UNIVERSIDAD DE MÁLAGA

Arquitectura Global

Proyecto VIGIA

Autores:

Manuela Ruiz Montiel
Francisco Moyano Lara
Javier Ríos Pérez
Rafael Martínez González
Israel Rodríguez Martín
Miguel Ángel Lorente López

Versión: 3.0

Fecha: 17/12/2008

Tabla de contenido

Arquitectura Global

La arquitectura global de la aplicación se ajusta a un modelo cliente-servidor. Individualmente, los sistemas cliente y servidor pueden describirse de acuerdo a un modelo de capas.

Cliente

La aplicación cliente se encarga, a grosso modo, de detectar la posición del usuario con respecto a la pantalla y traducir esta información a peticiones que se envían al servidor. Las peticiones son órdenes para la cámara remota. Las diferentes capas de la aplicación, de menor a mayor nivel, se describen a continuación:

1. **Bluetooth stack:** software independiente de nuestra aplicación que, necesariamente, se estará ejecutando en el cliente, para que el sistema operativo detecte apropiadamente el dispositivo Wiimote.

2. **API Mando:** esta capa permite acceder a la información que el mando proporciona al PC en forma de eventos, lo cual hace posible la adquisición de las coordenadas del Wiimote con respecto a los LED's infrarrojos que maneja el usuario.

3. **Driver API-Mando:** debido a la gran cantidad de API's disponibles para el Wiimote, es imprescindible disponer de una capa de aislamiento que permita definir una interfaz para nuestra aplicación, de forma que podamos cambiar la API del mando (o, incluso, el propio mando) sin necesidad de modificar la aplicación principal del cliente, simplemente realizando una nueva implementación de la interfaz definida por este driver.

4. **Aplicación principal:** esta capa se encarga de convertir las coordenadas del mando a órdenes a enviar a la cámara remota.

5. **Comunicaciones:** módulo paralelo a la aplicación cliente que se ocupa de enviar y recibir, a través de una red, toda la información necesaria.

    5.1. El módulo de datos se encarga de recibir las imágenes de la cámara remota.

    5.2. El módulo de control envía las órdenes para la cámara remota y recibe mensajes de estado y/o error desde el servidor.

    5.3. La interfaz proporciona independencia entre la aplicación y los módulos anteriores, de modo que, si es necesario hacer nuevas

implementaciones para nuevas redes, la aplicación no se verá afectada.

6. **GUI:** interfaz de la aplicación para el usuario. Muestra el vídeo remoto por pantalla y permite al usuario acceder, de forma intuitiva, a las diferentes funcionalidades del sistema.

Servidor

Las funciones principales del servidor son las de transmitir las órdenes a la cámara y enviar las imágenes que ésta capte. Las diferentes capas del servidor son, de menor a mayor nivel, las siguientes:

1. **API's Hardware:** software propio del hardware que se utilice para grabar imágenes. Hablamos de hardware, y no de cámara solamente, porque es posible que exista un soporte que proporcione movimiento (de hecho, nosotros trabajaremos con uno).

2. **Módulo de tratamiento de imágenes:** esta capa, situada al mismo nivel que las API's Hardware, proporciona servicios que entran en juego si el hardware no es capaz de proporcionarlos por sí mismo. Por ejemplo, si sólo tenemos una cámara web, sin posibilidad de ningún tipo de movimiento, entonces tendrá que ser simulado mediante software, llevando a cabo transformaciones de perspectiva, ampliaciones, recortes, etc. Por otra parte, aunque el hardware pueda ofrecer movimiento por sí mismo, siempre será necesario un pequeño tratamiento de la imagen para lograr un mejor efecto de "ventana virtual".

