

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



MEMORIA DEL TRABAJO FINAL

## **Depuración interactiva en CIAABOT**

**Autor:**  
**Ing. Jenny Chavez**

Director:  
Esp. Ing. Eric Pernía (UNQ, FIUBA)

CoDirector:  
Esp. Ing. Leandro Lanzieri Rodríguez (FIUBA)

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

*Este trabajo fue realizado en las Ciudad Autónoma de Buenos Aires, entre marzo de 2018 y diciembre de 2018.*



## *Resumen*

En la presente memoria se describe la implementación de un *debugger* interactivo para CIAABOT, que consiste en un entorno de programación mediante un lenguaje gráfico que se utiliza como una herramienta para la educación. El objetivo es mejorar la plataforma CIAABOT agregándole la capacidad a sus usuarios de identificar rápidamente los errores de programación de las plataformas de hardware en tiempo real, mediante el control desde la PC en la ejecución del programa y la observación del estado de sus variables.

Para cumplir con los objetivos planteados se aplicaron técnicas de diseño de programación y modularización de los servicios, herramientas de desarrollo de control de versiones, protocolos de comunicación, test unitario e integración continua.



## *Agradecimientos*

A Dios, que sin su apoyo no habría sido posible cumplir este sueño.

A mi familia, por darme la fuerza para continuar.

A mi director de proyecto final, Eric Pernía, por su paciencia, ayuda y entusiasmo.



# Índice general

<b>Resumen</b>	<b>III</b>
<b>1. Introducción General</b>	<b>1</b>
1.1. Introducción	1
1.1.1. CIAABOT	1
1.1.2. <i>Debug y Debugger</i>	2
1.2. Motivación	3
1.3. Objetivos y alcance	3
<b>2. Introducción Específica</b>	<b>5</b>
2.1. Componentes CIAABOT	5
2.1.1. CIAABOT IDE	5
2.1.2. CIAABOTS	7
2.1.3. Firmware	7
2.2. Requerimientos	8
2.2.1. Requerimientos asociados con el Proyecto CIAABOT	8
2.2.2. Componentes establecidos en CIAABOT	8
2.2.3. Firmware	8
2.2.4. Procesos Finales	8
2.3. Planificación	9
2.3.1. Desglose en tareas	9
2.3.2. Activity On-node	11
2.3.3. Diagrama de Gantt	12
<b>3. Diseño e Implementación</b>	<b>15</b>
3.1. Descripción general	15
3.2. Caso de estudio: depuración con eclipse	15
3.2.1. Firmata	16
3.3. Sistema para depuración propuesto	17
3.4. Características del entorno CIAABOT debug	18
3.5. Interfaz gráfica de CIAABOT Debug	18
3.6. Máquina de estados de CIAABOT Debug	19
3.6.1. Desconectado	19
3.6.2. Conectando	19
3.6.3. Conectado Sin Firmata4CIAA	21
3.6.4. Descargando Firmata4CIAA	22
3.6.5. Conectado Con Firmata4CIAA	22
3.6.6. Conectado En Sesión De Depuración	22
3.7. Sesión de depuración	24
3.7.1. Iniciar/detener sesión	24
3.7.2. Herramientas de control de ejecución	25
3.7.3. Menú de visualización	27
3.8. Edición de programa	28

3.9. Implementación . . . . .	28
3.9.1. Implentación de la GUI . . . . .	29
3.9.2. Herramientas utilizadas . . . . .	30
3.9.3. Archivo de estado de CIAABOT Debug . . . . .	32
3.9.4. GitLab . . . . .	33
<b>4. Ensayos y Resultados</b>	<b>35</b>
4.1. Pruebas funcionales del hardware . . . . .	35
<b>5. Conclusiones</b>	<b>37</b>
5.1. Conclusiones generales . . . . .	37
5.2. Próximos pasos . . . . .	37
<b>Bibliografía</b>	<b>39</b>



# Índice de figuras

1.1. Área de flujo de trabajo de CIAABOT-IDE. . . . .	2
2.1. Componentes CIAABOT. . . . .	5
2.2. Editor gráfico de CIAABOT-IDE. . . . .	6
2.3. Diagrama del funcionamiento principal de la plataforma CIAABOT. . . . .	7
2.4. Placa EDU-CIAA-NXP . . . . .	9
2.5. Diagrama de Node. . . . .	11
2.6. Tabla de colores diagrama Activity . . . . .	12
2.7. Diagrama de Gantt - Parte 1. . . . .	12
2.8. Diagrama de Gantt - Parte 2. . . . .	13
2.9. Diagrama de Gantt - Parte 3. . . . .	13
2.10. Diagrama de Gantt - Parte 4. . . . .	14
3.1. Modelo conceptual. . . . .	15
3.2. Uso de firmata con la Educiaa. . . . .	17
3.3. Composición de la intergaz gráfica de CIAABOT Debug. . . . .	18
3.4. Composición de la intergaz gráfica de CIAABOT Debug. . . . .	19
3.5. Diagrama de máquina de estados de los modos de funcionamiento. . . . .	20
3.6. Estado Donectado. . . . .	21
3.7. Descargar firmata4CIAA. . . . .	21
3.8. Configuración de paths. . . . .	22
3.9. Estado Conectado con firmata4CIAA. . . . .	23
3.10. Estado Depurando. . . . .	24
3.11. Estados de los botones de depuración. . . . .	24
3.12. Herramientas de control de ejecución habilitada. . . . .	24
3.13. Resaltado del flujo de control (Parte 1). . . . .	26
3.14. Resaltado del flujo de control (Parte 2). . . . .	26
3.15. Resaltado del flujo de control (Parte 3). . . . .	26
3.16. Establecer una bandera de punto de interrupción. . . . .	27
3.17. Punto de interrupción. . . . .	27
3.18. Quitar Punto de interrupción. . . . .	27
3.19. Desactivar los Puntos de interrupción activos. . . . .	27
3.20. Menú de visualización de variables. . . . .	27
3.21. Ventana de Puntos de ruptura ( <i>breakpoints</i> ). . . . .	28
3.22. Diágrama de Bloques para el manejo de los periféricos. . . . .	29
3.23. Ejemplo de código en bloques para el barrido de un servo. . . . .	29
3.24. Barrido de servomotor ejecutado con johnny five . . . . .	30
3.25. Ejemplo de wait . . . . .	30
3.26. Diagrama de secuencia. . . . .	31



# Índice de Tablas

3.1. Comandos de control de ejecución. . . . .	25
------------------------------------------------	----







# Capítulo 1

## Introducción General

En este capítulo se presenta una breve introducción a la plataforma CIAABOT, y se indica la necesidad que dio origen a desarrollar este proyecto. También se detallan los motivos que llevaron a realizarlo, cuáles son los objetivos y el alcance.

### 1.1. Introducción

En el presente trabajo se describe la implementación de un *debugger* interactivo para CIAABOT IDE<sup>1</sup>.

CIAABOT es una plataforma de software y hardware abierto que forma parte del proyecto CIAA (Computadora Industrial Abierta Argentina)[1].

La implementación de este trabajo tiene la misión de contribuir a la formación académica de los alumnos, así como también brindar un aporte al proyecto CIAA. De esta manera se desarrolla el presente proyecto para que los usuarios que se encuentren realizando sus desarrollos de programas en CIAABOT IDE, tengan también, como alternativa la depuración.

En la sección 1.1.1 se introduce el proyecto CIAABOT y las partes que lo componen; también se listan los pasos que son parte del flujo de trabajo en el IDE. En la sección 1.1.2 se exponen los conceptos involucrados en el contexto del desarrollo del presente proyecto, y en la sección 1.2 se introducen los motivos que llevaron a realizarse. Por último, en la sección 1.3 se presentan los objetivos propuestos y el alcance.

#### 1.1.1. CIAABOT

CIAABOT es una plataforma de robótica educativa. Tiene como propósito principal introducir de forma sencilla la programación de robots.

Las partes componentes que lo integran son:

- CIAABOT IDE: es un entorno de programación en lenguaje CIAABOT (basado en Blockly[2]), que es un lenguaje gráfico donde un programa se crea encastrando bloques. Permite crear el programa, compilarlo y descargarlo a las plataformas de hardware.

---

<sup>1</sup>IDE son las siglas en inglés de entorno de desarrollo integrado.

- CIAABOTS: son las plataformas de hardware que se programan desde CIAA-BOT IDE.
- Firmware: el programa creado en CIAABOT IDE genera internamente código C que se combina con el firmware pre-existente del proyecto CIAA y se compila para su posterior descarga a la plataforma.

Se observa en la figura 1.1 el área en dónde se desarrolla el flujo de trabajo, las cuales son:

1. Armar un programa con los bloques: encastrando bloques predefinidos por la plataforma.
2. Visualizar el código generado en C: actualizado en tiempo real cuando se manipulan los bloques.
3. Compilar: a partir del código sintácticamente correcto se traduce al código para usar en la placa.
4. Descargar: instalar el código generado en la plataforma de hardware.
5. Usar la placa con el programa: comenzar a realizar los ensayos sobre la placa.

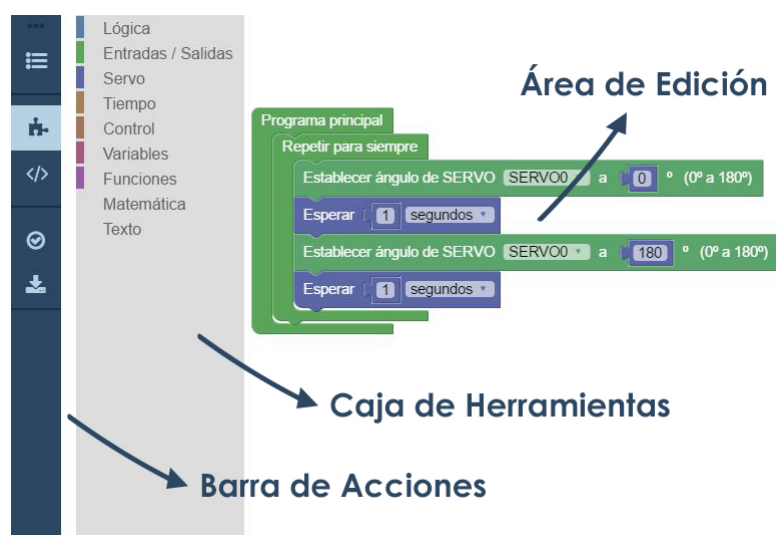


FIGURA 1.1: Área de flujo de trabajo de CIAABOT-IDE.

### 1.1.2. Debug y Debugger

El depurador es un programa que ejecuta ciertas rutinas de código de otros programas (programa objetivo), con el fin de encontrar y eliminar errores de ejecución. Esta técnica permite al código ser examinado y obtener información sobre el estado en el que se está ejecutando.

Mediante el proceso de depuración de programas, se puede identificar y corregir errores propios de programación, corriendo un programa paso a paso, parándolo o pausándolo, de esta manera se realiza el seguimiento de valores de las variables, dando en consecuencia la posibilidad al programador de corregir errores y pulir el funcionamiento de su programa.



Como el software en general y los sistemas electrónicos se vuelven generalmente más complejos, se da importancia al desarrollo de técnicas y herramientas de depuración, precisamente por las ventajas que tiene, como son el detectar anomalías en cada paso, corregir y mejorar las funcionalidades.

