Sistemas Informáticos
Curso 2004-05

Protocolo de transmisión de señales estereoscópicas para control remoto de un robot autónomo

Francisco José Mansilla Albornos
David Martínez Bermejo
David Miguel Águeda

Dirigido por:
Prof. Gonzalo Pajares Martinsanz
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Autorización

Autorizamos a la facultad de Informática de la Universidad complutense de Madrid, así como al resto de los centros adscritos a la Universidad Complutense de Madrid a la utilización y modificación de todos los componentes que forman este proyecto. Siempre y cuando no se derive en una utilización comercial de cualquiera de los componentes o de parte de ellos.

Francisco José Mansilla Albornos

David Martínez Bermejo

David Miguel Águeda

Lista de palabras clave para la búsqueda bibliográfica:

Estéreo
Vídeo
TCP
UDP
Wavelets
Transmisión
Control Remoto

Agradecimientos

Nuestro más profundo agradecimiento al catedrático D. Luis Jáñez Escalada como Director del Instituto de Tecnología del Conocimiento en la Universidad Complutense de Madrid y al equipo bajo su dirección sin cuya inestimable ayuda el proyecto que presentamos no hubiera sido posible. Es igualmente destacable la valoración e interpretación de los resultados por parte de este mismo equipo experto en temas de percepción visual humana y artificial.

Agradecimientos al profesor Gonzalo Pajares Martinsanz por su dedicación y ayuda en el diseño y realización de este proyecto.

Agradecemos a la dirección de la delegación de alumnos de la facultad de Veterinaria de la Universidad Complutense de Madrid por su colaboración en las pruebas de dicho proyecto, así como ha todas las personas que directa o indirectamente han colaborado en la realización del mismo, especialmente a Gonzalo Fernandez Hernández por su colaboración en las pruebas.

Resumen:

Este proyecto se basa en la investigación de distintas tecnologías de transmisión de imágenes estereoscópicas en tiempo real, así como en el desarrollo e implementación de una aplicación que realice esta transmisión.

Nuestro trabajo está ubicado dentro de un proyecto de mayor envergadura que tiene como objetivo desarrollar un sistema de navegación no tripulado para el rescate de náufragos en condiciones extremas.

Dentro de un ámbito más particular de este proyecto, el Instituto del Conocimiento, la Facultad de Electrónica y la Facultad de Informática de la Universidad Complutense de Madrid, forman un departamento de investigación encargado del ámbito de la percepción del conductor en un entorno remoto.

El campo de investigación de nuestra área del proyecto está orientado a la optimización de las transmisiones de imágenes estereoscópicas, para que la percepción de la situación en cada momento, esté dentro de unos márgenes que aseguren la correcta precisión del rescate y la seguridad de la instrumentación a utilizar.

El motivo de la transmisión de imágenes estereoscópicas esta fundamentado en la creación de un entorno remoto de visión 3D, el cual simule con la máxima veracidad posible el estado del entorno donde se encuentre el sistema de visión. Aparte de la aplicación de nuestro proyecto a un ámbito naval, también puede ser utilizado para el control remoto de otros vehículos, instrumentación médica, etc.

Estos márgenes fijan dos parámetros en el desarrollo de nuestro proyecto: la fiabilidad y la rapidez de transmisión. Si un sistema es poco fiable mucha información del entorno se perderá, dando lugar a posibles fallos.

Por otro lado, si un sistema de transmisión no es lo suficientemente rápido, la interacción en tiempo real con la instrumentación se pierde.

Al encontrarnos en el inicio de las investigaciones y del desarrollo del proyecto general, tenemos como objetivo fijar las bases de la transmisión, descartando sistemas mediante los cuales no sea posible una transmisión eficiente.

Por este mismo motivo, parámetros como el medio de transmisión, el formato y las características de los equipos de la instrumentación, y el formato y las características de las imágenes a tratar no están definidos. Esto nos ocasiona una gran perdida de precisión en las líneas de investigación, abordando la problemática inicial de una manera muy general.

La fase de investigación de este proyecto se desarrolla en tres ámbitos:

- Transmisión de imágenes mediante el protocolo TCP.
- Transmisión de imágenes mediante el protocolo UDP.

- Procesamiento, antes y después de la transmisión por los protocolos antes enunciados, de las imágenes mediante la transformada wavelets.

Gracias a estas tres líneas de investigación, vamos a poder fundamentar cuál de los dos protocolos de transmisión es el más recomendable, así como la viabilidad de un procesamiento previo y posterior al envío y recepción de las imágenes.

Abstract:

This project is based upon the investigation of several stereoscopic images transmission technologies in real time, furthermore the development and the implementation of an application which makes this transmission.

This is a subproject for the development of an unmanned robotic system, under the objective of navigation in adverse environments and driven by a user from a remote host.

This subproject has been proposed by the Instituto de Tecnología del Conocimiento and developed in the Facultad de Informatica. Hence, two teams have been involved in such development.

The goal is the transmission optimization for stereo video signal, so that a remote user can perceive the 3-Dimensional structure in the scene under a real time specification. This kind of transmissions should be also acceptable for other possible applications such as tele-medicine.

The above specification requires an acceptable quality of the transmitted signal in real time. This implies that a trade-off must be achieved between the speed and the quality. With such purpose we have implemented studied the TCP and UDP protocols. The first is safer than the second but it is slow for real time purposes.

Hence we have investigated some kind of preprocessing and postprocessing in order to achieve the mentioned trade-off. The processing has been directed towards video compression through wavelets. Nevertheless, this topic is still open for future research.

1. Índice

2. Introducción

2.1. Antecedentes

El trabajo recogido en la presente memoria tiene sus orígenes en la propuesta realizada por el Instituto de Tecnología del Conocimiento, adscrito a la Universidad Complutense de Madrid, a la Facultad de Informática de esta misma universidad con el propósito de iniciar una colaboración en materias de investigación.

El proyecto se enmarca dentro de un proyecto más amplio en el campo de la Robótica. Dicho proyecto, pretende desarrollar un prototipo de robot teledirigido desde un puesto remoto. Este robot tendrá múltiples aplicaciones:

Un pequeño barco o sistema de navegación con un equipamiento, que le permitirá moverse en entornos marinos. La idea es utilizarlo en determinadas situaciones donde la presencia humana puede ser arriesgada, no aconsejable o incluso imposible. Particularmente, se piensa en grandes catástrofes o situaciones límite.

Un robot de Tele-asistencia a intervenciones quirúrgicas, para que un cirujano especialista pueda inspeccionar libremente el desarrollo de una operación quirúrgica realizada lejos, y pueda también dar directrices sobre su ejecución a los médicos que la estén realizando.

Un robot de conducción de vehículos en entornos hostiles (p. e. ambientes radiactivos) desde un sistema de realidad virtual alejado.

El sistema consta de un elemento robot equipado con un sistema de visión estereoscópica que transmite vídeo estéreo al puesto remoto desde el que actúa un operador humano. Este operador debe actuar en consecuencia sobre el robot en función de las imágenes que recibe. Por este motivo y dadas las circunstancias de operatividad del sistema en el entorno hostil, la señal de vídeo debe llegarle con el tiempo suficiente para que las respuestas del operador sean efectivas. Se requiere además que la calidad de las imágenes sea lo suficientemente buena como para que permita la maniobrabilidad del sistema.

Las actividades previstas para el completo desarrollo del sistema son las siguientes:

1. Estudio, diseño e implementación del sistema de cámaras estéreo
2. Adquisición de las secuencias de video estéreo.
3. Codificación de las secuencias, de manera que las pérdidas de datos no comprometan la fusión estereoscópica.
4. Transmisión de las secuencias codificadas a través de una red IP.
5. Decodificación de la secuencia recibida.
6. Visualización utilizando gafas de obturación óptica

7. Determinar y ajustar los parámetros de adquisición de las cámaras para conseguir que el operador tenga una visión realista: la métrica del espacio percibido ha de coincidir con la del espacio físico situado ante el robot.
8. Transmisión por internet de
    - señales de control sobre las cámaras.
    - señales de control de los movimientos del robot
9. Ídem para audio

El objetivo que nuestro proyecto se plantea es doble:

- Transmitir las imágenes estereoscópicas en tiempo real, entendiendo por tal el que requiere el operador para percibir en 3D

- Enviar las señales de vídeo con la máxima calidad posible

En consecuencia, estos serían los requisitos de usuario planteando el proyecto como un desarrollo Software.

2.2. Problemática

Como se ha mencionado anteriormente, la problemática surgida para la realización de este proyecto tiene su base en el envío de vídeo estereoscópico. En el momento de abordar el proyecto no se dispone del Hardware necesario para producir vídeo estereoscópico, y los sistemas de recepción de vídeo estereoscópico del Instituto del conocimiento necesitan una actualización para adaptarse a las nuevas tecnologías.

Se inicia el proyecto tomando como base las imágenes en un formato conocido técnicamente como "*raw*"en terminología inglesa. Este formato esta asociado a la extensión informática *".raw"* y consiste en imágenes en escala de grises con un byte de profundidad, es decir contiene imágenes cuya intensidad varía en el rango 0 a 255.

Se ha tomado este formato debido a que las cámaras de vídeo que se planificaron para el sistema guardan las imágenes tomadas bajo esta extensión.

Estas imágenes deben ser enviadas desde una unidad emisora hasta otra unidad receptora, a través de un medio todavía no fijado, en tiempo real y con un número de pérdidas en la composición de las imágenes mínimo y controlado.

De la misma manera en la que el medio de transmisión no está totalmente definido, existen otros parámetros como el tamaño de las imágenes y los márgenes de pérdidas de las propias imágenes que no están todavía suficientemente definidos.

2.3. Enfoque

El enfoque utilizado para el tratamiento del problema consistió en la implementación en el lenguaje de programación *JAVA* de los protocolos propios de transmisión de datos a nivel de sesión. Los cuales pudieran ser universalmente utilizados en cualquier sistema compatible con el *Entorno de tiempo de ejecución Java de Sun microsystems* ® que posea una versión de la máquina virtual de JAVA igual o superior a la 1.4.

El software desarrollado, aunque guiado a los tipos de datos y tamaños concretos necesarios para esta aplicación, se implementaría de acuerdo a criterios de reutilizado para sortear posibles inconsistencias en los requisitos externos, bastante probables dada la baja concreción del proyecto global.

2.4. Contenido de la Memoria

La memoria se compone de diversos apartados en los que vamos a sintetizar todo el desarrollo del proyecto llevado a cabo durante este último curso lectivo.

Dentro de estos apartados se concretan las soluciones aportadas por nuestro equipo de proyecto a la problemática descrita anteriormente, una explicación tanto de los requisitos a tener en cuenta por los usuarios así como por el entorno donde se quiera desarrollar nuestra aplicación y una descripción detallada de todos las fases y decisiones tomadas a lo largo de este curso para la realización de el proyecto.

Se incluye una sección encaminada a explicar el desarrollo del proyecto entendido como un proyecto software y su ciclo de vida.

También se incluirá en esta memoria un apartado con las pruebas y resultados obtenidos, así como las conclusiones a las que hemos llegado al término de este proyecto.

Además existirá un apartado en el cual explicaremos la manera de dar continuidad a este proyecto, para que en años sucesivos se pueda continuar con el desarrollo procedente de la investigación, contando con la información obtenida y demostrada, de la manera más eficiente posible.

3. Solución aportada

La solución final aportada por nuestro equipo de proyecto se basa en dos principios notablemente independientes, a saber: el protocolo de comunicación y el preprocesamiento-postprocesamiento de las imágenes para ser enviadas y recibidas.

Gracias a la permutación de estos dos principios, llegamos a la distribución final de las soluciones del proyecto, que se engloban en cuatro campos: protocolo de envío TCP (*Transmission Control Protocol*), protocolo de envío TCP con compresión-descompresión de imágenes, protocolo de envío UDP (*User Datagram Protocol*) y protocolo de envío UDP con compresión descompresión de imágenes.

En un principio el proyecto no estaba tan claramente dividido en estos cuatro apartados, ya que al tener un alto contenido de investigación tuvimos que ir añadiendo nuevos campos al mismo a medida que se nos abrían o cerraban puertas dependiendo de los resultados.

Durante la realización del proyecto se han ido probando diferentes soluciones al problema, que se han descartado por diversos motivos, quedándonos finalmente con la implementación descrita en esta memoria.

En un primer momento y tras una fase de documentación sobre otros temas análogos al que trataba nuestra investigación, decidimos partir de la comparativa entre los protocolos de envío TCP y UDP. La decisión de empezar a probar con estos dos protocolos fue debida a que a pesar de ser muy utilizados y conocidos, no encontramos ningún documento consistente en el que se reflejara fielmente el comportamiento de estos dos sistemas en un campo parecido al de nuestro desarrollo e investigación.

Algunas de las referencias utilizadas como base de nuestro trabajo son los trabajos [1], [2] y [3]. En estas referencias se realizó un estudio sobre la migración y costumbres de los salmones. Al final de esa tesis se llego a la conclusión de que debido al sistema de transmisión entre los dispositivos instalados en los salmones y el centro de control, se perdió la suficiente información como para no poder dar una conclusión fiable sobre las costumbres migratorias del salmón. Esta conclusión a simple vista parece no tener mucha relación con el tema de nuestra investigación, pero nada mas lejos de la realidad, ya que la transmisión existente en este experimento de los salmones entre cada ejemplar y el centro de control se realizó por medio del protocolo UDP, ya que se necesitaba gran velocidad de envío para saber en cada momento la situación de cada ejemplar.

Por otro lado en el comienzo de nuestro proyecto estaba en pleno nacimiento el mercado en España de la tecnología 3G entre terminales móviles, la cual permite la videoconferencia entre dos terminales. Según las conclusiones que pudimos obtener de diversos documentos, la tecnología 3G se fundamentaba en el protocolo de envío UDP.

Al principio no podíamos entender como un mismo protocolo podía dar tan buenos resultados en un campo y tan malos en otro, siendo ambos campos tan conceptualmente semejantes.

Por estas razones decidimos estudiar los protocolos de transmisión de datos TCP y UDP por nuestros medios para poder sacar unas conclusiones lo suficientemente fiables que nos pudieran llevar a resolver la problemática sugerida para la realización de este proyecto.

Poco después de afianzar el estudio en los campos de transmisión de datos TCP y transmisión de datos UDP, se nos planteó la idea de realizar un pequeño preprocesamiento consistente en la reducción del tamaño de las imágenes para el envío y su posterior recuperación al tamaño original en el puesto receptor.

La técnica utilizada para el preprocesamiento de las imágenes responde al nombre de *wavelets*.

Este preprocesamiento, no debería influir demasiado en el concepto de transmisión en tiempo real pero tendría la misión de reducir el tamaño de datos a enviar causando la mínima perdida de datos y de tiempo de procesamiento de las imágenes.

3.1. Aspectos teóricos

En esta sección mostramos una breve iniciación en los aspectos teóricos fundamentales para el buen entendimiento del trabajo realizado en este proyecto. Estos aspectos teóricos abarcan los tres principales campos en los que se ha movido la investigación y el desarrollo de este proyecto: protocolo de transmisión de datos TCP, protocolo de transmisión de datos UDP y procesamiento de imágenes mediante la transformada de wavelets.

El objetivo de esta parte relativa al procesamiento tiene su fundamento en el hecho de probar métodos de tratamiento de imágenes que reduzcan al máximo el tamaño de éstas con la mínima pérdida de información posible. Esto es así para ganar tiempo en las transmisiones y conseguir el objetivo de transmisión en tiempo real comentado inicialmente.

3.1.1. Protocolo de transmisión de datos TCP

El protocolo TCP (*Transmission Control Protocol*, protocolo de control de transmisión) está basado en IP que es no fiable y no orientado a conexión y sin embargo este protocolo es:

- Orientado a conexión. Es necesario establecer una conexión previa entre las dos máquinas antes de poder transmitir ningún dato. A través de esta conexión los datos llegarán siempre a la aplicación destino de forma ordenada y sin duplicados. Finalmente, es necesario cerrar la conexión.

- Fiable. La información que envía el emisor llega de forma correcta al destino.

El protocolo TCP permite una comunicación fiable entre dos aplicaciones. De esta forma, las aplicaciones que lo utilicen no tienen que preocuparse de la integridad de la información: dan por hecho que todo lo que reciben es correcto.

El flujo de datos entre una aplicación y otra viajan por un *circuito virtual*. Sabemos que los datagramas IP pueden seguir rutas distintas, dependiendo del estado de los encaminadores (router) intermedios, para llegar a un mismo destino. Esto significa que los datagramas IP que transportan los mensajes siguen rutas diferentes aunque el protocolo TCP de la impresión de que existe un único circuito por el que viajan todos los paquetes que componen un mensaje uno detrás de otro (algo así como una tubería entre el origen y el destino). Para que esta comunicación pueda ser posible es necesario abrir previamente una conexión. Esta conexión garantiza que los todos los datos lleguen correctamente de forma ordenada y sin duplicados. La unidad de datos del protocolo es el *byte*, de tal forma que la aplicación origen envía bytes y la aplicación destino recibe estos bytes.

