Visión por Computador para Robots Mindstorms NXT

Miguel Angel Astor Romero, David Perez Abreu, Maria Elena Villapol
Centro CICORE, Laboratorio ICARO, Escuela de Computación
Facultad de Ciencias, UCV
Caracas, Venezuela
miguel.astor@ciens.ucv.ve, david.perez@ciens.ucv.ve, maria.villapol@ciens.ucv.ve

***Resumen* — El uso de dispositivos que permitan incorporar visión en robots es una característica deseable, pero realmente costosa; por lo que su uso en instituciones modestas de investigación es realmente difícil. Partiendo de la premisa anterior en el presente trabajo de investigación se presenta una solución factible para incorporar visión por computador a un robot Lego Mindstorms NXT haciendo uso de un smartphone vía una conexión Bluetooth. La incorporación de visión por computador se logra mediante la ejecución de una aplicación en el smartphone, la cual está basada en la biblioteca OpenCV4Android. Específicamente en esta investigación se implementó una clase dentro de la aplicación encargada de la recepción de imágenes de la cámara que permite que la imagen fuese procesada con OpenCV, y desplegada por pantalla haciendo uso de una textura de OpenGL ES.**

Palabras claves—Robot; Lego Mindstorms NXT; Visión por Computador; Smartphone; Android.

I. INTRODUCCIÓN

La robótica es una rama de la tecnología con una larga trayectoria, ya que ha sido muy estudiada y desarrollada; en particular la palabra robot se usó por primera vez en 1921 en la obra de teatro titulada Rossum's Universal Robots de Karel Čapek [1]. Desde entonces el uso de dicha palabra y todos los elementos que la rodean se han hecho presentes y cotidianos en nuestra sociedad. En la década de los sesenta el uso de un robot solo tenía sentido cuando su intención era la de relevar a un trabajador humano de una labor aburrida, desagradable o demasiado precisa; algunos ejemplos son el uso de robots industriales diseñados para trabajos en ambientes peligrosos, robots soldadores, robots usados en áreas de seguridad y militares, exploraciones espaciales, etc.

En la última década el surgimiento de diferentes proyectos y productos comerciales relacionados a la robótica como: kits de ensamblaje, módulos programables y pequeños juguetes autónomos; y en particular proyectos como el MIT Programmable Brick [2] y el kit Mindstorms de Lego [3], han permitido modificar el paradigma sobre el uso de la robótica ampliando sus espacios e incursionando en nichos de índole educativo, permitiendo de este modo el acercamiento de la robótica al público general. Un ejemplo son los llamados robots tortuga, inventados entre los años 1967 y 1969 junto al lenguaje de programación Logo por Seymour Papert y Wallace Feurzeig del grupo de investigación Bolt, Benarek and Newman de Massachusetts [4]. Estos robots junto al lenguaje de programación fueron exitosamente utilizados por Papert y sus colaboradores en varios niveles de educación, principalmente primaria, para fomentar el interés de los estudiantes, y difundir la práctica de temas relacionados con la programación y la tecnología.

El MIT Programable Brick y los robots Mindstorms de Lego, son ejemplos más recientes de este enfoque. El kit de Lego permite a sus usuarios diseñar y programar diferentes modelos de robots utilizando un enfoque modular. El kit en si no corresponde a un robot completo, sino que en su lugar provee una amplia variedad de piezas (incluyendo sensores), las cuales pueden combinarse de distintas maneras para formar robots u otros tipos de máquinas autónomas. El kit incluye un bloque especial que contiene un micro computador el cual actúa como unidad de procesamiento del robot. Lego incluye 4 sensores básicos en el kit Mindstorms NXT 2.0, la versión más reciente del kit. Estos son un sensor de tacto, un sensor de sonido, un sensor de ultrasonido, y un sensor puntual de color. Adicionalmente se pueden adquirir sensores especializados, ya sean de Lego o de otros fabricantes. Ejemplos de estos sensores son los de temperatura, orientación (brújula), acelerómetros, cámaras (detectores de manchas), entre otros. Estos sensores pueden combinarse y programarse de múltiples maneras dependiendo de las necesidades del robot en cuestión; sin embargo, ninguno de estos sensores facilita funciones complejas de visión al robot, término que en este contexto se define como el uso de técnicas de procesamiento de imágenes para digitalizar información sobre el ambiente en el cual se desenvuelve el robot.

Teniendo en cuenta la situación planteada anteriormente, y luego de realizar una búsqueda exhaustiva de trabajos relacionados, en la presente investigación se plantea modificar la aplicación MINDdroidCV de Richárd Szabó [5] con el objetivo de incorporar verificación de sensores de forma que el robot sea capaz de obtener información más completa sobre su entorno. Adicionalmente se propone la integración de un smartphone (Motorola Milestone) basado en el sistema operativo Android con los robots NXT por medio de Bluetooth. Dicha integración tiene como ventaja no solo el hecho de incorporar visión al robot (usando la cámara embebida en dicho dispositivo); sino que además se tiene a disposición la capacidad de procesamiento del smartphone la cual es considerablemente superior a la provista por el bloque inteligente de Lego.