La mayoría de las computadoras modernas tienen el soporte de su hardware para realizar la depuración, en el caso de la EDU-CIAA-NXP cuenta con un puerto USB para realizar la depuración de un programa en el microcontrolador. El *debugger* utilizado en el IDE-Eclipse puede realizar la depuración conectándose al puerto USB.

## 1.2. Motivación

La plataforma CIAABOT no posee ninguna forma que permita *debuggear* un programa. Para realizar los ensayos se tiene que compilar el programa, descargarlo y luego manualmente probar la lógica en la placa usando, como por ejemplo, los leds, botones, mensajes por UART, etc.

Se plantea el desarrollo de un *debugger* interactivo para CIAABOT, debido a la importancia de realizar pruebas de un programa en tiempo real, y de esta manera verificar la lógica del programa.

El desarrollo del *debugger* interactivo para CIAABOT, seguirá los lineamientos de diseño de CIAABOT, implementando el desarrollo con una visión de fácil utilización, simple e intuitivo, teniendo en cuenta quienes serán los usuarios finales.

Se aprovechará con el desarrollo de esta implementación poder aplicar todos los conocimientos aprendidos durante la carrera de especialización de sistemas embebidos.

## 1.3. Objetivos y alcance

El objetivo de este proyecto es brindarle al usuario una herramienta útil para la corrección e identificación de los errores de programación cuando está usando la plataforma CIAABOT. Para lograrlo se pretende desarrollar un *debugger* para CIAABOT que cumpla con las siguientes características:

- Ejecutar un programa línea a línea.
- Detener la ejecución temporalmente en un bloque encastrable concreto.
- Visualizar el contenido de las variables en un determinado momento de la ejecución.
- Entorno gráfico amigable, tooltips con valores sobre el código.
- Importar el programa de bloques creado en el IDE de desarrollo de CIAABOT.

De esta manera el aprendizaje del usuario programador primerizo será didáctica, identificando y subsanando los errores de ejecución.



## Capítulo 2

# Introducción Específica

En este capítulo se presenta los componentes de CIAABOT, con más detalle, luego se establecen los requerimientos y la planificación para el desarrollo del presente trabajo.

### 2.1. Componentes CIAABOT

La plataforma CIAABOT esta conformada por tres partes fundamentales, tal como se muestra en la figura 2.1.

A continuación se describirá en detalle cada una de las partes de CIAABOT.

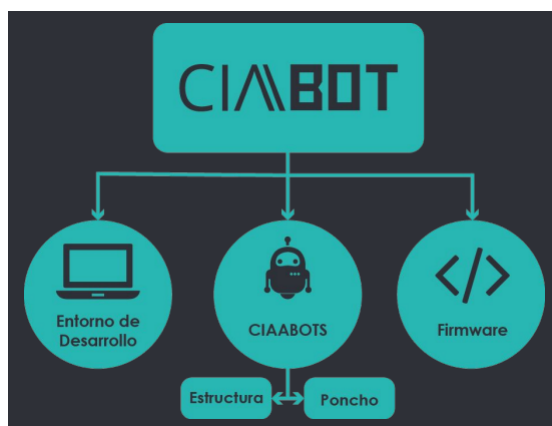


FIGURA 2.1: Componentes CIAABOT.

#### 2.1.1. CIAABOT IDE

El IDE esta basado en el paradigma reactivo que permite armar programas propios, que pueden ser programas puntuales como manejo de actuadores y motores en un sistema embebido, así como también, un prototipado rápido.

El IDE de CIAABOT tiene como componente principal al editor. A partir de allí el usuario puede desarrollar su propio programa gráfico encastrando de manera fácil bloques predefinidos creados en lenguaje javascript, como se observa en la figura 2.2

El entorno de desarrollo integrado permite de manera gradual ir comprendiendo como realizar el mismo programa en lenguaje C, debido a que permite ver en tiempo real el código C generado mientras se van encastrando los bloques.

Dentro del entorno se brinda al usuario una barra de herramientas con funcionalidades para crear un nuevo programa, compilarlo, y realizar la descarga del código en la placa conectándola por USB.

El IDE proporciona al usuario la opción de guardar el programa creado en un archivo con extensión .cbp, el cual contiene toda la información del programa creado, que podría ser utilizado posteriormente.

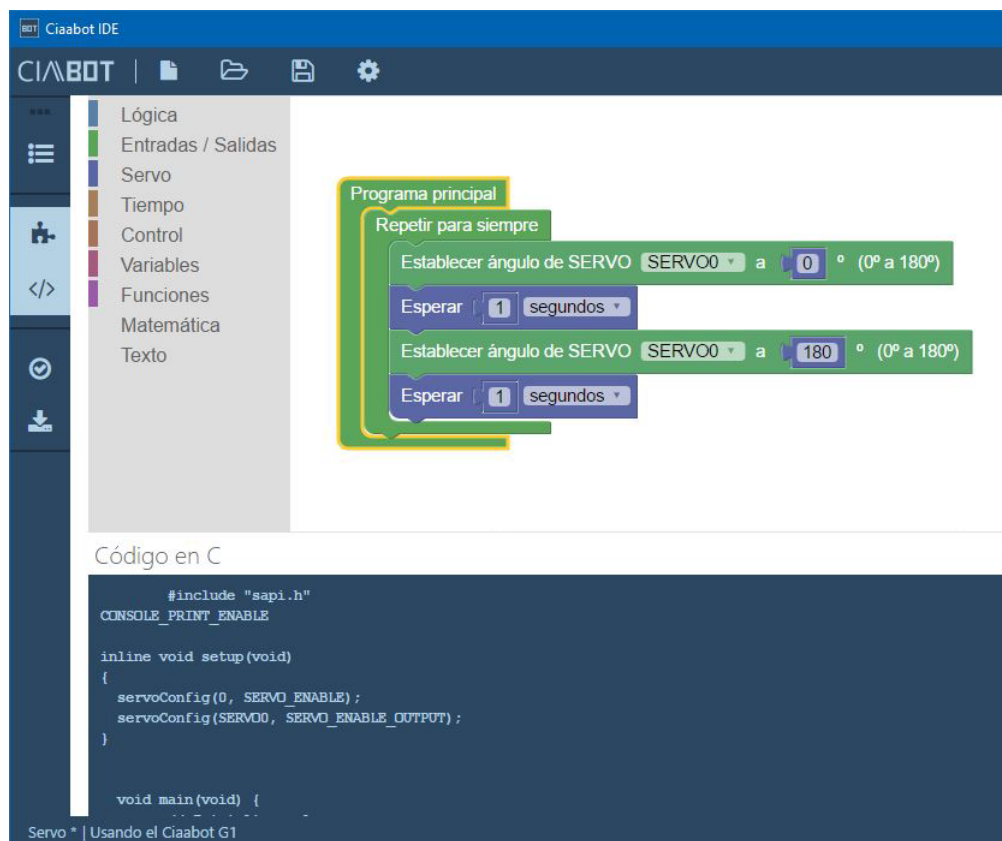


FIGURA 2.2: Editor gráfico de CIAABOT-IDE.

La plataforma CIAABOT esta programado usando las siguientes tecnologías:

- Angular[3]: es una plataforma usada para desarrollar aplicaciones web en HTML y JavaScript.
- Blockly[2]: es una biblioteca que utiliza bloques gráficos encastrables para representar conceptos de código.
- Electrón[4]: es una biblioteca de código usado para armar aplicaciones de escritorio multiplataforma utilizando tecnologías web.
- NodeJS[5]: es un entorno de ejecución de JavaScript que utiliza el motor V8 de Google.
- TypeScript[6]: es un superconjunto de JavaScript, esencialmente añade tipado estático y objetos basados en clases.

El funcionamiento principal en la plataforma CIAABOT es la siguiente:

- Genera el archivo principal: archivo main.c
- Genera el código en C: cada uno de los bloques genera el correspondiente código en C y lo inyecta en el archivo main.c del programa.
- Permite la ejecución del compilador de C para el código en la placa. Invoca al OpenOCD.
- Descarga el binario compilado en flash y realiza el reset.

En la figura 2.3 se expone el funcionamiento principal de la plataforma CIAABOT.

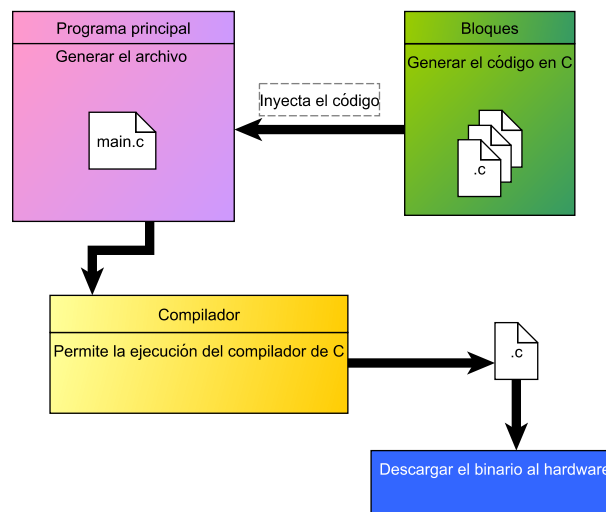


FIGURA 2.3: Diagrama del funcionamiento principal de la plataforma CIAABOT.

### 2.1.2. CIAABOTS

Se llama CIAABOTS a los robots que se pueden programar utilizando CIAABOT IDE. Estos CIAABOTS tienen un diseño estructural de impresión en 3D, correspondiente al modelo de la placa CIAA.

Para ser usados por las impresoras 3D, es necesario armar el poncho de diseño abierto y tener los sensores y actuadores del CIAABOT a imprimir.

### 2.1.3. Firmware

Debido a que la plataforma CIAABOT está basada en el firmware v2 [7] del proyecto CIAA, puede crear desde funciones simples a más complejas, para realizarlo utiliza las siguientes herramientas:

- *Makefile*[8]: para la gestión de dependencias, de esta manera puede construir el software desde sus archivos fuente.
- *OpenOCD*[9]: una herramienta OpenSource, usado para el grabado del firmware en las placas.

- *sAPI*[10]: permite manejar los periféricos del microcontrolador de una manera muy sencilla.

## 2.2. Requerimientos

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

### 2.2.1. Requerimientos asociados con el Proyecto CIAABOT

- El entorno de programación deberá poder ejecutarse minimamente dentro del entorno Linux y Windows.
- El uso debe ser sencillo, rápido e intuitivo.
- El entorno de debugger debe tener un diseño basado en ventanas cómodas y que permitan tener mucha información a la vista.
- El entorno de debugger debe ocupar muy poca memoria.
- El diseño de la herramienta debe seguir los estilos de interfaz establecidos en el Proyecto CIAABOT.
- La herramienta debe poder permitir el monitoreo de las entradas y salidas de los diferentes periféricos de la placa.

### 2.2.2. Componentes establecidos en CIAABOT

- El presente proyecto deberá integrarse al entorno gráfico establecido que permita la programación de los robots.
- Se usará la placa EDU-CIAA-NXP (figura 2.4) para el control de los robots.

### 2.2.3. Firmware

- Se actualizará a la última versión, respetando el control de versiones establecido.
- Se utilizarán las principales bibliotecas firmata que permita interactuar con el cliente que está corriendo en la placa.
- Se contará con un mecanismo de ahorro de la flash, para saber si la placa ya tiene firmata.

