

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 Pernia (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 y 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 unitarios 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 Pernia, por su paciencia, ayuda y entusiasmo.

Índice general

Resumen	III
1. Introducción General	1
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
2. Introducción Específica	5
2.1. Componentes CIAABOT	5
2.1.1. CIAABOT IDE	5
2.1.2. CIAABOTS	7
2.1.3. Firmware	7
2.2. Requerimientos	7
2.2.1. Requerimientos del entorno de depuración	8
2.2.2. Hardware soportado por el entorno a desarrollar	8
2.2.3. Firmware	8
2.2.4. Procesos Finales	9
2.3. Planificación	9
2.3.1. Desglose en tareas	9
2.3.2. Activity On-node	11
2.3.3. Diagrama de Gantt	11
3. Diseño e Implementación	15
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 <i>CIAABOT Debug</i>	18
3.5. Interfaz gráfica de <i>CIAABOT Debug</i>	18
3.6. Máquina de estados de <i>CIAABOT Debug</i>	19
3.6.1. Desconectado	19
3.6.2. Conectando	19
3.6.3. Conectado Sin Firmata4CIAA	21
3.6.4. Descargando Firmata4CIAA	21
3.6.5. Conectado Con Firmata4CIAA	23
3.6.6. Conectado En Sesión De Depuración	23
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	29
3.9.1. Implentación de la GUI	29
3.9.2. Herramientas utilizadas	32
3.9.3. Archivo de estado de <i>CIAABOT Debug</i>	33
3.9.4. GitLab	33
4. Ensayos y Resultados	35
4.1. Ensayos preliminares sobre acerca del control de la plataforma mediante protocolo Firmata.	35
4.2. Ensayo de funcionamiento en los sistemas operativos Windows y Linux.	35
4.2.1. Instalación en cada sistema.	37
<i>Test</i> funcional posterior a la instalación en cada sistema . . .	37
4.2.2. Chequeo de conexión y descarga de Firmata4CIAA.	38
<i>Test</i> funcional descargar Firmata4CIAA	38
4.2.3. Chequeo de conexión con la plataforma EDU-CIAA-NXP. . .	38
<i>Test</i> funcional de conexión con la plataforma EDU-CIAA-NXP	39
4.3. Comprobación de funcionamiento de una sesión de depuración. . .	39
4.3.1. Sesión de depuración uso del comando <i>Ejecutar</i>	39
4.3.2. Sesión de depuración uso del comando <i>Suspender</i>	40
4.3.3. Sesión de depuración uso del comando <i>Pasar por encima (step over)</i>	40
4.3.4. Sesión de depuración uso del comando <i>Pasar adentro (step into)</i>	41
4.3.5. Sesión de depuración uso del comando <i>Paso de regreso (step return)</i>	41
4.3.6. Sesión de depuración con puntos de ruptura (<i>breakpoints</i>) . .	42
4.3.7. Sesión de depuración uso del comando desactivar puntos de interrupción	42
4.3.8. Sesión de depuración desactivar puntos de interrupción desde la ventana de visualización de <i>breakpoints</i>	43
4.3.9. Sesión de depuración uso del comando <i>Detener depuración</i> .	43
4.3.10. Guardado y restauración de los puntos de ruptura	44
4.4. Ensayos de edición de programas en lenguaje CIAABOT.	45
4.4.1. Edición de programa	45
4.5. Ensayos de interfaz de usuario.	45
4.6. Validación con usuarios.	46
5. Conclusiones	47
5.1. Conclusiones generales	47
5.2. Próximos pasos	48
Bibliografía	49

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	8
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.	13
3.1. Modelo conceptual.	15
3.2. Uso de firmata con la Educiaa.	17
3.3. Diseño de la interfaz gráfica de <i>CIAABOT Debug</i>	18
3.4. Composición de la interfaz gráfica de <i>CIAABOT Debug</i>	19
3.5. Diagrama de máquina de estados de los modos de funcionamiento.	20
3.6. Estado Desconectado.	21
3.7. Descargar Firmata4CIAA.	22
3.8. Configuración de paths.	22
3.9. Estado Conectado con Firmata4CIAA.	23
3.10. Estado Depurando.	24
3.11. Estados del botón de depuración.	24
3.12. Herramientas de control de ejecución habilitada.	25
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.	28
3.21. Ventana de Puntos de ruptura (<i>breakpoints</i>).	28
3.22. Diagrama de Bloques para el manejo de los periféricos.	29
3.23. Ejemplo de código en bloques para el barrido de un servo.	30
3.24. Barrido de servomotor ejecutado con johnny five	30
3.25. Ejemplo de wait	30
3.26. Diagrama de secuencia.	31
4.1. Programa Firmata test	36
4.2. Firmata4CIAA	37

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 explica 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 (IDE son las siglas en inglés de entorno de desarrollo integrado). 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, mediante el agregado de capacidad de depuración a los programas desarrollados con CIAABOT IDE.

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.
- CIAABOTS: son las plataformas de hardware que se programan desde CIAABOT 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: encastrando bloques predeterminados 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.

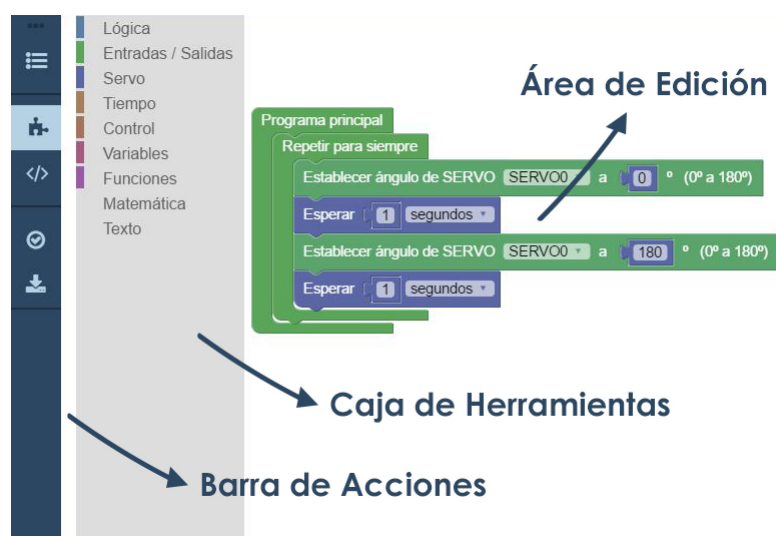


FIGURA 1.1: Área de flujo de trabajo de CIAABOT-IDE.

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. 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.

1.2. Motivación

La plataforma CIAABOT no posee la capacidad de *debuggear* un programa. Para realizar los ensayos se tiene que compilar el programa, descargalo 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 bloque por bloque.
- 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.
- Importar el programa de bloques creado en el IDE de desarrollo de CIAABOT.

De esta manera el aprendizaje del usuario programador primerizo será más enriquecedor, 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. Las cuales se detallan en las siguientes secciones.

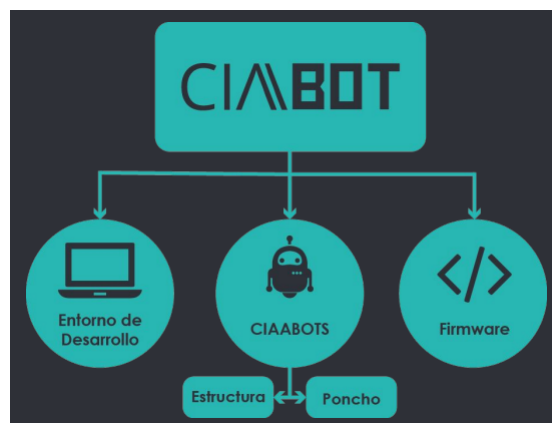


FIGURA 2.1: Componentes CIAABOT.

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, 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 generado a 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.

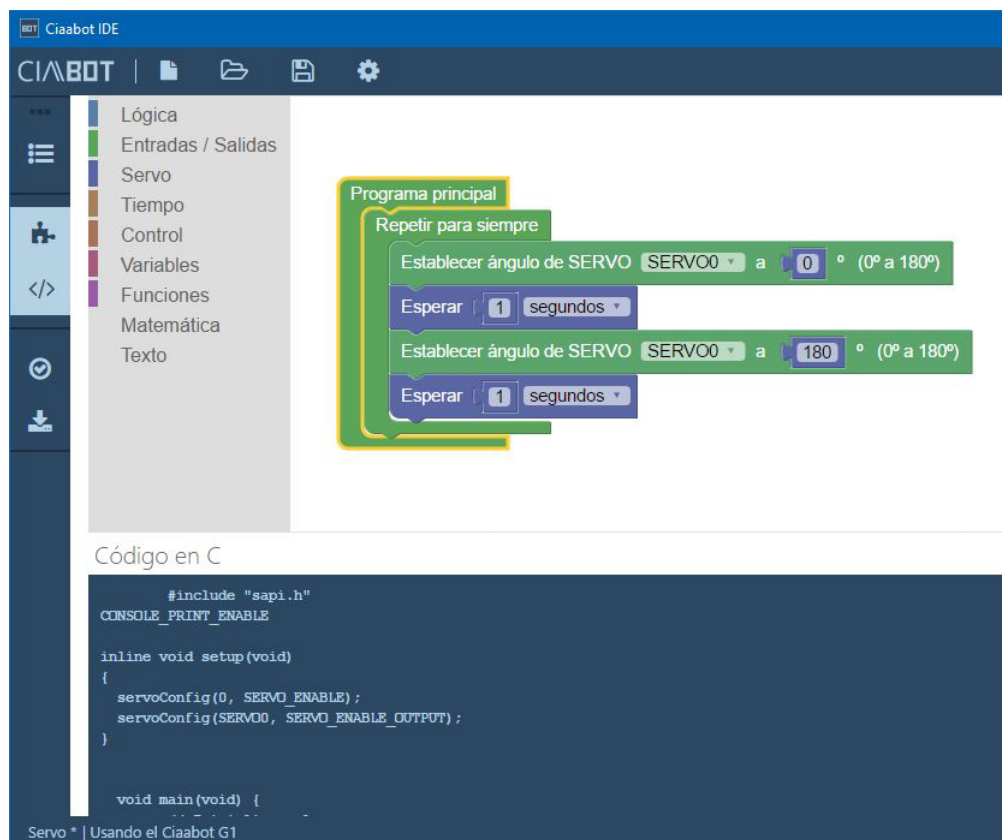


FIGURA 2.2: Editor gráfico de CIAABOT-IDE.

La plataforma CIAABOT esta programada 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 programas.
- Electrón [4]: es una biblioteca para armar aplicaciones de escritorio multi-plataforma 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:

- Diseñar un programa con bloques encastrables. A medida que esto se realiza va generando internamente código C para implementar la misma lógica que describen los bloques.

- Compilar el código C generado a partir de los bloques, utilizando el *tool-chain* GNU ARM Embedded [7]. Este proceso genera un archivo binario.
- Descargar el archivo binario conectándose a la plataforma mediante OpenOCD [8] y resetearla para que comience a funcionar.