Una integración robot-smartphone como la planteada anteriormente brindará la oportunidad de incorporar

mecanismos de control de mayor complejidad a los que pueden obtenerse usando los sensores básicos incluidos en el kit de Lego. Todo esto teniendo en cuenta que no será necesario incorporar un equipo especializado, ya que cualquier smartphone ejecutando Android (mayor a la versión 2.2) puede integrarse a la solución planteada. Un ejemplo de un comportamiento complejo haciendo uso de la fusión robot-smartphone es la posibilidad de incorporar el uso de tecnologías de realidad aumentada a dicho conjunto, logrando de esta manera que el robot pueda interactuar con objetos o entes virtuales los cuales son percibidos a través del smartphone.

El resto del artículo se encuentra estructurado de la siguiente manera. En la Sección II se presenta un estado del arte donde se describe el estado actual de la investigación que involucra a robots Mindstorms NXT. La descripción de las diferentes tecnologías utilizadas en este trabajo de investigación, específicamente la biblioteca OpenCV, los robots Mindstorms NXT y el teléfono Motorola Milestone se exhiben en la Sección III. Una reseña detallada de la integración de la visión por computador usando el smartphone, así como una descripción minuciosa de todos los aspectos tomados en consideración conforma la Sección IV. Finalmente, la Sección V presenta las conclusiones de la investigación y los trabajos futuros.

II. Trabajos Relacionados

La robótica móvil es un área de trabajo e investigación que actualmente cuenta con amplia aceptación en diversos ámbitos; un ejemplo son los robots Lego Mindstorms. Estos robots son publicitados como un paquete dirigido a aficionados; sin embargo, al ser de hardware abierto proporcionan una impresionante versatilidad que los hace aptos para su uso en ambientes académicos y estudiantiles, donde es difícil o incluso imposible adquirir equipos de robótica profesionales.

Muchos trabajos se han realizado con la intención de aplicar estos robots para fomentar el estudio de carreras científico-tecnológicas. Trabajos como [6], [7] y [8] se enfocan en el uso de los robots Mindstorms NXT en ambientes educativos. En [6] Menegatti y colaboradores plantean un esquema de enseñanza construccionista basado en la robótica; en este trabajo de investigación se diseñaron diversas actividades y evaluaciones para niveles de educación desde secundaria hasta maestría. Por su parte, Behrens en [7] diseña un sistema que permite ejecutar programas en robots NXT de manera remota utilizando MATLAB [9], orientado a los estudiantes de Ingeniería y Ciencias de la Computación. En [8] se describe un experimento social que utiliza robots NXT para incentivar el estudio de asignaturas relacionadas con la tecnología en estudiantes alemanes de secundaria.

El uso de robots en áreas diferentes a la educación también es una realidad; por ejemplo, la creación de enjambres de robots se ha investigado en los trabajos [10], [11] y [12]; sistemas distribuidos para control basados en robots se detallan en [7], [12] y [13]; así como frameworks de alto nivel para programar a los robots son descritos en [7], [13], [14] y [15]. Pásztor y colaboradores describen en [10] el trabajo que realizaron para diseñar un esquema de comunicación para robots basado en Bluetooth usando una red de tipo Scatternet. Este esquema de comunicación permite formar redes de robots NXT con más de 4 dispositivos, con la finalidad de que los mismos puedan resolver tareas en conjunto vía una constante comunicación; el caso de estudio de esta investigación es una simulación de hormigas buscando alimento. En [12] se presenta el diseño de una arquitectura distribuida para controlar múltiples robots móviles, utilizando una serie de computadoras de escritorio clientes para realizar el control; las cuales a su vez se encuentran subordinadas a una computadora servidor encargada de coordinarlas. Magnenat y colaboradores en [11] describen el diseño e implementación de un entorno de desarrollo integrado llamado ASEBA, el cual permite escribir y depurar programas que se ejecutan sobre enjambres de robots móviles basados en microcontroladores.

Los trabajos relacionados con el estudio de frameworks de alto nivel para robots representan la mayor cantidad de trabajos de investigación en el área; siendo el principal interés de todos los trabajos examinados el ocultar la complejidad de la interacción a bajo nivel entre dispositivos Bluetooth y los robots. En particular en las investigaciones donde se utiliza el kit de Lego Mindstorm, el objetivo principal es tratar de ocultar la complejidad del protocolo LCP (Lego Communication Protocol – Protocolo de Comunicación Lego) definido por el Bluetooth Developers Kit [16].

El uso de dispositivos móviles (smartphones, PDAs, etc.) como sistemas de control para estos robots también ha sido un área de interés para la investigación y desarrollo; sin embargo, cabe destacar que al menos en el ámbito académico, la implementación de soluciones de visión por computador como un sensor adicional para los robots Mindstorms NXT no ha sido del todo explorada. En [17] Stephan Göbel y colaboradores implementan un sistema de control remoto utilizando dos dispositivos Android (uno como “ojo” usando la cámara y otro como controlador por medio de Bluetooth); otra investigación relacionada es la desarrollada por Tae-Hoon en Lee en [18] donde se implementa un sistema de rastreo usando reconocimiento facial con cámaras web inalámbricas incorporadas al robot. Luego de una revisión amplia del tema solo estas dos investigaciones en el área de visión por computador en robots Lego Mindstorms fueron encontradas; hecho que denota que existe una línea de investigación potencial en dicha área.

