UNIVERSIDAD DE MÁLAGA

Formación

Proyecto VIGIA

Autores:

Manuela Ruiz Montiel
Javier Ríos Pérez
Jesús Manuel Rodríguez Sánchez
Daniel Héctor Stolfi Rosso

Versión: 2.0

Fecha: 28/11/2008

Tabla de contenido

Recorte de imágenes en Java

Primera forma

Primera forma, usando la clase BufferedImage del paquete java.awt.image.

Esta clase contiene un método llamado getSubimage(...).

Funciona de la siguiente manera (traducido de la documentación oficial de Java):

public BufferedImage getSubimage(int x, int y, int w, int h);

Devuelve una subimagen definida por una específica región rectangular. El BufferedImage devuelto comparte el mismo array de datos de la imagen original.

Parámetros:

x - Coordenada x de la esquina superior izquierda de la imagen.

y - Coordenada y de la esquina superior izquierda de la imagen.

w - El ancho de la región que queremos.

h - El alto de la región que queremos.

Devuelve una BufferedImage que es una subimagen del BufferedImage original.

Lanza una RasterFormatException si el área especificada no está contenida dentro del BufferedImage.

Esto es una traducción libre, no literal de lo que viene en el javadoc.

Segunda forma

Segunda forma, usando la clase Toolkit del paquete java.awt.

```
Image i2 = Toolkit.getDefaultToolkit().createImage(
                        new FilteredImageSource(i1,
                        new CropImageFilter(x,y,w,h)));
```

Este método acepta como argumento ImageProducer, byte[], byte[] con las dimensiones, un String con la ruta de un archivo y una dirección url con una imagen.

Tratamiento de imágenes

Introducción

Librerías/APIs para el tratamiento de imágenes en JAVA:

- Java 2D
  - Paquete java.awt.geom (transformaciones geométricas de objetos en 2D)
    - AffineTransform: Representa una transformación afín de unas coordenadas a otras.
  - Paquete java.awt.image (crear y modificar imágenes, procesamiento de imágenes)
    - AffineTransformOp: Clase que usa una AffineTransform para hacer un mapeo de una imagen.
- Java 3D
  - …
- Java Advanced Imaging (JAI)
  - Paquete javax.media.jai (transformaciones geométricas sobre imágenes)
    - AffineTransform: Representa una transformación afín de unas coordenadas a otras.
    - PerspectiveTransform: Representa una transformación de perspectiva de una imagen (corrige la distorsión introducida por la perspectiva).
    - PlanarImage: Elimina las distorsiones introducidas por diversas causas en una imagen.
- JHLabs Filter Clases
    - PerspectiveFilter: Realiza una transformación de la perspectiva de una imagen.
- Java Image I/O

- Java Media Framework
- Java Binding for OpenGL

Código de ejemplo

So, here is a full program that cause the segfault.
What it does:
- it loads an image stores it into a bufferedImage and then get a subImage of it (in the real app, it depends on the user input)
- it converts it into a PlanarImage to remove perspective effects on the texture
- and then it creates a new BufferedImage from the PlanarImage corrected in the previous step.

It seems that the nullPointerException comes only when using both perspective correction and (bufferedimage.)subImage.

To be complete, the software used:
jdk 1.5.06 and jai 1.1.2 under linux

Here is the code, to use it you need a 1600x1200 image named blank.jpg

```
import java.awt.image.*;
import javax.swing.*;
import javax.media.jai.*;
import java.awt.geom.*;

public class Test {

public static void main(String[] args) {

// some variables coming from user input in the original code
Point2D.Float v1=new Point2D.Float(745.0981f,162.0915f),
v2=new Point2D.Float(1283.6602f,394.77124f),
v3=new Point2D.Float(1299.3464f,486.2745f),
v4=new Point2D.Float(755.55554f,266.66666f);

float xMin=(float)745.0981,
yMin=(float)162.0915,
xMax=(float)1299.3464,
yMax=(float)486.2745;


// we load a file and put it into a temporaryImage
ImageIcon source=new ImageIcon("blank.jpg"); // blank.jpg is a 1600x1200 image

BufferedImage temporaryImage = new BufferedImage(source.getIconWidth(),
```

