios que se generan por el tipo de contacto que tienen los sensores. Para esto, se pretende hacer uso de la unidad de punto flotante presente en el LPC4337. Para las primeras pruebas de manejo de hardware y lectura de los sensores musculares, se utilizaron programas armados en CIAABOT IDE.

Capítulo 5

Conclusiones

Este capítulo final resume el trabajo que se realizó hasta este punto, junto con su evaluación y conclusiones, y se esbozan las posibilidades de un trabajo futuro que continúe el proyecto.

5.1. Trabajo realizado

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

5.2. Trabajo futuro

Acá se indica cómo se podría continuar el trabajo más adelante.

Bibliografía

- [1] CIAA Firmware v2. 2016. URL: https://github.com/ciaa/firmware_v2.
- [2] Diseño del poncho CIAABOT G1. 2017. URL: <https://github.com/leandrolanzieri/ciaabot-g1-poncho>.
- [3] Documentación de Angular. URL: <https://angular.io/docs>.
- [4] Documentación de Blockly. URL: <https://developers.google.com/blockly/>.
- [5] Documentación de CIAABOT. 2017. URL: <http://leandrolanzieri.github.io/ciaabot-ide/documentacion>.
- [6] Free Software Foundation. GNU General Public Licence v3. 2007. URL: <https://www.gnu.org/licenses/gpl-3.0.en.html>.
- [7] Lenguaje de dominio específico. URL: https://es.wikipedia.org/wiki/Lenguaje_de_dominio_espec%C3%ADfico.
- [8] Licencia MIT. URL: <https://es.wikipedia.org/wiki/Licencia/MIT>.
- [9] The Linux Information Project. BSD License Definition. 2004. URL: <http://www.linfo.org/bsdlicense.html>.
- [10] Proyecto BlocklyDuino. URL: <https://github.com/BlocklyDuino/BlocklyDuino>.
- [11] Proyecto CIAA. URL: <http://www.proyecto-ciaa.com.ar>.
- [12] Repositorio CIAABOT IDE. 2017. URL: <https://github.com/leandrolanzieri/ciaabot-ide>.
- [13] sAPI. 2017. URL: <https://github.com/epernia/sAPI>.
- [14] Sobre Electron. URL: <https://electron.atom.io/docs/tutorial/about/>.
- [15] Sobre NodeJS. URL: <https://nodejs.org/en/about/>.
- [16] Wiki del proyecto CIAA. URL: <http://www.proyecto-ciaa.com.ar>.