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

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción General

En este capítulo se presenta una breve introdución a la plataforma CIAABOT, 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[1].

CIAABOT es una plataforma de software y hardware abierto que forma parte del proyecto CIAA (Computadora Industrial Abierta Argentina)[5].

La implementacion 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[3]), que es un lenguaje gráfico donde un programa se crea encastrando bloques. Permite crear el programa, compilarlo y descargarlo a las plataformas de hardware.

[1] IDE son las siglas en inglés de entorno de desarrollo integrado.

- CIAABOTS: son las plataformas de hardware que se programan desde CIAA-BOT IDE. Actualmente disponible únicamente la EDU-CIAA-NXP.
- Firmware: el programa creado en CIAABOT IDE genera internamente código C que se combina con el firmare 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:

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

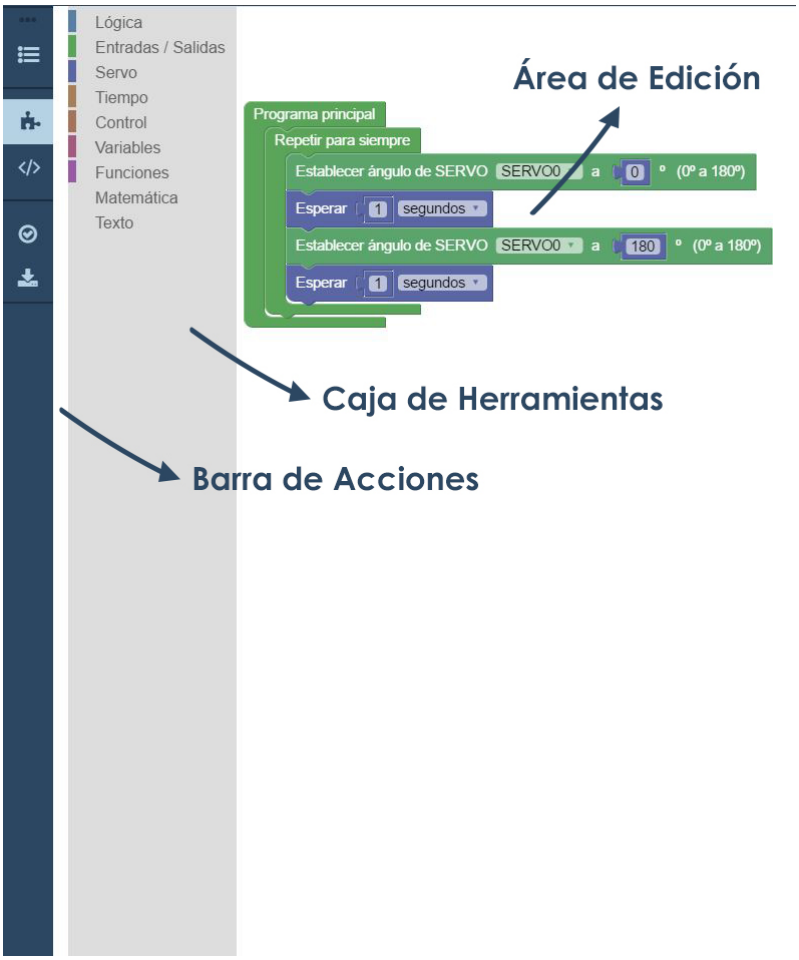

FIGURA 1.1: Área de flujo de trabajo de CIAABOT-IDE. Recuperado de https://laboratorios.fi.uba.ar

1.1.2. *Debug* y *Debugger*

La depuración 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 obteniendo información muy importante 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 depurando el código, parandolo o pausandolo, de esta manera se realiza el seguimiento de valores de las variables críticas, dando en consecuencia la posibilidad al programador de corregir errores y pulir el funcionamiento.

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 modernos 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.

La plataforma CIAABOT no posee ninguna forma que permita *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.

1.2. Motivación

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.

La importancia de usar un *debugger* a nivel de bloques, es que permite al usuario ir ejecutar las rutinas de código de manera mas pausada, ademas de ir monitoreando el estado de las variables que son parte de su programa, ayudándolos a identificar y subsanar los errores de ejecución.

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 implementacion 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 un poco más de detalle, se analiza una alternativa real para depuración y se justifica el diseño propuesto de *debugging* a implementar, luego se establecen los requerimientos y la planificación para el desarrollo del presente trabajo.

2.1. Componentes CIAABOT

La plataforma 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.

