Detección de objetos utilizando el sensor Kinect

Ing. Juan Eduardo Salvatore
UNAJ, Florencio Varela, Buenos Aires, Argentina, juaneduardosalvatore@gmail.com

Ing. Jorge Osio
UNAJ, Florencio Varela, Buenos Aires, Argentina, josio@unaj.edu.ar

Ing. Martin Morales
UNAJ, Florencio Varela, Buenos Aires, Argentina, martinmorales@unaj.edu.ar

RESUMEN

El objetivo de este trabajo fue realizar detección de rostros por medio del algoritmo de Haar que provee la librería OpenCV, de tipo open source ampliamente utilizada para el procesamiento de imágenes, conjuntamente con el sensor Kinect de Microsoft. A través de esta tecnología se lleva a cabo la captura de frames que posteriormente son procesados con el fin de poder detectar los ojos, nariz y boca de una persona. Este algoritmo es muy potente, pero a su vez puede llegar a ser lento por la gran cantidad de procesamiento que requiere, uno de los objetivos de las pruebas realizadas es justamente encontrar un equilibrio entre el procesamento de frames y el uso de las librerías. La potente combinación de una herramienta como kinect y la detección de rostros posibilita la detección de proximidad de las personas detectadas al equipo de procesamiento.

Palabras claves: Kinect, imágenes, Haar, OpenCV, rostros.

ABSTRACT

The purpose of this study was to perform face detection using the Haar Algorithm that is provided by the OpenCV library, which is open source type and widely used in image processing, along with the Microsoft Kinect sensor. With this technology the frames are captured to be subsequently processed in order to detect the eyes, nose and mouth of a person. This is a very powerful algorithm, but it can be slow due to the large amount of processing requiered, an objective of the tests performed is to find a balance between the processing frames and the use of the libraries. The strong combination of a tool like Kinect and face detection enables proximity detection of people that is detected by the processing equipment.

Keywords: Kinect, image, Haar, OpenCV, face.

1. INTRODUCTION

El procesamiento de imagenes tiene como objetivo mejorar el aspecto de las imágenes y hacer más evidentes en ellas ciertos detalles que se desean hacer notar. El campo de aplicación es amplio y nuevos estudios como así también nuevas tecnologías, tal y como lo es el caso del sensor Kinect mencionado previamente, han sido desarrollados.

Si bien el reconocimiento de rostros humanos puede tornarse difícil debido a que hay varios parámetros implicados, es un tema de interés cada vez mayor en diversas áreas de aplicación como por ejemplo la identificación personal. De esta forma lo que se pretende mostrar en esta publicación es una aplicación para la detección de un rostro en tiempo real, donde la implementación esta basada en OpenCV. Para esto se hara uso del

Kinect junto con el algritmo de Haar, mediante la combinacion de varias de sus librerias incluidas en OpenCV, para la detección de proximidad de sujetos al sensor Kinect.

2. ESTADO DEL ARTE

A continuación se desarrollan a modo introductorio los conceptos teóricos sobre lo que se conoce como "Luz Estructurada", el funcionamiento de la tecnología Kinect y sobre el final de esta sección se hará una reseña teórica sobre el algoritmo de Haar provisto por la librería OpenCV.

2.1 LUZ ESTRUCTURADA

Es el principio que usa el sensor Kinect (utilizado en este trabajo) para su funcionamiento. Consiste en proyectar un patrón de luz sobre una escena y observar la deformación del patrón en la superficie de los objetos. A la izquierda de la figura 1 se muestra un patrón de líneas y la derecha un patrón de puntos utilizado por el sensor Kinect, cuyo funcionamiento se detalla más adelante.