Sin embargo, cada byte no se envía inmediatamente después de ser generado por la aplicación, sino que se espera a que haya una cierta cantidad de bytes, se agrupan en un *segmento* y se envía el segmento completo. Para ello son necesarias unas *memorias intermedias* o *búferes*. Cada uno de estos segmentos viaja en el campo de datos de un datagrama IP. Si el segmento es muy grande será necesario fragmentar el datagrama, con la consiguiente pérdida de rendimiento; y si es muy pequeño, se estarán enviando más cabeceras que datos. Por consiguiente, es importante elegir el mayor tamaño de segmento posible que no provoque fragmentación.

El protocolo TCP envía un *flujo de información no estructurado*. Esto significa que los datos no tienen ningún formato, son únicamente los bytes que una aplicación envía a otra. Ambas aplicaciones deberán ponerse de acuerdo para comprender la información que se están enviando.

Cada vez que se abre una conexión, se crea un canal de comunicación bidireccional en el que ambas aplicaciones pueden enviar y recibir información, es decir, una conexión es *full-dúplex*.

➢ *Fiabilidad*

¿Cómo es posible enviar información fiable basándose en un protocolo no fiable? Es decir, si los datagramas que transportan los segmentos TCP se pueden perder, ¿cómo pueden llegar los datos de las aplicaciones de forma correcta al destino?

La respuesta a esta pregunta es sencilla: cada vez que llega un mensaje se devuelve una confirmación (*acknowledgement*) para que el emisor sepa que ha llegado correctamente. Si no le llega esta confirmación pasado un cierto tiempo, el emisor reenvía el mensaje.

Veamos a continuación la manera más sencilla (aunque ineficiente) de proporcionar una comunicación fiable. El emisor envía un dato, arranca su temporizador y espera su confirmación (ACK).

Si recibe su ACK antes de agotar el temporizador, envía el siguiente dato. Si se agota el temporizador antes de recibir el ACK, reenvía el mensaje. Los siguientes esquemas de las figuras 1, 2 representan este comportamiento:

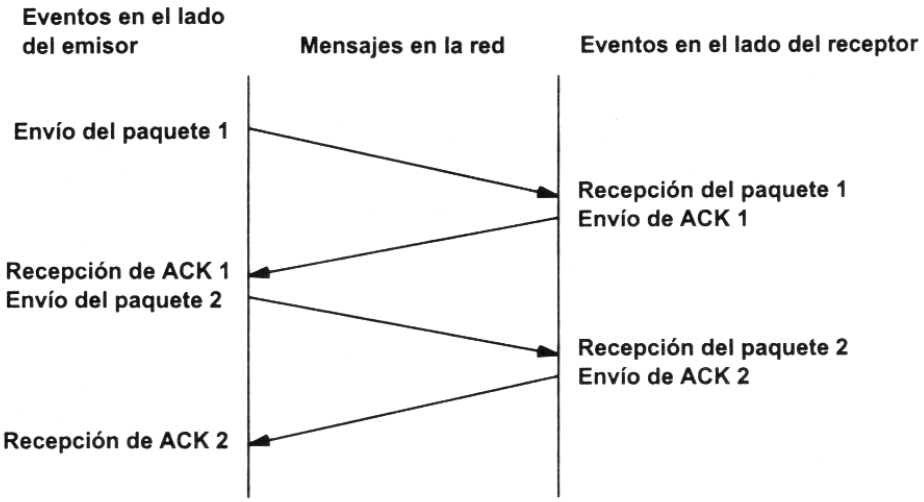


Figura 1. Esquema de envío desde el emisor sin perdida de paquete en el envío

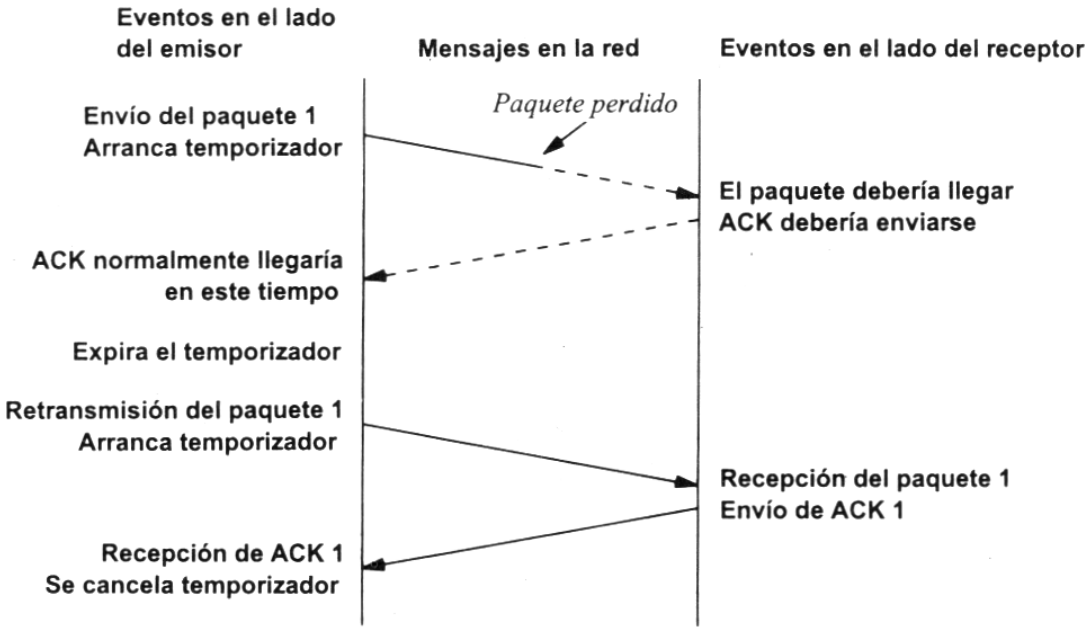


Figura 2: Esquema de la transmisión de un paquete fallido y su recuperación

Este esquema es perfectamente válido aunque muy ineficiente debido a que sólo se utiliza un sentido de la comunicación a la vez y el canal está desaprovechado la mayor parte del tiempo. Para solucionar este problema se utiliza un *protocolo de ventana deslizante*, que se resume en el siguiente esquema de la figura 3. Los mensajes y las confirmaciones van numerados y el emisor puede enviar más de un mensaje antes de haber recibido todas las confirmaciones anteriores.

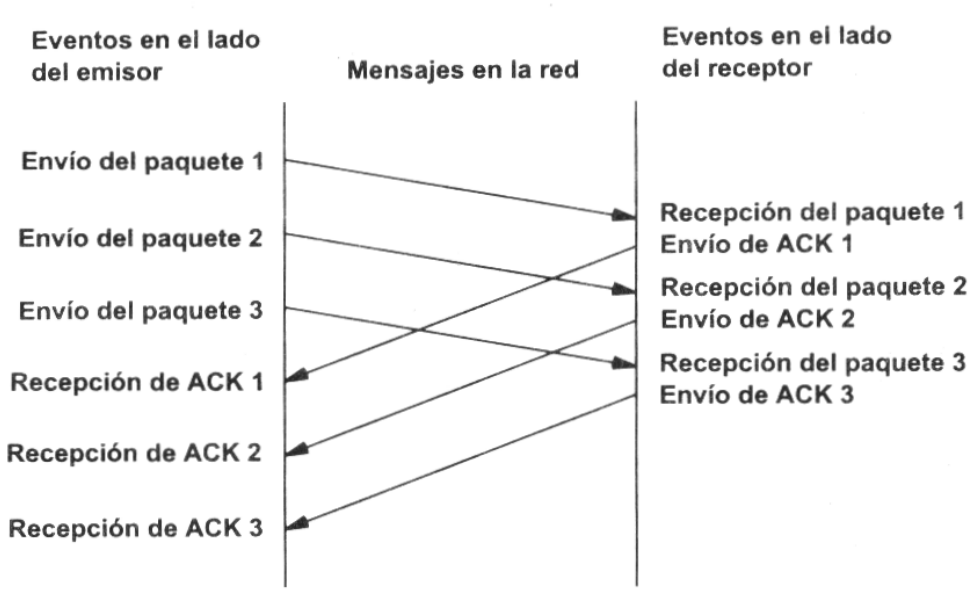


Figura 3: Esquema del protocolo de ventana deslizante

➢ *Conexiones*

Una conexión consta de dos pares *dirección IP : puerto*. No puede haber dos conexiones iguales en un mismo instante en toda la Red. Aunque bien es posible que un mismo ordenador tenga dos conexiones distintas y simultáneas utilizando un mismo puerto. El protocolo TCP utiliza el concepto de conexión para identificar las transmisiones. En el siguiente ejemplo de la figura 4, se han creado tres conexiones. Las dos primeras son al mismo servidor Web (puerto 80) y la tercera a un servidor de FTP (*File Transfer Protocol*) (puerto 21).

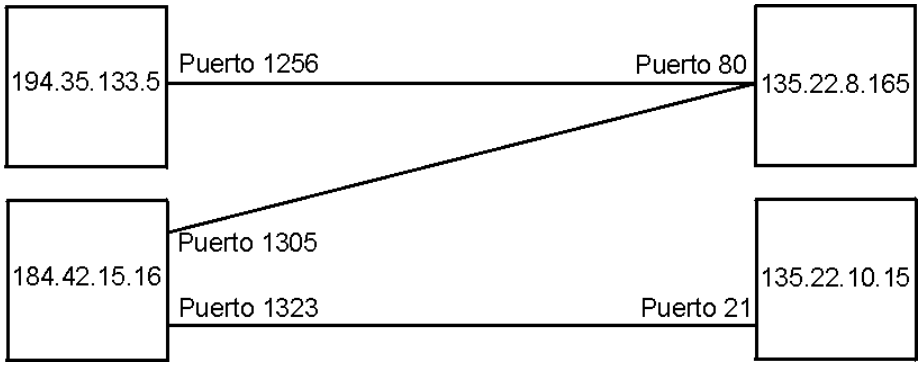


Figura 4: Esquema de conexiones

Para que se pueda crear una conexión, el extremo del servidor debe hacer una *apertura pasiva* del puerto (escuchar su puerto y quedar a la espera de conexiones) y el cliente, una *apertura activa* en el puerto del servidor (conectarse con el puerto de un determinado servidor).

➢ *Formato del segmento TCP*

Ya hemos comentado que el flujo de bytes que produce una determinada aplicación se divide en uno o más segmentos TCP para su transmisión. Cada uno de estos segmentos viaja en el campo de datos de un datagrama IP. Para facilitar el control de flujo de la información los bytes de la aplicación se numeran. De esta manera, cada segmento indica en su cabecera el primer byte que transporta. Las confirmaciones o acuses de recibo (ACK) representan el siguiente byte que se espera recibir (y no el número de segmento recibido, ya que éste no existe).

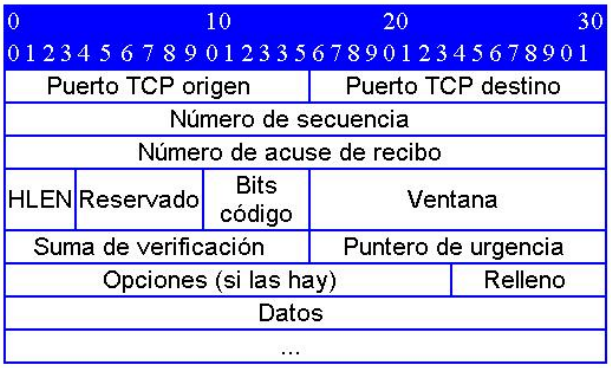


Figura 5: Esquema del formato del segmento TCP

- **Puerto fuente** (16 bits). Puerto de la máquina origen. Al igual que el puerto destino es necesario para identificar la conexión actual.
- **Puerto destino** (16 bits). Puerto de la máquina destino.
- **Número de secuencia** (32 bits). Indica el número de secuencia del primer byte que transporta el segmento.
- **Número de acuse de recibo** (32 bits). Indica el número de secuencia del siguiente byte que se espera recibir. Con este campo se indica al otro extremo de la conexión que los bytes anteriores se han recibido correctamente.
- **HLEN** (4 bits). Longitud de la cabecera medida en múltiplos de 32 bits (4 bytes). El valor mínimo de este campo es 5, que corresponde a un segmento sin datos (20 bytes).
- **Reservado** (6 bits). Bits reservados para un posible uso futuro.
- **Bits de código** o indicadores (6 bits). Los bits de código determinan el propósito y contenido del segmento. A continuación se explica el significado de cada uno de estos bits (mostrados de izquierda a derecha) si está a 1.
- **URG.** El campo *Puntero de urgencia* contiene información válida.
- **ACK.** El campo *Número de acuse de recibo* contiene información válida, es decir, el segmento actual lleva un ACK. Observemos que un mismo segmento puede transportar los datos de un sentido y las confirmaciones del otro sentido de la comunicación.
- **PSH.** La aplicación ha solicitado una operación *push* (enviar los datos existentes en la memoria temporal sin esperar a completar el segmento).
- **RST.** Interrupción de la conexión actual.
- **SYN.** Sincronización de los números de secuencia. Se utiliza al crear una conexión para indicar al otro extremo cual va a ser el primer número de

secuencia con el que va a comenzar a transmitir (veremos que no tiene porqué ser el cero).

- **FIN.** Indica al otro extremo que la aplicación ya no tiene más datos para enviar. Se utiliza para solicitar el cierre de la conexión actual.
- **Ventana** (16 bits). Número de bytes que el emisor del segmento está dispuesto a aceptar por parte del destino.
- **Suma de verificación** (24 bits). Suma de comprobación de errores del segmento actual. Para su cálculo se utiliza una *pseudo-cabecera* que también incluye las direcciones IP origen y destino.
- **Puntero de urgencia** (8 bits). Se utiliza cuando se están enviando datos urgentes que tienen preferencia sobre todos los demás e indica el siguiente byte del campo *Datos* que sigue a los datos urgentes. Esto le permite al destino identificar donde terminan los datos urgentes. Nótese que un mismo segmento puede contener tanto datos urgentes (al principio) como normales (después de los urgentes).
- **Opciones** (variable). Si está presente únicamente se define una opción: el tamaño máximo de segmento que será aceptado.
- **Relleno.** Se utiliza para que la longitud de la cabecera sea múltiplo de 32 bits.
- **Datos.** Información que envía la aplicación.

➢ *Establecimiento de una conexión*

Antes de transmitir cualquier información utilizando el protocolo TCP es necesario abrir una conexión. Un extremo hace una *apertura pasiva* y el otro, una *apertura activa*.

El mecanismo utilizado para establecer una conexión consta de *tres vías*.

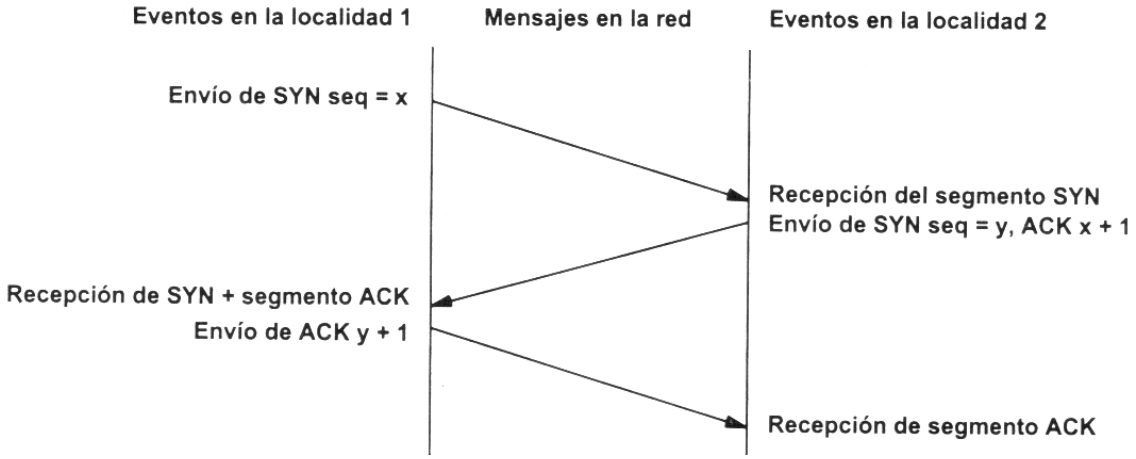


Figura 6: Establecimiento de la conexión TCP

1. La máquina que quiere iniciar la conexión hace una apertura activa enviando al otro extremo un mensaje que tenga el bit SYN activado. Le informa además del primer número de secuencia que utilizará para enviar sus mensajes.
2. La máquina receptora (un servidor generalmente) recibe el segmento con el bit SYN activado y devuelve la correspondiente confirmación. Si desea abrir la conexión, activa el bit SYN del segmento e informa de su primer número de secuencia. Deja abierta la conexión por su extremo.
3. La primera máquina recibe el segmento y envía su confirmación. A partir de este momento puede enviar datos al otro extremo. Abre la conexión por su extremo.
4. La máquina receptora recibe la confirmación y entiende que el otro extremo ha abierto ya su conexión. A partir de este momento puede enviar ella también datos. La conexión ha quedado abierta en los dos sentidos.

