Multiplataforma de Programación, Adquisición y Visualización Interactiva de Datos para Diseño, Simulación e Implementación de Sistemas de Control.

DEIVIS ALEXANDER ROSERO YELA

UNIVERSIDAD DE NARIÑO

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

SAN JUAN DE PASTO

2014

Multiplataforma de Programación, Adquisición y Visualización Interactiva de Datos para Diseño, Simulación e Implementación de Sistemas de Control.

DEIVIS ALEXANDER ROSERO YELA

TRABAJO DE GRADO PARA OPTAR POR EL TITULO DE INGENIERO ELECTRÓNICO

Supervisado por:
PhD. Andrés Pantoja Bucheli

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA
SAN JUAN DE PASTO
2014

NOTA DE RESPONSABILIDAD

“La Universidad de Nariño no se hace responsable por las opiniones y resultados obtenidos en el presente trabajo y para su publicación priman las normas sobre el derecho del autor”.

Acuerdo 1. Artículo 324. Octubre 11 de 1966, emanado del honorable Consejo Directivo de la Universidad de Nariño.

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

_____

_____

Firma Presidente del Jurado

_____

Firma Jurado 1

_____

Firma Jurado 2

San Juan de Pasto, 29 de septiembre de 2014

AGRADECIMIENTOS

Agradecimientos a:

Dios y mi familia

RESUMEN

En este documento se presenta el proceso de desarrollo e implementación de una plataforma versátil y robusta con las siguientes características: i) funciones de un sistema de adquisición de datos normal e inalámbrica compatible con Matlab, Simulink, Processing, Java; ii) capacidad de un minicomputador y un sistema operativo como Ubuntu para permitir su programación, el desarrollo de software e interactividad con múltiples programas; iii) con una arquitectura electrónica y un programa exclusivo para la programación interna y externa pensado en el desarrollo de hardware embebido; iv) manejo de redes a través de internet y compatibilidad con protocolo Zigbee para redes de sensores; y v) aplicaciones Android para usarla como osciloscopio, puente interactivo de datos y para Simulación virtual usando el toolbox Simulink 3D Simulation y dispositivos móviles. El enfoque central de este dispositivo es permitir el diseño, simulación e implementación de diversas estrategias de control como Fuzzy, PID, PID Adaptativo y Dinámica de replicadores.

En este trabajo inicialmente se presenta el diseño del software necesario para cubrir todas las capacidades preestablecidas incluyendo la parte de control, posteriormente se define la arquitectura electrónica y finalmente se presenta el desarrollo matemático de las librerías de control junto con la implementación y descripción del uso.

Índice de Términos—Plataforma de Desarrollo, Sistema de Adquisición de Datos, Matlab, Estrategias de Control.

ABSTRACT

THIS DOCUMENT IS PRESENTED IN THE PROCESS OF DEVELOPMENT AND IMPLEMENTATION OF A ROBUST VERSATILE PLATFORM WITH THESE FEATURES I) FUNCTIONS OF A DATA ACQUISITION SYSTEM COMPATIBLE WITH NORMAL AND WIRELESS MATLAB, SIMULINK, PROCESSING, JAVA; II) A MINICOMPUTER CAPACITY AND AN OPERATING SYSTEM UBUNTU AS TO ENABLE PROGRAMMING AND SOFTWARE DEVELOPMENT OF MULTIPLE INTERACTIVE PROGRAMS; III) WITH AN ELECTRONIC ARCHITECTURE AND A UNIQUE PROGRAM FOR INTERNAL AND EXTERNAL PROGRAM DEVELOPMENT THOUGHT OF EMBEDDED HARDWARE; IV) NETWORK MANAGEMENT THROUGH INTERNET AND COMPATIBILITY WITH ZIGBEE PROTOCOL FOR SENSOR NETWORKS; YV) ANDROID APPLICATIONS FOR USE AS SCOPE, INTERACTIVE DATA BRIDGE AND VIRTUAL SIMULATION USING SIMULINK TOOLBOX 3D SIMULATION AND MOBILE DEVICES. THE CENTRAL FOCUS OF THIS DEVICE IS ALLOWING THE DESIGN, SIMULATION AND IMPLEMENTATION OF DIFFERENT STRATEGIES OF FUZZY CONTROL AS, PID, PID AND ADAPTIVE DYNAMICS REPLICATORS.

INITIALLY IN THIS WORK NECESSARY SOFTWARE DESIGN IS PRESENTED TO COVER ALL OF PRESETS INCLUDING PART OF CONTROL, ELECTRONIC LATER DEFINED ARCHITECTURE AND FINALLY THE MATH OF THE LIBRARIES OF CONTROL WITH THE IMPLEMENTATION OF USE DEVELOPMENT AND DESCRIPTION IS PRESENTED.

INDEX TERMS-PLATFORM DEVELOPMENT DATA ACQUISITION SYSTEM, MATLAB, CONTROL STRATEGIES.

Índice general

I. INTRODUCCIÓN 15
1. DISEÑO DE SOFTWARE 16
1. DISEÑO DE LIBRERÍA DE MATLAB y SIMULINK 16
1.1 Bases preliminares 16
1.2 Parámetros de diseño 17
1.3 Construcción de librería de Teslamat 18
1.4 Construcción de la librería Teslasim para Simulink 23
2. DISEÑO Y CONSTRUCCION DE APLICACIÓN ANDROID 30
2.1 Bases Preliminares. 30
2.2 Parámetros de Diseño. 30
2.3 Estructura General de la Aplicación. 31
2.4 Interfaz Gráfica 32
2.5 Resumen de Resultados 37
3. MÉTODO PARA SIMULACIÓN VIRTUAL 39
3.1 Bases Preliminares 39
3.2 Parámetros de diseño 39
3.3 Estructura de Hardware y Software. 40
3.4 Algoritmo programado en la Tarjeta Tesla 41
4. DISEÑO Y ELABORACIÓN DE UN PROGRAMA PARA LA COMPILACIÓN Y CARGA DE CÓDIGO. 45
4.1 Bases preliminares 46
4.2 Parámetros de Diseño 47
4.3 Estructura del programa 48
4.4 Lenguaje Nativo 50
4.5 Resultados 54
5. SISTEMA OPERATIVO y REDES 56
5.1 Bases preliminares 56
5.2 Parámetros de configuración y selección 57
5.3 Sistema Operativo 57
5.4 Modificación del núcleo 57
5.5 Instalación de programas 58
5.6 Redes. 59
II. DISEÑO DE HARDWARE 60
1. ESTRUCTURA ELECTRÓNICA GENERAL 60

2. FUNCIONES DEL SISTEMA ELECTRÓNICO 63
3. ENTRADAS ANALÓGICAS, CIRCUITO DE PROTECCIÓN Y MULTIPLEXADO 66
4. SALIDAS PWM 69
5. SALIDAS Y ENTRADAS DIGITALES 70
6. PANTALLA Táctil 71
7. SALIDAS DE CONTROL AC 73
8. BLUETOOTH, MÓDULO XBEE, TARJETAS USADAS EN EL DISEÑO 74
9. IMPLEMENTACIÓN 75
III. librerías de control 79
1. LIBRERÍA DE CONTROL PID 79
1.1 Bases conceptuales. 79
1.2 Entradas y Salidas de Algoritmo 80
1.3 Controlador y Planta 80
1.4 Consideraciones para la elaboración del algoritmo PID en lenguaje C. 81
1.5 Algoritmo PID 84
1.6 Funciones de la Librería PID 85
1.7 Ejemplo de uso 86
2. LIBRERÍA DE CONTROL FUZZY 87
2.1 Bases Teóricas 87
2.2 Entradas y salidas de algoritmo 93
2.3 Consideraciones para la elaboración del algoritmo Fuzzy en lenguaje C. 93
2.4 Funciones de la librería 96
2.5 Ejemplo de aplicación 97
3. LIBRERÍA DE DINÁMICA DE REPLICADORES 98
6. Bases Conceptuales 98
3.2 Entradas y salidas del algoritmo 99
3.3 Implementación del algoritmo en lenguaje C 100
3.4 Funciones de la librería 101
IV. Discusión y Conclusiones 103
V. Referencias 105
VI. ANEXOS 106

Índice de figuras

		Pág.
Figura 1	Librería Arduino IO para Matlab y Simulink	17
Figura 2	Librería Lego Mindstorms para Matlab y Simulink	17
Figura 3	Algoritmo del Class de la librería TESLAMAT	19
Figura 4	Proceso para llevar un dato desde el dispositivo a bloques de Simulink	24
Figura 5	Editor de Máscaras y Algoritmo para generar la máscara del bloque de Android	25
Figura 6	Interfaz de usuario para configuración de bloque.	25
Figura 7	Adición al library browser de Simulink	27
Figura 8	Librería Teslasim de Simulimk para plataforma Tesla	28
Figura 9	Estructura de la aplicación Android.	31
Figura 10	Modificación de un Slider con código de rogramación	32
Figura 11	Interfaz de inicio de la aplicación.	33
Figura 12	Interfaz de Puente	33
Figura 13	Opción de escalado y ampliación de la señal.	34
Figura 14	Opción de cambio de color.	35
Figura 15	Adquisición en una Tablet Samsung de 7 Pulgadas.	35
Figura 16	Control PS3 para robótica.	36
Figura 17	Consola Serial	37
Figura 18	Aplicación Android Tesladroid. La imagen ilustrada muestra las 6 aplicaciones: la imagen 1 es el demo tesla., la 2 es la interfaz robótica, la 3 es la Interfaz para Matlab, la 4 es el osciloscopio digital, la 5 es una aplicación para agregar otras aplicaciones y la 6 es una consola serial.	39
Figura 19	Entornos creados con el toolbox Simulink 3d Animation.	40
Figura 20	Interfaz de Matlab para la aplicación Tesladroid	41

Figura 21	Método para simulaciones virtuales usando el dispositivo Tesla	41
Figura 22	Algoritmo en diagrama de bloques para lectura de tramas de la Interfaz Android para Matlab	42
Figura 23	Implementación del método de simulación virtual, usando la librería Teslasim y el toobox de Simulink 3D Animación.	43
Figura 24	Resultado de la simulación, Control sobre un eje.	44
Figura 25	Estructura de Teslasoft	47
Figura 26	Proceso de elaboración de Teslasoft	49
Figura 27	Programas para la compilación de Teslasoft	50
Figura 28	Generación del archivo .hex a partir de librerías y el compilador	51
Figura 29	Demostración de la librería de segundo nivel Tesla y código nativo en Teslasoft	52
Figura 30	Programa Teslasoft. La imagen muestra el programa compilando un sencillo código.	54
Figura 31	Librerías, ejemplos y opciones del programa. La imagen indica la estructura de opciones que ofrece al desarrollador y las librerías de control principales.	55
Figura 32	Arquitecturas de red para la Tarjeta	59
Figura 33	Transmisión de datos vía internet	60
Figura 34	Diagrama de bloques del sistema electrónico del dispositivo Tesla	62
Figura 35	Funciones de la plataforma Tesla y acciones de software y hardware para ejecutar cada función.	64
Figura 36	Interfaz entre Processing y el microcontrolador de desarrollo de la plataforma Tesla	66
Figura 37	Estructura general de las entradas analógicas para la plataforma Tesla	67
Figura 38	Circuito de protección de pines analógicos de los microcontroladores	68
Figura 39	Voltaje de salida ante la variación del voltaje de entrada del circuito de protección	68

Figura 40	Circuito de protección y multiplexado para pines analógicos	69
Figura 41	Circuito de protección y variación de amplitud de salidas PWM	70
Figura 42	Conversión TTL para entradas y salidas digitales	70
Figura 43	Conexión entre el LCD táctil y el microcontrolador máster	71
Figura 44	Conjunto de pasos para programar la pantalla táctil	71
Figura 45	Interfaz de la pantalla táctil	72
Figura 46	Circuito de potencia para accionamiento de cargas AC	74
Figura 47	Disposición modular de la tarjeta Tesla	75
Figura 48	Implementación de la capa superior	76
Figura 49	Implementación de la capa media	76
Figura 50	Implementación de la capa inferior	77
Figura 51	Estructura del dispositivo Tesla	78
Figura 52	Control PID aplicado a una planta.	79
Figura 53	Entradas y Salidas del Algoritmo	80
Figura 54	Estructura del control con la tarjeta Tesla	80
Figura 55	Integral real e integral aproximada, áreas bajo la curva.	81
Figura 56	Efecto Windup en un control PID	83
Figura 57	Efecto del cambio de sintonización en un control PID	84
Figura 58	Algoritmo en diagrama de flujo implementado para el programa Teslasoft	85
Figura 59	Sistema de control Fuzzy con realimentación	87
Figura 60	Proceso de Fuzzificación.	88
Figura 61	Funciones de Pertenencia para las variables: humedad y temperatura.	89
Figura 62	Evaluación de reglas, segunda fase del proceso de un control Fuzzy	90
Figura 63	Matriz de reglas de Inferencia	90

Figura 64 Funciones de pertenencia para Defuzzificación. 92

Figura 65 Proceso de Defuzzificación. 92

Figura 66 Entradas y Salidas del Algoritmo 93

Figura 67 Construcción de funciones de membresía a partir de vectores. 94

Figura 68 Proceso de configuración y ejecución del algoritmo 96

Figura 69 Ejemplo de aplicación de la librería Fuzzy 97

Figura 70 Entradas y salidas del algoritmo por dinámica de replicadores 100

Figura 71 Estructura del controlador y la planta 100

Índice de Tablas

		Pág.
Tabla 1	funciones de usuario de la librería Teslamat	20
Tabla 2	Tramas De Datos Enviadas Desde Teslamat	22
Tabla 3	Archivos con S-FUNCIONES de segundo nivel de MATLAB	27
Tabla 4	Código para leer los objetos de la interfaz robótica	36
Tabla 5	Tramas de bytes para leer la interfaz de android para Matlab	42
Tabla 6	Algunas sentencias de lenguaje nativo para TESLASOFT	54
Tabla 7	Programas multimedia instalados internamente	59
Tabla 8	Funciones y sentencias de la librería PID	86
Tabla 9	Código PID implementado con la plataforma Tesla y el uso de la Libreria	87
Tabla 10	Reglas y grado de verdad	92
Tabla 11	Creación de funciones de membresía	95
Tabla 12	Funciones y sentencias de la librería para dinámica de replicadores	97
Tabla 13	Funciones y sentencias de la librería para dinámica de replicadores	102

I. INTRODUCCIÓN

La electrónica, junto con el hardware y software libre están evolucionando muy rápidamente debido a que en un corto periodo de tiempo se ha pasado de la programación a bajo nivel de microcontroladores a programación de grandes procesadores con ciclos de reloj de hasta 1GHz. Personas sin formación en sistemas digitales pueden tener acceso a la información y poder programar estas complejas arquitecturas gracias a la existencia de fuentes libres de información que se conoce como "Open Source" [1]. Todo este desarrollo especialmente en tarjetas electrónicas se ha encaminado en el desarrollo de proyectos sencillos, de bajo costo y corto tiempo de programación; pero no ha tenido un enfoque profesional que permita utilizar toda la capacidad de procesamiento y cómputo en áreas específicas como sistemas de control, adquisición robusta y desarrollo a un alto nivel de programación. Otra característica de las tarjetas de desarrollo actuales es que el programador en la mayoría de los casos solo puede utilizarla para una función específica, es decir, si se utiliza para adquisición simple no se puede usar para programación y viceversa. Además, ningún desarrollo de hardware libre permite simulación virtual en unión con Simulink.

En cuanto al costo de sistemas de adquisición especializados, es bastante elevado, por lo que no son asequibles a estudiantes o como equipo de laboratorio experimental en materias básicas. Por ejemplo, las tarjetas de National Instruments NI PCIe-6343 y NI PCIe-6341 para usar con Matlab y Simulink pueden alcanzar los $5.000.000 [2]. Muchos de estos sistemas se enfocan en adquisiciones con conversores analógico-digital de gran cantidad de bits, olvidando que la tendencia de los sensores en la actualidad es mudarse de una salida análoga a una salida digital con protocolos como SPI, I2C, Serial y LAN; tal como lo están haciendo grandes empresas como Sparkfun y Adafruit industries. Por otra parte, la adquisición actual es puramente alámbrica, es decir que el sensor tiene que tener una conexión física por cable, por lo que las redes de sensores inalámbricos tienen serias dificultades de acople con este tipo de tarjetas.

El anterior planteamiento permite sentar las bases de la necesidad de construir una plataforma lo suficientemente robusta, flexible y con un claro enfoque en sistemas de control; constituida como una multiplataforma que permite al usuario realizar adquisición normal e inalámbrica, programación, simulación, permitir la función de un minicomputador, uso de redes, protocolos digitales y tener un enfoque en estrategias de control. De esta manera, el objetivo central de esta investigación es apoyar los desarrollos del grupo de investigación GIIEE de la universidad de

Nariño, especialmente en el área de sistemas de control, sistemas digitales, comunicaciones y otras áreas dentro del amplio espectro de posibilidades y usos que puede tener la tarjeta.

La estructura de hardware está conformada por un procesador Vivante con frecuencia de reloj de 1GHz, brindado por la tarjeta Udoo; dos CPU ATSAM3X8E con la función de dos microcontroladores con frecuencia de reloj a 84 MHz; módulos de comunicación bluetooth: wi-fi y xbee; pantalla lcd TFT táctil; un circuito de adaptación de señales de entrada y de protección y un circuito de adaptación de señales de salida. En cuanto a software posee: un programa denominado Teslasoft con un lenguaje nativo de programación creado en la Universidad de Nariño para generar algoritmos de programación, librerías para usarse con Matlab, Simulink, Processing y Java, una aplicación Android para interactuar con dispositivos móviles, un método para simulación virtual usando el toolbox de "Simulink 3d Simulation" y Android, un programa en Java para enviar y recibir datos por internet y un sistema operativo adaptado a las características del dispositivo.

De acuerdo a las consultas realizadas en bibliografía actualizada, bases de datos de trabajos de grado de las principales universidades del país y en las principales páginas de internet especializadas en este tipo de dispositivos, aún no se presenta un dispositivo comercial con estas características. A esta plataforma se le ha denominado "Tesla" en reconocimiento al gran aporte a la ciencia brindado por Nikola Tesla, término que se seguirá usando durante el desarrollo de este documento.

Este trabajo está organizado así: en la Sección I y correspondientes subsecciones se presenta el diseño del software, en la Sección II se explica la arquitectura electrónica y en la sección III se describe el diseño de las librerías de control.

I. DISEÑO DE SOFTWARE

1. DISEÑO DE LIBRERÍA DE MATLAB y SIMULINK

1.1 Bases preliminares

El equipo de Mathworks, quien diseña y crea toolbox y librerías para soportar a múltiples dispositivos en Matlab, ha creado soporte para arduino en todas sus gamas y versiones. La primera librería que lanzó para arduino se denominó ARDUINO IO [3], mostrada en la Figura 1, que presenta excelente velocidad de transmisión. Sin embargo, esta librería usa la tarjeta cargándole un programa esclavo que debe estarse ejecutando constantemente, por lo que queda útil solo para adquisición y para ninguna otra función como la programación. Posteriormente, se lanzó soporte para arduino uno y

mega, para los que se podía correr en modo "External" dentro de Simulink, y Matlab compilaba y cargaba a la tarjeta un determinado código que fuese compatible con el diagrama de bloques que diseñaba el programador. El problema actual que presenta esta librería es la carencia de velocidad para usarse en sistemas de control y en plantas que requieren tiempos de muestreo (envío de datos de los sensores y retorno de la señal del actuador) bastante cortos.

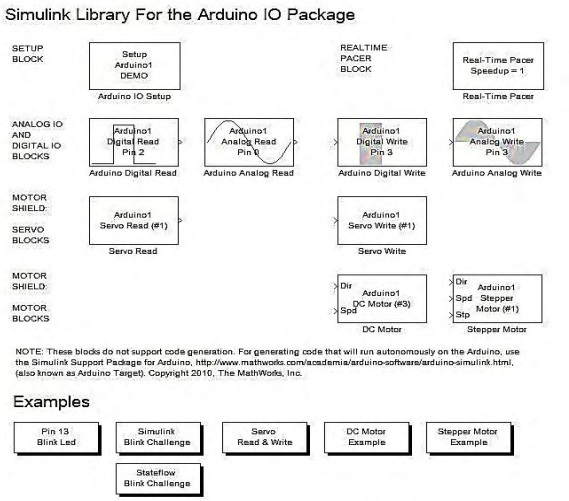


Figura 1. Librería Arduino IO para Matlab y Simulink

Las máscaras que se utilizan para plataformas no son llamativas, pero hay otras librerías como las de Lego Mindstorms que se muestra en la Figura 2, que poseen diagramas de bloques llamativos que impulsan el fácil entendimiento de los bloques y configuración.

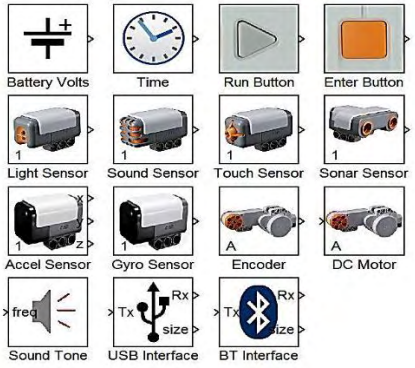


Figura 2. Librería Lego Mindstorms para Matlab y Simulink

1.2 Parámetros de diseño

Los parámetros esenciales para la construcción son:

- La librería debe funcionar a una buena velocidad para permitir el enfoque en control que se requiere.
- Las máscaras en Simulink deben ser muy simples y con un diseño estético llamativo para el estudiante.
- Tanto la librería de Matlab como la de Simulink deben instalarse en un solo paquete.
- La librería debe ser flexible para aceptar comunicación con Android y dispositivos móviles.
- Se debe permitir la adquisición por medio de protocolos digitales como I2C o SPI.
- Se debe poder intercomunicar los módulos xbee directamente con Matlab.
- Se debe facilitar la interfaz con entradas analógicas y digitales, así como el control de salidas digitales, PWM y analógicas.

Para efectos de diseño, a la librería de Matlab se la denominó TESLAMAT y a la librería de Simulink TESLASIM.

1.3 Construcción de librería de Teslamat

Según la revisión bibliográfica realizada, no se encontraron documentos que indiquen el modo de crear una librería para un dispositivo de adquisición de datos. El tema es muy hermético y se mantienen las características de diseño dentro del grupo de desarrollo de las aplicaciones particulares y de las interfaces con Matlab. La mínima información que se encuentra en internet es para la construcción de librerías a partir de otras librerías ya creadas por el grupo de Mathworks, por tal razón, se realizó un estudio a las librerías existentes como: Arduino IO y lego mindstorms, para extraer el proceso de construcción general de una librería y a partir de este conocimiento crear una librería con aspectos mejorados y que se adapte a las características de hardware de la tarjeta Tesla. Se determinó que la librería internamente posee un archivo con extensión .m con una función classdef que define una clase [4]. Esta clase, mediante otro archivo, necesita ser guardada en la lista de rutas de Matlab, que se conoce como "path" para que cualquier usuario en cualquier momento puede usar las funciones que internamente se encuentran en la clase.

1.4 Archivos principales de la librería

Los archivos principales son:

1.4.1 Clase Principal

Este archivo permite contener las funciones principales, crear funciones dinámicas, funciones estáticas y parámetros de configuración. Se define a partir de la función "classdef" y la estructura

se indica en la Figura 3. En la sección "methods" se incluyen todas las funciones principales para escritura y lectura de datos, funciones que el usuario utiliza en un script de Matlab o en la ventana de comandos. Las funciones auxiliares incluidas en la sección "methods (static)" brindan un apoyo de cómputo y corrección para las funciones principales y en la etapa inicial se configura y asigna propiedades. Las funciones de Matlab nativas que utilizan las funciones principales son:

- Serial(COMX, 'BaudRate', Velocidad): función para configuración del puerto serial y la comunicación con la plataforma Tesla
- fopen(puerto): abre el puerto serial para iniciar comunicación
- fwrite(valor): envía un valor por el puerto serial
- fscanf(): lee un valor enviado por la tarjeta
- fclose(puerto): cierra el puerto serial

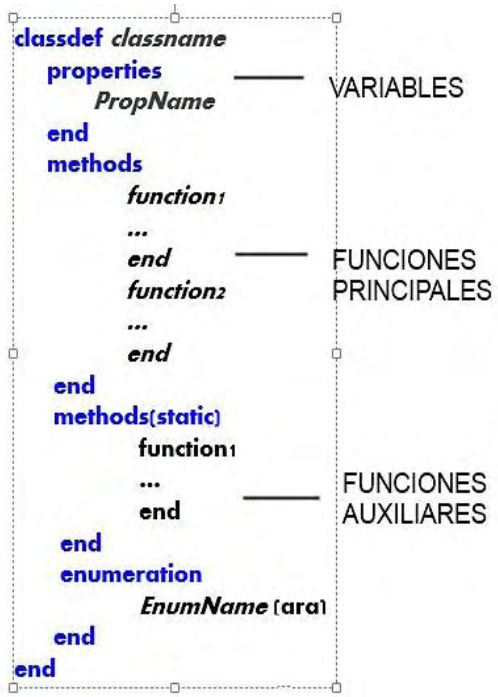


Figura 3. Algoritmo del Class de la librería TESLAMAT

1.4.2 Archivo de adición a la ruta de direcciones:

Es un archivo con extensión .m que permite adicionar la librería al "path" de Matlab para que este programa pueda identificar la dirección del archivo y pueda cargarlo cada vez que se abra Matlab y se requiera usar la librería.

1.5 Funciones de usuario de la librería

El usuario podrá utilizar las funciones indicadas en la TABLA I para acceder a las funcionalidades de la tarjeta desde cualquier script de Matlab. Estas funciones son codificadas, muy sencillas de entender y aplicar (inclusive diseños avanzados con GUIDE por ejemplo, podrán llevarse a cabo usando estas funciones). Una vez añadido la clase principal a la ruta de archivos o "path", las funciones de la clase añadida se transforman en sentencias nativas de Matlab como las funciones "plot(x)", "clear all", "clc", "size(A)", etc.

Tabla 1. Funciones de usuario de la Librería Teslamat

Número	Sentencia	Funcionamiento
1	t=TESLA('COMx')	Crea un objeto TESLA en la variable t y le asigna el puerto serial COMx
2	t.EscrituraI2C(a,b)	Escribe mediante el protocolo I2C en un dispositivo conectado al tesla mediante pines I2C con una dirección a, el valor de b, a es necesariamente una dirección de 8 bits es decir valores de 0-255, y b puede ser cualquier valor.
3	val=t.LecturaI2C(a,b,c)	Realiza una lectura en un dispositivo conectado por I2C al tesla que tenga la dirección a, b es un parámetro de configuración necesario en algunos casos para configurar la lectura y c es la cantidad de bytes que se desea leer, esta lectura la asigna a la variable val.
4	val=t.lecturaDigital(a)	Ejecuta una lectura de un pin digital número a y asigna el valor a la variable val.
5	t.EscrituraDigital(a,b)	Permite escribir en un pin digital a, el valor digital de b, que es un valor binario de 0 o 1.
6	val=t.LecturaAnalogica(a)	Realiza una lectura analógica del pin a, y guarda el valor en la variable val.
7	t.EscrituraAnalogica(a,b)	Escribe un valor analógico b, en un pin a.
8	t.EscrituraPWM(a,b)	Escribe un valor PWM a en un pin PWM b
9	t.EscrituraBluetooth()	Ejecuta una escritura en el módulo bluetooth interno del dispositivo tesla.
10	val=t.LecturaBluetooth()	Asigna en la variable val un valor obtenido del módulo bluetooth
11	t.EscrituraRF(val)	Realiza el envío de datos por medio del módulo Xbee a otro módulo emparejado con el xbee interno del tesla.

12	val=t.LecturaRF()	Asigna a la variable val un valor leído del módulo Xbee
13	val=t.Android(a)	Lee valores enviados desde dispositivos móviles, el valor a indica el widget (slider, botón, entre otros) que se desea leer dentro de la aplicación Android.

1.6 Proceso para comunicación entre tarjeta Tesla y librería Teslamat

La comunicación se establece de forma bidireccional usando el puerto serial. Inicialmente, se debe colocar la tarjeta Tesla en modo adquisición, la librería Teslamat (2) envía una trama de dos bytes enteros {12,7}, a la cual el dispositivo Tesla debe contestar con el byte entero 16 para iniciar el emparejamiento y la transmisión de datos. Cuando Matlab y el dispositivo tesla están emparejados, la librería Teslamat puede usar cualquier función de la clase principal establecidas en la TABLA II para leer y/o para escribir datos, el flujo de datos es controlado principalmente desde esta librería, por ejemplo, para realizar una lectura analógica se envía la trama {1,40+pin} conformada por 2 números enteros, el 1 corresponde a la función de lectura analógica y el byte de 40+pin corresponde al pin con un valor adicional para evitar errores en la lectura desde el dispositivo Tesla, cuando recibe esta cadena la tarjeta, esta le responde con el valor analógico del pin correspondiente. La TABLA II indica las tramas de bytes que se envían desde Matlab y la respuesta desde la tarjeta tesla para una determinada acción que se requiere realizar. La trama de datos puede estar conformada por números enteros de 8 bits hasta de 12 bits, el primer byte siempre es el modo y los siguientes pueden ser pines, valores, direcciones y datos de configuración.

