UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Expandiendo el escritorio con infrarrojos

Apellidos, nombre	Agustí i Melchor, Manuel (magusti@disca.upv.es)
Departamento	Departamento de Informática de Sistemas y Computadores (DISCA)
Centro	Escola Tècnica Superior d'Enginyeria Informàtica Universitat Politècnica de València

1 Resumen de las ideas clave

En este artículo vamos a presentar el uso de luz infrarroja como instrumento para facilitar la interacción del usuario con un computador. El uso de un dispositivo que emita este tipo de luz servirá como extensión del ratón en el escritorio tradicional. El computador hará uso de técnicas de Visión por Computador (en adelante VxC) para detectar la posición de ese dispositivo en la escena. Esta es una temática actual en el campo de la interacción con el computador. De hecho, es de uso habitual en las consolas de videojuegos de última generación con dispositivos como el WiiMote[1] o el Kinect[2], las pizarras interactivas o los escenarios de realidad virtual[3] entre otros.

El interés de la aproximación que aquí se expone reside en:

- Su bajo presupuesto. Se utiliza **un mando infrarrojo cualquiera**, p. ej, un mando a distancia de un equipo doméstico (TV, refrigeración o calefacción, equipo de música, proyector de vídeo, etc.) puede servir. En su defecto, se podrá indicar el uso de un fichero de vídeo para probar y validar la reacción de la aplicación ante una situación particular.
- Su bajo consumo de recursos del computador. El **uso de la luz infrarroja** simplifica el algoritmo de detección de la posición del mando en la escena.

¿Qué se puede hacer con esa información de dónde está el dispositivo? Lo que la imaginación proponga: pasar transparencias, sustituir al ratón, … Le haré alguna sugerencia en cuanto le haya contado qué operaciones necesita realizar para utilizar este tipo de dispositivos.

2 Objetivos

Una vez que el lector haya explorado los contenidos relacionados con la aplicación que aquí se diseña, será capaz de:

- Identificar cómo se procesa una secuencia de vídeo tanto está guardada en un fichero como si proviene de una cámara digital conectada al computador.
- Analizar los parámetros que permiten detectar la presencia de una fuente de luz infrarroja en la escena.
- Identificar la simplificación de la tarea de detección que supone sobre el uso de objetos de color en la imagen de color RGB.
- Identificar las condiciones ambientales en las que es factible el uso de este tipo de luz.

La estrategia para llevar a cabo el tema propuesto consiste en desarrollar la idea inicial en una serie de pasos en secuencia:

---

[1]Brain, Marshall. "How the Wii Works" 05 September 2007. HowStuffWorks.com. <http://electronics.howstuffworks.com/wii.htm> Consultada el 1 de Junio de 2015.

[2] Kinect features <https://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx> Consultada el 1 de Junio de 2015.

[3]J. Chung. Wii Projects. <http://johnnylee.net/> Consultada el 1 de Junio de 2015.

- La aplicación deberá identificar una fuente de vídeo como entrada de datos.
  - Se valdrá de la existencia de una cámara digital (p. ej., una cámara web con conexión USB) para seguir las acciones del usuario delante del computador.
    - También deberá ser posible utilizar un nombre de fichero de vídeo, para utilizar su contenido como fuente de datos.
- La aplicación ha ser capaz de segmentar el objeto del fondo. Esto es, identificar la luz infrarroja como el objeto de interés en la escena. Aquel cuyas propiedades (color, posición, etc.) nos interesa averiguar en nuestra aplicación. El resto de la escena lo denominaremos “fondo”, esto es, no aporta información para la tarea que tiene que hacer la aplicación. El fondo deberá ser ignorado.
- Experimentar en diferentes situaciones de luz ambiente para validar el sistema.

No es objetivo de este artículo describir como se realiza la interacción con un sistema operativo en concreto. En [4] puede encontrar cómo generar los eventos de posición de ratón o de pulsación de los botones del mismo de manera sintética y desde su propia aplicación.

3 Introducción

Este ejemplo nace de la inspiración del trabajo de Millán [1] que utilizaba un emisor de luz láser, véase fig. 1 para ampliar la interacción del usuario, utilizando objetos del mundo real, es decir, que están fuera del “escritorio virtual”. Creando así un “escritorio extendido” en el que el computador recibe eventos de objetos del mundo real y responde a ellos como lo hace al ratón o al teclado.