### 2.2.4. Procesos Finales

- Se actualizará el manual de usuario, y se incluirán ejemplos básicos funcionales del entorno de depuración.
- Se utilizarán las principales bibliotecas firmata que permita interactuar con el cliente que está corriendo en la placa.

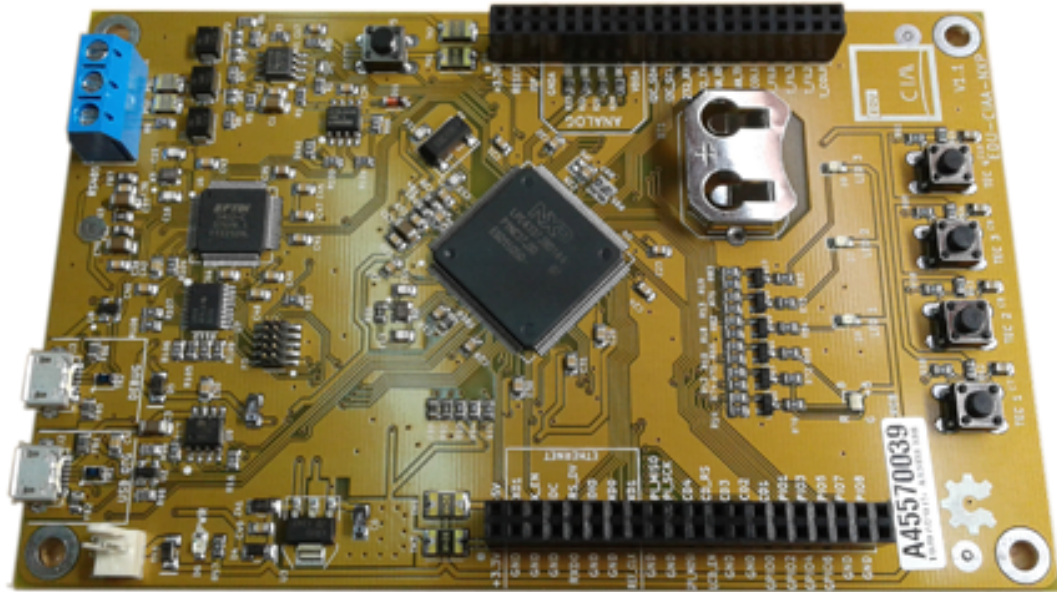


FIGURA 2.4: Placa EDU-CIAA-NXP

- Se implementará el *debugger* de la aplicación sobre una maqueta o robot adaptado a funcionar con la placa EDU-CIAA.
- Se evaluarán los resultados del proyecto y su facilidad de uso en ámbitos de enseñanza reales.

## 2.3. Planificación

Para lograr los objetivos propuestos, se realizó el desglose en tareas, y se utilizaron las herramientas del diagrama de Activity on-node y gantt donde se esquematiza esas tareas que son parte del trabajo.

### 2.3.1. Desglose en tareas

Para alcanzar objetivos concretos, se plantean los entregables para el proyecto:

- Entorno de Debugger.
- Código fuente del proyecto.
- Actualización del Manual de usuario, mostrando ejemplos didácticos del uso del debugger en la plataforma.
- El presente informe final.

Se estimó un tiempo aproximado de 600 horas, distribuidas en grupos de tareas de la siguiente manera:

1. Planificación del proyecto (60 hs.).
  - Plan del proyecto.
  - Análisis de requerimientos.

- Análisis técnico y de factibilidad.
  - Gestión de riesgos.
  - Gestión de calidad.
2. Investigación Preliminar (40 hs.).
    - Búsqueda de plataformas de robótica educativa existentes, que en su interfaz de desarrollo implemente la herramienta de debugging.
    - Búsqueda de frameworks de JavaScript, para la implementación de las funciones de firmata.
    - Búsqueda de información acerca de la ejecución de debugging multi-plataforma.
    - Búsqueda de intérpretes Javascript para el debugging.
    - Búsqueda de información de mecanismos de ahorro de la flash.
  3. Selección de Frameworks (35 hs.).
    - Selección y pruebas preliminares de la biblioteca firmata para JavaScript.
    - Selección y pruebas preliminares del intérprete Javascript.
    - Búsqueda de información acerca de la ejecución de debugging multi-plataforma.
    - Selección y pruebas del mecanismo de ahorro de la flash.
    - Evaluar la correcta integración entre la aplicación CIAABOT y el debugger.
  4. Desarrollo del Debugger (85 hs.).
    - Desarrollo de estructura amigable e intuitiva para su uso.
    - Desarrollo de estilos de componente compatibles a la aplicación CIAABOT.
    - Desarrollo de módulo de configuración de mensajes al moverse los diferentes periféricos de la placa.
    - Desarrollo del mecanismo de ahorro de la flash.
  5. Implementaciones de funciones firmata javascript (90 hs.).
    - Implementar los módulos para JavaScript encargados de obtener datos de cada sensor.
    - Implementar los módulos para JavaScript encargados de manejar los actuadores.
    - Desarrollo de funciones complementarias utilizando la API de JS Interpreter.
    - Integración de las bibliotecas de programación gráfica.
  6. Programación por Interfaz serie y Monitoreo Firmata (40 hs.).



- Desarrollo para mostrar mensajes en la interacción de los diferentes periféricos de la placa cuando está conectada por interfaz serie.
- Desarrollo de monitoreo en modo debug, de los estados de entradas y salidas a través de firmata con visualización en la aplicación.

#### 7. Pruebas de Firmware (60 hs.).

- Pruebas Unitarias.
- Pruebas de Integración.
- Corrección de errores encontrados.

#### 8. Integración del Sistema (60 hs.).

- Integración de la aplicación Ciaabot para el modo debug.
- Pruebas iniciales de todo el sistema Ciaabot.
- Corrección de errores encontrados.

#### 9. Procesos Finales (130 hs.).

- Modificar el manual de usuario, agregando el uso del modo debug.
- Redacción de memoria de trabajo.
- Evaluar el cumplimiento de requerimientos.
- Preparación de la presentación del proyecto.

### 2.3.2. Activity On-node

En el diagrama de Activity on node de la figura 2.5 se muestran todas las tareas propuestas que se planificaron para realizar el proyecto, junto con su respectivo tiempo estimado en días para cada tarea. Todas las flechas entrantes a un nodo o tarea son las dependencias de la misma.

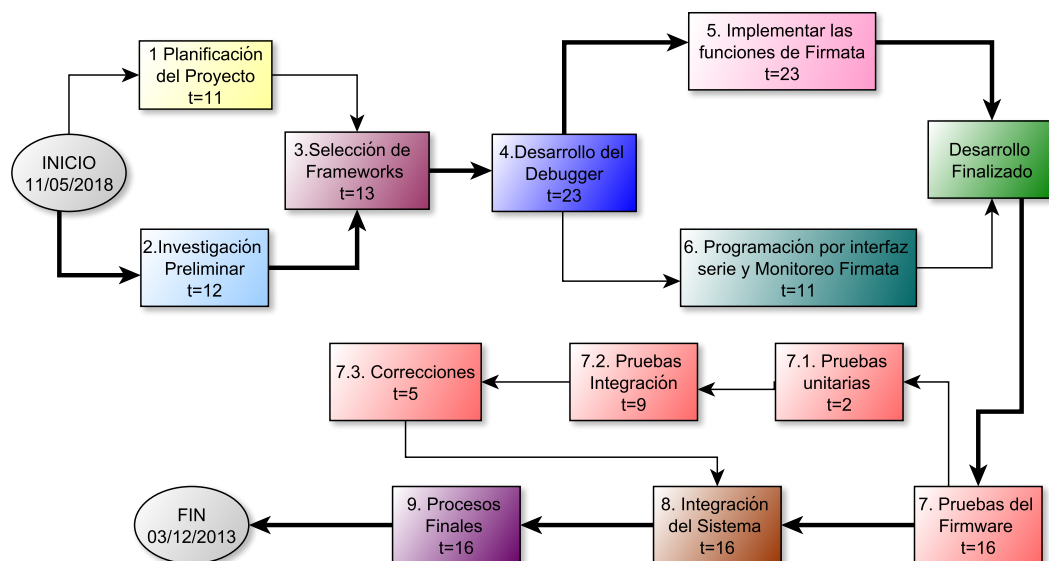


FIGURA 2.5: Diagrama de Node.

Los días están expresados en días laborales de aproximadamente 3 horas, y en días no laborales de aproximadamente 4 horas. A modo de referencia se muestra en la siguiente figura 2.6 una tabla de colores que se corresponde con cada una de las tareas.

Color	Tarea
	1. Planificación del Proyecto
	2. Investigación Preliminar
	3. Selección de Frameworks
	4. Desarrollo del Debugger dentro de la Aplicación de Escritorio
	5. Implementar las funciones de Firmata para JavaScript
	6. Programación por interfaz Serie y Monitoreo Firmata
	7. Pruebas del Firmware
	8. Integración del Sistema
	9. Procesos Finales

FIGURA 2.6: Tabla de colores diagrama Activity

### 2.3.3. Diagrama de Gantt

El diagrama de Gantt permite tener una referencia rápida de dónde se debería encontrar el desarrollo del proyecto según la planificación inicial. Por lo tanto, como parte de la planificación del proyecto, se definieron las tareas necesarias para completar el trabajo y se establecieron las relaciones de correlatividad entre ellas, teniendo en cuenta su duración.

En la figura 2.7 se puede observar la primera parte del diagrama para este proyecto. Las horas en la duración de cada una de las tareas están expresadas en días laborales y no laborales.

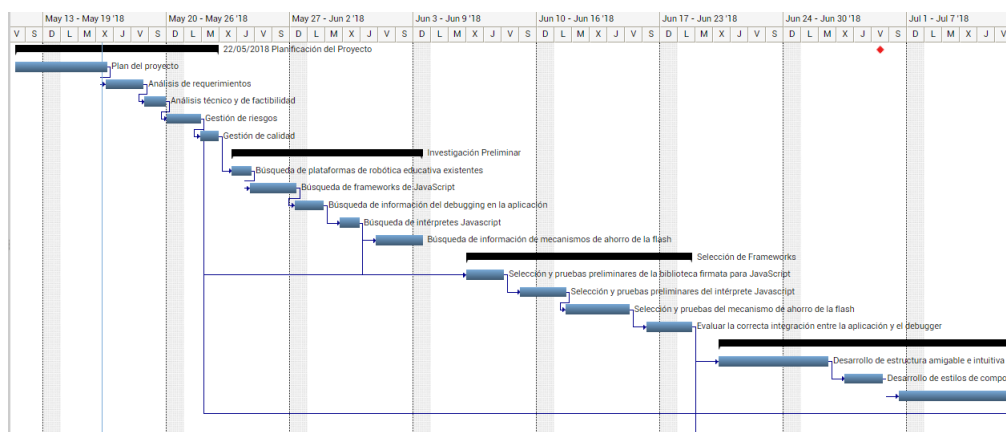


FIGURA 2.7: Diagrama de Gantt - Parte 1.

En la figura 2.8 se puede observar la segunda parte del diagrama para este proyecto.