- ***Cierre de una conexión***

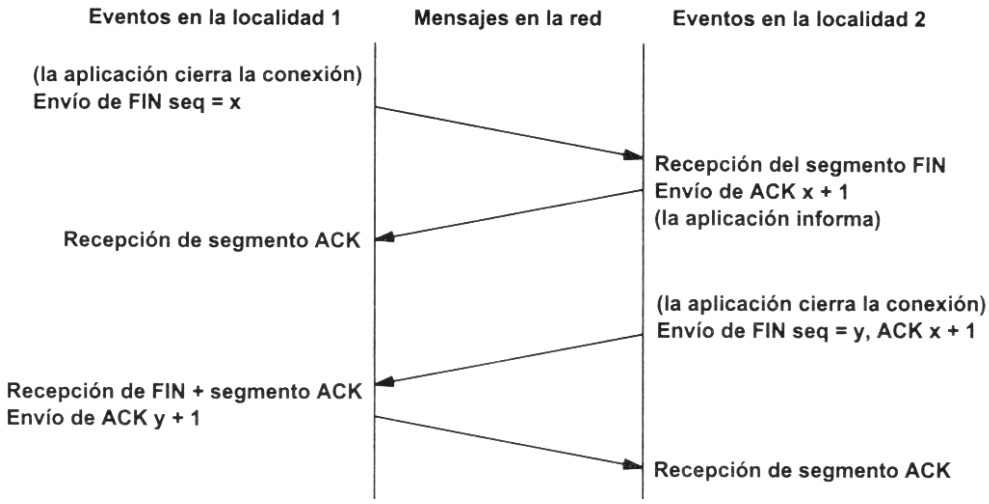


Figura 7: Cierre de una conexión TCP

Cuando una aplicación ya no tiene más datos que transferir, el procedimiento normal es cerrar la conexión utilizando una variación del mecanismo de 3 vías explicado anteriormente. Puede existir al posibilidad como en nuestro caso que la conexión se tenga que mantener siempre abierta.

El mecanismo de cierre es algo más complicado que el de establecimiento de conexión debido a que las conexiones son full-duplex y es necesario cerrar cada uno de los dos sentidos de forma independiente.

1. La máquina que ya no tiene más datos que transferir, envía un segmento con el bit FIN activado y cierra el sentido de envío. Sin embargo, el sentido de recepción de la conexión sigue todavía abierto.

2. La máquina receptora recibe el segmento con el bit FIN activado y devuelve la correspondiente confirmación. Pero no cierra inmediatamente el otro sentido de la conexión sino que informa a la aplicación de la petición de cierre. Aquí se produce un lapso de tiempo hasta que la aplicación decide cerrar el otro sentido de la conexión.
3. La primera máquina recibe el segmento ACK.
4. Cuando la máquina receptora toma la decisión de cerrar el otro sentido de la comunicación, envía un segmento con el bit FIN activado y cierra la conexión.
5. La primera máquina recibe el segmento FIN y envía el correspondiente ACK. Observemos que aunque haya cerrado su sentido de la conexión sigue devolviendo las confirmaciones.
6. La máquina receptora recibe el segmento ACK.

3.1.2. Protocolo de transmisión de datos UDP

El protocolo UDP (*User Datagram Protocol*, protocolo de datagrama de usuario) proporciona una comunicación muy sencilla entre las aplicaciones de dos ordenadores. Al igual que el protocolo IP, UDP es:

- No orientado a conexión. No se establece una conexión previa con el otro extremo para transmitir un mensaje UDP. Los mensajes se envían sin más y éstos pueden duplicarse o llegar desordenados al destino.

- No fiable. Los mensajes UDP se pueden perder o llegar dañados.

UDP utiliza el protocolo IP para transportar sus mensajes. Como vemos, no añade ninguna mejora en la calidad de la transferencia; aunque sí incorpora los puertos origen y destino en su formato de mensaje. Las aplicaciones (y no el protocolo UDP) deberán programarse teniendo en cuenta que la información puede no llegar de forma correcta.

- *Formato del segmento UDP*

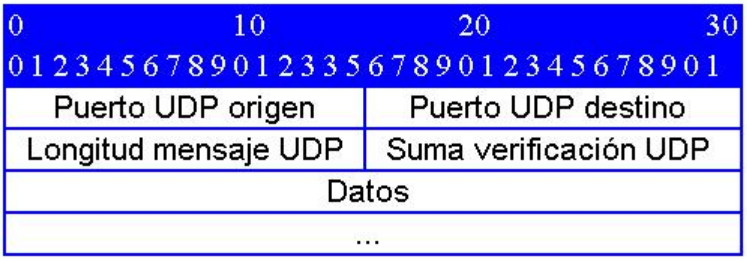


Figura 8: Esquema del formato del segmento UDP

- **Puerto UDP de origen** (16 bits, opcional). Número de puerto de la máquina origen.
- **Puerto UDP de destino** (16 bits). Número de puerto de la máquina destino.
- **Longitud del mensaje UDP** (16 bits). Especifica la longitud medida en bytes del mensaje UDP incluyendo la cabecera. La longitud mínima es de 8 bytes.
- **Suma de verificación UDP** (16 bits, opcional). Suma de comprobación de errores del mensaje. Para su cálculo se utiliza una *pseudo-cabecera* que también incluye las direcciones IP origen y destino. Para conocer estos datos, el protocolo UDP debe interactuar con el protocolo IP.
- **Datos**. Aquí viajan los datos que se envían las aplicaciones. Los mismos datos que envía la aplicación origen son recibidos por la aplicación destino después de atravesar toda la Red de redes.

2.1.3 Transformada de wavelets

Una *wavelet* es una forma de onda de duración limitada que tiene un valor medio cero. El análisis de wavelets consiste en la descomposición de una señal arbitraria *f* en versiones escaladas y trasladadas de la wavelet original (ver referencia [4] para más detalles). Es decir, la idea básica de esta transformada consiste en representar cualquier función arbitraria f como una superposición de un conjunto de dichas wavelets o funciones base.

La transformada de wavelets de una señal f es la familia de coeficientes C(a,b), que dependen de dos índices a y b que se asocian con la escala y la posición de la señal. En una sola dimensión estos coeficientes se obtienen de la forma de la ecuación siguiente

$$C(a,b)=\int_{-\infty}^{+\infty} f(x)\frac{1}{\sqrt{a}}\psi\left(\frac{x-b}{a}\right)dx \quad a\in R^{+}-\{0\} \quad \text{y} \quad b\in R$$

donde R es el conjunto de los números reales.

Partiendo de la ecuación anterior esta transformada se define como la suma sobre todo el intervalo de la señal multiplicada por las versiones escaladas y trasladadas de la función wavelet ψ. Multiplicando cada coeficiente por la apropiada wavelet escalada y trasladada obtenemos las wavelet que componen la señal original. Escalar una wavelet significa simplemente comprimirla o expandirla , por tanto a es el factor de escala.

Debido a la imposibilidad de calcular los coeficientes para todas las escalas y posiciones se recurre a una versión discreta denominada *transformada discreta de wavelets* (TDW) donde se elige un subconjunto de escalas y posiciones.

Una forma eficiente para implementar este esquema fue desrrollado inicialmente por Woods y O'Neill y posteriormente por Mallat mediante el uso de filtros. Descomponiendo la imagen en dos componentes: *aproximación de baja resolución* (contenido de baja frecuencia) y *detalle de la señal* (contenido de alta frecuencia). El

resultado de descomponer la señal de entrada en versiones paso bajo y paso alto, se conoce generalmente como sub-bandas. Cada una de estas sub-bandas se puede seguir descomponiendo por el mismo procedimiento.

Esta transformada descompone la imagen original en cuatro imágenes submuestreadas o diezmadas. El resultado consta de una imagen que ha sido filtrada mediante un filtro paso alto en vertical y horizontal, otra que ha sido filtrada en paso alto en vertical y paso bajo en horizontal, otra que ha sido filtrada en paso bajo en vertical y paso bajo en horizontal y una que ha sido filtrada en paso bajo en ambas direcciones.

Esta transformada se implementa típicamente en el dominio espacial utilizando filtros de convolución 1-D. Para satisfacer las condiciones de esta transformada, los filtros deben ser filtros de reconstrucción perfecta, lo que significa que cualquier distorsión introducida por la transformación directa, debe ser cancelada por la transformación inversa y deben cumplir algunas otras propiedades.

Entre los filtros propuestos para implementar la TDW uno de los más utilizados es el de Daubechies:

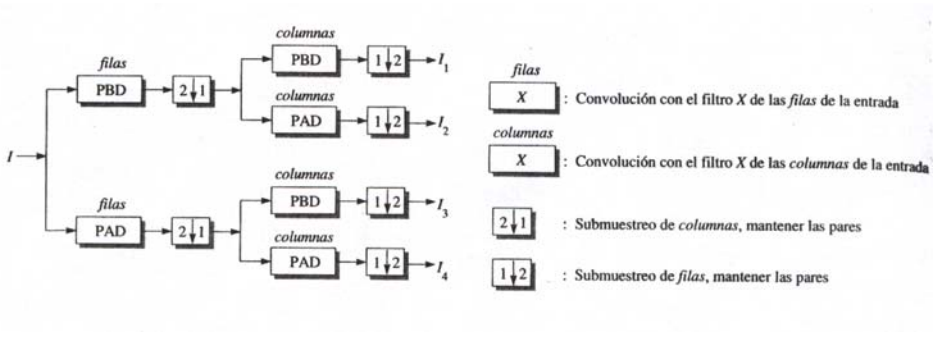
Paso Bajo

$$1/4\sqrt{2}\ [\ 1 - \sqrt{3},\ 3 - \sqrt{3},\ 3 + \sqrt{3},\ 1 + \sqrt{3}\ ]$$

Paso Alto

$$1/4\sqrt{2}\ [\ -1 - \sqrt{3},\ 3 + \sqrt{3},\ -3 + \sqrt{3},\ 1 - \sqrt{3}\ ]$$

Tomando PBD como Paso Bajo Directo y PAD filtro Paso Alto Directo, en el siguiente diagrama se muestra el proceso de filtrado por filas y columnas para la descomposición de la imagen original en cuatro subimagenes.



El proceso inverso trata de reconstruir la señal original a partir de sus elementos previamente descompuestos, la reconstrucción en el caso continuo resulta ser,

$$f(x) = (1/K_\psi) \int_R + \int_R C(a,b)\ (1/\sqrt{a})\ \boldsymbol{\psi}((x-b)/a)\ dadb/a^2$$

donde $K\psi$ es una constante que depende de ψ.

Y en el caso discreto resulta

$$f(x) = \sum_{j \in Z} \sum_{k \in Z} C(j,k)\, \psi_{j,k}(x)$$

Así la TDW inversa se realiza aumentando los datos de la TDW a su dimensión original, para luego insertar ceros entre cada valor, realizar la convolución correspondiente a cada imagen con los filtros inversos a los utilizados y sumar los resultados para obtener la original. Los filtros inversos de Daubechies son:

Paso Bajo

$$1/4\sqrt{2}\ [\ 1+\sqrt{3},\ 3+\sqrt{3},\ 3-\sqrt{3},\ 1-\sqrt{3}\ ]$$

Paso Alto

$$1/4\sqrt{2}\ [\ 1-\sqrt{3},\ -3+\sqrt{3},\ 3+\sqrt{3},\ -1-\sqrt{3}\ ]$$

3.2. Desarrollo de la solución aportada

Como ya hemos mencionado anteriormente, las tres vertientes por las que ha discurrido nuestra investigación han sido transmisión por protocolo TCP, transmisión por protocolo UDP, y tratamiento de imágenes basándonos en los protocolos anteriores. Si bien se han combinado con la utilización de transformada de wavelets para aumentar la velocidad de transmisión.

A continuación explicamos detalladamente cada una de estas tres líneas de trabajo.

3.2.1. Solución aportada mediante el protocolo TCP

La investigación de las soluciones a la problemática inicial de este proyecto, empezó por las transmisiones con este protocolo, principalmente por ser el más extendido y el que en teoría nos planteaba una seguridad total en el envío y recepción de imágenes.

Al implementar el proyecto se han usado sockets, y una arquitectura cliente/servidor, de forma que este proyecto está ideado para que el cliente se corresponda con el robot autónomo, el cual envía información al servidor, que es la estación de control remota. Esta implementación viene fundamentalmente motivada por el hecho de que en todo momento se conocerá la dirección ip de la estación de control (servidor), y por tanto será mucho más fácil para el robot autónomo conectar con esta, ya que por la especificación, no se podía garantizar que la dirección ip de este robot fuese estática o dinámica.

En un principio diseñamos un sistema en el cual el protocolo de control propio de la aplicación era bastante rudimentario de forma que entre cliente y servidor, se enviaban las fechas de inicio y fin de la transmisión, y en el servidor se controlaban una serie de estados muy simples definidos por constantes.

Las constantes que definían los estados, y lo que significaba que la transmisión se encontrase en alguno de ellos es lo siguiente:

- WAITING: Si estábamos en este estado, y se conectaba un cliente iniciábamos la comunicación imprimiendo por pantalla la fecha de inicio de la conexión, y pasábamos automáticamente al estado de transmisión.
- TRANSMISSION: Si estábamos en este estado guardábamos en un fichero de salida la información procedente del cliente, hasta que este cerraba la transmisión, momento en el cual se pasaba al estado de desconexión.
- DISCONNECT: Una vez alcanzado este estado cerrábamos la comunicación, y el servidor pasaba de nuevo al estado de espera (WAITING).

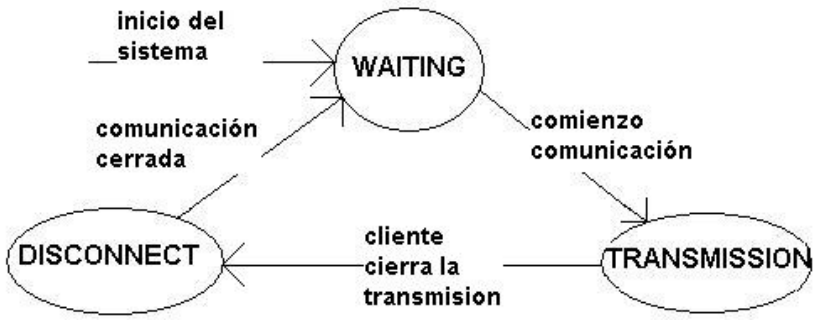


FIGURA 9: Diagrama de estados del protocolo del primer prototipo

Nos dimos cuenta de que esta primera aproximación tenía los siguientes problemas:

- En las reuniones con el equipo del Instituto del conocimiento se propuso la idea por parte de los integrantes de dicho equipo de que el proyecto se dejase listo para que en futuras ampliaciones se pudiese añadir mejoras tales como sonido estéreo en la transmisión, y con esta arquitectura no era posible que permitir en ningún momento la conexión de varios clientes al servidor, por lo tanto, no se podría incorporar la retransmisión simultánea de sonido.
- Como se verá más adelante, interesaba la idea de transmitir ciertas imágenes, o partes de estas siguiendo el protocolo UDP (protocolo de similares características al TCP, pero que no garantiza la integridad de la transmisión), y con esta arquitectura, no se podía garantizar el cambio de estados, ya que era posible que se perdiese la notificación de cambio.

Sin embargo, con este primer diseño, logramos tener una base operativa sobre la que construir posibles mejoras, e incluso nuevas arquitecturas.

Una vez comprobada la eficacia del protocolo en el envío de datos, mutamos nuestra aplicación para el envío de imágenes a través de un archivo. La aplicación empezó y finalizó basando sus estructuras de entrada salida en archivos físicos del sistema, ya que no existe en esta fase del proyecto un medio productor del vídeo estereoscópico objeto de la transmisión.

A continuación comprobamos la eficiencia de transmisión de datos, que cumplía a la perfección las características teóricas asociadas al protocolo TCP, es decir el envío y recepción de la totalidad de los datos de los archivos de salida, sobre ese primer prototipo. Las pruebas consistieron en el envío de archivos de varios formatos y tamaños a través de diversos tipos de redes.

Una vez verificado el correcto funcionamiento de nuestra herramienta, tras intentar definir de forma infructuosa modificaciones a los protocolos antiguos, y nuevos protocolos, decidimos modificar por completo la arquitectura de la aplicación de forma que se satisfacciesen de forma óptimas las características dadas por los organizadores del proyecto.

En este nuevo sistema el servidor seguiría localizado en la base de transmisiones, que es la unidad física donde llegarán todas las imágenes enviadas desde los distintos clientes.

El software que se proporciona al servidor tiene la característica de estar continuamente escuchando a sus puertos.

