ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA MECÁNICA

UTILIZACIÓN DEL MIDDLEWARE OROCOS PARA LA IMPLEMENTACIÓN DE CONTROLADORES EN TIEMPO REAL

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE MAGÍSTER EN DISEÑO, PRODUCCIÓN Y AUTOMATIZACIÓN

FABRICIO MANUEL TIPANTOCTA PILLAJO
fabricio.tipantocta@epn.edu.ec

DIRECTOR: ING. ANA RODAS
ana.rodas@epn.edu.ec

CODIRECTOR: ING. IVÁN ZAMBRANO, M.Sc.
ivan.zambrano@epn.edu.ec

Quito, abril, 2017

DECLARACIÓN

Yo, Fabricio Manuel Tipantocta Pillajo, declaro bajo juramento que el trabajo aquí descrito 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 los derechos de propiedad intelectual correspondiente a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Fabricio Tipantocta

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por el señor Fabricio Manuel Tipantocta Pillajo bajo mi supervisión.

Ing. Ana Rodas
DIRECTOR DE PROYECTO

MSc.Ing. Iván Zambrano
CO-DIRECTOR DE PROYECTO

DEDICATORIA

A mi esposa Daniela que es el pilar de mi hogar, a mis hijos Kelly y Gregory que son la fuente de inspiración y gracias a ellos me levanto día a día a continuar, a mi madre Charito que sin el esfuerzo de ella no llegaría a ser la persona que soy, a mis hermanos Katherine y Geovanny que con su apoyo incondicional he logrado avanzar tanto. A mis amigos que me acompañan para cada día realizar un mejor trabajo.

Fabry

AGRADECIMIENTO

Un total agradecimiento a toda mi familia y amigos pues siempre me han impulsado a ser una mejor persona.

A todos los docentes de la escuela politécnica nacional que fueron partícipes en mi formación profesional, gracias a la Ingeniera Ana Rodas y el Magister Iván Zambrano, por sus aportes en este trabajo.

Al Instituto tecnológico superior Sucre, por la colaboración brindada para la realización del proyecto.

Fabry

ÍNDICE DE CONTENIDO

Declaración ..... **i**

Certificación ..... **ii**

Dedicatoria ..... **iii**

Agradecimiento ..... **iv**

Índice de contenido ..... **v**

Índice de figuras ..... **ix**

Resumen ..... **xiv**

Abstract ..... **xv**

Presentación ..... **xvi**

1.MARCO TEÓRICO **1**

1.1. ROS. (Sistema Operativo de Robot)

1.1.2. Arquitectura de ros

1.1.2.1. Nivel de sistema de archivos 2

1.1.2.2. Nivel de computación gráfica 3

1.1.2.3. Nivel comunitario 5

1.2. Orocos Open Robot Control Software 6

1.2.1. Historia del proyecto orocos 7

1.2.2. Librería de cinemática y dinámica (kdl) 8

1.2.3. Librería bayesian filtering (bfl) 9

1.2.4. Orocos toolchain 10

1.3. Sistemas de control 11

1.3.1. Variable controlada y variable manipulada

1.3.2. Tipos de sistemas de control

1.3.2.1. Sistemas de control en lazo cerrado 12

1.3.2.2 Sistemas de control en lazo abierto

1.3.3. Modos de control 13
1.3.4. Controlador PID
1.3.6. Discretización de la ley de control pid 14
1.3.6.1. Acción proporcional
1.3.6.2. Acción integradora
1.3.6.3. Acción derivadora 15
1.3.6.4. Algoritmo de control
1.4. Linux ubuntu 17
1.4.1. Instalación de programas en linux ubuntu 18
1.4.1.1. Instalación de ros-jade
1.4.1.2. Instalación de orocos toolchain 19
1.5. Proyecto a desarrollar 20
2.MIDDLEWARE OROCOS 21
2.1. Introducción
2.1.1. Middlewares orientados a componentes 22
2.2. Sistema de tiempo real
2.2.1. Determinismo 23
2.2.2. Responsividad
2.2.3. Usuarios controladores
2.2.4. Confiabilidad
2.2.5. Operación a prueba de fallas duras 24
2.3. Aplicación de orocos toolchain
2.3.1. Real-time toolkit (rtt) 25
2.3.2. Librería de componentes en orocos (ocl) 26
2.3.2.1. Partes de un componente
2.3.2.2. Estructura interna de un componente 27
2.3.2.3. Guía para crear un componente 29
2.3.2.4. Ejemplo de creación de un componente 30

2.4. Selección de dispositivos 33
2.4.1. Requerimientos de la computadora
2.4.2. Tarjeta de adquisicion de datos pci-1711u 34
2.4.2.1. Instalación de la tarjeta de adquisición pci-1711u en linux 36
2.5. Conversor ac-dc semicontrolado 46
2.6. Sistema motor generador 49
2.6.1. Transmisión por poleas y correas
2.6.1.1. Poleas
2.6.1.2. Transmisión por correa 50
3.DISEÑO DEL CONTROLADOR 51
3.1. Modelo de la planta 52
3.1.1. Modelo de la planta en tiempo discreto 54
3.2. Diseño de controlador pid 55
3.2.1. Programa de aplicación en matlab 56
3.3. Diseño del componente para la creación del controlador 58
3.3.1. Compilación
3.3.2. Conexión con el driver pci-1711 59
3.3.3. Componente del controlador pid 60
3.4. Diseño del conversor ac-dc 63
3.4.1. Circuito de fuerza 64
3.4.2. Circuito de control 66
3.4.2.2. Microcontrolador 67
3.4.2.3. Etapa de disparo 68
3.4.2.4. Circuito completo
3.5. Relación de transmisión 70
4.PRUEBAS Y RESULTADOS 71
4.1. Simulación del controlador
4.1.1. Arranque del deployer 72

4.2. Pruebas de funcionamiento de controladores 73

4.2.1. Proporcional (P) 74

4.2.2. Proporcional - integral (PI) 76

4.2.3. PROPORCIONAL-INTEGRAL-DERIVATIVO (PID).....78

5.CONCLUSIONES Y RECOMENDACIONES 81

5.1. Conclusiones

5.2. Recomendaciones 82

BIBLIOGRAFÍA.....84

ANEXOS 86

Anexo 1. Programa del componte del controlador en tiempo real

ÍNDICE DE FIGURAS

Figura 1.1. Logotipo de Sistema Operativo de Robot. 1

Figura 1.2. Nivel de sistemas de archivos. 3

Figura 1.3. Nivel de computación gráfica.

Figura 1.4. Descripción de las formas en el entorno de ROS. 5

Figura 1.5. Librerías de OROCOS.....7

Figura 1.6. Logotipo del proyecto OROCOS

Figura 1.7. Brazo robot en serie con seis articulaciones de revolución. 9

Figura 1.8. Filtro de partículas para localización de robots móviles.

Figura 1.9 Control en lazo cerrado 12

Figura 1.10. Control en lazo abierto.

Figura 1.11. Escritorio Ubuntu 14.04 LTS.....17

Figura 1.12. Terminal de Ubuntu.

Figura 2.1. Esquema de un Middleware.....21

Figura 2.2. Orocos Toolchain como Middleware. 24

Figura 2.3. Real Time Toolkit (rtt). 25

Figura 2.4. Componente Orocos basado en la biblioteca Realtime Toolkit (RTT). 26

Figura 2.5. Constitución de un componente. 27

Figura 2.6. Estructura general de un componente. 29

Figura 2.7. Creación del componente PCI1711.....31

Figura 2.8. Carpeta raíz PCI1711.

Figura 2.9. Componente creado, extensión .cpp y .hpp.

Figura 2.10. Componente creado PCI1711 con Orocos.....32

Figura 2.11. Cabecera de archivo .hpp.....33

Figura 2.12. Tarjeta de adquisicion de datos pci-1711u. 35

Figura 2.13. I/O pines de conexión.....36

Figura 2.14. Terminal de Linux.

Figura 2.15. Actualización de Linux.....37

Figura 2.16. Archivo de configuración blclist.conf.

Figura 2.17. Ingreso de parámetros en el archivo de configuración.....38

Figura 2.18. Instalación de drivers en Linux.

Figura 2.19. Copia de archivos de driver en carpeta raíz.....39

Figura 2.20. Ingreso a la carpeta de drivers.

Figura 2.21. Instalación de controlador biokernbase.....40

Figura 2.22. Compilación de driver biokernbase.

Figura 2.23. Cambio de nombre de directorios necesarios. 41

Figura 2.24. Compilación del driver de la tarjeta.

Figura 2.25. Creación del directorio BIODAQ. 42

Figura 2.26. Copia de controladores BIODAQ.

Figura 2.27. Indexado de controladores.....43

Figura 2.28. Apertura del archivo Modules.

Figura 2.29. Escritura del driver instalado. 44

Figura 2.30. Copia de biblioteca en el directorio LIBS.

Figura 2.31. Copia de archivo de encabezado al directorio. 45

Figura 2.32. Creación de carpeta para almacenamiento de archivos JAR.

Figura 2.33. Copia de archivo JAR en directorio Advantech.

Figura 2.34. Cambio de atributos. 46

Figura 2.35. Conversor de onda completa Semicontrolado 47

Figura 2.36. Primer cuadrante.

Figura 2.37. Señal de salida del conversor AC-DC. 48

Figura 2.38. Fricción por poleas. 49

Figura 2.39. Correa trapezoidal. 50

Figura 3.1. Sistema de control de velocidad. 51

Figura 3.2 Modelamiento de un motor DC 53

Figura 3.3 Aproximación rectangular 55

Figura 3.4. Respuesta a una señal escalón. 57

Figura 3.5. Controlador PID discreto.

Figura 3.6 Respuesta a una señal escalón de un controlador PID discreto 58

Figura 3.7. Pantalla de compilación. 59

Figura 3.8. Driver de tarjeta PCI1711 usada en el programa del componente daq1711-component.hpp. 60

Figura 3.9. Cambio de dirección en la instrucción**.**

Figura 3.10. Atributos de ingreso al programa del controlador 61

Figura 3.11. Configuración de inicio. 62

Figura 3.12. Programa del controlador PID. 63

Figura 3.13. Tiristor BT151. 65

Figura 3.14. Circuito de Fuerza del conversor AC-DC semicontrolado.

Figura 3.15. Circuito de sincronización con la red. 66

Figura 3.16. Onda de sincronización con la red. 67

Figura 3.17. Microcontrolador ATmega16.

Figura 3.18. Circuito de disparo para los tiristores. 68

Figura 3.19 Circuito del Conversor AC-DC 69

Figura 3.20. Sistema Motor Generador acoplado con poleas y banda trapezoidal. 70

Figura 4.1. Sistema de Control de velocidad en Tiempo Real aplicado a un controlador PID 71

Figura 4.2. Arranque del deployer. 72

Figura 4.3. Script para arranque de componentes.....73

Figura 4.4. Pantalla de visualización de parámetros del Controlador en Tiempo Real. 74

Figura 4.5. **Pruebas con Control Proporcional con Kp=1.** 75

Figura 4.6. Pruebas con Control Proporcional con Kp=2.

Figura 4.7. Pruebas con Control Proporcional con Kp=0.5. 76

Figura 4.8. Pruebas con Control Proporcional-Integral con Kp=1 y Ti=0.5.....77

Figura 4.9. Pruebas con Control Proporcional-Integral con Kp=1 y Ti=0.9.

Figura 4.10. Pruebas con Control Proporcional-Integral con Kp=1 y Ti=1.5.....78

Figura 4.11. Pruebas con Control Proporcional-Integral-Derivativo con Kp=1,Ti=0.5 y Td=0.1. 79

Figura 4.12. Pruebas con Control Proporcional-Integral-Derivativo con Kp=1,Ti=0.5 y Td=0.9.

Figura 4.13. Pruebas con Control Proporcional-Integral-Derivativo con Kp=1,Ti=0.5 y Td=1.5. 80

RESUMEN

El presente trabajo, es parte del proyecto de investigación PIMI 15-04 que trata acerca del "Control adaptativo basado en inteligencia artificial aplicado a un sistema mecatrónico fundado en un robot paralelo para la diagnosis y rehabilitación", realizado por la facultad de Ingeniería Mecánica en colaboración con la Universidad Politécnica de Valencia. El objetivo del proyecto es implementar controladores en tiempo real mediante el Middleware Orocos, programando y desarrollando controladores por componentes. Para lograrlo, se trabajó en un ambiente creado exclusivamente para software libre como es el sistema operativo Linux Ubuntu, con los programas ROS y OROCOS. El Middleware Orocos es una herramienta de software, que permitió crear controladores en tiempo real, para desarrollar algoritmos en lenguaje de programación en C++, con la ventaja que puede enlazar software y hardware. Como aplicación se lo utiliza para la creación de un algoritmo de un controlador PID para el control automático en lazo cerrado de un motor DC, para lo cual se implementa una tarjeta de adquisición de datos. Para demostrar físicamente el funcionamiento del controlador, se diseñó un sistema de control de velocidad en lazo cerrado, para lo cual se usa un motor de corriente continua como planta. Los resultados obtenidos cumplen con el objetivo del trabajo ya que la regulación de velocidad del motor DC controlado con el algoritmo PID implementado y desarrollado en Middleware Orocos muestra resultados óptimos de regulación.

Palabras clave: Middleware, Ros, Orocos, Motor DC, Control

ABSTRACT

The present work is part of the research project PIMI 15-04 that deals with the "Adaptive control based on artificial intelligence applied to a mechatronic system based on a parallel robot for diagnosis and rehabilitation", carried out by the Faculty of Mechanical Engineering in Collaboration with the Polytechnic University of Valencia. The objective of the project is to implement real-time drivers using the Orocos Middleware, programming and developing component drivers. To achieve this, we worked in an environment created exclusively for free software such as the Ubuntu Linux operating system, with the ROS and OROCOS programs. The Orocos Middleware is a software tool, which allowed to create real-time controllers, to develop algorithms in C ++ programming language, with the advantage that it can link software and hardware. As an application it is used for the creation of an algorithm of a PID controller for the automatic control in closed loop of a DC motor, for which a data acquisition card is implemented. To physically demonstrate the operation of the controller, a closed loop speed control system was designed, for which a DC motor is used as the plant. The results obtained comply with the objective of the work since the speed regulation of the DC motor controlled with the PID algorithm implemented and developed in Orocos Middleware shows optimal results of regulation.

Keywords: Middleware, Ros, Orocos, Motor DC, Control

PRESENTACIÓN

El presente trabajo se centra en el diseño e implementación de un controlador en tiempo real basado en el Middleware Orocos, que permite desarrollar el sistema de control automático de una planta compuesta por un motor DC.

El capítulo1 contiene información pertinente acerca de los programas que se usarán, como lo son ROS y OROCOS en un ambiente de LINUX, referenciando a un sistema desarrollado en software libre, también se encuentra especificado de que se compone un sistema de control automático, pues se demostrará el funcionamiento del controlador en tiempo real diseñado por software en Orocos en este tipo de sistema.

El capítulo 2 trata acerca de cómo trabaja un sistema en tiempo real en base al Middleware Orocos, su forma de configuración para la creación de componentes que enlazarán tanto software como hardware, a la vez que trata los dispositivos que se utilizará para la aplicación del controlador diseñado.

El capítulo 3 presenta el diseño del sistema de control de velocidad, que permitirá demostrar el funcionamiento del controlador desarrollado en Middleware Orocos, partiendo del modelo y el diseño del sistema físico a implementar.

El capítulo 4, presenta las pruebas y resultados del controlador diseñado, aplicado a la planta propuesta, la simulación del controlador desarrollado bajo el sistema operativo Linux y en base a Middleware Orocos, así como también las gráficas correspondientes.

En el capítulo 5 se presentan las conclusiones y recomendaciones acerca del proyecto, las cuales permitirán continuar con la investigación en este campo.

CAPÍTULO 1

1. MARCO TEÓRICO

En este capítulo se detallará el marco conceptual de las partes fundamentales del proyecto como es la introducción al Sistema Operativo para Robot (ROS), la aplicación en el proyecto orocos, el soporte de Sistemas de Control y el software LINUX.

La integración de los cuatro elementos es la base para el desarrollo de aplicaciones en tiempo real aplicados a un sistema de control como se verá posteriormente.

1.1. ROS. (Sistema Operativo de Robot)

El Sistema Operativo de Robot (ROS) es un framework para la escritura de software de robots. Es una colección de herramientas, bibliotecas y convenciones que tienen como objetivo simplificar la tarea de crear comportamientos para robots complejos y robustos a través de una amplia variedad de plataformas robóticas. Esta plataforma común permite a la gente compartir el código y las ideas más fácilmente y, quizás más importante, significa que usted no tiene que pasar años escribiendo infraestructura de software antes de que los robots empiezan a moverse. (Quigley, Gerkey, & Smart, 2016)

Figura 1.1. Logotipo de Sistema Operativo de Robot.
(Fuente: Martinez & Fernández, 2013)

1.1.2. Arquitectura de ros

Según (Martinez & Fernández, 2013), la arquitectura de ROS ha sido diseñada y dividida en tres secciones o niveles de conceptos:

- Nivel de sistema de archivos
- Nivel de computación gráfica
- Nivel comunitario

El primer nivel es el de sistema de archivos. En este nivel, se explica cómo está formado ROS internamente mediante la estructura de carpeta, y los archivos mínimos que éste necesita para trabajar.

El segundo nivel es el nivel de Computación Gráfica, donde sucede la comunicación entre procesos y sistemas. En esta sección, se encuentran todos los conceptos y sistemas que ROS tiene que establecer para manejar los procesos y para comunicarse con más de un único equipo.

El tercer nivel es el nivel comunitario, el cual permite usar las herramientas y conceptos para compartir conocimientos, algoritmos, y el código de cualquier desarrollador. Este nivel es importante porque ROS puede crecer rápidamente con un gran apoyo de la comunidad.

1.1.2.1. Nivel de sistema de archivos

De manera similar a un sistema operativo, el nivel de sistema de archivos o "Filesystem level", de un programa de ROS se divide en carpetas, y estas carpetas tienen paquetes que describen las siguientes funcionalidades:

- **Stacks**: Cuando se reúnen varios paquetes con algunas funciones, se obtiene un stack. En ROS, existe una gran cantidad de estos stack con diferentes usos, por ejemplo, el stack de navegación.
- **Stack manifests**: Proporcionan datos sobre un stack, incluyendo su información de licencia y sus dependencias en otros stack.
- **Packages:** Paquetes que forman el nivel atómico de ROS. Un paquete tiene la estructura y el contenido mínimo para crear un programa dentro de ROS. Puede tener procesos de ROS de tiempo de ejecución (nodos) y archivos de configuración.
- **Manifests**: Proporcionan información sobre un paquete, la licencia, información, dependencias, opciones del compilador entre otros. Son administradas con un archivo llamado "manifests.xml".
- **Message (msg):** Un mensaje es la información que un proceso envía a otros procesos. ROS tiene una gran cantidad de tipos estándar de mensajes. Las descripciones de los mensajes se almacenan en "my_package/msg/MyMessageType.msg".
- **Service (srv):** Las descripciones de servicios, almacenados en "my_package/srv/MyServiceType.srv" definen la petición y respuesta de las estructuras de datos para servicios de ROS.

- **Code**: El código de ingreso de instrucciones es permitido por el lenguaje de programación utilizado, sirve para el ingreso de información a ROS generalmente el usado en el ambiente de Phyton.
- **Other:** Archivos de ayuda generados por la compilación y ejecución de varios programas.

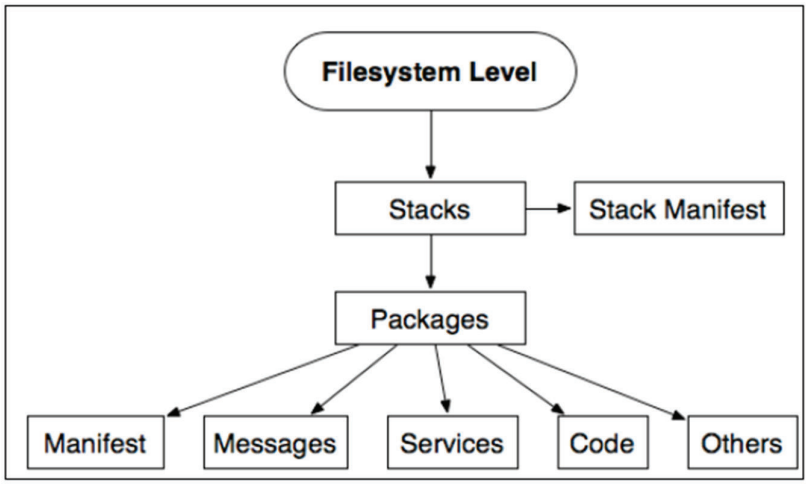

Figura 1.2. Nivel de sistemas de archivos.
(Fuente: Martinez & Fernández, 2013)

1.1.2.2. Nivel de computación gráfica

ROS crea una red en la que todos los procesos están conectados. Cualquier nodo del sistema puede acceder a esta red, interactuar con otros nodos, ver la información que está enviando y transmitir datos a la red.

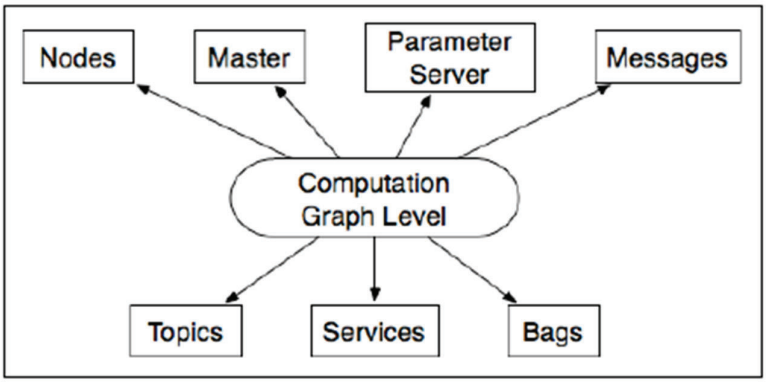

Figura 1.3. Nivel de computación gráfica.
(Fuente: Martinez & Fernández, 2013)

Los conceptos básicos de este nivel son los nodos, el maestro, el servidor de parámetros, los mensajes, los servicios, los temas y las bolsas, todos los cuales proporcionan datos al gráfico de diferentes maneras:

- **Nodes:** Los nodos son procesos donde se realiza el cálculo. Si se desea tener un proceso que pueda interactuar con otros nodos, se debe crear un nodo con este proceso para conectarlo a la red ROS. Normalmente, un sistema tendrá muchos nodos para controlar diferentes funciones. Es mejor tener muchos nodos que proporcionan una sola funcionalidad, en lugar de un nodo grande que hace todo en el sistema. Los nodos se escriben con una biblioteca cliente de ROS, por ejemplo: ros.cpp o ros.py.
- **Master:** El Maestro proporciona registro de nombres y búsqueda para el resto de los nodos. Si no se lo tiene en el sistema, no se puede comunicar con nodos, servicios, mensajes y otros.
- **Parameter service:** El Servidor de Parámetros da la posibilidad de tener datos almacenados usando las teclas en una ubicación central. Con este parámetro, es posible configurar nodos mientras se está ejecutando o cambiar el funcionamiento de los nodos.
- **Messages:** Los nodos se comunican entre sí a través de mensajes. Un mensaje contiene datos que envían información a otros nodos. ROS tiene muchos tipos de mensajes, y también puede desarrollar su propio tipo de mensaje usando mensajes estándar.
- **Topics:** Cada mensaje debe tener un nombre para ser encaminado por la red ROS. Cuando un nodo está enviando datos, se dice que el nodo está publicando un tema. Los nodos pueden recibir temas de otros nodos simplemente suscribiéndose al tema. Un nodo puede suscribirse a un tema y no es necesario que exista el nodo que está publicando este tema. Esto permite desacoplar la producción del consumo. Es importante que el nombre del tema sea único para evitar problemas y confusión entre los temas con el mismo nombre.
- **Services:** Cuando se publican temas, se está enviando datos de una manera muchos-a-muchos, pero cuando se necesita una solicitud o una respuesta de un nodo, no se puede hacerlo con temas. Los servicios dan la posibilidad de interactuar con nodos y deben tener un nombre único. Cuando un nodo tiene un servicio, todos los nodos pueden comunicarse con él, gracias a las bibliotecas cliente de ROS.
- **Bags:** Los bags son un formato para guardar y reproducir los datos del mensaje ROS. Son un mecanismo importante para almacenar datos, tales como datos de

sensores, que pueden ser difíciles de recopilar pero que son necesarios para desarrollar y probar algoritmos.

En la Figura 1.4, se puede ver la representación gráfica de este nivel, representando a un robot que trabaja en condiciones reales. La figura muestra los Nodos, los temas y qué nodo está suscrito a un tema.

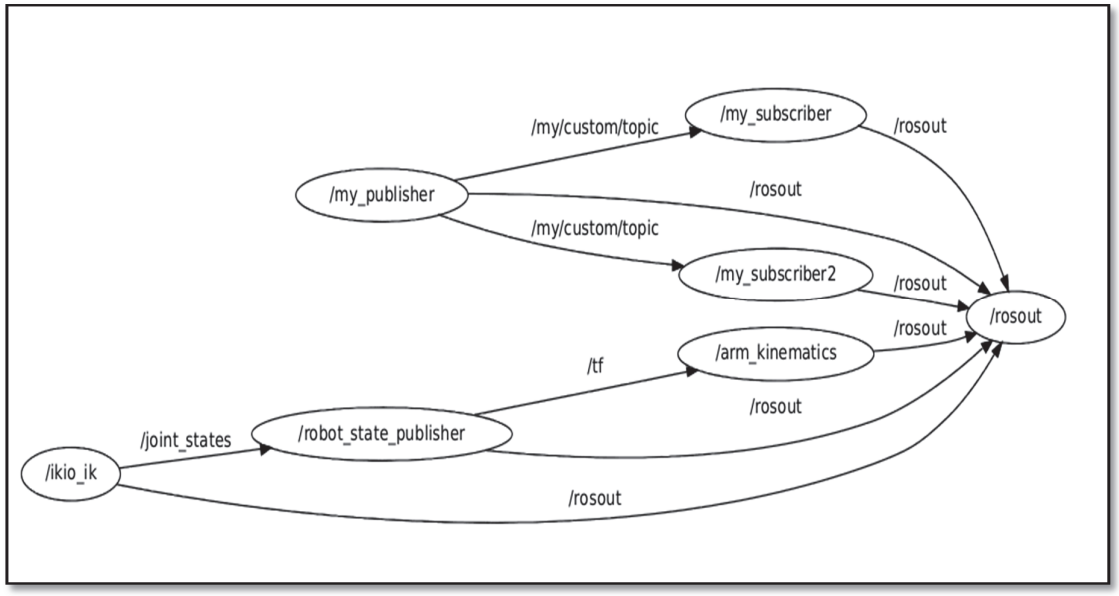

Figura 1.4. Descripción de las formas en el entorno de ROS.
(Fuente: Martinez & Fernández, 2013)

1.1.2.3. Nivel comunitario

Los conceptos de nivel comunitario de ROS son recursos ROS que permiten a comunidades separadas intercambiar software y conocimientos. Estos recursos incluyen:

- **Distribuciones**: las distribuciones ROS son colecciones de pilas versionadas que se pueden instalar. Las distribuciones ROS desempeñan un papel similar a las distribuciones de Linux. Facilitan la instalación de una colección de software y también mantienen versiones consistentes en un conjunto de software.
- **Repositorios:** ROS se basa en una red federada de repositorios de código, donde diferentes instituciones pueden desarrollar y liberar sus propios componentes de software de robot.
- **ROS Wiki:** El ROS Wiki es el principal foro para documentar información sobre ROS. Cualquier persona puede inscribirse en una cuenta y contribuir con su propia

documentación, proporcionar correcciones o actualizaciones, escribir tutoriales y más.

- **Listas de correo:** La lista de correo de usuarios de ROS es el canal de comunicación principal sobre las nuevas actualizaciones de ROS, así como un foro para hacer preguntas sobre el nuevo software ROS. (Quigley, Gerkey, & Smart, 2016)

En este proyecto se usará el sistema operativo de robots ROS para la instalación y el funcionamiento de la herramienta llamada *"Real Time Toolkit (RTT)",* la herramienta está basada en el software *Orocos Toolchain*, que permitirá crear componentes de software en tiempo real, scripts interacivos y generar códigos.

