"UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
ESCUELA DE INGENIERÍA EN MECATRÓNICA"

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN MECATRÓNICA

TITLE:

" INVERTED PENDULUM CONTROLLED VIA REAL-TIME TECHNIQUES: CONTROLLER DEVELOPMENT VIA A FIELD NETWORK"

TEMA:

"PÉNDULO INVERTIDO CONTROLADO CON TÉCNICAS DE TIEMPO REAL: DESARROLLO DEL CONTROLADOR VÍA CAMPO DE DATOS"

AUTOR:
MAURICIO FERNANDO HINOJOSA REA

DIRECTOR DE TESIS:
Ing. XAVIER ROSERO

Ibarra-Ecuador
2017

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA
AUTORIZACIÓN DE USO Y PUBLICACIÓN
A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

IDENTIFICACIÓN DE LA OBRA.

La Universidad Técnica del Norte dentro del proyecto Repositorio Digital Institucional, determinó la necesidad de disponer de textos completos en formato digital con la finalidad de apoyar los procesos de investigación, docencia y extensión de la Universidad. Por medio del presente documento dejo sentada mi voluntad de participar en este proyecto, para lo cual pongo a disposición la siguiente información:

Datos de Contacto :	
CÉDULA DE IDENTIDAD:	1003531082
APELLIDOS Y NOMBRES:	Mauricio Fernando Hinojosa Rea
DIRECCIÓN:	Jacinto Collahuazo 4ta Etapa Calles G y 7
EMAIL:	mfhinojosa@utn.edu.ec
TELÉFONO FIJO:	062-903484
TELÉFONO MÓVIL:	0980780421

DATOS DE LA OBRA:	
TTULO:	**INVERTED PENDULUM CONTROLLED VIA REAL-TIME TECHNIQUES: CONTROLLER DEVELOPMENT VIA A FIELD NETWORK**
AUTOR:	MAURICIO FERNANDO HINOJOSA REA
FECHA:	
PROGRAMA:	PREGRADO
TITULO POR EL QUE OPTA:	INGENIERO EN MECATRÓNICA
DIRECTOR:	Ing. XAVIER ROSERO

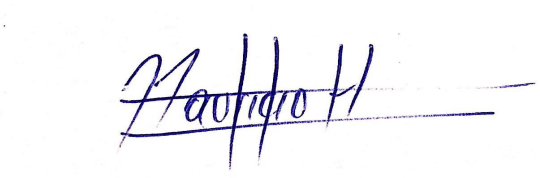
AUTORIZACIÓN DE USO A FAVOR DE LA UNIVERSIDAD

Yo, Mauricio Fernando Hinojosa Rea con cédula de identidad Nro. 100353108-2, en calidad de autor y titular de los derechos patrimoniales de la obra o trabajo de grado descrito anteriormente, hago entrega del ejemplar respectivo en formato digital y autorizo a la Universidad Técnica del Norte, la publicación de la obra en el Repositorio Digital Institucional y uso del archivo digital en la Biblioteca de la Universidad con fines académicos, para ampliar la disponibilidad del material y como apoyo a la educación, investigación y extensión; en concordancia con la Ley de Educación Superior Artículo 144.

CONSTANCIA DE ORIGINALIDAD Y RESPONSABILIDAD

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrollo sin violar derechos de autores de terceros, por lo tanto la obra es original, y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, 24/7/2017



Firma

Nombre: Mauricio Fernando Hinojosa Rea

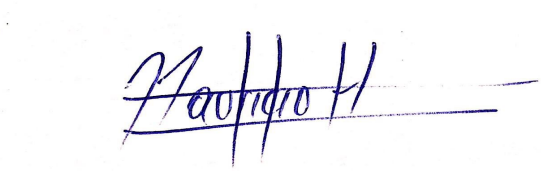
Cédula: 100353108-2

LA UNIVERSIDAD TÉCNICA DEL NORTE

CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE GRADO A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

Yo, Mauricio Fernando Hinojosa Rea, con cédula de identidad Nro. 100353108-2, manifiesto mi voluntad de ceder a la Universidad Técnica del Norte los derechos patrimoniales consagrados en la Ley de Propiedad Intelectual del Ecuador, artículos 4, 5 y 6, en calidad de autor (es) de la obra o trabajo de grado denominado: INVERTED PENDULUM CONTROLLED VIA REAL-TIME TECHNIQUES: CONTROLLER DEVELOPMENT VIA A FIELD NETWORK., que ha sido desarrollado para optar por el título de: Ingeniero en Mecatrónica, en la Universidad Técnica del Norte, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente. En mi condición de autor me reserva los derechos morales de la obra antes citada. En concordancia suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Técnica del Norte.

Ibarra, 24/7/2017



Firma

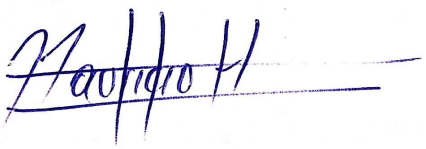
Nombre: Mauricio Fernando Hinojosa Rea

Cédula: 100353108-2

DECLARACIÓN

Yo, MAURICIO FERNANDO HINOJOSA REA, declaro bajo juramento que el trabajo aquí escrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Universidad Técnica del Norte - Ibarra, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normativa institucional vigente.

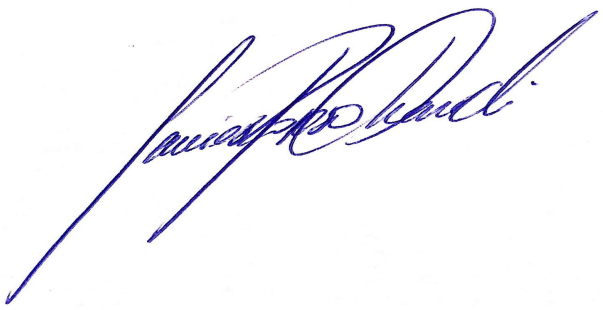


Nombre: MAURICIO FERNANDO HINOJOSA REA

Cédula: 100353108-2

CERTIFICACIÓN

Certifico que el presente Trabajo de Grado " INVERTED PENDULUM CONTROLLED VIA REAL-TIME TECHNIQUES: CONTROLLER DEVELOPMENT VIA A FIELD NETWORK", fue desarrollado por el egresado MAURICIO FERNANDO HINOJOSA REA, bajo mi supervisión, lo cual certifico en honor a la verdad.



Ing. XAVIER ROSERO

DIRECTOR DEL PROYECTO

AGRADECIMIENTO

No podría estar más satisfecho y extasiado con el sentimiento que me provoca terminar esta etapa de mi vida, de la que sé que mis padres, Ligia y Freddy se que siente orgullos. Esta tesis y mi trabajo durante todos estos años de universidad se lo agradezco a ellos; sin su paciencia y sin su amor no habría podido hacer. Ustedes han sido los promotores de mis sueños y los amo. Me dieron la vida y han cuidado de mi en cada uno de mis días. Me enseñaron todo lo que necesitaba para defenderme, para levantarme, para ser humilde. No hay palabra en el mundo o definición que alcance para llenar lo que ambos son para mi.

De pequeño adoraba observar el cielo; esas brillantes luces despertaron mi curiosidad; me hice muchas preguntas, de las que algunas tardé algún tiempo en contestarlas; fue sin duda mi primer amor y me prometí que nunca dejar de dudar de todo, porque para eso el cosmos me dotó de inteligencia. Y a pesar de que la vida me llevo por otro rumbo, ese pequeño niño está aún ansioso de que luche por sus sueños; mis sueños.

Claro, en la vida no se conseguiría mucho sin las personas que de alguna u otra manera me apoyaron en éste trabajo de tesis; la vida me dió la grata fortuna de haber encontrado personas increibles; de mis papás, de mis compañeros de universidad y mis amigos de toda la vida ya que a los amigos en todo momento son como hermanos. Jamás olvidaría a todos y cada uno de los ingenieros docentes de cuyas clases tomé la inspiración para mi vida como estudiante de ingeniería.

A mis abuelitos que son la joya mas valiosa que un humano podría tener, a mis tios y todas las personas que me aconsejaron en su día y sé que nunca dejarán de hacerlo; parte de este trabajo fluyó bajo sus consejos. A mi director de tesis, por mostrarme el camino para desarollar esta tesis.

Alexis hermano, nos esforzamos enormemente y no nos dejamos vencer aún en los días más difíciles.

Gracias a todos.

Mauricio Hinojosa

DEDICATORIA

Quiero dedicar con profunda algarabía y con el sentido de poner en sus mentes un consejo, y llenar un poco más sus mundos internos; a mis dos hermanas, Michelle y Micaela: En un día claro puedes ver kilómetros sobre el horizonte. En una noche clara, años luz.

No hay nada de malo en ver el cielo y el infierno. Sean mejores todos los día en cada uno de los aspectos de su vida; sean mill veces mejores y diferentes que su hermano. Abran su mente a todo y para todo. La vida no es buena ni mala, es simplemente maravillosa. Hagan todo lo que quieran. Cumplan sus sueños.

Si tú lo deseas puedes volar, sólo tienes que confiar mucho en ti y seguir; puedes contar conmigo, te doy todo mi apoyo.

Mauricio Hinojosa

Resumen

Un péndulo invertido es un dispositivo que consiste en una barra cilíndrica con libertad de oscilar alrededor de un pivot fijo, esta montado sobre un carro que sigue una trayectoria horizontal. El objetivo es mantener el péndulo perpendicular a la trayectoria del carro ante la presencia de perturbaciones en el sistema; el sistema corrige la posición angular del péndulo desplazando el carro con un sistema de banda-motor, según una acción de control calculada.

El controlador implementado para el sistema fue desarrollado mediante la técnica de espacio de estados a partir del modelo matemático y simulado en Matlab obteniendo sus ganancias, las mismas que sirven para modificar el comportamiento del sistema. El sistema de control está divido en dos, una parte de monitoreo de datos o HMI y otra de adquisición de datos y control.

El sistema de adquisición de datos está montado en 4 nodos controlados por Arduinos dentro de una red CAN; el nodo 0 o nodo central tiene la tarea de recibir datos desde la red, procesarlos y tomar una acción de control; también tiene la importante tarea de servir de enlace entre la HMI y el resto de la red. El nodo 1, adquiere el dato de posición del carro y la envia al nodo 2; éste a su vez toma el dato del ángulo del péndulo y lo envia al nodo 0. El nodo 4 en conjunto con un puente H toma la acción de control enviada desde el nodo 0 y la convierte en voltaje el cual controla el motor enlazado al carro.

El sistema de monitoreo o HMI desarrollada en PyCharm muestra los datos provenientes de la red, como la posición y velocidad del carro y del péndulo. Desde la HMI se controla la red de comunicaciones y se puede enviar datos de ganancias para el controlador en el nodo central.

Abstract

An inverted pendulum is a device consisting of a cylindrical bar, free to oscillate about a fixed pivot, is mounted on a carriage following a horizontal path. The objective is to maintain the pendulum perpendicular to the trajectory of the car in the presence of disturbances in the system; the system corrects the angular position of the pendulum by moving the carriage with a motor-band system, according to a calculated control action.

The controller implemented for the system was developed using the state space technique from the mathematical model and simulated in Matlab obtaining its gains, which are used to modify the behavior of the system. The control system is divided into two, one part of data monitoring or HMI and one of data acquisition. Data and control.

The data acquisition system is mounted on 4 nodes controlled by Arduinos within a CAN network; The node 0 or central node has the task of receiving data from the network, processing them and taking a control action; Also has the important task of serving as a link between the HMI and the rest of the network. Node 1 acquires the position data of the carriage and sends it to node 2; this in turn takes the data from the angle of the pendulum and sends it to node 0. The node 4 in conjunction with a bridge H takes the control action sent from node 0 and converts it to voltage which Controls the motor attached to the carriage.

The monitoring system or HMI developed in PyCharm shows the data coming from the network, such as the position and speed of the car and the pendulum. From the HMI the communication network is controlled and profit data can be sent to the controller at the central node.

Índice general

Índice de figuras

Índice de cuadros

Índice de ecuaciones

Capítulo 1

Introducción

El desarrollo de controladores es una constante que va de mano con el crecimiento económico y tecnológico de los pueblos y sus industrias; impulsando avances en los procesos industriales con el proposito de fomentar competitividad para los paises en materia tecnológica [2].

Al desarrollar controladores que funcionen de la manera más adecuada en sistemas lineales y no lineales representa en si un reto, en especial desde el punto de vista académico, al tratar de acoplar la utilidad de los controladores, con la seguridad que deben presentar, y lo confiables que deben ser a la hora de su implementación [2].

Existe una simple razón por la que el "Péndulo" es tan utilizado en investigación, es su dinámica no lineal, que permite comprender el comportamiento de sistemas mas complejos desde el punto de vista de su linealidad y dinámica, como son los sectores de transporte, telecomunicaciones, aeronáutica; con lo que los controladores son fácilmente aplicables a estos sistemas [2].

En el presente trabajo se realizará la implementación de un controlador con técnicas de tiempo real sobre arduinos a través de una red de campo (basada en protocolos de comunicación CAN multi-nodo); donde los datos de monitoreo así como el controlador serán mostrados en la HMI desarrollada para este propósito.

1.1. Objetivos

1.1.1. Objetivo Principal:

- Desarrollar e implementar el sistema de control del péndulo invertido a través de una red de campo de datos.

1.1.2. Objetivos Específicos:

- Revisar bibliografía y documentos acerca del estado del arte referente al control por red.
- Determinar los requisitos y parámetros necesarios para realizar el diseño del sistema de control en base a las características de la planta, y a su función de transferencia.
- Diseñar y probar el sistema de control del péndulo a través de software de simulación.
- Diseñar la HMI para adquisición de datos y supervisión.
- Implementar el sistema de control y probarlo para condiciones reales de trabajo.

1.2. Antescedentes

El sistema de péndulo es uno de los más factibles de resolver dentro del mundo físico; ya que de él parten varios problemas que se ponen de manifiesto para los sistemas de control[3].

El modélo matemático que se traslada de la realidad a lo abstracto de la física del péndulo, presenta una formulación basada en ecuaciones diferenciales, que se asemejan en muchos casos a otros sistemas reales más o menos complejos dentro de varios sectores tecnológicos: el control seguro de una planta química o un sistema de vuelo de aeronaves. Por ende, el estudio de este sistema sirve como punto de partida para desarrollar controladores especializados para sistemas complejos [3].

Basados en ese punto, el péndulo invertido presenta una enorme variedad de problemas que lo hacen uno de los sistemas más concretos y útiles a la hora del ensayo de leyes de control, en las últimas décadas. Los primeros péndulos nacieron en los 70, y aún después de tantos años se sigue utilizando el péndulo como caso de estudio e investigación [3].

1.3. Problema

En teoría de control uno de los problemas mas conocidos, clásicos e importantes es el así llamado “Péndulo Invertido”. Tiene como aplicaciones, desde el control de estabilidad de grúas, hasta el modelamiento de la bipedestación humana [4]; y no está demás decir que este tipo de sistemas esta presente en varias universidades, como por ejemplo: la Universidad Nacional de Colombia, la Universidad Complutense de Madrid, la Universidad Autónoma de Puebla, entre muchas otras al rededor del mundo; por lo que se puede ver que al rededor del mundo utilizan estre problema físico como sistema base para el desarrollo de controladores en investigaciones académicas.

Es accesible desde el punto de vista académico y de control relativamente fácil, en él se pueden observar características de sistemas de lazo abierto y cerrado [4].

Hay que tener presente que dentro de la UTN y especialmente en la facultad de ingeniería, no se cuenta con módulos ni plantas basados en hardware y software abierto orientados al uso didáctico o como base para el desarrollo de investigación científica.

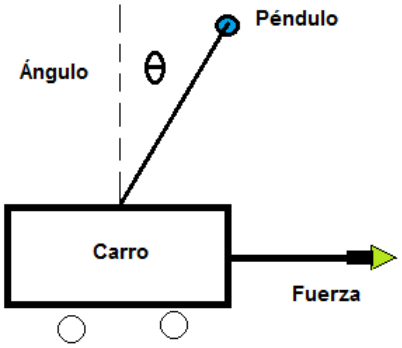


Figura 1.1: Diagrama simple del Péndulo.

1.4. Justificación

Al conseguir los objetivos del proyecto, se desea demostrar que es posible controlar un sistema de control de dinámica rápida a través de una red de campo (basada en una capa física o de enlace).

Por las características ampliamente estudiadas del péndulo invertido, siendo éste una planta no lineal de segundo orden, y considerando que muchos sistemas físicos reales son similares, esta planta se convierte en un sistema útil para el ensayo de soluciones de control, así como para el desarrollo de sistemas de control más eficientes.

Este proyecto permitirá cumplir dos metas:

- Afianzar el conocimiento teórico adquirido por el estudiante a través de la implementación de sistemas reales.
- Servir como plataforma para el ensayo de nuevas soluciones de control obtenidas a través de la investigación.

1.5. Alcance

La construcción del péndulo constituye un proyecto impulsado por el grupo de investigación sistemas inteligentes de la UTN en el que se podrá probar diferentes técnicas de diseño de controladores tales como control óptimo y control por posicionamiento

de polos (espacio de estados); implementación de observadores y observadores óptimos. Todo esto en base a funciones de transferencia sustentadas matemáticamente que podrían comprender el modelado de retardos de propagación y procesamiento. Se pondrá a prueba el sistema con un controlador basado en técnicas de estado de espacios; con el que se validará la funcionalidad de todo el sistema.

El control a implementarse en el péndulo se realizará a través de varios sistemas microprocesados comunicados mediante una red de campo (control distribuido). Además la planta contará con una HMI (interfaz humano-máquina) implementada en un ordenador, con el fin de establecer un sistema de supervisión y adquisición de datos del péndulo para un posterior análisis off-line. Todo los sistemas es basarán en software y hardware libre.

1.6. Datos generales

1.6.1. Sistema Mecánico

El sistema mecánico así como los componentes del mismo están descritos a detalle en la tesis que precede a éste trabajo; conformado de: una mesa, un sistema de rieles que permiten la movilidad, un carro, un sistema eje-péndulos, sensores de posición y un sistema de transmisión de fuerza con un motor acoplado.

1.6.2. Sistema Electrónico y de Control

En breves rasgos el sistema electrónico esta compuesto por 4 “Arduinos Uno” montadas en una red CAN (4 nodos). Cada nodo tendrá una funcionalidad diferente; a breves rasgos los nodos 1 y 2 obtendrán datos de los sensores, los procesarán y convertirán pertinentemente a las unidades adecuadas y los subirá a la red de forma digital. El nodo central o nodo 0 procesará los datos y calculará una acción de control que será enviada al nodo 3 donde se la convertirá en voltaje para accionar el motor acoplado al sistema; a dems de proveer la información necesaria para la HMI.

Capítulo 2

Fundamento Teórico

2.1. Diseño del sistemas de control en espacio de estados

2.1.1. Asignación de polos

Todas las variables de estado con las que el sistema cuenta, se suponen medibles y disponibles para su realimentación y si el sistema es completamente controlable, se pueden colocar los polos del sistema en cualquier posición dentro de una matriz de estado de ganancias (matriz de ganancias de a realimentación de estados). Se consiguen estos polos a partir de la respuesta transitoria, así como de las respuestas de frecuencia. Seleccionando una matriz de ganancias apropiado se puede conseguir que los polos del sistema en lazo cerrado tengan una posición adecuada [5].

2.1.2. Diseño mediante asignación de polos

Se deben especificar todos los polos en lazo cerrado del sistema, tomando en cuenta que para ello se deben tener medidas excelentes de las variables de estado o incluir un observador de estado en el sistema; a parte de que el requisito primordial es que el sistema sea completamente controlable [5].

Siendo un sistema de control con la forma:

$$\begin{cases} \dot{x} = Ax + BU \\ y = Cx + DU \end{cases}$$

donde: x =vector de estado (vector de dimensión n)
y =señal de salida (escalar)
u =señal de control (escalar)
A =matriz de coeficientes constantes $n * n$
B =matriz de coeficientes constantes $n * 1$
C =matriz de coeficientes constantes $1 * n$
D =constantes (escalar)

Se selecciona la señal de control (2.1):

$$u = -Kx \tag{2.1}$$

Es decir que **u** es es un estado instantáneo; siendo la matriz **K** de 1xn denominada como matriz de ganancia de realimentación de estado. Sistema mostrado en la Fig. 2.1 y donde **x** es un vector de dimensión $n * 1$; al realizar la operacin correspondiente matricial, se obtiene K con un valor escalar [5].