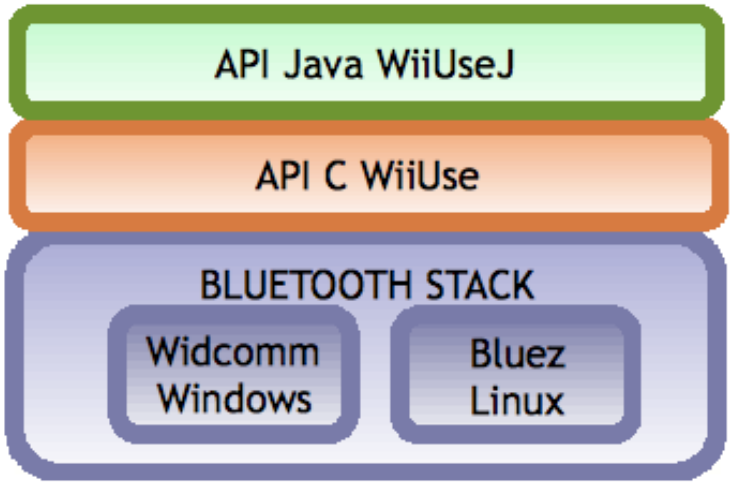
```
source.getIconHeight(), BufferedImage.TYPE_INT_RGB);

temporaryImage.getGraphics().drawImage(source.getImage(),0,0,null);
System.out.println("temporaryImage = "+temporaryImage);

BufferedImage temporaryImage2 = temporaryImage.getSubimage((int) xMin,(int) yMin,(int) (xMax - xMin + 1),(int) (yMax - yMin + 1));

System.out.println("temporaryImage = "+temporaryImage2);

// PlanarImage planarSource=PlanarImage.wrapRenderedImage(temporaryImage2);
PlanarImage planarSource=PlanarImage.wrapRenderedImage(temporaryImage2);

// Projective transformation of the texture

PerspectiveTransform pt = PerspectiveTransform.getQuadToQuad(0,0,xMax - xMin, 0,xMax - xMin,yMax - yMin,0,yMax - yMin,v1.x - xMin,v1.y - yMin,v2.x - xMin,v2.y - yMin,v3.x - xMin,v3.y - yMin,v4.x - xMin,v4.y - yMin);

ParameterBlockJAI pb = new ParameterBlockJAI("Warp");

pb.addSource(planarSource);
pb.setParameter("warp", new WarpPerspective(pt));
pb.setParameter("interpolation", new InterpolationBilinear());

RenderedOp rop=new RenderedOp("Warp",pb,null);
PlanarImage planarImage=rop.createInstance();
System.out.println("planarimage: "+planarImage);

BufferedImage tmpBuffImg=null;

try {
tmpBuffImg=planarImage.getAsBufferedImage();
} catch (Exception e)
{
System.out.println("Caught exception : "+e);
}
System.out.println("tmpBuffImg = "+tmpBuffImg);
}
}


// End of Code
```

API WiiUseJ

La API WiiUseJ

WiiUseJ es una API Java para el acceso al periférico Wiimote de la consola Wii de Nintendo. Se basa en una API C llamada WiiUse que accede directamente al bluetooth stack de nuestro sistema (como BlueSoleil, Widcomm o Bluez). La diferencia de WiiUsej con otras APIs Java reside en que no se basa en el paquete javax.bluetooth (implementación del estándar JSR-82), con lo cual resulta ser un mecanismo más eficiente para acceder al mando al estar implementada sobre un lenguaje de bajo nivel como es C.

La librería WiiUse en C está disponible para plataformas Windows y Linux (formatos .dll y .so).



Para que WiiUseJ acceda a la API C WiiUse, tenemos que copiar los archivos de esta última librería (.dll si estamos en Windows o .so si estamos en Linux) en la carpeta principal de nuestro proyecto Java.

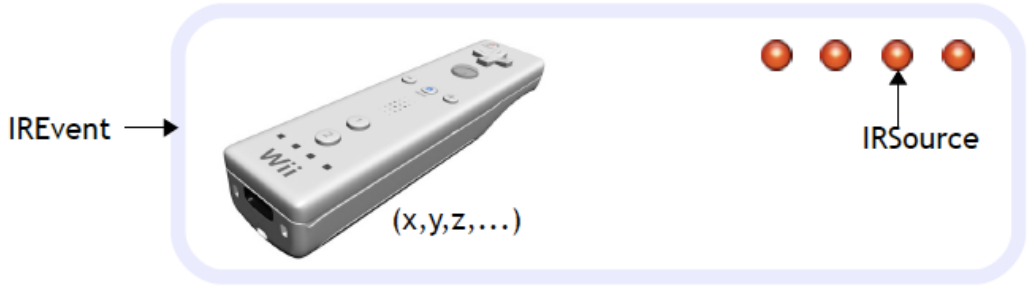
Funcionamiento de WiiUseJ

El aspecto que nos interesa del mando es la recepción de fuentes de luz infrarroja. El funcionamiento de WiiUseJ en este aspecto se detalla a continuación.