En el siguiente apartado se presenta una breve introducción al proyecto *orocos*, un entorno que permite trabajar en aplicaciones de tiempo real, el cual permitirá realizar el enlace entre el hardware que se utilizará como demostración de aplicación y el software.

1.2. Orocos Open Robot Control Software

El Orocos (Open Robot Control Software), es un framework de software en código abierto y desarrollado en C++ para la construcción de aplicaciones basadas en componentes en tiempo real en automatización y robótica. (Pal_robotic, Lauven, & FMTC, The Orocos Project, 2011)

El proyecto está estructurado en tres partes principales:

- Orocos Toolchain, para aplicaciones basadas en componentes desarrolladas en tiempo real.
- La librería de Cinemática y Dinámica (KDL) para la resolución de cadenas cinemáticas.
- La Librería de Filtrado Bayesiano (BFL) para filtrado y fusión de datos de sensores.

Las tres partes pueden utilizarse de forma independiente, pero las aplicaciones robóticas típicas utilizan todas ellas. En la Figura 1.5 se muestra las librerías que componen el proyecto Orocos.

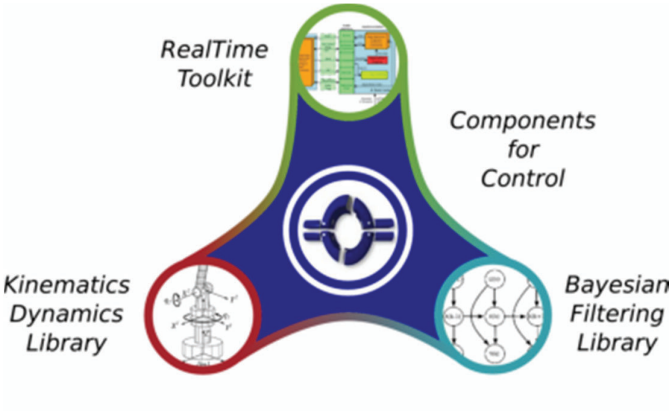

Figura 1.5. Librerías de orocos.
(Fuente: Pal_robotic, Lauven, & FMTC, 2011)

En la Figura 1.6 se puede observar el logotipo característico del proyecto orocos. El logotipo de orocos se parece a la letra 'O' que significa 'Abierto'. Está formada por dos pinzas robóticas que se alcanzan entre sí, simbolizando la colaboración. (Pal_robotic, Lauven, & FMTC, The Orocos Project, 2011)

Figura 1.6. Logotipo del proyecto orocos.
(Fuente: Pal_robotic, Lauven, & FMTC, 2011)

1.2.1. Historia del proyecto orocos

La idea de iniciar un proyecto de software libre para el control de robots nació en diciembre del 2000 como una idea de Herman Bruyninckx. La idea de control con framework “abierto” fue lanzada en la lista de correo de EURON, la Red Europea de Robótica. En la temporada de Navidad fue enviada la idea y después de dos semanas se preparó una propuesta y se envió a la Unión Europea.

La idea fue consolidada para tres socios K.U.Leuven en Bélgica (Orocos@KUL), LAAS Toulouse en Francia (Orocos@LAAS), and KTH Estocolmo en Suiza (Orocos@KTH). El

proyecto patrocinado por la UE comenzó en septiembre de 2001 y tuvo una duración de dos años.

Una primera versión del hard real-time core fue lanzada en el verano de 2002, pero fue muy preliminar y difícil de usar. En noviembre de 2002, se lanzó la primera versión con la que era posible un simple control de posición y velocidad de un brazo manipulador de seis grados de libertad.

Una vez finalizado el proyecto patrocinado por la UE, los socios del proyecto continuaron mejorando el software suministrado. Debido a su aplicabilidad a aplicaciones industriales, el proyecto Orocos en tiempo real se ha convertido en el campo del control de máquinas superando sus raíces de robótica. La modularidad de los paquetes Orocos refleja esta versatilidad.

Posteriormente se introdujeron al proyecto dos nuevos subproyectos, como complementos del núcleo "Real-Time Toolkit" (RTT): la "Bayesian Filtering Library" (BFL) y la "Cinemática y Dynamics Library "(KDL).

Hoy en día, The SourceWorks es el principal contribuyente a la infraestructura en tiempo real de la cadena de herramientas Orocos. Los usuarios y colaboradores de todo el mundo utilizan el software de Orocos para el procesamiento de datos de sensores, la cinemática de una máquina o el control de robots. Los derechos de autor sobre el código de Orocos son compartidos por más de 20 contribuyentes de diversos países del mundo. (Pal robotic, Lauven, & FMTC, 2011)

1.2.2. Librería de cinemática y dinámica (kdl)

La librería de Cinemática y Dinámica (KDL) desarrolla un framework independiente de aplicación para el modelado y cálculo de cadenas cinemáticas, tales como robots, modelos humanos biomecánicos, figuras animadas por computadora, máquinas herramientas, etc. Proporciona bibliotecas de clases para objetos geométricos, cadenas cinemáticas de varias familias, y su especificación de movimiento e interpolación. (Pal_robotic, Lauven, & FMTC, The orocos project, 2011)

Un esqueleto de un brazo robot en serie con seis articulaciones de revolución como el que se aprecia en la Figura 1.7 es un ejemplo de una estructura cinemática.

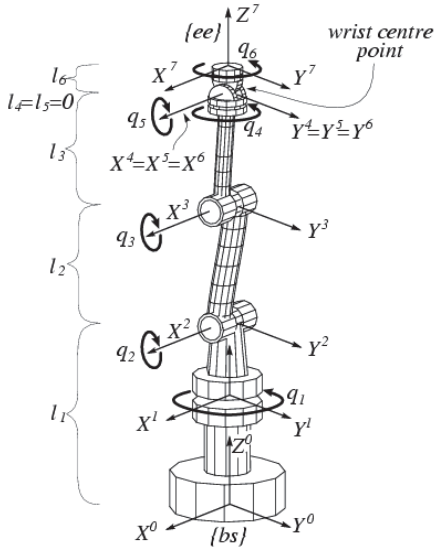

Figura 1.7. Brazo robot en serie con seis articulaciones de revolución.

(Fuente: Pal_robotic, Lauven, & FMTC, 2011)

1.2.3. Librería bayesian filtering (bfl)

La Bayesian Filtering Library (BFL) proporciona un framework independiente de aplicación para la inferencia en redes Bayesianas Dinámicas, es decir, procesamiento de información recursiva y algoritmos de estimación basados en la regla de Bayes, tales como Filtros de Kalman y Filtros de Partículas. Estos algoritmos pueden ejecutarse, por ejemplo, encima de los servicios en tiempo real, o ser utilizados para la estimación en aplicaciones de cinemática y dinámica. (Pal_robotic, Lauven, & FMTC, The orocos project, 2011)

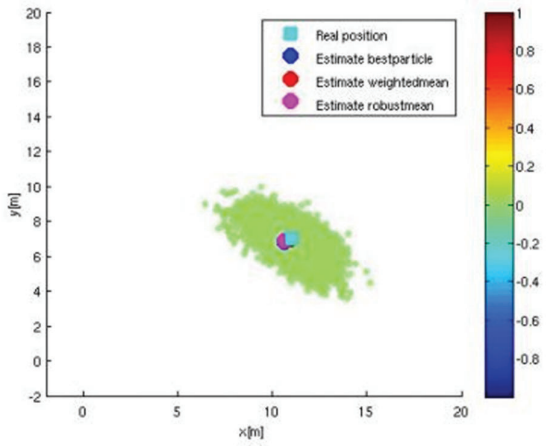

Figura 1.8. Filtro de partículas para localización de robots móviles.

(Fuente: (Pal_robotic, Lauven, & FMTC, 2011))

1.2.4. Orocos toolchain

Orocos Toolchain es la principal herramienta para crear aplicaciones de robótica en tiempo real utilizando componentes de software configurables, proporciona:

- Soporte multi-plataforma: Linux, Windows (Visual Studio) y Mac OS-X
- Extensiones a otros frameworks robóticos: ROS, Rock, Yarp
- Generadores de código para transferir datos definidos por el usuario entre componentes distribuidos.
- Componentes programables y configurables en tiempo de ejecución y en tiempo real.
- Registro e informes de eventos del sistema y datos comunicados.

Consiste en:

- **AutoProj**, una herramienta para descargar y compilar las bibliotecas necesarias.
- **The Real-Time Toolkit**, un marco de componentes que nos permite escribir componentes en tiempo real en C ++
- **La Librería de Componentes Orocos**, los componentes necesarios para iniciar una aplicación e interactuar con ella en tiempo de ejecución
- **OroGen y TypeGen,** herramientas para generar códigos listos para compilar y ejecutar desde encabezados existentes o archivos de descripción de componentes

En este proyecto se usará Orocos Toolchain para crear el componente o componentes de software necesarios para realizar una aplicación en tiempo real, que demuestre el uso y la efectividad de este tipo de sistema.

Como se ha mencionado en este apartado, Orocos Toolchain se encuentra integrado en la herramienta Real Time Toolkit, instalado sobre ROS con la cual se diseñará una aplicación que permita enlazar tanto software como hardware.

Por medio de estos componentes de software desarrollados en una computadora con bus PCI se enlazará a la tarjeta de adquisición de datos y a la planta que consiste en un motor DC con el fin de realizar el control en lazo cerrado de velocidad.

1.3. Sistemas de control

Según (Ogata, 2010), el control automático ha desempeñado un papel vital en el avance de la ingeniería y la ciencia. El control automático se ha convertido en una parte importante e integral en los sistemas de vehículos espaciales, en los sistemas robóticos, en los procesos modernos de fabricación y en cualquier operación industrial que requiera el control de temperatura, presión, humedad, flujo, etc.

1.3.1. Variable controlada y variable manipulada

La variable controlada es la cantidad o condición que se mide y controla. La señal de control o variable manipulada es la cantidad o condición que el controlador modifica para afectar el valor de la variable controlada.

En el estudio de la ingeniería de control, es necesario definir términos adicionales que se precisan para describir los sistemas de control.

- **Planta.** Una planta puede ser una parte de un equipo, tal vez un conjunto de los elementos de una máquina que funcionan juntos, y cuyo objetivo es efectuar una operación particular.
- **Proceso.** Un proceso como una operación o un desarrollo natural progresivamente continuo, marcado por una serie de cambios graduales, se llamará proceso a cualquier operación que se va a controlar.
- **Sistema.** Un sistema es una combinación de componentes que actúan juntos y realizan un objetivo determinado.
- **Perturbación.** Una perturbación es una señal que tiende a afectar negativamente el valor de la salida de un sistema.
- **Control realimentado.** El control realimentado se refiere a una operación que, en presencia de perturbaciones, tiende a reducir la diferencia entre la salida de un sistema y alguna entrada de referencia, y lo realiza tomando en cuenta esta diferencia.

1.3.2. Tipos de sistemas de control

Los sistemas de control se clasifican en dos grupos bien definidos:

- Sistemas de control en lazo cerrado
- Sistemas de control en lazo abierto

1.3.2.1. Sistemas de control en lazo cerrado

Los sistemas de control en lazo cerrado se denominan también sistemas de control realimentados. En un sistema de control en lazo cerrado, se alimenta al controlador la señal de error de actuación, que es la diferencia entre la señal de entrada y la señal de realimentación (que puede ser la propia señal de salida o una función de la señal de salida y sus derivadas y/o integrales), con el fin de reducir el error y llevar la salida del sistema a un valor deseado. El término control en lazo cerrado siempre implica el uso de una acción de control realimentado para reducir el error del sistema.

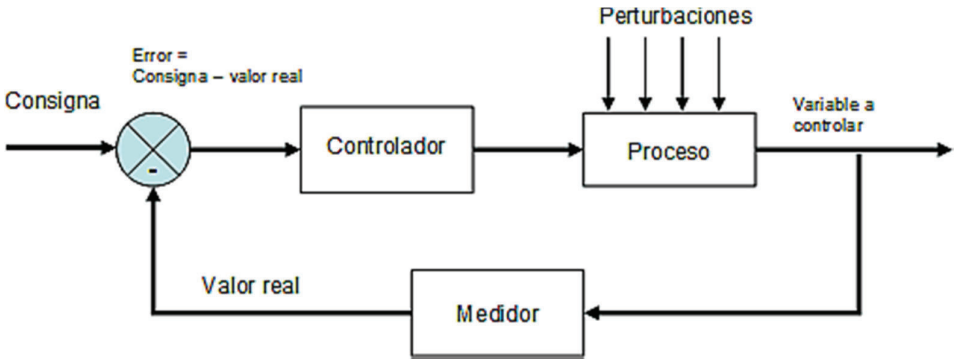

Figura 1.9 Control en lazo cerrado

(Fuente: Tipantocta, 2017)

1.3.2.2 Sistemas de control en lazo abierto

Los sistemas en los cuales la salida no tiene efecto sobre la acción de control se denominan sistemas de control en lazo abierto. En cualquier sistema de control en lazo abierto, la salida no se compara con la entrada de referencia. Así, a cada entrada de referencia le corresponde una condición de operación fija. Como resultado de ello, la precisión del sistema depende de la calibración del mismo.

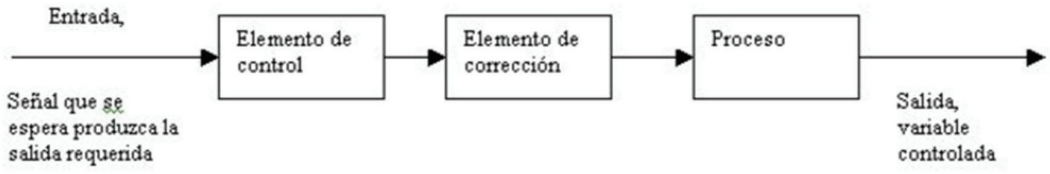

Figura 1.10. Control en lazo abierto.

(Fuente: Tipantocta, 2017)

1.3.3. Modos de control

Según (Bolton, 2014) una unidad de control puede reaccionar de diversas maneras ante una señal de error y proporcionar determinadas señales de salida para que actúen los elementos correctores:

- El modo proporcional produce una acción de control que es proporcional al error. La señal correctora aumentará en la medida en que lo haga el error. Si el error disminuye también disminuye la magnitud de la corrección y el proceso de corrección se desacelera.
- El modo derivativo produce una acción de control que es proporcional a la rapidez con la cual el error está cambiando. Cuando hay un cambio súbito en la señal de error, el controlador produce una señal correcta de gran magnitud y cuando el cambio es gradual, se produce una pequeña señal correctora.
- El modo integral produce una acción de control que es proporcional a la integral del error en el tiempo. Es decir, una señal de error constante producirá una señal de corrección que aumenta en forma constante.

Se puede combinar las acciones proporcionales y derivativas, proporcionales integrales y proporcionales, integrales y derivativas. A este último se le conoce como controlador de tres términos o PID.

1.3.4. Controlador PID

Al combinar los tres modos de control (proporcional, derivativo e integral) se obtiene un controlador que no tiene desviación en el error y disminuye la tendencia a que se produzcan oscilaciones. La ecuación que describe su comportamiento es:

$$Isal = Kpe + K_I \int_0^t edt + K_D \frac{de}{dt} + Io \qquad (1.1)$$

Donde Isal es la salida del controlador cuando existe un error e, el cual varía con el tiempo, t, Io es la salida del valor de referencia cuando no hay error, Kp es la constante de proporcionalidad, Ki la constante integral y Kd la constante derivativa. El controlador de tres modos se puede considerar como un controlador proporcional que a su vez tiene un control integral para eliminar la desviación en el error, así como un control derivativo para reducir los retrasos. Aplicado transformadas de Laplace, se obtiene:

$$(Isal - Io)(s) = KpE(s) + \frac{1}{s}K_I E(s) + sK_D E(s) \qquad (1.2)$$

y, por lo tanto:

$$\text{Función de transferencia} = Kp + \frac{1}{s}K_I + sK_D \tag{1.3}$$

$$\mathbf{PID(s)} = Kp\left[1 + \frac{1}{Tis} + Tds\right] \tag{1.4}$$

1.3.6. Discretización de la ley de control pid

Para implementar un controlador PID en una computadora digital es necesario formular aproximaciones discretas de la derivada y de la integral que aparecen en la ley de control. Si mediante algún método de diseño en el dominio continuo se ha determinado los valores de los parámetros adecuados para obtener el comportamiento deseado del sistema a lazo cerrado, se habrá definido totalmente los elementos de la expresión como la siguiente con mencionadas aproximaciones:

$$u(t) = k_p\left[r(t) - y(t) + \frac{1}{T_i}\int_0^t e(\tau)dt + T_d\left(c\frac{dr(t)}{dt} - \frac{dy(t)}{dt}\right)\right] \tag{1.5}$$

1.3.6.1. Acción proporcional

El término proporcional es

$$P(t) = k_p[r(t) - y(t)]$$

Este término se implementa reemplazando las variables continuas por sus versiones muestreadas,

$$P(t_k) = k_p[r(t_k) - y(t_k)] \tag{1.6}$$

Donde t_k denota los instantes de muestreo, es decir los valores de tiempo para los cuales se leen las variables analógicas.

1.3.6.2. Acción integradora

El término integral es

$$I(t) = \frac{k_p}{T_i}\int_0^t e(\tau)dt \tag{1.7}$$

y se deduce que

$$\frac{dI}{dt} = \frac{k_p}{T_i}e(t) \tag{1.8}$$

Si se aproxima la derivada con una diferencia hacia adelante se obtiene, empleando h =T para designar el período de muestreo

$$\frac{I(t_{k+1})-I(t_k)}{h} = \frac{k_p}{T_i} e(t_k) \tag{1.9}$$

Lo que conduce a la siguiente expresión recursiva para el cálculo del término integral

$$I(t_{k+1}) = I(t_k) + \frac{k_p h}{T_i} e(t_k) \tag{1.10}$$

1.3.6.3. Acción derivadora

El derivador puro es reemplazado con c=0 por un derivador restringido como el que se muestra,

$$D(s) = -\frac{sk_p T_d}{1+\frac{sT_d}{N}} Y(s) \tag{1.11}$$

Es decir:

$$\frac{T_d}{N}\frac{dD}{dt} + D = -k_p T_d \frac{dy}{dt} \tag{1.12}$$

Que puede ser aproximada por la siguiente ecuación en diferencias,

$$\frac{T_d}{N}\frac{D(t_k)-D(t_{k-1})}{h} + D(t_k) = -k_p T_d \frac{y(t_k)-y(t_{k-1})}{h} \tag{1.13}$$

y reordenando se obtiene

$$D(t_k) = \frac{T_d}{T_d+Nh} D(t_{k-1}) - \frac{k_p T_d N}{T_d+Nh}[y(t_k) - y(t_{k-1})] \tag{1.14}$$

Se ha determinado que el controlador PID puede ser aproximado mediante expresiones discretas. Estas expresiones constituyen quizás la más simple de las muchas discretizaciones posibles del algoritmo de control PID. (Cova, 2008)

$$u(t_k) = P(t_k) + I(t_k) + D(t_k) \tag{1.15}$$

1.3.6.4. Algoritmo de control

El resultado de discretizar un sistema continuo, es para procesar la información adquirida por medio de una tarjeta de adquisición DAQ, y por medio de software se puede desarrollar el o los controladores necesarios de aplicación en tiempo real. Gracias al apartado anterior

la suma de los controladores Proporciona, Integral y Derivativo, es como resultado el PID, y se comprende el algoritmo ya discretizado como:

```
//*********************************************************************************************************
//                                    Algoritmo de control PID
//*********************************************************************************************************


// El algoritmo se desarrolla en C++

// La primera parte adquiere el Error, el cual es la resta entre el Setpoint y la
Realimentación

                                Error=Setpoint-Realimentación;

// Se calcula Kp, y se ingresa como constante para multiplicar con el Error y se obtiene el
controlador Proporcional

                                P=Error*Kp;

// El tiempo de muestreo es de 0.01 segundos, se calcula Ti y se añade como constante
en el algoritmo para determinar el controlador Integral.

                                if(Ti==0) I=0;

                                else I = I_0 + (Kp/Ti)*error_0*T;

// El tiempo de muestreo es de 0.01 segundos, se calcula Td y se añade como constante
en el algoritmo para determinar el controlador Derivativo.

                                D=(Kp*Td)*(error-error_0)/T;

// El resultado es la suma de los tres controladores.

                                PID=P+I+D;
```

Este algoritmo se integrará posteriormente, determinando las constantes, Kp, Ti y Td, para desarrollar los controladores básicos e implementarlos mediante software, se aclara también que en C++ cuando una línea esta con doble slash “//” es para comentar y no se toma en cuenta en el algoritmo.

1.4. Linux ubuntu

El sistema operativo para el desarrollo de este proyecto es Linux Ubuntu 14.04, además de ser software libre y de código abierto es capaz de soportar los diferentes softwares usados de una manera óptima y eficiente; es un sistema operativo comercial como Windows o Mac OS. En la Figura 1.11 se muestra el escritorio de Ubuntu 14.04.

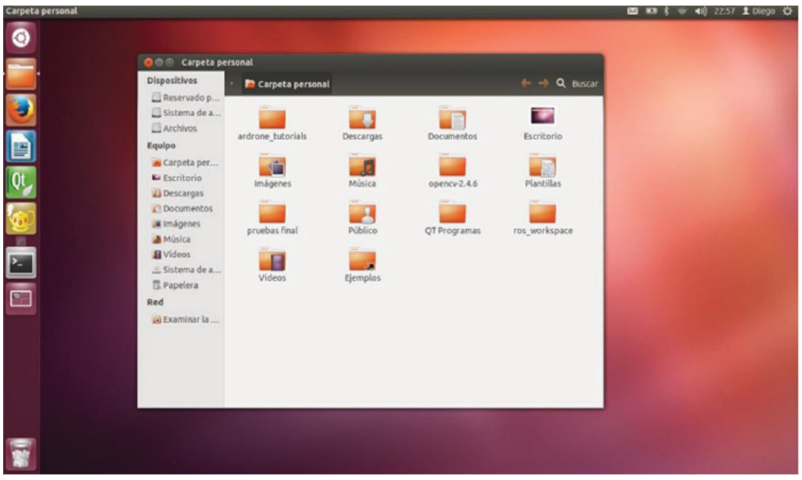

Figura 1.11. Escritorio Ubuntu 14.04 LTS.

(Fuente: Tipantocta, 2017)

El terminal es una herramienta esencial que permite al usuario ingresar códigos de comandos para interactuar entre los diferentes programas que utiliza el usuario para el desarrollo del proyecto, se puede activar con las teclas CTRL+ALT+T, en la Figura 1.12 se muestra el terminal.

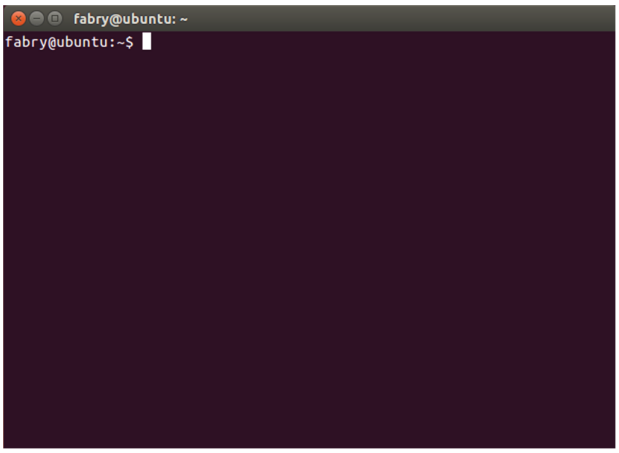

Figura 1.12. Terminal de Ubuntu.

(Fuente: Tipantocta, 2017)

1.4.1. Instalación de programas en linux ubuntu

Los programas que se usarán para el proyecto son: ROS y orocos toolchain. En los apartados siguientes se muestra como realizar la instalación de cada uno de ellos.

1.4.1.1. Instalación de ros-jade

Para la instalación se empezará con ROS ya que se ha mencionado será la plataforma base para la realización del proyecto, para ello se seguirán varios pasos, ingresando siempre los códigos uno por uno y desde una terminal.

- Añadir el repositorio de ROS Jade. (Válido para Ubuntu 14.04, 14.10 y 15.04)

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release
-
sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --
recv-
```

- Actualizar los repositorios

```
sudo apt-get update
```

- Instalar ROS Jade

```
sudo apt-get install ros-jade-desktop-full
```

- Inicializar rosdep. Esto servirá para instalar las dependencias necesarias para algunos Componentes de ROS.

```
sudo rosdep init
rosdep update
```

- Configuración del bash. Para cargar automáticamente las variables de entorno al abrir una terminal se ejecuta:

```
echo "source /opt/ros/jade/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

- Instalación de Python. Rosinstall ayudará a instalar paquetes de ROS con mayor facilidad como el lenguaje de progamación Python.

```
sudo apt-get install python-rosinstall
```

Cuando se haya instalado ROS en LINUX, se necesita configurar el espacio de trabajo, donde se va a tener todos los archivos fuente y donde la herramienta catkin, propia de ROS los compilará.

- Creación del espacio de trabajo

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

Aquí se ha llamado **catkin_ws** al directorio de trabajo, pero el nombre puede ser el que se quiera.

- Compilamos el espacio de trabajo. Cada vez que exista una modificación al o los componentes creados es necesario compilar.

```
$ cd ~/catkin_ws/
 $ catkin_make
```

- Variables de entorno. Para finalizar se debe asegurar de que se van a cargar todas las variables de entorno cuando se ejecute una terminal. En el directorio ~/catkin_ws/

```
$ source devel/setup.bash
```

1.4.1.2. Instalación de orocos toolchain

Para poder usar Orocos con ROS se debe instalar el paquete rtt_ros_integration. Este paquete contiene todo el software necesario para construir sistemas que utilizan tanto las infraestructuras Orocos como ROS.

```
$ echo $ROS_PACKAGE_PATH
/home/fabry/catkin_ws/src:/opt/ros/jade/share:/opt/ros/jade/stacks
$ echo "source /home/fabry/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

```
$ sudo apt-get install ros-jade-rtt-ros-integration
```

Ahora que ya se tiene instalado Orocos se va a proceder a crear un componente. A continuación se indica la creación del componente con el nombre my_component.

