IMPLEMENTACIÓN DE UN SISTEMA TELEOPERADO CON REALIMENTACIÓN VISUAL PARA EVASIÓN DE OBSTÁCULOS DE UN ROBOT MÓVIL

Luis Miguel Betancourt

Nelsón Botón Gómez

UNIVERSIDAD MILITAR NUEVA GRANADA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA EN MECATRÓNICA
BOGOTÁ D.C.
AGOSTO 2009

IMPLEMENTACIÓN DE UN SISTEMA TELEOPERADO CON REALIMENTACIÓN VISUAL PARA EVASIÓN DE OBSTÁCULOS DE UN ROBOT MÓVIL

Luis Miguel Betancourt
betancourtluismiguel@gmail.com
1800512

Nelsón Botón Gómez
NelsonBoton@gmail.com
1800513

Trabajo de grado para optar al título de
INGENIERO EN MECATRÓNICA
Trabajo de grado

DIRIGIDO POR:
Ing. NELSON VELASCO TOLEDO
nelson.velasco@unimilitar.edu.co

UNIVERSIDAD MILITAR NUEVA GRANADA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA EN MECATRÓNICA
BOGOTÁ D.C.
AGOSTO 2009

Dedico este proyecto, y toda mi carreara universitaria, a Dios y mi familia, que en cada momento, lucharon conmigo hombro a hombro, por conseguir el éxito. También dedico esto a todos, aquellos que creen en la formación académica integral, que quieren y respetan la ingeniería mecatronica, y luchan cada día por mejorar la calidad de esta disciplina.

-Nelsón Botón

El auténtico problema, es que la gente cree que nacimos para vivir en un cuerpo y no en un Alma. Este trabajo lo dedicó a todos aquellos que hacen de su vida la solución y entienden el verdadero sacrificio, necesario para convertirla en realidad.

-Luis Miguel

AGRADECIMIENTOS

Aquí habla botón. Son muchas las personas, sin las cuales, este trabajo de grado, no hubiera sido posible, nombrare las mas significativas, sin desmeritar la valiosa ayuda, de los que se me olvide nombrar. Primero a Dios, ya que es el autor intelectual de todo lo que en nuestra vida ocurre, a mi familia, y su apoyo constante en cada una de las etapas de mi carrera, a mi compañero de tesis Luis Miguel Betancourt, con quien se logro una pareja de trabajo ideal para la realización de este proyecto, al director de Tesis, Nelson Velasco Toledo, ya que con sus sabiduría y su apoyo, nos supo guiar en los momentos de duda, y a los grupos de investigación de la universidad, por la incalculable ayuda prestada en la realización de la tesis.

Nelsón Botón.

Creo, la palabra Gracias se me escapó de tanto sentirla. Huyó para vivir en todos aquellos que me permitieron estar aquí, que me dieron parte de sus enseñanzas y consejos. ¿Cómo mencionar lo que cada uno de ustedes me regalo? Tal vez, me sea más fácil contar todas las estrellas. Sin embargo procuraré, dibujar entre líneas, el verdadero sentir de tan inmensa bondad.

Primero, gracias a Dios. No solamente por estar vivo sin que mi Alma muriera lejos de estar preparada; sino por ser tan misericordioso con este simple y pecador mortal. Tal vez, si durante cada segundo de cada día, le diera gracias de corazón a Dios, por el tiempo que tarda un pájaro carpintero en desbastar con su pico el monte Everest hasta llegar al nivel del mar, entonces expresaría la verdadera gratitud que El se merece. Gracias Dios mío por esa casi eterna soledad humana, por darme el Alma que tanto le cuesta comprender su pertenencia a este lugar, por los dones, por mi familia y amigos, por la salud y especialmente por nunca abandonarme; aunque yo lo hiciera tantas veces. Ésta Alma es tuya.

A la virgen María, gracias Madre Santa, por cuidar de esta Alma escribiendo tú nombre en mi corazón. Gracias por interceder por nosotros ante Dios y aún más, por el inmenso Amor con el que ferviente, tu corazón nos llama y protege. Confío en tus palabras a mi Madre: “Todo va a estar bien, no te preocupes” para llenar de fe mi ser. Gracias, eternamente gracias.

A mi Ángel de la guarda, porque aunque he ahumado sus alas en el fuego de los réprobos, al meterme a mí mismo en tantas situaciones que nunca debí ni intentar; El siempre ha estado firme. Gracias por cuidarme y escribir mis actos en Aquel libro. Gracias por las palabras silentes, disfrazadas de arcos en blanco y leve. Gracias por tú guía, voluntad celestial.

A mi Madre, porque la palabra Amor se escribe con el eco de tú nombre entre tus abrazos. Gracias

por hacer de tú vida, el ejemplo de las enseñanzas que nos diste con tanto cariño. Por el inmenso Amor con el que nos educaste, por tus consejos llenos de sabiduría, por estar SIEMPRE ahí para darnos un abrazo, no importa si era en la madrugada ó al comienzo de la noche. Gracias por cuidarnos y educarnos para ser hermanos, no enemigos; como tanto se ve hoy en día. Gracias porque te preocupaste más por cuidar nuestras Almas, que lo demás.

A mi Padre, porque aunque le dijera que tendría que ir a recogerme al purgatorio; lo haría, y al llegar me diría: "Si ves hijo, siempre a tiempo". Por enseñarme a jugar ajedrez y hacerme ver, que la vida es una larga partida contra nosotros mismos: el blanco y el negro, piezas de un mismo tablero. Por enseñarme el Amor a los libros, ¡qué gran tesoro descubrí entre las hojas de sublimes líneas, mientras dibujaba mundos de fantasía en mi imaginación! Espero algún día, si Dios me permite tener hijos, poderles enseñar ese Amor tan grande.

A mi Hermano Fabio, por su protección y cuidados. Por la música que tantos recuerdos me permitió guardar. Por los consejos que a palabras simples no hubiera entendido. Por todas las noches que cazábamos juntos, entre libros de Agatha y música clásica. Siempre serás Poirot para mí. No importa que tenga 50 años, siempre me gustará el: "¿. . . madre y los muchachos?". Gracias por todos los detalles y presentes, pero especialmente por el Amor tan grande que me diste. Con él Alma, el Capitán Hastings.

A mi Hermano Diego, ¿qué tesoro más grande que el de tener un hermano gemelo? Nunca estuve solo; siempre que salía a jugar, te tenía a ti. Todas las cosas las compartíamos juntos. Las películas, los juegos, las tardes dibujando, los juegos de Rol, los torneos de Magic, todas las series que veíamos, las noches programando el motor de render, las noches cazando ejercicios de algebra y física . . . oh, aquellas épocas fueron tan maravillosas . . . tantos fueron los grandes momentos, que necesitaría un cofre del tamaño del Sol para guardar la alegría de siquiera un solo momento a tú lado. Sé que pronto nos separaremos para cada uno continuar con su vida y sentiré un vació tan grande . . . pero como le ocurrió a Ulic Qel-Droma antes de morir . . . la fuerza siempre estará contigo y nunca os abandonará. A ustedes dos mis hermanos, gracias por ser el Virgilio que éste Dante necesito para atravesar este largo camino. Gracias, eternas Gracias.

A Numa, gracias a Dios y a ti, por el inmenso Amor; contigo se fue un pedazo de mi corazón que jamás llenaré. Fueron más de 12 años de risas y grandes momentos. Gracias por la protección y por dar hasta el último momento, esa alegría que naciente despertaba cada mañana a mi lado. Espero verte al otro lado.

A mi abuelo Fabio, porque aunque nunca te conocí, la fama de tus actos aquí en la tierra, inspiraron en mí tanto respeto y admiración. Espero llegar al cielo y mostrarte lo bien que tu hijo me enseño a jugar ajedrez; me sentaré a tu lado y me contarás todas esas cosas que los abuelos suelen contarle a sus nietos, entre risas y abrazos. A todos mis familiares: mis abuelos, tíos y primos, gracias por el apoyo, la ayuda y el especial cariño. Muchas gracias.

Al Padre Edgar, por recordarme que: El verdadero sentido de esta vida es aprender a Amar, pero sólo hay algo que merece la pena Amar. Gracias por salvar mi Alma y explicarnos tantas cosas que estaban más allá de nuestra comprensión.

A mi compañero de Tesis, Nelson Botón que conozco desde hace 15 años. Gracias, porque si no

hubiera sido por su ayuda, estas palabras no estarán escritas en este documento. Porque, las muchas veces que quise desistir de este proyecto, él estuvo ahí para darme animo. De todas las personas que conocí en la universidad, no me hubiera embarcado en este proyecto con otra. Gracias muy especiales también, a toda su familia que me recibió en su casa; muchas gracias.

A Jhon Cardona, porque su amistad ha llenado de alegría este hogar; por los muchos días de juego, por los mundos conocidos y por los que él cartógrafo nunca paso; también por las alarmas y las infinitas risas que tanto acompañaron los momentos. Gracias buen hombre por todo.

A David y el Mono y todos sus familiares, gracia por todas las noches de juegos y risas, creo que lo que hacíamos era magia; todos esos buenos momentos, definieron la felicidad en mí. Ya el tiempo presente no es como el de antes, los miedos y la realidad se llevaron aquella infancia que parecía eterna. Aun en mí, quedan esos muchos recuerdos que nunca se irán. Cuídense y gracias, muchas gracias.

A Ernesto, mi profesor de matemáticas en el bachillerato. Gracias por ser exactamente como fue, gracias por enseñarme lo hermoso y maravilloso que puede ser el mundo de los números y las ecuaciones. Si me fue tan bien en la Universidad, fue gracias a vuestra ayuda. También un agradecimiento muy especial a todos mis profesores del colegio y Universidad, aprendí muchas cosas de la vida y el conocimiento junto a ustedes.

A mis amigos y compañeros de Universidad, a Vivi, porque más allá de ser una gran amiga, aprendí muchas cosas y salvó en gran parte esta Alma. A Diego Quintero, Erwin, Ruben, Daniel, Camilo y demás compañeros, gracias por la inmensa ayuda. A Juan Camilo un especial agradecimiento por los buenos momentos y permitir que nuestro proyecto se realice junto al de él.

A los que se me robaron la sonrisa del rostro a fuerza de mentiras y engaños; a ustedes, por muy irónico que suene, también gracias; porque me recordaron que lo único seguro en la vida, es que nada es seguro. Dios se apiade de ustedes.

Este también es un adiós, a mi infancia y toda mi vida sin responsabilidades aparte del estudio. Este es el punto donde el camino se divide, de un pasado en el que era un simple joven a un futuro en el que comenzaré realmente la vida. Este es el punto donde me convierto en adulto, dejando atrás los miedos conocidos para adquirir nuevos. A todos los que llevaron el Alma al aire para dejarme respirar y no mencioné aquí, os agradezco con todo mi ser.

Antes de morir, un naufrago, escribió: “Dale gracias a Dios por todos los que te dan Amor, porque es el único tesoro que te llevas al partir”. A todos ustedes, además de mi Alma, gracias por ser lo único que tengo.

Luis Miguel.

CONTENIDO

LISTA DE FIGURAS

Resumen del Trabajo

En el desarrollo de la vida humana, el hombre se ha visto expuesto a realizar diversas tareas para subsistir en la naturaleza. Muchas de estas, ponen en riesgo la integridad fsica del sujeto que se encuentra realizando dicho trabajo. Labores como la actividad militar, exploración en medios contaminados, investigación con químicos, entre otras; han sido un problema constante, ya que impedimentos, como la morfo-fisiologa humana o la vulnerabilidad a elementos peligrosos que se encuentran en el entorno, son un riesgo para la supervivencia del mismo. En la constante búsqueda de una posible solución al problema anteriormente mencionado, el hombre ha ido desarrollando herramientas útiles para realizar satisfactoriamente dichas tareas; uno de los métodos más usados en la actualidad es el empleo de la robótica móvil teleoperada. El uso de ésta, ofrece la posibilidad de operar distintos mecanismos a distancia, para que el usuario no interactué directamente en el proceso.

El presente proyecto consiste en la implementación de un sistema teleoperado con realimentación visual. Para esto se usa un sistema embebido (mini2440) colocado en un robot móvil con el propósito de controlarlo desde un computador distante, que se comunica con el sistema (mini2440-robot móvil) por medio del protocolo TCP. Junto al sistema embebido se adaptó una cámara USB que captura las imágenes, las procesa y luego, el sistema embebido las envía al sistema remoto por medio del protocolo TCP en el mismo canal. Allí, el usuario recibirá la imagen enviada por el sistema y será capaz de controlar el movimiento del robot al enviar señales de control por medio del protocolo TCP. El sistema Embebido, recibe dichas señales y se comunica con el robot por medio del puerto serial (RS232).

En la implementación se obtiene como resultado un sistema adaptable a cualquier plataforma móvil remota, siendo capaz de controlar su dirección. También logra obtener una realimentación visual, con el suficiente tamaño para que el usuario pueda monitorear el entorno del robot, con una aceptable velocidad (2.5 Frames Por Segundo), y la suficiente calidad para identificar objetos de la imagen. El video se obtiene con un retraso de aproximadamente 1.5 segundos, debido a la imposibilidad por parte de la librería grafica utilizada en el proyecto, para transmitir datos de mayor tamaño a 1300 bytes.

El presente documento esta dividido en cinco partes. En la primera parte, está la introducción, donde se exponen los objetivos, antescedentes y justificación. En la segunda parte, aparecen los fundamentos básicos. Estos son los conceptos principales que proveerán al lector de un marco de referencia para el entendimiento de este proyecto. En la tercera parte se expone el desarrollo de la metodología, donde se explica cómo se implementó el sistema teleoperado y la relación con la

plataforma robótica móvil. La cuarta parte es el conjunto de pruebas de desempeño del sistema. También aparecen las conclusiones, donde se comentan las ideas finales de la realización de este trabajo. Finalmente, aparecen los anexos que contienen la información básica de los manuales de usuario y el código fuente.

1. INTRODUCCIÓN

1.1. Planteamiento del problema

Existen diversas situaciones en la vida diaria que comprometen la integridad y salud humanas; como por ejemplo, incendios, derrumbes, reparaciones en plataformas submarinas, terrenos minados, entre otras. Debido a esto, la creación de este proyecto pretende maximizar la protección humana, al permitir su participación en dichas situaciones a través del uso de un sistema tele-operado; asegurando de ésta manera su bienestar y protección.

Un sistema tele-operado es aquel sistema controlado por un usuario a distancia desde una estación remota; permitiendo darle ordenes al mismo en cuento a su movimiento, y que trayectorias se deberían utilizar para completar su objetivo. Basándose en este concepto, se decide desarrollar un sistema tele-operado con realimentación visual; donde la cámara capturará la imagen que después será procesada para enviarse al sistema remoto, a través del sistema embebido. Éste desarrollo le dará al móvil la posibilidad de desplazarse por diferentes ambientes y evadir obstáculos. La cuestión básicamente radicaría en ¿cómo se implementaría un sistema tele-operado con realimentación visual en el robot móvil para evasión de obstáculos?

1.2. Justificación

En la actualidad, son muchos los avances tecnológicos junto a las áreas de investigación sobre las cuáles el hombre pretende buscar soluciones a problemas reales; como la búsqueda de un modo de energía alterna al uso del petróleo ó el reciclaje del agua. Si bien, estos son grandes interrogantes en un camino extenso; el presente proyecto pretende ser una aproximación al alcance de dichas metas; al generar herramientas modernas y eficientes en el desarrollo de conocimiento para la apropiación de tecnolgías.

Los proyectos de dicha índole, nacen de iniciativas gubernamentales y avances académicos en el sector privado ó público por parte de entidades educativas y empresas. Por tal razón, surge la gran importancia al seguimiento en los avances Universitarios; junto al desarrollo intelectual de la creación de proyectos en líneas de trabajo como la robótica; generando antecedentes para los grupos de investigación.

Por último, cabe recalcar los inmensos beneficios aportados por la participación en este trabajo de grado; al fortalecer la formación académica, en la implementación y desarrollo de las capacidades adquiridas durante la formación en ingeniería, junto a su aplicación en la resolución de problemas.

1.3. Antecedentes

Si bien la implementación de robots durante la última década ha incrementado y especialmente el conjunto de la robótica denominado robótica móvil ha evolucionado a tal punto que su comercialización es muy amplia y aceptada por la demanda de la comunidad estudiantil y profesional. La necesidad básica, es la de extender el campo de actividad del hombre hacia ambientes hostiles ó de tiempo de trabajo muy corto; permitiendo de esta manera ampliar nuestro cocimiento El primer robot móvil de la historia, fue ELSIE (Electro-Light-Sensitive Internal-External), construido en Inglaterra en 1953. ELSIE era un robot que aunque no tenía inteligencia muy compleja, era capaz de moverse siguiendo una fuente de luz. En 1968, apareció SHACKEY del SRI (standford Research Institute), que estaba provisto de una diversidad de sensores así como de una cámara de visión y sensores táctiles [1].

La idea básicamente con el pasar de los años fue la de proveer a los robots de autonomía tanto de movimiento como de control; en muchos casos se realizaba por medio de cables conectados al móvil y adaptados a alguna clase de base de operaciones estática; pero debido al avance tecnológico, se permitió el uso de baterías recargables portátiles y comunicaciones por medio de radio frecuencia, dándole a los robots móviles la autonomía necesaria para poder enfatizarse ya propiamente en la resolución del problema el cual se estaba tratando.

Con el tiempo, puesto que las cuestiones básicas de portabilidad eran en la mayoría de los casos prácticamente solucionables, los procesos a los cuales eran aplicados el análisis hecho por medio de robots móviles fueron creciendo y con esto el coste de procesamiento de la máquina; los robots comenzaron a tener sistemas de procesamiento y análisis de datos incorporados dentro de ellos mismos permitiendo de esta manera ampliar el valor autónomo de los mismos.

A este conjunto de sistemas incorporados, se le dio el nombre de microprocesadores embebidos, el primero en ser pionero fue la empresa Intel en 1970 con el famoso Intel 8080 y de ahí en adelante su uso fue ampliamente extendido, virtualmente todas las impresoras tienen al menos un microprocesador[12].