Tabla 2. Tramas De Datos Enviadas Desde Teslamat

Función	Trama de datos desde Matlab	Respuesta o acción a tomar en la tarjeta Tesla
Lectura analógica	[1, 40+pin]	Devolución de un valor que varía de (0-4096) generado a partir del modo establecido en el primer dato y el pin generado de la resta entre el segundo dato y 40.
Escritura analógica	[2,valor,80+pin]	Escritura analógica con valores de (0-4096) que corresponden al segundo dato (valor), sobre el pin, producto de restar 80 al tercer dato enviado desde Matlab
Escritura PWM	[3,120+pin, valor]	Escritura PWM con valores de (0-4096) en un pin PWM|
Lectura Digital	[4,160+pin]	Devolución de un valor booleano leído de un pin digital deducido del segundo dato proveniente de Matlab

Escritura Digital	[5,200+pin,valor]	Escritura booleana establecida en un pin digital deducido del segundo dato y con el valor leído del tercer dato.
Lectura I2C	[6, dirección, configuración, tamaño búfer]	Devolución de un valor de 12 bits, producto de la lectura de un dispositivo o sensor conectado en los pines I2C y que posea la dirección de 7 bits establecida en la lectura del segundo dato.
Escritura I2C	[7, dirección, valor]	Escritura I2C en algún dispositivo conectado con la tarjeta tesla que posea la dirección del segundo dato.
Lectura Bluetooth	[8]	Devolución de un valor de 8 bits proveniente del módulo bluetooth conectado a la tarjeta Tesla
Escritura Bluetooth	[9,valor]	Escritura de un valor deducido del segundo dato en el módulo bluetooth
Lectura Xbee	[10]	Devolución de un valor de 12 bits proveniente del módulo Xbee
Escritura Xbee	[11,valor]	Escritura de un valor deducido del segundo dato en el módulo Xbee de la tarjeta.
Conexión inicial con Matlab	[12,7]	Devolución del valor 16 para establecer que hay comunicación bidireccional entre Matlab y el dispositivo Tesla, para así iniciar el proceso de envio y recepción de datos en ambas partes.
Conexión Android	[13,widget]	Lee valores provenientes de dispositivos móviles que posean la aplicación Tesladroid.

El algoritmo programado en la tarjeta Tesla posee la misma estructura funcional que el presentado en la Figura 26, solo que en este caso no se almacenan valores si no se ejecutan acciones específicas. Para ello se estableció un algoritmo de un estado de máquinas con el cual constantemente se está leyendo el puerto serial, en caso que se reciba un modo (valores de 0-13) se realiza un cambio de estado y se empieza a leer dato por dato de cada trama enviada desde matlab, una vez leída la trama se vuelve a un estado base y se empieza a esperar un siguiente modo. El proceso debe correr a máxima velocidad, por lo que se optimiza el código y se realiza todo el proceso en una función “switch case ()” que es más eficiente que un conjunto de “if” anidados o cualquier otra sentencia en lenguaje C. El algoritmo final abarca 900 líneas de código y se puede visualizar en los anexos.

1.7 Consideraciones finales

El archivo para la adición a la ruta de archivos y el de la clase principal se adjuntan en los anexos. Una vez instalada la librería, no se debe mover ninguno de los dos archivos de la dirección en que inicialmente se los colocó, ya que la dirección para que Matlab los encuentre es diferente. Por tal razón se debe colocar en la carpeta de trabajo de Matlab, que es la dirección por defecto para cargar nuevas librerías.

1.8 Construcción de la librería Teslasim para Simulink

Con la elaboración de la clase principal "class" para la librería de Matlab Teslamat, es menos complejo la construcción de la librería Teslasim de Simulink, ya que estas funciones se pueden asociar a bloques de Simulink mediante la utilización de funciones de 2do nivel de Matlab, las cuales son sentencias usadas para crear bloques de Simulink personalizados, restringir variables de entrada y salida, crear puertos y controlar el flujo de señales [5]. El proceso de controlar la tarjeta y de realizar adquisición funciona de la siguiente manera:

El flujo de datos puede ocurrir de 4 a 1 o de 1 a 4, ya que entre la librería Teslamat y el dispositivo Tesla se puede recibir o enviar datos. Para enviar una determinada acción desde Simulink o realizar una adquisición de datos se debe seguir un proceso ordenado, el cual es mostrado en la Figura 4. Cuando se corre una simulación, esta recorre bloque a bloque ejecutando cada función de segundo nivel que le corresponde a cada bloque, esta función de segundo nivel llama a funciones de la librería Teslamat y esta transmite un conjunto de tramas de datos, ya sea para ejecutar una acción (como escritura PWM, escritura I2C,entre otras) o para realizar adquisición de datos (el proceso que se lleva entre Teslamat y Tesla se explicó en anteriores sub-secciones) una vez leída la trama con el dispositivo tesla (haciendo uso de un estado de máquinas), se ejecuta una acción o se envía un dato de vuelta, en el caso de que se envíe un dato de vuelta, este es recibido por la librería Teslamat y transmitido a la función de segundo nivel, la cual entrega por el puerto que construyó el dato.

Cabe destacar, que Simulink ejecuta cada bloque como si fuese un bucle de programación con un periodo regular, en cada periodo recorre todos los bloques que correspondan a la simulación y cada bloque tiene asociada una función de 2do nivel, la cual enlaza las funciones de la clase principal de Teslamat y permite realizar al bloque adquisición o controlar un determinado pin. Para cada bloque hay que diseñar las máscaras por medio del editor de máscaras que ofrece Matlab. Cuando ya se construye la librería gráfica y se le asigna a cada bloque una función de segundo nivel (función S de 2do nivel) se guarda el archivo con extensión ".mdl" y se debe otro archivo que

permita que la librería creada se agregue al buscador de librerías (en inglés Library Browser) que posee Matlab.

Figura 4. Proceso para llevar un dato desde el dispositivo a bloques de Simulink

1.9 Archivo .mdl y construcción de máscaras

El archivo .mdl define la parte gráfica de la librería, en donde cada bloque debe estar asociado con una S-función de 2do nivel de Matlab. La parte de la máscara se crea con el editor de máscaras y con un conjunto de funciones: de graficado (como patch(), color()), de adición de texto(como text(), dpoly(), fprintf()) y funciones matemáticas como coseno, seno, etc. Por ejemplo, la Figura 5 indica la forma de crear la imagen más conocida de Android y toda la parte visual del bloque que realiza una interfaz con dispositivos móviles. El código que se encuentra en el interior de la imagen es el necesario para formar la imagen y la máscara que se observa a la derecha. El editor de máscaras ofrece bastantes opciones, entre ellas: Parámetros, Inicialización y Documentación; estas opciones permiten establecer las opciones que presentará cada bloque al usuario, por ejemplo la Figura 6 muestra las opciones de usuario, para generar estas opciones se debe manipular la parte de parámetros, inicialización y documentación; para manipular estas opciones se puede consultar en [6].

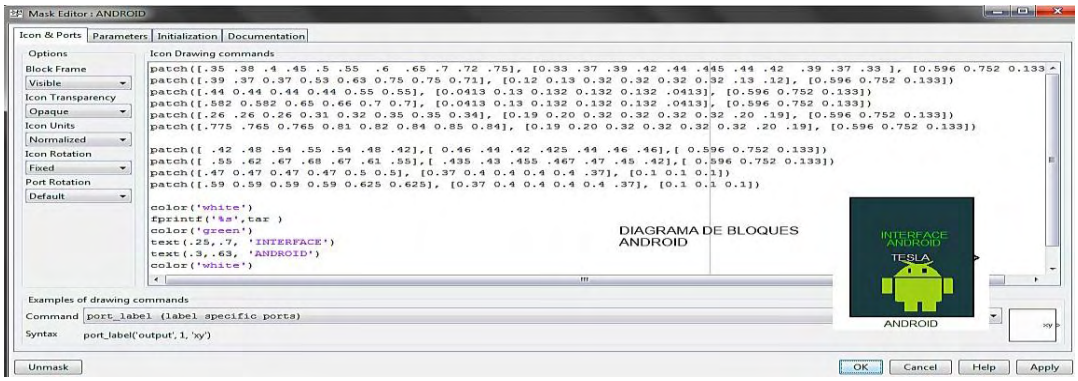


Figura 5. Editor de Máscaras y Algoritmo para generar la máscara del bloque de Android

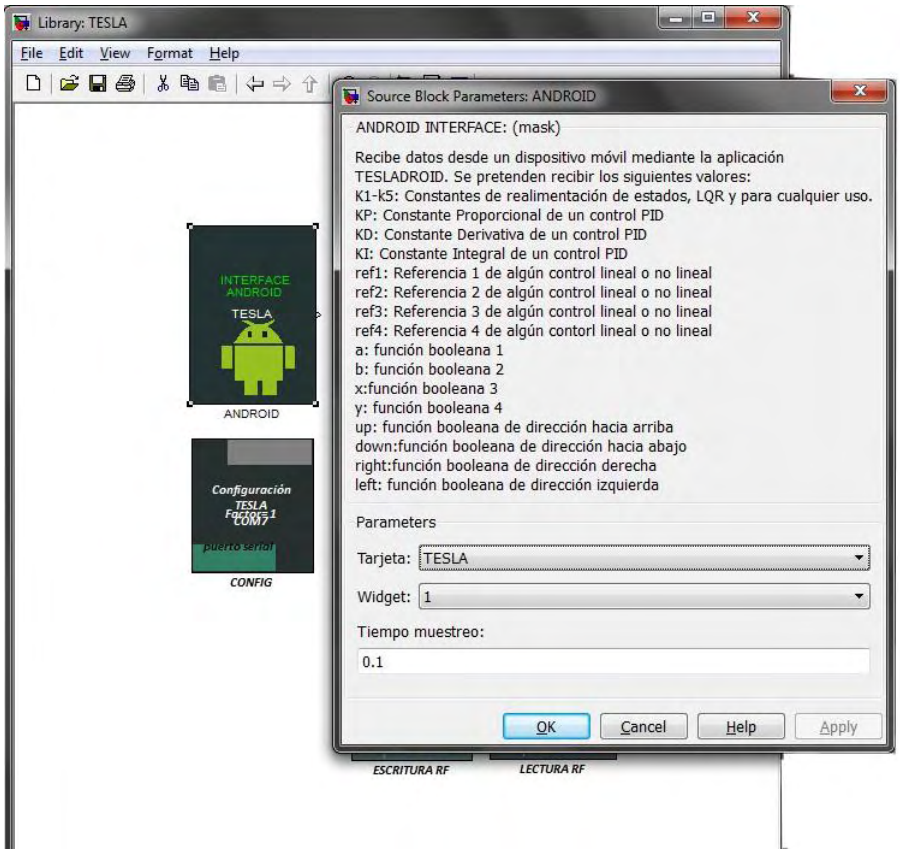


Figura 6. Interfaz de usuario para configuración de bloque.

1.10 Funciones de 2do Nivel de Matlab para las máscaras de los bloques de Simulink

Las funciones S de Matlab permiten crear bloques en Simulink personalizados con múltiples puertos de entrada y salida, capaces de manejar cualquier tipo de señal producida por un modelo de

Simulink. Una S-función de Nivel-2 es la función de Matlab que define las propiedades y el comportamiento de una instancia de un bloque. Son las funciones que unen una salida o entrada de datos de un bloque con una sencilla función de Matlab o una función que pertenece a una clase. El motor de Matlab para ejecutar un modelo recorre finitamente cada bloque ejecutando un archivo con una S-función de 2do nivel, que ejecuta una función cualquiera de Matlab para devolver finalmente un valor y presentarlo al usuario, transmitirlo a un bloque siguiente o realizar cualquier operación que Simulink requiera. La TABLA III contiene en la segunda columna el conjunto de funciones de 2do nivel que posee cada bloque de Simulink, en la tercera columna las sentencias o funciones de la clase principal que corresponden a cada función de 2do nivel, en la cuarta y quinta columna se indican los puertos creados a partir de estas funciones de segundo nivel. Cabe recordar que a cada bloque, mostrado en la Figura 8, que posee una cantidad de puertos de salida y/o entrada le corresponde una función de segundo nivel, que define los puertos y además asocia una función de Teslamat para ejecutar la acción adecuada (como lectura digital, analógica, etc).

Tabla 3. Archivos con S-FUNCIONES de segundo nivel de MATLAB

Número	Nombre de archivos con S-funciones de 2do nivel	Sentencias de la clase principal que le corresponde	Numero de Puertos de Entrada Generados	Número de Puertos de Salida Generados
1	InterfazAndroid	t.Android(a)	0	1
2	FMS_EscrituraAnaloga	EscrituraAnalogica(a,b)	1	0
3	FMS_LecturaRF	t.LecturaRF()	0	1
4	FMS_LecturaI2C	t.LecturaI2C(a,b,c)	0	1
5	FMS_LecturaDigital	t.lecturaDigital(a)	0	1
6	FMS_LecturaBluetooth	t.LecturaBluetooth()	0	1
7	FMS_LecturaAnalogica	t.LecturaAnalogica(a)	0	1
8	FMS_EscrituraRF	t.EscrituraRF()	1	0
9	FMS_EscrituraI2C	t.EscrituraI2C(a,b)	1	0

10	FMS_EscrituraDigital	t.EscrituraDigital(a,b)	1	0
11	FMS_EscrituraBluetooth	t.EscrituraBluetooth()	1	0
12	FMS_EscrituraPWM	t.EscrituraPWM(a,b)	1	0
13	FMS_Configuracion		0	0

1.11 Archivos Adicionales

Para adicionar la librería de Teslasim al archivo que contiene el conjunto de direcciones de los programas asociados a Matlab, "path", se necesita utilizar el mismo archivo de adición que la librería Teslamat descrito anteriormente. Para permitir que la librería Teslasim, mostrada en la Figura 8, se adicione al buscador de librerías o "library browser", tal como se indica en la Figura 7, se debe ejecutar un archivo que contenga la función principal "blkStruct()", este archivo posee extensión .m y se encarga de colocar el nombre dentro de la lista de librerías de Simulink y adicionar los bloques en el buscador de librerías de Simulink.

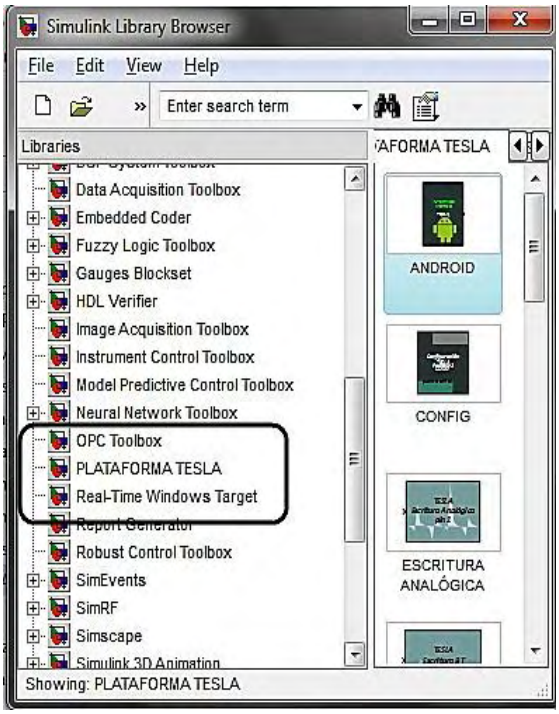


Figura 7. Adición al library browser de Simulink

1.12 Consideraciones finales

Una vez ejecutado el archivo de instalación ya se puede hacer uso de las dos librerías Teslamat y Teslasim, y esta última tendrá la imagen de la Figura 8. El bloque de CONFIG permite ejecutar los bloques de la librería Teslasim en tiempo real a partir de la modificación de un parámetro P por parte del usuario, cuando el parámetro p es igual a 1, se ejecutan los bloques en tiempo real, cuando p es igual a inf se ejecuta al máximo tiempo que proporcione la simulación. Este bloque no afecta en ningún caso la simulación del sistema, debido a que solo modifica el tiempo de muestreo que posee cada bloque de la librería Teslasim.

El objetivo final de la librería de Simulink, es conseguir una simulación o ejecución de algoritmos de forma rápida, con alta trasmisión de datos desde la plataforma, con alta velocidad de ejecución en Simulink y con formas llamativas e intuitivas de los bloques, muy similares a la librería de Lego Mindstorms mostrada en la Figura 2. Según las consultas realizadas, este desarrollo es por primera vez conseguido en una Universidad de Latinoamérica y convierte a la Universidad de Nariño, en pionera en desarrollos de este tipo de librerías para plataformas de software y hardware libre y que adicionalmente tenga compatibilidad con Arduino Due. Mathworks en la necesidad de sacar una librería lanzó el 6 de marzo de 2014 una librería para Arduino Due [7], que hace carga automática de código en la tarjeta con muchos errores de compilación y de velocidad, problemas evitados con el uso de la librería Teslasim.

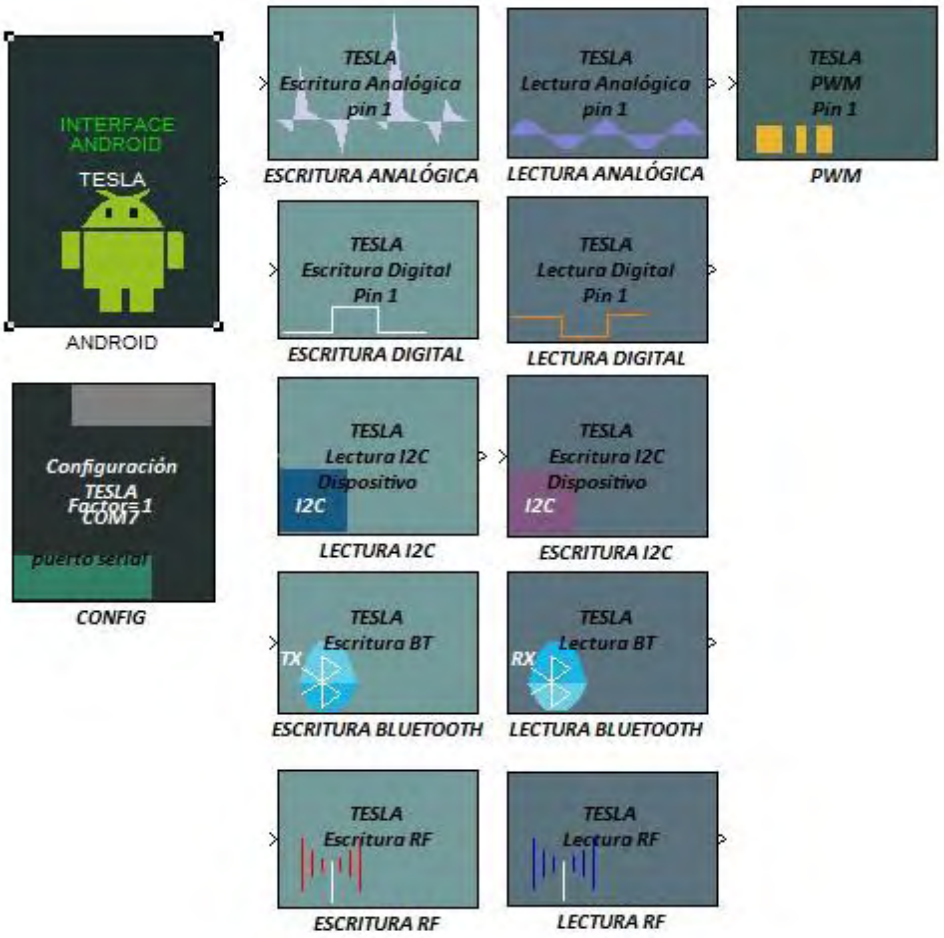


Figura 8. Librería Teslasim de Simulimk para plataforma Tesla

Las novedades que presenta esta librería son:

- Permite la adquisición normal: entradas digitales y analógicas y permiten el control digital con salidas digitales y PWM, tal como lo hacen otras librerías comunes.
- Permite la Adquisición inalámbrica a través de módulos Xbee
- Admite la adquisición de sensores con protocolo digital I2C
- Admite la conexión de otras tarjetas, alrededor de 255 a Simulink mediante el bloque de Lectura I2C de la librería Teslasim
- Presenta salida Analógica a una mínima resolución equivalente a 0.805 mV aproximadamente y valores de 0-4096 arrojados por el conversor DAC de 12 bits resolución.

- Presenta compatibilidad con dispositivos móviles mediante el bloque de Android
- Admite simulación virtual con el uso del bloque de Android
- Permite adquisición de datos Bluetooth.

2. DISEÑO Y CONSTRUCCION DE APLICACIÓN ANDROID

2.1 Bases Preliminares.

El diseño se realizó en el programa ECLIPSE ADT Bundle [8], que es un paquete de instalación que incluye Eclipse, el ADT de Android y el kit de desarrollo de software (SDK), que inmediatamente instalado, puede ser utilizado para desarrollar aplicaciones móviles. La aplicación construida se la llamó Tesladroid. Para realizar una breve introducción al tema, se necesitan conocer algunos términos como:

- SDK: Es el kit de desarrollo de software o conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones para Android. Contiene todo lo necesario para desarrollar aplicaciones.

- ADT: Es un "plugin" para el IDE Eclipse que está diseñado para potenciar el entorno de Eclipse y facilitar la construcción e integración en el desarrollo de aplicaciones de Android. Utiliza las herramientas del SDK y las adapta para usarse con Eclipse.

- Actividad: es una tarea realizada por una aplicación y también se podrían llamar ventanas. Android solo ejecuta una actividad a la vez. En otras palabras, una actividad es el algoritmo que realiza una tarea determinada.

- Layout: Son la parte visual de una aplicación, por lo general, por cada actividad hay un "layout". Son la estructura gráfica que permite la interacción con el usuario. Se estructuran en forma de árbol para organizar cada elemento gráfico en la aplicación.

2.2 Parámetros de Diseño.

El auge actual en cuanto a tecnología y comunicaciones son los dispositivos móviles y aplicaciones Android. Por esta razón se hizo necesario la inclusión de este tipo de desarrollo, en la plataforma Tesla, pero con un enfoque en Ingeniería Electrónica. El diseño debe ser muy flexible y debe tener en cuenta los siguientes planteamientos.

- Una Aplicación Android puede comunicarse con el exterior a través de cable, a través de la red o a través de bluetooth. En este sentido, Tesla se usa para adquirir las señales eléctricas que se pueden mostrar o manipular por medio del uso de la red o bluetooth.

- Contar con un dispositivo móvil (celular o Tablet) que pueda ser usado como un osciloscopio de mano es una gran ventaja para un estudiante de Ingeniería Electrónica, ya que podrá analizar en tiempo real el estado de señales. La medición de las señales se realizaría con la tarjeta y la visualización y procesamiento de las señales con el dispositivo móvil.

- En robótica y sistemas digitales es muy útil la incorporación de una interfaz de control para facilitar la manipulación de variables como dirección, velocidad y variables booleanas en general. Por esta razón un control similar al de una consola de juegos brinda una herramienta útil para incorporarse en este ámbito.

- Para la comprobación de paquetes de transmisión, tramas de bits y revisión de protocolos de transmisión es muy útil una consola serial que permite una rápida revisión de datos enviados.

- Matlab es la herramienta más útil para sistemas de control, así que permitir una comunicación unidireccional entre Android y Matlab, conlleva grandes ventajas a la hora de realizar simulaciones gráficas de sistemas de control.

2.3 Estructura General de la Aplicación.

La comunicación entre el dispositivo Tesla y la aplicación se lleva a cabo por medio de Bluetooth, por lo que se requiere solicitar permisos en el archivo "AndroidManifest" para el uso de este dispositivo. Se deben programar 8 actividades, como se muestra en la Figura 9, con 6 "layouts" principales y 2 auxiliares. Desde cada actividad principal se debe permitir la opción de pasar a cualquier otra actividad incluyendo la actividad 2. Por ejemplo, cuando el usuario este usando el osciloscopio, debe tener la opción de usar en cualquier momento la consola serial u otra actividad, y por tanto, eso implica que se necesitan 42 "intents" y uno adicional para pasar de la actividad 1 a la actividad 2. Teniendo en cuenta los parámetros de diseño, se incluyó el diseño de un osciloscopio, una consola serial, un control tipo videojuego para robótica, una interfaz para conexión directa con Matlab, un Demo y un espacio para agregar nuevas aplicaciones útiles.

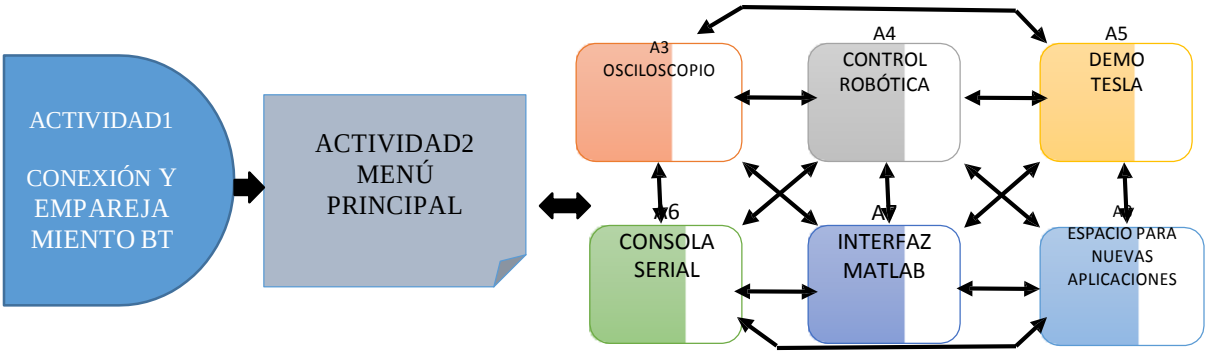


Figura 9. Estructura de la aplicación Android.

2.4 Interfaz Gráfica

La parte gráfica dentro de una aplicación es muy importante. Por tal razón se realizaron modificaciones a la paleta de "widgets" de Eclipse construyendo nuevas formas de: botones, barras de desplazamiento, sliders, entre otros. Por ejemplo, en la Figura 10 se indica el código y el objeto creado para un comando de tipo deslizamiento.

Cada forma y objeto es modificado para adecuarlo a la aplicación mediante código. Hay objetos que requieren desde 15 lineas (como el de la Figura 13) hasta objetos que requieren 1000 líneas para modificarse como sliders circulares.

Figura 10. Modificación de un Slider con código de programación

2.5 Interfaz de Inicio

La interfaz de inicio es la primera actividad que se ejecuta, en donde el usuario selecciona el dispositivo bluetooth de la tarjeta Tesla. Esta interfaz brinda la posibilidad de conectarse con un dispositivo Tesla a la vez, el nombre del bluetooth es Tesla, por tanto, se puede identificar fácilmente en la lista de dispositivos vinculados, en la izquierda de la Figura 11. Cuando se hay seleccionado el dispositivo, se presión el botón de conexión indicado en la Figura 10, si se logró emparejar el módulo de la Tablet con el módulo bluetooth de la tarjeta se puede acceder a las demás ventanas.

Figura 11. Interfaz de inicio de la aplicación.

2.6 Interfaz Puente

Es la interfaz mostrada en la Figura 12, la cual es desplegable cuando ya se garantiza la conexión bluetooth y se ha verificado que existe un dispositivo Tesla. Presenta diferentes opciones al usuario entre las que se encuentran seleccionar: una interfaz para Matlab, osciloscopio, un control para robots, botón para desconectar el módulo bluetooth, consola serial y un espacio para agregar nuevas aplicaciones.

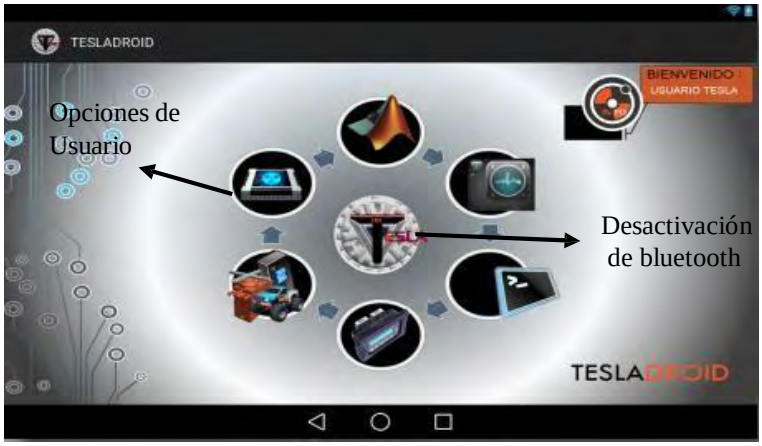


Figura 12. Interfaz de Puente

2.7 Osciloscopio

La aplicación contiene un algoritmo complejo en programación, solo la parte del osciloscopio cuenta con más de 900 líneas de código programadas. Esta interfaz ofrece un canal de visualización para señales que se adquieran con el dispositivo Tesla. Presenta características de pausa y análisis detallado de la señal, tal como se indica en la Figura 13. Para la construcción de la señal gráfica se hace uso del aporte generado por Jonas Gehring [9], con el cual se puede realizar gráficas profesionales mediante la concatenación de puntos. Las opciones que ofrece la interfaz de osciloscopio programada son las siguientes:

- Escalado y Zoom: Mediante un desplazamiento simultáneo de dedos a la izquierda y derecha se puede realizar un zoom a la señal y revisarla con más detenimiento.

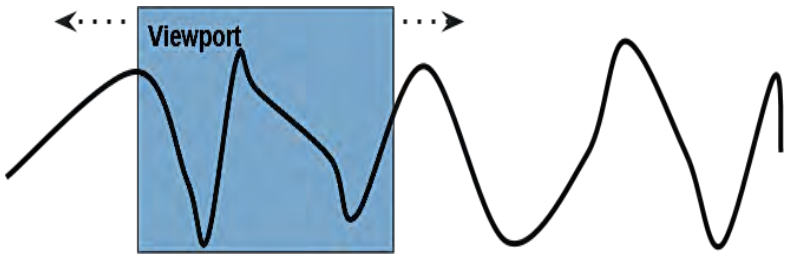


Figura 13. Opción de escalado y ampliación de la señal.