A un nivel bajo, podemos considerar las clases IRSource e IREvent (fuente infrarroja y evento infrarrojo, respectivamente).

- IRSource representa a una luz infrarroja detectada por el mando, vista como un punto bidimensional con un tamaño asociado.

- IREvent representa un evento infrarrojo formado por todos los IRSources (puntos de luz infrarroja) que estén dentro del radio de acción del Wiimote. Estos puntos se almacenan en un array. En los objetos de la clase IREvent se almacena también la posición absoluta del Wiimote (coordenadas x, y, z) y otra información relevante, como por ejemplo, si el mando se encuentra por encima o por debajo de los puntos de luz.



La librería WiiUse C recoge los eventos IREvent a bajo nivel y los comunica a la API Java. En esta última, la librería WiiUse se representa mediante la clase WiiUseApi, un singleton usado para llamar directamente a los archivos .dll o .so desde Java.

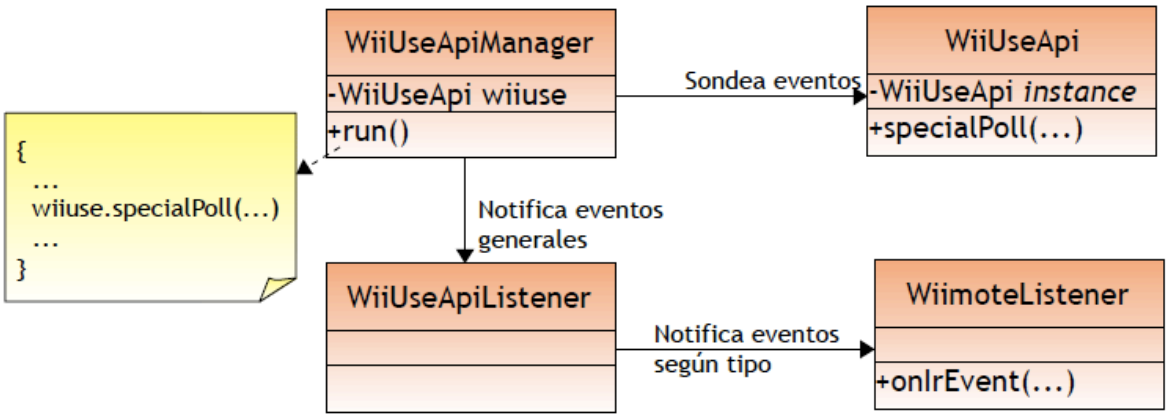
La clase que maneja a la librería WiiUse se llama WiiUseApiManager, que hereda de la clase Thread y se encarga de lanzar el "polling" de eventos, en su método run().

En la API WiiUseJ existen dos tipos de "listeners", representados por las dos interfaces siguientes:

- WiiUseApiListener: se encarga de escuchar los eventos que recoge WiiUseApiManager. La clase que implemente esta interfaz se encargará de llamar a los métodos adecuados dependiendo del tipo de evento escuchado. Si detecta que el evento escuchado es infrarrojo, llamará al método onIrEvent(IREvent arg) definido en la siguiente interfaz.

- WiimoteListener: la clase principal del proyecto tiene que implementar esta interfaz, que es la que define las acciones a llevar a cabo en la aplicación concreta cuando se producen los eventos recogidos por WiiUseApiListener. Define el método onIrEvent(IREvent arg).

Todo esto queda reflejado en el siguiente diagrama:



El método onIrEvent(IREvent arg0) accede a la información del evento infrarrojo, pudiendo tratar directamente las coordenadas del mando con respecto a las luces infrarrojas para usarlas como queramos.

Guía de usuario

¿Cómo realizar finalmente nuestra aplicación? Los pasos a seguir son los siguientes:

1. Copiar los archivos de la librería WiiUse (.dll si estamos en Windows o .so si estamos en Linux) en la carpeta principal de nuestro proyecto Java.

2. Crear una clase (MyListener) que implemente el interfaz WiimoteListener (en concreto, lo que nos interesa es el método onIrEvent).