```
$ cd ~/catkin_ws/src
$ rosrun rtt_ros orocreate-pkg my_component component
```

Esto creará un componente en el espacio de trabajo, ~/catkin_ws/src.

Cuando se tenga instalado los dos programas en Linux Ubuntu, se procederá a desarrollar la aplicación en tiempo real, pero teniendo el conocimiento de que se está trabajando sobre la plataforma de ROS, pues orocos toolchain se está integrando en dicha plataforma y no trabaja independientemente. Se debe aclarar que cuando se crea un componente, este trabaja con el lenguaje de programación en C++ y no en Python, pues generalmente en ROS se trabaja de esta manera.

1.5. Proyecto a desarrollar

El objetivo general del proyecto, es usar el middleware orocos para integración de hardware con controladores en tiempo real desarrollados por software.

Para ello se programará y desarrollará controladores por componentes en base a middleware en tiempo real, se realizará pruebas a los controladores desarrollados y se los implementará físicamente en un hardware compuesto por una tarjeta de adquisición de datos que permite enlazar tanto hardware como software. Se diseñará como aplicación un sistema de control de velocidad para un motor de corriente continua.

CAPÍTULO 2

2. MIDDLEWARE OROCOS

2.1. Introducción

El Middleware o lógica de intercambio de información entre aplicaciones ("interlogical") es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware y/o sistemas operativos. Éste simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones y sincronizaciones que son necesarias en los sistemas distribuidos. De esta forma, se provee una solución que mejora la calidad de servicio, así como la seguridad, el envío de mensajes, la actualización del directorio de servicio, etc. (WIKIPEDIA, 2016)

Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El middleware abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema a resolver y de las funciones necesarias, serán útiles diferentes tipos de servicios de middleware. (WIKIPEDIA, 2016) En la Figura 2.1 se puede apreciar cómo está configurado el middleware, con accesos a las aplicaciones y directamente al hardware.

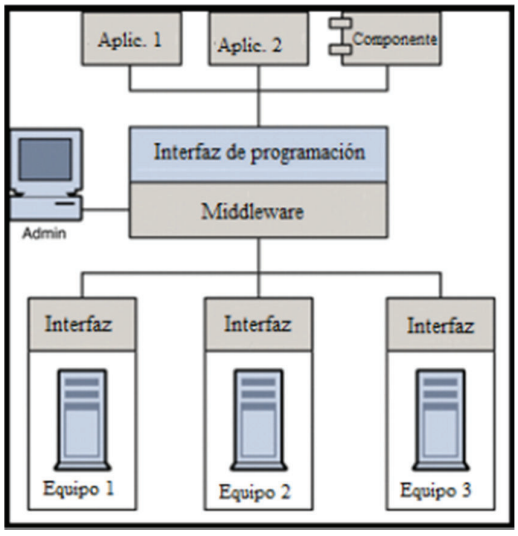

Figura 2.1. Esquema de un Middleware.
(Fuente: Martinez & Fernández, 2013)

Dependiendo de la categoría de integración, existen middlewares orientados a distintos aspectos:

- Orientados a procedimiento o procesos
- Orientados a objetos
- Orientados a mensajes
- Orientados a componentes

Para el proyecto en desarrollo todo lo que se basa en el middleware orocos, está orientado al uso de componentes y a continuación se observará cómo se configura cada uno de los módulos necesarios para crear una aplicación diseñada de esta manera.

2.1.1. Middlewares orientados a componentes

Un componente es un programa que realiza una función específica, diseñada para operar e interactuar fácilmente con otros componentes y aplicaciones. El middleware en este caso en una configuración de componentes. Los puntos fuertes de este middleware es que es configurable y reconfigurable. La reconfiguración se puede realizar en tiempo de ejecución, lo que ofrece una gran flexibilidad para satisfacer las necesidades de un gran número de aplicaciones. (WIKIPEDIA, 2016)

2.2. Sistema de tiempo real

El sistema de tiempo real es un sistema informático que interactúa con su entorno físico y responde a los estímulos del entorno dentro de un plazo de tiempo determinado. No basta con que las acciones del sistema sean correctas, sino que, tienen que ejecutarse dentro de un intervalo de tiempo determinado.

"Un ejemplo que ilustra el str[1] es el de un robot que necesita tomar una pieza de una banda sin fin. Si el robot llega tarde, la pieza ya no estará donde debía recogerla, por tanto, el trabajo se llevó a cabo incorrectamente, aunque el robot haya llegado al lugar adecuado. Si el robot llega antes de que la pieza llegue, la pieza aún no estará ahí y el robot puede bloquear su paso" (Villarroel, 2015)

Existen características en los sistemas de tiempo real como son:

- Determinismo

[1] STR: Sistemas en Tiempo Real

- Responsividad
- Usuarios controladores
- Confiabilidad
- Operación a prueba de fallas duras (*fail hard operation*)

2.2.1. Determinismo

Es la capacidad de determinar con una alta probabilidad, cuanto es el tiempo que se toma una tarea en iniciarse. Esto es importante porque los sistemas de tiempo real necesitan que ciertas tareas se ejecuten antes de que otras puedan iniciar, así que es importante determinar el tiempo que tardara el sistema en aceptar esta petición de servicio.

2.2.2. Responsividad

La responsividad se enfoca en el tiempo que tarda una tarea en ejecutarse una vez que la interrupción ha sido atendida. Los aspectos a los que se enfoca son:

- La cantidad de tiempo que se lleva el iniciar la ejecución de una interrupción
- La cantidad de tiempo que se necesita para realizar la tarea que pidió la interrupción.
- Los efectos de interrupciones anidadas.

Una vez que el resultado del cálculo de determinismo y responsividad es obtenido, se convierte en una característica del sistema y un requerimiento para las aplicaciones que correrán en él, (por ejemplo, si diseñamos una aplicación en un sistema en el cual el 95 % de las tareas deben terminar en cierto período entonces es recomendable asegurarse que las tareas ejecutadas de nuestra aplicación no caigan en el 5 % de bajo desempeño).

2.2.3. Usuarios controladores

En estos sistemas, el usuario (por ejemplo, los procesos que corren en el sistema) tienen un control mucho más amplio del sistema.

- El proceso es capaz de especificar su prioridad
- El proceso es capaz de especificar el manejo de memoria que requiere
- El proceso especifica qué derechos tiene sobre el sistema.

2.2.4. Confiabilidad

El sistema no debe solamente estar libre de fallas pero más aún, la calidad del servicio que presta no debe degradarse más allá de un límite determinado. El sistema debe de seguir

en funcionamiento a pesar de catástrofes, o fallas mecánicas. Usualmente una degradación en el servicio en un sistema de tiempo real lleva consecuencias catastróficas.

2.2.5. Operación a prueba de fallas duras

El sistema debe de fallar de manera que cuando ocurra una falla, el sistema preserve la mayor parte de los datos y capacidades del sistema en la mayor medida posible. Que el sistema sea estable, es decir, que si para el sistema es imposible cumplir con todas las tareas sin exceder sus restricciones de tiempo, entonces el sistema cumplirá con las tareas más críticas y de más alta prioridad.

2.3. Aplicación de orocos toolchain

Para crear las aplicaciones en tiempo real se mencionó en el capítulo anterior que se usará la herramienta orocos toolchain, esta herramienta funcionará sobre la plataforma de ROS, siendo este entorno el que sustente toda la creación de los componentes que se desea realizar. Pero desde ya se tiene que mencionar que tanto la plataforma de ROS y orocos toolchain funcionarán en el sistema operativo de Linux.

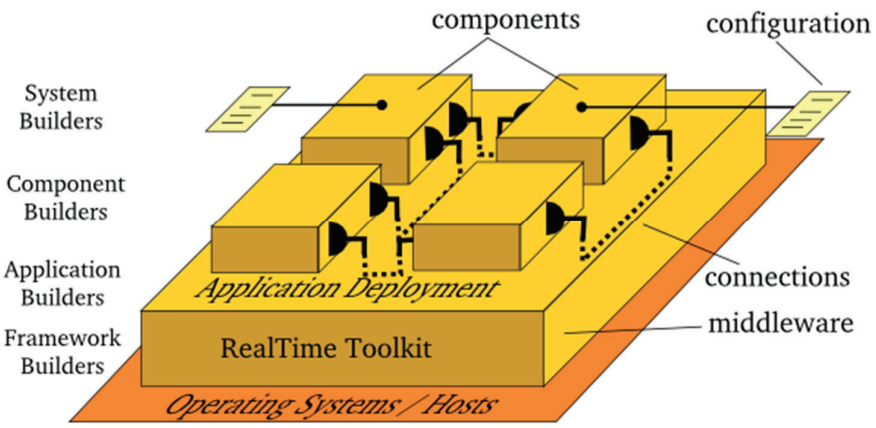

Figura 2.2. Orocos Toolchain como Middleware.
(Fuente: The orocos project, 2011)

Cabe destacar que la herramienta orocos toolchain, realizará el enlace de aplicación con el hardware como se ha mencionado el funcionamiento de middleware, que en este caso se convertirá en el middleware orocos.

Dada esta premisa se pretende analizar la creación de componentes en tiempo real con dos puntos específicos de herramientas de orocos toolchain que son:

- Real-Time Toolkit (rtt)
- Orocos Component Library (OCL)

2.3.1. Real-time toolkit (rtt)

El Orocos Real-Time Toolkit (RTT) proporciona un framework en C ++, o "runtime", dirigido a la implementación de sistemas de control. Será el encargado de proporcionar la aplicación para poder escribir los algoritmos necesarios para la ejecución en tiempo real de cualquier elemento implementado en dicha plataforma, se puede observar en la Figura 2.3. (Pal_robotic, Lauven, & FMTC, 2011)

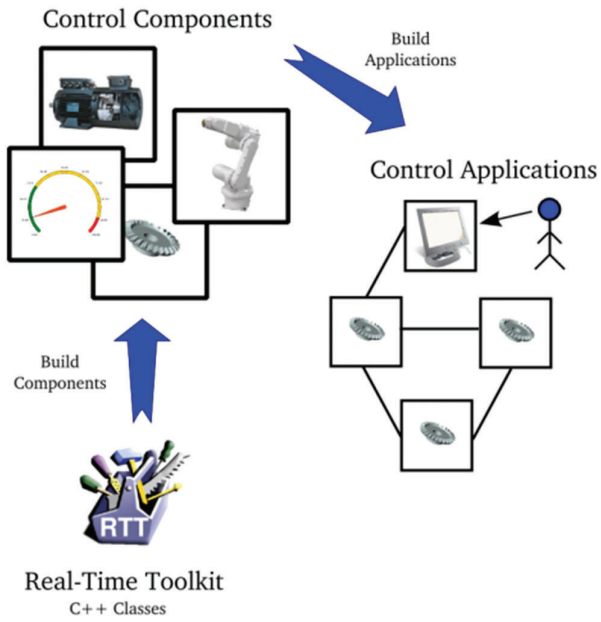

Figura 2.3. Real Time Toolkit (rtt).
(Fuente: Martinez & Fernández, 2013)

(Pal_robotic, Lauven, & FMTC, 2011) Entre otras cosas, la Real Time Toolkit permite a los desarrolladores la creación de componentes totalmente configurables (incluso en tiempo de ejecución) e interactivos basados en el control de aplicaciones de tiempo real, pudiéndose usar en aplicaciones con características tales como:

- Capturar y graficar el flujo de datos entre los componentes.
- Reasignar prioridades y modicar el algoritmo de control en tiempo de ejecución.
- Configurar los componentes y la aplicación a partir de archivos XML.

- Interactuar con otros dispositivos directamente desde una interfaz gráfica de usuario o mediante el prompt de Linux.
- Ampliar las aplicaciones con estructuras de datos propias.
- Ejecutar la aplicación tanto en sistemas operativos estándar como sistemas de tiempo real.

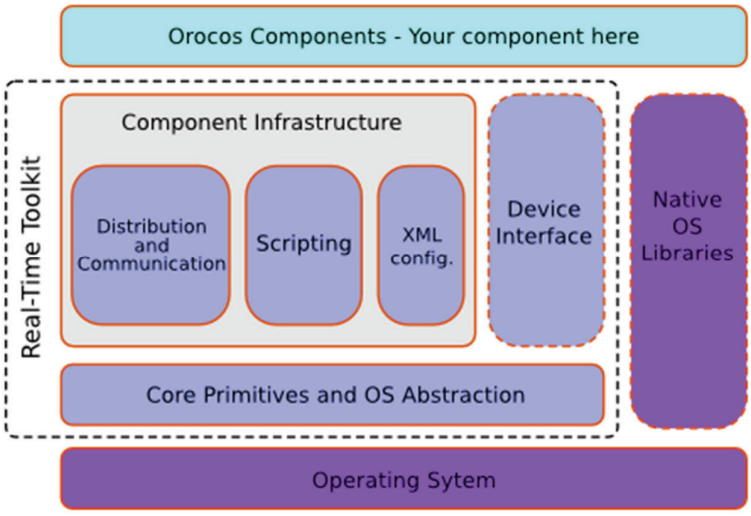

Figura 2.4. Componente Orocos basado en la biblioteca Realtime Toolkit (RTT).
(Fuente: Pal_robotic, Lauven, & FMTC, 2011)

2.3.2. Librería de componentes en orocos (ocl)

La librería de componentes orocos utiliza el Real-Time Toolkit (RTT) para construir todos sus componentes (Programas Base). Algunos componentes utilizan la librería de Cinemática y Dinámica (KDL), la Biblioteca de Filtrado Bayesiano (BFL) u otras bibliotecas. La OCL v2.0 es ahora una parte integral de la Herramienta orocos y sólo contiene componentes para configurar y supervisar aplicaciones. (THE OROCOS PROJECT, 2011)

2.3.2.1. Partes de un componente

La funcionalidad del middleware orocos es que el proyecto está implementado en C++, por lo que el diseño de todos los controladores serán realizados en dicha plataforma, esto será la creación de componentes que funcionen en tiempo real, cada componente debe tener una determinada estructura para que al compilarlo se obtenga un módulo que se pueda ejecutar de forma independiente

En la Figura 2.5 se muestra un componente que tiene diferentes partes, cada una de ellas se detalla a continuación: (The orocos project, 2011)

- Puertos de entrada. Mediante los puertos de entrada, se obtienen los datos de otros componentes o de sensores externos.
- Puertos de salida. Con los puertos de salida se envían los datos a otros componentes, o a actuadores externos.
- Atributos. Variables propias con un determinado valor.
- Métodos. Secuencia de código que se ejecuta cuando se llama a esa función.
- Eventos. Mediante los eventos, se consigue que se ejecute una secuencia de código cuando se produce un evento, tal como un nuevo dato por un puerto de entrada.

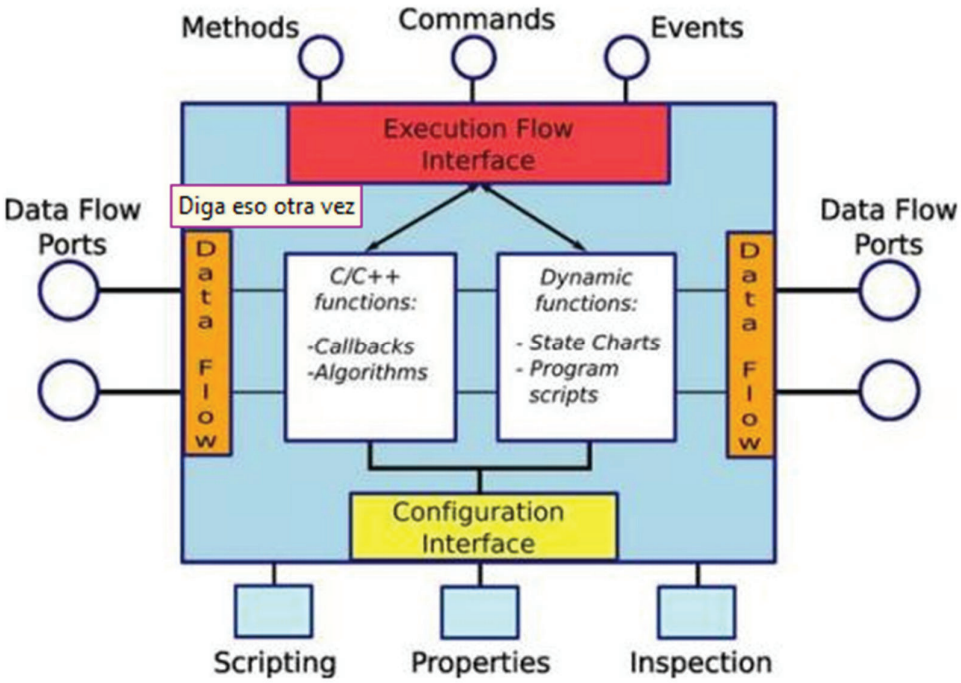

Figura 2.5. Constitución de un componente.

(Fuente: Pal_robotic, Lauven, & FMTC, 2011)

2.3.2.2. Estructura interna de un componente

Para crear un componente se necesita algunos puntos específicos que se detallan a continuación:

- **Zona de Declaración de variables globales**. En Orocos es muy importante tener en cuenta todas las variables que se van declarando. Es altamente recomendable

cuando se quiere que alguna variable vaya cambiando de valor en cada una de las iteraciones, que la variable se declare como variable global. (Casalilla, 2012)

- **StartHook().** El código que se añada en esta función se ejecutará inmediatamente antes de poner en modo running el componente. Generalmente en esta parte se suele establecer el periodo de ejecución y prioridad del módulo, en caso de no habérselo especificado anteriormente de forma interactiva. (Casalilla, 2012)

- **ConfigureHook().** En esta parte de la implementación del componente definen las posibles variables globales que se hayan declarado previamente (tales como ganancias, por ejemplo), inicializándose también todos los dispositivos de entrada/salida, como la tarjeta de los convertidores D/A, la tarjeta de encóders o sensor de fuerza. (Casalilla, 2012)

- **UpdateHook().** Al contrario que las funciones anteriores, que sólo se ejecutan cuando se les llama (y sólo una vez), el código perteneciente a la función UpdateHook() se ejecuta de forma iterativa (según el periodo de muestreo especificado) cuando el componente pasa al estado running. Se debe tener especial precaución a la hora de definir variables en esta sección ya que, al ejecutarse todo el código de forma periódica, se puede producir una declaración de alguna variable, dando un fallo en tiempo de ejecución. (Casalilla, 2012)

- **StopHook().** De forma análoga a la función StartHook(), el código que se añada en esta función se ejecutará inmediatamente antes de poner en stop el componente. De forma general, en esta zona de código se suele especificar un periodo de 0 (que indica que el componente se para), además de detener el hardware inicializado previamente. (Casalilla, 2012)

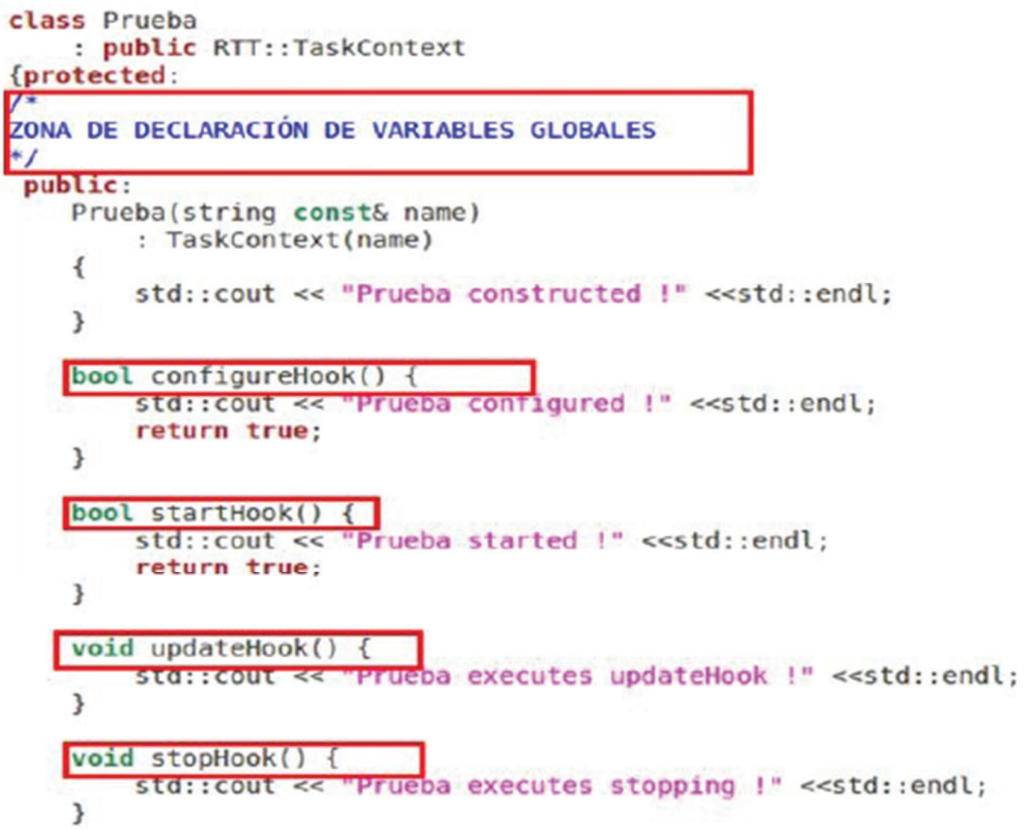

Figura 2.6. Estructura general de un componente.

(Fuente: Casalilla, 2012)

2.3.2.3. Guía para crear un componente

Para poder crear un componente, se deben seguir ciertos pasos: Se necesita abrir un terminal y digitar las siguientes instrucciones:

- Ir al espacio de trabajo y crear un componente con un nombre de identificación, en el ejemplo se muestra el nombre my_component.

```
$ cd ~/catkin_ws/src
$ rosrun rtt_ros orocreate-pkg my_component component
```

- Sustituir manifest.xml por package.xml, para que funcionen todos nuestros componentes se ha aconsejado que reemplazamos este documento con uno

elaborado por la politécnica de Valencia, este documento añade la librería OCL y RTT_ROS para trabajo en tiempo real.

- Comprobar si todo ha salido bien, se ha mencionado que la compilación es la verificación de que si el programa que está construyendo se encuentra de una manera correcta. Con las siguientes instrucciones se puede comprobar que los componentes estén al 100% compilados, de igual manera si tienen algún error el compilador mostrará las direcciones de donde posiblemente se encuentre dicho error.

```
$ cd ~/catkin_ws/
$ catkin_make
```

- Escribir el script que conecte el o los componentes. Este paso corresponde a la creación de un documento conocido como script, tiene una extensión OPS y este elemento sirve para poder arrancar el programa construido en base a componentes.
- Ejecutar el deployer. Para arrancar cualquier componente se debe abrir otro terminal el cual con la instrucción DEPLOYER permitirá arrancar el script con extensión OPS y así ejecutará todos los componentes creados.