- Pausa: Se puede mediante el botón de pausa parar la adquisición y revisar la señal con detenimiento. Una vez parada la imagen se puede hacer uso de la función de escalado y zoom.

- Configuración de Color: se ofrece múltiples opciones de color para el cambio de la línea y para el cambio del fondo como se muestra en la Figura 14.

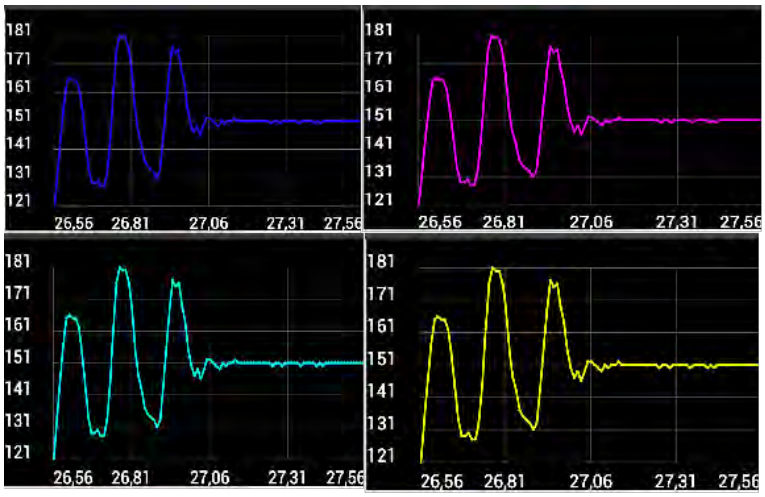


Figura 14. Opción de cambio de color.

El resultado final se indica en la Figura 15, en donde se realiza una adquisición analógica de un pin analógico del dispositivo Tesla y se muestran todas las partes y submenús.

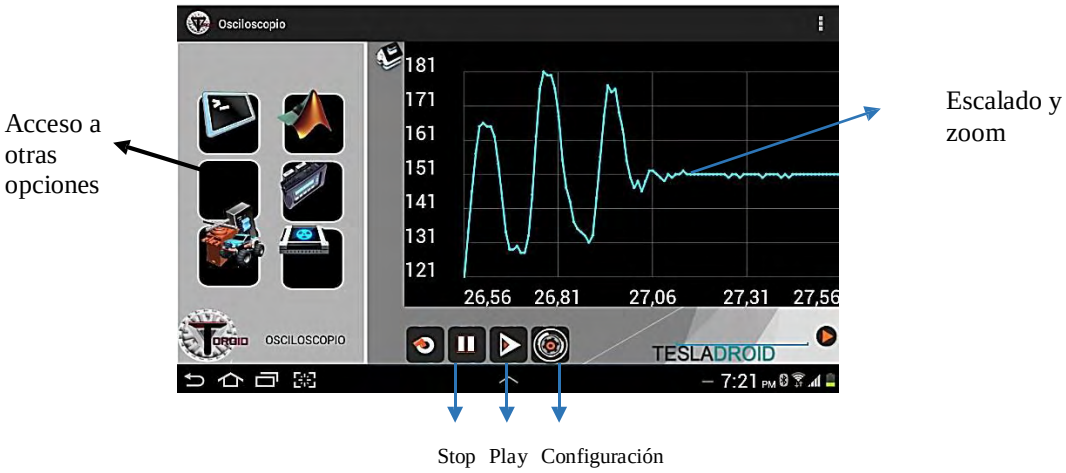


Figura 15. Adquisición en una Tablet Samsung de 7 Pulgadas.

2.8 Interfaz para robótica.

La interfaz para robótica, que se muestra en la Figura 16, está conformada por una serie de objetos que permitirán controlar dirección, velocidad, y otras variables booleanas o enteras. Cada objeto dentro de la aplicación tiene su identificación, por tal razón, cada vez que se manipule un objeto se estará enviando un byte de identificación y otro byte del valor.

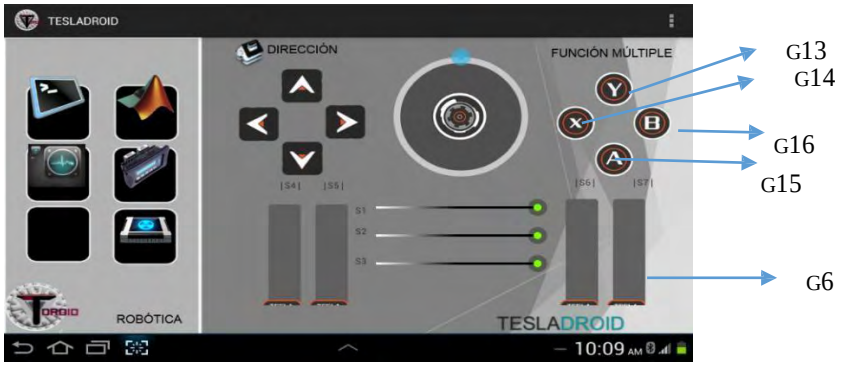


Figura 16. Control PS3 para robótica.

Por ejemplo, para leer desde el dispositivo Tesla los objetos: g13, g14, g15, g16 y g6, se necesita leer la identificación y el valor que representan. En este caso, los botones van a representar variables booleanas y el slider g6 va a representar una variable entera. Para leer estas variables desde Matlab o desde la programación interna en Tesla, es necesario realizar un algoritmo teniendo en cuenta el byte de identificación de cada objeto (presentados en la TABLA IV) y tener presente que al tocar el objeto en el dispositivo móvil, por el módulo bluetooth se va a enviar siempre su identificación y valor.

Tabla 4.Código para leer los objetos de la interfaz robótica

Objeto	Nombre	Valor del Byte de Identificación	Rango del Byte de Valor
G13	Botón y	113	[0,1]
G14	Botón x	114	[0,1]
G15	Botón A	115	[0,1]
G16	Botón B	116	[0,1]
G6	Slider S7	106	[0,255]

2.9 Consola Serial y Demo.

Finalmente, se realizó la construcción de una consola serial, que permita la transmisión y recepción de datos y los presente en el dispositivo móvil. El algoritmo para esta parte consta de una función principal, que crea un buffer de datos y está constantemente actualizándolo con los datos que llegan al bluetooth del dispositivo móvil. La consola serial ofrece bastantes utilidades como: flujo de datos

bidireccional, ofrece la opción de parada de recepción de datos, presentación de datos en serie o saltando líneas; limpieza de datos de pantalla con un botón y la presentación de un gran número de datos en una ventana pequeña. Esta consola permite testear o revisar tramas de bits de los puertos seriales, observar la variación de parámetros de señales que se envíen a ella, enviar datos o caracteres del celular para activar actuadores, revisar la transmisión de datos de pines seriales, I2C o SPI y cualquier utilidad similar al uso del programa Putty. Las partes de esta Interfaz pueden verse en la Figura 17. La interfaz del demo, simplemente es una demostración de activación de salidas digitales y analógicas.

Figura 17. Consola Serial

2.10 Resumen de Resultados

El resultado final es la obtención de 6 aplicaciones inmersas en una sola aplicación, con un código de complejidad alta que se evidencia en un archivo pesado. En los anexos se encuentra un link de descarga, tanto para la aplicación que posee una extensión ".apk" como para el código fuente. La aplicación se construye para correr bajo Android 4.3, compatible con versiones anteriores. La velocidad de transmisión se configura a 115200 baudios/s entre los dos módulos, el emparejamiento del módulo bluetooth del Tesla no exige contraseña e inicialmente se debe hacer una vinculación a la Tablet del módulo. La librería se la llamó Tesladroid y el conjunto de ventanas se muestran en la Figura 18.

Las pruebas de su correcto funcionamiento fueron llevadas a cabo, con el uso del módulo Bluetooth Smirf Gold, tipo Rn-41, con un alcance con línea de vista de alrededor 100 m, lo que brinda autonomía de control para las diferentes ventanas de la aplicación. En resumen la aplicación incluye: i) Osciloscopio portátil: tiene el aspecto de un osciloscopio DSO QUAD [4] que permite visualizar en tiempo real señales enviadas desde el dispositivo Tesla, analizarlas en tiempo y amplitud mediante la interfaz y controlar diferentes aspectos de visualización gráfica de la señal. ii) Interfaz para robótica que cuenta con una interfaz pensada para controlar cualquier prototipo robótico (motores, servomotores, dirección y cualquier función variable), ya que cuenta con un control similar a las consolas de juego. iii) Control para Simulink: esta aplicación incorpora un conjunto de sliders que permiten modificar ganancias LQR, de realimentación de estados, controles PID, entre otros; pero especialmente se diseña para ser compatible con simulaciones virtuales mediante el toolbox "Simulink 3D Animación" [5]. iv) Consola serial: permite enviar y recibir datos como cualquier consola serial donde cada comando permitirá establecer una función determinada (depende del programador asignar estas funciones). v) Demo Tesla: esta realiza una demostración de control sobre entradas y/o salidas digitales y analógicas del dispositivo desde el dispositivo móvil. vi) Aplicación extra: esta parte admite incorporar cualquier aplicación por parte del programador para controlar lo que se desee.

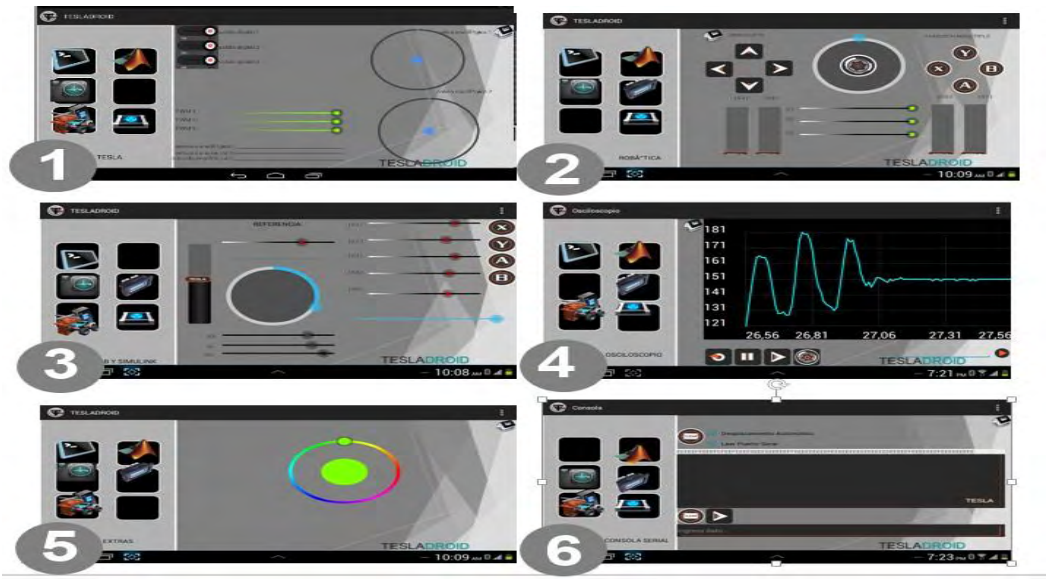


Figura 18. Aplicación Android Tesladroid. La imagen ilustrada muestra las 6 aplicaciones: la imagen 1 es el demo tesla., la 2 es la interfaz robótica, la 3 es la Interfaz para Matlab, la 4 es el osciloscopio digital, la 5 es una aplicación para agregar otras aplicaciones y la 6 es una consola serial.

3. MÉTODO PARA SIMULACIÓN VIRTUAL

3.1 Bases Preliminares

Simulink 3d Animación: Es un toolbox que ofrece la vinculación de objetos gráficos en 3D a bloques y modelos de Simulink y a algoritmos de Matlab que permite visualizar y verificar el comportamiento de un sistema dinámico en un entorno de realidad virtual. Los objetos se representan en el "Virtual Reality Modeling Language" (VRML), un lenguaje de modelado 3D estándar. Puede animar un modelo 3D, cambiando la posición, rotación, escala y otras propiedades de los objetos durante la simulación en tiempo real. También puede inyectar señales de los sensores virtuales y datos de animación 3D en Simulink y Matlab. Con el editor de "3D World", se puede crear escenas detalladas y montadas a partir de modelos 3D exportados a partir de fuentes web o diseños CAD, tal como se observa en la figura 19. Se puede incorporar múltiples vistas y escenas 3D e interactuar con periféricos como tarjetas de desarrollo, joystick, mouse u otro dispositivo de hardware.

Figura 19. Entornos creados con el toolbox Simulink 3d Animation.

3.2 Parámetros de diseño

El método de simulación que se quiere diseñar es una herramienta interactiva para cambiar parámetros importantes en sistemas de control como múltiples ganancias, referencias y valores de constantes; en una simulación que se está ejecutando en tiempo real. Estos cambios se realizan con la manipulación de dispositivos Android y debe afectar objetos 3d creados a partir del toolbox de Simulink 3d Animación. El diseño está sujeto a los siguientes parámetros.

- Mediante el uso de un dispositivo móvil se deben poder transferir datos de variables booleanas y enteras al entorno de desarrollo de Simulink
- Se debe hacer aprovechamiento del toolbox de Simulink 3D Animación [10] y la capacidad táctil de los dispositivos móviles
- El dispositivo Tesla debe ser el puente de conexión entre Android y Simulink
- El software que se construya no debe afectar en ningún caso el tiempo de simulación en Simulink

3.3 Estructura de Hardware y Software.

El desarrollo Android, explicado en la subsección B, contiene una ventana dedicada a Matlab y a permitir la variación de parámetros de sistemas de control y variables en general. La interfaz gráfica de esta parte de la aplicación se muestra en la Figura 20; en donde, todos los objetos que presenta se enfocan en variación de ganancias y referencias de controles como PID, LQR, realimentación de estados.

Figura 20. Interfaz de Matlab para la aplicación Tesladroid

La interfaz ofrece múltiples cambios de referencia, manipulación de variables booleanas por medio de los botones que están al extremo derecho y variaciones de constantes de todo tipo. El proceso de simulación virtual se puede ver en la Figura 21, en donde se muestra que la manipulación de un dispositivo móvil genera variaciones de los objetos de la aplicación, dichas variaciones son enviadas mediante bluetooth al dispositivo Tesla, quien mediante un algoritmo pre-programado, las lee y envía los datos a Simulink haciendo uso de la librería Teslasim. Una vez recibidos los datos,

estos se transfieren a los bloques del toolbox de Simulink 3d Animation, para que este permita en objetos 3d ejecutar movimientos de rotación y desplazamiento, entre otros.

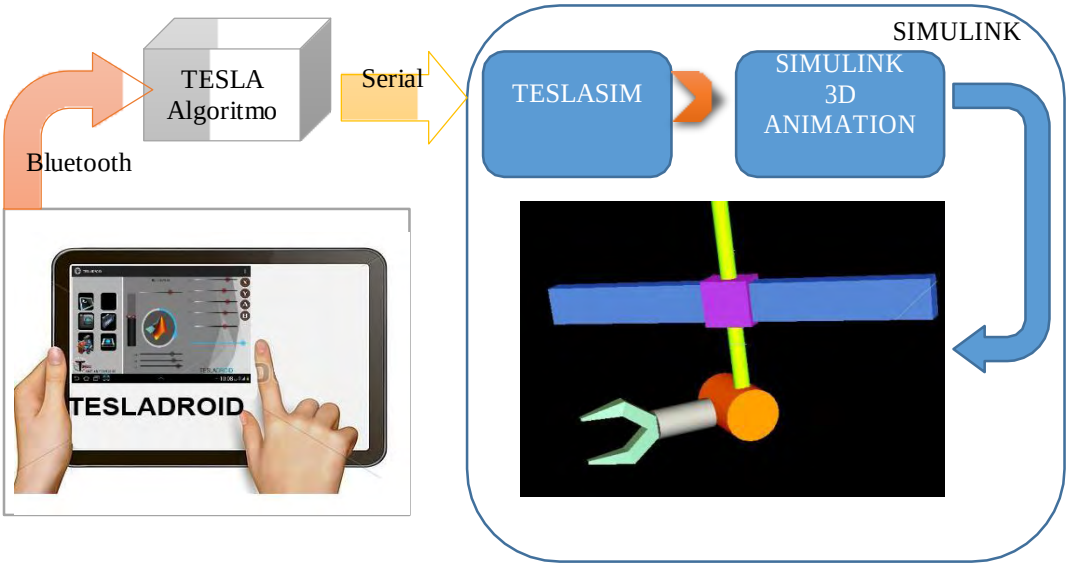


Figura 21 Método para simulaciones virtuales usando el dispositivo Tesla

3.4 Algoritmo programado en la Tarjeta Tesla

El algoritmo programado se basa en una máquina de estados que está constantemente almacenando valores que corresponden a los objetos de la aplicación. Por medio del puerto serie del dispositivo Tesla, en donde está conectado el módulo bluetooth, se recibe un byte de identificación y otro byte del valor del objeto, el algoritmo por máquina de estados se usa para actualizar las variables a partir de los 2 bytes recibidos. El algoritmo es muy eficiente, ya que puede recibir simultáneamente valores al azar y actualizar las variables que corresponden a los objetos de la aplicación, y si se presentan bytes erróneos por algún problema en la comunicación, la máquina de estados permanece en un estado de lectura, hasta que se presenta una correcta identificación del byte de identificación del objeto o "widget" (corresponde a cada elemento gráfico de una aplicación móvil). El código completo se puede revisar en los anexos. La TABLA V presenta la identificación y rango de valores que pueden tomar las variables.

Tabla 5. Tramas de bytes para leer la interfaz de android para matlab

Nombre	Valor del Byte de Identificación	Rango del Byte de Valor
F1	220	[0,1]
F2	221	[0,1]
F3	219	[0,1]
F4	218	[0,1]
F5	216	[0,1]
F6	214	[0,1]
F7	217	[0,1]
F8	215	[0,1]
V	213	[0,100]
R1	209	[0,100]
R2	210	[0,100]
R3	211	[0,100]
R4	212	[0,100]
K1	200	[0,100]
K2	201	[0,100]
K3	203	[0,100]
K4	204	[0,100]
K5	205	[0,100]
KP	206	[0,100]
KI	207	[0,100]
KD	208	[0,100]

La máquina de estados programada en el dispositivo Tesla tiene en cuenta la TABLA V para realizar el proceso indicado en la Figura 22. Se tienen dos variables principales: V, para almacenar la lectura serial y S para almacenar el estado, en donde el estado S=0 permite constantemente estar realizando una lectura y además calculado los siguientes estados. En caso de que se reconozca un byte de identificación que representa valores de 200 a 221, se realiza un cambio de estado y una vez identificado el objeto, se realiza una siguiente lectura serial que representa el valor del objeto (verificando que este byte dentro del rango permitido (valores de 0 a 100)). Si el dato es verificado

se realiza la asignación de la variable que le pertenece a ese objeto con el valor de la lectura serial y se vuelve al estado base S=0 para comenzar nuevamente la actualización de los valores correspondientes a las variables de cada objeto. Condiciones iniciales:

S=0,V=0

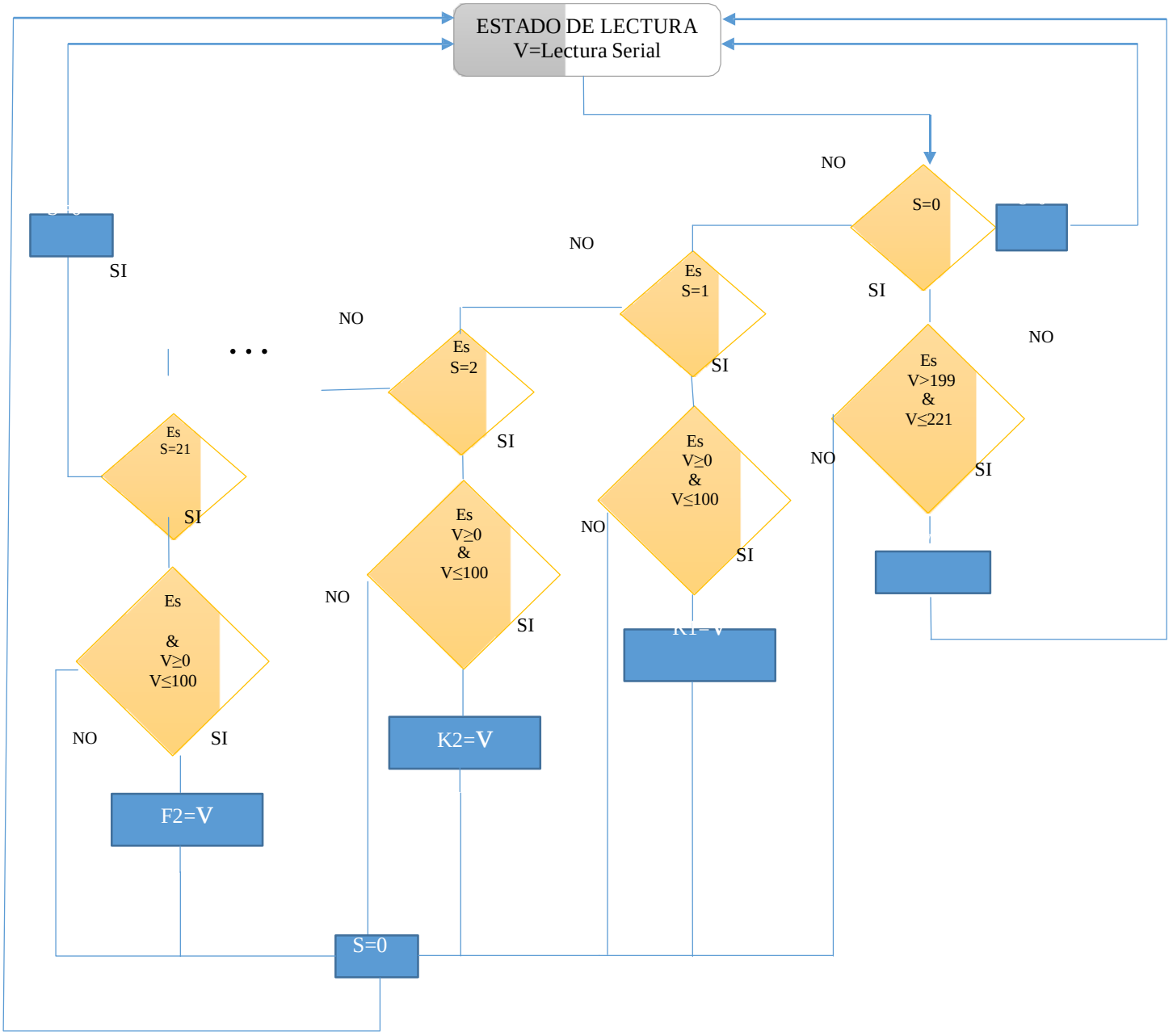


Figura 22. Algoritmo en diagrama de bloques para lectura de tramas de la Interfaz Android para Matlab

De manera interactiva se realizaron simulaciones virtuales llamativas, entre ellas se realizó el cambio de referencia de un levitador magnético, mostrado en la Figura 23 y Figura 24, al que se le implementa un control por realimentación de estados con integrador y se observa en un objeto 3d los cambios de referencia realizados desde una Tablet Samsung. En la simulación se ha integrado el bloque de Interfaz de Android, que viene incluido en la librería Teslasim.

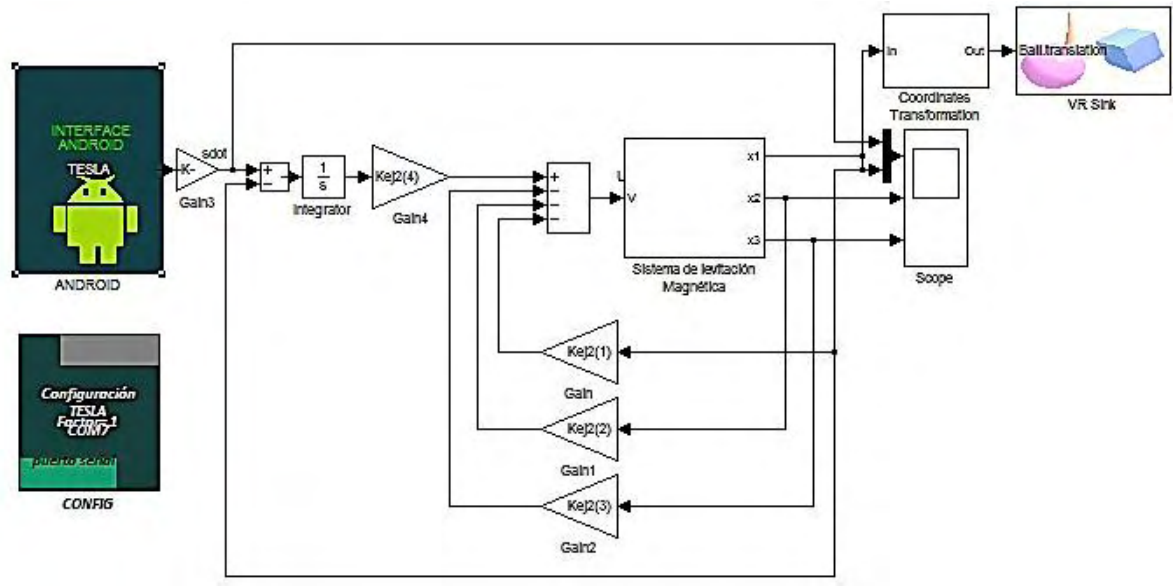


Figura 23. Implementación del método de simulación virtual, usando la librería Teslasim y el toobox de Simulink 3D Animación.

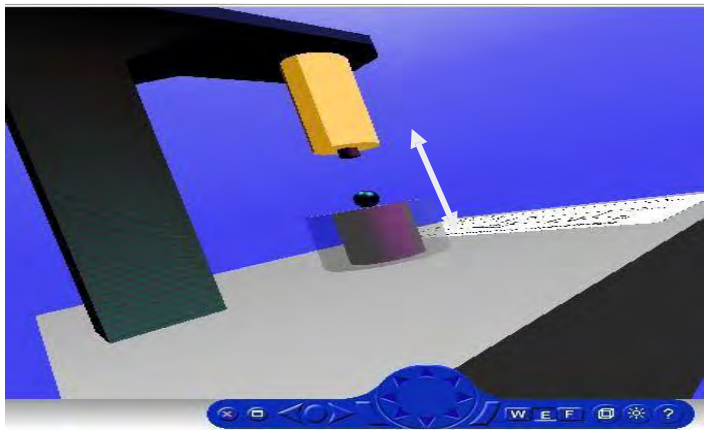


Figura 24. Resultado de la simulación, Control sobre un eje.

4. DISEÑO Y ELABORACIÓN DE UN PROGRAMA PARA LA COMPILACIÓN Y CARGA DE CÓDIGO.

La estructura del dispositivo Tesla (que será explicada en la sección siguiente) es diferente a la de un microcontrolador común o una tarjeta Arduino, debido a que: i) dos microcontroladores tienen conectados sus pines, por lo que para funcionar uno o funcionar otro, se debe cambiar el estado de impedancia de sus pines (para no generar cortos y dañar ambos microcontroladores). ii) Para distribuir los pines a los diferentes microcontroladores se hace necesario de un multiplexado, iii) cada pin posee un circuito de protección de acuerdo a su naturaleza, iv) se tiene un microcontrolador ATSAM3X8E sin el chip atmega16U2, al cual se necesita programarlo sin el uso de este chip. Un ejemplo que ilustra la necesidad de tener un programa de edición, compilación y carga de código debido a la estructura de hardware es: una simple adquisición analógica en arduino se realiza leyendo el valor del conversor ADC por lo que necesita una sentencia, en el Tesla una adquisición analógica necesita de 3 salidas digitales y una lectura del conversor(debido a su circuito de protección), por tanto, si se realiza con lenguaje arduino se necesitaría de 4 sentencias, por lo que es necesario crear un programa con un lenguaje en el cual se realice este proceso con una sola sentencia. La tarjeta Tesla posee un enfoque en sistemas de control y posee unas librerías a las cuales se le debe crear un lenguaje preferiblemente con sentencias simples y en español. La interfaz que ofrece arduino no es muy llamativa y al editor de código le hace falta una interfaz similar a la que ofrece entornos java como: Eclipse o Netbeans, especialmente la numeración de líneas de código y opciones de configuración. La tarjeta Tesla posee asociados módulos de comunicación (como módulos xbee, bluetooth, etc) por tanto, estos deben ser accesibles desde el lenguaje de programación. Esta tarjeta tiene asociado un procesador, el cual posee un sistema operativo pesado, por tanto, se hace necesario desde la programación permitir acceder al ventilador o cambiar los parámetros de refrigeración, mediante el uso del ventilador y el sensor de temperatura asociados al procesador.

Debido a que el dispositivo Tesla posee un sistema operativo interno derivado de Linux, se necesita crear un programa que sea compatible con una arquitectura ARM (debido a que el procesador posee esta arquitectura) y compatible con un sistema operativo Linux (Lubuntu 12.04) para poder programar la tarjeta con el uso de la misma, también debe ser compatible con versiones de Windows (7 y XP) para poder programar la tarjeta externamente con el uso de un PC cualquiera.

