Universidad Autónoma Metropolitana

Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Proyecto Terminal de Ingeniería en Computación

Automatización de un brazo robótico para clasificar objetos por medio de redes neuronales y RTLinux

Proyecto que presenta:

Cirilo Alejandro Bravo Soriano

Para obtener el Título de:

Ingeniero en Computación

Asesor en el Proyecto:

M. en C. Oscar Alvarado Nava

México, D.F. **Diciembre de 2013**

Resumen

Los sistemas en tiempo real son importantes para la precisión de los movimientos que tendrá el robot; es por ello que este proyecto utiliza un sistema operativo en tiempo real para la manipulación de un brazo robótico por medio de redes neuronales que permiten la clasificación de objetos de acuerdo a su forma (cuadrados, triángulos, círculos y rectángulos) encontrados en una banda magnética.

Agradecimientos

Este es el momento para que pueda expresar todo mi agradecimiento a todas aquellas personas que de una u otra manera han participado en este proceso tan arduo que me ha tocado vivir durante estos casi 5 años de carrera, en los que he vivido diferentes experiencias, aprendizajes y descalabros, pero en donde he tenido la fortuna de conocer a personas que han aportado grandes enseñanzas y me han permitido cultivar verdaderas amistades.

Primero que nada quiero agradecer a quienes me dieron la vida, mis padres Gloria y Cirilo quienes siempre han estado ahí para mí, alentándome a nunca darme por vencido y ser mejor persona, pues con su ejemplo me han enseñado que a pesar de los obstáculos siempre podemos lograr lo que nos proponemos, gracias papás por su apoyo incondicional, espero poder haberles dado un motivo de alegría al verme concluir una meta más; de igual manera agradezco a mis hermanos Adriana y Sergio quienes han sido un aliciente para querer ser mejor cada día.

Por otro lado extiendo mi agradecimiento a mi ahora esposa Marbella que me ha entregado su amor, su compañía y un apoyo incondicional; a mí asesor Oscar Alvarado quien me brindó su apoyo, sus consejos y me dirigió de manera correcta para lograr concluir este proyecto y por último pero no menos importante a mis amigos quienes fueron de gran ayuda en la realización de mí proyecto.

Índice general

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

Las Redes Neuronales Artificiales *(RNA*[1]*)* nacieron como un intento de modelar un órgano muy complicado del cuerpo humano: el cerebro. Estas redes neuronales constituyen un campo de trabajo amplio y como herramienta de solución de problemas se utilizan en muchas aplicaciones, desde el control de robots, los sistemas de rastreo infrarrojo, el reconocimiento de patrones, etc. Entre sus muchas cualidades, tienen la habilidad de aprender, clasificar, almacenar, recordar, cruzar referencias, interpolar, adaptar parámetros y realizar mantenimiento de la red.

Conociendo las utilidades y aplicaciones que nos ofrecen las RNA, se desarrollará un sistema que utilice las RNA, que sea capaz de automatizar el movimiento de un brazo robótico *Scorbot-Er VPlus*[2] de acuerdo al reconocimiento visual de objetos; los procesos de reconocimiento y automatización serán ejecutados en un sistema de tiempo real. Se plantea que las RNA puedan aprender, reconocer y clasificar objetos por su forma, a través del reconocimiento de patrones y puedan diferenciar entre las distintas formas y eviten cometer errores. Es por ello, que se requiere que la imagen captada sea de buena calidad (3 megapíxeles), ya que digitalizada la imagen se constituirá por un conjunto de elementos llamados píxeles[3], en donde cada píxel ofrece cierta información sobre una región elemental de la imagen, apoyándome de software como es Netbeans[4] 7.01, OpenCV[5], RTLinux[6] y un simulador robótico RoboCell[7].

[1]Conjunto de bibliotecas de C y C++ de código libre, orientadas a visualización artificial.
[2]Es un robot creado y diseñado para fines didácticos con seis grados de libertad.
[3]es la menor unidad homogénea en color que forma parte de una imagen digital.
[4]Es un entorno de desarrollo integrado libre, su principal lenguaje de programación JAVA.
[5]Es una biblioteca libre de visión artificial originalmente desarrollada por Intel.
[6]Sistema operativo de tiempo real que ejecuta Linux como un hilo de ejecución de menos prioridad.
[7]Es un paquete de software que simula los robots SCORBOT.

El funcionamiento general del sistema es el siguiente: una cámara web (webcam[8]) capta y envía una imagen a un sistema de reconocimiento a través del bus USB[9], la cual es procesada para el reconocimiento de objetos por medio de las redes neuronales y una vez reconocido el objeto de acuerdo con su forma (cuadrados,triángulos, círculos y rectángulos) por parte del sistema de reconocimiento, el sistema de automatización enviará las palabras de control necesarias para el movimiento a los servomotores del brazo robótico y así se pueda colocar al objeto en su zona correspondiente.

Los procesos anteriores se llevan a cabo a través de 5 módulos, los cuales llevan una secuencia de tiempo para que se puedan ejecutar y son dependientes. Como se manejarán en tiempo real los módulos, es necesario crear un sistema operativo en tiempo real para su realización de acuerdo al funcionamiento que se desea obtener, véase Figura 1.1.

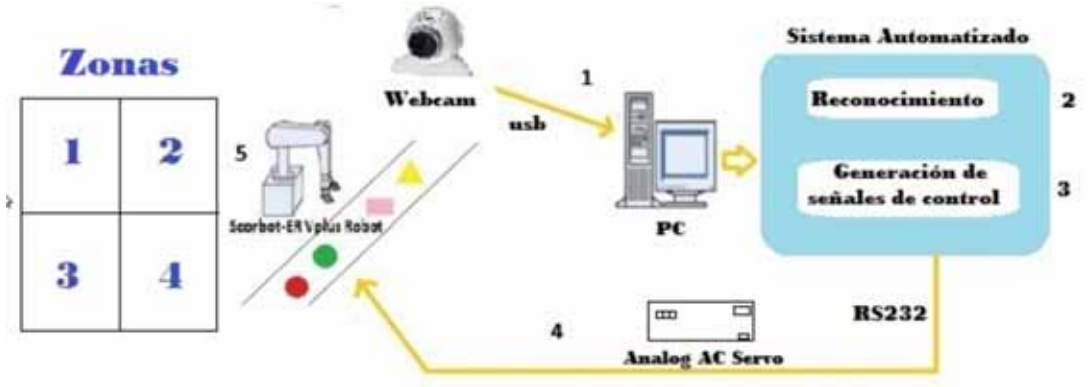

Figura 1.1: Clasificación de los 5 módulos.

[8]Es una pequea cámara digital conectada a una computadora la cual puede capturar imágenes.

[9]Sistema de comunicación entre dispositivos electrnicos-informáticos que sólo se transmite una unidad de información a la vez.

Capítulo 2

RTLinux

2.1. ¿Qué es RTLinux?

RTLinux es un sistema operativo de tiempo real que ejecuta Linux como un thread[1] de menos prioridad que las tareas de tiempo real, con este diseño estás tareas y los manejadores de interrupciones nunca se verán retrasados por operaciones que no sean de tiempo real.

Los RTLinux proporcionan la capacidad de ejecutar tareas de tiempo real y manejadores de interrupciones en la misma máquina que el Linux[2] estándar; siendo el peor caso de tiempo entre que se detecta la interrupción de hardware y el procesador ejecuta la primera instrucción del manejador de la interrupción.

RTLinux nació del trabajo de Michael Barabanov y Victor Yodaiken en New Mexico Tech, que posteriormente fundaron FSM Labs ofreciendo soporte técnico. En febrero de 2007, Wind River adquirió FSM Labs.

2.2. RTLinux su estructura y contenido

RTLinux está estructurado como un pequeño componente de la base y un conjunto de componentes opcionales; el componente central permite la instalación de controladores de interrupción de latencia muy bajos que no pueden demorarse por Linux, así como algunos de sincronización de bajo nivel y las rutinas de interrupción de control. Este componente del núcleo se ha extendido para soportar SMP[3] y al mismo tiempo se ha simplificado mediante la eliminación de parte de la funcionalidad que puede ser proporcionada fuera del núcleo.

[1] Hilo de ejecución con la secuencia más pequeña de las instrucciones programadas que se pueden administrar de forma independiente por un sistema operativo planificador.

[2] Es el núcleo libre del sistema operativo.

[3] Es un tipo de arquitectura de computadores en la que dos o más unidades de procesamiento comparten una única memoria central.

Un módulo de Linux no es más que un fichero objeto, el cual se crea mediante la compilación de un archivo de lenguaje C ordinario en el que la función **main()** se sustituye por un par de funciones:

int init_module ();
void cleanup_module ();

Como su nombre lo indica, la función *init_module()* es llamado cuando el módulo se carga por primera vez en el núcleo, debe devolver 0 en caso de éxito y un valor negativo en caso de fallo. Del mismo modo, la *cleanup_module* se llama cuando se descarga el módulo.

2.3. ¿Porqué usar un kernel de RTLinux?

La Figura 2.1 muestra el kernel de un Linux básico sin el apoyo en tiempo real, podemos ver que el kernel de Linux separa el hardware de tareas a nivel de usuario, el núcleo tiene la capacidad de suspender cualquier tarea a nivel de usuario, supongamos por ejemplo, que una tarea de usuario controla un brazo robótico, el kernel de Linux estándar potencialmente podría adelantarse a la tarea y dar a la CPU[4] una que es menos crítica en consecuencia, el brazo no cumplirá los requisitos de tiempo estrictos. Por lo tanto, al tratar de ser justo para todas las tareas, el kernel puede prevenir accidentes que se produzcan.

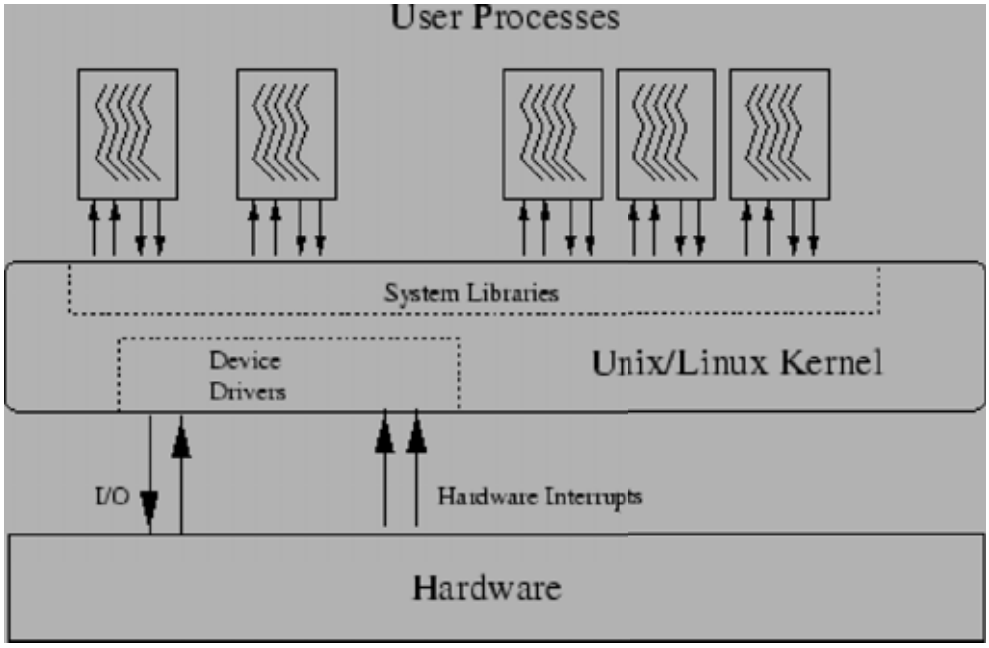

Figura 2.1: Kernel de un Linux básico.

[4]Unidad Central de Procesamiento, es el componente principal del ordenador.

La Figura 2.2 muestra un núcleo de Linux modificado para dar soporte en tiempo real, una capa adicional de abstracción denominada una máquina virtual en la literatura se a añadido entre el kernel estándar de Linux y el hardware del equipo. Por lo que el núcleo de Linux estándar se concentra, esta nueva capa parece ser hardware real, esta nueva capa presenta su propio programador de prioridad fija, este planificador asigna la prioridad más baja al kernel de Linux estándar, que a continuación, se ejecuta como una tarea independiente, entonces se le permite al usuario tanto introducir y establecer prioridades para cualquier número de tareas en tiempo real.

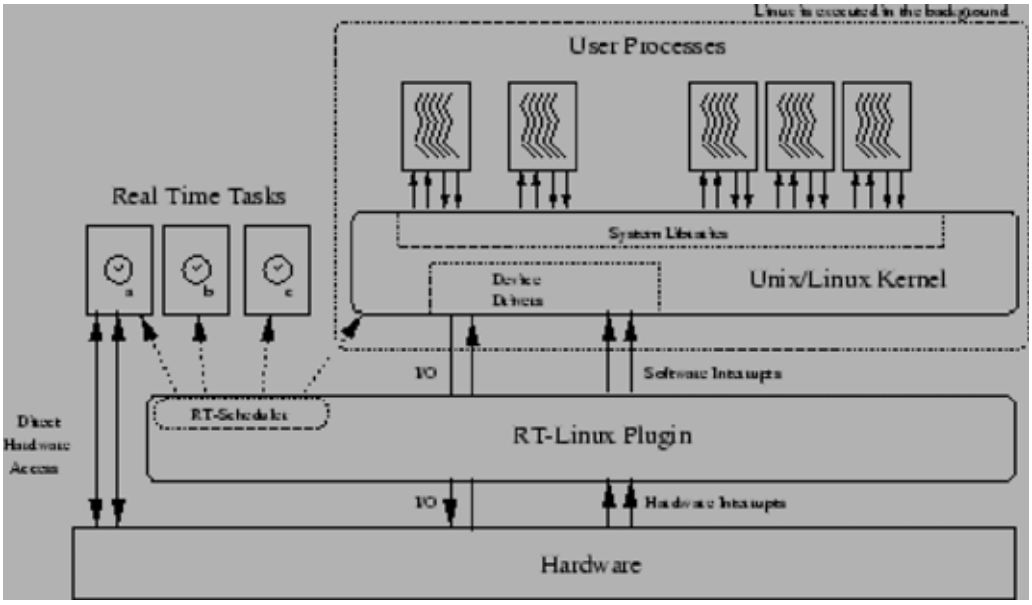

Figura 2.2: Kernel de un RTLinux.

2.4. Instalación de RTLinux

Para la instalación de RTLinux es necesario que antes tengamos disponible un kernel de Linux y un patch[5] (en RTLinux), que deben ser de la misma versión para evitar conflictos durante la instalación, en el caso del kernel lo podemos obtener desde: `http://kernel.org/pub/linux/kernel/v3.x/` y para el caso del patch podemos obtener una distribución desde: `http://kernel.org/pub/linux/kernel/projects/rt/3.2/`

[5]Parche con una utilidad presente en los sistemas basados en Linux.

Una vez que tenemos descargados los códigos fuentes procederemos a descomprimir con el siguiente comando en consola:

```
cd /home/alex/Downloads
tar xzf linux-3.2.51.tar.gz
```

Es necesario mover los archivos a **/usr/src/** con el siguiente comando en consola:

```
cp -r linux-3.2.51 /usr/src/
cp -r patch-3.2-51-rt72.patch.bz2 /usr/src/
```

Antes de aplicar el parche del kernel comprobamos si disponemos del compilador adecuado, para ello escribimos en consola:

```
gcc -v
```

Debemos comprobar que al menos tenemos la versión 2.7.2.3 o superior de gcc y la 1.1.2 o superior de egcs, si no es así debemos actualizar la versión de nuestro compilador. Ahora continuamos con el parcheado del kernel, que significa la instalación patch de RTLinux con nuestro kernel, para esto debemos ir al directorio actual de nuestro kernel y escribimos el siguiente comando en consola:

```
cd /usr/src/linux-3.2.51
bzcat ../patch-3.2.51-rt72.patch.bz2 | patch -p0 -p1
```

Ahora configuraremos el kernel, lo compilaremos y obtendremos la imagen de nuestro Linux en tiempo real, el compilado del kernel depende de la máquina que disponemos y puede ser un proceso largo.

Escribimos en consola lo siguiente:

```
cd /usr/src/linux-3.2.51
make menuconfig
```

Durante la instalación aparecerá una pantalla donde podremos configurar nuestro kernel con las características que requerimos, vea Figura 2.3.

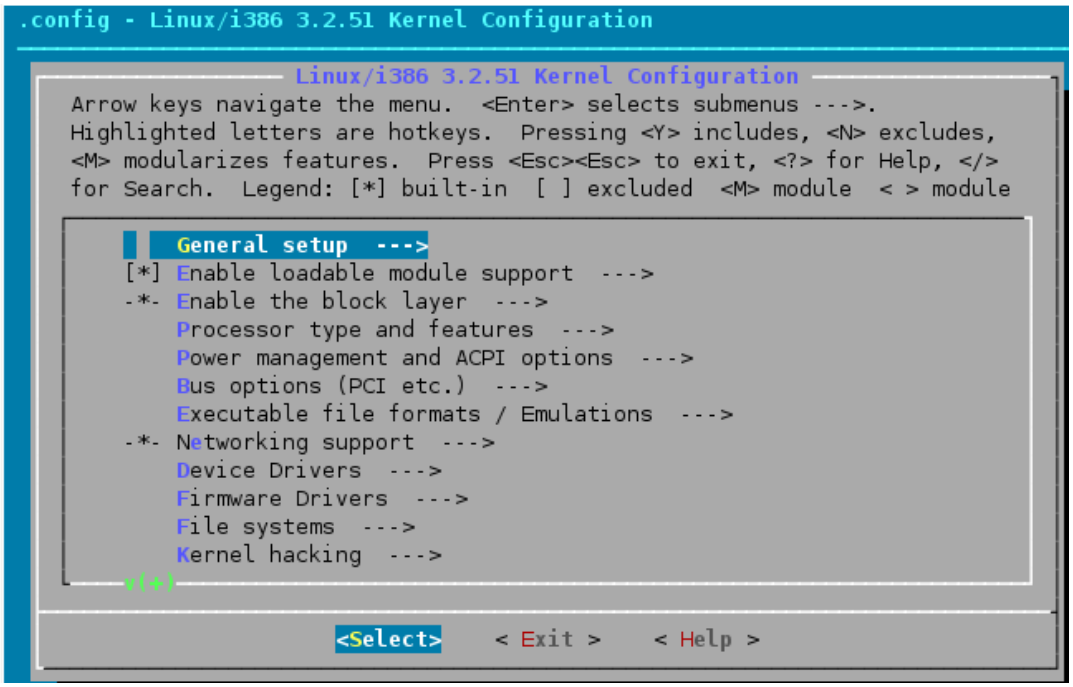

Figura 2.3: Pantalla de configuración de RTLinux.

Hay unas opciones importantes que se deben configurar adecuadamente para que el sistema RTLinux funcionen correctamente, estas son:

- Especificar correctamente el tipo de CPU que disponemos.
- Si solo disponemos de una CPU, desactivar el soporte para SMP.
- No activar el soporte de APM[6]. APM BIOS Call puede producir efectos impredecibles en el rendimiento de un sistema de tiempo real.
- Deshabilitar el soporte ACPI[7].
- En Preemption Model debemos seleccionar Fully Preemtible Kernel(RT) como se ve en la Figura 2.4.

[6]Es un API que permite que el BIOS administre la energía, tal como reducir la velocidad de la CPU, apagar el disco duro o apagar el monitor después de un período de inactividad para conservar corriente eléctrica.

[7]Es un estándar resultado de la actualizacin de APM a nivel de hardware, que controla el funcionamiento del BIOS y proporciona mecanismos avanzados para la gestión y ahorro de la energía.

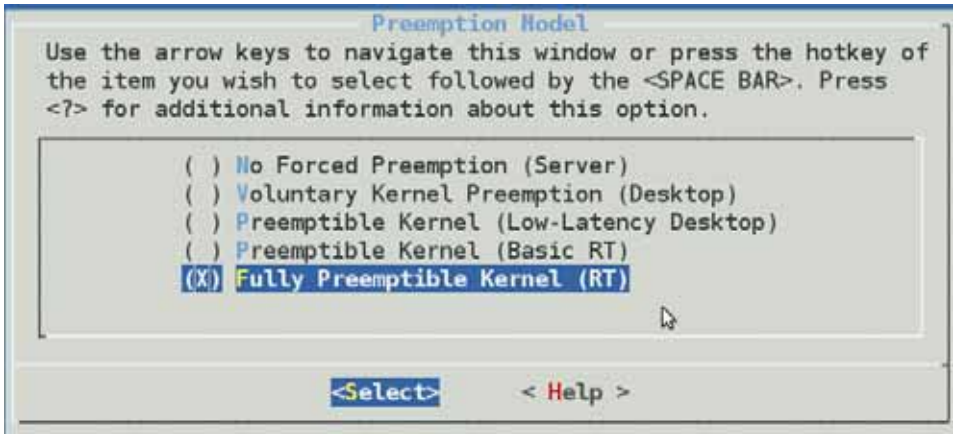

Figura 2.4: Instalación completa de RTLinux.

