Universidad Nacional Autónoma Metropolitana Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Ingeniería en Computación

Programa de Desarrollo Profesional en Automatización

Reporte del Proyecto Terminal

Clasificación de Objetos en un Sistema de Posicionamiento Automático mediante Visión Robótica Activa

Autor:
Eric Miguel González Jiménez

Asesor
M. en C. Gerardo Aragón González

México D.F. Julio 2011

Tabla de contenido

Resumen

Se desarrolla un sistema para la clasificación y reconocimiento de objetos de posicionamiento automático mediante visión robótica activa. La clasificación y reconocimiento de objetos se realiza mediante una cámara acoplada a un robot cartesiano XYZ, y el software requerido fue implementado en C++ y MATLAB. En el desarrollo del sistema se empleó el algoritmo para visión artificial (SIFT) propuesto por David Lowe [12], y dos módulos: uno para controlar la trayectoria del robot cartesiano, y el otro para el reconocimiento de objetos. Se construyó una interfaz gráfica a nivel usuario la cual consiste de tres botones para inicializar, buscar y reconocer, y terminar.

Abstract

A system for automatic positioning object's classification and recognition is developed by means of active robot vision. The object's classification and recognition is performed by a camera coupled to a Cartesian robot XYZ, and the required software was implemented in C++ and MATLAB. Throughout the system's development, the algorithm for artificial vision proposed (SIFT) by David Lowe [12] was employed, and two modules: one to control the Cartesian robot's trajectory, and the other to recognize objects. A graphic interface, which consists of three buttons to initialize, search and recognize, and finish, was built in a user level.

Introducción

El **Programa de Desarrollo Profesional en Automatización** (PDPA) se fundó en 1992, impulsado por tres profesores titulares del departamento de Energía, enmarcado en un acuerdo de colaboración signado por la Rectoría de la Unidad Azcapotzalco y la empresa Parker Hannifin de México. Entre las actividades primordiales del PDPA se cuentan la producción de prototipos y soluciones tecnológicas para el sector productivo, desarrollados con base en acuerdos de colaboración técnica con diferentes actores económicos de nuestro país. Las actividades de desarrollo tecnológico *incluyen la participación de alumnos de* las licenciaturas de ingeniería, con el propósito de ponerlos en contacto directo con las aplicaciones de la potencia fluida, el control de movimiento y la automatización de procesos de manufactura.

En las instalaciones del PDPA existe un robot posicionamiento cartesiano (robot cartesiano) anclado en una mesa denominada XYZ, el cual fue ensamblado y puesto a punto mediante un proyecto terminal de ingeniería mecánica [1]. El robot de posicionamiento cartesiano interpreta las instrucciones dadas desde una computadora que tiene previamente instalado el software del fabricante y las transforma en pulsos o pasos de motor y cuenta con 3 drives que le permiten el movimiento sobre los ejes X-Y-Z. En [2] un programa en Visual Basic fue desarrollado para manipular el robot cartesiano para seguir trayectorias simples. En el robot cartesiano se encuentra una cámara de alta definición montada y adaptada en el eje Z del sistema de posicionamiento. Con lo que se pueden obtener imágenes dentro del área de trabajo.

El robot cartesiano constituyó la base para desarrollar este proyecto terminal cuyo objetivo general es la clasificación de objetos con un sistema de posicionamiento automático mediante el paradigma de visión activa. El proyecto consistió en varias etapas:

Construir una base de datos con las imágenes entrantes que se utilizaran como referencia para la clasificación. Extraer las características específicas de las imágenes de la base de datos tomando muestras mediante la detección de esquinas. Obtener los descriptores usando el algoritmo SIFT [12] para el conjunto de imágenes de referencia. Comparar la imagen tomada con la cámara con las existentes en la base de datos. Integrar estas etapas en un solo módulo de programación. Desarrollar otro módulo que controle al robot cartesiano. Se implementó visión activa para que el robot explore el área de trabajo y realice reconocimiento de objetos dentro del área de trabajo. Se elaboró otro módulo que calcula el desplazamiento del sujetador (gripper) respecto al centro del objeto aplicando una regresión lineal de píxeles versus pasos de motor. Los módulos fueron desarrollados con base en lenguajes de programación Visual C++ y MATLAB. Se construyó una interfaz de control la cual es muy intuitiva para poder controlar el robot cartesiano y el reconocimiento de objetos.

El reporte consta de Preliminares y tres Capítulos. En los Preliminares se hace una breve descripción de los dispositivos que componen al sistema de posicionamiento automática, del control maestro de la serie 6k4 [18] y del Software Motion Planner del propietario (Parker Hannifin). Una descripción más detallada de este sistema se encuentra en [1].

El Capítulo 1 está dedicado al reconocimiento de imágenes en la literatura. Se discuten algunos aspectos sobre la percepción, el conocimiento cognitivo, el cerebro y visión artificial para después describir lo correspondiente a procesamiento digital de imágenes, la forma en que se representa una imagen mediante mapas de bits y la forma que la computadora interpreta a un pixel. Se describe la biblioteca libre Opencv desarrollada por Intel para visión artificial la cual es útil para la comprensión del procesamiento digital de imágenes y los diversos filtros (suavizado, gaussiano, etc.).

El Capítulo 2 está dedicado exclusivamente al algoritmo fundamental de visión artificial SIFT [12] (por sus siglas en inglés corresponde a *Scale Invariant features transform*; transformada de características invariantes a la escala), creado para obtener las características locales de una imagen. Este algoritmo, es uno de los más adecuados para describir las características de una imagen para hacer coincidir (*matching*) objetos en una imagen o una escena. Las características de los objetos resultan ser invariantes a la escala y rotación de la imagen y parcialmente invariante al cambio de iluminación y el punto de vista 3D (tridimensional) de la cámara. Estas características están bien localizadas tanto en el ámbito espacial y la frecuencia, lo que reduce la probabilidad de interrupción de oclusión, desorden o ruido. Un gran número de características se pueden extraer de imágenes con el SIFT. Además, las características son muy distintivas, y eso permite que una sola característica sea correctamente mapeada con una probabilidad alta en concordancia con una base de datos de características previamente implementada. Se discuten las etapas fundamentales del SIFT y algunas de las fases de estas etapas como son la detección de extremos escala-espacio, localización de los Keypoints (puntos clave), la asignación de la orientación y el descriptor de Keypoints.

El Capítulo 3 constituye la parte medular del reporte está dedicado a la descripción del sistema desarrollado para que el robot cartesiano realice el reconocimiento de objetos. Aunque, la interfaz del software propietario es limitada ya que solamente permite desarrollar scripts con varias sentencias y ejecutar todas ellas en conjunto. La ventaja es que al instalar este software propietario también se instala la biblioteca de funciones COM6SRVR la cual puede ser integrada con cualquier lenguaje de programación que maneje objetos OLE. Por lo que es posible desarrollar aplicaciones visuales propias simplemente integrando esta librería. El primer módulo (trayectoria del robot) fue desarrollado en Visual C++ 6.0 y el segundo (reconocimiento de objetos) fue desarrollado en MATLAB. La interfaz gráfica para el usuario, que integra y oculta ambos módulos, fue simplificada a tres botones: Iniciar, Buscar y Terminar.

Finalmente, como Apéndice se incluye el código completo para el sistema de posicionamiento automático para el robot cartesiano para clasificar y hacer el reconocimiento de objetos.

Preliminares

Dispositivos que componen el sistema de posicionamiento automático

Descripción del sistema

El sistema cuenta con 3 diferentes tipos de actuadores. Cada actuador es capaz de transformar la energía eléctrica en movimiento, los actuadores reciben la instrucción de un controlador y en función de ésta realizan el movimiento. Se utilizan actuadores lineales por las ventajas ofrecidas como:

- Facilidad de montaje en cualquier posición: horizontal, vertical o inclinada.
- También por su funcionamiento independiente y autónomo de otras instalaciones.
- El movimiento se efectúa a velocidad uniforme.
- Funcionamiento silencioso.

Específicamente, los ejes X e Y utilizan actuadores sin vástago modelo ER32 (Figura 1), marca Parker. Para este modelo de actuador están disponibles tres presentaciones los que son accionados por correa, los que utilizan un mecanismo de tornillo de bolas y los de transporte de carga con apoyo ya sea por las ruedas de rodamiento de rodillos de precisión o un riel montado en el interior. El eje X utiliza un mecanismo de tornillo de bolas sin vástago. El eje Y utiliza el mismo mecanismo con la diferencia de que incluye un carro de rodillos. En el eje Z se instaló un cilindro eléctrico, impulsado con un tornillo de bolas y un módulo guía para el vástago, marca Parker serie ET.

Figura 1. Actuador lineal serie ER32

Los actuadores son impulsados por 3 motores a pasos marca Parker-Compumotor (Figura 2). Los motores a pasos tienen la característica de transformar una serie pulsos eléctricos en desplazamientos angulares. Este tipo particular de motor no recibe corriente continua ni corriente alterna como sucede con otros motores, sino una serie de pulsos que definen una secuencia previamente definida, cada vez que se aplica un pulso el motor se desplaza un paso y queda fijo en esta posición de esta forma es posible controlar el desplazamiento del motor. Además si se varía la frecuencia en que se aplican los pulsos también se varía la velocidad con que se mueve el motor, lo que permite realizar un control de velocidad. Por último al invertir la secuencia de los pulsos de alimentación aplicados a las bobinas, se realiza una inversión en el sentido de giro del motor.

Un inconveniente que presentan los motores a pasos es velocidad angular limitada. Dicha limitación surge al momento de realizar un paso, el motor requiere un tiempo para alcanzar la posición de equilibrio. Si dicho tiempo no se respeta (esto ocurriría si la frecuencia de los pulsos es demasiado elevada) el motor puede no encontrar nunca esa posición de equilibrio y perderíamos el control sobre éste (se mueve en forma de vaivén, no se mueve, o incluso se mueve en sentido contrario al deseado).

Figura 2. Motor a pasos

Estos motores tienen la característica de tener alta precisión respecto a la posición ya que dependiendo los impulsos se puede tener un movimiento que va desde 1.8° hasta 90° de acuerdo a la secuencia que se indique, esta característica resulta útil en el sistema al demandar una exactitud milimétrica al momento de cambiar la posición .

Cada uno de los motores a pasos empleados en el sistema se controla con un drive serie E marca Parker, esta unidad de micro paso opera directamente con 120V de corriente alterna por lo que no es necesario un transformador para su uso. El drive recibe las señales de paso y dirección de la unidad por parte del controlador maestro 6k4. Para cada paso del pulso que recibe el drive conmuta el motor para incrementar la posición del rotor (Figura 3).

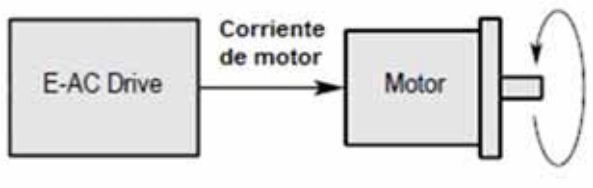


Figura 3. Pulsos enviados al motor a pasos incrementan la posición del rotor.

Todos los motores paso a paso están sujetos a la resonancia. La unidad de E-AC tiene un circuito interno para evitar este problema. Este es un circuito de propósito general que proporciona amortiguación agresiva y efectiva. La anti-resonancia se puede desactivar con un interruptor DIP ubicado en la parte posterior de la unidad.

El controlador maestro serie 6k4 (Figura 4) se encarga de traducir las instrucciones del usuario transmitidas a través de una computadora, de un controlador programable o de algún dispositivo de entrada-salida que se encuentre conectado en el controlador. La información es redirigida a los dispositivos que tiene bajo su mando. La comunicación se hace mediante el servidor de comunicación 6k (COM6SRVR). Es un servicio de automatización tipo OLE de 32 bits para facilitar la comunicación entre los controladores 6k y las aplicaciones software.

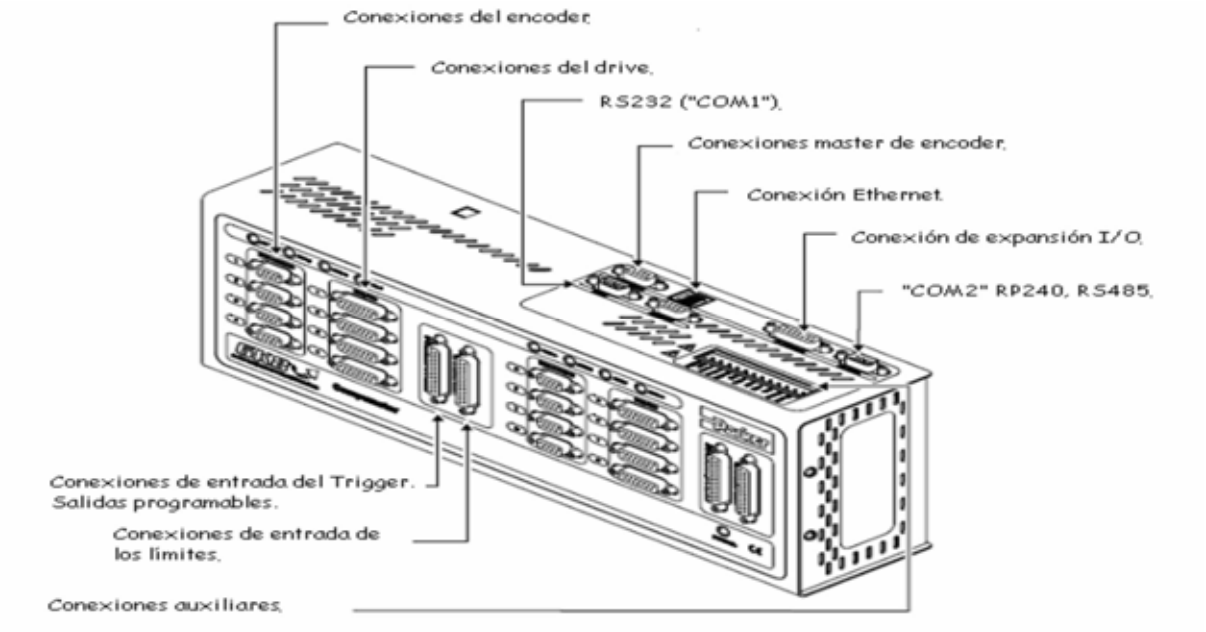


Figura 4. Conexiones de los dispositivos 6k

Como se puede observar en la Figura 4, en el controlador 6k hay dos métodos de conexión diferentes para realizar la conexión con un dispositivo de mando, se cuenta con una entrada Ethernet para conectar mediante un ruteador (router) para realizar múltiples conexiones mediante una red LAN. Otra forma de conexión se realiza mediante un cable cruzado de

esta forma es posible conectar el dispositivo directamente a una computadora en una red PTP (peer to peer, por sus siglas en inglés). La segunda forma de conexión es mediante los puertos COM para realizar la conexión con la computadora o controlador programable. Se cuentan con dos puertos para conectar el mismo número de dispositivos. El controlador 6k4 se encarga de transmitir el número de pasos, la frecuencia que se aplicará a los motores a pasos y el sentido que estos tienen a los 3 drives serie E que controlan los ejes coordenados del robot cartesiano. La conexión del controlador maestro con los drives se ilustra en la Figura 5.

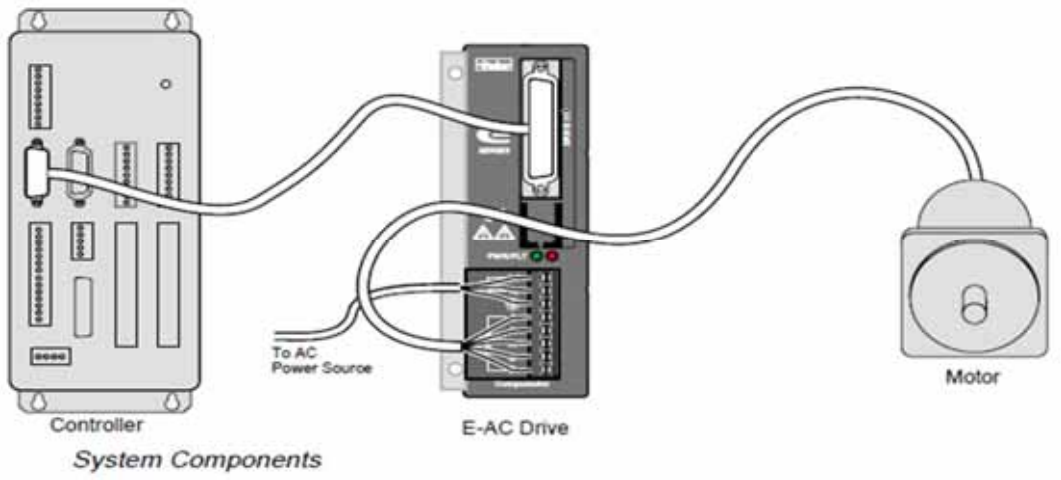


Figura 5. Conexión entre los dispositivos que efectúan el movimiento

El controlador 6k4 cuenta con dos conexiones de propósito general en su parte frontal para expandir el sistema, con base en dispositivos de entrada y salida, de posicionamiento automático y cuenta con un dispositivo que ofrece entradas para sensores de posicionamiento. Este dispositivo se denominaVM25 por el fabricante. Este dispositivo ofrece terminales con tornillo para conexiones de E / S en los conectores de 25 pines, los cuales pueden ser salida de Triggers y conectores de límite de carrera. El VM25 incluye un cable de 2 metros de altura que proporciona una fácil conexión entre el VM25 y el conector del 6K4 de 25 pines (Figura 6).

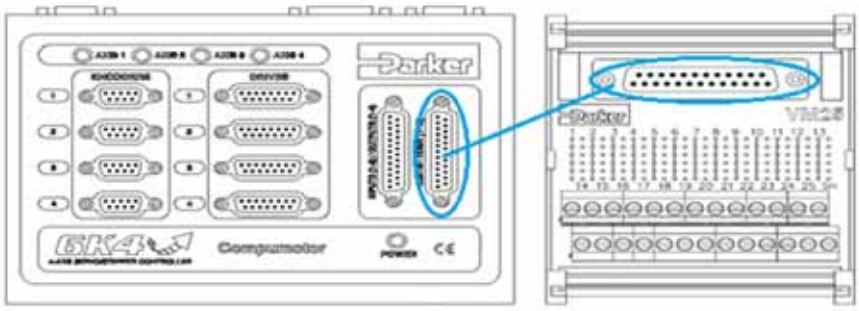


Figura 6. Panel de conexiones del controlador 6k4 con el dispositivo VM25.

Como se mencionó anteriormente el módulo VM25 sirve para agregar y controlar señales entrada/salida con el fin de instalar nuevos dispositivos en el sistema. Para establecer el sistema de posicionamiento se emplearon sensores de posición inicial y límite de carrera, cada uno de los sensores se conecta en este dispositivo. Los cables de los sensores son blindados, con forro aislante amarillo, marca Ballffur. En el extremo de estos cables aparecen tres alambres con forro aislante azul, café y negro. Los alambres color negro de cada sensor se conectan en los puertos numerados del módulo VM25.

El sistema cuenta con 9 sensores tipo efecto Hall (Hall -effect) para establecer la posición (Figura 7). Este tipo de sensores varían su salida de tensión en respuesta a un campo magnético. Los sensores de efecto Hall se utilizan para la conmutación de la proximidad, el posicionamiento, la detección de velocidad y las aplicaciones reales de detección. Se utilizan tres por cada eje, de los cuales dos son para indicar los límites positivo y negativo. Y un tercer sensor establece la posición central (Home). Los sensores que especifican el límite positivo y negativo se colocan arbitrariamente debido a que la configuración se realiza mediante la interfaz de Motion Planner.

Figura 7. Sensor tipo efecto Hall.

Los sensores indican la posición permitida de cada eje, con el fin de establecer el área en la cual se pueden desplazar y así evitar que los ejes impacten con las barras que los sostienen. Con los sensores el sistema de control recibe una retroalimentación en tiempo real.

También, se cuenta con una fuente de alimentación (Figura 8) marca SOLA/HEVI-DUTY tipo SDN 2.5-24-100P que proporciona energía al controlador maestro y a los sensores de posición; ya que estos dispositivos requieren alimentación eléctrica de 24 V de corriente directa, a partir de una fuente externa. La máxima corriente que soporta es 2.5 Ampere, con un voltaje ajustable entre 24 y 28 V. En el controlador como en los sensores hay que efectuar tres conexiones, para los polos positivo y negativo y el neutro

Figura 8. Fuente de alimentación.