Con el objetivo de brindar un aporte en el área de visión por computador en robots Lego Mindstorms en el presente trabajo de investigación se propone incorporar la verificación de sensores de forma que el robot sea capaz de obtener información más completa sobre su entorno, en particular la capacidad de visión vía una cámara. A diferencia de los trabajos que han sido descritos anteriormente, este trabajo de investigación incorpora un smartphone al kit de Lego Mindstorms NXT, con la finalidad de brindar capacidades de visión (usando la cámara embebida en el smartphone) y de procesamiento digital de imágenes a dicho dispositivo, haciendo uso de Bluetooth para la comunicación y la biblioteca OpenCV; todo esto tomando como referencia la aplicación MINDdroidCV de Richárd Szabó [5] disponible para el sistema operativo Android.

III. MARCO TECNOLÓGICO

En esta sección se introducen los conceptos y detalles concernientes a las herramientas de hardware y software utilizadas durante este trabajo de investigación.

A. *Lego Mindstorms NXT*

El kit Mindstorms NXT [3] es un paquete de robótica modular vendido por Lego como un sistema de nivel introductorio para entusiastas de la robótica. Es muy popular entre aficionados y en entornos educativos. El kit consiste en un conjunto de 577 piezas entre las cuales se encuentran el bloque controlador, un CD con software asociado y algunas etiquetas. El bloque controlador es la unidad central de procesamiento de los robots que se construyen con el kit. Este bloque está compuesto por un procesador ARM Atmel de 32 bits funcionando a 45 MHz. Este procesador dispone de 64 KB de memoria RAM, y 256 KB de memoria flash para almacenamiento persistente. Adicionalmente el bloque contiene un coprocesador Atmel AVR de 8 MHz. Es posible establecer comunicación con el bloque utilizando ya sea la interfaz USB 2.0 o el protocolo de comunicaciones Bluetooth. Si se desea establece una comunicación usando Bluetooth, el bloque dispone de un chipset Bluecore 4 que implementa el perfil de puerto serial de Bluetooth.

Lego provee documentación asociada al robot llamada *Development Kits* (Kits de Desarrollo), dirigida a usuarios avanzados del sistema. Existen 4 Development Kits: el BDK (*Bluetooth Development Kit* – Equipo de Desarrollo de Bluetooth) que detalla el funcionamiento del protocolo de comunicación LCP utilizado por el bloque y el hardware de la interfaz Bluetooth; el HDK (*Hardware Development Kit* – Equipo de Desarrollo de Hardware) que detalla el funcionamiento y la estructura electrónica del hardware del bloque y sus sensores; y por último 2 SDK (*Software Development Kits* – Equipos de Desarrollo de Software), donde uno de ellos detalla el software controlador del robot, y otro el funcionamiento de la máquina virtual que ejecuta programas de usuario en el bloque. Los robots Mindstorms NXT pueden ser programados usando una gran cantidad de tecnologías y lenguajes de programación. El entorno básico es provisto por Lego con el kit y consiste en un IDE (*Integrated Development Environment* – Ambiente de Desarrollo Integrado) visual llamado NXT-G. Este entorno se caracteriza por ser sumamente fácil de aprender y utilizar. Los robots se programan combinando una serie de bloques que se ejecutan linealmente sobre carriles paralelos. Estos bloques controlan los distintos componentes electrónicos del robot y pueden combinarse utilizando diferentes estructuras de control como lo son condicionales y ciclos.

De manera alternativa, dada la calidad abierta del hardware y el software de los kits Mindstorms NXT, existen una gran variedad de entornos de desarrollo no oficiales relacionados con Lego. Algunos de los más conocidos son el proyecto LeJOS [19] , que suplanta al firmware estándar de los bloques NXT y permite ejecutar programas escritos en un subconjunto del lenguaje Java sobre el robot. Otro es el proyecto NXC [20] (Not eXactly C – No eXactamente C), que permite compilar programas escritos en un lenguaje similar a C para ser ejecutados sobre el firmware básico. También es posible utilizar entornos especializados para robótica como lo son el Microsoft Robotics Developer Studio 4 [21], y el sistema Player/Stage [22].

Para este trabajo de investigación se utilizó un modelo de robot basado en la propuesta de Tovar y Tosiani en [23]. El modelo propuesto por ellos puede verse en la Fig 1.

Fig 1. Robot propuesto por Tovar y Tosiani

La configuración de los sensores del modelo propuesto en [23] fue modificada como puede observarse en la Fig 2. Adicionalmente al robot se le incorporó un soporte que permitió la inserción y traslado del smartphone.

Fig 2. Modificaciones realizadas al robot propuesto por Tovar y Tosiani

B. *OpenCV*

OpenCV [24] es una compleja biblioteca de algoritmos de visión por computador y aprendizaje de máquinas. Actualmente es desarrollada por Itseez como software libre distribuido bajo una licencia BSD de dos cláusulas. Su propósito es proveer una interfaz de programación completa para el desarrollo de soluciones de visión digital, control de robots, así como análisis de imágenes y video.