3. **Driver cámara/plataforma:** esta capa es específica del hardware que tengamos, y, si éste cambia, habrá que hacer una nueva implementación de la interfaz que se defina para ella. Esto permite que el hardware que utilicemos sea transparente a la aplicación principal del cliente, que, simplemente, enviará órdenes para mover la cámara, sin conocer absolutamente nada del tratamiento de imágenes. Así, la implementación del driver "conocerá" perfectamente el hardware que hay por debajo y descompondrá cada orden de movimiento adecuadamente.

4. **Módulo de comunicaciones:** paralelo al driver. La función de este módulo en el servidor es completamente simétrica a la ya comentada para el cliente, con la salvedad de que, obviamente, envía imágenes correctamente capturadas y tratadas, en lugar de recibirlas.

A continuación, se detalla un diagrama con las diferentes capas:

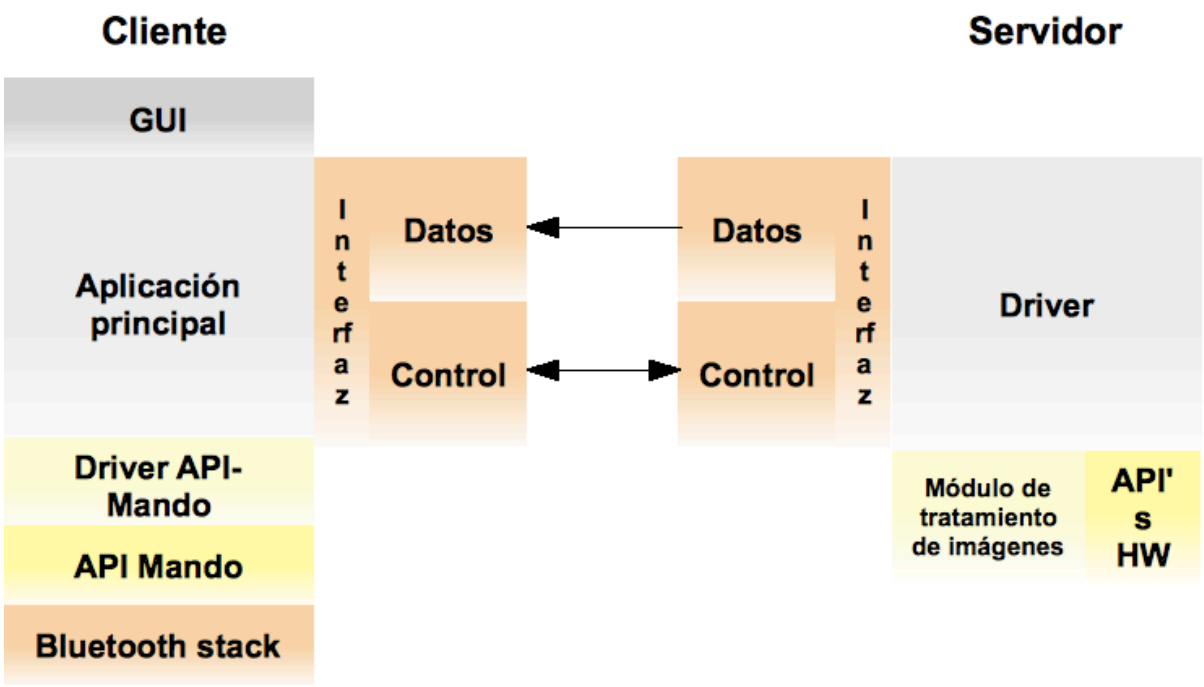


Diagrama: Esquema Cliente-Servidor

Diseño de la GUI

Descripción de la ventana

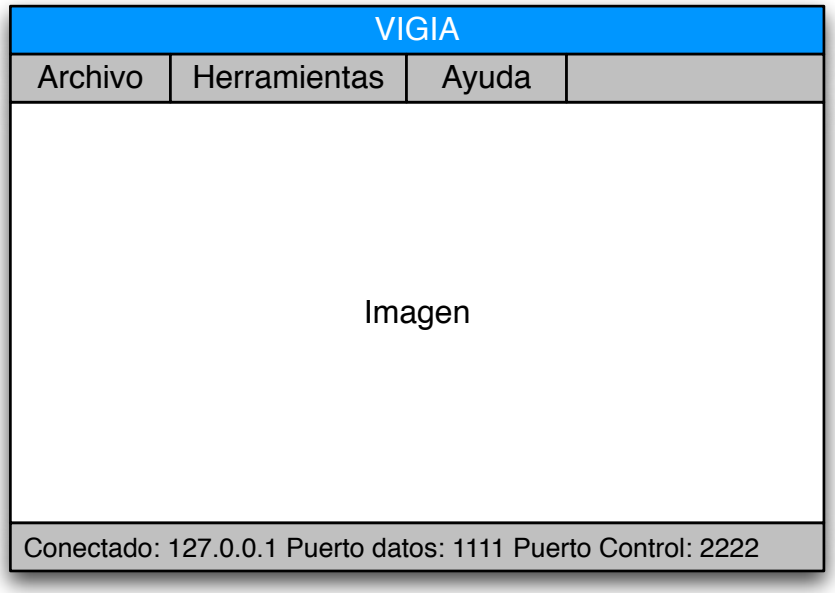


Diagrama: Esquema de la ventana

- **Archivo:**

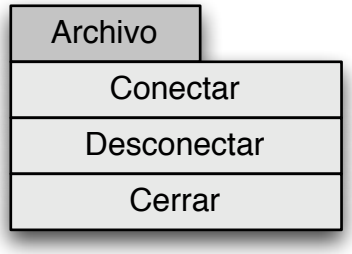


Diagrama: Esquema del menú de Archivo

- **Conectar:** conectar a un servidor. Aparecerá una nueva ventana.

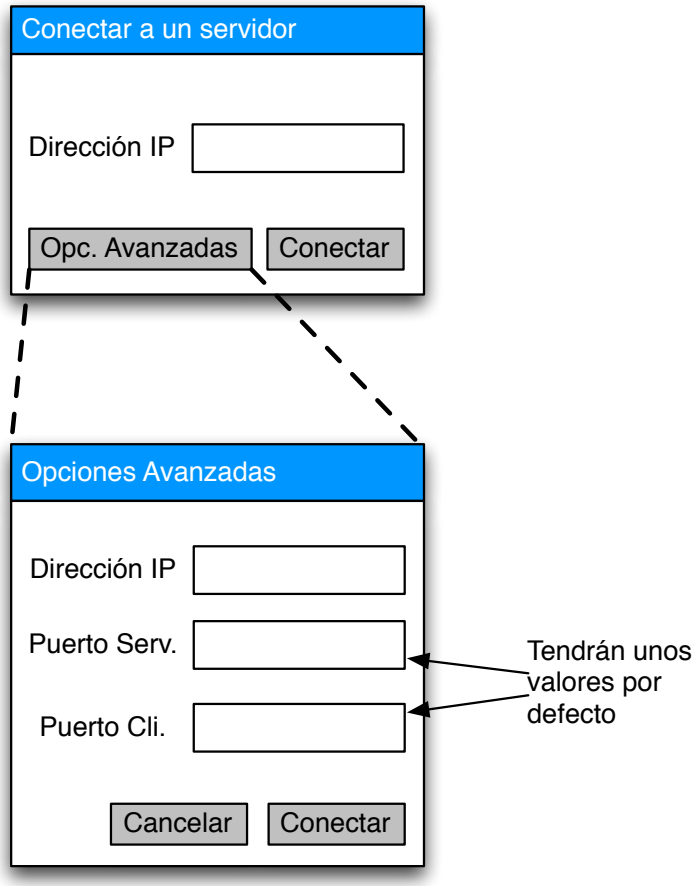


Diagrama: Esquema de las ventanas de conexión

- **Desconectar:** desconectar de un servidor.
- **Cerrar:** cerrar la aplicación.