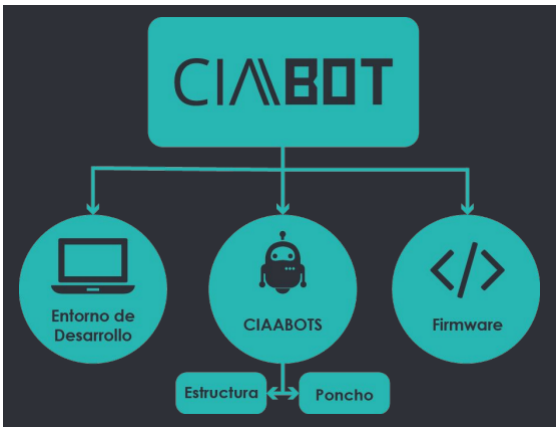

FIGURA 2.1: Componentes CIAABOT. Recuperado de https://laboratorios.fi.uba.ar

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 empezar a desarrollar su propio programa gráfico encastrando de manera fácil bloques predifinidos creado en lenguaje javascript, como se observa en la figura 2.2

El entorno de desarrollo integrado está diseñado para crear programas de forma sencilla encastrando bloques gráficos. Asimismo, 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.

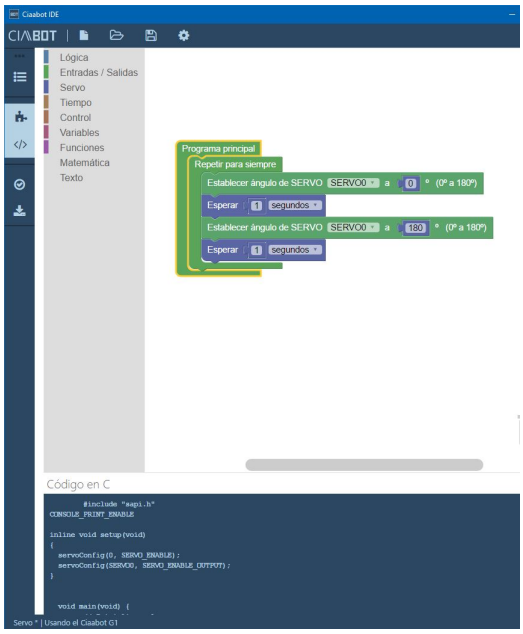

FIGURA 2.2: Editor gráfico de CIAABOT-IDE.

La plataforma CIAABOT esta programado usando las siguientes tecnologías:

- Angular[2].
- Blockly[3].
- Electrón[7].
- NodeJS[8].
- TypeScript[4]

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 bloque genera el correspondiente codigo 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.

2.1.2. CIAABOTS

La plataforma usa CIAABOTS para llamar así 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 esta basado en el firmware v2 [1] del proyecto CIAA, puede crear desde funciones simples a más complejas, para realizarlo utiliza las siguientes herramientas:

- *Makefile*: para la gestión de dependencias, de esta manera puede construir el software desde sus archivos fuente.
- *OpenOCD*: una herramienta OpenSource, usado para el grabado del firmware en las placas.
- *sAPI*[6]: permite manejar los periféricos del microcontrolador de una manera muy sencilla.

2.2. Alternativas de diseño para CIAABOT debugger

Para diseñar el software de depuración se toma como caso de estudio, la depuración de un programa en lenguaje C mediante Eclipse IDE:

- *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.
- *arm-none-eabi-gdb* [2]: 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 brakpoints. Se podría debuggear también, por línea de comandos directamente sobre el código C abriendo GDB en una terminal con comandos GDB-MI y usando los archivos .elf y .bin generados.
- *OpenOCD (Open On-Chip Debugger)*[3]: prevee 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"*).

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

[2]Software de debug originario de linux.

[3]Software de código abierto que interactúa con el puerto JTAG de un depurador de hardware.

- *Debugger (HW)*: en el caso de la EDU-CIAA es la placa de interfaz fisica para pasar de JTAG a USB (modo puerto serie virtual). En la EDU-CIAA viene incluido en la misma placa que esta el micro y usa todo el circuito de hardware para debug.
- *Microcontrolador a debuggear*: El microcontrolador posee un periferico específico para *debug* con interfaz JTAG [4], teniendo acceso para modificar la RAM, Flash (externas al microcontrolador) 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 visto anteriormente, 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 quiere manipular.

Debido a que existe como antecedente el proyecto firmata4CIAA se decidió utilizarlo como protocolo para acceso al Hardware.