El patrón de luz puede ser proyectado, bien por un proyector LCD (luz no coherente) o bien por un barrido laser. Una cámara ligeramente desplazada respecto del proyector, captura la deformación de las líneas (o puntos) y calcula la distancia de cada punto utilizando una técnica similar a la triangulación. (Córdova Lucero, 2012)

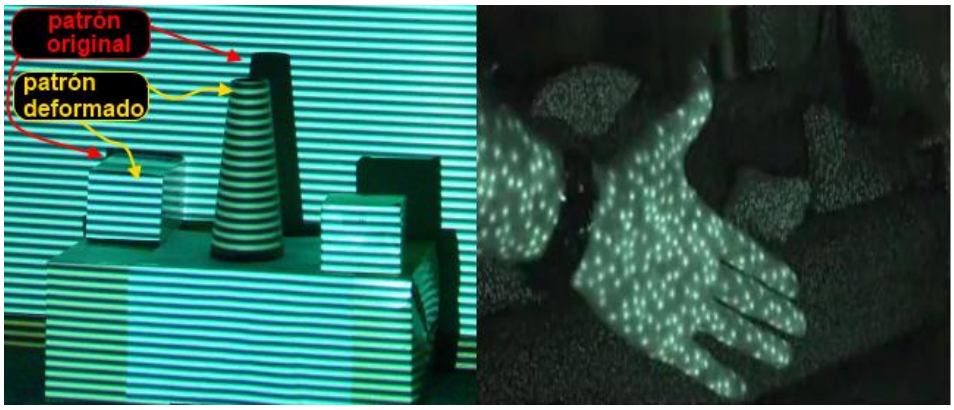

Figura 1: Patrones de luz estructurada con líneas y puntos (sensor Kinect). (Córdova Lucero, 2012)

En la figura 2 se puede observar cómo se emplea la técnica de triangulación para el caso de patrones de líneas. En una superficie plana, es de esperar que la línea capturada por la cámara sea recta. Una pequeña deformación en la misma puede ser convertida a una imagen con coordenadas 3D. Para ello se tiene que identificar cada línea, que se logra mediante el rastreo de cada línea (método de reconocimiento de patrones) o simplemente contándolas. Otro método muy común consiste en proyectar patrones alternativos formando una secuencia de código Gray que identifica el número de cada línea proyectada sobre el objeto. (Córdova Lucero, 2012)

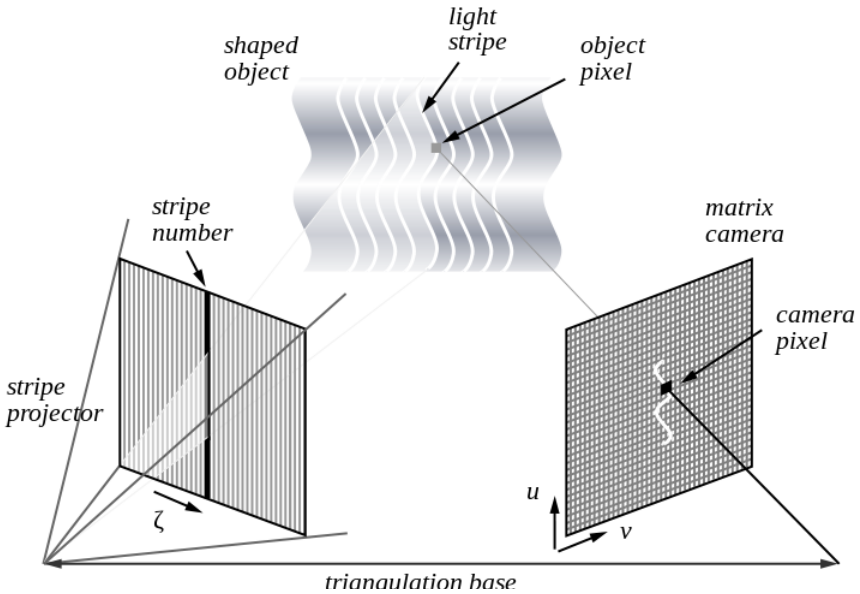

Figura 2: Principio de la triangulación aplicada a patrones de líneas (Córdova Lucero, 2012)