Después de haber configurado el kernel se guardarán los cambios en un archivo *.config* y salimos de la configuración, como se ve en la Figura 2.6.

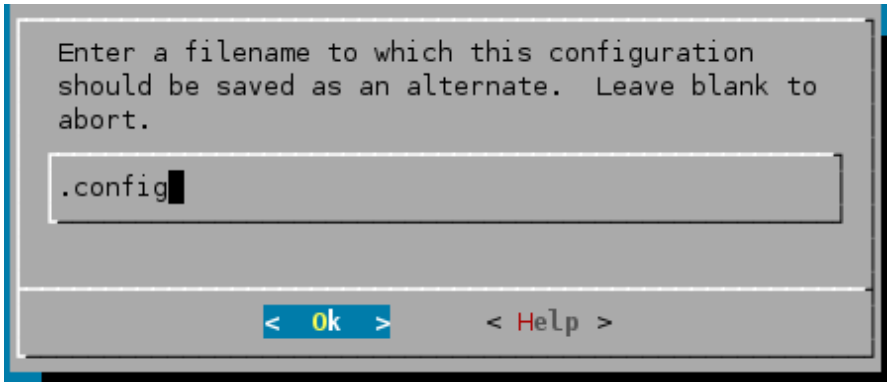

Figura 2.5: Guardando cambios de configuración.

Ahora compilamos nuestro kernel, el tiempo del compilado para este proyecto fue de 195 minutos con la siguiente instrucción en consola:

```
make
```

Después instalamos los códigos fuentes con la siguiente instrucción en consola:

```
make install
```

Ahora debemos instalar los módulos con la siguiente instrucción en consola:

```
make modules_install
```

Una vez compilado el kernel debemos agregarlo al grub del sistema, el cual se realizó con la siguiente instrucción en consola:

```
mkinitramfs -k -o /boot/initrd.img-3.2.51-rt72 3.2.51-rt72
```

Es necesario actualizar nuestro grub con la siguiente instrucción en consola:

```
update-grub
```

Si no hubo algún error durante el proceso de instalación se debe visualizar el nuevo kernel, para ello ejecutamos la siguiente instrucción en consola:

```
ls -l /boot/
```

Y podemos ver que se ha agregado el nuevo kernel a nuestro sistema operativo como se muestra en la Figura 2.6

```
root@debian:/# ls -l /boot/
total 25648
-rw-r--r-- 1 root root  111156 Sep 23 21:59 config-2.6.32-5-686
-rw-r--r-- 1 root root  131568 Oct 14 05:24 config-3.2.51-rt72
drwxr-xr-x 3 root root    4096 Nov  8 17:30 grub
-rw-r--r-- 1 root root 8700673 Oct 13 23:27 initrd.img-2.6.32-5-686
-rw-r--r-- 1 root root 9590262 Oct 14 05:27 initrd.img-3.2.51-rt72
-rw-r--r-- 1 root root 1296686 Sep 23 21:59 System.map-2.6.32-5-686
-rw-r--r-- 1 root root 1489675 Oct 14 05:24 System.map-3.2.51-rt72
-rw-r--r-- 1 root root 2303648 Sep 23 21:58 vmlinuz-2.6.32-5-686
-rw-r--r-- 1 root root 2558448 Oct 14 05:24 vmlinuz-3.2.51-rt72
root@debian:/#
```

Figura 2.6: Visualización del nuevo kernel.

Ahora solo debemos reiniciar nuestro equipo y a la hora de iniciar elegimos la opción del nuevo kernel y podemos trabajar con nuestro RTLinux.

Capítulo 3

OpenCV

3.1. ¿Qué es OpenCV?

Las siglas OpenCV provienen de los términos anglosajones Open Source Computer Vision Library, la cual definimos como una biblioteca de tratamiento de imágenes, destinada principalmente a aplicaciones de visión por computador en tiempo real.

Esta biblioteca libre de visión artificial originalmente fue desarrollada por Intel. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos; debido a que su publicación se da bajo licencia BSD[1], que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

3.2. OpenCV su estructura y contenido.

Cuenta con interfaces de C++, C, Phyton, Java y MATLAB y es compatible con Windows, Linux, Android y Mac OS, contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estéreo y visión robótica.

OpenCV es un conjunto de 10 librerías de las cuáles solo utilizaremos 4 en este proyecto: *CV, CXCORE, HIGHGUI* y *ML*, además de una librería adicional que es *CvBlobs*, la cual no viene con la versión 2.4 de OpenCV que utilizamos. A continuación se describen las librerias utilizadas en este proyecto.

[1]Es un sistema operativo derivado del sistema Unix.

- **CV**. Realiza el procesamiento de la imagen, es quien convierte la imagen de RGB[2] a imagen binarizada, además contiene una variedad amplia de posibles métodos que se pueden emplear para realizar esta operación.

- **CXCORE**. Permite utilizar texto en las imágenes y en general permite añadir distintas formas a las imágenes para un mayor entendimiento de la persona que utiliza la aplicación. Con esta librería se ha graficado un rectángulo alrededor de cada región calculada por la librería CvBlobs, también permite dibujar el centroide de cada región como un punto en la imagen binarizada y permite poner texto en imágenes.

- **HIGHGUI**. Permite realizar las capturas de imágenes y video, además de crear ventanas y mostrarlas en el proceso de ejecución de la aplicación e ir mostrando diferentes imágenes y de la misma forma permite eliminar las ventanas creadas cuando ya no sean necesarias; también permite cargar imágenes con las que ya se cuenta.

- **ML**. Es la librería conocida como Machine Learning, esta librería permite implementar una serie de soluciones tales como arboles de decisión, arboles aleatorios, redes neuronales, etc. Con esta librería se implementó una red perceptrón multicapa para poder hacer el entrenamiento y predicción de los objetos.

- **CvBlobs**. Esta librería permite reconocer las regiones en color negro de una imagen binarizada y extraerlas como objetos, además permite calcular el área, perímetro y compacidad de las regiones localizadas; que son datos que ingresarán a la red neuronal e incluso permite grabarlos en un archivo de texto para poder utilizar en otro momento, así mismo permite calcular el centroide de cada región.

3.3. ¿Por qué usar OpenCV?

OpenCV es una excelente opción debido a que se puede trabajar con punteros en esta librería y además está desarrollado en C++, lenguaje de programación en la que ya tengo experiencia, por lo tanto, se me hizó un poco más sencillo utilizar esta librería; se señala que OpenCV es una biblioteca de visión artificial por lo que cuenta con innumerables problemas que ya se han solucionado, facilitando y complementando así el trabajo para este proyecto transformándose de esta manera en una ventaja competitiva dentro de las herramientas principales para el desarrollo óptimo de una solución al problema propuesto.

[2]Es la composición del color en términos de la intensidad de los colores primarios de la luz.

Un aspecto importante es que al trabajar con OpenCV es código abierto no como otras aplicaciones como podría ser MatLab en la cual debemos pagar una licencia para su utilización.

3.4. Trabajar OpenCV en Netbeans

Usar Netbeans como plataforma de desarrollo de la aplicación permitió que se integrara de una manera fácil y sencilla OpenCV, esto a que ambos tienen aplicaciones en un lenguaje común como lo es C++.

El trabajar OpenCV en Netbeans ayudó al rápido desarrollo del proyecto, debido a que una vez que se tiene instalado OpenCV solo se debe agregar la librería al proyecto para que a la hora de ejecutar nuestra aplicación reconozca las librerías con las que trabajaremos, como se puede observar en la Figura 3.1.

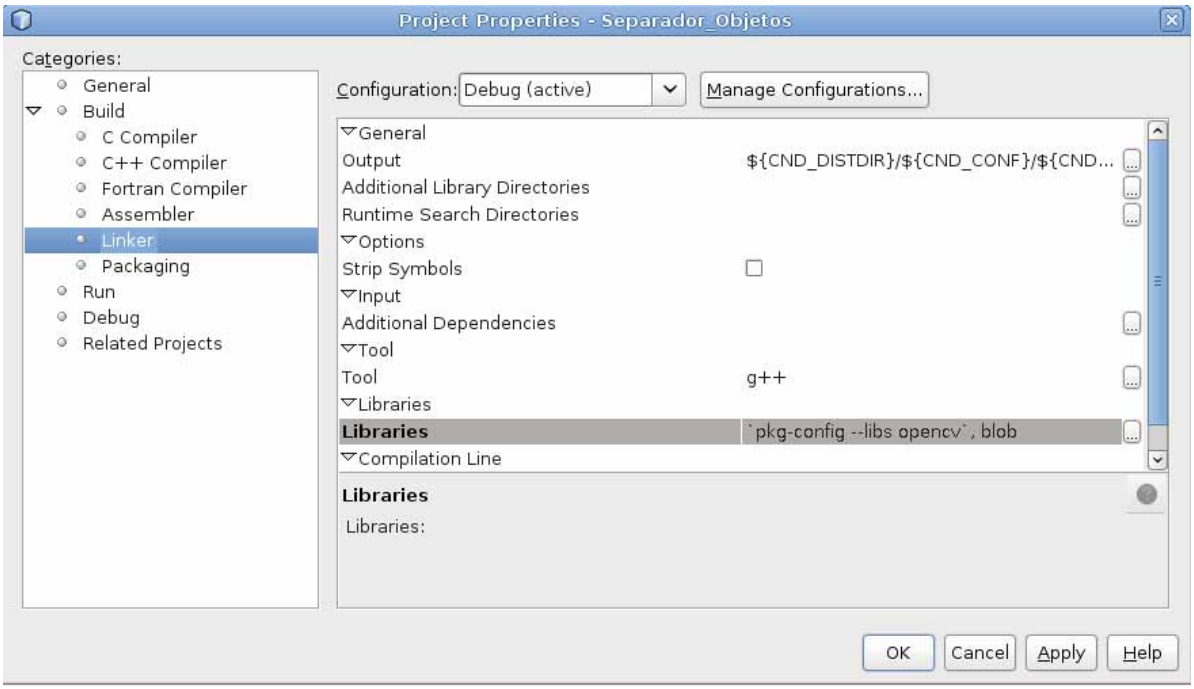

Figura 3.1: Agregando la librería de OpenCV al proyecto de Netbeans.

3.5. Instalación de OpenCV

Para el proceso de instalación de OpenCV es necesario tener algunas dependencias que se requieren para que funcione, para ello en consola escribimos las siguientes instrucciones que corresponde a la instalación de las mismas:

```
# apt-get install build-essential
# apt-get install libgtk2.0-dev
# apt-get install libavcodec-dev
# apt-get install libavformat-dev
#  apt-get install libjpeg62-dev
# apt-get install libtiff4-dev
# apt-get install cmake
# apt-get install libswscale-dev
# apt-get install libjasper-dev
```

Una vez que tenemos instaladas las dependencias, procedemos a descargar nuestro OpenCV, para este proyecto utilicé la versión 2.4.4 para Linux el cual lo podemos encontrar en: http://sourceforge.net/projects/opencvlibrary/

Una vez descargado descomprimimos el archivo y accedemos a la carpeta con el siguiente comando en consola:

```
cd /home/alex/downloads/opencv-2.4.4
```

Ahora construimos e instalamos el código fuente con la herramienta de construcción *Cmake*[3] de la siguiente forma:

```
# cmake.
# make
# make install
```

La Figura 3.2 muestra el proceso de instalación con el comando *make*[4] en consola, estas tres instrucciones duran aproximadamente 20 minutos.

[3]Es una herramienta multiplataforma de generación o automatización de código.

[4]Es una herramienta de generación o automatización de código, muy usada en los sistemas operativos tipo Linux.

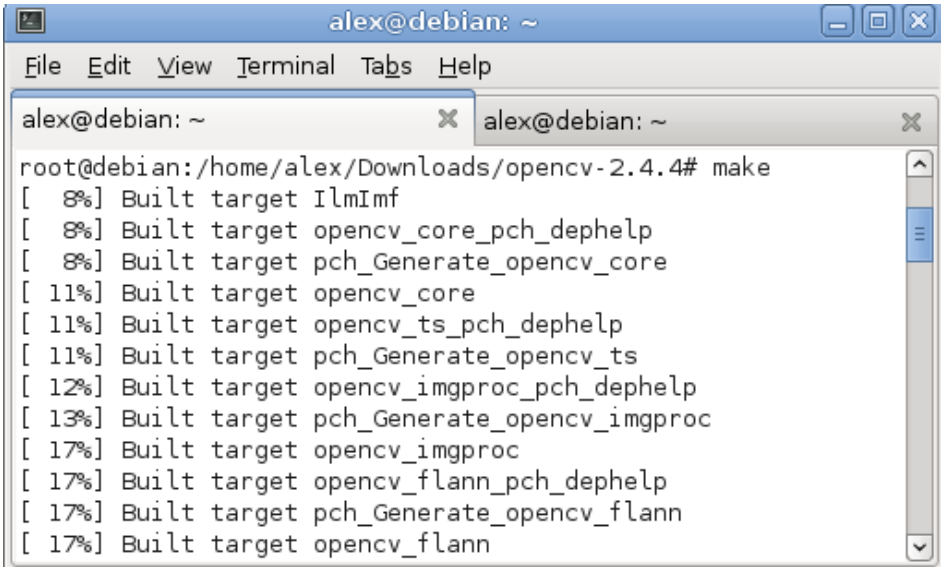

Figura 3.2: Ejecución de make en OpenCV.

Luego de instalar OpenCV-2.4.4 es necesario configurar las librerías a manera de variable de entorno indicando su ubicación para poder vincularlas en cualquier proyecto, abrimos el archivo de configuración **opencv.conf** con el editor de texto gedit desde la terminal con el siguiente comando:

```
# gedit /etc/ld.so.conf.d/opencv.conf
```

Una vez abierto el archivo de configuración escribimos lo siguiente:

```
/usr/local/lib/
```

Guardamos los cambios y cerramos el editor; ahora es necesario vincular las librerías recientes ejecutando el siguiente comando:

```
# ldconfig
```

Es necesario abrir el archivo **bash.bashrc** y debemos agregar las siguientes líneas al final del texto:

```
# gedit bash.bashrc
PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
export PKG_CONFIG_PATH
```

Finalmente guardamos los cambios y cerramos gedit, nuestra instalación de OpenCV se encuentra finalizada y lista.

3.6. Captura de imágenes

El sistema que permite la captura de la imagen, está formado por los siguientes elementos:

- **Cámara**. Es el dispositivo encargado de transformar las señales luminosas que aparecen en la escena en señales analógicas, capaces de ser transmitidas por un cable usb; se divide en dos partes, el sensor, que captura las propiedades del objeto en forma de señales luminosas y lo transforma en señales analógicas, y la óptica que se encarga de proyectar los elementos adecuados de la escena ajustando una distancia focal adecuada.

  La codificación de la brillantez de cada elemento de imagen o pixel obtenido de la digitalización espacial, se hace generalmente en 8 bits, mientras que la resolución de la discretización espacial de una imagen puede ser por ejemplo de 320*240 pixeles.

- **Sistema de iluminación**. La iluminación de la escena desempeña un papel importante en el desarrollo de un sistema visual, antes de intentar corregir un problema de iluminación por medio de algoritmos muy complicados, es mejor implantar un sistema de iluminación adecuado que intentar corregir ese problema por software, pues la velocidad de procesamiento será mayor con algoritmos más sencillos.

3.7. Binarización de imágenes

La binarización es un caso particular de la segmentación que consiste en transformar los píxeles de una imagen en 0 o 1 dependiendo de su nivel de grises, por esto es necesario que la imagen que se va a binarizar se encuentre a nivel de grises; pero la forma de binarizar más utilizada consiste en utilizar un umbral[5] que determina a partir de qué nivel de gris se va a considerar como primer plano, de tal forma que se preserven las propiedades esenciales de la imagen.

La Figura 3.3 muestra una binarización con un umbral de 217, donde se aprecia la presencia de ruido y distorsión en la imagen, no siendo apropiada para el desarrollo de este proyecto.

[5]Es la cantidad mínima de señal que ha de estar presente para ser registrada por un sistema.

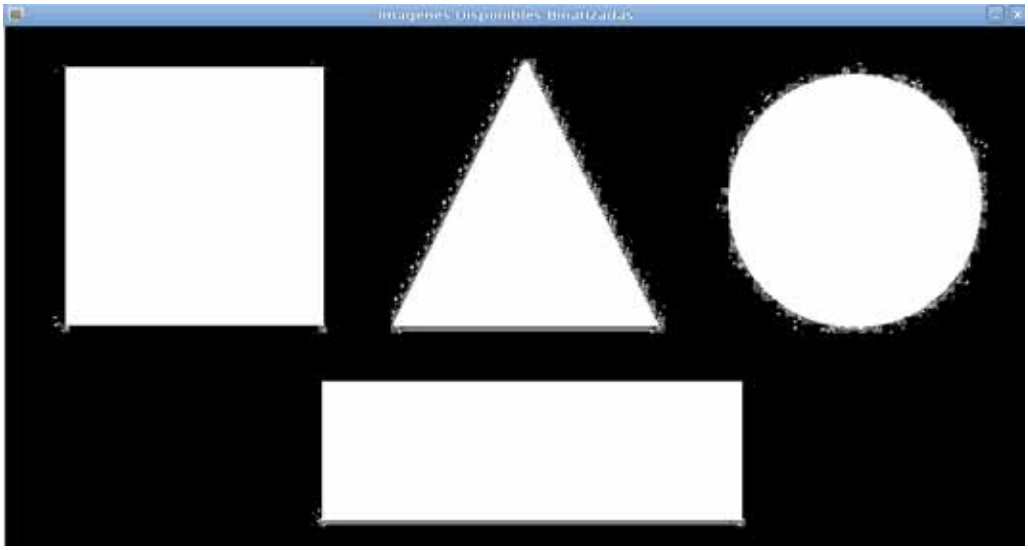

Figura 3.3: Imagen binarizada con un umbral de 217.

La Figura 3.4 nos muestra la misma imagen que se utilizó en la Figura 3.3 pero con un umbral de 115, en el cual podemos ver que no hay presencia de ruido o distorsión de la imagen, por lo que este valor de umbral es el apropiado para trabajar en el proyecto.

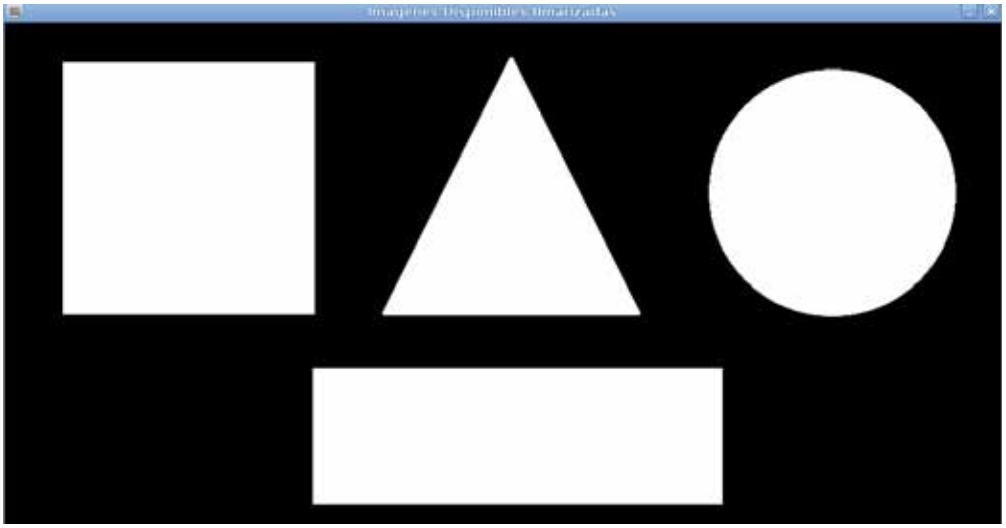

Figura 3.4: Imagen binarizada con un umbral de 115.

Capítulo 4

Red Neuronal como reconocedor de patrones

4.1. ¿Qué es una red neuronal?

Las redes neuronales son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales; se trata de un sistema de interconexión de neuronas que colaboran entre sí para producir un estímulo de salida. En inteligencia artificial es frecuente referirse a ellas como redes de neuronas o redes neuronales.

Una red neuronal se compone de unidades llamadas neuronas, en donde cada neurona recibe una serie de entradas a través de interconexiones y emite una salida dada por tres funciones:

1. **Una función de propagación.**

   Consiste en el sumatorio de cada entrada multiplicada por el peso de su interconexión. Si el peso es positivo la conexión se denomina excitatoria y si el peso es negativo se denomina inhibitoria.

2. **Una función de activación.**

   Modifica a la anterior, puede o no existir siendo en este caso la salida la misma función de propagación.

3. **Una función de transparencia.**

   Se aplica al valor devuelto por la función de activación. Se utiliza para acotar la salida de la neurona y generalmente viene dada por la interpretación que queramos darle a dichas salidas, algunas de las más utilizadas son la función sigmoidea (para obtener valores en el intervalo [0,1]) y la tangente hiperbólica (para obtener valores en el intervalo [-1,1]).