2.3. Firmata

Firmata es un protocolo genérico para comunicarse con microcontroladores desde el software en una computadora (o teléfono inteligente / tableta, etc.).

Este protocolo fue diseñado para la comunicación directa entre un microcontrolador y un objeto de software en una computadora host Cliente Firmata. En la figura 2.3 se muestra la Arquitectura de este protocolo.

Firmata4CIAA es un programa que implementa el protocolo firmata en la EDU-CIAA-NXP.

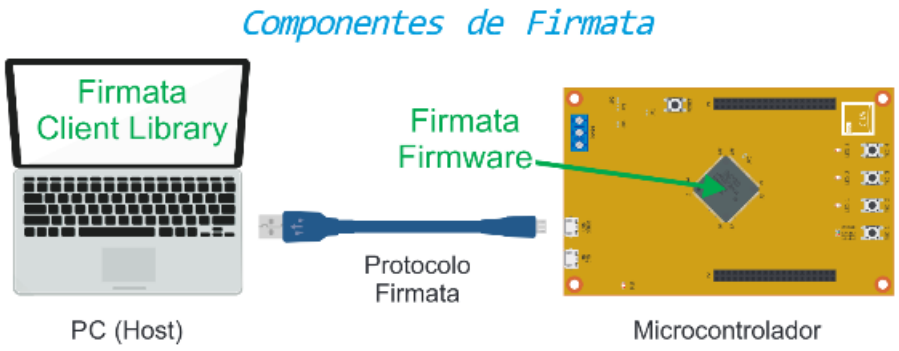

FIGURA 2.3: Arquitectura de Firmata.

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

2.3.1. Johnny-Five

Cliente firmata, basado en el lenguaje JavaScript. Es un marco de programación de fuente abierta, basado en el protocolo firmata, IoT y Robótica. En la figura 2.4 se presenta a este framework.

FIGURA 2.4: Cliente firmata: Johnny-Five. Recuperado de http/ /johnny-five.io

De esta manera se propone la comunicación entre la aplicación y la placa EDUCIAA-NXP.

2.4. Sistema para depuración propuesto

Después de lo visto en las secciones anteriores, se propone el siguiente sistema a diseñar:

- Ejecutar un programa línea a línea, que ejecute los bloques en javascript en la PC del usuario..
- Enviar comandos firmata a la placa, solo 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 protocolo firmata de la CIAA, sólo en el caso de ser necesario.
- Entorno gráfico amigable.

2.4.1. Limitación

En este sistema propuesto, existen dos tipos de ejecución de bloques:

- 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.

- 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.

Los bloques gráficos que acceden a los periféricos, en tiempo de acceso de ejecución, serán más lentos que si se tuviera que implentar la alternativa de diseño de la sección 2.2, 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.

En conclusion si medimos los tiempos de ensayos, cuando se instala el firmware del codigo C generado en la placa contra la ejecución de los bloques con firmata, estos tiempos no serán los mismos.

2.4.2. Ventaja

Para el usuario de CIAABOT, este método será una ventaja, ya 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.

2.5. Requerimientos

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

2.5.1. 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.

2.5.2. 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 2.5) para el control de los robots.

FIGURA 2.5: Placa EDU-CIAA-NXP

2.5.3. 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 mecanismos de ahorro de la flash, para saber si la placa ya tiene firmata.

2.5.4. 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.

2.6. Planificación

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

2.6.1. Desglose en tareas

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

- Pluggin Debugger
- Código fuente del proyecto debugger.
- 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 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 traves 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.

2.6.2. Activity On-node

En el diagrama de Activity on node de la figura 2.6 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.

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

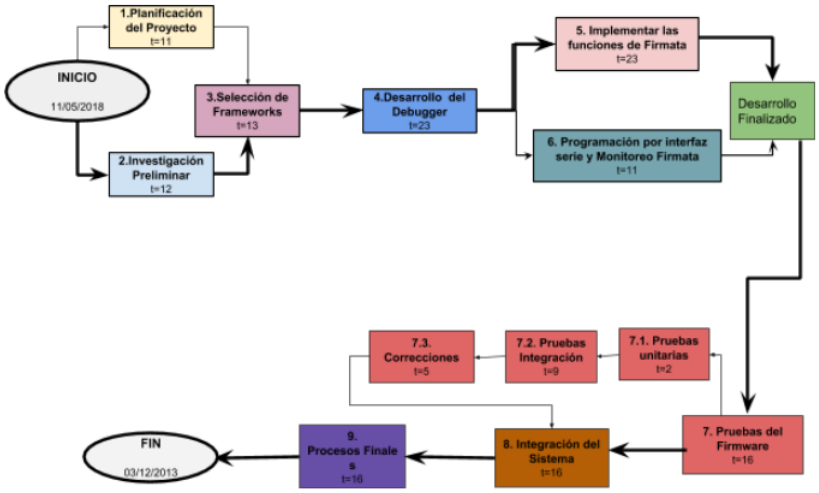

FIGURA 2.6: Diagrama de Node.

en la siguiente figura 2.7 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 2.7: Tabla de colores diagrama Activity

2.6.3. 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 2.8 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.