Fue entonces cuando al léxico informático se agrego el término Sistema Embebido, muchas han sido las aproximaciones conceptuales por parte de la autoría en el área de la robótica y la electrónica pero básicamente se puede ver como un sistema de procesamiento de información incorporado dentro de productos como carros, telecomunicación ó equipos de fabricación. Siguiendo el éxito que tuvo la información tecnológica para la oficina y las aplicaciones comerciales, los sistemas embebidos están considerados como el área de aplicación más importante de la información tecnológica en los próximos años. Debido a este hecho, en el futuro los computadores estándar serán el tipo de hardware menos dominante; procesadores y software serán utilizados en sistemas más pequeños y en muchos casos serán invisibles[9].

El presente proyecto consiste en el uso de un sistema embebido colocado en un robot móvil con el propósito de controlarlo desde un computador distante, que se comunica por medio del protocolo TCP/IP. Se adaptará una cámara USB junto al sistema embebido, que procesará las imágenes para enviarlas al sistema remoto, donde el usuario recibirá la imagen y será capaz de controlar el movimiento del robot. El sistema embebido se comunicará con el robot por medio del puerto serial (RS232). También se explicarán los tópicos referentes al manejo y uso del sistema embebido para una persona con conocimientos avanzado en la programación de software usando el lenguaje de programación C++, la libreria gráfica Qt y Qtopia[1]. Las plataformas sobre las cuáles se desarrollo este proyecto fueron Linux (para el sistema embebido con la distribución Fedora 10) y Windows (para el software utilizado por el usuario a distancia).

1.4. Alcances y limitaciones

El rediseño básico del robot móvil, consiste en el mejoramiento del sistema de navegación. Se removerán los sensores de ultrasonido (que permiten la transfiguración de los datos por medio de la inserción de ruido) para ser reemplazados por un sistema tele-operado usando realimentación visual. La planeación de trayectorias ya no será realizada por medio del sistema de posición global (GPS); sino que el robot se encargará, de viajar a través de un ambiente con obstáculos, guiado por el usuario desde la estación teleoperada.

1.5. Objetivo General

Implementar el sistema para la teleoperación de un robot móvil usando realimentación visual.

1.6. Objetivos Específicos

- Hacer uso de un sistema embebido basado en un microprocesador como unidad central de procesamiento en el robot móvil.
- Implementar un programa para realizar adquisición, compresión y transmisión de imágenes entre el robot móvil y la interfaz de usuario.
- Implementar una técnica de control para los motores desde el sistema embebido teniendo en cuenta las instrucciones dadas por el usuario.
- Hacer uso del protocolo TCP/IP y wireless para transmitir los datos entre el móvil y la interfaz de usuario.

[1] Qt software anunció que para después del 30 de septiembre de 2008, la librería Qtopia sería renombrada a Qt Extended, pero para el 3 de marzo de 2009 Qt software anunció que Qt Extended no se continuaría como un producto individual sino que algunas capacidades migrarán dentro del framework de Qt. Para los usuarios de Qt Extended, puesto que el proyecto era software libre se continuo por parte de la comuninad libre y se creó Qt Extended Improved.

1.7. Contenido del DVD

Junto a este trabajo, se anexa un DVD que contiene los siguientes archivos.

- Código fuente programa Master.
- Código fuente programa del Sistema Embebido.
- Código fuente programa microcontrolador.
- Fotografías de las pruebas realizadas en la plataforma robótica.
- Video de las pruebas realizadas.
- Hojas de datos de los componentes usados en éste proyecto.
- Diagramas del circuito para la adquisición de datos desde el sistema Embebido.
- Diagramas UML de las clases.
- Documento del trabajo de grado en versíón pdf.

2. FUNDAMENTOS BÁSICOS

Los temas incluidos en el desarrollo de éste trabajo abarcan una amplia cantidad de conceptos. A continuación se presentan los tópicos generales, analizando básicamente tres grandes áreas: Sistemas Robóticos Teleoperados, Comunicación en Redes usando el protocolo TCP y Sistemas Embebidos Linux.

2.1. Sistemas Robóticos Teleoperados

Los sistemas robóticos móviles son sistemas desarrollados por el hombre, para posibles soluciones de problemas como los anteriormente descritos. La robótica móvil es frecuentemente implementada para el desarrollo de plataformas, las cuales son generalmente utilizadas para recorrer aquellos espacios, que son de alguna manera inaccesibles para el hombre.
La robótica teleoperada es usada para el desarrollo de robots los cuales son aquellos controlados por usuarios a distancia desde una estación remota, desde la que pueden darle ordenes a los robots, en cuanto a su movimiento y las capacidades que tienen para completar su objetivo. Este tipo de manejo ofrece al operador una ventaja, ya que otorgan seguridad y protección al usuario, además de brindar más rapidez, eficiencia y precisión en la ejecución de trabajos determinados[4].
Por consecuencia la robótica móvil teleoperada es el desarrollo de robots, no solo con la capacidad de moverse en distintos ambientes, sino también la capacidad de operarlos a distancia, lo cual permite no solo ofrecer seguridad al usuario; si no que además, la capacidad de interactuar con el entorno de trabajo para hacer operaciones en el.

2.2. Componentes de un Sistemas Teleoperado

Los sistemas teleoperados como se dijo anteriormente se componen principalmente de tres elementos.

- Estación de teleoperación ó Máster
- Sistema de Comunicación
- Estación de Trabajo ó Slave

La estación de teleoperacion o Máster es la que se encarga de controlar la estación de trabajo o Slave desde un espacio lejano a esta. El máster está implementado generalmente en un computador,

el cual tiene dispositivos de entrada para poder controlar el Slave, a través de un software en el cual se encuentra la implementación del sistema de comunicaciones del Máster, este atiende llamadas del sistema de Comunicaciones del Slave, y responde a ellas, este software también tiene implementado una interfaz grafica, en la cual se tiene acceso a señales propias del Slave, que dice al Máster propiedades físicas del entorno de trabajo, y a las cuales se deben responder mediante señales de control desde el Máster al Slave mediante la interfaz de usuario. Para la implementación del software, se utilizan herramientas como Labview, Matlab o se desarrollan software propios de cada sistema, en lenguajes como c++, java, phyton, entre otros.
El sistema de comunicaciones es el que se encarga del correcto entendimiento entre el Máster y el Slave. Este se implementa para que las señales de control del Máster, sean correctamente entendidas e implementadas por el Slave, este sistema se tiene que implementar en el software de ambas estaciones, y se tiene que definir antes de la implementación, si el sistema de comunicaciones tiene que ser sincrónica o asincrónica dependiendo de la tarea a realizar y del tiempo de acción y reacción del sistema. La comunicación sincrónica es aquella en la cual el Máster envía señales de control de la comunicación. Se implementan controles al principio de cada mensaje para que el Slave sepa que va a recibir uno, para el envió del siguiente mensaje, el Slave tiene que contestar al mensaje previo con señales de control hacia el Máster, para que este sepa que el Slave ya recibió el paquete. Esta técnica supone mejor entendimiento y seguridad en la integridad del mensaje pero reduce bastante el tiempo de la comunicación por las señales de control. La comunicación asincrónica es aquella en la cual el Máster envía señales al Slave y no se preocupa por saber si el Slave recibió el mensaje de forma correcta, en cambio el Máster sigue enviando paquetes al Slave. Esta técnica supone un tiempo de comunicación reducido pero el hecho de no saber de la integridad del mensaje y tampoco la correcta recepción en el Slave puede suponer perdidas de información restando eficiencia al sistema. Estos sistemas de comunicaciones suelen ser implementados en interfaces LAN, radiofrecuencia, microondas, infrarrojos y wireless entre otras.
La estación de trabajo o Slave es el que se encarga de recibir señales de control desde el Máster, de acuerdo a estas toma acciones sobre el entorno de trabajo, además de esto se encarga también de entregar información al Máster, tomando magnitudes físicas del entorno, otorgando al teleoperador información importante sobre el proceso que se está desarrollando.

2.3. Funcionamiento del Robot

Para el funcionamiento óptimo del Slave está dotado de un “cerebro”, el cual se encarga de tomar decisiones sobre las señales de control que llegan desde el Máster, este “cerebro” es implementado mediante un microcontrolador, este para las tareas más comunes y con menos requerimiento de procesamiento de señales. Cuando el proceso requiere que desarrollen tareas más difíciles en cuanto a capacidad de procesamiento, se recurren a sistemas como las FPGA o sistemas embebidos, ya que sus características de procesamiento y velocidad son más altas, esto permite hacer procesamiento sobre señales, imágenes o estructuras de datos, otorgando al Slave una mayor independencia del

usuario. El “cerebro” es el que tiene inmerso el software del Slave, el cual se encarga de comunicarse con el Máster para un correcto entendimiento entre las partes. También se encarga de tomar las magnitudes físicas del entorno, y enviarlas al Máster.

2.3.1. Tipos de locomoción

Los sistemas Slave principalmente son robots móviles o robots antropomórficos como brazos o actuadores de diversos tipos, esto para que se puedan manejar en el entorno de trabajo, y puedan realizar las acciones sobre este de una manera optima. Cuando estos sistemas Slave son robots móviles, existen diferentes sistemas de locomoción tales como ruedas, orugas, patas entre otras, las cuales permiten al sistema Slave moverse en el entorno de trabajo. Estos tipos de locomoción se escogen mediante el previo estudio del entorno en el cual el robot móvil se va a desplazar, ya que cada uno de estos ofrece diferentes características especiales para diversos tipos de ambientes. A continuación se citan algunos ejemplos de locomoción [3].

Ackerman: Es el utilizado en vehículos de cuatro ruedas convencionales. De hecho los vehículos robóticos para exteriores resultan normalmente de la modificación de vehículos convencionales tales como automóviles o incluso vehículos más pesados. El mayor problema de la locomoción Ackerman es la limitación de la maniobrabilidad.

Triciclo clásico: En este sistema de locomoción la rueda delantera sirve tanto para la tracción como para el direccionamiento. El eje trasero, con dos ruedas laterales, es pasivo y sus ruedas se mueven libremente. La maniobrabilidad es mayor que en la configuración anterior pero puede presentar problemas de estabilidad en terrenos difíciles. El centro de gravedad tiende a desplazarse cuando el vehículo se desplaza por una pendiente, causando la perdida de tracción. Debido a su simplicidad, es bastante frecuente en vehículos robóticos para interiores y exteriores pavimentados.

Direccionamiento diferencial: El direccionamiento viene dado por la diferencia de velocidades de las ruedas laterales. La tracción se consigue también con estas mismas ruedas. Adicionalmente existen una o más ruedas para soporte. Esta configuración es la más frecuente en robots para interiores.

Skid Steer: Se disponen varias ruedas en cada lado del vehículo que actúan de forma simultánea. El movimientos el resultado de combinar las velocidades de las ruedas de la izquierda con la derecha.

Pistas de deslizamiento: Son vehículos tipo oruga en los que tanto la impulsión como el direccionamiento se consiguen mediante pistas de deslizamiento. Pueden considerarse funcionalmente análogas al skid steer. De forma más precisa, las pistas actúan de forma análoga a ruedas

Locomoción mediante patas: Permiten aislar el cuerpo del terreno empleando únicamente puntos

discretos de soporte. Es posible adoptar el polígono de soporte para mantener la estabilidad y pasar sobre obstáculos. Por consiguiente, tiene mejores propiedades que las ruedas para avanzar en terrenos difíciles llenos de obstáculos. Asimismo, mediante patas, es posible conseguir la omnidireccionalidad y el deslizamiento en la locomoción es mucho menor. En los robots con patas la complejidad de los mecanismos necesarios es mayor, así como el consumo de energía en la locomoción.

2.3.2. Sensores para estaciones Slave

En los sistemas teleoperados, los Slave están dotados de diversos tipos de sensores, como se dijo antes el sistema Slave se encarga de medir variables físicas del entorno de trabajo y estos sensores son los encargados de tomar estas medidas que otorguen al usuario características del espacio de trabajo. Estas variables pueden ser del tipo térmico, mecánico, electrónico, entre otros. Es necesario estudiar previamente el lugar de trabajo, para saber que variables físicas puede tener, y así elegir adecuadamente los sensores de la estación Slave. Estos sensores pueden ser del tipo ultrasonido, infrarrojo, resistivo, voltaico entre otros los cuales cada uno sirven para un tipo determinado de variable física a medir.

También se puede dotar al sistema Slave, con una cámara que ofrezca realimentación visual a la estación Máster. Esto da al teleoperador una visión en tiempo real de lo que ocurre no solo con el sistema Slave, sino con el entorno de trabajo, para la adaptación de estas cámaras en un sistema Slave, se tiene que contar con un sistema que pueda identificar la cámara, y enviar las imágenes del entorno de trabajo al Máster. Para esto es necesario contar con una FPGA o un sistema embebido. Si no se cuenta con uno de estos sistemas, la cámara puede comunicarse con el Máster mediante sistema wireless siendo necesario que sea una cámara IP, que provee este tipo de comunicación y se puede implementar en el sistema Slave.

2.4. Comunicación en Redes usando el protocolo TCP

En la actualidad la gente usa el computador para hacer llamadas, ver televisión, enviar mensajes a sus amigos, jugar con otras personas y comprar casi cualquier cosa. La capacidad que tiene estas máquinas de comunicarse entre ellos, hacen todo esto posible. El número de personas conectadas a internet, crece cada vez más a través el tiempo, y con el desarrollo de mayores velocidades en la red, las aplicaciones desarrollables, a través de internet son casi inimaginables.

¿Pero como hacen los computadores para conectarse a otros a través de la red? Esto se logra mediante la programación de un software, capaz de interactuar con el usuario, y que tiene inmerso una API (Application Programming Interfaces) conocida como socket, que se desarrollo por primera vez en lenguaje C, y que debido a su aplicación, ha sido extendida para su desarrollo, en los lenguajes

más comunes hoy en día. Es necesario hacer una breve mirada en el panorama de las redes y los protocolos que se usan en estas, términos como TCP (Transmission Control Protocol), UDP (User Datagram Protocol) y demás protocolos que serán posteriormente explicados, esto para una correcta implementación de nuestro socket[10].

2.4.1. Redes, Paquetes y Protocolos

Una red se compone de máquinas interconectadas por canales de comunicación. Estas máquinas son llamadas host y routers. Los host son los computadores que corren aplicaciones, como el explorador web, mensajería instantánea, etc., estos programas son los verdaderos usuarios de la red. Los routers se encargan de transmitir la información, desde un canal de comunicación a otro. Un canal de comunicación, es el medio de transmisión de secuencias de bytes de un host a otro, este puede ser cableado, wireless u otro tipo de conexión.
La información que se envía, es una secuencia de bytes que son construidos y posteriormente interpretados por los programas. En el contexto de las redes de computadores, esta información es generalmente llamada paquetes. Un paquete contiene información de control, que es utilizada por la red para realizar la comunicación. También puede incluir los datos del usuario. Los routers utilizan esa información, para saber cómo y adonde enviar cada paquete.

Un protocolo es un acuerdo que dice como deben ser interpretados los paquetes intercambiados. Los protocolos están normalmente diseñados para resolver problemas determinados de comunicación. Para la implementación de una red optima, se utilizan conjuntos de protocolos como el TCP/IP. Este puede usarse para establecer una comunicación sobre internet o bien sobre una red privada. Los protocolos TCP/IP más usados son el IP (Internet Protocol), TCP (Transmission Control Protocol) y UDP(User Datagram Protocol).

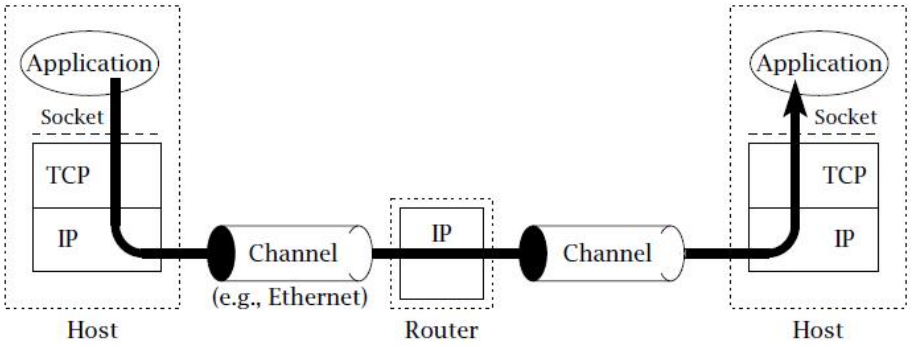


Figura 2.1: Relaciones entre los protocolos

La figura muestra las relaciones entre los protocolos, aplicaciones, los sockets en los hosts y routers y el flujo de dados de una aplicación a otra. Los bloques de TCP e IP representan la implementación

de estos protocolos en una comunicación sobre una red, estos protocolos residen en la aplicación de los host. Las solicitudes de acceso a los servicios prestados por los sockets, están representados con una línea discontinua. La flecha representa el flujo de dados de la aplicación.
En TCP/IP, la capa inferior se compone de los canales de comunicación. Estos canales son utilizados por la capa de la red, que se ocupa del problema de la transmisión de paquetes hacia su destino. El protocolo IP resuelve el problema de hacer, que la secuencia de canales y routers entre dos host, parezca una conexión simple host-to-host. El IP proporciona un servicio de datagramas, cada paquete se maneja y entrega por la red de forma independiente. Para realizarlo cada paquete IP contiene la dirección del destino.
La capa superior de TCP/IP es la capa de transporte, esta ofrece dos posibilidades: TCP y UDP. Cada uno ofrece distintos tipos de transmisión de los datos, dependiendo de la aplicación. Pero tiene una cosa en común, la de Direccionar los paquetes, para esto usan los números de los puertos, para identificar las solicitudes de las aplicaciones de los host. Estos protocolos (TCP y UDP) se llaman de extremo a extremo, ya que estos se encargan de enviar los datos de un programa a otro.

2.4.2. Direccionamiento

Antes de que un programa pueda comunicarse con otro programa, tiene que decirle algo a la red para identificar al otro programa. En TCP/IP, se tiene dos piezas de información para identificar un programa en particular: una dirección de internet y un número de puertos. Las direcciones de internet son números binarios. Vienen en dos versiones dadas por el protocolo de Internet: IPv4 e IPv6. La más común es la IPv4, la cual es de 32 bits, logrando así cerca de 4 millones de posibilidades distintas, para identificar los diferentes destinos. La IPv6 es de 128 bits pero no es muy común, ya que hasta ahora se está implementando.