Esto se debe por una parte a la necesidad de que el receptor esté siempre alerta ante posibles fallos en la conexión con el cliente, ya que el medio de la conexión podría fallar y de esta manera no se tendría que volver a cargar el sistema del cliente y del servidor. Mediante este diseño sólo haría falta cargar el sistema del cliente, esto permitiría que si existen diversos clientes en el sistema, los fallos no fueran globales. Así ante el hipotético fallo de un cliente los demás pudieran seguir en servicio sin ningún problema.

Por otro lado, así aportamos la posibilidad de manejar más de un equipo, es decir, si se quisiese con el mismo software tratar de manera individualizada cada una de las cámaras, o tener más de un equipo funcionando bajo el mismo control, esto es ahora posible con la nueva arquitectura.

La nueva arquitectura del sistema funciona a tres niveles distintos y bien diferenciados:

- En un primer nivel el servidor está escuchando por el puerto que se le asigne, a la espera de la llegada de nuevos clientes. Esto se hace por medio de instrucciones de espera “bloqueantes” para evitar malgastar recursos con sistemas de espera activa. Cada vez que se reciba un nuevo cliente, se iniciara un proceso de gestión de conexión con ese cliente, pero el sistema seguirá a su vez a la escucha del puerto correspondiente esperando la llegada de nuevas conexiones.
- Cada vez que se reciba un cliente, se iniciará con él un proceso de control, es decir, se señalizará la hora de inicio de la conexión, y se

establecerá una conexión de control, por la que se realizará la comunicación, y sincronización entre cliente y servidor. Este proceso contará con los siguientes estados:

- HOLA: Cuando llega esta palabra reservada por parte del cliente, el servidor inicia las gestiones necesarias para inicializar la conexión de control.
- FICHERO: Cuando llega esta palabra reservada el servidor se encarga de cambiar de estado, y abrir un nuevo hilo para la transmisión de la imagen. Almacenando la hora de comienzo de la transmisión.
- FINTRANS: En este estado, el cliente manda esta palabra reservada para informar de que se finalizó la transmisión, en este caso, el servidor gestiona el cierre del hilo por el que se transmitían las imágenes, y el almacenamiento de esta última en su correspondiente archivo. Así mismo el servidor se encarga de almacenar la hora en la que terminó dicha transmisión. En este caso SOLO se finaliza la transmisión, de forma que la conexión entre cliente y servidor puede seguir abierta.
- ADIOS: Cuando el servidor recibe esta palabra reservada, cambia al estado que lleva el mismo nombre, y se encarga de gestionar el cierre de la conexión con el cliente, así como el cierre del puerto de comunicaciones. Cuando ha finalizado le manda al cliente la palabra reservada "Xaoo" indicándole que todo ha terminado correctamente, para que este realice su desconexión de manera apropiada.

- Cuando se dispone a enviar una imagen se abre el tercer proceso, en el cual, por el puerto siguiente al que se usa para control se envía y recibe la imagen.

Cada vez que se pasa de un nivel a otro se abrirá un nuevo hilo para la gestión del nivel.

El esquema de cómo trabajan los distintos niveles es el siguiente:

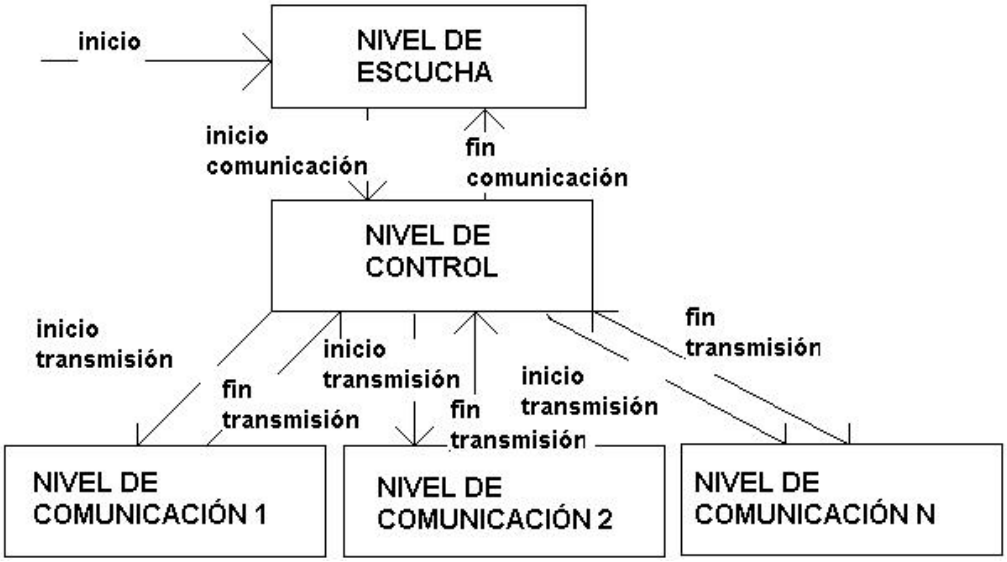


FIGURA 10: Diagrama de niveles de la versión definitiva

En estos momentos, como la aplicación no lo requiere el servidor solo atiende un cliente en cada momento, aunque si por cualquier cosa este se cae (se pierde la conexión), se le atenderá inmediatamente en caso de que quiera reconectar. Para que se escuchen varios clientes a la vez únicamente habría que añadir un nuevo puerto de escucha en para el primer nivel.

Este sistema de control actúa mediante el envío de balizas, que validan los diferentes estados de *last transmission* en los dos puestos (emisor, receptor). De esta manera el protocolo de emisión de balizas sería:

1. El servidor abre el puerto de comunicación de control, y se mantiene a la espera de la conexión de cualquier cliente.

2. El cliente notifica al servidor por el puerto de comunicaciones de control el estado de deseo de transmisión.

3. El servidor recibe la notificación del deseo de transmisión, configura su estado para la recepción de los datos y valida la transmisión al cliente.

4. El cliente recibe la validación de transmisión, configura su estado para el envío, empieza la transmisión por el puerto de transmisión de datos y por el puerto de transmisiones de control envía la fecha y hora del comienzo de la transmisión.

5. El servidor recibe los datos y la fecha de comienzo de la transmisión, y cuando le han llegado la totalidad de los datos, por el puerto de transmisiones de control notifica al cliente la finalización del envío y la fecha y hora de esa finalización.

La aplicación está diseñada para que ambas transmisiones (datos y control) se realicen por puertos diferentes, el puerto de la transmisión de datos siempre será una unidad superior al número de puerto especificado para la transmisión de control.

Además de la independencia de puertos en las transmisiones de control y de datos, también las ejecuciones de las operaciones de transmisión de datos y de control están ejecutándose en hilos paralelos (según la filosofía JAVA), lo que nos garantiza una independencia total entre ambos canales que se verá reflejada en la homogeneidad de transmisión de datos.

Un requisito del diseño de la transmisión de control, es la necesidad de realizarla mediante el protocolo TCP, ya que no se podrían admitir pérdidas en las transmisiones de control porque se descoordinaría toda la transmisión.

A parte del puerto de transmisión de datos es necesario especificar la dirección del servidor en la ejecución del cliente para que este pueda localizar al servidor. Estos parámetros son configurables en tiempo de ejecución por medio de la interfaz de nuestra aplicación. Esta disponibilidad está sujeta a la falta de información sobre direcciones y puertos por los que transmitir, por esto decidimos poder configurarlos en cada ejecución hasta que estos parámetros sean fijados.

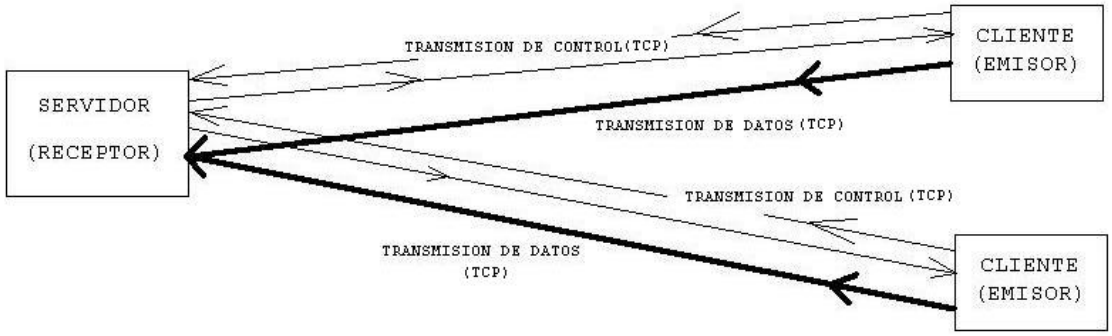


Figura 11: Esquema de la transmisión mediante el protocolo TCP

3.2.2. Solución aportada mediante el protocolo UDP

La solución aportada mediante este protocolo de transmisión, se tomó como alternativa al protocolo de transmisión TCP basándonos en las características aportadas por la documentación teórica previa al desarrollo de esta solución. Estas características nos mostraban un protocolo de transmisión menos seguro, debido a la ausencia de conexión entre emisor receptor, que no garantizaba la llegada de todos los paquetes al receptor.

Al margen de esta característica, en teoría este protocolo podría proporcionarnos una mayor velocidad de transmisión gracias a esa ausencia de comunicación de control propia del protocolo de transmisión.

Así pues fijamos el objetivo de este estudio en la posibilidad de tolerar las pérdidas a cambio de una mayor velocidad de transmisión de datos.

Como en el caso del protocolo TCP, diseñamos una primera arquitectura que posteriormente cambiamos, pero que nos sirvió como una base fiable para posteriormente desarrollar la versión definitiva de la herramienta para este protocolo.

Este primer prototipo únicamente enviaba archivos de un equipo a otro, cerrándose la conexión en el momento que finalizaba el envío del fichero del emisor al destinatario.

No pudimos implementar un protocolo como en el primer prototipo que construimos para TCP debido a que UDP no garantiza la recepción de los paquetes, y por tanto, la información de control podría perderse en una transmisión dando resultados inesperados.

Como solución a los problemas derivados de las características de transmisión de estos protocolos propusimos este otro modelo, que es el que se ha implementado definitivamente.

Para el desarrollo de esta solución diseñamos un sistema que necesitaba, como en el caso del TCP, una transmisión de control independiente de la transmisión de datos, y que a su vez fuera segura ya que tenía que controlar todo el proceso. Por estas razones decidimos acoplar el diseño de la transmisión de control, desarrollado para la resolución del problema mediante el protocolo TCP cuyo funcionamiento ya ha sido explicado en el apartado anterior.

El esquema de esta nueve herramienta consta de otros 3 niveles, como en el caso del protocolo TCP, de los cuales, los dos primeros son idénticos en ambas implementaciones, y solo varía el tercero, que se ajusta a las características de UDP.

Esto nos llevaba a un sistema en el que existen dos transmisiones: la primera (por el puerto especificado) encargada de transmitir todas las balizas de control mediante un protocolo seguro TCP, y una segunda independiente (por el puerto igual al numero de puerto de las transmisiones de control mas uno) encargada de la transmisión de todos los datos por el protocolo UDP.

La transmisión de control esta diseñada análogamente a la transmisión de control para la resolución del problema mediante el protocolo TCP. Únicamente existen diferencias mínimas en la implementación del diseño respecto a la del protocolo TCP.

La transmisión de datos de esta solución se realiza a través del protocolo UDP. En este protocolo tiene la característica no se puede controlar ni la llegada de paquetes, ni el orden en el cual estos llegan al receptor, por lo que en el diseño tuvimos que controlar posibles riesgos como la llegada descontrolada y desordenada de paquetes al receptor y la pérdida de algún paquete por el camino.

De no ser asumidos estos riesgos se podría llegar a la situación de una mala reconstrucción del objeto enviado ya sea por el desorden de sus partes constituyentes o por el extravío de alguna de ellas.

Para controlar la consistencia de los datos recibidos, decidimos diseñar un patrón de paquete, en el cual además de los datos a enviar, también se incluyera un índice el cual nos indicaría a la llegada en que posición debe situarse ese paquete para su reconstrucción.

Además de estos campos en el patrón paquete diseñamos otro más, el cual nos permitiría descartar el paquete si hubiera superado un retardo fijado para su transmisión. Este campo no fue implementado formalmente dado que no se nos facilitaron las medidas de tiempos necesarias para descartar los paquetes. Estas medidas de tiempo están básicamente diseñadas para controlar la característica de tiempo real de la aplicación. De esa manera un paquete recibido sería descartado si su correspondiente objeto a formar ya hubiera sido procesado y por lo tanto el retardo de la transmisión pudiera propiciar su entrada en otro objeto, descontrolando la reconstrucción total de la aplicación. Nosotros salvamos este inconveniente, ya que hemos comprobado que con el tamaño habitual de las imágenes que poseen formato raw, el contador que usamos es suficiente para identificar una cantidad razonable de paquetes, hasta que se vuelva a introducir el mismo identificador.

Una vez explicado el encapsulamiento de los datos en nuestro patrón de paquetes, vamos a explicar el procesamiento que se realiza en cada extremo de la transmisión a los objetos, para su correcto encapsulamiento y reconstrucción.

El procesamiento servido por el cliente (emisor) consiste básicamente en la creación de los objetos del patrón paquete, los cuales contienen un byte para el encapsulamiento del índice (números consecutivos en la misma transmisión, nos aseguramos que el rango fuera lo suficientemente amplio para que no hubiera conflictos de índices en la transmisión) y 1024 bytes para el encapsulamiento de los datos en cada paquete. Estos datos se extraen consecutivamente del objeto a transmitir.

En el receptor la conexión nos sirve objetos que siguen nuestro patrón de paquete, por lo tanto vamos almacenando en memoria los paquetes que nos van llegando para en el momento indicado proceder a la reconstrucción del objeto enviado por el emisor.

Esta reconstrucción se realiza ordenando el contenido de los paquetes en un búfer de salida según el orden indicado por el índice de cada objeto. Los objetos descartados por fallos en su recepción no pueden ser incluidos en la imagen final, esto nos llevaría a un fallo estructural de la imagen en la reconstrucción si no se tuvieran en cuenta.

Lo que diseñamos para solventar esta ausencia de datos, fue la inserción de 1024 ceros por cada paquete con algún fallo en la recepción del mismo, así nos aseguramos la completitud estructural de la imagen con la única pega de la sustitución de los datos perdidos por píxeles negros. Esto se realiza así para todos los paquetes que llegan al servidor, excepto con el último, ya que como el tamaño de la imagen es conocido, solo se rellenarán con ceros (en caso de no llegar este último paquete) los bytes que se corresponderían con el tamaño de este.

Aparte de todo esto también hay que reseñar que para que exista la mayor independencia del envío de datos respecto a todas las demás tareas a realizar por la aplicación, cada una de estas se ha diseñado para que sea ejecutada en un hilo de ejecución diferente. Esto nos asegura además de una total independencia, una mayor eficiencia en los envíos, debido a la ausencia de retardos producidos por tareas externas a la transmisión sobre ella misma.

Para finalizar la explicación de esta solución aportada, también hay que señalar que existe una interfaz gráfica que ayuda al usuario a configurar parámetros como puertos, dirección del servidor y los objetos destino y final necesarios para la correcta ejecución de la aplicación.

Esta interfaz se ha desarrollado con una imagen análoga a la de la solución aportada por protocolo TCP para una mayor facilidad de manejo de ambas y una comparación más sencilla.

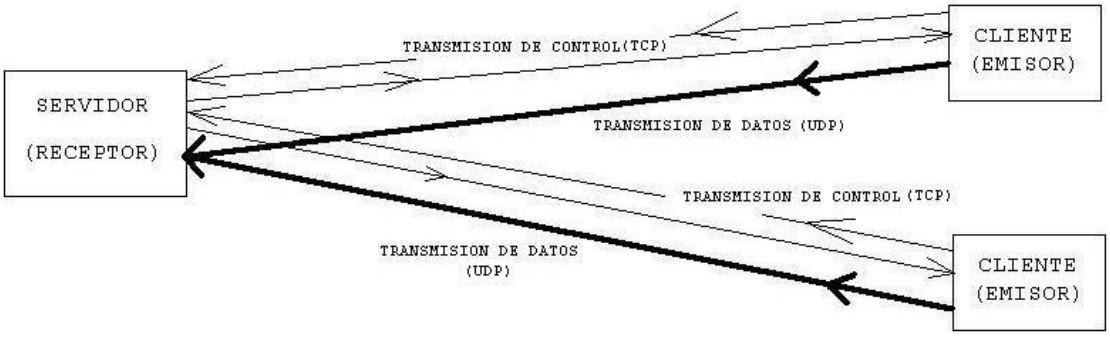


Figura 12: Esquema de la transmisión mediante el protocolo UDP

3.2.3. Solución aportada mediante el filtro Wavelets

Después de documentarnos en diversas técnicas de procesamiento de imagen decidimos optar por la aplicación de la transformada de Wavelets, en su versión discreta aplicando los filtros de Daubechies.