OpenCV tiene interfaces de programación para los lenguajes C, C++, Java y Python. Tiene versiones disponibles para los sistemas operativos Windows, GNU/Linux, Mac OS X, Android e iOS. La versión para Android se llama OpenCV4Android y al momento de realizar esta investigación se encuentra en la versión 2.46. Puede ser instalada en dispositivos Android usando el IDE Eclipse con el Android SDK, o directamente desde la Google Play Store. Puede ejecutarse en Android 2.2 o superior.

C. OpenGL ES

OpenGL ES es un API (*Application Programming Interface* – Interfaz de Programación de Aplicaciones) que define una biblioteca para despliegue de gráficos 2-D y 3-D basada en un subconjunto de OpenGL, diseñada para su funcionamiento en sistemas embebidos. Al igual que con OpenGL, el estándar es mantenido por el consorcio *Kronos Group*, compuesto por una gran diversidad de empresas productoras de hardware y software gráfico [28].

Actualmente existen dos grandes versiones del estándar OpenGL ES. La primera versión es el estándar OpenGL ES 1.1, basado en OpenGL 1.5, el cual define una biblioteca para despliegue por medio de un *pipeline* gráfico fijo, lo que quiere decir que todas las operaciones de transformación e iluminación están predefinidas por la biblioteca. La segunda versión es OpenGL ES 2.0, que se basa en OpenGL 2.0 e introduce un pipeline gráfico programable con la posibilidad de ejecutar ciertos programas llamados *shaders* en el hardware gráfico, los cuales permiten a los desarrolladores definir sus propias operaciones de transformación e iluminación. Esta versión no es retro compatible con OpenGL ES 1.1. El sistema operativo Android cuenta con soporte para OpenGL ES 1.1 en todas sus versiones, y para OpenGL ES 2.0 a partir de Android 2.2 (API 8).

D. Motorola Milestone

El smartphone Motorola Milestone (también conocido como Motorola Droid) es un dispositivo Android fabricado y distribuido por Motorola desde noviembre de 2009. Por defecto ejecuta el sistema Android 2.1, actualizable a Android 2.2. Es completamente compatible con CyanogenMod 7 (Android 2.3.7), y tiene soporte parcial para CyanogenMod 9. Dicho smartphone cuenta con un procesador Cortex A8 a 600 Mhz, y un GPU PowerVR SGX530, además de soporte para redes 2G, 3G, WiFi y Bluetooth 2.1. Dispone de unos 200 MB de memoria RAM, una pantalla táctil capacitiva y un teclado QWERTY. Su tamaño modesto (11,5 x 6 x 1,3 cm), poco peso (165 gramos) y su cámara posterior lo hacen particularmente interesante como candidato a ser utilizado como sensor de visión en los robots Mindstorms NXT.

IV. Implementación de la Solución y Resultados

El presente trabajo comenzó por probar la aplicación MINDdroidCV en múltiples dispositivos. Esta aplicación se deriva de la aplicación MINDdroid [25] de Lego desarrollada por Shawn Brown y Günther Hölzl, la cual es distribuida según los términos de la licencia GNU GPL (*General Public License* – Licencia Pública General) versión 3. MINDdroid permite controlar un robot Mindstorms utilizando el acelerómetro de un dispositivo Android para determinar la potencia de los motores, lo que altera el movimiento del robot.

MINDdroidCV es una modificación de MINDdroid que incorpora un tipo de control adicional, el cual utiliza OpenCV4Android para realizar rastreo de luces, haciendo que el robot se mueva en dirección de la fuente de luz más intensa visible a la cámara del dispositivo Android. Este control funciona utilizando los momentos centrales de la imagen capturada por la cámara; sin embargo, esta imagen primero debe pasar por un proceso de binarización por umbral. Al calcular los momentos centrales de la imagen se obtiene información sobre la ubicación del centroide (centro de masa de los píxeles blancos) de la misma. Este cálculo permite a la aplicación determinar en que dirección relativa al centro del campo de visión de la cámara se encuentra la(s) fuente(s) de luz visible(s) de mayor intensidad.

A. Experimentación

Al leer la documentación y los códigos de ejemplo de OpenCV4Android, se observa que el esquema básico de una aplicación de OpenCV4Android que utiliza la cámara nativa de Android consiste en por lo menos los siguientes componentes: una actividad principal como lo dictamina el SDK de Android; y una clase que implementa la interfaz *Surface Holder*. Esta clase debe registrar un objeto *PreviewCallback* que a su vez implementa el método *onPreviewFrame*, el cual recibe las capturas de la cámara que serán procesadas.

Bajo este esquema la actividad principal se encarga de la inicialización de componentes, el manejo de la interacción con el usuario, el ciclo de vida de Android, y lo más relevante a OpenCV, la creación de la interfaz de usuario y la superficie que despliega el *preview* de la cámara. *Surface Holder* se encarga por su parte de recibir este *preview* y pasarlo a OpenCV4Android para su procesamiento.