```
rosrun rtt_ros deployer /directory/to/my/script/my_script.ops
```

- Dentro del DEPLOYER, ejecutar el start del componente principal para que empiece la ejecución.
  componente.start

2.3.2.4. Ejemplo de creación de un componente

Para crear el componente que servirá para el desarrollo del controlador, es preferible abrir otro terminal e ingresar al espacio de trabajo pero esta vez se deberá ingresar también a la carpeta src/ y se debe ejecutar la instrucción **rosrun rt_ros orocreate-pkg,** esta instrucción permite crear las aplicaciones en c++, creando el nombre del componente y sobre todo la aplicación en tiempo real.

$ cd /home/fabry/catkin_ws/src/

$ rosrun rtt_ros orocreate-pkg PCI1711 component

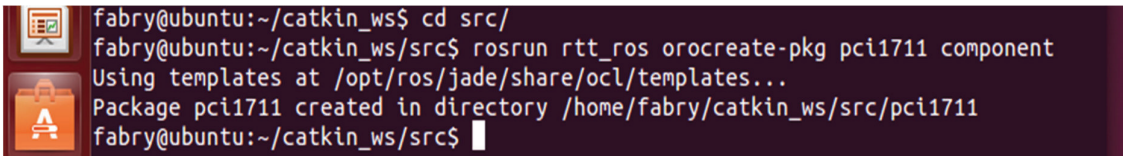

Figura 2.7. Creación del componente PCI1711.

(Fuente: Tipantocta, 2017)

En el ejemplo anterior se creó el componente PCI1711, creando un directorio completo de recursos como son:

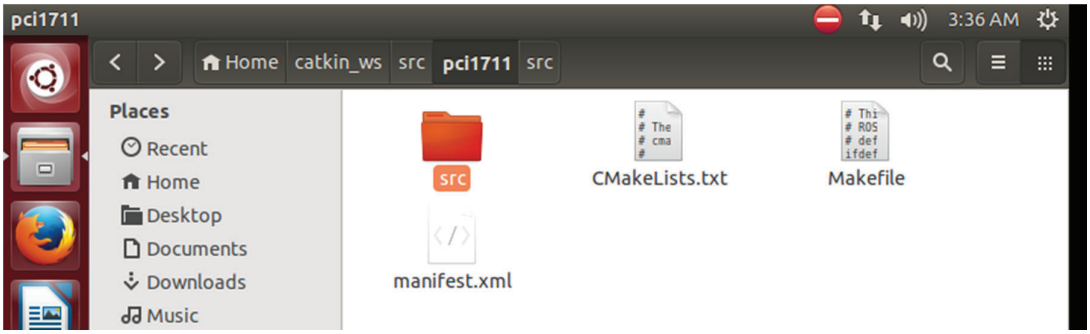

Figura 2.8. Carpeta raíz PCI1711.

(Fuente:(Tipantocta, 2017))

Donde la carpeta src/ almacena documentos como: “pci1711-component.cpp” y “”pci1711-component.hpp”, que son los elementos indispensables para trabajar con el componente en lenguaje C++.

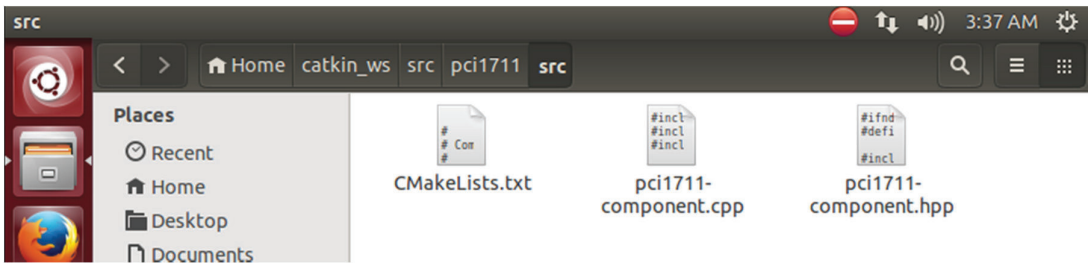

Figura 2.9. Componente creado, extensión .cpp y .hpp.

(Fuente: Tipantocta, 2017)

Cuando se abre el contenido con extensión CPP corresponderá al documento en C++ y es como se estructura el programa para el trabajo en tiempo real. En la Figura 2.10 se muestra

cómo está inicialmente estructurado el programa al ser generado desde el terminal con la instrucción siguiente:

$ rosrun rtt_ros orocreate-pkg PCI1711 component

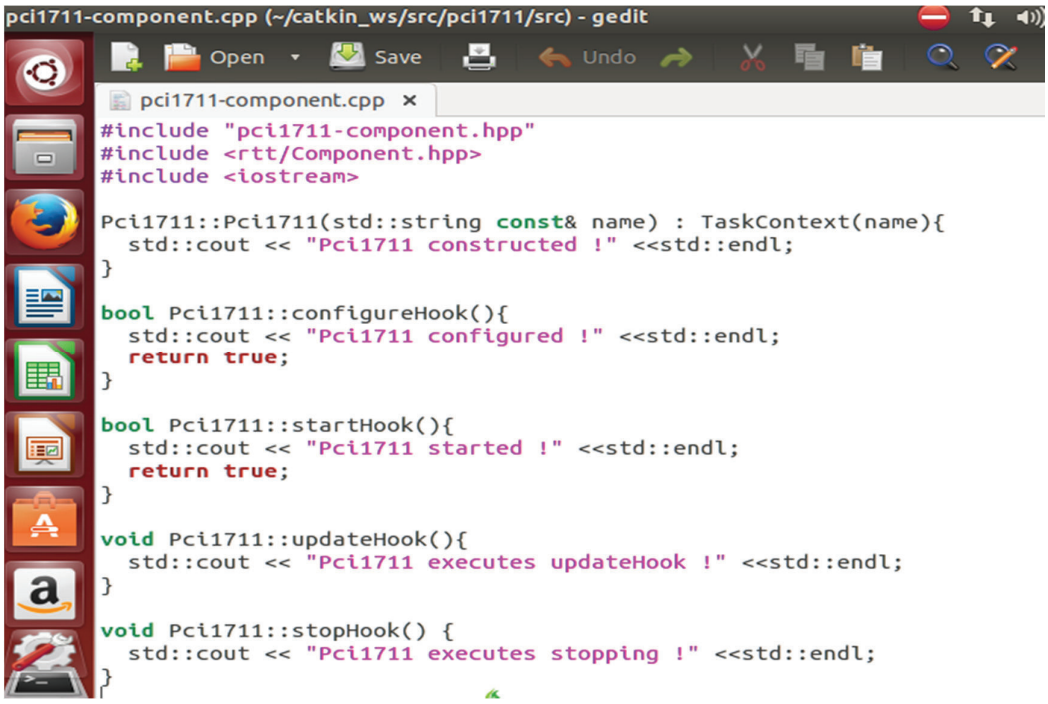

Figura 2.10. Componente creado PCI1711 con Orocos.

(Fuente: Tipantocta, 2017)

Cuando se abre el contenido de la extensión hpp corresponde a la cabecera de inicio del programa en C++ donde generalmente extrae la información necesaria para que el programa funcione correctamente en la figura 2.11 muestra cómo está estructurado el programa.

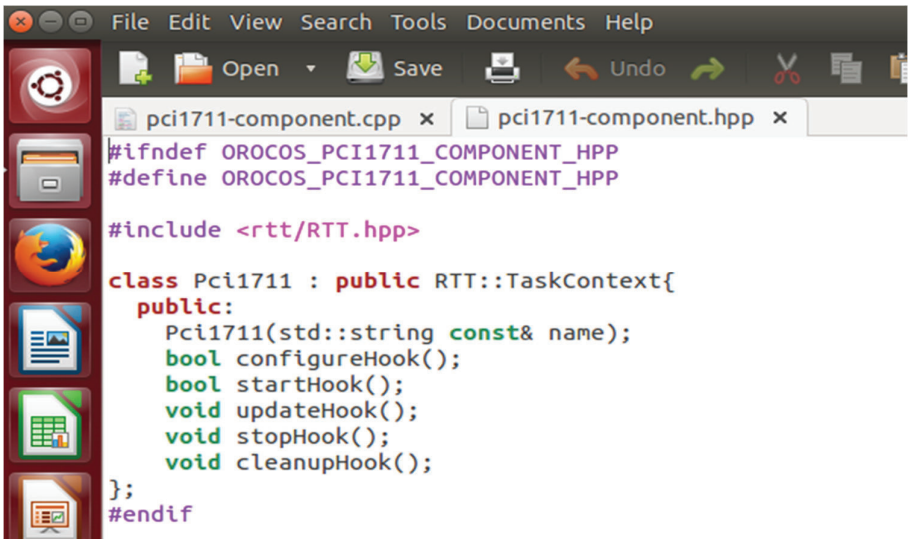

Figura 2.11. Cabecera de archivo .hpp.

(Fuente: Tipantocta, 2017)

Con este ejemplo se muestra los recursos necesarios como la terminal para ingresar las líneas de comandos necesarios y así crear el componente.

Para el presente trabajo, se ha creado un componente que maneja la lectura y escritura en registros de una tarjeta de adquisición de datos, siendo éste el medio para poder realizar el controlador el cual servirá para ejecutar la demostración de un sistema de control para un motor de corriente continua, en el capítulo tres se empezará a diseñar todo el sistema.

2.4. Selección de dispositivos

Como el objetivo principal del proyecto, es demostrar el uso de un controlador en tiempo real diseñado por software usando Orocos, es necesario integrar el software con el hardware conectadolo a una computadora.

2.4.1. Requerimientos de la computadora

Para la demostración del proyecto se ha conseguido una computadora Pentium cuatro de las siguientes características:

- Procesador pentium 4 de 2.8ghz
- Discos duro de 80 gb
- Memoria de 1 gb en ram
- Cdwriter

La razón de por qué escoger esta computadora, fue por la Main Board del procesador de la Pentium cuatro, esta tarjeta tiene dos slot de Bus PCI. Por consejo de un investigador de la politécnica de Valencia quien ha mencionado que es necesario trabajar con tarjetas de adquisición de datos que permitan la conectividad del bus PCI en los proyectos que trabaja la universidad conjuntamente con la EPN.

Otra característica, es que las computadoras nuevas ya no vienen con el bus PCI, si no, con el bus PCI Express, y las tarjetas de adquisición de datos actuales no tienen este tipo de bus de trabajo.

Definida la tarjeta de adquisición de datos que se usará para el proyecto, se instalará el sistema operativo Linux Ubuntu versión 14.04, y después las herramientas de ROS y orocos, pero cabe recalcar que se deberá instalar también el driver de la tarjeta de adquisición de datos.

2.4.2. Tarjeta de adquisicion de datos pci-1711u

Las características necesarias para la realización del proyecto, es que la tarjeta de adquisición pueda convertir señales analógicas a digitales y viceversa, para lo cual se ha buscado en internet y específicamente en la página de la marca ADVANTECH, y se ha podido observar varias tarjetas de adquisición pero con los recursos mencionados por separado.

La tarjeta de adquisición PCI-1711U es una tarjeta multifunción de bus PCI universal de la marca ADVANTECH. Ofrece las cinco funciones de medición y control más necesitadas:

- Conversión A / D de 12 bits
- Conversión D / A
- Entrada digital
- Salida digital
- Temporizador / contador

Tarjeta de Multifunción Universal de 100 kS / s, 12-bit, 16-ch (Advantech, 2010)

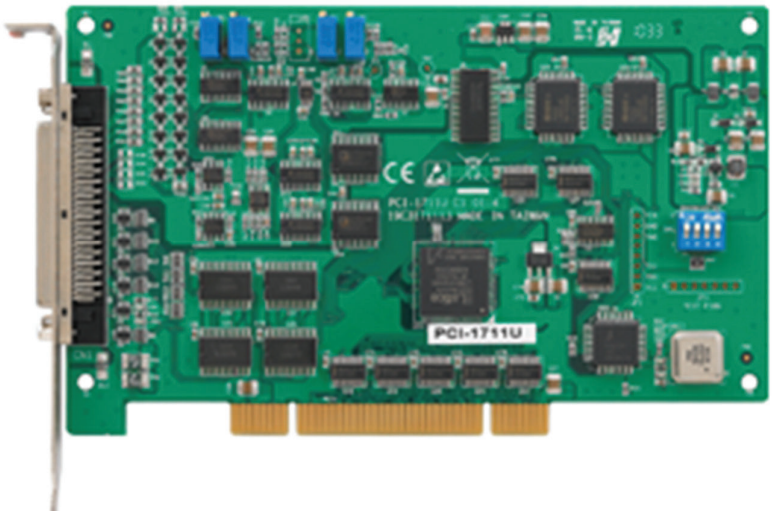

Figura 2.12. Tarjeta de adquisicion de datos pci-1711u.

(Fuente: Advantech, 2010)

Esta tarjeta de adquisición, tiene los recursos necesarios para poder trabajar con el proyecto, y específicamente tiene las siguientes características:

- 16 entradas de un solo extremo
- Convertidor A / D de 12 bits, velocidad de muestreo de hasta 100 kHz
- Ganancia programable para cada canal de entrada
- Exploración automática de canal / ganancia
- Memoria intermedia FIFO de 1K con interrupción seleccionable por software
- 16 entradas digitales y 16 salidas digitales compatibles con TTL
- Dos canales de salida analógica de 12 bits
- Contador programable incorporado
- Universal PCI-Bus (señal de bus 3.3V o 5V PCI-Bus)
- Interruptor de identificación de la tarjeta

Los pines de distribución de la tarjeta PCI-1711U se muestran en la figura 2.13:

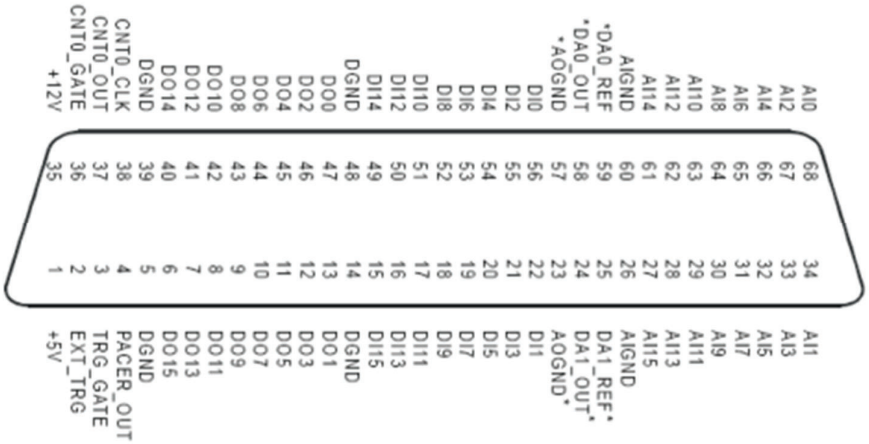

Figura 2.13. I/O pines de conexión.

(Fuente: Advantech, 2010)

2.4.2.1. Instalación de la tarjeta de adquisición pci-1711u en linux

Para instalar la tarjeta de adquisición de datos, es necesario seguir una serie de pasos. Debemos abrir un terminal y empezar e ingresar líneas de código, siempre y cuando la computadora esté conectado a Internet; y tener el CD de instalación de la tarjeta de adquisición que se quiera utilizar.

Paso 1: Abra una terminal. (Sugiera usar la tecla corta "Ctrl + Alt + t" para abrir el terminal)

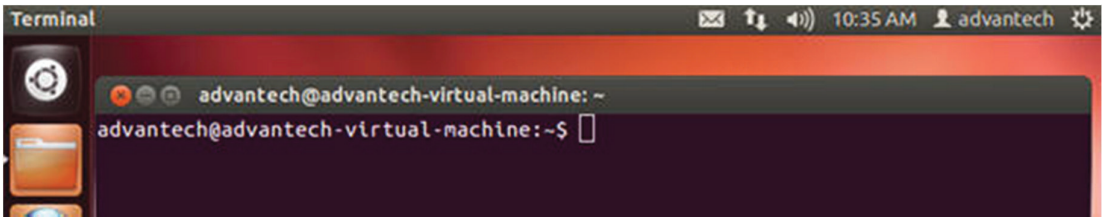

Figura 2.14. Terminal de Linux.

(Fuente: Tipantocta, 2017)

Paso 2: Asegúrese de que el origen del software del sistema esté actualizado. Utilice el comando "sudo apt-get update" para actualizar.

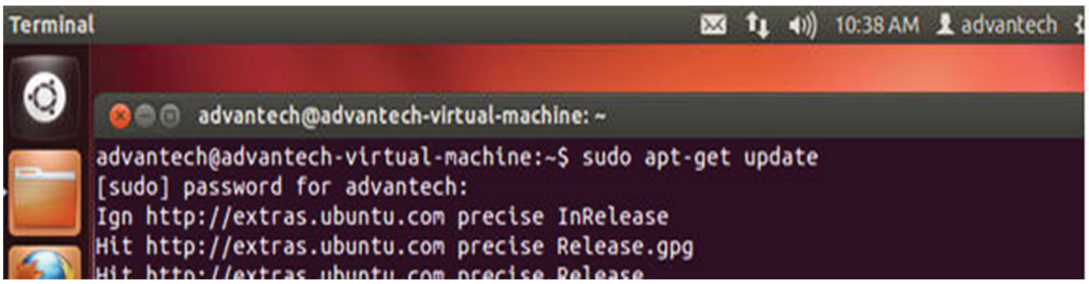

Figura 2.15. Actualización de Linux.

(Fuente: Tipantocta, 2017)

Paso 3: Deshabilite el controlador comedi, porque este controlador entrará en conflicto con algunos de los controladores de DAQ.

Nota: antes de hacer esto, no inserte ningún dispositivo DAQ, especialmente el dispositivo de la serie PCI.

Abra el archivo "/etc/modprobe.d/blacklist.conf" con el comando "sudo gedit /etc/modprobe.d/blacklist.conf"

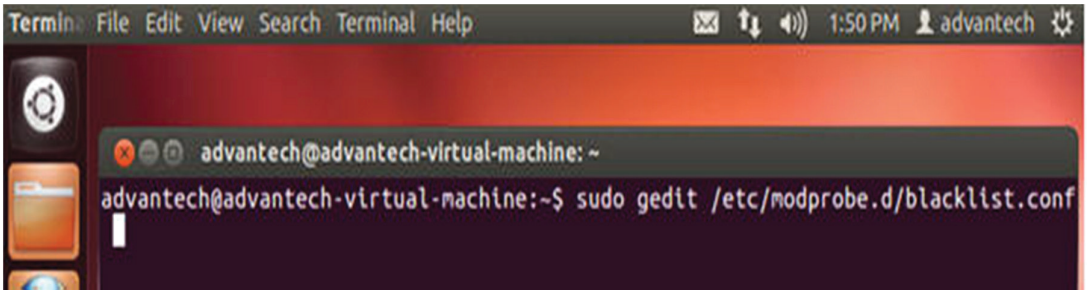

Figura 2.16. Archivo de configuración blclist.conf.

(Fuente: Tipantocta, 2017)

A continuación, agregue "blacklist comedi" "lista negra adv_pci_dio" " blacklist adv_pci1711" al final del archivo.

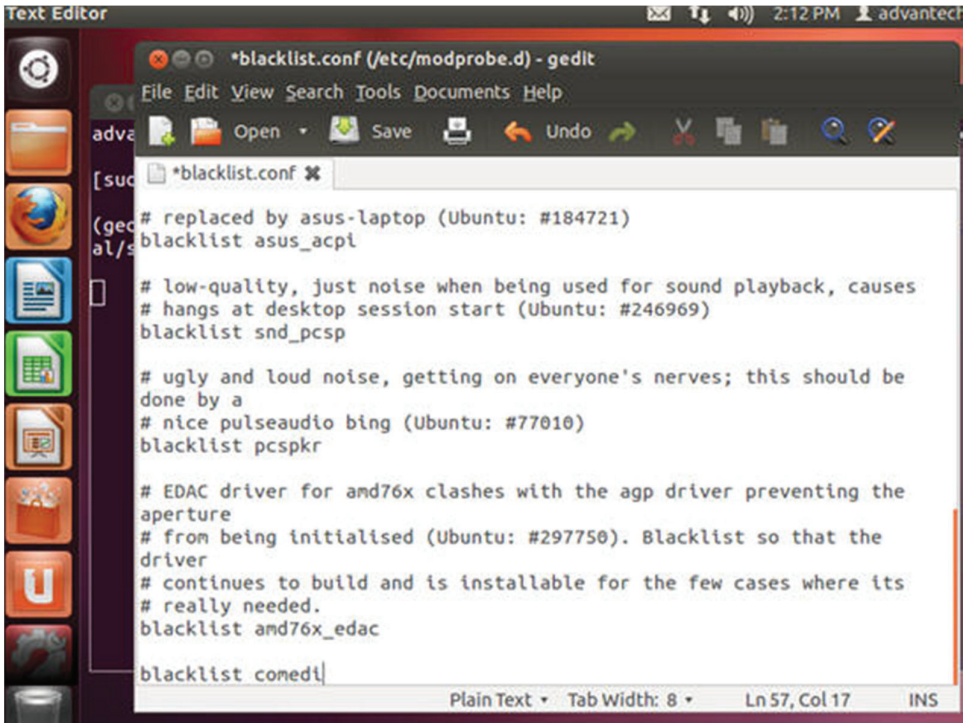

Figura 2.17. Ingreso de parámetros en el archivo de configuración.

(Fuente: Tipantocta, 2017)

Paso 4: Apague el sistema. (Aquí, también puede insertar el dispositivo DAQ en la máquina, pero se sugiere inserte el dispositivo después de completar todos los pasos de instalación del SDK y del controlador).

Paso 5: Encienda el sistema. Abra el terminal, introduzca el siguiente comando. "Sudo apt-get instala build-essential linux-headers - $ (uname -r)".

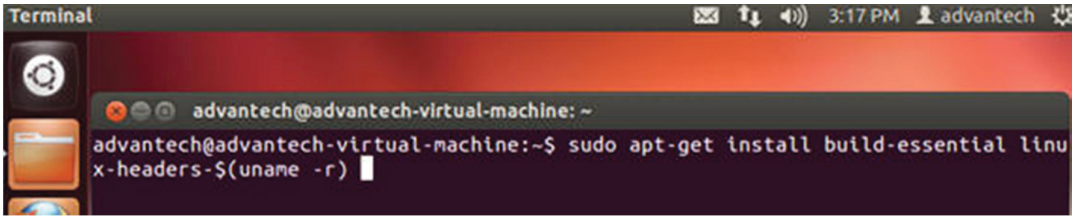

Figura 2.18. Instalación de drivers en Linux.

(Fuente: Tipantocta, 2017)

Paso 6: Copie el paquete de compresión en la carpeta principal. Si su sistema es de 32 bits, copie el archivo "linux_driver_package_x.x.x.x_32bit.zip" y

"linux_driver_source_x.x.x.x_32bit.zip". Si su sistema es de 64 bits, copie el archivo "linux_driver_package_x.x.x.x_64bit.zip" y "linux_driver_source_x.x.x.x_64bit.zip". Aquí utilizamos el sistema de 64 bits como ejemplo. Paquete de compresión del controlador de descompresión. (Nombre como linux_driver_source_3.1.4.0_64bit.zip).

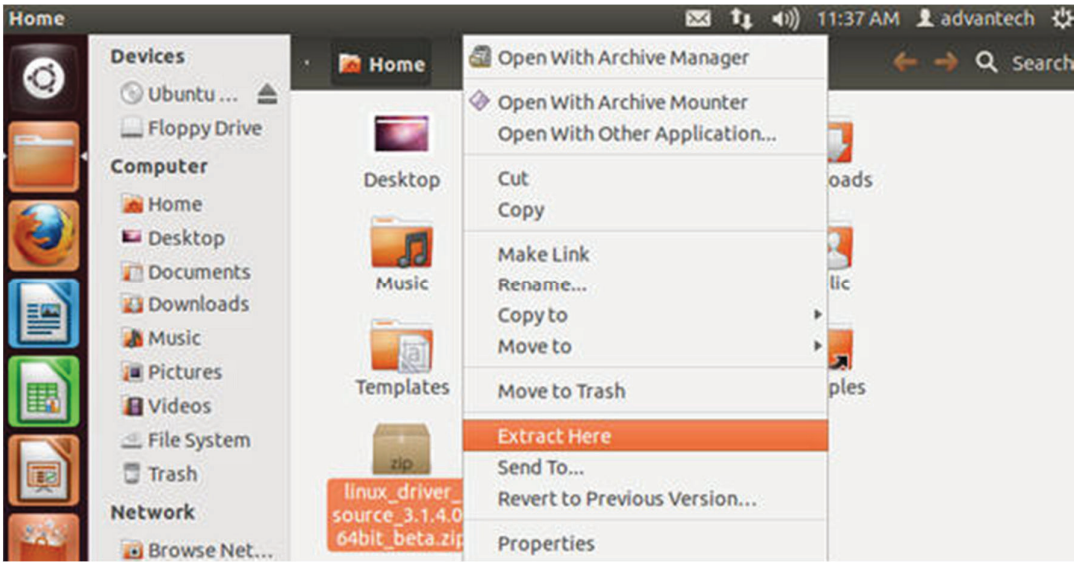

Figura 2.19. Copia de archivos de driver en carpeta raíz.

(Fuente: Tipantocta, 2017)

Paso 7: Introduzca el siguiente comando "cd linux_driver_source_3.1.4.0_64bit_beta /" e ingrese al directorio 'drivers'.

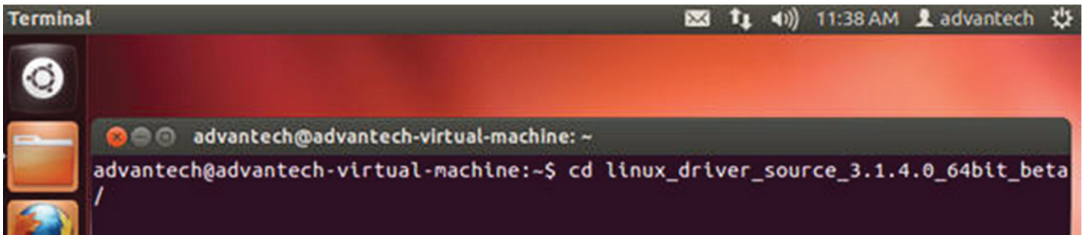

Figura 2.20. Ingreso a la carpeta de drivers.

(Fuente: Tipantocta, 2017)

Paso 8: Instale el controlador de biokernbase. Introduzca el directorio 'driver_base / src / lnx_ko'.

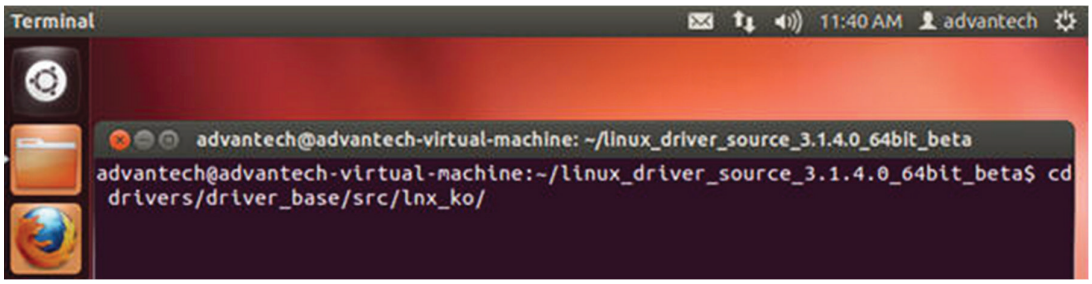

Figura 2.21. Instalación de controlador biokernbase.

(Fuente: Tipantocta, 2017)

Paso 9: Compile los controladores de la base de datos.

Paso 9-1: Obtenga privilegios de root con el comando "sudo -s".

Paso 9-2: Código fuente del controlador de biokernbase compilado. Introduzca el siguiente comando. "make".

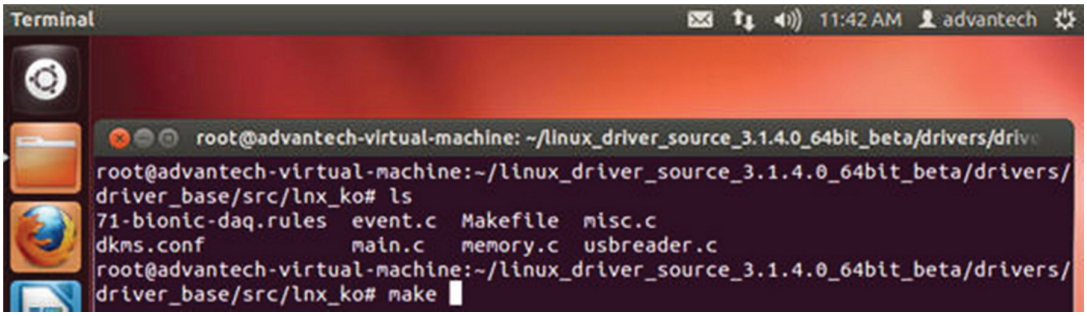

Figura 2.22. Compilación de driver biokernbase.

(Fuente: Tipantocta, 2017)

Paso 10: Compile el controlador de dispositivo DAQ. Por ejemplo: usb4716.

Paso 10-1: Cambie el directorio a usb4716. "Cd ../../../usb4716/src/lnx_ko"

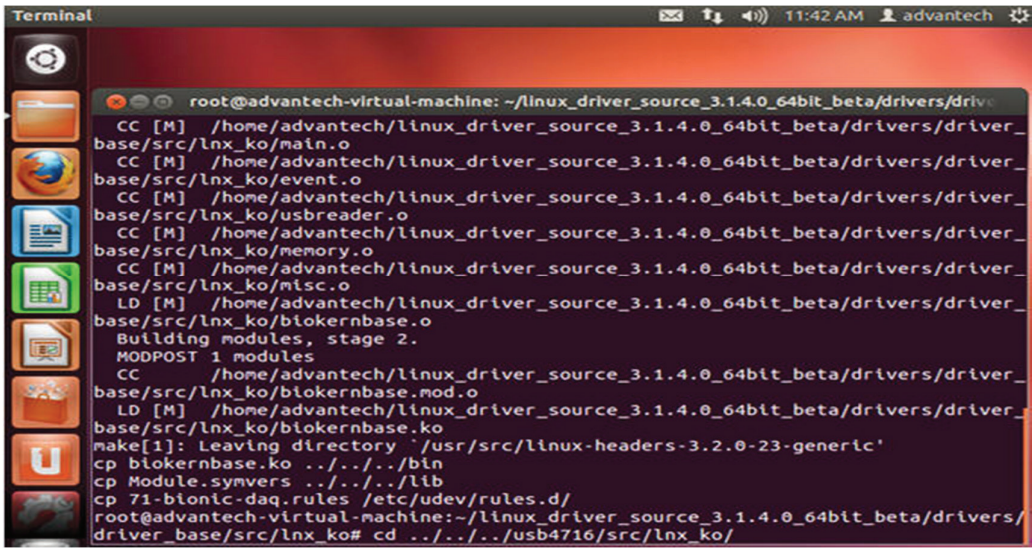

Figura 2.23. Cambio de nombre de directorios necesarios.

(Fuente: Tipantocta, 2017)

Paso 10-2: Código fuente del controlador usb4716 compilado. Introduzca el siguiente comando. "make".

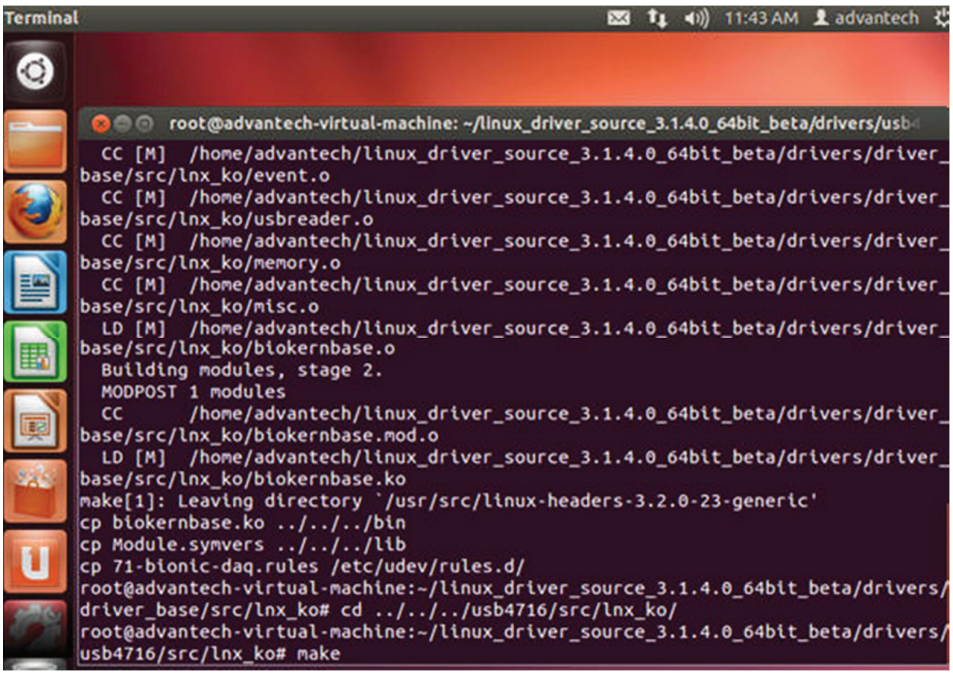

Figura 2.24. Compilación del driver de la tarjeta.

(Fuente: Tipantocta, 2017)

Paso 11: Ubuntu carga automáticamente el controlador DAQ.

Paso 11-1: En el directorio del módulo del controlador linux cree el directorio BIODAQ. Introduzca el siguiente comando: "mkdir / lib / modules / $ (uname -r) / biodaq".

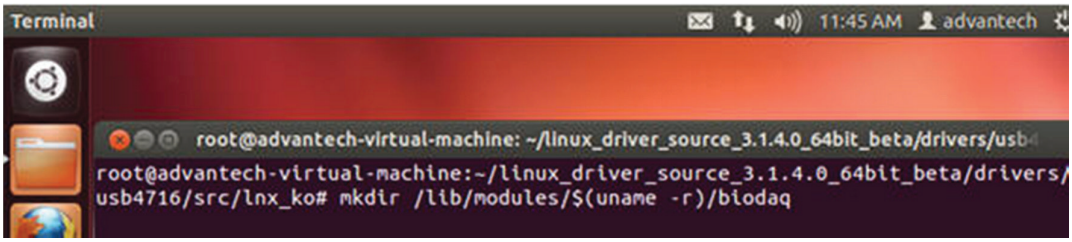

Figura 2.25. Creación del directorio BIODAQ.

(Fuente: Tipantocta, 2017)

Paso 11-2: Volver al directorio 'drivers'. Introduzca el siguiente comando: "cd ../../../". Antes de hacer esto, asegúrese de que el directorio de trabajo actual es "~ / linux_driver_source_3.1.4.0_64bit_beta / drivers / usb4716 / src / lnx_ko"

Paso 11-3: Copie los controladores al directorio BIODAQ. Introduzca el siguiente comando: "cp bin / * .ko / lib / modules / $ (uname -r) / biodaq /".

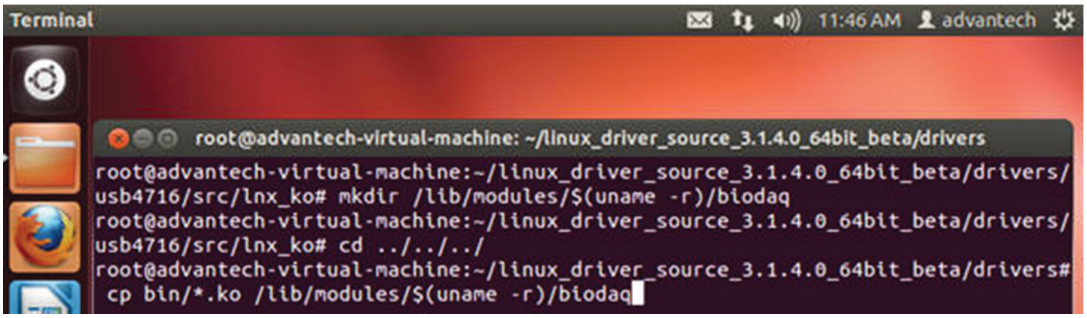

Figura 2.26. Copia de controladores BIODAQ.

(Fuente: Tipantocta, 2017)

Paso 11-4: Permitir que Linux Ubuntu indexe todos los controladores. Introduzca el siguiente comando: "depmod".

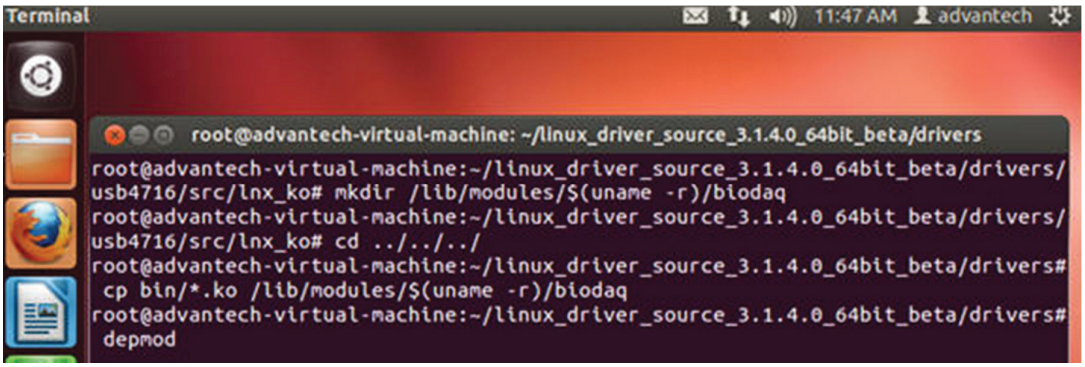

Figura 2.27. Indexado de controladores.

(Fuente: Tipantocta, 2017)

Paso 11-5: Abra el archivo / etc / modules. Introduzca el siguiente comando: "gedit / etc / modules".

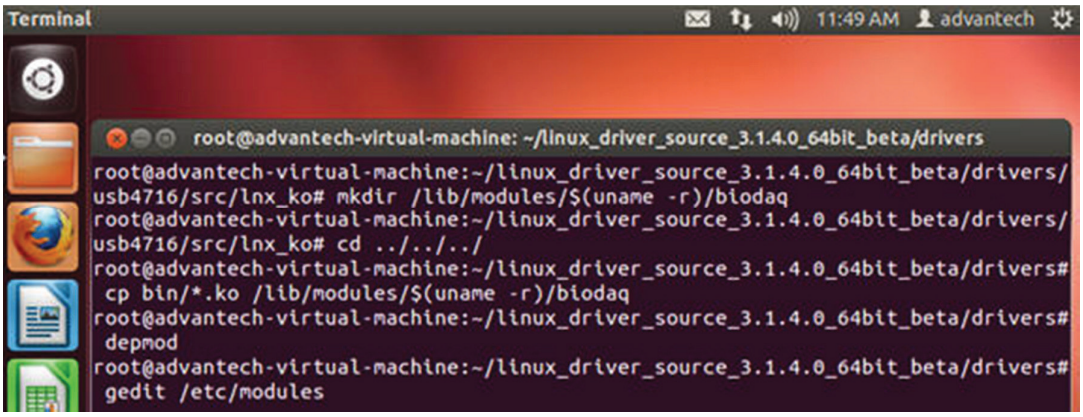

Figura 2.28. Apertura del archivo Modules.

(Fuente: Tipantocta, 2017)

Paso 11-6: El nombre que se necesita para cargar automáticamente se agregó a este archivo. Cierre y guarde el archivo.

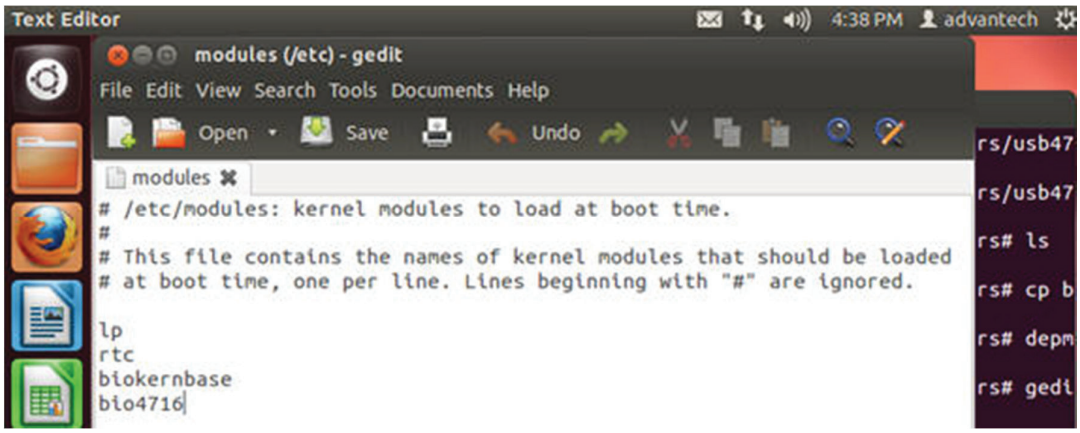

Figura 2.29. Escritura del driver instalado.

(Fuente: Tipantocta, 2017)

Paso 12: Copie la biblioteca dinámica en el directorio del sistema.

Cambie el directorio a 'libs'. Introduzca el siguiente comando: "cd ../libs/" y luego copie toda la biblioteca en el directorio '/ usr / lib /'. Introduzca el siguiente comando. "Cp * / usr / lib /"

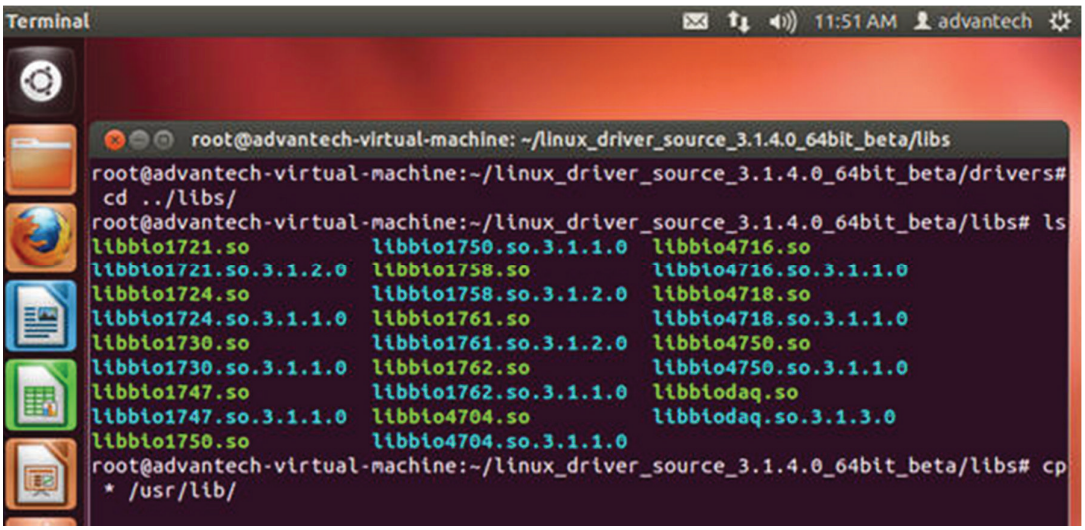

Figura 2.30. Copia de biblioteca en el directorio LIBS.

(Fuente: Tipantocta, 2017)

Paso 13: Copie el archivo de encabezado al directorio del sistema.

Paso 13-1: Cambie el directorio a "inc" usando el comando "cd ../inc/", luego copie el archivo de encabezado al directorio "/ usr / include /" usando el comando "cp bdaqctrl.h / usr / include/

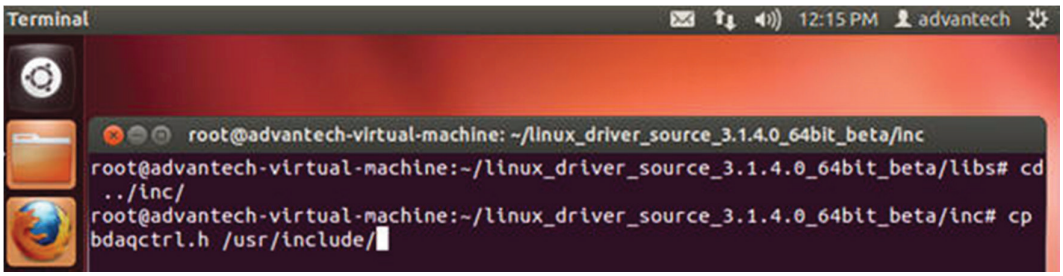

Figura 2.31. Copia de archivo de encabezado al directorio.

(Fuente: Tipantocta, 2017)

Paso 13-2: Cree una carpeta para almacenar el archivo JAR. Introduzca el siguiente comando. "Mkdir / usr / share / advantech"

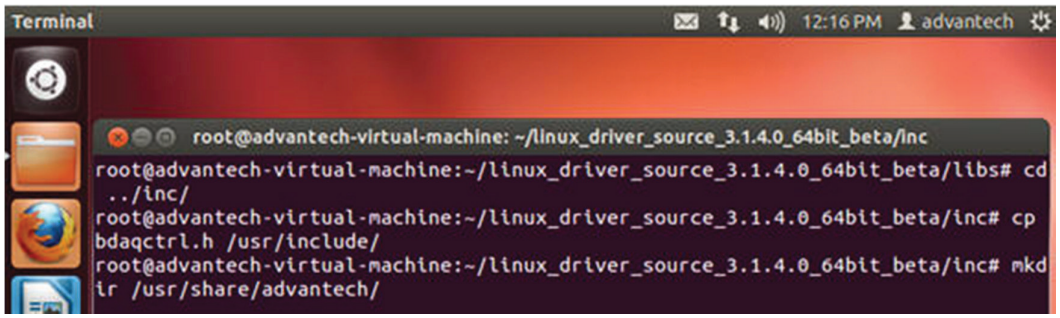

Figura 2.32. Creación de carpeta para almacenamiento de archivos JAR.

(Fuente: Tipantocta, 2017)

Paso 13-3: Copiar JAR al directorio 'advantech'. Introduzca el siguiente comando "cp Automation.BDaq.jar / usr / share / advantech /".

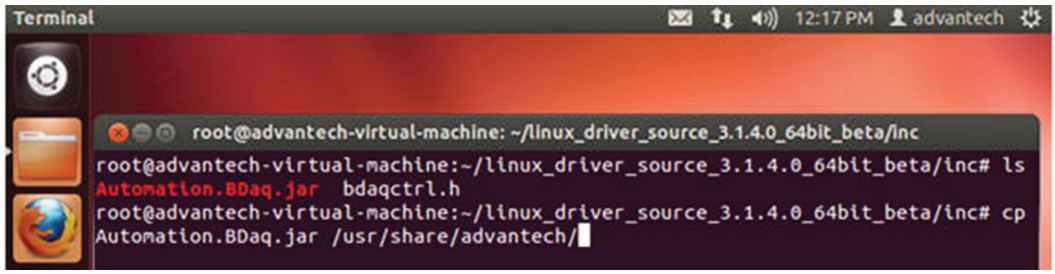

Figura 2.33. Copia de archivo JAR en directorio Advantech.

(Fuente: Tipantocta, 2017)

Paso 13-4: Cambiar el archivo Automation.BDaq.jar a todos los atributos legibles.

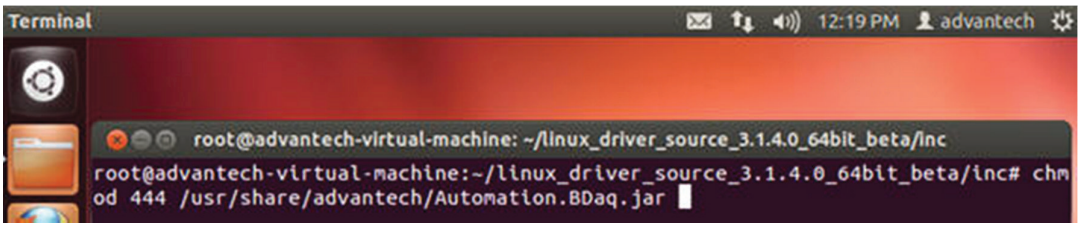

Figura 2.34. Cambio de atributos.

(Fuente: Tipantocta, 2017)

2.4.2.2. Instalación de la tarjeta pci-1711u

Después de instalar el controlador de dispositivo DAQNavi y los SDK, se debe seguir los siguientes pasos para instalar el dispositivo DAQ en la computadora.

Paso 1: Apague el ordenador y los accesorios conectados, apague la alimentación.

Nota: Apague la fuente de alimentación de la computadora siempre que instale o retire tarjetas, o conecte y desconecte los cables.

Paso 2: Si el ID de la tarjeta del dispositivo puede ser configurado por Switch, marque el conmutador y configure el ID de la tarjeta como un valor distinto de cero.

Paso 3: Sostenga con cuidado el borde del dispositivo e insértelo en su sistema. Asegure el dispositivo con tornillos.

Seguidos todos los pasos de instalación, se puede trabajar ya con la tarjeta de adquisición, en Linux.

2.5. Conversor ac-dc semicontrolado

Los conversores Semicontrolados son un tipo de convertidor de un sólo cuadrante ver Figura 2.35 y tiene una misma polaridad de voltaje y de corriente de salida. Emplean en sus configuraciones ramas rectificadoras con, cada una de ellas, un diodo y un tiristor.

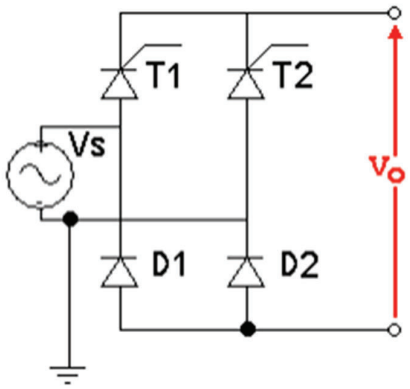

Figura 2.35. Conversor de onda completa Semicontrolado

(Fuente: Tipantocta, 2017)

En la figura 2.36 se muestra el cuadrante trabajo del conversor AC-DC semicontrolado.

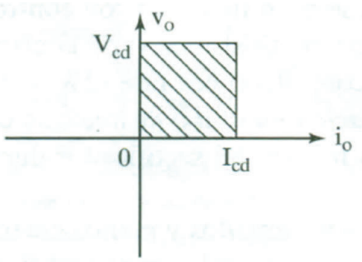

Figura 2.36. Primer cuadrante.

(Fuente: Tipantocta, 2017)

El circuito rectificador monofásico semicontrolado que se muestra en la figura 2.35, permite variar la componente de la tensión de salida en función del troceado producido por una pareja de tiristores de acuerdo con el ángulo de fase de disparo de los mismos. El Vdc se determina con:

$$V_{DC} = \frac{1}{T}\int v(t)dt \qquad (2.1)$$

El voltaje promedio de salida se calcula con los límites de fase con los que va a funcionar como muestra la figura 2.37.

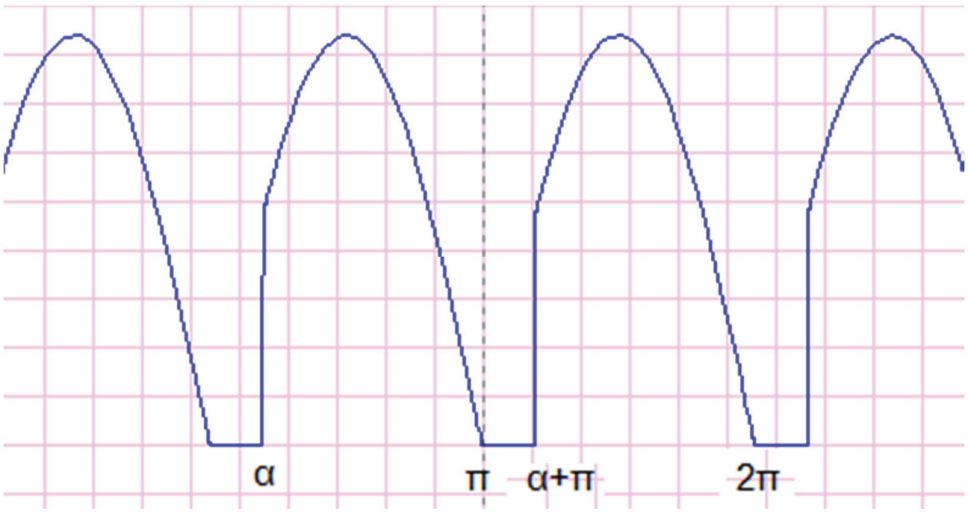

Figura 2.37. Señal de salida del conversor AC-DC.

(Fuente: Tipantocta, 2017)

Por lo tanto el voltaje de salida DC será:

$$V_{DC} = \frac{1}{T}\int_{\alpha}^{\pi} v(t)dt \tag{2.2}$$

Donde

V(t) = Voltaje de entrada en función del tiempo = $Asen(wt)$

T = Periodo de onda = π

w = 2π*f

f = frecuencia

Resolviendo

$$V_{DC} = \frac{A}{T}(\cos(\alpha) + 1) \tag{2.3}$$

Para la demostración del controlador en tiempo real, se tomará en cuenta la aplicación de un sistema de control de velocidad en lazo cerrado, para ello es necesario equipar un sistema motor-generador y así exponer la aplicación del controlador en base a orocos,y el hardware necesario para su funcionamiento, en el siguiente apartado se muestra que es el sistema Motor-Generador.

2.6. Sistema motor generador

La carga a usar es un motor de corriente continua, el cual va a ser regulado por un conversor AC-DC, diseñado y controlado a través de la tarjeta PCI-1711.

Para demostrar el funcionamiento del controlador se debe realimentar uno o varios parámetros de la planta, para lo cual sería un sistema en lazo realimentado, por tal motivo se ha escogido colocar otro motor de corriente continua acoplado por una banda al primer motor y así armar un sistema motor-generador, con este método se pretende medir la velocidad del motor, con respecto al voltaje generado.

En el siguiente apartado se explicará mecánicamente cómo funciona el sistema de transmisión de velocidad en base al movimiento rotatorio del motor de corriente continua.

2.6.1. Transmisión por poleas y correas

Un sistema de transmisión por correa es un conjunto de dos poleas acopladas por medio de una correa con el fin de transmitir fuerzas y velocidades angulares entre árboles paralelos que se encuentran a una cierta distancia.

2.6.1.1. Poleas

Las poleas no son más que una rueda (llanta) con un agujero en su centro para acoplarla a un eje en torno al cual giran. Para asegurar el contacto entre polea y correa se talla en la polea un canal o garganta que "soporta" a la correa.

En un sistema de transmisión de poleas son necesarias dos de ellas:

- **Una conductora**, de entrada o motora, que va solidaria a un eje movido por un motor.
- **Otra conducida,** de salida o arrastrada, también acoplada a un eje y que es donde encontraremos la resistencia que hay que vencer.

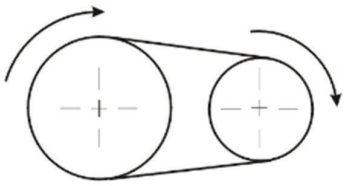

Figura 2.38. Fricción por poleas.
(Fuente: Tipantocta, 2017)

2.6.1.2. Transmisión por correa

La correa seleccionada es del tipo Trapezoidal, son las correas más utilizadas, pues se adaptan firmemente al canal de la polea evitando el posible deslizamiento entre polea y correa.

Figura 2.39. Correa trapezoidal.
(Fuente: Tipantocta, 2017)

La fuerza se transmite por efecto del rozamiento que ejerce la correa sobre la polea. El proceso de transmisión del movimiento con correa es un proceso de elevado rendimiento (95-98%) y precio reducido.

La ecuación fundamental de velocidades para transmisiones por correa es:

$$\emptyset 1 * n1 = \emptyset 2 * n2 \tag{2.4}$$

En el siguiente capítulo, se diseñará el controlador, tomando en cuenta lo visto en este capítulo, tanto en hardware como en software y con esto se pretende demostrar la aplicación con el middleware Orocos.

CAPÍTULO 3

3. DISEÑO DEL CONTROLADOR

Para demostrar el uso del Middleware Orocos, se pretende diseñar un controlador en tiempo real creado por componentes, este será integrado en un sistema de control de velocidad como muestra la figura 3.1.

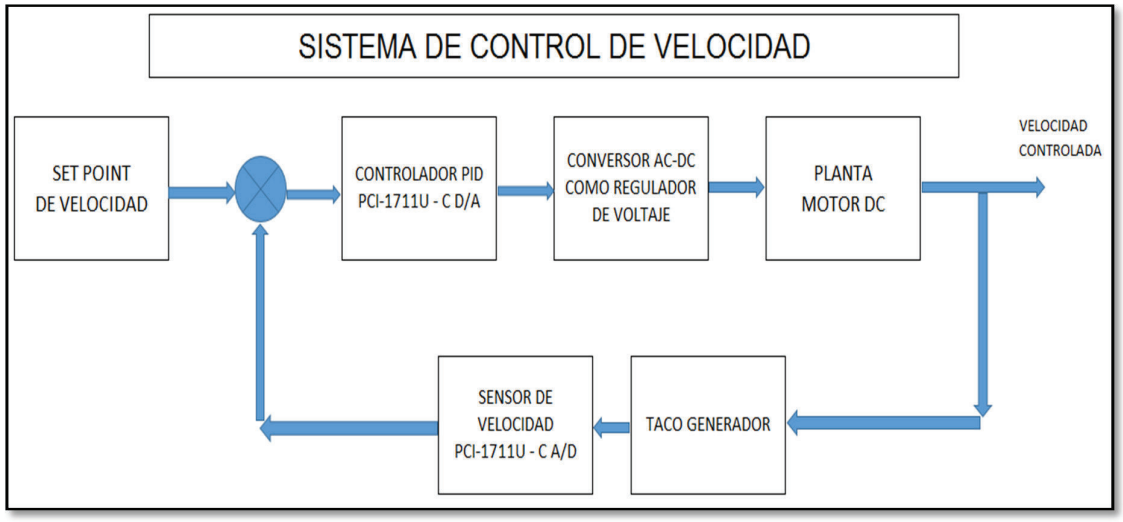

Figura 3.1. Sistema de control de velocidad.

(Fuente: Tipantocta, 2017)

Para el diseño de este sistema de control se detallan los elementos que lo constituirán:

- Set Point. Será constituido por un potenciómetro el cual cambiará al valor de referencia de seteo entre 0 y 1000 RPM. La señal de este elemento será adquirido por la tarjeta de adquisición de datos PCI-1711U, por medio de un canal análogo a digital y procesando la información internamente.
- Controlador PID. Se creará el controlador en tiempo real en base a la librería Orocos e internamente por software, el cual medirá el error que existe entre el set point y la realimentación, para así determinar la acción de control.
- Conversor AC-DC. El conversor regulará el voltaje en DC, el cual permitirá aumentar o disminuir la velocidad de giro de un motor de corriente continua. La señal de regulación lo realizará en base al controlador en tiempo real diseñado,

dicha señal saldrá de un canal digital a análogo de la tarjeta PCI-1711U y el conversor aceptará la señal de control.

- Planta. La planta escogida es un motor de corriente continua de laboratorio que tiene ciertas características mecánicas y eléctricas. Este elemento servirá para su modelamiento matemático y el diseño del controlador.
- Taco Generador. Para sensar la velocidad del motor, se armará un juego de poleas con una banda entre el motor que servirá de planta y otro motor de corriente continua, convirtiéndose en un sistema motor-generador. La señal que emitirá el generador será adquirida por la tarjeta PCI 1711U, por medio de un canal análogo a digital, procesando así la señal de realimentación.

Con el sistema presentado se pretende demostrar la efectividad que tiene el trabajar en controladores de tiempo real, y la facilidad de trabajar con el Middleware Orocos.

3.1. Modelo de la planta

Para diseñar el controlador en tiempo real es primordial empezar con el modelo matemático de la planta a controlar. La planta escogida es un motor de corriente continua de laboratorio, el cual tiene parámetros tanto eléctricos como mecánicos.

El motor es un dispositivo analógico, y las características eléctricas y mecánicas que describen su comportamiento se lo realizarán en forma de tiempo continuo. Este modelo será descrito en una forma discreta, en el dominio de Z para el uso de un controlador digital. El medio ZOH es usado para la transformación del modelo en una forma discreta.

En general, las características eléctricas de un motor de corriente continua o DC están descritas por:

$$L\frac{di}{dt} + Ri = V - emf \tag{3.1}$$

Donde:

L = Inductancia del motor

R = Resistencia

V = Voltaje aplicado

I = Corriente

Emf= Fuerza electromotriz = Ke x θ

Las características mecánicas están descritas por:

$$J_M \frac{d^2\theta}{d_t{}^2} + B\frac{d\theta}{d_t} + K\theta = T_L - J_L \frac{d^2\theta}{d_t{}^2} \quad (3.2)$$

Donde:

J_M = Inercia del motor

θ = Desplazamiento angular

K = Constante de rigidez

B = Constante de amortiguación

J_L = Carga inercial

T_L = Torque de inercia = Kt x i

Kt = Torque constante

La Figura 3.2 muestra una representación del modelo eléctrico y mecánico de un motor DC.

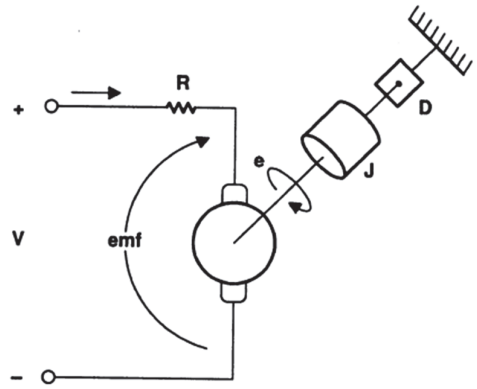

Figura 3.2 Modelamiento de un motor DC

(Fuente: Ogata, 2010)

El motor seleccionado es uno de corriente continua de laboratorio con los siguientes parámetros:

R = 6.4Ω

L=3 mH

J = $1.54x10^{-6}kgxm^2$

Kt = 0.0207(Nxm)/A

Ke = 0.0206(volt)/(rad/s)

En función de las características eléctricas y mecánicas de la ecuación (3.1) y (3.2) la función de transferencia resultante del modelo de la planta es:

$$G(s) = \frac{53.906}{s(s+1.16)} \tag{3.3}$$

Dado que la tarjeta de adquisición PCI-1711U tiene 16 canales para señales de entrada analógicas con una resolución de 12 bits, se toma en cuenta la información del elemento para poder discretizar el controlador y la planta. En el siguiente apartado se demostrará cómo realizar un controlador PID en el dominio de Z o discreto, por tal motivo se utilizará la herramienta computacional de Matlab para la demostración del controlador.

3.1.1. Modelo de la planta en tiempo discreto

El software permite realizar la transformación del tiempo continuo de una función de transferencia a tiempo discreto por medio de una función **c2d,** pero para nuestro fin también se crea la función de transferencia por medio de la instrucción **zpk**; en las siguientes líneas se realiza la operación de creación de la función de transferencia y pasar al dominio de Z.

- Modelo de la planta en MATLAB