- **Herramientas:**

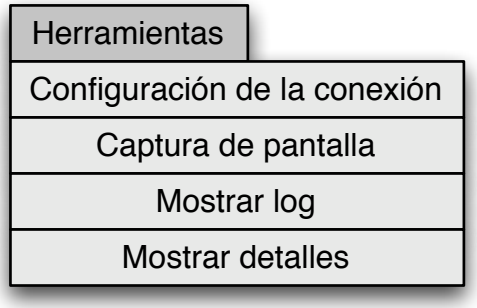


Diagrama: Esquema del menú Herramientas

  - **Configuración de la conexión:** se muestra la misma ventana de "Opciones Avanzadas" que salía en la opción Conectar del menú Archivo.

  - **Captura de pantalla:** mostrará una ventana donde elegir la ruta de destino de la imagen tomada.

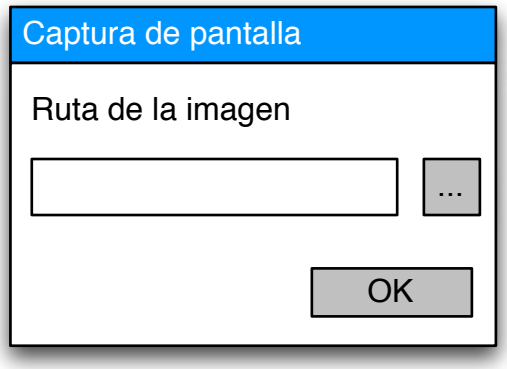


Diagrama: Diseño de la ventana de Captura de Pantalla

  - **Mostrar log:** hace que salga una sección añadida bajo la ventana principal, mostrando los errores.

  - **Mostrar detalles:** ídem que los errores, pero con detalles.

- **Ayuda:**

  - **Acerca de…:** ventana externa con información sobre el nombre y la versión de la aplicación.

Descripción del patrón seguido

El patrón seguido se asemeja a un Modelo-Vista-Controlador, con la salvedad de que el modelo también producirá eventos sobre el controlador, y no será solamente la vista la generadora de eventos.

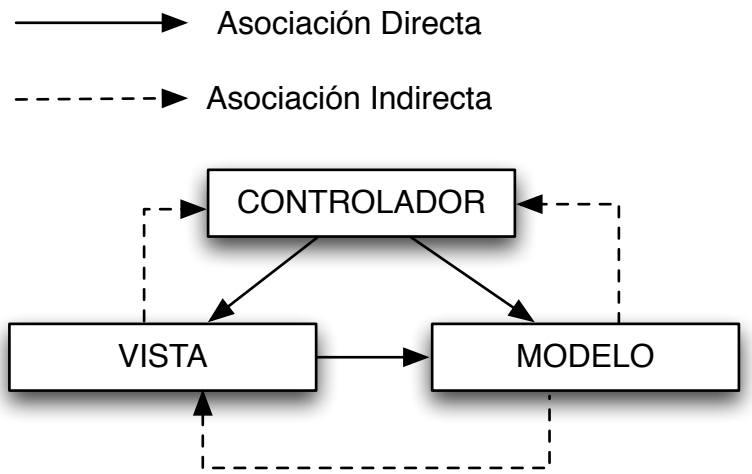


Diagrama: MVC - Modelo Vista Controlador

- **VISTA:** Implementará la parte visible de la aplicación y proveerá métodos para actualizar la imagen que muestra.
- **MODELO:** es el grueso de la aplicación, define toda la lógica de funcionamiento y permitirá conectarse y desconectarse de un servidor, así como cambiar la configuración del sistema.
- **CONTROLADOR:** es el encargado de mediar entre la vista y el modelo, traducirá los eventos provocados por el usuario en operaciones sobre el modelo o cambiará la vista.

Diseño de comunicaciones

Descripción

La comunicación cliente-servidor se realizará por medio de dos canales, uno para los datos y otro para el control.