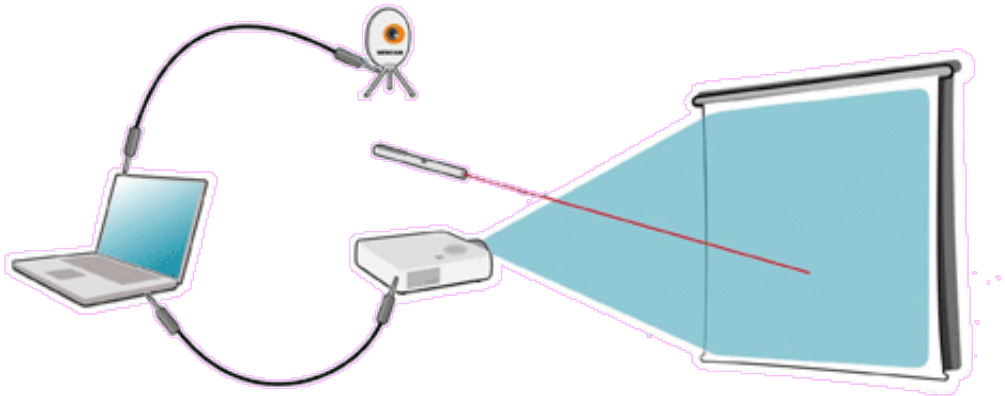


Figura 1: Esquema de la interacción con el puntero láser [1].

Constituye una alternativa a las ideas básicas de manipulación directa de objetos, habitualmente realizadas con el teclado o el ratón. La metáfora del escritorio se amplia así con los objetos que están “fuera”, esto es, los elementos del mundo real. Así, el usuario podría trabajar con un “escritorio extendido” al permitir la interacción de los objetos del mundo real con el escritorio virtual que se muestra en la pantalla de su computador. El uso de la VxC para crear esta extensión se basa en el trabajo de Baermann [3].

La solución de Millán [4] se integra con el equipamiento habitual en las aulas donde el docente emplea un computador y un proyector de vídeo. La mayor parte de los mandos del proyector incorporan un emisor láser, por lo que solo es necesario incorporar la cámara para extender el área de trabajo del cursor al entorno de trabajo habitual. La utilización del ratón sigue siendo posible y no es competitivo, en cuanto a que los periféricos habituales siguen siendo funcionales, es decir, en cuanto se mueve el ratón toma el control de cursor. Aunque, claro está, hay que establecer un modo de empleo puesto que no tendría sentido tener dos dispositivos apuntadores que dirijan al mismo cursor.

El resultado de este trabajo es una solución de bajo coste, por lo que se refiere al *hardware* empleado y la potencia de cálculo necesaria. El sistema propuesto es portable (funciona en diferentes plataformas software y *hardware*) y el interfaz exige el mínimo posible de adaptación o aprendizaje, al respecto de su uso.

4 Desarrollo

Veamos ahora cómo se ha diseñado la solución, para después pasar a hablar de la implementación del algoritmo en la aplicación final. El algoritmo utilizado se puede ver como una secuencia de bloques que la aplicación repite una y otra vez mientras el usuario no disponga lo contrario. Por brevedad de esta exposición, se ha obviado gran parte del control de errores.

Como parámetro de la llamada de la aplicación deberá ser posible inicializar la captura al contenido de un archivo de vídeo. Estos vídeos deben estar en un formato soportado por OpenCV, por sencillez del proceso se ha optado por AVI en color, sin compresión.

4.1 Diseño de la aplicación

Para implementar los objetivos propuestos, esa iteración que se ha de repetir un número indeterminado de veces, se muestra en forma de diagrama de bloques en la fig. 2. El computador ha de **analizar la escena** que tiene delante para tomar la decisión de cuál debe ser su repuesta. Para ello llevará a cabo la detección de la presencia de luz infrarroja utilizando el API de OpenCV. En nuestro caso son objetos caracterizados por una alta luminosidad.

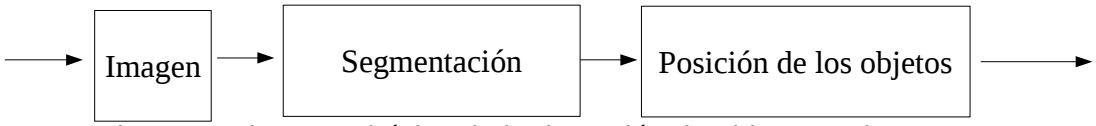


Figura 2: Diagrama básico de la detección de objetos en la escena.