Lo anterior lleva a concluir la necesidad de poseer un programa que permita:

- La carga de código es de una manera diferente a la plataforma Arduino, debido a que no se cuenta con el chip atmega16U2 en uno de los microcontroladores, por tanto, no se puede realizar el proceso como la hace Arduino. Se debe encontrar un método que permita realizar este proceso y que se incluya dentro del programa.
- El uso de un lenguaje diferente (lenguaje nativo) ya que se tienen librerías de control a las cuales se les necesita inventar el lenguaje, funciones como lectura analógica o salida PWM que deben ejecutarse con una sola sentencia y con el programa de Arduino no se lograría debido a la estructura de hardware.
- Poseer una interfaz gráfica intuitiva y llamativa con un enfoque en sistemas de control y modificación de aspectos como: menús, submenús, colores, librerías externas, objetos de compilación, animaciones de botones, animación de inicio, etc.
- Tener compatibilidad con Linux y Windows, también con una arquitectura ARMhf.

Este proceso de crear este tipo de programas a partir de un archivo fuente, fue realizado por el equipo de Intel junto con el equipo de arduino para crear la plataforma Intel Galileo [11], este hecho brinda la posibilidad de decir, que esta plataforma tiene un gran desarrollo de software que va a la vanguardia con el diseño software realizado por el equipo de Intel.

4.2 Bases preliminares

Para una mejor comprensión del diseño del programa propio de Tesla, se presenta el siguiente marco conceptual,

- Software Libre: Arduino es una plataforma de hardware libre que incluye un programa para compilación y carga de código a la tarjeta. Este programa está elaborado en plataforma java a partir de un código base utilizado en la elaboración de Processing [12] y la inclusión de librerías en lenguaje C. Esta documentación está abierta al público en general para ser modificada y mejorada.

- Lenguaje de programación nativo: Es el código por medio del cual entiende una máquina o un microcontrolador que corresponde a las sentencias de código digitadas sobre un

programa para que cumpla una función determinada. El lenguaje nativo de Arduino, se deriva de un conjunto de librerías internas de II nivel, este tema será tratado más adelante.

- JDK de Java: Es un software que provee herramientas de desarrollo para la creación de programas en Java. Puede instalarse en una computadora local o en una unidad de red.

- Cygwin: es una colección de GNU y "Open Source", herramientas que proporcionan una funcionalidad similar a una distribución de Linux en Windows. Estas herramientas sirven para proporcionar un comportamiento similar a los sistemas Unix en Microsoft Windows y su objetivo es portar software que ejecuta en sistemas POSIX a Windows con una re-compilación a partir de sus fuentes.
- Apache Ant: Es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción. Es, por tanto, un software para procesos de automatización de compilación desarrollado en lenguaje Java y requiere de esta plataforma para ejecutarse. Esta herramienta tiene la ventaja de no depender de las órdenes de archivos de cada sistema operativo, sino que se basa en archivos de configuración XML y clases Java para la realización de las distintas tareas, siendo idónea como solución multi-plataforma.

- IDE Arduino: Corresponde al entorno de desarrollo integrado donde se realiza la programación de las tarjetas Arduino. Permite la edición, compilación y carga de código.

4.3 Parámetros de Diseño

Para construir un programa que permita programar una tarjeta con una arquitectura de hardware diferente a un Arduino común se tienen en cuenta los siguientes parámetros

- El programa debe poder ser ejecutado en Windows y en distribuciones de Linux para arquitecturas ARM
- EL programa debe permitir la inclusión de lenguaje nativo
- Debe permitir: edición, compilación y carga de código
- La interfaz debe presentar una estética llamativa
- Debe contener librerías enfocadas en sistemas de control

4.4 Estructura del programa

Para efectos de diseño, el programa se denomina Teslasoft y se basa en el código abierto de arduino IDE 1.5 [13]. Con modificaciones sustanciales, cambio del lenguaje interno, modificación de la interfaz gráfica, adición de librerías de control y carga de código diferente. La estructura general se muestra en la Figura 25. El trabajo se centró en cambiar las librerías de segundo nivel, para modificar el lenguaje nativo; modificar las librerías de III nivel para cambiarlas por librerías de control; realizar modificaciones del archivo bossac para realizar una carga de código diferente debido a la estructura del hardware y renovar la interfaz gráfica por medio del cambio de botones, ventanas, numeración, menús, colores entre otros. Cada parte será explicada en las siguientes subsecciones.

El archivo bossac es un ejecutable con extensión ".exe" que se encarga de cargar cualquier programa precompilado en microcontroladores de la familia arduino, se ejecuta cada vez que se carga un programa realizando procesos de: borrado de la memoria, configuración de la transmisión, carga de código, etc. Para la tarjeta Tesla es necesario modificarlo debido a que se necesitan señales de "set" y "reset" para permitir que se cargue un programa a la tarjeta. En las tarjetas arduino este proceso lo realiza un chip (generalmente ATmega16U2,), pero en la tarjeta Tesla se tiene un microcontrolador ATSAM83XE que no posee este chip, por lo cual se modifica el archivo bossac para que el procesador de la tarjeta Udoo sea el encargado de realizar esta tarea.

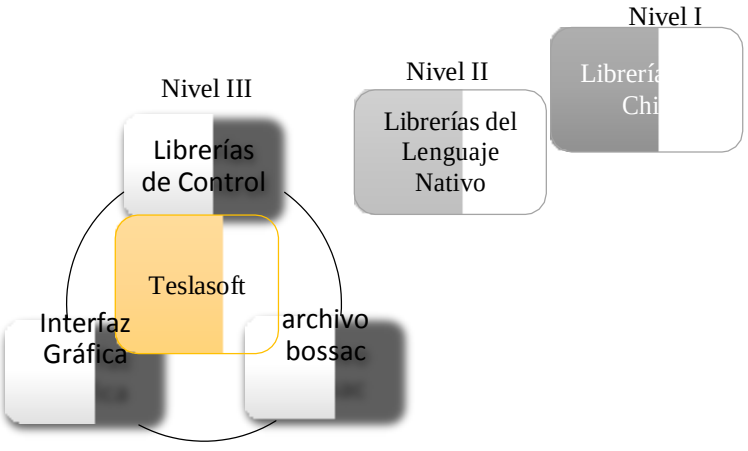


Figura 25. Estructura de Teslasoft

4.5 Proceso para construir el programa

La elaboración del programa a partir de un código fuente (arduino 1.5 IDE), involucra la serie de pasos mostrados en la Figura 26, en donde, el software de compilación y elaboración del ejecutable debe ser flexible para permitir que el programa modificado pueda ejecutarse en sistemas operativos Linux, Windows y Mac. El código fuente es el programa que posee el equipo de arduino para generar el entorno de desarrollo de arduino, las modificaciones son descritas más adelante y el producto final es el programa Teslasoft.

Figura 26. Proceso de elaboración de Teslasoft

4.6 Adecuación del Software para compilación y código fuente

Para obtener un programa compatible con Windows y con una arquitectura Linux ARM, se deben instalar los programas de Cygwin, Ant Apache y el JDK de java. A la hora de instalar Cygwin se deben también instalar los siguientes paquetes: git, make, gcc-MinGW, g++, Perl, ActiveState, unzip, zip, coreutils (o textutils), gzip, alquitrán, openssh y nano. Cada paquete cumplirá un papel importante dentro de la compilación, por ejemplo los paquetes MinGW y g++ permiten construir el ejecutable final Teslasoft.exe. El proceso de compilación de Teslasoft se realizó con tres programas, mostrados en la Figura 27, todo el desarrollo de Teslasoft está basado en java, Cygwin usa sentencias de Apache Ant para compilar y generar el ejecutable de Teslasoft. Se debe tener especial cuidado en instalar estos programas, especialmente en la parte de agregar variables del sistema para que así se pueda ejecutar correctamente la compilación del programa.

Figura 27. Programas para la compilación de Teslasoft

4.7 Descarga y Modificación del Código Fuente

El código fuente de arduino IDE 1.5 se puede descargar de internet para modificar funcionalidades internas y externas, incluyendo la interfaz gráfica del programa, para lo cual cerca de 32 archivos deben ser modificados. En los anexos se muestra el proceso más detallado de cómo se realiza la construcción del programa Teslasoft, se agrega el código fuente de Teslasoft y un manual detallado del proceso para crear un entorno de desarrollo para cualquier microcontrolador de la familia Atmel.

4.8 Compilación y elaboración del ejecutable

Con la consola de Cygwin se accede al archivo modificado y por medio de sentencias de Apache Ant se logra generar el ejecutable y los archivos de soporte. A partir del código fuente y en especial del archivo build.xml, al final se obtiene una versión compatible con sistemas operativos: Mac, Linux y Windows.

4.9 Lenguaje Nativo

En general, un microcontrolador ejecuta un determinado proceso cuando se le carga un archivo con extensión ".hex" que es el lenguaje por medio del cual el microcontrolador "entiende". Para generar este archivo el programador ingresa un conjunto de instrucciones usando directa o indirectamente las librerías de I, II y III nivel, mostradas en la Figura 25. Posteriormente este archivo es revisado por medio de un compilador, y si alguna sentencia está mal escrita se advierte del error al programador. De lo contrario, si todo está correcto, este compilador generará el archivo .hex que finalmente es cargado a la memoria flash, de tal manera que una vez alimentado el microcontrolador empezará a ejecutar todas las funciones que el programador diseñó a partir de sentencias simples. El ciclo correcto de compilación y generación del archivo ".hex" se puede visualizar en la Figura 32.

El lenguaje nativo se construye a partir de la modificación de las librerías de segundo nivel ya que las librerías de primer nivel no se las debe manipular puesto que corresponden a cada arquitectura

de procesador o microcontrolador. Por ejemplo, para el ATSAM3X8E hay un conjunto de librerías de 1er nivel que acceden a todas sus funcionalidades, entre ellas la librería madre se denomina "chip.h". Las librerías de segundo nivel son aquellas que no se agregan con la sentencia "#include <librería.h>" de tal forma que son librerías ocultas. Para el caso de Teslasoft estas librerías pueden encontrarse en un conjunto de archivos en la siguiente dirección del software: "Teslasoft\build\windows\work\hardware\arduino\sam\cores\arduino".

Para crear un lenguaje nativo se tiene que manipular las librerías de II nivel agregando librerías o quitando las que no sean necesarias para la plataforma. Estas librerías se las debe elaborar en lenguaje C, por lo que su construcción es diferente a las comunes librerías de III nivel. Una vez que se construye o modifica librerías existentes se las debe agregar a la librería madre (en este caso "Arduino.h"). Una vez agregadas, ya se puede construir código en el editor del programa de Teslasoft. La prueba final de que todo funciona bien se basa en ejecutar la compilación y que no se genere ningún tipo de error en la consola inferior, tal como indica la Figura 28.

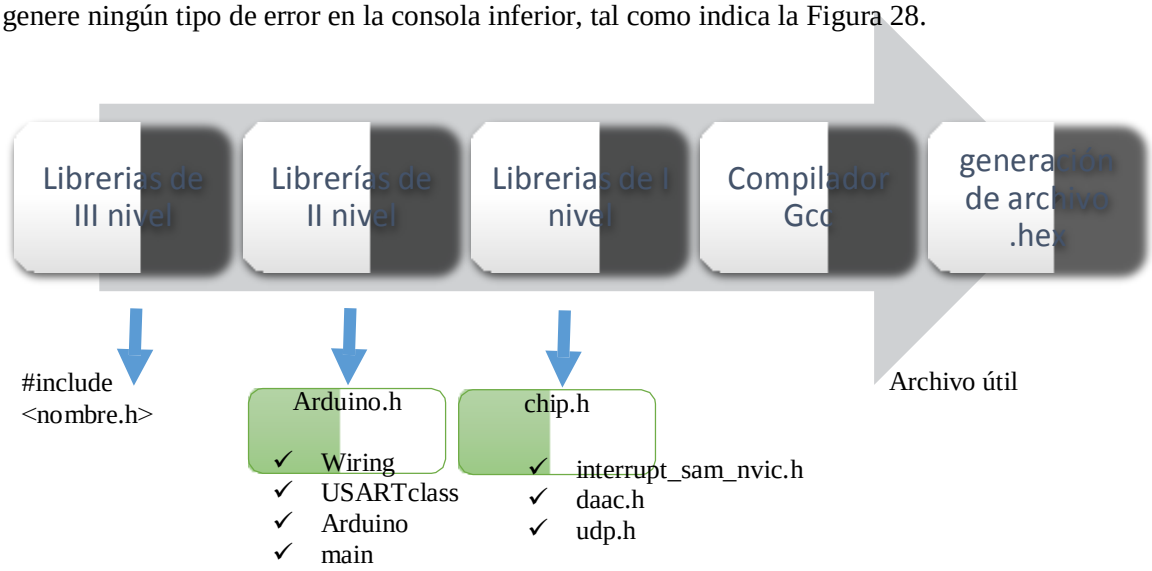


Figura 28. Generación del archivo .hex a partir de librerías y el compilador

En conclusión, el lenguaje nativo son las sentencias que usa un programador para crear un programa lógico que realice una determinada función en el microcontrolador, para ello es necesario modificar las librerías de II nivel, en estas librerías es donde se almacena el conjunto de sentencias que usa el programador en el software Teslasoft; estas sentencias deben ser sencillas y prácticas con el fin de motivar la programación y realizar procesos complejos con una reducida cantidad de sentencias.

Para cumplir con el objetivo de crear un lenguaje nativo generado en la Universidad de Nariño, se incluyó una librería "TESLA.h" en las librerías de 2do nivel que cambia el lenguaje de arduino y lo convierte en un lenguaje para ser usado con la plataforma Tesla", esta librería se diseñó en lenguaje C y contiene sentencias simples que realizan procesos complejos, algunas sentencias se incluyen en la TABLA VI. El resultado de la compilación a partir del uso de Teslasoft y el lenguaje nativo por medio de la librería TESLA.h se indica en la Figura 29, en donde se muestra que un código creado únicamente para la tarjeta puede generar un archivo .hex a partir de una librería de II nivel y lenguaje nativo.

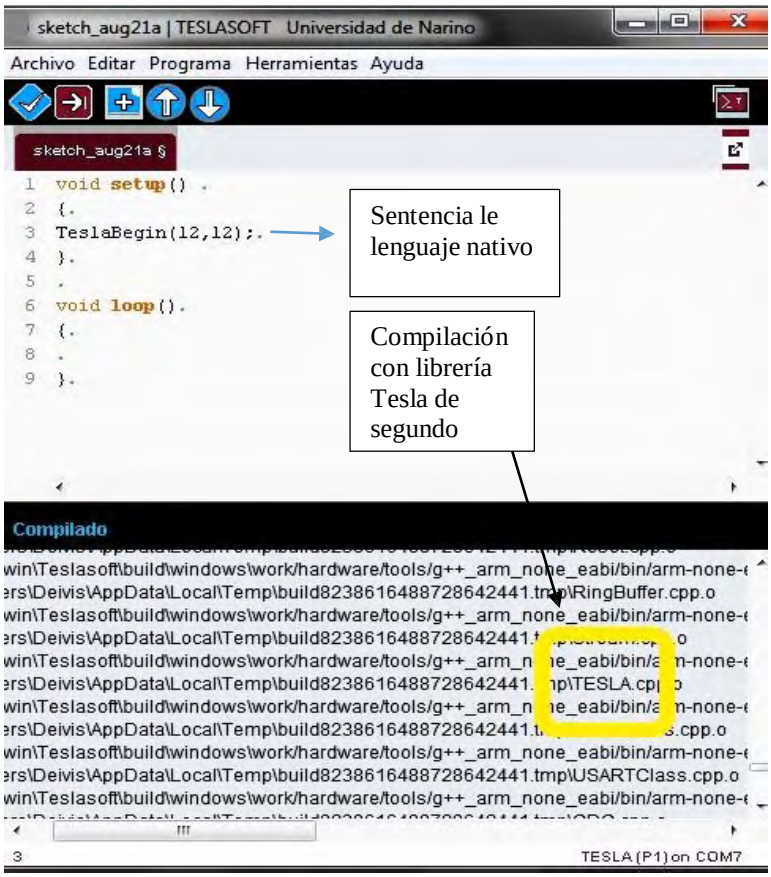


Figura 29. Demostración de la librería de segundo nivel Tesla y código nativo en Teslasoft

Tabla 6. Algunas sentencias de lenguaje nativo para TESLASOFT

Sentencia	Uso
teslaConfig(a,b)	Inicia los conversores ADC a un valor de a bits y el conversor DAC a un valor b bits
lecturaDigital(a)	Establece una lectura digital del pin a
escrituraDigital(a,b)	Establece el pin digital a en un valor booleano b
lecturaPulso(a,alto)	Devuelve un valor en microsegundos calculado a partir del tiempo en que permanece un pin digital a en un estado alto.
LecturaPulso(a,bajo)	Devuelve un valor en microsegundos calculado a partir del tiempo en que permanece un pin digital a en un estado bajo.
Osciloscopio(val)	Envía un valor val al osciloscopio de la aplicación Tesladroid
EscrituraPWM(a,b,INT)	Envía al pin a una señal PWM con un valor de ciclo útil b y una amplitud interna de 5V.
EscrituraAC(a,ALTO)	Acciona una carga AC conectado a los pines AC dispuestos en el Tesla.
EscrituraPWM(a,b, EXT)	Envía al pin a una señal PWM con un valor de ciclo útil y con una amplitud de voltaje externo que se ingresa a través de un pin ubicado a lado de los pines PWM.
lecturaXbee()	Lee una trama de bits desde el módulo Xbee
LecturaAnalogica(a,SEG)	Lee un pin analógico a y devuelve el valor, la palabra “SEG” indica que se lee un pin protegido contra altos voltajes de entrada.
LecturaAnalogica(a,INSEG)	Realiza una lectura analógica de un pin que no posee protección contra altos voltajes.
lecturaAndroid(a)	Devuelve el valor booleano o entero de un objeto dentro de la aplicación Tesladroid
Color(a,b,c)	Establece un led rgb en un color cualquiera, producto de la manipulación de la escala rgb dada por las variables a, b, c respectivamente. Se usa de muestra y como remplazo del popular ejemplo básico “hola mundo”
Ventilador(on)	Acciona el ventilador, que refrigera el procesador
temperaturaCPU()	Devuelve el valor de la temperatura en grados centígrados que posee
ControlTempCPU(automatico,a b,c)	Establece un control automático de la temperatura del procesador, el control aplicado es PID y es ajustado con las variables a, b y c; las cuales representan las ganancias Kp, Kd y Ki del control PID respectivamente.

4.10 Resultados

Realizando todo el proceso anterior, se rediseñó una plataforma de programación a partir del software original de arduino y con la utilización del programa Eclipse se modificó la interfaz gráfica y algunos opciones del programa para adecuarla a sistemas de control. La plataforma de software posee 4 librerías principales: control PID, fuzzy y dinámica de replicadores. Posee una forma de carga de algoritmos al microcontrolador diferente a arduino, un lenguaje de programación sencillo, interfaz gráfica llamativa y con una forma de compilación conocida y modificable.

Por otra parte, se logró compilar el código del software mediante el uso del programa Cygwin Terminal y obtener versiones compatibles con Linux y Windows. La versión de Linux servirá para elaboración, compilación y carga de código sin el uso de un computador (solo mediante el uso de la tarjeta Tesla, ya que posee Ubuntu internamente) y la versión de Windows permitirá la carga de código de manera externa con el uso de un computador. En la Figura 30, se muestra la interfaz gráfica.

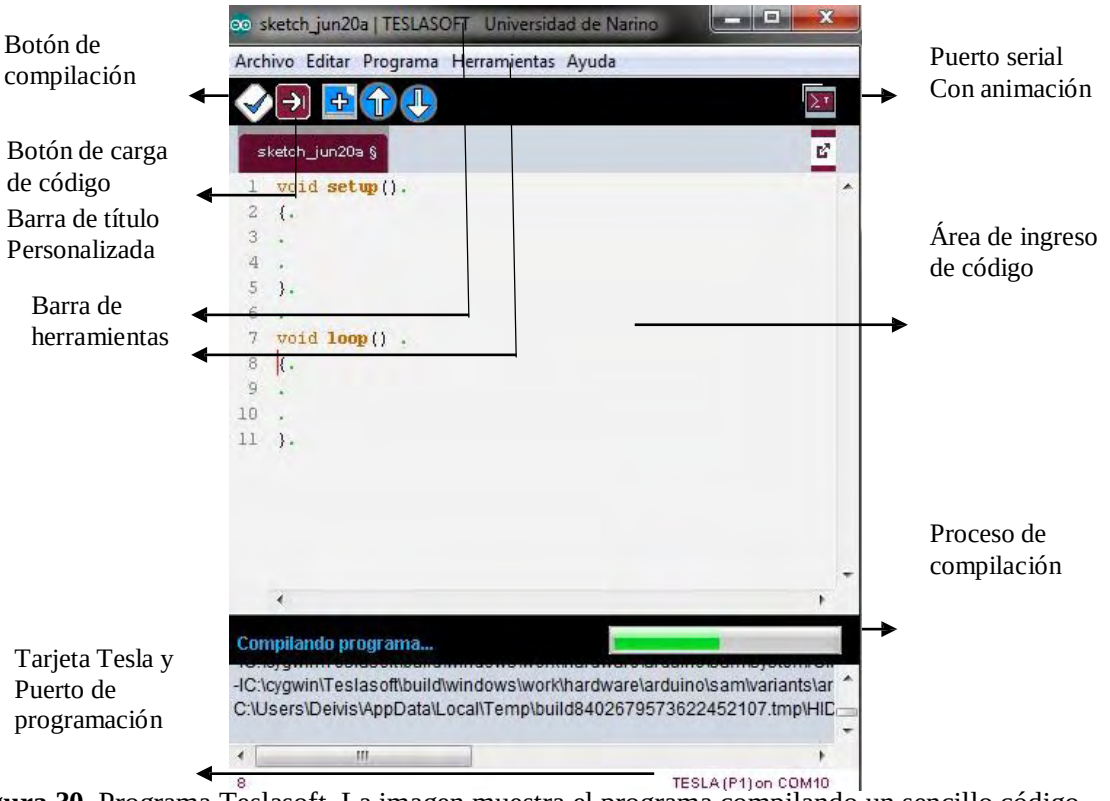


Figura 30. Programa Teslasoft. La imagen muestra el programa compilando un sencillo código.

En la interfaz gráfica también se hicieron modificaciones como se presenta en la Figura 31. Dentro de las opciones se puede mostrar la opción de TESLA-BOOK, donde los estudiantes podrán almacenar proyectos de control o de cualquier naturaleza, mientras que en la opción EJEMPLOS-CONTROL se tiene ejemplos de las librerías principales y de configuración inicial del dispositivo.

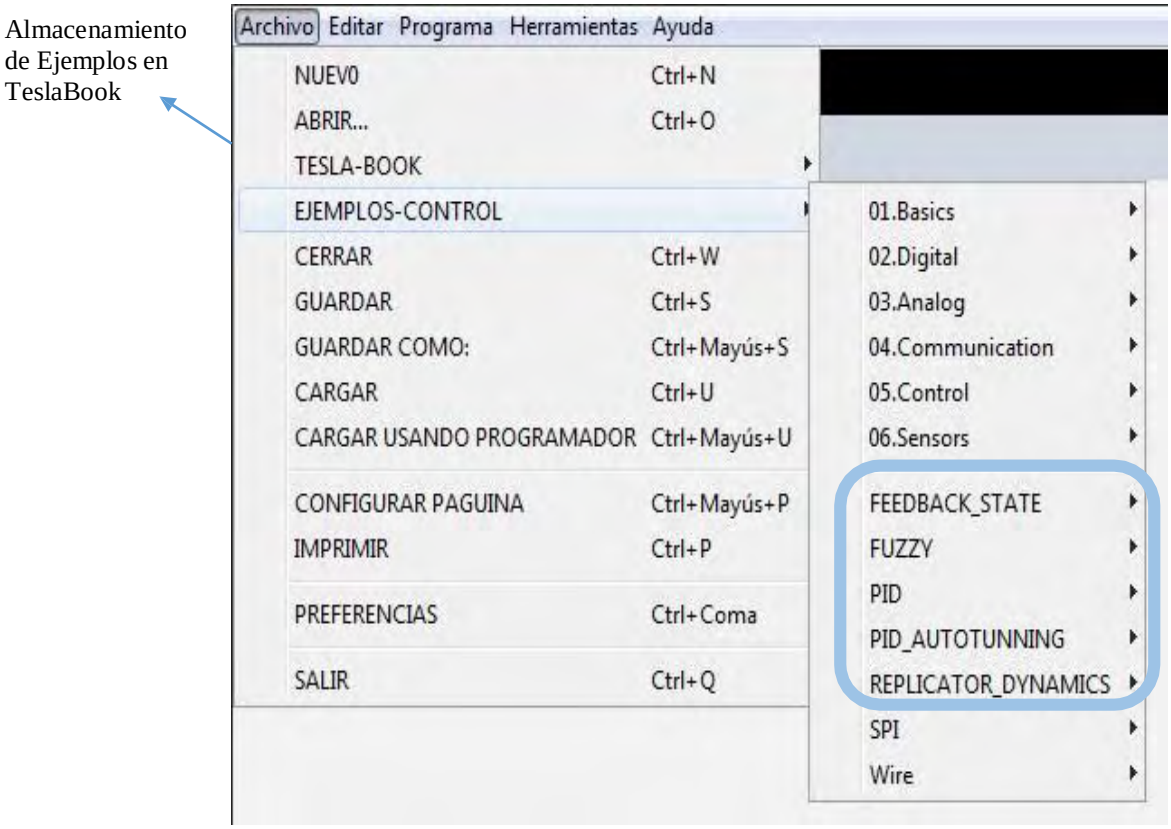


Figura 31. Librerías, ejemplos y opciones del programa. La imagen indica la estructura de opciones que ofrece al desarrollador y las librerías de control principales.

En cuanto a las librerías de control, este es un tema a tratar en una sección superior, ya que implica un desarrollo matemático y algunos parámetros de control. En los ejemplos básicos se establece como iniciar la plataforma con máxima resolución del conversor análogo-digital y el conversor digital-análogo, protocolos de comunicación y sencillos ejemplos de manipulación de la tarjeta.

5. SISTEMA OPERATIVO y REDES

5.1 Bases preliminares

Un sistema operativo que corre en una memoria posee 3 archivos principales, el núcleo, el sistema de archivos y el gestor de arranque. Para la tarjeta Tesla se cuenta con una tarjeta Udoo con un sistema operativo derivado de Linux que permita cargar programas basados en Java como Teslasoft, Processing y programas pre-compilados en plataformas como Eclipse y Netbeans [14], Este sistema operativo debe ser flexible para permitir la implementación de múltiples tipos de red. A continuación se describen los archivos principales del sistema operativo.

Núcleo: (Kernel): es un software que constituye una parte fundamental del sistema operativo y se define como la parte que se ejecuta en modo privilegiado. Es el principal responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o en forma básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema. Como hay muchos programas y el acceso al hardware es limitado, también se encarga de decidir qué programa podrá hacer uso de un dispositivo de hardware y durante cuánto tiempo, lo que se conoce como multiplexado. Acceder al hardware directamente puede ser realmente complejo, por lo que los núcleos suelen implementar una serie de abstracciones del hardware. Esto permite esconder la complejidad, y proporciona una interfaz limpia y uniforme al hardware subyacente, lo que facilita su uso al programador.

Sistemas de Archivos (File System): Los sistemas de ficheros son contenedores de archivos, que se almacenan, probablemente en un árbol de directorios, junto con los atributos, como el tamaño, propietario, fecha de creación y similares.

Gestor de Arranque (U-boot): Es un software que permite mediante un conjunto de procesos y ejecución de un conjunto de comandos, el arranque del sistema a partir de la utilización del kernel. Este es universal y soporta múltiples arquitecturas, entre ellas la ARM.

Arquitectura ARM: es una arquitectura de 32 bits desarrollada en 1983 por la empresa Acorn Computers Ltd para usarse en procesadores que manejan un sistema de instrucciones realmente simple lo que le permite ejecutar tareas con un mínimo consumo de energía.

5.2 Parámetros de configuración y selección

- Se necesita escoger un sistema operativo que permita ejecutar Teslasoft y archivos Java.
- Se necesita una arquitectura que permita la ejecución rápida de programas.
- Se debe tener en cuenta que no sea una arquitectura pesada.
- Debe presentar un equilibrio entre una ejecución rápida y una arquitectura pesada

5.3 Sistema Operativo

El sistema operativo seleccionado fue Lubuntu 12.04 LTS armHF, el cual es un sistema rápido, muy liviano y modificable. Adicionalmente permite la instalación de Arduino 1.5 y la inclusión de ejecutables realizados en Java. Fue seleccionado por ser el sistema operativo que posee un equilibrio en todos los aspectos que se necesitan citados anteriormente.

El sistema fue instalado en una memoria y se lo hizo correr en la tarjeta para hacerle las modificaciones de fondo. Entre estas se encuentran las modificaciones del núcleo, la interfaz gráfica y la instalación de programas y paquetes necesarios para brindar un soporte de software robusto. El kernel de Linux se modifica para agregar nuevas funcionalidades a la tarjeta, establecer los pines de salida y entrada en determinados valores, agregar nuevos puentes de comunicación entre el microcontrolador y procesador, entre otros. En este caso, se realiza para manipular los puertos y para agregar puertos seriales a la tarjeta.

5.4 Modificación del núcleo

Para modificar el kernel o núcleo se descarga un kernel pre-existente basado en Linux ARM que se almacena en un directorio propicio y se le realizan las modificaciones (como agregar un puerto serial adicional). Posteriormente se compila haciendo uso de sentencias nativas del sistema pre-instalado. El proceso completo de la modificación del kernel o núcleo se puede consultar en la guía [15].

5.5 Instalación de programas

La TABLA VII indica las características de los programas instalados sobre la plataforma Tesla (en la memoria que alberga el sistema operativo).