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

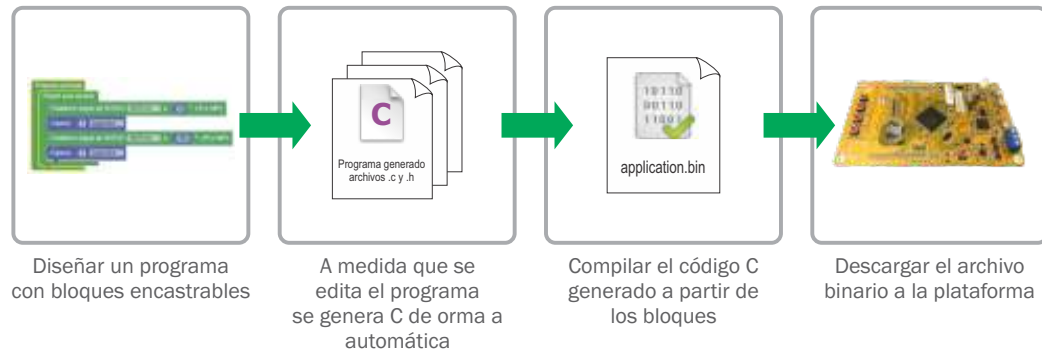


FIGURA 2.3: Diagrama del funcionamiento principal de la plataforma CIAABOT.

2.1.2. CIAABOTS

Se denominan 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 [9] del Proyecto CIAA, puede crear desde funciones simples a más complejas, para realizarlo utiliza las siguientes herramientas:

- *Makefile* [10]: para la gestión de dependencias, de esta manera puede construir el software desde sus archivos fuente.
- *OpenOCD* [8]: una herramienta OpenSource, usado para el grabado del firmware en las placas.
- *sAPI* [11]: 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 del entorno de depuración

- REQ1: El entorno de depuración deberá poder utilizarse dentro de los sistemas operativos Windows y Linux.
- REQ2: Debe controlar de la ejecución de un programa corriendo en la plataforma de hardware.
- REQ3: Exponer el estado de las variables del programa en tiempo de ejecución.
- REQ4: Permitir la edición de programas para corregir errores de lógica encontrados en tiempo de ejecución.
- REQ5: La interfaz gráfica debe seguir los lineamientos de estilo establecidos en el Proyecto CIAABOT.
- REQ6: Debe presentar un diseño de interfaz gráfica intuitiva, con elementos similares a los hallados en otras herramientas de depuración.

2.2.2. Hardware soportado por el entorno a desarrollar

- Se usará la placa EDU-CIAA-NXP (figura 2.4) para el control de los robots.

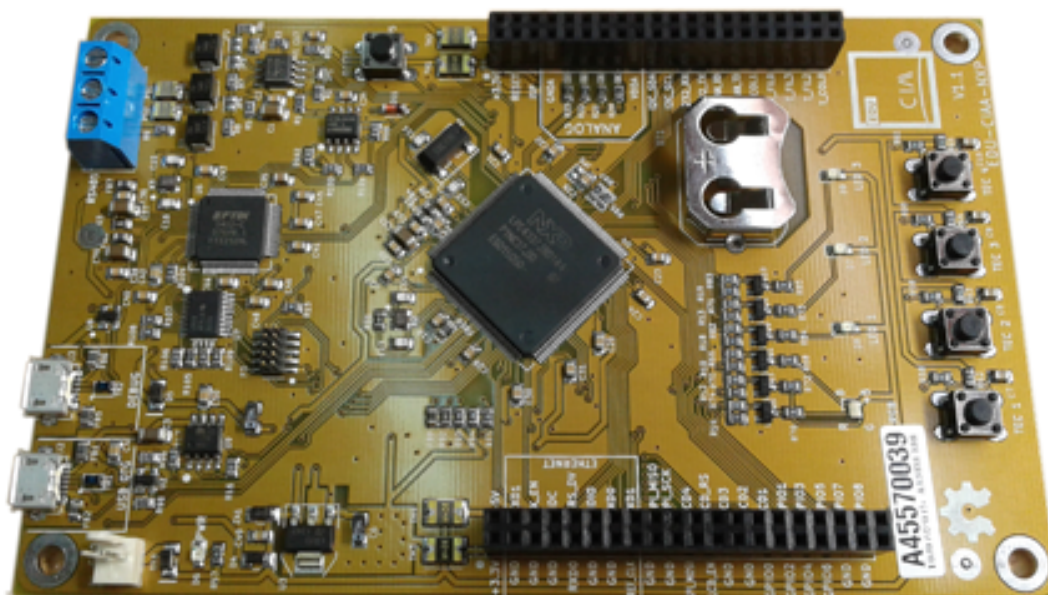


FIGURA 2.4: Placa EDU-CIAA-NXP

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.
- 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 un 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 sesiones de depuración multiplataforma.
 - 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* multiplataforma.
 - 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.

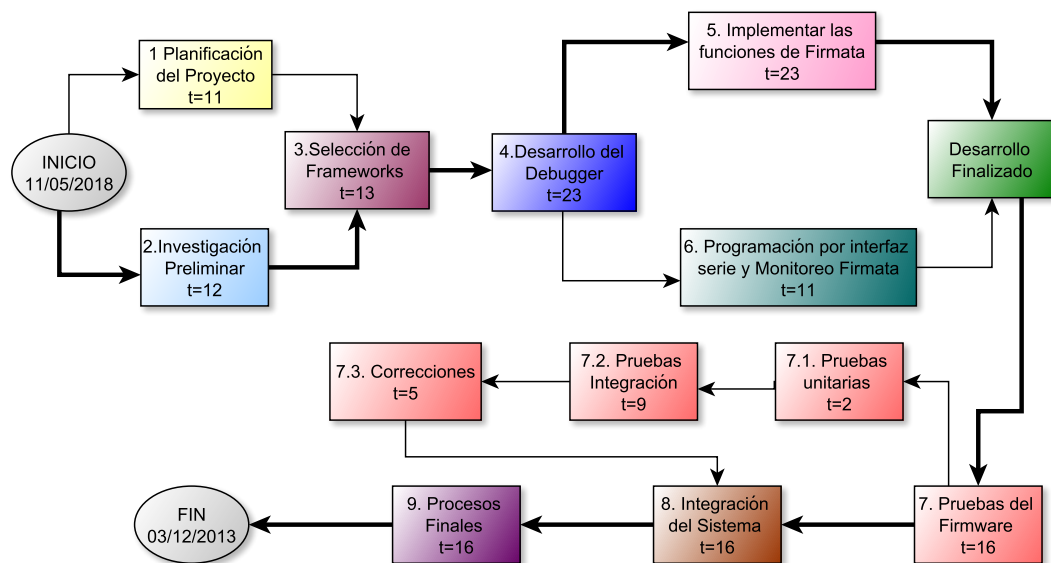


FIGURA 2.5: Diagrama de Node.

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.

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,

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

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 laborables y no laborables.

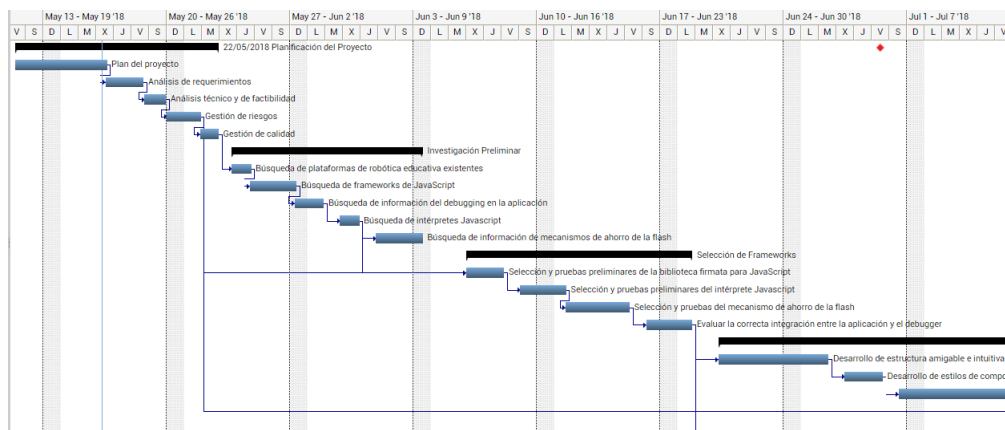


FIGURA 2.7: Diagrama de Gantt - Parte 1.

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

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

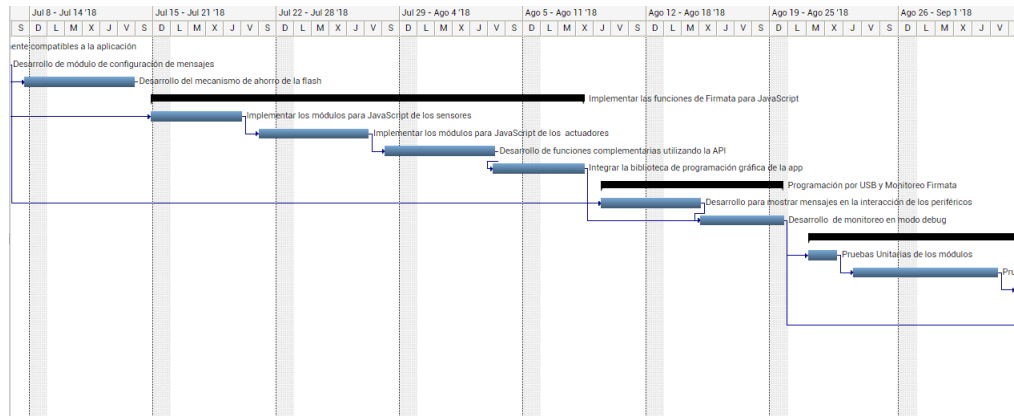


FIGURA 2.8: Diagrama de Gantt - Parte 2.

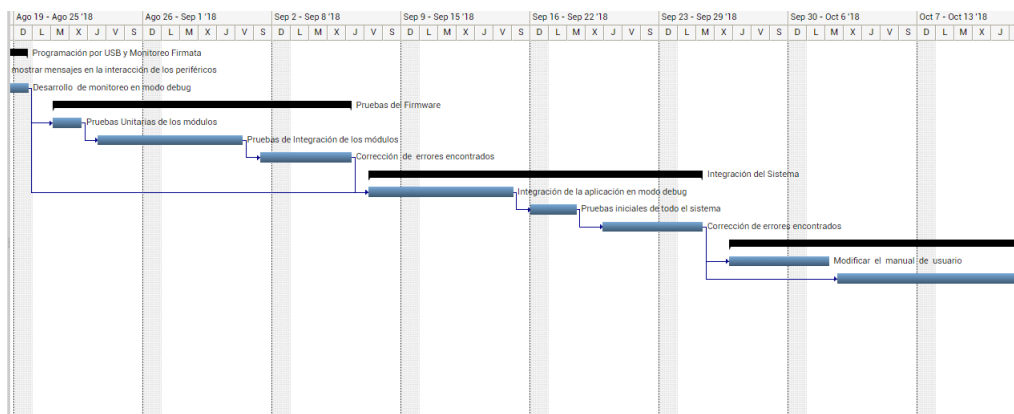


FIGURA 2.9: Diagrama de Gantt - Parte 3.