2.2 EL SENSOR KINECT

El sensor Kinect es un dispositivo lanzado en Noviembre de 2010 por microsoft, orientado principalmente a la industria de los videojuegos, concretamente, como periférico del video-consola Xbox 360 de Microsoft. Su principal innovación es que permite a los usuarios controlar e interactuar con la consola sin necesidad de tocar ningún controlador de juego físicamente, a través de una interfaz de usuario natural basada en gestos y comandos de voz. La apariencia de este dispositivo es la se muestra en la figura 3. (Microsoft Corporation, 2013)

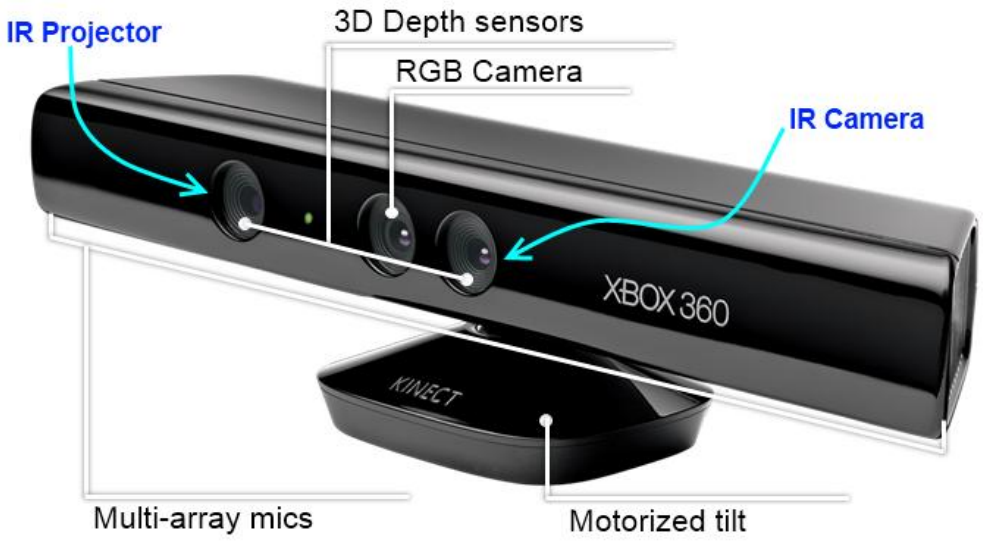

Figura 3: El sensor Kinect de Microsoft (Córdova Lucero, 2012)

Como se puede ver en la figura 3, son cuatro los elementos principales que componen el dispositivo: una cámara RGB en resoluciones de 640x480 (VGA) y 1280x1024 píxeles, sensor de profundidad (proyector IR + cámara IR) en resoluciones de 640x480 (VGA), 320x240 (QVGA) y 80x60 píxeles; un motor para controlar la inclinación del dispositivo y un arreglo de cuatro micrófonos distribuidos a lo largo del sensor. A continuación se describen los que resultan de interés para este proyecto. (Microsoft Corporation, 2013)

Cámara RGB: Dependiendo del SDK (Software Development Kit) utilizado, esta cámara es capaz de operar con dos formatos de imagen: formato RGB y formato YUV. En formato RGB, hasta treinta imágenes pueden ser generadas por segundo (30fps). Las imágenes en formato YUV sólo están disponibles en una resolución de 640x480 píxeles y a solo 15fps. (Microsoft Corporation, 2013)

Sensor de profundidad: Este sensor utiliza luz estructurada infrarroja para su funcionamiento. La fuente de luz infrarroja (laser más rejilla de difracción), proyecta un patrón de puntos sobre la escena que es leído por un sensor de infrarrojos monocromático CMOS. El sensor detecta los segmentos de puntos reflejados y estima la profundidad a partir de la intensidad y la distorsión de los mismos. Microsoft, la empresa propietaria del sensor, no ha hecho ninguna publicación en relación al algoritmo empleado para estimar la profundidad. Sin embargo, varios investigadores han intentado deducirlo por medio de un proceso de ingeniería inversa. La fig. 4 muestra un patrón de puntos claros y oscuros proyectados por el sensor Kinect. Según ROS (Robot Operating System), el algoritmo comienza calculando la profundidad de un plano de referencia a partir de los nueve puntos que aparecen muy marcados y guarda el patrón para ese plano. Posteriormente, la profundidad para cada pixel se calcula eligiendo una ventana de correlación pequeña (9x9 ó 9x7) y se compara el patrón local en ese píxel con el patrón memorizado en ese píxel y los 64 píxeles vecinos en una ventana horizontal. (Córdova Lucero, 2012)