Tabla 7. Programas multimedia instalados internamente

Programa	ícono	Descripción
Teslasoft		Programa basado en el programa de Arduino, reprogramado en Eclipse IDE y adaptado a sistemas de control. Está compilado para correr en Linux y Windows y es el programa que permite la elaboración, compilación y carga de algoritmos a la tarjeta. Es únicamente compatible para programar tarjetas Tesla.
Processing		Programa utilizado para realizar algoritmos de procesamiento de imágenes e interfaces gráficas que permiten obtener datos analógicos y digitales de la plataforma Tesla e incorporarlos a una interfaz gráfica de usuario, permitiendo una comunicación bidireccional entre el programa y la tarjeta.
Python		Software que genera algoritmos para múltiples usos. Para la tarjeta se lo utiliza para manejo de bases de datos y para crear redes que permitan conexiones remotas con otros dispositivos mediante el uso de internet.
Scilab		Software matemático, con un lenguaje de programación de alto nivel, para cálculo científico. Es interactivo de libre uso y similar a Matlab
Pure Data		Lenguaje de programación visual que permite a los artistas, investigadores y desarrolladores crear software de forma gráfica, sin necesidad de escribir líneas de código. Se utiliza para procesar y generar gráficos de sonido, vídeo, 2D / 3D, y sensores de interfaz, dispositivos de entrada, y MIDI. Puede trabajar fácilmente a través de redes locales y remotas.
Geany		Es un editor de texto, con las características básicas de un entorno de desarrollo integrado. Fue desarrollado para proporcionar un pequeño y rápido IDE que tiene sólo unas pocas dependencias de otros paquetes. Es compatible con muchos tipos de archivos.
Code Blocks		Multiplataforma para programar en C, C++ y Fortran++, construido para satisfacer necesidades exigentes de los usuarios. Está diseñado para ser extensible y totalmente configurable.
Scratch		Entorno de programación gráfica para crear animaciones, permitir conexiones seriales para interactuar con hardware externo, entre otras.
JDK de Java		Software necesario para soportar aplicaciones en Java que pueden enfocarse en redes, manejo de bases de datos, interfaz gráfica, comunicación serial, entre otros usos de la plataforma Java.

Se instalaron otros programas tales como. Calculadora, PHP, MySQL y Mozilla Firefox.

5.6 Redes.

Para el manejo de redes hay tres formas de permitir la transferencia de datos que pueden ilustrarse en la Figura 32. La forma de corto alcance se realiza haciendo uso de protocolos de comunicación SPI, serial o por I2C, lo que implica la comunicación por medio de cable. La segunda opción es la comunicación por medio del uso del módulo xbee de la tarjeta, con lo que se puede obtener un alcance de 45 km, usando módulos XBEE PROHP 900 MHz y antenas de alta ganancia, según su fabricante [16]. La tercera forma es haciendo uso de internet con la ejecución de programas elaborados en Java que permitan la recepción de datos seriales y la asociación de servicios web.

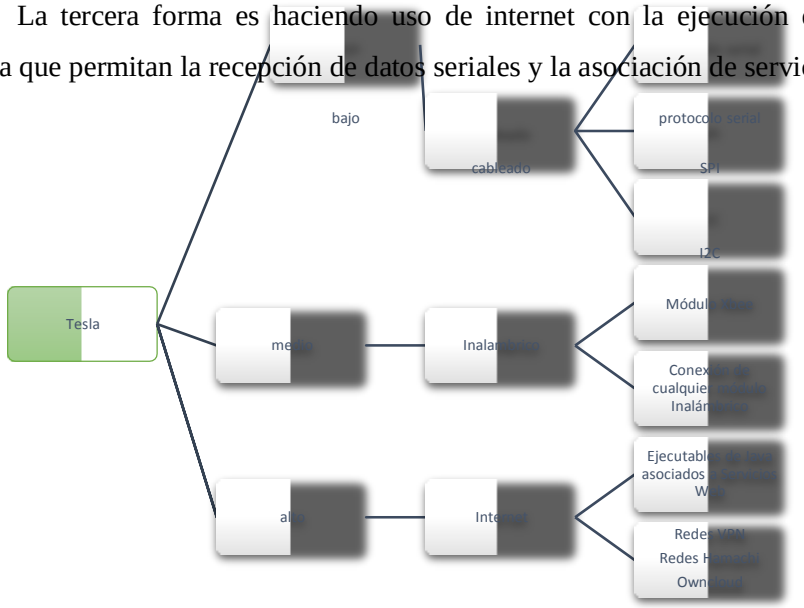


Figura 32. Arquitecturas de red para la Tarjeta

Para acceder a la tarjeta desde una red local se configuró una el programa VNC (siglas que en español significan computación en red, es un software que permite acceder a un escritorio remoto de un pc externo por medio de internet) y se instaló el servidor en la tarjeta de tal manera que mediante un computador conectado a la red local se puede acceder a las funcionalidades del dispositivo, incluyendo la programación de la plataforma.

La parte más interesante es el manejo de datos por medio de internet, con lo cual se puede enviar datos del dispositivo a cualquier parte del mundo, crear laboratorios virtuales y programar la tarjeta remotamente, entre otras aplicaciones. La aplicación más probada del dispositivo fue el acceso a la red por ejecutables de Java transmitiendo datos a un servidor remoto, lo que prueba la correcta transmisión. Para este proceso se utiliza un programa compilado en Java y compatible con el JDK instalado en la tarjeta. A este programa se le debe asociar un servicio web que permita la conexión con una fantasma web (servicio web sin máscara) y una página web sin máscara gráfica. Esta página se encarga de enviar los datos directamente a un servidor que se encuentre en cualquier parte del mundo. Este proceso se ilustra en la Figura 33.

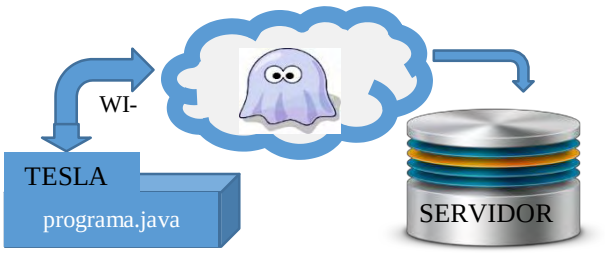


Figura 33. Transmisión de datos vía internet

II. DISEÑO DE HARDWARE

1. ESTRUCTURA ELECTRÓNICA GENERAL

En general, el sistema electrónico del dispositivo Tesla (mostrado en la Figura 34) se basa en una tarjeta de hardware libre que incluye un procesador (puede ser raspberry pi, Cubieboard, Udoo, entre otras) y dos microcontroladores ATSAM8X3E, uno de ellos usado para desarrollo y el otro usado como máster. El microcontrolador de desarrollo se puede programar tanto con el procesador como externamente por medio del uso de un computador. Este microcontrolador es el único que se puede programar de los dos y permite el desarrollo e implementación de algoritmos por medio del programa Teslasoft. El microcontrolador master se encarga de la interfaz de usuario de la tarjeta por medio de una pantalla LCD táctil, gestiona la comunicación con aplicaciones Android (en especial la comunicación bidireccional con la aplicación Tesladroid), permite controlar el proceso de simulaciones virtuales entre Android y Matlab y la comunicación con programas exteriores como Matlab, Simulink, Processing y Java. Este diseño se acopla a la estructura de software planteada en la sección II.

La tarjeta de desarrollo o procesador posee dos conexiones seriales con el microcontrolador de desarrollo, aunque de forma nativa se presenta una sola conexión. Para habilitar la segunda conexión es necesario modificar el núcleo o "kernel" de Linux, compilarlo y volverlo a cargar (proceso explicado en la sección Sistema Operativo y Redes, Sección II). Estas conexiones seriales son fundamentales, ya que son un puente de comunicación entre los programas internos (Teslasoft, Processing, Java e Internet) y las entradas analógicas, digitales, SPI e I2C, entre otras. En síntesis, son el puente entre el mundo físico (sensores) y la parte intangible del software.

Los dos microcontroladores poseen pines compartidos. Por ejemplo, los pines analógicos y digitales son usados por los dos dispositivos con el fin de poder usar la tarjeta tanto para adquisición de datos con Matlab como para realizar un manejo de señales con programación independiente. Por otra parte, los microcontroladores se pueden comunicar por medio de 3 conexiones: 1 serial y 2 con protocolo I2C. Esta transferencia directa de datos es importante, ya que permite establecer los estados de cada microcontrolador y controlar el modo de funcionamiento de Tesla por medio de la habilitación de los pines de propósito general a la hora de cambiar de función (adquisición, simulación virtual, programación y modo pc).

El microcontrolador master controla las funciones que debe ejecutar todo el sistema. Está a cargo de la visualización de funciones en el display LCD táctil y el manejo de la comunicación bluetooth. Con el uso del bluetooth realiza todo el proceso de simulación virtual y con el uso del LCD permite el cambio de función, de acuerdo a lo que el usuario escoja. El microcontrolador de desarrollo tiene asociado el módulo Xbee y los protocolos de comunicación digital como SPI, I2C y Serial. Los pines de entrada analógica, salida PWM, entrada y salida digital poseen circuitos adicionales de adaptación diseñados como protección y/o para mejorar características electrónicas como compatibilidad de niveles lógicos. Esta parte será discutida más adelante.

Este diseño electrónico particular de Tesla no se presenta en ninguna otra tarjeta existente, por el hecho de: habilitar entradas y salidas de acuerdo al microcontrolador actuante, compartir pines de entradas y salidas analógicas y digitales, utilizar dos microcontroladores con características definidas como master y desarrollador, tener circuitos de acondicionamiento y protección para los pines, asociar módulos de comunicación a los dos microcontroladores, tener puentes de comunicación entre el procesador y los microcontroladores y permitir realizar cuatro funciones

específicas (tarjeta de programación, sistema de adquisición, puente para simulaciones virtuales y minicomputador)

En vista de la necesidad de usar dos microcontroladores que posean una alta velocidad para poder tener tiempos de muestreo cortos y para implementar estrategias de control que exigen un despliegue matemático importante, se decidió usar el microcontrolador ATSAM8X3E del Arduino Due como máster. Para el procesador se escoge a la tarjeta Udoo ya que tiene una gran ventaja frente a las demás tarjetas al permitir correr Teslasoft y ejecutables de Java, además de permitir correr el sistema operativo y tener un microcontrolador ATSAM8X3E empotrado, que será el microcontrolador de desarrollo.

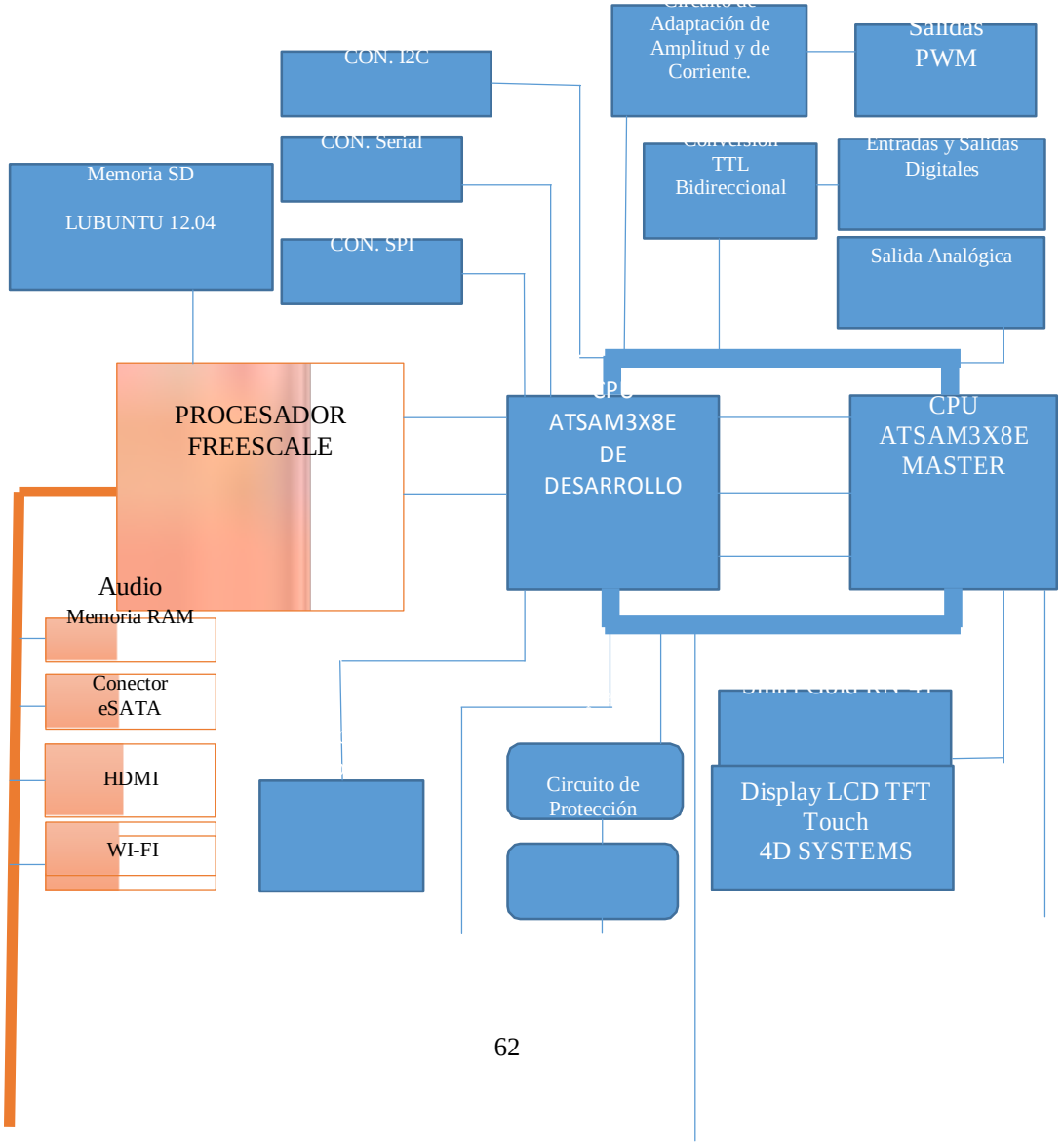
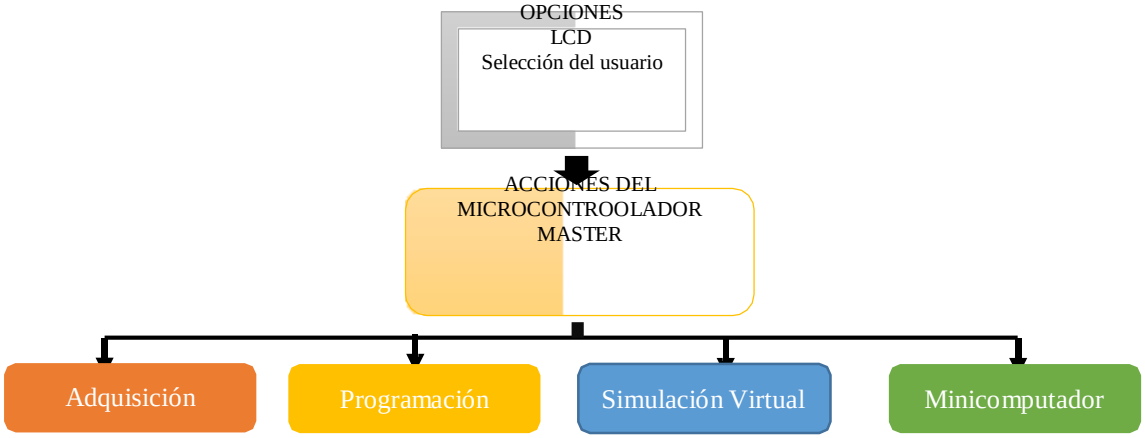


Figura 34. Diagrama de bloques del sistema electrónico del dispositivo Tesla

2. FUNCIONES DEL SISTEMA ELECTRÓNICO

El sistema electrónico tiene cuatro funciones seleccionables desde el LCD táctil (modo: adquisición, programación, simulación virtual y minicomputador), cada función es controlada por el microcontrolador master, cuando un usuario selecciona alguna operación, el LCD táctil envía vía puerto serial una trama de bits, la cual se interpreta (por el microcontrolador master) y de acuerdo a su valor(producto de transformar la trama de bits en valores enteros) se ejecuta una determinada acción; Por ejemplo, si el usuario selecciona(del menú que presenta el LCD, mostrado en la Figura 42) el modo adquisición, el microcontrolador master desactiva el procesador y el microcontrolador de desarrollo (desactivando la tierra del voltaje de alimentación de ambos mediante un relé, accionado por un pin digital de este microcontrolador) y ejecuta un algoritmo cíclicamente para permitir la conexión con Matlab y Simulink. Cada función que selecciona el usuario implica una acción ejecutada por el microcontrolador master, los cambios en hardware y software de acuerdo a la función que seleccione el usuario se muestra en la Figura 35.



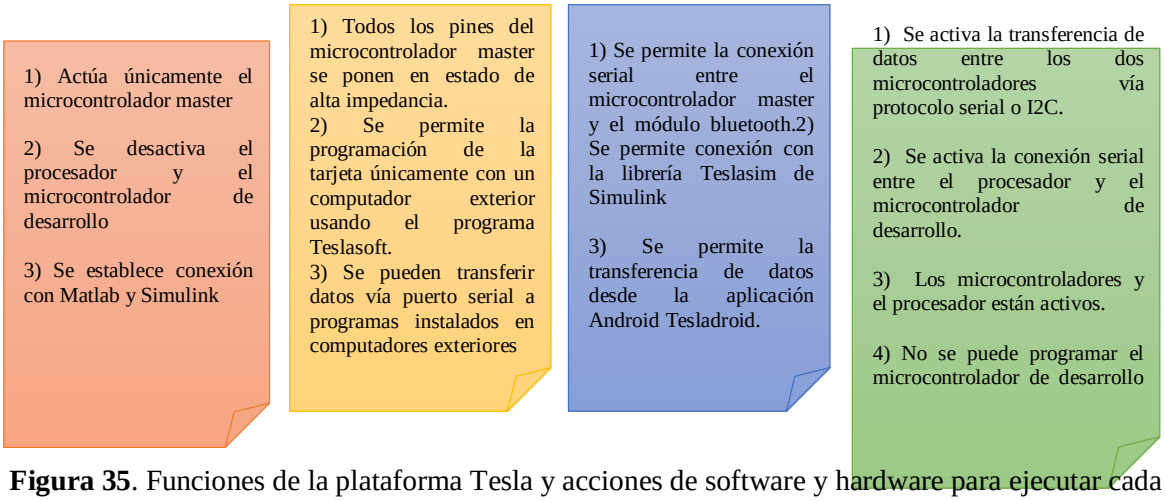


Figura 35. Funciones de la plataforma Tesla y acciones de software y hardware para ejecutar cada función.

Las funciones y los cambios realizados en cada una de estas son:

- Modo adquisición: en el modo adquisición el procesador y el microcontrolador de desarrollo se desactivan (la línea de tierra de la alimentación de voltaje, por medio del accionamiento de un pin digital en un relé) a fin de que el microcontrolador master pueda ejecutar la adquisición de los datos y poder ahorrar energía, ya que una tarjeta de adquisición no debe realizar un gran consumo. El microcontrolador master ejecuta automáticamente un algoritmo que permite enviar y recibir datos de Matlab mediante un algoritmo basado en una máquina de estados finitos, en donde Matlab (por medio de las librerías Teslamat y Teslasoft) envía comandos por el puerto serial y coordina la transmisión de datos. En este modo el microcontrolador master es capaz de controlar todos los pines (digitales, analógicos, PWM, seriales y de protocolo I2C). Es importante establecer que los pines analógicos y digitales de ambos microcontroladores están conectados, para no afectar la lectura del conversor DAC y de pines digitales del microcontrolador de desarrollo basta con desconectar la tierra del microcontrolador de desarrollo, esta conclusión surge producto de pruebas hechas sobre los dos microcontroladores.

- Modo programación: En el modo de programación el procesador y los dos microcontroladores están activos. El microcontrolador master ajusta la impedancia de sus pines con

el fin de evitar cortocircuitos generados por las salidas o entradas de ambos microcontroladores. Es decir, se da autonomía al microcontrolador de desarrollo para que se pueda programar cualquier tipo de algoritmo, incluyendo algoritmos para sistemas de control. El microcontrolador de desarrollo se puede programar únicamente con el uso de un computador externo o con el uso del sistema operativo de la tarjeta, ya que se corta la comunicación entre el procesador interno y el microcontrolador de desarrollo, para programar la tarjeta (el microcontrolador de desarrollo) se debe hacer uso de Teslasoft instalado de Windows o en Mac Os, cabe destacar que ningún otro programa podrá programar la tarjeta y que Teslasoft(descrito en la sección II) tiene: las librerías de control, el editor de código, ejemplos, librerías y programa (bossac.exe) de compilación, lenguaje de programación y la interfaz adecuada para ejecutar cualquier tipo de algoritmo.

- Modo Simulación Virtual: En este modo, el microcontrolador master habilita la conexión con dispositivos Android y ejecuta el modo adquisición para poder transferir datos desde dispositivos móviles a Simulink. El procesador y el microcontrolador de desarrollo quedan deshabilitados por medio del uso de un relé y se establece comunicación directa con el módulo bluetooth. Cabe destacar que se hará uso de la librería Teslasim y de la aplicación Tesladroid, también se podrá transferir datos a la aplicación y hacer uso del osciloscopio y la consola serial (que posee la aplicación Tesladroid, descrita en la sección II).

- Modo PC: Se habilita el procesador y los microcontroladores haciendo ajuste de impedancias para evitar cortocircuitos en las entradas y salidas. La diferencia con el modo de programación es que únicamente con el procesador se puede programar el microcontrolador de desarrollo. También se habilita la programación por medio de escritorio remoto, se puede usar los programas multimedia que se encuentran en la TABLA VII, se pueden establecer redes a través del uso de ejecutables java y enviar datos se sensores adquiridos por medio de un algoritmo ejecutado en el microcontrolador de desarrollo. Este modo es muy flexible y permite al usuario usar la tarjeta para cualquier aplicación que implique el uso de software en conjunto con el uso de microcontroladores. En este modo se puede hacer interfaces gráficas de todo tipo, la Figura 36 presenta una interfaz realizada con Processing y el microcontrolador de desarrollo y sin hacer uso de un computador externo y usando el modo de minicomputador, esto prueba la comunicación entre procesador y microcontrolador de desarrollo y el amplio espectro de posibilidades de desarrollo de software y hardware.

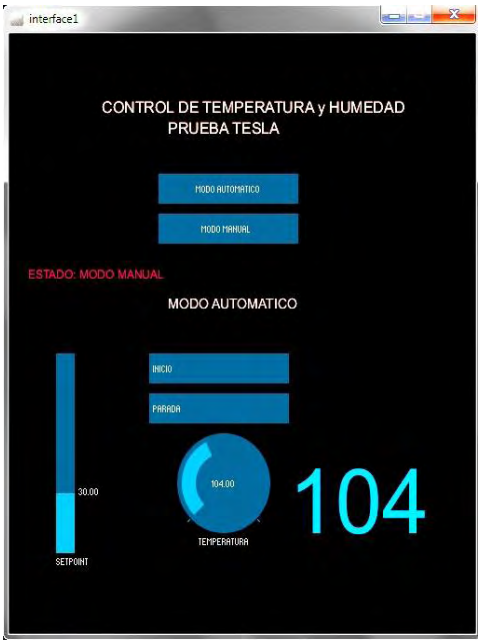


Figura 36. Interfaz entre Processing y el microcontrolador de desarrollo de la plataforma Tesla

3. ENTRADAS ANALÓGICAS, CIRCUITO DE PROTECCIÓN Y MULTIPLEXADO

El dispositivo Tesla tiene 16 entradas analógicas, 8 de ellas protegidas contra alto voltaje y 8 entradas no protegidas, la estructura general es presentada en la Figura 37. El circuito de multiplexado se usa para leer 8 entradas (8 pines en la borneras de entrada) usando una sola entrada analógica de los microcontroladores y usando únicamente 1 circuito de protección, multiplexar es útil ya que se usa menos recursos (1 circuito de protección en lugar de 8) y se amplía las entradas analógicas. El circuito de protección y multiplexado se explica mas adelante.

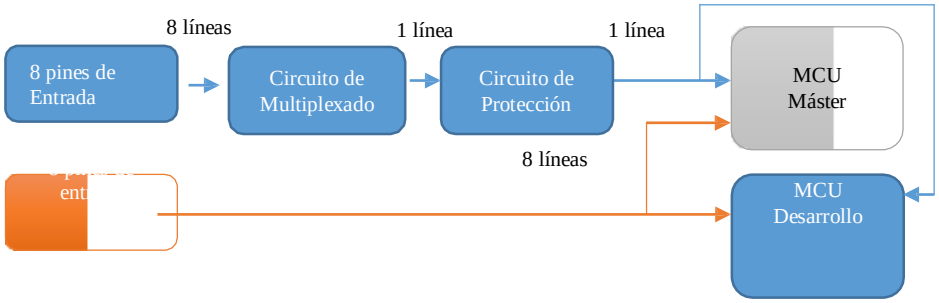


Figura 37. Estructura general de las entradas analógicas para la plataforma Tesla

Proteger las entradas analógicas de los microcontroladores es esencial, ya que una subida de voltaje por encima del voltaje del conversor (3.3 V), puede dañar totalmente los microcontroladores o los

pines en cuestión. Para esto se realizaron pruebas con amplificadores operacionales en diferentes configuraciones, tanto en amplificadores riel a riel (que poseen un costo elevado), como en amplificadores operacionales normales (TL084, LM324, entre otros). Finalmente se obtuvo un equilibrio entre precio y funcionalidad con la saturación de la fuente de alimentación de un amplificador LM324 configurado en modo seguidor tal como muestra la Figura 38. La función que cumple este circuito se muestra en la figura 39, por tanto, al ingresar cualquier señal que varíe de 0 V a 3.3 V en la entrada, en la salida se va a obtener la misma señal, si la señal de entrada supera los 3.3 V a la salida siempre se va a obtener este valor (3.3 V), debido a que se manipula el voltaje de alimentación del integrado logrando saturar la salida. Este comportamiento representa una ventaja a la hora de proteger los microcontroladores, ya que según la hoja de datos [17], a la entrada del amplificador se puede inyectar señales de hasta 30 V sin sufrir daño. Este resultado se generó como parte de la investigación de lograr una tarjeta bastante robusta. Cabe aclarar que en la Figura 38, la entrada (Vi) es el voltaje que se quiere leer (por tanto, esta parte estaría conectada a borneras de entrada y sería donde se conecten sensores analógicos) y la salida (Vo) se conecta directamente a pines de entrada analógicos pertenecientes a los microcontroladores. El circuito utilizado para las entradas analógicas es el mostrado en la Figura 37, en donde el amplificador LM324 puede operar solo con voltajes positivos de alimentación. La fuente de alimentación de este amplificador se la varía hasta obtener un voltaje máximo de salida de 3.3 V ante cualquier entrada de voltaje que supere este valor. El voltaje de alimentación encontrado experimentalmente para que cumpla con el comportamiento descrito en la Figura 39, es de 4.5797 V.

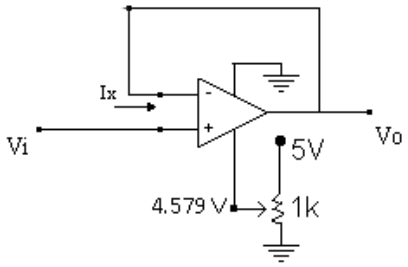


Figura 38. Circuito de protección de pines analógicos de los microcontroladores

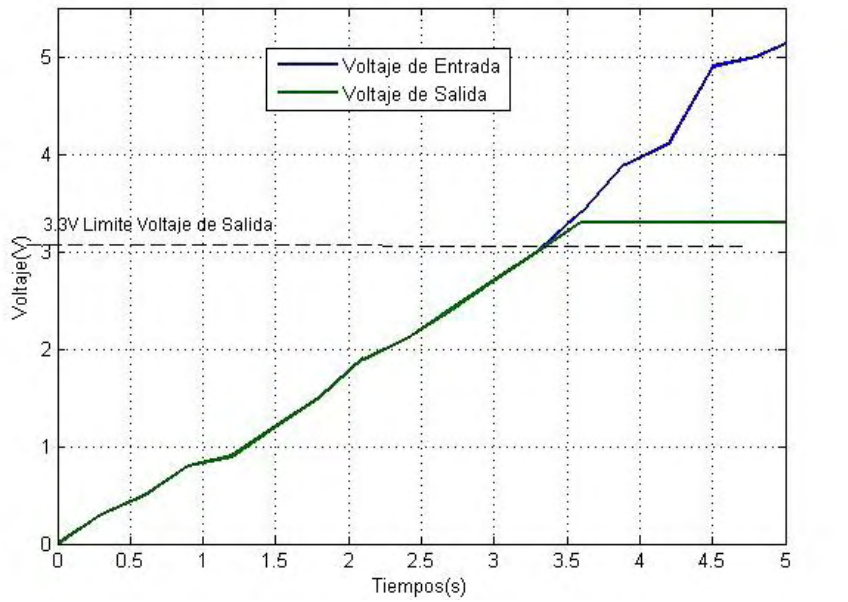


Figura 39. Voltaje de salida ante la variación del voltaje de entrada del circuito de protección

El circuito de multiplexado de acuerdo a la Figura 40, se utiliza para tener un solo circuito de protección (1 solo amplificador operacional) en lugar de 8 circuitos de protección por cada pin de entrada analógico, también se usa para ampliar el número de entradas analógicas a 16, ya que los microcontroladores ATSAM8X3E tienen 12 entradas analógicas (una de ellas ocupada para el sensor de temperatura del procesador). Para realizar el multiplexado de la señal se utiliza el circuito de la Figura 40, en donde se necesita un solo pin analógico para leer 8 entradas analógicas que poseen protección y 8 entradas analógicas que no poseen protección. Para este circuito se utiliza el multiplexor analógico 4051, al que se le ingresan 3 señales digitales que seleccionan cualquiera de las 8 señales disponibles, que posteriormente son protegidas con el uso del anterior circuito.
Mediante el uso de las sentencia "LecturaAnalogica(pin, SEG)", se puede acceder a la lectura de pines protegidos dentro del programa Teslasoft y mediante el uso de la sentencia "LecturaAnalogica(pin, INSEG)" se accede a la lectura de los pines no protegidos; en la variable pin se indica un número de 1 a 8 que corresponde al pin que se desea leer.