4.2. Red perceptrón multicapa

El perceptrón multicapa define una relación entre las variables de entrada y las variables de salida de la red, esta relación se obtiene propagando hacia adelante los valores de las n variables de entrada donde cada neurona de la red procesa la información recibida por sus entradas y produce una respuesta o activación que se propaga a través de las conexiones correspondientes hacia las neuronas de la siguiente capa.

La Figura 4.1 muestra una red neuronal artificial perceptrón multicapa con n neuronas de entrada y m neuronas en su capa oculta y una neurona de salida.

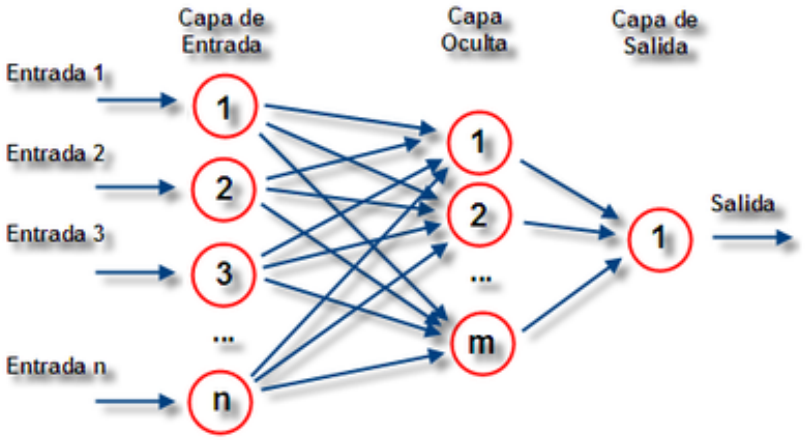

Figura 4.1: Estructura de una red neuronal perceptrón.

La red neuronal que emplearemos para el proyecto es de tipo perceptrón multicapa, con las siguientes características:

- **Una capa de entrada con 3 neuronas.**

  Para la cual estamos definiendo área, perímetro y compacidad que utilizaremos para poder clasificar nuestros objetos.

- **Dos capas ocultas con 13 neuronas.**

  Definimos 2 capas ocultas con 13 neuronas cada una para incluir todas las posibilidades a la hora de clasificar los objetos.

- **Una capa de salida con 4 neuronas.**

  Definimos 4 neuronas de salidas debido a que buscamos que nuestra red neuronal pueda clasificar 4 objetos: cuadrado, triángulo, círculo y rectángulo.

La función de transferencia que se usó fue la Sigmoid[1], la cual calcula la salida para las capas desde la red ingresada. La Figura 4.2 muestra la estructura de la red neuronal utilizada.

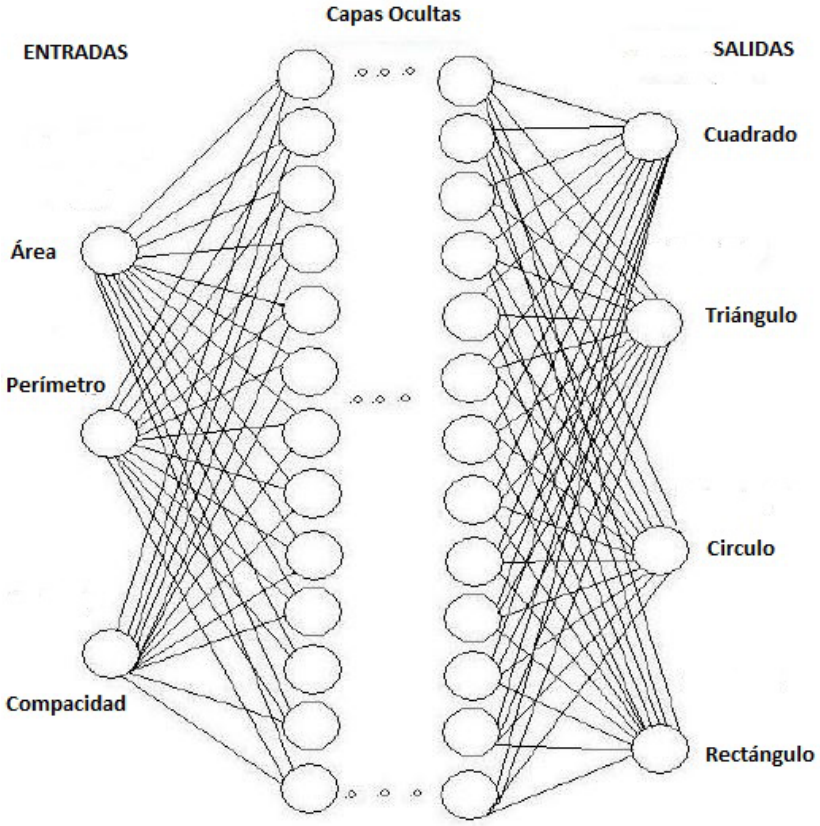

Figura 4.2: Estructura de la red neuronal utilizada.

4.3. Métodos de descripción (área, perímetro y compacidad)

- **Área**. En cualquier imagen digital, el área de un objeto esta definida por el número de píxeles que la representan, por lo tanto el cálculo del área se realiza contando el número de píxeles que lo integran. El cálculo del área del objeto se realiza mediante la función **calculos()**, en la cual se tiene definida una función llamada: **blob.area** que es la encargada de realizar los cálculos y determinar el área del objeto, esta función es extraída de *cvBlobResult*[2].

[1]Es una función matemática que tiene una forma de "S".

[2]Es una biblioteca para la visión artificial para detectar regiones conectadas en imágenes digitales binarios.

- **Perímetro**. El perímetro es un parámetro geométrico que al igual que el área, puede ser calculado a partir del código de cadena, para realizar este cálculo es necesario contar la longitud del código y tomar en consideración que los pasos en direcciones diagonales deben ser multiplicados por un factor igual a raíz cuadrada de dos. La función **blob.perimetro** realiza los cálculos y nos proporciona el perímetro del objeto, esta función es extraída de *cvBlobResult.*

- **Compacidad**. La compacidad es una propiedad topológica de un espacio topológico, que define un contorno alrededor de cada punto, en el cual el espacio tiene propiedades similares a las de un espacio compacto. La función:
pow(blob.perimeter,2)/(4*CV PI*blob.area) permite calcular la compacidad del objeto y es extraída de *cvBlobResult.*

4.4. Entrenamiento de la red

Para el entrenamiento de la red perceptrón multicapa que utilicé como arquitectura, lo primero que realicé fue obtener una imagen patrón con las imágenes a clasificar (cuadrados, triángulos, círculos y rectángulos) y posteriormente se binariza la imagen como se ve en la Figura 4.3 y Figura 4.4.

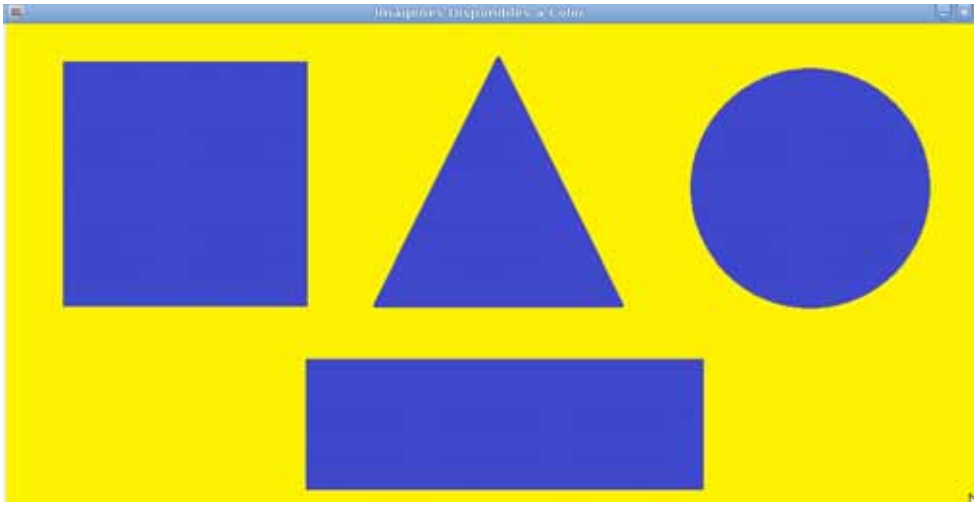

Figura 4.3: Imagen patron a color.

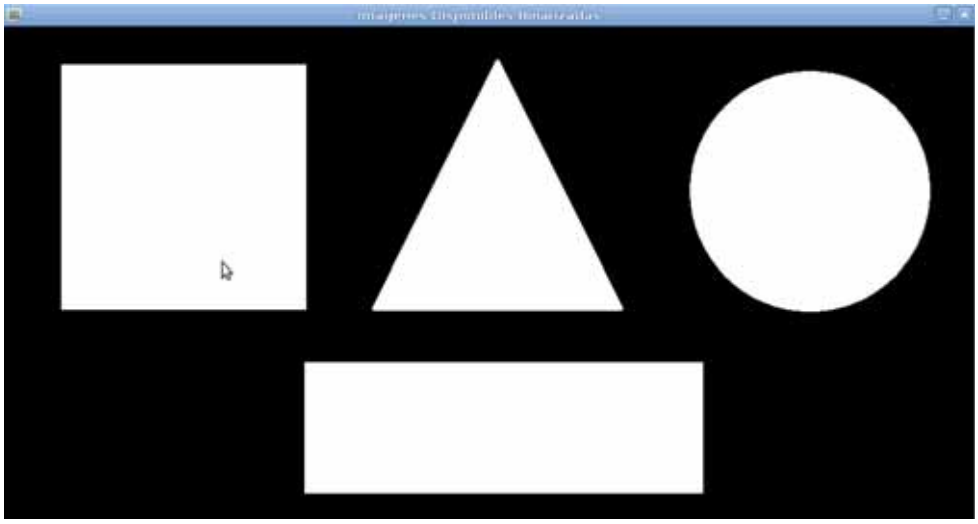

Figura 4.4: Imagen patron binarizada.

Luego de la binarización la Figura 4.5 muestra como la aplicación reconoce cada objeto a identificar y los enumera, durante este proceso se calcula el área, perímetro y compacidad de cada objeto que son almacenados en un vector fila llamada *in* por ser la entrada de datos a la red.

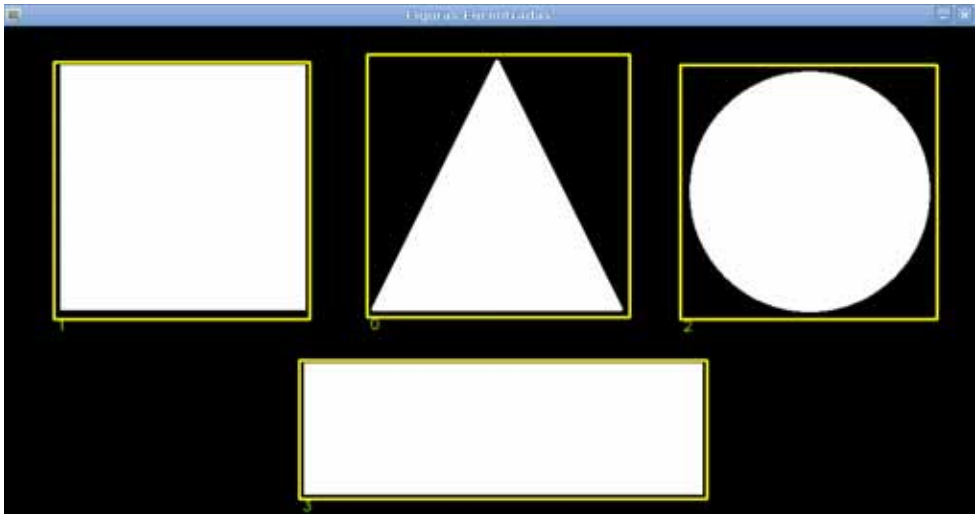

Figura 4.5: Figuras Encontradas.

Una vez que tenemos las figuras enumeradas es necesario crear una matriz llamada out con la salida deseada:

```
1          //Q=cuadrado, T=triangulo,C=Circulo, R=Rectangulo
2  float out[]={    0,1,0,0,    //0
3                   1,0,0,0,    //1
4                   0,0,1,0,    //2
5                   0,0,0,1};   //3
```

En el código anterior la línea 2 describe un arreglo utilizado para definir las figuras disponibles a clasificar, donde *0,1,0,0* representa que se tata de una figura de clase triángulo, debido a que en la Figura 4.1 nos muestra que el número 0 le corresponde, lo mismo se hace para las líneas 3,4 y 5.

Ahora para poder entrenar a la red tenemos:

```
1. Los datos de entrada a la red estan en input
CvMat *input= cvCreateMat (num,3,CV_32FC1);
cvInitMatHeader (input,num,3,CV_32FC1,in);

2. La salida de la red se encuentra en target
CvMat *target= cvCreateMat (num,4,CV_32FC1);
cvInitMatHeader (target,num,4,CV_32FC1,out);

3. Establecimiento de la estructura de la red
int layer [ ]= {3 ,13 ,13 ,4};      //capas
CvMat *layersize= cvCreateMat (1,4,CV_32SC1);
cvInitMatHeader (layersize,1,4,CV_32SC1,layer);
CvANN_MLP net ( layersize, CvANN_MLP::SIGMOID_SYM,1,1);

4. Entrenamiento de la red
int iter= net.train (input,target,NULL,0, CvANN_MLP_TrainParams
(cvTermCriteria(CV_TERMCRIT_ITER | CV_TERMCRIT_EPS,5000,0.001),
CvANN_MLP_TrainParams::BACKPROP, 0.1,0.1));
```

Donde:

- num: es el número de objetos encontrados.
- input: data de entrada.
- target: la salida deseada.
- NULL: es un parámetro que en este caso toma el valor de NULL porque se define solo cuando la red es del tipo RPROP[3].
- 0: indica las filas a tomar, cero significa considerar todos los datos.

[3]Es un aprendizaje heurístico para aprendizaje supervisado en redes neuronales artificiales.

Las siguientes funciones forman parte del entrenamiento de CvANN_MLP_TrainParams:

```
cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,5000,0.001)
Define los criterios para la terminaci'on del algoritmo, donde
   definimos dos criterios:
1. Definimos el numero de iteraciones (CV_TERMCRIT_ITER=500)
2. El error objetivo con (CV_TERMCRIT_EPS = 0.001).

CvANN_ MLP_TrainParams::BACKPROP
Define el algoritmo de entrenamiento, backpropagation en mi
   caso.

{0.1,0.1}
El primer parametro es la velocidad o tasa de aprendizaje y el
   segundo define el momento, el cual multiplica la diferencia
   de pesos de la iteracion anterior y la suma al calculo de la
   variaci'on.
```

4.5. Prueba de la red neuronal

Para realizar el test se realizan los mismos pasos descritos anteriormente, primero tomamos una foto con la webcam como se muestra en la Figura 4.6.

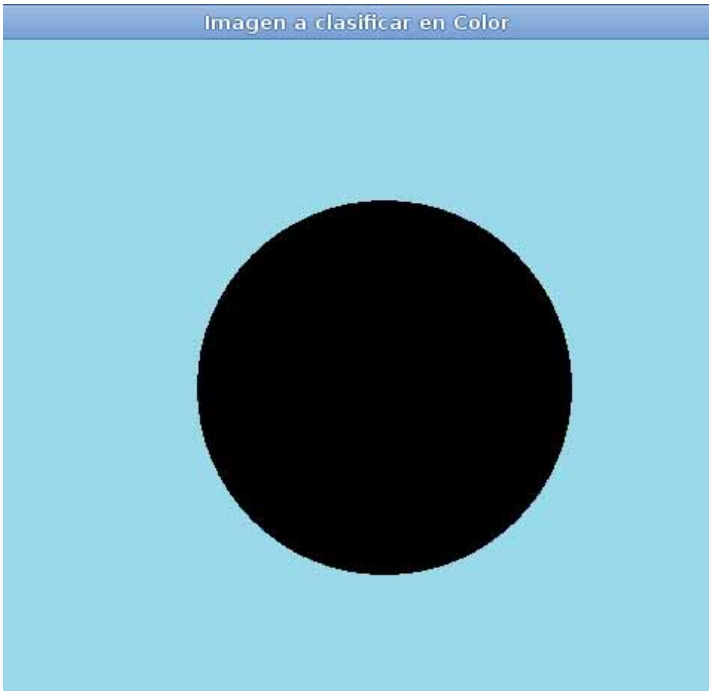

Figura 4.6: Foto tomada con la webcam.

Una vez que se tiene la imagen a clasificar con la webcam se guarda con el nombre de *clasifica.jpg* luego será binarizada como lo muestra la Figura 4.7.

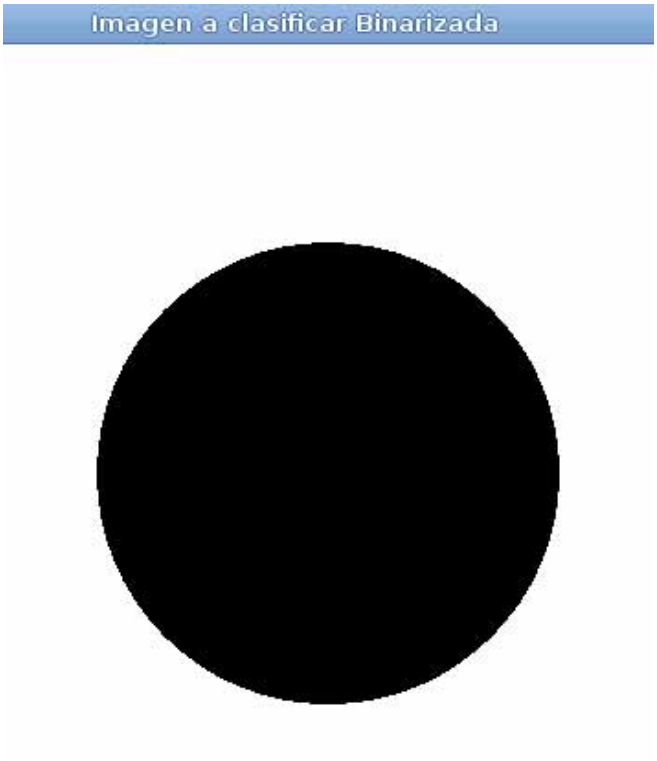

Figura 4.7: Imagen a clasificar binarizada.

Después de terminar el proceso de binarización se llama a la función **calculos()**, la cual será la encargada de obtener el área, perímetro y compacidad de la imagen, cuando el proceso termine mostrará una ventana enumerando el objeto encontrado como se ve en la Figura 4.8.

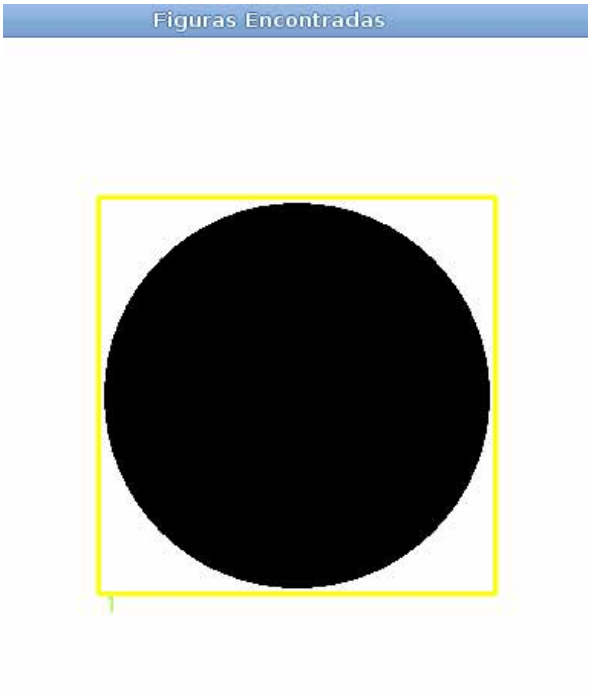

Figura 4.8: Imagen a clasificar encontrada.

Una vez que se conoce el área, perímetro y compacidad de la imagen a clasificar se llama a la función **entrenamiento()** donde se lleva el procedimiento de clasificar al objeto con uno de los que tenemos en el patrón, es decir que tengan o se aproximen a la misma área, perímetro y compacidad, como lo muestra la Figura 4.9 dónde la imagen ya es clasificada.

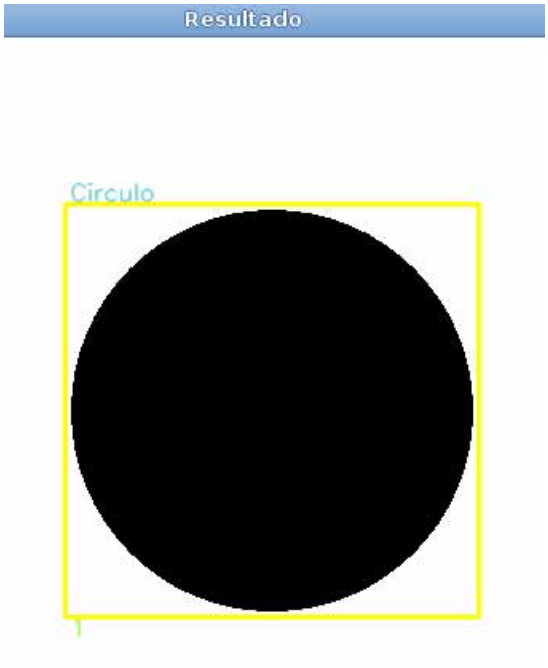

Figura 4.9: Resultado de la imagen a clasificar.