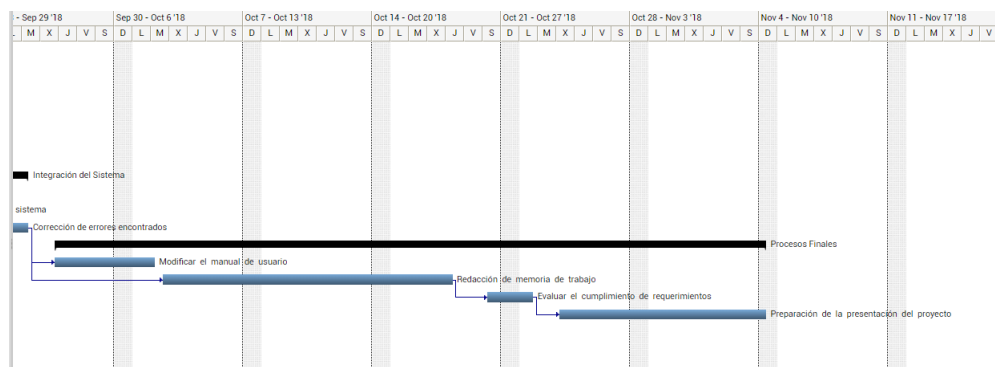


FIGURA 2.10: Diagrama de Gantt - Parte 4.

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 IDE 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 EDU-CIAA-NXP 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*¹: 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.

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

- *arm-none-eabi-gdb*: software de debug originario de Linux que 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 permitir ejecutar los comandos de parar, continuar o de uso de breakpoints.
- *OpenOCD (Open On-Chip Debugger)*: software de código abierto que interactúa con el puerto JTAG de un depurador de hardware, 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 se encuentra el microcontrolador a depurar.
- *Microcontrolador a depurar*: El microcontrolador posee un periférico específico para *debug* con interfaz JTAG (Acrónimo de Joint Test Action Group, es utilizado como mecanismo para depuración de sistemas embebidos, proveendo una puerta trasera para acceder al sistema) 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, para 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 elige utilizar el protocolo firmata, debido a que existe una implementación del mismo para la EDU-CIAA-NXP [9] 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.

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 EDU-CIAA-NXP.

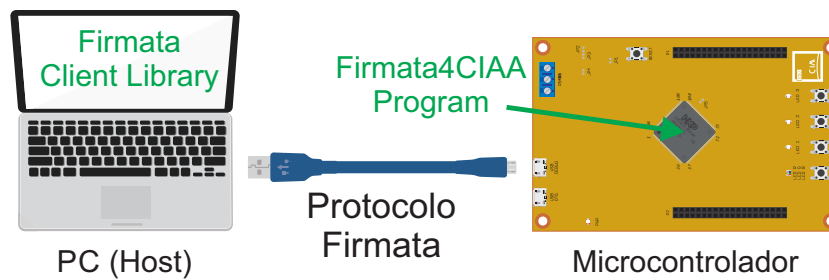


FIGURA 3.2: Uso de firmata con la Educiaa.

3.3. Sistema para depuración propuesto

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

- Ejecutar un programa bloque a bloque en la PC, 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 EDUCIAA-NXP, 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 de la máquina virtual 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 el tiempo de compilación del código C generado a partir del programa en bloques (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.
- Descarga 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 cargada y el diagrama en bloques es mostrado.

3.5. Interfaz gráfica de *CIAABOT Debug*

La figura 3.4 expone el diseño realizado para la interfaz gráfica de *CIAABOT Debug*.

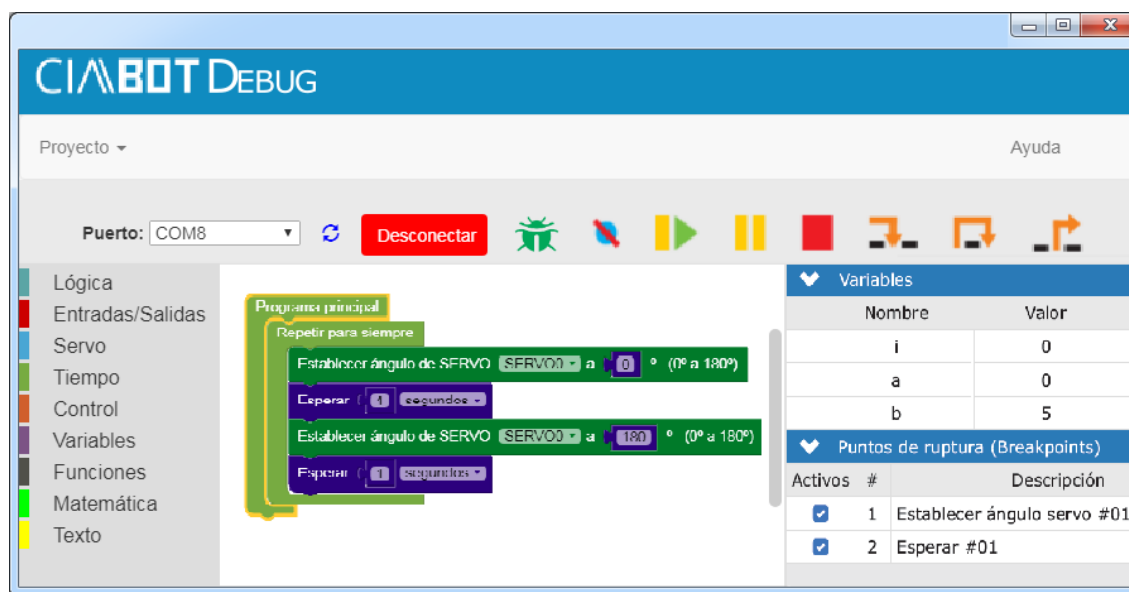


FIGURA 3.3: Diseño de la interfaz gráfica de *CIAABOT Debug*.

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 del 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, así 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 se presiona el botón "Conectar" pasa al estado *Conectando*. En este estado se 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*.

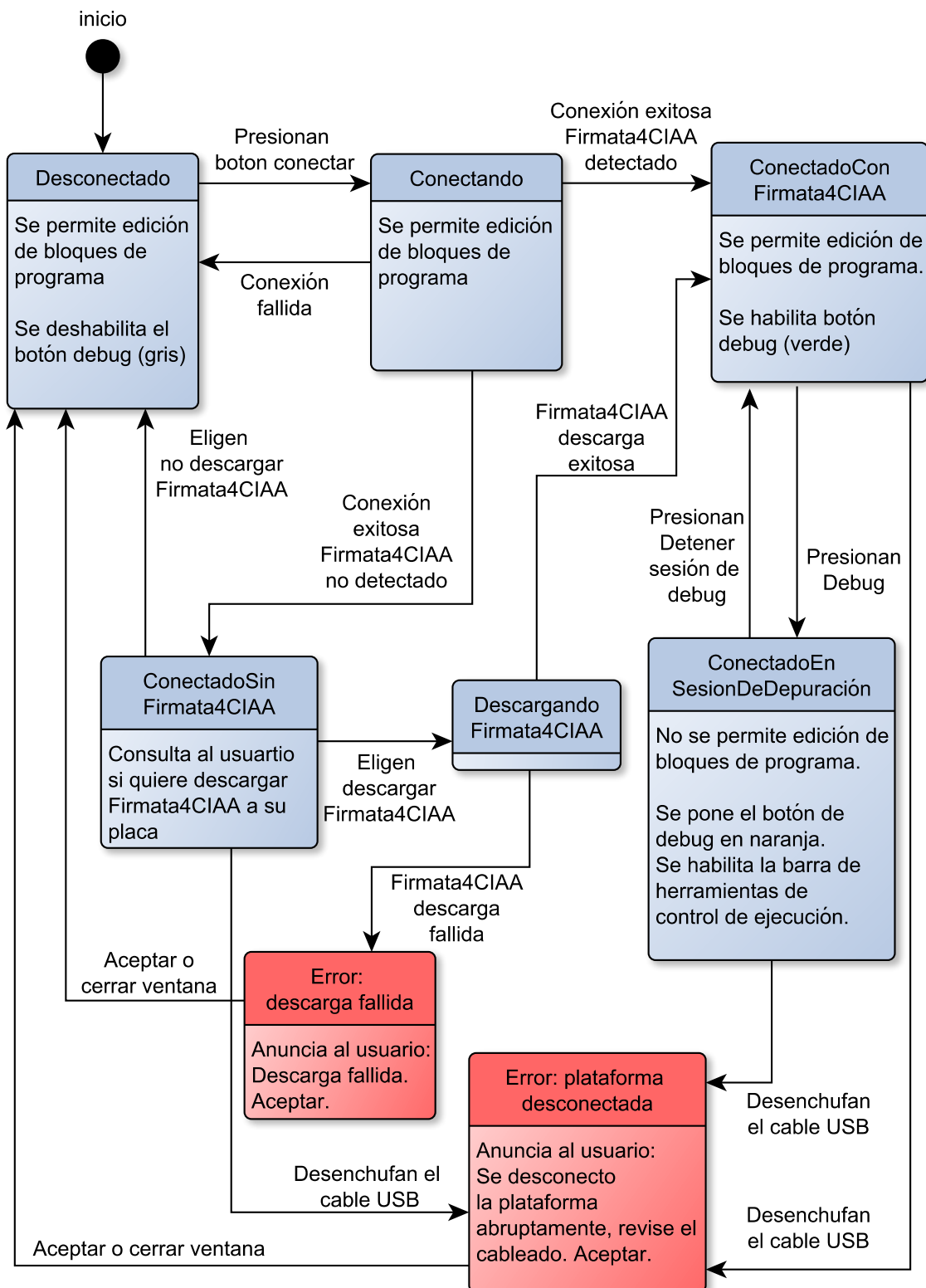


FIGURA 3.5: Diagrama de máquina de estados de los modos de funcionamiento.

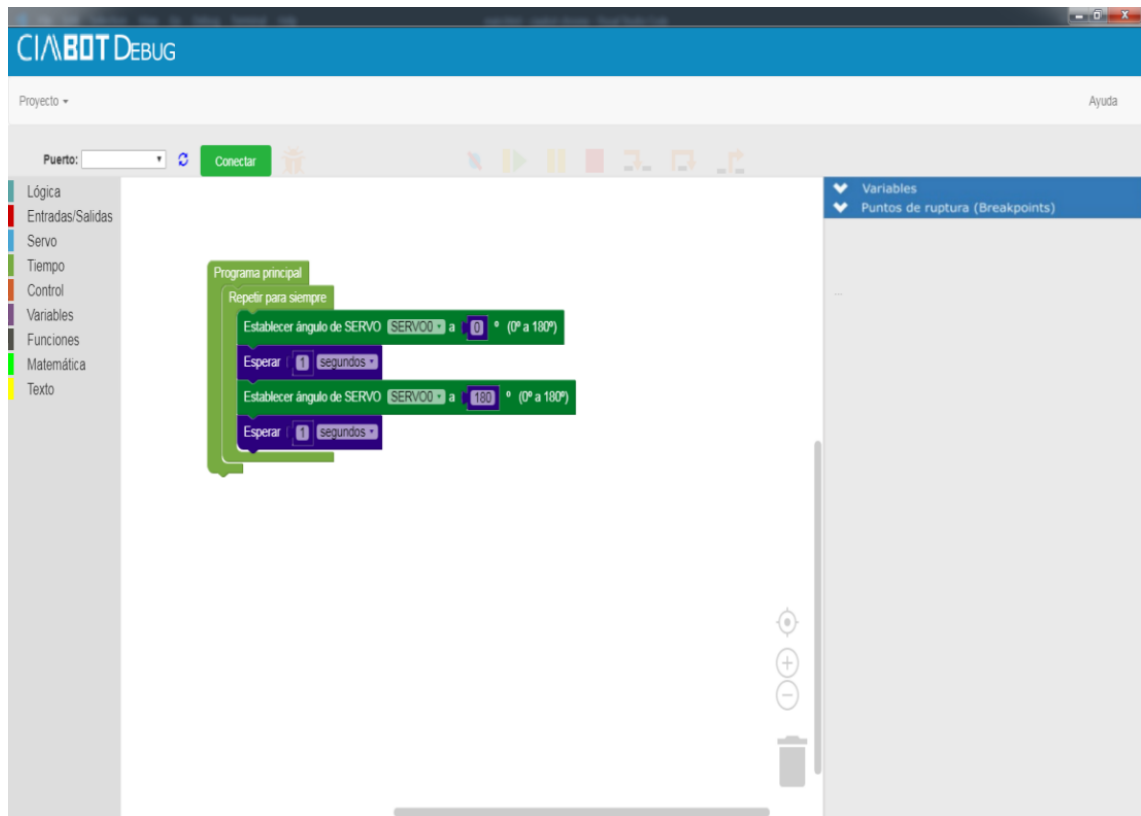


FIGURA 3.6: Estado Desconectado.