Este procesamiento tenía la función de reducir la imagen a un cuarto de su tamaño antes de ser enviada. Esto se consigue mediante el filtro paso bajo – paso bajo que mantiene la aproximación de baja resolución, pero pierde el detalle de la señal, es decir, mantiene la información de intensidad de la original pero difumina los bordes. Debido al uso pretendido para esta aplicación (la visualización por parte de un operador humano en tiempo real) esto parece ser una buena opción, dado que el ojo humano difícilmente distingue entre la imagen preprocesada y la postprocesada.

En la figura 13 se muestra un ejemplo de imagen original representativa del conjunto de imágenes utilizadas durante el desarrollo del proyecto; la imagen comprimida, la que se transmite entre emisor y receptor; y la imagen descomprimida por el receptor aplicando la transformada inversa de wavelets. Se trata de una imagen en escala de grises con un byte de profundidad (256 niveles de intensidad) para cada píxel, y un tamaño de 600*800 píxeles. La imagen comprimida tiene por tanto 300*400 píxeles de tamaño.

Figura 13. Imagen original.

Figura 14. Imagen comprimida.

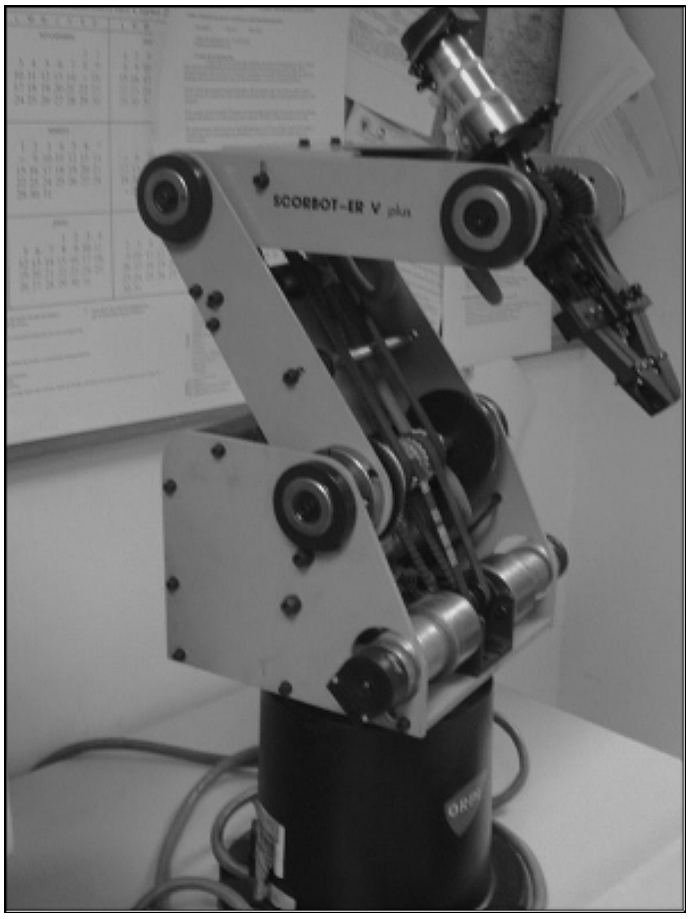


Figura 15. Imagen descomprimida.

Como se puede observar las formas de los objetos y la claridad de éstos se pueden distinguir perfectamente en la imagen resultado (figura 15), sin embargo los textos presentes en la imagen original están ligeramente más difusos en la imagen resultado.

El tiempo necesario para el procesamiento de las imágenes se reparte entre emisor se reparte entre la maquina emisora y la receptora, lo cual ayuda a la minimización del tiempo total transcurrido entre recepción de la imagen en el emisor y su exposición en el receptor, necesario para la percepción de transmisión en tiempo real en el caso de que se transmitiera una secuencia de imágenes.

En los siguientes apartados mostraremos unos breves aspectos teóricos sobre los caminos utilizados en el proyecto, así como la relación entre ellos en este proyecto y su modo de uso en el mismo.

4. Diseño Software del proyecto

A la vista de las consideraciones expuestas hasta este momento, el proyecto se ha centrado en el desarrollo de una aplicación Software donde se contemplan todos los aspectos relatados en las secciones anteriores.

El proyecto se fundamenta en el desarrollo del ciclo de vida de un proyecto software tal y como se contempla en la Ingeniería del Software. A continuación se proporcionan los detalles relativos al mencionado ciclo de vida, que contempla: Arquitectura del sistema, Diseño e Implementación.

4.1. Arquitectura del Sistema

El sistema de transmisión de video esta compuesto por un sistema de captación (maquina cliente) y un sistema de recepción (maquina servidor). Ambas maquinas tienen que ser totalmente compatibles con Java y disponer de un entorno en tiempo de ejecución Java instalado.

La comunicación entre cliente y servidor se realiza por medio de una red TCP/IP en la que existe una conexión posible entre ambas maquinas.

El paquete ServidorTCP/UDP estará instalado sobre la maquina servidor. El paquete ClienteTCP/UDP estará instalado sobre la maquina cliente.

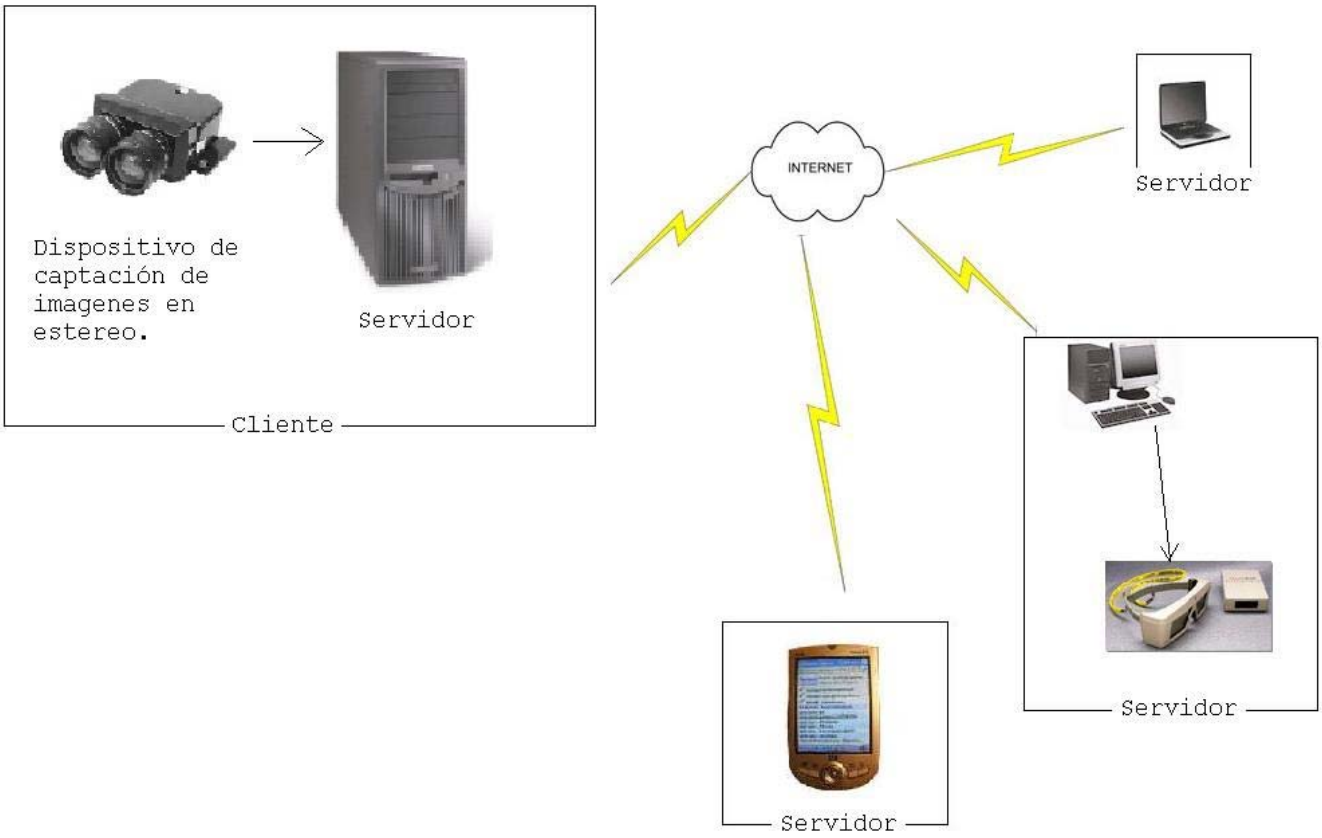


FIGURA 16: Arquitectura del sistema

4.2. Diseño orientado a objetos

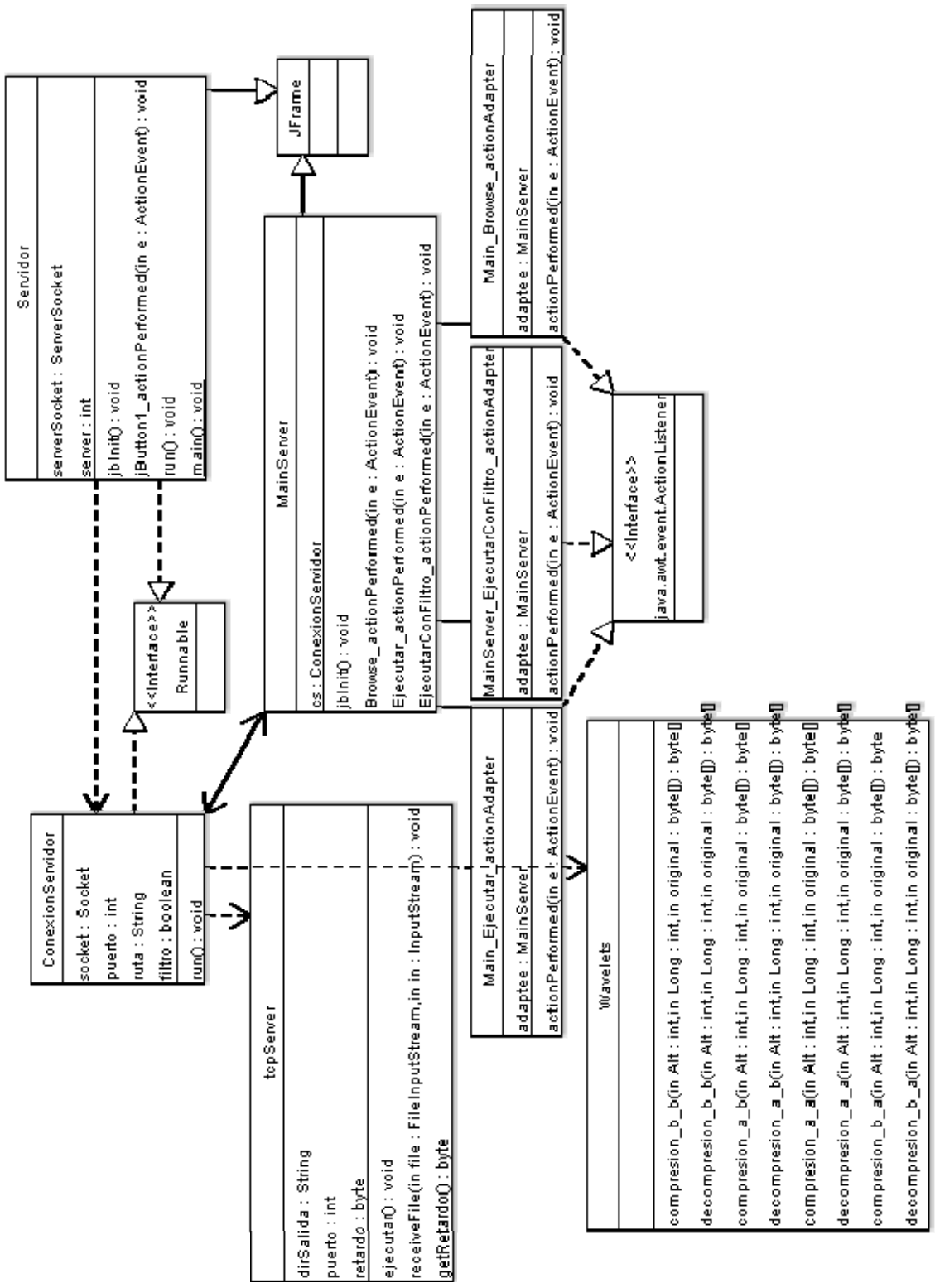


Figura 17 : Diagrama de Clases del Servidor TCP

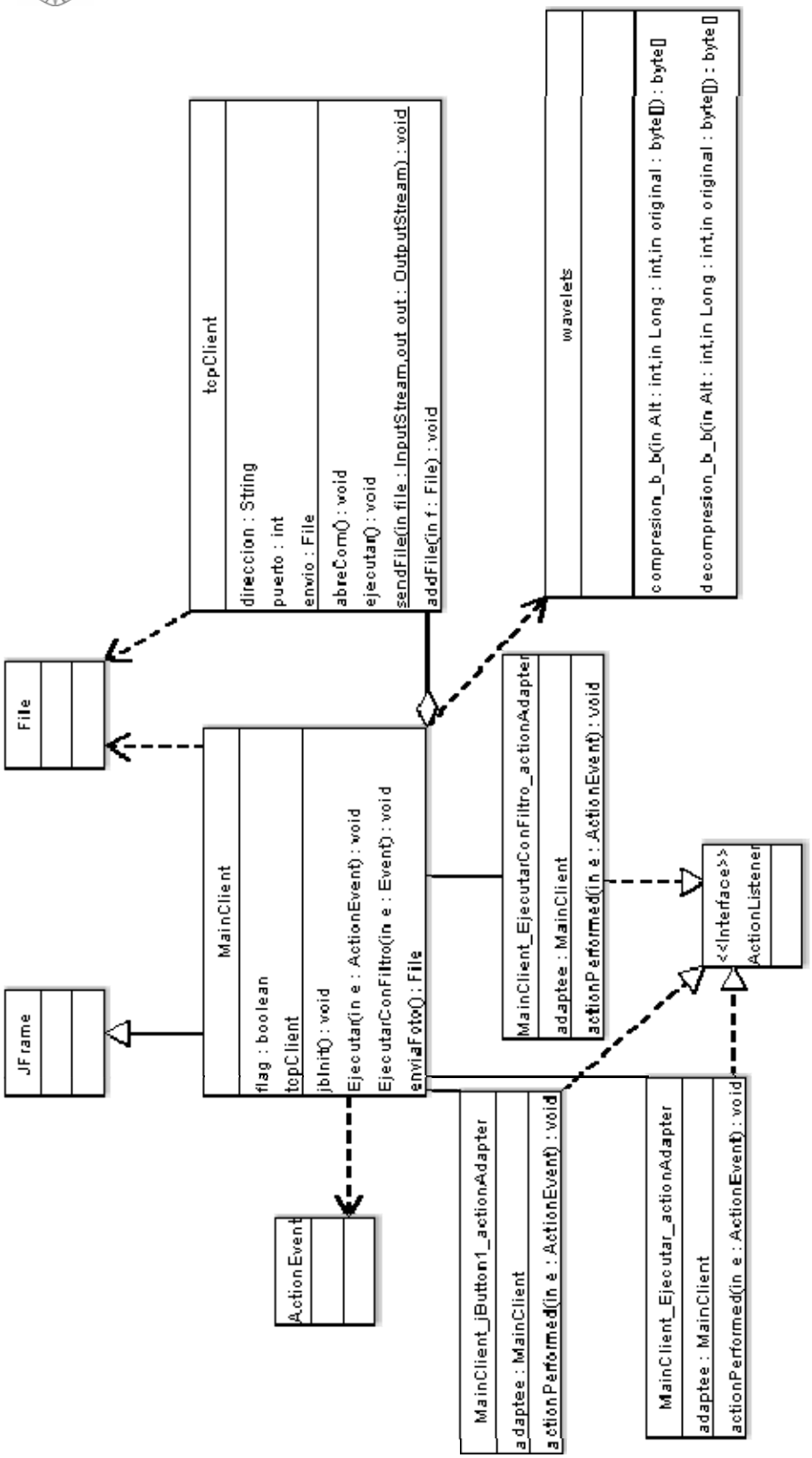


Figura 18: Diagrama de clases del cliente TCP.