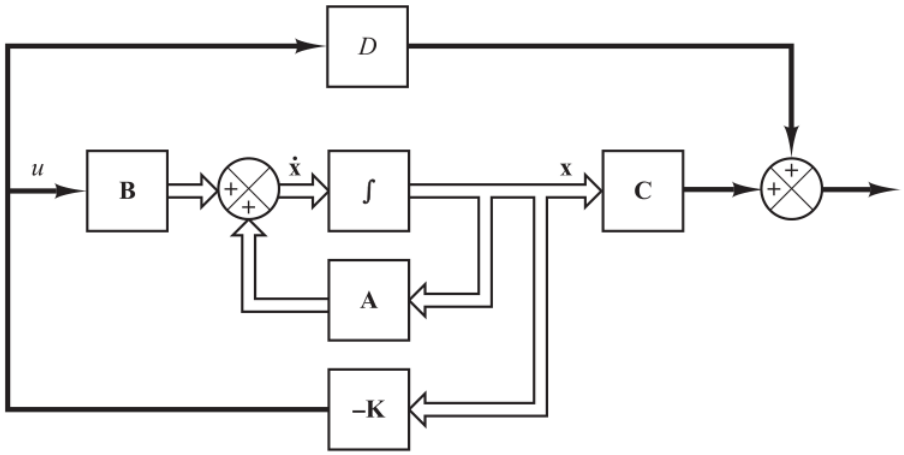


Figura 2.1: Sistema de control en lazo cerrado
Extraído: [5]

2.2. Entorno de desarrollo integrado (IDE)

El propósito más importante de un IDE es ayudar a acelerar el desarrollo preciso, rápido y robusto de una aplicación. Muchos IDEs se encuentran en desarrollo o están implementadas en universidades [6].

Kavitha and Sindhu [7] Muestra que algunos IDE contienen un compilador,intérprete o ambos, como NetBeans y Eclipse. Muchos IDEs modernos también tienen un navegador de clase y un diagrama de jerarquía de clases.

2.2.1. Entorno de desarrollo integrado abierto

Open IDEs son IDEs desarrollados, depurados y montados bajo una Licencia de Software de Código Abierto, y permiten al programador de desarrollo de software nuevo, con todo tipo de licencias. Como PyCharm, NeatBeans, Matlab y otros.

2.2.2. Herramientas de los IDEs

"Hay muchas herramientas IDE disponibles para el código fuente editor, herramientas de automatización y depurador. Algunos de los Herramientas son, ***Eclipse,NetBeans, Code::Blocks, Code Lite, Dialog Blocks***" [6].

Worldwide, Aug 2016 compared to a year ago:

Rank	Change	IDE	Share	Trend
1	↑	Visual Studio	22.94 %	-0.1 %
2	↓	Eclipse	22.64 %	-6.1 %
3	↑	Android Studio	10.29 %	+2.7 %
4	↓	Vim	8.03 %	-0.0 %
5		NetBeans	5.58 %	-0.3 %
6		Xcode	5.39 %	-0.2 %
7		Sublime Text	4.5 %	+0.1 %
8	↑	IntelliJ	4.18 %	+1.2 %
9	↓	Komodo	3.81 %	+0.3 %
10		Xamarin	3.34 %	+1.6 %
11		Emacs	1.88 %	+0.1 %
12		PhpStorm	1.58 %	+0.3 %
13		pyCharm	1.56 %	+0.6 %
14		Light Table	1.1 %	+0.2 %
15		Cloud9	0.9 %	-0.1 %

Figura 2.2: Top 15 IDES.
Extraído de: `http://pypl.github.io/IDE.html`

El top mostrado en la Fig. 2.2 contiene la frecuencia con la que se realizan búsquedas de IDEs en Google. El índice puede ayudarle a decidir qué IDE utilizar [8].

Como se puede ver en la Fig. 2.2, están los IDE más importantes que se buscan en Google, y están clasificadas por el lenguaje compatible como: ActionScript, Ada, Assembly, BASIC, C/C++, C#, Common Lisp, Component Pascal, Eiffel, Fortran, Haxe, Java, JavaScript, Lua, Pascal, Object Pascal, Perl, PHP, Python,Racket, Ruby, Scala, Small Basic, Smalltalk, Tcl.

En esta sección elegimos el lenguaje Python porque es una herramienta gratuita para el desarrollo y también es un lenguaje de programación de alto nivel, de propósito general, interpretado y dinámico; todas estas particularidades hacen que este lenguaje sea perfecto para nuestro trabajo, y van a ser explicados en los siguientes capítulos.

2.2.3. Software de código abierto

El software de código abierto da acceso al código fuente; luego los usuarios pueden cambiar o modificar el producto original [9].

En otras palabras los usuarios tienen la libertad para ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software. Algunos autores lo denominan "software libre" (tiene un contexto de libertad no de precio) [10].

2.3. Lenguajes de programación

Lenguaje creado para para interpretar instrucciones elementales de la arquitectura mismo de los ordenadores; facilitando la tarea de programador, y que a partir de estos lenguajes se crean otros de nivel superior o lenguajes de alto nivel, que son mucho más productivos [11].

2.3.1. Paradigmas de programación

"Programming paradigms o Paradigmas de programación", según Rodríguez et al. [11] son patrones que moldean la forma de pensar y de formular soluciones asi como la de estructurar programas. Los paradigmas de programación son:

- Programación imperativa.
- Programación funcional.
- Programación lógica.
- Programación orientada a objetos.

Lenguajes de alto nivel

Del mismo modo Rodríguez et al. [11] nos muestra que existen algunas características dentro de estos lenguajes:

Es un lenguaje independiente de la máquina. Programas legibles y más fácil de entender; mucho más natural. Repertorio de instrucciones amplio, potente y fácilmente utilizable. Estructura próxima a los lenguajes naturales, mantenimiento y corrección de errores más sencilla [11].

Traductores (Translators)

Se establece que un traductor migra el programa creado de un lenguaje fuente a un lenguaje máquina. El proceso de conversión puede ser: por interpretación o por

compilación [11].

Intérpretes (Interpreters)

Según [11] es un programa que toma como entrada un programa escrito en lenguaje fuente y lo va traduciendo y ejecutando instrucción por instrucción (de una en una).

Compiladores (Compilers)

Es un programa que toma como entrada un programa fuente y genera un programa equivalente llamado programa objeto [11].

2.4. Python

2.4.1. Introducción

Python nace en los noventa bajo la direccion de guido Van Rossum, es un lenguaje similar a Perl, pero con una sintaxis muy limpia y un código legible. Es un lenguaje interpretado, multiplataforma y orientada a objetos[12].

2.4.2. Lenguaje interpretado o script

Lenguaje que ejecuta un programa intermedio o de interpretación en lugar de compilar a un lenguaje máquina, haciendo que este pueda ser interpretado directamente por la computadora. Python tiene muchas de las características de los lenguajes compilados, por lo que se podría decir que es semi interpretado[12].

2.4.3. Tipado dinámico

Esta característica se refiere a que no es necesario declarar el tipo de dato de una variable, sino que este se determinará en tiempo de ejecución; haciendo posible que el tipo de variable pueda cambiar con el tiempo [12].

2.4.4. Tipado fuerte

Es primordial convertir de forma explícita una variable a un nuevo tipo previo a su uso [12].

2.4.5. Python

Python debería ser universal, su sintaxis es sencilla y clara; el tipado es fuerte, es orientada a objetos, tiene gran cantidad de librerías disponibles así como la calidad y potencia del lenguaje; por lo que desarrollar en python es una tarea sencilla [12].

Python es usado en Google, Yahoo, la NASA, Industrias Ligh & Magic [12].

Figura 2.3: Logo Python ™
Extraído de: [12]

2.5. Sistemas operativos de tiempo real

2.5.1. Definición

Así Mall [13] define que un sistema se denomina sistema en tiempo real, cuando se necesita una expresión cuantitativa del tiempo (i.e. real-time) para describir el comportamiento de un sistema.

Ortiz López [14] nos brinda otro par de definiciones que son igualmente aceptables:

- Un sistema a tiempo real debe producir tantas salidas como respuesta a unas entradas dentro de unos límites de tiempo plazos específicos.
- Un sistema en tiempo real debe producir respuestas correctas dentro de unos límites de tiempo.

De estas definiciones el autor Ortiz López [14] se plantea un par de interrogantes a las que da respuesta:

- Qué significa limites de tiempo específicos?
  - No necesitamos el mismo tiempo de respuesta en unas aplicaciones que en otras.
- Por qué se dice “debe producir” y no dice “produce”?
  - No en todas las aplicaciones es necesario que el sistema responda siempre dentro de unos límites de tiempo concretos (ej. aplicaciones multimedia), mientras que en otras es crucial (piloto automático de un avión).

2.5.2. Aplicaciones

Sobre las aplicaciones de los sistemas de tiempo real Mall [13] muestra algunos ejemplos, pero para el caso se presentarán solo dos ejemplos que ilustran la situación de los sistemas operativos de tiempo real:

Aplicaciones industriales

Algunos ejemplos de aplicaciones industriales de sistemas en tiempo real son: sistemas de control de procesos, aplicaciones SCADA, equipos de prueba y medición, sistemas de automatización industrial y todos los materiales en equipos robóticos [13].

- **Ejemplo 1: Control de una planta química**
  Un ordenador en tiempo real implementado en una planta química automatizada, monitorea periódicamente las condiciones de la planta como: lecturas actuales de presión, temperatura y concentración química de la cámara de reacción. Los parámetros son muestreados periódicamente. Para mantener la reacción química a una determinada velocidad, la computadora en tiempo real decide las correcciones de las acciones necesarias, basándose en estos valores; cambiar la presión, temperatura u otros valores. En todos los sentidos, los límites de tiempo en un control de esta planta química varían desde unos pocos segundos hasta varios milisegundos [13].
- **Ejemplo 2: Control, supervisión y Adquisición de Datos (SCADA)**
  SCADA es una categoría de sistemas de control distribuidos que se utilizan en muchas industrias. Un sistema SCADA ayuda a supervisar y controlar un gran número de eventos distribuidos de interés. En los sistemas SCADA, los sensores están dispersos en varios lugares geográficos para recopilar datos en bruto (llamados eventos de interés). Estos datos se procesan a continuación Y almacenados en una base de datos en tiempo real. Los eventos de interés están recibiendo datos de sensores en un sector específico dentro de una planta, luego se almacenan en una base de datos en tiempo real. Esta base de datos se está actualizando con frecuencia para convertirla en un modelo realista del estado de actualización del entorno. La restricción de tiempo en tal aplicación SCADA es que los sensores deben detectar el estado del sistema a intervalos regulares (cada pocos milisegundos) y el mismo debe ser procesado antes de que se detecte el siguiente estado[13].

Otros sistemas comunes de tiempo real dentro de algunas áreas:

- Sistemas de inyección de conbustible multipunto.
- Impresora laser.
- Video conferencias.
- Celulares.
- Sistemas de guía de misiles.

2.5.3. Modelo básico de un sistema en tiempo real

Mall [13] explica el concepto básico sobre el modelo de un sistema en tiempo real:

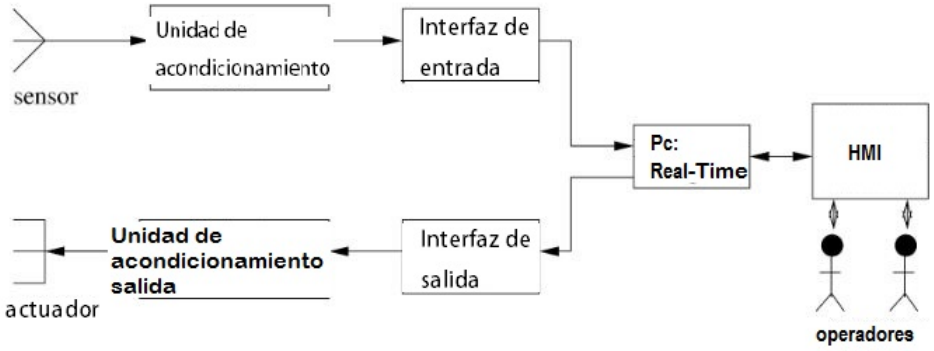


Figura 2.4: Modelo básico de un sistema en tiempo real
Extraído de: [13]

La Fig. 2.4 muestra un simple modelo básico de un sistema en tiempo real. Mall [13] indica que los sensores están interconectados con el bloque de acondicionamiento de entrada, que a su vez está conectado a la entrada interfaz. La interfaz de salida, el acondicionamiento de salida y el accionador están interconectados de una manera complementaria. A continuación están brevemente los roles de los diferentes bloques funcionales de un sistema en tiempo real:

Sensor: Un sensor convierte alguna acción física o característica de su entorno en señales eléctricas [13].

Actuador: Un actuador es cualquier dispositivo que toma sus entradas de la interfaz de salida de un PC o microcontrolador y convierte estas señales eléctricas en acciones físicas [13].

Acondicionamiento de señales: Mall [13] muestra que las señales eléctricas producidas por un ordenador rara vez se pueden utilizar para accionar directamente un actuador por lo que es necesario algunos acondicionamientos realizados en señales brutas generadas por sensores y señales digitales generadas por computadoras.

Conversión análoga-digital: Mall [13] explica que los ordenadores digitales no pueden procesar señales analógicas. Por lo tanto, las señales analógicas necesitan ser convertidas a forma digital. Las señales analógicas pueden convertirse a forma digital utilizando un circuito cuyo diagrama de bloques se muestra en la Fig. 2.7. Utilizando el diagrama de bloques mostrado en Fig.2.7, las señales analógicas se convierten normalmente en forma digital a través de los dos pasos principales siguientes:

- Muestra la señal analógica (mostrado en la Fig. 2.5) a intervalos regulares. Este muestreo puede realizarse mediante un circuito de condensadores que almacena los niveles de voltaje. El nivel de voltaje almacenado puede ser discretizado. Después

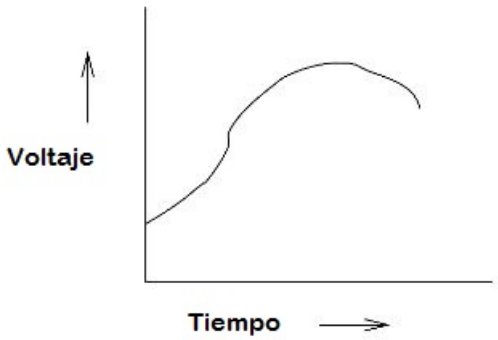


Figura 2.5: Voltaje análogo continuo
Extraído de: [13]

de muestrear la señal analógica, una forma de onda escalonada como se muestra en la Fig. 2.6 es obtenida [13].

- Convierte el valor almacenado en un número binario utilizando un convertidor analógico a digital (ADC) como se muestra en la Fig. 2.7 y almacenar el valor digital en un registro [13].

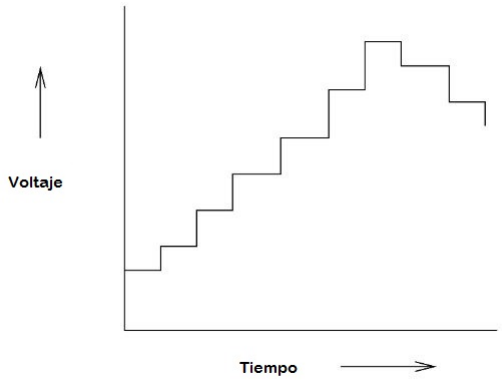


Figura 2.6: Voltaje análogo convertido a la forma discreta
Extraído de: [13]

2.5.4. Tipos de tareas en tiempo real

Tareas del tipo hard

Una tarea del tipo hard en tiempo real es aquella que está limitada a producir sus resultados dentro de ciertos límites de tiempo predefinidos. Se considera que el sistema ha fallado cuando alguna de sus tareas en tiempo real no produce los resultados requeridos antes del límite de tiempo especificado [13].

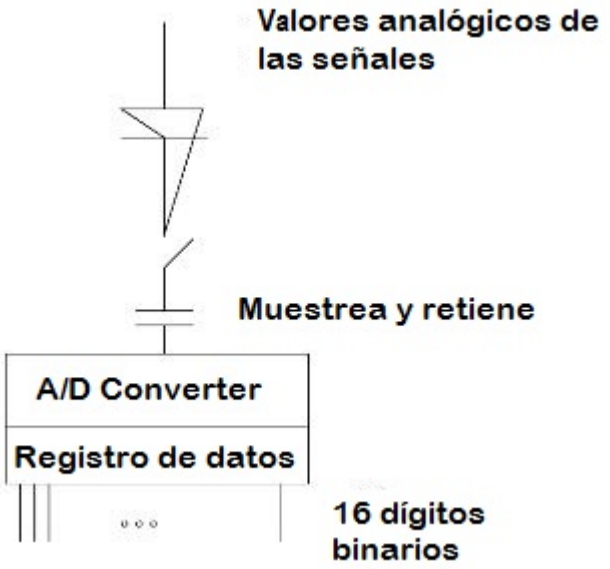


Figura 2.7: Conversión de una señal análoga a número digital de 16 bits.
Extraído de: [13]

Tareas del tipo firm

Cada tarea en tiempo real del tipo firm está asociada con algún plazo predefinido antes del cual se producen sus resultados. Sin embargo, a diferencia de una tarea en tiempo real hard, incluso cuando una tarea en tiempo real firm no completa dentro su plazo la tarea requerida, el sistema no falla. Los resultados tardíos son simplemente descartados. En otras palabras, la utilidad de los resultados calculados por una tarea en tiempo real firm se convierten en cero después de la fecha límite. Se puede decir que si la respuesta en tiempo de una tarea excede el plazo especificado, entonces la utilidad de los resultados se convierte en cero y los resultados se descartan [13].

2.6. Open Hardware

2.6.1. Definición

Aplicando ciertas libertades y definiciones antes dadas al software libre, entre ellas: libertad de uso, modificación, distribución y la redistribucion en mejoras con base al software original, con la diferencia de que para obtener un producto tangible tuvo que haber de antemano un sistema o proyecto con planos, estudios de costo, entre otras cosas. [15].

Russell et al. [16] presenta un resumen más específico sobre el ”hardware abierto”: El hardware de código abierto es un tipo de hardware en el que los esquemas y diseños se hacen sin restricciones y están disponibles para todos.

2.6.2. Propiedades

Russell et al. [16] da algunas propiedades del hardware abierto: Primero se acompañan a menudo del software abierto; Esto puede aportar fiabilidad y facilidad de depuración. En segundo lugar Open Hardware viene con desarrollo modular para prototipos rápidos usando bibliotecas preescritas.

El hardware de código abierto es hardware cuyo diseño se hace disponible públicamente para que cualquier persona pueda estudiar, modificar, distribuir, fabricar y vender el diseño o el hardware basado en ese diseño [16].

Hardware de código abierto

Se refiere al hardware para el cual toda la información del diseño se pone a disposición del público en general. Open source hardware se puede basar en un free hardware design, o el diseño en el cual se basa puede ser restringido de alguna manera [15].

Hardware libre

Es un término usado de vez en cuando como sinónimo para el open source hardware; la cual busca ser directamente paralelo entre el hardware y el software. El término de free hardware es particularmente confuso puesto que implica el estado físico del hardware, más que su diseño, el cual es libre de alguna manera. Esto no es del todo cierto en el sentido del costo, y tiene poca importancia en el sentido social. Lo más simple es evitar este término totalmente, exceptuando su significado de costo [15].

2.6.3. Hardware libre en Ecuador

El interés de los modelos de Hardware Libre para Ecuador procede de su potencial como régimen de producción y distribución de tecnología, así como del desarrollo de nuevos vínculos sociales en torno a ella. El Hardware Libre tiene notables ventajas comparativas, la situación de partida del país invita a prestar atención al desarrollo de hardware libre con el objetivo de que su expansión favorezca el crecimiento económico, sin que tal desarrollo limite el brillante potencial del Hardware Libre para la economía del país [17].

Actualmente existen muchos problemas para el desarrollo de un hardware sustentable. Uno de los principales problemas son los altos costos de producción, habitualmente derivados de la dependencia tecnológica que afecta a muchos países, como Ecuador [17].

Por otra parte, los fabricantes de hardware y los titulares de los derechos de autor aplican la Gestión de Derechos Digitales (DRM, por sus siglas en inglés) para controlar el uso de contenidos y dispositivos digitales [17].

Frente a esto, el Hardware Libre es relativamente barato y está profundamente

integrado en los niveles de alta innovación. Además, puede ser fácilmente modificado a fin de servir ciertos propósitos educativos [17].

2.7. Arduino

2.7.1. Introducción

Entre las implementaciones de HL (Hardware Libre) más representativas está Arduino, una plataforma informática basada en un tablero microcontrolador simple y un ambiente de desarrollo para escribir software en él. Está dirigido a artistas, diseñadores, aficionados y demás interesados en crear dispositivos o ambientes interactivos. Este microcontrolador permite el funcionamiento de varios dispositivos derivados como el Arduino Geiger (detector de radiación), pHduino (medidor de pH), Xoscillo (osciloscopio) y OpenPCR (análisis de ADN) [17].