- *Direcciones IP*: Las direcciones IP son un grupo de cuatro números decimales separados por puntos. (Por ejemplo 192.168.1.1). Los cuatro números representan la cadena contenida en los cuatro bytes de la dirección de internet. Por lo tanto cada uno es un número entre 0 y 255. Técnicamente, cada dirección de internet se refiere a la conexión entre un host y un canal de comunicación ósea una interfaz de red. Debido a que cada conexión de red pertenece a un único host, una dirección de internet identifica a un host, como su conexión a la red.
- *Puertos*: Se mención anteriormente, que se necesitan dos piezas de la dirección para recibir un mensaje de un programa. El numero del puerto TCP o UDP, es interpretado relativamente a la dirección de Internet. El puerto se puede interpretar como un hilo de la comunicación sobre la dirección IP utilizada. Sobre este puerto es que el host va a estar esperando la comunicación desde la otra aplicación.
- *Clientes y Servidores*: En las comunicaciones en internet, los términos servidor y cliente son comúnmente utilizados. El programa Cliente inicia la comunicación, mientras que el programa

servidor espera pasivamente, y a continuación, responde a los clientes que entren en contacto con él. En conjunto el cliente y el servidor componen la aplicación. Cuando un programa actúa como servidor o cliente, determina la forma general como se usan los sockets para establecer la comunicación con sus pares. Además el enlace entre cliente-servidor es importante porque el cliente necesita saber la dirección del servidor, y el puerto al que se va conectar y no viceversa. Con los sockets, el servidor puede, obtener la información de dirección del cliente cuando recibe la primera comunicación del cliente.

2.5. Sistemas Embebidos Linux

El trabajo realizado sobre sistemas embebidos Linux crece de manera exponencial debido a la evolución del kernel, junto a las distribuciones diseñadas para estaciones de trabajo y dispositivos embebidos. Este kernel permite el uso de una gran variedad de hardware, dándole gran versatilidad y aplicabilidad a estos sistemas.
Uno de los campos ampliamente estudiados radica en la autonomía de los sistemas que trabajan en campos de poca o ninguna interacción humana. El logro de dicha autonomía, se basa en la capacidad de procesamiento y el número de tareas que es capaz de realizar el sistema. Aquí es donde propiamente los sistemas embebidos son de gran ayuda, porque aunque pretenden simular el comportamiento de una estación de trabajo, realizan tareas con un amplio grado de efectividad.
Los conceptos explicados a continuación permitirán ubicarse dentro de un marco específico, enfocándose hacia el desarrollo de sistemas embebidos.

2.5.1. Linux

Técnicamente hablando, Linux se refiere solamente al kernel de un sistema operativo originalmente escrito por Linus Torvalds. El kernel provee una gran variedad de facilidades requeridas por cualquier sistema basado en Linux para operar correctamente. El software de aplicación depende de características específicas del kernel, como por ejemplo la capacidad de manejar los dispositivos de hardware y, su disposición al uso de una gran variedad de abstracciones fundamentales; como la memoria virtual, tareas, sockets, archivos, descriptores y demás. El kernel es iniciado básicamente por un *bootloader*[1] ó un system firmware, pero una vez inicializado, nunca se apaga (aunque el dispositivo parezca que entra a un estado de bajo consumo de energía).
Actualmente, el término "Linux" se utiliza de manera intercambiable junto al término Linux kernel. Por si sólo, se puede entender un sistema Linux como una distribución pre-compilada (o solamente con el código fuente) construida sobre un kernel junto con otros softwares. Pero esta definición podría tomarse a varios malentendidos, puesto que el kernel es un nucleo, que contiene software para la comunicación con los periféricos y demás unidades no centrales para el entendimiento del usuario.

[1] Bootloader significa: *cargador de arranque*. Durante todo el documento se usará los nombres originales y no se traducirán (a excepción de algunos conceptos cuya traducción ayudaría a su uso) puesto que esto podría llevar a confusiones en la comprensión del proyecto.

Por esta razón, Richard Stallman y la Free software foundation colocan el prefijo "GNU/" (como en "GNU/Linux") para referirse a un sistema completo trabajando con un linux kernel y una gran variedad de software GNU.
Puesto que es posible trabajar con distribuciones Linux que pueden usar software no-GNU, se amplia la definición anterior refiriéndonos a un sistema Linux como a un sistema completo o distribución trabajando con software GNU y no-GNU en un kernel.
Un sistema Linux puede ser creado desde cero a deseo del usuario (una gran ventaja comparada con otros tipos de sistemas operativos), o puede basarse en una distribución ya realizada. A pesar de la gran cantidad existentes, para el uso con sistemas embebidos y dispositivos embebidos Linux; muchas empresas prefieren crear sus propias distribuciones, potencializando su aplicabilidad de acuerdo a las necesidades del mercado.
Cuando una persona usa el término Linux, por lo general se refiere a una distribución Linux, como por ejemplo Red Hat, Fedora, Suse, Ubuntu, Kubuntu, etc. Las distribuciones de Linux varían en propósito, tamaño y precio; pero todas comparten una meta en común: proveer al usuario final con un conjunto de archivos pre-compilados, un procedimiento para instalar el kernel y otro conjunto de software para cierto tipo de hardware.
Desde el punto de vista embebido, existen una gran variedad de distribuciones, como por ejemplo; MontaVista, Wind River, Timesys, Denx, entre otros. Estas distribuciones embebidas especializadas, generalmente no están destinadas al uso de computadores de escritorio, estaciones de trabajo o servidores; esto significa que estas distribuciones, típicamente no incluirán software que no esta diseñado para su uso.
Otra de las grandes ventajas de utilizar Linux, es la posibilidad de cambiar y configurar el sistema operativo al gusto del usuario, basándose en el uso de diversas herramientas de libre distribución. Como por ejemplo el escritorio de trabajo GNOME, utilizado en distribuciones como Ubuntu; y KDE, utilizado en distribuciones como Kubuntu. En la mayoría de las distribuciones, existen repositorios de software libre, permitiéndole al usuario tener una gran variedad de alternativas a la hora de realizar una tarea específica. Los campos de aplicación varían desde, programación, diseño gráfico, audio, video, hasta entretenimiento.

2.5.2. Linux Embebido

Linux Embebido típicamente se refiere a un sistema completo, ó a una distribución destinada a dispositivos embebidos. Aunque el término "embebido" es usado comúnmente en el concepto de kernel; no hay un tipo de kernel diseñado para aplicaciones embebidas; en cambio, el mismo código fuente del kernel, está destinado para ser construido en una gran variedad de dispositivos, estaciones de trabajo y servidores. Para más inforamción, ver [6].
El kernel es (de manera general) un núcleo encargado de traducir los deseos del usuario al sistema. Por supuesto, se puede configurar el kernel para un uso más especializado.
En el contexto del desarrollo embebido, se pueden encontrar sistemas ó dispositivos con Linux embebidos que usan un kernel junto a una gran variedad de software para aplicaciones especiales.

Los distribuidores de sistemas embebidos, proveen una gran variedad de herramientas como cross-compilers[2], depuradores, software, boot image builders[3], entre otros. Una gran cantidad de distribuidores han decidido integrar estas herramientas dentro de plug-ins especializados para sus propias versiones disponibles a la venta.

2.5.3. Tipos de Sistemas Linux Embebido

Dentro de la creación de sistemas embebidos, pueden surgir muchos interrogantes a la hora de diseñar estos sistemas: ¿Qué tipos de sistemas embebidos están construidos con Linux? ¿por qué las personas eligen Linux? ¿cómo usarlo? ¿qué distribución usar?, entre otras. Por eso es necesario tener un conocimiento general de todos los aspectos que incluyen la creación y uso de un sistema embebido. Aunque se podría ejemplificar el uso de sistemas específicos, la mejor forma de diseñar un sistema embebido (o canalizar su uso a una tarea específica) es clasificándolos de acuerdo a un criterio general, que proveerá información acerca de su estructura. Estos criterios son los siguientes: Tamaño, limitaciones de tiempo, capacidad de red y la interacción con el usuario.

- *Tamaño*

  El tamaño de un sistema embebido Linux depende de una gran variedad de factores. Primero, está el tamaño físico. Algunos sistemas pueden ser modestamente grandes, como los construidos por agrupaciones, o pueden ser pequeños, como el reloj de pulsera construidos por IBM. El tamaño físico de un sistema embebido es a menudo una determinante importante en las capacidades del hardware (el tamaño físico de los componentes dentro del dispositivo terminado) y también es necesario tener en cuenta, el tamaño de los componentes de la máquina. Estas consideraciones son muy importantes para los diseñadores de sistemas; incluyendo también, la velocidad de la CPU, el tamaño de la RAM y el tamaño de la memoria permanente (el cuál puede ser un disco duro, pero a menudo es un dispositivo flash NOR o NAND; de acuerdo al uso).

  En términos de tamaño, se usarán tres variantes, pequeña, mediana y grande. Sistemas pequeños se caracterizan por una CPU de baja capacidad con un mínimo de 4 MB de ROM y entre 8 y 16 MB de RAM. Esto no quiere decir que Linux no se pueda ejecutar en pequeños espacios, pero si tomará mucho esfuerzo y dedicación, hacer tareas que requieran grandes cantidades de procesamientos de datos.

  Sistemas de tamaño medio se caracterizan por una CPU de capacidad media con 32 MB o más de ROM y 64-128 MB of RAM. La mayoría de usuarios dentro del mundo embebido pertenecen

[2]El término en inglés es *cross-compilers* cuya traducción literal sería compilador-cruzado se refiere al compilador cuyo código fuente es declarado en un tipo de máquina (host) y se implementa para otro tipo de máquina (target). Más adelante se hablará sobre este tema, pero de aquí en adelante se utilizará el término en inglés.

[3]Muchos de los paquetes utilizados en los sistemas embebidos vienen (por parte de los vendedores) como paquetes comprimidos en un sólo formato definido como *imagen* al igual que los formatos conocidos para los sistemas no embebidos ISO, BIN, NRG, CDI, DAA.

a ésta categoría; incluyendo varios PDAs (por ejemplo the Nokia Internet Tablets), reproductores de MP3, sistemas de entretenimiento, aparatos que usan red, memorias extraíbles, entre otros. Estos tipos de dispositivos tienen suficiente potencia y almacenamiento para manejar una gran variedad de pequeñas tareas, o pueden servir para un único propósito que requiera muchos recursos.

Sistemas de tamaño grande se caracterizan por una gran capacidad de CPU o colección de CPUs combinadas con una gran cantidad de RAM y memoria permanente. Usualmente estos sistemas son usados en ambientes que requieran gran cantidad de cálculos para realizar una tarea determinada; como por ejemplo los simuladores de vuelo utilizados en la investigaciones del gobierno. Típicamente en estos sistemas no importa el dinero o la cantidad de recursos necesarios, lo único importante es la funcionalidad; mientras que el costo, tamaño y complejidad permanecen en un segundo plano.

- *Limitaciones de Tiempo*

  Hay dos tipos de limitaciones de tiempo para los sistemas embebidos: limitaciones rigurosas y limitaciones leves. Las limitaciones rigurosas de tiempo, requieren que el sistema actúe en un marco predefinido de tiempo; de otra forma, eventos catastróficos pueden ocurrir. Por ejemplo, en una fábrica donde los empleados deben manipular material que debe ser cortado por una máquina. Cómo una medida preventiva, se colocan sensores ópticos alrededor de las cuchillas de corte, para detectar guantes especiales de colores, usados por los empleados. Cuando el sistema es advertido de la presencia del guante de un usuario, éste debe detenerse automáticamente. La máquina no puede esperar por algún tipo de operación I/O. De otra forma, alguien perdería un brazo.

  Sistemas y dispositivos de audio como reproductores MP3 y teléfonos celulares, podrían calificarse como limitaciones de tiempo rigurosas; porque cualquier atraso en la transmisión de la señal, repercutiría en la calidad del sonido. Aunque las fallas de tiempo (de existir) no serían consideradas catastróficas, la calidad de los dispositivos, es algo primario en el diseño de dichos sistemas.

  Las limitaciones leves de tiempo, varían de gran manera en los requisitos de diseño, pero generalmente se aplican a sistemas donde las demoras de tiempo no son tan críticas. Si un sistemas se demora 5 segundos en realizar una transacción, generalmente no es un problemas (aunque el usuario puede pensar de manera diferente). Lo mismo es para las PDAs que toman tiempo de más, para inicializar alguna aplicación. El tiempo extra puede hacer que el sistema parezca lento, pero esto no va a afectar el resultado final. Lo importante es notificarle al usuario del tiempo requerido por el sistema para terminar su tarea.

- *Capacidad de Red*

  La capacidad de Red, define la disposición de un sistema para conectarse a un red. Hoy en día, es de esperarse que cualquier dispositivo sea accesible a través de una red. Esto hace que el sistema deba cubrir ciertos requerimientos (un factor definitivo en la elección de Linux para el diseño de sistemas embebidos, radica en su capacidad para la conexión de red); permitiendo,

que la caída en los precios y a estandarización de los componentes de red están acelerando está tendencia. La mayoría de los dispositivos que usan Linux tiene una u otra forma de capacidad de red, bien sea inalámbrica o no.

- *Interacción con el Usuario*

  El grado de interacción con el usuario varia ampliamente de un sistema a otro. Algunos sistemas, como las PDAs y la Nokia Internet Tablet, están centrados en la interacción del usuario; mientras que otros tipos, como los sistemas industriales, sólo tienen LEDs y botones para interactuar con el usuario. Otros sistemas, en cambio no interactúan con el usuario, como por ejemplo los sistemas de piloto automático.

2.5.4. Tipos de Estaciones de Desarrollo

Para los sistemas embebidos que trabajan con Linux, es importante elegir el tipo de distribución para el computador *host*, de acuerdo al compilador que creará los binarios para ejecutar en el sistema embebido. Por tal razón, es necesario revisar la documentación existente (preferiblemente con el distribuidor del sistema embebido) acerca de cuáles versiones de los compiladores y depuradores funcionan sin presentar problemas. Los criterios para la elección de una distribución específica son materia de grandes artículos, pero aquí sólo se mencionará la importancia de adquirir una distribución que funcione (sin problemas) con los programas a ejecutar.

Tipos de distribuciones Linux

Más adelante se discutirán los tipos de hardware más comunes encontrados en sistemas Linux embebidos. Pero aunque el hardware es muy importante para el diseño general del sistema, cada posible hardware puede ser desarrollado usando una gran variedad de *hosts*. En la siguiente sección se discutirán los tipos de distribuciones comúnmente usados para los *hosts* y sus particularidades.

Estaciones de Trabajo Linux

Éste es el tipo más común de host para el desarrollo de sistemas embebidos. Éste es recomendado por muchos usuarios, puesto que sus facilidades de trabajo y código abierto ayudan en gran medida al desarrollo de los proyectos. Usar Linux diariamente permite que se conozcan muchos de los problemas usuales (o errores por parte del usuario) que suelen ocurrir; de igual manera se pueden resolver en el sistema embebido (puesto que pueden usar un kernel igual o similar). Un PC estándar es la típica estación de trabajo Linux. De cualquier manera, Linux no corre solamente en hardware tipo PC sino en una gran variedad de sistemas. A continuación se mencionan algunas distribuciones.

- Debian GNU/Linux (http://www.debian.org)
  Está popular y común distribución, es mantenida por un grupo internacional de desarrolladores y tiene soporte bajo la organización de caridad “Software In The Public Interest”. Debian tiene

altos estándares, pero es (ocasionalmente) conocido por presentar pequeños problemas en la instalación y para el uso de usuario nuevos. No se actualiza bajo ningún tipo de calendario.

- Fedora (http://www.fedoraproject.org)

  La continuación libre del famoso "Red Hat Linux" que ya no existe, sino únicamente bajo la obsoleta referencia (Red Hat Linux 9.0) de soporte, que persiste aún en Internet. Fedora es desarrollado internacionalmente, pero tiene tradicionalmente una fuerte relación con la empresa Red Hat Enterprise Linux Distribution. Fedora es actualizado aproximadamente de 6-9 meses. Éste tiene una actualización no estable conocida como "rawhide" la cuál es actualizada diariamente de manera individual a Fedora.

- OpenSusE (http://www.opensuse.org)

  Si Fedora es la continuación moderna de Red Hat Linux, OpenSuSE es lo mismo, pero de la distribución conocida como SuSe. Después que Novell adquirió SuSE y comenzó a distribuir SuSE Linux Enterprise Server (SLES), OpenSuSE se convirtió en la incubadora de futuras tecnologías. Se actualiza de manera similar a Fedora y también tiene una versión no estable conocida como "Factory" que es actualizada de manera diaria.

- Red Hat Enterprise Linux (RHEL) (http://www.redhat.com)

  Ésta es la versión comercial de Red Hat y un heredero directo de la línea Red Hat Linux. Se actualiza cada 18 meses y tiene soporte a usuarios desde sus inicios. El soporte es en la forma de suscripción. Aunque muchos han logrado efectivamente re-empaquetar RHEL en una distribución Enterprise Linux libre (la cuál no viene con soporte, pero los binarios de la versión comercial con la libre son prácticamente iguales).

- SuSE Linux Enterprise Server (SLES) (http://www.suse.com)

  Ésta es una distribución comercial de Novell, la cual produce también SLED(SuSE Linux Enterprise Desktop) y una variedad de otros productos. Uno de esos productos es producto comercial de tiempo-real. SLES es actualizado de manera similar a Red Hat Enterprise Linux y compite con él directamente.

- Ubuntu Linux (http://www.ubuntulinux.org)

  Ésta es una derivación de Debian GNU/Linux, pero desarrollada para un mercado y uso masivo, con actualizaciones bianuales. Es apoyado por Canonical, una compañía formada por el multimillonario Mark Shuttleworth, él cual se hizo rico a través de una compañía que dependía enteramente en Debian y el desarrollo de su estructura. Un gran número de desarrolladores de PCs están considerando suplir el proyecto, ó ya están involucrados dejando PCs con Ubuntu pre-instalado.

- Yellow Dog Linux (http://www.terrasoftsolutions.com)

  Ésta distribución es una derivación para sistemas PowerPC, como los de IBM (basados en procesadores POWER/OpenPOWER/PowerPC) y formalmente aquellos de Apple Computer. Aunque no es tan común, se mantiene como el ejemplo de una distribución intentada específicamente para arquitecturas no-PC en primera instancia.