Descripción del entorno de comunicación

Para llevar a cabo la comunicación se cuenta con software propietario (Motion Planner) desarrollado por Parker con el cual se establece la comunicación entre sus dispositivos y la computadora. Cuando se instala el software también se instala el servidor de comunicación (COM6SRVR.EXE). Este servidor es el encargado de la comunicación entre los dispositivos. Está basada en comunicación de objetos OLE (Object Linking and Embedding; por sus siglas en inglés). OLE proporciona un estándar consistente que permite a los objetos, aplicaciones y componentes ActiveX, comunicarse entre sí con la finalidad de usar código de otros lenguajes. Los objetos no requieren tener conocimiento anticipado de los objetos con los que se va a comunicar, incluso ni su código necesita estar escrito en un mismo lenguaje.

El servidor de comunicación es compatible con cualquier aplicación software o programa desarrollado que pueda utilizar componentes OLE automatizados incluyendo:

- Visual Basic
- Visual C++
- Delphi
- Paquetes software que soportan Microsoft Component Object Model (COM):
    - Wonderware Factory Suite 2000
    - National Instruments LabVIEW

Debido a la compatibilidad del servidor de comunicación no se requiere el software incluido para establecer la comunicación y así poder desarrollar aplicaciones propias mediante las bibliotecas existentes. Hay diversas opciones de conexión con la computadora. Por ejemplo, se puede realizar mediante cable cruzado Ethernet 10 base T. De esta forma es posible hacer la conexión directa desde la computadora hacia el dispositivo 6k4. Otra manera de hacerlo de la forma punto a punto es mediante un cable RS-232 si se cuenta con una entrada de este tipo en la computadora. Para la conexión mediante un cable RS-232 se necesita especificar el puerto PC COM en el que se conectará.

La forma recomendada para establecer la comunicación es con base en una red LAN con un router. De esta manera tenemos la posibilidad de conectar más de una computadora con el controlador maestro 6k4 (Diagrama 1).

Para iniciar la comunicación con una aplicación simplemente hay que hacer una solicitud de conexión para cada controlador 6k4. De esta manera, se puede alimentar la información desde una conexión a múltiples aplicaciones cliente. Esto significa por ejemplo que hay una aplicación en Visual Basic y otra en Motion Planner que pueden estar conectados a al mismo tiempo al controlador 6k4. Las dos terminales recibirán las mismas respuestas provenientes del controlador.

El Diagrama 1 ilustra las múltiples conexiones que se pueden realizar con el 6k4. Para este tipo de conexión se necesita especificar la dirección IP del controlador. El controlador por defecto tiene la dirección IP 192.168.10.30. Si existe conflicto con otro dispositivo dentro de la red se recomienda cambiar la dirección con el comando NTADDR. El servidor de comunicación soporta hasta 2 conexiones RS-232 e ilimitadas conexiones Ethernet.

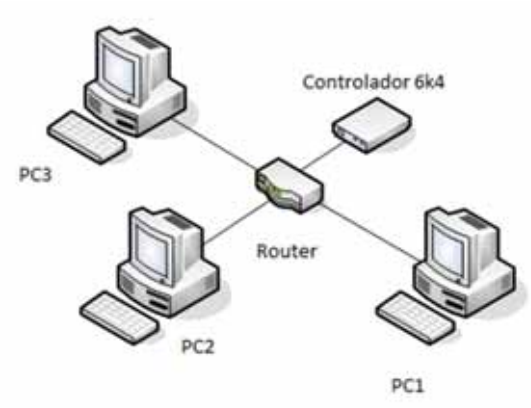


Diagrama 1. Configuración LAN entre PC´s y controlador 6k4

Descripción del principio de Movimiento

Con la conexión del controlador maestro establecida con al menos una computadora se puede controlar la trayectoria del robot mediante el comando GO100 el cual mueve el eje X con la distancia especificada en centímetros. De esta forma se podrán describir la trayectoria y jerarquías del robot cartesiano. Para iniciar la conexión se necesita la petición de conexión desde la computadora hacia el controlador 6k4 para describir la trayectoria del robot cuando se teclea un comando en la interfaz de Motion Planner. Al inicializar este programa automáticamente hace la petición de conexión con lo que se inicializa el servidor de comunicación, siempre y cuando la comunicación este establecida. Se despliega la pantalla principal de la Figura 9 sin marcar ningún error. Se habilita el drive que controla al eje X con el comando DRIVE100 esto hace que el drive se encuentre en estado de espera para recibir instrucciones. Si se desea se puede establecer la aceleración y la velocidad con los comandos A10 y V5 para establecer, por ejemplo, una a aceleración de 10 cm/s^2 y velocidad de 5 cm/s.

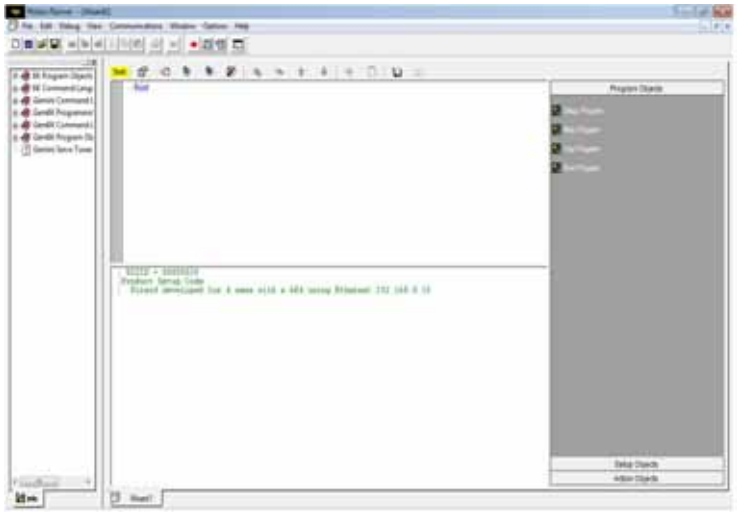


Figura 9. Pantalla inicial Motion Planner

Los comandos descritos solo permiten establecer las variables que determinan el movimiento; no son útiles para la comunicación entre la computadora y el robot cartesiano. En este momento el sistema está listo para enviar la solicitud de movimiento. Enseguida se puede ingresar el comando que cambia de posición al robot a lo largo del eje X. Esto se realiza desde la terminal, en donde está instalado el Motion Planner, con el comando GO100 (Figura 10).

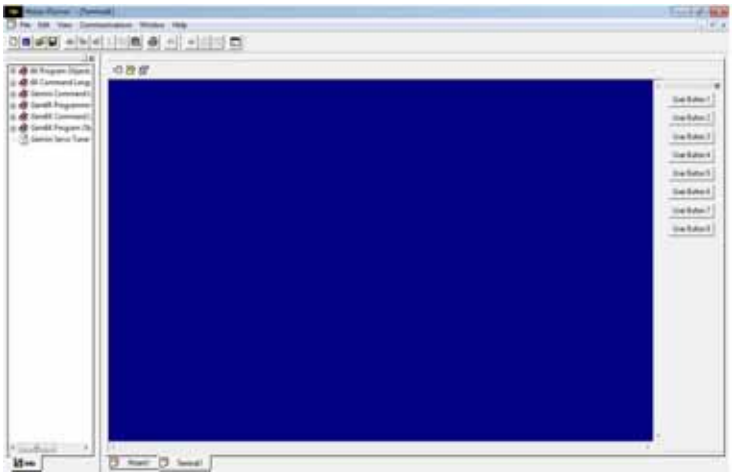


Figura 10. Terminal de comandos de Motion Planner

El comando GO001 se envía a través de la conexión que se eligió (Ethernet o RS-232) al controlador maestro, el controlador maestro codifica la instrucción y la envía al dispositivo correspondiente; en este caso la instrucción se envía al drive E que controla al eje X. Cada uno de los drives se encarga de decodificar la señal analógica y la transforma en pulsos eléctricos que son enviados a los motores a pasos. Los motores al recibir el pulso eléctrico realizan el movimiento angular de acuerdo a la cantidad de pulsos recibidos. Al girar el eje de los motores activan a los actuadores y el robot se desplaza en el eje en forma precisa en función del número de motores a pasos con el cual el sistema fue calibrado. Con los sensores incorporados en el sistema se realiza una retroalimentación de la posición. De esta forma se tiene la ubicación del robot en todo momento.

Después del movimiento del robot, los sensores envían la información a los drives E. Estos a su vez envían esta información al controlador maestro. El controlador maestro procesa esta información. De esta forma, si el usuario realiza un movimiento del robot fuera de rango, el controlador despliega un mensaje de error al usuario. El Diagrama 2 presenta la jerarquía y como se realiza la instrucción de movimiento.

Finalmente, una descripción más detallada del movimiento del robot y de los comandos del Motion Planner puede verse en [1].

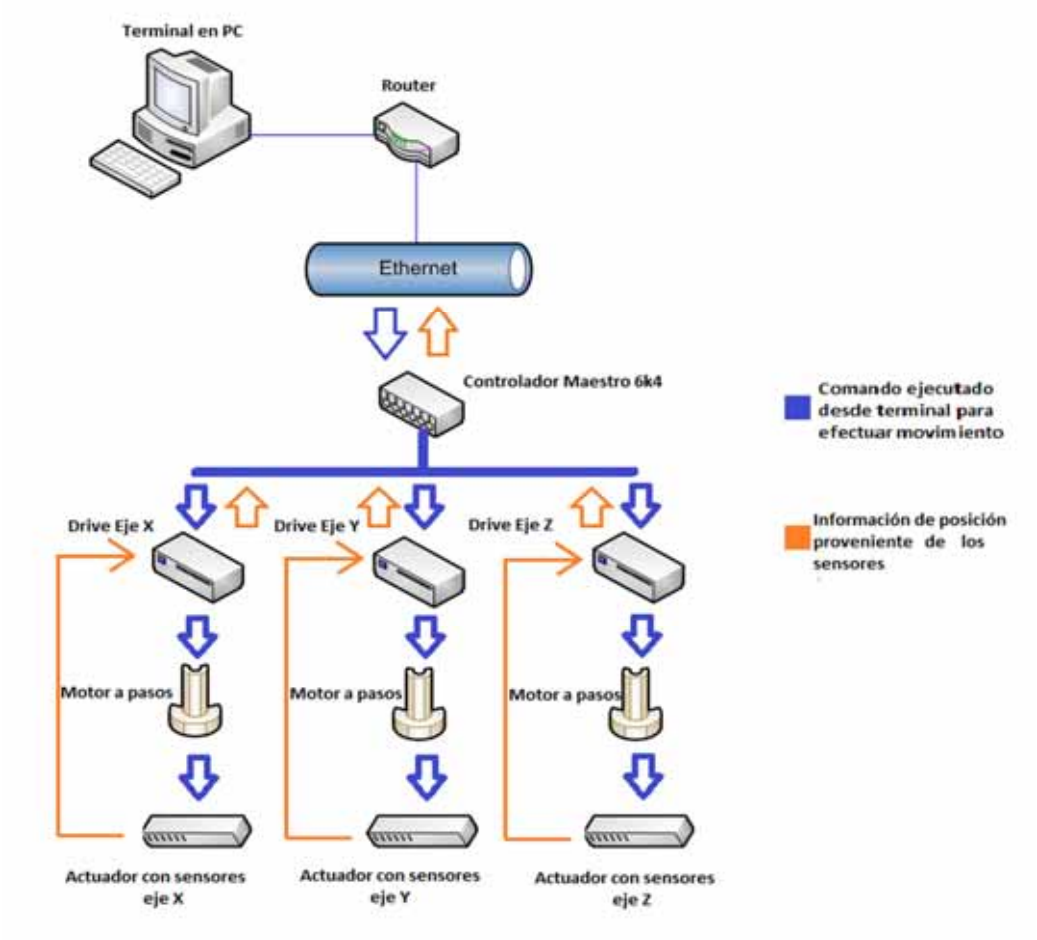


Diagrama 2. Flujo de datos al momento de ejecutar un comando de movimiento

Capítulo 1

Reconocimiento de Imágenes

Concepto de percepción

Desde el nacimiento los mamíferos incluyendo el ser humano reciben información del mundo que los rodea a través de los sentidos. Esta forma natural es un componente del conocimiento en donde el sujeto aplica interactuar con el mundo con el objetivo de percibirlo. No hay que confundir la sensación con el proceso de percepción. La sensación son estímulos que recibimos a través de los sentidos, en cuanto que la percepción es la interpretación, organización y recreación de esos estímulos procesados para tener una idea del mundo exterior.

Es así como el ser humano adquiere conciencia sobre sí mismo y sobre el mundo que los rodea. Identificar la realidad por las impresiones que se producen mediante los sentidos es una evidencia de la misteriosa perfección de la mente humana.

La psicología explica la diferencia entre las sensaciones recibidas y la realidad del mundo real, aunque están implicadas otras muchas ciencias, como la geometría, la física o la biología.

Al acotar la percepción solo al sentido de la vista se podría decir que la percepción visual es el estímulo de la luz registrada por los ojos. Este acto repetitivo y cotidiano que se realiza de forma automatizada es complejo. Prácticamente es el mismo proceso en cualquier individuo de vista sana, sin importar pigmentación o diferencias entre los órganos visuales.

La luz proyecta imágenes en la retina sensible a la luz, donde se detecta y se transmite una señal correspondiente a través del nervio óptico. El ojo por lo general es aproximadamente esférico, lleno de una sustancia transparente gelatinosa, que rellena el espacio comprendido entre la retina, el gelatina transparente, que se encuentra situado en el espacio existente entre el cristalino y la córnea transparente, cuya función es la de controlar el estado óptimo de la presión, con un lente de enfoque llamado cristalino y, a menudo, un músculo llamado iris que regula cuánta luz entra y sale del ojo [3].

Cabe mencionar que el ojo solo es el encargado de captar la luz de nuestro entorno, el cerebro es el intérprete de toda la información y se encarga de completar el proceso de visión.

Conocimiento Cognitivo

La psicología cognitiva tiene a su cargo el estudio de lo cognitivo, es decir de los procesos mentales implicados en el conocimiento, el objeto es conocer desde los mecanismos básicos hasta la formación de conceptos y razonamiento lógico. Esta rama surge del interés de saber cómo las personas entienden el mundo en el que viven y se abordan las cuestiones de cómo los seres humanos toman la información sensorial entrante y la transforman, sintetizan, elaboran, almacenan, recuperan y finalmente hacen uso de ellas [4].

Para entender mejor la forma en el que nuestro cerebro interpreta las imágenes capturadas mediante la luz que entra a través de nuestros ojos tenemos que tener conocimiento de dos conceptos, los cuales nos ayudaran a comprender como se reconocen los objetos de una forma sistemática, estos son la memoria y el aprendizaje.

La memoria es una función del cerebro con la cual es posible codificar la información proveniente del exterior, almacenarla dependiendo de su importancia para después de cierto tiempo tener la opción de recuperarla. Esto se debe a las conexiones sinápticas entre las neuronas, lo que crea una gran conexión de redes neuronales. La memoria permite retener experiencias pasadas y dependiendo del tiempo se clasifica en: memoria a corto plazo, memoria a mediano plazo y memoria a largo plazo.

Se denomina aprendizaje al proceso de adquisición de conocimientos, habilidades, valores y actitudes. Esto se logra mediante la enseñanza, el estudio o la experiencia. El proceso fundamental del aprendizaje es la imitación, este proceso observado implica tiempo, espacio, habilidades y otros recursos. Esta capacidad no es exclusiva de la especie humana, aunque es un factor dentro de la rama evolutiva.

El cerebro, cómo se perciben las cosas

Anteriormente se había mencionado la importancia de la vista, pero el encargado de realizar la codificación del mundo real es el cerebro. Cabe mencionar que con cada nueva experiencia las neuronas entablan nuevas conexiones, cada sistema del cuerpo es complejo pero solo hay uno que percibe todo lo que hacemos, nuestro cerebro guía, vigila y da órdenes definiendo lo que somos. Todo esto a través de una autopista de información hecha de nervios que se expanden en nuestro cuerpo.

El cerebro va teniendo en cuenta cada parte del cuerpo (ojos, oídos, piel, huesos, corazón, músculos, etc.) mediante 100 mil millones de células especializadas llamadas neuronas transmitiendo millones de señales eléctricas y químicas a 320 km/h.

La información visual sigue una regla compleja: los nervios ópticos de los dos ojos se unen en un punto llamado el quiasma óptico, y la mitad de las fibras de cada nervio se separan para unirse a la otra. El resultado es que las conexiones de la mitad izquierda de la retina en ambos ojos, van hacia el lado izquierdo del cerebro, mientras que las conexiones de la mitad derecha de la retina van hacia el lado derecho del cerebro [5].

Debido a que cada mitad de la retina recibe la luz procedente de la mitad opuesta del campo visual, la consecuencia funcional es que la información visual desde el lado izquierdo del mundo va al lado derecho del cerebro, y viceversa. Así, el lado derecho del cerebro recibe información somato-sensorial del lado izquierdo del cuerpo, e información visual del lado izquierdo del campo visual, una disposición que, presumiblemente, ayuda a la coordinación viso-motora [5].

El proceso en el cual las imágenes tienen sentido es el siguiente: inicia cuando la luz que entra es convertida en electricidad por los bastones y conos. La información de la luz que entra es volteada cabeza abajo, después esta información se fragmenta para seguir su camino a la corteza visual primaria, esta se ubica en la parte posterior del cerebro. Esta región filtra y codifica la información, después la información codificada es dividida en porciones (alrededor de 32 locaciones) para un procesamiento más fino.

Las neuronas empiezan a ordenar la escena aun entremezclada, conforme continua el proceso miles de tareas a la velocidad de la luz son realizadas dentro del cerebro, se trata de reconocer contornos redondeados, para hallar en la mezcolanza incomprensible algo o alguien quien reconozcamos. Todos estos fragmentos visuales son procesados y codificados por las neuronas asignadas, y cuando todo es ensamblado aparece una imagen conocida. Los expertos especulan que el cerebro construye sobre lo que aprende mientras procesa patrones simples que le permiten percibir formas de una complejidad cada vez mayor u otras formas de información [6].

Visión Artificial

Es un subcampo de la inteligencia artificial que trata de emular la capacidad que tienen algunos seres vivos para ver una escena y ser capaces de entenderla. Es una disciplina compleja que involucra interdisciplinariamente a la Física, Matemática, Ingenierías como son eléctrica y computación. Es una rama relativamente nueva de la computación, pero el continuo desarrollo de algoritmos, funciones, métodos y aplicaciones hacen de la visión por computadora una ciencia en constante evolución. Desde la aparición de la primer maquina capaz de realizar procesamientos matemáticos hasta la actualidad se han tenido avances significativos en cuestión de desarrollo tecnológico.

La tecnología crece a un ritmo acelerado que la humanidad ha tenido que adaptarse rápidamente a estos cambios, puede ser esta adaptabilidad la que nos lleve a dejar de lado lo complejo de que puede ser el desarrollo de una innovación tecnológica. Nos acostumbramos rápido a las cosas nuevas que no distinguimos el alcance de algunas ciencias. La visión artificial tiene varios aportes en la industria, la medicina, educación y entretenimiento, a continuación se mencionan algunas aplicaciones de la visión artificial.

- Detector de esquinas
- Reconocimiento de objetos
- Mapear imágenes para hacerlas coincidir
- Seguimiento de un objeto en una secuencia de imágenes
- Mapeo de una escena para generar objetos tridimensionales
- Detector de patrones
- Conteo de coincidencias

En la actualidad la visión artificial juega un papel importante dentro de la sociedad ya sea con entretenimiento mediante un videojuego en el cual se necesite traducir los movimientos del jugador hasta aplicaciones médicas capaces de detectar en radiografías algún tipo de daño o enfermedad. No cabe duda que con el crecimiento constante de esta disciplina la humanidad tendrá grandes beneficios a futuro [7].

Procesamiento Digital de Imágenes

El procesamiento digital de imágenes (PDI) es un conjunto de técnicas aplicadas a una imagen, la finalidad es obtener una imagen de salida la cual pueda tener uno o varios procesos. Esto se hace con el objetivo de mejorar la imagen o prepararla para obtener información.