Lazalde et al. [17] exponen que Arduino bajo la licencia de Creative Commons .Attribution ShareAlike 3.0"(2010), lo que significa que:

- Quienquiera puede producir copias, rediseñarlo o incluso vender placas de hardware.
- Quienquiera que vuelva a publicar el diseño de referencia debe atribuirlo al equipo original de Arduino.

2.7.2. Aspéctos Técnicos

García Osés [1] especifica la conformación del Arduino : El Arduino UNO es muy popular por su sencillez, coste y dimensiones. Utiliza el microcontrolador ATmega328P, fabricado por ATMEL y posee los siguientes aspectos técnicos:

- Voltaje: 5V.
- Voltaje de entrada (recomendado): 7-12V.
- Voltaje de entrada (límites): 6-20V.
- Pines de entradas y salidas digitales: 14.
- Corriente en pines de entrada y salida: 40mA.
- Pines de entrada analógica: 6.
- Corriente en pin de 3.3V: 50mA.
- Memoria Flash: 32KB.
- SRAM: 2KB.

- EEPROM: 1KB.
- Velocidad reloj: 16MHz.
- Dimensiones: 68.6x53.4mm.

Arduino UNO Pinout Diagram

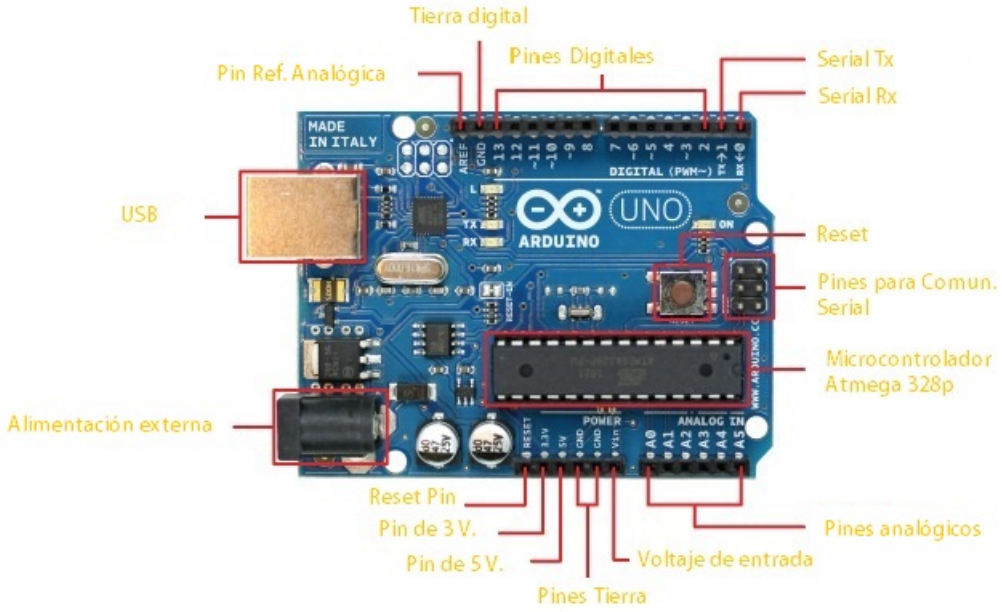


Figura 2.8: Arduino UNO Pinout Diagram
Extraído de: http://www.webondevices.com/wp-content/uploads/2015/06/callouts.jpg

EL arduino IDE al momento de esta investigación se encuentra en la versión 1.6.12 en su página oficial.

2.7.3. Arduino en Ecuador

En Ecuador los estudiantes son los principales impulsodores del desarrollo de proyectos de hardware libre. En la Campus Party celebrada en Quito en 2013, estudiantes de la Universidad Politécnica de Chimborazo, la Universidad Técnica de Loja y la Universidad Salesiana de Quito desarrollaron dispositivos electrónicos basados en Arduino. Un ejemplo de Hardware Libre producido en el Ecuador es la aeronave no tripulada llamada Gavilán UAV-2, diseñada por las Fuerzas Aéreas Ecuatorianas (FAE) para vigilancia de áreas de difícil acceso como la selva [17].

2.8. Red de área de comunicación (CAN)

2.8.1. Can Bus Shield para Arduino: aspectos técnicos

El hardware que va a ser embebido en la placa de Arduino es un shield de marca SPARK FUN (CAN COMMUNICATION):

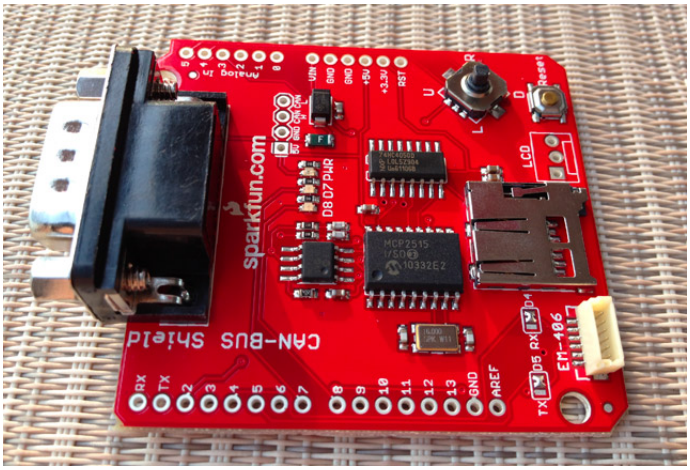


Figura 2.9: Placa CAN de marca Sparkfun
Extraído de: [17]

Permite montar la red CAN con 4 nodos incrustados en ella. Consta de las siguientes caractrísticas:

- CAN v2.0B con velocidad de 1 Mb/s (1000Kbps).
- Interfaz SPI de alta velocidad (10 MHz).
- Datos estándar y extendidos y marcos remotos.
- Conexión CAN tipo sub-D estándar de 9 vías (DB-9).
- La energía puede suministrar a Arduino por sub-D a través del fusible dedicado.
- Enchufe para el módulo del GPS EM506.
- Puerto Micro SD.
- Connector para LCD serial.
- Botón de reinicio.
- Control de navegación del menú con un joystick.
- LEDs indicadores.

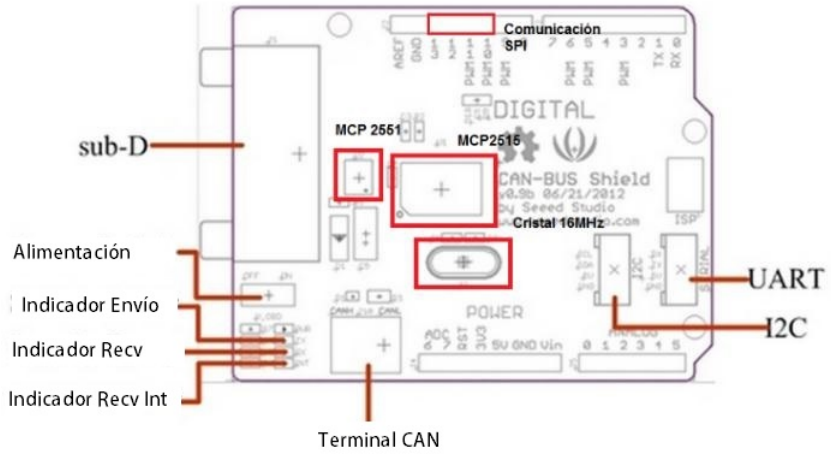


Figura 2.10: Partes del Shield CAN
Extraído de: [1]

2.8.2. Introducción

El bus CAN surge de la necesidad encontrar una forma de interconectar y conectar los distintos dispositivos de un automóvil en una sola red de una manera sencilla y reduciendo significativamente las conexiones, luego estandarizada en la norma ISO 11898-1; CAN bus cubre la capa de Enlace de datos y la Física dentro de la pila de protocolo OSI [1].

2.8.3. Capa de enlace

El protocolo de acceso es CSMA/CD + AMP (Carrier Sense Multiple Access/ Collision Detection + Arbitration on Message Priority). Bajo este protocolo, los medios se ponen a la escucha de tramas de datos enviados en la red desde cualquier nodo, evitando así enviar mensajes mientras la red está ocupada; en el caso de que se envien mensajes al mismo tiempo desde dos o mas puntos de la red, el mensaje enviado es el que tenga como emisario al del identificador más bajo; cada nodo tiene un identificador que deberá ser único, es designado por software [1].

El campo de desición es al principio de la trama y la desición de prioridad se toma al final de la misma como se ve a continuación:

2.8.4. Capa física

La capa física debe recibir y enviar mensajes a la vez; a demás de presentar el estado recesivo y dominante en los nodos. tiene tres sub capas:

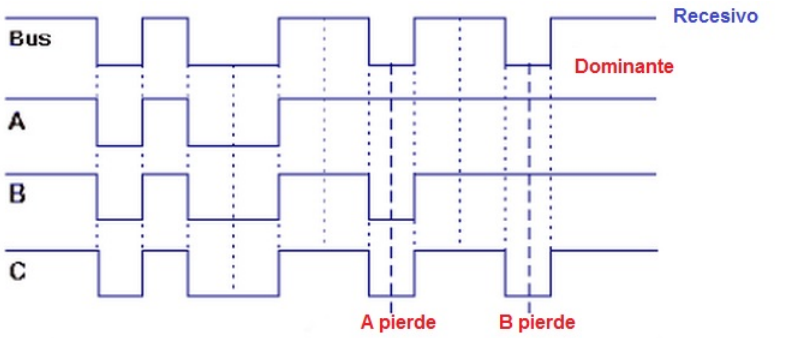


Figura 2.11: Buffer de salida. Obtenido de García Osés [1].
Extraído de: [1]

PSL

Physical Signaling Layer, sincroniza y temporiza los bits (modificación por software) [1].

PMA

Convierte los niveles lógicos de transmisión y recepción al lenguaje del protocolo [1].

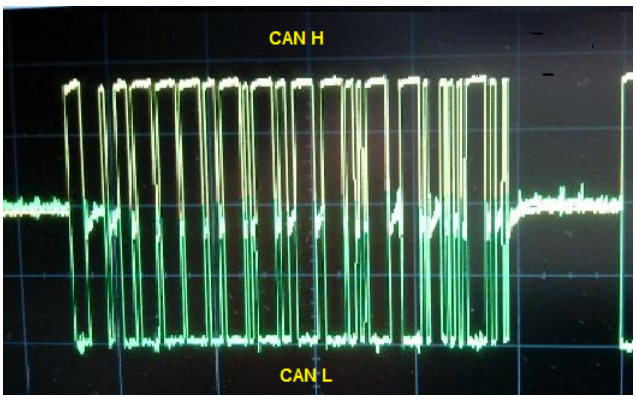


Figura 2.12: Forma de onda generada por los canales de salida CAN H y CAN L [1].
Extraído de: [1]

La diferencia entre CAN H y CAN L que provee la comunicación va de 0 a 2 volts, lo que da un nivel logico a un bit; el modo diferencial permite eliminar ruido [1].

MDI

Medium Dependent Interface, o interfaz dependiente del medio, indica como y bajo que medio se hará la transmisión [1].

- Cables con resistencias de 120 Ω .
- Cable trenzado o apantallado.
- Evitar derivaciones.

Trama de CAN bus

El siguiente gráfico representa una trama de datos de la comunicación CAN:

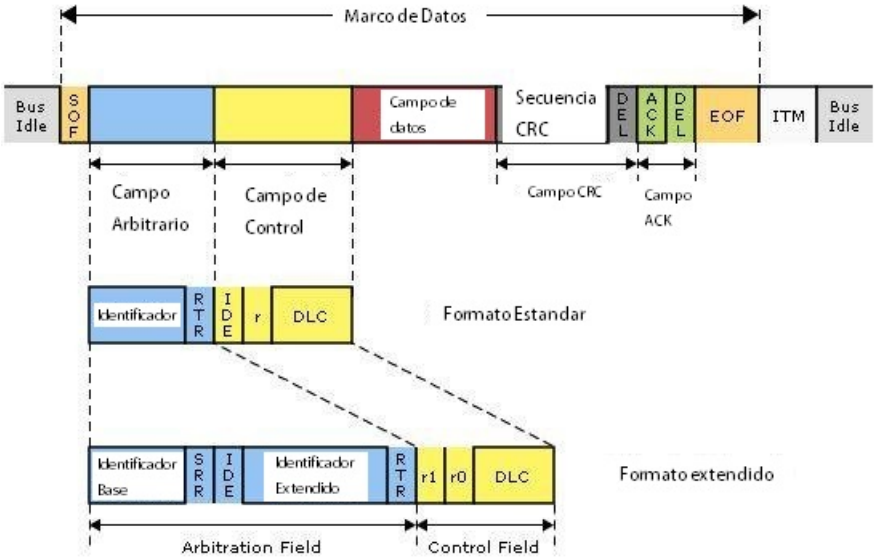


Figura 2.13: Trama generada en la comunicación CAN[1].
Extraído de: [1]

Consta de los siguientes elementos:

- SOF (Start of Frame bit) (Bit de comienzo de trama).
- Campo de arbitrio.
- Campo de control.
- Campo de datos.
- Campo de verificación por redundancia cíclica CRC.
- Campo de reconocimiento.
- Campo de fin de trama.

Trama remota

Un nodo tiene la capacidad de solicitar un mensaje de otro nodo usando tramas remotas. Luego el nodo enviará su información al solicitante; el nodo que solicite la información deberá tener el mismo identificador [1]. Este tipo de trama no será usada en el presente trabajo; en su lugar se utilizará una trama estandar de 8 bytes de datos.

2.9. Driver, encoder, motor y finales de carrera

El driver a utilizarse es DRIVER VNH2SP30, driver de hasta 16 volts. El encoder es HEDM-5500 con una resolución de 2000 puntos cada 360 (2 encoders). El motor dc tiene una potencia de 180 watts a 24 volts. Finales de carrera del tipo mecánico. Todos estos sensores y actuadores están descritos por Montalvo [18] en la tesis previa a este trabajo y están montadas en el sistema mecánico.

Capítulo 3

Metodología

3.1. Sistema de control

3.1.1. Función de Transferencia

El controlador está basado en un sistema de equilibrio en bajo o mejor llamado *Down-position* o *crank-down*; y cuya función de transferencia obtenida en la tesis previa de Montalvo [18]:

$$\ddot{x} = \frac{F - ml(\ddot{\theta}\cos\theta + \dot{\theta}^2\sin\theta) - f_c\dot{x}}{(M+m)} \tag{3.1}$$

$$\ddot{\theta} = \frac{mgl\sin\theta - ml\ddot{x}\cos\theta - f_p\dot{\theta}}{(I_p + ml^2)} \tag{3.2}$$

Cuyos parámetros estan representados en el cuadro 3.1 y son:

x_p	Coordenada X para el centro de gravedad del péndulo.
y_p	Coordenada Y para el centro de gravedad del péndulo.
x_c	Coordenada X para el centro de gravedad del carro.
y_c	Coordenada Y para el centro de gravedad del carro.
y_{pp}	Coordenada Y para el punto de pivote.
l	Distancia del centro de gravedad del péndulo al punto de giro.
θ	ángulo del péndulo desde la dirección Y positiva.
m	Masa de péndulo (carga y varilla).
M	Masa del carro.
F	Fuerza aplicada al carro.
V	Fuerza de reacción vertical del péndulo.
H	Fuerza de reacción horizontal del péndulo.
I_p	Momento de inercia para el péndulo con respecto al punto de giro.
f_p	Fricción viscosa para péndulo en el punto de pivote.
f_c	Fricción dinámica para el carro.

Cuadro 3.1: Parámetros de la función de transferencia

3.2. Punto de equilibrio del sistema

La ecuación 3.1 muestra la representación del sistema de una forma no-lineal. Equilibrio está definido como un punto (o una curva) en donde todos los estados permanecen sin cambio, sus derivadas son ceros (0). Después de que 3.1 haya sido reemplazada en 3.2 y viceversa aparece un nuevo sistema de ecuaciones:

$$\beta\ddot{x} = (I_p + ml^2)(F + ml\dot{\theta}^2 \sin\theta - f_c\dot{x}) - ml\cos\theta(mgl\sin\theta - f_c\dot{\theta}) \tag{3.3}$$

$$\beta\ddot{\theta} = (M+m)(mgl\sin\theta - f_p\dot{\theta}) - ml\cos\theta(F + ml\dot{\theta}^2\sin\theta - f_c\dot{x}) \tag{3.4}$$

$$\beta = (M+m)(I_p + ml^2) - (ml\cos\theta)^2 \tag{3.5}$$

Las condiciones de $\dot{x} = 0, \dot{\theta} = 0$ y $F = 0$ entonces las ecuaciones 3.3 y 3.4 resultan:

$$\beta\ddot{x} = -ml\cos\theta(mgl\sin\theta - f_c\dot{\theta}) = 0 \tag{3.6}$$

$$\beta\ddot{\theta} = (M+m)(mgl\sin\theta) = 0 \tag{3.7}$$

Ya que $\beta \neq 0$ los únicos puntos que satisfacen estas ecuaciones son $\theta = 0$ y $\theta = \pi$. Esto tiene sentido ya que el péndulo puede permanecer directamente abajo; así como hacia arriba si nada lo perturba.

3.2.1. Linealización en posición bajo

Si el área enfocada para el controlador en bajo es ($\theta = \pi$, $\dot{\theta}_0 = 0$), entonces tiene sentido linealizar para estos puntos, definiendo θ como una desviación de θ_π.La linealización usando las series de Taylor se muestra a continuación.

$$\sin\theta \approx \sin\theta_0 + \left.\frac{\mathrm{d}\sin\theta_0}{\mathrm{d}x}\right|_{\theta_0=\pi} * \theta + \varepsilon \approx 0 - 1 * \theta = -\theta \tag{3.8}$$

$$\cos\theta \approx \cos\theta_0 + \left.\frac{\mathrm{d}\cos\theta_0}{\mathrm{d}x}\right|_{\theta_0=\pi} * \theta + \varepsilon \approx -1 - 0 * \theta = -1 \tag{3.9}$$

Para $(\theta_0 = \pi, \dot{\theta}_0 = 0)$ las versiones de ecuaciones linealizadas de las ecuaciones 3.8 y 3.9 son:

$$\ddot{x} = \frac{F + ml\ddot{\theta} - f_c\dot{x}}{(M+m)} \tag{3.10}$$

$$\ddot{\theta} = \frac{-mgl\theta - ml\ddot{x} - f_p\dot{\theta}}{I_p + ml^2} \tag{3.11}$$

Usando las ecuaciones 3.8 y 3.9 para encontrar el espacio de estados; tomando en cuenta que para ellos los términos deben ser de bajo orden. Por tanto $\ddot{x}$ debe ser sustituido en 3.11 por $\ddot{\theta}$ de 3.10 y viceversa.

$$\ddot{\theta}(I_p + ml^2) = -mgl\theta + ml(\tfrac{F + ml\ddot{\theta} - f_c\dot{x}}{(M+m)}) - f_p\dot{\theta}$$
$$\Leftrightarrow$$
$$\ddot{\theta}(I_p + ml^2)(M+m) = (M+m)(-mgl\theta - f_p\dot{\theta}) + ml(F + ml\ddot{\theta} - f_c\dot{x})$$
$$\Leftrightarrow$$
$$\ddot{\theta}((I_p + ml^2)(M+m) - m^2l^2) = (M+m)(-mgl\theta - f_p\dot{\theta}) + ml(F - f_c\dot{x})$$
$$\Rightarrow$$
$$\text{Si } \alpha = (I_p + ml^2)(M+m) - m^2l^2, \text{ entonces:}$$
$$\Rightarrow$$
$$\ddot{\theta} = (M+m)(\tfrac{-mgl\theta}{\alpha} - \tfrac{f_p\dot{\theta}}{\alpha}) + ml(\tfrac{F}{\alpha} - \tfrac{f_c\dot{x}}{\alpha})$$

Ahora $\dot{\theta}$ deberá ser sustituido en 3.10:

$$\ddot{x}(I_p + ml^2)(M+m) = ml(-mgl\theta + ml\ddot{x} - f_p\dot{\theta}) + (F - f_c\dot{x})(I_p + ml^2)$$
$$\Leftrightarrow$$
$$\ddot{x}((I_p + ml^2)(M+m) - m^2l^2) = ml(-mgl\theta - f_p\dot{\theta}) + (F - f_c\dot{x})(I_p + ml^2)$$
$$\Rightarrow$$
$$\text{Si } \alpha = (I_p + ml^2)(M+m) - m^2l^2, \text{ entonces:}$$
$$\Rightarrow$$
$$\ddot{\theta} = ml(\tfrac{-mgl\theta}{\alpha} - \tfrac{f_p\dot{\theta}}{\alpha}) + (F - f_c\dot{x})(\tfrac{I_p}{\alpha} + \tfrac{ml^2}{\alpha})$$