El flujo normal de una aplicación OpenCV4Android es entonces el siguiente (este flujo puede variar según las necesidades de cada aplicación):

1) *Inicio:* La actividad principal inicializa la cámara y el estado global de la aplicación. Luego crea una instancia de la clase *Surface Holder*.
2) *Creación de la superficie:* La clase *SurfaceHolder* puede o no crear una superficie sobre la cual se mostrarán los *previews* de la cámara.
3) *Captura:* Durante el ciclo de vida de la aplicación la cámara captura imágenes que son pasadas a *Surface Holder* por medio del método *onPreviewFrame* el cual es llamado por cada captura realizada.
4) *Procesamiento:* Cada captura es procesada por OpenCV según las necesidades de la aplicación.
5) *Despliegue: Surface Holder* muestra la imagen resultante del procesamiento de OpenCV.

Este flujo puede observarse en detalle en la Fig 3. En particular los pasos 8-10 (que equivalen a los pasos 3, 4 y 5 de la lista anterior) se repiten indefinidamente mientras se necesite

la cámara, alrededor de 30 iteraciones por segundo dependiendo del rendimiento global de la aplicación.

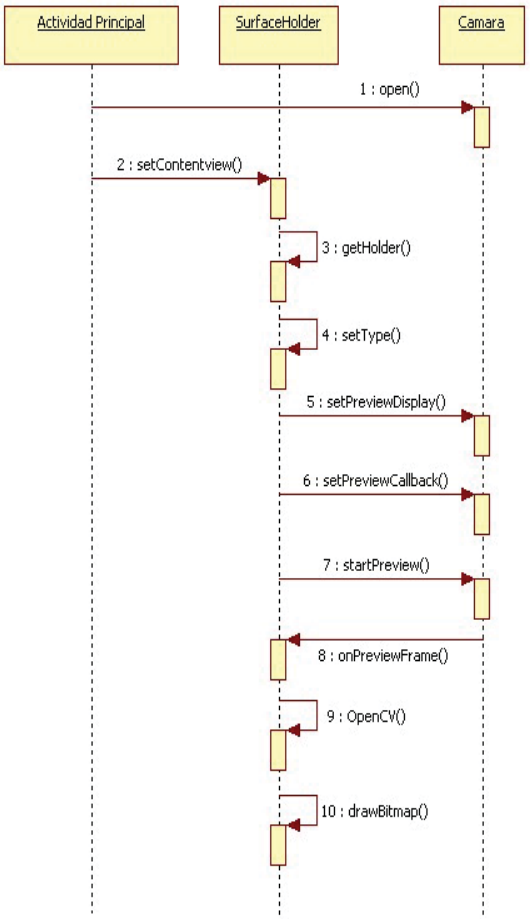

Fig 3. Secuencia de actividades de una aplicación OpenCV4Android

Lo ideal es que la aplicación no muestre por pantalla las imágenes recibidas de la cámara cuando estas no han sido procesadas aún, de forma que no se incurra en bajas de rendimiento por el costo computacional adicional que esto supone. Para varios dispositivos Android esto es perfectamente posible; sin embargo, existen dispositivos con los cuales no es posible utilizar OpenCV4Android si el *preview* de la cámara no es visible en pantalla. Este problema está documentado como el *bug* número 1244 en el *bugtracker* del proyecto OpenCV [26]. Dicho *bug* está registrado con prioridad normal, y se pretende resolverlo en la versión de OpenCV 3.0. Al momento de realizar esta investigación se reporta un 0% de avance en su resolución.

Al tratar de utilizar las muestras de OpenCV4Android en el teléfono Motorola Milestone (en este caso a través de la aplicación MINDdroidCV) se presentan una serie de errores que impiden el correcto funcionamiento de la aplicación. El problema más notable es que la pantalla del teléfono permanece negra mientras que la aplicación nunca procesa imágenes. En la herramienta *logcat* de Android se aprecia el siguiente mensaje al momento en que la aplicación inicializa la cámara *"Could not initialize Camera preview: App passed null surface"*. Acto seguido la aplicación continúa, aunque a partir de ese momento no recibirá imágenes de la cámara.

En primera instancia se pensó que el problema se trataba de una incompatibilidad entre OpenCV y CyanogenMod 7; para corroborar esta hipótesis se realizaron pruebas con diferentes firmwares; primero actualizando el teléfono a CyanogenMod 9, ensayo que falló de la misma manera. También se probó utilizando una tableta Galaxy Tab P1010, en primera instancia con el firmware de fábrica (Android 2.2 de Samsung), obteniendo un resultado positivo (funcionó), y luego con CyanogenMod 7 el experimento fue fallido, es decir, se obtuvo el mismo resultado que con el Motorola Milestone. Cabe resaltar que dicha tableta falló de la misma manera cuando se le aplicó la actualización de Samsung a Android 2.3.3. Finalmente se decidió retornar el dispositivo Milestone al sistema y firmware de fábrica provistos por Motorola, prueba que falló con exactamente el mismo resultado. Una vez realizados estos ensayos y luego de haber analizado cada uno de los escenarios posible, se descartó que el problema estuviera exclusivamente en CyanogenMod; concluyendo además que esta situación podría presentarse no como consecuencia del equipo o del sistema en sí, sino de las actualizaciones presentes o aplicadas al smartphone.