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 estado *Desconectado*.

En la figura 3.7 muestra la consulta enviada al usuario.

3.6.4. Descargando Firmata4CIAA

Firmata4CIAA, la aplicación pasa al estado *Descargando Firmata4CIAA*. Si en medio del proceso existiese algún problema de descarga, se informará al usuario y volverá a estado *Desconectado*.

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 comando *make download* en la ruta de dicho proyecto.

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

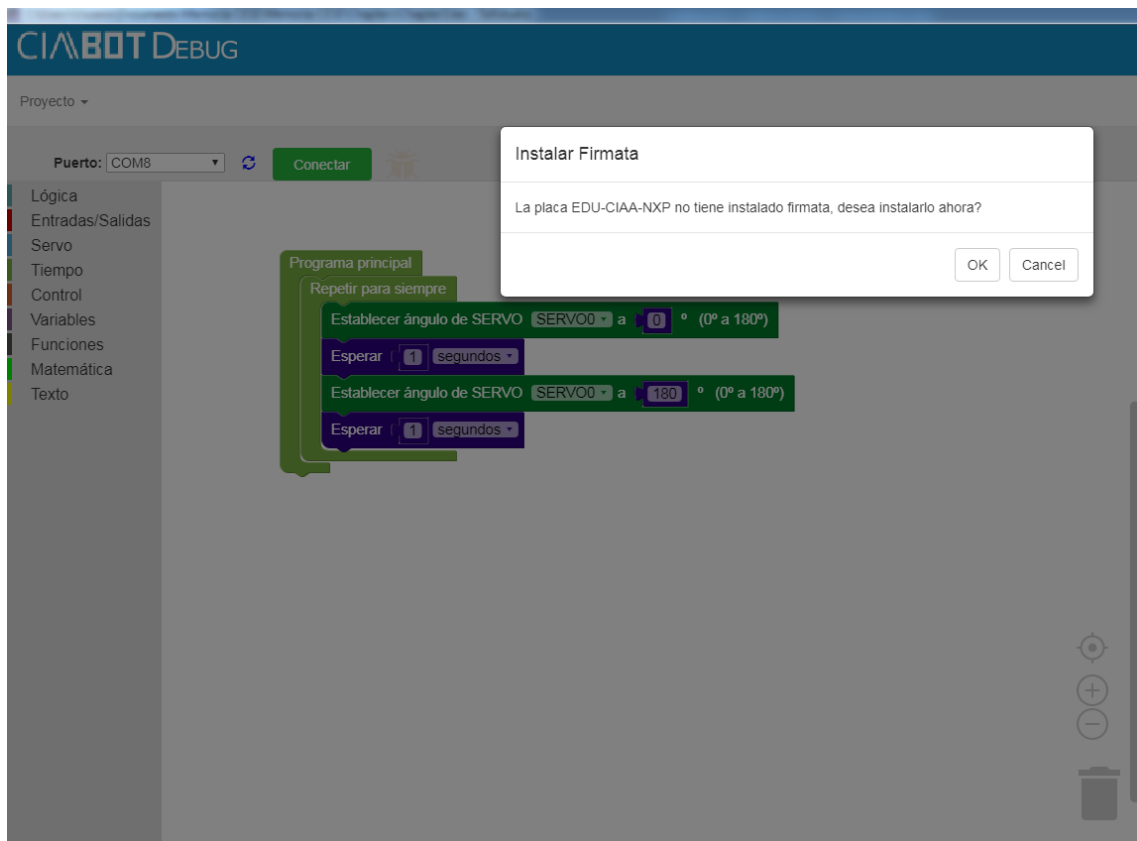


FIGURA 3.7: Descargar Firmata4CIAA.

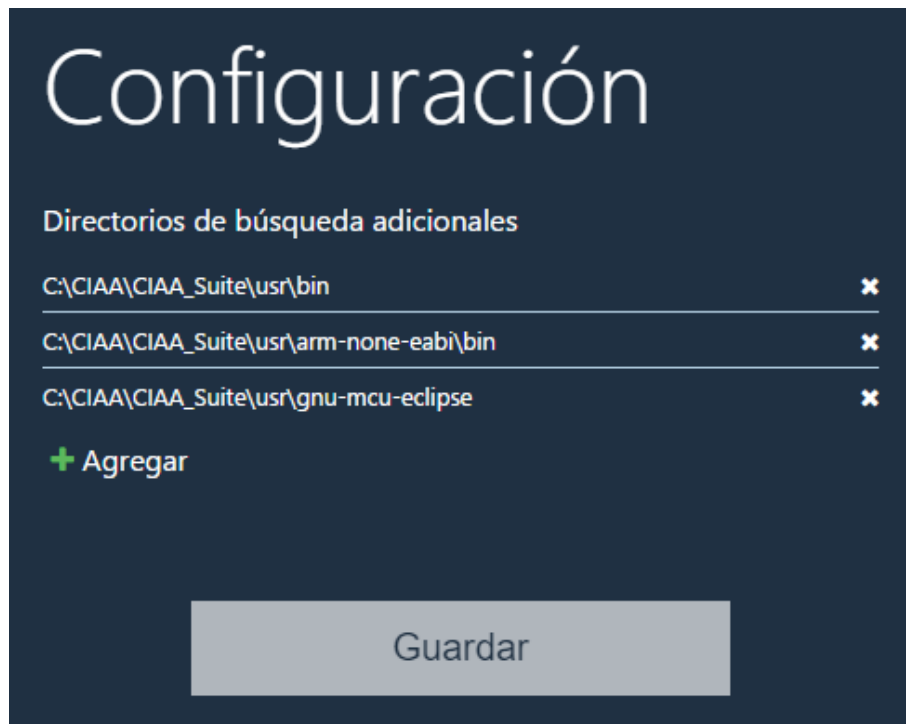


FIGURA 3.8: Configuración de paths.

3.6.5. Conectado Con Firmata4CIAA

En caso de que la conexión resulte exitosa y se compruebe la disponibilidad de Firmata en la plataforma, la aplicación pasa al estado *Conectado Con Firmata4CIAA*, se habilita el inicio de sesión de depuración. En la figura 3.9 se muestra este modo de funcionamiento.

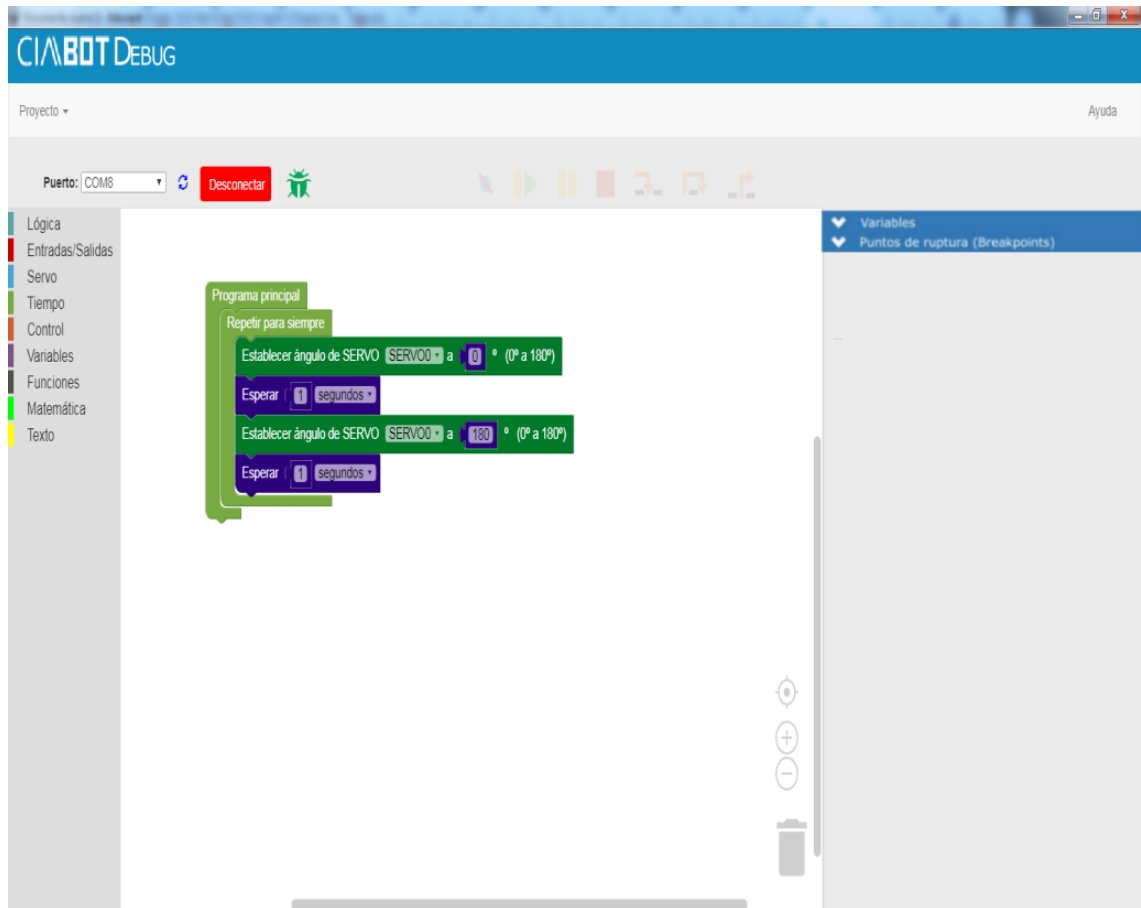


FIGURA 3.9: Estado Conectado con Firmata4CIAA.

3.6.6. Conectado En Sesión De Depuración

Si en el estado Conectado el usuario inicia la sesión de depuración, el entorno pasa a estado *Conectado En Sesión de Depuración*. En este estado habilita la barra de herramientas de control de ejecución, permitiéndole realizar el ensayo de su programa y deshabilita el menú de edición. La figura 3.10 muestra la aplicación con este estado.