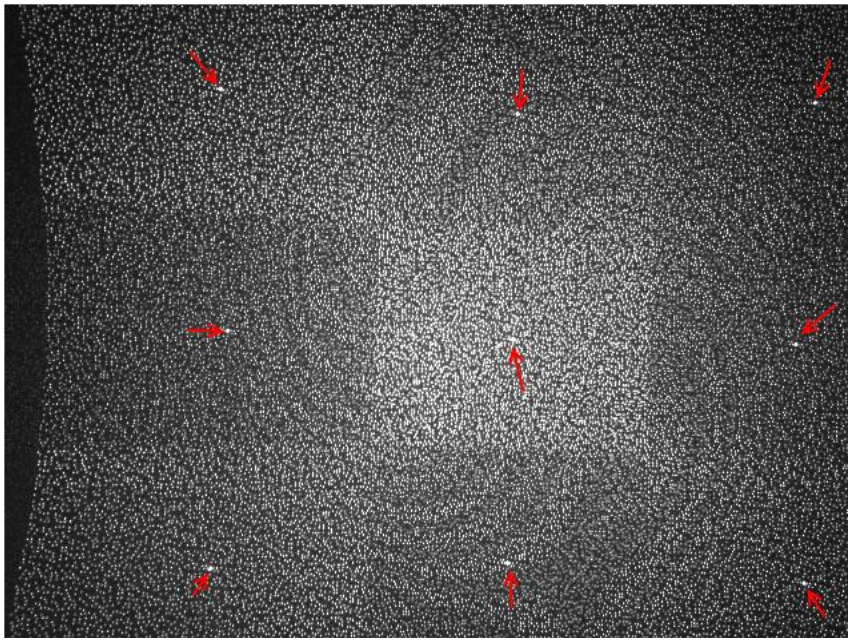

Figura 4: Patrón de puntos emitidos por el sensor Kinect (Córdova Lucero, 2012)

La mejor coincidencia da un offset (disparidad) respecto a la profundidad del plano de referencia, en términos de píxeles. Dadas la profundidad del plano memorizado y la disparidad, una profundidad estimada para cada píxel puede ser calculada por triangulación, de acuerdo a la expresión utilizada por un sistema estéreo normal. Por otro lado, la resolución en la dimensión z es de aproximadamente un centímetro, mientras que la resolución espacial (ejes x e y) es del orden de milímetros. La información de profundidad se devuelve en un mapa de píxeles con una frecuencia máxima de 30 imágenes por segundo. Cada pixel está representado por dos bytes (16 bits), cuyo valor

representa teóricamente la distancia del objeto al sensor. Para un funcionamiento correcto, deben satisfacerse las siguientes condiciones teóricas:

- Máxima (mínima) distancia del objeto al sensor: 3.5 (1.2) metros.
- Los objetos deben estar dentro del ángulo de visión: $\pm 43^{o}$ verticalmente y de $\pm 57^{o}$ horizontalmente.
- Relacionadas con el tipo de objeto (translucido, especular, cóncavo, etc.).
- Relacionadas con la iluminación (luz solar fuerte).