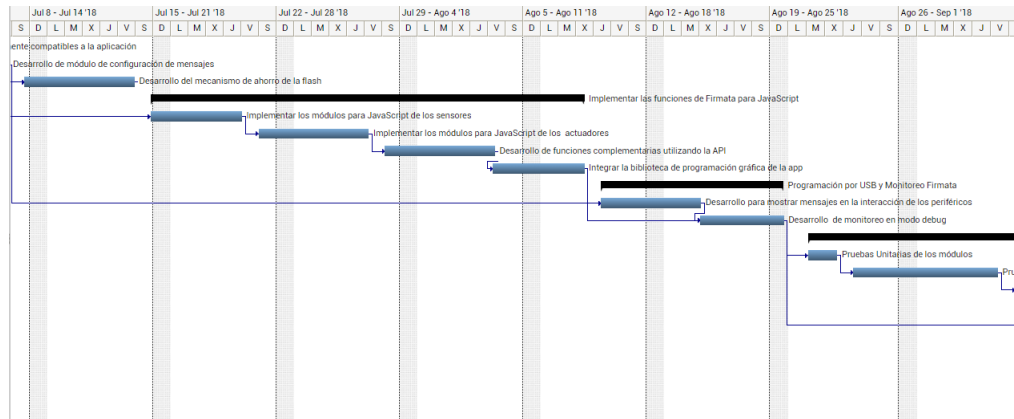


FIGURA 2.8: Diagrama de Gantt - Parte 2.

En la figura 2.9 se puede observar la tercera parte del diagrama para este proyecto. Y la cuarta parte del diagrama se puede observar la figura 2.10

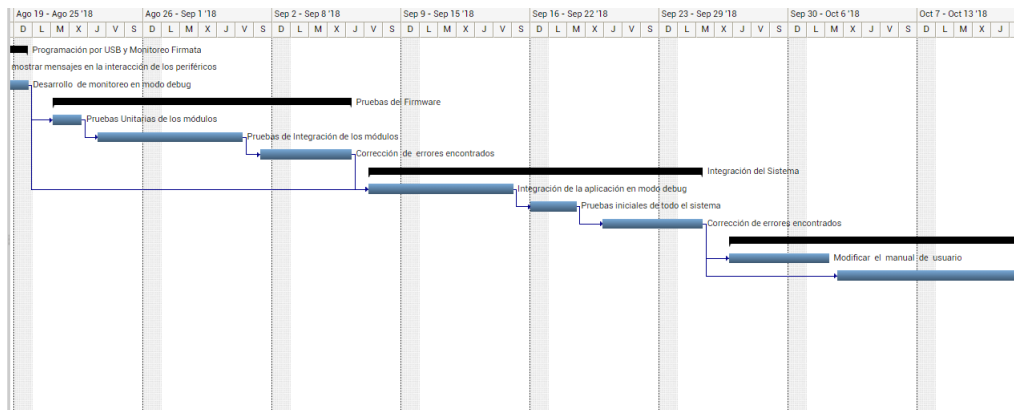


FIGURA 2.9: Diagrama de Gantt - Parte 3.

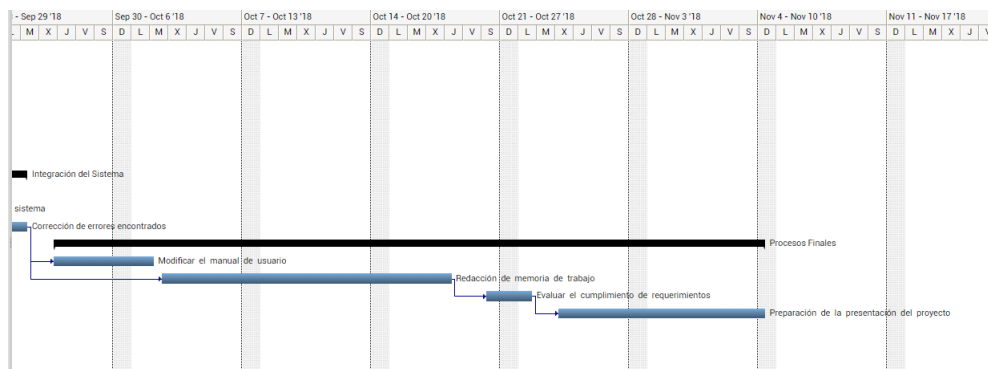


FIGURA 2.10: Diagrama de Gantt - Parte 4.

## Capítulo 3

# Diseño e Implementación

En este capítulo se presenta como caso de estudio el funcionamiento del software de depuración para programas en C mediante eclipse y se describe en detalle el desarrollo del diseño e implementación del Sistema para depuración elegido.

### 3.1. Descripción general

El sistema se compone de una computadora ejecutando la depuración de un programa en lenguaje CIAABOT, y la placa EDUCIAA conectado a la computadora, por medio de una interfaz, desde donde el IDE del *debug* realice la comunicación con el hardware como se muestra en la figura 3.1.



FIGURA 3.1: Modelo conceptual.

### 3.2. Caso de estudio: depuración con eclipse

Para diseñar el software de depuración se toma como caso de estudio la depuración de programas escritos en lenguaje C con Eclipse IDE sobre la EDU-CIAA-NXP:

- *Plugin de eclipse arm-none-eabi-gdb*<sup>1</sup>: realiza la interpretación de comandos de GDB-MI y muestra los resultados en la interfaz gráfica del editor de texto de C en el IDE del Eclipse.

---

<sup>1</sup>Software de configuración de eclipse para usar el depurador GNU para procesadores ARM Cortex-A/R/M.

- *arm-none-eabi-gdb*<sup>2</sup>: realiza el mapeo de los símbolos de C con las instrucciones en código máquina que se ejecutan en el microcontrolador (mapea las funciones al código binario en flash), para luego ejecutar los comandos de parar, continuar o de uso de breakpoints.
- *OpenOCD (Open On-Chip Debugger)*<sup>3</sup>: provee a GDB el remote interface protocol para permitirle acceder al hardware. Traduce transacciones JTAG o SWD (mediante el puerto serie sobre USB) a comandos remote-protocol de GDB. OpenOCD utiliza scripts de configuración (archivos \*.cfg) donde se describe el microcontrolador a depurar y la interfaz de hardware para acceder al mismo (en adelante "*Debugger HW*").
- *Debugger HW*: en el caso de la EDU-CIAA es el circuito de interfaz física para pasar de JTAG a USB (modo puerto serie virtual). En la EDU-CIAA viene incluido en la misma placa que esta el microcontrolador a depurar.
- *Microcontrolador a depurar*: El microcontrolador posee un periférico específico para *debug* con interfaz JTAG<sup>4</sup>, teniendo acceso para modificar la RAM, Flash y los registros del Microcontrolador.

Una alternativa para la programación de un software de depuración a nivel de bloques de CIAABOT es, entonces, mantener el mapeo entre los bloques de programa de CIAABOT y su C generado y comunicarse con GDB, mediante GDB-MI como lo realiza el plugin de Eclipse.

Teniendo en cuenta la complejidad de implementar lo expuesto, se propone como una alternativa factible, el de emular la funcionalidad de debugger mediante la ejecución del programa de CIAABOT en la propia PC, de manera que al momento de ejecutarse los bloques gráficos de acceso a los periféricos, se realice la comunicación con la placa mediante un protocolo, y de esta manera realizar la ejecución de comandos de lectura y escritura en los periféricos que se requiera manipular.

Como protocolo de comunicación para acceso al hardware se elije utilizar el protocolo firmata, debido a que existe una implementación del mismo para la EDU-CIAA-NXP [7] y existen múltiples bibliotecas firmata para diferentes lenguajes de programación en la PC.

### 3.2.1. Firmata

Firmata es un protocolo genérico y abierto, fue diseñado para la comunicación directa entre un microcontrolador previamente instalado con un programa que implementa firmata y un objeto de software que implementa un cliente firmata en una computadora host.

El protocolo se puede implementar en cualquier arquitectura de microcontroladores, así como en cualquier paquete de software.

---

<sup>2</sup>Software de debug originario de linux.

<sup>3</sup>Software de código abierto que interactúa con el puerto JTAG de un depurador de hardware.

<sup>4</sup>Acrónimo de Joint Test Action Group, es utilizado como mecanismo para depuración de sistemas embebidos, proveendo una puerta trasera para acceder al sistema.

Firmata4CIAA es un programa que implementa el protocolo firmata en la EDU-CIAA-NXP. En la figura 3.2 se muestra el uso de firmata con la Educiaa.

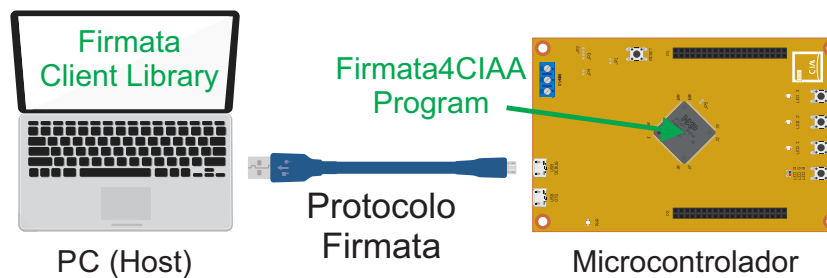


FIGURA 3.2: Uso de firmata con la Educiaa.

### 3.3. Sistema para depuración propuesto

El funcionamiento del entorno propuesto se puede resumir en los siguientes pasos:

- Ejecutar un programa bloque a bloque, usando Javascript internamente, hasta que se encuentre con un bloque que requiera acceso a algún periférico del hardware.
- Enviar comandos firmata a la placa cuando se encuentre un bloque de acceso a algún periférico.
- Visualizar el contenido de las variables en un determinado momento de la ejecución.
- Instalación del programa que implementa el protocolo firmata en la CIAA, sólo en el caso de ser necesario.

Cabe destacar que los bloques gráficos que acceden a los periféricos, serán más lentos que si se tuviera que implementar mediante GDB-MI, ya que ese programa accedería al hardware a la velocidad en la que se llama a las instrucciones del hardware, debido al código binario compilado instalado en la placa.

Por el contrario, los bloques gráficos que no acceden a los periféricos, en tiempo de ejecución, serán más rápidos, debido a que se ejecutarán directamente en la misma PC del usuario. El programa se ejecutará dentro del contexto del VM de javascript, que es más rápido que el código c compilado en el microcontrolador.

De esta manera si se miden los tiempos de ensayos, cuando se instala el firmware del código C generado en la placa contra la ejecución de los bloques con firmata, estos tiempos no serán los mismos.

Una ventaja para el usuario de CIAABOT, es que podrá depurar el programa sin necesidad de esperar la generación del código C a partir del programa en bloques, su compilación (especialmente en Windows) y descarga a la plataforma reduciendo los tiempos de desarrollo del programa.

### 3.4. Características del entorno CIAABOT debug

El Entorno de Programación de CIAABOT debug debe ser capaz de permitirle al usuario realizar las siguientes tareas:

- Abrir o importar un programa realizado en CIAABOT IDE.
- Ejecutar un programa bloque a bloque.
- Establecer puntos de interrupción (puntos de parada en los bloques).
- Permitir la visualización del valor de las variables durante la ejecución del programa.
- Ejecución del programa con o sin puntos de interrupciones activos (*break-points*).
- Pausar y Parar la ejecución del programa desde cualquier momento dado.
- Instalación del programa que implementa el protocolo firmata en la EDU-CIAA-NXP, sólo en el caso de ser necesario.