B. Desarrollo

Una vez realizada la fase de experimentación y analizados los resultados obtenidos, fue necesario implementar alguna solución que permitiese utilizar esta biblioteca en los sistemas mencionados, por lo menos de manera temporal mientras el *bug* es resuelto por el OpenCV Dev Team.

Lo primero que ha de resolverse sería el problema del *preview* visible. Esto resultó ser sencillo y básicamente la solución consistió en la creación de una superficie concreta que debe ser colocada dentro de algún contenedor en la interfaz de usuario; o sustituyendo la vista de contenido completa de la interfaz (la manera de implementar dicha solución se detalla claramente en la documentación para desarrolladores de Android [27]). Una vez solucionado el problema del *preview* es posible pasar cuadros directamente a OpenCV para que esta los procese. Sin embargo, no es posible visualizar el resultado obtenido por OpenCV de esta manera, ya que la especificación de Android establece que un *preview* de cámara no puede ser modificado, puesto que Android solamente entrega una copia de este a la aplicación [27].

La situación descrita anteriormente planteó la necesidad de manipular el modo de visualización presente en Android utilizando una clase adicional que tenga lo necesario para desplegar gráficos de OpenGL ES. Esta clase que se llamó *CameraGLRenderer* se define como una implementación de las interfaces *Renderer* y *PreviewCallback* como se aprecia en la Fig 4.

```
public class CameraGLRenderer implements Renderer,
PreviewCallback;
```

Fig 4. Declaración de la clase *CameraGLRenderer*

Definir la clase de esta manera permite registrarla dentro de *Surface Holder* para recibir los cuadros capturados por el *preview* de la cámara como se observa en la Fig 5.

```
private android.hardware.Camera mCamera;
protected CameraGLRenderer mGLRender;
…
public void surfaceChanged(…){
   …
   mCamera.setPreviewCallback(mGLRender);
   …
}
```

Fig 5. Implantación del método *PreviewCallback* de la cámara en la clase *SurfaceHolder*

CameraGLRenderer contiene una clase anidada llamada *FrameProcessor* que implementa la interfaz *Runnable*, la cual se encarga de procesar un cuadro cada 100 milisegundos. Este valor puede establecerse a voluntad; en esta investigación se estableció en 100 milisegundos para compensar la sobrecarga en tiempo que implica la solución planteada, aunado al retraso en que incurre el bloque controlador del robot cuando se hacen solicitudes de valores de sensores utilizando Bluetooth [16]. Específicamente en la ejecución de la solución se observó un retraso de al menos 30 milisegundos antes de comenzar la transmisión de datos; esto limita a la aplicación a un máximo efectivo de 10 cuadros por segundo. El valor mínimo de este número dependerá de la tasa de cuadros por segundo que puede capturar la cámara para el *preview*. Esta clase se implementa dentro de *CameraGLRenderer* como se observa en la Fig 6.

```
...
int[] glCameraFrame = null;
byte[] frame = null;
private Handler handler = new Handler();
...
private void processFrame(byte[] data) {
   synchronized (glCameraFrame) {
      /* Aqui va la llamada a OpenCV. */
      cv_call(w, h, data,
            glCameraFrame);
   }
}
private Runnable FrameProcessor = new Runnable(){
   public void run(){
      synchronized(frame){
         processFrame(frame);
      }
      if(handler != null)
         handler.postDelayed(this, 100);
   }
};
```

Fig 6. Implementación de la clase anidada *FrameProcessor*

Una vez solucionados los inconvenientes descritos anteriormente, solo resta el despliegue de la imagen resultante sobre un rectángulo del tamaño necesario utilizando OpenGL. En el caso de MINDdroidCV, se utilizó un rectángulo a pantalla completa. Para esto se creó una textura utilizando OpenGL ES, la cual contendrá la imagen procesada por OpenCV. Esta textura debe ser recreada cada vez que se procese un nuevo cuadro. Finalmente para que esta solución tenga éxito, es necesario colocar la superficie definida por CameraGLRenderer sobre la superficie de despliegue de la cámara, siguiendo el método descrito en la Fig 7.

```
/* Creamos la superficie para OpenGL. */
GLSurfaceView glView = new GLSurfaceView(this);
/* Configuramos sus parámetros. */
glView.setEGLConfigChooser(8, 8, 8, 8, 16, 0);
glView.getHolder().setFormat(
      PixelFormat.TRANSLUCENT);
/* Creamos el renderer y lo registramos. */
CameraGLRenderer gl_c = new CameraGLRenderer();
glView.setRenderer(renderer);
/* Creamos la superficie para la camara. */
CameraView cView = new CameraView(
            getApplicationContext(),
            renderer,
            this);
/* Sustituimos la interfaz de usuario con
 * la superficie de la cámara y colocamos la
 * superficie de OpenGL encima de esta. */
setContentView(cView);
addContentView(glView, new LayoutParams( ));
```

Fig 7. Inicialización de la superficie de despliegue

C. *Análisis y Resultados*

Las pruebas realizadas en este trabajo de investigación han demostrado que OpenCV puede ejecutarse en el Motorola Milestone y en CyanogenMod de forma funcional. Sin embargo, la solución planteada no es del todo perfecta, dado que no se corrigió de forma directa el *bug* encontrado en OpenCV. En su lugar se emplean mecanismos adicionales para realizar el despliegue de la imagen procesada en dos pasos, lo cual requiere la inclusión de OpenGL ES; siendo este procesamiento adicional el causante de una merma en el rendimiento de la aplicación, ya que la imagen debe pasar en primera instancia por el procesamiento de conversión inherente a la diferencia entre espacios de color asociados a OpenCV y OpenGL ES, y luego ser transferida a un segundo espacio de memoria.