```
gs=zpk([],[0 -1.116],53.906)
T=0.01;
gz=c2d(gs,T)
```

Donde el software da como resultado la transformación en el dominio Z de la planta.

$$G(z) = \frac{0.0026853\ (z+0.9963)}{(z-1)\ (z-0.9889)} \tag{3.4}$$

3.2. Diseño de controlador pid

El método que se tomará para el diseño del controlador PID discreto será de aproximación rectangular, el diseño se encuentra realizado en el dominio analógico y convertido en el dominio de Z. El algoritmo del PID está dado por:

$$u(t) = K_p e(t) + K_i \int dt + K_d \frac{de}{dt} \tag{3.5}$$

Donde:

Kp,Ki y Kd = PID constantes

u(t) = salida del controlador

e(t) = Señal de error

Para discretizar se asume el intervalo de muestreo para el sistema que es T=0.01 s. Con la aproximación rectangular que es el método más sencillo, obteniendo resultados satisfactorios, ésta asume la integral $\int edt$ que es una acumulación de pequeños rectángulos y el diferencial puede ser aproximado como:

$$\frac{de}{dt} = \frac{e(n) - e(n-1)}{T} \tag{3.6}$$

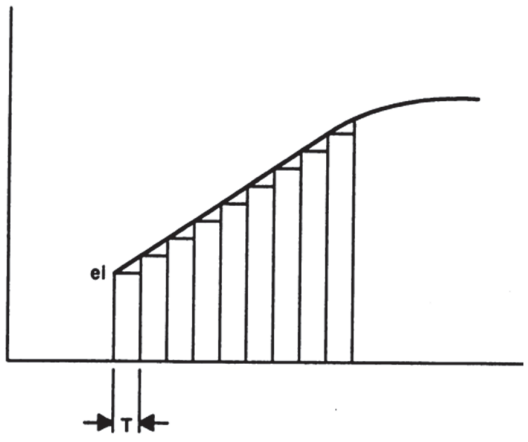

Figura 3.3 Aproximación rectangular

(Fuente: Tipantocta, 2017)

La fórmula general del controlador PID por el método de aproximaciones queda como:

$$u(n) = u(n-2) + K1e(n) + K2e(n-1) + K3e(n-2) \qquad (3.7)$$

Donde:

K1 = Kp + Kd/T + Ki/T

K2 =KiT – 2Kd/T

K3 =Kd/T – Kp

u(n) = señal de control en el intervalo de tiempo n

u(n-2) = n-2 muestras

e(n) = señal de error en el tiempo n

e(n-1) = señal de error en el tiempo n-1

T = Intervalo de muestreo

3.2.1. Programa de aplicación en matlab

En el siguiente programa de aplicación realizado en el software computacional MATLAB, se extrae las constantes necesarias el controlador PID, entregando las constantes K1, K2, K3 y presentando en el gráfico del controlador, de aquí parte la necesidad de las constantes TI y TD.

- Se extrae parámetros
  [n,d]=tfdata(gz,'V');
- Discretización de la aproximación rectangular
  b1=n(2);b2=n(3);a1=d(2);a2=d(3); %
- Parámetro necesarios para la extracción de la aproximación calculamos Ku y Tu:
  Ku=(1-a2)/b2
  alfa=-1/2*(Ku*b1+a1);
  Tu=2*pi*T/acos(alfa)
- Sintonización por Ziegler Nichols
  Kc=0.6*Ku
  Ti=Tu/2;Td=Tu/8;
  Ki=Kc/Ti
  Kd=Kc*Td

- Parámetros del controlador discreto

  K1=Kc+Ki*T+Kd/T;

  K2=-Kc-2*Kd/T;

  K3=Kd/T;
- Función de Transferencia del controlador PID

  Cz=tf([K1 K2 K3],[1 -1 0],T)
- Resultado

  step(feedback(gz*Cz,1))

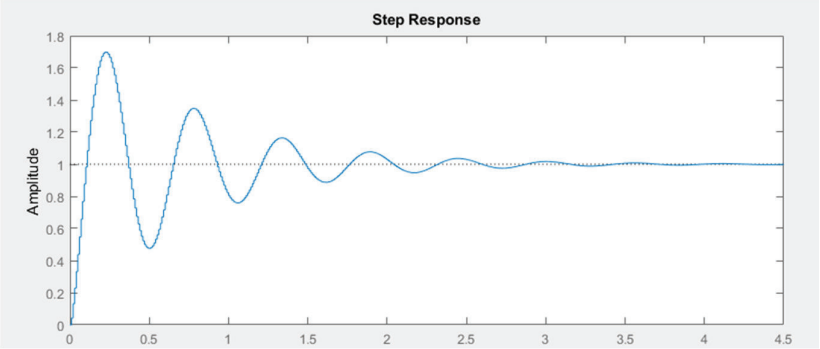

Figura 3.4. Respuesta a una señal escalón.

(Fuente: Tipantocta, 2017)

Por medio de SIMULINK se puede realizar la práctica de discretización de una manera más efectiva como muestra la Figura 3.5

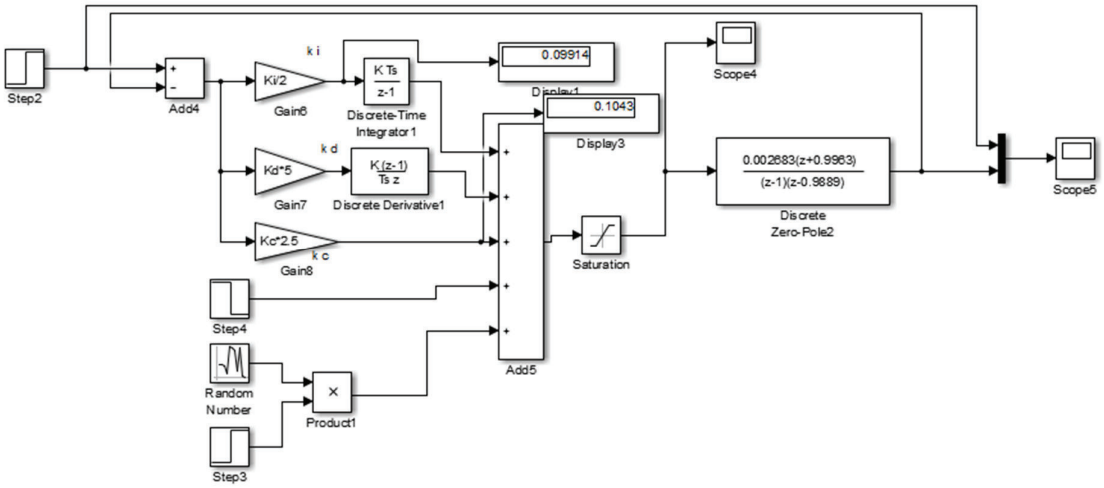

Figura 3.5. Controlador PID discreto.

(Fuente: Tipantocta, 2017)

El diagrama muestra las constantes Kc, Ki, Kd con los parámetros de ganancia, derivación, e integración sumadas y conectadas al modelo de la planta, también se coloca una perturbación randomica y entregando una respuesta controlada como se muestra en la figura 3.6.

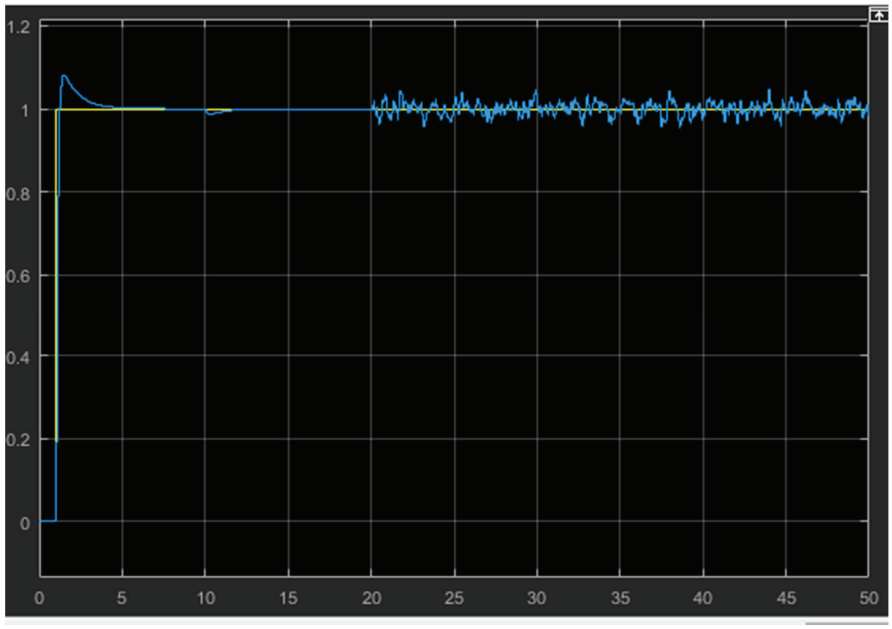

Figura 3.6 Respuesta a una señal escalón de un controlador PID discreto

(Fuente: Tipantocta, 2017)

3.3. Diseño del componente para la creación del controlador

3.3.1. Compilación

Para saber si todo va bien lo mejor es compilar, si no devuelve ningún error al compilar entonces se puede ejecutar el deployer con el script. Para compilar se debe ingresar a la carpeta de trabajo catkin_ws y ejecutar la instrucción catkin_make:

$ cd /home/fabry/catkin_ws/

$ catkin_make

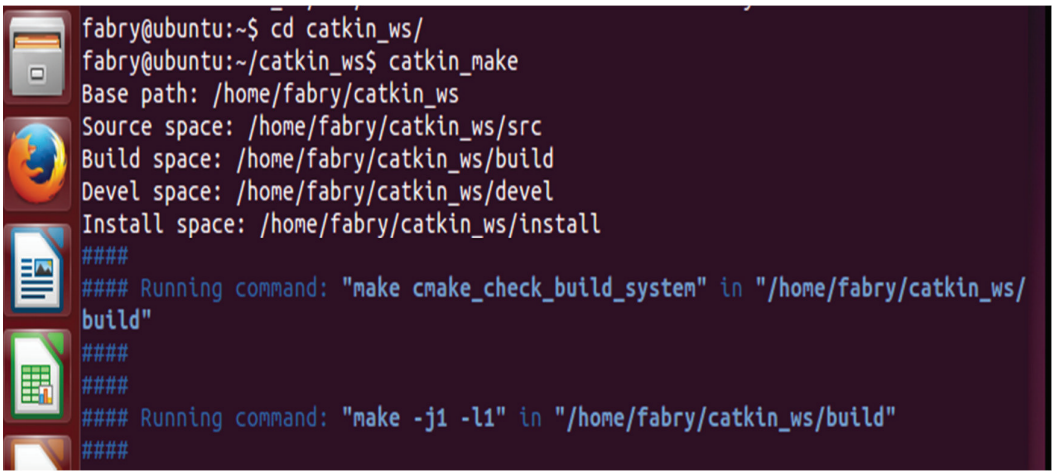

Figura 3.7. Pantalla de compilación.

(Fuente: Tipantocta, 2017)

Si todo es correcto el compilador muestra un 100% sin ningún error, pero si existiese algún error en la compilación, mostrará las direcciones de donde posiblemente se encuentra el error cometido.

3.3.2. Conexión con el driver pci-1711

Para poder incluir el driver de la tarjeta PCI-1711, se debe colocar algunas instrucciones que ayudarán a que el mismo sea compatible con Orocos. En primera instancia se incluyen dos librerías que deben ser copiadas en la carpeta raíz donde se ha generado el componente, y en el archivo con extensión “daq-component.hpp” se copia las librerías como son: “compatibility.h” y “bdaqctrl.h” y se declara un nombre de uso de espacio conocido como “Automation::Bdaq”, se muestra en la Figura 3.8.

Para el diseño del controlador PID, se creó otro componente denominado dac1711, en el cual se programó todos los drivers necesarios para que funcione la tarjeta y de igual manera el controlador digital.

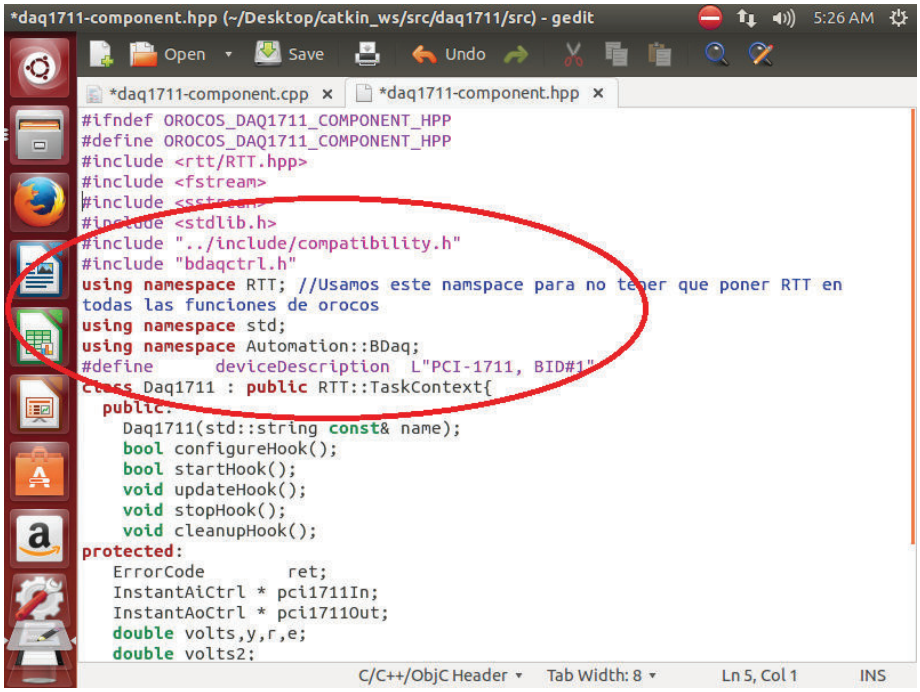

Figura 3.8. Driver de tarjeta PCI1711 usada en el programa del componente daq1711-component.hpp.

(Fuente: Tipantocta, 2017)

Uno de los problemas por el cual no se reconocía la tarjeta, es de una constante donde se define la descripción de la tarjeta, y con la ayuda de la empresa distribuidora de las mismas se hizo mención del cambio de una dirección que la figura 3.9 se muestra señalado como:

```
#define      deviceDescription  L"PCI-1711, BID#1"
```

Figura 3.9. Cambio de dirección en la instrucción.

(Fuente: Tipantocta, 2017)

Esta instrucción permite el reconocimiento como hardware y en el ejemplo que se entrega en el CD de instalación hacen referencia a cualquier elemento como “Device” y para uso exclusivo de esta tarjeta se cambió por “PCI-1711”.

3.3.3. Componente del controlador PID

Para la primera sección del programa “daq1711-component.cpp” mostrado en la Figura 3.10, se anotan los atributos de conectividad, estos atributos permiten la conexión del componente con diferentes tipos de programas, en nuestro caso son los parámetros de

inicio del componente para poder cambiar las constantes del controlador PID como son: “Kp”, “Ti”, y “Td” y así no tener que compilar el programa siempre con valores preseteados.

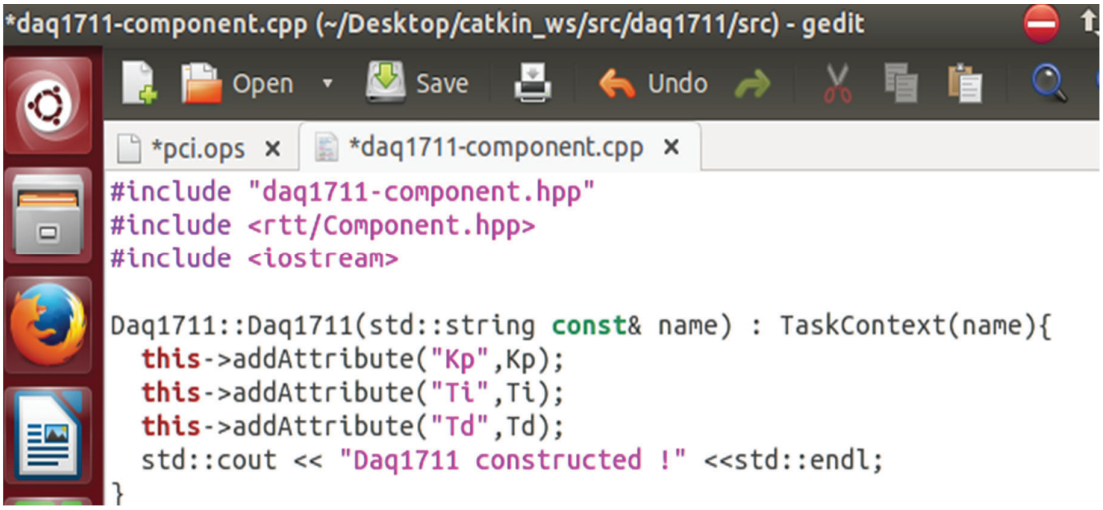

Figura 3.10. Atributos de ingreso al programa del controlador.
(Fuente: Tipantocta, 2017)

En la siguiente sección, permite configurar todas las variables de ingreso que se pretenden utilizar en el programa y generalmente todas ellas se colocan en su valor de inicio. También en esta sección es preferible realizar la configuración de las variables que interactúan con el driver de la tarjeta PCI-1711 como por ejemplo:

- pci1711In = AdxInstantAiCtrlCreate();
- pci1711Out= AdxInstantAoCtrlCreate();

Donde dichas instrucciones permiten sincronizar a variables creadas, con el ingreso y salida de datos del conversor análogo-digital y digital-análogo que tiene la tarjeta, y toda la configuración necesaria se encuentra en la figura 3.11.

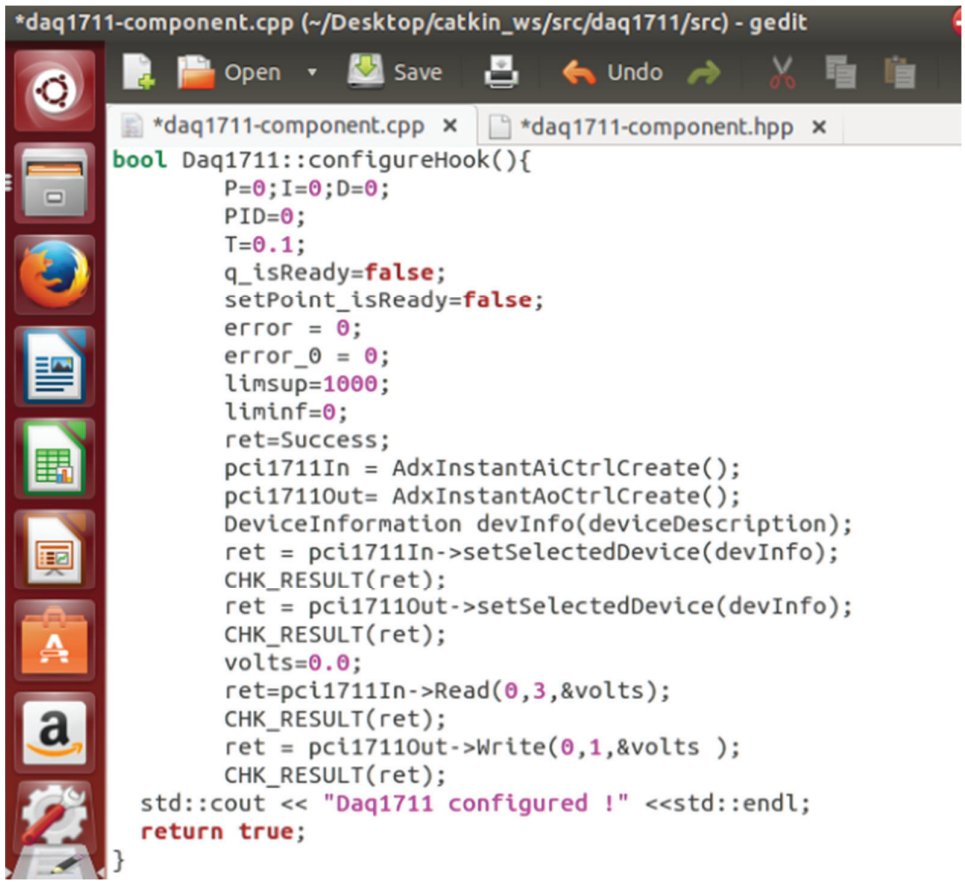

Figura 3.11. Configuración de inicio.

(Fuente: Tipantocta, 2017)

La siguiente sección será configurar el algoritmo del controlador PID, que será el programa principal el cual contendrá todas las instrucciones respectivas para que el sistema de control maneje a la planta modelada por estudio anterior e ingresar los valores de las constantes de proporcionalidad, integración y derivación creando así el controlador PID para la planta modelada que será nuestro motor DC.

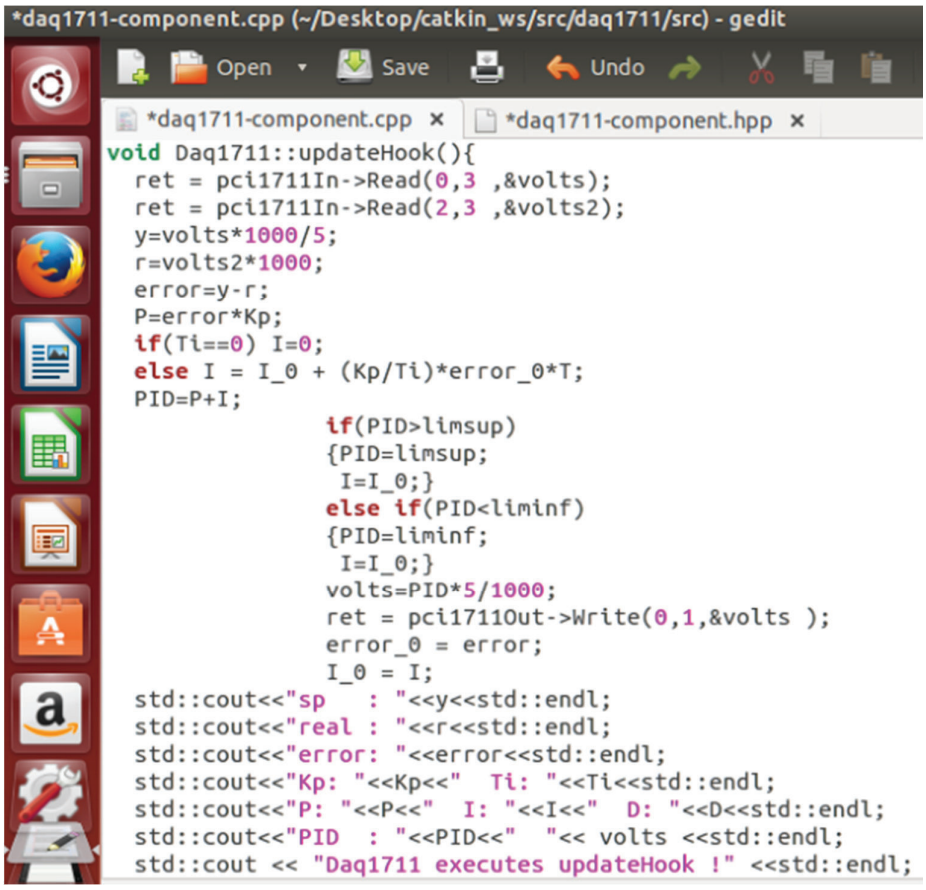

Figura 3.12. Programa del controlador PID.

(Fuente: Tipantocta, 2017)

El algoritmo del controlador PID se encuentra sustentado en bases del modelamiento de la planta, la discretización y la simulación realizada en la herramienta computacional Matlab.

3.4. Diseño del conversor ac-dc

Se ha mencionado anteriormente que el conversor regulará la velocidad del motor usado como planta, este circuito recibirá la señal análoga desde la tarjeta PCI-1711U instalada en la PC.El conversor estático AC a DC, es un circuito electrónico que permite regular el voltaje de carga, dependiendo el control y el tipo de conversor que vaya a usarse. Para el proyecto se pretende realizar un conversor AC-DC del tipo semicontrolado que por medio de control de fase, permita variar el voltaje de 0 a un máximo de 90VDC. En los siguientes apartados se diseña los elementos necesarios para poder construirlo.

3.4.1. Circuito de fuerza

Para el caso práctico, se asumirá el voltaje de la red eléctrica para realizar la conversión de energía de alterna a continua, el conversor a utilizar será un conversor AC-DC semicontrolado que dependiendo de la regulación de energía por control de fase se tendrá un valor máximo de hasta 108V hasta un valor mínimo de 0 V, y sabiendo las reglas de electrónica de potencia se puede construir el conversor de tal forma que:

$$Vdc = \frac{2V_m}{\pi} \tag{3.8}$$

$$Vdc = \frac{2*170}{\pi} = 108V\ para\ \propto= 0 \tag{3.9}$$

El valor de 108 V será el valor máximo de voltaje que se entregará la carga, para el cálculo de los tiristores y diodos se tomará en cuenta los parámetros del motor de resistencia e inductancia y de igual forma para $\propto= 0$:

$$Vrms = \frac{Vm}{2}\left[\frac{1}{\pi}\ (\pi - \alpha + \frac{sen2\alpha}{2}\right]^{\frac{1}{2}} \tag{3.10}$$

$$Vrms = \frac{Vm}{2}\left[\frac{1}{\pi}\ (\pi - 0 + \frac{sen(2*0)}{2}\right]^{\frac{1}{2}}\ para\ \alpha = 0 \tag{3.11}$$

$$Vrms = \frac{170}{2}\ para\ \alpha = 0 \tag{3.12}$$

$$Vrms = 85\ V\ para\ \alpha = 0 \tag{3.13}$$

$$Irms = \frac{Vrms}{Z} \tag{3.14}$$

$$Z = R + jXL \tag{3.15}$$

Donde:

$$R = 6.4\Omega; Xl = 2 * \pi * 60 * 10mh = 3.76\Omega \tag{3.16}$$

$$Z = 6.4 + j3.76\ \Omega \tag{3.17}$$

$$Z = 7.42/30.4°\ \Omega \tag{3.18}$$

$$Irms = \frac{85V}{7.42\Omega} = 11.45\ A \tag{3.19}$$

Por lo cual a una corriente de 11.45 A, el elemento seleccionado es un tiristor de 12 A, el BT151

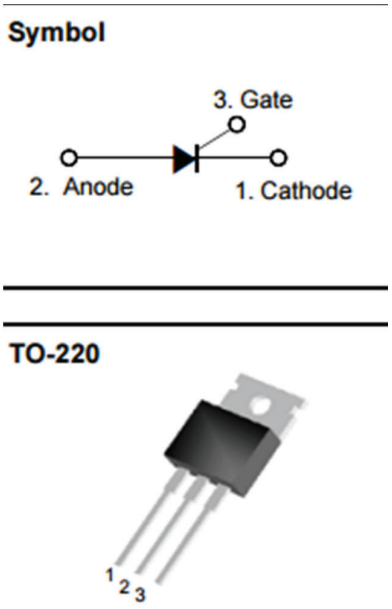

Figura 3.13. Tiristor BT151.

(Fuente: Bolton, 2014)

Siendo el circuito de fuerza como en la figura 3.14:

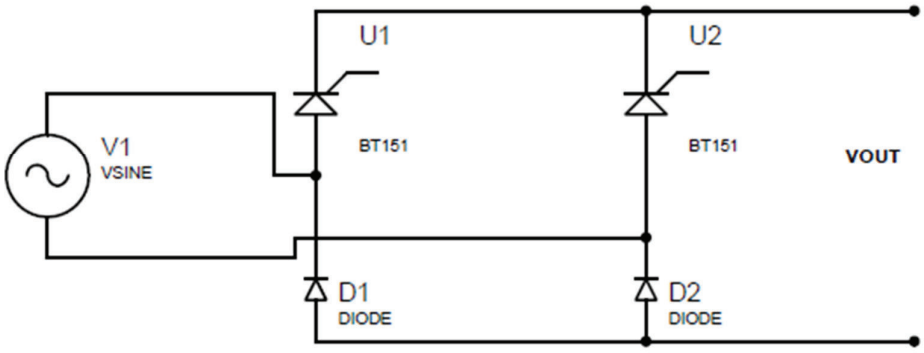

Figura 3.14. Circuito de Fuerza del conversor AC-DC semicontrolado.

(Fuente: Tipantocta, 2017)

3.4.2. Circuito de control

Este tipo de control permitirá regular el ángulo de fase entre cero y 180°, regulando así el voltaje de salida a la carga entre cero y el valor máximo en DC y que por las fórmulas anteriores se había calculado un valor de 108 V. Para poder controlar fuerza se necesita de tres etapas en el circuito de control, las cuales son:

- Circuito de sincronización o cruce por cero
- Microcontrolador
- Etapa de disparo

3.4.2.1. Circuito de sincronización o cruce por cero

Para la sincronización con la red, se usa un puente de diodos que rectifica la señal de corriente alterna e ingresa a un circuito serie entre una resistencia y el diodo interno de un opto acoplador, para entregar así una señal pulsatoria y sincronizada con la red.

Este circuito posee aislamiento por el uso del opto acoplador como muestra la figura 3.15.

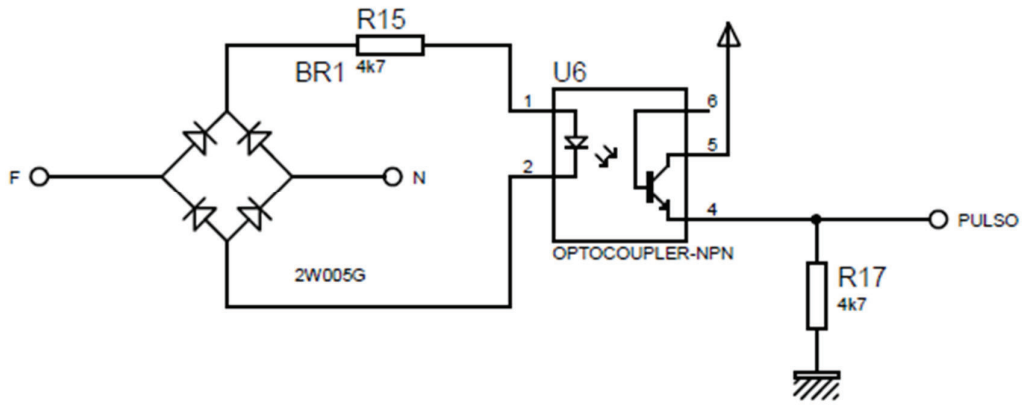

Figura 3.15. Circuito de sincronización con la red.

(Fuente: Tipantocta, 2017)

En la figura 3.16 se puede apreciar la forma de onda de salida del circuito.

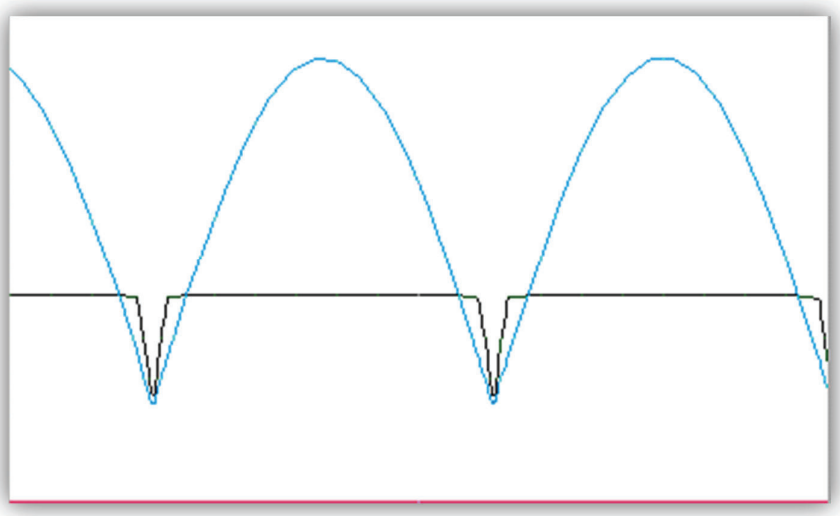

Figura 3.16. Onda de sincronización con la red.

(Fuente: Tipantocta, 2017)

3.4.2.2. Microcontrolador

Esta etapa se encuentra constituida por un microcontrolador de 8 bits de la marca AVR, específicamente el ATmega 16.

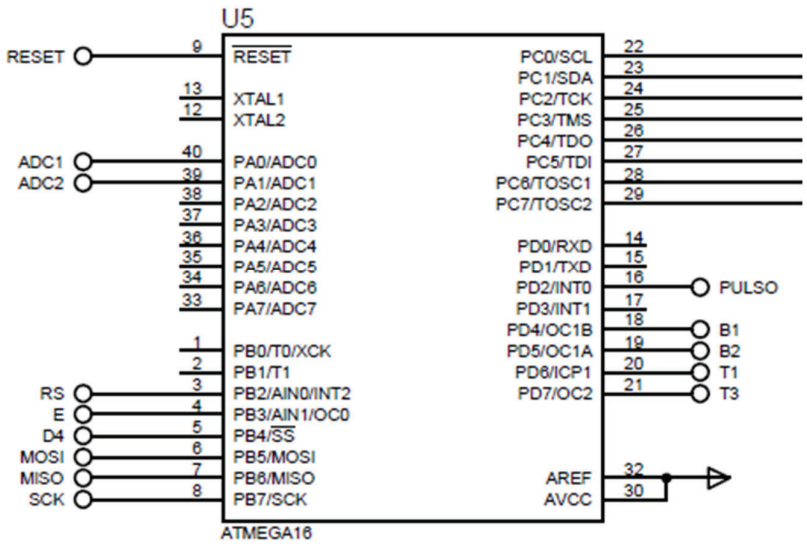

Figura 3.17. Microcontrolador ATmega16.

(Fuente:Tipantocta, 2017)

La señal sincronizada con la red ingresa por la interrupción del microcontrolador en el pin 16 y en base a esta entrada se genera la señal de control para realizar control de fase a la carga.

Se tiene un valor entre 0 y 5 voltios generados por la tarjeta PCI-1711U, el cual conecta a una entrada analógica del microcontrolador, para regular el ángulo de disparo del conversor entre 0 y 180°, y así obtener la regulación de carga.

3.4.2.3 Etapa de disparo

Para la etapa de disparo se usa un opto triac, el cual permite encender elementos de potencia, separando el circuito de fuerza con el circuito de control.

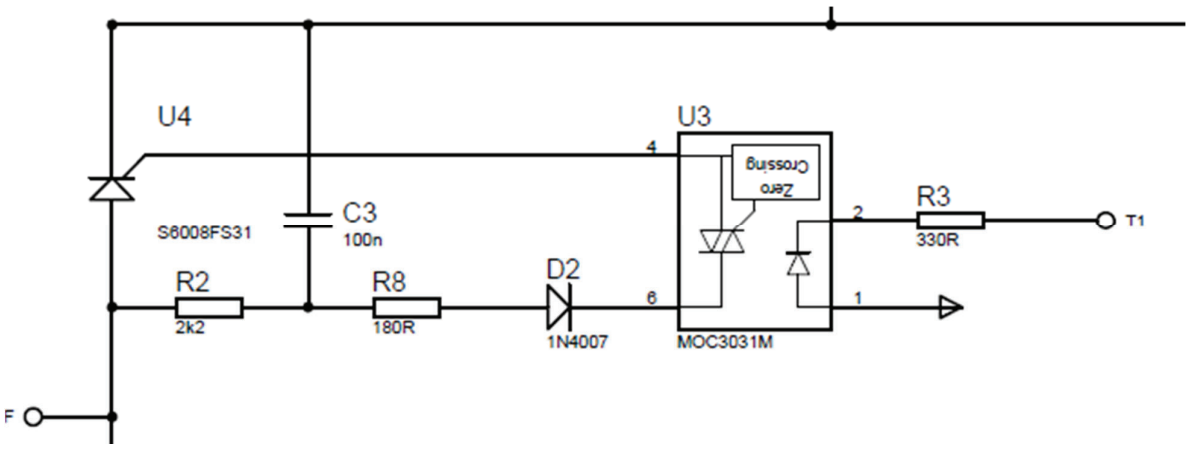

Figura 3.18. Circuito de disparo para los tiristores.

(Fuente: Tipantocta, 2017)

La señal de pulso de disparo generada por el microcontrolador es enviada al opto acoplador por medio de una resistencia, el esquema del circuito se puede visualizar en la figura 3.18.

3.4.2.4. Circuito completo

Todas las etapas del circuito pueden verse en la figura 3.19.

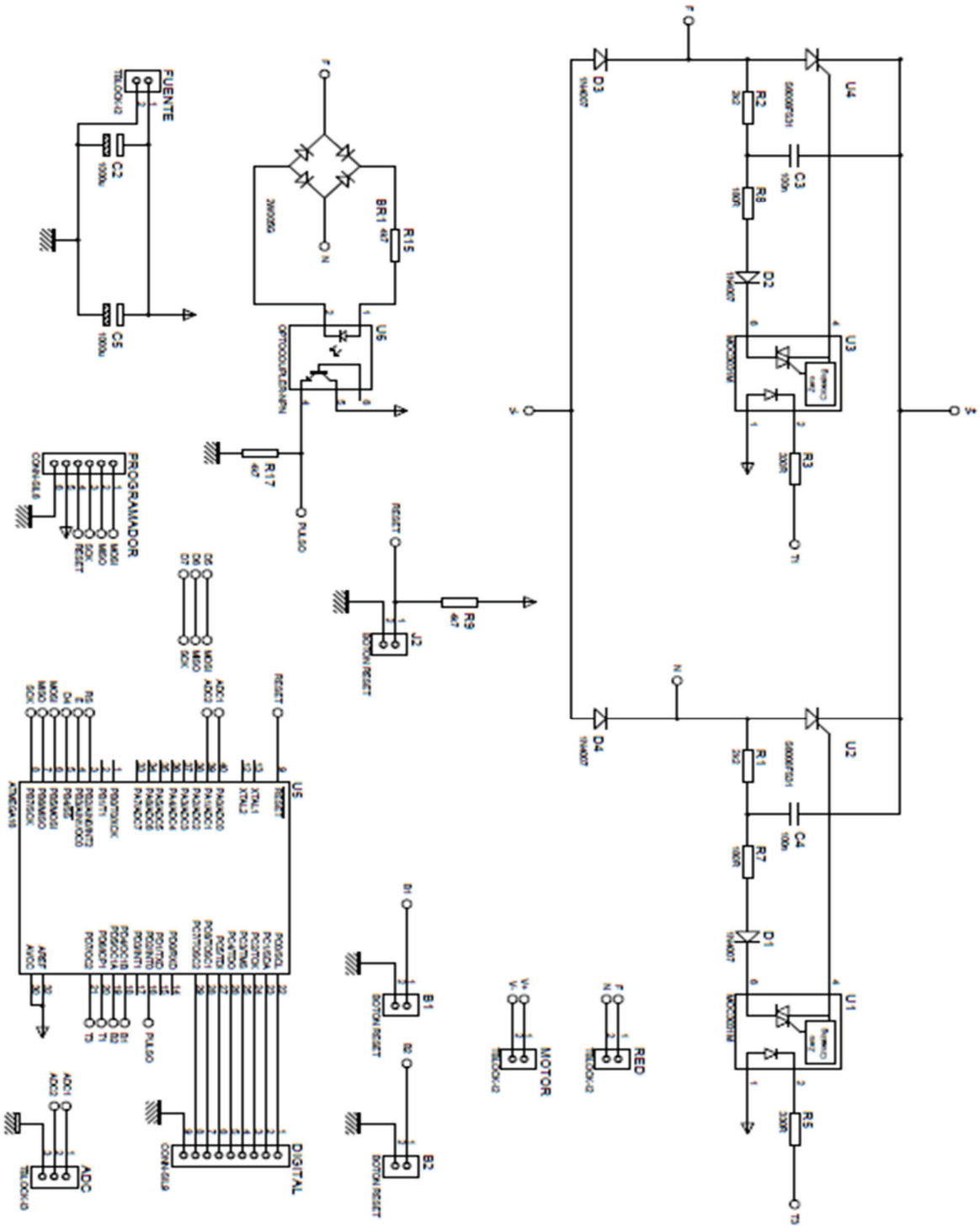

Figura 3.19 Circuito del Conversor AC-DC

(Fuente: Tipantocta, 2017)

3.5. Relación de transmisión

Para la relación de transmisión, se mencionó que se utilizará dos poleas acopladas a los motores, para lo cual se ha elegido dos del mismo diámetro por que siendo $\emptyset 1$ diámetro de la polea motriz y n1 su velocidad de giro, $\emptyset 2\ y\ n2$ son el diámetro y la velocidad de la polea conducida la ecuación de la relación de transmisión, es la relación que existe entre la velocidad de giro del árbol del motor y la velocidad del árbol resistente.

$$\delta = \frac{n2}{n1} = \frac{\emptyset 2}{\emptyset 1} \tag{3.20}$$

Para fines prácticos se ha seleccionado una polea de dos pulgadas tanto para el motor número uno como para el motor número dos, de igual forma por esta relación se trasmite la misma velocidad del motor que está regulando su voltaje de carga hacia el generador.

Por tal motivo la ecuación de la relación de transmisión es 1.

$$\delta = \frac{2"}{2"} = 1 \tag{3.21}$$

En la figura 3.20 se puede observar el sistema motor-generador acoplado por poleas correa trapezoidal.

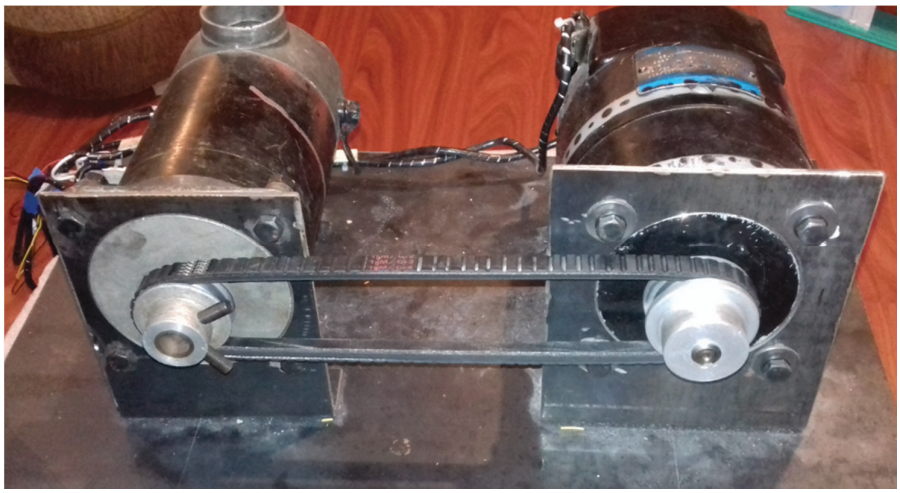

Figura 3.20. Sistema Motor Generador acoplado con poleas y banda trapezoidal.
(Fuente: Tipantocta, 2017)

CAPÍTULO 4

4. PRUEBAS Y RESULTADOS

4.1. Simulación del controlador

En el siguiente apartado se demuestra cómo el objetivo de usar el Middleware Orocos para realizar controladores se ha cumplido.

Luego de trabajar con el Middleware Orocos, se ha programado un sistema de control, basado en un controlador PID para un motor DC, en lazo cerrado y controlado por una señal de Set Point; este sistema se encuentra trabajando con soporte del sistema ROS montado sobre el sistema operativo Linux, y sobre todo con la herramienta Real Time Toolkit que funciona como librería sobre ROS, el cual le da la propiedad de trabajar en tiempo real para cualquier proceso que se quiera incorporar en este u otros sistemas de trabajo.

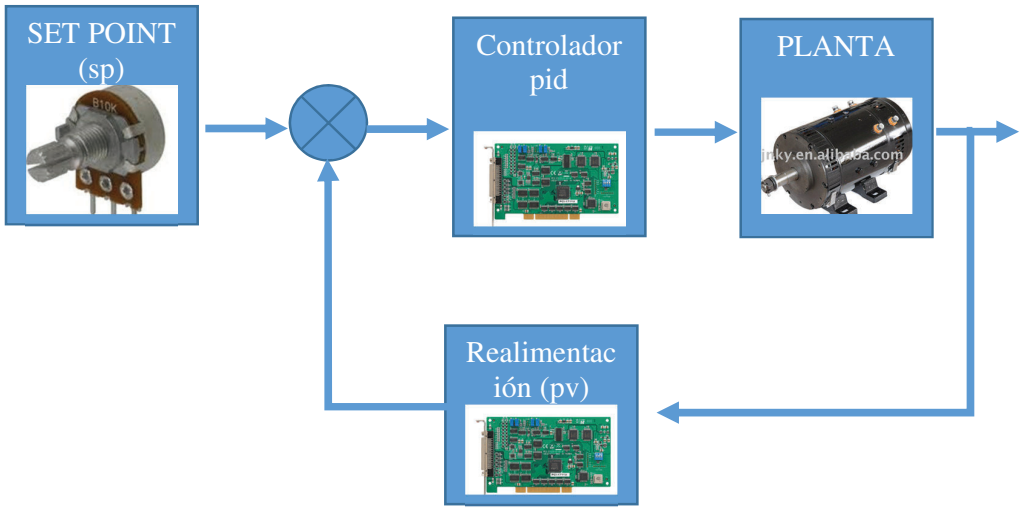

Figura 4.1. Sistema de Control de velocidad en Tiempo Real aplicado a un controlador PID

(Fuente: Tipantocta, 2017)

4.1.1. Arranque del deployer

El DEPLOYER es el entorno de funcionamiento o de ejecución de los programas en tiempo real, para ello es necesario escribir las instrucciones adecuadas para iniciarlo, este es el entorno en el cual funcionarán los componentes, la forma de activarlo es con la instrucción:

rosrun rtt_ros deployer /home/fabry/catkin_ws/src/script/pci.ops

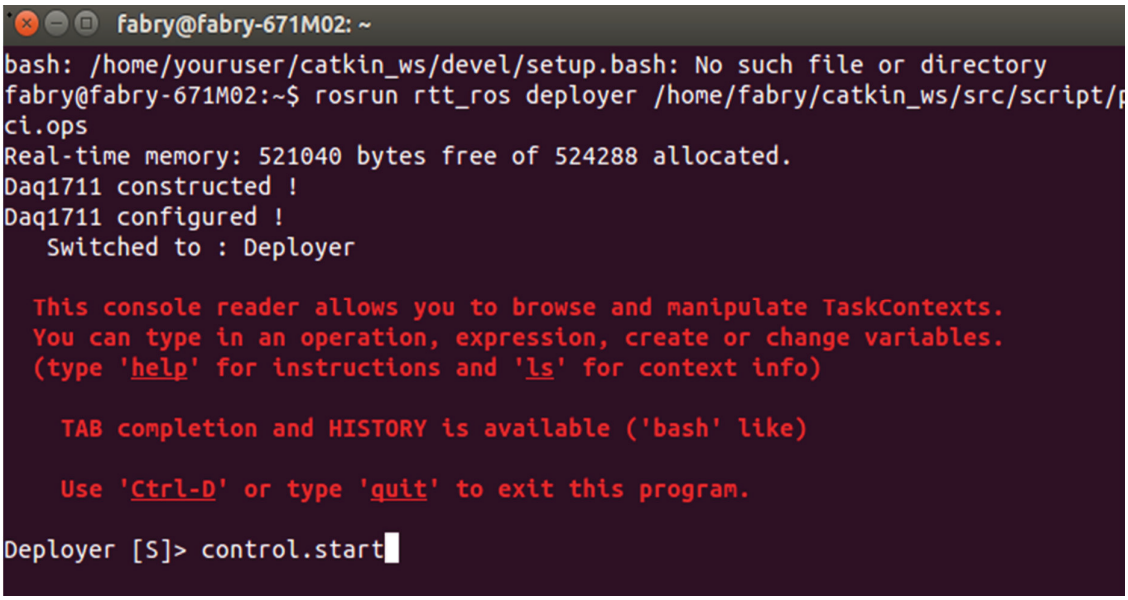

Figura 4.2. Arranque del deployer.
(Fuente: Tipantocta, 2017)

Para poder arrancar con el funcionamiento de un componente es recurrente abrir un terminal y ejecutar el **DEPLOYER,** para ello es necesario crear un script que no es más que una secuencia de inicio, grabado en un documento con extensión ".ops"; este elemento puede enviar atributos como parámetros de inicio de variables que el componente tenga para poder acceder al programa sin necesidad de compilar; en nuestro caso, para el controlador vamos enviar las constantes que manejan al PID como son "Kp, Ti, Td".

En la figura 4.3 muestra el script "pci.ops" con los parámetros de inicio.

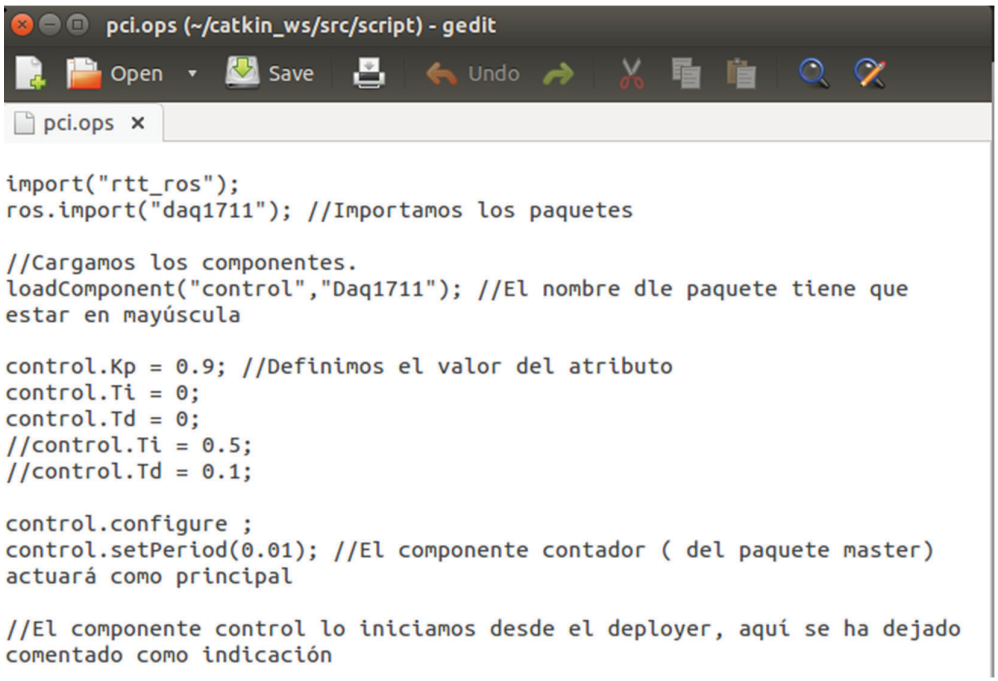

Figura 4.3. Script para arranque de componentes.

(Fuente: Tipantocta, 2017)

Como se muestra en script, se coloca las constantes del PID como atributos de clase del que llamaremos control que conectan directamente a los atributos del componente creado.

En el programa se incluyó la manera de crear una lista de datos, a la cual llamaremos fichero, que permite guardar la información adquirida tanto del Set Point como de la realimentación, el cual almacena los datos de revoluciones por minuto RPM para poder realizar las gráficas y así demostrar el funcionamiento del controlador digital en tiempo real. Por este método, el fichero tiene información que por medio de una tabla realizada en Excel presenta cada 0.01 segundos una muestra de datos y así se puede manejar la información.