Los proyectos creados en CIAABOT IDE son guardados como archivos con extensión .cbp, la misma contiene la información del proyecto, su nombre, el modelo de CIAABOT utilizado y el diagrama en bloques. Un proyecto puede abrirse desde el entorno de *debug*, la información del proyecto es cargado y el diagrama en bloques es mostrado.

### 3.5. Interfaz gráfica de CIAABOT Debug

La figura 3.4 expone la composición de la interfaz gráfica de CIAABOT *debug*.

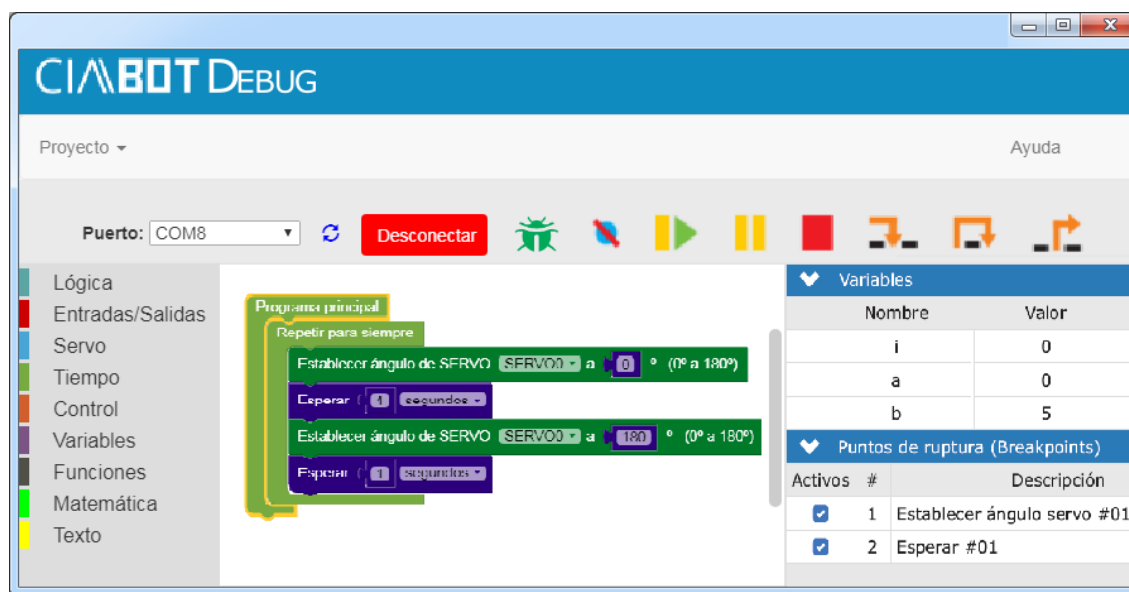


FIGURA 3.3: Composición de la interfaz gráfica de CIAABOT Debug.

Se compone de las siguientes partes:



- Conexión con la plataforma.
- Editor de programa.
- Iniciar sesión de depuración.
- Herramientas de control de ejecución.
- Menú de visualización de variables y puntos de interrupción.

En la figura 3.4 se resaltan cada una de ellas:



FIGURA 3.4: Composición de la interfaz gráfica de CIAABOT Debug.

### 3.6. Máquina de estados de CIAABOT Debug

En la figura 3.5 se expone el diagrama de estados de el software CIAABOT debug.

#### 3.6.1. Desconectado

La aplicación inicia en estado *Desconectado*. En este modo de funcionamiento se permite la edición de bloques de programa, el botón de inicio de sesión de depuración se encuentra deshabilitado, como las herramientas de control de ejecución. Mientras no este depurando se habilita la edición del programa. La figura 3.6 expone la interfaz gráfica en este estado.

#### 3.6.2. Conectando

Cuando la aplicación esta en estado *Conectando* permite la edición de bloques de programa y verifica si la EDU-CIAA-NXP tiene descargado el programa firmata4CIAA, si lo tiene pasa a estado *Conectado Con Firmata4CIAA*, sino pasa a *Conectado Sin Firmata4CIAA*.

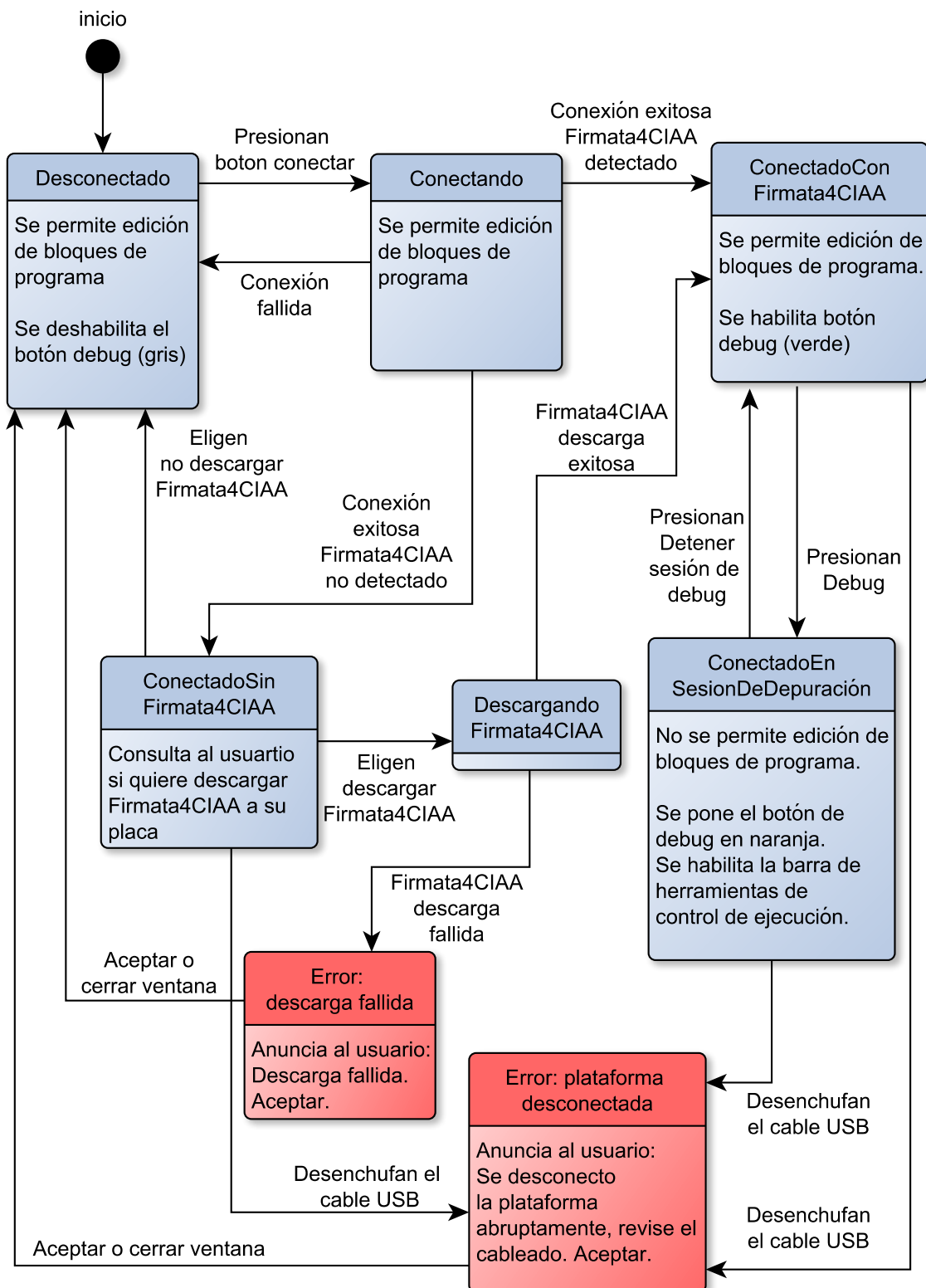


FIGURA 3.5: Diagrama de máquina de estados de los modos de funcionamiento.

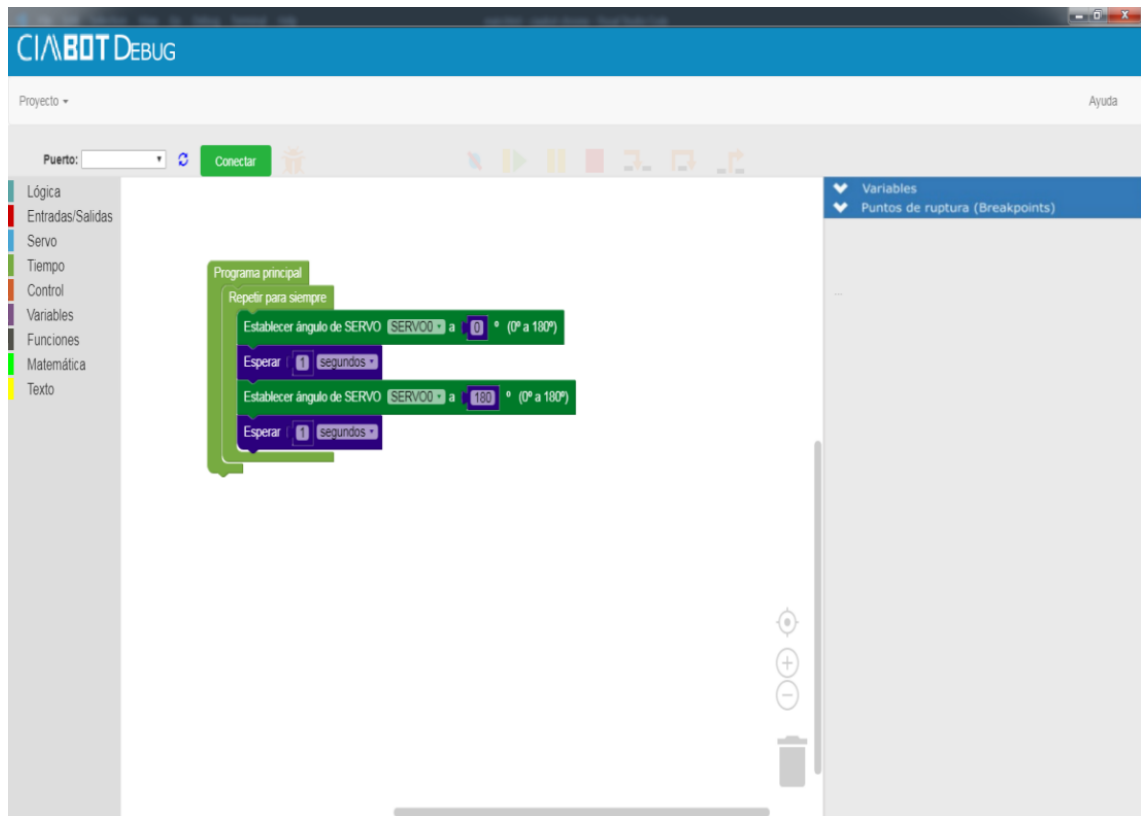


FIGURA 3.6: Estado Donectado.

### 3.6.3. Conectado Sin Firmata4CIAA

Si la aplicación paso a estado *Conectado Sin Firmata4CIAA*, realiza la consulta al usuario si quiere descargar el programa *firmata4CIAA*, si el usuario elige que sí, entonces se procede a realizar la descarga, y si elige que no entonces la aplicación vuelve al modo *Desconectado*.