3. En la clase main: Wiimote[] wiimotes = WiiUseApiManager.getWiimotes(X, true);

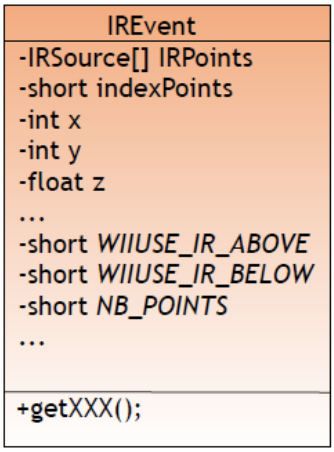
    3.1. Esto detecta los mandos que haya. Se encarga de lanzar la hebra de WiiUseApiManager. Wiimote es una clase que implementa el interfaz WiiUseApiListener, así que lo que devuelve es un array de listeners del WiiUseApi.

    3.2. El parámetro booleano "true" hace que se produzca un sonido la primera vez que se detectan los Wiimotes.

4. En la clase main: wiimote[0].addWiiMoteEventListeners(new MyListener());

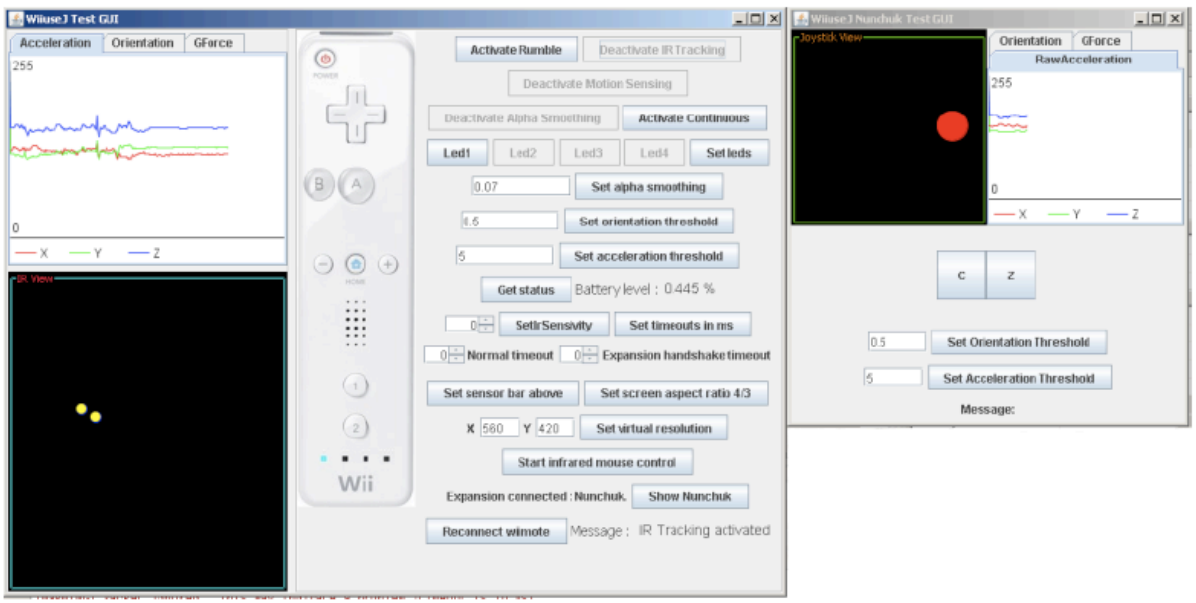
    4.1. Registra el primer Wiimote con nuestro listener implementado en el primer paso. Cuando el mando lance un evento infrarrojo, sucederá lo que hayamos implementado en el método onIrEvent(IREvent arg0) de esta clase.

Finalmente, se detallan algunos atributos de la clase IREvent para ilustrar la información más relevante que se puede obtener de ella.



Pruebas

Para probar la API, ésta incluye clases que implementan una interfaz gráfica. Esta GUI se lanza desde la clase Main que viene con la API, o ejecutando wiiusej.jar.



Referencias

Todas las fuentes de las APIs, wiki, información general, etc. se puede encontrar en **http://code.google.com/p/wiiusej/**.