El principal problema es de desempeño, el cual se ve considerablemente afectado por la sobrecarga en la cual se incurre al copiar cada cuadro después de haber sido modificado por OpenCV en una textura. Adicionalmente, el estar recreando la textura de OpenGL ES en cada cuadro implica que la textura del cuadro anterior debe ser destruida para no abarrotar la memoria, esto lo realiza Android automáticamente por medio del recolector de basura después de la llamada correspondiente de OpenGL para borrar la textura.

Con base al análisis anterior es directo inferir que la frecuencia de llamadas al recolector de basura aumenta de manera considerable y esto por supuesto tiene impacto sobre el rendimiento del sistema, el cual es visible en la cantidad de cuadros que pueden ser procesados por segundo por la aplicación. Adicionalmente, el *logcat* de la aplicación se inunda de mensajes del sistema indicando la ejecución del recolector de basura, lo que disminuye aún más el rendimiento.

Un segundo problema corresponde al tamaño del *preview*. En la solución planteada en esta investigación es necesario que el tamaño del *preview* sea fijo, capaz de ser embebido dentro

de una textura de OpenGL ES, y preferiblemente pequeño para disminuir la sobrecarga de la creación de texturas. El tamaño utilizado en esta investigación fue de 240 por 160 píxeles, atendiendo a las limitaciones del Milestone en cuanto a procesamiento y GPU (Graphical Processor Unit – Unidad de Procesamiento Gráfico).

Un preview pequeño es suficiente para el funcionamiento de MINDdroidCV, pero podría ser insuficiente para otras aplicaciones, especialmente las que necesiten de mayor precisión. Adicionalmente, pruebas realizadas en otros dispositivos arrojan un error al tratar de solicitar un *preview* tan pequeño (240 por 160 píxeles) para la cámara. Una posible solución a este inconveniente sería establecer el tamaño del *preview* al mínimo requerido por el dispositivo y crear las texturas de OpenGL ES de acuerdo a ese tamaño.

Una vez incorporada esta solución a la aplicación MINDdroidCV, se procedió a probarla sobre el dispositivo Milestone, obteniendo resultados satisfactorios, tal como se observa en la Fig 8. Esto permitió cumplir el objetivo principal de esta investigación el cual fue la incorporación de visión por computador a un robot Lego Mindstorm NXT; y con base a esta importante incorporación abrir el abanico de posibilidades en cuanto a proyectos e investigaciones posibles en este campo de la ciencia de la computación.

Fig 8. MINDdroidCV ejecutándose en el dispositivo Milestone

V. Conclusiones y Trabajos Futuros

En este trabajo de investigación se presentó una solución basada en la aplicación MINDdroidCV que permite utilizar OpenGL ES para desplegar imágenes procesadas por la biblioteca OpenCV4Android en la pantalla de dispositivos Android que son afectados por un *bug* conocido de esta biblioteca; acción que se realiza cubriendo completamente una superficie de despliegue que muestra la imagen capturada por la cámara sin modificaciones.

El objetivo principal de esta investigación se basó en la premisa de utilizar un dispositivo Motorola Milestone como un sistema de visión para los robots Mindstorms NXT, esto con la finalidad de ampliar el rango de funcionalidades del robot. Una vez resuelto el obstáculo que representa el *bug* 1244 de OpenCV4Android, se obtuvo un código funcional que cumple a cabalidad con el objetivo planteado. Los resultados obtenidos y el análisis de los mismos permiten asegurar que la solución presentada es funcional y representa un primer intento para poder utilizar OpenCV en dispositivos. La fortaleza de la solución planteada radica en el hecho de que solamente utiliza métodos compatibles con el API 8 del SDK de Android, lo que permite utilizar en dispositivos que ejecuten cualquier versión de Android desde la 2.2 en adelante. Sin embargo, algunas mejoras son posibles; por ejemplo, podría utilizarse el método *onPreviewTexture* definido en el SDK de Android el cual permite capturar las imágenes de la cámara directamente a una textura de OpenGL ES; esto evitaría el tener que desplegar las imágenes no procesadas a una superficie no visible y luego tener que copiar la imagen procesada a una nueva textura por cada captura. No obstante este método solo está disponible desde el API 11 del SDK de Android, es decir, Android 3.0; lo que evita que pueda ser utilizado en dispositivos que ejecuten versiones anteriores.