El procesamiento de imágenes es relativamente nuevo ya que aparece tardíamente en la historia de la computación. Esto se debe principalmente a que no existía el hardware apropiado para poder hacerlo y los procesos gráficos son siempre una limitante aunque con avances significativos recientemente. Aparece por el hecho de imitar y utilizar algunas características de los seres vivos como apoyo para resolver problemas.

En el PDI se pueden distinguir tres etapas principales:

1. Adquisición de la imagen.
2. Procesamiento de la imagen.
3. Presentación de la imagen resultante

Estas etapas están presentes en el proceso de visión de los seres humanos. Desde que la luz reflejada es percibida por los ojos, la cual por medio de los conos y bastones se envía mediante impulsos eléctricos al cerebro para procesarla y al final del proceso obtener información de la imagen resultante. Este proceso imitado con una computadora resulta ser demasiado complejo ya que nuestro dispositivo de captura depende de muchas variables.

El ojo humano tiene la característica de regular la luz al momento de enfocar un objeto, desgraciadamente al tratar de capturar una imagen con un dispositivo digital estamos limitados con la resolución y la intensidad natural de la luz en ese instante de tiempo, estos factores repercuten en la calidad de la imagen y dificultan el reconocimiento de objetos dentro de la escena.

El procesamiento de la imagen consiste en eliminar la mayor cantidad de ruido que se agrega durante la adquisición así como también mejorar las características de la imagen capturada, características tales como intensidad de color, contraste, brillo, etc., valiéndose de procesos y técnicas matemáticas. En esta etapa también pueden existir técnicas para el almacenamiento de la información.

La última etapa consiste en el método empleado para exponer la imagen al observador, esta puede ser impresa o presentada a través de medios electrónicos como la televisión o el monitor de una computadora. En esta etapa es importante tener conocimientos de la percepción de la vista humana, así como la velocidad de despliegue de la imagen [8].

Mapa de Bits (Bit Map)

La manera más fácil de representar una imagen a color en la memoria de una computadora es con un bitmap (por sus siglas en inglés). Un bitmap es un arreglo de pixeles donde cada uno determina su color. Este valor está formado por tres números en un rango que va desde el 0 hasta el 255, estos son asociados a los colores rojo, verde y azul. Cualquier color visible al ojo humano se puede representarse de esta manera.

Viendo desde este punto de vista una imagen es un arreglo bidimensional de pixeles cada uno codificado en 3 bytes que puede tener $256 \times 256 \times 256 = 16.8$ millones de diferentes colores. Esta técnica se conoce como codificación RGB (por sus siglas en inglés) y está adaptada a la visión humana. Sin embargo hay otras técnicas de codificación donde las cámaras o dispositivos de medición tienen un papel predominante.

¿Porque es difícil la visión artificial?

Debido a que somos seres visuales se llega a pensar que la visión artificial es algo sencillo de realizar, pero existen muchas diferencias al momento de realizar el proceso de visión, los humanos realizan un proceso complejo al estar observando una escena en la cual se encuentra un coche. Una computadora al momento de captar una imagen solamente recibe una malla de números por cada pixel que se encuentra en la imagen. En esta matriz de números se especifica la información correspondiente a cada uno de los pixeles tales como coloración, brillo, contraste, etc. (Figura 11).

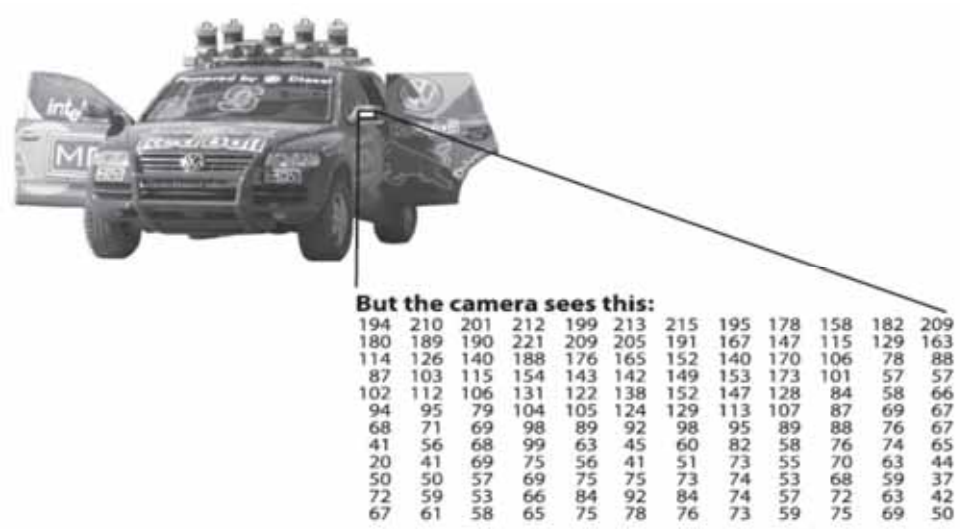


Figura 11. Lo que "visualiza" una computadora

Cabe resaltar que una imagen en escala de grises se obtiene una matriz en dos dimensiones, pero se complica el reconocimiento cuando se trata de una imagen en color, ya que se obtiene una matriz cubica y por lo tanto es más información para procesar. El problema que se plantea es más difícil de resolver cuando se tiene una escena en dos dimensiones a partir de una en 3 dimensiones, y como un problema mal planteado no hay una única manera de hacer la reconstrucción en 3D. (REFORZAR ESTE PARRAFO)

Sin embargo, los datos contienen ruido y distorsiones. Esa corrupción se deriva de las variaciones en el mundo (el clima, la iluminación, reflexiones, los movimientos), las imperfecciones del lente y la configuración mecánica, tiempo finito de integración en el sensor (desenfoque de movimiento), el ruido eléctrico en el sensor o la electrónica, y artefactos de compresión para la captura de imágenes.

Opencv

Opencv es una biblioteca libre de visión artificial originalmente desarrollada por Intel, destinada principalmente a aplicaciones de visión por computador en tiempo real. Es multiplataforma, Existiendo versiones para Linux, Mac OS X y Windows. La librería está escrita en C y C++ con la que se pueden desarrollar interfaces para Python, Ruby, Matlab y otros lenguajes. La biblioteca puede tomar ventaja de los procesadores con varios núcleos.

Uno de los aciertos que tiene Opencv es que provee a los usuarios de una infraestructura simple de usar que ayuda a la gente construir de manera sencilla sofisticadas aplicaciones de visión por computadora. 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.

Debido a que la visión por computadora y el lenguaje máquina deben ir siempre juntos, Opencv incluye librerías de propósito general Machine Learning Library (MLL, por sus siglas en inglés). Esta sub librería se centra en el reconocimiento estadístico de patrones y la agrupación. La MLL es de gran utilidad para las tareas de visión, pero es lo suficientemente general como para ser utilizado para cualquier problema de aprendizaje automático.

Opencv se usa generalmente por la mayoría de los programadores prácticos que están al tanto de algún aspecto de la función que desempeña la visión por computadora. Muy poca gente es consciente de todas las formas en la que se usa la visión por computadora ya que tienen solamente una idea vaga de su aplicación (v.gr. cámaras de vigilancia), pero en la actualidad la visión por computadora está presente en la mayoría de productos producidos a escala que lleven a cabo una inspección automatizada.

Desde su versión alfa Opencv se ha utilizado en muchas aplicaciones, productos y esfuerzos de investigación tales como la unión de mapas capturados de satélite, reducción de ruido en imágenes médicas, análisis de objetos, calibración de cámaras, sistemas de seguridad y detección de intrusos, sistemas de aplicación, aplicaciones militares tales como inspección de vuelo no tripulado, vehículos bajo el agua, Incluso se ha utilizado en el sonido, en el reconocimiento de música, entre muchas otras aplicaciones más [9].

HighGUI

Opencv tiene un kit de herramientas de gráficos llamada HighGUI (por sus siglas en inglés) que proviene de las palabras anglosajonas high-level graphical user interface que significa interfaz de usuario grafica de alto nivel. HighGUI y es la que permite abrir ventanas para poder desplegar una imagen o una secuencia de imágenes dentro de ellas, leer y escribir en archivos relacionados con gráficas (ya sea imágenes o video) y manejar tanto eventos del ratón como del teclado (Figura 12).

También es posible crear otras herramientas como deslizadores (trackbars) y después añadirlos a nuestras ventanas. Sin embargo, si se conocen otras interfaces gráficas resulta claro que HighGUI es algo redundante en aspectos como creación de ventanas, no obstante una de sus grandes ventajas es su portabilidad y facilidad de operación en cualquier lenguaje de programación. La biblioteca HighGUI en Opencv se puede dividir en tres partes: hardware, sistema de archivos, y la parte GUI (por sus siglas en inglés). La parte de hardware, se refiere principalmente a la operación de las cámaras.

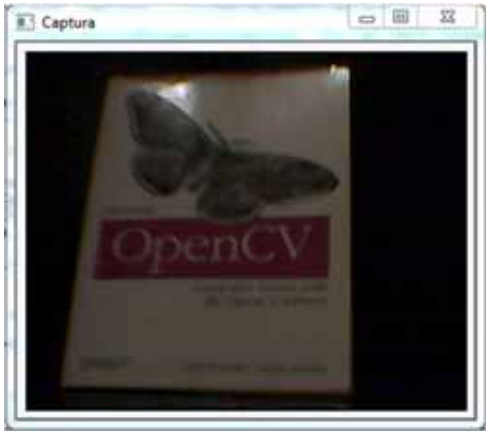


Figura 12. Captura de una cámara web mostrada con la herramienta HighGUI

En la mayoría de sistemas operativos, la interacción con una cámara es una tarea tediosa y pesada. HighGUI permite en forma fácil consultar una cámara y recuperar la última imagen de la cámara. La parte del sistema de archivos se ocupa principalmente de cargar y guardar imágenes. Una característica interesante de la biblioteca es que permite leer vídeo usando los mismos métodos que se utilizan para obtener la captura en una cámara. Por lo tanto, se logra abstraer del dispositivo en particular que se está usando. HighGUI también proporciona un par (relativamente) universal de funciones para cargar y guardar imágenes fijas. Estas funciones se basan simplemente en la extensión del archivo y automáticamente manejan toda la decodificación o codificación que sea necesaria [9].

Funciones más importantes de la Herramienta HighGUI

Como se ha señalado, la interfaz gráfica de usuario de Opencv es una herramienta sencilla de usar, ya que en comparación con otros lenguajes que contiene su propia GUI no se necesita de un estudio profundo para aplicar las características de la interfaz. Es suficiente tener conocimientos de estructuras de datos y apuntadores para poder utilizar funciones como reacción de ventanas y almacenamiento de una imagen para mostrarla dentro del marco de una ventana ya creada con anterioridad. A continuación describiremos las herramientas más sobresalientes de la HighGUI incluida en Opencv.

Espera de un evento del teclado (Waitkey)

La función cvWaitkey (por sus siglas en inglés) realiza la espera un número específico en milisegundos a que el usuario oprima la tecla adecuada. Si la tecla se oprime dentro del lapso de tiempo planteado la función regresa la tecla que se oprimió, en caso contrario se devuelve un cero, a continuación se encuentra parte del código de esta función:

```
while( 1 ) {
if( cvWaitKey(100)==27 ) break;
}
```

En el ejemplo se propone un bucle el cual se repetirá cada 100 milisegundos, dentro del bucle está la opción cvWaitkey con la cual se espera que el usuario presione una tecla, que es la única forma de salir de este bucle siempre y cuando el valor de la tecla se ASCII de 27, es decir, la tecla Escape. También es posible poner como argumento dentro de la función el cero, esto es para esperar a que el usuario presione una tecla y sin importar que tecla se presione romper el ciclo y ejecutar la siguiente instrucción.

Eventos del ratón

Como en cualquier aplicación de software diseñado a nivel usuario se puede requerir más interacción que la que proporcionan los eventos del teclado, puesto que Opencv se implementó en un ambiente de ventanas se tiene la opción de recibir y enviar instrucciones a través del ratón de la computadora.

Esto se logra mediante un mecanismo de llamada típico, es decir, recibir respuesta de los clics del ratón que haga el usuario; lo primero que se debe de hacer es una rutina de devolución de llamada en caso de que exista un evento. Una vez hecho esto, hay que registrar la devolución de llamada con Opencv, con lo que se informa a Opencv que esta es la función correcta de usar cada vez que el usuario hace clic con el ratón en una ventana en particular. Esto ofrece una gran gama de posibilidades al programar una acción con las diferentes combinaciones del ratón desde un doble clic, un clic sencillo con cualquiera de los dos botones, así como de cualquier movimiento con el scroll (desplazamiento).

Deslizadores y Switches

Opencv cuenta con otra herramienta para manejar las diferentes opciones que puede tener nuestras aplicaciones desarrolladas, se trata de una barra deslizante comúnmente llamada dentro del HighGui como trackbar (deslizador). Inicialmente los deslizadores fueron creados para que en una secuencia de video se conservara el manejo en cualquier momento de los frames (cuadros). De esta manera se tiene la opción de elegir sobre toda la línea del tiempo que dura el video el frame requerido.

Desafortunadamente la interfaz gráfica de usuario de Opencv no cuenta, como sucede con otras GUI, con una opción para utilizar botones. Es por esto que los usuarios de Opencv utilizan los deslizadores de varias formas dentro de sus aplicaciones, una manera de utilizar los trackbars es simular un botón a manera de switch (encendido-apagado), es decir, en una aplicación se deben proporcionar dos valores, de esta forma tenemos la opción de switch.

Los desarrolladores de la HighGUI crearon los deslizadores (trackbars, de aquí en adelante) para ser utilizados en secuencias de video ya que Opencv con todas las funciones dentro de su biblioteca permite crear un reproductor de video. No obstante, la ausencia de botones ha hecho que los trackbars sean usados de maneras sumamente variadas (Figura 13). Para crear un trackbar, como a cualquier ventana, se le asigna un nombre con una secuencia de caracteres, a continuación se muestra la sintaxis de la creación de trackbar.

```
Int cvCreateTrackbar(
const char* trackbar_name,
const char* window_name,
int* value,
int count,
CvTrackbarCallbackon_change
);
```

Primero se hace una llamada a la función cvCreateTrackbar con esto Opencv tiene que crear trackbar, los primeros dos argumentos se refieren al nombre de la función, así como al nombre de la ventana la cual estará anclada a la barra. Cuando trackbar se crea, este se añade en la parte superior o inferior de la venta para no obstruir ninguna imagen contenida en la ventana. El tercer y cuarto argumento se refiere a valores que cuentan las posiciones de desplazamiento, el tercer argumento es un entero a un apuntador que proporciona el valor de la posición especificada, esto es unas referencia en todo momento del valor en que hay desplazamiento hacia cualquier dirección. El cuarto argumento solo es el valor límite del desplazamiento. El último argumento es el nombre de la función la cual se llama cada vez que ocurre un cambio de posición.

Figura 13. Ventana con 2 deslizadores integrados

Esto es similar a programar eventos con el ratón. Si no se incluye una función a llamar, se considera como NULL y no se tendrá ningún efecto en la imagen, simplemente se apreciara el cambio de posición de nuestro trackbar. Para finalizar se presentan dos rutinas para establecer o leer los valores del trackbar.

```
intcvGetTrackbarPos(
const char* trackbar_name,
const char* window_name
);

voidcvSetTrackbarPos(
const char* trackbar_name,
const char* window_name,
int pos );
```

Estas dos rutinas permiten establecer la acción a realizar si el trackbar está en determinada posición, así como actualizar la posición en la cual se encuentra. Los trackbars son una herramienta que permiten interactuar de una manera sencilla y proporcionan una vista agradable al usuario, así como de ser una manera práctica y de control de establecer cambios en nuestro programa.

Procesamiento Digital de Imágenes con Opencv

Anteriormente, se proporcionó una breve introducción de lo que es procesamiento digital de imágenes. A continuación, se presentan algunas funciones para manipular estructuras de imágenes mediante Opencv.

Smoothing (Suavizado)

Este proceso también es llamado blurring (visión borrosa), es una operación simple y de uso frecuente en el procesamiento digital de imágenes, existen muchos motivos por los cuales al momento de capturar la imagen se presenta esta situación, la principal razón es la velocidad del objeto en movimiento al momento de capturar la escena.

Hay muchas razones para suavizar, pero generalmente se realiza para reducir el ruido o los efectos de la cámara. El suavizado también es importante cuando se quiere reducir la resolución de una imagen. El suavizado es un filtro que se aplica a cada pixel para así obtener una imagen nueva a partir de una imagen fuente.

Opencv ofrece cinco diferentes operaciones de suavizado como se describe en la Tabla 1. Antes de describir las opciones que existen para suavizar una imagen, se destaca la sintaxis de Opencv para comprender el funcionamiento de cada filtro. Se utilizan de 2 a 4 parámetros dentro de cada función que son utilizados como grano (kernel) para aplicar el proceso correspondiente a una imagen. La sintaxis de suavizado en Opencv es la siguiente.

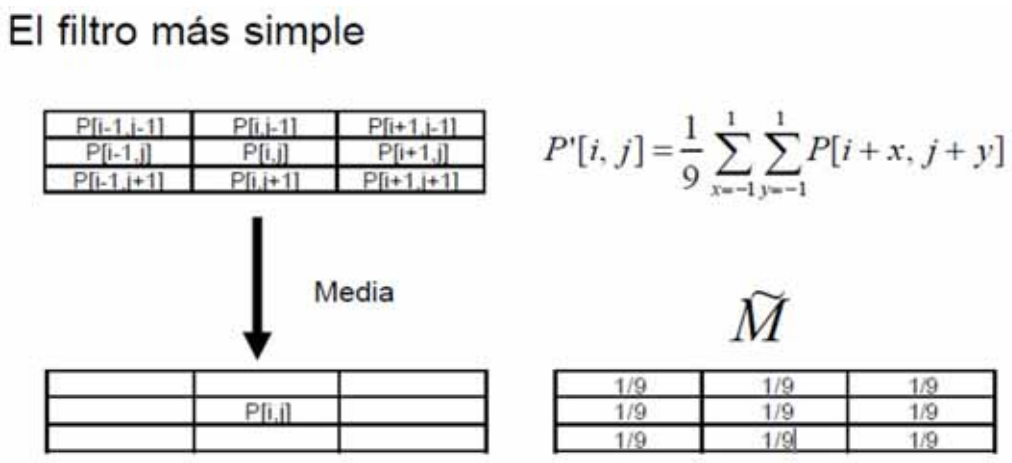
```
void cvSmooth (
constCvArr* src,
CvArr* dst,
int smoothtype = CV_GAUSSIAN,
int param1 = 3,
int param2 = 0,
double param3 = 0,
double param4 = 0);
```

Función	Nombre	Breve Descripción
CV_BLUR	Filtrado Simple	Suma a partir de param1×param2 la vecindad con la subsecuente escala 1/(param1×param2)
CV_BLUR_NO_SCALE	Filtrado simple sin escala	Suma a partir de la vecindad param1xparam2
CV_MEDIAN	Filtrado de mediana	Busca la media a partir de param1xparam2 en una vecindad cuadrada
CV_GAUSSIAN	Filtrado Gaussiano	Suma a partir de la vecindad param1xparam2
CV_BILATERAL	Filtrado Bilateral	Aplica un filtro bilateral de 3x3 con el color de sigma= param1 y espacio sigma= param2

Tabla 1. Tipos de suavizado que ofrece Opencv

En la estructura de suavizado los primeros dos argumento se refieren a la imagen fuente y destino que se utilizaran. Debido a que existen algunas operaciones de suavizado que no permiten aplicar el filtrado en la imagen original se requiere una imagen destino en algunos casos, el tercer argumento se refiere al tipo de filtrado que utilizaremos, se puede elegir entre cualquiera de los 5 filtros mencionados en la Tabla 1. El significado de los últimos 4 parámetros depende del valor elegido en la variable smoothype (tipo de suavizado). En seguida se explicara cada uno de los filtros así como la función que desempeñan los parámetros.

En el **filtrado simple** cada píxel de la salida es la media simple de todos los píxeles de una ventana alrededor del píxel correspondiente en la entrada, como se muestra a continuación.