Demostración de recorte y zoom

JPImg.java

```
/**
 *
 */
package isp.zoom;

import java.awt.Graphics;
import java.awt.Image;
import java.text.NumberFormat;

import javax.swing.JPanel;

/**
 * @author Daniel Héctor Stolfi Rosso
 * ISP - VIGIA
 * panel para imagen
 */
public class JPImg extends JPanel {

	private
		Image img;
		String ruta;
		int factor, algoritmo;

	/**
	 * Constructor
	 * @param ruta Ruta de las imágenes
	 */
	public JPImg(String ruta) {
		this.ruta = ruta;
	    factor = 0;
	    algoritmo = 0;
	    aplicaZOOM();
	}

	/**
	 * Pinta imagen
	 */
	public void paint(Graphics g) {
		g.drawImage(img, 0, 0, img.getWidth(this),
img.getHeight(this), this);
	}

	/**
	 * Función que acerca
	 */
	public void zoomIN() {
		if (factor < 10) {
			factor++;
```

```
                aplicaZOOM();
            }
        }

        /**
         * Función que aleja
         */
        public void zoomOUT() {
            if (factor > 0) {
                factor--;
                aplicaZOOM();
            }
        }

        /**
         * Función que cambia el algoritmo
          *  @param  algoritmo  Algoritmo  de  redimensionado  a
utilizar
         */
        public void setAlgoritmo(int algoritmo) {
            this.algoritmo = algoritmo;
            aplicaZOOM();
        }

        /**
         * Función que cambia el factor del ZOOM
         */
        private void aplicaZOOM() {
            // Transforma imagen
            this.factor = factor;
            img = corte(factor);
            img = zoom(factor);
            repaint();
        }

        /**
         * Función zoom
         * @param factor Factor ZOOM
         */
        private Image zoom(int factor) {
            int alg;
            switch (algoritmo) {
            case 0: alg = Image.SCALE_AREA_AVERAGING; break;
            case 1: alg = Image.SCALE_DEFAULT; break;
            case 2: alg = Image.SCALE_FAST; break;
            case 3: alg = Image.SCALE_REPLICATE; break;
            case 4: alg = Image.SCALE_SMOOTH; break;
            default: alg = Image.SCALE_DEFAULT;
            }
            return  img.getScaledInstance(800,  600,
alg);
        }

        /**
         * Simulación Función Corte
```

```
             * @param factor Índice de imagen a utilizar para simular
corte
             */
            private Image corte(int factor) {
                 String fichero = ruta
+String.format("%02d.jpg",factor);
                 System.out.println(fichero);
                 return getToolkit().getImage(fichero);
            }
    }
```

Zoom.java

```
package isp.zoom;

import java.awt.GridLayout;
import java.awt.Insets;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * @author Daniel Héctor Stolfi Rosso
 * ISP - VIGIA
 * Zoom imagen
 */

public class Zoom extends JFrame {

      JPImg jPImg;
      JPanel jPBotonera;
      JButton jBZin, jBZout;
      JComboBox jCBAlg;

      /**
       * Constructor
       * @param fichero Ruta y patrón de nombre de los ficheros de imagen
a visualizar
       */
      public Zoom(String fichero) {
            // Inicializa
            setLayout(null);

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
         setTitle("ZOOM");
         setSize(815, 715);
         setResizable(false);

         Insets interior = getInsets();

         // Imagen
            jPImg = new JPImg(fichero);
            add(jPImg);
            jPImg.setBounds(interior.left,
```

```
                                    interior.top,
                                    800, 600);
            // Botones
            jBZin = new JButton("ZOOM IN");
          jBZin.addActionListener(new java.awt.event.ActionListener() {
              public void actionPerformed(java.awt.event.ActionEvent evt)
{
                  jPImg.zoomIN();
              }
          });
            jBZout = new JButton("ZOOM OUT");
          jBZout.addActionListener(new java.awt.event.ActionListener() {
              public void actionPerformed(java.awt.event.ActionEvent evt)
{
                  jPImg.zoomOUT();
              }
          });
          // Combo
          jCBAlg = new JComboBox();
          jCBAlg.setModel(new javax.swing.DefaultComboBoxModel(new
String[]
{ "SCALE_AREA_AVERAGING","SCALE_DEFAULT","SCALE_FAST","SCALE_REPLICATE",
"SCALE_SMOOTH" }));
          jCBAlg.addActionListener(new java.awt.event.ActionListener() {
              public void actionPerformed(java.awt.event.ActionEvent evt)
{
                  jPImg.setAlgoritmo(jCBAlg.getSelectedIndex());
              }
          });

            // Botonera
            jPBotonera = new JPanel(new GridLayout(1,3));
            jPBotonera.add(jBZin);
            jPBotonera.add(jCBAlg);
            jPBotonera.add(jBZout);
            add(jPBotonera);
            jPBotonera.setBounds(interior.left,
                                    interior.top + jPImg.getHeight() +
2,
                                    800, 80);
      }

      /**
       * @param args Se le pasa la ruta y patrón de nombre de los
ficheros de imagen a visualizar
       */
      public static void main(String[] args) {
            if (args.length < 1) {
                  System.out.println("Uso: Zoom ruta");
            } else {
                  Zoom z = new Zoom(args[0]);
               z.setVisible(true);
            }
      }

}
```

API TrackerPod

Software de control existente para TrackerPod

- PTZdriver: Software de control básico para la base móvil TrackerPod.
- TrackerCam: Software de control avanzado que trabaja conjuntamente con la base móvil TrackerPod y con la cámara web usada.