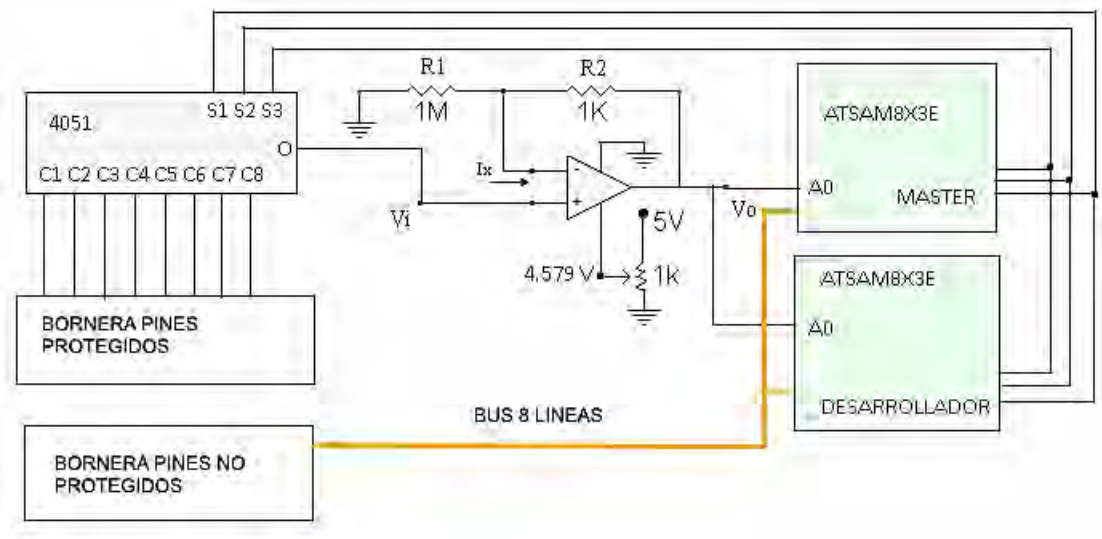


Figura 40. Circuito de protección y multiplexado para pines analógicos

4. SALIDAS PWM

Las salidas PWM de un microcontrolador pueden entregar un máximo nivel de corriente (120 mA para microcontroladores ATSAM8X3E) y entregan una señal cuadrada con una frecuencia constante y con una amplitud constante (3.3 V para un ATSAM8X3E); cuando se conecta una carga que exige más de la corriente que puede proporcionar el microcontrolador, esta carga puede dañar el microcontrolador o bajar la tensión de este. Por esta razón, las salidas PWM se las protege con el uso de transistores y resistencias que limitan la corriente suministrada por los microcontroladores. Se hace uso del transistor 2n2222 [18] con el cual se puede proporcionar hasta 1 A según su hoja de datos. El circuito de protección es el mostrado en la Figura 41, la resistencia conectada en serie con cada pin PWM protege y limita la corriente de salida, la resistencia se la calcula de acuerdo a la limitación de corriente, para la plataforma Tesla la corriente de cada pin PWM se la limita a 15mA y el voltaje de salida es 3.3 V, se aplica Ley de Ohm y se obtiene una resistencia de 220 Ω. También se encuentra un circuito adicional para permitir el cambio de amplitud, el cual está constituido de un transistor por cada pin PWM y un relé(activado por un pin digital) que selecciona el voltaje de alimentación del transistor entre un voltaje interno de 5 V o un voltaje externo que puede ingresar el usuario por un pin habilitado para ese propósito, ajustar el voltaje de alimentación del transistor implica ajustar la amplitud de salida del voltaje del colector y por ende la amplitud de la señal. La salida final es una señal de voltaje en el colector del transistor a la cual se puede controlar la amplitud entre un voltaje interno y un voltaje externo de hasta 40 V

(según las limitaciones de la hoja de datos del transistor). Las salidas PWM dentro del programa Teslasoft se pueden acceder (de acuerdo a la TABLA VI) con las sentencias de código: "salidaPWM(pin, valor, INT)", se ingresa el pin, el valor útil y la palabra INT para ajustar la amplitud a 5V y la sentencia "salidaPWM(pin, valor, EXT)" para que la amplitud se la ajuste con un voltaje ingresado por un pin establecido para esta función. Las salidas PWM son 10 y están conectadas tanto al microcontrolador de desarrollo como el microcontrolador master.

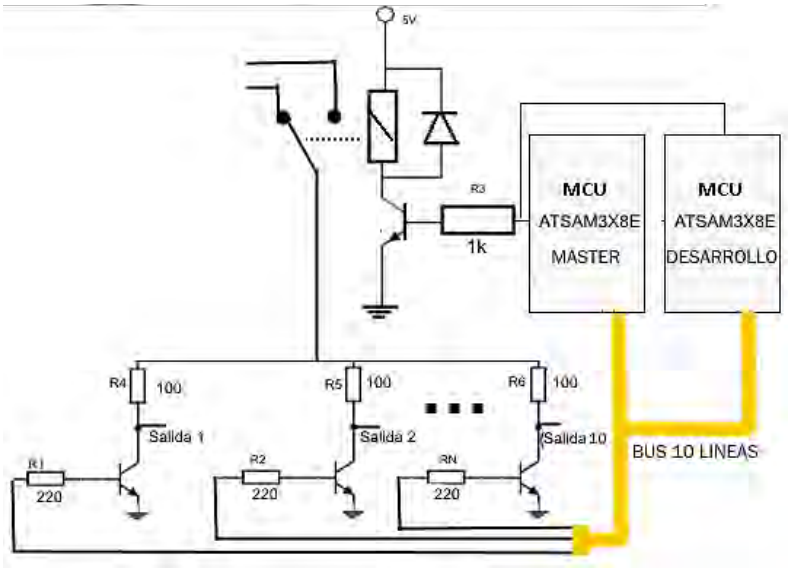


Figura 41. Circuito de protección y variación de amplitud de salidas PWM

5. SALIDAS Y ENTRADAS DIGITALES

Las entradas y salidas digitales son 16 pines que se conectan al microcontrolador master y de desarrollo, estos pines tienen un conversor bidireccional de voltaje ADA-395 [19], que utiliza el chip TXB0108 y realiza conversiones de voltaje de 3.3 a 5 V y viceversa. Realizar la conversión es muy útil ya que la mayoría de integrados, escudos de arduino y circuitos comerciales son tecnología TTL y los microcontroladores funcionan a 3.3 V. De esta manera, este circuito es necesario ante un eventual uso de algún tipo de circuito que funcione a 5 V. Con este integrado se puede convertir en ambas direcciones, por lo que se puede usar un pin digital como entrada o como salida. El circuito usado es el de la Figura 42 duplicado para poder tener 16 pines de entrada y/o salida digital.

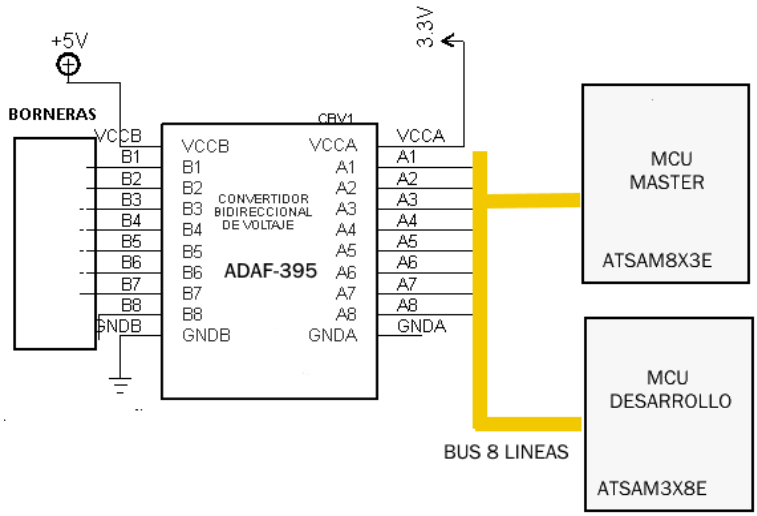


Figura 42. Conversión TTL para entradas y salidas digitales

6. PANTALLA Táctil

La pantalla usada es fabricada por la empresa 4D Systems, con un tamaño de 3.2 pulgadas, táctil y compatible con protocolo serial. Para programar la tarjeta se deben seguir los siguientes pasos ordenados. La estructura de conexión es la presentada en la Figura 43 y los pasos ordenados para cargar un programa se indican en la Figura 44.

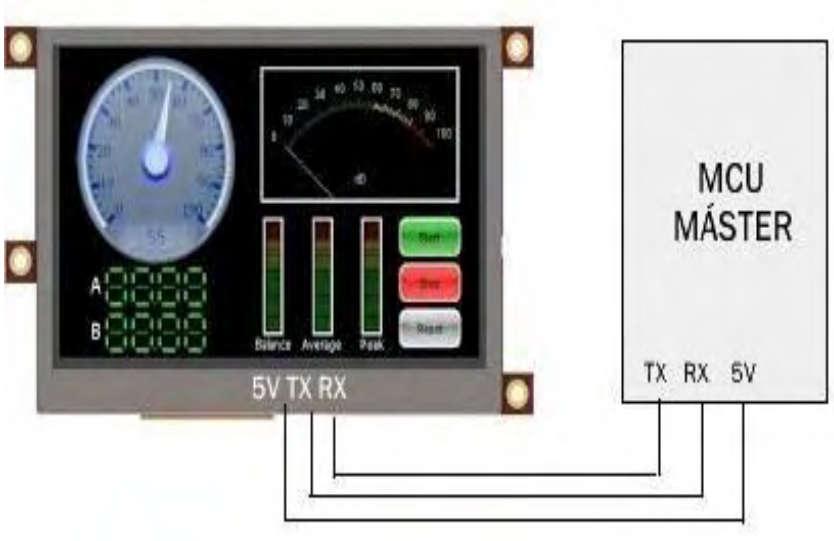


Figura 43. Conexión entre el LCD táctil y el microcontrolador máster

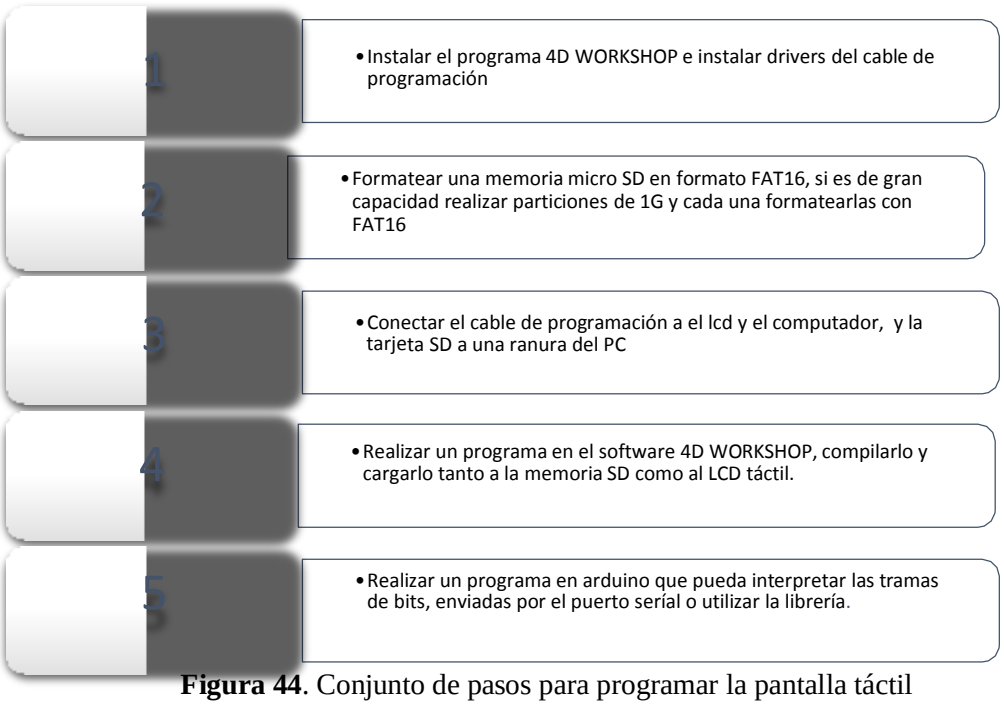


Figura 44. Conjunto de pasos para programar la pantalla táctil

El programa desarrollado en 4D Workshop, tiene la estructura mostrada en la Figura 45. Inicialmente se presenta una pantalla de inicio y posteriormente una interfaz de selección, donde el usuario escoge que función quiere establecer en el dispositivo Tesla. Cada pantalla de cada opción permite trasladarse a otras funciones. Cada vez que se selecciona mediante el uso de los botones, se envía una trama de bytes en hexadecimal, que debe ser interpretada por el microcontrolador para poder cambiar de función

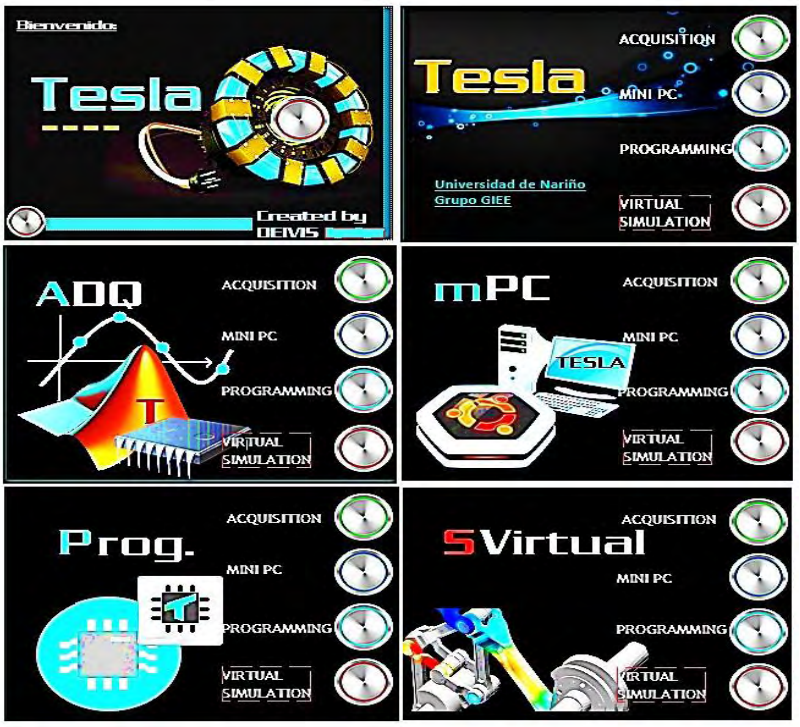


Figura 45. Interfaz de la pantalla táctil

7. SALIDAS DE CONTROL AC

El dispositivo Tesla posee la capacidad de controlar cargas AC, se le adaptó una etapa de potencia basada en opto-acopladores (MOC3011) y triacs (BT137) con lo cual es posible conectar cargas con corriente efectiva(RMS) de hasta 8 A (basado en la capacidad del triac BT137). El circuito es el presentando en la Figura 46, el microcontrolador master controla los opto-acopladores por medio de salidas digitales, para controlar desde el microcontrolador de desarrollo basta con mandar una trama de bits vía serial para que el microcontrolador master accione los opto-acopladores y estos disparen los triacs. Las resistencias R1 y R2 son calculadas con las ecuaciones (1) y (2)

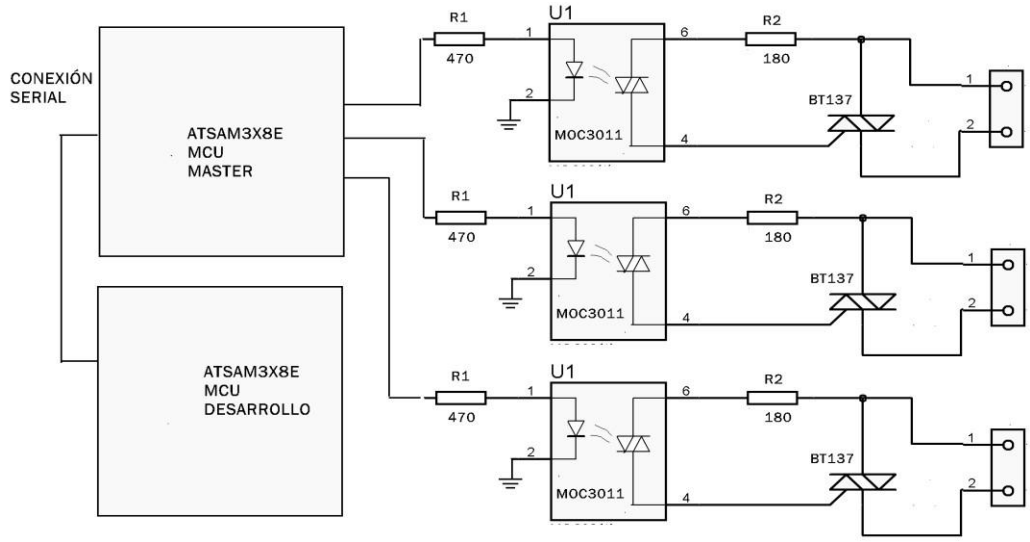


Figura 46. Circuito de potencia para accionamiento de cargas AC

$$R1 = \frac{V - V_d}{I_d} \quad (1)$$

Dónde, Id es la corriente del diodo proporcionada por la hoja de datos del opto-acoplador (6mA), V es el voltaje de microcontrolador (3.3V) y Vd es el voltaje del diodo led (0.3 V).

$$R2 = \frac{V_{pico}}{I_{max}} \quad (2)$$

Dónde, Vpico es el voltaje pico de la señal AC de la red (169.7 V para la red de 110V), Imax es la máxima corriente de entrada del opto-acoplador (1A, proporcionada por la hoja de datos).

Los valores de resistencia R1 y R2 generados con las ecuaciones (1) y (2) se aproximan a valores comerciales

8. BLUETOOTH, MÓDULO XBEE, TARJETAS USADAS EN EL DISEÑO

Los módulos Xbee y Bluetooth van conectados a los puertos seriales disponibles en los microcontroladores, ya que solo implican una lectura seria. El módulo bluetooth es fundamental para el proceso de simulación virtual, por tanto, se decidió conectarlo en el puerto serial del microcontrolador máster y el módulo xbee se conecta en un puerto serial del microcontrolador de desarrollo.

9. IMPLEMENTACIÓN

Una vez diseñados todos los circuitos, se procedió a graficarlos en el programa Eagle. Aunque algunos componentes no estaban presentes dentro de las librerías de este software, se realizaron librerías de los elementos faltantes para no depender de algún diseño erróneo. Los archivos adicionales están disponibles en los anexos. Para ahorrar espacio se procedió a hacer un diseño modular de tres capas: i) en la capa superior se va a ubicar la pantalla táctil junto con un circuito para pines digitales y analógicos; ii) en la capa media se va a ubicar el Arduino Due, el módulo bluetooth, circuito para el resto de pines digitales y un circuito para la adaptación de entradas PWM; y iii) en la capa inferior se ubica la tarjeta Udoo con el módulo xbee, las salidas AC, un conector USB, el control de potencia para la Udoo y el ventilador, entre otros. La Figura 46 indica la disposición de las capas.

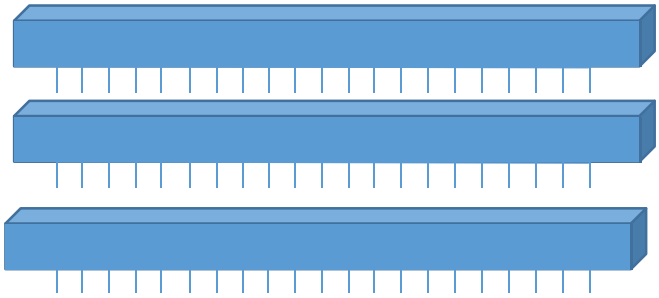


Figura 47. Disposición modular de la tarjeta Tesla

Las capas son los siguientes:

Capa superior: En la capa superior se incorpora la pantalla táctil, 8 salidas digitales, 8 salidas analógicas protegidas, 4 salidas analógicas, un circuito digital y analógico para realizar protección, multiplexado y conversión TTL de las entradas y salidas. Las borneras son industriales con resortes. El circuito impreso es desarrollado en una capa con una capa de pintura para protección anti soldadura. El potenciómetro que se observa permite calibrar la saturación del amplificador operacional LM324 en el circuito de la Figura 38. El circuito de la capa superior es el mostrado en la Figura 48.

Figura 48. Implementación de la capa superior

Capa media: En la capa media están las borneras para 10 salidas PWM, 8 entradas y/o salidas digitales, módulo bluetooth, conector Jack para alimentación, circuito para protección de entradas y salidas, Arduino Due, circuito para control de amplitud de las salidas PWM, circuito de conversión TTL para entradas digitales y conector molex.

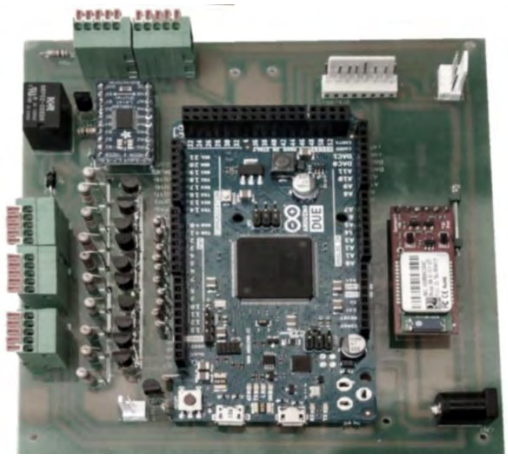


Figura 49. Implementación de la capa media

Capa inferior: En la capa inferior se conecta el módulo Xbee, sensor LM35 pegado al procesador, circuito de salidas AC, conector para salidas seriales y de protocolo I2C, procesador Vivante proporcionado por la tarjeta Udoo, módulo wi-fi, ventilador, conector USB para comunicación con

Matlab, botones de reseteo para el procesador y el microcontrolador, conector mini-USB para programación de la plataforma.

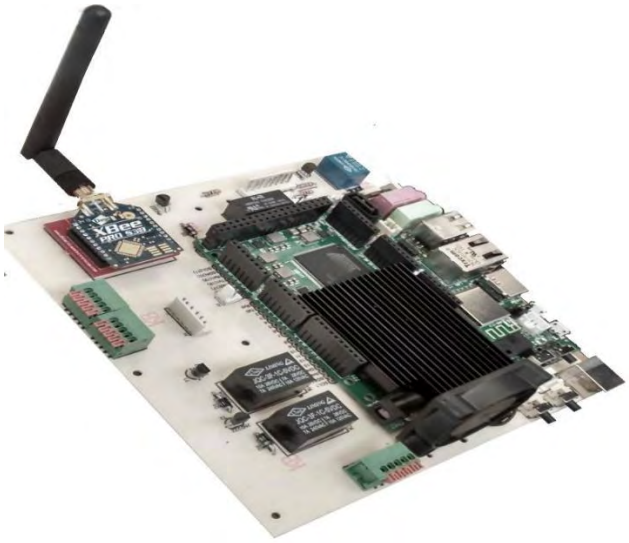


Figura 50. Implementación de la capa inferior

Finalmente la plataforma posee las siguientes características:

- 16 entradas analógicas, 8 protegidas y 8 no protegidas, 12 bits.
- 3 salidas para control de cargas AC
- 16 entradas y/o salidas digitales
- 10 salidas PWM con amplitud variable y protegidas contra grandes cargas
- 4 salidas analógicas de 12 bits
- 1 entradas serial
- 1 entrada para protocolo I2C
- Xbee pro de 900MHz
- Módulo Bluetooth RN41 (Smirf Gold)
- Circuito de control de temperatura para el procesador
- Procesador Freescale ARM
- LCD TFT, 3.2 pulgadas, táctil, marca 4D Systems
- Circuitos de protección para cada salida y/o entrada
- 1 conector para protocolo SPI

- Entrada Ethernet, módulo wi-fi, conector e-sata para disco duro
- Conexión para cámara, monitor, 2 entradas USB.
- Salida HDMI
- 2 microcontroladores ATSAM8X3E
- Memoria mini USB con sistema operativo LUBUNTU 12.04 modificado para sistemas de control y multimedia de programas instalados (Teslasoft, Processing, etc)

Se realizaron 2 dispositivos Tesla, en la Figura 51 se muestra los pines de salida y entrada, así como también parte la estructura del dispositivo Tesla final.

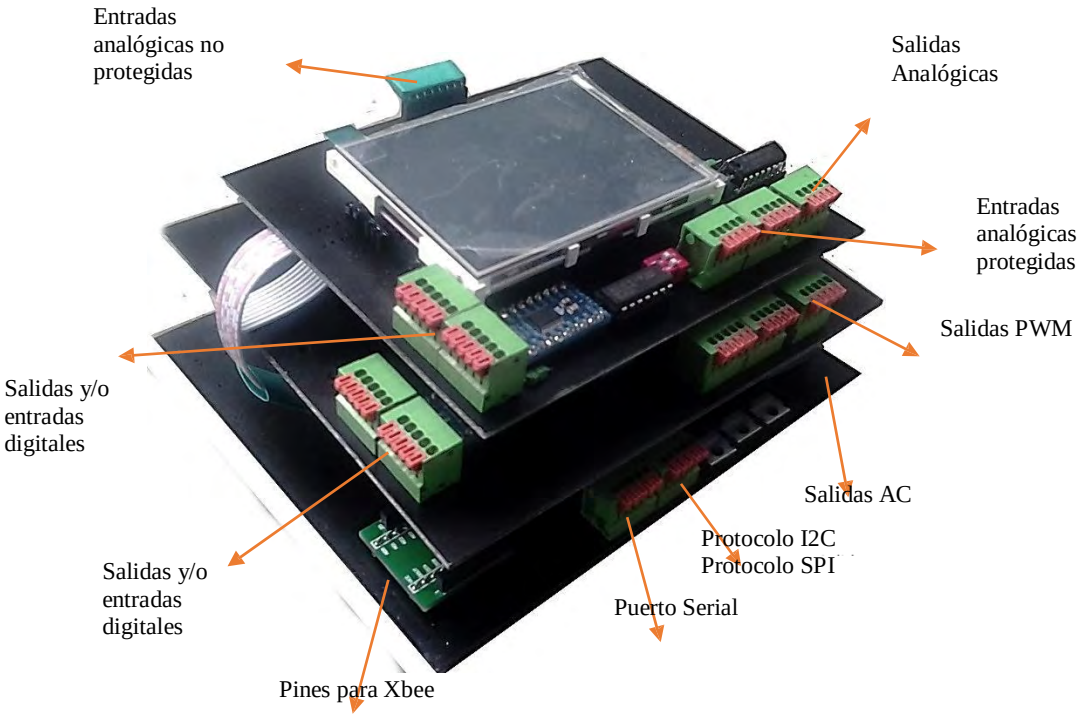


Figura 51. Estructura del dispositivo Tesla

III. LIBRERÍAS DE CONTROL

1. LIBRERÍA DE CONTROL PID

1.1. Bases conceptuales.

Un controlador PID computa una señal de control u(t) a partir del error entre la referencia y la salida de la planta de acuerdo a la fórmula (3), la cual depende esencialmente de tres partes: la parte proporcional, la parte integral y la parte derivativa. La Figura 52 muestra el controlador en conjunto de la planta y los diferentes bloques que contiene. El primer bloque (parte proporcional) consiste en el producto entre la señal de error y la constante proporcional. El segundo bloque (integral) es el producto entre la integral del error y una constante integral, este bloque pretende eliminar el error en estado estacionario y actuar cuando hay una desviación entre la variable y la referencia, integrando esta desviación en el tiempo y sumándola a la acción proporcional. El bloque derivativo consiste en el producto de la derivada del error y una constante y su función principal es la de acortar el tiempo de respuesta del proceso y estabilizar utilizando una acción predictiva basada en los cambios de la salida del proceso.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt} \qquad (3)$$

Donde, e(t) es el error de la señal, u(t) salida del controlador, Kp constante proporcional, Kd constante derivativa y Ki constante integral.

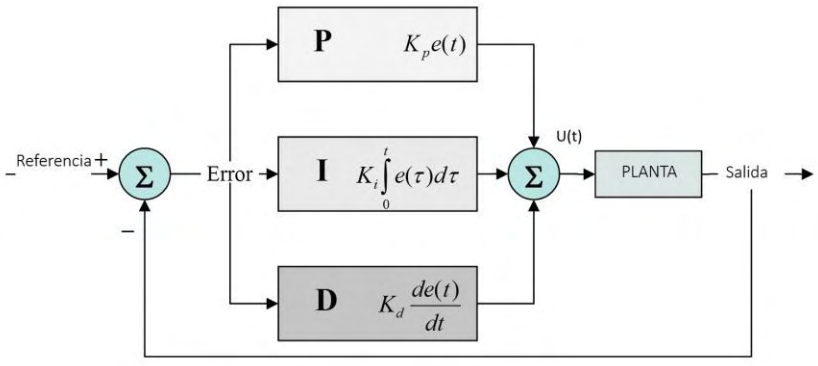


Figura 52. Control PID aplicado a una planta.