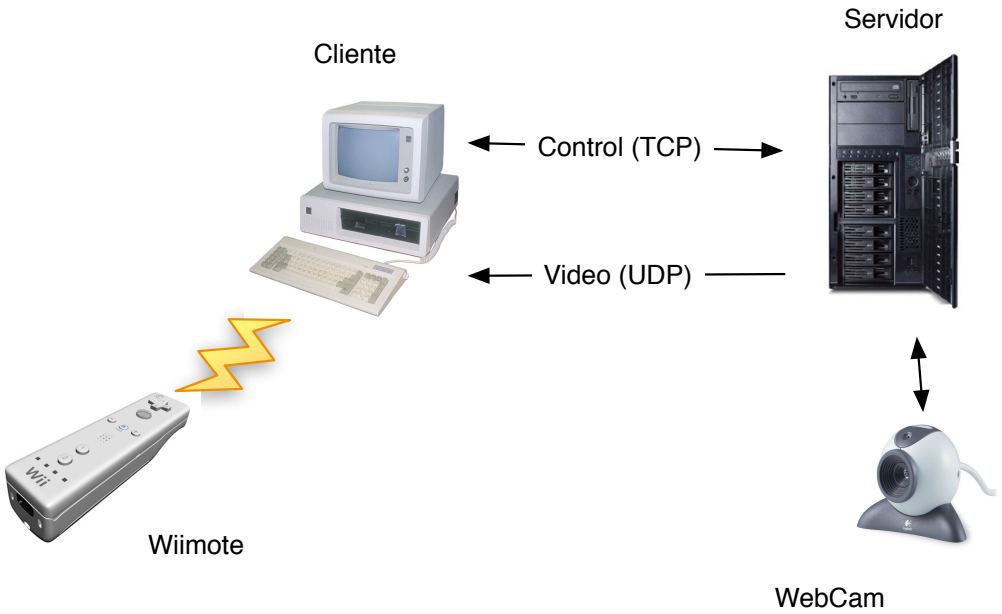


Diagrama: Esquema de las distintas comunicaciones entre los componentes del sistema

El servidor está escuchando por un puerto TCP esperando alguna conexión. Cuando esto suceda el cliente deberá proporcionar al servidor la dirección IP y puerto UDP al que deberá mandar el video.

Cuando el vigilante se mueva, el cliente mandará al servidor, a través del canal de control, la orden necesaria para mover la cámara y simular la visión del vigilante.

El servidor mandará al cliente la imagen capturada por la cámara constantemente a la dirección y puerto indicada por el cliente al establecer la comunicación.

Los comandos podrán ser:

- Inicializar (IP, puerto): indica al servidor donde debe mandar la trama de video.

- Mover_cámara (p, t, z, x, y): hace que el servidor mueva la cámara.
  - p: giro azimutal.
  - t: giro en altura.
  - z: zoom.
  - x: movimiento horizontal.
  - y: movimiento vertical.