Por simplicidad esta detección se realizará en base a un valor umbral decido a priori. Aunque no aparece en el diagrama, se dejará disponible un control para que el usuario pueda alterar esta variable y así explorar su influencia.

La etapa de **Imagen** comprende la toma de imágenes y la aplicación de los parámetros que el usuario pueda variar en tiempo de ejecución.

La etapa de **Segmentación** consiste en analizar la escena, con el objetivo de aislar los puntos que pertenecen al objeto de interés del resto de la imagen, el fondo.

La etapa de **Posición de los objetos** corresponde a la determinación de la posición del objeto de interés en coordenadas de la imagen.

Previamente a estos pasos es necesaria la inicialización de la cámara digital o del fichero de partida. Se ha obviado el control de errores por brevedad del código presentado, pero es totalmente recomendable comprobar que se puede acceder a la cámara o, si se indica, al fichero que contiene el vídeo; así como también a cada operación de captura de imagen que se pide a la cámara.

4.2 Implementación de la aplicación

Se ha obviado, en esta explicación, la parte del código correspondiente a las cabeceras, la declaración de tipos, constantes y funciones que constituyen elementos auxiliares a la ejecución del resto del código. No profundizaré en su detalle para dejar espacio a las explicaciones de la parte central del código.

El listado 1 contiene el grueso del programa principal que implementa la secuencia de la fig. 2:

- La inicialización de la aplicación comprende, líneas 11 a la 23, la de la cámara digital (*cvCreateCameraCapture*) o la del fichero de vídeo (*cvCreateFileCapture*) como fuente de entrada de la aplicación.
- El bucle principal, líneas 25 a la 44, que procesa la entrada del usuario (*cvWaitKey*), analiza escena, muestra los resultados y actualiza la imagen a procesar (*cvQueryFrame*).
- El punto central, el análisis de la escena (línea 36), lo vemos a continuación detallado.
- La posición de las diferentes fuentes de infrarrojos, el centro de gravedad (en adelante cdg) de todos ellos, la realiza la función *pintaCruz* (línea 38) que proporciona una forma visual de representar los valores numéricos correspondientes al cdg o centro de masas del objeto detectado.

  Esta decisión es puramente particular, por si se quiere utilizar un vector o una matriz de diodos led de luz infrarroja para tener mayor potencia luminosa. En otro contexto, esta función podría variarse para adaptarse a otras exigencias. En las pruebas realizadas, solo se muestran resultados con un diodo por ser el tipo de dispositivo más común de entre los de uso doméstico y, por tanto, facilitando el llevar a cabo esta experiencia.
- La liberación de recursos, líneas 46 a la 48, antes de finalizar la aplicación con *cvDestroyAllWindows*, *cvReleaseImage* y *cvReleaseCapture*.

```
int main(int argc, char* argv[])
{
 CvCapture* capture = NULL;
 IplImage *imgOrg, *imgDst;
 CvCapture* videoOrg;
 int tecla, nCuadre,  int tempsEntreCuadres = 25; // milisegons
BOOL bContinuar = TRUE;
 char* nombreFichero;
 CvPoint posCDG;

 if (argc > 1)  videoOrg = cvCreateFileCapture( argv[1] );
 else videoOrg = cvCreateCameraCapture( CV_CAP_ANY );

 cvNamedWindow( fORG, CV_WINDOW_AUTOSIZE );   cvMoveWindow( fORG, 0, 0 );
 cvCreateTrackbar( "Valor umbral", fORG, &valorUmbral, MAXPIXEL, cambioDeUmbral );

 imgOrg = cvQueryFrame( videoOrg );
 if (imgOrg == NULL ) { bContinuar = FALSE; }
 else { cvShowImage( fOriginal, imgOrg ); }

 imgDst = cvCreateImage(cvGetSize(imgOrg),imgOrg->depth, 1);
 cvNamedWindow( fResultat, CV_WINDOW_AUTOSIZE );
 cvShowImage( fResultat, imgDst ); cvMoveWindow( fResultat, imgOrg->width + 10, 0);

 while ( bContinuar)
 {
  tecla = cvWaitKey(tempsEntreCuadres) & 255;  // Espera una tecla los mseg. que haga falta
   switch ( tecla )
  {
   case ESC: case 'q':case 'Q': bContinuar = FALSE;  break;
   case 'u':  valorUmbral--; break;
   case 'U': valorUmbral++; break;
  default: ;
  } // Fin de "switch ( tecla )"

  analisisDeLaEscena( imgDst, imgDst, valorUmbral, &posCDG );
  cvShowImage( fResultat, imgDst );
  pintaCruz( imgOrg, posCDG.y, posCDG.x, 12 );
  cvShowImage( fOriginal, imgOrg );

   imgOrg = cvQueryFrame( videoOrg );
   if (imgOrg == NULL )  bContinuar = FALSE;

}// Fin de  "for(;;)"

 cvDestroyAllWindows( );
 cvReleaseImage( &imgOrg );  cvReleaseImage( &imgDst );
 cvReleaseCapture( &videoOrg );

 return 0;
} // Fi del programa principal
```

Listado 1: Código de la función principal de la aplicación.