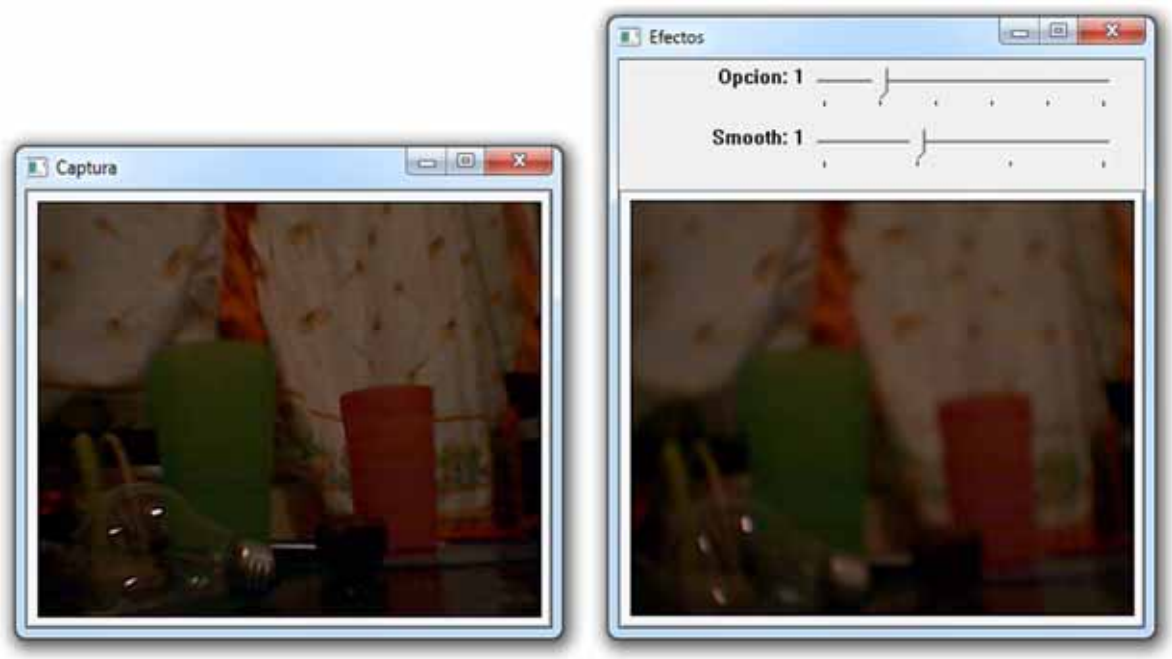


Figura 14. Ejemplo Filtro simple sin escala

El **filtro simple sin escala** es muy parecido al filtrado simple, con la diferencia de que no existe una división para crear un promedio. Por lo tanto la imagen fuente y destino deben de tener la precisión numérica diferente para que durante la operación no se produzca un desbordamiento. Es decir este proceso no se puede aplicar directamente en la imagen original. El filtro sin escala a veces es elegido con mayor frecuencia porque es un poco más rápido que el filtro simple y produce los mismos resultados (Figura 14).

El **filtro de mediana** tiene la ventaja de que el valor final del pixel es un valor real presente en la imagen y no un promedio, de este modo se reduce el efecto borroso que tienen las imágenes que han sufrido un filtro de media. Además el filtro de mediana es menos sensible a valores extremos. El inconveniente es que resulta más complejo de calcular ya que hay que ordenar los diferentes valores que aparecen en los pixeles incluidos en la ventana y determinar cuál es el valor central [10].

20	23	30	31
22	21	29	30
23	24	32	33
29	31	34	37

Filtro de mediana

N	N	N	N
N	23	30	N
N	29	31	N
N	N	N	N

El **filtro gaussiano** es probablemente el más usado aunque no es muy eficiente numéricamente. Actúa sobre cada píxel de la capa activa, estableciendo su valor como el promedio entre los valores de todos los píxeles incluidos en un radio definido. El valor máximo aparece en el pixel central y disminuye hacia los extremos cuanto menor sea el parámetro de desviación típica (Figura 15).

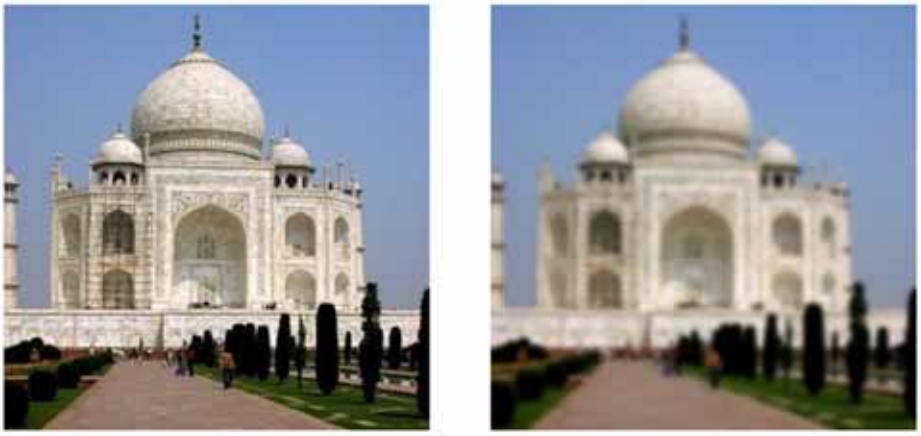


Figura 15. A la izquierda foto original, en la parte derecha imagen con filtro Gaussiano

Los dos primeros parámetros (param1 y param2) proporcionan ancho y altura de la ventana de filtro, el tercer parámetro (opcional) indica el valor de la desviación estándar (la mitad del ancho máximo de la mitad de la ventana). Si el tercer parámetro no se especifica, el Gaussiano determinará automáticamente el tamaño de la ventana usando las siguientes fórmulas [9]:

$$\sigma_x = \left(\frac{n_x}{2} - 1\right) \cdot 0.30 + 0.80, \quad n_x = \text{param1}$$

$$\sigma_y = \left(\frac{n_y}{2} - 1\right) \cdot 0.30 + 0.80, \quad n_y = \text{param2}$$

Si desea que el filtro sea asimétrico, entonces también es posible (opcionalmente) proporcionar un cuarto parámetro, en este caso, los parámetros tercero y cuarto serán los valores de la desviación estándar en la dirección horizontal y vertical, respectivamente. Si los parámetros tercero y cuarto se proporcionan, pero los dos primeros se establecen en 0,

entonces el tamaño de la ventana se determina automáticamente a partir del valor de sigma (desviación estándar) [9].

El **filtrado bilateral** es el producto de dos Gaussianas.

1.- Gaussiana espacial
2.- Gaussiana en el espacio de los valores de los pixeles.

En Opencv este filtro toma dos parámetros el primero es el tamaño del kernel gaussiano usado como dominio espacial. El segundo parámetro corresponde al ancho del kernel Gaussiano en el dominio del color, cuanto mayor sea el segundo parámetro, más amplio es el rango de intensidades que se incluirán en el suavizado.

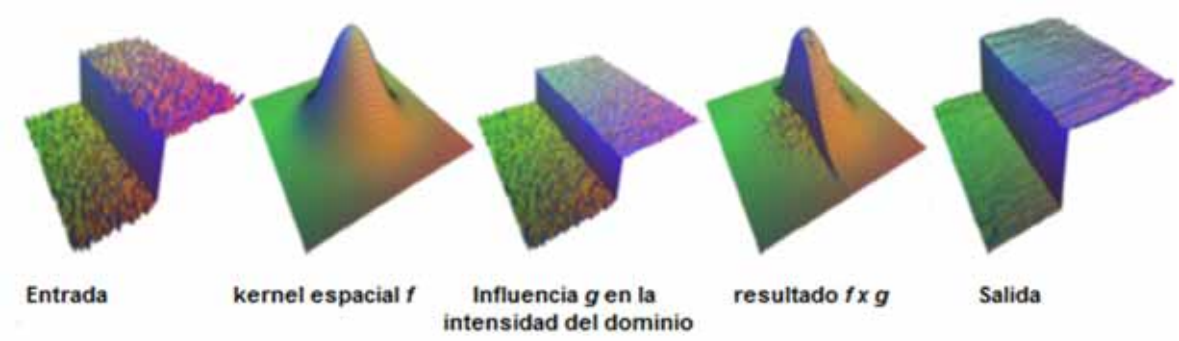


Figura 16. Filtrado bilateral

Transformaciones Morfológicas

Opencv provee de una interfaz rápida para crear transformaciones morfológicas. La morfología matemática se basa en operaciones de teoría de conjuntos. En el caso de imágenes binarias, los conjuntos tratados son subconjuntos de Z_2 y en el de las imágenes en escala de grises, se trata de conjuntos de puntos con coordenadas en Z_3.

Las transformaciones morfológicas básicas se llaman dilatación y erosión. La morfología se puede usar, entre otros, con los siguientes objetivos:

- Pre-procesamiento de imágenes (supresión de ruidos, simplificación de formas).
- Destacar la estructura de los objetos (extraer el esqueleto, detección de objetos, envolvente convexa, ampliación, reducción,...)
- Descripción de objetos (área, perímetro,...)

La palabra morfología significa forma y estructura de un objeto. Para imágenes binarias se definen operaciones morfológicas. Y con estas se constituye una herramienta de extracción de componentes de imagen útiles en la representación y descripción de la forma de las regiones. La dilatación se puede simplificar es decir agregar pixeles a un objeto, hacerlo más grande, y la erosión es hacerlo más pequeño. La erosión saca los "outliers* del objeto".

Erosión-Dilatación

La morfología binaria por lo general trata de ensayar dentro de una imagen una forma predefinida simple para poder sacar conclusiones sobre cómo esta forma encaja o no dentro de la imagen**. La dilatación** corresponde a una convolución* de una imagen (o parte de la imagen) con un kernel o elemento estructurante, este kernel puede ser de cualquier forma o tamaño teniendo un único punto de anclaje predefinido.

Existen tres elementos estructurantes muy utilizados dentro de la morfología a continuación se explican todos ellos denotados por A (Figura 17).

- A es un círculo abierto de radio r, centrado en el origen.
- A es un cuadrado 3×3, esto es A= {(-1,-1), (-1,0), (-1,1), (0,-1), (0,0), (0,1), (1,-1), (1,0), (1,1)}.
- A es una cruz dada por: B = {(-1,0), (0,-1), (0,0), (0,1), (1,0)}.

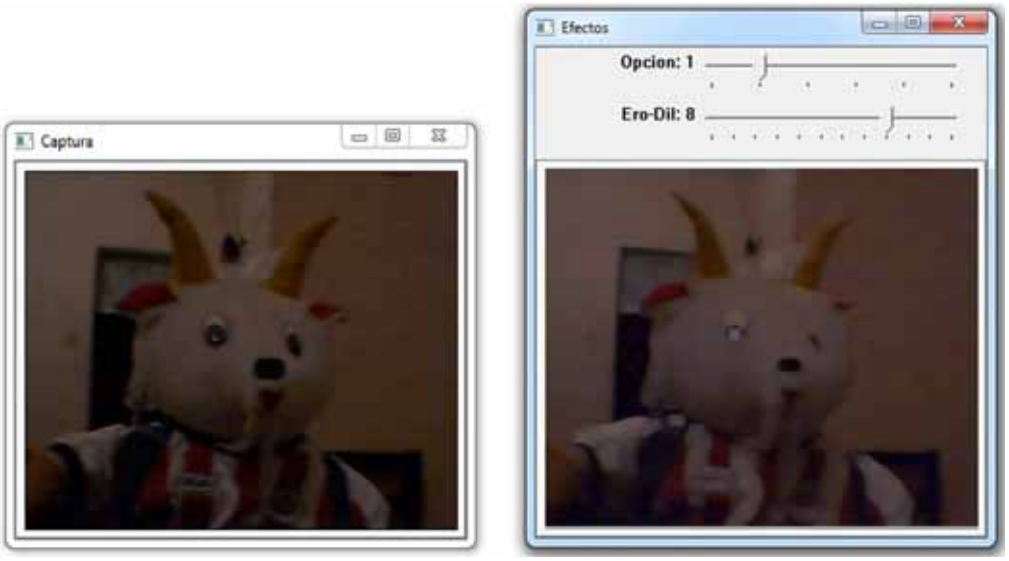
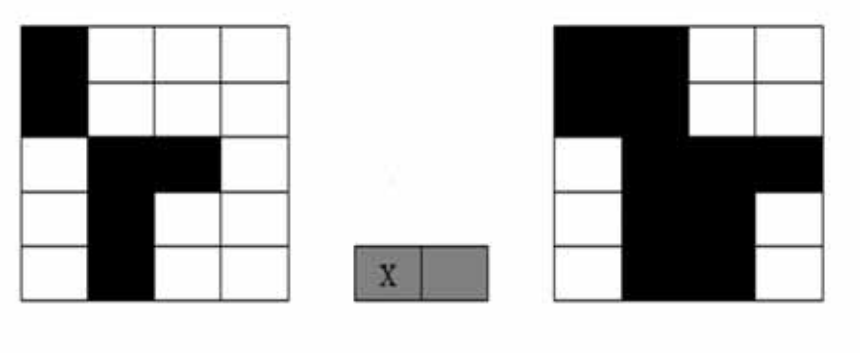
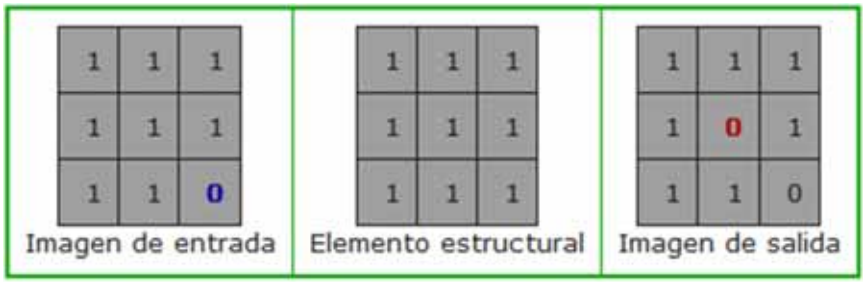


Figura 17. Efecto dilatación morfológica.

Gráficamente la dilatación se calcula de la siguiente manera: Se recorre la imagen y cuando se encuentra un 1 se ubica al origen del elemento estructural. Se realiza la unión de la imagen con el elemento estructural en los puntos que coincidan. Si todos los 1 de la imagen coinciden con 1 del elemento estructural entonces marcamos el píxel del origen del elemento con un 1. A continuación se muestra un pequeño ejemplo donde queda más claro cómo es que funciona la dilatación.



La erosión es la operación inversa se basa en reducir el nivel de píxeles del entorno de un objeto. Para ello se selecciona el mínimo valor en la vecindad del punto a tratar (en vez del máximo, como cuando se aplica la dilatación). Por ejemplo



Como se puede apreciar, se ha sustituido el píxel central por el mínimo valor de los píxeles de la imagen original definidos por la vecindad contenida en el elemento estructural. En general, mientras que la dilatación extiende la región, la erosión reduce la región. Por otra parte, la dilatación tiende a suavizar las concavidades y la erosión tiende a suavizar las protuberancias. Por supuesto, el resultado exacto dependerá del kernel, pero por lo general estas suposiciones son ciertas en la mayoría de los elementos estructurales.

En Opencv podemos aplicar el efecto de estas funciones mediante el siguiente código:

```
void cvErode (
IplImage* src,
IplImage* dst,
IplConvKernel* B = NULL,
int iterations = 1
);
```

```
voidcvDilate(
IplImage* src,
IplImage* dst,
IplConvKernel* B = NULL,
int iterations = 1
);
```

Ambas funciones toman como sus primeros dos argumentos apuntadores a imágenes; el primer argumento es la fuente y el segundo argumento es el destino, ambas funciones soportan aplicar el efecto in situ siempre y cuando la imagen fuente y destino sean la misma imagen. El tercer argumento es el elemento estructural o kernel, Opencv permite crear tu propio kernel (por default es nulo), en caso de que sea Nulo el elemento estructural es un cuadrado de 3 × 3 con un anclaje en el centro. Finalmente el último argumento es el número de iteraciones, si no se establece el valor por default es 1, la operación se aplicara múltiples veces durante la aplicación de una llamada.

La operación de erosión a menudo se utiliza para eliminar ruido en una imagen. La idea aquí es que las motas se erosionan mientras que las grandes regiones que tienen contenido visual importante no son afectadas. La operación de dilatación es utilizado muchas veces cuando se trata de encontrar los componentes conectados (es decir, las grandes regiones discretas de color de los píxeles similares o la intensidad). La utilidad de la dilatación se debe a que en muchos casos, una región grande puede ser descompuesta en varios componentes (Figura 18).

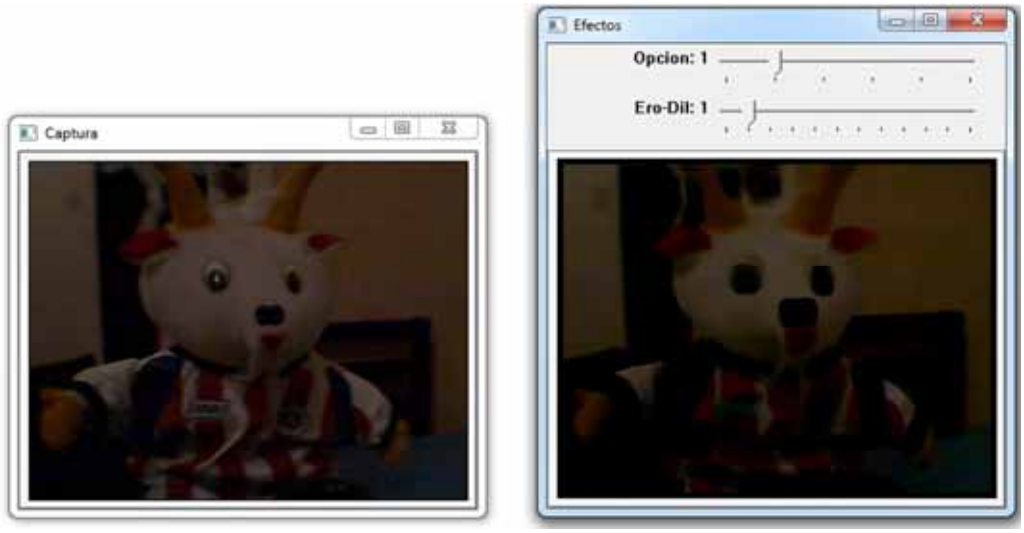


Figura 18. Efecto de erosión morfológica.

Umbral

Umbral (threshold del inglés) es una herramienta simple pero eficaz para aislarlos objetos de interés del fondo. Sus aplicaciones incluyen varias características tales como el análisis de imágenes de documentos, cuyo objetivo es extraerlos caracteres impresos, logotipos, contenido gráfico, o partituras musicales, también se utiliza para el procesamiento de mapas que tiene como objetivo localizar las líneas, leyendas y personajes. También se utiliza para el procesamiento de escenas, con el objetivo dela detección de objetos y marcado. Del mismo modo, se emplea para la inspección de calidad de los materiales para desechar materiales defectuosos

Frecuentemente cuando se realizan varias capas de procesamiento se necesita tomar una decisión acerca de pixeles en la imagen y rechazar pixeles que estén por encima o por debajo de cierto valor y solamente mantener los que estén dentro de cierto rango. La función de Opencv cvThreshold realiza estas tareas. La idea básica es: que dado un array (arreglo) básico junto con un umbral, se le proporciona un valor a cada elemento del arreglo dependiendo si esta por abajo o por encima del umbral. El código de la función es:

```
doublecvThreshold(
CvArr* src,
CvArr* dst,
double threshold,
doublemax_value,
intthreshold_type
);
```

En Opencv existen cinco tipos de umbral que pueden ser aplicados, cada uno de ellos corresponde a una operación en particular con el fin de comparar el pixel fuente y el umbral aplicado. El pixel destino puede tomar 3 valores distintos como observamos en la tabla.