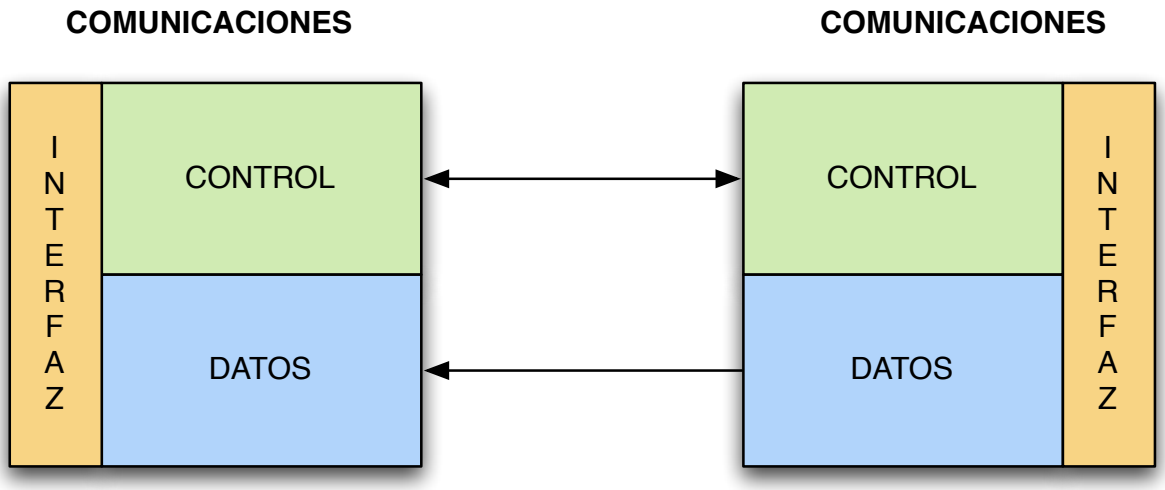


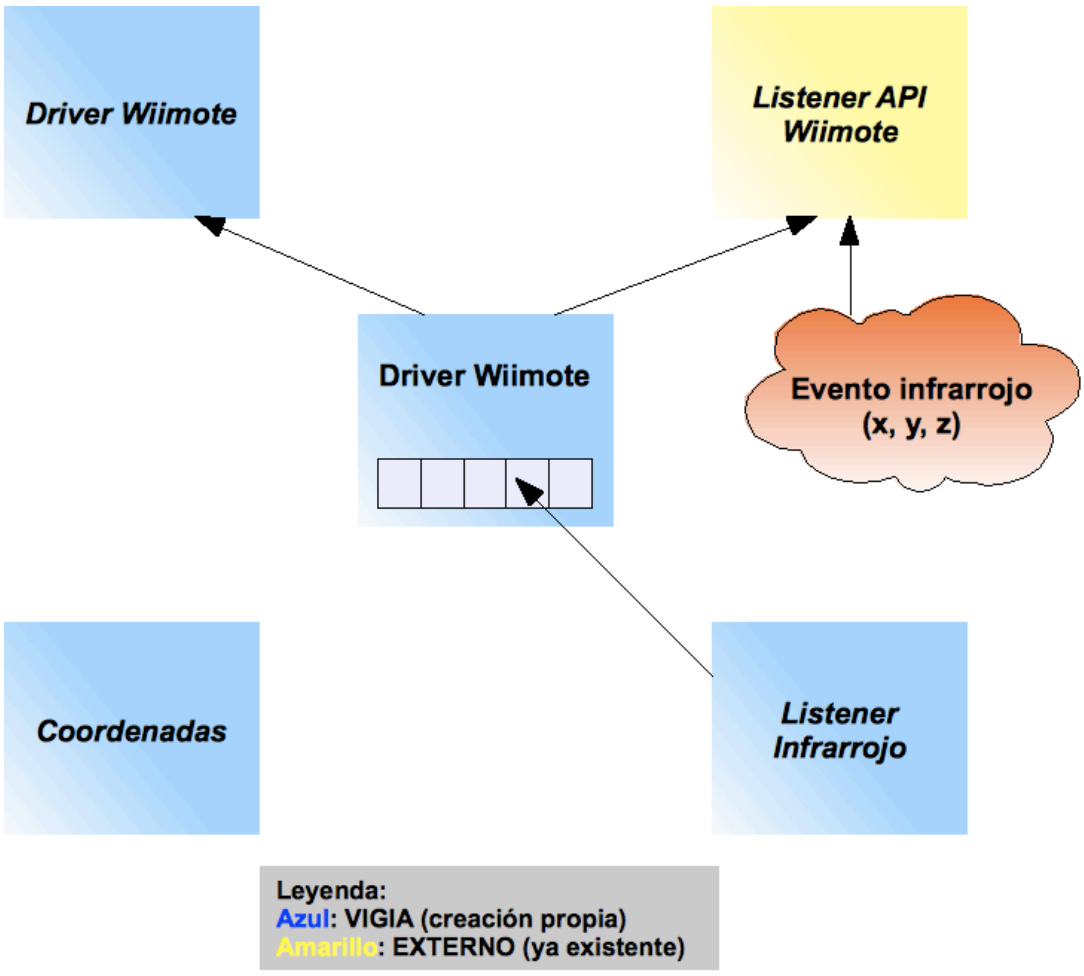
Diagrama: Esquema de las comunicaciones

El módulo de comunicación se dividirá en dos bloques: uno se encargará de los datos y el otro de los comandos de control. Para facilitar la integración se proveerá una única interfaz que hará transparente el uso de los distintos canales de comunicación.

Diseño del driver de la API del mando

Debido a la gran cantidad de API's disponibles para el Wiimote, es necesaria una capa de aislamiento que permita definir una interfaz para nuestra aplicación, de forma que podamos cambiar la API del mando (o, incluso, el propio mando) sin necesidad de modificar la aplicación principal del cliente, simplemente realizando una nueva implementación de la interfaz definida por este driver.

Cuando el mando Wiimote detecte un movimiento de los LED's infrarrojos, comunicará un evento a la API en concreto que se esté utilizando. En general, todas las API's proporcionan un listener para estos eventos, así que la implementación de nuestro driver tendrá que obedecer a la interfaz que defina a ese listener concreto de la API. Por supuesto, también tendrá que implementar la interfaz del driver propio de nuestra aplicación, que lo único que exigirá es un método que añada un listener infrarrojo también definido por nosotros.



Luego el driver propio del Wiimote (Driver Wiimote) tiene que implementar dos métodos:

- Añadir listener de evento infrarrojo (de la interfaz Driver Wiimote). Recibirá un listener y lo añadirá a su lista. En concreto, este listener lo que recibirá son coordenadas, como veremos más adelante.

- Método (de la interfaz Listener API Wiimote)que defina la acción a llevar a cabo cuando se recibe un evento infrarrojo. Esta acción consistirá en comunicar a todos los listener infrarrojos almacenados que se ha producido un evento infrarrojo, pasándoles como argumento las coordenadas del mando. Dichas coordenadas se obtendrán a partir del evento infrarrojo, de la forma que defina la API concreta que estemos utilizando.