Como se mencionó anteriormente, existen muchas más distribuciones, por ejemplo Slackawre, Gentoo, Rock Linux, Tom Root Boot, Mandriva, entre muchas más.

Para mayor información acerca de las diferentes distribuciones, sus requerimientos y los tipos de hardware que mejor encajan para un proyecto determinado, consulte los diferentes sitios en Internet de cada distribución, también distrowatch.com y demás artículos en la red acerca del tema.

Estaciones de Trabajo Unix

De acuerdo a las circunstancias, se puede requerir usar una estación tradicional de trabajo Unix, aunque es recomendable no hacer esto si la opción de usar Linux está a la mano. Estaciones de trabajo Solaris por ejemplo, son muy comunes entre desarrolladores en el área de telecomunicaciones. Aunque el uso de estas estaciones de trabajo no es tan común como las comparadas con Linux para desarrollar sistemas embebidos, es aún posible realizar proyectos de esa magnitud. De hecho, modernos sistemas Solaris (y OpenSolaris) incluyen una gran cantidad de software GNU pre-compilado, como por ejemplo el famoso compilador gcc.
Puesto que Linux es en si muy parecido a Unix, se puede decir que lo que aplica a Linux también aplica a Unix y otros sistemas tipo Linux, como por ejemplo BSD (OpenBSD, FreeBSD y demás); especialmente cuando se trata de la famosa herramienta (toolchain) del desarrollo GNU y otras herramientas como la libreria C que ya estaban creadas antes que Linux comenzará en el mercado de los computadores. Aunque la mayoría de las recomendaciones dadas anteriormente son aplicables de igual manera a los sistemas Unix, es recomendable no obstante usar sistemas Linux.

Estaciones de Trabajo Windows (Vista, XP, 2000, NT, 98, etc.)

A comienzo de los 90s, el desarrollo de sistemas embebidos cambio para ser desarrollado en estaciones de trabajo Windows. Esto fue debido al gran crecimiento del desarrollo de herramientas gráficas disponibles, las cuales llegaron a tiempo para un mercado de sistemas embebidos en crecimiento. Desde entonces muchos desarrolladores se acostumbraron a este tipo de estaciones de trabajo para realizar sus proyectos. Por esta y otras razones, algunos desarrolladores les gustaría continuar usando estaciones de trabajo Windows para desarrollar (irónicamente) sistemas embebidos Linux. Aunque es recomendable trabajar en estaciones de trabajo Linux para cualquier desarrollo, y a pesar de la existencia de herramientas GUI multiplataforma (como por ejemplo Eclipse), existe la necesidad para muchos usuarios del uso de herramientas únicamente para Windows. Esto incluye un gran número

de herramientos por parte de los distribuidores como depuradores, programadores de hardware FPGA/PROM/Flash y similares. Se sabe que no todos, desearán mantener dos sistemas de desarrollo Linux para proyectos únicamente en Windows, usando herramientas de este sistema. Lo mejor de todo, es que es posible usar Windows en el host y compilar programas para targets Linux embebidos. A primer vista, parecería que el principal problema usando Windows para desarrollar targets con Linux embebido, es la falta de desarrollo para herramientas GNU. Actualmente, esto no es un problema, porque Red Hat provee el desarrollo de Cygwin la cual es la herramienta GNU compatible con Windows. Muchas personas se acostumbraron a compilar herramientas multi-plataforma en Linux, incluyendo un gran número de estaciones. Si se va a trabajar en un ambiente embebido para Linux, el distribuidor de dicho sistema debe por lo general proveer las herramientas necesarias para compilar targets Windows (basados invariablemente en Eclipse) con lenguajes de programación como C++ [7] o java.

2.6. Modulación por Ancho de Pulsos (PWM)

La modulación por ancho de pulso es un modo de digitalmente codificar los niveles de una señal análoga. A través del uso de contadores de alta resolución, el ciclo de trabajo de una señal cuadrada es modulado para codificar un nivel específico de una señal análoga. La señal del PWM sin embargo es digital, porque en cualquier instante de tiempo la señal de alimentación DC está completamente apagada (off) ó encendida (on). EL voltaje o la corriente de alimentación se aplica a la carga en una serie repetitiva de pulsos on y off. El tiempo de encendido (on) ocurre cuando la señal DC es aplicada a la carga; y el tiempo de apagado (off), ocurre cuando la alimentación no se aplica a la carga. Con un valor suficiente de ancho de pulso, cualquier señal análoga puede ser codificada con PWM.[2]

2.6.1. Controladores PWM

Muchos microcontroladores incluyen controladores PWM. Por ejemplo el PIC 16F877A de Microchip incluye dos PWM, permitiendo en cada uno seleccionar el tiempo de *on* y el periodo. El ciclo de trabajo es el cociente entre el tiempo *on* y el periodo; la frecuencia de modulación es el inverso del periodo. Para empezar la operación del PWM, la hoja de datos sugiere las siguientes configuraciones:

- Configurar el periodo de encendido (on) del timer, que provee la señal cuadrada de modulación.
- Configurar el tiempo de encendido en el registro de control del PWM.
- Configurar la dirección de salida del PWM (Pins I/O).
- Inicializar el timer.
- Activar el controlador PWM.

Para la configuracion del PWM, se empieza hallando el periodo de la señal, el cual esta en funcion de:

- Frecuencia del Cristal (FC).
- Valor del Timer (T#_DIV).
- Prescaler del Timer (PR#).

Donde el # indica el tipo especifico de registro que se este utilizando. A continuacion se muestra la ecuacion, con la que se halla el periodo de la señal PWM.

$$T \equiv (4/FC)(T\#\_DIV)(PR\# + 1)$$

Una vez obtenido el periodo de la señal, se halla el valor para cargar el PWM en el microcontrolador; luego se encuentra el valor en el que la señal estará activa (on), en el cien por ciento del tiempo. Cuando ya se tiene este valor, se sabrá el rango entre 0 y la magnitud hallada para variar el PWM. Este valor (duty_PWM) está en función de:

- Periodo (T).
- Frecuencia del Cristal (FC).
- Valor del Timer (T#_DIV)

$$duty\_PWM\# \equiv (T)(FC)/T\#\_DIV$$

Una de las ventajas del PWM, es que la señal permanece digital, desde el procesador hasta el sistema controlado; no es necesaria una conversión análoga-digital. Manteniendo la señal digital, los efectos del ruido son minimizados. El ruido solo puede afectar una señal digital, si es suficientemente fuerte para cambiar un 1 lógico a un 0 lógico, o viceversa.

3. DESARROLLO

Una vez abarcados los conceptos fundamentales, continúa el desarrollo, presentando un enfoque general a través de un diagrama de bloques, explicando la funcionalidad de cada elemento junto a su participación dentro de este proyecto.

3.1. Metodología

El sistema tele-operado consiste en un conjunto de elementos desarrollados e implementados para lograr la autonomía necesaria para controlarlo desde una estación remota de trabajo (Master). A continuación se muestra el diagrama de bloques del sistema tele-operado. Los colores de los bloques indican un conjunto especial de componentes que se encargan de realizar operaciones especiales afines. Las uniones entre los bloques, describen el tipo de conexión realizada dando a la presentación del diagrama la claridad necesaria para comprender el sistema tele-operado.

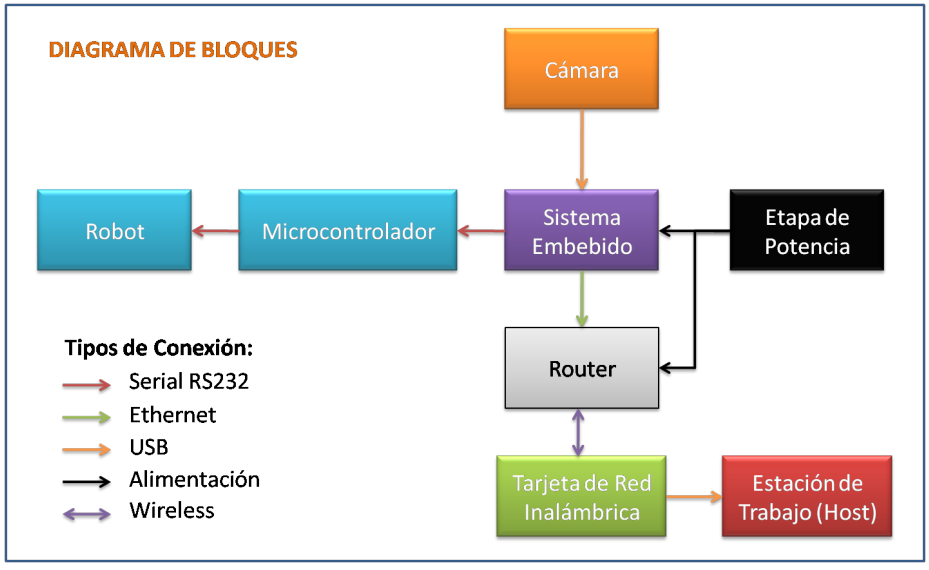


Figura 3.1: Diagrama de Bloques Sistema Teleoperado

En las siguientes secciones se explicarán cada uno de los componentes de acuerdo al conjunto de elementos al cual pertenecen, junto con su funcionalidad, ficha técnica e implementación.

3.1.1. Plataforma Robótica

Microcontrolador

El microncotrolador 16F877A, es el encargado de establecer la comunicación entre el sistema Embebido y el robot. Recibe los datos por el puerto serial (Pines 25 y 26) y de acuerdo a estos, envía señales de control al driver del motor por medio de los dos puertos PWM (Pines 16 y 17) (para mayor información sobre el microncontrolador, véase figura 4.1 en la sección Anexos). A continuación se muestra el diagrama de flujo del programa del microcontrolador.

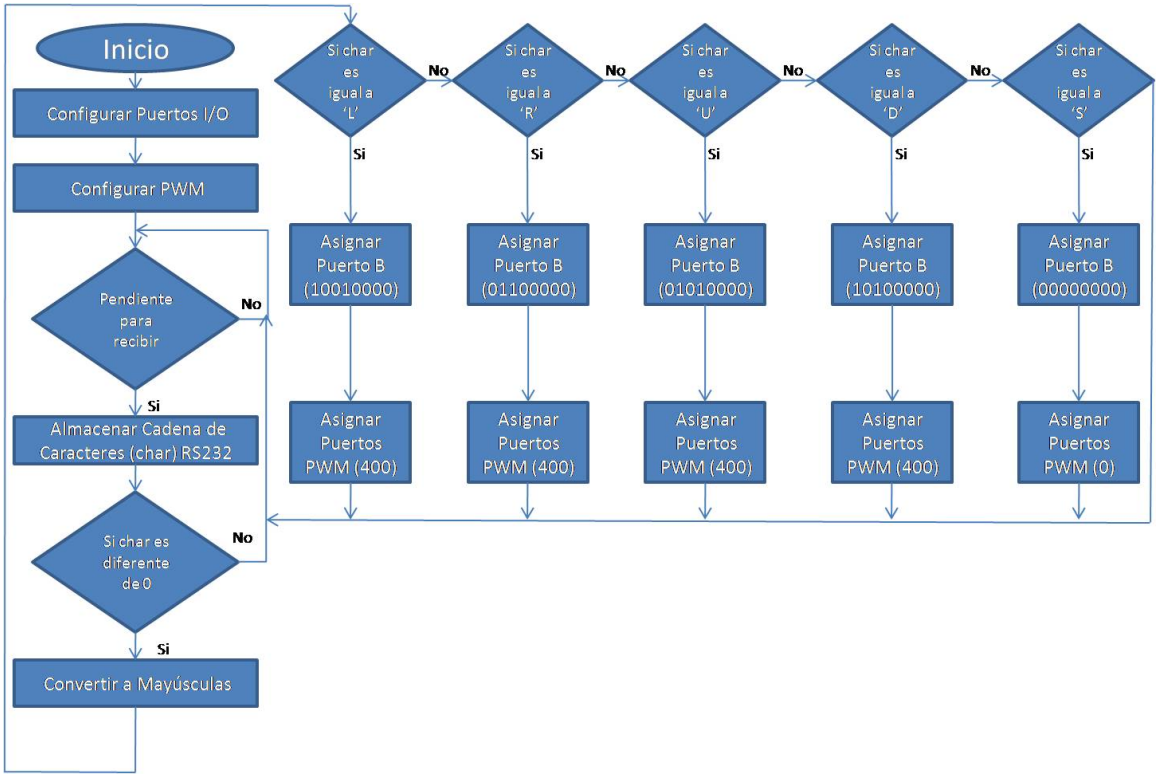


Figura 3.2: Diagrama de Flujo Software Pic

El cálculo de los valores de los registros para la configuración del PWM (teniendo en cuenta que el fabricante del driver que controla los motores requiere una frecuencia menor a 20 KHz); se realiza con un periodo de 10 KHz, utilizando la formula:

$$T \equiv (4/FC)(T\#\_DIV)(PR\# + 1)$$

Despejamos el valor de PR# utilizando el prescaler del timer 2, una frecuencia del cristal equivalente a 4 MHz y el valor de T2_DIV igual a 1:

$$PR2 \equiv (T)(FC)/(4)(T2\_DIV) - 1$$

$$PR2 \equiv (1/10e + 3)(4e + 6)/(4)(1) - 1$$

El resultado de PR2 es 99.

Ahora se calcula el valor en el que el PWM está 100 % encendido:

$$duty\_PWM2 \equiv (T)(FC)/T2\_DIV$$

$$duty\_PWM2 \equiv (1/10e + 3)(4e + 6)/1$$

El valor de duty_PWM2 es 400.

3.1.2. Cámara

El proyecto inicial utilizaba para la captura de video, una cámara CMOS (que es la cámara que viene con el sistema embebido mini2440), pero el uso de una cámara USB ya estaba implementado por medio de la libreria V4L2 (video for Linux 2).
Básicamente existen dos métodos para la transmisión de frames, el primer método, únicamente usa los métodos de lectura y escritura directamente sobre el dispositivo de video (éste método es soportado por muchos tipos de cámaras). El segundo método es el explicado en esta sección y se usa por lo general por su realización en la I/O de video, aunque también es usado cuando el dispositivo de video no soporta la escritura y lectura directa.
Con la funciones de lectura y escritura `read()` y `write()`, específicamente cada frame se copia entre la aplicación y el driver de video como parte de la operación de I/O. Cuando se usa streaming I/O, en cambio, ésta copia no ocurre, la aplicación y el driver de video intercambian buffers a través de punteros. Estos buffers seran mapeados en la dirección del espacio de memoria de la aplicación.
Existen dos tipos diferentes de buffers en el streaming I/O:

- **Buffers de memoria mapeada** (type `V4L2_MEMORY_MMAP`), los cuales son asignados en el espacio del kernel. La aplicación los mapea dentro de esta dirección de memoria con la llamada a la función `mmap()`.

- **Buffer User-space** (`V4L2_MEMORY_USERPTR`), los cuales son asignados en el espacio del usuario. En este caso, ninguna llamada a `mmap()` es requerida, pero el driver puede tener que trabajar de más, para soportar eficientemente este tipo de buffers.

Los dirvers que soportan streaming I/O deben informar a la aplicación de este hecho simplemente por medio de la bandera `V4L2_CAP_STREAMING` en la llamada a la función `vidioc_querycap()`.

La estructura `v4l2_buffer`

Cuando el streaming I/O está activo, los frames se pasan entre la aplicación y el driver en la forma de la estructura `v4l2_buffer`. Esta estructura es amplia y larga de describir. Un buen punto de partida es notar que hay tres estados fundamentales en los cuales un buffer puede estar:

- **En la cola de entrada del driver.** Los buffers son colocados en esta cola por la aplicación, en espera que el driver haga algo útil con ellos. Para un dispositivo de captura, los buffers en la cola de entrada estarán vacios, esperando por el driver para llenarlos con datos de video. Para un dispositivo de salida, estos buffers tendrán datos de los frames de video, que serán enviados al dispositivo.
- **En la cola de salida del driver.** Estos buffers han sido procesados por el driver y están esperando que la aplicación los reclame. Para dispositivos de captura, los buffers de salida tendrán nuevos datos de los frame de video. Para dispositivos de salida, estos buffers estarán vacíos.
- **En ninguna cola.** En este estado, el buffer pertenece al espacio del usuario y normalmente no será tocado por el driver. Este es el único momento en el que la aplicación no deberá hacer nada con el buffer. Se llamará a este estado, el estado "user-space".

La estructura `v4l2_buffer`:

```
struct v4l2_buffer
{
    __u32                   index;
     enum v4l2_buf_type     type;
    __u32                   bytesused;
    __u32                   flags;
    enum v4l2_field         field;
    struct timeval          timestamp;
    struct v4l2_timecode    timecode;
    __u32                   sequence;
    /* memory location */
    enum v4l2_memory        memory;
    union {
        __u32               offset;
        unsigned long       userptr;
    } m;
    __u32                   length;
    __u32                   input;
    __u32                   reserved;
};
```

La variable `index` es un número de secuencia que identifica el buffer; solamente es usado con buffers `V4L2_MEMORY_MMAP`. Como otros objetos que pueden ser enumerados en la interfaz de la V4L2, los buffers mapeados empiezan con el índice 0 y se incrementan secuencialmente apartir de éste valor. La variable `type` describe el tipo de buffer, usualmente `V4L2_BUF_TYPE_VIDEO_CAPTURE` ó `V4L2_BUF_TYPE_VIDEO_OUTPUT`.

El tamaño del buffer esta dado por la variable `length` la cuál está en bytes. El tamaño de la imagen contenida en el buffer se almacena en la variable `bytesused`; obviamente `bytesused` debe ser menor o igual a `length`. Para dispositivos de captura, el driver inicializará la variable `bytesused`. Para dispositivos de salida la aplicación debe inicializar esta variable. La variable `field` describe cual campo de la imagen es almacenado en el buffer.