Threshold type	Operation
CV_THRESH_BINARY	$dst_j = (src_j > T) \,?\, M : 0$
CV_THRESH_BINARY_INV	$dst_j = (src_j > T) \,?\, 0 : M$
CV_THRESH_TRUNC	$dst_j = (src_j > T) \,?\, M : src_j$
CV_THRESH_TOZERO_INV	$dst_j = (src_j > T) \,?\, 0 : src_j$
CV_THRESH_TOZERO	$dst_j = (src_j > T) \,?\, src_j : 0$

Capítulo 2

Algoritmo SIFT

SIFT

SIFT, por sus siglas en inglés corresponde a Scale Invariant features transform (Transformada de características invariantes a la escala), es un algoritmo de visión artificial creado para obtener las características locales de una imagen. El algoritmo fue patentado en Estados Unidos y la propietaria es la Universidad de British Columbia y fue inventado por David Lowe (profesor de esta Universidad) en 1999. Este algoritmo es adecuado para describir las características de una imagen para hacer coincidir objetos en una imagen o una escena. Estas características resultan ser invariantes a la escala y rotación de la imagen y parcialmente invariante al cambio de iluminación y el punto de vista 3D (tridimensional) de la cámara. Las características están bien localizadas tanto en el ámbito espacial y la frecuencia, lo que reduce la probabilidad de interrupción de oclusión, desorden o ruido. Un gran número de características se pueden extraer de imágenes típicas con el SIFT. Además, las características son muy distintivas, y eso permite que una sola característica sea correctamente mapeada con una probabilidad alta en concordancia de una base de datos de características de gran magnitud [12].

Mediante la adopción de un enfoque de filtrado en cascada, el costo de extracción de características se minimiza. A continuación se presentan las principales etapas de cálculo utilizado para generar el conjunto de características de la imagen:

1. Detección de extremos escala-espacio: La primera etapa busca en todas las escalas y las ubicaciones de la imagen. Se lleva a cabo de manera eficiente mediante el uso de una función Gaussiana para identificar posibles puntos característicos los cuales son invariantes a la escala y la orientación.

2. Localización de los keypoints (puntos clave): en cada localidad propuesta se realiza un ajuste para determinar la ubicación y la escala. Los keypoints se seleccionan dependiendo su estabilidad.

3. Asignación de orientación: uno o más orientaciones se asignan a cada ubicación de keypoints. Todas las operaciones futuras se realizan en los datos de la imagen que se ha transformado con respecto a la orientación, la escala y la ubicación asignada de cada función, proporcionando así la invariancia de estas transformaciones.

4. Descriptor de keypoints: Los gradientes de la imagen local se miden en la escala seleccionada en la región alrededor de cada keypoints. Estos se transforman en una representación que permite importantes niveles de distorsión de la forma local y el cambio en la iluminación.

Un aspecto importante de este enfoque es que genera un gran número de características que cubren densamente la imagen. Una imagen típica de tamaño 500×500 pixeles dará lugar a cerca de 2000 características estables (aunque esta cifra depende del contenido de la imagen y los diferentes parámetros). La cantidad de características es particularmente importante para el reconocimiento de objetos, donde la capacidad de detectar objetos pequeños en fondos desordenados requiere que al menos tres características se emparejen correctamente para una identificación fiable.

Etapas fundamentales del SIFT

Cabe destacar que para el reconocimiento de objetos, no solo es necesario obtener las características en la imagen, si no que se necesita de al menos de un conjunto de características extraídas de otra imagen para determinar con base en las correspondencias entre los puntos clave si existe una hipótesis de objetos dentro de la escena.

El algoritmo recibe como parámetro de entrada la trayectoria en donde se encuentra la imagen y después de que se aplica el SIFT se obtiene como salida los correspondientes descriptores, así como la información correspondiente a cada descriptor encontrado; específicamente:

- La posición en una matriz N×2 que corresponden a las coordenadas (x,y) las cuales son almacenada en filas.
- La escala en una matriz N×3 almacenadas en la siguiente forma: la columna 1 almacena el intervalo, la columna 2 almacena los octavos y la tercera columna almacena la desviación estándar de cada descriptor encontrado.
- La orientación de cada descriptor, en una matriz N×1 en un rango de $[-\pi, \pi]$.

Son cuatro etapas fundamentales del SIFT. A continuación se describe las fases cada una de estas etapas.

Detección de extremos escala-espacio

En la primera etapa, se recibe la imagen fuente y hay que resolver el problema de en qué escala se encuentra. Para resolver este problema, en el procesamiento digital, se genera una serie de imágenes (pirámide de imágenes, Figura 19). Las pirámides de imágenes permiten encontrar características en una imagen ya que contienen información de las imágenes a diferentes resoluciones. La imagen de la base de la pirámide contiene la más alta resolución, mientras que la punta de la pirámide contiene la más baja resolución. Conforme nos movemos hacia arriba de la pirámide, tanto el tamaño como la resolución de las imágenes disminuyen.

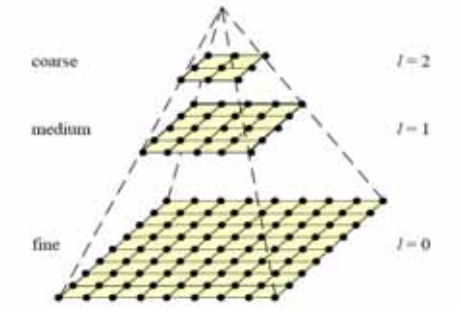


Figura 19. Pirámide de imágenes [13].

Para encontrar los keypoints en una imagen el algoritmo de SIFT utiliza una pirámide Gaussiana es decir a una serie de imágenes con diferentes escalas los filtros Gaussianos para suavizarlos son aplicados. En una pirámide Gaussiana cada nivel de la pirámide es suavizado por un kernel Gaussiano simétrico y sub muestreado para obtener la siguiente capa. Si no se especifican los octavos y lo intervalos por octavo por default los valores de 4 y 2 son tomados, respectivamente.

Las imágenes convolucionadas se encuentran agrupadas por octavas (una octava corresponde a duplicar el valor de la desviación estándar σ), el valor de σ se selecciona de manera que obtenemos un número determinado de imágenes convolucionadas por octava (Figura 20).

Después se realiza la diferencia de las imágenes ya suavizadas. Los keypoints corresponden al máximo y mínimo de la diferencia de Gauss (ver ecuación (6)), como suele ocurrir con imágenes a múltiples escalas (Figura 21).

Figura 20. Pirámide Gaussiana de la foto de Einstein

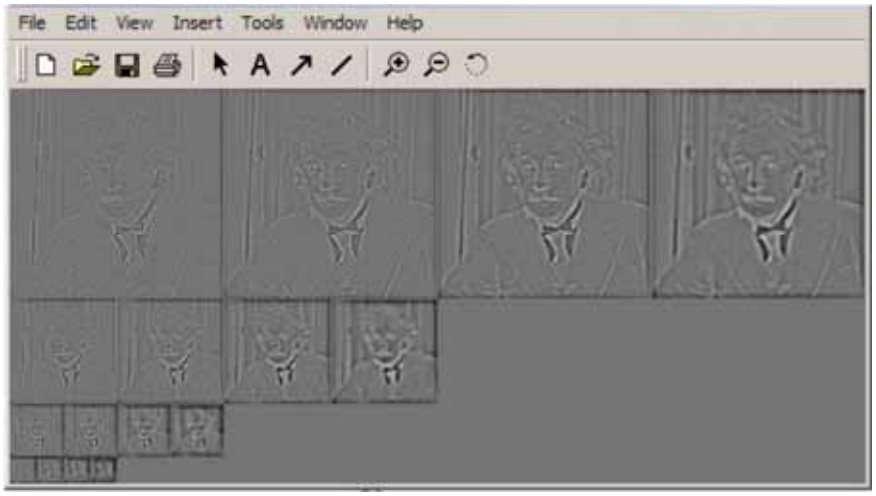


Figura 21. Diferencia de los barridos

Lindeberg [14] demostró que el laplaciano de Gauss normalizado por σ^2 (ssLoG) da como resultado:

$$\sigma^2\nabla^2 G(x,y,\sigma) = -\frac{1}{2\pi\sigma^2}\left(2-\frac{x^2+y^2}{\sigma^2}\right)\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right), \qquad (1)$$

Se puede demostrar que ssLog está estrictamente relacionada con la diferencia de Gauss (DoG). De la ecuación de difusión presentada por Lowe[12] y reemplazando el parámetro del tiempo *t* por σ nos da:

$$\frac{\partial G(x, y, \sigma)}{\partial \sigma} = \sigma \nabla^2 G(x, y, \sigma), \tag{2}$$

Aproximando el miembro derecho de la ecuación (2) por las diferencias finitas:

$$\frac{\partial G(x, y, \sigma)}{\partial \sigma} = \lim_{\Delta\sigma \to 0} \frac{G(x, y, \sigma + \Delta\sigma) - G(x, y, \sigma)}{\Delta\sigma}. \tag{3}$$

Reemplazando $\nabla\sigma = \sigma(k - 1)$ en la ecuación (3) tenemos :

$$\frac{\partial G(x, y, \sigma)}{\partial \sigma} = \lim_{k \to 1} \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{\sigma\,(k - 1)}. \tag{4}$$

Por lo tanto la ecuación (2) es aproximada mediante:

$$(k - 1)\,\sigma^2 \nabla^2 G(x, y, \sigma) \approx G(x, y, k\sigma) - G(x, y, \sigma), \tag{5}$$

En el miembro derecho de la ecuación anterior corresponde a la diferencia de Gauss la cual es definida por:

$$D(x, y, \sigma) = G(x, y, k\sigma) - G(x, y, \sigma). \tag{6}$$

De aquí se obtiene que: $\sigma\, 2\nabla^2 G(x, y, \sigma) \sim D(x, y, \sigma)$ (CHECAR), para k suficientemente cercano a 1. Lowe reporta buenos resultados en la práctica para resultados $2^{1/s}$ donde s> 1 es un entero que indica el número de intervalos dentro de un nivel de escala simple. En comparación con muchos detectores de características conocidos, ssLoG proporciona una extracción de características de la imagen más estable en el grupo de distorsión de la imagen

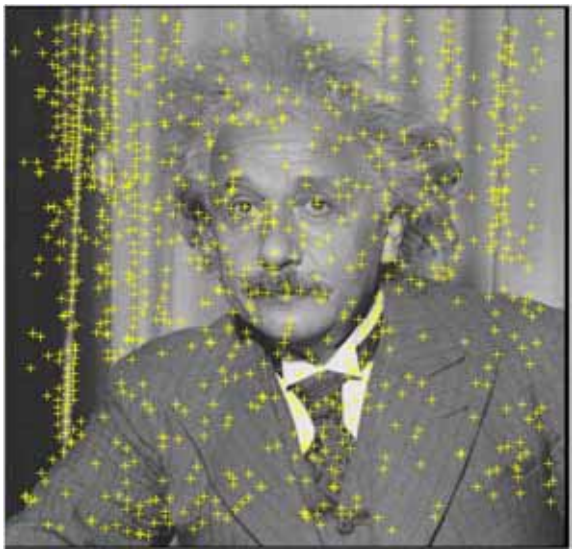


Figura 22. Keypoints obtenidos con la detección de extremos escala – espacio

Una vez que a las imágenes se les aplica la diferencia de Gauss (DoG) se obtienen los keypoints (Figura 22). Esto se hace mediante la comparación de cada píxel de la imagen con DoG a sus ocho vecinos en la misma escala. Si el valor del píxel es el máximo o mínimo entre la comparación de todos los píxeles, este es seleccionado como un punto clave candidato.

Localización de keypoints

La localización escala-espacio produce diversos keypoints los cuales la gran mayoría son inestables. Por lo que hay que realizar un ajuste de ellos para rechazar aquellos que sean sensibles al ruido, no están bien localizados en los bordes o tenga bajo contraste.

En primer lugar, para cada keypoint seleccionado, se utiliza la interpolación de vecinos cercanos para determinar con exactitud su posición. Inicialmente se procedía en localizar sólo cada keypoint en cuanto a ubicación y escala del punto clave candidato. Ahora, se calcula la localización del extremo mediante interpolación, lo cual mejora sustancialmente la correspondencia y la estabilidad. La interpolación se realiza mediante una expansión de Taylor (hasta sus términos cuadráticos) aplicada a la diferencia Gaussiana de la función escala-espacio, D(x, y, σ), con el keypoint seleccionado como el origen [12]. La función está dada por:

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^{\mathbf{T}}\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \qquad (7)$$

Donde D y sus derivadas son evaluadas en un punto de muestreo x=(x, y, σ)^T que es el desplazamiento de este punto. La localización del extremo $\hat{x}$ se determina al derivar esta función con respecto a x e y para después anularla:

$$\hat{\mathbf{x}} = -\frac{\partial^2 D^{-1}}{\partial \mathbf{x}^2} \frac{\partial D}{\partial \mathbf{x}}. \tag{8}$$

Como propone Brown [15], la matriz hessiana y la derivada de D se aproximan si usamos las diferencias de los puntos de muestreo de los vecinos. Lo que resulta en un sistema lineal 3×3 el cual es muy fácil de resolver. Si el desplazamiento de $\hat{x}$ es mayor que 0.5 en cualquier dimensión, esto significa que el valor del extremo se encuentra más cercano a un punto de muestreo diferente. En este caso se cambia el punto de muestreo y la interpolación se realiza alrededor de ese punto. El desplazamiento final de $\hat{x}$ se añade a la localización de su punto de muestreo para obtener la estimación de la interpolación de la ubicación del extremo. El valor de la función en el extremo D($\hat{x}$) , es útil para rechazar extremos inestables con bajo contraste. Esto se puede obtener por la ecuación sustituyendo (8) en (7), para obtener:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}} \hat{\mathbf{x}}. \tag{9}$$

Para la estabilidad, no es suficiente rechazar los keypoints con bajo contraste. La función de diferencia de Gauss permite tener una respuesta buena a lo largo de los bordes, incluso si la ubicación a lo largo de la orilla es poco determinada y por lo tanto inestable a pequeñas cantidades de ruido.

Para los picos en la función de diferencia de Gauss, la curvatura principal a través del borde será mucho más grande que la curvatura principal a lo largo de ella. Para encontrar la cantidad de estas curvaturas principales es necesario encontrar los eigenvalores de segundo orden de la matriz Hessiana H:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{10}$$

Las derivadas son estimadas tomando las diferencias de los vecinos en lo puntos de muestreo. Los eigenvalores de H son proporcionales a la curvatura principal de D. Utilizando la aproximación de Harris y Stephens [16]. Sea α el eigenvalor con la magnitud más larga y β el de la magnitud más pequeña, entonces nosotros podemos computar la suma de los eigenvalores con la traza de H y el producto de su el determinante:

$$\begin{aligned} \mathrm{Tr}(\mathbf{H}) &= D_{xx} + D_{yy} = \alpha + \beta, \\ \mathrm{Det}(\mathbf{H}) &= D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta. \end{aligned} \tag{11}$$

Si el determinante es negativo, las curvaturas tienen diferentes signos (punto hiperbólico) por lo que el punto se descarta ya que no es un valor extremo. Sea r el radio entre las magnitudes más larga y más corta de los eigenvalores, entonces $\alpha = r\beta$ por lo tanto:

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}, \tag{12}$$

Como dependemos del valor individual de los eigenvalores, la cantidad $(r+1)^2/r$ es minima cuando los dos eigenvalores son iguales y se incrementa con r. Por lo tanto, para comprobar que la proporción de curvaturas principales se encuentra por debajo de cierto umbral r, hay que verificar:

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}. \tag{13}$$

La figura 3.? Nos muestra la interpolación aplicando la expansión cuadrática de Taylor a la función de diferencia de Gauss (DoG). Esto es eficiente para computar con menos de 20 operaciones de punto flotante requeridos para cada punto clave.

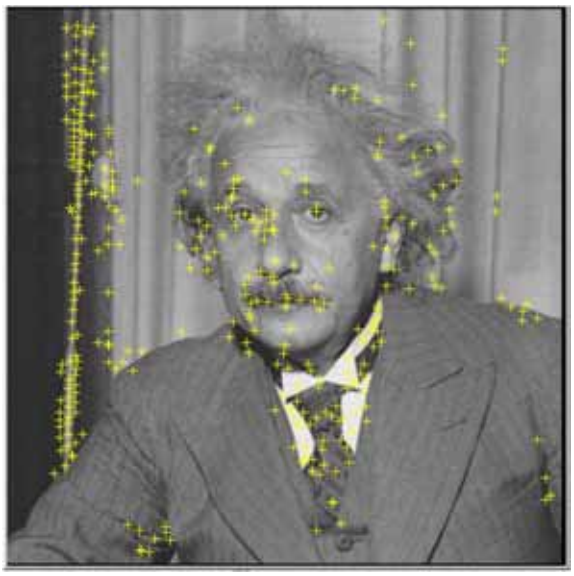


Figura 23. Localización de los Keypoints.

Asignación de Orientación

Mediante la asignación de una orientación consistente a cada Keypoint basado en las propiedades locales de una imagen, el Keypoint se puede representar en relación con esta orientación y por lo tanto, lograr la invariancia a rotación de la imagen. La escala de los Keypoint se utiliza para seleccionar la imagen suavizada de Gauss, L, con la máxima escala, de modo que todos los cálculos se realizan invariantes a escala. Para cada muestra la imagen, L (x, y), la magnitud del gradiente, m (x, y), y la orientación, θ(x, y), se calcula previamente mediante las diferencias de píxeles:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

Un histograma de orientación se forma a partir de las orientaciones del gradiente de puntos de muestreo dentro de una región alrededor del Keypoint. El histograma de orientación cuenta con 36 lugares que cubren el rango de 360°. Cada muestra añadida al histograma es ponderado por su magnitud y por el gradiente de una ventana circular de Gauss con una desviación estándar σ que es 1.5 veces la de la escala del Keypoint.

Los picos en la orientación del histograma corresponden a las direcciones principales de los gradientes locales. El pico más alto el en histograma se detecta y después cualquier otro pico local que está dentro del 80% del pico más alto se utiliza para crear un Keypoint con esa orientación

Por lo tanto, para lugares con múltiples picos de magnitud similar, existen varios Keypoint creados con la misma ubicación y escala, pero con diferentes orientaciones. Sólo alrededor del 15% de los puntos se asignan múltiples orientaciones, pero contribuyen significativamente a la estabilidad del reconocimiento. Finalmente, una parábola es ajustada para 3 valores del histograma más cercanos a cada pico para interpolar la posición del pico, esto con el fin de lograr una mayor precisión.

Descriptor de Keypoints

Todo lo anterior ha servido para encontrar los Keypoints, descartar aquellos que tienen bajo contraste o que presentan ruido y aquellos puntos que quedaron dentro del rango se les asigne una ubicación, escala y orientación. El siguiente paso consiste en calcular un descriptor de la región de la imagen local que debe de ser tan invariable como sea posible a las variaciones restantes, tales como el cambio en la iluminación o el punto de vista en 3D.

Para encontrar este descriptor invariante Lowe [12] utilizó lo desarrollado por Edelman et. al [16]. Su propuesta se basó en la representación de un modelo de visión biológica complejo, en particular de las neuronas de la corteza visual primaria. Estas neuronas responden a un gradiente en una orientación particular y la frecuencia espacial, pero la ubicación del gradiente en la retina permite el cambio en un campo de recepción pequeño en lugar de ser localizado con precisión.

Edelman et al. propusieron la hipótesis de que la función de estas neuronas permitían la coincidencia y el reconocimiento de objetos 3D a partir de un rango de puntos de vista. Se han realizado experimentos detallados con modelos por computadora en 3D de objetos y figuras de animales que muestran los gradientes de coincidencia incluyendo rotaciones en 3D con buenos resultados (Figura 24).

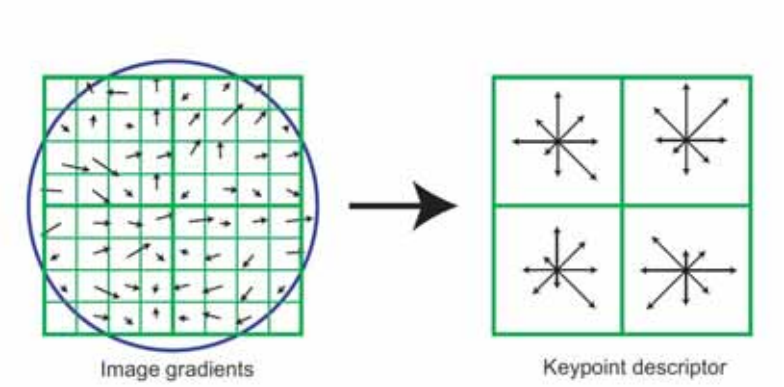


Figura 24. A la izquierda está la imagen del gradiente, del lado derecho el Keypoint descriptor, imagen tomada de [12].