Para el siguiente apartado se realizan pruebas de los controladores en tiempo real como son el Control Proporcional (P), Proporcional-Integral (PI) y Proporcional-Integral-Derivativo (PID), modificando las constantes Kp, Ti y Td.

4.2. Pruebas de funcionamiento de controladores

Para las pruebas se utilizó la señal del Potenciómetro que representa el Set Point que ingresa a un canal del conversor análogo-digital de la tarjeta de adquisición, la cual se encuentra escalada de 0 a 5 voltios a 0 y 1000 RPM; la manera de realimentar el sistema

se optó por el valor de voltaje que entrega el taco generador acoplado directamente al motor principal, este elemento entrega un valor de 0 a 1 voltio acoplada a otro canal de la tarjeta de adquisición y escalada con la velocidad de 0 a 1000RPM, en la Figura 4.4 se muestra la pantalla de visualización del controlador en tiempo real con los parámetros como el Set Point (sp), la realimentación (real) y mide los parámetros en RPM.

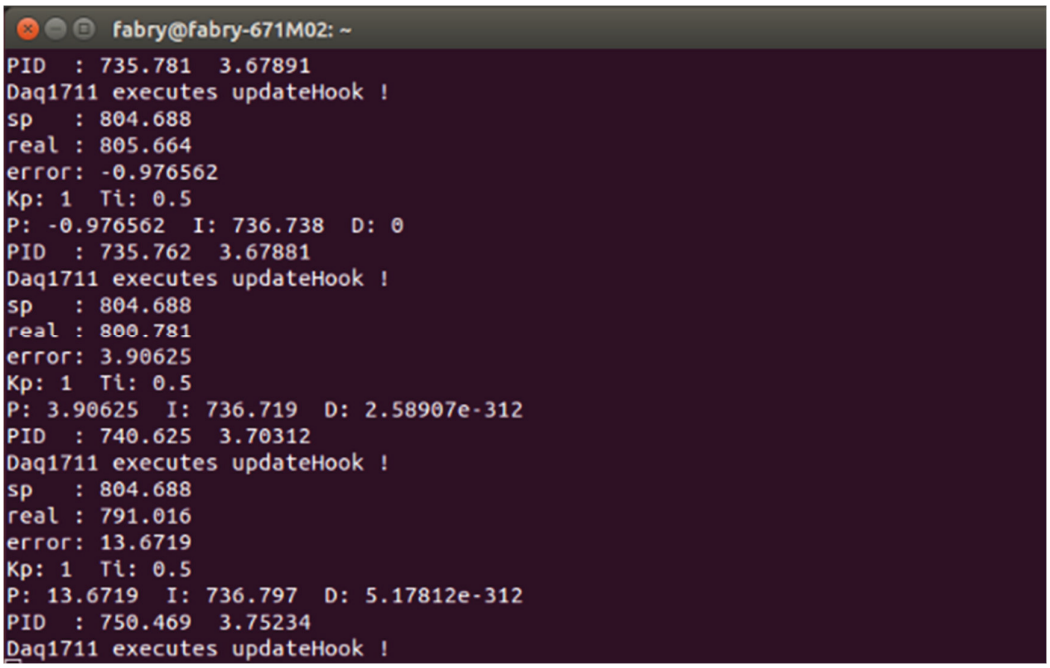

Figura 4.4. Pantalla de visualización de parámetros del Controlador en Tiempo Real.
(Fuente: Tipantocta, 2017)

4.2.1. Proporcional (P)

Por definición el control proporcional tiene error en estado estacionario, por lo tanto la señal de realimentación no iguala a la señal de Set Point. Para lo cual se toma como error a la resta del Set Point con la realimentación.

$$error = Sp - pv$$

$$P = Kp * error$$

Para todos los ejercicios se propone trabajar en un valor de Set Point de 800 RPM y ver cómo puede mejorar el resultado sintonizando las constantes, en este caso KP. Para Kp=1 se puede apreciar el error en estado estacionario, pero mantiene estabilidad esto se puede evidenciar en la Figura 4.5

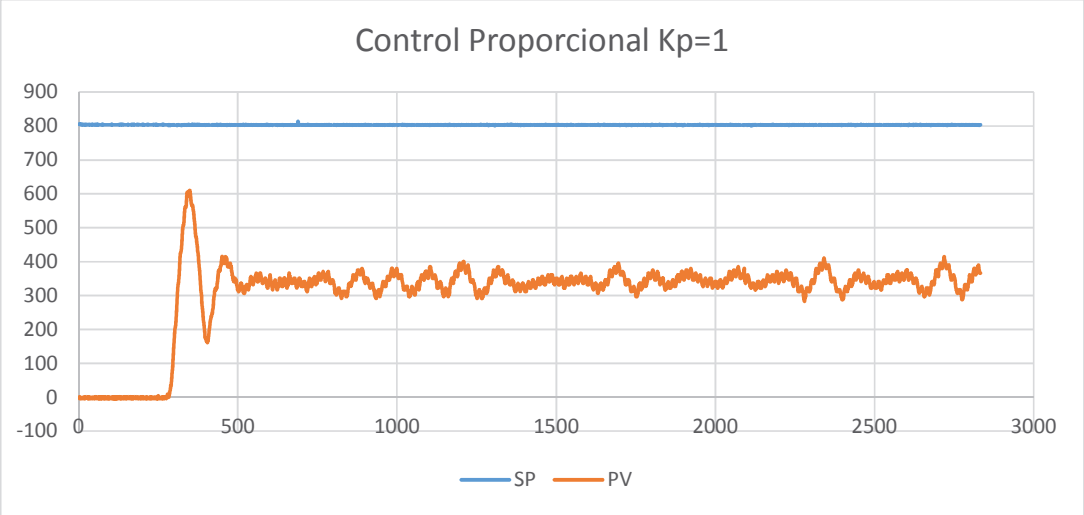

Figura 4.5. Pruebas con Control Proporcional con Kp=1.

(Fuente: Tipantocta, 2017)

Una segunda prueba puede ser para Kp=2 donde demuestra inestabilidad esto se puede evidenciar en la Figura 4.6

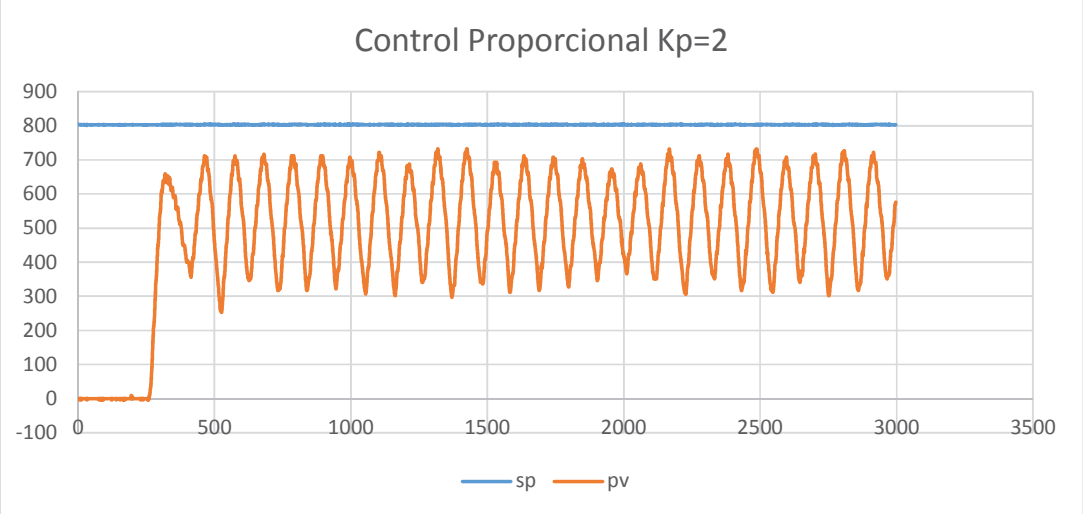

Figura 4.6. Pruebas con Control Proporcional con Kp=2.

(Fuente: Tipantocta, 2017)

Para Kp=0.5 demuestra estabilidad, pero se nota más aun el error en estado estacionario, esto se puede evidenciar en la Figura 4.7

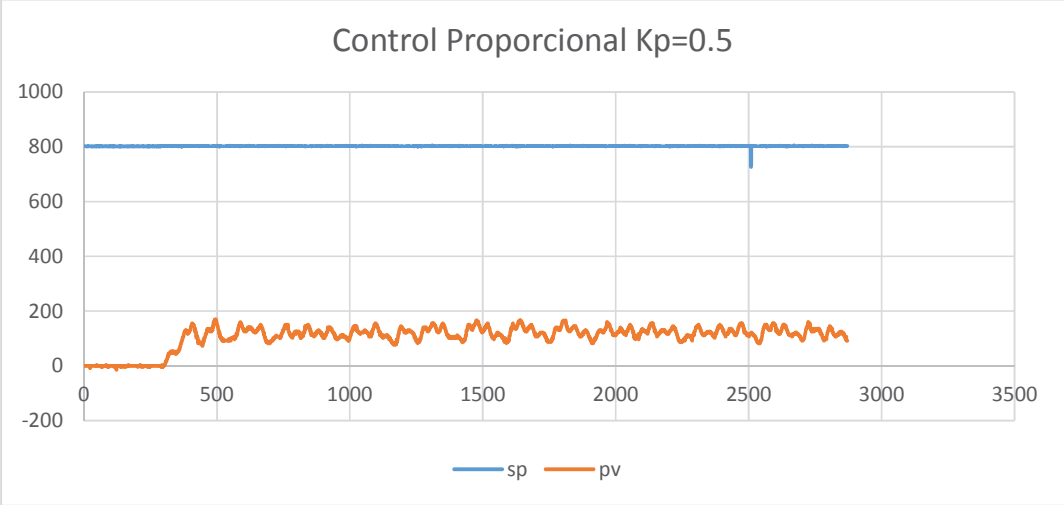

Figura 4.7. Pruebas con Control Proporcional con Kp=0.5.

(Fuente: Tipantocta, 2017)

Las pruebas demuestran que para el control proporcional la señal de realimentación (pv) no igualará al Set Point (sp).

4.2.2. Proporcional - integral (PI)

El efecto de combinar las acciones proporcional e integral, es un cambio en la salida del controlador sin desviación en el error. Al aplicar una señal de entrada tipo escalón al sistema de control producirá un valor de estado estacionario en el que no hay error.

Nuevamente en el script se toma en cuenta la sintonización de las constantes “Kp” y “Ti”; entregando los siguientes resultados mostrados en las siguientes figuras.

Para Kp=1 y Ti=0.5

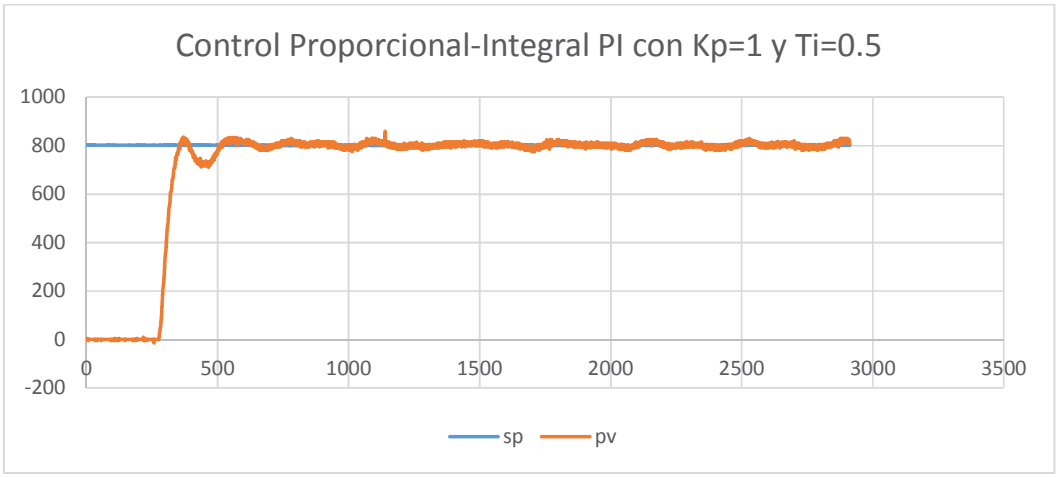

Figura 4.8. Pruebas con Control Proporcional-Integral con Kp=1 y Ti=0.5.

(Fuente: Tipantocta, 2017)

Generalmente la acción integral produce un sobrepaso considerable del error cuando se propone valores más altos antes de lograr finalmente su establecimiento.

Para Kp=1 y Ti=0.9

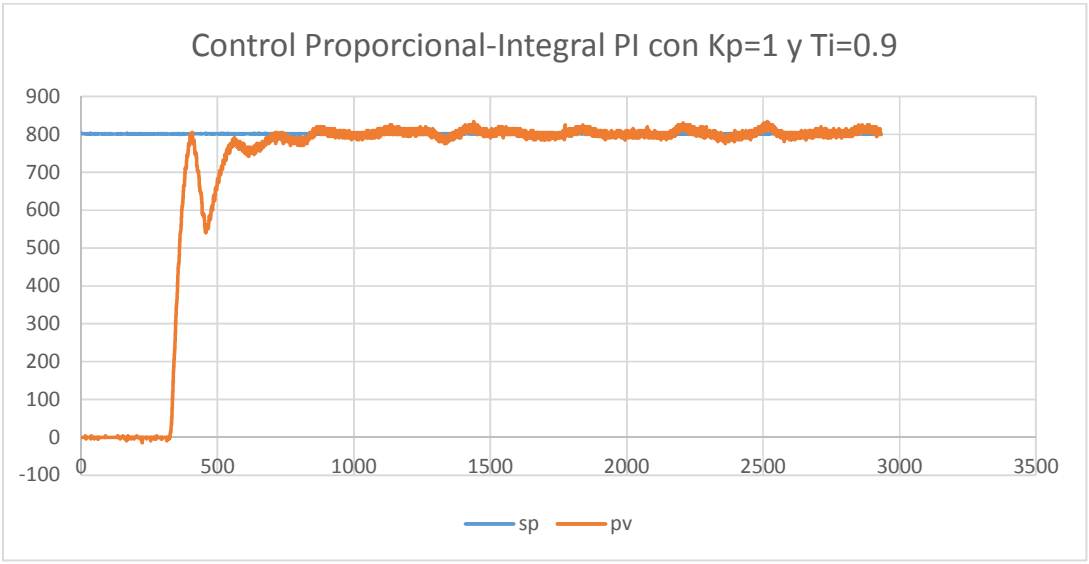

Figura 4.9. Pruebas con Control Proporcional-Integral con Kp=1 y Ti=0.9.

(Fuente: Tipantocta, 2017)

El sistema se vuelve lento al final llega la estabilidad por el efecto integral con valores más altos.

Para Kp=1 y Ti=1.5s

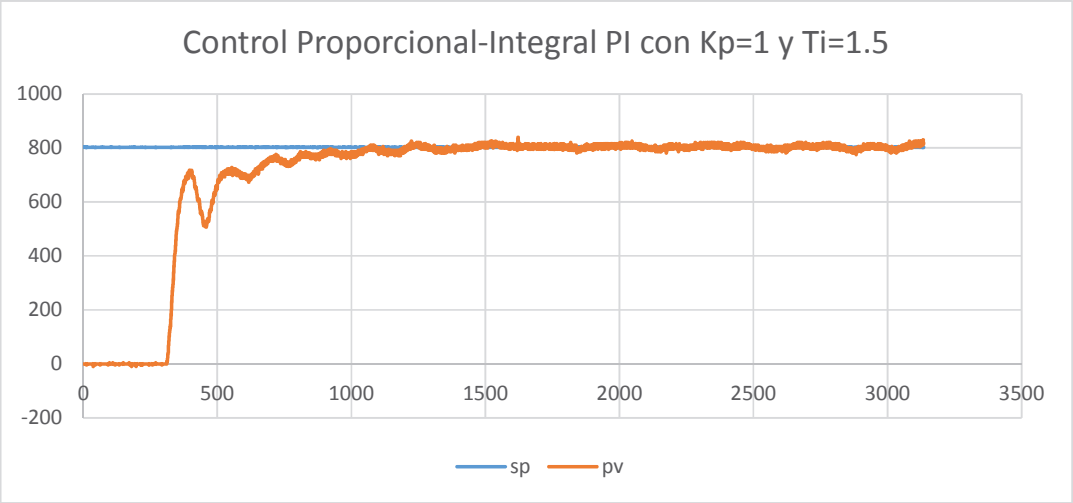

Figura 4.10. Pruebas con Control Proporcional-Integral con Kp=1 y Ti=1.5.
(Fuente: Tipantocta, 2017)

4.2.3 PROPORCIONAL-INTEGRAL-DERIVATIVO (PID)

Al combinar los tres modos de control (proporcional, derivativo e integral) se obtiene un controlador que no tiene desviación en el error y disminuye la tendencia a que se produzcan oscilaciones. Con el script se puede sintonizar y obtener diferentes respuestas, como muestra las siguientes figuras.

Con el valor derivativo bajo tiende a obtener una respuesta rápida, y se mantiene estable a cualquier cambio brusco.

Para Kp=1 ; Ti=0.5 ; Td=0.1

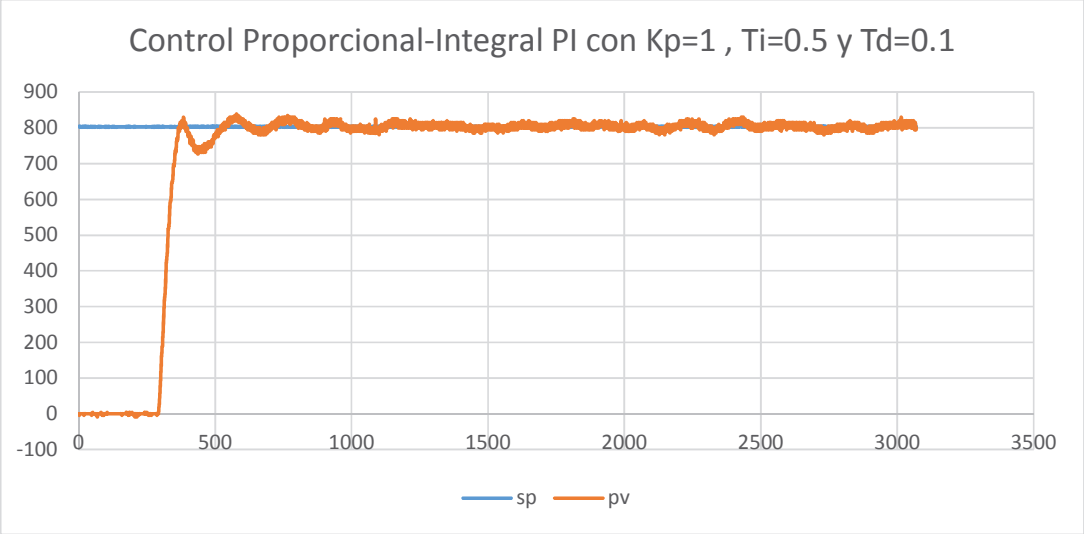

Figura 4.11. Pruebas con Control Proporcional-Integral-Derivativo con Kp=1,Ti=0.5 y Td=0.1.

(Fuente: Tipantocta, 2017)

Para Kp=1 ; Ti=0.5 ; Td=0.9

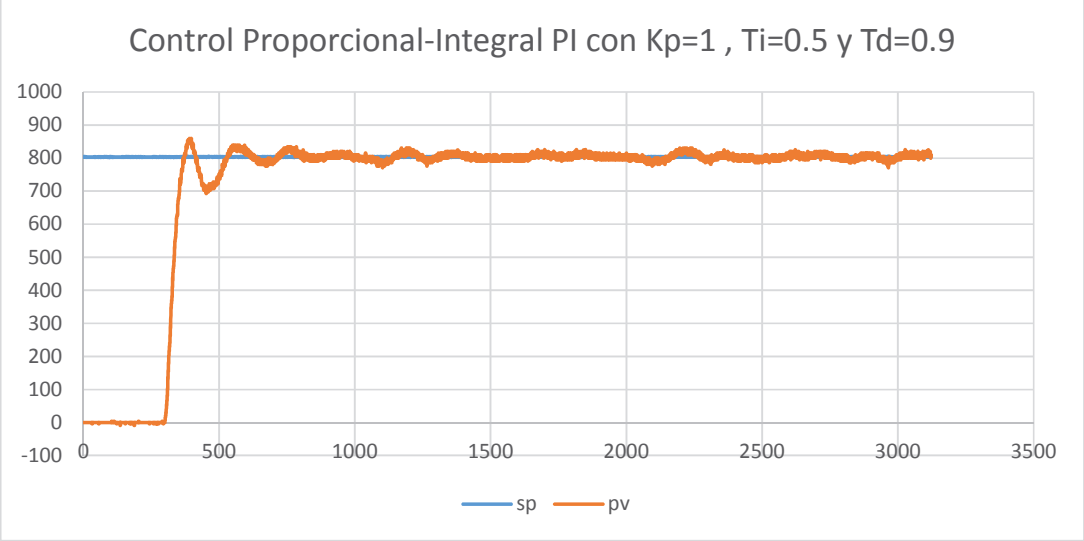

Figura 4.12. Pruebas con Control Proporcional-Integral-Derivativo con Kp=1,Ti=0.5 y Td=0.9.

(Fuente: Tipantocta, 2017)

Para Kp=1 ; Ti=0.5 ; Td=1.5

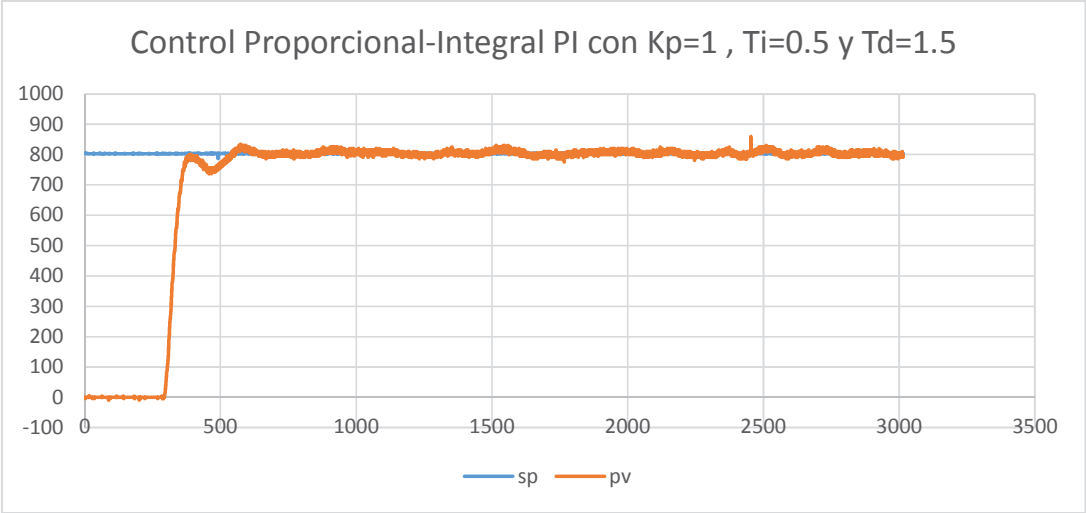

Figura 4.13. Pruebas con Control Proporcional-Integral-Derivativo con Kp=1,Ti=0.5 y Td=1.5.
(Fuente: Tipantocta, 2017)

Con esto queda demostrado, que el controlador PID discreto, obtiene buenos resultados a cambios bruscos de la señal de entrada, y sobre todo se demuestra que la implementación del sistema de controladores en tiempo real cumple su objetivo, que es el de mantener la variable de entrada controlada con respecto a la salida.

CAPÍTULO 5

5. CONCLUSIONES Y RECOMENDACIONES

Finalizada la implementación del sistema de control de velocidad, en base al Middleware Orocos se establecen conclusiones y recomendaciones, las cuales constituyen un aporte de la realización del proyecto.

5.1. Conclusiones

- Se ha implementado el controlador en tiempo real, diseñado en el Middleware Orocos en base a componentes, demostrando así como trabajar en programas de software libre, y el desarrollo en lenguaje de programación C++.

- El controlador PID diseñado por componentes, se conecta sin dificultad a la tarjeta de adquisición de datos, pues a un período de muestreo de 0.01 segundos, se obtiene una gran cantidad de datos que permitieron observar la veracidad del proyecto realizado.

- El sistema de control de velocidad implementado para la demostración del funcionamiento del controlador diseñado en tiempo real, responde a los diferentes parámetros programados como son el control: proporcional, proporcional-integral, proporcional-integral-derivativo, respondiendo de mejor manera en el sistema de control PID.

- La tarjeta de adquisición con bus PCI, es la mejor opción para trabajar a velocidades altas de procesamiento de información, ya que la computadora instalada tiene Linux Ubuntu para 32 bits, y la tasa de transferencia de datos es de 133 MB/s, con esta característica se da confiabilidad de transferencia de datos para que el Middleware Orocos procesando la información con rapidez y confiabilidad.

- Seleccionar un sistema de control de velocidad fue necesario, para demostrar el funcionamiento del controlador en tiempo real diseñado, para ello fue una buena elección el escoger un motor de corriente continua como planta y así realizar el modelamiento matemático y así realizar tanto la simulación en Matlab como la

implementación en Middleware Orocos, esto permitió saber que sólo por simulación el controlador funcionaría.

- La computadora seleccionada tiene un procesador Pentium cuatro, a la cual se le puede instalar Linux Ubuntu 14.04, y también permite la adaptación de la tarjeta de adquisición de datos con bus PCI.

5.2. Recomendaciones

- Este primer análisis del proyecto con controladores en tiempo real, apertura al trabajo con diferentes tipos de sensores y actuadores; se recomienda el estudio previo de los elementos mencionados y así implementarlos en el sistema del robot paralelo.

- Otro aspecto importante es la supervisión de datos, pues este primer trabajo permite la realización y el funcionamiento del controlador, mas no una interfaz que sea amigable con el usuario, y permita la visualización tanto de los parámetros ingresados al controlador como parámetros de salida.

- El proyecto desarrollado en base a componentes de software, falta por desarrollar, lo recomendable es seguir trabajando con más proyectos donde se pueda este tipo de programas y aprovechar todos los recursos y atributos que tiene el Middleware Orocos.

- Uno de los aspectos complicados que se tuvieron en el proyecto, fue la selección de la computadora, pues las tarjetas de adquisición de datos que se ha trabajado para el proyecto tiene un bus PCI, es decir permite la transferencia de información tanto de entrada como salida, pero en las computadoras actuales ha desaparecido y se tiene otro conector que es el bus PCI Express, que tiene una ranura más pequeña y no permite el funcionamiento de la tarjeta de adquisición de datos, por ello, en proyectos de otra universidad lo que hacen es adquirir computadoras bajo pedido, las cuales se puede ordenar características como el número de conectores de bus PCI.

- Otra característica recomendable es trabajar con computadoras de última generación, con procesadores I7, de alta capacidad en RAM y alta capacidad del

disco duro, pues la computadora que se utilizó, apenas funciona el sistema operativo Linux Ubuntu.

- Es primordial probar los controladores en un sistema físico, donde se pueda demostrar el 100% de la utilidad de trabajar con Middleware Orocos.

- Se recomienda continuar con el trabajo de investigación con algoritmos más robustos y elaborados en C++, y así mejorar los tipos de control.

- Desarrollar el hardware del robot paralelo y realizar el estudio de instrumentación requerida con el tipo de sensores que se deberá utilizar para el ingreso y salida de datos en tarjetas PCI óptimas de trabajo.

BIBLIOGRAFÍA

Advantech. (2010). *PCI-1711U User manual.* USA: ADVANTECH.

Bolton, W. (2014). *Mecatrónica - sistemas de control electrónico en la ingeniería mecánica y eléctrica.* Mexico: Alfaomega.

Casalilla, J. (2012). *Implementación basada en el middleware orocos de controladores dinámicos para un robot paralelo.* Valencia: UPV.

Cova, W. (2008). *Sistemas de control aplicado.* Argentina: Universidad Tecnológica Nacional.

Martinez, A., & Fernández, E. (2013). *Learning ROS for Robotics Programming.* BIRMINGHAM - MUMBAI: PACKT.

Ogata, K. (2010). *INGENIERÍA DE CONTROL MODERNA.* Madrid: PEARSON EDUCACIÓN.

Pal robotic, Lauven, & FMTC. (2011). Recuperado el 06 de 11 de 2016, de The orocos project: http://www.orocos.org/content/history

Pal_robotic, Lauven, & FMTC. (2011). Recuperado el 06 de 11 de 2016, de The orocos project: http://www.orocos.org/kdl

Pal_robotic, Lauven, & FMTC. (2011). Recuperado el 6 de 11 de 2016, de The orocos project: http://www.orocos.org/rtt

Pal_robotic, Lauven, & FMTC. (2011). Recuperado el 6 de 11 de 2016, de The orocos project: http://www.orocos.org/bfl

Pal_robotic, Lauven, & FMTC. (2011). Obtenido de The Orocos Project: http://www.orocos.org/content/history

Quigley, M., Gerkey, B., & Smart, W. D. (2016). *Programming Robots with ROS.* United States of America: O'REILLY.

THE OROCOS PROJECT. (2011). *Components for Control*. Recuperado el 6 de 11 de 2016, de The orocos project : http://www.orocos.org/ocl

Villarroel, J. (2015). *SISTEMAS DE TIEMPO REAL.* Zaragoza: Universidad de Zaragoza.

WIKIPEDIA. (29 de 10 de 2016). *Middleware*. Recuperado el 6 de 11 de 2016, de https://es.wikipedia.org/wiki/Middleware

ANEXOS

Anexo 1. Programa del componte del controlador en tiempo real.