La variable `timestamp`, para dispositivos de entrada, dice cuando el frame fue capturado. Para dispositivos de salida, el driver no deberá enviar el frame, antes que pase el tiempo que toma almacenar ésta variable. Inicializar ésta variable a cero significa capturar la imagen "tan pronto como sea posible". El driver inicializará ésta variable al tiempo que el primer byte del frame, haya sido transferido al dispositivo - ó lo más aproximado posible.
La variable `timecode` puede ser usada para almacenar códigos de tiempo; útiles para las aplicaciones de edición de video. Si se inicializa a 1, indica 24 frames por segundo, 2 indica 25 frames por segundo, y 3 indica 30 frames por segundo.
El driver mantiene un contador incremental de los frames que se le pasan al dispositivo; almacenando el número de secuencia actual, a medida que cada frame es transferido. Para dispositivos de entrada, la aplicación puede observar esta variable para detectar frames caídos. La variable `memory` es la encargada de decir si el buffer es memory-mapped ó user-space. Para buffers de memory-mapped, `m.offset` describe dónde estará el buffer. La especificación la describe como "la compensación del buffer desde el comienzo del dispositivo de memoria", pero la verdad es que ésta variable, es usada por la aplicación por medio de la función `mmap()` para especificar cual buffer esta siendo mapeado. Para buffers user-space, en cambio, `m.userptr` indica la dirección user-space del buffer.

La vairable `input` puede ser usada para rápidamente cambiar entre las entrada del dispositivo de captura (asumiendo que el dispositivo soporta esto). La variable `reserved` nunca debe ser puesta a cero, por especificación del creador de la librería.

Finalmente, hay varias banderas definidas:

- `V4L2_BUF_FLAG_MAPPED` indica que el buffer ha sido mapeado en el user-space. Solamente es aplicable a los buffer con memory-mapped.
- `V4L2_BUF_FLAG_QUEUED`: El buffer está en la cola de entrada del driver.
- `V4L2_BUF_FLAG_DONE`: El buffer esta en la cola de salida del driver.

- `V4L2_BUF_FLAG_KEYFRAME`: El buffer mantiene un key frame - útil para capturas comprimidas.
- `V4L2_BUF_FLAG_PFRAME` y `V4L2_BUF_FLAG_BFRAME` también usadas en capturas comprimidas; indican frames previstos o diferencias entre los mismos.
- `V4L2_BUF_FLAG_TIMECODE`: Si la variable `timecode` es valida.
- `V4L2_BUF_FLAG_INPUT`: Si la variable `input` es valida.

Configuración del Buffer

Una vez la aplicación que usa el método de streaming I/O, está configurada básicamente, es momento de organizar la forma en que trabajarán los buffers. El primer paso es establecer un conjunto de buffers con la llamada a la función `ioctl()` junto con el parámetro `VIDIOC_REQBUFS`; lo cual realmente es hacer una llamada a la función `vidioc_reqbufs()` del driver por medio de la libreria V4L2: `int (*vidioc_reqbufs) (struct file *file, void *private_data, struct v4l2_requestbuffers *req);` Todo lo necesario para está operación está dentro de la estructura `v4l2_requestbuffers`:

```
struct v4l2_requestbuffers
{
    __u32                   count;
    enum v4l2_buf_type      type;
    enum v4l2_memory        memory;
    __u32                   reserved[2];
};
```

La variable `type` describe el tipo de I/O a realizar, el cual puede ser V4L2_BUF_TYPE_VIDEO_CAPTURE para adquisición de video ó V4L2_BUF_TYPE_VIDEO_OUTPUT para un dispositivo de salida. Hay también otros tipos, pero están más allá del alcance de este documento.

Streaming I/O

Cuando la aplicación obtiene los buffers con VIDIOC_REQBUFS, todos esos buffers están en el user-space, lo cual significa que aún no existen. Antes que la aplicación pueda empezar el streaming I/O, debe poner al menos un buffer dentro de la cola de entrada del driver. Para un dispositivo de salida, estos buffers deben ser llenados con datos validos.
Para almacenar en la cola los buffers, la aplicación debe hacer una llamada a la función `ioctl()` con el parámtro `VIDIOC_QBUF`, la cual, V4L2 mapea en una llamada a la función `vidioc_qbuf()` del driver:

```
int (*vidioc_qbuf) (struct file *file, void *private_data, struct v4l2_buffer *buf);
```

Para buffers de memoria mapeada, sólo las variables de `type` e `index` de `buf` son válidas. El driver puede realizar rutinas de comprobación, poner el buffer en la cola de entrada (con la bandera V4L2_BUF_FLAG_QUEUED) y retornar.

Una vez el streaming I/O comience, el driver tomará los buffers desde la cola de entrada, cuando el dispositivo haya solicitado la transferencia, entonces mueve el buffer a la cola de salida. La banderas del buffer deberán ajustarse de acuerdo a como ocurra la transición. Variables como `sequence` y `timestamp` deberán también ser llenadas en el tiempo de la misma. Eventualmente la aplicación requerirá los buffers en la cola de salida, para retornarlos al estado de user-space. Ese es el trabajo de `VIDIOC_DQBUF`, el cual ocurre con la llamada a la rutina:

```
int (*vidioc_dqbuf) (struct file *file, void *private_data, struct v4l2_buffer *buf);
```

El driver removerá el primer buffer de la cola de salida, almacenando la información relevante en `*buf`. Normalmente, si la cola de salida está vacía, está llamada deberá bloquearla hasta que un buffer esta disponible. Los drivers de la V4L2 estarán esperando para no bloquearlo, aunque, si el dispositivo de video fue abierto con la bandera `O_NONBLOCK`, el driver deberá retornar `-EAGAIN` en el caso de una cola vacía. Es importante notar que el dispositivo debe soportar la llamada a la función `poll()` para el streaming I/O.

El último paso que falta, es el de indicarle actualmente al dispositivo que comience el streaming I/O. Los métodos del driver de Video4Linux2 para ésta tarea son:

```
int (*vidioc_streamon) (struct file *file, void *private_data, enum v4l2_buf_type type);
int (*vidioc_streamoff)(struct file *file, void *private_data, enum v4l2_buf_type type);
```

La llamada a la función `vidioc_streamon()` iniciará el dispositivo después de chequear que la variable `type` tenga sentido. El driver puede, si es necesario, requerir un cierto número de buffers de la cola de entrada antes que el streaming pueda comenzar.

Cuando la aplicación termina, deberá generar una llamada a la función `vidioc_streamoff()`, la cual debe detener el dispositivo. El driver también deberá remover todos los buffers de la cola de entrada y salida, dejándolos en el estado user-space. Por supuesto el driver debe ser preparado para simplemente cerrar el dispositivo sin detener el streaming primero. Para más información de la libreria V4L2, ver [5].

3.1.3. Sistema Embebido

Los sistemas Linux están hechos de muchos componentes, definiendo la arquitectura principal de un sistema Linux, cómo un gran conjunto de elementos, con el fin de comunicar el usuario con el hardware a través del kernel, para obtener el mayor rendimiento. El hardware debe tener ciertos características generales para ejecutarse en un sistema Linux:

- Linux normalmente requiere de al menos una CPU de 32-bit conteniendo una unidad de manejo de memoria.
- Una cantidad suficiente de memoria RAM (mínimo 32 MB) debe estar disponible para el sistema.
- Mínimas capacidades de I/O son necesarias, si cualquier desarrollo se quiere implementar en el target por razonables de facilidad y depuración.
- El kernel debe ser capaz de cargar archivos del sistema raíz a través de algún almacenamiento permanente, o acceso a través de la red.

Inmediatamente arriba del hardware, se encuentra el kernel, que es el componente principal del sistema operativo. Su propósito es el de manejar el hardware en una forma coherente, mientras provee abstracciones familiares de alto nivel, al nivel del usuario por medio del software. Como ocurre en otros kernels de tipo Unix, Linux maneja los dispositivos, acceso de entrada y salida de datos (I/O), control de la programación de procesos, aplica el compartimiento de memoria, maneja las señales de distribución y otro tipo de tareas administrativas. Es por lo general de esperarse que las aplicaciones que usan los APIs dados por el kernel, serán portables entre diferentes arquitecturas soportadas por éste kernel sin ningún cambio. Éste es usualmente el caso con Linux, como se puede ver en la estructura general y uniforme disponible en todas las arquitecturas que soporta. Dentro del kernel hay dos categorías generales, mostradas como capas de servicio existentes, para la funcionalidad requerida por las aplicaciones. Las interfaces de bajo nivel son específicas, para la configuración del hardware; en las cuales el kernel trabajaría para el control directo de los recursos del mismo, usando APIs independientes de la estructura del hardware. Manejar registros ó páginas de memoria, ocurre de manera diferente en un sistema PowerPc que en un sistema ARM (Advanced RISC Machine); pero será accesible a través de un API común a los componente de alto nivel del kernel (con algunas excepciones); típicamente: los servicios de bajo nivel manejados por las operaciones específicas de la CPU, operaciones de memoria específicas de la arquitectura y las interfaces básicas de los dispositivos. Estos son, abstraídos a un código de alto nivel a través de encabezados, macros y funciones de intercambio.

Arriba de los servicios de bajo nivel proveídos por el kernel, los componente de alto nivel proveen las abstracciones comunes a todos los sistemas Unix, incluyendo procesos, archivos, sockets y señales. Desde que las APIs de bajo nivel provistas por el kernel, son comunes a través de diferentes arquitecturas; los códigos para las abstracciones de alto nivel, son casi constantes; a excepción de

la arquitectura subyacente. Existen otras excepciones donde el código de alto nivel del kernel, incluirá casos especiales de diferentes funciones para ciertos tipos de arquitecturas.

Entre estos dos niveles de abstracción, el kernel algunas veces necesita, lo que podría llamarse la interpretación de los componentes, para entender e interactuar con datos estructurados de o hacia ciertos dispositivos. Tipos de sistemas de archivos y protocolos de red son los ejemplos primarios de datos estructurados que el kernel necesita para entender e interactuar en orden de proveer acceso a los datos que van o vienen de esas fuentes. Los dispositivos de disco han sido y son los principales medios de almacenamiento de datos. En el mundo embebido, los dispositivos flash tienden a proveer la misma funcionalidad, inclusive usando interfaces compatibles en muchos casos. Todavía dispositivos de disco y muchos otros dispositivos de almacenamiento, para esa tarea, contienen pequeñas estructuras. Su contenido debe ser accesible por referencia al sector apropiado en cierto disco, pero este nivel de organización, es bastante insuficiente para acomodar, el siempre cambiante contenido de los archivos y directorios. El acceso al nivel de archivos, es logrado usando una organización especial de los datos en el disco, dónde la información en los archivos y directorios es almacenada en una forma particular (ext3), para ser reconocido cuando sea leído de nuevo. Esto es la forma en que los sistemas de archivos están implementados.

Durante la evolución de los sistemas operativos, muchos tipos de sistemas de archivos incompatibles, han visto la luz del día. Para acomodar dichos sistemas de archivos existentes, como se haría con sistemas nuevos en desarrollo; el kernel tiene un número de motores de sistemas de archivos, que pueden reconocer la estructura de un disco en particular y recuperar o agregar archivos y directorios a partir de dicha estructura. Los motores proveen todos, la misma API a las capas superiores del kernel, a través de la abstracción del Linux Virtual File System (VFS); de esta manera pueden acceder a los diferentes tipos de sistemas de archivos. La misma API es provista a la capa del sistema de archivos virtual del kernel, por ejemplo, por el sistema de archivos FAT y el sistema de archivos ext3; pero las operaciones que el sistema de archivos conduce en el bloque del driver del dispositivo, va de acuerdo a las estructuras respectivas usadas por FAT y ext3, para almacenar los datos en el disco (las cuales son muy diferentes entre si). Durante esta operación normal, el kernel requiere que al menos un sistema de archivos esté bien estructurado: el sistema de archivos root. Desde este sistema de archivos, el kernel carga la primera aplicación que corre en el sistema. Este también normalmente, depende de éste sistema de archivos por ciertas operaciones avanzadas, como por ejemplo cargar módulos y proveer cada proceso con un directorio que trabaje (aunque estas actividades pueden tomar lugar en otros sistemas de archivos, montados entre el el conjunto de sistemas, que comienzan con el sistema de archivos root). El sistema de archivos root puede ser almacenado y operado desde un dispositivo de almacenamiento hardware, o cargado en la RAM durante el inicio del sistema y operado desde allí
Los servicios exportados por el kernel, a menudo no encajan para ser manejados directamente por

las aplicaciones. Más bien, la dependencia de las aplicaciones recae en librerías y especial daemons[1] del sistema, para proveer APIs familiares y servicios abstractos que interactuan con el kernel, en las aplicaciones, para obtener la funcionalidad deseada. La principal librería usada en la mayoría de aplicaciones Linux, es la librería C GNU glibc. Para los sistemas embebidos, substitutos pueden usarse para compensar el principal defecto que tiene la librería glibc: su tamaño. Librerías a parte de la glibc, como por ejemplo Qt, XML, MD5; proveen varias APIs útiles y funcionales, que sirven para todo tipo de propósitos. Mientras tanto, importantes procesos del sistema ("daemons") proveen servicios explotados por las aplicaciones. Por ejemplo, el dispositivo administrador de sistemas de archivos udev, en el directorio /dev. Como cuando un dispositivo de almacenamiento USB es agregado y removido de un sistema.

Las librerías son enlazadas típicamente de manera dinámica por las aplicaciones, lo cual permite que dichas librerías, no sean parte del binario de la aplicación, sino que más bien, sean cargadas en el espacio de memoria de la aplicación, durante el inició de la misma. Esto permite a muchas aplicaciones, el uso de la misma instancia de la librería, en vez de estar haciendo copias de la misma, para cada aplicación (lo cual es una gran forma de aumentar la efectividad de las aplicaciones). La librería C, que se encuentra en el sistema de archivos, es cargada únicamente una vez en la RAM del sistema, y ésta misma copia es usualmente compartida entre todas las aplicaciones que usan la librería. Pero en algunas situaciones, que incluyen sistemas embebidos, enlaces estáticos (por lo cual las librerías son parte de los binarios de las aplicaciones) es preferible a los enlaces dinámicos. Cuando solamente una parte de la librería es usada por una o dos aplicaciones, los enlaces estáticos ayudan a evitar la necesidad de almacenar la librería entera en el dispositivo de almacenamiento del sistema embebido. Este problema se vuelve más complejo cuando se enlazan aplicaciones propietarias con ciertas librerías cubiertas solo por una licencia estricta GPL en vez de una LGPL.

Linux trabaja en una gran cantidad de arquitecuras, pero no todas estas arquitecturas son en realidad usadas en configuraciones embebidas, como se mencionó anteriormente; una rápida hojeada al código fuente del subdirectorio principal del kernel de Linux, muestra 24 arquitecturas, las cuales soporta junto a otras arquitecturas, mentenidas por otros desarrolladores. En esas 24 arquitecturas, principalmente están (en order alfabético) ARM, AVR32, Intel x86, M32R, MIPS, Motorola 68000, PowerPC, and Super-H. Sólo se describirá la arquitectura usada en éste proyecto, la ARM.

ARM

ARM (Advance RISC Machine), es una familia de procesadores mantenida y promovida por ARM Holdings Ltd. En contraste a otras fabricantes de chips como IBM, Freescale e Intel; ARM Holdings no fabrica su propio procesador; más bien, diseña el núcleo de la CPU, para sus clientes basado en el núcleo ARM, permitiendo a sus clientes fabricar el chip donde crean que encaja en el diseño. Esto

[1] La traducción literal de esta palabra no puede tomarse para el entendimiento del kernel en Linux, sino más bien se entiende como un proceso en segundo plano.

ofrece varias ventajas a todos los involucrados, pero crea cierto tipo de confusión para los desarrolladores que emplean estas arquitectura por primera vez. De todas maneras, hay una característica unificadora que es importante recordar: Todo los procesadores ARM comparten el mismo conjunto de instrucciones, lo cual hace que todas las variantes, soporten una revisión particular del conjunto de instrucciones ARM, compatible completamente con el software. Esto no significa que todas las CPUs ARM y tarjetas puedan ser programadas y preparadas en la misma forma; sólo que el lenguaje assembler y el código de los binarios resultantes, son idénticos para todos los procesadores ARM que reúnen cierta revisión similar de la arquitectura. Revisiones de la arquitectura actual en uso incluyen ARMv4T (introduciendo el conjutno de instrucciones Thumb) ARMv5TE (las bases para las partes "Xscale"), ARMv6 (dispositivos basados en TI-OMAP de Nokia y similares como el ARMv6KZ, del cual se base el iPhone de Apple) y el ARMv7. Cada una de estas revisiones de arquitectura, mejora capacidades dentro de la "familia" de procesadores ARM ARM7TDMI, ARM9E, Xscale, ARM11 y demás. El nombramiento, se vuelve más complejo en realidad, porque las revisiones de las arquitecturas incluye letras ó banderas indicando la forma como se agregan más capacidades. Por ejemplo, el ARMv4T introducía la versión condensada del conjunto de instrucciones "Thumb" que apuntaba al menor uso de memoria para las instrucciones de almacenamiento, mientras mantenía un adecuado nivel de ejecución. También están los procesadores ARM con mejoras DSP en ejecucción ("E"), soporte Java bytecode ("J"), capacidades virtuales y un gran número de otras banderas.
Actualmente las CPUs ARM son fabricadas por Marvell (formalmente Intel, bajo la marca "Xscale"), Toshiba, Samsung, entre otros. La arquitectura ARM es muy popular en muchos campos de aplicación, desde teléfonos celulares y PDAs hasta equipos de red. Existen muchos distribuidores que proveen productos y servicios para todo el mundo. Hasta agosto de 2008 Linux soportaba 40 distintos tipos de CPUs ARM y un total de 1,832 diferentes tipos de máquinas. La lista completa y al día de los sistemas que soportan ARM y sus detalles, se encuentra en http://www.arm.linux.org.uk/developer/machines. Suficiente es decir que Linux soporta gran cantidad de CPUs y boards, como la CPUs de la Texas Instrument OMAP usada por Nokia en sus bien conocidas Linux Internet Tablets y el procesador de red IXP usado en diferentes dispositivos de red. Para más información con respecto a la arquitectura ARM en sí y su conjunto de instrucciones, consulte http://www.arm.com. Información de las otras arquitecturas puede ser encontrada en [13].