Un descriptor de Keypoint se crea mediante el cálculo de la primera magnitud y orientación del gradiente en cada punto de muestra en una región alrededor de la ubicación del Keypoint, como se muestra a la izquierda de la Figura 18. Estos son ponderados por una ventana gaussiana, indicado por el círculo superpuesto. Estas muestras se acumulan en los histogramas de orientación que resumen los contenidos a través de las subregiones 4×4, como se muestra a la derecha, con longitud de cada flecha correspondiente a la suma de las magnitudes de los gradientes cerca de esa dirección dentro de la región.

El descriptor está formado por un vector que contiene los valores de todas las entradas de la orientación de histograma, que corresponden a las longitudes de las flechas (lado derecho de la Figura 18). La figura muestra una matriz de 2×2 de los histogramas de orientación, mientras que los experimentos muestran que los mejores resultados se logran con una serie de histogramas 4×4 con 8 depósitos de orientación en cada uno. Por lo tanto, se utiliza un vector 4×4×8 = 128 elementos de características para cada Keypoint.

Por último, la función vectorial se modifica para reducir los efectos del cambio de iluminación. En primer lugar, el vector se normaliza, se hace unitario. Se aplica un cambio en el contraste de la imagen en cada valor de píxel, el cual se multiplicará por una constante que también multiplicará a los gradientes, por lo que este cambio de contraste puede cancelarse por la normalización de vectores.

El brillo que se agrega con esta constante se añade a cada píxel de la imagen por lo tanto no afectará a los valores de los gradientes, ya que se calcula a partir de las diferencias de píxeles. Por lo tanto, el descriptor es invariante a los cambios en la iluminación.

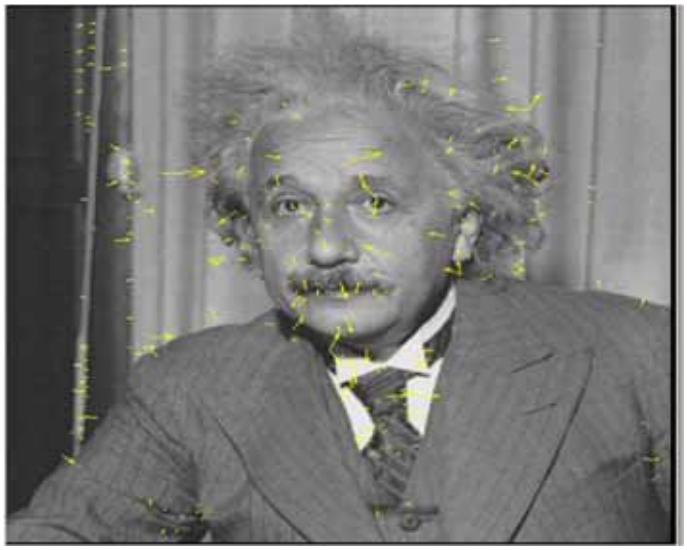


Figura 25. Se calcula el vector descriptor para cada Keypoint. Este vector es invariante a la rotación y la iluminación.

En la Figura 25 se muestran los vectores descriptores que resultan al aplicar el SIFT en una imagen de Einstein, cada uno de estos descriptores contiene la información de la orientación, escala, posición y las características del descriptor. Estos descriptores se almacenan en matrices por lo que es posible guardar estos descriptores en una base de datos y utilizarlos para encontrar las coincidencias entre las escenas de dos imágenes. Esto se hace encontrando la mejor coincidencia para cada Keypoint, mediante la identificación de su vecino más cercano en la base de datos de Keypoints de las imágenes. El vecino más cercano se define como el punto clave de la distancia euclidiana mínima para el vector descriptor invariable.

Capítulo 3

Descripción del sistema desarrollado para el robot cartesiano

Con base en los preliminares y capítulos 1 y 2 se ha descrito la estructura del sistema para el robot cartesiano. Inicialmente se intentó implementarlo usando como base Opencv para capturar y almacenar la escena mediante la cámara digital de alta definición, incluso se aplicaron diferentes filtros para suavizar la imagen.

Sin embargo, integrar múltiples sistemas dificultó el desarrollo de una aplicación que pudiera englobar los módulos requeridos para el reconocimiento de objetos. Por esta razón no se incluyó la biblioteca de procesamiento digital de imágenes (PDI) desarrollada por Intel. Aunque si fue útil para la comprensión del PDI y entender mejor los procesos realizados por el algoritmo de SIFT.

El algoritmo de SIFT es una serie de módulos que existen dentro de la visión artificial para obtener las características locales de una imagen, es un proceso el cual implementa el método de cascada para ser eficiente y minimizar el costo de obtener los Keypoints, y está dividido en cuatro etapas que mejoran los algoritmos para obtener los vectores descriptores que contienen la información de ubicación, escala y características de cada Keypoint, estos descriptores son robustos a cambios de iluminación, rotación y escala.

El sistema desarrollado consiste esencialmente en dos principales módulos, el primer módulo (módulo A) se encarga de manipular el robot cartesiano en su recorrido del área que se quiere explorar en busca de objetos. El segundo módulo (módulo B) es el que realiza el reconocimiento de objetos en el área escaneada. Como anteriormente se había mencionado el proveedor de los dispositivos que componen al robot cartesiano proporciona un software para manipular los dispositivos mediante una interfaz gráfica (Motion Planner). Esta interfaz es limitada a pesar de que permite desarrollar scripts con varias sentencias y ejecutar todas ellas en conjunto. La ventaja es que al instalar este software propietario también se instala la biblioteca de funciones COM6SRVR la cual puede ser integrada con cualquier lenguaje de programación (software) que maneje objetos OLE. Por lo que es posible desarrollar aplicaciones visuales propias simplemente integrando esta librería.

Para desarrollar el módulo A se eligió el entorno de desarrollo integrado (IDE por sus siglas en Ingles) Visual C++ 6.0 la cual corresponde a la versión requerida para ejecutar el servidor de comunicación COM6SRVR.

El módulo B que comprende al procesamiento de imágenes y la extracción de características fue desarrollado en MATLAB (por sus siglas en inglés, matrix laboratory) que resultó ser el más adecuado para la información obtenida con el algoritmo de SIFT dado que las características de la imagen se encuentran almacenadas mediante matrices.

El objetivo inicial de este proyecto era recorrer toda el área de exploración en busca de objetos, en caso de haber reconocido un objeto la pinza de sujeción (gripper) bajara a tomar el objeto para después llevarlo a una posición predefinida de esta forma se clasificaran los objetos de acuerdo a la clase a la que pertenecen, por ejemplo, si se reconoce un desarmador se coloca con la clase herramientas, si reconoce un carrito lo agrupe con la clase juguetes. Por las restricciones del robot cartesiano, el eje Z en el cual se encuentra adaptado el gripper no baja lo suficiente para alcanzar los objeto además de que se actúa neumáticamente.

Así en lugar de sujetar los objetos mediante el gripper, fue suficiente calcular el centro del objeto aplicando una relación pasos de motor versus pixeles. Puesto que conocido el centro se obtienen las coordenadas del centro de la imagen, los actuadores correspondientes a los ejes X-Y se moverán los pasos de motor exactos para posicionarse debajo de la ubicación del centro. Con el actuador del eje Z se indica que el objeto fue encontrado o más exactos reconocido.

A continuación describiremos los dos principales módulos que componen el reconocimiento de objetos mediante visión activa, cabe destacar que para mayor eficiencia cada módulo se divide en pequeñas funciones las cuales mediante programación modular se ilustra la forma en que se resolvió el problema.

Módulo de Reconocimiento de Imágenes

El módulo fue desarrollado en lenguaje M, el cual es nativo de MATLAB el cual es ideal para manipular cualquier formato de imágenes mediante su caja de herramientas toolbox. Por lo que dada una imagen esta es transformada en una malla numérica la que se almacena en matrices, cada uno los números representa ya sea color, intensidad y brillo de cada pixel que conforma la imagen.

La cámara SONY XCD-710 con la que se cuenta el robot cartesiano, solo captura escenas en escala de grises, esto no representa una desventaja por el contrario facilita la

manipulación de la imagen adquirida. Una imagen a color queda representada por una matriz tridimensional mientras que si se encuentra en un formato en blanco y negro solamente se recibe una matriz de dos dimensiones. En otras palabras como una primera aproximación conviene manipular así la imagen y facilitar los cálculos a cada pixel.

El algoritmo empleado para el reconocimiento de imágenes trata de imitar las funciones del cerebro, los seres humanos tenemos la sensación de tener una gran agudeza visual pero en realidad nuestros ojos se están moviendo sutilmente pero constantemente para barrer nuestro entorno e ir recopilando la información que nos permite construir un mapa mental de nuestro entorno. Esto nos proporciona la ilusión de que lo vemos todo en detalle. El concepto de Visión Activa, reconoce que solamente existe esa ilusión, en realidad el sentido de la vista es pobre y limitado. Sólo es gracias a todas las asociaciones y experiencias pasadas el motivo por el cual le podemos dar significado.

Debido a que en una computadora resulta complejo aprender mediante la experiencia se utiliza para simular la memoria humana una serie de imágenes que componen nuestra base de datos inicial, este conjunto de imágenes son en realidad los objetos que nuestro sistema puede reconocer. En este trabajo se utilizaron solamente dos objetos diferentes para encontrar dentro de la escena: una cajetilla de cigarros y una caja de medicina (Figura 26). Fueron seleccionadas para el desarrollo de este módulo debido a que cuentan con las suficientes características necesarias para poder obtener los Keypoints suficientes para garantizar el reconocimiento y analizar lo que "observa" la cámara.

Figura 26. Objetos que reconoce el sistema

El módulo de reconocimiento necesita de una base de datos de imágenes para tener referencia de los objetos que el sistema tomara como conocidos, esta característica nos permite agregar o cambiar los objetos que se identificaran dentro de la imagen. En la práctica solo se tomaron dos objetos a reconocer debido a que al tratar de simular la memoria asociativa se compara la imagen tomada en tiempo real contra las imágenes almacenadas en la base de datos, es decir, entre más objetos se incluyan en la base de datos el algoritmo tardara más tiempo comparando si en la escena se encuentra algún objeto con las características similares a las que el sistema puede identificar.

El algoritmo se divide en varias funciones que dividen el trabajo, esto facilita e ilustra el cálculo necesario para el reconocimiento de objetos. Cuenta con una función principal en la que se centra todo el contenido, esta función hace la conexión con los módulos que realizan una operación específica, estos módulos pueden contener dentro de ellos otros módulos para finalizar su tarea y de ser necesario cada uno de estos segmentos de código regresan los valores que necesitan la función dependiente para seguir su ejecución. A continuación se describe cada una de las partes que componen el sistema de reconocimiento de imágenes.

Función principal (reconoce)

Esta función es la encargada de englobar todas las operaciones que son necesarias para decidir si dentro de una imagen adquirida como parámetro de entrada se encuentra algún objeto que el sistema reconoce. Al final esta función se transforma de lenguaje M nativo de Matlab a un ejecutable que podemos llamar desde cualquier otro programa o ejecutarlo desde línea de comandos. Más a delante se describirá el algoritmo en conjunto.

Sub modulo Foto

Como su nombre lo indica este sub módulo es el encargado de capturar el entorno exterior dentro de la región en la que se desplaza el robot cartesiano. Admite como parámetro de entrada la ruta donde se almacenara la fotografía, así como el nombre y el formato que tendrá la imagen, con esta información queda seleccionado el canal de video y la resolución de la cámara que utilizaremos, el motivo de ajustar estos valores es que se puede tener montada más de una cámara y cada una de ellas puede aceptar valores de resolución diferente. La cámara que empleamos soporta una definición de 1024 x 768, establecidos estos parámetros, se inicializa el canal y se toma una instantánea, después se limpia el canal de video y se guarda la captura para su posterior comparación.

Sub modulo para cargar base de datos

Este módulo recibe como parámetros la ruta donde se guarda la imagen, el nombre del archivo, un cache que determina que operación se realizara, los intervalos y octavos para el algoritmo de SIFT en caso de que se aplique. La función de este módulo es obtener los puntos clave dentro de una base de datos de descriptores o aplicando el algoritmo SIFT a nuestra base de datos de imágenes con el fin de guardarlos en matrices para su posterior manejo.

Lo primero que se realiza es verificar el valor del cache, si el valor es uno revisa si se encuentra el archivo "nombre_imagen.key" que almacena los descriptores de la imagen que se da como parámetro de entrada, si existe se manda llamar otra función (read_keypoint_file) esta función carga el archivo que contiene los descriptores y le aplica un mascara para filtrar los puntos clave, para finalizar la función carga la imagen que se representa por una matriz de números, los descriptores y la posición de cada uno de ellos para regresarla a la función que la mando llamar. En caso de que el cache sea uno y el archivo nombre_imagen.key no exista entonces se aplica el algoritmo SIFT a la imagen de entrada, el algoritmo nos arrojara la representación de la imagen, los descriptores y su ubicación. Al haber obtenido esta información será necesario guardarla, esto se hace con el fin de ahorrar tiempo en caso de necesitar la información de las características locales de la imagen a procesar, si hacemos esta acción ya no tendremos que aplicar el extractor de características a cada imagen cada vez que se quiera comparar una escena con las imágenes de nuestra base de datos. Se manda llamar a la función write_keypoint_file la cual escribe los descriptores en un archivo, recibe como parámetros de entrada el nombre del archivo donde se guardaran los descriptores así como la ruta donde se encuentra. Automáticamente se agrega la extensión .key.mat, la posición una matriz N×2 que contiene las coordenadas (x,y) de los descriptores almacenados en columnas, una matriz N×3 con las filas que describe la escala de cada punto clave (i.e., la primera columna especifica los octavos, la segunda columna especifica el intervalo, y la tercer columna especifica el sigma). Un vector Nx1 que contiene la orientación de cada punto clave y por último los descriptores una matriz Nx128 con columnas que contienen las características de los correspondientes puntos clave.

Si el cache tiene un valor igual a dos entonces se ignora si existe el archivo que guarda los descriptores. Los puntos son calculados nuevamente y un nuevo archivo .key.mat es recreado, al igual que con la opción anterior se guarda la información de las características para mejor eficiencia. Si el cache es 3 se aplica una versión del extractor de características desarrollado por Lowe, en esta condición la información de los descriptores es almacenada temporalmente hasta que termina la ejecución del programa.

Este es uno de los dos componentes más importantes del módulo de visión y la función principal es recolectar la información que se tiene de nuestra base de datos de imágenes. En caso de que no exista el archivo que guarda las características de alguna imagen dentro de nuestra BD entonces se calculan los descriptores y se almacenan, esta información es pasada al siguiente bloque para su posterior comparación.

Submodulo de comparación

Los dos primeros bloques (sub modulo foto y sub modulo que carga la base de datos) son los encargados de obtener las características locales de la imagen proveniente del entorno en el que se desplaza el sistema de posicionamiento y de nuestra base de datos. Teniendo esta información se procede a comparar la imagen capturada por la cámara con las que se encuentran en la base de imágenes con el fin de determinar si existe en la escena un objeto que coincida con alguna imagen que el módulo reconozca.

Esta función junto con el algoritmo de SIFT son las que demandan mayor tiempo de ejecución. Para el desarrollo de este proyecto se contó con una computadora con procesador CORE i7 aunado a que afortunadamente matlab soporta la computación multihilos se pueden usar algunas funciones que incorpora esta herramienta para mejorar algunos procesos. En MATLAB basta con especificarle a una función el número de hilos que utilizaremos y automáticamente se refleja en algunas funciones las cuales se ejecutan más rápidamente. Se estableció como valor máximo de hilos cuatro, la función admite como parámetros de entrada la información de los descriptores de la imagen capturada con la cámara y de una imagen dentro de la base de datos. Como parámetros de salida proporciona el número de coincidencias existentes entre las imágenes. Se establece el valor del radio para determinar al vecino más cercano, entre menor sea se encontraran pocos Keypoints, esto hace que se tenga mayor precisión al descartar Keypoints que no estén en el rango. Se comprueba si el vecino más cercano tiene un valor menor a dos veces la distancia del radio en caso de ser cierto se coloca un uno en el contador de las correspondencias encontradas. En caso contrario se coloca un cero, al terminar de recorrer todos los Keypoints se suman aquellas celdas que contienen un valor diferente de cero. Este valor se retorna a la función principal. Como resultado final se dibuja la línea entre el descriptor encontrado en la imagen a identificar y la imagen en nuestra base de datos. Obteniendo los resultados de la Figura 26.

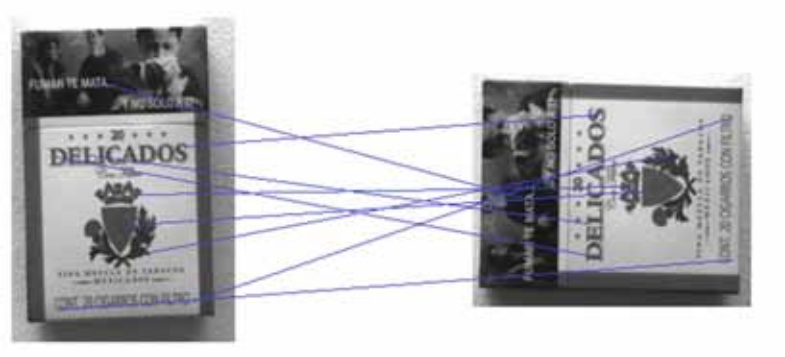


Figura 26. Correspondencias entre la imagen capturada y la almacenada en la base de datos de imágenes

Sub modulo para obtener el centro de la imagen

Una vez que se tiene la hipótesis de que el objeto puede ser reconocido, hay que hacer que la pinza de sujeción del eje Z del apunte al objeto que encontrado dentro del área de en donde está. El sub módulo de comparación realiza esta tarea:

Con la información obtenida de filtrar las características mediante el algoritmo del vecino más cercano, este método obtiene conjuntos agrupados de descriptores, al tener una base de fondo blanco la mayoría de las características residen en el lugar donde se encuentre un objeto, una vez que se obtiene los descriptores se filtra la lista de índices de las coordenadas (x, y), se eliminan los ceros para limpiar las celdas que no contienen información y se calcula la media de los puntos. Esta forma de calcular el centro de la imagen nos da una coordenada aceptable para establecer el lugar dentro del área en donde se encuentra el objeto.

Descripción General del modulo

La función principal engloba los sub módulos antes mencionados y es el encargado de encapsular todas las funciones necesarias para el reconocimiento de objetos, la primera instrucción que se realiza es limpiar el entorno de trabajo, después para tener una referencia de donde buscar los archivos requeridos se agrega la ruta de almacenamiento de la base de datos de imágenes (Figura 27).