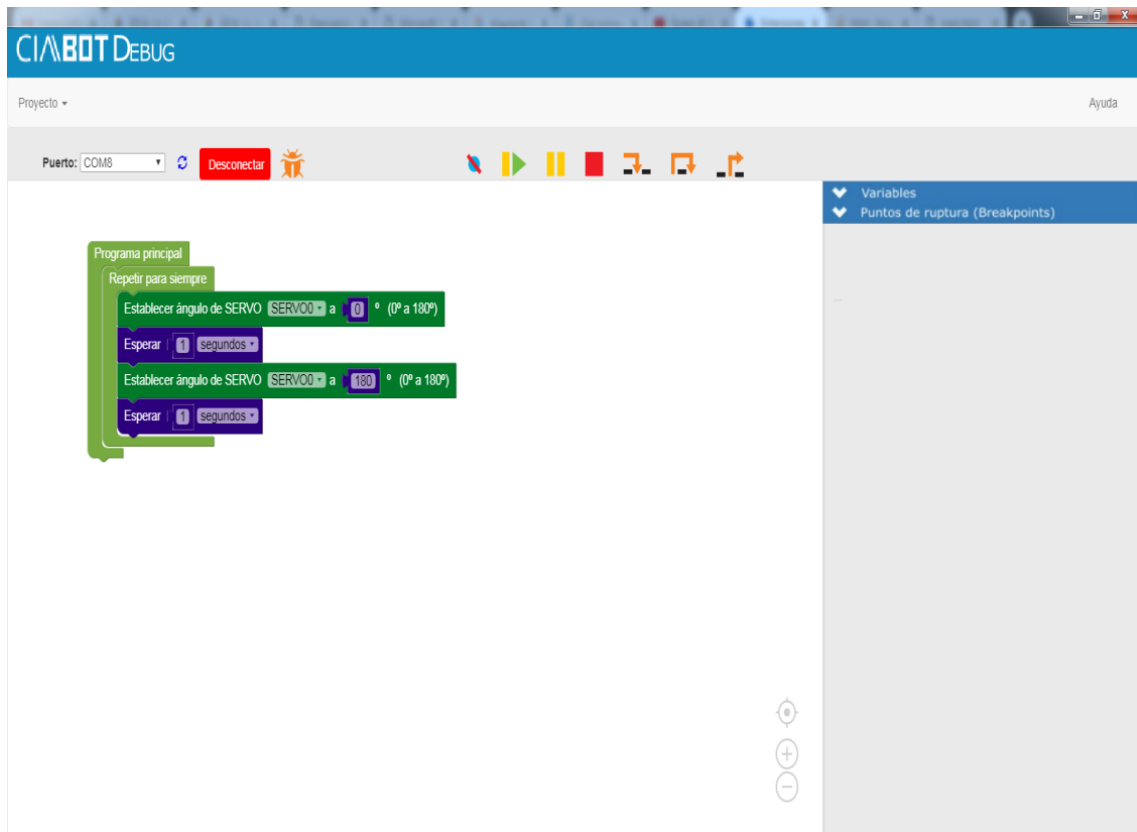


FIGURA 3.10: Estado Depurando.

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 (deshabilitado) a color verde (habilitado), de esta manera se permite al usuario mediante un click en el mismo, comenzar con la sesión de depuración. Al presionarlo, el botón se pone en color naranja, indicando que se ha iniciado una sesión de depuración y se deshabilita la acción de click sobre el mismo. Con la sesión de depuración iniciada el entorno habilita las herramientas de control de ejecución como se muestra en la figura 3.12.



FIGURA 3.11: Estados del botón de depuración.

La sesión de depuración finaliza cuando el usuario presiona el botón *Detener depuración*, o si se cierra el programa en la aplicación.



FIGURA 3.12: Herramientas de control de ejecución habilitada.

3.7.2. Herramientas de control de ejecución

En la tabla 3.1 se listan los botones 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 (<i>step into</i>)	Ingresa a ejecutar el código en caso de ejecutar un llamado a función.
	Pasar por encima (<i>step over</i>)	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 (<i>step return</i>)	Ejecutar hasta encontrar un retorno de función.

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*.

En la figura 3.13 se expone un ejemplo de programa donde se ejecutan cada uno de los bloques con el comando *Pasar por encima (step over)*, 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



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

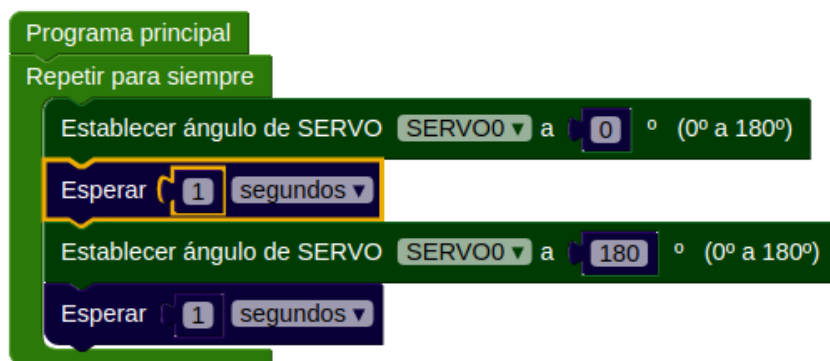


FIGURA 3.14: Resultado del flujo de control (Parte 2).

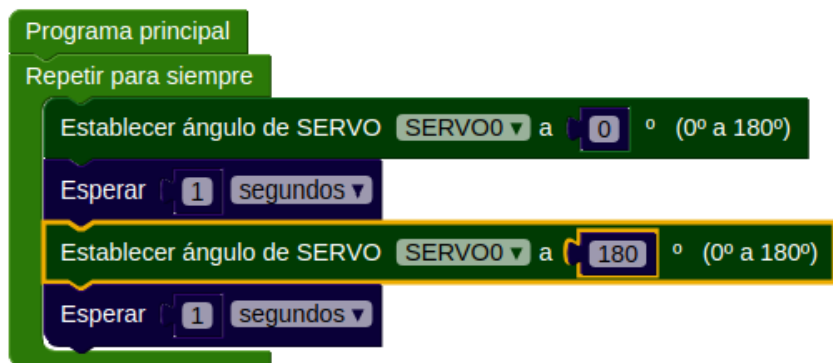


FIGURA 3.15: Resultado del flujo de control (Parte 3).

programa, mediante el estableciendo de puntos de interrupción (*breakpoints*). Para establecer un de punto de interrupción, el entorno permite que el usuario haga 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).

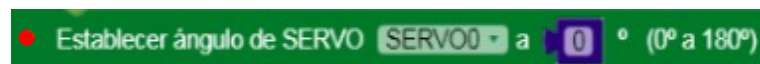


FIGURA 3.17: Punto de interrupción.

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.

Utilizando el botón *Desactivar puntos 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 de este botón.



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.

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*.

El entorno maneja la ejecución en los siguientes estados:

Variables	
Nombre	Valor
i	0
a	0
b	5

FIGURA 3.20: Menú de visualización de variables.

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*).

- 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 está desarrollado en el lenguaje Javascript. Permite importar un proyecto desde CIAABOT IDE con extensión .cbp y 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 EDU-CIAA-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.

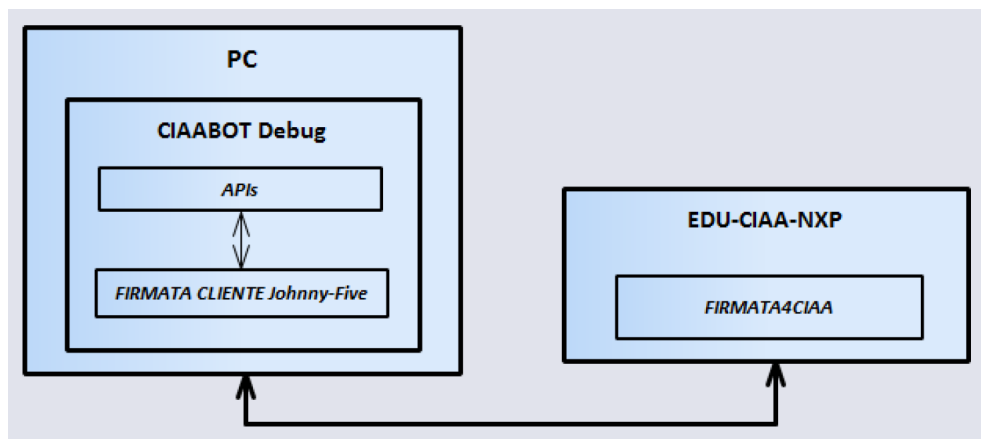


FIGURA 3.22: Diagrama de Bloques para el manejo de los periféricos.

3.9.1. Implentación de la GUI

A continuación se explicará las partes de bloques gráficos que ejecutan solo código javascript y las partes que ejecutan comandos de 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.

El código de salida en lenguaje Javascript de la figura 3.23 se muestra en el Algoritmo 3.1:

```

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.



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

- 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.

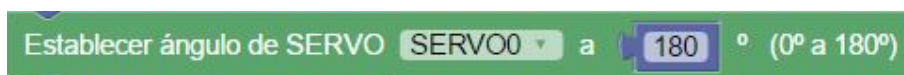


FIGURA 3.24: Barrido de servomotor ejecutado con johnny five

```

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.

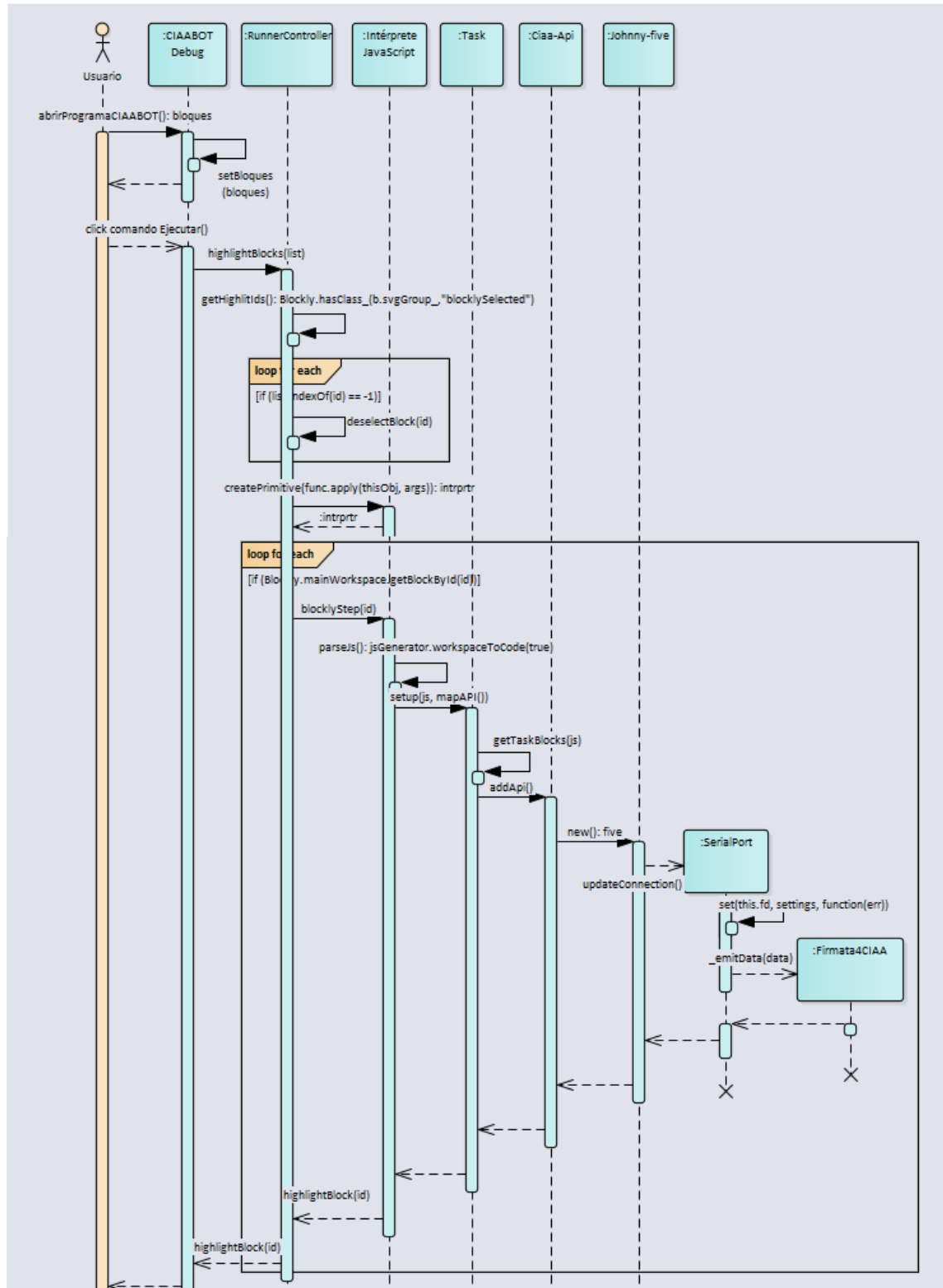


FIGURA 3.26: Diagrama de secuencia.