El listado 2 muestra el detalle del análisis de la escena, siguiendo el esquema de bloques de la anterior fig. 2:

- La etapa de **Imagen** abarca las líneas 5 a la 15. Es recomendable comprobar que se puede acceder a la cámara o al fichero que contiene el vídeo.
- La etapa de S**egmentación** consiste en tres pasos:
    - Transformar la imagen de color (RGB) a niveles de gris con *cvCvtColor*, en la línea 17.
    - Umbralizar utilizando la función *cvThreshold* (línea 19,). Todos los puntos con valor de gris entre *umbral* y 255 se convierten en 255 y el resto se dejan a 0. Esta elección es para facilitar la visualización de los resultados intermedios, pero podría ser cualquier otra pareja de valores.
    - Determinar la posición mediante el cálculo (línea 20) del **cdg.** Consideramos objeto lo etiquetado a valor 255. El listado 3 muestra el

```
int analisisDeLaEscena( IplImage *imgOrg, IplImage *imgDst, int umbral, CvPoint *posCDG )
{
 IplImage *imgOrgGris;

 if (imgOrg->nChannels > 1)
 {
  imgOrgGris = cvCreateImage(cvGetSize(imgOrg), imgOrg->depth, 1);
  cvCvtColor( imgOrg, imgOrgGris, CV_RGB2GRAY );
 }
 else
  imgOrgGris = cvClone(imgOrg);

 if (imgDst == NULL)
    imgDst = cvCreateImage(cvGetSize(imgOrg), imgOrg->depth, 1);
 else
 if (imgDst->nChannels > 1)
     imgDst = cvCreateImage(cvGetSize(imgOrgGris), imgOrgGris->depth, 1);

cvCvtColor(imgOrg, imgDst, CV_RGB2GRAY);

cvThreshold (imgOrgGris, imgDst, umbral, 255, CV_THRESH_BINARY);

 cdg( imgDst, posCDG, 255);
 //pero podría ser al reves y tomar el valor 255 como etiqueta.

 return( 0 );
} // Fin de "int procesarImg( IplImage *imgOrg, int umbral, IplImage *imgDst )"
```

Listado 2: Código para la etapa de análisis de la imagen.

código de la función *cdg*.

El *cdg* se define como el punto central donde se cruzan las diagonales de la caja que contiene todos los puntos del objeto. Se puede calcular de forma iterativa, atendiendo a su formulación estadística de promedio de las coordenadas del objeto.

Esto es, recorriendo la imagen y acumulando las posiciones (*suma_x* y *suma_y* en la línea 12*)* de los puntos etiquetados como pertenecientes al objeto (con el valor que contiene el parámetro *grisObjeto), a*l tiempo que se incrementa el

contador de puntos que pertenecen al objeto (variable *n*). Si el número de puntos no es nulo (línea 15), se puede dividir el valor acumulado de coordenadas en cada eje por el cardinal de puntos. Si no (línea 17) se asigna un valor nulo.

```
int cdg( IplImage *imagen, CvPoint *posCDG, int grisObjeto )
{
 int suma_x, suma_y, n, i, j;
 CvScalar s;

 suma_x = 0;   suma_y = 0;   n = 0;

 for(i=0; i<imagen->height; i++)
  for(j=0; j<imagen->width; j++)
  {
     s=cvGet2D(imagen,i,j);
    if(s.val[0] == grisObjeto)  {    suma_x += j;              suma_y += i;       n++;   }
  }

 if( n != 0 )
 {        posCDG->x = suma_x / n;      posCDG->y = suma_y / n; }
  else{  posCDG->x = 0;                posCDG->y = 0;
 }

 return( 0 );
} // Fin de "int cdg( ...
```

Listado 3: Código para la etapa de cálculo del cdg.