La clasificación fue la correcta como se puede observar en la Figura 4.9, debido a que nuestro patrón de imágenes a clasificar se encuentra bien definido, es decir no existe ruido ni distorsiones en las imágenes, por ello se obtiene un resultado favorable.

Si se colocará una imagen ajena a las del patrón la podría clasificar y le asignaría un resultado aproximado a las del patrón por medio del área, perímetro y compacidad.

Capítulo 5

Simulador RoboCell SCORBOT-ER-9Pro

5.1. ¿Qué es RoboCell SCORBOT-ER-9Pro?

Robocell es un software que integra el control robótico *SCORBASE*[1] el cual es un software de simulación de modelado de sólidos en 3D, robots y dispositivos virtuales de Robocell, replican con exactitud las dimensiones reales y las funciones de los equipos *SCORBOT*[2] permitiendo enseñar posiciones, escribir programas y depurar aplicaciones robóticas sin conexión antes de ejecutarlas en una celda de trabajo real, además de que permite a los usuarios experimentar con una variedad de células de trabajo simuladas, incluso los usuarios avanzados pueden diseñar objetos 3D e importarlos en Robocell para su uso en células de trabajo virtuales.

5.2. ¿Por qué usar RoboCell SCORBOT-ER-9Pro?

El uso de RoboCell es un software de *Intelitek*[3] que permite familiarizarse y hacer pruebas con los brazos robóticos en sus diferentes modos de operación y programación antes de utilizar el **SCORBOT-ER Vplus.** En lo particular me fue de gran ayuda, pues obtuve muchos beneficios que facilitaron el desarrollo de mi aplicación permitiendome:

[1]Software diseñado para brazos robóticos.
[2]Es un robot versátil y fiable para uso educativo.
[3]Es una empresa privada de educación tecnológica enfocada en robótica y visión artificial.

- Simular la programación empleada.
- Grabado de posiciones.
- Obtención de coordenadas cartesianas de los objetos.
- Manipulación de los movimientos con el teach pendant[4] o por medio del teclado.
- Manipulación de los objetos a clasificar.

Además de que es un software que tiene una licencia gratuita de 14 días y si se considera necesario se puede llenar un formato para obtener la licencia por 1 año para fines académicos.

5.3. Características de RoboCell SCORBOT-ER-9Pro

El SCORBOT-ER-9Pro es un robot vertical articulado con cinco juntas rotativas, con el agregado de la mordaza y el riel, además de que posee 7 grados de libertad, los movimientos de las juntas se describen en la Tabla 5.1.

Número de eje	Nombre de la junta	Movimiento producido
1	Base	Rota la base
2	Hombro	Sube y baja el brazo superior
3	Codo	Sube y baja el antebrazo
4	Elevación	Sube y baja la pinza
5	Giro	Gira la pinza
6	Pinzas	Abre o cierra la pinza
7	Riel	Mueve el robot sobre el riel

Tabla 5.1: Movimiento de las juntas del SCORBOT-ER-9Pro.

[4]Es un dispositivo que se puede utilizar para controlar un robot de forma remota.

5.3.1. Modos de operacion del Scorbot-ER-9PRO

Modo Manual

Este modo manual es accionable cuando el sistema se encuentra en modo directo, permitiendo el control directo de los ejes sin la necesidad de usar el *TEACH PENDANT*, para activarlo sólo basta presionar las teclas: **<ALT >+ m.** La Tabla 5.2 muestra los movimientos de los ejes.

Teclas	Acción
Q - 1	Axes 1
W - 2	Axes 2
E - 3	Axes 3
R - 4	Axes 4
T - 5	Axes 5
Y - 6	Axes 6
U - 7	Axes 7

Tabla 5.2: Movimientos de los ejes.

Modo Directo

Este modo directo es accionable cuando el usuario tiene el control directo de los ejes del robot, es decir, el controlador va ejecutando los comandos a medida que van siendo ingresados por el operador, cuando se está en este modo el *prompt*[5] aparece en pantalla de la siguiente manera: >_

5.4. Instalación de RoboCell SCORBOT-ER-9Pro

Este software lamentablemente solo se encuentra disponible para Windows Xp y Windows 7 de 32 bits y se puede obtener desde la siguiente dirección electrónica: `http://www.intelitekdownloads.com/Software/WIN/Robotics/ER9%20PRO/ER-9%20PRO6.1%20build%2018.zip`, una vez que tenemos el archivo, los descomprimimos y buscamos la carpeta Install y damos doble clic en Setup.exe. El ejecutable nos desplagará los términos de la licencia, damos clic en *I Accept* y procedemos con la instalación.

[5]Carácter o conjunto de caracteres que se muestran en una línea de comandos para indicar que está a la espera de órdenes.

La Figura 5.1 nos muestra los diferentes brazos robóticos con que cuenta la aplicación y que deseamos instalar, en mi caso elegí SCORBOT-ER-9Pro porque es muy similar al SCORBOT-ER-Vplus.

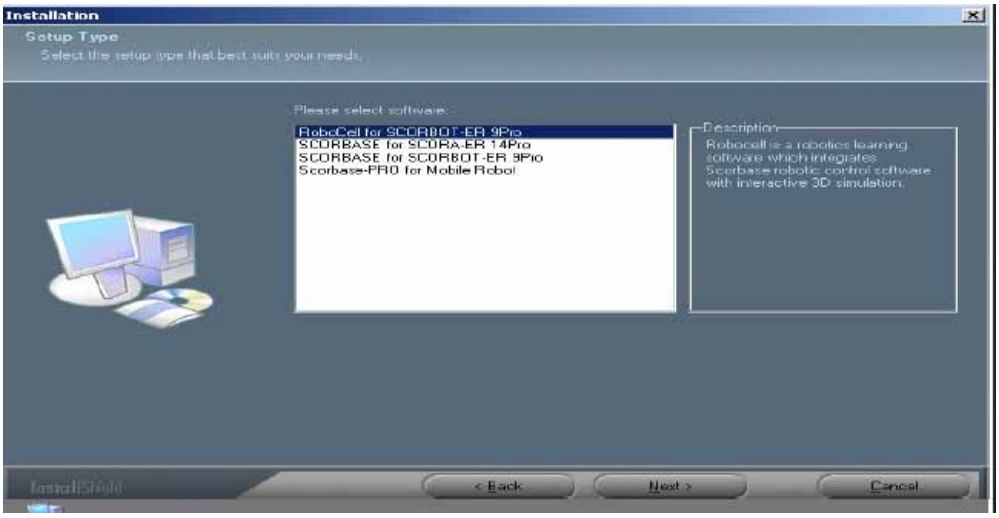

Figura 5.1: Selección del software a instalar.

Una vez seleccionado el software damos clic en *Next* y nos pedirá que demos la ruta donde deseamos que se encuentre instalado el software, una vez elegida la ruta damos clic nuevamente en *Next* y el proceso de instalación comenzará, el cual durará aproximadamente 10 minutos y posterior a este tiempo se desplegará un mensaje que señala que la instalació a finalizado tal como se muestra en la Figura 5.2.

Figura 5.2: Instalación completa del SCORBOT-ER-9Pro.

Damos clic en reiniciar equipo y al iniciar veremos que ya tenemos el software instalado para poder trabajar con RoboCell SCORBOT-ER-9pro, al ingresar al programa RoboCell por primera vez nos pedirá que registremos el producto para obtener una licencia y si no lo consideramos necesario se puede utilizar la versión de 14 días gratuita como se ve en la Figura 5.3.

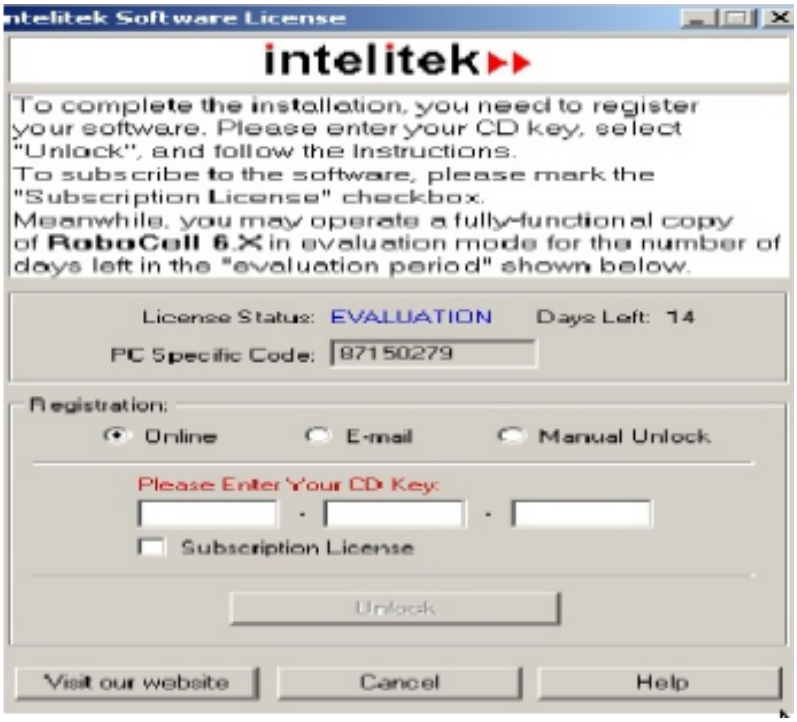

Figura 5.3: Registrar licencia del software SCORBOT-ER-9Pro.

Ya registrado el software, el programa nos mostrará la pantalla principal de RoboCell en la cual podremos crear proyectos y generar simulaciones con movimientos del brazo robótico.

5.5. Programación ACL del sistema robótico

En este apartado desarrollé las pruebas del SCORBOT-ER-9pro con la ayuda de los comandos que contiene el brazo robótico, primero elaboré un proyecto sobre el cual iba a trabajar llamado *prueba_circulo* para mayor entendimiento de la programación, el software proporciona una simulación en 3D. Lo primero que aprendí fue a conocer los movimientos del brazo en modo manual, es decir por medio del teclado y la manera de obtener las coordenas de los objetos de interés y a grabar posiciones.

Una vez que se desarrollen estas actividades se esperará que la simulación del set, de instrucciones para ver la simulación del brazo robótico, la Figura 5.4 nos muestra la pantalla principal una vez que se ha creado el proyecto y un script[6] en 3D para la simulación del brazo robótico.

[6]Es un programa usualmente simple por lo regular se almacena en un archivo de texto plano.

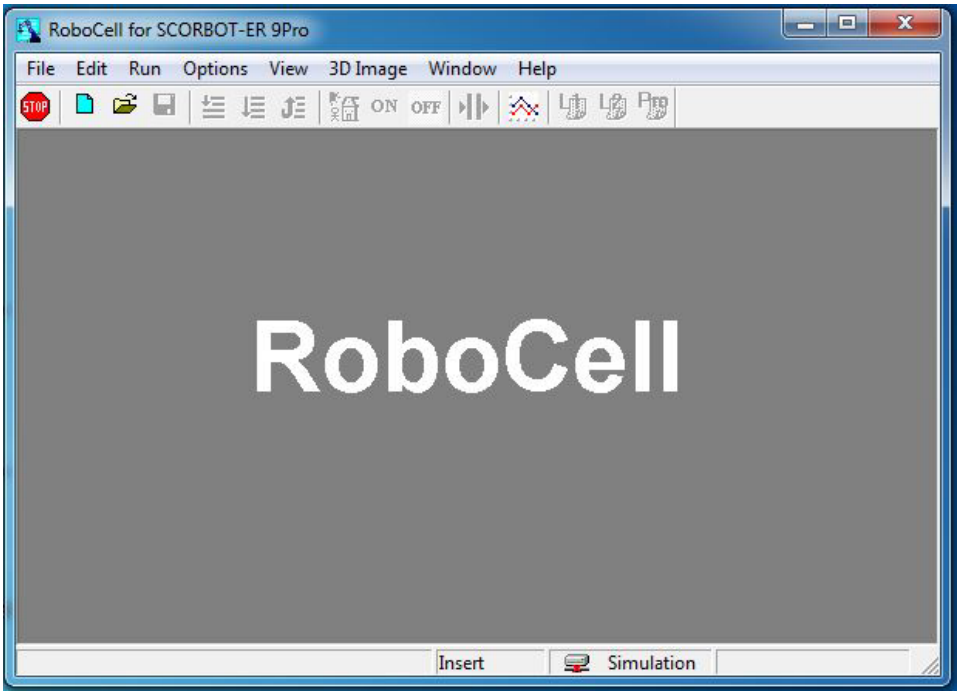

Figura 5.4: Pantalla principal de RoboCell.

Es necesario crear un proyecto, esto lo hacemos en el menú *File\New_Project*, elegimos un nombre para el proyecto y damos clic en *OK* para crear el proyecto, esto se ve reflejado en la Figura 5.5 la cual nos muestra todo lo que esta disponible para la utilización de nuestro proyecto, en ella se puede observar:

- *3D Image*
  Sirve para simular los movimientos que se envían al brazo robótico.
- *Program*
  Es el editor de programa permite crear el set de instrucciones que serán enviados al brazo.
- *Manual Movement*
  Permite mover todos los ejes del brazo robótico manualmente por medio del teclado.
- *Teach positions*
  Permite obtener posiciones y el grabado de ella.

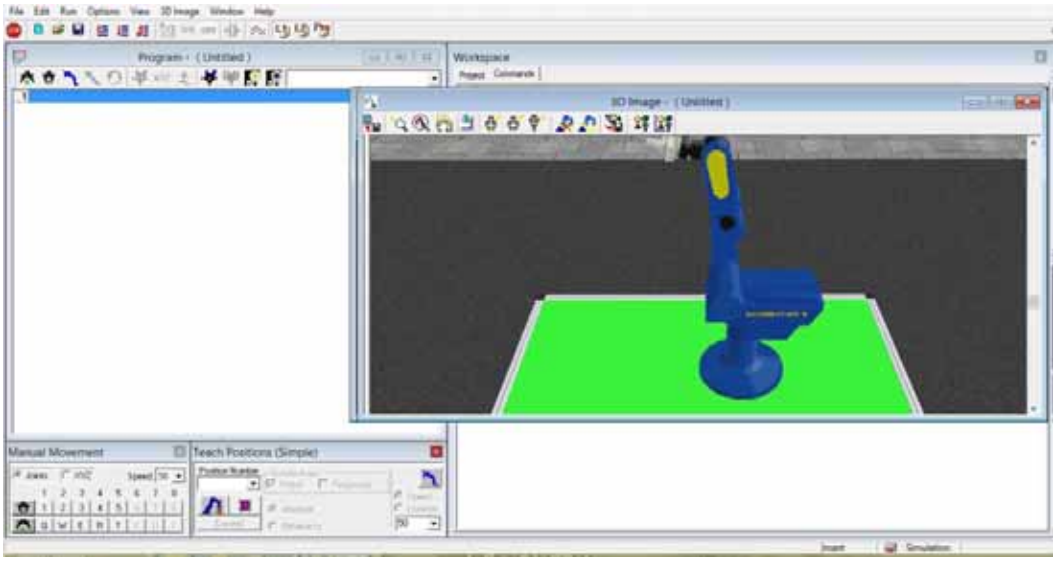

Figura 5.5: Elementos del proyecto.

La Figura 5.6 y Figura 5.7 muestran los movimientos que tiene el brazo robótico en modo manual.

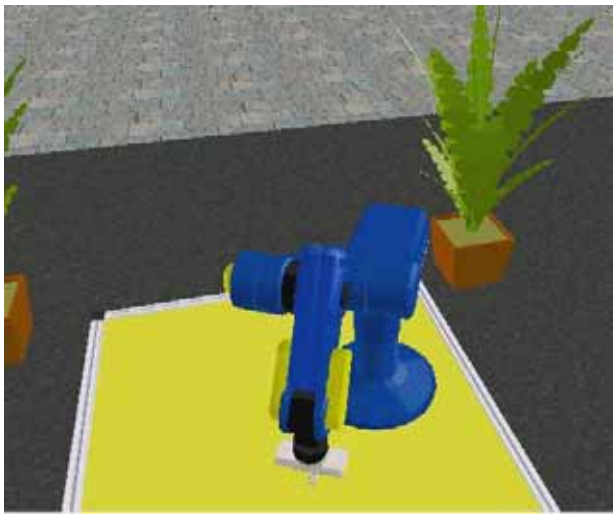

Figura 5.6: Movimientos del brazo SCORBOT-ER-9Pro.

Figura 5.7: Movimientos del brazo SCORBOT-ER-9Pro.

Los movimientos que se muestran en la Figura 5.6 y Figura 5.7 fueron pruebas que utilicé para el entendimiento del uso del brazo robótico con la apoyo del *Maual Movement* permitiendome llevar al brazo robótico a un posición deseada.

Para el proyecto fue necesario obtener las coordenadas de los objetos, es aquí donde utilicé *Teach positions* su funcionamiento permite que una vez que se tiene la posición a obtener sus coordenadas (X, Y, Z, P y R), solo se requiere Presionar el botón de *Get Position* para obtener sus valores referidos en mm como se parecía en la Figura 5.8.

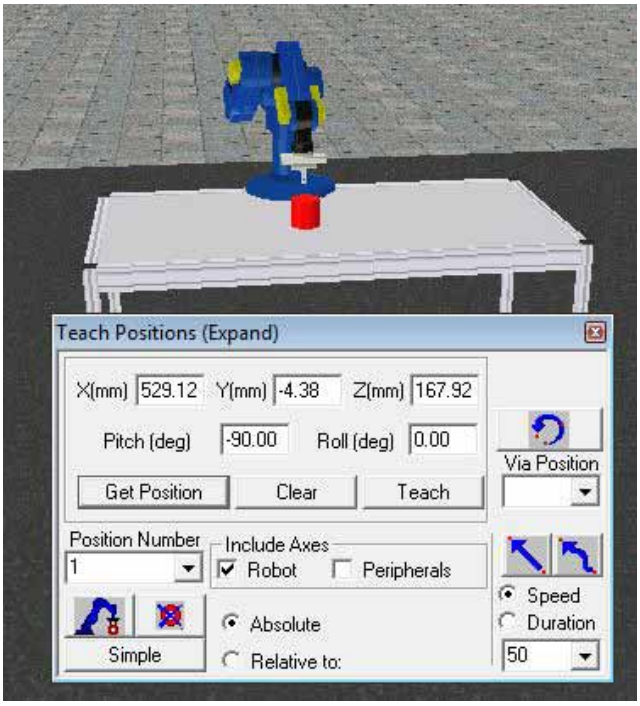

Figura 5.8: Obtención de coordenadas con Get Position.

Es necesario grabar posiciones para que el brazo robótico pueda saber los movimientos d donde debe moverse durante la ejecución, este procedimiento se realiza con ayuda del *Teach Positions*, primero se lleva al brazo robótico a la posición a grabar y una vez ahí se asigna un número a esa posición para que sirva de referencia y se termina el proceso presionando el botón de grabado (que es representado por un brazo de color azúl) como se ve en la Figura 5.9 y la posición con el número asignado queda lista para usarse durante el set de instrucciones.

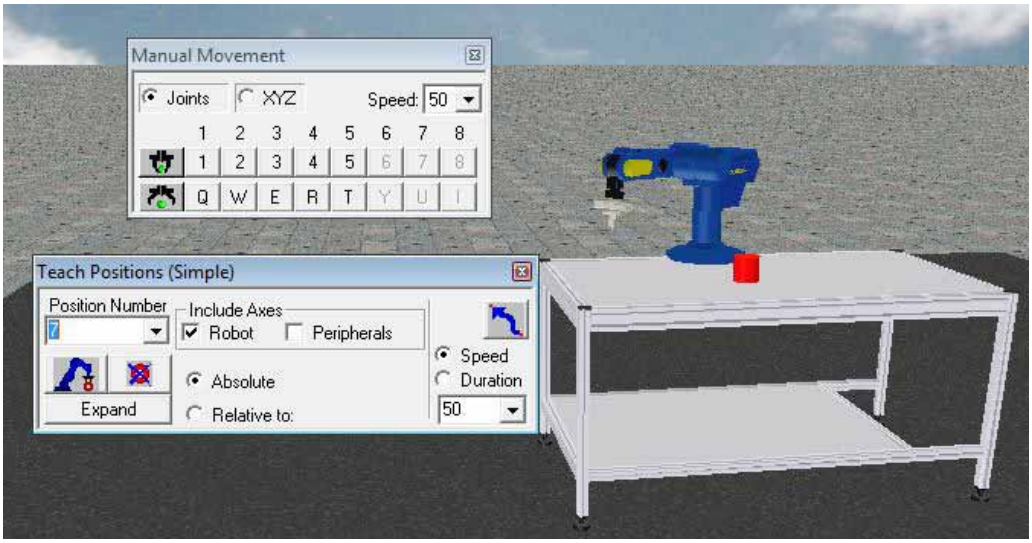

Figura 5.9: Grabado de posiciones.