3.9.2. Herramientas utilizadas

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

- Blockly [2]: el lenguaje gráfico de CIAABOT 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 [12]: se uso esta biblioteca dentro de la aplicación para darle comportamiento a los botones 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 [13]: el proyecto utiliza este intérprete de JavaScript ya que permite la ejecución línea por línea de código JavaScript generado para ejecutar el programas de usuario en bloques, y también para resaltar los bloques del programa a medida que se vayan ejecutando internamente. De esta manera los estudiantes podrán realizar un paso a través de su código para ver qué hace qué durante la depuración.
- Acorn [14]: 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 [15]: 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 [16]: 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 [17]: esta basado en Node.js y se integra muy bien con bibliotecas javascript que usamos en el entorno. Se utiliza para la comunicación con la EDU-CIAA-NXP y trabajar con sus periféricos.
- Express [18]: *framework* de aplicación web ligero, rápido y muy útil, esta basado en http para Node.js, se usa para crear la aplicaciones web que luego son embebidas en una aplicación de escritorio. Esta biblioteca proporciona funcionalidades como el enrutamiento, opciones para gestionar sesiones y cookies.
- Serialport [19]: junto a firmata es usada para combinar JavaScript y hardware. Esta biblioteca permite el acceso a puertos series en la computadora.
- Electron [4]: 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 [20]. 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 o no, de acuerdo a la elección realizada por el usuario en la última sesión de depuración.

Si un proyecto luego es modificado por CIAABOT IDE, por ejemplo eliminando algún bloque que contenía un *breakpoint*, al abrirlo nuevamente desde CIAABOT Debug se actualiza la información del archivo *.cbd* asociada al proyecto *.cbp* para mantener la coherencia entre los mismos.

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.

En el Algoritmo 3.4 se expone un ejemplo de la estructura del archivo nombrado *debug.cbd*.

```

1 {
2   "type": "ciaabotDebugState",
3   "breakpoints": [
4     {
5       "type": "asociation",
6       "key": {
7         "type": "logic_operation\\",
8         "id": "Ju6).aSXM}[dFL_Xw/p,\\",
9       },
10      "value": {
11        "type": "breakpoint",
12        "id": 1,
13        "active": true
14      }
15    },
16    {
17      "type": "asociation",
18      "key": {
19        "type": "ciaa_sapi_gpio_digital_read",
20        "id": "Oj7~RU|??CeufGPNV$IW\\",
21      },
22      "value": {
23        "type": "breakpoint",
24        "id": 2,
25        "active": false
26      }
27    }
28  ]
29 }
```

ALGORITMO 3.4: Estructura del archivo *debug.cbd*

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

Se realizaron pruebas funcionales sobre los requerimientos del proyecto.

4.1. Ensayos preliminares sobre acerca del control de la plataforma mediante protocolo Firmata.

Para comprobar el funcionamiento del entorno de depuración primero se comprobó mediante otro software el correcto acceso a la plataforma de hardware. Para esto se utilizó:

- El programa *Firmata test* [21], permite ensayar desde la PC el funcionamiento de una plataforma que soporte protocolo Firmata. La figura 4.1 expone la interfaz gráfica del software libre *Firmata test*.
- La placa EDU-CIAA-NXP corriendo el programa *Firmata4CIAA*, el cual fue compilado y descargado de forma manual.

Se realizaron pruebas puntuales sobre los sensores y actuadores, demostrando el correcto acceso a todos los pines con sus respectivos modos de funcionamiento (los cuales pueden observarse en la figura 4.2).

Para la verificación de Firmata de lado del cliente, se realizaron ensayos puntuales sobre la biblioteca *Johnny five*. En la asignatura de Protocolos de Comunicación en Sistemas Embebidos de la CESE FIUBA [22] se trabajó de forma puntual sobre el protocolo Firmata, realizando pruebas sobre un software web creado como trabajo final de la asignatura e instalado en la computadora, con el objetivo de ensayar todos los comandos del protocolo cliente sobre periféricos específicos conectados a la placa EDU-CIAA-NXP con *Firmata4CIAA*, y de esta manera realizar la manipulación de los mismos.

4.2. Ensayo de funcionamiento en los sistemas operativos Windows y Linux.

Este ensayo corresponde al “REQ1: El entorno de depuración deberá poder utilizarse dentro de los sistemas operativos Windows y Linux” y para llevarlo a cabo se realizaron los siguientes pruebas:

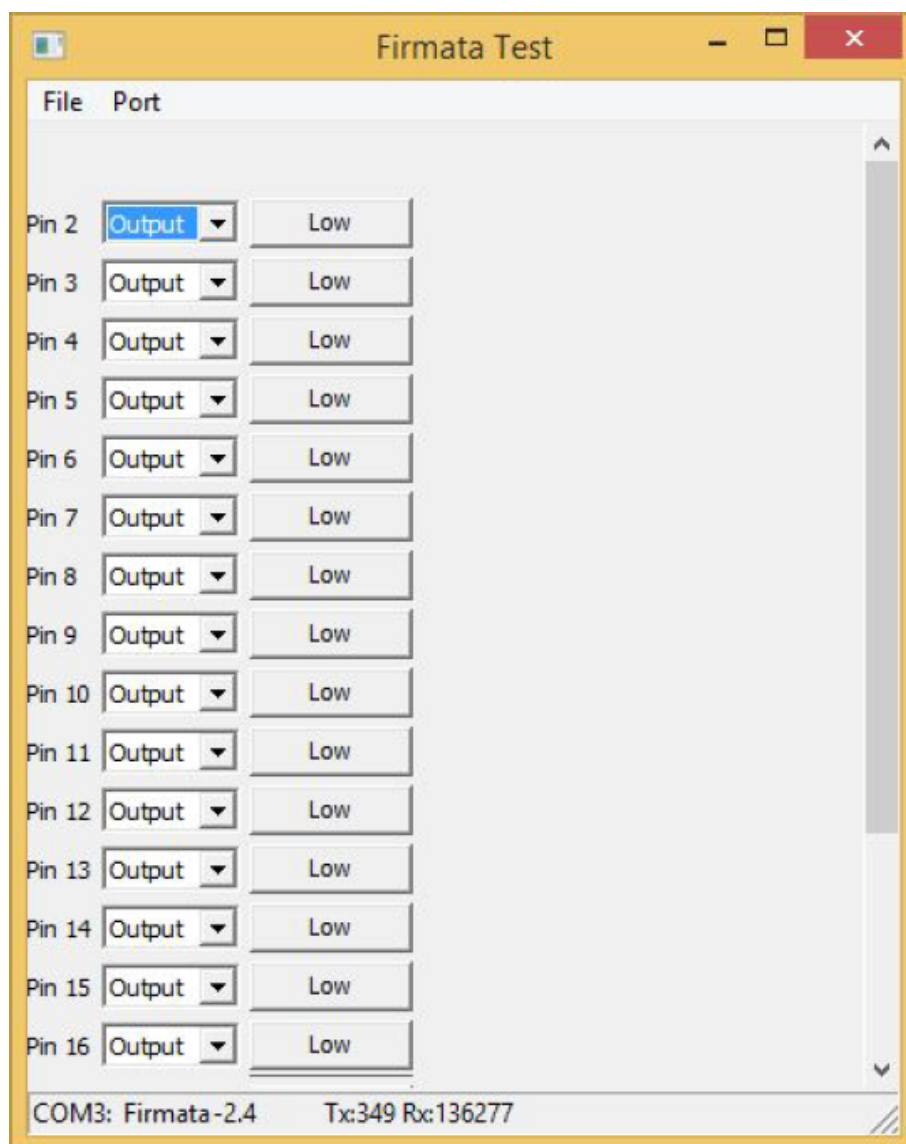


FIGURA 4.1: Programa Firmata test

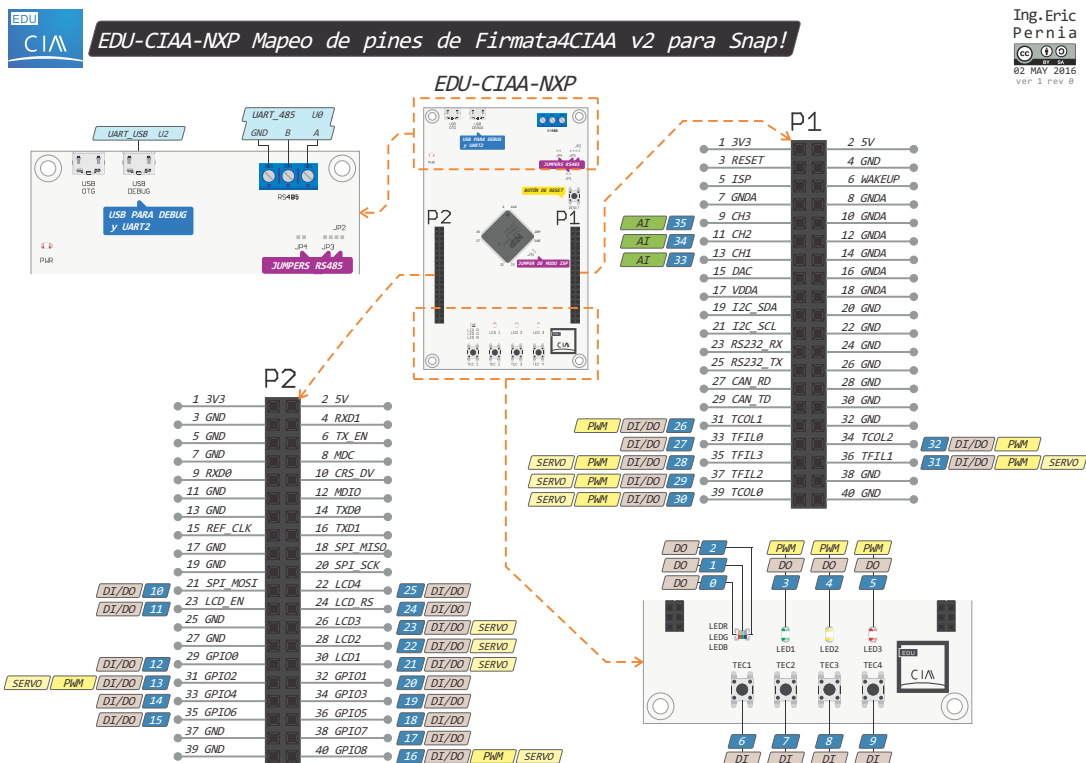


FIGURA 4.2: Firmata4CIAA

4.2.1. Instalación en cada sistema.

La instalación del sistema involucra la copia de los archivos necesarios para que el entorno de *CIAABOT Debug* pueda funcionar sin problemas, en ambos sistemas operativos. Para llevar a cabo este *test* se hizo la instalación en dos máquinas diferentes, una con el sistema operativo Windows y la otra con el sistema operativo Linux.