Usando el vector:

$$X = [x, \dot{x}, \theta, \dot{\theta}]$$

El sistema linealizado en espacio de estados esta descrito por:

$$\begin{cases} \dot{X} = A_{down}X + B_{down}U \\ Y = CX + DU \end{cases}$$

Donde A,B,C,D son matrices y vectores definidos por espacio de estados y son:

$$A_{down} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{(I_p+ml^2)f_c}{\alpha} & \frac{-m^2l^2g}{\alpha} & \frac{-mlf_p}{\alpha} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlf_c}{\alpha} & -\frac{(M+m)mlg}{\alpha} & -\frac{(M+m)f_p}{\alpha} \end{bmatrix}$$

$$B_{down} = \begin{bmatrix} 0 \\ \frac{(I_p+ml^2)}{\alpha} \\ 0 \\ \frac{ml}{\alpha} \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad D = 0 \quad U = F$$

3.2.2. Matriz numérica con parámetros del sistema

Tomando los siguientes parámetros descritos por Montalvo [18] para las constantes del sistema mostrados en 3.2 :

$M =$	0.972 Kg
$m =$	0.144 Kg
$L =$	0.38 m
$I_p =$	0.02627 Kg/rad
$f_p =$	0.01 Ns/rad
$f_c =$	1.1 Ns/m

Cuadro 3.2: Constantes del sistema

Las matrices obtenidas son las siguientes:

$$A_{down} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -06498 & -0{,}5930 & -0{,}011 \\ 0 & 0 & 0 & 1 \\ 0 & -1{,}21 & -12{,}08 & -0{,}2253 \end{bmatrix}$$

$$B_{down} = \begin{bmatrix} 0 \\ 4{,}04 \\ 0 \\ 1{,}10 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad D = 0 \quad U = F$$

3.3. Controlador del sistema en lazo cerrado

El siguiente algoritmo está desarrollado en Matlab para obtener las ganancias para sistemas dinámicos en lazo cerrado:

```
%% 1. STATE FEEDBACK DISCRETE TIME CONTROLLER
clear all, close all, clc,

A=[0 1 0 0;0 -0.6498 -0.5930 -0.011; 0 0 0 1; 0 -1.21 -12.08 -0.2253];
B=[0;4.04;0;1.10];
C=[1 0 0 0; 0 1 0 0 ;0 0 1 0;0 0 0 1];
D=[0;0;0;0];

tTop = 0.3; %maximum time for simulation
h = 0.01; %sampling period (10msec)
tick = 0.01; %granularity

sys = ss(A, B, C, D);
sys_d = c2d(sys, h);

pole1 = -5+25j; %poles in continous time
pole2 = -5-25j;

z1 = exp(pole1*h); %poles in discrete time
z2 = exp(pole2*h);

K = acker(sys_d.a, sys_d.b, [z1 z2]); %controller design
CONTROLLER = K
```

3.3.1. Ganancias obtenidas

Estas son las ganancias obtenidas a partir del algoritmo de Matlab: $G1 = 59{,}86$ (Ganancia de la posición del carro).
$G2 = 17{,}9$ (Ganancia de la velocidad del carro).
$G3 = -3{,}17$ (Ganancia de la posición del péndulo).
$G4 = -10{,}713$ (Ganancia de la velocidad del péndulo).
$G5 = 0{,}9345$ (Ganancia del error integral).

3.4. Diseño del sistema de control electrónico y de comunicación

3.4.1. Can Bus Shield

El shield descrito en el capítulo anterior, conectará todos los nodos a continuación descritos.

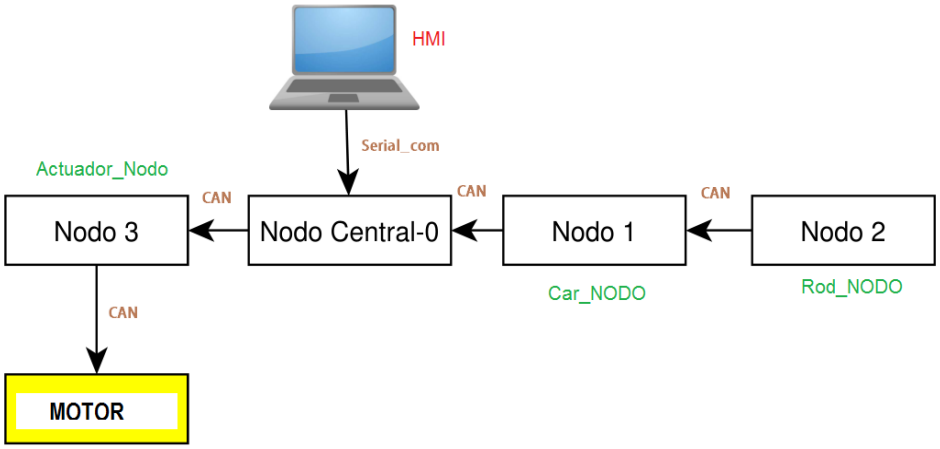


Figura 3.1: Diagrama general de la red

3.4.2. Diseño lógico general de red

Se prevee un esquema general de la red con todos sus nodos como se muestra en la Fig. 3.1. Se utiliza el IDE de Arduino para el desarrollo del control de todos los nodos. El nodo primordial es el así llamado nodo central o nodo 0.

3.4.3. Diseño lógico de la estructura de la red: Nodo 0

En esté nodo se realizarán las acciones de control en base a la información proveniente del nodo 2; se encapsulará la acción de control y se enviará al nodo 3; también tendra la importante función de servir de enlace entre la HMI y la red CAN enviando y recibiendo información desde y hacia la PC, por tanto este nodo es el que más carga de trabajo tendrá, ya que experimentará dos tipos de comunicación: CAN y serial y la misión de calcular el controlador para el sistema Fig. 3.2.

Máquina de estado Nodo 0

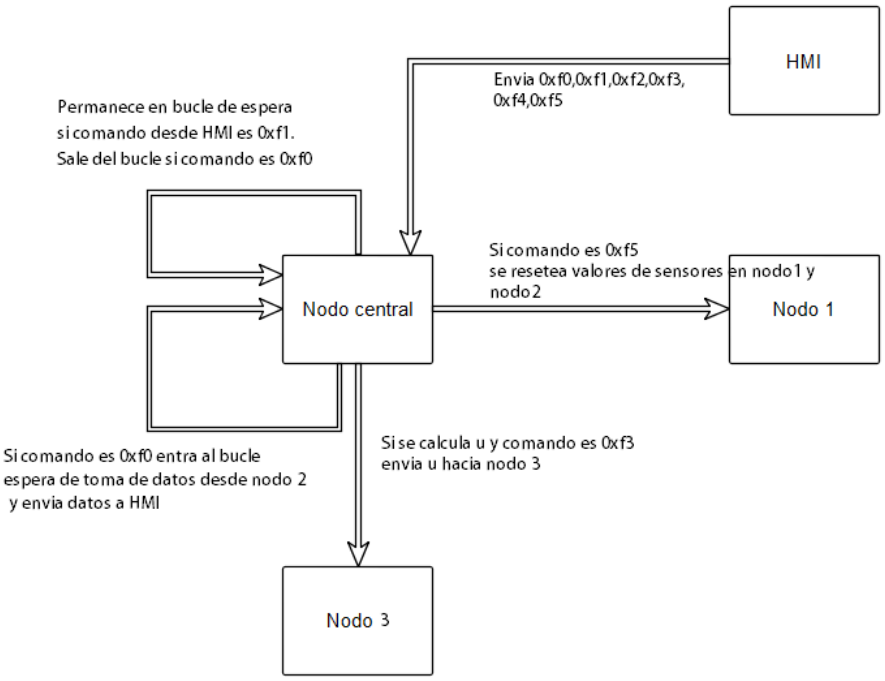


Figura 3.2: Diagrama de máquina de estado del Nodo central

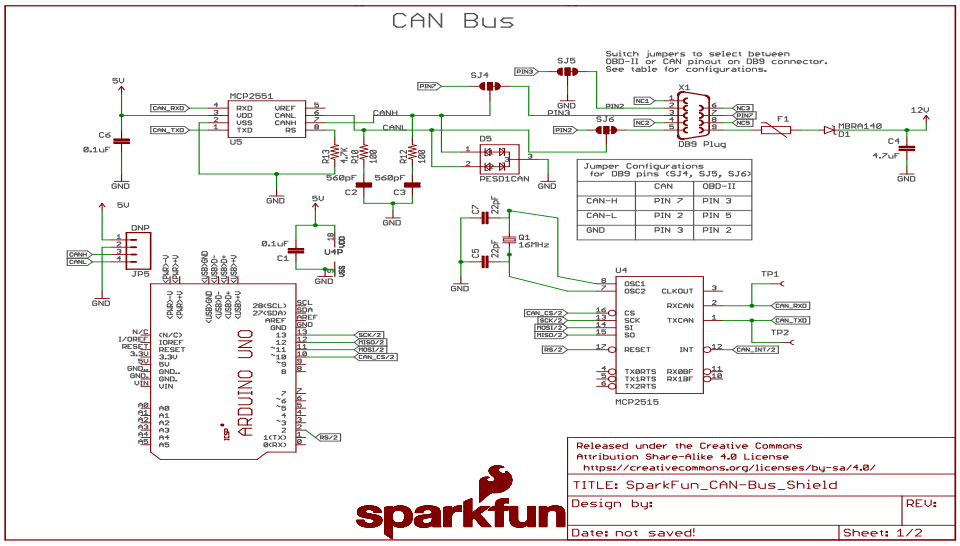
Controlador implementado en el nodo 0

El controlador del sistema implementado en el nodo 0 está basado en la ecuación 2.1 y es:

```
u = -gain1*cartPos - gain2*cartVel - gain3*rodPos - gain4*rodVel + gain5*error;
```

Sistema eléctrico del nodo 0

A continuación se detalla el sistema de conexión del sistema eléctrico del nodo 0 en la Fig. 3.3:



11/2/2017 19:10 C:\EAGLE-7.6.0\lbr\SparkFun_CAN-Bus_Shield.sch (Sheet: 1/2)

Figura 3.3: Diagrama eléctrico del nodo 0

3.4.4. Diseño lógico de la estructura de la red: Nodo 1

Máquina de estado Nodo 1

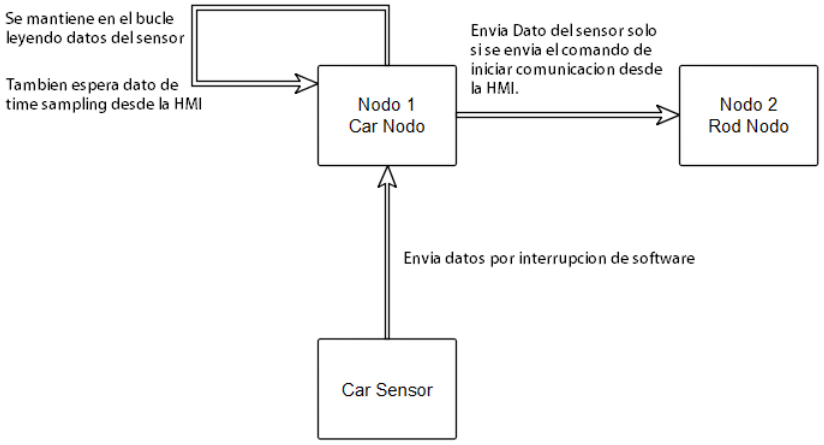
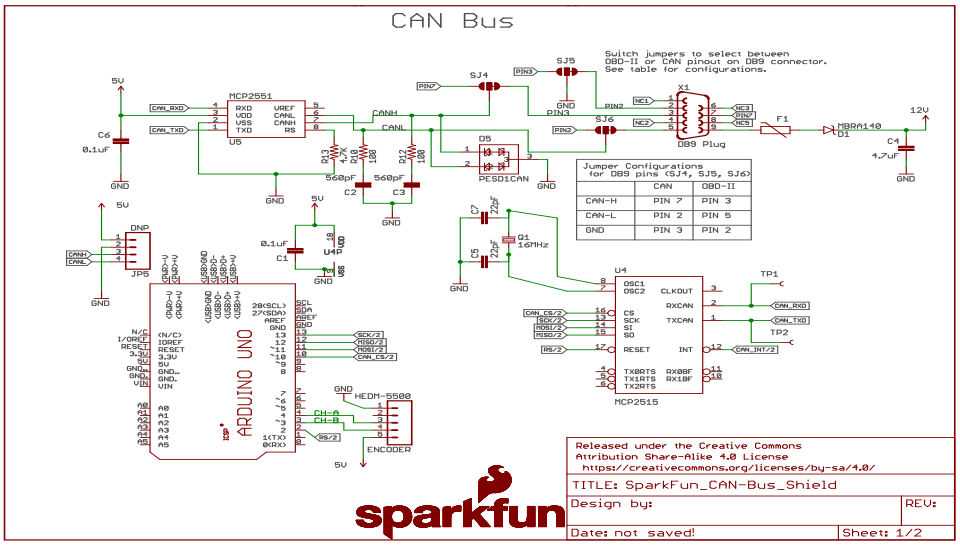


Figura 3.4: Máquina de estado del Nodo 1

En esté nodo estará conectado el sensor ubicado en el motor con una particularidad, en este nodo se disparará las banderas y la toma de datos del sensor cada ciclo de muestreo (ts) especificado en la HMI, que tendrá que ser enviada primero al nodo central y luego a este nodo; por defecto el nodo tiene un tiempo de muestreo cada 10 (ms), siendo el rango permitido desde 10ms hasta 50 ms. En éste nodo sólomente se toman datos y se envian al nodo 2 como se muestra en la Fig. 3.4.Todas las librerias usadas se encuentran disponibles en la página oficial de Arduino o en su defecto en la de terceros.



11/2/2017 19:11 C:\EAGLE-7.6.0\lbr\SparkFun_CAN-Bus_Shield.sch (Sheet: 1/2)

Figura 3.5: Diagrama eléctrico del nodo 1 y nodo 2

Sistema eléctrico del nodo 1

Se detalla el sistema de conexión del sistema eléctrico del nodo 1 en la Fig. 3.5:

3.4.5. Diseño lógico de la estructura de la red: Nodo 2

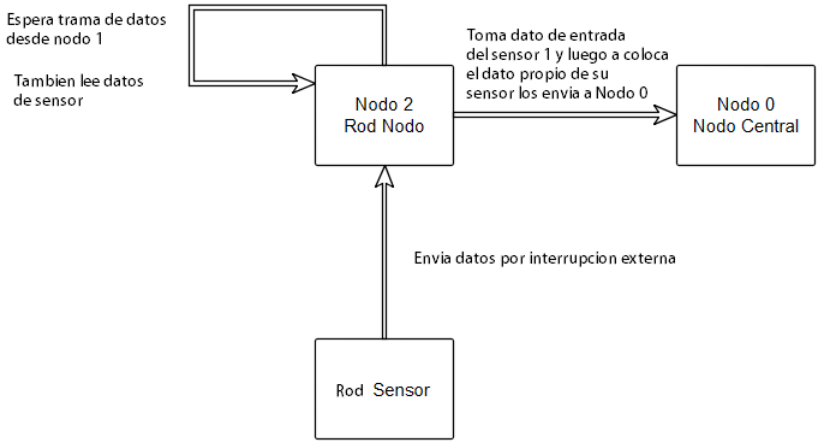


Figura 3.6: Máquina de estado del Nodo 2

Máquina de estado del Nodo 2

En este nodo se toman datos del sensor ubicado en el péndulo con una particularidad; sólo se disparará el envio de datos al nodo central cada vez que le llegue información del nodo 1; luego éste tomará el dato del sensor, lo encapsulará, lo pondrá en la cola a

continuación de los datos del nodo 1 que pasan directamente de la entrada a la salida del buffer de la red y enviará ambos datos al nodo central, como se muestra en la Fig. 3.6.

Sistema eléctrico del nodo 2

La Fig. 3.5 muestra la conexión a utilizarse en el nodo 2, la cual es similar al nodo 1.

3.4.6. Diseño lógico de la estructura de la red: Nodo 3

Máquina de estados del Nodo 3

Nodo del motor: Nodo 3

En este nodo se convierte la información a cerca del controlador enviada desde el nodo central (Fig. 3.7).

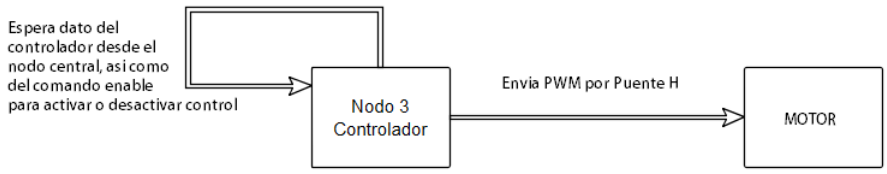


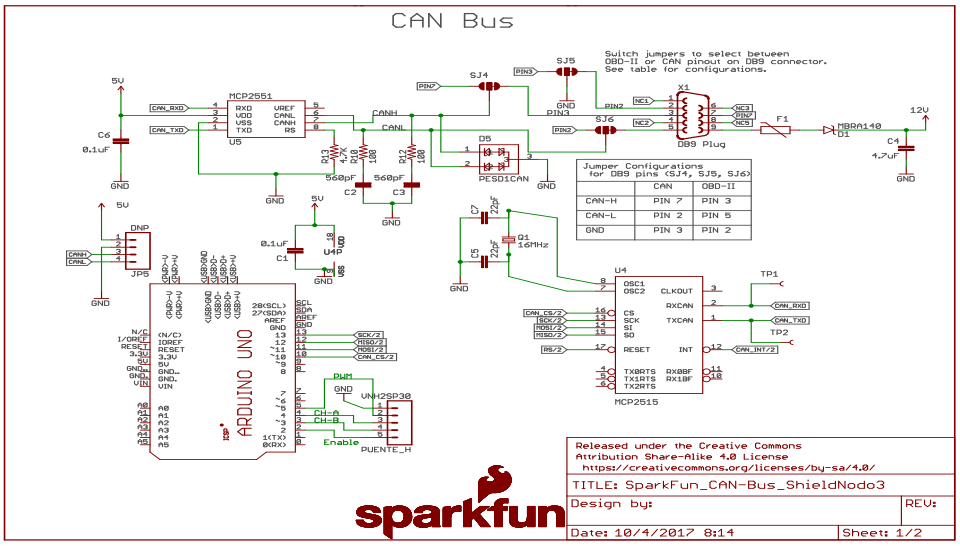
Figura 3.7: Máquina de estados del Nodo 3

Sistema esquemático eléctrico del nodo 3

Sistema de conexión eléctrico del nodo 3 en la Fig. 3.8:

3.4.7. Red física CAN e interconexión con la HMI

En esta sección se muestra la interconexión de la red CAN con la HMI, así como de los sensores implementados en la red y el puente H del sistema como se muestra en la Fig.3.9.



10/4/2017 8:14 C:\Users\asus\Dropbox\Noveno\tesis\Contenedor inteligente\documentos\software\Tesis_mauricioHinojosa\SparkFun_CAN-Bus_ShieldNodo3.sch (Sheet: 1/2)

Figura 3.8: Diagrama eléctrico del nodo 3

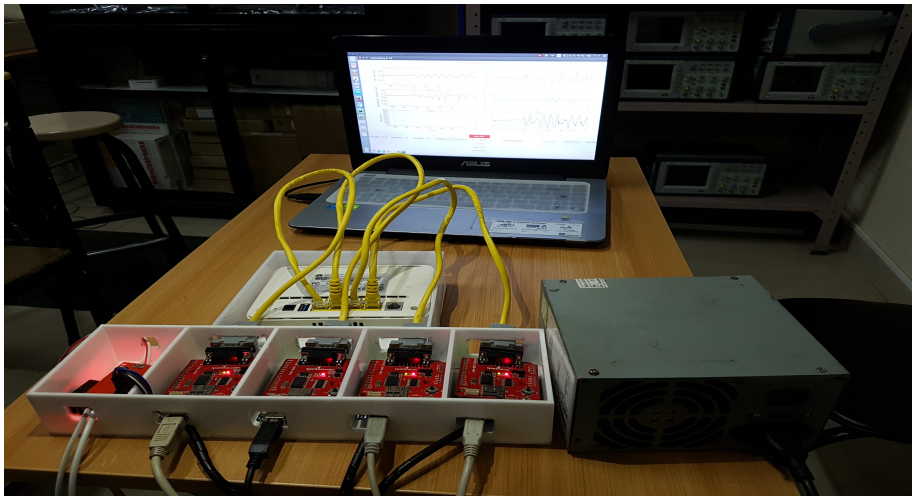


Figura 3.9: CAN y HMI

3.5. Diseño de la HMI

3.5.1. Instalación de Python en Ubuntu

La instalación de Python dentro de Ubuntu no tiene ninguna complicación:

- Se debe abrir una terminal **(Ctrl + Alt + t)**.
- Con la correspondiente conexión a internet y se debe escribir en consola lo siguiente:
  - `sudo add-apt-repository ppa:fkrull/deadsnakes`
  - `sudo apt-get update`

- `sudo apt-get install python2.7`

- Por último:

```
sudo apt-get update

sudo apt-get upgrade
```

3.5.2. Instalación de PyCharm

PyCharm es un IDE, que tiene una version paga y una versión (community) de libre de desarrollo la cual será implementada para el desarrollo del sistema HMI, y utiliza como estructructura y lenguaje de desarrollo el antes instalado Python 2.7.2, entre algunos otros lenguajes; y contiene varias herramientas de desarrollo, como ya se explicó en secciones anteriores; logo de PyCharm en la Fig. 3.10.