La Figura 5.10 muestra el set de instrucciones que debe ejecutar el brazo robótico, en el se incluyen posiciones grabadas para poder clasificar un objeto.

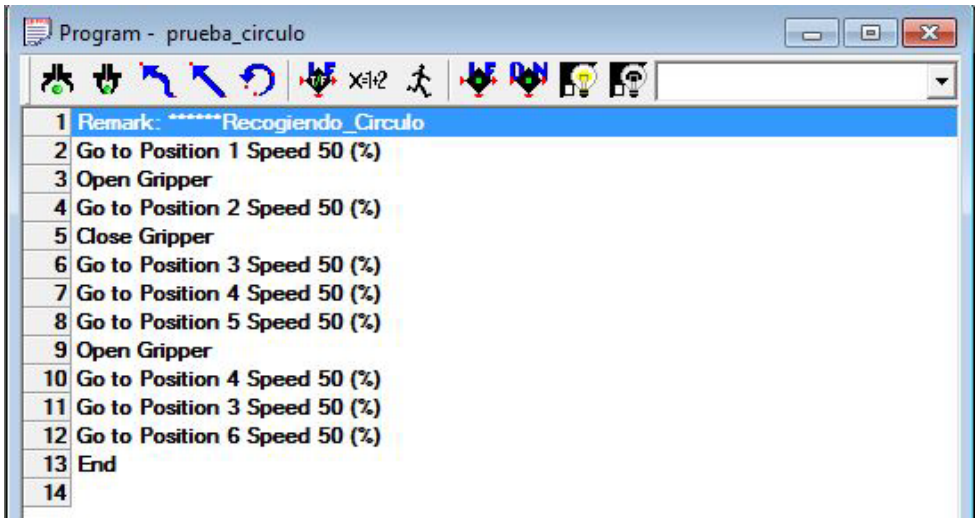

Figura 5.10: Set de instrucciones a ejecutar.

Después de tener las posiciones grabadas se requiere que se ejecute el brazo el cual desarrollará las siguientes fases:

- Se colocará debajo del objeto.
- Abrirá pinzas.
- Tomará el objeto.
- Cerrará pinzas.
- Levantará el objeto.
- Llevará el objeto a su nueva posición.
- Bajará el objeto.
- Abrirá pinzas.
- Suelta el objeto.
- Sube brazo.
- Cierra pinzas.
- Regresa a home[7].
- Esperará otro objeto.

La Figura 5.11, Figura 5.12, Figura 5.13, Figura 5.14, Figura 5.15 y Figura 5.16 muestran los movimientos que hace el brazo robótico una vez que se ejecuta el proyecto para poder llevar el objeto de su posición de origen a su nueva posición de acuerdo con el objeto que se desea clasificar.

[7]Posición de origen del brazo robótico pregrabado.

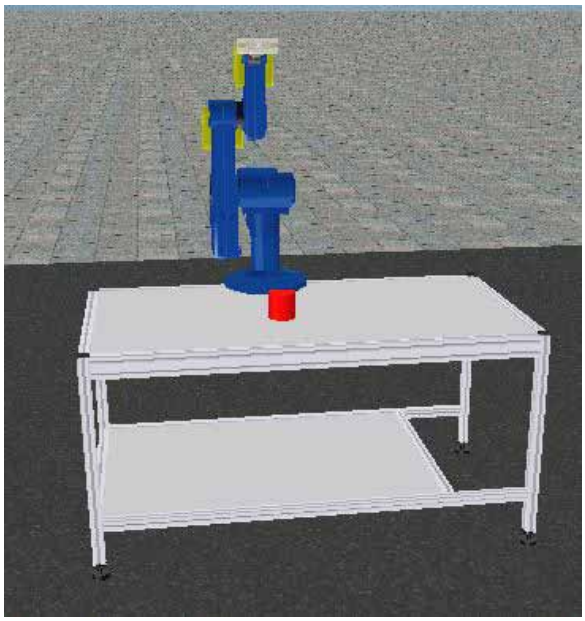

Figura 5.11: Objeto a clasificar.

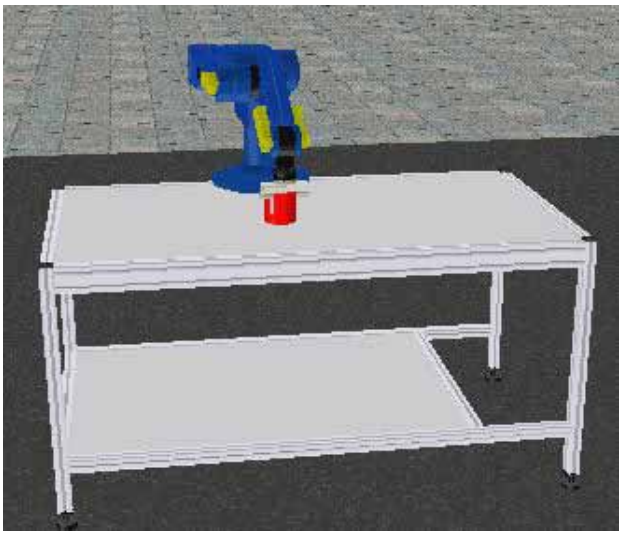

Figura 5.12: Tomando el objeto a clasificar.

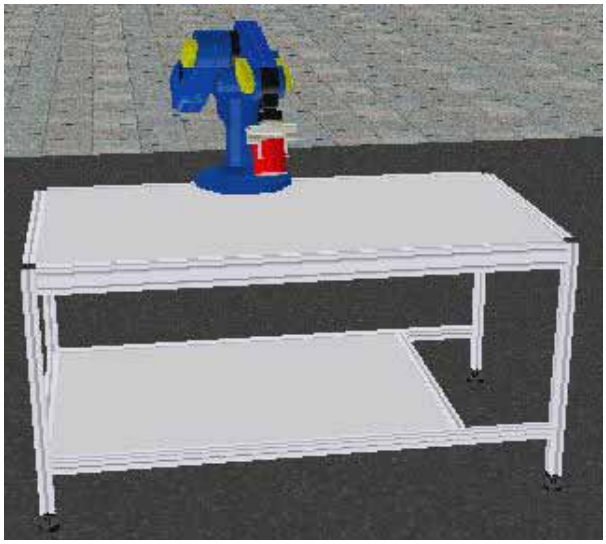

Figura 5.13: Levantando el objeto a clasificar.

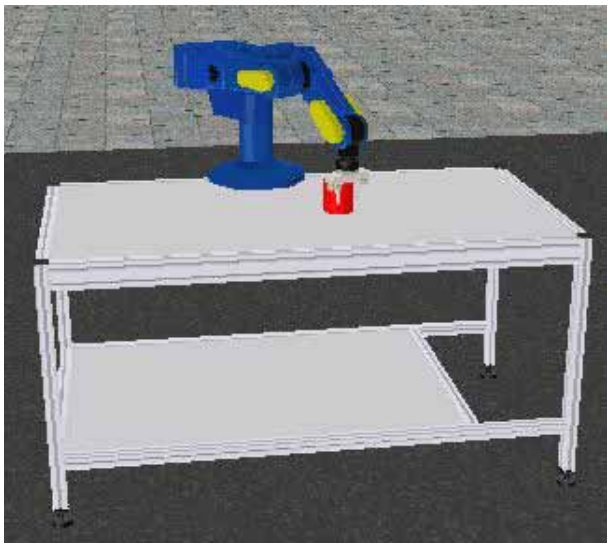

Figura 5.14: Colocando al objeto en su nueva posición.

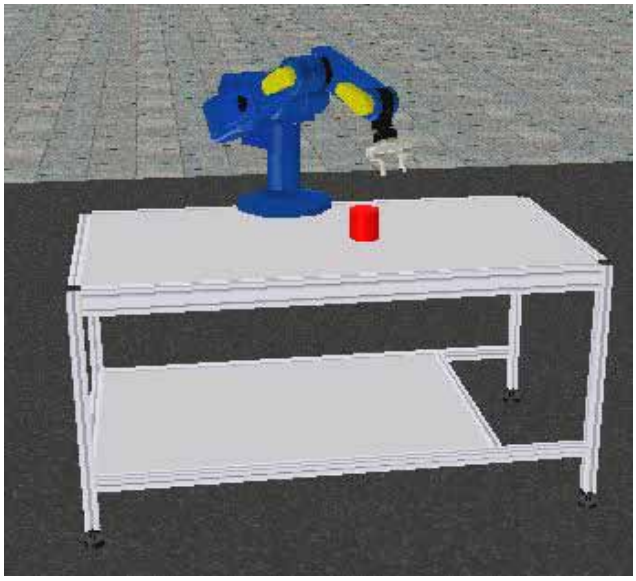

Figura 5.15: Regresando el brazo a su posición de origen.

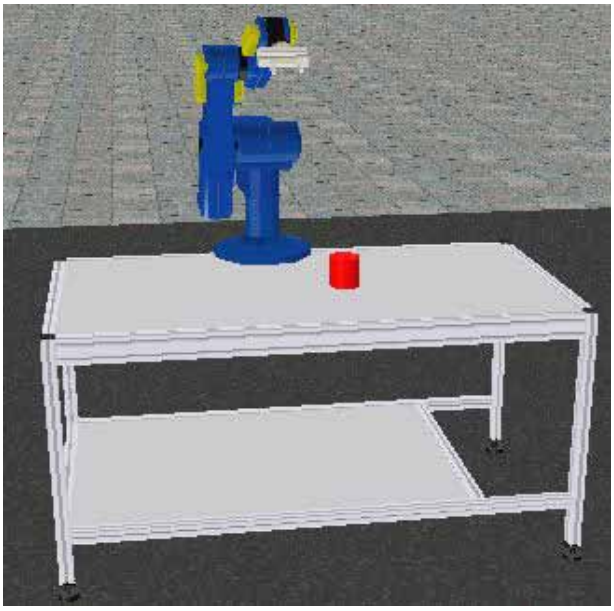

Figura 5.16: Brazo en posición de inicio en espera de otro objeto.

Capítulo 6

Brazo robótico SCORBOT-ER-Vplus

6.1. SCORBOT-ER-Vplus

Este brazo robótico es un robot vertical articulado con cinco juntas rotativas que posee seis grados de libertad, su controlador autónomo y multitarea trabaja en tiempo real permitiendo ejecutar simultáneamente e independientemente hasta un total de 20 programas en memoria, podemos decir también que es un robot rápido, seguro, flexible y fiable. La Figura 6.1 Muestra las partes que componen al robot.

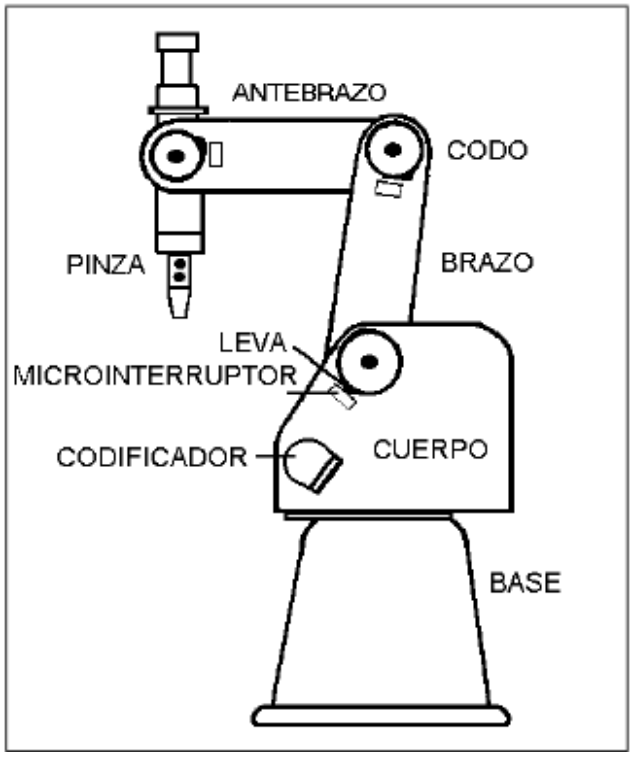

Figura 6.1: Partes del robot.

La Tabla 6.1 nos muestra los movimientos de las juntas del SCORBOT-ER Vplus.

Número de eje	Nombre	Movimiento	Motor
1	Base	Rotación del cuerpo	1
2	Hombro	Sube y baja el hombro	2
3	Codo	Sube y baja el antebrazo	3
4	Elevación	Sube y baja la inclinación de la pinza	4 + 5
5	Giro	Gira la pinza	4 + 5

Tabla 6.1: Movimiento de las juntas del SCORBOT-ER-Vplus.

La Tabla 6.2 señala las especificaciones con las que cuenta el SCORBOT-ER Vplus.

Componentes	Descripción
Estructura mecánica	Robot de articulación vertical
Número de ejes	Cinco ejes más pinza
Movimiento de ejes	
Eje 1: Base	310
Eje 2: Hombro	+ 130 / -35
Eje 3: Codo	+- 130
Eje 4: Elevación	+- 130
Eje 5: Giro	Sin límite mecánico eléctricamente +-570
Rango de operación	610mm
Elemento terminal	DC pinza servo con codificador óptico
Máxima apertura de pinza	75 mm sin almohadillas y 65 mm con almohadillas
Inicio	Posición fija en cada eje
Realimentación	Codificadores ópticos en cada eje
Impulsores	Motores servo 12 VCC
Potencia de motores	70 W potencia en pico de par
Transmisión	Engranes, correas y husillos
Máxima carga de trabajo	1 Kg. incluyendo la pinza
Peso	11.5 Kg
Velocidad máxima	600 mm/seg

Tabla 6.2: Especificaciones del SCORBOT-ER-Vplus.

6.2. Métodos de operación

El SCORBOT-ER Vplus puede ser programado y operado por medio de software como son: *ACL*, *ATS* y *SCORBASE* o por un hardware denominado *botonera de enseñanza*.

- **ACL**. (Advanced Control Language) es un lenguaje de programación robótico moderno de tareas múltiples, desarrollado por Eshed Robotec, el cual es programado en un grupo de *EPROMs* en el ControladorA y puede ser accedido desde cualquier terminal estándar o PC por medio de un canal de comunicación *RS232*[1], las cualidades de ACL incluyen:
  - Control directo del usuario de los ejes robóticos.
  - Programación del sistema robótico por parte del usuario.
  - Control de datos de entrada y salida.
  - Ejecución de programas simultanea, sincronizada e interactiva.
  - Administración simple de archivos.
- **ATS**. (Advanced Terminal Software) es la interface del usuario al ACL del controlador, el cual es suministrado en disquete y opera en cualquier PC; el software por su parte es un emulador de terminal que permite acceso al ACL desde el PC y sus cualidades incluyen:
  - Configuración abreviada del controlador.
  - Definición de dispositivos periféricos.
  - Uso de teclas para insertar órdenes.
  - Editor de programa.
  - Administración de respaldos.
- **SCORBASE**. Se conoce como un paquete de software de control robótico que puede ser utilizado con el ControladorA, su estructura está basada en menús y sus habilidades de operación *off-line*[2] facilitan la programación y la operación robótica, el cual es suministrado en disquete y opera en cualquier sistema de PC. SCORBASE se comunica por medio de ACL el lenguaje interno del controlador que usa un canal RS232.

[1]Es una interfaz que designa una norma para el intercambio de una serie de datos binarios entre un DTE (Equipo terminal de datos) y un DCE(Equipo de comunicación de datos).

[2]Se dice que está fuera de línea cuando está desconectado del sistema.

- **Botonera de eneñanza**. La botonera de enseñanza es una terminal sostenida en mano usada para controlar el robot y ejes conectados al ControladorA, es muy práctica para mover los ejes, guardar posiciones, envíar los ejes a posiciones guardadas y activar programas. Se puede ejecutar funciones adicionales desde la botonera de enseñanza, su visualizador es una pantalla de cristal líquida de 2 líneas, 32 caracteres, muestra el estado del controlador, la orden del usuario y mensajes del sistema, además de poseer 30 teclas en donde muchas de sus ordenes son de ACL.

6.3. Sistema de Coordenadas

El robot SCORBOT-ER Vplus puede ser operado y programado en dos sistemas diferentes de coordenadas: *sistema de coordenadas de ejes (Joints)* y *coordenadas cartesianas (XYZ)*.

- **Coordenadas de ejes (Joints)**. Este tipo de coordenadas especifican la locación de cada eje según la cuenta de su codificador, cuando el eje se mueve al codificador óptico genera una serie de señales eléctricas altas y bajas alternadas, dicho número es proporcional a la cantidad de movimientos del eje, el controlador las cuenta y determina qué distancia recorrió el eje. Similarmente un movimiento o una posición del robot pueden ser definidas como un específico número de cuentas del codificador para cada eje con relación a la posición de inicio a otra coordenada, estos se mueven inndividualmente según la orden dada.

- **Coordenadas Cartesianas**. El sistema de las coordenadas cartesianas o *XYZ* como también es conocido, se denomina un sistema geométrico usado para especificar la posición del *PCH* (punto central de la herramienta, generalmente de la pinza) del robot por medio de la distancia en unidades lineales de su punto de origen (el centro de la base) a lo largo de los tres ejes lineales como se ve en la Figura 6.2.

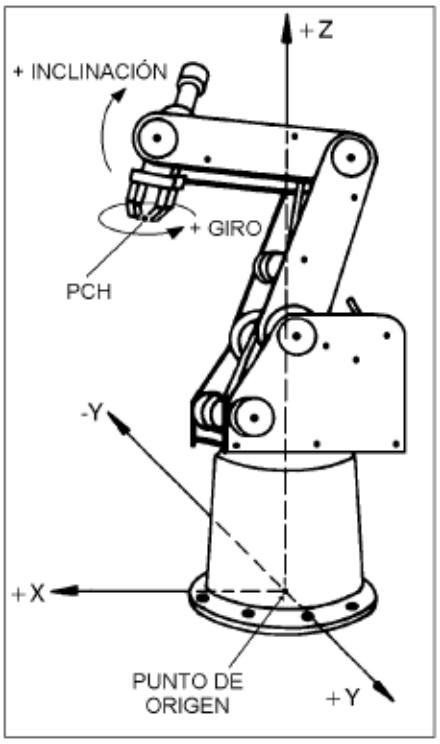

Figura 6.2: Coordenadas cartesianas.

6.4. Comunicación por puerto serial

Para desarrollar la comunicación serial es neceario que primero este conectado el brazo robótico directamente a nuestra computadora por medio del conector RS232, la comunicación que se establece con el robot es bit a bit, es decir permite envíar un bit a la vez al contrario de dispositivos de transmisión más modernos que envían varios bits simultáneamente, por lo tanto, se requiere de un lenguaje de programación en el cual pueda comunicarme a través del puerto serial con la red neuronal y además debe ser compatible en la integración de la aplicación que junta todas las fases de desarrollo del proyecto.

La conexión del puerto serial con el SCORBOT la desarrolle en un lenguaje C, debido a la compatibilidad que hay entre la interfaz del sistema que es desarrollada en C++ destacando así la línea de eficiencia en la aplicación y la portabilidad. Es necesario agregar la librería *LnxComm*[3] la cual está creada para la comunicación con el brazo robótico por medio del RS232, esta librería está diseñada en su totalidad por el lenguaje C y las funciones utilizadas para el desarrollo del proyecto son las siguientes:

[3]Librería que permite conectar el brazo robótico con el puerto serial con Linux

- **Open_port**. Esta corresponde a la función de léctura del puerto serial, es decir abre el puerto de comunicaciones.

```
HANDLE Open_Port(char comx[]);
```

Esta función recibe como parámetro una cadena con el nombre del puerto y devuelve una variable de tipo *HANDLE* que es el manejador del puerto, el cual es un índice a una tabla que sirve para almacenar datos dependiendo de la arquitectura.

- **Get_Configure_Port**. Proporciona la configuración actual del puerto serie.

```
DCB Get_Configure_Port(HANDLE fd);
```

Esta función devuelve un variable de tipo DCB con la configuración actual del puerto serie, la función recibe un parámetro de tipo *HANDLE* que es el manejador con la configuración actual del puerto devuelto por la función Open_port.

  - *fd*
    Es el HANDLE manejador del puerto devuelto por Open_port.
  - *return*
    Regresa una estructura DCB con la configuración actual del puerto.

- **Configure_Port**. Con esta función se stablece la nueva configuración del puerto serial dependiendo del periférico que se desee manejar, en este corresponde al SCORBOT ER-V Plus, que necesita manejarse a 9600 baudios, bits de datos de tamaño 8, ninguna paridad y bit de parada 1.

```
DCB Configure_Port( HANDLE fd, unsigned int BaudRate,
   char CharParity[]);
```

Esta función configura el puerto serie con los parámetros fd, BaudRate y CharParity.