Comparativa entre el software existente para el TrackerPod

	 TrackerCam	 PTZdriver
Works with webcams	✓	✗
Works with camcorders	✓	✓
Pant/tilt control	✓	✓
Runs w/o Tracker/PowerPod	✓	✗
Digital zoom control	✓	✓
Firewire optical zoom control	✓	✓
LANC optical zoom control	✓	✓
ClearPTZ optical zoom control	✓	✓
Serves video to Web	✓	✗
Pan/tilt over Web	✓	✓
Person tracking	✓	✗
Motion tracking	✓	✗
Use video in up to 7 apps	✗	✗
Capture multiple streams	✗	✗

Still image capture	✓	✓
Movie capture	✓	✗
Scheduled capture	✓	✗
Motion triggered capture	✓	✗
Motion triggered alarms	✗	✗
FTP upload captured images	✓	✗
Access via cell phone	✓	✗
Built-in chat	✓	✗
Windows 98SE, ME, 2000, XP	✓	✓

¿Qué se puede hacer con la API de TrackerPod?

Tanto la librería como el objeto COM sólo incluyen funcionalidad para controlar los movimientos de "pan" y "tilt" sobre TrackerPod, no se incluye código para manipular la imágenes/video de la cámara.

Métodos de comunicación con la base PTZ (TrackerPod)

Existen tres formas de comunicación con la base motorizada PTZ:

- directamente a través de peticiones/respuestas al servidor web integrado en el software de comunicación con la base (mediante JavaScript y rutinas PHP).
- usando la librería propia existe para el control de la base en nuestros programas desarrollados en C++ o C#.
- usando el componente COM creado para el control de la base en nuestros programas Visual Basic o ActiveX.

Descripción de la comunicación mediante peticiones/ respuestas HTTP

Peticiones/respuestas POST para las imágenes

```
<form name="Stills" action="RemoteProcessN.trackercamimage"
method="post" target="Invisible">
<input type="hidden" name="UserID">
<input type="hidden" name="Busy">
<input type="hidden" name="RefreshFlag">
<input type="hidden" name="Frame">
<input type="hidden" name="ConsoleType" value="Full">
<input type="hidden" name="ConsoleSize" value="Standard">
<input type="hidden" name="RefreshRate">
<input type="hidden" name="WhereIsTheFunctions" value="top.Main">
</form>
```

Peticiones/respuestas POST para el control de la base

```
<form name="Controls" action="RemoteProcessN.trackercamcontrol"
method="post" target="Invisible">
<input type="hidden" name="XPos">
<input type="hidden" name="YPos">
<input type="hidden" name="UserID">
<input type="hidden" name="RequestType">
<input type="hidden" name="StepSize">
<input type="hidden" name="WhereIsTheFunctions" value="top.Main">
</form>
```

Ejemplo de petición POST:

```
-----------------------------------------------------------------------
POST /RemoteProcessN.trackercamcontrol HTTP/1.0
User-Agent: TrackerCamHttp/1.0
Accept: www/source; text/html; image/gif; */*
Content-type:application/x-www-form-urlencoded
Content-length: <longitud_datos>
variableX=<valorX>&variableY=<valorY>&…
-----------------------------------------------------------------------
```

Donde las posibles variables a usar junto con sus valores se muestran a continuación:

```
XPos={H o un número entero}
YPos={H o un número entero}
ZPos=[10..500] -> Normal 100
UserID=<nombre_usuario>
RequestType={move, zoompos}
StepSize=[1..7]
WhereIsTheFunctions={unknown}
PixelMove={Absolute} (creo que no se usa actualmente)
```

Descripción de la librería para proyectos C++\C#

1. *¿Qué incluye?*

La librería incluye 6 ficheros :

1.1. "include\TrackerPod.h" – contiene las declaraciones de las funciones.

1.2. "lib\6.0\TrackerPod_ml.lib" and "lib\6.0\TrackerPod_mt.lib" – librerías simple y multi-hebra para Visual Studio 6.0.

1.3. "lib\7.1\TrackerPod_ml.lib" and "lib\7.1\TrackerPod_mt.lib" – librerías simple y multi-hebra para Visual Studio .NET 2003 (7.1).

1.4. "lib\8.0\TrackerPod_mt.lib" – librería multi-hebra para Visual Studio 2005 (8.0)

También se incluye un proyecto de ejemplo "TrackerPodClient" hecho en C para la plataforma Win32 para mostrar como usar la librería.