Figura 3.10: Logo identificativo de Pycharm
Extraído de: https://cdn.worldvectorlogo.com/logos/pycharm.svg

La instalación de PyCharm dentro de Ubuntu no tiene ninguna complicación:

- Se debe abrir una terminal.

- Con la correspondiente conexión a internet y escribir en la consola lo siguiente:

  - `sudo add-apt-repository ppa:mystic-mirage/pycharm`

  - `sudo apt-get update`

  - `sudo apt-get install pycharm-community`

- Al final :

```
sudo apt-get update

sudo apt-get upgrade
```

3.5.3. Desarrollo del software para la interfaz humano-máquina (HMI)

EL desarrollo del software de supervisión de la HMI, partió de un software básico de intercambio de información con un Arduino programado en lenguaje **Python 2.7.2** mostrado en los anexos; en el que el programa se envía dos datos y el arduino respondía devolviendo los mismos datos cada cierto tiempo, para graficarlos en la HMI. Basados en esta premisa se tuvo principalmente tres etapas de desarrollo: expansión el buffer del puerto Usb para el envío de datos, desarrollo de los sistemas gráficos y de procesamiento de información y el desarrollo de controles para el HMI.

3.5.4. Expansión el buffer del puerto Usb

Al principio, con el progama base de python, la información que se envía desde la HMI es de dos datos por el buffer de salida hacia el nodo central; se procedió a expandirlo hasta que se puedan enviar los datos necesarios para el control desde la HMI, modificando la sección del codigo que se encarga del montaje de datos en el buffer (ambos códigos, el base y el programa final modificado están en la sección Anexo B y Anexo C, respectivamente), dando como resultado 10 datos seccionados en 4 bytes, con un total 40 bytes de información que se envían a la targeta de Arduino que corresponde al nodo 0 conectado por Usb a la Pc.

3.5.5. Desarrollo de los sistemas gráficos y de procesamiento de información

Se contaba con una sóla gráfica implementada en el programa origen, de la cual se partió para conseguir 6 gráficas, de las que la primera representa la posición del sensor del carro, en la segunda muestra la velocidad de cambio de posición del carro; la tercera datos de posición del sensor del péndulo; en la cuarta se regresenta la variación de velocidad del sensor pendular; la quinta el tiempo de procesamiento que lleva el nodo 0 desde su arranque en microsegundos y la sexta representa el controlador.

3.5.6. Procesamiento de información y el desarrollo de controles

Se implentementó todo tipo de herramientas que permiten, desde la creación de botones, cuadros de texto y modificadores gráficos que permiten la manipulación externa de la HMI, hasta la creación de variables que guardan información o se utilizan para el envío y recepción de datos. También se añadieron las clases y objetos que permiten manipular la HMI internamente junto con librerias especiales como la de la Usb o la librería para el empaquetado de datos (todas las librerías incluidas en el IDE tool de Python) y estructuras lógicas o matemáticas necesarias para el desarrollo de la interfaz.

3.5.7. Máquina de estado de la HMI

En la gráfica 3.11 se representa la máquina de estado de la hmi.

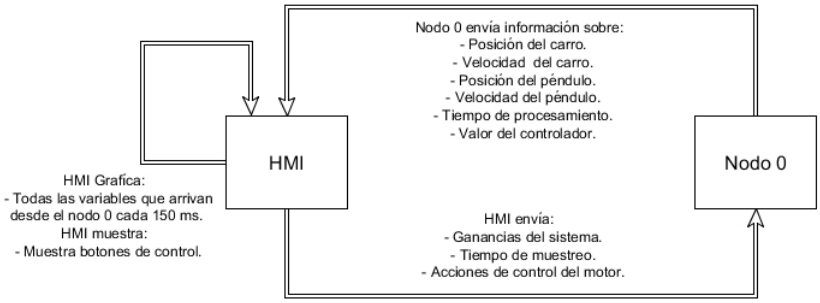


Figura 3.11: Máquina de estado de la HMI

3.5.8. Ventana HMI terminada

Concluida la programación se tienen dos ventanas; la primera es una pequeña ventana que permite seleccionar entre diferentes puertos Usb, posibles en Ubuntu como muestra la Fig. 3.12.

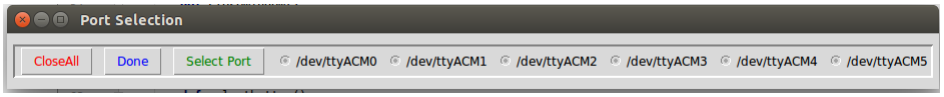


Figura 3.12: Ventana de selección Usb

La segunda ventana es la HMI utilizar en el sistema como muestra Fig. 3.13, consta de las partes:

1. Gráfica sensor 1.
2. Gráfica de la variación de velocidad sensor 1.
3. Gráfica sensor 2.
4. Gráfica de la variación de velocidad sensor 2.
5. Gráfica del tiempo de procesamiento en microsegundos.
6. Botones y texto de información y modificación de los valores de control del péndulo.
7. Barra de manipulación de las gráficas.

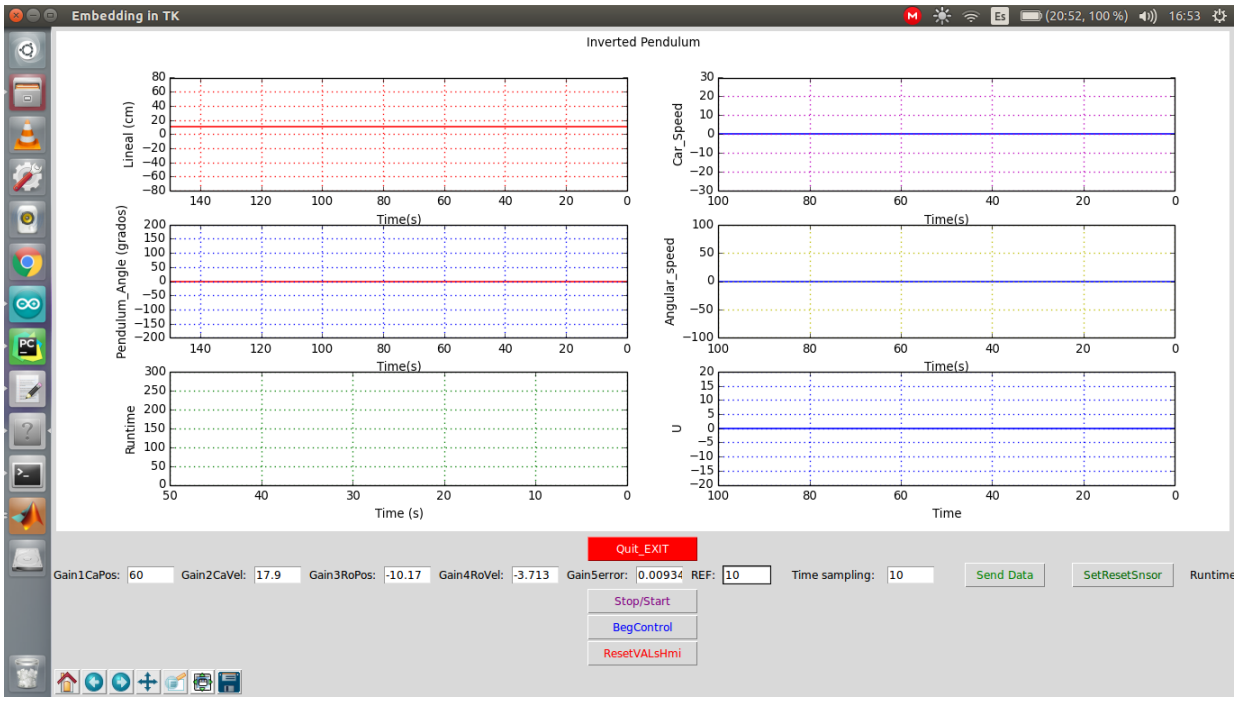


Figura 3.13: HMI

3.6. Trámas de datos enviados desde HMI hacia el nodo central

Esta es la trama de datos enviado por la HMI hacia el nodo central:

```
([command.direccion, ord(value0[0]), ord(value0[1]), ord(value0
    [2]), ord(value0[3]),
        ord(value1[0]), ord(value1[1]), ord(value1[2]), ord(
            value1[3]),
        ord(value2[0]), ord(value2[1]), ord(value2[2]), ord(
            value2[3]),
        ord(value3[0]), ord(value3[1]), ord(value3[2]), ord(
            value3[3]),
        ord(value4[0]), ord(value4[1]), ord(value4[2]), ord(
            value4[3]),
        ord(value5[0]), ord(value5[1]), ord(value5[2]), ord(
            value5[3]),
        ord(value6[0]), ord(value6[1]), ord(value6[2]), ord(
            value6[3]),
        ord(value7[0]), ord(value7[1]), ord(value7[2]), ord(
            value7[3]),
        ord(value8[0]), ord(value8[1]), ord(value8[2]), ord(
            value8[3]),
        ord(value9[0]), ord(value9[1]), ord(value9[2]), ord(
            value9[3]),0x0A])
```

Contiene 42 bytes estructurados en una sola trama de datos que controla el sistema en los nodos arduinos.

Donde: value0= estructura en 4 bytes el valor de la ganancia G1.

value1= estructura en 4 bytes el valor de time sampling.
value2= estructura en 4 bytes el valor del *on/off* del sistema de comunicación.
value3= estructura en 4 bytes el valor del reset de sensores.
value4= estructura en 4 bytes el valor de la ganancia G2.
value5= estructura en 4 bytes el valor de la ganancia G3.
value6= estructura en 4 bytes el valor de la ganancia G4.
value7= estructura en 4 bytes el valor de la ganancia G5.
value8= estructura en 4 bytes el valor de la referencia.
value9= estructura en 4 bytes el valor de enable del control (enciende o paga el controlador).

El valor de ***command.direccion*** permite seleccionar las acciones en los nodos:

- Si command.direccion =0xf1 entonces se apagan y cierran comunicaciones con HMI.
- Si command.direccion =0xf0 entonces se detienen o reinician comunicaciones con HMI.
- Si command.direccion =0xf2 entonces se recepcionan ganancias variables para el controlador.
- Si command.direccion =0xf3 entonces se activa o desactiva el controlador.
- Si command.direccion =0xf4 entonces se resetean valores tanto en nodo central como en HMI.
- Si command.direccion =0xf5 entonces se envia mensaje para resetear sensores.

El último valor (0x0A) es estandar para final de trama.

3.7. Trámas de datos enviados y recibidas entre nodos

3.7.1. Nodo 0

En la sección anterior se mostró la trama de datos enviada desde la HMI hacia el nodo central (nodo0).

3.7.2. Nodo 0 a HMI

La siguiente trama 3.3 de datos representa los bytes enviados desde el nodo central hacia la HMI.

La trama 3.3 está compuesta por 28 bytes donde:
cartposHmi= estructura en 4 bytes el valor de la posición del carro.

Byte	1	2	3	4
Value	pointer-cartposHmi0	pointer-cartposHmi1	pointer-cartposHmi2	pointer-cartposHmi3
	5	6	7	8
	pointer-rodposHmi0	pointer-rodposHmi1	pointer-rodposHmi2	pointer-rodposHmi3
	9	10	11	12
	pointer-carVelHmi0	pointer-carVelHmi1	pointer-carVelHmi2	pointer-carVelHmi3
	13	14	15	16
	pointer-rodVelHmi0	pointer-rodVelHmi1	pointer-rodVelHmi2	pointer-rodVelHmi3
	17	18	19	20
	pointer-rtime0	pointer-rtime1	pointer-rtime2	pointer-rtime3
	21	22	23	24
	pointer-u0	pointer-u1	pointer-u2	pointer-u3
	25	26	27	28
	pointer-eof0	pointer-eof1	pointer-eof2	pointer-eof3

Cuadro 3.3: Trama de datos Nodo Central-HMI

rodposHmi0= estructura en 4 bytes el valor de la posición del péndulo.
cartVelHmi= estructura en 4 bytes el valor de la velocidad del carro.
rodVelHmi0= estructura en 4 bytes el valor de la velocidad del péndulo.
u=estructura en 4 bytes el valor del controlador. eof= end of line.

3.7.3. Nodo 0 a Nodo 1

La siguiente trama 3.4 de datos representa los bytes enviados desde nodo 0 a nodo 1: La trama 3.4 está compuesta por 8 bytes donde:

Byte	1	2	3	4
Value	pointer-timesampl0	pointer-timesampl1	pointer-timesampl2	pointer-timesampl3
	5	6	7	8
	pointer-auxsensores0	pointer-auxsensores1	pointer-auxsensores2	pointer-auxsensores3

Cuadro 3.4: Trama de datos Nodo Central- Nodo 1

timesampl= estructura en 4 bytes el valor del tiempo de muestreo.
auxsensores= estructura en 4 bytes el valor de un auxiliar que permite el reseteo de sensores.
Si auxsensores= 1 entonces se envian los datos de sensor al nodo 2.
Si auxsensores= 2 entonces se resetea el valor del encoder.

3.7.4. Nodo 0 a Nodo 3

La siguiente trama 3.5 de datos representa los bytes enviados desde nodo 0 a nodo 3: La trama 3.5 está compuesta por 8 bytes donde:

Byte	1	2	3	4
Value	pointer-senduCan0	pointer-senduCan1	pointer-senduCan2	pointer-senduCan3

Cuadro 3.5: Trama de datos Nodo Central - Nodo 3

senduCan= estructura en 4 bytes el valor del controlador(-16:+16).

3.7.5. Nodo 1 a Nodo 2

La siguiente trama 3.6 de datos representa los bytes enviados desde el nodo 1 a nodo 2: La trama 3.6 está compuesta por 8 bytes donde:

Byte	1	2	3	4
Value	pointer-sensorvalue0	pointer-sensorvalue1	pointer-sensorvalue2	pointer-sensorvalue3
	5	6	7	8
	pointer-auxsensores0	pointer-auxsensores1	pointer-auxsensores2	pointer-auxsensores3

Cuadro 3.6: Trama de datos Nodo 1 - Nodo 2

sensorvalue=estructura en 4 bytes el valor del sensor encoder del carro.
auxsensores= estructura en 4 bytes el valor de un auxiliar que permite el reseteo de sensores.
Si auxsensores= 1 entonces se envian los datos de sensor al nodo 2.
Si auxsensores= 2 entonces se resetea el valor del encoder.

3.7.6. Nodo 2 a Nodo 0

La siguiente trama 3.7 de datos representa los bytes enviados desde el nodo 2 a nodo 0: La trama 3.7 está compuesta por 8 bytes donde:

Byte	1	2	3	4
Value	pointer-bufCan= 0	pointer-bufCan1	pointer-bufCan2	pointer-bufCan3
	5	6	7	8
	pointer-rodPos0	pointer-rodPos1	pointer-rodPos2	pointer-rodPos3

Cuadro 3.7: Trama de datos Nodo 2 - Nodo 0

bufCan= Guarda la lectura del buffer desde CAN y lo pone dirección al nodo 0.

Capítulo 4

Pruebas y análisis de resultados

4.1. Prueba de adquisición de datos con el sensor HEDM-5500

De esta prueba se obtuvo la siguiente gráfica con una variación de 2000 puntos de resolución del sensor y 1500 muestras, como se muestra en la Fig.4.1:

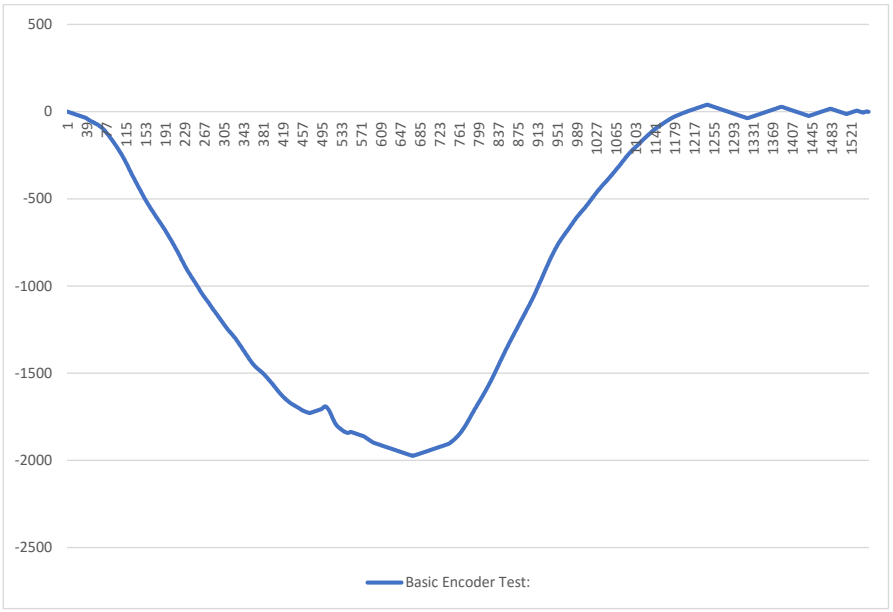


Figura 4.1: Diagrama de adquisión de datos del sensor HEDM-5500

4.1.1. Análisis de los datos obtenidos de los sensores

Se puede observar una correlación entre los datos obtenidos en la interfaz serial de Arduino IDE (Fig. 4.2) con los puntos de resolución del sensor mostrados en HMI Fig. 4.3; cabe resaltar que estos datos fueron enviados a través de CAN hacia el nodo central y luego a la HMI.