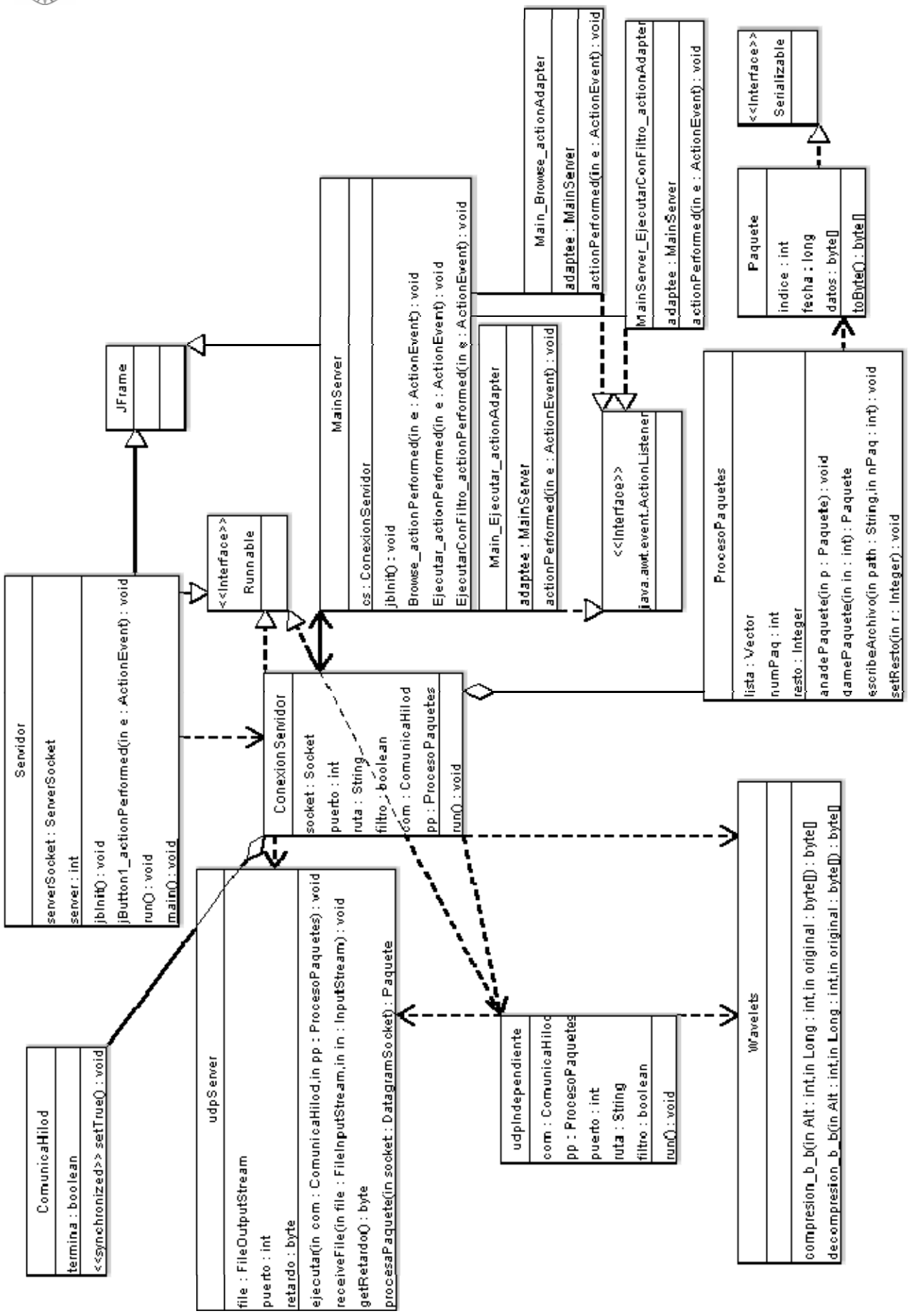


Figura 19: Diagrama de clases del Servidor UDP.

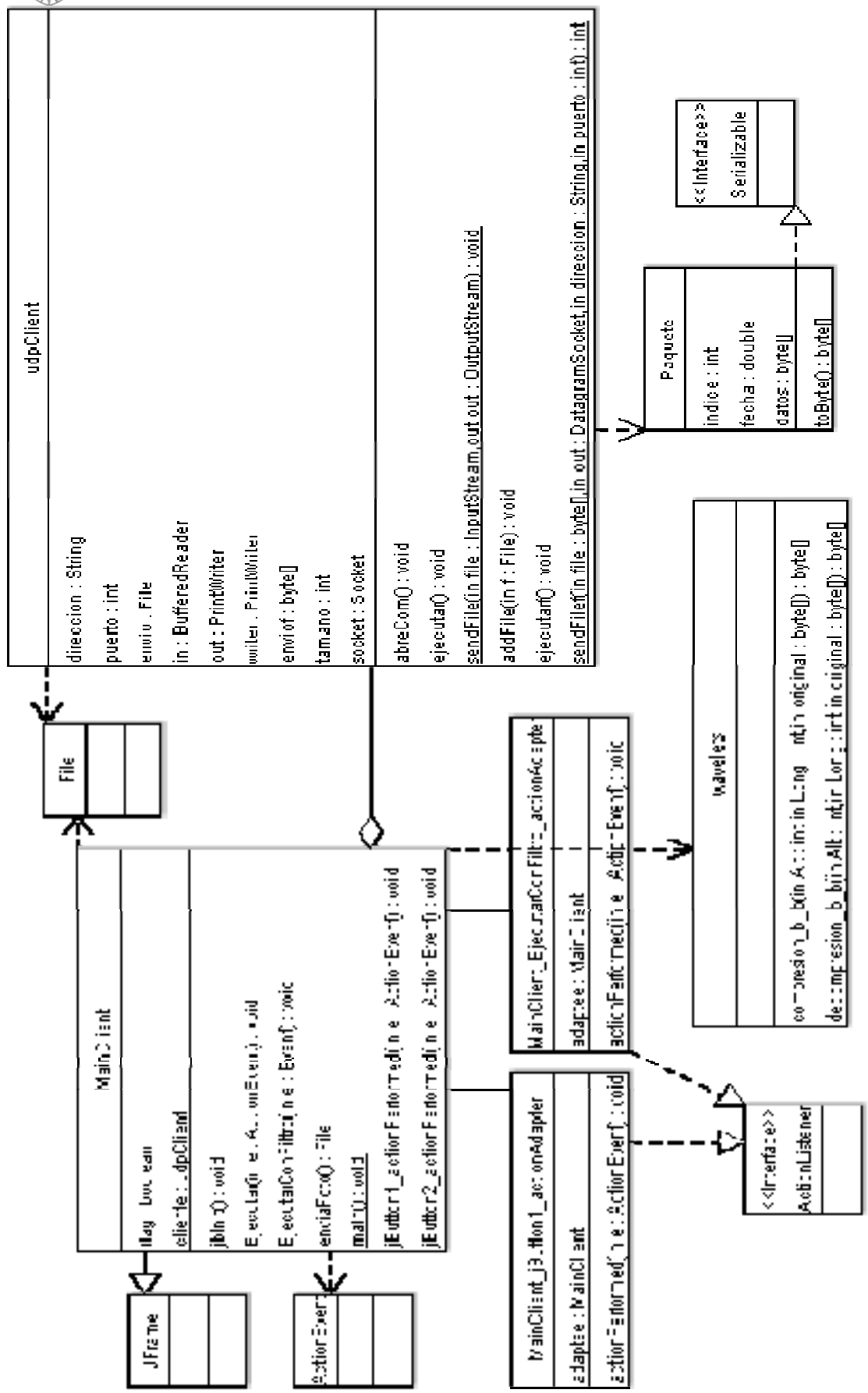


Figura 20: Diagrama de Clases del Cliente UDP.

4.3. Detalles de implementación

El proyecto se ha dividido básicamente en tres módulos:

- Transmisión de datos e imágenes mediante la implementación de una herramienta sobre el protocolo TCP
- Transmisión de datos e imágenes mediante la implementación de una herramienta sobre el protocolo UDP
- Compresión y descompresión de imágenes empleando un algoritmo basado en WAVELETS.
- Creación de una interfaz intuitiva y amigable para el usuario mediante la cual se ejecuten cada una de las herramientas.

Más adelante dentro de este apartado, daremos los detalles de implementación tanto de los protocolos de envío como de la compresión y descompresión de imágenes, pero primero explicaremos el uso de la interfaz para el usuario, explicando también los elementos empleados para su implementación.

4.3.1. Implementación de la interfaz gráfica

La interfaz gráfica está realizada en JAVA heredando dela clase Jframe, y usando librerías proporcionadas por Borland para generar los layouts.

Cada una de las clases que generan alguna parte de la interfaz gráfica presentará una estructura en la que se observarán al comienzo de esta, los atributos propios de una clase gráfica, como son botones, etiquetas, menús, campos de texto, etcétera.

Todas las clases que tienen una parte gráfica presentarán un método llamado jbInit, que a demás será el primero al que llame la constructora de la clase, el cual se encargará de configurar la apariencia del gráfico, es decir, posicionará los botones, etiquetas, y campos de texto en su posición correcta, será la que marque cual será el layout de la imagen, y posicionará correctamente los paneles de la interfaz en la pantalla del equipo. Por último este mismo método se encargará de dimensionar, y hacer visibles en su debido momento los elementos gráficos.

Una labor muy importante del método jbInit consistirá en la inclusión de comandos de la forma :

```
ElementoGráfico.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      Elemento_actionPerformed(e);
    }
```

```
});
```

Los cuales se explican a continuación.

Todas las clases gráficas poseen varios métodos orientados a eventos, es decir, cuando ocurre el suceso que estaban esperando (por ejemplo hacer click en un botón de la interfaz gráfica, o cerrar alguna de las ventanas, o componentes gráficos de esa clase) ejecutarán la secuencia de instrucciones que el método tenga previsto para el evento en cuestión. Estas funciones están caracterizadas por que (esto se debe a nuestro estándar de código) están nombradas de la siguiente manera: Nombre_actionPerformed(ActionEvent e). Donde "Nombre"hace referencia a algún rasgo significativo del objeto o evento esperado, y "e" es el evento que se esperaba que se pasa como parámetro.

A continuación vamos a representar una secuancia de ejecución tanto para la herramienta TCP, como para la herramienta UDP donde se mostrará la actuación de las interfaces gráficas, intercalando el uso que se haría por parte tanto del cliente como del servidor (como se ha dicho antes en cada secuencia de ejecución es necesario que tanto cliente como servidor estén sincronizados, y se esperen a que el otro realice una labor antes de comenzar la suya).

Para la herramienta que implementa el protocolo TCP, la secuencia de funcionamiento es esta:

En primer lugar se iniciará el servidor en el centro de transmisiones, y lo primero que pedirá el programa será que el operador inserte el puerto por el cual se estará escuchando, a la espera de que un cliente (el robot que se encuentra en un lugar remoto se conecte a él). La figura siguiente muestra cual es la interfaz que recoge estos datos.

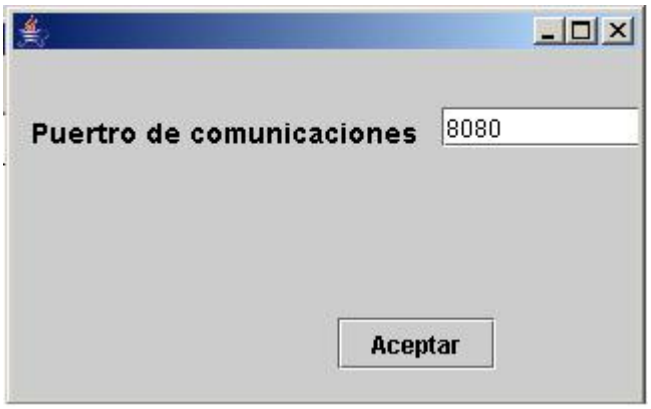


FIGURA 21: Interfaz que recoge el puerto de escucha

Esta imagen está implementada en la clase Servidor.java, la cual pertenece al código de la herramienta que actúa como servidor en el protocolo TCP. Una vez presionado el botón aceptar, la clase generará un nuevo hilo, que será el encargado de esperar la conexión del cliente.

Una vez el servidor está en estado de "espera" será ahora el cliente (el robot) el que deberá inicializarse, y deberá conectarse. Cuando de ejecuta el cliente, aparecerá en la pantalla el siguiente gráfico:

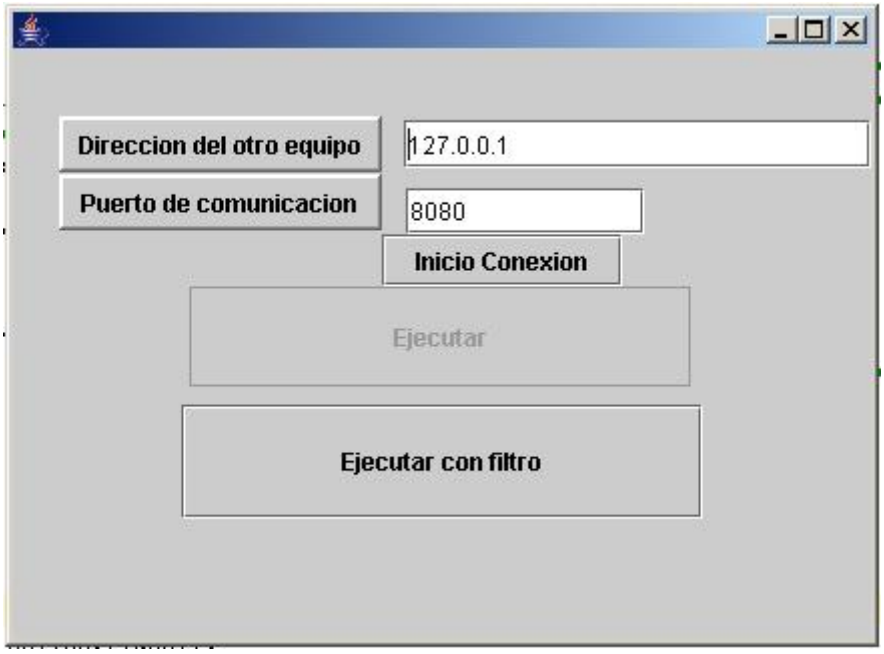


FIGURA 22: Interfaz que recoge el puerto de conexión y la dirección ip.

Esta imagen es generada por la clase MainClient.java perteneciente al cliente del protocolo TCP; en ella se recogen la dirección ip a la que se quiere conectar el cliente (es decir, hay que introducir la dirección ip del servidor), y el puerto por el que se realizará la conexión. En este caso, os valores que tiene la imagen por defecto se corresponden con 127.0.0.1 para la dirección ip, y 8080 para el puerto de conexión, pero estos valores pueden ser cambiados por el usuario. Una vez introducidos estos valores, el siguiente paso consistirá en pulsar sobre el botón de "Inicio conexion", para establecer el contacto con el servidor. Como se puede ver, en este caso está desactivado el botón de "Ejecutar" esto es debido a que en este momento no se requiere su utilización, pero más adelante sí.

El servidor reconocerá la llamada del cliente, e inicializará la conexión. En este momento en el equipo donde se halle el servidor aparecerá la siguiente imagen para seleccionar la ruta en la que se desee guardar el archivo.

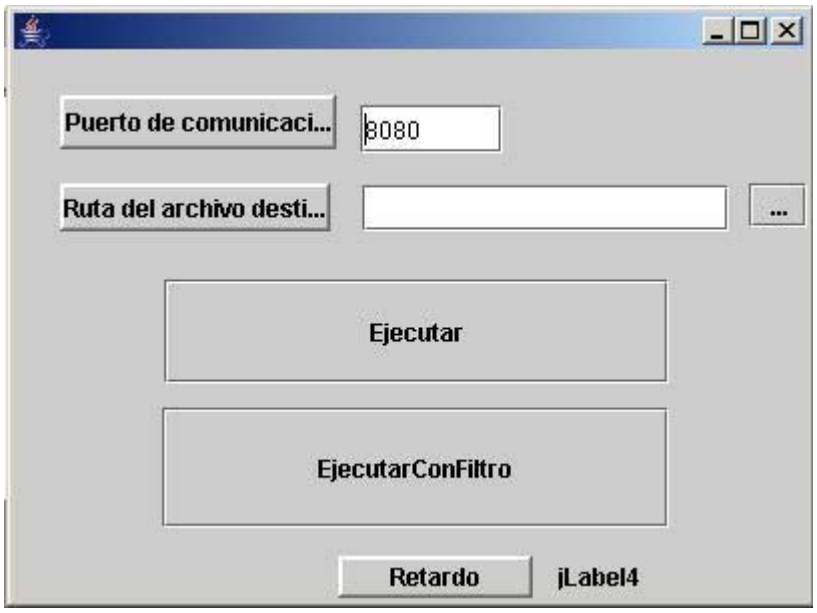


FIGURA23: Interfaz que recoge la ruta para guardar el archivo.

Esta imagen esta generada por la clase Mainserver.java perteneciente al servidor.

En esta imagen se ve que aparece el puerto de comunicaciones; este dato es meramente indicativo, ya que aunque el usuario cambie su valor, el programa seguirá empleando el puerto que se eligió en un principio.

Para la elección de la ruta del archivo a guardar se pueden emplear dos opciones:

- Insertar en el cuadro de texto la ruta directamente desde teclado.
- Pulsar el botón que aparece a la derecha del cuadro de texto donde aparece la siguiente imagen: "...".