2. *¿Cómo usar la librería?*

Antes de poder usar TrackerPod, se debe ejecutar "trackerpod_server.exe". Esta aplicación instalará el administrador de TrackerPod en el directorio Windows. Después se debe incluir el fichero de cabeceras "TrackerPod.h" en el archivo de código y enlazar la librería correspondiente, la simple "TrackerPod_ml.lib" o la multihebra "TrackerPod_mt.lib", en la versión de Visual Studio existente.

Una vez hecho lo anterior, basta con una llamada al método CTrackerPod::CreateTrackerPod() para obtener una instancia CTrackerPod sobre la que poder empezar a trabajar.

Ver el proyecto de ejemplo "TrackerPodClient" para más detalles.

3. *¿Cómo funciona la librería TrackerPod?*

Cuando un programa hace uso de la librería, la librería internamente ejecuta el programa llamado Administrador TrackerPod. Esta aplicación realiza las siguientes funciones:

A). Inicializar el dispositivo TrackerPod

B). Monitorizar los eventos de conexión/desconexión del dispositivo USB

C). Permitir al usuario asignar el dispositivo a sus programas

D). Asignar el dispositivo apropiado en el arranque de los programas

4. *La clase CTrackerPod*

```
typedef void (__stdcall *_dev_change_notify)(int devindex, char
*version, int status, unsigned long user_data);
class CTrackerPod
{
public:
CTrackerPod() { }
virtual ~CTrackerPod() { }
virtual bool initialize(char *client_id,
_dev_change_notify pfn,
unsigned long user_data,
bool b_show_on_event) = 0;
virtual bool begin_enum_device() = 0;
virtual bool enum_reset() = 0;
virtual long enum_next(char *ver) = 0;
virtual bool use_device(long devid) = 0;
virtual bool move_to(float x, float y) = 0;
virtual bool move_by(float x, float y) = 0;
virtual long control(char cmd[1024]) = 0;
virtual bool get_pos(float *x, float *y) = 0;
virtual bool get_info(char info[1024]) = 0;
virtual void show_on_event(bool show) = 0;
```

```
virtual void show_manager(bool show) = 0;
virtual bool is_manager_visible() = 0;
virtual void ReleaseTrackerPod() = 0;
static CTrackerPod* CreateTrackerPod();
};
static CTrackerPod* CreateTrackerPod()
```

Esta función estática retorna un puntero a una instancia CTrackerPod o NULL en caso de fallo. Se debe llamar a ReleaseTrackerPod para liberar los recursos una vez que no se va a usar más la instancia.

```
bool initialize(char *client_id, _dev_change_notify pfn, unsigned long user_data, bool b_show_on_event)
```

Prepara el control del dispositivo TrackerPod. Esta es la primera función en ser llamada después de static CTrackerPod* CreateTrackerPod().

client_id: cadena que contiene la identidad del cliente. El programador debe proporcionar una cadena de identidad de cliente única ya que esta información será usada por el Administrador TrackerPod para asignar el dispositivo a la aplicación cliente.

pfn: función de callback usada para notificar al programa eventos de cambios en el dispositivo. Este valor puede ser NULL.

user_data: datos definidos por el usuario

b_show_on_event: true muestra o false oculta el Administrador TrackerPod

Devuelve true en caso de realizarse satisfactoriamente, false en caso de fallo.

Nota: Si no se llama a la función use_device(), el Administrador TrackerPod asignará el ultimo dispositivo usado por la aplicación del cliente.

```
bool begin_enum_device()
```

Para enumerar los dispositivos disponibles, invocar a esta función. Devuelve true en caso de éxito, false en caso de fallo.

```
long enum_next(char *ver)
```

Devuelve la identidad del dispositivo (long integer) y la versión del mismo (string). Si el valor devuelto es menor que 0, indica que no hay más dispositivos. A continuación un ejemplo,

```
char ver[16];
pTrackerPod->begin_enum_device();
while(true)
{
int devid = pTrackerPod->enum_next(ver);
if(devid < 0)
break;
}
bool enum_reset()
```

Si se quieren enumerar los dispositivos de nuevo, se debe llamar a esta función en lugar de begin_enum_device(). Devuelve true en caso de éxito, false en caso de fallo.

```
bool use_device(long devid)
```

Usar el dispositivo con identificador devid. Si devid = -1, no uar ningún dispositivo. Devuelve true en caso de éxito, false en caso de fallo.

```
bool get_pos(short *x,short *y)
```

Obtiene el valor actual de pan y tilt. Devuelve true en caso de éxito, false en caso de fallo.

```
bool move_to(short x, short y)
```