El sistema embebido mini2440 es el núcleo del sistema tele-operado, puesto que es el encargado de procesar la información de la imagen y transferirla a la estación de trabajo. A su vez, se encarga de recibir los comandos enviados por la estación de trabajo, traducirlos y enviarlos al microcontrolador. El sistema embebido mini2440 tiene la siguientes características:

- Dimension: 100 x 100 mm
- Masa: 280 gramos.
- CPU: 400 MHz Samsung S3C2440A ARM920T (Max freq. 533 MHz)
- RAM: 64 MB SDRAM, 32 bit 100 MHz Bus

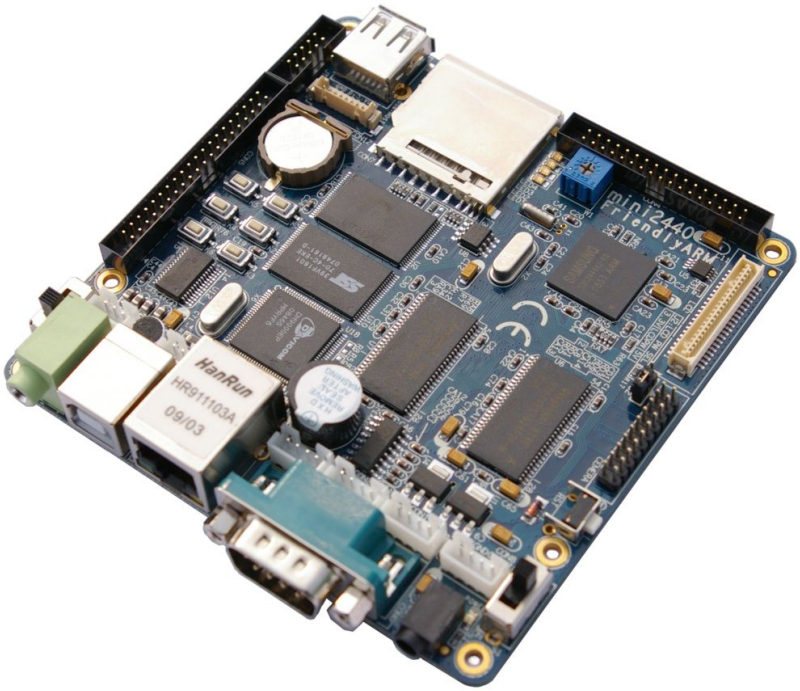


Figura 3.3: Tarjeta Mini2440

- Flash: 64 MB NAND Flash and 2 MB NOR Flash with BIOS
- EEPROM: 1024 Byte 24C08 (I2C)
- Ext. Memory: SD-Card socket
- Serial Ports: 1x DB9 connector (RS232), total: 3x serial ports on the PCB
- USB: 1x USB Host, 1x USB Device
- Audio Output: 3.5 mm stereo jack
- Audio Input: on PCB + condenser microphone
- Ethernet: RJ-45 10/100M (DM9000)
- RTC: Built in Real Time Clock with battery
- Beeper: On board PWM buzzer
- Camera: 20 pin Camera interface
- LCD Interface:
  - STN Displays:
    - 4 bit dual scan, 4 bit single scan or 8 bit single scan display type

        - monochrome, 4 gray levels, 16 gray levels, 256 colors or 4096 colors
        - Max: 1024x768, 4096 colors
    - TFT Displays:
        - 1, 2, 4 or 8 bpp palletized color displays
        - 16 or 24 bpp non-palletized true-color displays
        - Max: 1024x768, 64k colors
- Touch Panel: 4 wire resistive
- User Inputs: 6x push buttons and A/D pot
- User Outputs: 4x LEDs
- Expansion: 40 pin System Bus, 34 pin GPIO (2.0mm)
- Debug: 10 pin JTAG (2.0mm)
- Power Connector: 5V with switch and LED
- OS Support
    - Linux 2.6
    - Windows CE 5

Transmisión de datos

La transmisión de datos en el sistema embebido ocurre de dos maneras y en dos direcciones específicas. El sistema embebido se comunica con el router por medio de Ethernet. La libreria gráfica Qtopia provee clases para el manejo de sockets y comunicación por medio de TCP. El sistema embebido comienza su funcionamiento al crear dos servidores, el servidor TCP y el servidor Serial.
El servidor TCP es el encargado de recibir conexiones realizadas por cualquier tipo de `host` que desee conectarse para recibir las imágenes de la cámara. El hecho que el sistema embebido pueda recibir más de un cliente le permite al robot móvil ser controlado y/o vigilado por diferentes usuarios, potencializando la aplicabilidad del mismo.
Una vez establecida la conexión del servidor TCP con el cliente, el sistema embebido creará el servidor serial cuya función es la de encargarse de traducir las señales de control que envia el cliente a lenguaje que el microcontrolador pueda entender.

3.1.4. Etapa de Potencia

La autonomía del sistema embebido radica en el uso de una bateria de 12 voltios a 5 amperios, de acuerdo al siguiente diagrama:

El sistema embebido requiere de 5 voltios a 2 amperios y el router requiere 5 voltios a 1.2 amperios. Debido al alto valor de la corriente necesaria por el sistema en general, fue necesario agregar dos reguladores de voltaje de 5 voltios a 3 amperios, asegurando la protección de los componentes.

3.1.5. Interfaz de Usuario del `Master`

El `Master` es la estación de trabajo encargada de recibir las imágenes del sistema embebido para que el usuario pueda apreciar el trayecto del robot móvil (figura 4.3). A su vez se encarga de transmitir las señales de control junto con el valor del PWM a cada uno de los motores. A continuación aparace la interfaz gráfica del `Master`:

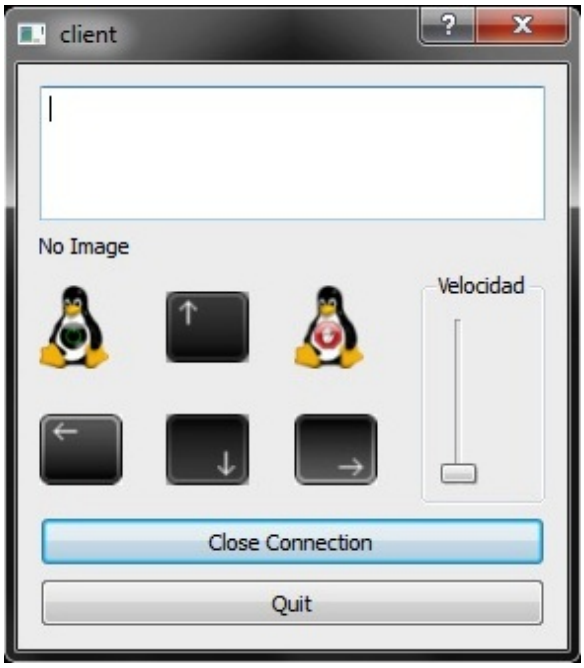


Figura 3.4: Interfaz Gráfica Usuario

- Cuadro de mensaje (QTextEdit) donde se le informa al usuario si la conexión con los servidores del sistema embebido tuvo lugar (`Connected to Image Server` y `Connected to Serial Server`) o no (`Error, TCPError: Connection timed out`).
- Un QLabel encargado de mostrar la imagen de la cámara transmitida por el sistema embebido.
- Botón (QPushButton) encargado de realizar el inicio de la transmisión TCP (icono de color verde).
- Botón (QPushButton) encargado de detener el robot móvil.
- Cuatro botones (ButtonLabel) encargado del control de movimiento del robot móvil.

- Una barra (QSlider) encargada del valor PWM enviado a cada motor.
- Botón (QPushButton) encargado de cerrar la conexión TCP con el sistema embebido.
- Botón (QPushButton) encargado de salir de la aplicación.

3.1.6. Señales de Control del Sistema

EL proceso general para la transmisión de datos desde el usuario hasta el robot móvil, ocurre de acuerdo al siguiente diagrama, que representa una iteración del algoritmo general.

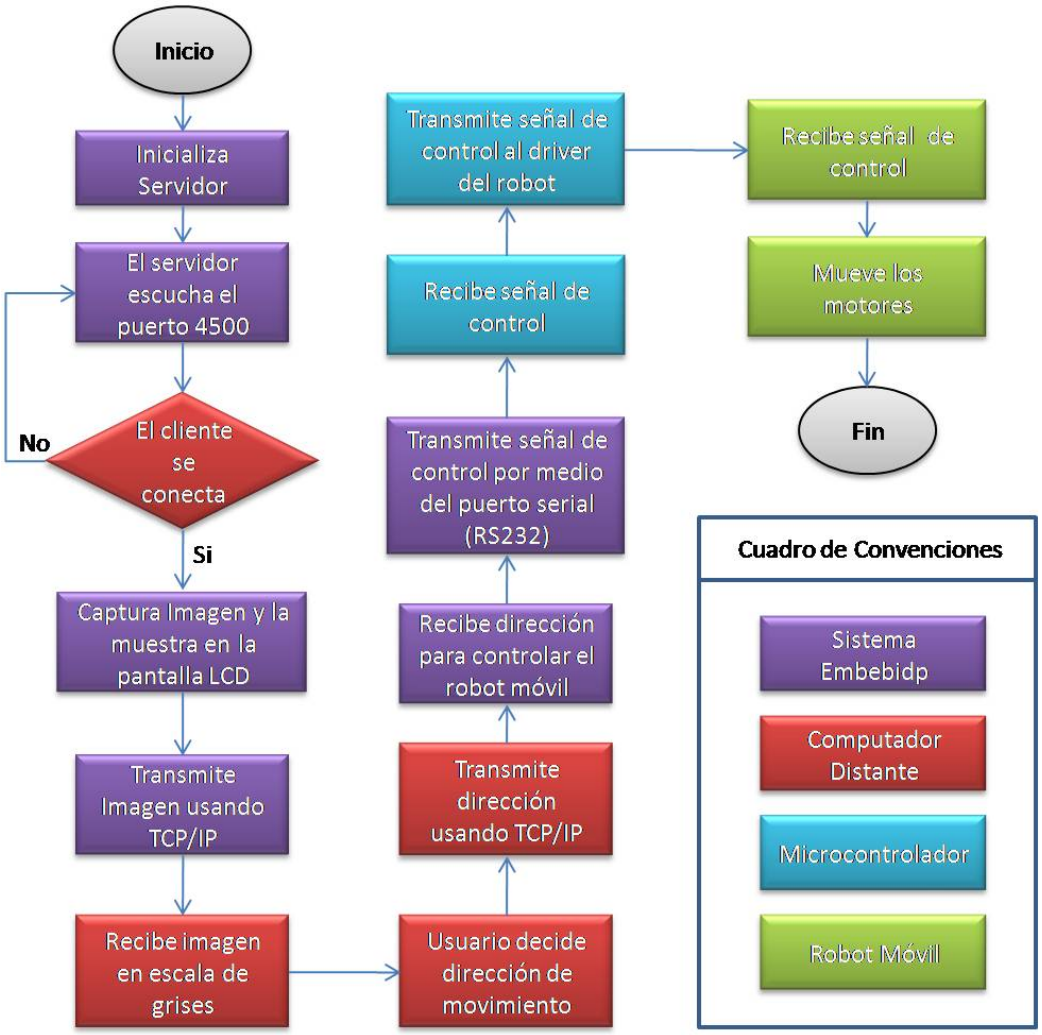


Figura 3.5: Iteración del algoritmo general de transmisión de datos

El usuario después de ver la imagen, determina una dirección para el movimiento deseado del robot móvil. Cómo se puede ver a partir de la figura anterior, el usuario recibirá constantemente (mientras

este conectado con el sistema embebido) la imagen y la señal de control enviada será recibido por el servidor (mini2440) y de ahí, se comunica con el microcontrolador. A continuación aparece la tabla con las señales de control, enviadas al PIC:

	Señal de Control Enviada		
Dirección	char[0]	char[1]	char[2]
Izquierda	'L'	'E'	PWM (400)
Derecha	'R'	'I'	PWM (400)
Adelante	'F'	'O'	PWM (400)
Atrás	'B'	'A'	PWM (400)
Stop	'S'	'T'	PWM (0)

Figura 3.6: Tabla de señales de control enviadas por el Master

Para el control de los motores, se utilizó la técnica de control On/Off mediante el uso de los valores de los PWM del microcontrolador. Para el control de los motores, se utilizó la tabla dada por el fabricante del driver, utilizando el puerto B del microcontrolador, para la comunicación de las señales de control.

M3	M2	M1	ACTION
1	0	0	Brake
1	0	1	Forward
1	1	0	Reverse
1	1	1	Brake
0	X	X	Coast

Figura 3.7: Conjunto de instrucciones para el controlador del robot móvil

3.2. Pruebas de Desempeño del Sistema

3.2.1. Prueba Tarjeta Wireless

- *Prueba:* Se conectó al sistema embebido, una tarjeta wireless, para probar la conexión entre las estaciones. Para comprobar el enlace, se envió un ping, desde la estación Master al Slave; separados por una distancia aproximada a un metro, con línea de vista entre el Router y la tarjeta inalámbrica.

- *Resultado:* Al conectar la tarjeta al sistema embebido, el OS reconoció el chip vendor del módulo wireless, pero no activó el módulo `wlan0`, necesario para las conexiones inalámbricas. Se trató de activar el módulo `wireless extensions` encargado del manejo del `wlan0`; para esto, se recompiló el kernel del sistema embebido sin resultados satisfactorios, puesto que la imagen recompilada creaba errores en tiempo de ejecución del sistema embebido (kernel panic). Se contactó a los fabricantes de la board mini2440 para conocer la solución a dicho problema, pero respondieron que el módulo `wireless extensions` no era soportado actualmente por el

sistema operativo de la mini2440.
Para la implementación de la comunicación entre las estaciones, se procedió entonces a colocar un router wireless (DLink), en la estación Slave; para que la estación Master se comunicará a través de él. Esta prueba resultó satisfactoria. Para probar la velocidad de transmisión del sistema, se envió un ping, con paquetes de 32 bytes demorándose cada uno, un tiempo menor a 2 milisegundos.

3.2.2. Programación Cámara en Sistema Embebido

- *Prueba:* Se conectó una cámara USB, al sistema embebido, con el fin de implementar la adquisición de video en tiempo real. Se programó cada una de las clases necesarias en lenguaje C++, usando la librería V4L2, comprobando la correcta apertura del dispositivo, y el tipo de adquisición soportado por el mismo.

- *Resultado:* Se utilizó una cámara USB Genius Slim; se prosiguió a comprobar si el kernel tenía pre-compilado el driver del dispositivo lo cual ocurrió satisfactoriamente. Se procedió a la implementación de las clases para el manejo de la adquisición de la imagen. La apertura del descriptor de archivo asociado al dispositivo se ejecutó correctamente, debido a la comprobación de las banderas en el uso de las clases de la V4L2. Se comprobó si el tipo de apertura era *read-write* o tipo *streaming*, el dispositivo usado soportaba el tipo streaming únicamente. Se hizo la implementación de las clases encargadas de obtener la imagen del dispositivo para ser mostrada en el programa del sistema embebido.
  El resultado final fue la correcta visualización del video en tiempo real, dentro del sistema embebido.

3.2.3. Programación Sockets

- *Prueba:* Se utilizó la librería Qt, para la implementación de una comunicación, orientada a conexión, como la TCP; por medio de las clases QSocket y QServer en la estación Slave, y QTcpSocket, en la estación Master. Se programó la estación Slave (remota), y una estación Master para comunicarse entre si.

- *Resultado:* El desarrollo del programa de la estación Slave se realizó satisfactoriamente, permitiendo la creación y uso de un servidor que es capaz de comunicarse con cualquier cliente que se conecte a dicha dirección y puerto. También se implementó la estación Master, que fue capaz de conectarse con el servidor. Se probó dicha conexión con el envió de caracteres de un solo byte de tamaño, para luego incrementar dicho tamaño hasta enviar frases de una estación a otra. En dicha prueba se concluyó que el tamaño del buffer de envió era máximo de 1300 bytes, puesto que al enviar un buffer de mayor cantidad, se perdian datos.

3.2.4. Programación Transmisión de Video

- *Prueba:* Se realizó la unión de los programas implementados anteriormente, en la estación Master y Slave, con el fin de lograr la transmisión de video, adquirido en el sistema embebido, a la estación Master.

- *Resultado:* Para la transmisión de la imagen, primero se codificó la imagen adquirida a escala de grises, con el fin de reducir el tamaño de la imagen a ser enviada. Se trató de enviar la imagen completa (76800 bytes), pero la clase QSocket sólo permitió el envió de paquetes de tamaño menores a 1300 bytes, como se estableció en la prueba de comunicación. Por tal motivo, se dividió la imagen en 64 paquetes, cada uno de 1200 bytes. La transmisión se realizó de manera sincrónica, para el envió correcto de la imagen. Cuando llegaban los 64 paquetes a la estación Master, se mostró en la interfaz de usuario y se procedió a tomar otra imagen en la estación Slave.
  La transmisión ocurrió sin perdida de datos y con una velocidad de 2.5 fps, pero en la prueba se observó un retraso de aproximadamente 1.5 segundos; esto debido a la capacidad del buffer de envió (1300 bytes), lo cual no permitió transmitir toda la imagen en un sólo paquete.

3.2.5. Programación RS232

- *Prueba:* Se implementó un software capaz de comunicarse con otros dispositivos serialmente, a través del puerto RS232. Se desarrolló un algoritmo, necesario para el correcto entendimiento entre el sistema embebido, y el robot móvil. Esto con el fin de realizar el control de la estación Slave, desde la estación Master.

- *Resultado:* Se implementaron y desarrollaron, las clases para la apertura del descriptor de archivo `ttyS0`, encargado del manejo del puerto serial RS232. Se probó la comunicación con el microcontrolador 16F877A, enviando caracteres de control de un byte, de manera satisfactoria. Luego se implementó la comunicación entre el microcontrolador y el robot móvil, obteniendo como resultado, el movimiento de los motores del mismo.

3.2.6. Prueba del sistema

- *Prueba:* Fue realizada en el laboratorio de robótica de la Universidad Militar Nueva Granada. El ambiente al que fue expuesto, constó de un conjunto variado de obstáculos, no solo para la comunicación entre las estaciones, debido a que las paredes del laboratorio irrumpían la línea de visión entre el Master y el Slave; sino también, para el libre movimiento del robot. Ya que el robot enfrentó elementos móviles aleatorios, como por ejemplo, personas caminando a través del pasillo. A la vez, evadió objetos inmóviles como por ejemplo, lockers, puertas, canecas y muros.

- *Resultado:* El robot móvil evadió de manera eficiente todos los obstáculos presentados en el recorrido. La transmisión entre el Slave y el Master fue probada en un rango aproximado de 15 metros, funcionando dentro de las especificaciones técnicas del Router. Uno de los aspectos a tener en cuenta, es el retraso del video, el cuál es de aproximadamente 1.5 segundos; esto se remite a la prueba de streaming anteriormente mencionada, en la cual, se explica el porque de este retraso.