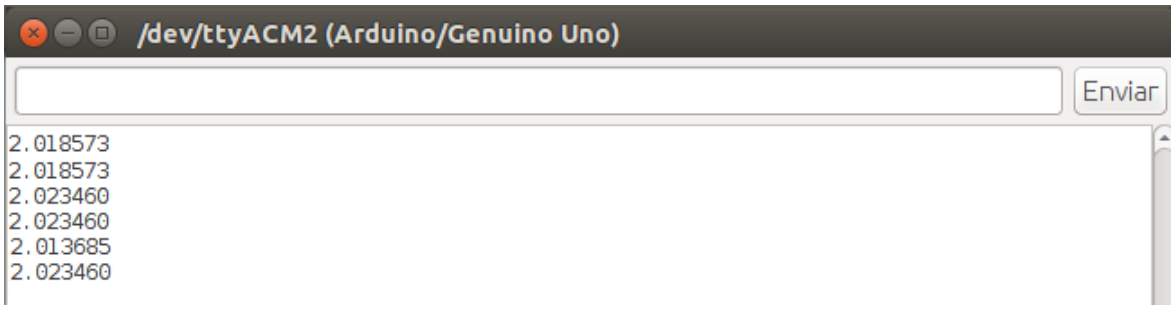


Figura 4.2: Datos obtenidos por Arduino IDE

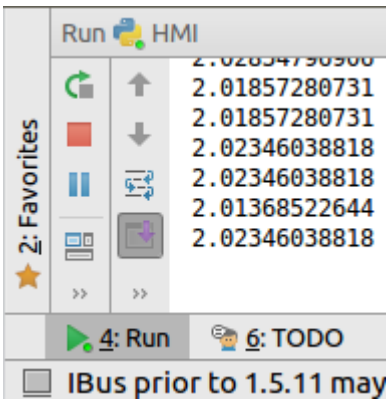


Figura 4.3: Datos obtenidos por HMI

4.2. Prueba del sistema simulado de control

4.2.1. Prueba del controlador en estado estable para el sistema desarrollado en Matlab

En el capítulo anterior se mostró la utilidad de Matlab a la hora de obtener ganancias de un sistema en espacio de estados, también se puede añadir el siguiente código para obtener la simulación en condiciones iniciales dadas y el tracking a una señal de entrada:

```
%% 2. SIMULATION IN FRONT OF GIVEN CONDITIONS
x = [1; 1]; %given initial conditions
history = []; %buffer initialization
history = [history sys_d.c*x];

for i = 0 : tick : tTop
    u = -K*x; %applies control
    x = sys_d.a*x+sys_d.b*u;
    y = sys_d.c*x;
```

```
    history = [history y]; %updates buffer
end

time = 0:tick:tTop+tick;
figure
plot(time, history(1,:));
xlabel('Time [s]');
ylabel('x[1]');
legend('State');
axis([0 tTop -1 1])

%% 3. SIMULATION TRACKING A REFERENCE SIGNAL
tTop = 5;
x = [0; 0]; %given initial conditions

history = []; %buffer initialization
history = [history x];

ref = 0;
real_ref = ref - 0.5;
refHist = [];
refHist = [refHist, ref];

u = 0;
controlHist = [];
controlHist = [controlHist, u];

for i = 0 : tick : tTop
    if mod(i,1) == 0 && i ~= 0
        ref = ~ref;
        real_ref = ref - 0.5; %to simulate the behavior in the
            flexboards
    end
    u = K(1)*(real_ref-x(1)) - K(2)*x(2);
    x = sys_d.a*x+sys_d.b*u;
    history = [history x]; %updates buffer
    refHist = [refHist real_ref];
    controlHist = [controlHist u];
end

time = 0:tick:tTop+tick;

figure
hold on
plot(time, history(1,:));
plot(time, refHist(1,:),'color', 'red')
plot(time, controlHist(1,:),'color', 'green')
xlabel('Time [s]');
ylabel('Voltage');
legend('State','Reference','Control');
axis([0 tTop -2 2])
```

4.2.2. Resultados adquiridos de Matlab

A continuación se muestran los datos obtenidos de Matlab

```
Crank system is controllable!
Convergence was reached at
iter =  2194
Crank control gains:
K_ = [ 72.49, 25.91, 14.25, -2.096]
KI_ = 0.9615
eigenvalues =    0.9669 + 0.0000i
   0.9817 + 0.0277i
   0.9817 - 0.0277i
   0.9920 + 0.0303i
   0.9920 - 0.0303i
```

4.2.3. Gráfico obtenido de la simulación

Dando como resultado que el sistema sea completamente controlable como muestra la Fig. 4.4:

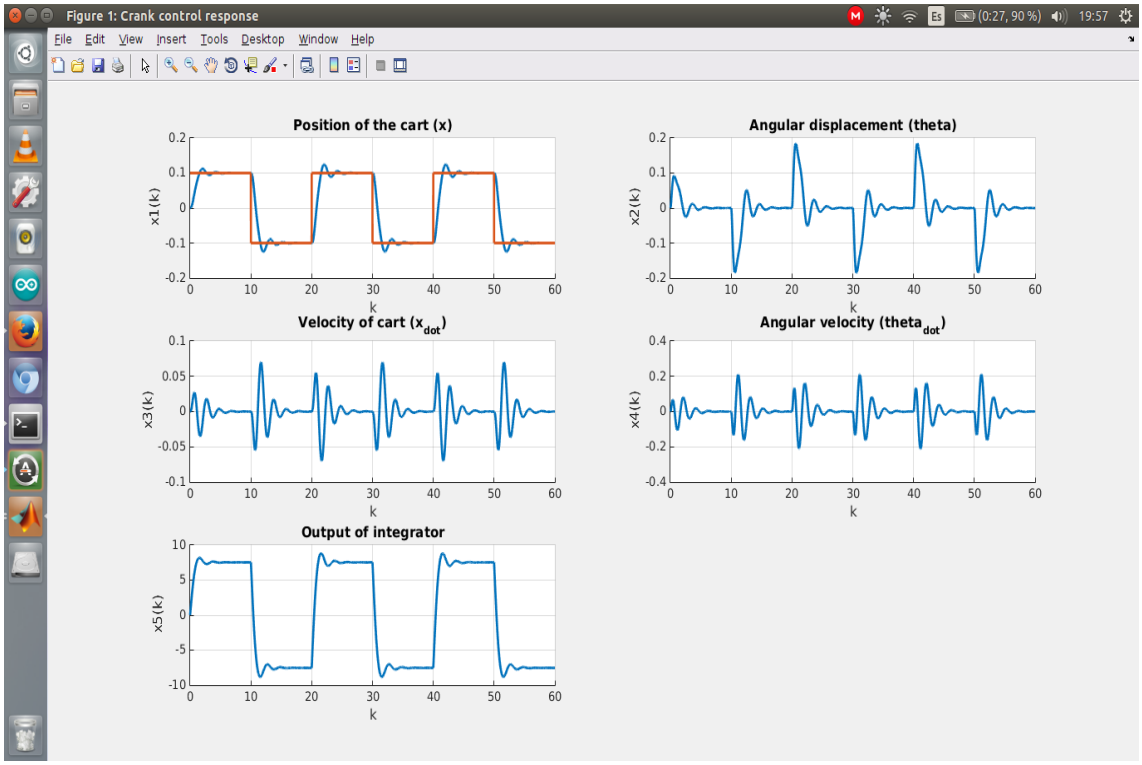


Figura 4.4: Simulación obtenida con Matlab

4.2.4. Análisis de los resultados

Los resultados obtenidos por Matlab muestran que el sistema es controlable, con una itinerancia de software de 2149; donde se encontraron los polos del sistema así como sus ganancias, las cuáles fueron puestas en un sistema de realimentación cerrado y simulado por Matlab Fig. 4.4.

4.3. Prueba del sistema real con las ganancias obtenidas

Prueba del sistema real con las ganancias obtenidas en el capítulo anterior en donde fueron modificadas para reducir el tiempo de estabilización en estado estable, dando la siguiente tabla 4.1:

Prueba	G1CarPos	G2CarVel	G3RosPos	G4RosVel	G5error	Tiempo Estabilizacion(seg)
1	59.86	17.9	10.17	-3.713	0.98	Inestable
2	59.86	17.9	-10.17	-3.713	0.15	20
3	59.86	17.9	-40.17	-3.713	0.15	14
4	59.86	17.9	-40.17	-10.713	0.15	13
5	59.86	17.9	-40.17	-1.713	0.15	15
6	59.86	17.9	-40.17	-5.713	0.15	15
7	59.86	17.9	-40.17	-7.713	0.15	14
8	59.86	17.9	-40.17	-20.713	0.15	11
9	59.86	17.9	-40.17	-10.713	0.1	14
10	59.86	17.9	-40.17	-3.713	0.08	12
11	59.86	17.9	-3.17	-3.713	0.1	20
12	59.86	17.9	-40.17	-3.713	0.25	Inestable

Cuadro 4.1: Tabla comparativa de ganancias

4.3.1. Análisis de los ganancias modificadas en el sistema real

Los datos de la tabla 4.1 muestra que las ganancias más críticas son G3RosPos y G5error; tomando la primera valores menores a 0 el sistema se vuelve inestable y para la segunda con valores mayores a 1.5 sucede lo mismo. El mejor tiempo de estabilización se obtuvo con G3RosPos=-40.17,G2CarVel=-20.713 y G5error =0.15 con un tiempo de estabilización de 11 seg.

4.4. Análisis costos

A continuación se detallan los costos del trabajo realizado en la tabla 4.2:

Egresos	Mes1 USD	Mes2 USD	Mes3 USD	Mes4 USD	Mes5 USD	Mes6 USD	Total USD
Materia prima	200				200		400
Mano de obra directa					20		20
Mano de obra indirecta							0
Materiales de oficina	10	10	10	5	20	5	60
						Costo	480
						Precio	520

Cuadro 4.2: Tabla de costos

Capítulo 5

Conclusiones y Recomendaciones

5.1. Conclusiones

1. Los documentos recopilados referentes al control por red mostraban únicamente el envio y recepción de varios tipos de datos a través de una red sin que estos documentos estén relacionados al diseño de controladores; por otra parte, cuando se encontraba documentación relacionada al desarrollo de controladores para péndulos, era dentro de un mismo microprocesador con el que se adquiria, procesaba, calculaba y se tomaba una acción de control. Por tanto en base a que no se encontró información completa para desarrollar este trabajo se tomaron en cuenta 4 fundamentos ampliamente desarrollados en otras tesis y son: adquisición y procesamiento de datos, redes CAN, HMI's y diseño de controladores.

2. Los requerimientos necesarios para el sistema fueron tomados de documentos en los que se realizaba el control en sistemas de péndulos invertidos pero la información relacionada a la implementación de controladores en una red de campo de datos era prácticamente nula, por lo que la misma fué especialmente diseñada y desarrollada para la planta.

3. La Fig. 4.4 es resultado de introducir los parámetros y constantes de la planta, que están especificados en la seccion 3.2.2, en el código desarrollado en Matlab para obtener el controlador diseñado bajo la técnica de *espacio de estados*. Como se muestra en esta simulación, el control cumple con la condición de estabilizar el sistema; el proceso antes mensionado se puede ver claramente en la gráfica que representa la posición del carro, a la que se le sometió una entrada escalón (línea naranja) y se produjo una respuesta en la posición simulada del carro (línea azul). Básicamente, la respuesta tiene una variación que sobrepasa el 10 % de la entrada escalón (aceptable en términos de diseño de controladores teóricos). Las ganancias obtenidas se utilizaron de base en el sistema real.

4. La adquisición, monitoreo y control de datos de la planta se muestra en la Fig. 3.13. La interfaz tiene dos partes; la parte superior, desarrollada para la adquisicón y monitoreo de datos, representa las gráficas de las variables del sistema y está seccionado en 6 siendo: la posición del carro y su velocidad (superior izquierda y superior derecha); la posición del péndulo y su derivada (medial izquierda y medial

derecha); tiempo de procesamiento (línea amarilla) y el controlador (línea azul, posicionada en la parte inferior derecha de la sección de gráficas). La parte de control de la propia HMI y de la red, está en la parte inferior de la ventana, con la que se puede modificar las ganancias del controlador del sistema, resetear sensores, apagar motor, parar y reiniciar el sistema y la comunicación, entre otros. Con estas dos secciones de la interfaz queda comprobado que se cubren los aspectos planteados en el diseño; a demás de la práctica utilidad que brinda para manejar el sistema.

5. Después de haber implementado la red, HMI y el controlador para el sistema, las pruebas fueron realizadas y se pudieron obtener varios valores donde las ganancias estabilizan la planta. La tabla 4.1 muestra que despues de 12 pruebas en las que se variaron 3 ganancias (G3Rospos: ganancia de la posición del péndulo; G4RosVel: ganancia de la velocidad del péndulo y G5error: ganancia del error integral), se pudo demostrar que sólo en dos pruebas con valores de G5error distintas (el primero con el que se empezó la prueba y cuyo valor fué tomado de la simulación y el segundo con valor randómico de 0.25, correspondiente a la prueba 12), la planta se volvía *inestable*; mientras que para el resto de ganancias modificadas la variación en tiempo de estabilización tiene una diferencia de 9 segundos entre los valor máximo y mínimo de la tabla; a lo que podemos concluir que el diseño del controlador esta dentro de los parámetros aceptables de funcionamiento (10 de 12 pruebas).

5.2. Recomendaciones

5.2.1. Recomendaciones del sistema electrónico

1. El correcto funcionamiento de la planta con Arduino Uno muestra la efectividad en cuanto a trabajo de procesamiento que tienen estos shields y control que proveen para con los sensores; sin embargo se recomienda el uso y desarrollo en Arduino Due o Mega que cuentan con más pines para interrupciones externas que podrían ser usados para conectar los canales A y B de cada uno de los encoders para mejorar el rendimiento del sistema.

2. Habría que cambiar los shields de comunicación CAN si se usan algun otro tipo de shield de control, ya que los que están montados en la red son exclusivos para los Arduinos Uno o en su defecto manufacturar un shield propio.

5.2.2. Recomendaciones del sistema de control

1. El sistema podría responder de manera más o menos estable, dependiendo de las ganancias que sean calculadas a posteriori para el mismo controlador o para controladores que sean implementados con diferentes técnicas de diseño de controladores.

2. Para posteriores trabajos se debería tomar en cuenta los tiempos de propagación dentro del sistema de comunicación CAN, aquellos que afectan al sistema de control de la planta.

3. Realizar las modificaciones del sistema de control dentro de los nodos para conseguir la estabilización del sistema en alto.

5.2.3. Recomendaciones para la HMI

1. Se podría utilizar la HMI para trabajos no tan rigurosos, aún que la proyección es utilizarlo como punto de convergencia para intercambiar controladores sin tener que desmontar totalmente la red.

2. Se podrían mostrar más datos en la pantalla de la HMI que por el momento no son necesarias.

3. No utilizar mas de dos conectores USB al momento de correr el programa en Python, ya que puede causar fallas en la HMI.

5.2.4. Otras recomendaciones

1. Si se trata de conseguir la estabilización en alto se aconseja reaalizar el desarrollo matemático de la planta así como de su simulación previo a la implementación de la misma.

2. Como parte integral del desarrollo de este trabajo se utilizó la herramienta Latex para el documento escrito; por lo que se recomienda utilizarlo en posteriores trabajos.

Bibliografía

[1] A. García Osés, *Diseño de una red CAN bus con Arduino*. Universidad Pública de Navarra, 2015, trabajo Fin de Grado en Ingeniería en Tecnologías Industriales.

[2] O. Montoya, J. Valenzuela, and D. Buitrago, "Balanceo y estabilización del péndulo invertido empleando redes neuronales artificiales y un regulador lineal óptimo con criterio cuadrático (LQR)," *Scientia et Technica*, vol. 14, mar 2013, universidad Tecnológica de Pereira, Pereira, Colombia.

[3] F. Ortega, *Modelado y control del péndulo invertido sobre carro mediante sistemas híbridos.* Universidad de Sevilla, Escuela Superior Técnica de Ingenieros, 2012.

[4] A. Hernández, M. Legaspi, and J. Peláez, *Control inteligente del péndulo invertido*. Universidad Complutense de Madrid, 2012.

[5] K. Ogata, *Ingeniería de Control Moderna 5ł ED.* Prentice-Hall, 2010.

[6] Text Analysis International Inc, "Integrated development environments for natural language processing," 2011.

[7] S. Kavitha and S. Sindhu, "Comparison of integrated development environment (ide) debugging tools: Eclipse vs netbeans," *International Research Journal of Engineering and Technology (IRJET)*, vol. 02, jul 2015, universidad Tecnolgica de Pereira, Pereira, Colombia.

[8] P. Carbonnelle, "Topide top ide index (recuperado de: `http://pypl.github.io/IDE.html`)," 2015, online; accessed 31 August 2016.

[9] S. Lochhaas and M. Moore, *Open Source Software Libraries*. B sides, Journal of the University of Iowa School of library and information science, 2010.

[10] R. Arango, A. Navarro, and J. Padilla, "Sistemas open hardware y open source aplicados a la enseńanza de la electrónica," *Revista de Investigaciones- Universidad del Quindo*, mar 2014, universidad del Quindo.

[11] J. Rodríguez, L. Arana, A. Rabasa Dolado, and O. Martiínez Bonastre, *Introdicción a la programación. Teoría y práctica.* Editorial Club Universitario, 2013.

[12] R. González, *Python para todos*. Creative Commons Reconocimien- to 2.5 Espaa., 2014, espaa.

[13] R. Mall, *Real-Time Systems Theory and Practicev*. Dorling Kinderley (India) Pvt. Ltd., licensees of Pearson Education in South Asia, 2012, department of Computer Science and Engineering Indian Institute of Technology Kharagpur India.

[14] M. Ortiz López, *Sistemas en Tiempo Real*. Universidad de Crdoba, 2014, Área de Arquitectura y Tecnología de Computadores Departamento de Arquitectura de Computadores, Electrónica y Tecnología Electrónica.

[15] H. Fernández, *OPEN HARDWARE*. Universidad Católica, 2012.

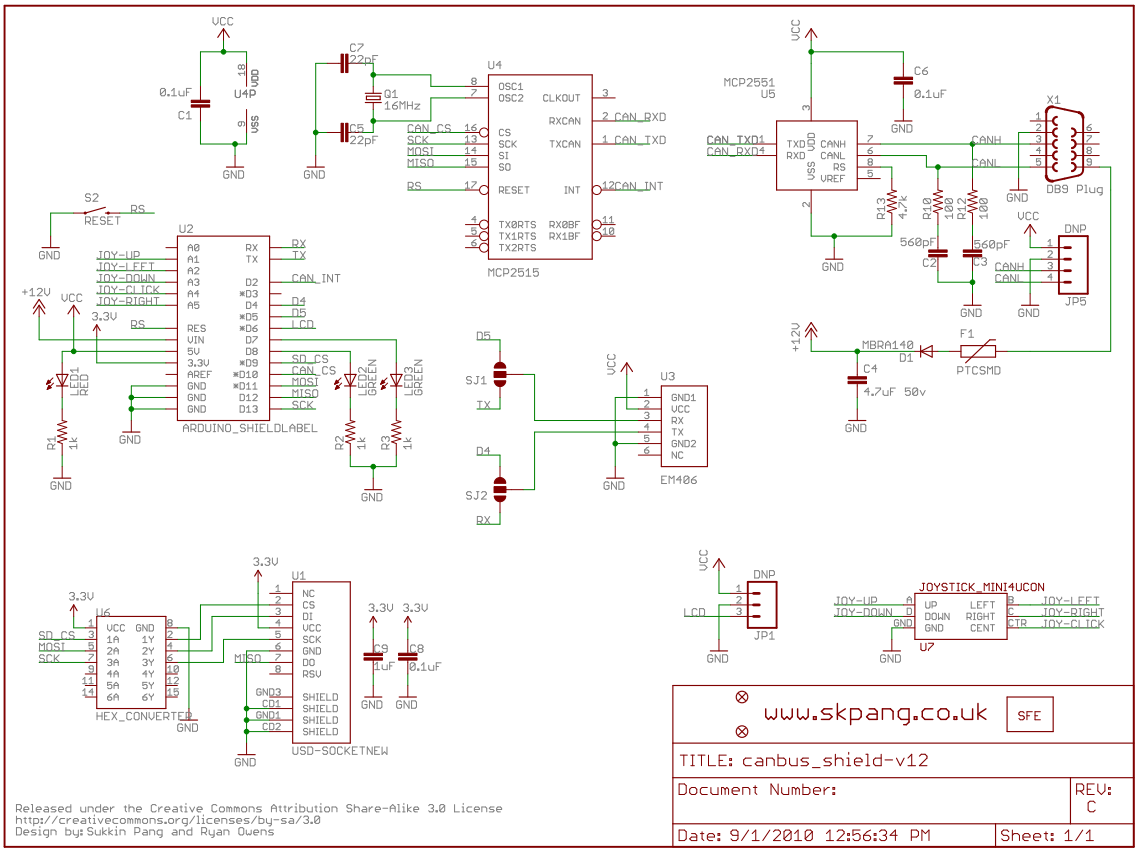
[16] L. Russell, A. Steele, and R. Goubran, "Low-cost, rapid prototyping of imu and pressure monitoring system using an open source hardware design," *IEEE Advancing Technology for Humanity*, jul 2012, dOI: 10.1109/I2MTC.2012.6229719.

[17] A. Lazalde, J. Torres, and D. Vila-Viñas, "Hardware libre (v1.2)," Buen Conocer - FLOK Society documento de política pública, jan 2015, proyecto realizado bajo convenio con el Ministerio Coordinador del Conocimiento y Talento Humano, la Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación y el Instituto de Altos Estudios Nacionales del Ecuador (IAEN).

[18] A. Montalvo, *Inverted pendulum controlled via real-time techniques: Plant development and modeling.* UTN-Ecuador, 2017, facultad de Ingeniería en Ciencias Aplicadas.