Ordena al dispositivo que se mueva a la posición absoluta indicada por x e y. Devuelve true en caso de éxito, false en caso de fallo.

```
bool move_by(short x, short y)
```

Ordena al dispositivo que se mueva a la posición relativa indicada por x e y, movimiento respecto a la última posición. Devuelve true en caso de éxito, false en caso de fallo.

```
void ReleaseTrackerPod()
```

Cuando no se necesite más la instancia, llamar a ReleaseTrackerPod para liberar memoria.

```
long control(char *cmd)
```

Ordenar al dispositivo que realice el commando apuntado por cmd. Devuelve un valor distinto de cercen case de éxito, 0 en caso de fallo.

```
bool get_info(char info[1024])
```

Obtiene información sobre el dispositivo. Devuelve un valor distinto de cercen case de éxito, 0 en caso de fallo.

```
void show_on_event(bool show)
```

Muestra, si show es true, TrackerPod (PowerPod) Manager cuando algún evento es notificado.

```
void show_manager(bool show)
```

Muestra, si show es true, TrackerPod (PowerPod) Manager, en otro caso lo oculta.

```
bool is_manager_visible()
```

Si el valor devuelto es true, indica que el administrador se está mostrando, en otro caso está oculto.

Descripción del componente TrackerPod para proyectos C++ \ Visual Basic

1. *¿Cómo se registra el componente TrackerPod?*

Lo único que hay que hacer es ejecutar el archivo compodsrv_setup.exe. Este programa extrae el componente TrackerPod a la carpeta Windows y lo registra. El nombre del componente instalado es trackerpod_com.dll.

2. *¿Cómo usar el componente TrackerPod?*

Se necesita inicializar la librería COM, crear una instancia del componente TrackerPod y a continuación ya podemos usar la interfaz definida para este componente, llamada ITrackerPodC. Hay que tener en cuenta que se debe usar un identificador único por cada instancia del componente.

3. *¿Cómo funciona el componente TrackerPod?*

Cuando un programa hace uso del componente, el componente internamente ejecuta el programa llamado Administrador TrackerPod. Esta aplicación realiza las siguientes funciones:

A). Inicializar el dispositivo TrackerPod

B). Monitorizar los eventos de conexión/desconexión del dispositivo USB

C). Permitir al usuario asignar el dispositivo a sus programas

D). Asignar el dispositivo apropiado en el arranque de los programas

4. *La interfaz ITrackerPodC*

```
interface ITrackerPodC : IUnknown
{
virtual HRESULT __stdcall initialize(BSTR app_name) = 0;
virtual HRESULT __stdcall move_to(short x, short y) = 0;
virtual HRESULT __stdcall move_by(short x, short y) = 0;
virtual HRESULT __stdcall get_pos(short *x, short *y) = 0;
virtual HRESULT __stdcall devcontrol(BSTR cmd, long *result) = 0;
virtual HRESULT __stdcall get_info(BSTR *pVal) = 0;
};
```

A). *HRESULT get_pos(short *x,short *y)*

Obtiene los valores actuales de pan y tilt del dispositivo. En caso de éxito, devuelve S_OK.

B). *HRESULT move_to(short x, short y)*

Ordena al dispositivo que se mueva a la posición absoluta indicada por los valores x e y. En caso de éxito, devuelve S_OK.

C). *HRESULT move_by(short x, short y)*

Ordena al dispositivo que se mueva a la posición relativa indicada por x e y, movimiento respecto a la última posición. En caso de éxito, devuelve S_OK.

D). *HRESULT initialize(BSTR app_name)*

Inicializa la aplicación para controlar dispositivos TrackerPod. Esta debe ser la primera función de la interfaz que se llama en el programa, si queremos que funcione correctamente.

El programador debe proporcionar el identificador único de cliente, app_name. Esta información será usada por el Administrador TrackerPod para asignar el dispositivo a la aplicación cliente. En case de éxito, devuelve S_OK.

E). *HRESULT devcontrol(BSTR cmd, long *result)*

Ordena al dispositivo TrackerPod que ejecute el comando indicado por cmd. En caso de éxito, devuelve S_OK.

F). *HRESULT get_info(BSTR *pVal)*

Obtiene información relativa al dispositivo TrackerPod. En caso de éxito, devuelve S_OK.

NOTA: Podrá obtener más información en el documento anexo "Presentacion_API_TrackerPod.pdf".

Formación JMF (Java Media Framework)

NOTA: La referencia a esta formación se encuentra en el documento adjunto "JMF.pdf".