3.3. Desempeño del Robot Móvil

El robot móvil tiene como tipo de locomoción el sistema de orugas. Las medidas del robot son:

- Ancho: 0.43 m.
- Largo: 0.87 m.
- Altura: 0.37 m.

El robot consta de dos motores de corriente continua de gran potencia, que son controlados por el driver `RL AMC 50NP04`; el cual es un avanzado Puente-H basado en MOSFETs de alto rendimiento. El robot móvil fue desarrollado para el grupo de investigación DaVinci, por parte del estudiante de la facultad de Ingeniería Mecatrónica de la Universidad Militar Nueva Granada Juan Camilo Hernandez. El cual implemento y desarrollo la plataforma móvil sobre la cual está colocado el sistema embebdio.

La velocidad lineal del móvil fue aproximadamente 25 cm/s(datos administrados por el diseñador del robot). El alcance de control del móvil es de aproximadamente 50 metros, esto debido a las especificaciones dadas por el fabricante del Router usado.

Figura 3.8: Robot Móvil

3.4. Sugerencias y Recomendaciones

Para la optimización de la adquisición de video en tiempo real, el sistema puede ser mejorado, mediante la implementación de una librería más actualizada, que la utilizada en el desarrollo de la

aplicación. Un ejemplo de esta librería, sería la Qt-Extended 4.5, la cual permite enviar una imagen completa en un solo paquete por medio del protocolo TCP. En el presente proyecto no se utilizó, debido a que hasta la fecha no hay soporte para el sistema embebido mini2440.

El uso del sistema embebido, permite la posible implementación de un sistema autónomo mediante procesos como visión estereoscópica, redes neuronales, inteligencia artificial; ya que la aplicación permitiría el procesamiento a partir de la imagen adquirida por la cámara dejando al sistema embebido la capacidad de procesamiento suficiente para el desarrollo de diferentes y más avanzados algoritmos.

La posibilidad de controlar el sistema desde diferentes estaciones Master en forma paralela, amplia el rango de aplicaciones para este tipo de sistemas, permitiendo la múltiple supervisión de diferentes procesos en la estación Slave.

El uso de una antena de mayor ganancia mejoraría el rango de tele-operación del sistema y su aplicabilidad en diferentes campos de trabajo.

Teniendo en cuenta todos las sugerencias presentadas anteriormente, los posibles trabajos futuros a realizar aplicando los conocimientos adquiridos, serían:

- Robots autónomos.
- Robots cooperativos.
- Tele-cirugía.
- Tele-medicina.
- Supervisión de ambientes aislados.
- Broadcasting.
- Implementación de aulas virtuales.
- Desactivación de minas.

4. CONCLUSIONES

Se implementó satisfactoriamente el sistema para la tele-operación de un robot móvil, gracias al uso de un sistema embebido, como unidad central de procesamiento. La importancia de este, radica en el incremento del número posible de aplicaciones y la autonomía de la que se podría dotar a la estación Slave, para el análisis de información en el ambiente de desarrollo.

Se dotó al sistema de realimentación visual, debido al uso del sistema embebido que puede adquirir imágenes por medio de una cámara USB, para luego ser enviadas a la estación Master a través de redes Wireless. Esto permite suministrar al sistema la capacidad de disminuir el error en la supervisión del ambiente analizado.

El uso de la librería multiplataforma Qt, para el desarrollo e implementación del software necesario para la adquisición, compresión y transmisión de imágenes entre el robot móvil y la interfaz de usuario; permite que la estación Master funcione en plataformas Windows, Linux y Mac. El uso de esta librería posee la gran ventaja de trabajar bajo la licencía GPL, permitiendo el libre desarollo de las aplicaciones open source; sin embargo también es posible comprar las licencias, para que el código fuente de las aplicaciones, permanezcan ocultas a los demás desarrolladores.

La aplicación del protocolo TCP/IP y wireless para la transmisión de datos entre el móvil y la interfaz de usuario, permite mayor alcance en el control de la tele-operación, así como la integridad de los paquetes enviados desde una estación a otra; contrario a técnicas usadas en las comunicaciones tele-operadas, no orientadas a la conexión, sin tener en cuenta la reducción del error; como aplicaciones anteriormente vistas en la universidad. Algunas de éstas, requieren el conocimiento previo del terreno, también el uso de técnicas como la radio-frecuencia restringe el rango de trabajo de dichos sistemas, por el alcance y las capacidades físicas del mismo.

La utilidad del trabajo realizado en este opción de grado, no se restringe al uso de un único robot móvil, sino a un amplio rango de aplicaciones; ya que es adaptable a cualquier plataforma en la que sea necesaria el uso de técnicas como la tele-presencia, como por ejemplo, la educación a distancia, la supervisión médica en cirugías; y la tele-operación, como por ejemplo, exploración de terrenos peligrosos, desactivación de minas, entre otros.

El desarrollo de proyectos que usan herramientas como los sistemas embebidos, con comunicación

orientada a conexión y la posibilidad de que varias estaciones Slave se conecten a una Master, ó viceversa, permite el avance en áreas del conocimiento de robótica, cómo la implementación de algoritmos más complejos, un ejemplo de esto, es la creación de robots cooperativos, ó sistemas que sean necesarios supervisar desde varias estaciones Master.

El uso de un sistema embebido como el núcleo principal de la aplicación, permite tener un amplio rango de aplicaciones. Se pueden implementar robots autónomos ó algoritmos más avanzados que no podrían desarrollarse con el uso de microcontroladores únicamente. Todo esto, permitirá el avance tecnológico e investigativo de los desarrollos presentados en la Universidad, al mostrar un precedente en el uso de nuevas y más avanzadas herramientas.

5. ANEXOS

Toda la información complementaria a la opción de grado, se presenta a continuación, junto a las hojas de datos de los componentes usados, código fuente y manuales de usuario.

5.1. Manual de Usuario Configuración Sistema Embebido

Para poder crear aplicaciones en el sistema embebido (mini2440) es necesario realizarlos en una estación de trabajo Linux (los programas creados en este documento son solamente para Linux), luego el programa debe ser compilado en un cross-compiler cuyo programa (en realidad crea un binario) resultado esté destinado para ser ejecutado en un sistema embebido con un procesador de arquitectura ARM. Este trabajo puede llevar un buen tiempo si la persona no esta familiarizada con el entorno de trabajo, pero a continuación se presentán los pasos necesarios para configurar debidamente el entorno de trabajo en la estación de trabajo, para crear aplicaciones que funcionen en la mini2440.

Instalar el sistema operativo Linux con Fedora 10

Para instalar este sistema operativo[1], sólo debe tenerse a mano una versión del CD o el DVD y luego seguir los pasos de la instalación teniendo en cuenta el tamaño con el cual se va a trabajar el sistema operativo. Por lo general, un espacio de 10 GB es suficiente para instalaro junto con los paquetes necesarios.

Descargar paquetes de instalación del Compilador

Descargue (`http://www.arm123.com.cn/linux/arm-linux-gcc-4.3.2.tgz`) el compilador. Este es el Cross-Compiler (el compilador encargado de generar los binarios para el sistema embebido desde la estación de trabajo) que de ahora en adelante se llamará arm-linux-gcc. La sintaxsis de la estructura de nombramiento se divide en tres partes; la primera indica el sistema-objetivo al cual se crean los binarios (en este caso para un procesador con arquitectura ARM), la segunda parte indica la estación de trabajo y la tercera parte indica el compilador usado (en este caso el gcc).

[1] Para el momento en que se creó este documento (Julio 2009), el proyecto de grado se desarrollo con el uso de esta distribución. También se sabe que la creación de aplicaciones para la mini2440 funciona en la distribución Fedora 11.

Luego descomprima el archivo en la siguiente locación (si no tiene estas carpetas creadas en Fedora, créelas): `/opt/FriendlyARM/mini2440`.

Descargue la libreria en el sigiuente enlace, `http://www.arm123.com.cn/linux/arm-qtopia.tgz`. Esta es la librería Qtopia, con la cual se crean las aplicaciones en el sistema embebido. Si el lector no está familiarizado con esta librería, se recomienda la lectura de [8] [11] [14]. La creación de aplicaciones con Qtopia no varía mucho de las de Qt, solamente las clases a usar. Se recomienda la lectura del siguiente enlace `http://doc.trolltech.com/qtopia2.2/html/classes.html` en donde se encontrarán las clases para Qtopia 2.2. El archivo descargado debe ser descomprimido en la siguiente locación: `/opt/FriendlyARM/mini2440`.

Agregar locaciones en la variable del sistema PATH

Luego deben ser agregadas en la variable del sistema PATH algunas locaciones (es necesario que lo realice en súper-usuario también):

- `/opt/FriendlyARM/mini2440/usr/local/arm/4.3.2/bin`
- `/opt/FriendlyARM/mini2440/usr/local/arm/4.3.2/arm-none-linux-gnueabi/bin`

Compilación de la librería Qtopia

En consola, entrar al modo súper-usuario, luego dirigirse a la siguiente carpeta donde quedará la librería, `/opt/FriendlyARM/mini2440/arm-qtopia`. Ejecutar el comando `./build-all`. Este comando configurará la forma en que será compilada la librería Qtopia junto con otros paquetes necesarios para el uso de la misma. El proceso puede tardar aproximadamente una hora y media, pero depende de la capacidad de la estación de trabajo donde se esté compilando la librería. NOTA: El comando `./build-all` esta configurado para instalar la librería en la carpeta `/opt/FriendlyARM/mini2440/arm-qtopia`, si los archivos fueron colocados en otra locación; debe cambiar el archivo `build-all`.

Comprobación de la librería

Para comprobar si la librería quedo instalada y compilada de manera correcta, se ejecuta un ejemplo.
En consola (modo súper-usuario) se accede a la carpeta: `/opt/FriendlyARM/mini2440/arm-qtopia/hello`.
En esta carpeta hay varios archivos mostrados a continuación (en orden alfabético):

- build

- hello.cpp
- hello.h
- hello.pro
- Hello2440.desktop
- hello_base.ui
- main.cpp
- Makefile

Hay tres archivos que es importante analizar. El primero es `build`; este archivo se encarga de la compilación de la aplicación ejemplo. Todos los programas que se realicen deben tener este archivo, sin cambiar ninguna línea. El segundo archivo es el `Makefile`; el cual contiene toda la configuración acerca de cómo debe ser compilado el programa y que tipo de compilador usar. El tercer archivo, es el `hello.pro`; en él aparecen los archivos de la aplicación incluidos (*.h, *.cpp, *.ui) y el directorio de destino, dónde se creará la aplicación; si no se cambia, por defecto crea los binarios en el directorio `$(QPEDIR)/bin` que en realidad seria el directorio ubicado dentro del sistema en la siguiente locación: `/opt/FriendlyARM/mini2440/arm-qtopia/qtopia-2.2.0-FriendlyARM/qtopia/bin`. El nombre del binario depende del valor que se le dé al argumento `TARGET`. Por defecto para este ejemplo, el nombre será `hello`. Los demás archivos, son los típicos archivos usados en la creación de una aplicación en Qtopia o Qt (más adelante se usará el archivo llamado `Hello2440.desktop`). Ahora se ejecuta el comando `./build`.

La bandera al final de la compilación `-lqte`, indica que el programa fue compilado con éxito. Ahora se puede buscar el binario en la carpeta indicada arriba. De esta manera quedará configurada la estación de trabajo.

Configuración del sistema embebido mini2440

Para La configuración de la mini2440, irónicamente el procedimiento se realiza en una plataforma Windows; puesto que el driver para la conexión de la mini2440 por USB está para dicha plataforma. Primero se debe descargar el siguiente paquete del sitio web de los creadores del sistema embebido `http://www.arm123.com.cn/linux/linux-images-20090616.rar`, el cual contiene los siguientes archivos:

- root_qtopia.img
- supervivi_mini2440
- zImage_A70

- zImage_N35
- zImage_VGA1024x768

Ahora se debe conectar la mini2440 al puerto serial. Para observar la manera en que los paquetes se cargan a la mini2440, es mejor utilizar el programa hyperterminal que viene con windows (a una velocidad de 115200 y ningún flujo de control). La mini2440 tiene dos switch, cada uno en un extremo superior e inferior del lado izquierdo de la misma. Para configurar la mini2440 por medio del puerto hyperterminal, el switch superior de la izquierda (el que dice: NOR) debe cambiarse hacia la izquierda. Ahora debe encenderse el sistema (switch inferior de la izquierda) y en el hyperterminal se puede ver un menú de la configuración propia del sistema. Se conecta ahora, la mini2440 a windows por medio del puerto USB. El driver está en la siguiente locación del DVD que viene con el sistema embebido (traducido del chino al inglés) `F:/instrument windows platform/USB driver`. Luego se abre el programa que se encarga de la transmisión de datos desde windows a la mini2440 en la siguiente locación (traducido del chino al inglés) `F:/instrument windows platform/dnw`. Es importante que en la barra superior del programa `dnw`, aparezca `USB:OK` indicando que el driver se instala correctamente y el programa esta listo para enviar datos.

En el hyperterminal, después del menú que aparece, se realiza la siguiente operación:

1. Oprimir la tecla `x`, esto formateará el sistema, preparándolo para la instalación de la nueva imagen del kernel.

2. Oprimir la tecla `v`, aparecerá un mensaje indicando que está esperando por la transferencia del archivo. Ahora se abre el programa `dnw` y en el menú: USB Port/Transmit/Restore se abre el archivo `supervivi_mini2440`. Este archivo permitirá encender el sistema embebido sin necesidad de tener el switch superior izquierdo activado.

3. Oprimir la tecla `k`, aparecerá un mensaje indicando que está esperando por la transferencia del archivo. Ahora se abre el programa `dnw` y en el menú: USB Port/Transmit/Restore se abre el archivo `zImage_N35`. Este archivo es la imagen del kernel del sistema.

4. Oprimir la tecla `y`, aparecerá un mensaje indicando que está esperando por la transferencia del archivo. Ahora se abre el programa `dnw` y en el menú: USB Port/Transmit/Restore se abre el archivo `root_qtopia.img`. Este archivo instala el sistema operativo de la mini2440.

El sistema embebido ahora está preparado para ser utilizado, solo hace falta desconectar los cables serial RS232, USB y cambiar de estado el switch superior izquierdo para luego reiniciar el sistema embebido.

Probando el programa `Hola Mundo`

De la estación de trabajo Fedora, copiar los archivos `Hello2440.desktop` y el binario `hello` en una memoria USB (o si se tiene una memoria SD).
Dentro del sistema embebido, en la pestaña del menú `FriendLyARM`, hacer clic en el *File Browser*. Dejar el archivo `Hello2440.desktop` en la siguiente locación `/opt/Qtopia/apps/Applications`. Luego dejar el archivo `hello` en la locación `/opt/Qtopia/bin`. Ahora en la pestaña del menú `Applications` se podrá ver un icono que dice `Hello2440`, hacer click sobre éste icono y se abrirá un programa que mostrará una pantalla gris con un botón, hacer clic sobre éste y finalmente el sistema estará preparado para hacer aplicaciones y probarlas.

Interfaz gráfica del `Master`

El uso de la interfaz gráfica es sencillo. Primero antes de inicializar el programa, debe el sistema embebido comenzar el programa del servidor. Una vez listo, al comenzar la interfaz gráfica del `Master`, el QTextEdit mostrará el mensaje de conexión al servidor. A continuación, el botón verde del pingüino de Linux inicializará el video transmitido por el sistema embebido. Las flechas se encargarán de los comandos de dirección enviados al robot. El QSlider de la derecha es el encargado de determinar el valor del PWM que será enviado a los motores.
Finalmente para terminar la aplicación, sólo basta oprimir el botón rojo del pingüino de Linux para cerrar el servidor Serial y después el botón que dice *Close Connection* para cerrar el servidor Image.

MICROCHIP

PIC16F87X

28/40-Pin 8-Bit CMOS FLASH Microcontrollers

Devices Included in this Data Sheet:

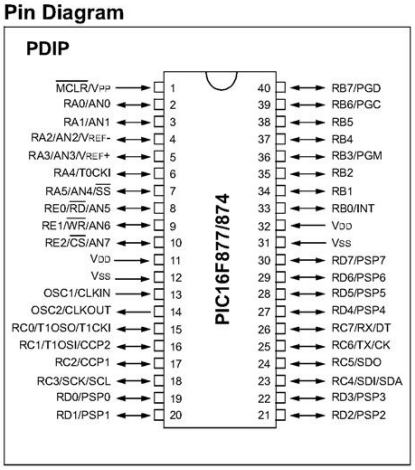
- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

Microcontroller Core Features:

- High performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory,
  Up to 368 x 8 bytes of Data Memory (RAM)
  Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and
  Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low power, high speed CMOS FLASH/EEPROM technology
- Fully static design
- In-Circuit Serial Programming™ (ICSP) via two pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0V to 5.5V
- High Sink/Source Current: 25 mA
- Commercial, Industrial and Extended temperature ranges
- Low-power consumption:
  - < 0.6 mA typical @ 3V, 4 MHz
  - 20 µA typical @ 3V, 32 kHz
  - < 1 µA typical standby current

Pin Diagram

PDIP



Peripheral Features:

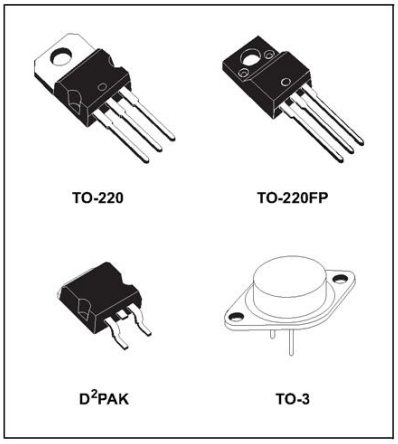
- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during SLEEP via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I²C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external RD, WR and CS controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

Figura 5.1: Hoja de Datos Microcontrolador 16F877A

L7800 SERIES

POSITIVE VOLTAGE REGULATORS

- OUTPUT CURRENT TO 1.5A
- OUTPUT VOLTAGES OF 5; 5.2; 6; 8; 8.5; 9; 12; 15; 18; 24V
- THERMAL OVERLOAD PROTECTION
- SHORT CIRCUIT PROTECTION
- OUTPUT TRANSITION SOA PROTECTION



DESCRIPTION

The L7800 series of three-terminal positive regulators is available in TO-220, TO-220FP, TO-3 and D^2PAK packages and several fixed output voltages, making it useful in a wide range of applications. These regulators can provide local on-card regulation, eliminating the distribution problems associated with single point regulation. Each type employs internal current limiting, thermal shut-down and safe area protection, making it essentially indestructible. If adequate heat sinking is provided, they can deliver over 1A output current. Although designed primarily as fixed voltage regulators, these devices can be used with external components to obtain adjustable voltage and currents.