Anexos A

Can Bus Shield Sparfun Schematic



Anexos B

Programa base para la HMI en Python

Autores: Carlos Xavier Rosero, Manel Velasco García. Nombre y extensión de archivo: ***testDimitris.py***.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
@authors: carlos xavier rosero
          manel velasco garca

best performance tested with python 2.7.3, GCC 2.6.3, pyserial 2.5
"""
from __future__ import division

import serial, struct
from collections import deque
import time

import matplotlib.pyplot as plt
import matplotlib.animation as animation

class command:
    startCx = 0xf0
    stopCx = 0xf1


#use this part only with old versions of python and libraries
#tested specifically with python 2.6.5, GCC 4.4.3, serial 1.3.5
class adapt(serial.Serial):
    def write(self, data):
        super(self.__class__, self).write(str(data))

class serialCx:

    def __init__(self, usbPort, frameLen, maxLen, EOL, bauds):

        self.ay1 = deque([0.0]*maxLen)
        self.ay2 = deque([0.0]*maxLen)

        self.maxLen = maxLen

        self.eol = EOL #end of line
        self.lenEol = len(self.eol) #length of the end of line

        self.frameLen = frameLen - self.lenEol

        self.float0 = float(0)
        self.float1 = float(0)

        # open serial port
        self.ser = adapt(port=usbPort, baudrate=bauds, timeout=1, parity=serial.PARITY_NONE,
                stopbits=serial.STOPBITS_ONE)

        self.ser.open
        self.ser.isOpen

        time.sleep(2) #waits until arduino gets up

        inputS = [] #initializes buffer
        while self.ser.inWaiting() > 0: #neglects the trash code received
            inputS.append(self.ser.read(1)) #appends a new value into the buffer


        self.float0ToTx = float(9.5)  #initial values of the floating numbers to be sent
        self.float1ToTx = float(-9.5)

        value0 = struct.pack('%sf' %1, self.float0ToTx) #splits the float value into 4 strings
        value1 = struct.pack('%sf' %1, self.float1ToTx) #splits the float value into 4 strings

        #this buffer sends the command to start tx from arduino and both floating registers
        buffer = bytearray([command.startCx, ord(value0[0]), ord(value0[1]), ord(value0[2]), ord(value0[3]),
                                             ord(value1[0]), ord(value1[1]), ord(value1[2]), ord(value1[3])])

        self.ser.write(buffer) #sends the command to start the reception


  # add to buffer
    def addToBuf(self, buf, val):
        if len(buf) < self.maxLen:
```

```
            buf.append(val)
        else:
            buf.pop()
            buf.appendleft(val)

    # update plot
    def update(self, frameNum, a1, a2):
        line = bytearray()
        try:
            while True:
                c = self.ser.read(1)
                if c:
                    line.append(c)
                    if line[-self.lenEol:] == self.eol: #verifies if EOF has been received
                        line = line[0:-self.lenEol] #removes EOF
                        break
                else:
                    break

            if len(line) == self.frameLen:

                self.float0 = struct.unpack('f', line[0:4])
                self.float0 = self.float0[0] #takes out the only one element from the tuple

                self.float1 = struct.unpack('f', line[4:8])
                self.float1 = self.float1[0] #takes out the only one element from the tuple

                print self.float0,
                print self.float1

                # add data to buffer
                self.addToBuf(self.ay1, self.float0)
                self.addToBuf(self.ay2, self.float1)

                a1.set_data(range(self.maxLen), self.ay1)
                a2.set_data(range(self.maxLen), self.ay2)

            else:
                print 'Incorrect frame length:',
                print len(line)
                self.ser.flushInput() #gets empty the serial port buffer

        except KeyboardInterrupt:
            print('Exiting...')

    # clean up
    def close(self):
        # close serial

        buffer = bytearray([command.stopCx, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
                            0x01, 0x01]) #the (0x01) is used only to fill the buffer
                                         #in order to accomplish the standard length

        self.ser.write(buffer) #sends the command to start the reception

        self.ser.flushInput()
        self.ser.close()

def plotting(givenData):

    fig = plt.figure()

    ax1 = plt.subplot(211)
    ax1.grid(True)
    a1, = ax1.plot([0], [0], color=(0, 1, 0))
    plt.xlim(0, 200)
    plt.ylim(-10, 10)
    plt.ylabel('Float 0')

    ax2 = plt.subplot(212)
    ax2.grid(True)
    a2, = ax2.plot([0], [0], color=(0, 1, 0))
    plt.xlim(0, 200)
    plt.ylim(-10, 10)
    plt.ylabel('Float 1')

    anim = animation.FuncAnimation(fig, givenData.update, fargs=(a1, a2), interval=50)

    plt.show() # show plot

#####################################
######here starts the main program#####
#####################################
if __name__ == "__main__":

    testCarlos = serialCx('/dev/ttyACM9', 12, 200, 'cXrC', 115200)
    plotting(testCarlos)
    serialCx.close(testCarlos)
```

Anexos C

Programa base para el desarollo del Nodo 0 en Arduino

Autores: Carlos Xavier Rosero, Manel Velasco García, incluidas librerías propias de arduino. Nombre y extensión de archivo ***testDimitris.ino***.

```
boolean txOn = false;  // whether the frame is complete
unsigned char counterRx = 0;
unsigned char bufferRx[20] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float float0 = 0; //numbers to be sent
float float1 = 0;
unsigned char *pointer_float0 = (unsigned char *)&float0;
unsigned char *pointer_float1 = (unsigned char *)&float1;
float float2 = 0; //numbers to be received
float float3 = 0;
float *pointer_float2 = (float *)&bufferRx[1];
float *pointer_float3 = (float *)&bufferRx[5];
void setup()
{
  Serial.begin(115200);   // initialize serial:
  pinMode(13,OUTPUT);
  digitalWrite(13,LOW);
}
void loop()
{
  unsigned char i;
  unsigned char data0[4] = {0xcd, 0xcc, 0x0c, 0x40};
  unsigned char data1[4] = {0xcd, 0xcc, 0x4c, 0x40};
  unsigned char eof[4] = {'c','X','r','C'};
  if (txOn)
  {
    for (i=0; i<4; i++)
                    Serial.write(pointer_float0[i]);
    for (i=0; i<4; i++)
      Serial.write(pointer_float1[i]);
    for (i=0; i<4; i++)
      Serial.write(eof[i]);
  }
  delay(100);
  float0 = float2;
  float1 = float3;
}
void serialEvent()
{
  while (Serial.available())
  {
    bufferRx[counterRx] = (unsigned char)Serial.read(); //get the new byte
    if (counterRx >= 8)
    {
      counterRx = 0;
       digitalWrite(13, !digitalRead(13)); //toggles led each time the buffer is 9 bytes long
      switch (bufferRx[0]) //analyze command on first byte
      {
        case 0xf0:
          txOn = true;
          //here make a union of bytes to build the float registers
          float2 = (*pointer_float2);
          float3 = (*pointer_float3);
        break;
        case 0xf1:
          txOn = false;
        break;
      }
    }
    else
      counterRx++;
  }
}
```

Anexos D

Desarrollo del código de programación multinodal en el IDE de arduino

Programación nodo 0

```
// demo: CAN-BUS Shield, send data
#include "mcpointer-can.h"
#include <SPI.h>
#include "TimerOne.h"

boolean txOn = false;  // whether the frame is complete
unsigned char counterRx = 0;
unsigned char bufferRx[20] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
unsigned char len = 0;
unsigned char buf[8]= {0,0,0,0,0,0,0,0};
float float0 = 0; //numbers to be sent
float float1 = 0;
unsigned char *pointer-float0 = (unsigned char *)&float0;
unsigned char *pointer-float1 = (unsigned char *)&float1;

float float2 = 0; //values of analog read for tests
float float3 = 0;//values of analog read for tests
float float4 = 0; //numbers to be received from HMI
float float5 = 10; //numbers to be received from HMI
unsigned char *pointer-float15 = (unsigned char *)&float5; ////numbers to be received time for sampling (ts)(float)
float float6 = 0; //numbers to be received from HMI
float float7 = 0; //numbers to be received  from HMI
float *pointer-float4 = (float *)&bufferRx[1]; //value of gain
float *pointer-float5 = (float *)&bufferRx[5]; // time sampling (10-50)
float *pointer-float6 = (float *)&bufferRx[1]; // on/off
float *pointer-float7 = (float *)&bufferRx[5]; //tipe of control

//numbers to be received  from CAN
float    float10=0;
float *pointer-float10 = (float *)&buf[0]; //value of Time sampling from CAN

float    float11=0;
float *pointer-float11 = (float *)&buf[4]; //value of Time sampling from CAN

float dxSensor1[]={0,0};
float dxSensor2[]={0,0};
float  dxres1=0;
float  dxres2=0;
float  rtime;
unsigned char *pointer-dx1 = (unsigned char *)&dxres1;
unsigned char *pointer-dx2 = (unsigned char *)&dxres2;
unsigned char *pointer-rtime = (unsigned char *)&rtime;
int cont2=0;


// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 10;
const int ledHIGH    = 1;

const int LED=8;
const int LED2=7;
int cont=0;
boolean ledON=1;
unsigned char stmp[8] = {0,0,0,0,0,0,0,0};
unsigned char stmp2[8] = {0,0,0,0,0,0,0,0};
MCpointer-CAN CAN(SPI_CS_PIN);                                    // Set CS pin

void setup()
{
   Timer1.initialize(140000);          // Dispara cada 140 ms
  Timer1.attachInterrupt(ISR_timer); // Activa la interrupcion y la asocia a ISR_Blin
    Serial.begin(115200);

    while (CAN_OK != CAN.begin(CAN_1000KBPS))              // init can bus : baudrate = 500k
    {

    }
   CAN.sendMsgBuf(0x70,0, 8, stmp);

}
void ISR_timer()
```

```
  {
  unsigned char i;
  unsigned char eof[4] = {'c','X','r','C'};
  dxSensor1[1]=float2;
  dxSensor2[1]=float3;
  dxres1=(dxSensor1[1]-dxSensor1[0])/float5;
  dxres2=(dxSensor2[1]-dxSensor2[0])/float5;

  //this is is for take values of analog in uncomment analog read
  float2=float2;
  float3=float3;
  float0 = float2;
  float1 = float3;

  //this is for test if HMI is sending the two first values 4 controls

  //float0 = float4;
  //float1 = float5;
  //this is for test if HMI is sending the two first values 4 control
  //float0 = float6;
  //float1 = float7;


    //Serial.println(CAN.getCanId());
    digitalWrite(LED,1);


     //delay(float5);                          // send data 100ms per cycle


    if (txOn)
  {

    for (i=0; i<4; i++)
      //Serial.write(*pointer-float0 + i);
      Serial.write(pointer-float0[i]);


    for (i=0; i<4; i++)
      //Serial.write(*pointer-float1 + i);
      Serial.write(pointer-float1[i]);


    for (i=0; i<4; i++)
      Serial.write(pointer-dx1[i]);


    for (i=0; i<4; i++)
      Serial.write(pointer-dx2[i]);

    for (i=0; i<4; i++)
      Serial.write(pointer-rtime[i]);

    for (i=0; i<4; i++)
      Serial.write(eof[i]);


  }

    if(float6==0){
      unsigned char p[4]={0,0,0,0};
      unsigned char e[4] = {'c','X','r','C'};
    for (i=0; i<4; i++)
      //Serial.write(*pointer-float0 + i);
      Serial.write(p[i]);


    for (i=0; i<4; i++)
      //Serial.write(*pointer-float1 + i);
      Serial.write(p[i]);


    for (i=0; i<4; i++)
      Serial.write(p[i]);


    for (i=0; i<4; i++)
      Serial.write(p[i]);

    for (i=0; i<4; i++)
      Serial.write(p[i]);

    for (i=0; i<4; i++)
      Serial.write(e[i]);


    }
  dxSensor1[0]=dxSensor1[1];
  dxSensor2[0]=dxSensor2[1];


    }
void loop()
{
   interrupts();
               read();
        rtime = millis();

}

void read(){
   // unsigned char len = 0;
   // unsigned char buf[8];

    if(CAN_MSGAVAIL == CAN.checkReceive())            // check if data coming
    {
      CAN.readMsgBuf(&len, buf);    // read data,  len: data length, buf: data buf
```

```
      unsigned char canId = CAN.getCanId();

      if(CAN.getCanId()==0x20){
       float10=(*pointer-float10);
       float11=(*pointer-float11);
       // if(float5>=10){
       // CAN.sendMsgBuf(0x70,0, 8, stmp);
       // }

       float2=float10;
                u = -gain1*cartPos - gain2*cartVel - gain3*rodPos - gain4*rodVel + gain5*error;
    }
    digitalWrite(LED,1);
  }
  else{

      if(float6==1){
        CAN.sendMsgBuf(0x70,0, 8, stmp);
        CAN.sendMsgBuf(0x10,0, 8, stmp2);
        }
     }

   digitalWrite(LED,1);

}
int aux=0;
void serialEvent()
{
  digitalWrite(LED2,0);

  while (Serial.available())
  {

   digitalWrite(LED2,1);
    bufferRx[counterRx] = (unsigned char)Serial.read(); //get the new byte

    if (counterRx >= 8)
    {
      counterRx = 0;


      switch (bufferRx[0]) //analyze command on first byte
      {
        case 0xf0:
          txOn = true;
//          here make a union of bytes to build the float registers

          float4 = (*pointer-float4);
          float5 = (*pointer-float5);
   //here we send the value of ts from HMI to CAN
          stmp[0] = pointer-float15[0];
          stmp[1] = pointer-float15[1];
          stmp[2] = pointer-float15[2];
          stmp[3] = pointer-float15[3];

            aux+=1;
        break;
        case 0xf2:
          txOn = true;
//          here make a union of bytes to build the float registers
          float6 = (*pointer-float6);
          float7 = (*pointer-float7);
          aux+=1;
        break;
        case 0xf1:
          txOn = false;
          aux=0;
        break;
      }
    }
    else
      counterRx++;
    }
 }
```

Programación nodo 1

Para el envio de datos se hace un direccionamiento indirecto; a continuación toda la programación:

```
#include <SPI.h>
#include "mcpointer-can.h"
#include <Encoder.h>
#include <Timer.h>

Encoder myEnc(5, 6);
// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 10;
const int LED=8;
unsigned char len = 0;
unsigned char buf[8]= {0,0,0,0,0,0,0,0};
boolean ledON=1;
float    float0=0;
unsigned char *pointer-float0 = (unsigned char *)&float0; //
unsigned long    float1=10; //this value never be less than 10 ms
```

```
unsigned long float2=10;

float *pointer-float1 = (float *)&buf[0]; //value of Time sampling from Central Controller
int   aux=0;
float   aux2=0;
float   aux3=0;
float   aux4=0;
//int aux5=10;

Timer t;

unsigned char stmp[8] = {0, 0, 0, 0, 0, 0, 0, 0};
MCpointer-CAN CAN(SPI_CS_PIN);                                    // Set CS pin

void setup()
{
  //Timer1.initialize(10000);          // Dispara cada 10 ms
//  Timer1.attachInterrupt(ISR_timer); // Activa la interrupcion y la asocia a ISR_Blin
interrupts();
int tickEvent = t.every(float1, ISR_timer);
    Serial.begin(115200);
    pinMode(LED,OUTPUT);

    while (CAN_OK != CAN.begin(CAN_1000KBPS))              // init can bus : baudrate = 500k
    {
        Serial.println("CAN_BUS_Shield_init_fail");
        Serial.println("_Init_CAN_BUS_Shield_again");
        // delay(100);
    }
    Serial.println("CAN_BUS_Shield_init_ok!");

}
void loop()
{

  int tickEvent = t.every(float1, ISR_timer);
 // Timer1.stop();

 float0= myEnc.read();
  t.update();
  read();
 // delay(10);

}
void ISR_timer()
   {
    //float0= myEnc.read();

       Serial.println(float0);
       Serial.println(float1);
      //Serial.println(float1);
      //Serial.println(float2);
      //Serial.println(float3);


       //Timer1.restart();


  //   Serial.println(float2);

         stmp[0]= pointer-float0[0];
         stmp[1]= pointer-float0[1];
         stmp[2]=pointer-float0[2];
         stmp[3]=pointer-float0[3];
         CAN.sendMsgBuf(0x60,0, 8, stmp);
   //Serial.println(float0);
    //Serial.println(float2);

   }
void read(){
   //unsigned char len = 0;
   // unsigned char buf[8];

    if(CAN_MSGAVAIL == CAN.checkReceive())            // check if data coming
    {
      CAN.readMsgBuf(&len, buf);    // read data,  len: data length, buf: data buf
      unsigned char canId = CAN.getCanId();
      // Serial.println(CAN.getCanId());
      if(CAN.getCanId()==0x70){

        float1=(*pointer-float1); //add to float1 the value of the pointer from de CAN buffer (time sampling).


         if(float1<=10) float1=10;
         if(float1>=50) float1=50;

         if(float2!=float1){

         Serial.println("TIme_sampling_has_changed");
         }
         float2=float1;

    }
     digitalWrite(LED,1);
  }

  }

/*********************************************************************************************************
  END FILE
*********************************************************************************************************/
```

Programación nodo 2

Para el envio de datos se hace direccionamiento indirecto de datos; se crean dos variables de clase CAN para guardar el dato de entrada y luego ponerlo en el buffer de salida.

```
#include <SPI.h>
#include "mcpointer-can.h"
#include <Encoder.h>
Encoder myEnc(5, 6);
// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 10;
const int LED=8;
unsigned char len = 0;
unsigned char buf[8]= {0,0,0,0,0,0,0,0};
float    float0=0;
unsigned char *pointer-float0 = (unsigned char *)&float0;
int    aux=0;
float    aux2=0;
 float u=0;
  float v=0;
   float w=0;
   int z=0;
unsigned char stmp[8] = {0, 0, 0, 0, 0, 0, 0, 0};

float    float10=0; //keep the data from the arduino w sensor 1
float *pointer-float10 = (float *)&buf[0]; //value

MCpointer-CAN CAN(SPI_CS_PIN);                                    // Set CS pin

void setup()
{
    Serial.begin(115200);
    pinMode(LED,OUTPUT);

    while (CAN_OK != CAN.begin(CAN_1000KBPS))              // init can bus : baudrate = 500k
    {
        //Serial.println("CAN BUS Shield init fail");
     //    Serial.println(" Init CAN BUS Shield again");
      }
    //Serial.println("CAN BUS Shield init ok!");
}
void loop()
{
  float0=  myEnc.read();
  read();
}

void read(){
    if(CAN_MSGAVAIL == CAN.checkReceive())            // check if data coming
    {
      CAN.readMsgBuf(&len, buf);    // read data,  len: data length, buf: data buf
      unsigned char canId = CAN.getCanId();
       //Serial.println(CAN.getCanId());
      if(CAN.getCanId()==0x60){

        float10=(*pointer-float10);

        Serial.println(float0);

        // v=(buf[0]+(buf[1]*3.917647)/1000);
         //u=(buf[2]*3.917647)/1000;
         //z=v*100;
        //w=(1.00*z/100)+((u/100));
        //w=v+((u/1000));
        stmp[0]= buf[0];
        stmp[1]= buf[1];
        stmp[2]= buf[2];
        stmp[3]= buf[3];
        //stmp[3]= float0;
        stmp[4]= pointer-float0[0];
        stmp[5]= pointer-float0[1];
        stmp[6]= pointer-float0[2];
        stmp[7]= pointer-float0[3];
        CAN.sendMsgBuf(0x20,0, 8, stmp);

             digitalWrite(LED,1);

    }
    digitalWrite(LED,1);
  }
  }
```

Programación nodo 3

```
#include <SPI.h>
#include "mcpointer-can.h"

// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 10;
```

```
const int LED=8;
float   aux2=0;
boolean ledON=1;
unsigned char stmp[8] = {0, 0, 0, 0, 0, 0, 0, 0};
MCpointer-CAN CAN(SPI_CS_PIN);                                    // Set CS pin

void setup()
{
    Serial.begin(115200);
    pinMode(LED,OUTPUT);

    while (CAN_OK != CAN.begin(CAN_1000KBPS))                    // init can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println(" Init CAN BUS Shield again");

    }
    Serial.println("CAN BUS Shield init ok!");
}
void loop()
{
  read();
}

void read(){
  unsigned char len = 0;
    unsigned char buf[8];

    if(CAN_MSGAVAIL == CAN.checkReceive())            // check if data coming
    {
      CAN.readMsgBuf(&len, buf);    // read data,  len: data length, buf: data buf
      unsigned char canId = CAN.getCanId();
       //Serial.println(CAN.getCanId());
      if(CAN.getCanId()==0x10){

        Serial.println("-----------------------------");
        Serial.println("get data from ID: ");
        Serial.println(canId);

        for(int i = 0; i<len; i++)    // print the data
        {
            Serial.print(buf[i]);
            Serial.print("\t");

             if(!buf[0])
            {

                digitalWrite(LED,1);
                ledON=1;
            }
        }
        Serial.println();
    }
  }
  }