En la figura 2.9 se puede observar la segunda parte del diagrama para este proyecto.

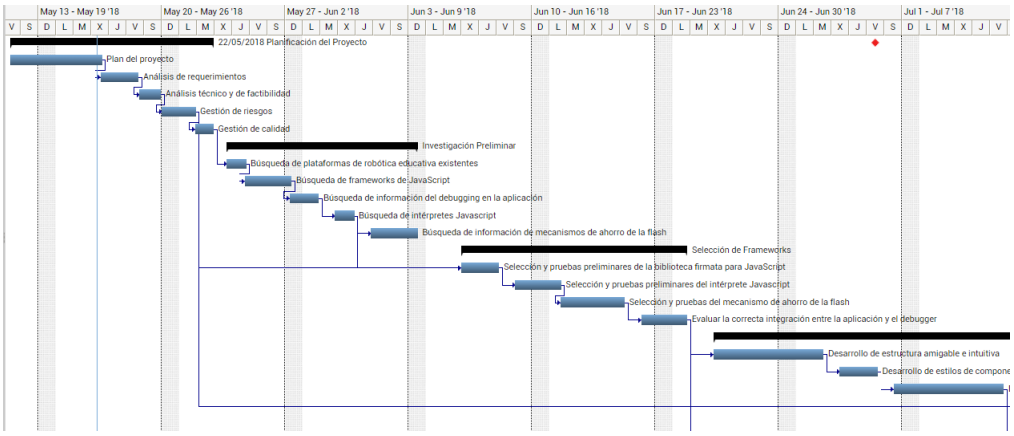

FIGURA 2.8: Diagrama de Gantt - Parte 1.

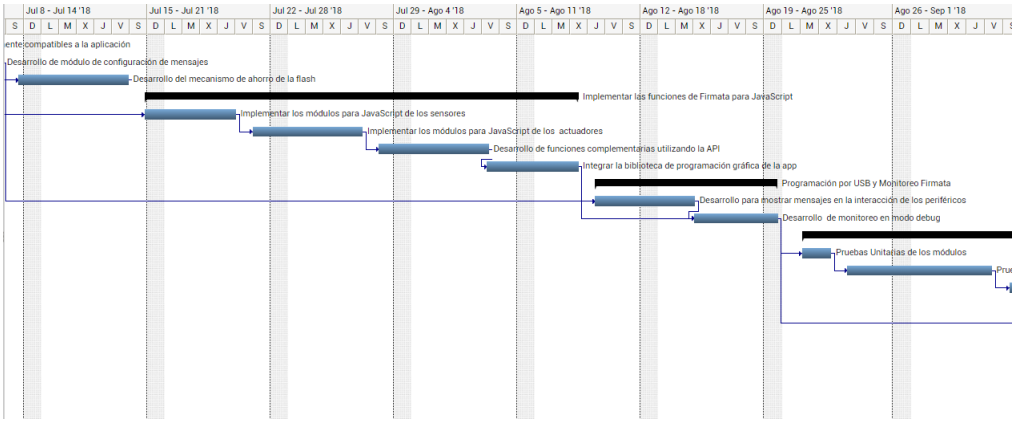

FIGURA 2.9: Diagrama de Gantt - Parte 2.