```
//*************************DAC1711-COMPONENT.CPP*****************************************
//*****************************************************************************************************

#include "daq1711-component.hpp"
#include <rtt/Component.hpp>
#include <iostream>

Daq1711::Daq1711(std::string const& name) : TaskContext(name){
  this->addAttribute("Kp",Kp);
  this->addAttribute("Ti",Ti);
  std::cout << "Daq1711 constructed !" <<std::endl;
}

bool Daq1711::configureHook(){

        P=0;
        I=0;
        D=0;
        PID=0;
        T=0.1;
        //Kp=1;
        //Ti=0.5;
        q_isReady=false;
        setPoint_isReady=false;

        error = 0;
        error_0 = 0;
        limsup=1000;
        liminf=0;

ret=Success;
pci1711In = AdxInstantAiCtrlCreate();
pci1711Out= AdxInstantAoCtrlCreate();
DeviceInformation devInfo(deviceDescription);
ret = pci1711In->setSelectedDevice(devInfo);
CHK_RESULT(ret);
ret = pci1711Out->setSelectedDevice(devInfo);
CHK_RESULT(ret);
volts=0.0;
ret=pci1711In->Read(0,3,&volts);
CHK_RESULT(ret);
ret = pci1711Out->Write(0,1,&volts );
CHK_RESULT(ret);

  std::cout << "Daq1711 configured !" <<std::endl;
  return true;
}
```

```
bool Daq1711::startHook(){

  std::cout << "Daq1711 started !" <<std::endl;
  return true;
}

void Daq1711::updateHook(){

 ret = pci1711In->Read(0,3 ,&volts);
 ret = pci1711In->Read(2,3 ,&volts2);
 y=volts*1000/5;
 r=volts2*1000;
//r=500;
 error=y-r;
 P=error*Kp;
 if(Ti==0) I=0;
 else I = I_0 + (Kp/Ti)*error_0*T;
 PID=P+I;

 /*AntiWindUp*/
                    if(PID>limsup)
                    {
                              PID=limsup;
                              I=I_0;
                    }
                    else if(PID<liminf)
                    {
                              PID=liminf;
                              I=I_0;
                    }

                    volts=PID*5/1000;
               ret = pci1711Out->Write(0,1,&volts );

                    error_0 = error;
                    I_0 = I;


std::cout<<"sp   : "<<y<<std::endl;
std::cout<<"real : "<<r<<std::endl;
std::cout<<"error: "<<error<<std::endl;
std::cout<<"Kp: "<<Kp<<"  Ti: "<<Ti<<std::endl;
std::cout<<"P: "<<P<<"  I: "<<I<<"  D: "<<D<<std::endl;
std::cout<<"PID  : "<<PID<<"  "<< volts <<std::endl;

  std::cout << "Daq1711 executes updateHook !" <<std::endl;
}

void Daq1711::stopHook() {
  std::cout << "Daq1711 executes stopping !" <<std::endl;
}
```

```
void Daq1711::cleanupHook() {
  std::cout << "Daq1711 cleaning up !" <<std::endl;
}

/*
 * Using this macro, only one component may live
 * in one library *and* you may *not* link this library
 * with another component library. Use
 * ORO_CREATE_COMPONENT_TYPE()
 * ORO_LIST_COMPONENT_TYPE(Daq1711)
 * In case you want to link with another library that
 * already contains components.
 *
 * If you have put your component class
 * in a namespace, don't forget to add it here too:
 */
ORO_CREATE_COMPONENT(Daq1711)
```

//********DAC1711-COMPONENT.CPP**********

```
//*************************************************************************************************
#ifndef OROCOS_DAQ1711_COMPONENT_HPP
#define OROCOS_DAQ1711_COMPONENT_HPP


#include <rtt/RTT.hpp>
#include <fstream>
#include <sstream>
#include <stdlib.h>
#include "../include/compatibility.h"
#include "bdaqctrl.h"
using namespace RTT; //Usamos este namspace para no tener que poner RTT en todas
las funciones de orocos
using namespace std;
using namespace Automation::BDaq;
#define      deviceDescription  L"PCI-1711, BID#1"
class Daq1711 : public RTT::TaskContext{
  public:
    Daq1711(std::string const& name);
    bool configureHook();
    bool startHook();
    void updateHook();
    void stopHook();
    void cleanupHook();
```

```
protected:
 ErrorCode        ret;
 InstantAiCtrl * pci1711In;
 InstantAoCtrl * pci1711Out;
 double volts,y,r,e;
 double volts2;
 bool q_isReady, setPoint_isReady;
 double  Td, limsup, liminf, T, PID;
 double q, setPoint, error,  P, D;

private:
 double Kp,I, I_0,Ti,error_0;
};
#endif
```