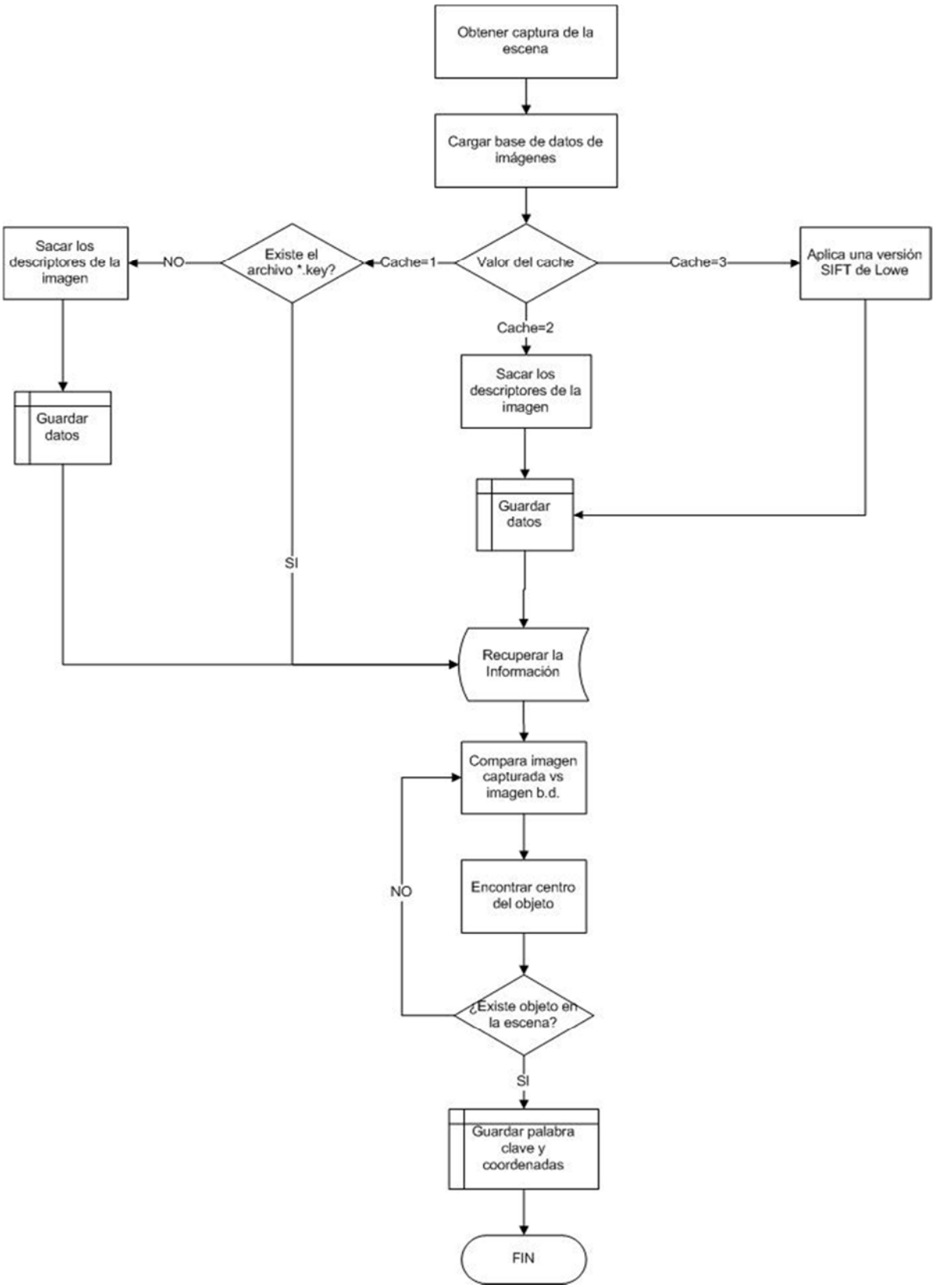


Figura 27. Diagrama de flujo del sistema de reconocimiento de objetos con el robot cartesiano.

Enseguida se crea un documento de texto llamado conexión.txt, al cual se le otorgan permisos de escritura; para establecer comunicación entre Visual C++ y MATLAB. Se establecen los octavos, los intervalos y el cache que se utilizaran para obtener las características locales de la imagen, los cuales se establecen con los valores 4, 2 y 1 respectivamente.

Para la declaración de variables se necesita un contenedor para almacenar el nombre de las imágenes de nuestra base de datos, así como también variables para almacenar la imagen, los descriptores, sus características y las correspondencias que se encuentren entre ambas imágenes. Para almacenar los nombres de las imágenes se guardan en un arreglo de cadenas, asignado cada una de las imágenes a usar. Para almacenar los descriptores MATLAB tiene un tipo de dato especial para guardar este tipo de información que son los llamados arreglos de matrices, cada celda de este arreglo nos permite guardar una matriz M x N de diferente tamaño para cada localidad.

Con el entorno inicializado se llama a la función foto la cual se encarga de capturar la imagen de la escena en el que se encuentra el eje Z, esta fotografía se guarda en el directorio donde se almacenan todas las imágenes con el nombre de captura.jpg. Se cargan las imágenes de la base de datos mediante un sub modulo y la información de las características locales de cada imagen es guardada en su respectivo arreglo de matrices, se realiza una iteración hasta el número de veces el tamaño del arreglo de imágenes. Ya que se obtiene la información de la base de datos solamente resta obtener los descriptores de la imagen capturada. Se aplica el algoritmo SIFT a la imagen y se guarda la información en un arreglo de matrices.

Con los resultados obtenidos con el algoritmo SIFT aplicado a todas las imágenes se busca verificar que dentro de la captura se encuentra un objeto y que este sea reconocido, el módulo de comparación se encarga de obtener las correspondencias entre la imagen proveniente de la cámara y el nombre que corresponde a cada celda de nuestro arreglo de caracteres en donde se especifica el nombre de la imagen con la cual se compara. Se realiza una iteración comparando la imagen contra cada una de las fotografías de nuestra base de datos, en cada comparación se comparan los descriptores y se almacenan las correspondencias, se manda llamar al módulo centrar objeto para obtener las coordenadas del objeto cuando ya fue reconocido, por lo general si no existe objeto reconocido en la escena, entonces, no hay correspondencias.

De estos dos sub módulos (comparar y centrar objeto), se obtiene la información del número de correspondencias encontradas y las coordenadas (x, y) de una ubicación posible

del centro del objeto. Como se están comparando cada imagen dentro del ciclo. Para necesitamos ahorrar tiempo y no tener que recorrer toda la base de datos se aplica el siguiente criterio; si hay un número mayor a 50 coincidencias, entonces, dentro de nuestra escena está un objeto reconocido, en el archivo se marca con la palabra objeto y las coordenadas corresponden a la ubicación del objeto, de esta forma se establece la comunicación con el modulo que manipula el robot cartesiano, al momento de marcar el ciclo se interrumpe, se cierra el archivo y se limpia todas las variables del módulo.

Para la integración del módulo de visión desarrollado en MATLAB con el módulo que controla al robot cartesiano se utilizó la herramienta de implementación deploytool, ya que es una forma sencilla de unir todas las funciones de un proyecto (Figura 28) para que la herramienta compile nuestros archivos y como resultado final un archivo ejecutable.

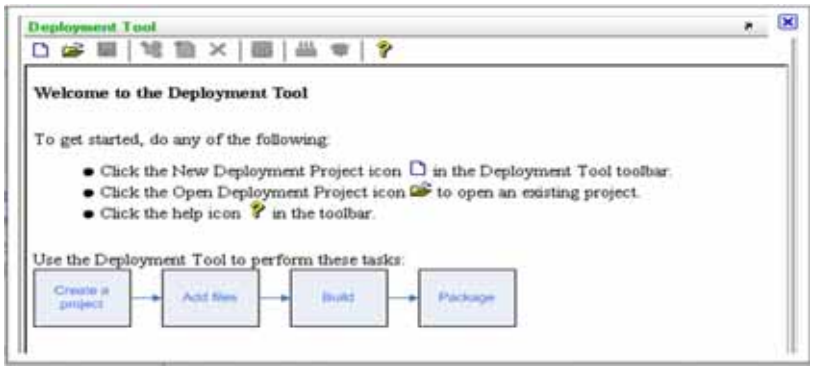


Figura 28. Herramienta de implementación de Matlab

Módulo de control del robot cartesiano

Para el desarrollo de este módulo se usó la biblioteca del software propietario para manejar dispositivos de la marca Parker, esto permite crear aplicaciones en un entorno grafico mediante las definiciones de los métodos para manejar el controlador maestro.

Por lo general cuando se desarrolla una interfaz visual se tiene que emplear la programación orientada a eventos, debido a que se desconoce cuál será la primera instrucción que suceda durante la ejecución ya que depende de las acciones realizadas por el usuario, a diferencia del módulo de reconocimiento de objetos donde se desarrollaron

varios sub-módulos para dividir las tareas, para la creación de este bloque solo se crea un solo archivo que contiene a todas las funciones. En este conjunto de funciones se busca que en el eje Z se explore el área que le sea posible recorrer, para buscar de forma automática objetos que coincidan con alguno de la base de datos.

El sistema de posicionamiento tiene un rango de desplazamiento en el eje X como en el eje Y de (-24,24) cada unidad es equivalente a un centímetro, es decir entre el segmento positivo y el negativo tenemos un desplazamiento de 48 centímetros, la idea es dividir el área de búsqueda en regiones de igual tamaño así que se midió el marco que capturaba la cámara desde la posición donde se encuentra montada, lo que resulto en un rectángulo de 16 cm por 12 cm. Al dividir el área de exploración por el tamaño del marco capturado por la cámara fueron obtenidas 12 regiones para realizar la búsqueda. Con 12 regiones es posible tener en forma predefinida las coordenadas de cada segmento y tener un sistema de posicionamiento relativo, es decir nuestra referencia siempre será la posición actual de la región en la que nos encontremos (Figura 29).

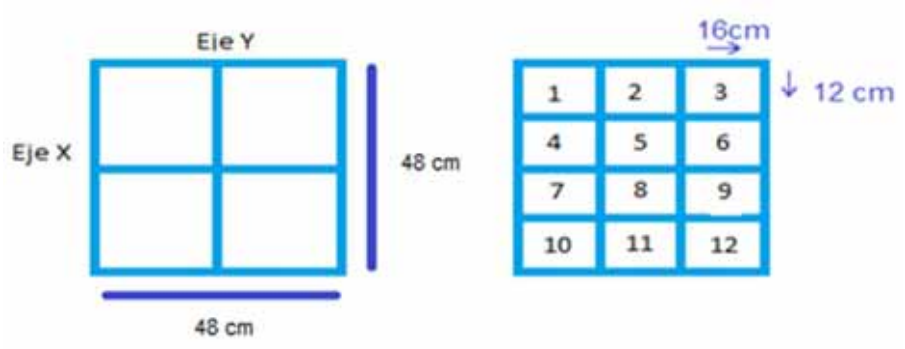


Figura 29. Regiones del área de búsqueda

El tipo de posicionamiento usado permite marcar el área que se recorrió evitando repetir la búsqueda en una región ya explorada. Para el control del robot cartesiano existe un software de calibración y así poder establecer la cantidad de pasos que se desplazara por cada instrucción de movimiento (corresponde al Comando GO del software del propietario Motion Planner). Para cada eje se calcula el número de pasos necesarios, para esto se emplean catálogos de los actuadores lineales [17] de donde se obtienen los datos de cuantos pasos por revolución se requieren para cada actuador, se calcula el inverso y se multiplica por la resolución del drive[18]. De lo anterior tenemos la los resultados siguientes.

	Distancia		Velocidad		Aceleración/Desaceleración		Etiqueta
	Unidades	Escala	Unidades/s	Escala	Unidades/s^2	Escala	
Eje 1	pasos	34246	rev/s	34246	rev/s^2	34246	Eje X
Eje 2	pasos	78740	rev/s	78740	rev/s^2	78740	Eje Y
Eje 3	pasos	78740	rev/s	78749	rev/s^2	78740	Eje Z

De esta manera quedan establecidos los pasos necesarios para que el actuador se desplace un centímetro en cualquiera de los tres ejes. Con la calibración y el tipo de posición a usar, a continuación se describen los eventos que sucederán en la ejecución del software, es decir las funciones que componen el módulo.

Función cargar

Esta función se encarga de almacenar la información que se analizó previamente. Almacena los datos correspondientes en cada segmento en el que se divide el área de exploración, cada bloque contiene 6 celdas que corresponden a los valores X e Y que componen las coordenadas de cada fragmento (Figura 30).

Los otros 4 valores son las referencias de los vecinos alrededor de la celda en la que se encuentra el robot cartesiano, los vecinos localizados diagonalmente se omiten debido a que hay que considerar la distancia más cercana a partir del punto donde se encuentra el robot: la cual es una distancia diagonal que implica una mayor distancia dentro de un malla de ubicaciones. Por este motivo solo se toman las celdas que se encuentran a los costados y los ubicados en la parte superior e inferior.

Con esta función se almacena la información mediante una lista ligada en el que cada región tiene dos parámetros para guardar las coordenadas y el recorrido a las demás celdas se hace mediante apuntadores a los vecinos de cada bloque. Este método implicaba crear otras funciones para obtener información guardada en memoria en cada bloque y manejar apuntadores representa aumentar el número de líneas de código.

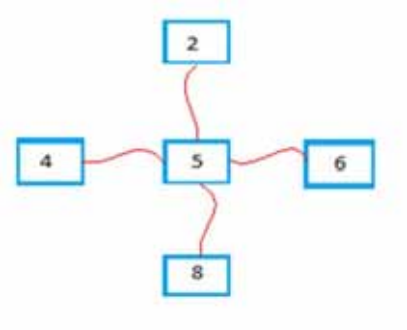


Figura 30. Segmento de una región de búsqueda con 4 apuntadores a sus celdas vecinas

Por lo que se tomó la decisión de crear mejor un arreglo de 6 localidades para almacenar los datos conocidos, así simplemente se recibe como parámetro el número de segmento que se recorrerá, se cargan los valores en el arreglo y se regresa a la función desde donde se llamó. Este arreglo de números puede ser pasado a otras funciones para realizar el movimiento, para marcar la región ya recorrida o para calcular la nueva región a donde se desplazara el robot.

Función reconoce

Uno de los aspectos importantes es sincronizar el bloque que reconoce los objetos con el módulo de control del robot cartesiano. Ambos bloques deben funcionar en forma coordinada para evitar adelanto o retraso en la captura de la imagen aun cuando continúe el movimiento en cualquiera de los ejes. Esta función ejecuta el módulo de visión artificial desarrollado en MATLAB y establece los parámetros indispensables para mantener en forma oculta la aplicación del reconocimiento de objetos. Para esto se usa una función nativa del sistema operativo llamada ShellExecute. Los parámetros de su estructura se modifican mediante una máscara para forzar que el proceso anterior espere a que termine de ejecutar la acción que se le indique. De esta manera, se establece una pausa en la búsqueda que realiza el robot. Es importante especificar la ruta en donde se encuentra el archivo. En un futuro se construirá un paquete de instalación por lo que es conveniente ubicar el módulo de visión en el mismo directorio del proyecto que busca objetos. De este modo, hay una ruta absoluta para todos los componentes de proyecto. El ultimo parámetro es una opción que especifica la forma en que se ejecutará el proceso, para que se realice la tarea de forma transparente, la ejecución se lleva a cabo de tal forma que no se despliega ninguna ventana o algún evento que indique que se está ejecutando un módulo.

Esta función no recibe parámetros solo es una llamada al módulo reconoce escrito en lenguaje M, hay que recordar que en caso de que se encuentre un objeto la función marca en un archivo de texto la palabra “objeto”, estos caracteres indican que se debe invocar a

otras funciones para ir en busca de alguna de las imágenes de la base de datos. También se obtienen las coordenadas del centro del objeto mediante el punto exacto en donde se encuentra.

Función siguiente

Esta función realiza la exploración de una forma automatizada. Una vez que se recorrió la primera región, calcula cual será la siguiente celda dentro del área a la cual se moverá el robot. La función admite como parámetros el arreglo que contiene las coordenadas y los 4 vecinos de la celda y una variable entera del número de la región en la que se encuentra el robot. Para evitar que se realice la búsqueda en una región antes explorada se cuenta con un arreglo que lleva el registro de las 12 regiones en las que se divide el área de exploración (Figura 29). Lo primero que realiza esta función es ubicar la celda que representa la región actual y establecerla como una región ya recorrida.

Con un segundo arreglo se almacena la distancia entre la región actual y uno de sus cuatro vecinos. Como cada vez que se ingresa a esta función se utiliza este arreglo, por lo que hay que limpiar el arreglo para quitar la basura. En seguida, se aplica la fórmula de la distancia euclidiana entre dos puntos y se aplica a todos los puntos vecinos validos alrededor del punto donde se encuentra el robot, se omiten aquellas regiones que ya fueron recorridas y aquellas celdas que representan a los vecinos cuyo valor sea igual a cero. Por ejemplo, la región que solamente tiene vecinos a su derecha y en la parte inferior pero no en la parte superior y el lado izquierdo, i. e. no hay celdas colindantes. Con esto se ahorra tiempo durante el proceso. Para realizar lo anterior, con el arreglo de entrada que contiene el número de la región de los 4 vecinos se usa la función Load para cargar las coordenadas del vecino que se está analizando, se aplica la fórmula de la distancia y el resultado se almacena para ser comparado posteriormente. En caso de que no haya un vecino o ya se hubiera visitado se coloca el valor de 100 para indicarnos que no se realizó ningún cálculo. Este número es una referencia para dos posibles casos que pueden suceder si las cuatro celdas para almacenar la distancia entre dos puntos equivalen a este valor, el primero de los casos es que la trayectoria que se está siguiendo el robot llegó a un punto en donde no es posible visitar ninguna región debido a que todos los vecinos alrededor ya fueron visitados. La segunda opción sucede cuando se terminó de explorar todas las regiones del sistema y no hay posibilidad de elegir otra región (Figura 31).

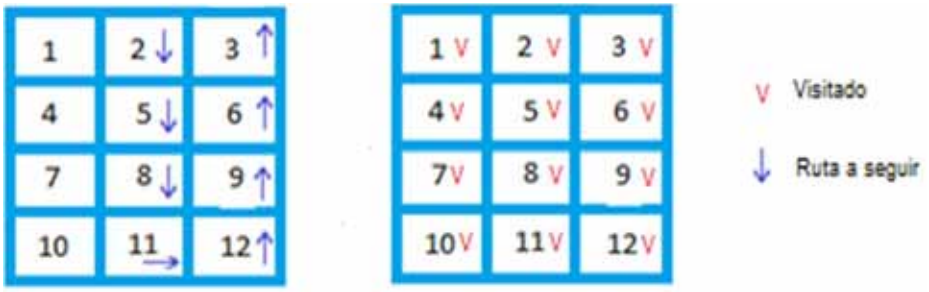


Figura 31. Casos cuando los vecinos alrededor de una región ya fueron visitados

Para resolver este inconveniente se crea un ciclo en donde se revisa el arreglo que lleva el registro de las celdas en donde ya se realizó la exploración, al encontrar una región a la cual no se ha visitado inmediatamente regresa el valor a la función que la manda llamar, en caso de que no encuentre ningún bloque al cual pueda desplazarse debido a que todas las regiones ya están marcadas, se escribe en el archivo de conexión la palabra "fin", esta es la orden para avisarle al sistema que ya se ha recorrido toda el área de trabajo y que debe de finalizar su ejecución.

En caso de que no todas las distancias sean igual a 100, se comparan los resultados de las celdas para encontrar el valor mínimo, el resultado es asociado a la posición de la celda (Figura 30), es decir, el valor de la celda superior se guarda en la celda 1, la distancia del vecino de la izquierda se almacena en la celda 2, el de la derecha en el tercer espacio y por último la distancia del vecino localizado en la parte inferior se localiza en la última celda del arreglo, el valor mínimo es regresado indicando la región siguiente a explorar.

La Figura 32 muestra los procesos que se realizan para automatizar el movimiento de los tres ejes en busca de objetos, al finalizar el reconocimiento de objetos en una región este bloque regresa la siguiente celda a explorar, esta función también determina la finalización de todo el modulo que controla el sistema de movimiento.