  - *fd*
    Es el manejador del puerto devuelto por la función Open_Port.
  - *BaudeRate*
    Es la velocidad del puerto como puede ser (B115200, B19200, B9600, ...).
  - *CharParity*
    Indica el número de bits de la transmisión. (8N1,7E1,7O1,7S1)
  - *return*
    Regresa una estructura DCB con la configuración del puerto.

- **Set_Configure_Port** . Establece la configuración del puerto, es decir reestablecé la configuración antigua del puerto una vez que se termina de usar, esto debe ocurrir antes de cerrar el puerto serie.

```
int Set_Configure_Port( HANDLE fd,DCB PortDCB);
```

Restituye la configuración del puerto serie, los parámetros serán pasados mediante una variable tipo DCB.

- *fd*
  Es el HANDLE manejador del puerto devuelto por Open_port.
- *return*
  Regresa un valor entero.

- **Write_Port**. Esta función escribe una cadena de datos en el puerto serie.

```
long Write_Port( HANDLE fd, char Data[], int SizeData);
```

Escribe los SizeData primeros de Data en el puerto serie.

- *fd*
  Es el HANDLE manejador del puerto devuelto por Open_port.
- *DATA*
  Es el dato a mandar.
- *SizeDATA*
  Es el número de bytes que se quieren escribir.
- *return*
  Regresa el número de bytes escritos en caso de éxito.

- **Read_Port**. Esta es para leer un bloque de datos en el puerto serial.

```
long Read_Port( HANDLE fd, char *Data, int SizeData);
```

Lee los SizeData primeros del puerto y los carga en Data.

- *fd*
  Es el HANDLE manejador del puerto devuelto por Open_port.
- *DATA*
  Es la variable en donde se reciben los datos
- *SizeDATA*
  Es el número de bytes que se quieren recibir.
- *return*
  Regresa en caso de éxito el número de bytes leídos, cero indica que no se a leído nada y en caso de error devuelve 1.

- **Create_Thread_Port** . Crea una función hilo que se ejecuta cuando existan caracteres en el buffer de entrada del puerto.

```
pthread_t Create_Thread_Port( HANDLE *fd);
```

  Recibe como parámetro el manejador del puerto y devuelve una variable de tipo pthread_t.

  - *fd*
    Es el HANDLE manejador del puerto devuelto por Open_port.
  - *return*
    Regresa el manejador del hilo creado.

- **Close_Port** . Cierra el puerto serie que se abrio previamente.

```
int Close_Port( HANDLE fd);
```

  Recibe la variable fd y cierra el puerto.

  - *fd*
    Es el HANDLE manejador del puerto devuelto por Open_port.
  - *return*
    Regresa TRUE si se ha cerrado el puerto y FALSE en caso contrario.

Teniendo en cuenta las funciones anteriores y comenzando con nuestra comunicación por puerto serial con el brazo robótico, debemos tener en consideración que antes de envíar cualquier comando lo primero que se hace es abrir el puerto, se obtiene su configuración y se le indica que va a ser de tipo no bloqueante y se envía la siguiente cadena de carácteres para inicializar:

```
char mensaje[]="|echo\r\r\r";           // comado ACL
Write_Port(fd,mensaje,strlen(mensaje)); // escribir en puerto
```

Realizado el paso anterior podemos envíar los comandos que queramos para que el brazo robótico los interprete y ejecute, por ejemplo:

```
char mensaje2[]="open\r";                 // comado ACL
Write_Port(fd,mensaje2,strlen(mensaje2)); // escribir en puerto
```

Es importante decirle al brazo robótico que ya no serán enviados más comandos y esto se hace mediante la siguiente instrucción:

```
char mensaje3[]="show par 1 \r\r\r";      // comando ACL
Write_Port(fd,mensaje3,strlen(mensaje3)); // escribir en puerto
```

6.5. Programación ACL del sistema robótico

Los comandos del Lenguaje ACL que se utilizarón en la aplicación cumplieron con el objetivo de poder mover el brazo robótico, las secuencias de comandos estuvieron programadas internamente en la aplicación desarrollada. La Tabla 6.3 muestra los comandos que se necesitaron en la aplicación:

Comando	Descripción
ECHO	Es el primer comando que se necesita antes de envíar cuálquier acción al controlador.
MOVE 0	Realiza un proceso de inicialización de los ejes del robot.
SPEED 50	La velocidad se asigna en porcentajes, la máxima es 100. y la mínima es 1.
DEFP pos	Crea una variable llamada pos en el grupo A. Crea una posición en el grupo A, en el grupo B o en un eje del grup o C, si no hemos especificado un grupo supone el grupo A.
HERE pos	Graba la posición actual en la variable de memoria pos creada con DEFP o DIMP.
LISTPV pos	Despliega las coordenadas de la posición especificada pos en valores de las articulaciones y coordenadas cartesianas.
LISTPV POSITION	Es un nombre reservado y se emplea para desplegar las coordenadas actuales. Las coordenadas cartesianas x, y, z se expresan en décimas de de mílimetros e indican la distancia entre el origen del sistema global de coordenadas del robot y el TCP (Tool Center Point= Coordenada de la Punta de la Tenaza).
SETPVC pos X 2799	Este comando sirve para modificar posiciones, permitiendo cambiar una coordenada cartesiana en una posición previamente registrada con el comando SETPV.
MOVE pos	Este comando mueve el brazo a la posición definida en pos, esta instrucción deposita un comando de movimiento en el buffer de movimientos, y no espera a que la operación haya sido terminada para envíar otra instruciión de movimiento, el desplazamiento lo realiza de acuerdo a la velocidad actual.
CLOSE	Cierra la pinza hasta el final de su movimiento.
OPEN	Abre la pinza hasta el final de su movimiento.
DELP pos	Borra las posiciones y los vectores de posición.

Tabla 6.3: Lista de comandos.

Capítulo 7

Integración del proyecto

7.1. Implementación del hardware

A continuación se va describir la integración del proyecto en el sistema clasificador de objetos por medio de redes neuronales que permite seleccionar objetos mediante un brazo robótico SCORBOT-ER-Vplus. En el desarrollo de este capítulo se describirá todo el hardware empleado y las partes más importantes del código de los programas de cada etapa del sistema, de tal forma que sea fácil su entendimiento en el desarrollo del software.

7.1.1. Instalación de la webcam

La webcam que se utiliza es una HP de 3 Megapíxeles, se requiere instalar un paquete para que sea reconocida por nuestro sistema operativo y para ello debemos escribir el siguiente comando en consola:

```
# apt-get install cheese
```

Una vez que se tiene instalada la webcam, podemos verificar que se instalo de manera correcta y ejecutamos el siguiente comando en consola como lo muestra la Figura 7.1.

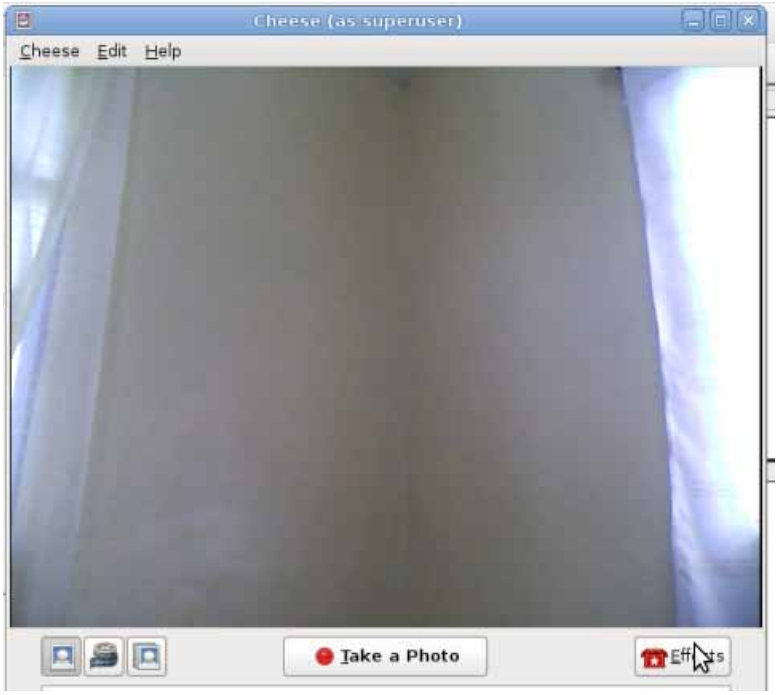

Figura 7.1: Ejecución de webcam.

Una vez que la webcam es ejecutada de manera correcta, se debe agregar el código necesario para que sea ejecutado por la aplicación por lo que el código de captura de la imagen es el siguiente:

```
1           IplImage* frame = NULL;
2           CvCapture* capture= NULL;
3
4                   capture= cvCaptureFromCAM(1);  //para webcam
                       externa=1 y =0 para la de la Laptop
5                   cvNamedWindow( "Imagen a Capturar",
                       CV_WINDOW_AUTOSIZE) ;
6                   cvMoveWindow( "Imagen a Capturar",380,120);
7           while(1)
8           {
9                   frame = cvQueryFrame(capture) ;
10                  cvShowImage( "Imagen a Capturar",frame);
11                  char c = cvWaitKey(10);
12                  if(c=='q'|| c =='Q')
13                  {
14                        //guarda la imagen como prueba.jpg
15                        cvSaveImage("prueba.jpg",frame);
16                        break;
17                  }
18          }
```

7.1.2. Interface PC-Robot SCORBOT

Esta aplicación se desarrolla en una Computadora Laptop Marca HP Modelo 435 con procesador AMD Athlon(tm) II P360 Dual-Core Processor 2.30 Ghz, Memoria RAM de 6 GB en la cual se anexa una webcam marca HP de 3 megapíxeles que es la encargada de la captura la imagen de los objetos, esta computadora deberá envíar las consignas al robot como resultado del análisis realizado, dichas consignas son la información necesaria (posición de los objetos, orientación, número de objetos, etc) para que el robot realice la tarea determinada y se envía mediante una conexión RS232 entre el controlador del robot y la PC.

7.1.3. Sistema de iluminación

La iluminación es un factor muy importante en el proceso de visión para la obtención de resultados óptimos, por lo que una deficiente iluminación implica obtener imágenes de baja calidad que a su vez con llevan a resultados erróneos o a un procesamiento excesivo. Observando la Figura 7.2 nos muestra el sistema de iluminación empleado en el proyecto.

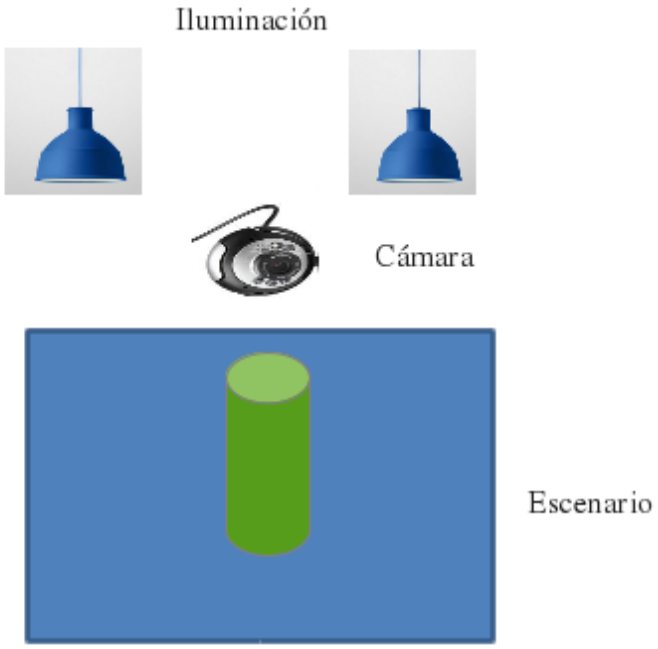

Figura 7.2: Sistema de iluminación empleado.

7.2. Implementación del Software

La implementación se desarrollo en la plataforma de Netbeans, para poder realizar cambios en una primera etapa y poder diseñar los algoritmos correspondientes para la aplicación y posteriormente implementarlos en C\C++ siendo el lenguaje en el que finalmente se implementó el sistema de red.

7.2.1. Binarización de la imagen

El código empleado para llevar a cabo una binarización se muestra a continuación:

```
1           IplImage* imagenColor;          //imagen de color
               base
2           IplImage* imagenEscalaGrises;   //imagen escala
               grises
3           IplImage* imagenBinarizada;     //imagen binaria
               conseguida a partir de la imagen en escala de
               grises
4
5           int threshold= 115;                  //valor del
               umbral 115
6           int maxValue= 255;                   //valor maximo
7           int thresholdType= CV_THRESH_BINARY_INV;    //tipo de
                binarizacion INV para de color  y sin INV para
               negras
8
9           imagenColor= cvLoadImage(imagenOrigen,1);      //
               cargamos imagen
10          imagenEscalaGrises= cvCreateImage(cvSize(imagenColor
               ->width, imagenColor->height),IPL_DEPTH_8U,1);//1
               para escala de grises
11          printf("\n---- Imagen Sample ----\n");
12          printf("\n");
13          //la imagen de intensidad tendra la misma
               configuracion que la fuente pero con 1 canal
14          cvCvtColor(imagenColor, imagenEscalaGrises,
               CV_BGR2GRAY); //pasamos la imagen color a escala
               de grises
15          imagenBinarizada= cvCloneImage(imagenEscalaGrises);
                        //copiamos la escala en escala de grises
16          cvThreshold(imagenEscalaGrises,imagenBinarizada,
               threshold, maxValue,thresholdType); //binarizamos
               imagen
17          cvNamedWindow("Imagenes Disponibles Binarizadas",1);
                           //representamos la imagen de color
               binarizada
18          cvShowImage("Imagenes Disponibles Binarizadas",
               imagenBinarizada);
```

```
19          cvMoveWindow("Imagenes Disponibles Binarizadas"
                ,200,90);
20          cvSaveImage(imagenDestino,imagenBinarizada);   //
                guardamos la imagen binarizada
21          cvNamedWindow("Imagenes Disponibles a Color",1);
                                    //representamos la imagen
22          cvShowImage("Imagenes Disponibles a Color",
                imagenColor);
23          cvMoveWindow("Imagenes Disponibles a Color",200,90);
                 //horizontal vs vertical
24
25          cvWaitKey(0);                                       //
                pulsamos tecla para terminar
26          cvDestroyWindow("Imagenes Disponibles a Color");
                           //destruimos todas las ventanas
27          cvDestroyWindow("Imagenes Disponibles Binarizadas");
```

7.2.2. Cálculo de las características

Cuándo se obteniene la imagen libre de ruido y sólo el objeto de nuestro interés, se procede a extraer sus características como son su área, perímetro y compacidad, para después poder clasificar los demás objetos con uno de los patrones antes definido, el código empledo para la obtención de las caracteríticas de los objetos apoyados de la librería CvBlobs es el siguiente:

```
1           //variables para calcular el centoroide
2           //momento de orden 1 en X
3           CBlobGetMoment m10(1,0);
4           //momento de orden 1 en Y
5           CBlobGetMoment m01(0,1);
6           //obtencion de datos
7           in[j]=blob.Area();
8           j=j+1;
9           in[j]=blob.Perimeter();
10          j=j+1;
11          in[j]=(pow(blob.Perimeter(),2)/(4*CV_PI*blob.Area()))
                ;
12          j=j+1;
```

7.2.3. Resultados de la red neuronal

Los resultados que se obtuvieron del diseño de red neuronal implementada fueron los esperados, debido a que la red neuronal podía clasificar los objetos por medio de su forma, una vez que se obtenia el área, perímetro y compacidad de cada objeto. El código que se empleo para definir la red neuronal fue el siguiente:

```
1          //3 capas de entrada= a,p,c
2          //2 capas ocultas c/u de 13
3          //4 capas de salida= c,t,c,r
4           //layer= capas o neuronas en la red
5           int layer[]={3,13,13,4};
6
7           //se define la arquitectura de red
8           //definimos la entrada input de la red con CvMat
9           //cvCreateMat(num,3,CV_32FC1) regresa una matriz de
               punto flotante de los vectores de entrada x un
               vector fila
10          //      los parametros de entrada son (num=filas=#
               blobs, col=3=a.p.c,CV_32FC1=tipodatoqueva a la
               matriz)
11          CvMat *input= cvCreateMat(num,3,CV_32FC1);
12          //definimos la salida target
13          CvMat *target= cvCreateMat(num,4,CV_32FC1);
14          //definimos las neuronas en cada capa incluida la
               entrada y salida(layersize) de la red
15          CvMat *layersize= cvCreateMat(1,4,CV_32SC1);
16
17          //se copian los valores a la arquitectura de red
18          cvInitMatHeader(input,num,3,CV_32FC1,in);
19          cvInitMatHeader(target,num,4,CV_32FC1,out);
20          cvInitMatHeader(layersize,1,4,CV_32SC1,layer);
21
22          //las neuronas se crean a partir de la clase
               CvANN_MLP
23          //que permite usa r la funcion net(layersize,
               CvANN_MLP::SIGMOID_SYM,1,1);
24          //donde 1 parametro vector q especifica el numero de
               neuronas en cada campo
25          //2 parametro funcion de activacion para cada neurona
               funcion simetrica sigmoidal
26          //3 y 4 parametro son libres los cuales son los
               valores que se le da a alfa y beta de la fun
               sigmoid
27          //la red la cual es de tipo perceptron multicapa
28
29          //se asignan pesos a la RN de acuerdo a la funcion de
               transferencia Sigmoide
30          CvANN_MLP net(layersize,CvANN_MLP::SIGMOID_SYM,1,1);
```

```
31
32          //ITERACIONES REALIZADAS EN EL ENTRENAMIENTO
33          //5000 MAXIMO NUMERO DE ITERACIONES
34          //0.001 ERROR DE OBJETIVO
35          int iter= net.train(input,target,NULL,0,
               CvANN_MLP_TrainParams(cvTermCriteria(
               CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,5000,0.001),
               CvANN_MLP_TrainParams::BACKPROP,0.2,0.1));
```

7.2.4. Sincronización del brazo robótico

Una vez que se tiene realizada la conexión con el brazo robótico por medio de la aplicación se debe mandar los comandos para que el brazo pueda realizar los movimientos necesarios para poder clasificar al objeto en su zona dependiendo su forma, el código implementado para que el brazo pueda clasificar un cuadrado a la zona 1 es el siguiente:

```
1             if (objeto==0){ //debe ser un cuadrado y va a la
                 izquierda zona 1
2               printf("FUE UN CUADRADO MOVIENDO A ZONA 1\n");
3               imagenColor= cvLoadImage("scorbot1.jpeg");    //
                   cargamos la imagen
4               cvNamedWindow("MOVIENDO A ZONA 1", 1);        //
                   representamos la imagen
5               cvShowImage("MOVIENDO A ZONA 1",imagenColor); //
                   mostramos la imagen
6               cvMoveWindow("MOVIENDO A ZONA 1",35,15);      //
                   movemos la ventana
7               cvWaitKey(0);                                 //
                   pulsamos tecla para terminar
8               cvDestroyWindow("MOVIENDO A ZONA 1");         //
                   destruimos todas las ventanas
```

El código implementado para realizar cada movimiento del SCORBOT-ER-Vplus se muestra en el set de instrucciones del objeto cuadrado, el código empleado se muestra a continuación:

```
1                //haz los pasos para mover a zona1
2
3                    char inicializar[]="|echo\r\r\r";        //
                        inicializa el envio de comandos
4                    char speed[]="speed 50\r";               //
                        velocidad a 50%
5                    char moved1[]="moved demo1\r";           //
                        posicion arriba del objeto
6                    char open[]="open\r";                    //
                        abre la pinza
7                    char moved2[]="moved demo2\r";           //
                        toma el objeto
8                    char close[]="close\r";                  //
                        cierra la pinza
9                    char moved3[]="moved demo3\r";           //
                        alza objeto
10                   char moved4[]="moved demo4\r";           //
                        mueve a la izquierda
11                   char moved5[]="moved demo5\r";           //
                        baja objeto
12                   char open2[]="open\r";                   //
                        abre la pinza
13                   char moved6[]="moved demo6\r";           //
                        levanta brazo
14                   char moved7[]="moved demo7\r";           //
                        mueve a la derecha
15                   char moved8[]="moved demo8\r";           //
                        regresa a home
16                   char finalizar[]="|show par 1\r\r\r";    //
                        finaliza comunicaci n
17
18                   fd=Open_Port("dev/ttyUSB0");             //
                        Abre el puerto serie fd=Open_Port("/dev/
                        ttyUSB0");
19                   OldConf=Get_Configure_Port(fd);          //
                        Guardo la configuraci n del Puerto
20                   Configure_Port(fd,B9600,"8N1");          //
                        Configuro el puerto serie
21                   IO_Blocking(fd,FALSE);                   //
                        Seleccionamos lectura no bloqueante
22                   escrituraLecturaScorbot(inicializar,fd);//Se
                        llama a las funciones que escribe y lee
                        del puerto serial
23                   escrituraLecturaScorbot(speed,fd);
24                   escrituraLecturaScorbot(moved1,fd);
```

```
25                      escrituraLecturaScorbot(open,fd);
26                      escrituraLecturaScorbot(moved2,fd);
27                      escrituraLecturaScorbot(close,fd);
28                      escrituraLecturaScorbot(moved3,fd);
29                      escrituraLecturaScorbot(moved4,fd);
30                      escrituraLecturaScorbot(moved5,fd);
31                      escrituraLecturaScorbot(open2,fd);
32                      escrituraLecturaScorbot(moved6,fd);
33                      escrituraLecturaScorbot(moved7,fd);
34                      escrituraLecturaScorbot(moved8,fd);
35                      escrituraLecturaScorbot(finalizar,fd);
36                      Set_Configure_Port(fd,OldConf);          //
                           Restituyo la antigua configuraci n del
                           puerto
37                      Close_Port(fd);                          //
                           Cierro el puerto serie
```

Las posiciones se obtuvieron por medio del comando *LISTPV position*, pues se maneja al brazo robótico a la posición deseada y una vez que nos encontrabamos en el lugar deseado basta con ejecutar el comando en la terminal ATS para que nos proporcione las coordenadas en milímetros (mm) como lo muestra la Figura 7.3.