Si el valor de un píxel del mapa es cero, significa que el sensor no pudo estimar la profundidad para esa región por no cumplirse alguna de estas condiciones. (Córdova Lucero, 2012) (Microsoft Corporation, 2013)

Limitaciones: El sensor Kinect tiene varias limitaciones que hacen que la profundidad de ciertas regiones de la escena no se pueda estimar o si se estima, la fiabilidad de los datos no es aceptable. Estas limitaciones vienen condicionadas tanto por factores internos, debidos a la arquitectura del dispositivo; como externos, debidos a la naturaleza de la escena. En el primer grupo (factores internos) se tienen las siguientes limitaciones:

- Los puntos de luz no cubren de forma continua la superficie de los objetos, lo que conlleva a que algunos píxeles de la imagen de profundidad tienen que ser interpolados. Esto implica que el valor de profundidad de un píxel determinado tiene asociado un margen de error. Este margen es mayor cuanto más alejado está el objeto, puesto que, para una misma superficie, los puntos de luz están más separados. A mayores distancias, los valores de profundidad devueltos para objetos cercanos entre sí tienden a ser muy similares. Sin embargo, si el objeto está a demasiada distancia del sensor, no se calcula ninguna distancia para ese punto. Esto ocurre así, porque la potencia de luz del haz de infrarrojos se atenúa en el trayecto recorrido, haciendo que sea imperceptible para el sensor de infrarrojos.

Figura 5: Imagen de profundidad para distancias muy grandes (Córdova Lucero, 2012)

- Del problema descrito en el punto anterior se deriva también una imprecisión en los bordes de los objetos. Dado que para la estimación de la profundidad considera más de un píxel y al estar éstos dispersos, unas veces se tomará la profundidad del objeto más cercano y otras las del más lejano.
- Objetos cóncavos o cavidades reflectantes: éstos pueden producir reflexiones dobles e inter-reflexiones. Es decir, un punto de luz que caiga sobre una superficie cóncava, puede rebotar en otra zona del mismo objeto (cóncavo), lo que produce un solapamiento de los puntos de luz, haciéndolos nuevamente irreconocibles para el sensor.

Las zonas cuya profundidad no pudo ser resuelta, aparecerán como zonas negras (píxeles de valor cero) en la imagen de profundidad. Aparte, como ya se mencionó anteriormente, también hay que tener en cuenta la atenuación que sufre la luz, por lo que para objetos muy lejanos, tampoco se podrá determinar su profundidad. Asimismo, la inclinación de la superficie de los objetos respecto al proyector del haz de luz limita la detección de la profundidad. Si el rayo de luz es casi paralelo a la superficie no podrá incidir sobre la misma, haciendo imposible la estimación de la profundidad. En la fig. 6 se puede ver este efecto en la mesita de la esquina inferior derecha y en la superficie del suelo. (Córdova Lucero, 2012)

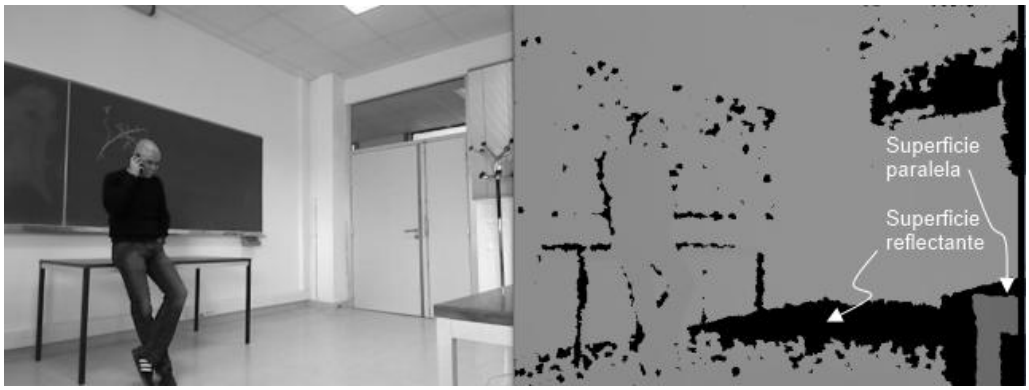

Figura 6: Efecto de superficie reflectante y paralela a los rayos de luz (Córdova Lucero, 2012)