/*********************************************************************************************************
  END FILE
*********************************************************************************************************/
```

Anexos E

Desarrollo del código de programación para la HMI con IDE de pycharm

E.0.1. Programación de la HMI en Python

A continuación se muestra la programación realizada en Python para la HMI.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
@authors: carlos xavier rosero
          manel velasco garc'ia
          Maurico Hinojosa Rea
best performance tested with python 2.7.3, GCC 2.6.3, pyserial 2.5
"""

from __future__ import division
import serial  #serial commun library
import struct  #library to reestructure bytes on buffer
import time #time for delay library
from collections import deque #library for structure of data
import matplotlib
matplotlib.use('TkAgg')
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2TkAgg
# implement the default mpl key bindings
from matplotlib.backend_bases import key_press_handler
from matplotlib.pyplot import Figure
import matplotlib.animation as animation #library for plotting
import matplotlib.pyplot as plt #library for plotting
from matplotlib.widgets import Slider, Button, RadioButtons
from Tkinter import *
import sys
if sys.version_info[0] < 3:
    import Tkinter as Tk
else:
    import tkinter as Tk
import os #library native

class puerto:
    puerto0='/dev/ttyACM0'
    j=0
    j=1
    i=0
########################################
# ######################################################
########################################                              ##################################
########################################  VENTANA DE Puertos ##################################
########################################                              ##################################
#########################################################################################################

class AllTkinterWidgets:
    def __init__(self, master):
        frame = Tk.Frame(master, width=500, height=400, bd=1)
        frame.pack()

        iframe1 = Tk.Frame(frame, bd=2, relief=SUNKEN)

        def closeall():

            master.quit()  # stops mainloop
            master.destroy()
            puerto.j=2

        def closewindow():

            master.quit()  # stops mainloop
            master.destroy()


        def selectbutton():
            puerto.j=1
            k = v.get()
            puerto.puerto0 = k
            print k
            print puerto.puerto0
```

```
        Tk.Button(iframe1, text='CloseAll', fg="red",
                                command=closeall).pack(side=LEFT, padx=5)
        Tk.Button(iframe1, text='Done', fg="blue",
                                command=closewindow).pack(side=LEFT, padx=5)
        Tk.Button(iframe1, text='Select Port',fg="green",
                                command=selectbutton).pack(side=LEFT, padx=5)
        #Tk.Checkbutton(iframe1,
                                #text='CheckButton').pack(side=LEFT, padx=5)

        v = StringVar()

        a=Tk.Radiobutton(iframe1, text='/dev/ttyACM5', variable=v,
                    value='/dev/ttyACM5').pack(side=RIGHT, anchor=W)
        b=Tk.Radiobutton(iframe1, text='/dev/ttyACM4', variable=v,
                    value='/dev/ttyACM4').pack(side=RIGHT, anchor=W)

        c=Tk.Radiobutton(iframe1, text='/dev/ttyACM3', variable=v,
                    value='/dev/ttyACM3').pack(side=RIGHT, anchor=W)

        d=Tk.Radiobutton(iframe1, text='/dev/ttyACM2', variable=v,
                    value='/dev/ttyACM2').pack(side=RIGHT, anchor=W)

        e=Tk.Radiobutton(iframe1, text='/dev/ttyACM1', variable=v,
                    value='/dev/ttyACM1').pack(side=RIGHT, anchor=W)
        f=Tk.Radiobutton(iframe1, text='/dev/ttyACM0', variable=v,
                        value='/dev/ttyACM0').pack(side=RIGHT, anchor=W)
        iframe1.pack(expand=1, fill=X, pady=10, padx=5)


root = Tk.Tk()
# root.option_add('*font', ('verdana', 10, 'bold'))
all = AllTkinterWidgets(root)
root.title('Port Selection')
root.mainloop()

########################################
# ##########################################################
##########################################                    ##################################
########################################  VENTANA DE MONITOREO ##################################
########################################                    ##################################
#########################################################################################################

class command:
    startCx = 0xf0  #startcx value for comunication
    startCx2 = 0xf2  #startcx value for comunication 2
    stopCx = 0xf1   #stopcx value for comunication


class cons:

    ######################################
    ###### here we create global variables#####
    ######################################

    div=1
    max1=0
    runtime1=float(0)
    k=float(0)
    ts=float(50)
    onoff=float(1)
    typecontroll=float(1)




# use this part only with old versions of python and libraries
# tested specifically with python 2.6.5, GCC 4.4.3, serial 1.3.5
class adapt(serial.Serial):
    def write(self, data):
        super(self.__class__, self).write(str(data))

################################################

class serialCx:

    def __init__(self, usbPort, frameLen, maxLen, EOL, bauds):
        #SELF (VARIABLES for the _init_ class)
        self.ay1 = deque([0.0] * maxLen)
        self.ay2 = deque([0.0] * maxLen)
        self.ay3 = deque([0.0] * maxLen)
        self.ay4 = deque([0.0] * maxLen)
        self.ay5 = deque([0.0] * maxLen)
        self.maxLen = maxLen
        self.eol = EOL  # end of line
        self.lenEol = len(self.eol)  # length of the end of line
        self.frameLen = frameLen - self.lenEol
        self.float0 = float(0) #var analog1
        self.float1 = float(0) #var analog2
        self.valor1 = float(0) #var analog1 dy/dt
        self.valor2 = float(0) #var analog2 dy/dt
        self.runtime = float(0)  # var analog2 dy/dt
                            #this all var are going to the plot
        # open serial port
        self.ser = adapt(port=usbPort,
                                baudrate=bauds, timeout=1, parity=serial.PARITY_NONE,
                            stopbits=serial.STOPBITS_ONE)

        ######################################
        ######here we send initial values to the arduino0 #####
        ######################################

        #self.ser.open
        #self.ser.isOpen
        time.sleep(2)  # waits until arduino gets up
```

```
        inputS = []  # initializes buffer
        while self.ser.inWaiting() > 0:  # neglects the trash code received
            inputS.append(self.ser.read(1))  # appends a new value into the buffer

        self.float0ToTx = cons.k  # initial values of the floating
                                  #numbers to be sent--------> value of k gain
        self.float1ToTx = cons.ts # initial values of the floating
                                  #numbers to be sent--------> value of ts (tme sampling) 10-50 ms
        self.float2ToTx = cons.onoff  # initial values of the floating
                                  #numbers to be sent--------> on/of arduino's communication
        self.float3ToTx = cons.typecontroll  # initial values of the floating
                                  #numbers to be sent--------> Tipe of controller crank/inverted

        cons.div=self.float1ToTx
        value0 = struct.pack('%sf' %1, self.float0ToTx)  # splits the float value into 4 strings
        value1 = struct.pack('%sf' %1, self.float1ToTx)  # splits the float value into 4 strings
        value2 = struct.pack('%sf' %1, self.float2ToTx)  # splits the float value into 4 strings
        value3 = struct.pack('%sf' %1, self.float3ToTx)  # splits the float value into 4 strings

        time.sleep(0.2) #this wait before buffer finish write
        # this buffer sends the command to start tx from arduino and both floating registers
        buffer = bytearray([command.startCx,
                                  ord(value0[0]), ord(value0[1]), ord(value0[2]), ord(value0[3]),
        ord(value1[0]), ord(value1[1]), ord(value1[2]), ord(value1[3])])

        self.ser.write(buffer)  # sends the command to start the reception

        time.sleep(0.075) #this wait before buffer finish write

        buffer = bytearray([command.startCx2,
                                  ord(value2[0]), ord(value2[1]), ord(value2[2]), ord(value2[3]),
        ord(value3[0]), ord(value3[1]), ord(value3[2]), ord(value3[3])])
        self.ser.write(buffer)  # sends the command to start the reception
        # add to buffer
        time.sleep(0.1)  # this wait before buffer finish write


    def addToBuf(self, buf, val):
        if len(buf) < self.maxLen:
            buf.append(val)
        else:
            buf.pop()
            buf.appendleft(val)


    # update plot
    def update(self, frameNum, a1, a2, a3, a4,rtime):

        ######################################
        ######here we get the data from the buffer for update on plots #####
        ######################################

        line = bytearray()
        try:
            while True:
                c = self.ser.read(1)
                if c:
                    line.append(c)
                    if line[-self.lenEol:] == self.eol:  # verifies if EOF has been received
                        line = line[0:-self.lenEol]  # removes EOF
                        break
                else:
                    break

            if len(line) == self.frameLen:
                self.float0 = struct.unpack('f', line[0:4])
                self.float0 = self.float0[0]  # takes out the only one element from the tuple
                self.float1 = struct.unpack('f', line[4:8])
                self.float1 = self.float1[0]  # takes out the only one element from the tuple
                self.valor1 = struct.unpack('f', line[8:12])
                self.valor1=self.valor1[0]   # takes out the only one element from the tuple
                self.valor2 = struct.unpack('f', line[12:16])
                self.valor2 = self.valor2[0]  # takes out the only one element from the tuple
                self.runtime = struct.unpack('f', line[16:20])
                self.runtime = self.runtime[0]   # takes out the only one element from the tuple

                print self.float0, #coma ad a espace on the plot window
                print self.float1,
                print self.valor1,
                print self.valor2,
                print self.runtime,
                print cons.ts,
                print cons.k,
                print cons.onoff,
                print cons.typecontroll
                cons.max1 = self.valor1
                cons.runtime1=self.runtime

                # add data to buffer
                self.addToBuf(self.ay1, self.float0)
                self.addToBuf(self.ay2, self.float1)
                self.addToBuf(self.ay3, self.valor1)
                self.addToBuf(self.ay4, self.valor2)
                self.addToBuf(self.ay5, self.runtime)
                a1.set_data(range(self.maxLen), self.ay1)
                a2.set_data(range(self.maxLen), self.ay2)
                a3.set_data(range(self.maxLen), self.ay3)
                a4.set_data(range(self.maxLen), self.ay4)
                rtime.set_data(range(self.maxLen), self.ay5)
            else:
                print 'Incorrect frame length:',
                print self.float0,  # coma ad a espace on the plot window
                print self.float1,
                print self.valor1,
                print self.valor2,
```

```
                print self.runtime
                print self.frameLen
                print len(line)
                #self.ser.write(buffer)
                #self.ser.flushInput()  # gets empty the serial port buffer


            except KeyboardInterrupt:
                print('Exiting...')

        def printrtime(self):
            print self.runtime
            return self.runtime
        # clean up
        def close(self):
            # close serial

            buffer = bytearray([command.stopCx, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
                                0x01, 0x01])  # the (0x01) is used only to fill the buffer
            # in order to accomplish the standard length

            self.ser.write(buffer)  # sends the command to start the reception

            self.ser.flushInput()
            self.ser.close()



def plotting(givenData):

    ########################################
    ###### here we create the variables for canvas and plt #####
    ########################################

    root = Tk.Tk()
    root.wm_title("Embedding in TK")
    #f = plt.figure(figsize=(9, 6), dpi=95, facecolor = 'w', edgecolor = 'k')
    fig = plt.figure(num = 1, figsize = (9, 5.5),
                dpi = 95, facecolor = 'w', edgecolor = 'k') #modifie plot's properties


    ########################################
    ######this are just propierties of plt and sub plt#####
    ########################################

    plt.rc('lines', linewidth=1.2) #modifie weigh of lines
    plt.rc('font', size=10) # modifie weight of text
    plt.suptitle('Inverted Pendulum')# add subplot title
    plt.subplots_adjust(hspace=0.30)


    ########################################
    ###### here we can change properties from each subplots #####
    ########################################

    #ax1 =  f.add_subplot(111)
    ax1 = plt.subplot(321)
    ax1.grid(True)
    a1, = ax1.plot([0], [0], color=(0, 1, 1)) ##comma means continuous plotting
    plt.xlim(0, 150)
    plt.ylim(-2500, 2500)

    plt.xlim(plt.xlim()[::-1])
    plt.ylabel('Analog1')
    plt.xlabel('Time')
    plt.grid(color = 'r', linewidth = 1)
    #plt.legend(loc=2)
    plt.title('', fontsize=20, color='0.5', verticalalignment=
                'baseline', horizontalalignment='center')

    ax2 = plt.subplot(323)
    ax2.grid(True)
    a2, = ax2.plot([0], [0], color=(1, 1, 0))
    plt.xlim(0, 150)
    plt.ylim(-2500, 2500)
    plt.xlim(plt.xlim()[::-1])#invert x axe's direction if
                #this line is disable plotting change direction
    plt.ylabel('Analog2')
    plt.xlabel('Time')
    plt.grid(color = 'b', linewidth = 1)
    plt.title('_', fontsize=20, color='0.5', verticalalignment='baseline',
                horizontalalignment='center')

    ax3 = plt.subplot(322)
    ax3.grid(True)
    a3, = ax3.plot([0], [0], color=(0, 1, 0))
    plt.xlim(0, 15)
    plt.ylim(-50, 50)  # modifie limits on y axis
    plt.xlim(plt.xlim()[::-1])  # invert x axe's direction
    plt.ylabel('Derivative1')
    plt.xlabel('Time')
    plt.grid(color='m', linewidth=1)
    plt.title('_', fontsize=20, color='0.5', verticalalignment='baseline',
                horizontalalignment='center')

    ax4 = plt.subplot(324) #_first num add y subplots _second_num add
                #x subplot _3_num put in the correct place the subplot
    ax4.grid(True)#this add grid on subplot
    a4, = ax4.plot([0], [0], color=(0, 0, 1)) #
    #a3, = ax4.plot([0], [0], color=(0, 1, 0)) #unline to plot two lines in the same plot
    plt.xlim(0, 15) #modifie limits on x axis
    plt.ylim(-50, 50) #modifie limits on y axis
    plt.xlim(plt.xlim()[::1])  # invert x axe's direction
```

```
plt.ylabel('Derivative2') #this add ylabel title
plt.xlabel('Time') #this add xlabel title
plt.grid(color='y', linewidth=1) #modifie properties of the grid
plt.title('␣', fontsize=20, color='0.5',
            verticalalignment='baseline', horizontalalignment=
            'center')#modifie title's properties on subplots

ax5 = plt.subplot(325) #_first num add y subplots _second_num add
            #x subplot _3_num put in the correct place the subplot
ax5.grid(True)#this add grid on subplot
rtime, = ax5.plot([0], [0], color=(0, 0, 1)) #

plt.xlim(0, 50) #modifie limits on x axis
plt.ylim(0, 100000) #modifie limits on y axis
plt.xlim(plt.xlim()[::-1])  # invert x axe's direction
plt.ylabel('runtime') #this add ylabel title
plt.xlabel('Time') #this add xlabel title
plt.grid(color='g', linewidth=1) #modifie properties of the grid
plt.title('␣', fontsize=20, color='0.5',
            verticalalignment='baseline', horizontalalignment=
            'center')#modifie title's properties on subplots
plt.subplots_adjust(left=0.1)
plt.subplots_adjust(right=0.95)
# selection2 = str(cons.runtime1)
# label3.config(text=selection2)
#anim = animation.FuncAnimation(f,
            givenData.update, fargs=(a1, a2, a3, a4, rtime),frames=1000, interval=1)
            #update values on plot #200 frames each 1 ms

anim = animation.FuncAnimation(fig,
            givenData.update, fargs=(a1, a2, a3, a4, rtime), frames=1000,interval=1)
            # update values on plot #200 frames each 1 ms

#######################################
######bottons inside canvas #####
#######################################


#here we made the canvas for the bottons and other things
#####################################################
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.show()
canvas.get_tk_widget().pack(side=Tk.TOP, fill=Tk.BOTH, expand=1)

toolbar = NavigationToolbar2TkAgg(canvas, root)
toolbar.update()
canvas._tkcanvas.pack(side=Tk.TOP, fill=Tk.BOTH, expand=1)

def on_key_event(event):
    print('you␣pressed␣%s' % event.key)
    key_press_handler(event, canvas, toolbar)

canvas.mpl_connect('key_press_event', on_key_event)

def _quit():
    root.quit()  # stops mainloop
    root.destroy()  # this is necessary on Windows to prevent
    # Fatal Python Error: PyEval_RestoreThread: NULL tstate
def reset():
     cons.runtime1 = float(0)
     cons.k = float(0)
     cons.ts = float(50)
     cons.onoff = float(1)
     cons.typecontroll = float(1)
     print cons.runtime1,cons.typecontroll,cons.onoff, cons.ts, cons.k
     serialCx(puerto.puerto0, 24, 300, 'cXrC',
                 115200)# Fatal Python Error: PyEval_RestoreThread: NULL tstate

buttonreset = Tk.Button(master=root, text='Reset', command=reset,fg="red",width=14)
buttonreset.pack(side=Tk.BOTTOM)
button = Tk.Button(master=root, text='Quit', command=_quit, fg="red", width=14)
button.pack(side=Tk.TOP)

label = Tk.Label(master=root, text="␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣")
label.pack(side=Tk.LEFT, padx=4)

labelgain = Tk.Label(master=root, text="Gain␣:␣(x.xx)")
labelgain.pack(side=Tk.LEFT,padx=4)
e = Tk.Entry(master=root,width=10)
e.insert(0,0)
e.pack(side=Tk.LEFT,padx=20)
e.focus_set()
labelts = Tk.Label(master=root, text="Time␣sampling␣:␣(x)")
labelts.pack(side=Tk.LEFT,padx=4)
f = Tk.Entry(master=root,width=10)
f.insert(0,10)
f.pack(side=Tk.LEFT,padx=35)

#e.focus_set()

label = Tk.Label(master=root, text="␣␣␣␣␣␣␣␣")
label.pack(side=Tk.LEFT)


def selectcontrolbutton():
    cons.typecontroll = v.get()
    print v.get()
    print cons.typecontroll

buttoncontrol = Tk.Button(master=root, text='Type␣Control',
            command=selectcontrolbutton, fg="purple", width=14)
buttoncontrol.pack(side=Tk.LEFT)
rootframe=Tk.Frame(root, bd=1, relief=SUNKEN)
v = IntVar()
c = Tk.Radiobutton(rootframe, text='Control1', variable=v,
```

```
                        value=1).pack(side=TOP)
    d = Tk.Radiobutton(rootframe, text='Control2', variable=v,
                        value=2).pack(side=TOP)
    t = Tk.Radiobutton(rootframe, text='Control3', variable=v,
                        value=3).pack(side=TOP)
    r = Tk.Radiobutton(rootframe, text='Control4', variable=v,
                        value=4).pack(side=TOP)

    rootframe.pack(side=Tk.LEFT, fill=X)

    label = Tk.Label(master=root, text="__________")
    label.pack(side=Tk.LEFT)


    def changevalbutton():
        k = e.get()
        l = f.get()
        cons.k=float(k)
        cons.ts = int(l)
        print k,l,cons.ts,cons.k
        serialCx(puerto.puerto0, 24, 300, 'cXrC',
                                115200) # This values modifie serialport,
                            #Length of bytes in tx,Length of data plotting, and bauds
        print k, l, cons.ts, cons.k


    def stopcanbutton():

        if puerto.i==1:
            cons.onoff=1
            puerto.i=0
        else:
            cons.onoff = 0
            puerto.i = 1
        print cons.onoff
        serialCx(puerto.puerto0, 24, 300,
                                        'cXrC',
                115200)
                        # This values modifie serialport,
      #Length of bytes in tx,Length of data plotting, and bauds

        print cons.onoff
        print puerto.i

    b3 = Tk.Button(master=root, text="Send_Data", command=changevalbutton, fg="green")
    b3.pack(side=Tk.LEFT,pady=10,padx=10)

    label = Tk.Label(master=root, text="__________")
    label.pack(side=Tk.LEFT)

    b4 = Tk.Button(master=root, text="Stop/Start_CAN_COM", command=stopcanbutton)
    b4.pack(side=Tk.BOTTOM, pady=10)


    labelruntime = Tk.Label(master=root,text="Runtime")
    labelruntime.pack(side=Tk.LEFT,pady=10)
    label = Tk.Label(master=root, text="____")
    label.pack(side=Tk.LEFT)
    def runtimebutton():
        s = cons.runtime1
        labelruntime.config(text=s)

    b2 = Tk.Button(master=root, text="Runtime", command=runtimebutton, fg="blue")
    b2.pack(side=Tk.LEFT,pady=10)
    label = Tk.Label(master=root, text="______")
    label.pack(side=Tk.LEFT)

    #    b2 = Tk.Button(master=root, text="runtime", command=runtimebutton)
#    b2.pack(side=Tk.RIGHT)



    ########################################
    ######update data from plt from pyplot#####
    ########################################

    #here the plt or figure is updating
    anim = animation.FuncAnimation(fig, givenData.update, fargs=(a1, a2, a3, a4, rtime), frames=1000,
                                   interval=1)  # update values on plot #200 frames each 1 ms
    #plt.show()
    Tk.mainloop()


   # plt.close()
    #plt.show()

    #Tk.mainloop()

########################################
######here starts the main program#####
########################################


if __name__ == "__main__":

    testMauricio = serialCx(puerto.puerto0, 24, 300, 'cXrC', 115200) #This values modifie serialport,
                #Length of bytes in tx,Length of data plotting, and bauds
    if puerto.j==1:
        plotting(testMauricio) #plotting
    serialCx.close(testMauricio) #close all process
```