La interfaz Listener Infrarrojo tendrá que definir un método que implemente las acciones a llevar a cabo en nuestra aplicación cuando se reciban nuevas coordenadas del mando. Podemos hacer la implementación que queramos dependiendo de lo que vayamos a hacer con estas coordenadas: imprimirlas por pantalla, tratarlas para convertirlas en órdenes para la cámara remota, etc.

Módulo tratamiento de imágenes

Definimos un módulo de tratamiento de imágenes, en el cual tenemos tres métodos de clase: crop, scale y perspective.

De los tres, el único que implementamos es el de crop o recorte, ya que será el único que necesitaremos para nuestra aplicación. La funcionalidad de los otros métodos lo proporciona el hardware de la cámara, pero se han definido por completitud o por si en el futuro se quiere implementar dicha funcionalidad por software.

Descripción del modelo

Para nosotros, el modelo está formado por:

- El driver del mando.
- El módulo de comunicaciones.
- El procesador de movimientos.

El controlador tendrá una instancia de cada uno de ellos almacenada. Tanto el driver del mando como el módulo de comunicaciones enviarán eventos al controlador, que serán gestionados adecuadamente.

Diseño del controlador

El controlador integra todas las funcionalidades necesarias para el funcionamiento de la aplicación. Se encargará de controlar todos los eventos posibles en el lado del cliente, de forma que implementará los siguientes listeners:

- El encargado de gestionar los eventos de la vista.
- El que recibe los eventos infrarrojos del mando.
- El que recibe los frames de la cámara remota.

- El que recibe los mensajes de log del servidor.