2.3 DETECCIÓN DE OBJETOS (ALGORITMO DE HAAR)

La siguiente figura muestra la función principal de detección de objetos de OpenCV: **cvHaarDetectObjects**. Esta función, que se aplicó en este trabajo, depende de algunos parámetros: image, cascade, storage, storagescale_factor, min_neighbors, flags, min_size. (Bradski and Kaehler, 2008)

```
CvSeq* cvHaarDetectObjects(
const CvArr* image,
CvHaarClassifierCascade* cascade,
CvMemStorage* storage,
double scale_factor = 1.1,
int min_neighbors = 3,
int flags = 0,
CvSize min_size = cvSize(0,0)
);
```

Figura 7: Función de detección de objetos (García Chang, 2009)

Image es la imagen,

Cascade es el clasificador de cascada resultado del entrenamiento (ver párrafo más abajo),

Storage sirve para la memoria de almacenamiento para guardar los resultados,

Scale_factor es un factor de escala. La función **cvHaarDetectObjects**() escanea la imagen de entrada para rostros de cualquier escala, habilitando el factor de escala determina que tan grande será el salto entre cada escala, si se le pone a un mayor valor significará que el tiempo de computación será más rápido,

Min_neighbors es un factor que sirve para prevenir falsas detecciones, si se le coloca el valor (3) este indicará que solamente se decidirá si el rostro está presente después de haber hecho al menos tres detecciones,

Flags es un parámetro que puede tomar a cuatro factores válidos los cuales pueden ser combinados con un operador booleano **OR**:

1. cv_haar_do_canny_pruning, causa que cuando las regiones sean planas (por ejemplo un fondo homogéneo) se evita procesarlas.
2. cv_haar_scale_image, el cual le dice al algoritmo que escale la imagen en el lugar de escalar el detector.
3. cv_haar_find_biggest_object, permite regresar nada más el objeto más grande encontrado (por lo tanto el número de objetos regresados será uno o ninguno).
4. cv_haar_do_rough_search, el cual es utilizado solamente con cv_haar_find_biggest_object, utilizada para determinar la búsqueda de objeto cuando el primer candidato sea encontrado (con suficientes vecinos para que sea un acierto).

Min_size es el tamaño de la ventana más pequeña en el cual se hace la búsqueda del rostro, colocando este a un mayor valor reducirá el proceso de computación evitando encontrar rostros pequeños.

Los resultados del entrenamiento se encuentran catalogados en un archivo tipo xml. Este archivo describe el árbol de funciones de clasificaciones que se utilizo. Varios archivos se encuentran en la librería OpenCV entre los que pueden citar:

haarcascade_eye.xml
haarcascade_eye_tree_eyeglasses.xml
haarcascade_mcs_nose.xml
haarcascade_mcs_mouth.xml

Una función de clasificación (ver figura más abajo) incluye un descriptor de Haar, un umbral y dos valores predefinidos por la función de clasificación. La salida toma una de estos valores según que el resultado del descriptor sea superior o inferior al umbral de la función de clasificación. (Bradski and Kaehler, 2008)

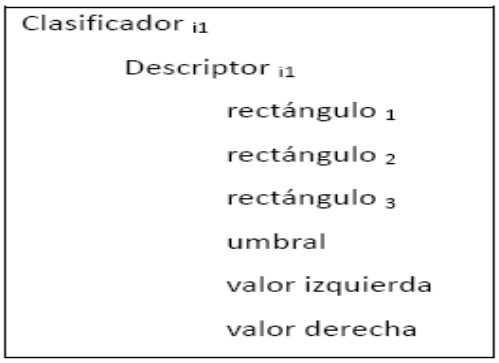

Figura 8: Estructura de la función de clasificación (García Chang, 2009)

Como la mayoría de la funciones de OpenCV son optimizadas para tener respuestas en tiempo real, un manera de tratar de mejorar el sistema de detección de rostro es el uso de detección del ojo. El archivo de Haarcascades para la detección de ojo lo cual es haarcascade_eye.xml, mencionado previamente. La afirmación o negación de la presencia de un rostro está condicionada a la presencia de un ojo en la zona de rostro.