El ensayo incluyó el proceso de descarga de los archivos del programa, y al ser exitosa, se ejecutó manualmente el siguiente *test* funcional sobre el estado *Desconectado* para verificar de manera general el funcionamiento de *CIAABOT Debug*.

Test funcional posterior a la instalación en cada sistema

Pre-condición:

- *CIAABOT Debug* se encuentra descargado en la computadora.

Flujo principal:

1. Al Abrir *CIAABOT Debug* se muestra al usuario la pantalla principal del entorno.
2. El botón de inicio de sesión de depuración se muestra deshabilitado junto con las herramientas de control de ejecución.
3. La edición de bloques del programa se encuentra habilitada.

Resultado obtenido: en ambas computadoras se ejecutaron todos los pasos del *test* sin problemas y de manera exitosa.

4.2.2. Chequeo de conexión y descarga de Firmata4CIAA.

Entre los archivos de *CIAABOT Debug* se encuentra el archivo compilado que corresponde al proyecto de Firmata4CIAA para la EDU-CIAA-NXP. Para proceder a la descarga de Firmata4CIAA, el entorno requiere que el usuario configure la ruta del *toolchain* y de *OpenOCD*. Una vez realizado esto se comprobó mediante el siguiente *test* manual la correcta comunicación entre el entorno, el *toolchain* y *openOCD*.

Test funcional descargar Firmata4CIAA

Pre-condición:

- *CIAABOT Debug* se encuentra descargado en la computadora.
- La placa EDU-CIAA-NXP se encuentra conectada a la PC mediante el puerto marcado como *USB Debug* en la serigrafía. La placa no tiene grabado el programa Firmata4CIAA.

Flujo principal:

1. Al Abrir *CIAABOT Debug* se muestra al usuario la pantalla principal del entorno.
2. Al hacer click sobre el icono de configuración se muestra al usuario una ventana emergente para la configuración de los *paths*.
3. El usuario ingresa las rutas desde donde tiene el binario para el *toolchain* y *openOCD* y presiona el botón "Guardar".
4. El usuario presiona el icono de actualizar desde la conexión con la plataforma.
5. El entorno al estar en estado *Conectado Sin Firmata4CIAA* consulta al usuario si quiere descargar Firmata4CIAA.
6. El entorno avisa al usuario que la descarga fue exitosa.

Resultado obtenido: en ambas computadoras se ejecutaron todos los pasos del *test* sin problemas y de manera exitosa.

4.2.3. Chequeo de conexión con la plataforma EDU-CIAA-NXP.

El ensayo incluyó la comprobación del funcionamiento del inicio de la sesión de depuración para verificar que los bloques gráficos que acceden al hardware lo hagan de forma exitosa. Se ejecuto manualmente el siguiente *test* funcional:

Test funcional de conexión con la plataforma EDU-CIAA-NXP

Pre-condición:

- *CIAABOT Debug* se encuentra descargado en la computadora.
- La placa EDU-CIAA-NXP se encuentra conectada a la PC y con el programa *Firmata4CIAA*.
- El editor de programa de *CIAABOT Debug* tiene importado un programa realizado en *CIAABOT IDE*.

Flujo principal:

1. El programa de bloques se encuentra importado en el editor de programa de *CIAABOT Debug*.
2. Al usuario inicia la sesión de depuración haciendo click sobre el botón verde de *debug*.
3. El entorno habilita las herramientas de control de ejecución.
4. El usuario hace click en el botón del icono *ejecutar*.
5. El entorno al detectar que existe un bloque gráfico con acceso a un periférico en particular, procede a realiza la conexión con la placa EDU-CIAA-NXP y muestra la manipulación del dispositivo al usuario.

Resultado obtenido: en ambas computadoras se ejecutaron todos los pasos del *test* sin problemas y de manera exitosa.

4.3. Comprobación de funcionamiento de una sesión de depuración.

Para poder verificar los siguientes requerimientos:

- *REQ2: Debe controlar de la ejecución de un programa corriendo en la plataforma de hardware.*
- *REQ3: Exponer el estado de las variables del programa en tiempo de ejecución.*

fueron necesarios realizar los siguientes ensayos de forma manual:

4.3.1. Sesión de depuración uso del comando *Ejecutar*

Pre-condición:

- *CIAABOT Debug* se encuentra descargado en la PC.
- La placa EDU-CIAA-NXP se encuentra conectada a la PC y con el programa *Firmata4CIAA*.
- Existe un programa realizado en *CIAABOT IDE* con bloques gráficos de acceso al hardware.

Flujo principal:

1. Al Abrir *CIAABOT Debug* se muestra al usuario la pantalla principal del entorno.
2. El usuario elige el puerto serial desde donde tiene conectada la placa EDU-CIAA-NXP.
3. El entorno detecta que la placa EDU-CIAA-NXP tiene descargado el programa Firmata4CIAA y habilita el inicio de sesión de depuración, mostrando el botón de *debug* en color verde.
4. El usuario presiona el botón verde de *debug*.
5. El entorno deshabilita la edición de programas, habilita las herramientas de control de ejecución y cambia el color del botón debug de verde a naranja.
6. El usuario hace click en el botón de comando *ejecutar*.
7. El entorno ejecuta bloque por bloque el programa y al detectar que existe un bloque con acceso a un periférico envía la acción a la placa EDU-CIAA-NXP utilizando el protocolo firmata y muestra como salida la manipulación del dispositivo según lo programado por el usuario.
8. El entorno actualiza el menú de visualización de variables con los nombres y valores de esas variables que se encuentran encastradas dentro de los bloques que están ejecutándose.

Resultado obtenido: ejecución de todos los pasos del *test* en forma exitosa.

4.3.2. Sesión de depuración uso del comando *Suspend*

Pre-condición:

- Sesión de depuración iniciada y programa en ejecución con el uso del comando *ejecutar*.

Flujo principal:

1. El entorno se encuentra ejecutando el programa con los bloques gráficos que acceden a periféricos como los que no acceden a periféricos.
2. El usuario presiona el botón de *suspend*.
3. El entorno suspende la ejecución del programa, es decir, termina de ejecutar el bloque actual y se detiene antes del próximo bloque a ejecutar, el cual lo resalta con borde naranja.
4. El usuario reanuda la ejecución del programa mediante el botón *ejecutar*.

Resultado obtenido: ejecución de todos los pasos del *test* en forma exitosa.

4.3.3. Sesión de depuración uso del comando *Pasar por encima (step over)*

Pre-condición:

- Conexión con la plataforma establecida y la misma cuenta con Firmata4CIAA.
- Sesión de depuración iniciada.

Flujo principal:

1. El usuario hace click en el botón de comando *Pasar por encima (step over)*.
2. El entorno muestra resaltado el primer bloque del programa pero sin ejecutarlo.
3. El usuario vuelve a hacer click en el botón de comando *Pasar por encima (step over)*.
4. El entorno muestra resaltado el bloque siguiente, ejecuta el bloque que fue resaltado anteriormente y actualiza el valor de las variables en el menú de visualizaciones de variables.

Resultado obtenido: ejecución de todos los pasos del *test* en forma exitosa.

4.3.4. Sesión de depuración uso del comando *Pasar adentro (step into)*

Pre-condición:

- Conexión con la plataforma establecida y la misma cuenta con Firmata4CIAA.
- Sesión de depuración iniciada.

Flujo principal:

1. El usuario hace click en el botón de comando *Pasar adentro (step into)*.
2. El entorno muestra resaltado el primer bloque del programa pero sin ejecutarlo.
3. El usuario vuelve a hacer click en el botón de comando *Pasar adentro (step into)*.
4. El entorno muestra resaltado el bloque siguiente, detecta que el bloque tiene un llamado a función y saltara a seleccionarlo.
5. El usuario vuelve a hacer click en el botón de comando *Pasar adentro (step into)*.
6. El entorno muestra resaltado el bloque siguiente dentro de la función y ejecutara el bloque anterior.

Resultado obtenido: ejecución de todos los pasos del *test* en forma exitosa.

4.3.5. Sesión de depuración uso del comando *Paso de regreso (step return)*

Pre-condición:

- Conexión con la plataforma establecida y la misma cuenta con Firmata4CIAA.
- Sesión de depuración iniciada y programa en ejecución con el uso del comando *Pasar adentro (step into)*.

Flujo principal:

1. El entorno se encuentra ejecutando el programa con los bloques gráficos resaltado los bloques con el uso del comando *Pasar adentro (step into)* y ahora se encuentra dentro de una función.
2. El usuario hace click en el botón de comando *Paso de regreso (step return)*.
3. El entorno ejecuta el bloque seleccionado y los siguientes sin seleccionarlos hasta encontrar el retorno de la función, luego seleccionara el siguiente bloque a continuación de la función que ejecuto.

Resultado obtenido: ejecución de todos los pasos del *test* en forma exitosa.

4.3.6. Sesión de depuración con puntos de ruptura (*breakpoints*)

Pre-condición:

- Sesión de depuración iniciada.

Flujo principal:

1. El usuario hace *click* con el botón derecho del *mouse* en cualquiera de los bloques gráficos del programa.
2. El entorno muestra el menú contextual del bloque desde donde se puede establecer el punto de interrupción.
3. El usuario hace click dentro del menú contextual del bloque.
4. El entorno establece el punto de interrupción en ese bloque, agrega un círculo rojo al mismo, en señal de *breakpoint* y actualiza la ventana de visualización de puntos de ruptura con la información del nuevo *breakpoint*.
5. El usuario hace click en el botón del comando *ejecutar*.
6. El entorno ejecuta cada bloque de programa hasta llegar al bloque marcado con punto de interrupción donde detiene la ejecución y resalta el bloque con el *breakpoint* que aún no ejecutó.
7. El usuario hace click sobre el botón *ejecutar*.
8. El entorno continuará con la ejecución.

Resultado obtenido: ejecución de todos los pasos del *test* en forma exitosa.

4.3.7. Sesión de depuración uso del comando desactivar puntos de interrupción

Pre-condición:

- Sesión de depuración iniciada.
- Puntos de interrupción establecidos en algunos de los bloques.