1.2 Entradas y Salidas de Algoritmo

El algoritmo de control se implementó en una librería del programa Teslasoft, la Figura 31 entrega la ruta para acceder a esta librería y los ejemplos, las entradas del algoritmo (variables o constantes que debe ingresar el usuario que usa la librería) son la referencia (el punto donde quiere que se establezca la salida de la planta) y la salida de la planta, que es valor de la medición hecha por el sensor. Por tanto el algoritmo se programará en la tarjeta Tesla, específicamente en el microcontrolador de desarrollo y la forma de programarlo está determinada por la librería para control PID almacenada en el programa Teslasoft. La salida de la planta es la variable que se quiere controlar, esta es medida por un sensor y su valor se transfiere al algoritmo. La salida del algoritmo es una señal de PWM o una señal de voltaje. La Figura 53 muestra las entradas y salidas del algoritmo.

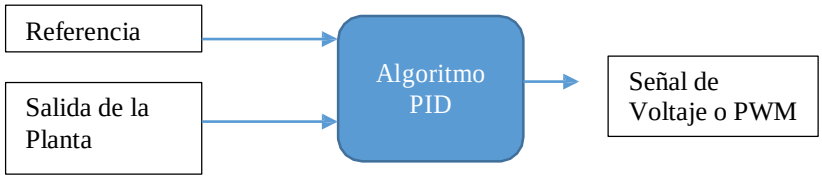


Figura 53. Entradas y Salidas del Algoritmo

1.3 Controlador y Planta

La plataforma Tesla es la encargada de ejecutar el algoritmo y generar una salida de PWM o de voltaje que se inyecta a un actuador y este actúa sobre la planta, el valor de salida de la planta deberá transferirse a la tarjeta para que esta realice el control del sistema. La Figura 54 muestra la estructura general del control. La referencia puede ser ingresada de forma exterior al dispositivo Tesla (con una señal analógica o digital) o ser ingresada internamente como una constante del algoritmo.

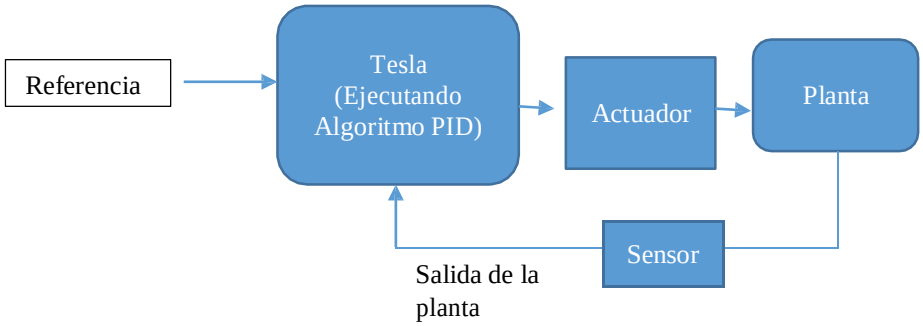


Figura 54. Estructura del control con la tarjeta Tesla

1.4 Consideraciones para la elaboración del algoritmo PID en lenguaje C.

La dificultad para crear un controlador PID digital radica en realizar el cálculo numérico de integrales y derivadas que se aproximen a la realidad. Por esta razón, se usaron algoritmos más sofisticados en los que se tienen en cuenta aspectos como saturación, cambios de sintonización y protección de sobre picos, entre otros.

Para el cálculo de la integral del error se hace uso de la regla de Simpson [20], la cual permite obtener una excelente aproximación a la integral real en comparación con métodos como: regla de trapecios, la regla del rectángulo y la del punto medio. Este cálculo se realiza por medio de la ecuación (4)

$$\int_{t_a}^{t_b} f(t)\,dt \approx \frac{b-a}{6}\left[f(a) + 4\cdot f\left(\frac{a+b}{2}\right) + f(b)\right] \quad (4)$$

Donde, ta es el tiempo inicial, tb el tiempo final

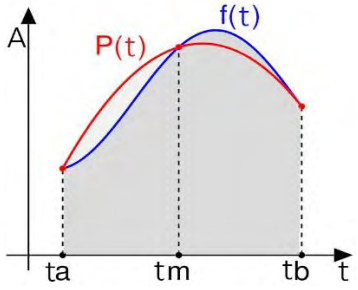


Figura 55. Integral real e integral aproximada, áreas bajo la curva.

.

Nótese en la Figura 55, que al aplicar cualquier método, finalmente el objetivo final es encontrar el área bajo la curva de una función dependiente del tiempo. Para el caso discreto una integral en el tiempo se puede aproximar usando la regla de Simpson por medio de la expresión (5). El producto de esta expresión con la constante integral define la parte integral. En lenguaje C, basta con almacenar el error presente y el error pasado, el termino $\left(\frac{\text{error}_{t-1}+\text{error}_t}{2}\right)$ es el promedio de estos dos errores y el término error_{t-1} (Δt representa el tiempo de muestreo del cálculo, el cual es constante (por defecto el tiempo es de 10ms), también es configurable por el usuario, la TABLA VIII muestra la función para cambiar de tiempo de muestreo.

$$\int f(x)dx \approx \sum_{1}^{n} \frac{x_n - x_{n-1}}{6}\left[f(x_{n-1}) + 4 \cdot f\left(\frac{x_{n-1} + x_n}{2}\right) + f(x_n)\right] \quad (5)$$

Esta expresión es usada en la librería, permitiendo mejores cálculos de la integral.

Para la derivada del error se utiliza el método de diferencias hacia atrás, definido por la ecuación (6)

$$f'(x) \approx \frac{f(x) - f(x - \Delta t)}{\Delta t} \quad (6)$$

Que reemplazando la función por el error del sistema, se tiene

$$e'(x) \approx \frac{e(x) - e(x - \Delta t)}{\Delta t} \quad (7)$$

Estableciendo que el error es la diferencia entre la referencia (r(t)) y la salida de la planta (o(t)), reemplazando el error se obtiene:

$$e'(x) \approx \frac{r(x_n) - o(x_n) - (r(x_{n-1}) - o(x_{n-1}))}{x_n - x_{n-1}} \quad (8)$$

$$e'(x) \approx \frac{o(x_{n-1}) - o(x_n)}{x_n - x_{n-1}} \quad (9)$$

Con lo cual se establece que la derivada del error depende de la salida de la planta y del tiempo de muestreo, es decir de Δt, este tiempo es constante y configurable por el usuario y es el mismo tiempo para el cálculo de la integral. Basta con multiplicar la expresión (9) con la constante derivativa y se tiene la parte derivativa.

La saturación de un control PID es importante ya que la salida va a ser PWM o una salida analógica, el ciclo útil y valores analógicos pueden manejar valores de 0 a 4096, por tanto, es necesario saturar la salida a estos valores ya que representan los extremos.

Algoritmos PID digitales hay de múltiples tipos, uno de los aportes importantes en este aspecto son las consideraciones de Brett Beauregard [21], este trata aspectos importantes de un PID a nivel industrial y elimina bajo técnicas de programación en lenguaje C los principales problemas, no tiene especial consideración en el cálculo de derivadas e integrales con exactitud y se centra en eliminar los siguientes problemas:

Windup**:** El efecto windup aparece al arrancar el sistema o en cualquier otra situación, donde aparece un error muy grande durante un tiempo prolongado. Esto hará que el término integral aumente para reducir el error, por lo que valores de PWM y de salida analógica se saturarán a sus valores máximos. Cuando el error se reduce, la parte integral también comenzará a reducirse, pero desde un valor muy alto, llevando mucho tiempo hasta que logre la estabilidad, generando fluctuaciones exageradamente grandes. El problema se manifiesta en forma de retrasos extraños de la salida respecto a la referencia. El problema se indica en la Figura 56.

Este problema se genera debido a que controles PID presentan una salida PWM o analógica que varía en un rango finito, el control PID puede enviar una señal de control muy grande (ante grandes variaciones de la referencia o perturbaciones de planta) la cual satura los valores máximos de PWM y señal analógica que puede proporcionar cualquier microcontrolador o sistema digital, esto produce un considerable retraso de la salida de la planta respecto la referencia. La solución práctica (planteada por Brett) ante este efecto es limitar: la señal de control, el valor que puede tomar la parte integral y la parte derivativa con un rango máximo dado por el valor máximo de ciclo útil que puede proporcionar una señal PWM o una señal analógica(para el caso de Tesla se limita entre 0 y 4096, que son los rangos máximos).

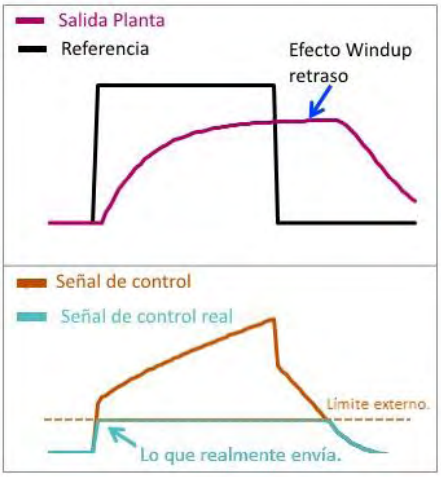


Figura 56. Efecto Windup en un control PID

Cambios de Sintonización: A nivel industrial es necesario el cambio en los parámetros de sintonización (cambios en las constantes principales: Kp,Ki y Kd), el cambio repentino de sintonización provoca pequeños sobresaltos o baches en la señal de control y la señal de salida de la planta, el efecto se muestra en la Figura 57. En la práctica este efecto se corrige con la introducción

de la constante integral en la parte interna de la integral del error, con lo cual no se realiza nada erróneo, este cambio elimina los baches de la Figura 57 (de acuerdo al análisis numérico de Brett), esta deducción puede consultarse en [21].

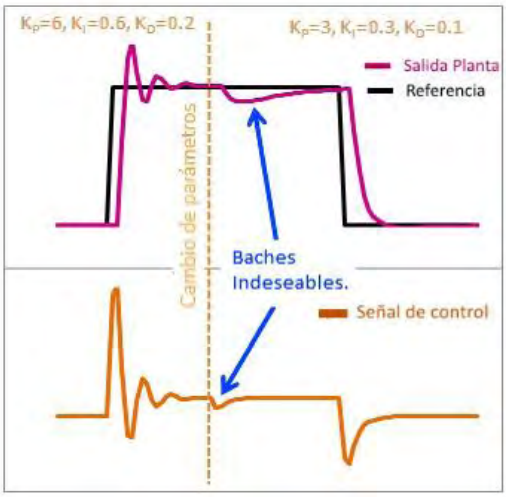


Figura 57. Efecto del cambio de sintonización en un control PID

Otros efectos que se analiza son el comportamiento de un PID en controles on/off, procesos industriales que son directos (cuando se aumenta la señal de control aumenta la salida de la planta) o indirectos (cuando se aumenta la señal de control disminuye la salida de la planta) y picos generados por la derivada; a todos estos efectos negativos se les brinda una solución traducida en lenguaje C en la guía [21]. Estas consideraciones son tomadas para la librería de la plataforma Tesla y en unión con el cálculo más exacto de la derivada e integral se realiza un algoritmo robusto que puede ser aplicado en procesos industriales.

1.5 Algoritmo PID

El algoritmo traducido en diagrama de flujo se presenta en la Figura 58, inicialmente se establecen dos variables de tiempo auxiliares: t1=0 y t2=0, la variable "tiempo" es el tiempo que corre durante todo el proceso y "tm" es el tiempo de muestreo. Nótese que el cálculo de la integral y la derivada de acuerdo a la fórmula (5) y (9) es más fácil debido a que el Δt es constante e igual al tiempo de muestreo, ya que el cálculo se realiza únicamente cuando se iguala o se supera el tiempo de muestreo.

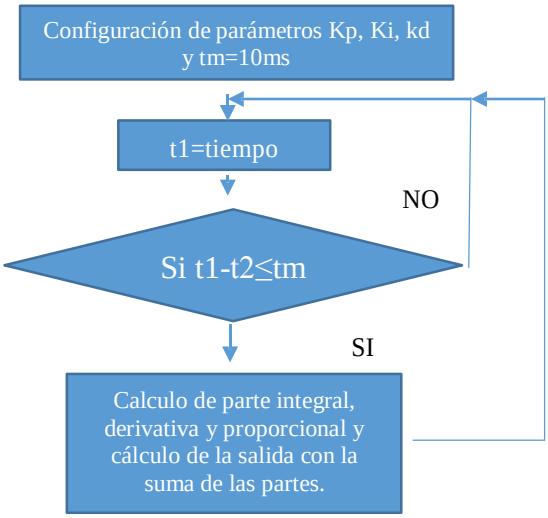


Figura 58. Algoritmo en diagrama de flujo implementado para el programa Teslasoft

1.6 Funciones de la Librería PID

Cabe resaltar que la librería PID está almacenada en el programa Teslasoft, con el uso de esta librería se puede programar un algoritmo PID dentro de la plataforma Tesla, dentro de la librería hay ejemplos de baja y mediana complejidad, las sentencias que se pueden usar con el lenguaje de programación de Tesla son las siguientes:

Tabla 8 Funciones y sentencias de la librería pid

Sentencia	Función
PID PID	Crea un objeto PID llamado
PID.sintonizacion(a, b, c, kp, ki, kd, p)	Realiza la sintonización del PID, a es la valor de entrada del controlador, b es nombre de la variable por donde el controlador saca el valor final, c es la referencia; kp, kd y ki son las ganancias y p es una variable booleana que indica si el proceso es directo o inverso, con uno coloca todas las ganancias positivas, con 0 coloca todas las ganancias negativas.
calculo()	Calcula la señal de salida, aplicando el algoritmo de la ecuación (12) y entrega en la variable b de la anterior

	sentencia el valor de salida del controlador, el cual puede ser inyectado como PWM o salida analógica a la planta.
setTiempoMuestreo(t)	Cambia el tiempo de muestreo a cualquier valor en milisegundos, por defecto está configurado el algoritmo a 10 ms
Satuacion(a,b)	Establece los parámetros de saturación de la señal de salida PWM o analógica, parte integral y par derivativa en el rango (a,b)
Directividad(a)	Establece si el control es directo o inverso, directo significa que las constantes siempre van a ser positivas e inverso las constantes las vuelve negativas, si la variable booleana "a" es 0 significa proceso directo, en caso contrario el proceso es inverso

1.7 Ejemplo de uso

Un ejemplo sencillo del uso de la librería es el mostrado en la TABLA IX, donde se evidencia que implementar un algoritmo PID con el uso de Teslasoft y la librería es relativamente sencillo, y un control de temperatura por ejemplo, llevaría solo 10 líneas de código implementarse(lo que normalmente implica más de 100 líneas de código). Este ejemplo es un control de temperatura en donde se recibe la temperatura como una lectura analógica del sensor de la planta, se calcula la señal de control a partir de las constantes de sintonización (KP=2, Ki=5 y Kd=1), esta señal de control es una salida PWM limitada en el rango de 0 a 4096.

Tabla 9. Código pid implementado con la plataforma Tesla y el uso de la Libreria

Líneas	Función
1	PID Controlador(&Entrada, &Salida, &Referencia,2,5,1, DIRECTO); / /parámetros de inicio del PID
2	void setup() {
3	TeslaBegin(12,12); // inicializa todos los componentes de la tarjeta tesla
4	Entrada = LecturaAnalogica(1); //la entrada del controlador es una lectura analógica de la planta
5	Referencia = 1000; //referencia en 1000
6	Controlador.Modo(AUTOMATICO); }
7	void loop() {
8	Entrada = LecturaAnalogica(1);
9	Controlador.Calcular(); //Se calcula la salida del PID con los parámetros establecidos
10	EscrituraPWM(3,Salida); }

2. LIBRERÍA DE CONTROL FUZZY

2.1 Bases Teóricas

Los controladores fuzzy, igual que otros controladores, toman el valor de las variables de entrada, procesan, y actúan sobre sus salidas a fin de controlar la planta. En fuzzy control se utilizan sistemas basados en reglas, que emplean como elemento central una lógica basada en la experiencia. Estos controles toman los valores de las variables de entrada, evalúan la veracidad de cada regla, y así, toman decisión sobre los cambios a realizar en las variables de salida. Una vez actualizadas las mismas, estas producirán un cambio sobre la planta, luego se vuelven a obtener los valores de las variables de entrada, comenzando un nuevo ciclo. La utilidad de este tipo de control radica en que no es necesario modelar una planta y procesar ecuaciones complejas sino reglas como: "SI la temperatura es baja ENTONCES aumente el ciclo de actividad del calefactor". El proceso de implementar un controlador fuzzy se basa en 3 partes: fuzzificación, evaluación de las reglas y defuzzificación, cada parte puede observarse en la Figura 59.

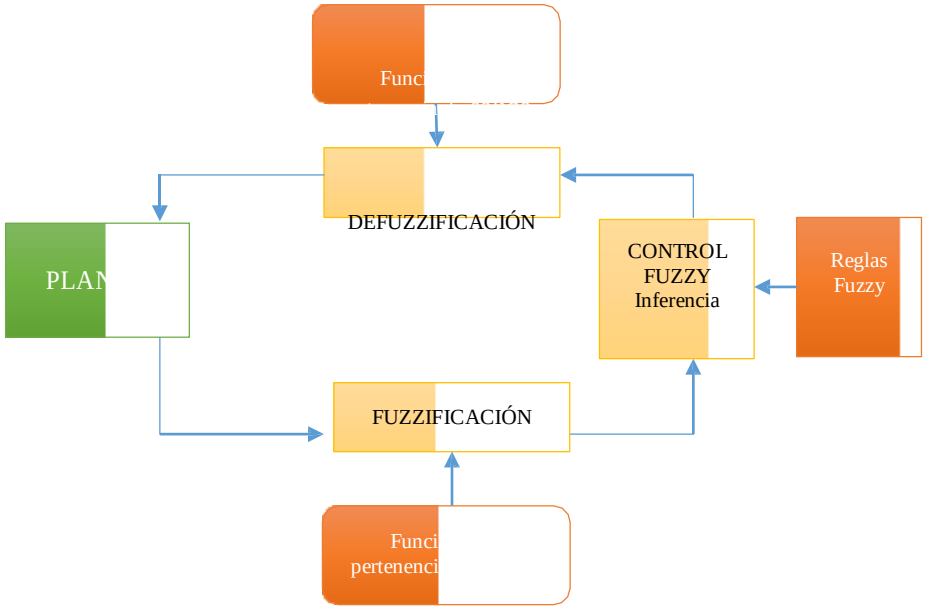


Figura 59. Sistema de control Fuzzy con realimentación

Para entender el proceso de un controlador fuzzy se procesa un ejemplo y se agregan definiciones de acuerdo al avance en el proceso. El ejemplo que se toma es un sistema de regadío, en donde el control fuzzy tendría por finalidad controlar el tiempo de regadío de la superficie de un terreno dependiendo de la temperatura del aire y de la humedad del suelo, por tanto, las variables de entrada

del controlador son (humedad y temperatura) y variable de salida del controlador es (tiempo de regadío).

Proceso de Fuzzificación: El primer paso en lógica difusa es convertir la señal proveniente de la planta (señal x, está también puede ser la señal de error del sistema de control) en un conjunto de variables difusas. Esto se hace asignando valores (que serán valores difusos) a partir de un conjunto de funciones de pertenencia (funciones de membresía). Los valores de cada función de pertenencia se etiquetan generalmente con μ(x), y son determinados por el valor de la señal x y la estructura de las funciones de membresía de entrada. La Fuzzificación divide los rangos posibles en los cuales puede clasificarse el valor de la señal x. Las funciones de pertenencia pueden ser de múltiples formas (trapezoidal, triangular, singleton, etc), una forma clásica es la mostrada en las Figura 61. La parte que corresponde a este proceso se indica en la Figura 60.

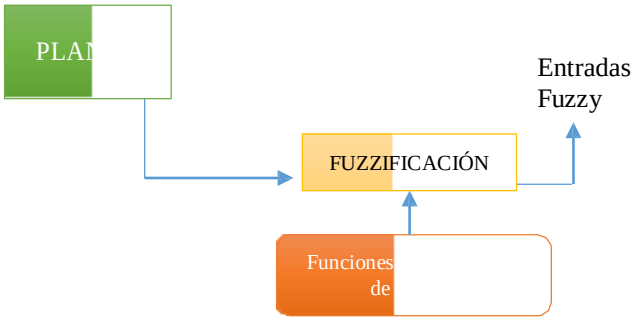


Figura 60. Proceso de Fuzzificación.

Para el ejemplo de estudio se realiza esta etapa, por tanto, el paso inicial es asignar etiquetas para cada variable de entrada al controlador (salida de la planta). Por ejemplo para la temperatura las etiquetas serían: congelado (CO), frio (FR), normal (NO), tibio (TI), caliente (CA); para humedad: seca (SE), húmeda (HU) y mojada (MO). A cada etiqueta se asigna una función de pertenencia, tal como muestra la Figura 61. Luego para cada variable de entrada (salida de la planta) se le asigna un valor de pertenencia a cada etiqueta de acuerdo a las funciones de membresía. Por ejemplo de acuerdo a las funciones de pertenencia de la Figura 61 y asumiendo que al proceso de fuzzificación le entran valores de 33 ºC y 11% de las variables de entrada temperatura y humedad respectivamente (provenientes de la planta), los valores de pertenencia para cada etiqueta serían:

La temperatura es tibia (TI) con un grado de verdad de 0.2: en resumen, $\mu_{TI}=0.2$.
La temperatura es caliente (CA) con un grado de verdad de 0.46, $\mu_{CA}=0.46$.

La humedad es seca (SE) con un grado de verdad 0.25, μ_{SE}=0.46.

La humedad es húmeda (HU) con un grado de verdad 0.75, μ_{HU}=0.75

Estos valores son las entradas fuzzy, útiles para el mecanismo de inferencia, el cual es el siguiente paso.

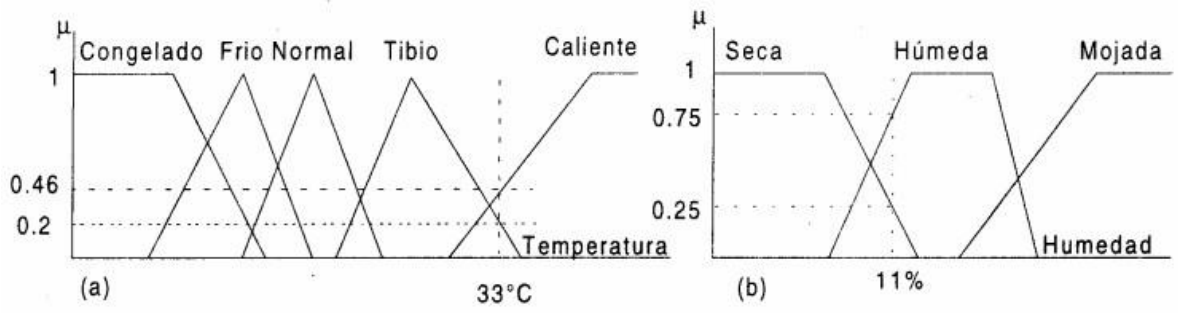


Figura 61. Funciones de Pertenencia para las variables: humedad y temperatura.

Evaluación de reglas (Mecanismo de Inferencia): Esta es la segunda etapa del proceso, en donde el controlador usa reglas lingüísticas (reglas fuzzy) sobre los resultados que fueron generados en la etapa de fuzzificación (denominados entradas Fuzzy) para generar unas salidas fuzzy; esta etapa del proceso corresponde a la Figura 62. Las reglas fuzzy son sentencias de tipo: Si-Antecedente Entonces Consecuente, describen la acción a ser tomada en respuesta a varias entradas fuzzy. La sintaxis de las reglas es la siguiente:

"SI antecedente1 AND antedecente2… ENTONCES consecuente1AND consecuente2…"

En donde el antecedente corresponde a las entrada fuzzy y el consecuente corresponde a la deducción a partir de los antecedentes y el operador, AND es un operador que comparara entre los valores de los antecedentes y toma el mínimo (en ocasiones es la multiplicación de los valores de los antecedentes), también puede presentarse el operador OR (que toma el máximo de la comparación) y el NOT(que niega un antecedente realizando la operación 1-μ_{xx}), los operadores en los consecuentes funcionan de la misma forma que en los antecedentes.

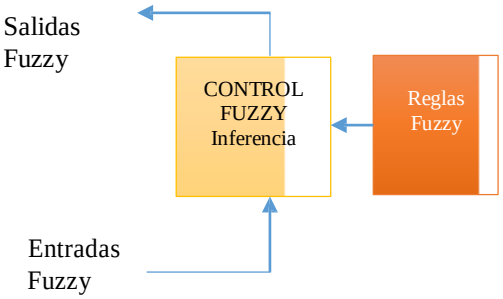


Figura 62. Evaluación de reglas, segunda fase del proceso de un control Fuzzy

Es importante establecer las etiquetas para la variable de salida, para el ejemplo en desarrollo la variable de salida es el tiempo de regadío y para este se le establecen las etiquetas: corto (CO), medio (ME) y prolongado (PRO). Por tanto, el conjunto de reglas se establece de acuerdo a la cantidad de etiquetas del antecedente1 y antecedente2, la Figura 63 muestra: antecedentes y consecuentes del ejemplo.

ANTECEDENTE 2	HUMEDAD	ANTECEDENTE 1: TEMPERATURA				
		Congelado	Frio	Normal	Tibio	Caliente
	Mojada	Corto	Corto	Corto	Corto	Corto
	Húmeda	Corto	Medio	Medio	Medio	Medio
	Seca	Prolongado	Prolongado	Prtolongado	Prolongado	Prolongado

Figura 63. Matriz de reglas de Inferencia

De acuerdo al ejemplo, existirían 15 reglas producto del cruce entre etiquetas de la variable de entrada temperatura y las etiquetas de la variable de entrada humedad. Para ilustrar el proceso se toman los antecedentes calculados en la etapa de fuzzificación y se establecen los antecedentes, por tanto, las reglas (parte de las reglas del ejemplo en proceso) quedarían así:

R1: SI la temperatura es caliente (CA) AND la humedad es seca (SE), ENTONCES la duración es prolongada (PRO)

R2: SI la temperatura es tibia (TI) AND la humedad es húmeda (HU), ENTONCES la duración es media (ME)
R3: Si la temperatura es tibia (TI) AND la humedad es seca (SE), ENTONCES la duración es prolongada (PRO)
R4: Si la temperatura es caliente (CA) AND la humedad es húmeda (HU), ENTONCES la duración es media (ME)

En base a las reglas, el mecanismo de inferencia deduce una salida Fuzzy (tal como indica la Figura 62), por tanto en base a las 4 reglas planteadas se induce una salida fuzzy, la cual se determina con la TABLA X

Tabla 10. Reglas y grado de verdad

Nro.	Regla	Grado de verdad
R1	SI temperatura es CA (0.46) AND humedad es SE (0.25) la duración es prolongada	0.25
R2	SI la temperatura es TI (0.2) AND la humedad es HU (0.75) la duración es media	0.2
R3	SI la temperatura es TI (0.2) AND la humedad es SE (0.25) la duración es prolongada	0.2
R4	SI temperatura es CA (0.46) AND humedad es HU (0.25) la duración es media	0.46

Nótese que la regla 1 y la regla 3 establecen un mismo consecuente, al igual que las reglas 2 y 4, cuando se ordena un mismo consecuente se escoge el de mayor peso. La cantidad de salidas Fuzzy es igual al número de etiquetas de la variable de salida, para el ejemplo serán 3 salidas Fuzzy determinadas por:

La duración de regadío es prolongado (PRO) con un grado de verdad de 0.25
La duración de regadío es media (ME) con un grado de verdad de 0.46
La duración de regadío es corto (CO) con un grado de verdad de 0

Defuzzificación: El proceso de defuzzificación consiste en traducir los valores de las salidas fuzzy en una señal de control para la planta por medio del uso de funciones de membresía(de salida), por tanto, es necesario tener una función de membresía por cada etiqueta(PRO,ME y CO) de la variable de salida (en el ejemplo, tiempo de regadío), el grado de pertenencia de las salidas fuzzy con cada función de pertenencia influye en la altura del área que se toma de la función de pertenencia(para

realizar el cálculo final). La Figura 64 muestra la relación de las salidas fuzzy respecto a cada función de pertinencia. Para determinar la salida de la planta se puede utilizar la técnica del centroide descrito por la ecuación (10)

$$\text{[illegible]} = \frac{\sum_1^{\text{[illegible]}} \text{[illegible]}}{\sum_1^{\text{[illegible]}} \text{[illegible]}} \qquad (10)$$

Donde [illegible] son las salidas fuzzy(para el ejemplo: 0; 0.25; 0.46) y el término corresponde a los puntos centrales (primer máximo) de la funciones de membresía, para el ejemplo son (7.5, 27.5 y 57,5), al hacer el cálculo el valor de salida(que corresponde a la señal de control de la planta) será de 38.