3. DISEÑO DE LA APLICACIÓN

Antes de mencionar detalles respecto al diseño de la aplicación vale la pena mencionar aspectos de las herramientas de programación utilizadas.

3.1 OPENCV

Es una biblioteca open source escrita en C/C++, desarrollada inicialmente por Intel, que es utilizada para el procesamiento de imágenes y visión computarizada. Entre las características principales de la misma se pueden mencionar:

- Disponible en Linux, Mac y Windows.
- Posee estructuras básicas de datos para operaciones con matrices y procesamiento de imágenes.
- Permite visualizar datos muy fácilmente y extraer información de imágenes y videos.
- Tiene funciones de captura y presentación de imágenes.

OpenCV fue diseñado para eficiencia computacional y con un fuerte enfoque en aplicaciones de tiempo real. Está escrito en C optimizado y puede tomar ventaja de procesadores de núcleo múltiple. (Bradski and Kaehler, 2008)

Se compone de cuatro módulos fundamentales:

- cv: Contiene las funciones principales de la biblioteca.
- cvaux: Contiene las funciones auxiliares.
- cxcore: Contiene las estructuras de datos y funciones de soporte para algebra lineal.
- Highgui: Funciones para el manejo de la GUI.

3.2 ENTORNO DE DESARROLLO INTEGRADO (IDE)

El elegido para el desarrollo de la aplicación fue Visual Studio de Microsoft debido a que los distintos desarrollos realizados con Kinect están hechos en el IDE mencionado. En esta oportunidad se utilizo la versión Visual C++ 2010 Express.

3.3 SDK – CODELABORATORIES

CodeLaboratories provee soluciones en áreas como seguridad digital, vigilancia y procesamiento de señales. Entre sus productos encontramos la plataforma CL NUI, que distribuyen de manera gratuita. Esta plataforma es recomendada para usuarios de Kinect debido que ofrece un controlador y SDK (software developer kit) para múltiples dispositivos Kinect en Windows XP, vista y 7 tanto con los sistemas x64 y x86.

El link de descarga es http://codelaboratories.com/downloads/

Si bien existe la posibilidad de utilizar el Microsoft SDK se opto por hacer del descripto previamente debido a que cuenta con los drivers necesarios para utilizar el kinect y ocupa poca memoria de disco. La desventaja de esta elección es que no hay mucha documentación o soporte para el mismo.

3.4 DISEÑO DEL ALGORITMO

Luego de configurar correctamente el Visual Studio a fin de poder utilizar la librería OpenCV y SDK se procedió a diseñar el algoritmo que, una vez inicializado el Kinect, sobre la imagen RGB obtenida hace la detección de los

ojos, nariz y boca a través de tres funciones para detectar cada uno de los parámetros mencionados. A continuación se muestra el código que estable el clasificador de Haar y la función que lleva a cabo la detección de ojos. Para los otros parámetros los cambios a realizar son mínimos.

```
// Se establecen los clasificadores a utilizar.
   cascade =(CvHaarClassifierCascade*)cvLoad("haarcascade_eye.xml",  0, 0, 0 );
   storage = cvCreateMemStorage( 0 );

// Definición de función que detecta los ojos.
void detectFaces( IplImage *img )
{
int i;
CvSeq *faces = cvHaarDetectObjects(
img,
cascade,
storage,
1.1,
3,
0,
cvSize( 40, 40 ) );
for( i = 0 ; i < ( faces ? faces->total : 0 ) ; i++ ) {
CvRect *r = ( CvRect* )cvGetSeqElem( faces, i );
cvRectangle( img,
cvPoint( r->x, r->y ),
cvPoint( r->x + r->width, r->y + r->height ),
CV_RGB( 255, 0, 0 ), 10, 8,0);
}
cvShowImage( "video", img );
}
```

4. RESULTADOS

La siguiente imagen ilustra el resultado de la detección en la que puede observarse en los rectangulos en color rojo, y a pesar de que el sujeto no esta mirando de frente al Kinect la detección puede lograrse, con la salvedad que solo se detecta un ojo. La mínima distancia a la que se realiza la detección es a 50cm, debido a las características funcionales de kinect.