Figura 7.3: Ejecución del comando Listpv position.

Esto se hizó para cada una de las posiciones que se iban a grabar para que el brazo realizará los movimientos necesarios, la Tabla 7.1 muestra las coordenadas del set de instrucciones para mover un cuadrado a la zona 1.

Movimientos	X	Y	Z	P	R
moved1	566.19	15.8	672.16	-23.75	0
moved2	443.53	-22.18	18	-90	0
moved3	583.56	-15.92	279.88	-58.2	0
moved4	268.06	-518.59	269.88	-68.2	0
moved5	196.94	-402.33	21.91	-90	0
moved6	268.06	-518.59	269.88	-68.2	0
moved7	583.56	-15.92	279.88	-58.2	0
moved8	283.07	-421.67	390.38	-90	0

Tabla 7.1: Posiciones de movimientos grabados.

Capítulo 8

Conclusiones

Este proyecto trata áreas muy complejas que no son estudiadas a profundidad a lo largo de nuestra carrera, por lo cual elegí un tema en donde pueda involucrar los conocimientos adquiridos y poder indagar a mayor profundidad en ellos, pues estamos preparados para convertirnos en profesionistas capaces de enfrentarse a cualquier situación, teniendo como habilidad el diseño, la innovación y la creación de nuevas técnologias enfocadas a la mejorá de nuestra sociedad.

El proyecto habla de temas relacionados con la visión artificial, robótica, inteligencia artificial y programación; en cada uno de los capítulos se logro explicar el funcionamiento que se obtuvo para concluir el proyecto llamado *Automatización de un brazo robótico para clasificar objetos por medio de redes neuronales y RTLinux*. Esto no fue fácil pues durante la elaboración del proyecto surgieron inconvenientes, en primer lugar la falta de conocimiento en determinadas áreas de estudio, por ello tuve que dedicar gran parte de mí tiempo a estudiar cada unas de esas áreas debido a que no estaban contempladas en mi plan de estudios, pero que eran necesarias para efecto del proyecto, en segundo lugar fue enfrentarme a definir claramente el proyecto aterrizandolo y no dejando de lado el principal objetivo que era lograr diseñar un sistema de automatización con redes neuronales en tiempo real, que permita el reconocimiento de objetos y el movimiento de un brazo robótico Scorbot-Er VPlus de seis grados de libertad siendo totalmente funcionable; por último realizar la programación de la red neuronal que permita al brazo robótico ejecutar las instrucciones que se le determinen mediante ordenes previamente establecidas, teniendo la capacidad de modificarlas conforme a los intereses.

La forma en que se abordo cada capítulo, en resumen lo defino como correcta, ya que estas se tomaron como procesos que a partir de los datos de entradas se debían generar salidas que servirían para el siguiente proceso, por lo tanto entre proceso y proceso debían haber integraciones ya que se deben establecer compatibilidades entre cada uno de ellos, entonces a lo largo del estudio se plantearon distintas formas de integración que fueran detalladas en el informe, que en resumidas cuentas sirvieron como puentes entre cada fase hasta lograr una comunicación de todas ellas que entregaron como resultado la creación de una aplicación.

Los resultados obtenidos de esta aplicación fueron satisfactorios debido a su desarrollo y pruebas realizadas, afinando los máximos detalles para lograr que a partir de una simple imagen tomada con una cámara web, a través de nuestro programa, se pudiera analizarla hasta llegar a identificar de que objeto se trataba por medio de la red neuronal y con esta información se le indico al brazo robótico la manera en que tenía que clasificar el objeto y el lugar en donde debía ser posicionado.

Apéndice A

Código de la aplicación

Listado A.1: Codigo en un archivo

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include <ctype.h>
#include <cv.h>
#include <highgui.h>
#include <cxcore.h>
#include <ml.h>

#include "/home/alex/Downloads/opencv-2.4.4/cvblobs8.3_linux/BlobResult.h"
#include "/home/alex/Downloads/opencv-2.4.4/cvblobs8.3_linux/blob.h"

#define __LINUX_COM__
#include "/home/alex/Downloads/Lnxcomm/com/serial.h"

IplImage *imagen=0;   //IplImage estructura que contiene informacion de una imagen
IplImage *image, *gray, *dst;

CBlobResult blobs;    //Se utiliza para manipular los blobs que aparecen con todas las regiones a calcular
CBlob blob;           //Almacena en cada momento el blob sobre el que se quiere trabajar.
CvFont dfont;         //Escribir texto en ventana que se desee

void capturaImagen();//char nombre[255], int argc, char ** argv); //captura de imagen
void binariza(char [255], char [255]);          //binariza
```

```
    imagen
26  int calculos(char [255], float [100]);           //calculo de
        regiones y datos
27  int entrenamiento(int num, float in[], float inP[]);   //
    entrenar y probar red
28  void comandoScorbot(char [],HANDLE, int );             //
    lista de comandos en ACL
29  void escrituraLecturaScorbot(char [],HANDLE);          //
    lectura y escritura de los comandos
30
31  HANDLE fd;
32
33
34  int main()
35  {
36          int num=0;          //numero de objetos de la imagen
                de entrenamiento
37          float in[100];      //Se guarda el vector de Area,
                perimetro Y Compacidad
38          float inP[100];
39          int objeto=0;
40          int salout0, salout1,salout2,salout3;
41          //capturaImagen();//"imgRGBTrain.jpg",argc,argv);
                    //captura de la imagen para entrenar RN
42          binariza("sample2.jpg","imagenBINEntrenamiento.jpg");
                 //imagen de samples de patrones
43          num= calculos("imagenBINEntrenamiento.jpg", in);
                     //regresa el numero de blobs de la
                imagenEntrena
44          objeto=entrenamiento(num,in,inP);
                                    //entrenamiento
45          comandoScorbot("mensaje[]", fd, objeto);
                           //lista de comandos en ACL
46          escrituraLecturaScorbot ("mensaje[]",fd);
                          //escritura y l ctura de cada
                comando
47
48   return 0;
49  }
50
51  void capturaImagen(){
52
53          IplImage* frame = NULL;
54          CvCapture* capture= NULL;
55
56                  capture= cvCaptureFromCAM(1);  //para webcam
                        externa=1 y =0 para la de la Laptop
57                  cvNamedWindow( "Imagen a Capturar",
                        CV_WINDOW_AUTOSIZE) ;
```

```
58                  cvMoveWindow( "Imagen a Capturar",380,120);
59          while(1)
60          {
61                  frame = cvQueryFrame(capture) ;
62                  cvShowImage( "Imagen a Capturar",frame);
63                  char c = cvWaitKey(10);
64                  if(c=='q'|| c =='Q')
65                  {
66                       //guarda la imagen como prueba.jpg
67                       cvSaveImage("prueba.jpg",frame);
68                       break;
69                  }
70          }
71          cvReleaseCapture( &capture);
72          cvDestroyWindow( "Imagen a Capturar");
73          printf("\ n         Foto tomada          \n");
74          printf("\n");
75   return;
76  }
77
78  void binariza(char imagenOrigen[255], char imagenDestino
       [255]){
79
80          //IplImage estructura que contiene informacion de una
               imagen
81          //cvCreateImage crea una imagen con un tama o ,
               formato y devuelve un apuntador a la estructura
               IplImage
82          IplImage* imagenColor;          //imagen de color
               base
83          IplImage* imagenEscalaGrises;   //imagen escala
               grises
84          IplImage* imagenBinarizada;     //imagen binaria
               conseguida a partir de la imagen en escala de
               grises
85
86          int threshold= 115;                  //valor del
               umbral 115
87          int maxValue= 255;                   //valor maximo
88          int thresholdType= CV_THRESH_BINARY_INV;    //tipo de
               binarizacion INV para de color  y sin INV para
               negras
89
90          imagenColor= cvLoadImage(imagenOrigen,1);      //
               cargamos imagen
91          imagenEscalaGrises= cvCreateImage(cvSize(imagenColor
               ->width, imagenColor->height),IPL_DEPTH_8U,1);//1
               para escala de grises
92          printf("\n---- Imagen Sample ----\n");
```

```
93          printf("\n");
94          //la imagen de intensidad tendra la misma
               configuracion que la fuente pero con 1 canal
95          cvCvtColor(imagenColor, imagenEscalaGrises,
               CV_BGR2GRAY); //pasamos la imagen color a escala
               de grises
96          imagenBinarizada= cvCloneImage(imagenEscalaGrises);
                     //copiamos la escala en escala de grises
97          cvThreshold(imagenEscalaGrises,imagenBinarizada,
               threshold, maxValue,thresholdType); //binarizamos
               imagen
98          cvNamedWindow("Imagenes Disponibles Binarizadas",1);
                        //representamos la imagen de color
               binarizada
99          cvShowImage("Imagenes Disponibles Binarizadas",
               imagenBinarizada);
100         cvMoveWindow("Imagenes Disponibles Binarizadas"
               ,200,90);
101         cvSaveImage(imagenDestino,imagenBinarizada);   //
               guardamos la imagen binarizada
102         cvNamedWindow("Imagenes Disponibles a Color",1);
                                    //representamos la imagen
103         cvShowImage("Imagenes Disponibles a Color",
               imagenColor);
104         cvMoveWindow("Imagenes Disponibles a Color",200,90);
                 //horizontal vs vertical
105
106         cvWaitKey(0);                                       //
               pulsamos tecla para terminar
107         cvDestroyWindow("Imagenes Disponibles a Color");
                          //destruimos todas las ventanas
108         cvDestroyWindow("Imagenes Disponibles Binarizadas");
109
110         cvReleaseImage(&imagenColor);                      //
               eliminamos todas las imagenes
111         cvReleaseImage(&imagenEscalaGrises);
112         cvReleaseImage(&imagenBinarizada);
113
114  return;
115 }
116
117 void binarizaPrueba(char imagenOrigen[255], char
       imagenDestino[255]){
118
119         //IplImage estructura que contiene informacion de una
               imagen
120         //cvCreateImage crea una imagen con un tama o ,
               formato y devuelve un apuntador a la estructura
               IplImage correspond
```

```
121             IplImage* imagenColor;          //imagen de color
                    base
122             IplImage* imagenEscalaGrises;   //imagen escala
                    grises
123             IplImage* imagenBinarizada;     //imagen binaria
                    conseguida a partir de la imagen en escala de
                    grises
124
125             int threshold= 115;                     //valor del
                    umbral
126             int maxValue= 255;                      //valor
                    maximo
127             int thresholdType= CV_THRESH_BINARY;    //tipo de
                    binarizacion INV para de color  y sin INV para
                    negras
128
129             imagenColor= cvLoadImage(imagenOrigen,1);       //
                    cargamos imagen
130             imagenEscalaGrises= cvCreateImage(cvSize(imagenColor
                    ->width, imagenColor->height),IPL_DEPTH_8U,1);//1
                    para escala de grises
131             printf("\n**** Imagen a Clasificar ****\n");
132             printf("\n");
133             //la imagen de intensidad tendra la misma
                    configuracion que la fuente pero con 1 canal
134             cvCvtColor(imagenColor, imagenEscalaGrises,
                    CV_BGR2GRAY); //pasamos la imagen color a escala
                    de grises
135             imagenBinarizada= cvCloneImage(imagenEscalaGrises);
                        //copiamos la escala en escala de grises
136             cvThreshold(imagenEscalaGrises,imagenBinarizada,
                    threshold, maxValue,thresholdType); //binarizamos
                    imagen
137             cvNamedWindow("Imagen a clasificar Binarizada",1);
                        //representamos la imagen de color
                    binarizada
138             cvShowImage("Imagen a clasificar Binarizada",
                    imagenBinarizada);
139             cvMoveWindow("Imagen a clasificar Binarizada",200,90)
                    ;
140             cvSaveImage(imagenDestino,imagenBinarizada);
                    //guardamos la imagen binarizada
141             cvNamedWindow("Imagen a clasificar en Color",1);
                    //representamos la imagen
142             cvShowImage("Imagen a clasificar en Color",
                    imagenColor);
143             cvMoveWindow("Imagen a clasificar en Color",200,90);
144
145             cvWaitKey(0);           //pulsamos tecla para terminar
```

```
146          cvDestroyWindow("Imagen a clasificar en Color");
                //destruimos todas las ventanas
147          cvDestroyWindow("Imagen a clasificar Binarizada");
148
149          cvReleaseImage(&imagenColor);                    //
                eliminamos todas las imagenes
150          cvReleaseImage(&imagenEscalaGrises);
151          cvReleaseImage(&imagenBinarizada);
152
153   return;
154  }
155
156  int calculos(char imgBin[255], float in[30]){
157
158      float hscale=0.5f, vscale= 0.5f;
159      float italicscale= 0.0f;
160      int thickness= 1;
161      char text[255]= "";
162      //CV_AA linea suavizado de contorno
163      //funcion que toma un grupo de argumentos que configura
            alguna fuente en particular para uso en pantalla
164      cvInitFont(&dfont,CV_FONT_HERSHEY_SIMPLEX,hscale,vscale,
            italicscale,thickness,CV_AA);
165      image= cvLoadImage(imgBin,1);
166      //funcion que permite crear imagen en especificacion 1=
            escala de grise, 3= RGB
167      gray= cvCreateImage(cvSize(image->width, image->height),
            IPL_DEPTH_8U,1);
168      dst= cvCreateImage(cvSize(gray->width,gray->height),
            IPL_DEPTH_8U,3);
169      //convierte a partir de un espacio de color a otro,
            mientras que el tipo de datos esperando a ser el mismo
170      //CV_BGR2GRAY convierte BGR o RGB a escala de grises
171      cvCvtColor(image,gray,CV_BGR2GRAY);
172      // copia estruct. imagen con estruct. dst
173      cvCopy(image,dst);
174
175      //extrae rgiones (blobs)
176      //blobs= CBlobResult(image,gray,true);//tambien funciona
177      blobs= CBlobResult(image,gray,115);
178
179      //filtra regiones que tengan un area entre 1000 y 100 000
            pixeles
180      blobs.Filter(blobs,B_INCLUDE,CBlobGetArea(),B_INSIDE
            ,1000,1000000);
181
182      int blobnum;
183      blobnum= blobs.GetNumBlobs();
184
```

```
185         //calculo de a,p,c apartir de localizar centroide
186         CvPoint p1,p2;
187         int centx, centy;    //ubicar centroide
188         int j=0;
189         for(int i=0; i<blobnum; i++){
190             blob= blobs.GetBlob(i);
191             p1.x=(int)blob.MinX();
192             p1.y=(int)blob.MinY();
193             p2.x=(int)blob.MaxX();
194             p2.y=(int)blob.MaxY();
195             //variables para calcular el centoroide
196             //momento de orden 1 en X
197             CBlobGetMoment m10(1,0);
198             //momento de orden 1 en Y
199             CBlobGetMoment m01(0,1);
200             //obtencion de datos
201             in[j]=blob.Area();
202             j=j+1;
203             in[j]=blob.Perimeter();
204             j=j+1;
205             in[j]=(pow(blob.Perimeter(),2)/(4*CV_PI*blob.Area()))
                    ;
206             j=j+1;
207 
208             sprintf(text,"%d",i);
209             //pone en la imgBIN el numero de blob
210             cvPutText(dst,text,cvPoint(p1.x,p2.y+15),&dfont,
                    CV_RGB(150,250,50));
211             printf("Blob %d: ",i);
212             printf("Area= %.2f, Perimetro= %.2f, Compacidad=%.2f
                    \n",in[j-3],in[j-2],in[j-1]);
213 
214             //coordenadas del centroide
215             centx=(int)m10(blob);
216             centy=(int)m01(blob);
217 
218             //enmarca cada blob en un recuadro amarillo
219             cvRectangle(dst,cvPoint(p1.x-3,p1.y-3),cvPoint(p2.x
                    +3,p2.y+3),CV_RGB(255,255,0),2,8,0);
220         }
221 
222         cvNamedWindow("Figuras Encontradas",CV_WINDOW_AUTOSIZE);
                //crea ventana llamada Blobs
223         cvShowImage("Figuras Encontradas",dst);
                                //muestra en la ventana la imagen
                binarizada
224         cvMoveWindow("Figuras Encontradas",180,80);
225         cvSaveImage("nombresBlob.jpg",dst);
226         cvWaitKey(0);
```

```
227         cvDestroyWindow("Figuras Encontradas");
228 
229      return blobnum;
230     }
231 
232 
233     int entrenamiento(int num, float in[], float inP[]){
234 
235     printf("Datos de Entrenamiento.....\n");
236 
237             //declaradas la matriz out con la salida deseada
238             //Q=cuadrado, T=triangulo,C=Circulo, R=Rectangulo
239     float out[]={     0,1,0,0,    //0
240                       1,0,0,0,    //1
241                       0,0,1,0,    //2
242                       0,0,0,1};   //3
243 
244 
245            //3 capas de entrada= a,p,c
246            //2 capas ocultas c/u de 13
247            //4 capas de salida= c,t,c,r
248             //layer= capas o neuronas en la red
249             int layer[]={3,13,13,4};
250 
251             //se define la arquitectura de red
252             //definimos la entrada input de la red con CvMat
253             //cvCreateMat(num,3,CV_32FC1) regresa una matriz de
                   punto flotante de los vectores de entrada x un
                   vector fila
254             //       los parametros de entrada son (num=filas=#
                   blobs, col=3=a.p.c,CV_32FC1=tipodatoqueva a la
                   matriz)
255             CvMat *input= cvCreateMat(num,3,CV_32FC1);
256             //definimos la salida target
257             CvMat *target= cvCreateMat(num,4,CV_32FC1);
258             //definimos las neuronas en cada capa incluida la
                   entrada y salida(layersize) de la red
259             CvMat *layersize= cvCreateMat(1,4,CV_32SC1);
260 
261             //se copian los valores a la arquitectura de red
262             cvInitMatHeader(input,num,3,CV_32FC1,in);
263             cvInitMatHeader(target,num,4,CV_32FC1,out);
264             cvInitMatHeader(layersize,1,4,CV_32SC1,layer);
265 
266             //las neuronas se crean a partir de la clase
                   CvANN_MLP
267             //que permite usa r la funcion net(layersize,
                   CvANN_MLP::SIGMOID_SYM,1,1);
268             //donde 1 parametro vector q especifica el numero de
```

```
            neuronas en cada campo
269         //2 parametro funcion de activacion para cada neurona
             funcion simetrica sigmoidal
270         //3 y 4 parametro son libres los cuales son los
            valores que se le da a alfa y beta de la fun
            sigmoid
271         //la red la cual es de tipo perceptron multicapa
272
273         //se asignan pesos a la RN de acuerdo a la funcion de
             transferencia Sigmoide
274         CvANN_MLP net(layersize,CvANN_MLP::SIGMOID_SYM,1,1);
275
276         //ITERACIONES REALIZADAS EN EL ENTRENAMIENTO
277         //5000 MAXIMO NUMERO DE ITERACIONES
278         //0.001 ERROR DE OBJETIVO
279         int iter= net.train(input,target,NULL,0,
            CvANN_MLP_TrainParams(cvTermCriteria(
            CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,5000,0.001),
            CvANN_MLP_TrainParams::BACKPROP,0.2,0.1));
280
281         printf("Entrenamiento de Red: %d iteracion\n",iter);
282         printf("Fin del entrenamiento\n");
283
284         ////////////////////////////PRUEBA
            ////////////////////////
285
286         //imagen de prueba
287         binarizaPrueba("circulo2.jpg","imgBINTest.jpg");  //
            imagen para prueba de la c mara que tomo
288         num=calculos("imgBINTest.jpg",in);    //regresa el
            numero de objetos de la imagenPrueba
289
290         int i=0,j=0;
291         char text[255];  //nombre de las figuras
292         CvPoint p1,p2;
293
294         printf("Datos de prueba..................\n");
295         CvMat *ineva= cvCreateMat(num,3,CV_32FC1);
296         CvMat *output= cvCreateMat(num,4,CV_32FC1);
297         cvInitMatHeader (ineva,num,3,CV_32FC1,in);
298
299         //se predice la salida
300         net.predict(ineva,output);
301         printf("Resultados de la prueba\n");
302         printf("Salidas obtenidas\n");
303
304         float vmax;
305         int jmax=0;
306         for(i=0; i<num; i++)
```