5 Construcción del ejecutable y experimentos

La solución aquí presentada utiliza una distribución de *GNU/Linux Ubuntu 12.04*, una versión 2.3.1 de OpenCV. Para obtener el ejecutable es necesario compilar la aplicación contra las bibliotecas de funciones de OpenCV al estilo de la orden que se muestra en el listado 4.

```
$ gcc cdg_enVideo.c -o cdg_enVideo `pkg-config opencv --cflags –libs`

$ cdg_enVideo ficheroVideoConLuzArtificial.avi
```

Listado 4: Generación del fichero ejecutable y su invocación con parámetros.

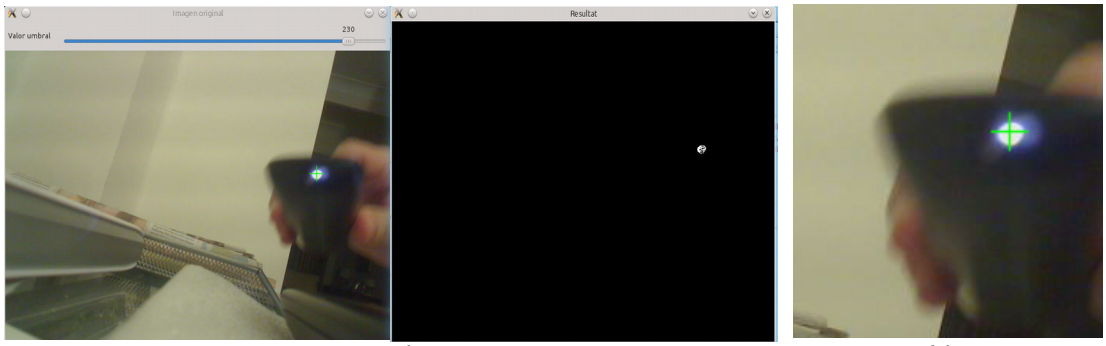
En ausencia de parámetros, se actuará sobre la cámara por defecto del sistema y, si se le pasa un argumento, se asumirá que es un vídeo sobre el que aplicar el proceso en lugar de sobre el flujo de imágenes en vivo de la cámara. Durante la realización de este documento se han generado[4] dos de características muy similares a partir de una cámara que, entre otros formatos, ofrecía la capacidad de enviar fotogramas a razón de 15 cuadros por segundo (*frames per second* o fps), con una resolución de 640x480 píxeles.

Para validar el algoritmo y para mostrar gráficamente las características de la escena donde se puede emplear este tipo de luz se han analizado dos posibles escenarios. El primer caso se refiere al uso de luz artificial, la fig.3a muestra el

[4] Se podría haber generado con OpenCV, lo dejo a la curiosidad del lector interesado. Véase <http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html?highlight=capture#bool%20VideoCapture::grab%28%29>. En este caso se han tomado con la aplicación *gucvview*.

resultado obtenido. La fig. 3a muestra la imagen RGB que devuelve la cámara y, al lado, el resultado del proceso de análisis de la imagen en la que los puntos del objeto de interés se ven en blanco y el fondo de la escena en negro. Si se mira con detenimiento, sobre el led encendido del mando utilizado, fig.3b, se verá pintada la cruz de color verde que muestra como se ha localizado el centro del objeto de interés en la escena.

Se ha utilizado un valor de 230 para obtener el resultado. Este valor se ha obtenido como resultado de un proceso de calibración consistente en variar el valor de umbralización hasta que maximice el número de puntos visibles (en blanco) de la imagen que pertenecen al led encendido. El resto de la imagen queda en negro.



a) *b)*

Figura 3: Un instante de la ejecución con luz artificial (a) y detalle (b).

Como segundo caso, la misma escena se ha grabado con doce horas de diferencia y sin otra fuente de luz que la natural presente en la imagen. Se ha mantenido el valor de umbral obtenido en la secuencia inicial de la fig. 3.