En la figura 3.7 muestra la consulta enviada al usuario.

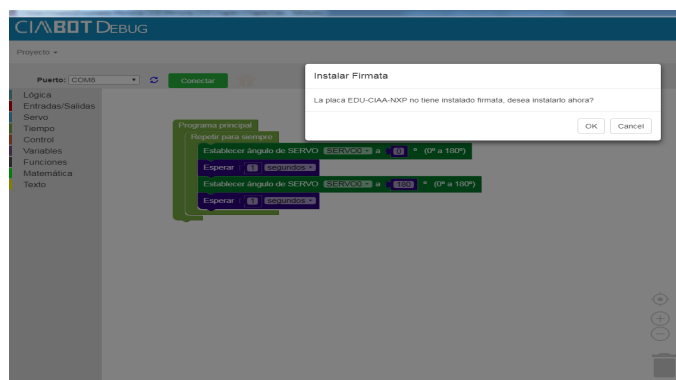


FIGURA 3.7: Descargar firmata4CIAA.

### 3.6.4. Descargando Firmata4CIAA

La aplicación pasa al estado *Descargando Firmata4CIAA* cuando el usuario eligió realizar la descarga del programa firmata4CIAA. Si en medio del proceso existiese algún problema de descarga, se avisará al usuario.

Entre los archivos de CIAABOT *Debug* se encuentra el proyecto de firmata4CIAA previamente compilado para la EDU-CIAA-NXP. Para descargarlo el entorno requiere configurar la ruta del toolchain y de OpenOCD. Con esa información simplemente ejecuta el siguiente comando en la ruta de dicho proyecto:

- make download.

La figura 3.8 muestra la ventana desde donde se puede realizar la configuración de paths.

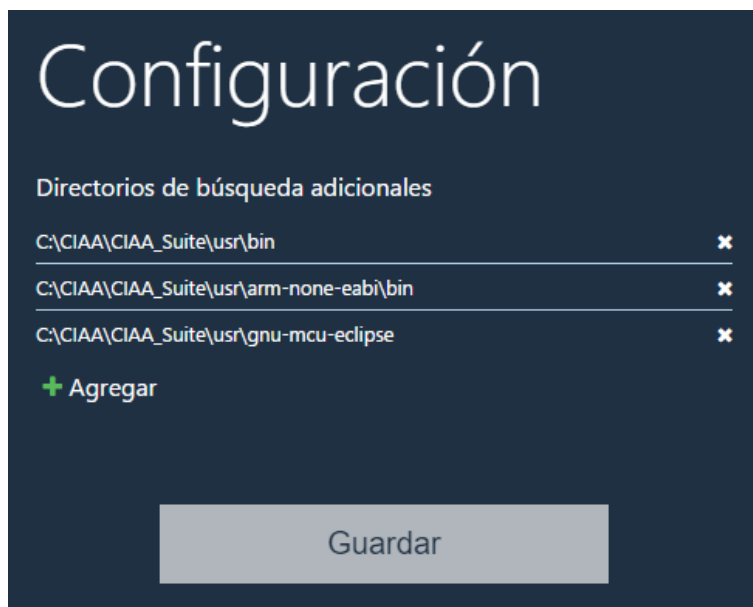


FIGURA 3.8: Configuración de paths.

### 3.6.5. Conectado Con Firmata4CIAA

Cuando la aplicación pasa al estado *Conectado Con Firmata4CIAA*, se habilita la edición de bloques de programa y también se habilita el inicio de sesión de depuración. En la figura 3.9 se muestra este modo de funcionamiento.

### 3.6.6. Conectado En Sesión De Depuración

En estado Conectado, y el usuario inicio la sesión de depuración, el entorno habilita la barra de herramientas de control de ejecución, permitiéndole realizar el ensayo de su programa sin poder editarlo. La figura 3.10 muestra la aplicación con este estado.

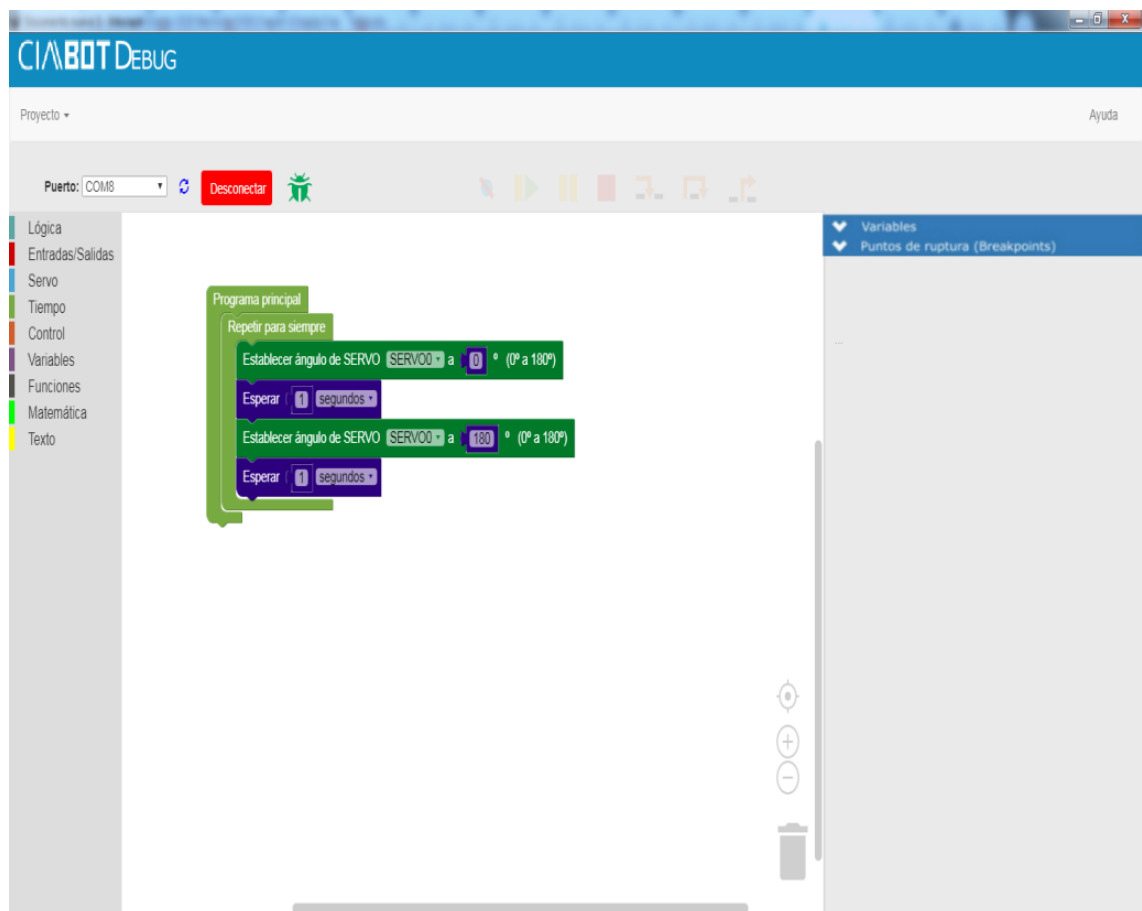


FIGURA 3.9: Estado Conectado con firmata4CIAA.

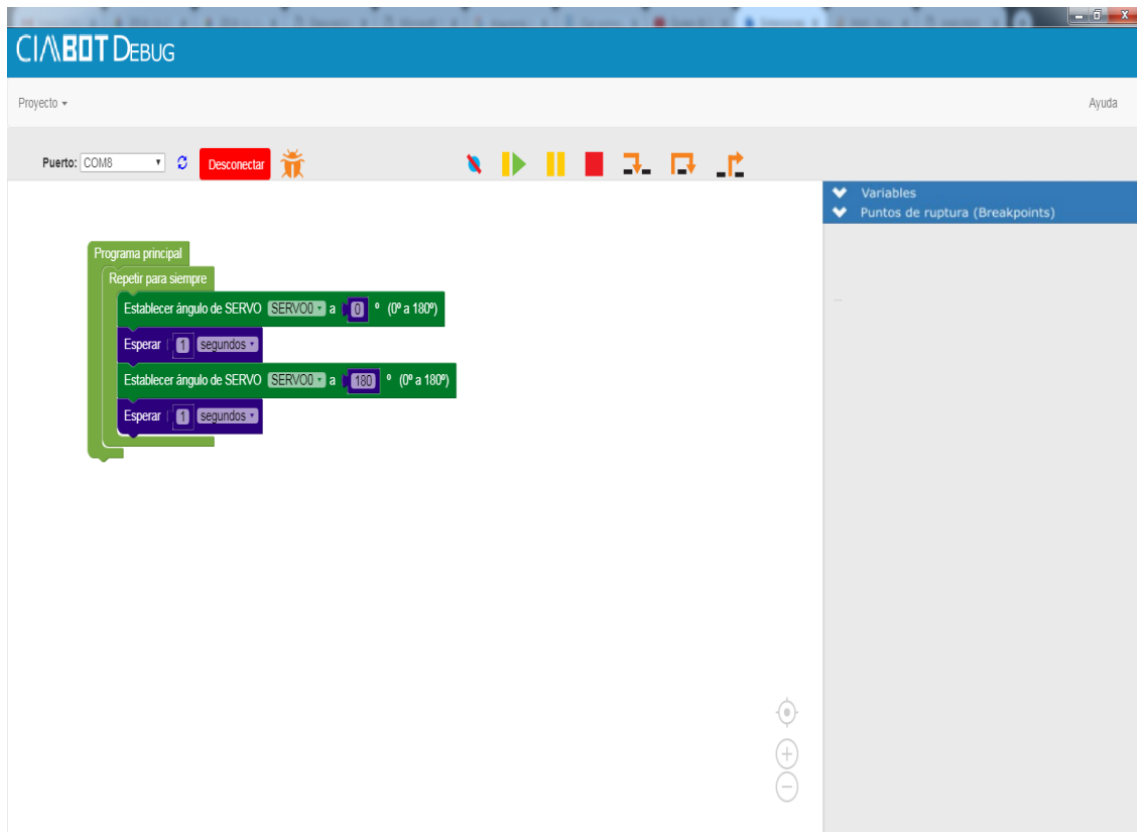


FIGURA 3.10: Estado Depurando.

## 3.7. Sesión de depuración

### 3.7.1. Iniciar/detener sesión

Cuando inicia la aplicación el icono de *debug* se muestra deshabilitado (color gris), como se muestra en la figura 3.11. Al tener la EDU-CIAA-NXP con firmata4CIAA, la aplicación estará lista para iniciar la sesión de depuración, cambiando el icono de *debug* de color gris a color verde (habilitado), de esta manera se permite al usuario hacer click, y luego de esta acción la sesión de depuración es iniciada y el entorno habilita las herramientas de control de ejecución como se muestra en la figura 3.12 y el icono de color verde habilitado pasa al color naranja de *debug*.



FIGURA 3.11: Estados de los botones de depuración.



FIGURA 3.12: Herramientas de control de ejecución habilitada.

La sesión de depuración finaliza cuando el programa termino de ejecutarse paso a paso o con puntos de ruptura (*breakpoints*), ó si el usuario presiono el comando de control de ejecución *Detener depuración*.