```
307             {
308                 //muestra la salida real obtenida de la
                        prediccion
309                 printf("Blob %d: < %.2f, %.2f, %.2f, %.2f>\n",i,
310                         CV_MAT_ELEM(*output, float, i ,0),
311                         CV_MAT_ELEM(*output, float, i ,1),
312                         CV_MAT_ELEM(*output, float, i ,2),
313                         CV_MAT_ELEM(*output, float, i ,3));
314                 //se aproxima la salida obtenida, guarda la
                        posicion del mayor valor de salida x cada
                        region
315                 vmax=CV_MAT_ELEM(*output,float,i,0);jmax=0;
316                 for(j=1; j<4; j++){
317                     if(vmax<CV_MAT_ELEM(*output,float,i,j)){
318                         vmax=CV_MAT_ELEM(*output,float,i,j);
319                         jmax=j;
320                     }
321                 }
322                 j=0;
323                 //se asigna 1 al maximo valor de salida
324                 CV_MAT_ELEM(*output,int,i,jmax)=1;
325                 //los demas valores se hacen 0
326                 while(j<4){
327                     if(j!=jmax)
328                         CV_MAT_ELEM(*output,int,i,j)=0;
329                         j++;
330                 }
331             }
332
333             printf("Desicion por Aproximacion.............\n");
334             int out0,out1,out2,out3;
335             for (i=0; i<num; i++){
336                 //salidas enteras de cada region
337                 out0=CV_MAT_ELEM(*output,int,i,0);
338                 out1=CV_MAT_ELEM(*output,int,i,1);
339                 out2=CV_MAT_ELEM(*output,int,i,2);
340                 out3=CV_MAT_ELEM(*output,int,i,3);
341                 //coordenadas limites en  cada region
342                 blob=blobs.GetBlob(i);
343                 p1.x=(int)blob.MinX();
344                 p1.y=(int)blob.MinY();
345                 p2.x=(int)blob.MaxX();
346                 p2.y=(int)blob.MaxY();
347
348                 sprintf(text,"Rectangulo");
349                 if(out0) sprintf(text,"Cuadrado");
350                 if(out1) sprintf(text,"Triangulo");
351                 if(out2) sprintf(text,"Circulo");
352
```

```
353                 //muestra resultados y desicion tomada
354                 printf("Blob %d-> (%6d, %4d, %.2f) (%d, %d, %d, %
                       d)\tFigura Encontrada: ",i,
355                       CV_MAT_ELEM(*ineva,int,i,0),
356                       CV_MAT_ELEM(*ineva,int,i,1),
357                       CV_MAT_ELEM(*ineva,float,i,2),
358                       out0,out1,out2,out3);
359                 printf(text);
360                 printf("\n");
361
362                 cvPutText(dst,text,cvPoint(p1.x,p1.y-5),&dfont,
                       CV_RGB(100,200,200));
363
364             }
365             ////////////////////////////fin de la prueba
                   ////////////////////////////
366
367             cvDestroyAllWindows();
368             cvNamedWindow("Resultado",CV_WINDOW_AUTOSIZE);
369             cvShowImage("Resultado",dst);
370             cvMoveWindow("Resultado",200,80);
371             cvSaveImage("resultado.jpg",dst);
372
373             cvWaitKey(0);
374             cvReleaseImage(&image);
375             cvReleaseImage(&gray);
376             cvReleaseImage(&dst);
377             cvDestroyAllWindows();
378
379             int salout0, salout1, salout2, salout3;
380
381             if (out0 == 1){//cuadrado
382                 salout0=0;
383                 return salout0;
384             }
385
386             if (out1 == 1){//triangulo
387                 salout1=1;
388                 return salout1;
389             }
390
391             if (out2 == 1){//circulo
392                 salout2=2;
393                 return salout2;
394            }
395             if (out3 == 1){//rectangulo
396                 salout3=3;
397                 return salout3;
398             }
```

```
399
400 }
401
402 void comandoScorbot(char mensaje[], HANDLE fd, int objeto)
403 {
404     DCB OldConf;
405     IplImage* imagenColor=0;
406
407           if (objeto==0){ //debe ser un cuadrado y va a la
                izquierda zona 1
408             printf("FUE UN CUADRADO MOVIENDO A ZONA 1\n");
409             imagenColor= cvLoadImage("scorbot1.jpeg");    //
                cargamos la imagen
410             cvNamedWindow("MOVIENDO A ZONA 1", 1);        //
                representamos la imagen
411             cvShowImage("MOVIENDO A ZONA 1",imagenColor); //
                mostramos la imagen
412             cvMoveWindow("MOVIENDO A ZONA 1",35,15);      //
                movemos la ventana
413             cvWaitKey(0);                                 //
                pulsamos tecla para terminar
414             cvDestroyWindow("MOVIENDO A ZONA 1");         //
                destruimos todas las ventanas
415
416             //haz los pasos para mover a zona1
417
418                 char inicializar[]="|echo\r\r\r";        //
                    inicializa el envio de comandos
419                 char speed[]="speed 50\r";               //
                    velocidad a 50%
420                 char moved1[]="moved demo1\r";           //
                    posicion arriba del objeto
421                 char open[]="open\r";                    //
                    abre la pinza
422                 char moved2[]="moved demo2\r";           //
                    toma el objeto
423                 char close[]="close\r";                  //
                    cierra la pinza
424                 char moved3[]="moved demo3\r";           //
                    alza objeto
425                 char moved4[]="moved demo4\r";           //
                    mueve a la izquierda
426                 char moved5[]="moved demo5\r";           //
                    baja objeto
427                 char open2[]="open\r";                   //
                    abre la pinza
428                 char moved6[]="moved demo6\r";           //
                    levanta brazo
429                 char moved7[]="moved demo7\r";           //
```

```
                    mueve a la derecha
430                 char moved8[]="moved demo8\r";          //
                    regresa a home
431                 char finalizar[]="|show par 1\r\r\r";   //
                    finaliza comunicaci n
432
433                 fd=Open_Port("dev/ttyUSB0");            //
                    Abre el puerto serie fd=Open_Port("/dev/
                    ttyUSB0");
434                 OldConf=Get_Configure_Port(fd);         //
                    Guardo la configuraci n del Puerto
435                 Configure_Port(fd,B9600,"8N1");         //
                    Configuro el puerto serie
436                 IO_Blocking(fd,FALSE);                  //
                    Seleccionamos lectura no bloqueante
437                 escrituraLecturaScorbot(inicializar,fd);//Se
                    llama a las funciones que escribe y lee
                    del puerto serial
438                 escrituraLecturaScorbot(speed,fd);
439                 escrituraLecturaScorbot(moved1,fd);
440                 escrituraLecturaScorbot(open,fd);
441                 escrituraLecturaScorbot(moved2,fd);
442                 escrituraLecturaScorbot(close,fd);
443                 escrituraLecturaScorbot(moved3,fd);
444                 escrituraLecturaScorbot(moved4,fd);
445                 escrituraLecturaScorbot(moved5,fd);
446                 escrituraLecturaScorbot(open2,fd);
447                 escrituraLecturaScorbot(moved6,fd);
448                 escrituraLecturaScorbot(moved7,fd);
449                 escrituraLecturaScorbot(moved8,fd);
450                 escrituraLecturaScorbot(finalizar,fd);
451                 Set_Configure_Port(fd,OldConf);         //
                    Restituyo la antigua configuraci n del
                    puerto
452                 Close_Port(fd);                         //
                    Cierro el puerto serie
453
454                 return;
455
456         }
457
458
459         if (objeto==1){ //debe ser un triangulo y va al
                frente zona 2
460              printf("FUE UN TRIANGULO MOVIENDO A ZONA 2\n");
461              imagenColor= cvLoadImage("scorbot2.jpeg");    //
                    cargamos la imagen
462              cvNamedWindow("MOVIENDO A ZONA 2", 1);        //
                    representamos la imagen
```

```
463                 cvShowImage("MOVIENDO A ZONA 2",imagenColor); //
                        mostramos la imagen
464                 cvMoveWindow("MOVIENDO A ZONA 2",515,215);
                        //movemos la ventana
465                 cvWaitKey(0);                                   //
                        pulsamos tecla para terminar
466                 cvDestroyWindow("MOVIENDO A ZONA 2");           //
                        destruimos todas las ventanas
467
468                 //haz los pasos para mover a zona2
469
470                     char inicializar[]="|echo\r\r\r";           //
                            inicializa el envio de comandos
471                     char speed[]="speed 50\r";                  //
                            velocidad a 50%
472                     char moved1[]="moved demo1\r";              //
                            posicion arriba del objeto
473                     char open[]="open\r";                       //
                            abre la pinza
474                     char moved2[]="moved demo2\r";              //
                            toma el objeto
475                     char close[]="close\r";                     //
                            cierra la pinza
476                     char moved3[]="moved demo3\r";              //
                            alza objeto
477                     char moved4[]="moved demo4\r";              //
                            mueve al frente
478                     char moved5[]="moved demo5\r";              //
                            baja objeto
479                     char open2[]="open\r";                      //
                            abre la pinza
480                     char moved6[]="moved demo6\r";              //
                            levanta brazo
481                     char moved7[]="moved demo7\r";              //
                            mueve atras
482                     char moved8[]="moved demo8\r";              //
                            regresa a home
483                     char finalizar[]="|show par 1\r\r\r";       //
                            finaliza comunicaci n
484
485                     fd=Open_Port("dev/ttyUSB0");                //
                            Abre el puerto serie fd=Open_Port("/dev/
                            ttyUSB0");
486                     OldConf=Get_Configure_Port(fd);             //
                            Guardo la configuraci n del Puerto
487                     Configure_Port(fd,B9600,"8N1");             //
                            Configuro el puerto serie
488                     IO_Blocking(fd,FALSE);                      //
                            Seleccionamos lectura no bloqueante
```

```
489                     escrituraLecturaScorbot(inicializar,fd);//Se
                           llama a las funciones que escribe y lee
                           del puerto serial
490                     escrituraLecturaScorbot(speed,fd);
491                     escrituraLecturaScorbot(moved1,fd);
492                     escrituraLecturaScorbot(open,fd);
493                     escrituraLecturaScorbot(moved2,fd);
494                     escrituraLecturaScorbot(close,fd);
495                     escrituraLecturaScorbot(moved3,fd);
496                     escrituraLecturaScorbot(moved4,fd);
497                     escrituraLecturaScorbot(moved5,fd);
498                     escrituraLecturaScorbot(open2,fd);
499                     escrituraLecturaScorbot(moved6,fd);
500                     escrituraLecturaScorbot(moved7,fd);
501                     escrituraLecturaScorbot(moved8,fd);
502                     escrituraLecturaScorbot(finalizar,fd);
503                     Set_Configure_Port(fd,OldConf);          //
                           Restituyo la antigua configuraci n del
                           puerto
504                     Close_Port(fd);                          //
                           Cierro el puerto serie
505
506                     return;
507
508             }
509
510
511             if (objeto==2){ //debe ser un circulo y va a la
                    derecha zona 3
512                  printf("FUE UN CIRCULO MOVIENDO A ZONA 3\n");
513                  imagenColor= cvLoadImage("scorbot3.jpeg");
                        //cargamos la imagen
514                  cvNamedWindow("MOVIENDO A ZONA 3", 1);
                        //representamos la imagen
515                  cvShowImage("MOVIENDO A ZONA 3",imagenColor);
                        //mostramos la imagen
516                  cvMoveWindow("MOVIENDO A ZONA 3",420,200);
                              //movemos la pantalla
517                  cvWaitKey(0);
                        //pulsamos tecla para terminar
518                  cvDestroyWindow("MOVIENDO A ZONA 3");
                        //destruimos todas las ventanas
519
520                  //haz los pasos para mover a zona3
521
522                      char inicializar[]="|echo\r\r\r";        //
                           inicializa el envio de comandos
523                      char speed[]="speed 50\r";               //
                           velocidad a 50%
```

```
524                     char moved1[]="moved demo1\r";          //
                           posicion arriba del objeto
525                     char open[]="open\r";                   //
                           abre la pinza
526                     char moved2[]="moved demo2\r";          //
                           toma el objeto
527                     char close[]="close\r";                 //
                           cierra la pinza
528                     char moved3[]="moved demo3\r";          //
                           alza objeto
529                     char moved4[]="moved demo4\r";          //
                           mueve a la derecha
530                     char moved5[]="moved demo5\r";          //
                           baja objeto
531                     char open2[]="open\r";                  //
                           abre la pinza
532                     char moved6[]="moved demo6\r";          //
                           sube brazo
533                     char moved7[]="moved demo7\r";          //
                           mueve a la izquierda
534                     char moved8[]="moved demo8\r";          //
                           regresa a home
535                     char finalizar[]="|show par 1\r\r\r";   //
                           finaliza comunicaci n
536
537                     fd=Open_Port("dev/ttyUSB0");            //
                           Abre el puerto serie fd=Open_Port("/dev/
                           ttyUSB0");
538                     OldConf=Get_Configure_Port(fd);         //
                           Guardo la configuraci n del Puerto
539                     Configure_Port(fd,B9600,"8N1");         //
                           Configuro el puerto serie
540                     IO_Blocking(fd,FALSE);                  //
                           Seleccionamos lectura no bloqueante
541                     escrituraLecturaScorbot(inicializar,fd);//Se
                           llama a las funciones que escribe y lee
                           del puerto serial
542                     escrituraLecturaScorbot(speed,fd);
543                     escrituraLecturaScorbot(moved1,fd);
544                     escrituraLecturaScorbot(open,fd);
545                     escrituraLecturaScorbot(moved2,fd);
546                     escrituraLecturaScorbot(close,fd);
547                     escrituraLecturaScorbot(moved3,fd);
548                     escrituraLecturaScorbot(moved4,fd);
549                     escrituraLecturaScorbot(moved5,fd);
550                     escrituraLecturaScorbot(open2,fd);
551                     escrituraLecturaScorbot(moved6,fd);
552                     escrituraLecturaScorbot(moved7,fd);
553                     escrituraLecturaScorbot(moved8,fd);
```

```
554                     escrituraLecturaScorbot(finalizar,fd);
555                     Set_Configure_Port(fd,OldConf);          //
                            Restituyo la antigua configuraci n del
                            puerto
556                     Close_Port(fd);                          //
                            Cierro el puerto serie
557
558                     return;
559
560             }
561
562
563             if (objeto ==3){ //debe ser un rectangulo y va a
                    atras zona 4
564                  printf("FUE UN RECTANGULO MOVIENDO A ZONA 4\n");
565                  imagenColor= cvLoadImage("scorbot4.jpeg");    //
                        cargamos la imagen
566                  cvNamedWindow("MOVIENDO A ZONA 4", 1);        //
                        representamos la imagen
567                  cvShowImage("MOVIENDO A ZONA 4",imagenColor); //
                        mostramos la imagen
568                  cvMoveWindow("MOVIENDO A ZONA 4",435,205);    //
                        movemos la ventana
569                  cvWaitKey(0);                                 //
                        pulsamos tecla para terminar
570                  cvDestroyWindow("MOVIENDO A ZONA 4");         //
                        destruimos todas las ventanas
571                  //haz los pasos para mover a zona4
572
573                     char inicializar[]="|echo\r\r\r";         //
                            inicializa el envio de comandos
574                     char speed[]="speed 50\r";                //
                            velocidad a 50%
575                     char moved1[]="moved demo1\r";            //
                            posicion arriba del objeto
576                     char open[]="open\r";                     //
                            abre la pinza
577                     char moved2[]="moved demo2\r";            //
                            toma el objeto
578                     char close[]="close\r";                   //
                            cierra la pinza
579                     char moved3[]="moved demo3\r";            //
                            alza objeto
580                     char moved4[]="moved demo4\r";            //
                            mueve atras
581                     char moved5[]="moved demo5\r";            //
                            baja objeto
582                     char open2[]="open\r";                    //
                            abre la pinza
```

```
583                     char moved6[]="moved demo6\r";          //
                           levanta brazo
584                     char moved7[]="moved demo7\r";          //
                           mueve adelante
585                     char moved8[]="moved demo8\r";          //
                           regresa a home
586                     char finalizar[]="|show par 1\r\r\r";   //
                           finaliza comunicaci n
587
588                     fd=Open_Port("dev/ttyUSB0");            //
                           Abre el puerto serie fd=Open_Port("/dev/
                           ttyUSB0");
589                     OldConf=Get_Configure_Port(fd);         //
                           Guardo la configuraci n del Puerto
590                     Configure_Port(fd,B9600,"8N1");         //
                           Configuro el puerto serie
591                     IO_Blocking(fd,FALSE);                  //
                           Seleccionamos lectura no bloqueante
592                     escrituraLecturaScorbot(inicializar,fd);//Se
                           llama a las funciones que escribe y lee
                           del puerto serial
593                     escrituraLecturaScorbot(speed,fd);
594                     escrituraLecturaScorbot(moved1,fd);
595                     escrituraLecturaScorbot(open,fd);
596                     escrituraLecturaScorbot(moved2,fd);
597                     escrituraLecturaScorbot(close,fd);
598                     escrituraLecturaScorbot(moved3,fd);
599                     escrituraLecturaScorbot(moved4,fd);
600                     escrituraLecturaScorbot(moved5,fd);
601                     escrituraLecturaScorbot(open2,fd);
602                     escrituraLecturaScorbot(moved6,fd);
603                     escrituraLecturaScorbot(moved7,fd);
604                     escrituraLecturaScorbot(moved8,fd);
605                     escrituraLecturaScorbot(finalizar,fd);
606                     Set_Configure_Port(fd,OldConf);         //
                           Restituyo la antigua configuraci n del
                           puerto
607                     Close_Port(fd);                         //
                           Cierro el puerto serie
608                     printf("\n\nPresione ENTER para terminar");
609                     getchar();
610                     return;
611
612             }
613
614     }
615
616     void escrituraLecturaScorbot(char mensaje[], HANDLE fd){
617
```

```
618       char resp[100]="";
619        int n;
620        Write_Port(fd,mensaje,strlen(mensaje));    /*Escribimos
              en el puerto serie */
621        Clean_Buffer(fd);                          /* Limpiamos
              el buffer*/
622        do
623        {
624            n=Read_Port(fd,resp,100);              /* Leimos del
                  puerto*/
625            printf("resp: %s" ,resp);
626            memset(resp,'\0', 100);                /*Limpiamos la
                   cadena*/
627            if(n!=0) sleep(1000);                  /*Dormimos el
                  proceso*/
628        else
629            sleep(500);
630        }
631        while(Kbhit_Port(fd));                    /*Se ejecuta
              hasta que sean los caracteres necesarios*/
632        printf("\nPresione ENTER para continuar");
633        getchar();
634        return;
635  }
```

Bibliografía

[1] A. Rodríguez Sánchez , O. Alvarado Nava and F.J. Zaragoza Martínez, *Real-Time Reconfigurable Embedded System for Remote Monitoring and Controling*, Julio 2013. [En línea]. Disponible en: http://ieeexplore.ieee.org/xpl/articleDetails.jsptp=&arnumber=6235441.

[2] O.Alvarado Nava, *Compilación Kernel de Linux*, Julio 2013. [En línea]. Disponible en: http://ce.azc.uam.mx/profesores/oalvarado/linux/compKernel.html

[3] A. Cuschieri, T. G. Frank, J.R. Hewit, A.P. Slade, *Intelligent interface considerations for surgical assist robotics*, 1997 IASTED International conference on intelligent information systems.

[4] F.M. Díaz Cabrera, *Clasificador de objetos en banda infinita por medio de procesamiento digital de imágenes*, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2009.

[5] M.I. Castillo Espínola,*Modelado en el accionar con fricción de un brazo robótico utilizando un controlador PD más Red Neuronal*, tesis de maestría, Universidad Autónoma Metropolitana Iztapalapa, México, 2010.

[6] C.H. Flores Arteaga and L.H. Ruíz Rodríguez, *Desarrollo de interfaz para el traslado de un objeto utilizando una celda flexible dentro del plano por medio de visión artificial*, tesis de licenciatura, Universidad del Bío-Bío, Concepción, Chile, 2010.

[7] E. Martínez Peña, *Control de un Robot tipo PUMA utilizando celdas neuronales analógicas*, tesis de maestría en ciencias, Instituto Tecnológico de Ciudad Victoria, México, 2008.

[8] *Tutorial de OpenCV*, [En línea]. Disponible en: http://docs.opencv.org/doc/user_guide/user_guide.html

[9] D. López Zamarron, *Manual de Instalación de RTLinux*,Julio 2013.

[10] C. Yeow Yeoh. (5 de Noviembre de 2012). *Real-Time Reconfigurable Embedded System for Remote Monitoring and Controlling*, November 2007. [En linea]. Disponible en: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4658600.

[11] C. Martín Syd, *Scorbot ER-Vplus y Scorbot ER-IX*, Abril 2000. [En línea]. Disponible en: http://iaci.unq.edu.ar/publicaciones/archivos/InformeIACI00_01.prn.pdf