En la figura 2.10 se puede observar la tercera parte del diagrama para este proyecto. Y la cuarta parte del diagrama se puede observar la figura 2.11

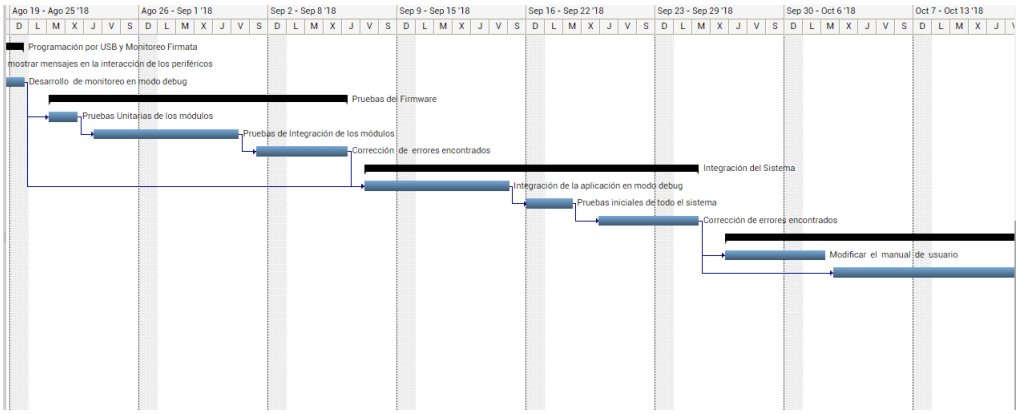

FIGURA 2.10: Diagrama de Gantt - Parte 3.

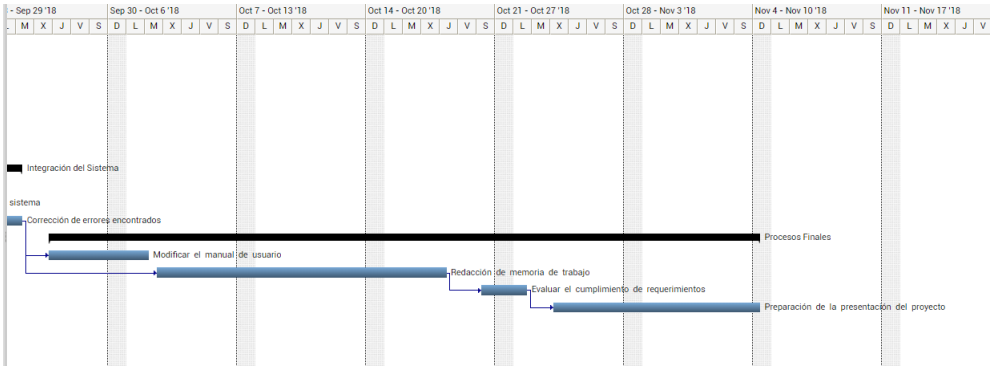

FIGURA 2.11: Diagrama de Gantt - Parte 4.

Capítulo 3

Diseño e Implementación

3.1. Estructura del software

La idea de esta sección es resaltar los problemas encontrados, los criterios utilizados y la justificación de las decisiones que se hayan tomado.

Se puede agregar código o pseudocódigo dentro de un entorno lstlisting con el siguiente código:

```
\begin{lstlisting}[caption= "un epígrafe descriptivo"]

las líneas de código irían aquí...

\end{lstlisting}
```

A modo de ejemplo:

```
#define MAX_SENSOR_NUMBER 3
#define MAX_ALARM_NUMBER  6
#define MAX_ACTUATOR_NUMBER 6

uint32_t sensorValue[MAX_SENSOR_NUMBER];
FunctionalState alarmControl[MAX_ALARM_NUMBER]; //ENABLE or DISABLE
state_t alarmState[MAX_ALARM_NUMBER];           //ON or OFF
state_t actuatorState[MAX_ACTUATOR_NUMBER];     //ON or OFF

void vControl() {

  initGlobalVariables();

  period = 500 ms;

  while(1) {

    ticks = xTaskGetTickCount();

    updateSensors();

    updateAlarms();

    controlActuators();

    vTaskDelayUntil(&ticks, period);
  }
```

```
28 }
```

ALGORITMO 3.1: Pseudocódigo del lazo principal de control.

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] *CIAA Firmware v2*. 2016. URL: https://github.com/ciaa/firmware_v2.
[2] *Documentación de Angular*. URL: https://angular.io/docs.
[3] *Documentación de Blockly*. URL: https://developers.google.com/blockly/.
[4] *Documentación de TypeScript*. URL: https://www.typescriptlang.org.
[5] Proyecto CIAA. *Computadora Industrial Abierta Argentina*. Disponible: 2016-06-25. 2014. URL: http://proyecto-ciaa.com.ar/devwiki/doku.php?id=start.
[6] *sAPI*. 2017. URL: https://github.com/epernia/sAPI.
[7] *Sobre Electron*. URL: https://electron.atom.io/docs/tutorial/about/.
[8] *Sobre NodeJS*. URL: https://nodejs.org/en/about/.