Si se pulsa el botón se accederá a otra interfaz gráfica para elegir visualmente la ruta destino. Esta otra imagen se corresponde con esta segunda interfaz gráfica:

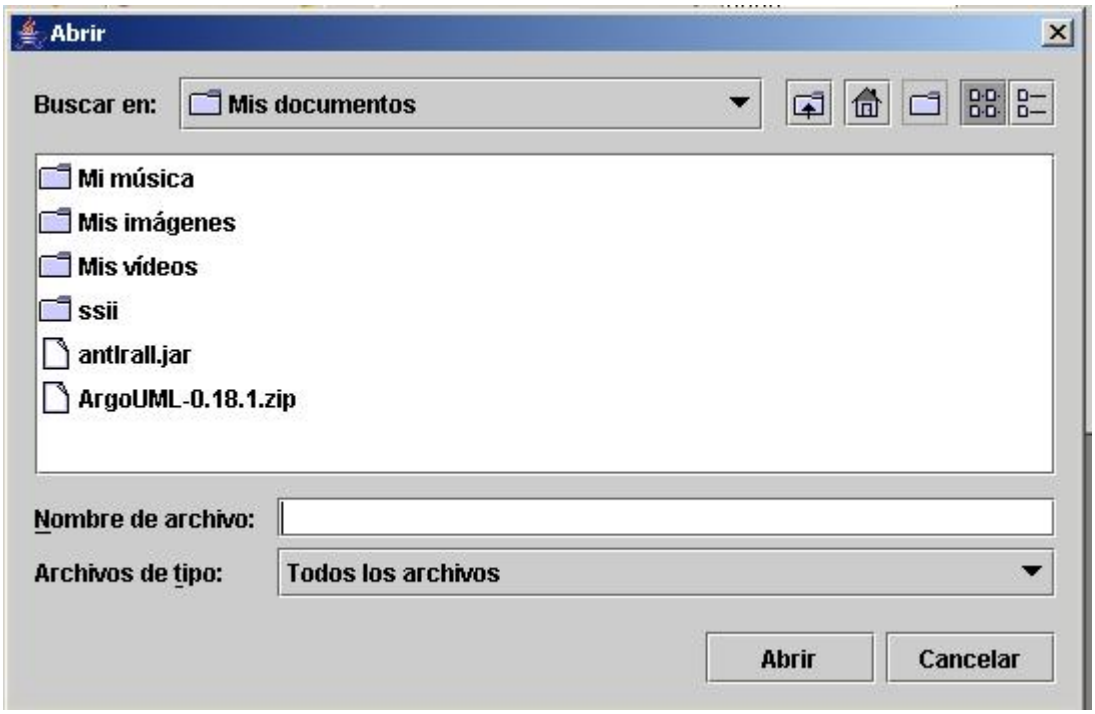


FIGURA 24: Interfaz que recoge la ruta para guardar el archivo

Aquí como se ve se puede elegir interactivamente el archivo a abrir. Si el archivo donde se va a guardar aún no se ha creado, lo nombraremos posicionandonos en el directorio donde queremos generarlo, y tecleando el nombre y la extensión en el campo de texto situado a la derecha de la etiqueta "Nombre de archivo". Una vez seleccionado pulsaremos el botón "Abrir", momento en el cual se cerrará esta interfaz, y aparecerá en la interfaz raíz, en el campo de texto situado a la derecha de la etiqueta "ruta del archivo destino" el nombre completo del archivo seleccionado (ruta completa donde se encuentra).

Después de seleccionar el archivo de destino se deberá pulsar sobre los botones "Ejecutar" o "Ejecutar con filtro" dependiendo de si se desea enviar la imagen sin comprimir o comprimida mediante el paso del filtro WAVELETS.

Cuando se haya realizado este paso en el servidor, en el cliente será el momento de enviar la imagen al servidor.

La interfaz qe aparecerá en ese momento en el cliente será la que se corresponde con la siguiente imagen.

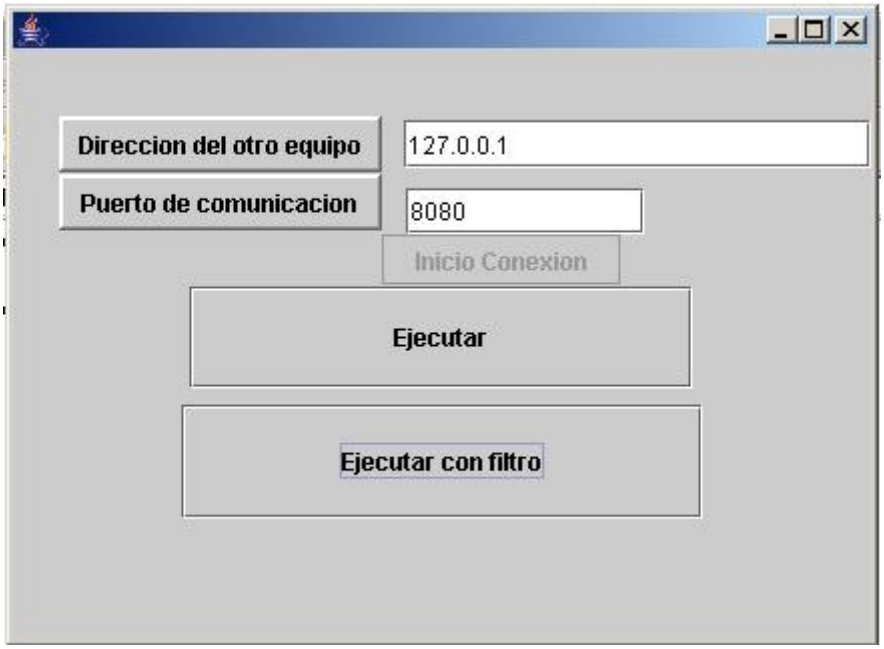


FIGURA 25: Interfaz que selecciona el modo de envío en el cliente.

Esta figura está generada por la clase Mainclient de JAVA perteneciente al cliente.

Los campos de "Dirección del otro equipo", y "Puerto de comunicación" aparecen a modo de recordatorio, de modo que si el usuario cambia alguno de esos parámetros, el programa ignorará dichos cambios, y seguirá empleando los valores previamente asignados.

Se puede ver que se ha desactivado el campo de Inicio de Conexión, esto se debe a que la conexión está ya establecida, y si se pulsase por error daría lugar a errores de ejecución.

Al igual que en el servidor, el usuario dispone de dos opciones de envío (ojo, debe de seleccionarse la misma opción de envío que se eligió en el servidor para el correcto funcionamiento de la aplicación). La opción de "Ejecutar" se corresponde con el envío de la imagen sin ningún tipo de filtrado, por otro lado la opción de "Ejecutar con filtro" se corresponde con el envío de la imagen previamente filtrada por medio de WAVELETS.

Una vez finalizado el envío, la aplicación está configurada para que el cliente envíe la señal del fin de transmisión, y acto seguido envíe la señal que indica el fin de la conexión.

Durante la transmisión el servidor ha ido sacando por pantalla los momentos en los que se inicio la conexión, el momento en el que se inició la transmisión de la imagen, y el momento en el que se finalizaron la transmisión de la imagen y la conexión. Todo esto se puede apreciar en la siguiente figura:

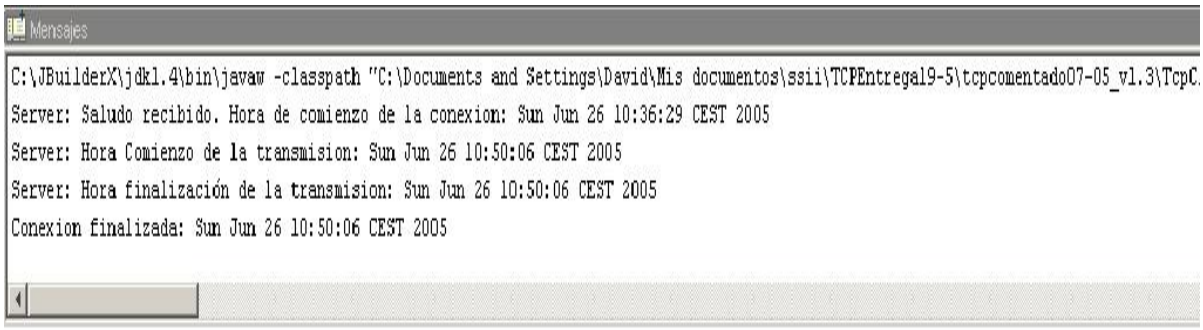


FIGURA 26: Datos de sincronización entre cliente y servidor

Para la herramienta UDP se empleo una interfaz gráfica muy similar, siendo además los pasos para la ejecución de los envíos de imágenes los mismos, con la diferencia de que en este caso el protocolo en envío es UDP. Lo único que varía es que el servidor saca por pantalla el número de paquetes recibidos, y el cliente informa de los enviados, como se ve en la siguiente figura:

FIGURA 27: Datos de sincronización cliente servidor UDP; el servidor informa de los paquetes recibidos.

4.3.2. Implementación del protocolo TCP

Como ya se describió en apartados anteriores, la arquitectura de la aplicación está dividida en tres niveles claramente diferenciados. En este apartado se describirá desde el punto de vista del lenguaje utilizado (JAVA) la implementación de la herramienta que envía imágenes empleando dicho protocolo.

La implementación del servidor se ha realizado de la siguiente forma:

El hilo principal de la aplicación (el que se ejecuta al iniciar la herramienta) comienza en la clase "Servidor.java"; desde aquí se selecciona el puerto por el que se esperara la llegada de un cliente. Cuando se selecciona el puerto de escucha, esta clase crea un nuevo hilo de ejecución para realizar la escucha.

La generación del hilo nuevo se realiza por medio de la interfaz de JAVA "Runnable".

La espera de recepción de un nuevo cliente se realiza dentro de un bucle infinito, pero con espera bloqueante (para evitar malgastar recursos), de forma que si se quiere, la aplicación puede gestionar el funcionamiento de varios clientes al mismo tiempo.

Una vez que se ha registrado un cliente, la aplicación crea un nuevo objeto de la clase "ConexiónServidor.java". Esta clase implementaría el llamado "segundo nivel" de la transmisión, nivel que realiza la comunicación de control entre el cliente y el servidor.

Cuando el nuevo cliente se ha registrado, ConexionServidor genera un nuevo hilo (también por medio de "Runable") con el fin de dar cobertura en exclusiva a esta nueva conexión, dejando a la espera al hilo padre para gestionar la llegada de nuevas conexiones.

La comunicación con el cliente de control y sincronización se llevará a cabo por medio de las directivas de control descritas en el apartado 2.2.1 de esta misma memoria.

Cuando el servidor recibe la primitiva "HOLA" generará un nuevo objeto de la clase "MainServer.java", clase que como hemos visto antes mostrará una pequeña interfaz gráfica donde el usuario seleccionará la ruta donde se almacenará la información que le llegue, así como el modo de ejecución (con o sin el uso del filtro WAVELETS)

Cuando el cliente decide enviar una imagen, se lo comunica al servidor por medio de la primitiva "FICHERO". En este momento el servidor dependiendo de si se marcó la opción de filtro o no, realiza dos caminos distintos.

En ambos casos se realiza un paso común, que es la generación de un nuevo objeto perteneciente a la clase "tcpServer.java".

En la clase tcpServer se abrirá un nuevo objeto de la clase Socket (clase que implementa en java el protocolo TCP). Este socket escuchará por el puerto siguiente al que se estuviese empleando para las transmisiones de control.

La clase tcpServer recibirá y almacenará la imagen enviada por el otro equipo (cliente) en forma de paquetes.

El objeto de la clase tcpServer se mantendrá en funcionamiento hasta que, por el puerto de control, el cliente mande al servidor la orden de cerrar la transmisión.

Cuando haya finalizado la transmisión se podrán seguir dos caminos. Si se ha elegido la opción de WAVELETS se llamará a la función correspondiente de la clase Wavelets (concretamente el método es "decompresion_b_b"), función que devolverá un

array de bytes que será el que posteriormente almacenaremos en la ruta especificada por el cliente.

Si no se eligió la opción de Wavelets se alamcenará en la ruta especificada por el usuario directamente lo que devuelva el objeto de la clase tcpServer.

En este momento el cliente podría volver a enviar una imagen, en cuyo caso se volvería a repetir el proceso anteriormente mencionado, o bien podría solicitar la desconexión enviando la primitiva correspondiente, en cuyo caso el servidor cortaría el hilo dedicado al cliente, y le mandaría al cliente la primitiva "Xaoo" para que este cerrase la comunicación.

La implementación del cliente TCP es la siguiente:

La clase donde se encuentra el método principal es "MainClient.java". Esta clase genera una interfaz gráfica, que como se vio en el apartado anterior sirve para introducir el puerto de comunicaciones y la dirección ip del servidor.

Al pulsar el botón "Inicio Conexión" se genera un nuevo objeto perteneciente a la clase "tcpClient.java". Este nuevo objeto se encargará de realizar la conexión de control con el servidor por medio de un objeto de la clase socket, y enviará la primitiva "HOLA" al servidor para que este vaya preparándose para el posterior envío de una imagen.

Una vez se ha realizado el inicio de la conexión con el servidor, como se ha visto en otros apartados, en la interfaz gráfica que mostró por pantalla la clase "MainClient.java" desactivará la opción de iniciar conexión de su gráfico, y activará las opciónes de "Ejecutar" y "Ejecutar con filtro".

En este momento, dependiendo de la opción que elija el usuario se podrán seguir dos caminos:

- Ejecutar
- Ejecutar con filtro

Si se elige la opcion de "Ejecutar", una vez seleccionado el archivo a enviar, se llamará al método "ejecutar" del objeto de tcpClient. Este método comenzará enviándole la plabra reservada "FICHERO" al servidor para que se prepare para recibir la información que le mande el cliente, y una vez hecho esto enviará a través del puerto siguiente al que se eligió en el cliente como control, los paquetes correspondientes a la imagen.

Si por el contrario se elige la opción de "Ejecutar con filtro" se creará un nuevo objeto de la clase "Wavelets.java", y se le pasará el archivo a filtrar con una llamada a la función "compresion_b_b", y se enviará el fichero por medio de la función "setEnvioF" del objeto tcpClient, cuyo funcionamiento es análogo al de la función "ejecutar" perteneciente a ese mismo objeto, pero con la salvedad de que "setEnvioF" envía un array de buffer (a diferencia de ejecutar que envía un objeto del tipo FILE) correspondiente a la imagen filtrada.

4.3.3. Implementación del protocolo UDP

La implementación del protocolo UDP comparte algunas clases con la del protocolo TCP debido a que se usa la misma interfaz gráfica, y a que como ya se ha descrito en otros puntos de la misma memoria la información de control y sincronización es compartida con el protocolo TCP, e implementado con las clases propias de este.

La implementación de la parte correspondiente al servidor de la herramienta es la siguiente:

La clase "Servidor.java" es análoga a la que lleva el mismo nombre en el protocolo TCP, y realiza las mismas funciones, e igualmente cuando intenta conectarse a ella un nuevo cliente, genera un objeto de la clase "ConexionServidor.java".

"ConexionServidor.java", como en el caso del protocolo TCP, generará un nuevo hilo para el tratamiento en exclusiva de la información de control y sincronización de cliente y servidor.

El comportamiento de "ConexionServidor" con las primitivas "HOLA", y "ADIOS", y "FINTRANS" es análogo al de su homónimo en el servidor TCP, pero en el caso de la primitiva "FICHERO" varía sensiblemente, ya que genera un nuevo objeto de la clase "udpIndependiente.java".

"UdpIndependiente.java" genera un hilo nuevo para llevar a cabo la transferencia de información por el protocolo UDP.

Es en esta clase (udpIndependiente) donde en UDP se hace la discriminación entre las opciones de ejecución filtrada con WAVELETS, y la opción de ejecución sin la presencia de dicho filtrado.

En el nuevo hilo generado por esta última clase se creará un nuevo objeto de la clase "udpServer.java" cuyo fin último consistirá en la realización de la recepción siguiendo el protocolo UDP.

Las clases de java empleadas para la recepción de la imagen por UDP se corresponden con "DatagramSocket", y se llaman en el objeto de la clase udpServer.

Para realizar la ejecución sin filtro el objeto de udpIndependiente llamará al método "ejecutar" del objeto de la clase udpServer, el cual por medio del objeto de la clase "DatagramSocket" recibirá la información en forma de arrays de bytes, y la almacenará en objetos de la clase "Paquete.java", que a su vez serán insertados en un tercer objeto perteneciente a la clase "ProcesoPaquetes.java".

La clase "Paquete.java" se encarga de procesar cada array de bytes recibido en udpServer. Para esto, toma de este array su primera posición que se corresponde con el

índice que ocupa dentro de la imagen, y lo almacena en un campo del objeto llamado "índice".

El resto del array (que se corresponde con la información enviada) lo guarda en otro atributo del objeto llamado "datos".

La clase "ProcesoPaquetes.java" se encarga de ir almacenando (y ordenando) todos los paquetes recibidos hasta el momento, y cuando se cierra la transmisión rellena con ceros las posiciones de paquetes que faltaban por llegar, de forma que la imagen recibida tenga el mismo tamaño que la original, y todos los paquetes recibidos ocupen la misma posición que ocupaban en la imagen de referencia.

Para almacenar los paquetes la clase "ProcesoPaquetes" se vale de unas estructuras definidas en la API de JAVA llamadas "Vector", que implementan listas ordenadas de una forma muy eficiente.