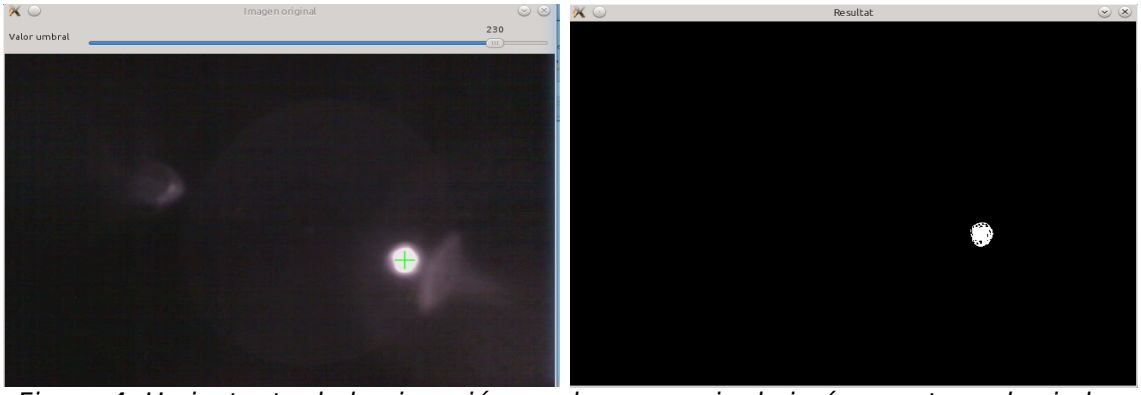


Figura 4: Un instante de la ejecución con la secuencia de imágenes tomada sin luz "ambiente".

La fig.4 muestra lo obtenido en ausencia de luz, lo que ejemplifica el buen comportamiento de este tipo de luz en ambientes con niveles bajos de luminosidad. La fig. 5, por el contrario, muestra la misma escena (imagen de la izquierda) en una franja horaria de alta incidencia de luz natural. Puesto que la luz solar también contiene componentes de luz infrarroja se observa (imagen de la derecha) que el algoritmo de detección ha detectado más puntos que los del emisor infrarrojo, lo que perturba el resultado obtenido, es decir, la posición de la cruz se ve desplazada respecto a la real del mando.

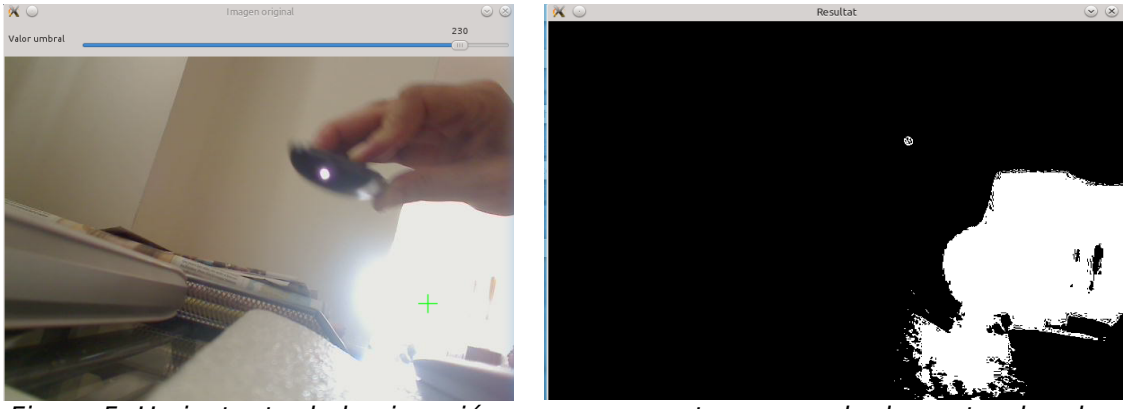


Figura 5: Un instante de la ejecución en un momento con mucha luz natural en la misma posición de la cámara.

6 Conclusión

Tras la lectura y experimentación con los contenidos relativos a este documento, el lector será capaz de:

- Procesar vídeos. Esto es, tomar secuencias de imágenes a través de una cámara digital o de un fichero de los formatos de vídeo soportados.
- Determinar las condiciones ambientales de luz que permiten trabajar con una fuente de luz infrarroja en la escena.
- Determinar las coordenadas de la imagen en las que se ubica el objeto de interés en la escena con el método de cálculo del centro de gravedad.

El usuario puede seguir explorando, a partir de aquí, cómo introducir las funciones necesarias para convertir el resultado de la aplicación desarrollada en eventos del sistema. La bibliografía le ayudará.

7 Bibliografía

[1] D. Millán y M. Agustí. (2004). Una aproximación a la integración del computador en el entorno de trabajo real. V Congreso Interacción Persona Ordenador. ISBN: 1-4020-4204-3.
[2] Open Computer Vision Library.<http://sourceforge.net/projects/opencvlibrary/>. Consultada el 17 de marzo de 2015.
[3] T. Baermann, M. Agustí y J. V. Benlloch. (2001). Expandiendo el escritorio; 2º Congreso Interacción Persona Ordenador. ISBN 84-7800-874-8.
[4] D. Millán. (2005). Desarrollo de un sistema señalador para presentaciones. PFC ETSInf.