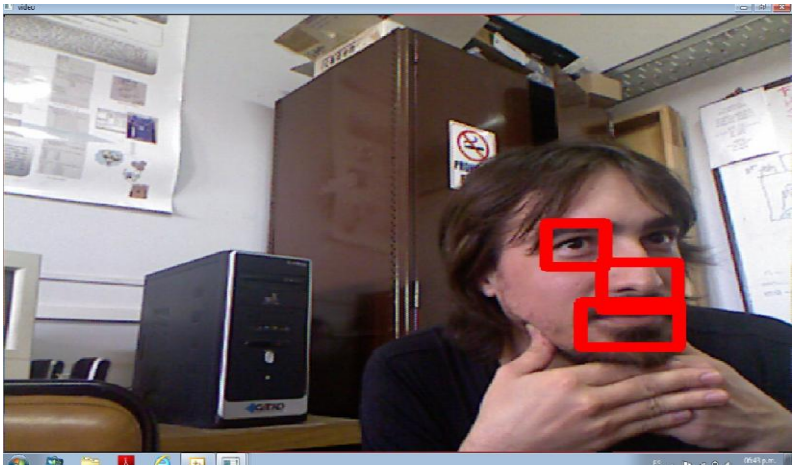

Figura 9: Detección realizada con el algoritmo de Haar (Frank and Salvatore, 2013)

Las distintas pruebas realizadas permitieron determinar que debido al alto procesamiento necesario para implementar este algoritmo, el hecho de utilizar tres librerías para detectar distintos aspectos del rostro hace que la máxima velocidad de procesamiento obtenida con suficiente calidad y fluidez de la imagen es de 20 frames por segundo, debido a esto es que se observa en la figura 9 que se están detectando 3 características a la vez, cuando en realidad la detección es secuencial.

5. CONCLUSIONES

El objetivo principal del trabajo era combinar dos herramientas tan poderosas como el Kinect y las librerías Open, lo cual se logro mediante una configuración fina de los parámetros esenciales de ambas librerías. Por otra parte se puede afirmar que las librerías de Haar son muy potentes para la detección de características en cuerpos humanos, pero requieren de un estudio profundo y de una configuración precisa para su uso en aplicaciones específicas, en este trabajo se logro encontrar un equilibrio para el uso optimizado y eficiente de estas librerías en la detección de rostros y lo interesante es que con que se pueda observar un rasgo tan detallado como labios u ojos el algoritmo puede detectar un rostro, aunque esté cubierto en gran medida. En cuanto a la herramienta utilizada, si bien la mínima distancia del objeto al Kinect debe ser 1.2m (idealmente), la detección pudo realizarse a partir de 50cm, lo que permite tanto detectar presencia para evitar proximidad a un objeto, como también a distancias relativamente grandes.

REFERENCIAS

Bradski Gary and Kaehler Adrian. (2008). Learning OpenCV.

Córdova Lucero Fabricio Alexander. (2012). "Detección de robo/abandono de objetos en interiores utilizando cámaras de profundidad", Tesis de Grado, Universidad Autónoma de Madrid, Madrid, España.

García Chang María Esther. (2009). "Diseño e implementación de una herramienta de detección facial", Tesis de Maestria, Instituto Politécnico Nacional, México DF.

Gonzalez Rafael and Woods Richard. (2007). Digital Image Processing (3rd edition).

Frank Andres and Salvatore Juan Eduardo. (2013). "Detección de objetos usando el sensor Kinect", Trabajo de investigación, Universidad Nacional de La Plata, La Plata, Argentina.

Microsoft Corporation. (2013). Kinect for Windows.

OpenCV Org. (2013). The OpenCV Reference Manual Release 2.4.6.0, 1. http://opencv.org/documentation.html

OpenCV Org. (2013). The OpenCV User Guide Release 2.4.6.0, 1. http://opencv.org/documentation.html

Szeliski Richard. (2010). Computer Vision: Algorithms and Applications.