En cuanto al uso de MINDdroidCV como pilar central de la solución desarrollada en este trabajo de investigación, se plantea como trabajo futuro el separar el código de verificación de sensores del código de procesamiento de imágenes. Esto permitiría procesar imágenes a más de 10 cuadros por segundo. Adicionalmente sería interesante incorporar la propiedad de visualización al comportamiento del modelo de robot planteado; esto requerirá el desarrollo de nuevos algoritmos para brindar la posibilidad de implementar esquemas de visión, por ejemplo, reconocimiento de objetos y detección de posibles amenazas. Como nota final cabe resaltar que nuestra implementación de la solución propuesta está disponible en el repositorio indicado en [29], siguiendo las pautas establecidas por la licencia GPL 3.

Referencias

[1] S. Kumar Saha, “Introducción a la Robótica”, McGraw-Hill, 2012.

[2] M. Resnik, F. Martin, R. Sargent y B. Silverman, “Toys to Think With”, IBM Systems Journal vol 35 No 3-4, pp 443 – 452, 1996.

[3] Lego, “Lego Mindstorm”, http://mindstorms.lego.com, 2012.

[4] S. Papert, “Teaching Children Thinking”, World Conference on Computer Education, 1970.

[5] S. Richárd, “Controlling Lego Mindstorms NXT Using OpenCV on Android”, http://www.jataka.hu/rics/nxt_android_opencv/index.html, 2012.

[6] E, Menegatti y M. Moro, “Educational robotics from high-school to master of science”, Proceedings of SIMPAR 2010 Workshops, pp 639 – 648, 2010.

[7] A. Behrens, et al.: “MATLAB meets Lego Mindstorms – A Freshman Introduction Course Into Practical Engineering”, IEEE Transactions on Education Volume 53 Number 2, pp 306 – 317, 2010.

[8] J. Mottok y A. Gardeia, “The Regensburg Concept of P-Seminars”, IEEE Global Engineering Education Conference, pp 917 – 920, 2011.

[9] MathWorks, “MathWorks – MATLAB and and Simulink for Technical Computer”, http://www.mathworks.com, 2012.

[10] A. Pásztor, T. Kovács y Z. Istenes, “Swarm Intelligence Simulation with NXT Robots Using Piconet and Scatternet”, 5th International Symposium on Applied Computational Intelligence and Informatics, pp 199 – 204, 2009.

[11] S. Magnenat, P. Retornáz, B. Noris y F. Mondada, “Scripting the swarm: event-based control of microcontroller-based robots”, Workshop Proceedings of SIMPAR 2008, pp 525 – 538, 2008.

[12] A. Valera et al, “Embedded Implementation of Mobile Robots Control.”, 17th IFAC World Congress, 2008.

[13] M. Cassini, A, Garulli, A. Giannitrapani y A. Vicino, “A Matlab-based Remote Lab for Multi-Robot Experiments”, 2008.

[14] D. Sakamoto, J. Kato, M. Inami, T. Igarashi, “A Toolkit for Easy Development of Mobile Robot Applications with Visual Markers and a Ceiling Camera”, UIST'09, 2009.

[15] R. Filipe Guedes, L. Paulo Reis y A. Sousa, “Efficient Robotics using the Lego NXT Platform and .Net”, 2007.

[16] The Lego Group, “Lego Mindstorms NXT Bluetooth Development Kit”, version 1.00, 2006.

[17] S. Göbel, R. Jubeh, S. Raesch y A. Zündorf, “Using the Android Platform to control Robots”, 2009.

[18] T. Lee, “Real-Time Face Detection and Recognition on LEGO Mindstorms NXT Robot”, 2007.

[19] LeJOS, “LeJOS, Java for Lego Mindstorms”, http://lejos.sourceforge.net/, 2012.

[20] NBC, “NeXT Bytes Codes, Not eXactly C, and SuperPro C”, http://bricxcc.sourceforge.net/nbc, 2012.

[21] Microsoft, “Microsoft Robotics Developers Studio”, https://www.microsoft.com/robotics, 2012.

[22] Player, “Player Project”, http://playerstage.sourceforge.net, 2012.

[23] C. Tovar y P. Tosiani, “Diseño, construcción y programación de robots móviles para la captura de eventos físicos vía Bluetooth”, T.E.G. Escuela de Computación, Facultad de Ciencias, UCV, 2007.

[24] OpenCV, “OpenCV|OpenCV”, http://opencv.org, 2012.

[25] Lego, “Lego.com MINDSTORMS : News”, http://mindstorms.lego.com/en-us/News/ReadMore/Default.aspx?id=227417, 2012.

[26] OpenCV, “OpenCV – Bug #1244”, http://code.opencv.org/issues/1244, 2012.

[27] Android Open Source Project, “Camera | Android Developers”, http://developer.android.com/reference/android/hardware/Camera.html#setPreviewCallback%28android.hardware.Camera.PreviewCallback%29, 2012.

[28] Khronos Group, “*OpenGL ES - The Standard for Embedded Accelerated 3D Graphics*”, http://www.khronos.org/opengles/, 2013

[29] M. Astor, D. Perez y M. Villapol, “*Controlling LEGO Mindstorms NXT using OpenCV on Android - Workaround for the OpenCV 1244 bug.*”, https://github.com/sagge-miky/MINDdroidCV, 2012.