SCHEMATIC DIAGRAM

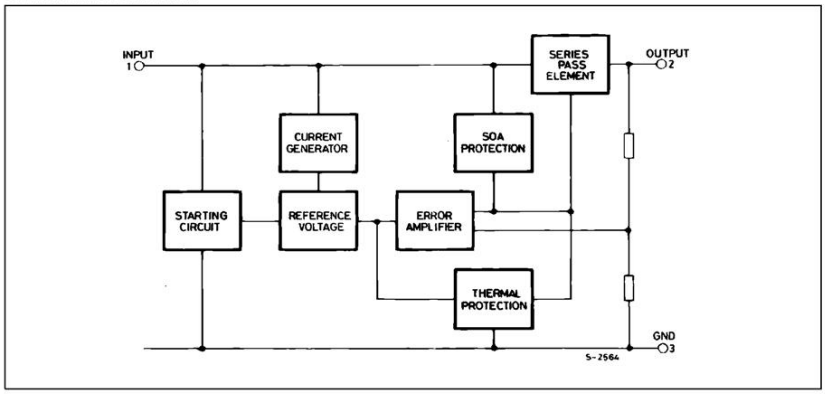


Figura 5.2: Hoja de Datos L7805CT

D-Link®

CREATE A WIRELESS NETWORK

DIR-300

WHAT THIS PRODUCT DOES

Share your broadband Internet connection with multiple computers in your house by simply connecting the D-Link Wireless G Router to your cable or DSL modem. Once connected, you can create your own personal wireless home network to share documents, music, and photos.

D-LINK WILL HELP YOU...

Set up your new D-Link networking hardware in minutes using our new Quick Setup Wizard. The wizard will guide you through an easy to follow process to install your new hardware and connect to your network.

ADVANCED WIRELESS FUNCTIONS

The D-Link Wireless G Router includes everything you need to get a wireless network up and running.

DIR-300 Wireless G Router:

+ IEEE 802.11b/g Compatible
+ Share Your Internet Connection with Built-In 4-Ports Switch
+ Advanced Firewall & Security
+ Advanced Scheduling and User Level Control
+ Supports VPN Passthrough
+ WFA (TKIP) and WFA2 (AES) Support
+ Interactive Install Guide
+ UPnP™ Support

Network Accessories & Aid:

+ CAT5 Ethernet Cable
+ Power Adapter
+ Manuals and Driver on CD-ROM
+ Quick Installation Guide
+ 1 Detachable Antenna

TECHNICAL SPECIFICATIONS

MINIMUM SYSTEM REQUIREMENTS

DIR-300:

+ Cable or DSL Modem with Ethernet Port
+ Computer with Windows XP SP2 or Windows 2000 SP4 or Mac OS X (v10.4/v10.3) or Linux-Based Operating System and an Installed Ethernet Adapter
+ Internet Explorer 6 or Firefox 1.5 or Later

STANDARDS

+ IEEE 802.11g
+ IEEE 802.11b
+ IEEE 802.3
+ IEEE 802.3u

WIRELESS SIGNAL RATES [1] WITH AUTOMATIC FALLBACK

+ 54, 48, 36, 24, 18, 12, 11, 9, 6, 5.5, 2, & 1Mbps

WIRELESS FREQUENCY RANGE

+ 2.4GHz to 2.462GHz

MODULATION TECHNOLGOY

+ Orthogonal Frequency Division Multiplexing (OFDM)
+ Complementary Code Keying (CCK)

WIRELESS TRANSMIT POWER

+ DIR-300: 15dBm +/- 2dBm

ANTENNA

+ DIR-300: Detachable Dipole Antenna (Reverse SMA Plug)

SECURITY

+ WEP 64/128-Bit Data Encryption (User-Selectable)
+ Wi-Fi Protected Access (WFA/WFA2)

ADVANCED FIREWALL FEATURES (DIR-300)

+ Network Address Translation (NAT)
+ Stateful Packet Inspection (SPI)
+ MAC Filtering
+ Website URL Filtering

DEVICE MANAGEMENT (DIR-300)

+ Internet Explorer v6 or later; or Firefox 1.5 or Later

DIAGNOSTIC LED

DIR-300:

+ Power
+ Status
+ Internet
+ WLAN (Wireless Connection)
+ LAN

POWER INPUT

+ DIR-300: 5VDC, 1.2A, Through External Power Adapter

DIMENSIONS

+ DIR-300: 112.6 x 147.5 x 31.8 mm

WEIGHT

+ DIR-300: 246 grams (0.5 lb)

CERTIFICATION

+ CE
+ FCC

OPERATING TEMPERATURE

+ DIR-300: 0˚ to 55˚ C (32˚ to 131˚ F)

OPERATING HUMIDITY

+ DIR-300: 95% maximum

1. Maximum wireless signal rate based on IEEE standard 802.11g specifications. Actual data throughput will vary. Network conditions and environmental factors, including volume of network traffic, building materials and construction, and network overhead lower actual data throughput rate and adversely affect the range.
2. The devices described herein are in compliance with the essential requirements and other relevant provisions of Directive 1999/5/EC.

CE FC

D-Link Worldwide Offices

U.S.A	TEL. 1-800-326-1688	FAX. 1-866-743-4905	Spain	TEL. 34 93 409 0778	FAX. 34 93 491 0795	Pakistan	TEL. 92-21-454-8158	FAX. 92-21-453-5163
Canada	TEL. 1-905-8295033	FAX. 1-905-8295223	Portugal	TEL. 351 21 8568463		Israel	TEL. 972-9-9715700	FAX. 972-9-9715601
Europe, UK & Ireland	TEL. 44-20-8955-9000	FAX. 44-20-8955-9002	Greece	TEL. 30 210 9914 512	FAX. 30 210 9910902	Latin America	TEL. 56-2-5838-950	FAX. 56-2-5838953
Germany	TEL. 49-6196-77990	FAX. 49-6196-7799300	Czech Republic	TEL. 420 274 247 500		Brazil	TEL. 55-11-2185-9300	FAX. 55-11-2185-9322
France	TEL. 33 1 30 23 86 86	FAX. 33 1 30 23 86 89	Hungary	TEL. 36 (0) 1 461 30 00	FAX. 36 (0) 1 461 30 09	South Africa	TEL. 27-12-665-2165	FAX. 27-12-665-2186
Netherlands	TEL. 31-10-282-1445	FAX. 31-10-282-1531	Poland	TEL. 48 (0) 22 583 92 75	FAX. 48 (0) 22 583 92 76	Russia	TEL. 7-495-744-0099	FAX. 7-495-744-0099
Belgium & Luxemburg	TEL. 32(0) 517 7111	FAX. 32(0) 517 6500	Singapore	TEL. 65-6774-6233	FAX. 65-6774-6322	Japan	TEL. 81-3-5781-8963	FAX. 81-3-5781-8965
Switzerland	TEL. 41 (0) 1 832 11 00	FAX. 41 (0) 1 832 11 01	Australia	TEL. 61-2-8899-1800	FAX. 61-2-8899-1868	Korea	TEL. 82-2-860-5491	FAX. 82-2-860-549
Sweden	TEL. 46-(0)8564-61900	FAX. 46-(0)8564-61901	India	TEL. 91-22-2652 6696	FAX. 91-22-2652 8914	China	TEL. 86-10-58635800	FAX. 86-10-58635799
Denmark	TEL. 45-43-969040	FAX. 45-43-424347	Middle East (Dubai)	TEL. 971-4-391-6480	FAX. 971-4-350-8581	Taiwan	TEL. 886-2-6600-0123	FAX. 886-2-6600-2639
Norway	TEL. 47 99 300 100	FAX. 47 22 30 95 80	Egypt	TEL. 202-791-5635	FAX. 202-791-5051	Headquarters	TEL. 886-2-6600-0123	FAX. 886-2-6600-5858
Finland	TEL. 358-10 309 6840	FAX. 358-10 309 6841	Turkey	TEL. 90-212-289-5659	FAX. 90-212-289-1656			
Italy	TEL. 39-02-2560-0616	FAX. 39-02-2560-1723	Iran	TEL. 90-212-289-5659	FAX. 90-212-289-1656			

Release 02 (Apr 2007)

WIRELESS

Figura 5.3: Hoja de Especificaciones Router D-Link

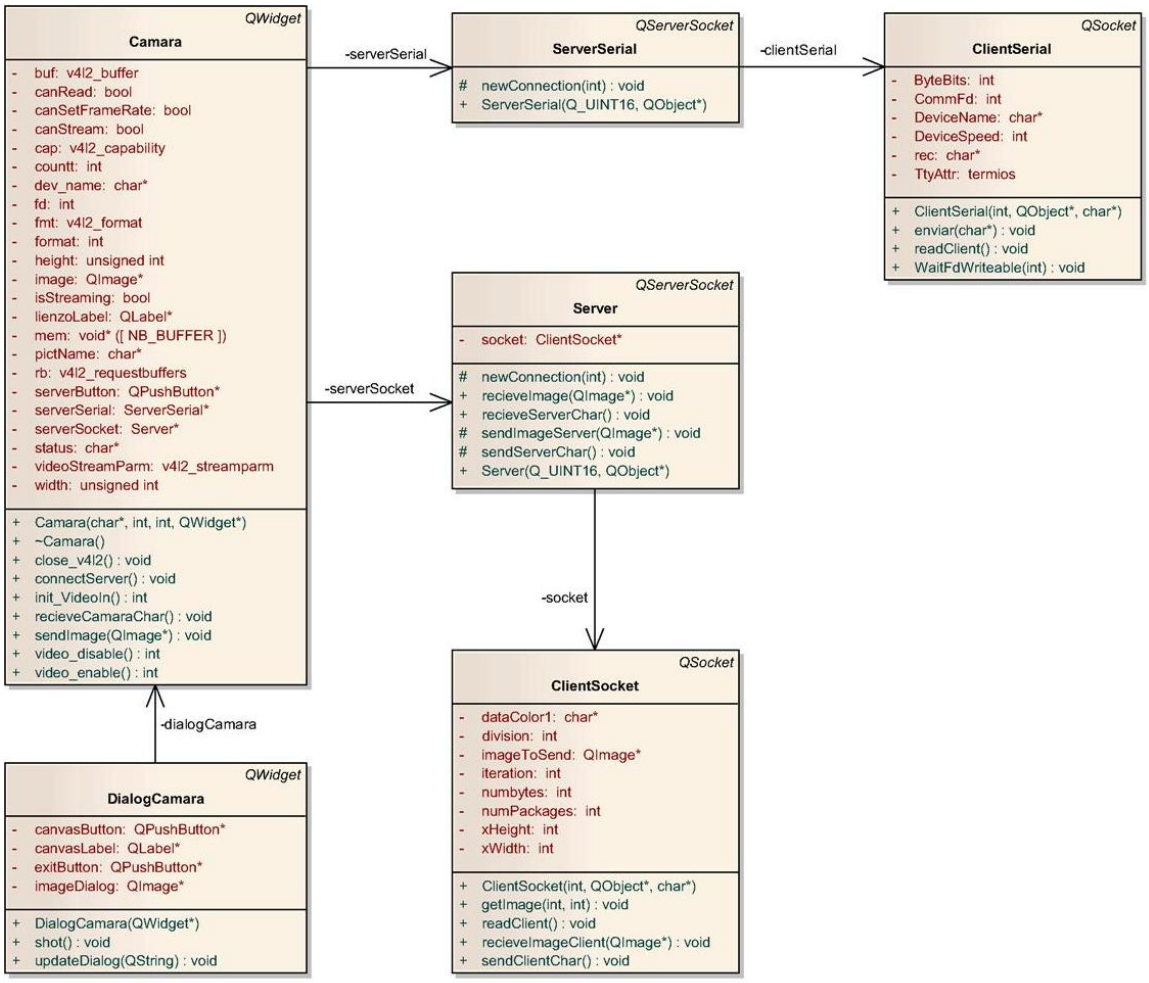


Figura 5.4: Diagrama UML del Software del Sistema Embebido

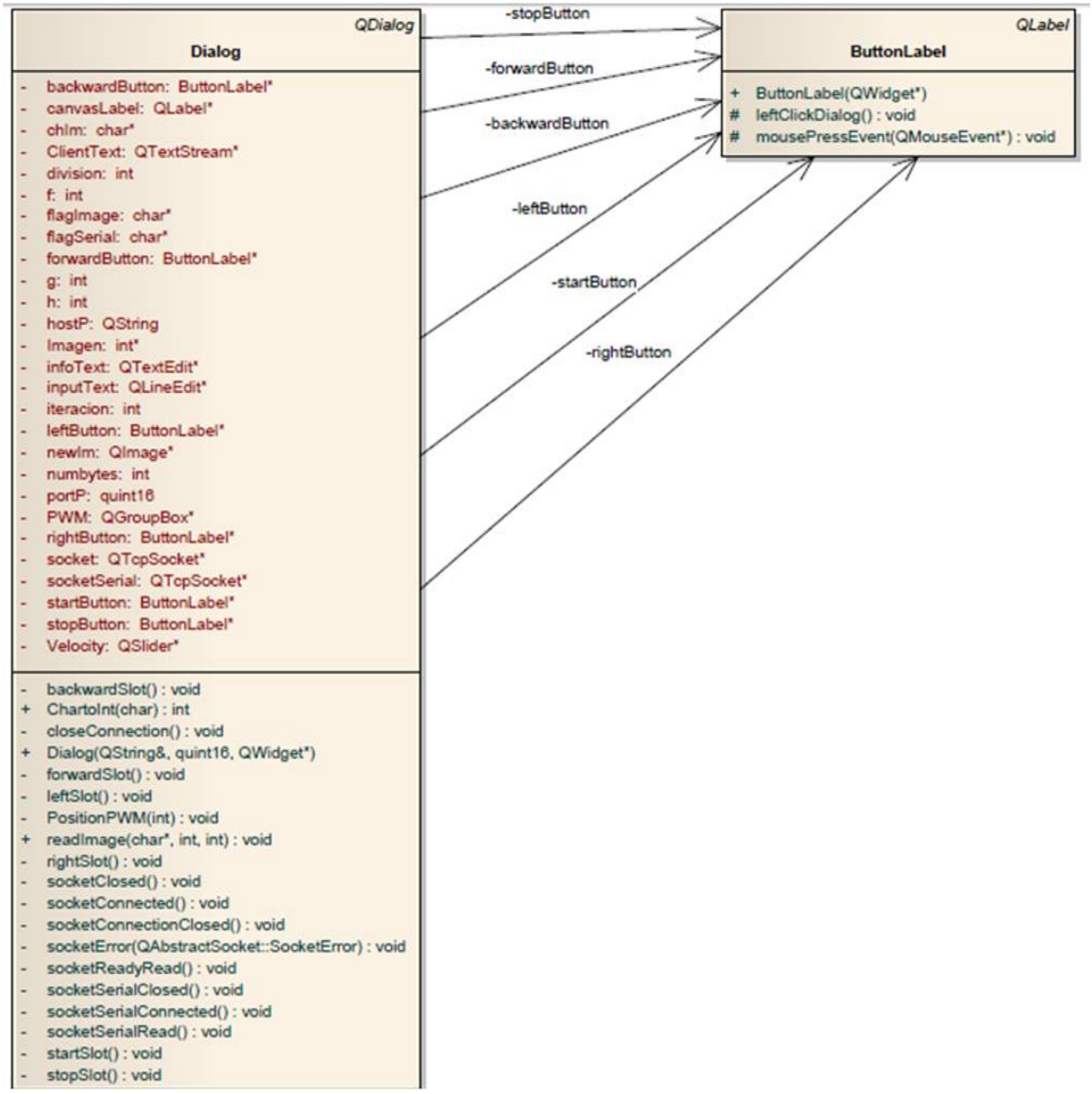


Figura 5.5: Diagrama UML del software del Client

BIBLIOGRAFÍA

[1] Ball, S. R. *Embedded Microprocessor Systems: Real World Design.* Butterworth-Heinemann, 2 edition, 2000.

[2] Barr, M. Introduction to pulse width modulation (pwm). *http://www.netrino.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation* (1999).

[3] Baturone, A. O. *Robotica, Manipuladores Y Robots Moviles.* Alfaomega - Marcombo; 1 ed. edition, April 1, 2007.

[4] Cerón, A. C. Sistemas robóticos teleoperados. *Ciencia e Ingeniería Neogranadina Número 15* (Noviembre de 2005), 8 páginas.

[5] Corbet. Video4linux2 part 6b: Streaming i/o. *http://lwn.net/Articles/240667/* (July 5, 2007).

[6] David G. Maxinez, J. A. *VHDL, El arte de programar sistemas digitales.* CECSA, 1 edición, Marzo, 2004.

[7] Harvey & Paul, M. D. . D. *C++ Cómo Programar.* Prentice Hall; 6 edition, Agosto 3, 2007.

[8] Jasmin Blanchette, M. S. *C++ GUI Programming with Qt 4.* Prentice Hall PTR; 2 edition, February 14, 2008.

[9] Marwedel, P. *Embedded System Design.* Springer, 1 edition, 2003.

[10] Michael J. Donahoo, K. L. C. . *TCP/IP Sockets in C.* Morgan Kaufmann; 2 edition, April 3, 2009.

[11] Molkentin, D. *The Book of Qt 4: The Art of Building Qt Applications.* No Starch Press; illustrated edition edition, July 19, 2007.

[12] Orozco, A. F., and Vélez, G. J. Historia de la robótica. *Universidad Pontificia Bolivariana* (2002).

[13] Philippe Gerum, Karim Yaghmour, J. M. G. B.-Y. *Building Embedded Linux Systems.* O'Reilly, 2 edición, 2008.

[14] Thelin, J. *Foundations of Qt Development.* Apress, August 3, 2007.