Flujo principal:

1. El usuario hace click en el botón de *desactivar puntos de interrupción*.

2. El entorno muestra el botón de *desactivar puntos de interrupción* deshabilitado en señal de que se saltarán todos los puntos durante la ejecución y actualiza el menú de visualización de puntos de ruptura (*breakpoints*) mostrándolos desactivados.
3. El usuario hace click en el botón del comando *ejecutar*.
4. El entorno resalta bloque por bloque el programa a medida que se va ejecutando sin detenerse al encontrar un punto de ruptura en el bloque establecidos anteriormente.
5. El usuario hace click en el botón de *desactivar puntos de interrupción*.
6. El entorno muestra el botón de *desactivar puntos de interrupción* habilitado y actualiza el menú de visualización de puntos de ruptura (*breakpoints*) mostrándolos activados nuevamente.

Resultado obtenido: ejecución de todos los pasos del *test* en forma exitosa.

4.3.8. Sesión de depuración desactivar puntos de interrupción desde la ventana de visualización de *breakpoints*

Pre-condición:

- Sesión de depuración iniciada.
- Puntos de interrupción establecidos en algunos de los bloques.

Flujo principal:

1. El usuario hace click en uno de los *checkbox* actualmente tildado dentro del menú de visualización de puntos de ruptura (*breakpoints*) de la columna "Activo".
2. El entorno muestra el *checkbox* destildado.
3. El usuario hace click en el botón del comando *ejecutar*.
4. El entorno ejecuta los bloques de programa sin detenerse cuando encuentra el punto de ruptura en el bloque que fue destildado en la ventana de visualización de puntos de ruptura.
5. El usuario hace click en el *checkbox* actualmente destildado dentro del menú de visualización de puntos de ruptura (*breakpoints*) de la columna "Activo".
6. El entorno ejecuta los bloques de programa y se detiene cuando encuentra el punto de ruptura en el bloque que fue tildado como activo desde la ventana de visualización de puntos de ruptura.

Resultado obtenido: ejecución de todos los pasos del *test* en forma exitosa.

4.3.9. Sesión de depuración uso del comando *Detener depuración*

Pre-condición:

- Conexión con la plataforma establecida y la misma cuenta con firmata.

Flujo principal:

1. El entorno se encuentra con una sesión de depuración iniciada (se comprueban los casos con sesión en ejecución y con sesión de suspensión en ejecución).
2. El usuario presiona el botón de *detener depuración*.
3. El entorno termina la ejecución del programa, finaliza la sesión de depuración, cambia el botón de *debug* de color naranja a color verde y por ultimo habilita la edición del programa.

Resultado obtenido: ejecución de todos los pasos del *test* en forma exitosa.

4.3.10. Guardado y restauración de los puntos de ruptura

Pre-condición:

- *CIAABOT Debug* se encuentra descargado en la PC.
- La placa EDU-CIAA-NXP se encuentra conectada a la PC y con el programa *Firmata4CIAA*.
- Existe un programa realizado en *CIAABOT IDE* y abierto en el editor de programas del entorno de *debug*.
- Sesión de depuración iniciada.

Flujo principal:

1. El usuario hace click derecho en cualquiera de los bloques gráficos que se muestran dentro del editor.
2. El entorno muestra el menú contextual del bloque desde donde se puede establecer el punto de interrupción.
3. El usuario hace click dentro del menú contextual del bloque.
4. El entorno establece el punto de interrupción en ese bloque, agrega un punto rojo en señal de *breakpoint* y actualiza la ventana de visualización de puntos de ruptura con la información del nuevo *breakpoint*.
5. El usuario hace click en el botón del comando *ejecutar*.
6. El entorno ejecuta los bloques de programa.
7. El usuario detiene la sesión de depuración y cierra el programa, o abre otro programa.
8. El entorno actualiza el archivo *debug.cbd*. dentro de la carpeta que contiene el proyecto automáticamente.
9. El usuario hace click nuevamente en el menú "Proyecto" de la aplicación y abre el proyecto usado previamente en el punto anterior.
10. El entorno carga nuevamente el programa en el editor de *debug*.
11. El usuario comprueba que el punto de interrupción establecido anteriormente para ese bloque aparece nuevamente en el editor.

Resultado obtenido: ejecución de todos los pasos del *test* en forma exitosa.

4.4. Ensayos de edición de programas en lenguaje CIAABOT.

Para poder verificar el requerimiento “REQ4: Permitir la edición de programas para corregir errores de lógica encontrados en tiempo de ejecución” fue necesario realizar el siguiente ensayo de forma manual:

4.4.1. Edición de programa

Pre-condición:

- CIAABOT Debug se encuentra descargado en la PC.
- La placa EDU-CIAA-NXP se encuentra conectada a la PC y con el programa Firmata4CIAA.
- Existe un programa realizado en CIAABOT IDE.

Flujo principal:

1. Al Abrir CIAABOT Debug se muestra al usuario la pantalla principal del entorno.
2. El usuario elige el puerto serial desde donde tiene conectada la placa EDU-CIAA-NXP.
3. El entorno detecta que la placa EDU-CIAA-NXP tiene descargado el programa Firmata4CIAA y habilita el inicio de sesión de depuración, mostrando el botón de *debug* en color verde.
4. El usuario realiza la modificación de los bloques gráficos, para ello agrega un nuevo bloque a la estructura de bloques del programa y guarda el programa actualizado desde el menú "Proyecto" de la aplicación.
5. El entorno guarda el programa actualizado y actualiza el archivo .cbp dentro de la carpeta que contiene el proyecto guardado.
6. El usuario hace click en el menú "Proyecto" de la aplicación y abre el proyecto que fue guardado en el punto anterior.
7. El entorno carga nuevamente el programa en el editor de *debug*.
8. El usuario comprueba que el bloque gráfico agregado en el punto anterior se encuentre en el programa cargado.

Resultado obtenido: ejecución de todos los pasos del *test* en forma exitosa.

4.5. Ensayos de interfaz de usuario.

Para poder verificar los requerimientos:

- REQ5: La interfaz gráfica debe seguir los lineamientos de estilo establecidos en el Proyecto CIAABOT.

- *REQ6: Debe presentar un diseño de interfaz gráfica intuitiva, con elementos similares a los hallados en otras herramientas de depuración.*

Como parte de la verificación en los estilos visuales del diseño de la interfaz del entorno de *debug* se realizaron reuniones con el director de este trabajo y con especialistas en diseño de interfaz de usuario para la evaluación de prototipos html, con la intención de descubrir errores en el funcionamiento que no cumplan los requerimientos, y además el de tener presente los posibles problemas de uso que podrían tener los usuarios.

4.6. Validación con usuarios.

Para verificar el requerimiento “*REQ6: Debe presentar un diseño de interfaz gráfica intuitiva, con elementos similares a los hallados en otras herramientas de depuración*” se realizaron ensayos de los usuarios finales en el marco de un laboratorio de programación llevado a cabo en la materia “*Computadores 2 / Técnicas Avanzadas de Programación UNQ 2018*” [23] de la Universidad nacional de Quilmes. Los usuarios con experiencia de depuración en lenguaje C pudieron utilizar el entorno sin necesidad de algún soporte en la utilización de la herramienta de comandos para depuración que ofrece la plataforma, siendo esto prueba que el diseño de interfaz gráfica es intuitiva y de fácil uso.

Capítulo 5

Conclusiones

5.1. Conclusiones generales

En el presente Trabajo Final se ha logrado obtener un entorno de depuración para la plataforma CIAABOT. El entorno de depuración *CIAABOT Debug* otorga una interfaz simple e intuitiva para depurar programas en lenguaje CIAABOT, utilizando la plataforma EDU-CIAA-NXP.

En el desarrollo de este Trabajo Final se aplicaron los conocimientos adquiridos en la Carrera de Especialización en Sistemas Embebidos, en especial las asignaturas:

- Programación de microprocesadores. En esta materia se aprendió a utilizar la EDU-CIAA-NXP y herramientas de depuración en lenguaje C. Estas herramientas permitieron entender como se utiliza la depuración de un sistema embebido desde la PC.
- Ingeniería de software en sistemas embebidos. Se utilizaron técnicas y metodologías de trabajo de la ingeniería del software. Se usaron los conocimientos de manejo de repositorio Git como sistema de control de versiones y se aplicaron metodologías de ensayo de software.
- Gestión de Proyectos en Ingeniería. Plan de Proyecto para organizar el Trabajo Final, permitió desde el inicio de la carrera tener una visión de todas las tareas a realizar y su estimación en tiempos.
- Protocolos de Comunicación. Se aplicaron los conocimientos obtenidos del protocolo Firmata.
- Sistemas Operativos de Propósito General. Se usaron los conocimientos sobre Linux, usando las herramientas que provee para usarse en las pruebas generales del software creado.

El entorno de *CIAABOT debug* le da la capacidad de depuración al proyecto CIAABOT, por lo tanto también aporta al Proyecto CIAA, y a la comunidad de sistemas embebidos en general.

Se concluye que los objetivos del trabajo han sido cumplidos satisfactoriamente, se logró un mejor conocimiento del hardware y el desarrollo del mismo ha favorecido en gran medida a la formación profesional de la autora.

5.2. Próximos pasos

Se propone agregar las siguientes características al entorno de depuración:

- Capacidad de evaluación de expresiones.
- Ver *stack* de llamadas a funciones.
- Consola de comandos del *debug*.

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] ARM. *GNU ARM Embedded Toolchain, herramientas para compilación cruzada para procesadores Arm Cortex-M y Cortex-R. Incluye Arm Embedded GCC compiler, bibliotecas y otras herramientas GNU necesarias*. Disponible: 2018-11-29. URL: <https://developer.arm.com/open-source/gnu-toolchain/gnu-rm>.
- [8] Dominic Rath. *Open On-Chip Debugger, herramienta para depuración, programación y boundary-scan para sistemas embebidos*. Disponible: 2018-11-29. URL: <http://openocd.org/>.
- [9] CIAA Firmware v2. 2016. URL: https://github.com/ciaa/firmware_v2.
- [10] *Sitio web de referencia*. URL: <https://cmake.org/>.
- [11] sAPI. 2017. URL: <https://github.com/epernia/sAPI>.
- [12] Mark Otto y Jacob Thornton de Twitter. *Repositorio de Bootstrap*. URL: <https://github.com/twbs/bootstrap>.
- [13] Neil Fraser. *Repositorio de JS-Interpreter*. URL: <https://github.com/NeilFraser/JS-Interpreter>.
- [14] Marijn Haverbeke. *Repositorio de Acorn*. URL: <https://github.com/acornjs/acorn>.
- [15] John Resig. *Repositorio de JQuery*. URL: <https://github.com/jquery/jquery>.
- [16] Robert Penner's. *Repositorio de Signals*. URL: <https://github.com/millermedeiros/js-signals>.
- [17] *Repositorio de Johnny five*. URL: <https://github.com/rwaldron/johnny-five>.
- [18] *Repositorio de Express*. URL: <https://github.com/expressjs/express>.
- [19] *Repositorio de Serialport*. URL: <https://github.com/node-serialport/node-serialport>.
- [20] *Sitio web oficial de Json*. URL: <https://www.json.org>.
- [21] *Sitio web de referencia*. URL: <http://firmata.org/>.
- [22] *Carrera de Especialización en Sistemas Embebidos (CESE) de la UBA*. URL: <http://laboratorios.fi.uba.ar/lse/especializacion.html>.
- [23] *Computadores 2/Técnicas avanzadas de programación*. URL: <https://sites.google.com/site/computadores2unq2018/>.