### 3.7.2. Herramientas de control de ejecución

En la tabla 3.1 se listan los comandos de control de ejecución junto a una breve descripción.








Comando	Nombre	Descripción
	Desactivar puntos de interrupción	Establece todos los puntos de interrupción para ser omitidos (Saltar todos los <i>breakpoints</i> ).
	Ejecutar	Ejecuta el programa y Reanuda el programa si fue suspendido
	Suspender	Pausa el programa con el objetivo de que se pueda examinar, inspeccionar datos, pasos, etc.
	Detener depuración	Termina la depuración del programa.
	Pasar adentro	Ingresa a ejecutar el código en caso de ejecutar un llamado a función.
	Pasar por encima	Continuar con el siguiente paso. La ejecución continuará en el siguiente bloque. En caso de que el bloque sea un llamado a función ejecutarlo sin ingresar al código de dicha función
	Paso de regreso	Ejecutar hasta encontrar un retorno de función. Esta opción detiene la ejecución después de salir de la función actual.

TABLA 3.1: Comandos de control de ejecución.

El usuario podrá ejecutar un programa haciendo click en el ícono *Ejecutar* de la barra de herramientas. De la misma manera para detener un programa en ejecución, podrá hacerlo mediante el icono *detener ejecución*. A medida que el programa se ejecuta, el flujo de control sobre los bloques es resaltado para que el alumno pueda seguirlo.

En la figura 3.13 se expone un ejemplo de programa donde se ejecutan cada uno de los bloques, comenzando por el bloque del programa principal, seguido por el bloque de repetir por siempre, y como siguiente paso, se muestra la ejecución de un bloque de establecimiento de un servomotor a 0 grados. En la figura 3.14 se muestra la continuación del ejemplo anterior, en este paso se muestra la ejecución de un bloque de espera de 1 segundo. Luego en la figura 3.15 se muestra la ejecución de un bloque de establecimiento de un servomotor a 180 grados.

La interfaz gráfica de usuario permite indicar al depurador cuándo pausar un programa, mediante el estableciendo de puntos de interrupción (*breakpoints*). Para establecer un de punto de interrupción, el entorno permite que el usuario haga

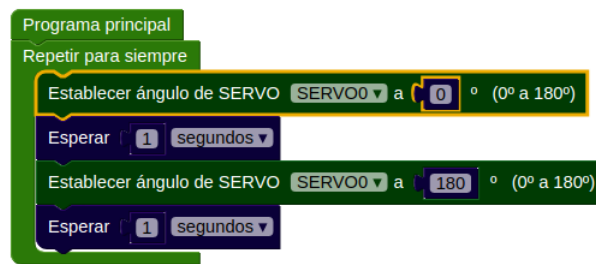


FIGURA 3.13: Resultado del flujo de control (Parte 1).

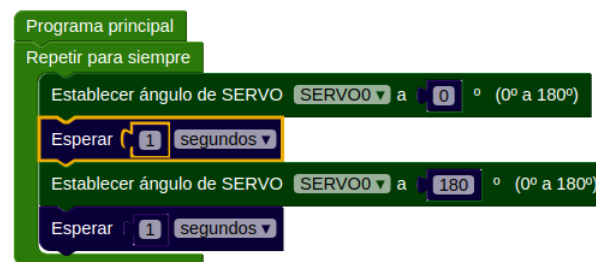


FIGURA 3.14: Resultado del flujo de control (Parte 2).

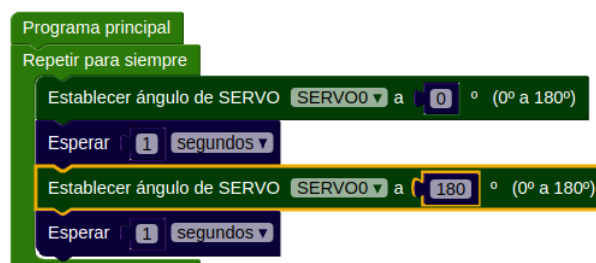


FIGURA 3.15: Resultado del flujo de control (Parte 3).



click en el bloque en donde quiere pausar el programa. Cuando la ejecución del programa llegue a este punto, el programa se detendrá. En la figura 3.16 se muestra el menú contextual del bloque desde donde se puede establecer el punto de interrupción.



FIGURA 3.16: Establecer una bandera de punto de interrupción.

Cuando se hace click en el bloque, debería mostrar un círculo rojo. Esto significa que el punto de interrupción se establece en ese bloque (figura 3.17).

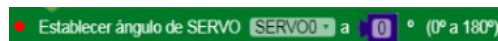


FIGURA 3.17: Punto de interrupción.

Para eliminar el punto de interrupción del bloque, el usuario podrá removerlo desde el menú contextual del mismo bloque. Después de realizar la acción, el círculo rojo del bloque se eliminará. Se ilustra el menú contextual en la figura 3.18.

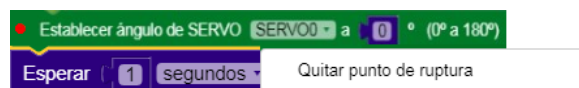


FIGURA 3.18: Quitar Punto de interrupción.

El entorno permite al alumno desactivar todos sus puntos de interrupción a la vez, sin eliminarlos, recordando cuales desactivo, entonces si se hace click nuevamente vuelven al estado original. En la siguiente figura 3.19 se muestra el ícono.



FIGURA 3.19: Desactivar los Puntos de interrupción activos.

### 3.7.3. Menú de visualización

El menú de visualización se divide en dos áreas, una para visualización de variables y otra para puntos de ruptura. La primera permite observar como cambian las variables de programa a medida que se ejecuta mientras está activa la sesión de depuración. En la siguiente figura 3.20 se muestra el panel de variables.

Variables	
Nombre	Valor
i	0
a	0
b	5

FIGURA 3.20: Menú de visualización de variables.

La segunda, permite la visualización de todos los puntos de ruptura (*breakpoints*) insertados en el programa, se permite modificar el estado activo/inactivo de cada punto de ruptura. En la siguiente figura 3.21 se muestra la ventana de *breakpoints*.

Puntos de ruptura (Breakpoints)		
Activos	#	Descripción
<input checked="" type="checkbox"/>	1	Establecer ángulo servo #01
<input checked="" type="checkbox"/>	2	Esperar #01

FIGURA 3.21: Ventana de Puntos de ruptura (*breakpoints*).

El entorno maneja la ejecución en los siguientes estados:

- Ejecución de bloque sin breakpoint.
- Ejecución de bloque con breakpoints activos.
- Ejecución de bloque con breakpoints inactivos.

### 3.8. Edición de programa

El área de edición de programa del entorno del *debug*, contiene la misma estructura de bloques de la caja de herramientas de CIAABOT IDE, las cuales se dividen en las siguientes categorías:

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

El propósito, es permitir la modificación del programa cuando el usuario encuentre errores en la sesión de debug y quiera corregirlo en el entorno del *debug*, de esta manera no tendrá la necesidad de guardar el programa para recién poder editarlo en CIAABOT IDE.

### 3.9. Implementación

CIAABOT *debug* esta desarrollado en el lenguaje Javascript, cuando se importa un proyecto desde CIAABOT IDE con extensión .cbp, el entorno genera internamente código javascript para ejecutar la misma lógica que el programa en bloques. Para realizar esto se utiliza el archivo .cbp tiene la estructura de archivo de texto xml. Se separan los bloques gráficos que acceden a los periféricos de los bloques

gráficos que no acceden a los periféricos, generando de esta manera el código correspondiente que integra todas las partes para que cuando se realice la ejecución del programa, lo haga de forma transparente para el usuario.

Mediante el cliente firmata Johnny-Five se establece la comunicación entre la aplicación y la placa EDUCIAA-NXP, para el manejo de los periféricos.

Johnny-Five es un cliente firmata, basado en el lenguaje JavaScript, y dentro del entorno de *debug*, es el encargado de interactuar con el microcontrolador con firmata4CIAA para la manipulación de los periféricos. En la siguiente figura 3.22 se muestra el diagrama de bloques.

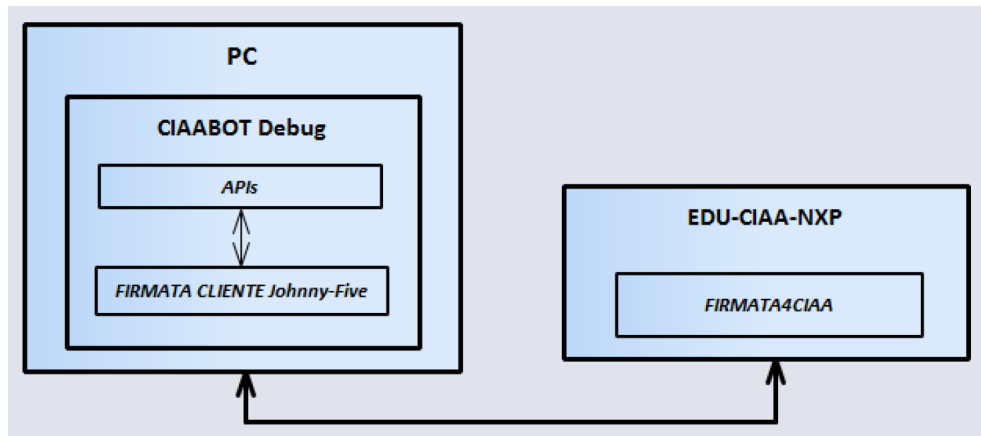


FIGURA 3.22: Diagrama de Bloques para el manejo de los periféricos.

### 3.9.1. Implementación de la GUI

A continuación se explicara las partes de bloques gráficos que ejecutan solo código javascript y las partes que ejecutan código johnny five.

A modo de ejemplo, en la figura 3.23 cómo se armaría en bloques el código para el barrido angular de un servomotor.



FIGURA 3.23: Ejemplo de código en bloques para el barrido de un servo.

El código de salida en lenguaje Javascript de la figura 3.23 se muestra a continuación:

```

1 while(true){
2   ciaa.setServo(0);
3   wait.for(1, sec);
4   ciaa.setServo(180);
5   wait.for(1, sec);
6 };

```

ALGORITMO 3.1: Salida del código en bloques de la figura 3.23.

- Bloques gráficos que acceden a los periféricos: son aquellos que usarán el protocolo firmata para enviar comandos a la placa, y mediante las funciones por UART se van comunicando con el hardware. Por ejemplo en el manejo de sensores, motores, etc. En la figura 3.24 se muestra un bloque de este tipo.



FIGURA 3.24: Barrido de servomotor ejecutado con johnny five

```

1 setServo: function(range) {
2   servo.to(range);
3 }
4

```

ALGORITMO 3.2: Código que ejecuta una de las funciones de johnny five del bloque de la figura 3.24.

- Bloques gráficos que no acceden a los periféricos: son aquellos que se ejecutan en la misma pc, por medio de javascript. Por ejemplo las sentencias loop-for, if-else, etc. En la figura 3.25 se muestra un bloque de este tipo.



FIGURA 3.25: Ejemplo de wait

```

1 for: function(time, unit) {
2   duration = time;
3   waitStart = getTime(unit);
4 }
5

```

ALGORITMO 3.3: Código en javascript del bloque de la figura 3.25.