Para preservar la "exclusión mutua" a la hora de tratar los objetos de Paquetes, nos valemos de la clase "comunicaHilod" que implementa un monitor, e impide que más de un proceso modifique atributos de un paquete al mismo tiempo.

Una vez concluida la transmisión se almacenará la lista contenida en el objeto de ProcesoPaquetes (habiendo incluido los ceros correspondientes en las posiciones vacías) en la ruta del archivo especificada por el usuario.

En el caso de querer transmitir bajo el modo de ejecución en el que se incluye el filtrado mediante WAVELETS después de ejecutar el método "ejecutar" de la clase udpServer, cuando se reciben todos los bytes se ejecuta el método de "decompresion_b_b" que nos devuelve una imagen similar a la transmitida originalmente en unn buffer de bytes, y estos a su vez se incluyen en unh archivo.

La implementación del cliente se describe así:

El método principal del programa se encuentra en la clase"MainClient.java", la cual a su vez posee una interfaz gráfica análoga a la de su homónima de la herramienta que implementa el protocolo TCP.

Este método se diferencia del de la aplicación TCP en los métodos que implementan la recepción de eventos por parte de los botones "Inicio conexión", "Ejecutar", y "Ejecutar con filtro".

Dentro del método que implementa la ocurrencia del evento de clickear en el botón "Inicio conexión", se crea unh objeto perteneciente a la clase "udpClient.java".

La clase "udpClient.java"es la encargada de la realización de las labores de control y sincronización con el servidor, así como de realizar la transmisión de imágenes bajo el protocolo UDP.

Al crearse un objeto de la clase udpClient, se abre una conexión con el servidor bajo el protocolo TCP (esto solo se usa para el control), y se le envía la palabra reservada "HOLA" para que el servidor se prepare para la recepción de una imagen.

Si se presiona el botón de "Ejecutar" perteneciente a la clase MainClient se deberá seleccionar la imagen a enviar de un archivo, y se le pasará al objeto udpClient; llamando seguidamente al método "ejecutar"de esta última clase.

El método ejecutar perteneciente a udpClient envía la palabra reservada "FICHERO" al servidor para que este se ponga en estado de recepción (bajo formato UDP); seguidamente el servidor envíara (bajo protocolo TCP) información referente al momento de inicio de la transmisión (esto se emplea como sincronización para evitar que el cliente se ponga a emitir antes de que el servidor esté listo para recibir), y sacará por pantalla dicha información.

Después de realizar las labores de control y sincronización, el objeto de udpClient se creará un elemento perteneciente a "DatagramSocket" para transmitir bajo protocolo UDP los datos de la imagen de salida, y comenzará a enviar.

Una vez finalizada la trasmisión, se enviará al servidor la primitiva "ADIOS" para cerrar la conexión.

Si en lugar de presionar sobre "Ejecutar" se hubiese presionado sobre "Ejecutar con filtro", se habría invocado previamente a la función "compresion_b_b" perteneciente a un objeto de la clase "Wavelets", y con el array de bytes devuelto por este método se hubiese llamado a la función "ejecutarf" perteneciente a la clase udpClient, cuya función es análoga a la de "ejecutar", pero que envía un array de bytes en lugar de un fichero.

4.3.4. Implementación de la transformada wavelets

La implementación de la transformada wavelets esta basadas en la implementación de cada una de las transformadas de compresión y descompresión en un metodo distinto.

Los algoritmos de compresión siguen el siguiente esquema;

Fijados los correspondientes coeficientes del filtrado correspondiente se realiza una conversión de los datos de entrada de array de byte a array de double.

A partir de los tamaños de la entrada se realizan los cálculos correspondientes para hallar las dimensiones de los arrays intermedios y de la solución. Siendo la solución del doble de alto como de ancho respecto al original, pero necesitandose unos arrays intermedios un píxel más grandes.

A continuación aplicamos la formula de la TDW (ver apartado 2.2.3 de este documento) por filas sólo para aquellos elementos que estarán en la solución final, de este modo ejecutamos un filtrado de 2 a 1, logrando una optimización en el coste del bucle. Una vez realizado el filtrado por filas aplica la misma formula sobre las filas del array intermedio obtenido anteriormente.

Con los valores calculados se realiza un reescalado de los valores de los píxeles de modo que se ajusten a 256 niveles de intensidad, que es el rango de valores valido teniendo una ratio de 1byte*píxel.

Para finalizar se realiza una conversión de array de double a array de byte eliminando los píxeles de los bordes.

Los algoritmos de descompresión siguen el mismo patrón de implementación excepto que se necesita realizar un sobredimensionado del array desde el tamaño de la imagen comprimida al tamaño calculado para los arrays intermedios, un filtro de 1 a 2.

5. Resultados obtenidos

Para la validación de los resultados y verificar la capacidad de transmisión de los diferentes sistemas y procesos hemos contado con la inestimable ayuda del equipo de expertos en percepción visual humana y artificial del Instituto de Tecnología del Conocimiento, gracias a la cual hemos podido corroborar nuestras estimaciones.

Para la realización de las pruebas se contó con la ayuda de la Facultad de Veterinaria y con la red de laboratorios de la Facultad de informática.

La razón de la elección de la facultad de veterinaria fue por motivos del medio de conexión. Esta conexión cumple los requisitos de la problemática expuesta en el proyecto al estar conectada al servidor central de la Universidad Complutense. Este tipo de red era el especificado ya que se requería un modelo de red WAN, i.e. una red privada que tenga enrutamiento externo sin ser un canal dedicado.

Gracias a las características de esta conexión, se pueden observar realmente situaciones de conflicto en la llegada de paquetes de una manera muy similar a lo que sería en la transmisión especificada en la problemática del proyecto.

Tanto la red de veterinaria como la de informática son redes de alta velocidad (superiores a 10 MB/s). Esto nos garantiza una alta velocidad de transmisión.

La primera prueba consistió en el envío de un archivo de texto de 13,4 MB. La aplicación esta diseñada para el envío de imágenes estereoscópicas, pero en pruebas sin filtrado de wavelets es posible probar el envío con cualquier tipo de archivo.

La elección de archivos de texto, se realizó para poder detectar cada una de las perdidas producidas y descubrir en que momento y situación se producen. La detección de errores es mas especifica en el envío de archivos de texto ya que es mas sencillo de detectar el conjunto de información que se ha perdido y a partir de ahí sacar las conclusiones pertinentes.

Aparte se realizó otra prueba con imágenes en la cual ya si se testó el funcionamiento de transformada wavelets. Estas imágenes tenían un tamaño de 600x800 píxeles y una resolución de 256 niveles de intensidad, un byte por pixel.

A continuación mostramos la tabla con las tomas de tiempo bajo las condiciones anteriormente descritas.

	TCP	UDP	TCP con Filtro	UDP con Filtro
Arch. Texto	3 112 ms	2 804 ms	-	-
Imagen	217 ms	193 ms	127 ms	112 ms

Tabla 1: Medición de tiempos en la primera prueba (red WAN)

En la siguiente tabla mostramos el factor de pérdida de datos.

	TCP	UDP	TCP con Filtro	UDP con Filtro
Arch. Texto	0.00	0.32	0.00	0.36
Imagen	0.00	0.42	0.00	0.44

Tabla 2: Tasas de Fallo en la primera prueba.

En la segunda prueba probamos el entorno optimo de transmisión consistente en el envío y recepción de datos en un mismo equipo.

Esta situación es la de menos retardo en el envío, esto lleva consigo un problema, ya que para el protocolo UDP, es necesario introducir un retardo entre el envío de paquetes. Este retardo permite que si un paquete se envía y no existe un retardo, si se envía otro paquete antes de que el primero haya sido recibido, el primero se pierde.

Por esta causa el protocolo UDP no resulta eficaz para transmisiones sin redes intermedias.

Esta prueba se realizo en un equipo AMD Athlon +3400 a 2.41 GHz , 1 GB de mem RAM y bajo entorno Windows XP.

A continuación mostramos la tabla con las tomas de tiempo bajo las condiciones anteriormente descritas.

	TCP	UDP	TCP con Filtro	UDP con Filtro
Arch. Texto	2 236 ms	1 710 ms	-	-
Imagen	102 ms	93 ms	117 ms	116 ms

Tabla 3: Medición de tiempos en la segunda prueba

En la siguiente tabla mostramos el factor de perdida de datos.

	TCP	UDP	TCP con Filtro	UDP con Filtro
Arch. Texto	0.00	0.82	0.00	0.88
Imagen	0.00	0.90	0.00	0.93

Tabla 4: Tasas de Fallo en la segunda prueba.

La tercera prueba se realizó sobre dos conexiones ADSL (512 Kb/s) entre servidores de wanadoo y telefónica.

En teoría esta sería la conexión mas lenta de las tres ya que la velocidad del medio es inferir a ambas y la complejidad del camino es semejante o superior a la situación de la primera prueba.

A continuación mostramos la tabla con las tomas de tiempo bajo las condiciones anteriormente descritas.

	TCP	UDP	TCP con Filtro	UDP con Filtro
Arch. Texto	272 500 ms	230 150 ms	-	-
Imagen	9 512 ms	7 800 ms	3 215 ms	2 700 ms

Tabla 5: Medición de tiempos en la tercera prueba

En la siguiente tabla mostramos el factor de perdida de datos.

	TCP	UDP	TCP con Filtro	UDP con Filtro
Arch. Texto	0.00	0.001	0.00	0.005
Imagen	0.00	0.005	0.00	0.005

Tabla 6: Tasas de Fallo en la tercera prueba.

6. Conclusiones

Después de analizar los resultados anteriores hemos llegado a las siguientes conclusiones:

- A medida que disminuye la velocidad de la red se produce una merma en los tiempos de transmisión: Según hemos observado en los resultados de las pruebas realizadas la velocidad de transmisión aumenta proporcionalmente con la velocidad de la red, por ejemplo en la segunda prueba no existe retardo inducido por la red, por lo que los resultados obtenidos en esta prueba son los más veloces. Por ese mismo motivo en la ejecución sobre la ADSL de 512 Kb/s, se produce un resultado mayor al margen permisible para catalogar la transmisión como tiempo real.

- A medida que aumenta la velocidad de la red, el porcentaje de perdida utilizando el protocolo UDP aumenta aritméticamente. Como se puede observar en los resultados de las pruebas, para tiempos de transmisión aceptables como tiempo real la tasa de perdidas de la transmisión mediante UDP adopta valores que hacen imposible el reconocimiento de las imágenes transmitidas.

  En el uso del protocolo UDP se desechan aquellos paquetes que no son recibidos completamente. Dado que no existe reenvío de paquetes perdidos, estos paquetes incompletos se pierden definitivamente. Cuando la velocidad de la red es comparable a la velocidad de la memoria principal, el tiempo transmisión de paquetes consecutivos es menor que el tiempo de almacenamiento en memoria, por lo que existe una alta probabilidad de colisión o perdidas de paquetes.

- El uso de la transformada wavelets disminuye el tiempo de transmisión a cerca de una cuarta parte del tiempo necesario para la transmisión de la misma imagen sin el uso de dichos filtros de compresión. Esto es debido a la disminución del tamaño de los datos a enviar a la cuarta parte del tamaño de los datos originales. A medida que aumenta la velocidad de la red la diferencia de tiempos entre el uso del envío normal y el envío con filtro se va reduciendo debido al tiempo invertido en la compresión/ descompresión de la imagen.
  Debido a las características de la compresión /descompresión por el filtrado paso bajo – paso bajo, se produce una perdida de nitidez en la imagen recibida, aunque esta perdida resulta difícilmente distinguible a la capacidad del ojo humano.
  Otro problema de la transmisión mediante compresión / descompresión con filtros mediante la transformada wavelets, es que al reducir el tamaño de la imagen a un cuarto para ser enviada, la perdida de un paquete ocasiona una ausencia de información en una región de tamaño cuádruple al tamaño de la región en una perdida equivalente en una transmisión sin filtrado.

- Debido a las especificaciones de la problemática inicial, en las que se definían unos parámetros de perdida y de tiempo de transmisión razonables para una perfecta interacción con el medio remoto, hemos llegado a la conclusión que el protocolo UDP da resultados muy favorables respecto a los tiempos de

transmisión, pero las perdidas ocasionadas mediante dicho protocolo hacen que el desarrollo no pueda continuar por ese camino.

Por este motivo hemos establecido que el mejor protocolo de transmisión de datos para las especificaciones de la problemática inicial es el protocolo TCP. Para equilibrar los retardos de tiempo inferidos por este protocolo, recomendamos utilizar la transformada de wavelets para disminuir el tamaño del envío, siendo las perdidas ocasionadas por la falta de precisión en la reconstrucción de la imagen por el filtro fácilmente asumibles.

7. Futuro

La aplicación desarrollada queda abierta ante la posibilidad de continuación del proyecto en cursos sucesivos.

Hemos considerado este aspecto como de vital importancia, dado que el hecho de desarrollar un proyecto bajo la asignatura de Sistemas Informáticos, supone que el equipo de desarrollo ha de estar preparado para iniciar su andadura profesional en el mercado laboral. Así, en cualquier empresa, los proyectos son propiedad de ésta independientemente del equipo que los desarrolle. Por este motivo, nuestra aplicación queda abierta para que cualquier otro equipo pueda continuarla con el menor esfuerzo posible.

Sobre la base de las conclusiones obtenidas, se deduce que será el protocolo TCP el candidato a ser utilizado, ello conlleva necesariamente el diseño e implementación de alguna técnica de compresión que minimice los efectos de seguridad requeridos por este protocolo para lograr la transmisión en tiempo real. En este sentido se debe continuar con la búsqueda de reducción de datos a transmitir sin pérdida apreciable de información, creemos que la apuesta más acertada sería la utilización de wavelets en zonas selectivas. A esta propuesta llegamos tras el contraste de opiniones entre nosotros y el equipo de psicólogos del Instituto de Tecnología del Conocimiento.

Se trataría de transmitir con seguridad las regiones foveales (zonas centrales) de las imágenes que son las que resultan de interés al usuario, dejando para el protocolo UDP las regiones periféricas, que conllevan menos información 3D desde el punto de vista del usuario humano.

Para poder implementar lo anterior, nuestro diseño está abierto a estas nuevas funcionalidades.

Asociar los dos protocolos a una interfaz, de manera que el sistema siga la misma estructura en la transmisión desarrollada por nosotros de las balizas de control, pero en la transmisión de datos se pueda elegir entre hacer el sendFile() y el receiveFile() mediante un objeto tcp o udp.

Habría que crearse una nueva clase o conjunto de clases encargado de gestionar la división y la reconstrucción de la imagen en dos buffer diferenciados. Uno para ser enviado por tcp y otro para ser enviado por udp. Esta clasificación podría darse por objetos similares a los de nuestra clase paquete con la única diferencia de los atributos de clase. En este nuevo diseño se podría caracterizar cada paquete por un índice o unas coordenadas en la imagen, que hagan referencia a su posicion, para luego poder reconstruirlo sin ningún problema.

Uno de los parámetros que habría que investigar a fondo es el del porcentaje de datos enviados por tcp y por udp, de esta manera se podría llegar a una solución optima para el envío de las imágenes. Una buena manera de diseño de este sistema es permitir variar el tamaño de la porción de imagen mandada por tcp y por udp.

Aparte del tratamiento de las imágenes individualmente, seria necesario empezar a fijar objetivos en lo que a pares de imágenes se refiere. Con esto quiero decir que habría que empezar a definir el formato del buffer que representa a las imágenes asociando a cada píxel del par una posición específica.

Una técnica de diseño para solucionar esto, sería encapsular en cada objeto de la clase paquete un buffer para la imagen izquierda y otro para la imagen derecha.

Aparte de esto también sería conveniente tratar las imágenes por recuadros en vez de por líneas, de esta forma si existiera alguna perdida seria de un conjunto de pixeles vecinos y no de un conjunto de líneas.

También este proyecto se podría ampliar de forma que no se traten las imágenes por pares en una misma transmisión, sino que exista un actor cliente por cámara, creando dos canales de transmisión independientes uno para cada ojo. Esta ampliación tiene su base diseñada en los resultados de este proyecto, lo único que se debería de añadir es un modulo de control entre las transmisiones de ambas cámaras de manera que los resultados obtenidos por ambas sean coherentes.

8. Bibliografía

[1] School of Plant Biology (Botany Building MO90), Faculty of Natural & Agricultural Sciences, The University of Western Australia, 35 Stirling Highway, Crawley, WA 6009, Australia.

[2] Faculty of Engineering, RMIT University, GPO Box 2476V, Melbourne 3001, Australia.

[3] Department of Geomatic Engineering, University College London, Gower Street, London WC1E 6BT, United Kingdom. "The implementation and validation of a stereo-video system for measuring the length of Southern Blue Fin Tuna during transfers".

[4] G. Pajares y J.M. de la Cruz, Visión por Computador: Imágenes digitales y Aplicaciones, RA-MA, 2001.

[5] http://www.sun.com

[6] http://argouml.tigris.org ArgoUml 0.18.1