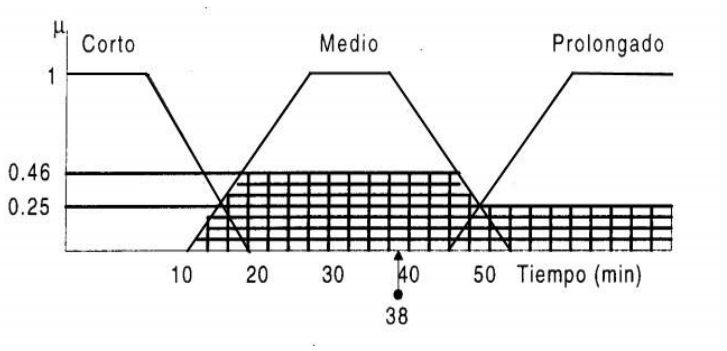


Figura 64. Funciones de pertenencia para Defuzzificación.

El proceso de defuzzificación corresponde al último paso que se hace, esta parte del proceso se muestra en la Figura 65, donde se evidencia que las Salidas Fuzzy son convertidas en una Señal de control a partir del uso de funciones de pertenencia.

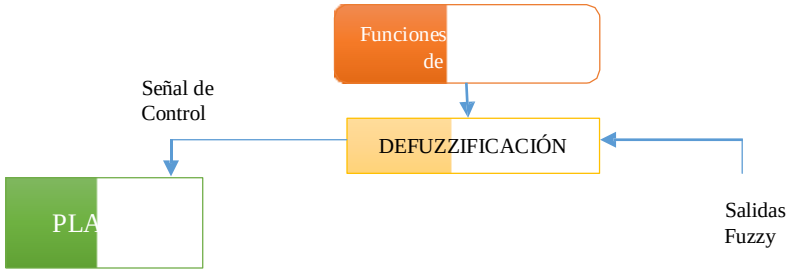


Figura 65. Proceso de Defuzzificación.

2.2 Entradas y salidas de algoritmo

El algoritmo de control se implementó en una librería del programa Teslasoft, la Figura 31 entrega la ruta para acceder a esta librería y los ejemplos, el algoritmo recibe una o varias señales de la planta (que pueden ser sensadas) o también una referencia y una señal de salida de la planta. El algoritmo recibe estas señales y a partir de los parámetros de configuración ingresados por el usuario (reglas fuzzy, etiquetas, funciones de membresía de entrada y salida, variables de entrada y salida) aplica: fuzzificación, mecanismo de inferencia y defuzzificación, para entregar una señal control. La señal de control (entregada por el algoritmo) no está limitada, ya que su valor depende de la forma y valores de las funciones de membresía, por tanto, el usuario debe tener especial cuidado en escoger adecuadamente estas funciones tanto para el proceso de fuzzificación como para el proceso de defuzzificación. La señal de control posteriormente puede ser transformada en señal PWM o señal analógica. El algoritmo se puede extender a otros usos aparte de sistemas de control, ya que simplemente aplica lógica fuzzy, por tanto puede ser usada en aplicaciones que impliquen toma de decisiones, clasificación de variables y cualquier uso que tenga que ver con lógica fuzzy. La Figura 66 muestra las entradas y salidas del algoritmo.

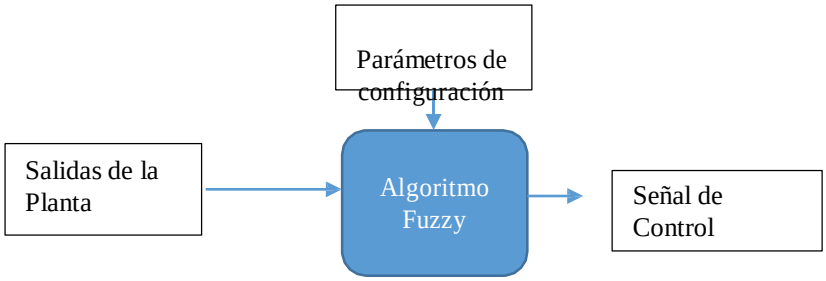


Figura 66. Entradas y Salidas del Algoritmo

2.3 Consideraciones para la elaboración del algoritmo Fuzzy en lenguaje C.

La principal dificultad de una librería de este tipo es la creación de las funciones de membresía o pertenencia para esto se construye un método en el cual el almacenamiento de 4 valores en un vector representa la geometría de una función de pertenencia. Por ejemplo la Figura 67 posee una

función de membresía cuya representación geométrica se basa en el almacenamiento del vector [10, 20, 30, 40]. La TABLA XI presenta las formas de crear las diferentes formas de las funciones de membresía para el proceso de fuzzificación y defuzzificación.

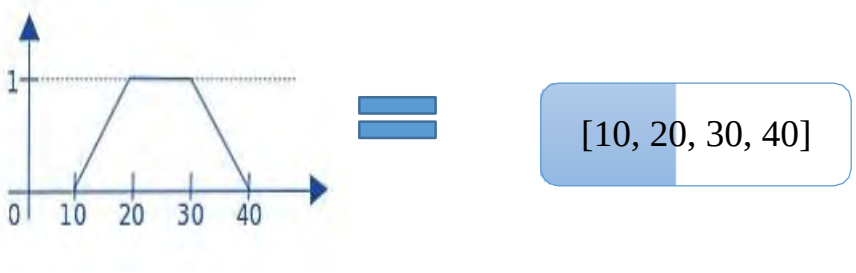
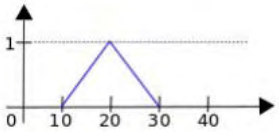
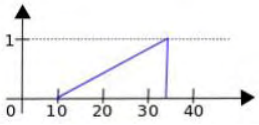
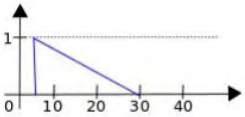
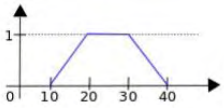
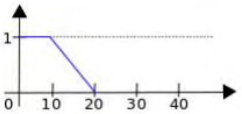
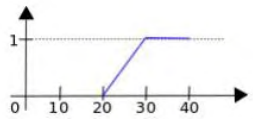
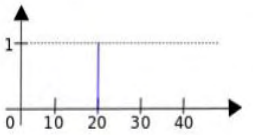


Figura 67. Construcción de funciones de membresía a partir de vectores.

Tabla 11. Creación de funciones de membresía

Tipo de Función	Sentencia	Resultado gráfico
Triangular.	FuzzyEtiqueta* f1 = FuzzyEtiqueta(10, 20, 20, 30);	
Rampa	FuzzyEtiqueta* f2 = FuzzyEtiqueta(10, 33, 33, 33);	
Rampa invertida	FuzzyEtiqueta* f3 = FuzzyEtiqueta(5, 5, 5, 30);	
Trapezoidal	FuzzyEtiqueta* f4 = FuzzyEtiqueta(10, 20, 30, 40);	
Rampa inicial	FuzzyEtiqueta* f5 = FuzzyEtiqueta(0, 0, 10, 20);	

Rampa final	FuzzyEtiqueta* f6 = FuzzyEtiqueta(20, 30, 40, 40);	
Escalón unitario Singleton	FuzzyEtiqueta* f7 = FuzzyEtiqueta(20, 20, 20, 20);	

Para el proceso de fuzzificación se tienen en cuenta las funciones de membresía de entrada creadas por el usuario(a partir del uso de la TABLA XI) y se deducen las entradas fuzzy a partir de cálculos de geometría básica (de la misma forma que el ejemplo planteado). Para la parte del mecanismo de inferencia se tiene en cuenta que hay operadores AND y OR (que depende de la selección del usuario), los cuales toman el valor máximo o mínimo de dos antecedentes y con esto generan las salidas Fuzzy, en caso de presentarse dos reglas que ordenan el mismo consecuente se escoge el valor de mayor peso, el planteamiento de las reglas fuzzy por parte del usuario en lenguaje C, es intuitivo y se basa principalmente en la siguiente función:

“[illegible]* [illegible]1 = [illegible](1, [illegible])”

El cálculo de la señal de control (salida de controlador) se realiza mediante el método del centroide, es decir, basado en la ecuación (9). Para implementar un algoritmo haciendo uso de la librería Fuzzy creada en lenguaje C basta con seguir los pasos (marcados en azul) ordenados en la Figura 68, la parte en azul es el proceso que se ejecuta automáticamente y la parte en gris son los parámetros de configuración, por lo cual esta parte se ejecuta una sola vez en el algoritmo.

Figura 68. Proceso de configuración y ejecución del algoritmo

2.4 Funciones de la librería

Las sentencias de código para la librería Fuzzy se presentan en la TABLA XII.

Tabla 12. Funciones y sentencias de la librería para dinámica de replicadores

Sentencia	Función
Fuzzy Fuzzy	Este objeto incluye todo el sistema Fuzzy, a través de él, se puede manipular los conjuntos difusos, reglas lingüísticas, entradas y salidas.
EntradaFuzzy variable1	Se usa para declarar una variable de entrada
SalidaFuzzy variable2	Se utiliza para declarar una variable de salida

FuzzyEtiqueta etiqueta1	Se usa para crear etiquetas fuzzy para las variables de salida o entrada, también permite la creación de funciones de membresía: triangulares, trapezoidales y singleton, basados en los puntos A, B, C y D, teniendo en cuenta la siguiente sentencia de uso: "FuzzyEtiqueta (float a, float b, c float, float d)".
ReglaFuzzy regla1	Se usa para establecer una regla fuzzy a partir de un antecedente y consecuente de la siguiente forma: "ReglaFuzzy r1 = new ReglaFuzzy (ID, antecedente, consecuente)", ID representa la numeración o identificación de la regla.
ReglaFuzzyAntecedente antecedente1	Se usa para declarar el antecedente que será usado en una regla Fuzzy declarada con la anterior sentencia
ReglaFuzzyConsecuente consecuente1	Es una sentencia que forma parte del montaje de una regla, a partir de la declaración del consecuente de esa regla

2.5 Ejemplo de aplicación

El ejemplo de aplicación desarrollado en Teslasoft establece un sistema de frenado para un carro que se dirige directamente hacia un obstáculo, el vehículo tiene incorporado un sonar, con el cual calcula la distancia que hay del vehículo al obstáculo, esta distancia es enviada al algoritmo de Tesla y el dispositivo entrega una señal de PWM que va disminuyendo a medida que el vehículo se acerca al obstáculo. La Figura 69 muestra la planta y el proceso que se quiere controlar, el código del ejemplo se puede encontrar dentro de la librería de Fuzzy que posee el programa Teslasoft, el cual se anexa al final, este incluye paso a paso la estructura de implementar un control por medio de lógica fuzzy, posee comentarios y secciones divididas para guiar al usuario en la implementación.

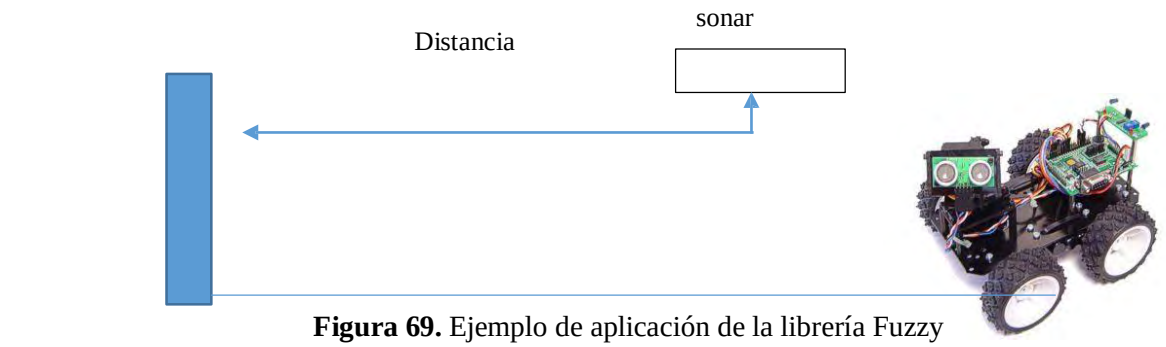


Figura 69. Ejemplo de aplicación de la librería Fuzzy

3. LIBRERÍA DE DINÁMICA DE REPLICADORES

3.1. Bases Conceptuales

La ecuación de dinámicas de replicadores describe cómo ciertas características de comportamiento de una población de individuos evolucionan a través de: la selección natural, interacciones mutuas y bienestar [22]. La dinámica de replicadores hace referencia al comportamiento de poblaciones de individuos que se descomponen en subgrupos en donde cada individuo escoge una estrategia que maximice su bienestar o "*fitness*". Los pagos en este juego representan el efecto de incrementar su función de *fitness* que depende tanto de la estrategia escogida como de la que escogen los demás individuos. El estado de la población corresponde al número de individuos en los diferentes subgrupos. Las dinámicas de los replicadores buscan que las reparticiones en los subgrupos sean estables en el tiempo [24] comparando su propia *fitness* con la *fitness* promedio de toda la población, en un conjunto de ecuaciones diferenciales de primer orden dadas por:

$$\dot{x}_i = \frac{x_i}{P_x}(f_i(x) - \bar{f}(x)) \qquad (11)$$

para i= 1, 2, …, N, donde N representa el número de estrategias puras, $x_i \geq 0$ es la cantidad de individuos jugando la estrategia i, $x = [x_1, \ldots, x_N]^T$ es el estado de la población, y P_x es el total de la población **¡Error! No se encuentra el origen de la referencia.**. Asumiendo que el número de jugadores en la población es grande como para aproximar la cantidad de individuos como una variable continua, las dinámicas de replicador están dadas por (12), donde $f_i(x)$ representa la función *fitness* percibida en el i-ésimo subgrupo y $\bar{f}(x)$ es el *fitness* promedio de la población dado por:

$$\bar{f}(x) = \frac{1}{P_x}\sum_{i=1}^{N} x_i f_i(x) \qquad (12)$$

El principio fundamental de este concepto consiste en que la población después de un proceso evolutivo, tiende a buscar un punto de equilibrio donde todos los individuos alcanzan el mismo *fitness* (13). En este sentido, cada individuo intenta maximizar su rentabilidad con la influencia de las estrategias de otros mientras que un bienestar social común se obtiene para la población total **¡Error! No se encuentra el origen de la referencia.**.

$$\dot{x}_i = \frac{\partial}{\partial x_i}(f_i - \bar{f}) = 0 \rightarrow f_i = \bar{f} \qquad (13)$$

Un replicador es una unidad fundamental en el proceso evolutivo, y es análogo a un individuo en una población. Los sistemas replicadores son fundamentales para muchos procesos naturales, y han sido usados para modelar diferentes tipos de sistemas. Estos sistemas son universales precisamente porque capturan las dinámicas esenciales de los sistemas que contienen múltiples agentes cooperando y compitiendo. En efecto, cualquier modelo de ecuaciones diferenciales de primer orden puede ser relacionado con un modelo replicador.

Para el caso discreto, la ecuación de la dinámica de replicadores se establece a partir de una ecuación en diferencias determinada por la ecuación (14).

$$x_i(t) = x_i(t-1)\frac{\alpha + f_i(t-1)}{\alpha + \bar{f}(t-1)} \qquad (14)$$

Donde

$$f_i = f + f_i - f_i$$
$$\bar{f} = f$$
$$f_i = \frac{1}{P_{ij}} \sum_{j=1} f_{ij} * f_{ij}$$

α es un parámetro ajustable que regula la razón de cambio de las estrategias, fij es la ecuación de "fitness" y $\bar{f}$ es el fitness promedio que tiene en cuenta a toda la población para el cálculo.

3.2 Entradas y salidas del algoritmo

En el programa Teslasoft se desarrolló una librería basada en dinámica de replicadores, con la cual se puede desarrollar algoritmos de este tipo. La salida del algoritmo son señales de PWM o voltaje que varían dentro del rango de [0-4096], ya que estos son los valores máximos del conversor DAC y de ciclo útil.

La librería se centra en el control de n zonas que posean: n referencias y n valores de pago. Para esto el usuario debe ingresar un vector de tamaño n para: referencias (Vr), señales de salida de la planta (Vss) (entrada del controlador) y pagos iniciales (Vpi), además de los parámetros de

configuración inicial como: B y Ptot. La Figura 70 muestra las entradas de acuerdo a lo planteado.

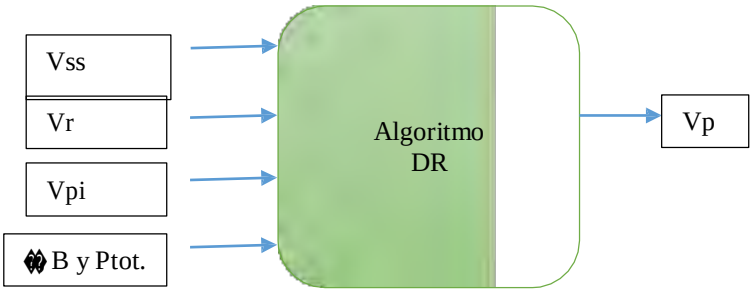


Figura 70. Entradas y salidas del algoritmo por dinámica de replicadores

El vector Vr, Vpi y las constantes B y Ptot. Se ingresan una única vez, pero el vector Vss que representa el conjunto de variables que son medidas por los sensores en la planta (señales de salida de la planta) deben estarse ingresando constantemente al algoritimo. La función del algoritmo consiste en actualizar el vector de pagos Vp (que serían las señales de control para la planta) a partir del vector Vr, Vpi, Vss y las constantes B y Ptot. El dispositivo Tesla será el controlador de la planta y en su interior correrá el algoritmo previamente configurado por el usuario. La Figura 71 muestra la estructura controlador-planta para aplicar el algoritmo por dinámica de replicadores.

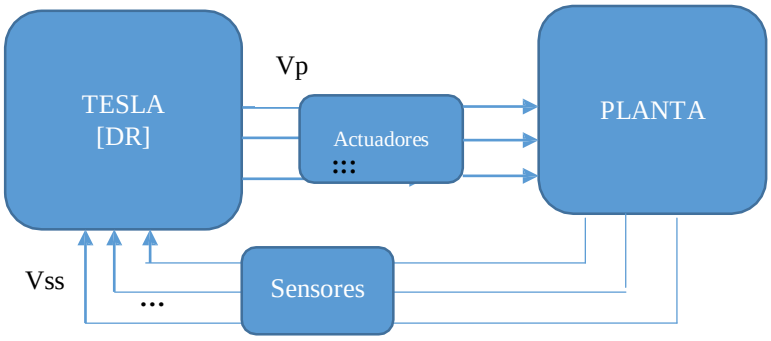


Figura 71. Estructura del controlador y la planta

3.3 Implementación del algoritmo en lenguaje C

Se establece una librería la cual ejecuta un conjunto de cálculos matriciales para obtener un vector de pagos Vp a partir de unas entradas Vss y parámetros de configuración. El proceso que ejecuta la librería se resume en los siguientes pasos ordenados.

- Se realiza el cálculo del bienestar o "fitness" a partir de la ecuación (14) usando: el vector Vss, el vector Vr y la constante B.
- Para la primera iteración se realiza el cálculo del "fitness" promedio teniendo en cuenta el anterior resultado, el vector Vpi y la constante Ptot. Para iteraciones posteriores solo se tiene en cuenta el vector Vp (Vpi ya no se tiene en cuenta), ya que el vector Vp se actualiza a partir de la segunda iteración y el Vpi ya no es necesario por ser un conjunto de elementos iniciales para el vector de pagos.
- Se calcula el vector Vp a partir de la aplicación de la ecuación (14), teniendo en cuenta el vector anterior Vp(k-1), la constante el vector de "fitness" calculado en el primer paso y el "fitness" promedio, es decir, se usa la ecuación (22), junto con todos los parámetros necesarios para su cálculo.

  Para la siguiente iteración se vuelve a aplicar el mismo procedimiento sin tener en cuenta el vecto Vpi.

En resumen, la implementación se reduce al manejo matricial y operaciones matemáticas básicas de suma, resta, multiplicación y división.

3.4 Funciones de la librería

Las funciones en la tabla XII son suficientes para plantear un algoritmo que incluya un control con dinámicas de replicadores.

Tabla 13. Funciones y sentencias de la librería para dinámica de replicadores

Sentencia	Función
REPLICATOR RD	Crea un objeto REPLICATOR llamado RD, con el fin de usarse en todo el algoritmo con las funciones principales
RD.Config(Vpi,Vr,α,B)	Realiza la configuración del control por dinámica de replicadores ingresando los principales parámetros de configuración inicial.

RD. Limites(a ,b)	Limita el conjunto de valores de salida del controlador a estar dentro de un rango permitido.
Vp=RD.calculo (Vss)	Calcula la señal de salida, aplicando el algoritmo de la ecuación (22), teniendo en cuenta los parámetros de configuración y entrega en la variable Pk2 el conjunto de señales de control que se deben aplicar en la planta. A la función solo se le debe ingresar el vector de medida de los sensores Vss.
Pwm=RD.escalizado(c)	Si se desea potencializar la señal de control se puede usar esta sentencia, en donde, el conjunto de valores Vp se multiplican por un factor c, y se almacenan en la variable Pwm. También realiza una limitación de los valores entre 0-4096, que son los valores de PWM y de salida analógica que posee el dispositivo Tesla

IV. DISCUSIÓN Y CONCLUSIONES

Revisando en conjunto el sistema conseguido es claro concluir que se trata de un desarrollo muy completo, que abarca elaboración de software, hardware y desarrollo matemático. Cada extensión del trabajo realiza un aporte a la ingeniería

La construcción de una librería para Matlab y Simulink es un trabajo tedioso pero gratificante, ya que rompe la barrera de imposición de herramientas de desarrollo impuestas por Mathworks y abre las puertas para que estudiantes e ingenieros puedan construir su propia librería y adapte el software de Matlab a algún requerimiento en particular de su hardware. En muchas ocasiones, estudiantes que elaboran plantas y requieren un pre-procesamiento de los sensores mediante un algoritmo en la tarjeta y luego enviar esta información a Simulink, no lo pueden hacer, sencillamente porque las librerías de Mathworks para Simulink no incorporan la posibilidad de adicionar código en la tarjeta y usarla como adquisición. Ahora el estudiante puede transferir sus datos como su necesidad lo requiera. El código fuente que se entrega contiene toda la información comentada a fin de que cualquier tarjeta de desarrollo se pueda volver lo flexible que el proyecto de desarrollo lo requiera.

La realidad virtual se distingue de una animación o una animación normal en que es una herramienta que brinda una inmersión sensorial a fin de que se muestre claramente en el mundo real como está actuando el algoritmo matemático sobre un elemento físico. Mediante el método de simulación virtual, donde se une controles hechos en aplicaciones Android con Simulink y el "toolbox" de Simulink 3d Animación; es una herramienta esencial tanto para docentes como para estudiantes, ya que posibilita mostrar y enseñar visualmente y de forma interactiva como actúa sobre un sistema físico la variación de parámetros importantes de control. En una simulación ya no es necesario parar la simulación o manipular opciones en la pantalla para cambiar parámetros, si no que se puede hacer desde un dispositivo móvil.

Se consiguió establecer el método de creación del programa de desarrollo de arduino, incluida la programación java para la interfaz, la forma de compilación, la forma de carga de código, se exploró desde lo más profundo como es crear un entorno de desarrollo y programación, lo que abre la puerta para crear cualquier sistema embebido, ya que ahora se le puede dar soporte de

programación e inventarse el programa que se desee con las características que necesite cualquier tipo de sistema embebidos. Inclusive ahora es posible compilar con el lenguaje que se desee, ya no es necesario someterse a algún lenguaje en particular con un idioma diferente, ahora ya es posible inventar o crear un lenguaje nativo en el que una determinada región de programadores esté de acuerdo. Esta posibilidad es importante, ya que se pone la información a la disponibilidad del estudiante a fin de que se creen nuevas formas de tarjetas y nuevas formas de impulsar el desarrollo del campo de los sistemas embebidos.

Con la manipulación de Linux se trataron temas muy profundos de este sistema operativo, se comprobó cómo es la arquitectura general, como se compone la madre del sistema operativo, para así lograr compilar un kernel que permitiese cumplir las funciones deseadas. Además de crear nuevas formas de software, es posible cargar estos programas en sistemas embebidos avanzados que incluyen procesadores y frecuencias de reloj muy altas. También se establece como realizar nuevas formas de conectar la tarjeta globalmente con el uso de internet. Toda esta parte tiene como objetivo demostrar que se puede crear a partir de hardware libre y software libre laboratorios virtuales, en donde, no se tenga que tener la tarjeta para programarse, esta puede estar en cualquier lugar cumpliendo una función específica y permitir al estudiante programarla desde cualquier lugar donde se encuentre. El método de utilizar ejecutables de java y transmitir datos por medio de un servicio web fantasma en internet, es un método probado en laboratorio, por lo que es una realidad que va impulsar el desarrollo de hardware en unión con internet y el diseño de software.

Las librerías desarrolladas y mejoradas para sistemas de control representa un aporte enorme para la implementación de estrategias de control en sistemas embebidos, el principal objetivo de esta parte del desarrollo es romper las barreras del uso de software avanzado para la implementación de algoritmos complejos y demostrar que se pueden llevar a cabo con el uso de hardware libre. Es posible en este momento realizar un control por dinámicas de replicadores usando únicamente la tarjeta de desarrollo y la planta a controlar; lo que es una reducción sustancial del costo de un sistema de control frente al uso de software especializado y tarjetas de adquisición muy avanzadas.

Finalmente, dado cumplimiento a los objetivos principales de este desarrollo se puede plantear que en el desarrollo de ellos, se obtuvo un gran conocimiento en muchas áreas de la ingeniería, especialmente ingeniería electrónica y de sistemas, este conocimiento debe ser aprovechado a fin de mejorar y darle más flexibilidad de uso a los sistemas actuales de hardware libre que se tiene. El comentario final fruto del aprendizaje de este trabajo es establecer que para la evolución de un

determinado desarrollo lo importante es generar el conocimiento suficiente para que este pueda evolucionar.

V. REFERENCIAS

[1] Perens, Bruce. The Open Source Definition. Documento disponible en: http://oreilly.com/openbook/opensources/book/perens.html

[2] National Instruments, *Selector de módulos online*, página disponible en: http://sine.ni.com/module-selector/app/module_selector?platform=pci

[3] DE LA HORA KÖLLMER, Mario. Sistemas de Adquisición de Datos basados en la plataforma Arduino. Madrid, 2013, pp. 36,48. Trabajo de grado. Universidad Carlos III de Madrid.

[4] Mathworks. Centro de documentación. Documento en línea disponible en: http://www.mathworks.com/help/matlab/quick-guide-to-class-syntax.html

[5] Mathworks. Centro de documentación. Funciones S de Matlab de 2do nivel. Disponible en: http://www.mathworks.com/help/simulink/sfg/writing-level-2-matlab-s-functions.html

[6] Mathworks. Centro de docuemntación.Editor general de máscaras, información disponible en : http://www.mathworks.com/help/simulink/gui/mask-editor-overview.html

[7] Mathworks. Centro de soporte. Soporte de Simulink para Android. Disponible en: http://www.mathworks.com/hardware-support/arduino-simulink.html

[8] Android Developers, Información disponible en: http://developer.android.com/tools/help/adt.html

[9] GEHRING JONAS. GraphView Library for Android, información disponible en: http://android-graphview.org/

[10] Mathworks. Simulink 3D Animation, Información disponible en: http://www.mathworks.com/products/3d-animation/

[11] Hoja de datos de Intel Galileo. Disponible en: http://www.intel.com/newsroom/kits/quark/galileo/pdfs/Intel_Galileo_Datasheet.pdf

[12] Processing. Página web oficial disponible en: http://www.processing.org/

[13] Arduino. Entorno de desarrollo y descargas. Disponible en: http://arduino.cc/en/pmwiki.php?n=main/software

[14] Netbeans. Página web oficial disponible en: https://netbeans.org/

[15] Solanes, Joshua. Kernel Personalizado en Udoo. Guía disponible en: http://joshua.solanes.us/operating-system-2/linux-os/2013/12/01/custom-kernel-on-the-udoo/

[16] Digi, Información de módulo Xbee PRO 900HP, disponible en: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-pro-900hp#overview

[17] Hoja de características del integrado LM324, disponible en: http://www.ti.com/lit/ds/symlink/lm124-n.pdf

[18] Hoja de datos del transistor 2N2222. Disponible en: http://www.fairchildsemi.com/ds/PN/PN2222A.pdf

[19] Adrafruit. Descripción y características disponible en: http://www.adafruit.com/products/395

[20] Rao, Sankara. *Numerical Methods for Scientists and Engineers* New Delhi (India): Prentice-Hall of India Learning Private. pp. 151–159.

[21] Beauregard, Brett. Guía de Uso de la Librería PID arduino. Guía disponible en:

http://brettbeauregard.com/blog/wp-content/uploads/2012/07/Gu%C3%ADa-de-uso-PID-para-Arduino.pdf
[22] Taylor, P. D. and Jonker, L. B., "Evolutionary stable strategies and game dynamics,"Mathematical Biosciences, vol. 40, pp. 145–156, 1978.
[23] Silva, A.; Laruelle, A.; Zuazo, P. "Replicator Dynamics and Evolutionary Stable Strategies in Heterogeneous Games". *Discussion Papers in Economics, Department of Economics, University of Leicester*. December 20, 2011.
[24] Pantoja, A.; Quijano, N. Distributed Optimization Using Population Dynamics with a Local Replicator Equation. Decision and Control (CDC), 2012 IEEE 51st Annual Conference, vol., no., pp.3790, 3795, 10-13 Dec. 2012.
[25] Pantoja, A. Distributed Optimization with Population Dynamics. Tesis Doctoral. Universidad de los Andes. Julio. 2012.

VI. ANEXOS

Toda la documentación es libre y podrá descargarse desde el siguiente link:

https://onedrive.live.com/redir?resid=92D09C62B7127EE8!816&authkey=!AHERsH3pjVH2qt4&ithint=folder%2c