En la figura 3.24 se expone el bloque gráfico que utiliza los comandos de la API Johnny-five.io. A modo de ejemplo se hizo referencia a un bloque gráfico que usa la siguiente función:

*to(degrees 0-180 [, ms [, rate]])// establece la posición de 0 a 180 grados*

La figura 3.26 muestra el diagrama de secuencia del acceso a un periférico mediante el protocolo firmata.

### 3.9.2. Herramientas utilizadas

Para el desarrollo del entorno de *debugger* se utilizaron los siguientes proyectos de código abierto:

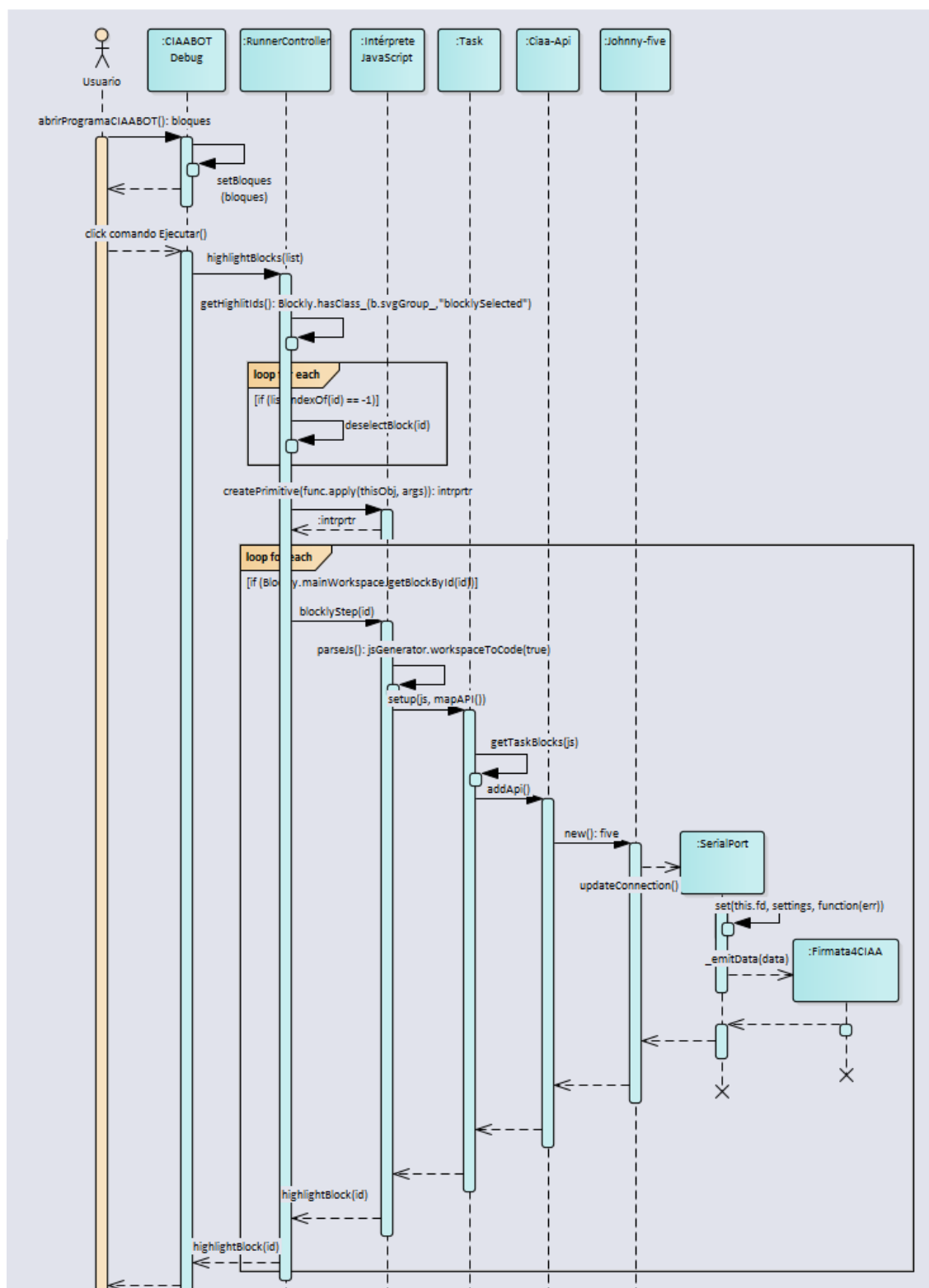


FIGURA 3.26: Diagrama de secuencia.

- Blockly[2]: CIABOOT IDE esta basado en esta biblioteca, por lo cual fue necesario la implementación de ciertas rutinas de código en javascript con la biblioteca blockly para lograr la visualización de los diagramas de bloques creados en CIABOOT IDE, así como también la manipulación de estos bloques gráficos para lograr la interacción con el interprete de Javascript.
- Bootstrap[11]: se uso esta biblioteca dentro de la aplicación para darle comportamiento a los botones de los comandos del entorno del *debug*, ya que provee todas las reglas CSS y HTML5 y se integra muy bien con las bibliotecas de javascript que usa el entorno.
- JS-Interpreter[12]: el proyecto utiliza este intérprete de JavaScript ya que permite la ejecución línea por línea de código escrito en el mismo lenguaje JavaScript, y también para resaltar el código del programa a medida que se vayan ejecutando los bloques. De esta manera los estudiantes podrán realizar un paso a través de su código para ver qué hace qué durante la depuración.
- Acorn[13]: es un rápido analizador tolerante a errores de JavaScript escrito en JavaScript, el analizador dentro del proyecto es usado para darle soporte al análisis/lectura de código del interprete de JavaScript.
- JQuery[14]: la biblioteca multiplataforma de JavaScript permite interactuar con los documentos HTML, manipular el árbol DOM de la página, agregar dinamismo a la aplicación y manejar eventos.
- Signals[15]: biblioteca escrita en JavaScript que implementa el patrón publicación/suscripción para agregar emisores de eventos al código cuando los bloques gráficos que se están ejecutando son resaltados.
- Johnny five[16]: esta basado en Node.js y se integra muy bien con bibliotecas javascript que usamos en el entorno. Lo usaremos para comunicarnos con la EDU-CIAA-NXP y trabajar con los periféricos como actuadores, sensores, etc.
- Express[17]: framework de aplicación web ligero, rápido y muy útil, esta basado en http para Node.js, lo usamos para crear la aplicaciones web que luego será embebida en una aplicación de escritorio. Esta biblioteca proporciona funcionalidades como el enrutamiento, opciones para gestionar sesiones y cookies.
- Serialport[18]: junto a firmata es usada para combinar JavaScript y hardware. Esta biblioteca proporciona listener que es propio del puerto serial.
- Electron[4]: Nos permite armar el desarrollo de la aplicación de escritorio multiplataforma utilizando tecnologías web.

### 3.9.3. Archivo de estado de CIAABOT Debug

Cuando se abre un proyecto (archivo *.cbp*), se crea un archivo *.cbd* donde se guardan los puntos de interrupción que utiliza el usuario durante su sesión de depuración.

El archivo *.cbd* se implementa mediante un archivo de texto en formato JSON[19]. El mismo se guarda al cerrar el proyecto. De esta forma se recuperan los puntos de

interrupción que fueron agregados al programa y si se encuentran activos ó no, de acuerdo a la elección realizada por el usuario en la última sesión de depuración.

Cuando se abre un archivo .cbp creado en CIAABOT IDE y actualizado también en CIAABOT *debug*, y este mismo archivo fue modificado desde la última sesión de depuración en el entorno de *debug* (por ejemplo si se elimina un bloque gráfico que tenía un punto de interrupción), el entorno detectará estas diferencias, actualizará el archivo .cbd con estos cambios y mostrará al usuario en los bloques coincidentes con las opciones de depuración de la última sesión.

Esta información es útil cuando el usuario cerro por algún motivo el entorno de CIAABOT *debug* y cuando lo vuelve a abrir no pierde el estado de los puntos de interrupciones utilizados en su última sesión de depuración.

A continuación se observa un ejemplo de la estructura del archivo nombrado *debug.cbd*.

```
{
  "type": "ciaabotDebugState",
  "breakpoints": [
    {
      "type": "asociation",
      "key": {
        "type": "logic_operation\\",
        "id": "Ju6).aSXM}[dFL_Xw/p,\\\"
      },
      "value": {
        "type": "breakpoint",
        "id": 1,
        "active" : true
      }
    },
    {
      "type": "asociation",
      "key": {
        "type": "ciaa_sapi_gpio_digital_read",
        "id": "Oj7~RU|??CeufGPNV$|W\\\"
      },
      "value": {
        "type": "breakpoint",
        "id": 2,
        "active" : false
      }
    }
  ]
}
```

### 3.9.4. GitLab

Como estrategia de mitigación del riesgo de pérdida de código, se propuso el versionado de código, para realizarlo se eligió la herramienta Gitlab, que es un

servicio web de control de versiones y desarrollo de software colaborativo. Esta basado en Git y permite repositorios de código públicos y privados.



## Capítulo 4

# Ensayos y Resultados

### 4.1. Pruebas funcionales del hardware

La idea de esta sección es explicar cómo se hicieron los ensayos, qué resultados se obtuvieron y analizarlos.



## Capítulo 5

# Conclusiones

### 5.1. Conclusiones generales

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. Próximos pasos

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



# Bibliografía

- [1] Proyecto CIAA. *Computadora Industrial Abierta Argentina*. Disponible: 2016-06-25. 2014. URL: <http://proyecto-ciaa.com.ar/devwiki/doku.php?id=start>.
- [2] Neil Fraser. *Documentación de Blockly*. URL: <https://developers.google.com/blockly/>.
- [3] *Documentación de Angular*. URL: <https://angular.io/docs>.
- [4] *Sobre Electron*. URL: <https://electron.atom.io/docs/tutorial/about/>.
- [5] *Sobre NodeJS*. URL: <https://nodejs.org/en/about/>.
- [6] *Documentación de TypeScript*. URL: <https://www.typescriptlang.org>.
- [7] *CIAA Firmware v2*. 2016. URL: [https://github.com/ciaa/firmware\\_v2](https://github.com/ciaa/firmware_v2).
- [8] *Sitio web de referencia*. URL: <https://cmake.org/>.
- [9] *Sitio web de OpenOCD*. URL: <http://openocd.org/>.
- [10] *sAPI*. 2017. URL: <https://github.com/epernia/sAPI>.
- [11] Mark Otto y Jacob Thornton de Twitter. *Repositorio de Bootstrap*. URL: <https://github.com/twbs/bootstrap>.
- [12] Neil Fraser. *Repositorio de JS-Interpreter*. URL: <https://github.com/NeilFraser/JS-Interpreter>.
- [13] Marijn Haverbeke. *Repositorio de Acorn*. URL: <https://github.com/acornjs/acorn>.
- [14] John Resig. *Repositorio de JQuery*. URL: <https://github.com/jquery/jquery>.
- [15] Robert Penner's. *Repositorio de Signals*. URL: <https://github.com/millermedeiros/js-signals>.
- [16] *Repositorio de Johnny five*. URL: <https://github.com/rwaldron/johnny-five>.
- [17] *Repositorio de Express*. URL: <https://github.com/expressjs/express>.
- [18] *Repositorio de Serialport*. URL: <https://github.com/node-serialport/node-serialport>.
- [19] *Sitio web oficial de Json*. URL: <https://www.json.org>.