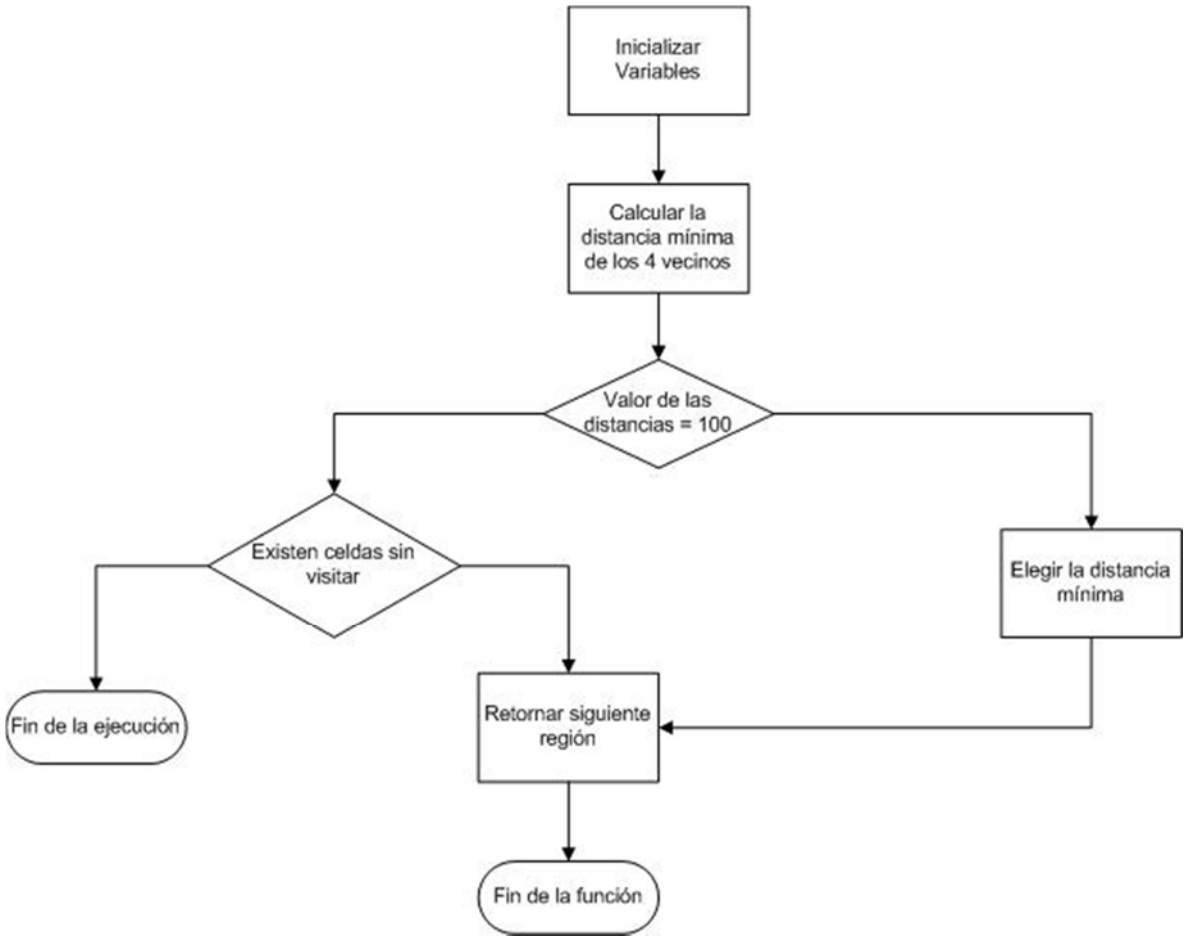


Figura 32. Diagrama de flujo la función siguiente

Función centrar

Como su nombre lo indica esta función se encarga de señalar el centro del objeto que ha sido reconocido, obtenemos la coordenada del centro de la imagen a partir del módulo reconoce, este punto dentro de la imagen proporcionada por la captura de la cámara está dada en pixeles, esto hace que sea necesario una forma de calcular el movimiento del sistema para ubicarse debajo de la posición requerida debido a que el sistema está calibrado en centímetros. Por este motivo es conveniente realizar una regresión lineal para establecer la relación que existe entre los pasos de motor y los pixeles, es decir cuantos pasos de motor son necesarios para desplazarse un pixel dentro de la imagen.

Para este proceso fue necesario desarrollar dos programas, uno desarrollado en MATLAB para realizar las capturas de la escena en donde se encuentra un objeto conocido y el segundo en visual C++ realiza el movimiento de N pasos de motor. Se sincronizan los dos programas para que realice un ciclo que se repite diez veces, se captura el objeto y en seguida se mueve la posición (en el eje X o en el eje Y) los pasos de motor que se le indiquen. Con las diez muestras donde varía la posición del objeto dentro de la imagen, se aplica el algoritmo SIFT a cada una de ellas, se almacena la información de los descriptores para aplicar otro ciclo y restar las coordenadas de los Keypoints, cada repetición del bucle se resta las características de la imagen con las imágenes restantes. De esta forma, se tiene el cambio de posición del objeto en coordenadas con respecto al tamaño de la imagen (Figura 33).

Figura 33. Capturas de nuestro objeto con un desplazamiento de 1000 pasos de motor.

Las coordenadas se almacenan en columnas, se filtra la lista de índices, se eliminan los espacios en blanco y se separan las coordenadas en columnas de cada imagen. Se obtiene la media aritmética de cada columna para comparar los resultados. Como el movimiento solo se aplica a un solo un eje, los resultados son lineales, para comprobar esto se grafican las medias de las diferencias por los pasos de motor, lo que se espera de esta impresión es una gráfica lineal (Figura 34).

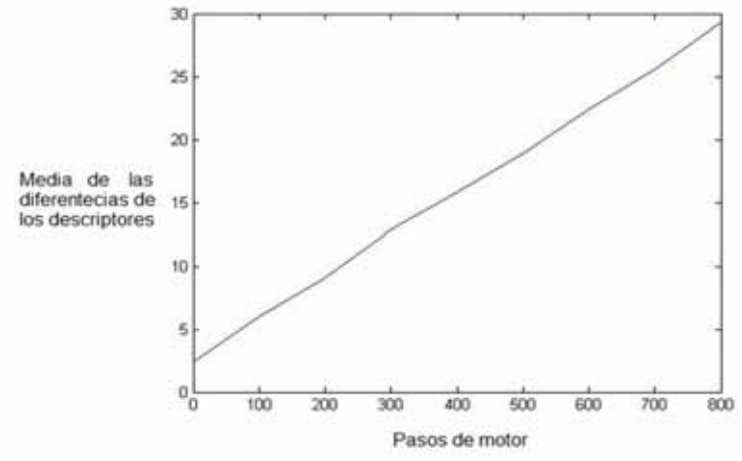


Figura 34. Grafica que representa la media aritmética de las diferencias por los pasos de motor

Si los resultados anteriores son verdaderos, se procede a aplicar una regresión lineal para obtener la relación de los pasos de motor necesarios para desplazarse un pixel dentro de la escena. Para crear un modelo de regresión lineal, es necesario que se cumpla con las siguientes hipótesis:

1. La relación entre las variables es lineal.
2. Los errores en la medición de las variables explicativas son independientes entre sí.
3. Los errores tienen varianza constante.
4. Los errores tienen una esperanza matemática igual a cero.
5. El error total es la suma de todos los errores.

Todo lo anterior lo satisface el sistema, entonces, se calcula el polinomio p que ajusta a los datos. El ajuste proporciona la relación pasos de pixeles por motor, adicionalmente se requerirá un coeficiente de correlación para medir la relación lineal entre las dos variables y mediante este dato asegurar que los resultados son fiables, lo anterior escrito en lenguaje M se obtiene de la siguiente manera.

```
p = polyfit(x,y,1);
R = corrcoef(x,y);
out = [p, R(1,2)];
```

donde "x" son las diferencias de X o Y y "y" los pasos de los motores en donde la diferencia fue registrada. El valor “p” que obtenemos da la relación de pixeles por pasos de motor y para obtener pasos por pixeles se calcula el inverso de "p". De “R” se tiene la intercepción y la correlación, estos deben ser < 1 y cerca de 1 respectivamente. Este

proceso se realizó para movimientos en el eje X, Y e XY (la diagonal), con diferentes pasos de motor, a continuación se muestran los resultados de este proceso.

Tabla de los resultados obtenidos al aplicar una regresión lineal

Pasos de motor	Desplazamiento del eje	
	X	Y
1000	*Slope=m: 603.218413* *Intercept=b: 3.073361* *Correlation: 0.999893*	*Slope=m: 1474.380592* *Intercept=b: 60.098747* *Correlation: 0.997374*
2000	*Slope=m: 602.173225* *Intercept=b: 127.788381* *Correlation: 0.999976*	*Slope=m: 1385.564137* *Intercept=b: 116.512066* *Correlation: 0.997369*
3000	*Slope=m: 612.542758* *Intercept=b: 1715.225212* *Correlation: 0.999990*	*Slope=m: 1413.599530* *Intercept=b: 369.213394* *Correlation: 0.998074*
4000	*Slope=m: 595.113613* *Intercept=b: 1747.549677* *Correlation: 0.999859*	*Slope=m: 1400.743214* *Intercept=b: -55.733428* *Correlation: 0.999173*
5000	*Slope=m: 607.904034* *Intercept=b: -19.500838* *Correlation: 0.999992*	*Slope=m: 1388.090015* *Intercept=b: 14.812633* *Correlation: 0.999832*
10000	*Slope=m: 608.080531* *Intercept=b: 1880.016147* *Correlation: 0.999985*	*Slope=m: 1387.278051* *Intercept=b: 274.570603* *Correlation: 0.999971*

De estos datos se calcula el promedio para obtener los pasos de motor necesarios para un desplazamiento de un pixel. El resultado obtenido fue X= 604 pasos y Y= 1408 pasos para un desplazamiento de un pixel dentro de la imagen. Con las coordenadas del centro de la imagen y la relación de pasos de motor por pixeles. Como el sistema de posicionamiento es relativo al centro de la imagen la cual es la posición de referencia, hay que dividir la imagen en cuatro partes donde se manejan valores negativos y positivos en cada uno de los cuadrantes.

Dado que las operaciones realizadas están dadas por la ubicación del pixel, los cuadrantes son numerados como se muestra en la Figura 35. Para realizar el movimiento basta con restar la coordenada de referencia contra la que se obtuvo del centro del objeto, dentro de la función dependiendo en que cuadrante se encuentra el robot. Se define el sentido en el que se moverá cada eje, se efectúa el movimiento y el eje Z que contiene el gripper, este baja para señalar la ubicación del objeto.

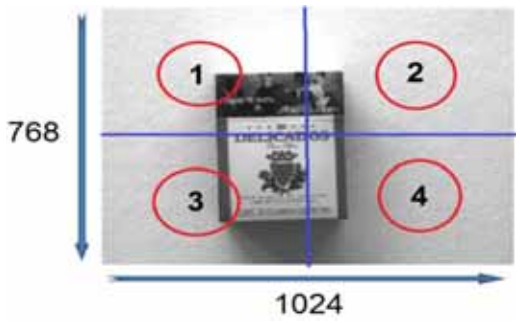


Figura. 35, Sentido de las regiones en las que se divide la imagen

Función habilitar botones

La interfaz desarrollada para este proyecto corresponde a una ventana que contiene únicamente tres botones (Figura 36). Al iniciar la aplicación solo está habilitado la opción conectar mientras que los otros dos botones restantes se encuentran deshabilitados, esto con el fin de evitar errores durante la implementación.

Es fundamental establecer comunicación con el servidor COM6SRVR para iniciar el movimiento. En el momento en que se presiona el botón Conectar la función es invocad y se encarga de establecer los parámetros de velocidad, aceleración, distancia y la escala. También, aplica la escala que se utilizara, se habilitan los drives que manejan los ejes y se establece la posición inicial en home. Para finalizar se deshabilita el botón Conectar y se habilitan los botones restantes.

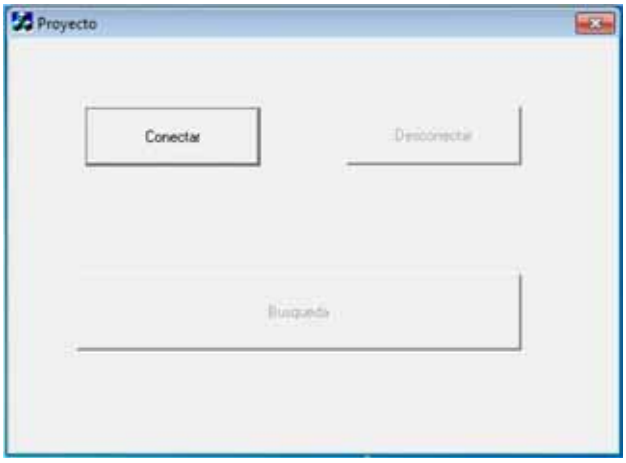


Figura 36. Interfaz visual del proyecto.

Botón Iniciar

Este botón establece la conexión con el drive 6k, que es el encargado de controlar todos los dispositivos y llevar el control del movimiento del sistema. Además mediante la función habilitar botones es posible establecer los parámetros iniciales para poder realizar la búsqueda. El método de conexión se realiza a través de una pequeña red que fue creada mediante un enrutador (router), por lo que se especifican valores para la conexión. Se guarda una variable con la dirección IP que se administró al controlador maestro, enseguida se crea una variable objeto para establecer la comunicación, ya que la biblioteca está implementada mediante objetos OLE. Si fue creada la variable, entonces, se establece una petición de conexión pasando la cadena de caracteres que contiene la dirección IP, el servidor de comunicación proporciona una respuesta booleana para tener conocimiento sobre el estado de la conexión. Si es verdadera se invoca a la función habilitar botones. Si la respuesta fue falsa se envía un mensaje de error desplegando la información de no hay un dispositivo con esa dirección IP. Este mensaje de error sirve para detectar si el controlador está alimentado a la corriente o si se hubo modificación en la configuración de este dispositivo. En caso de que la variable objeto no se establezca se despliega un mensaje de error informando sobre esta situación. Para evitar este error es necesario inicializar las librerías OLE antes de crear objetos. La Figura 37 muestra el diagrama de flujo de este botón.

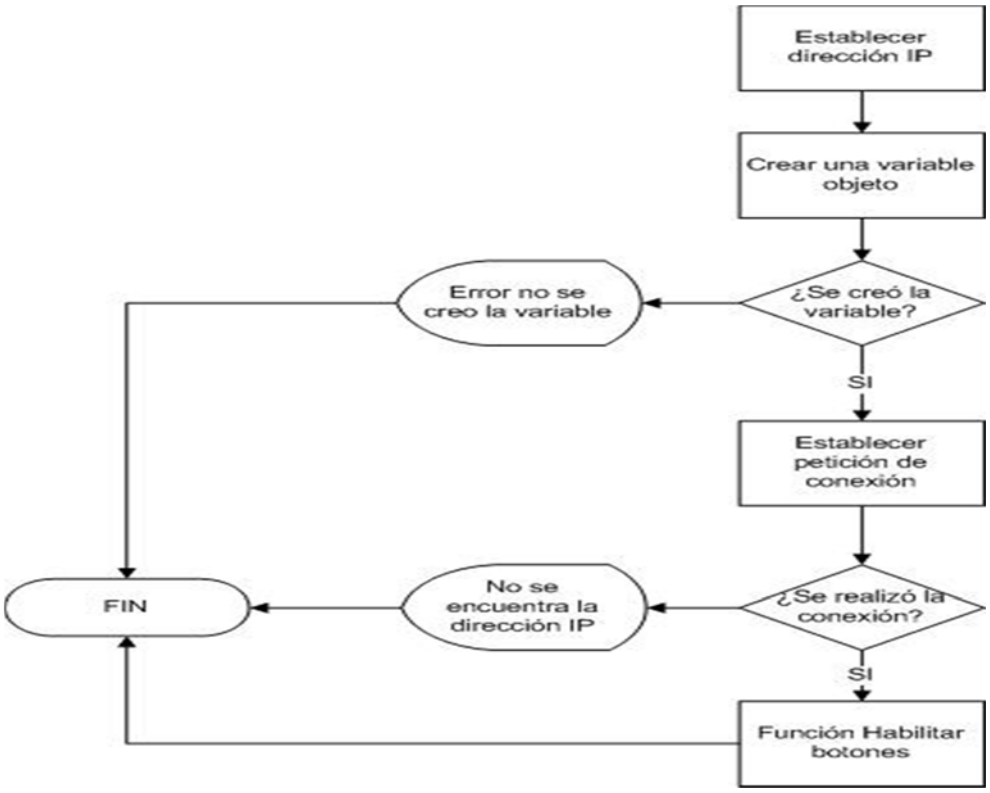


Figura 37. Diagrama de flujo del botón iniciar.

Botón Terminar

Como el servidor de comunicación es un ejecutable y no una dll (dynamic link library) permite establecer múltiples conexiones. Si existe más de una conexión el proceso no se cierra hasta que todas las estaciones cliente se desconecten, en muchos casos una correcta desconexión no se produce si existe un error no controlado en una aplicación cliente. Esto significa que se debe de tener cuidado al momento de terminar la comunicación con el servidor, es por lo que este botón es necesario para este proceso. La primera acción que realiza es deshabilitar los drives, con esto aseguramos que al momento de la desconexión no se estará efectuando ningún movimiento. Se procede a liberar la variable objeto que representa la conexión, se habilita el botón conectar y se deshabilitan los botones de búsqueda y desconectar. Para finalizar mandamos un mensaje en pantalla indicando que la desconexión se realizó sin problemas. La Figura 38 muestra el diagrama de flujo los procesos realizados para el botón terminar.

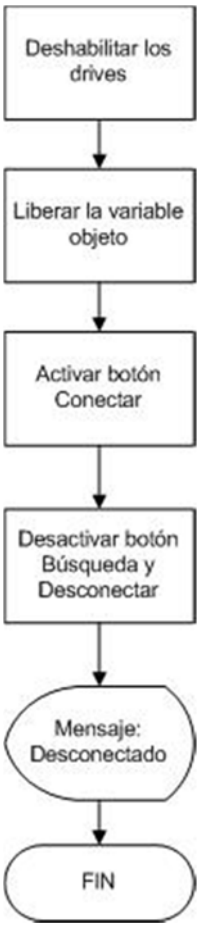


Figura 38. Diagrama de flujo del botón terminar.

Botón Buscar

Todas las funciones, módulos y botones antes mencionados se tienen que acoplar para que la búsqueda y el reconocimiento de objetos se automática. El botón buscar engloba todas las tareas necesarias para desarrollar esta tarea. La interfaz visual tiene la peculiaridad de que con este botón podemos ocultar todos los procesos realizados para que esté a nivel de usuario. Al presionar el botón, primero se declaran las variables necesarias. Hay un arreglo en donde se almacenan las coordenadas del centro de la imagen así como la orientación de los ejes X e Y, este arreglo se inicializa en ceros al comenzar el proceso y la comunicación inicializa con un conector en blanco.

Después, el sistema busca la primera región a visitar, para que no se repita la misma ruta cada vez que se realice la exploración en busca de objetos se usa un numero aleatorio. En la mayoría de los lenguajes de programación es difícil obtener el número al 100% debido a que el sistema toma el rango de los valores y los ordena en una secuencia predefinida cada vez que lo requiere. Por ejemplo, si se desea un número aleatorio entre 1 y 3, el primer

valor devuelto por la función podría ser 2, la segunda vez que se requiera el valor proporciona un 1 y para finalizar se obtiene el valor de tres. Pero si un valor se requiere más veces durante la ejecución se tiene la misma secuencia 2, 1 y 3.Para evitar tener una secuencia predefinida en la función se establece un valor que no sea constante, para esto se usa como semilla la hora actual de la máquina en donde se instaló el software, ya que no importa las veces que se requiera el número, la variable del tiempo cambia cada milisegundo. Con lo que no se obtiene la misma secuencia durante la ejecución.

Una vez que se tiene la primera región se itera con un ciclo que se repetirá durante 12 veces para recorrer todas las secciones del área de trabajo (Figura 31). Dentro del ciclo y con la información del segmento que se visitase invoca a la función cargar para obtener las coordenadas de la región así como los 4 vecinos que lo rodean. Por el sistema de referencia usado se calcula la distancia que el robot tiene que moverse del punto en el cual se encuentra; esto se realiza sumando a la región actual la coordenada del punto al que se moverá. Inicialmente, la posición es (0,0) y se suman las coordenadas de la variable aleatoria obtenida anteriormente: Con la distancia obtenida se ejecuta el comando de movimiento enviando los valores obtenidos de la función cargar, una vez que el controlador 6k recibe la instrucción distribuye el comando a los dispositivos para efectuar el cambio de posición. Después de unos segundos cuando el robot termine el movimiento en todos los ejes, se invoca a la función reconoce para realizar el reconocimiento de objetos.

Si la función reconoce encontró un objeto dentro de la escena, la función escribe la palabra objeto con sus respectivas coordenadas XY en el archivo de conexión. En caso contrario no se escribe ningún carácter. Se checa este archivo en busca de alguna palabra, en caso de que exista los caracteres y la primera palabra sea objeto, entonces, la coordenadas (x, y) y del centro de la imagen son enviadas como argumentos a la función centrar. La función centrar determina los pasos de motor necesarios para colocar el robot debajo del centro del objeto que encontrado. Enseguida, el eje Z regresa a su posición y una variable que lleva el conteo de los objetos encontrados se incrementa. Si al momento de revisar el archivo no se encontró la palabra clave objeto no se entra a la condicional y se salta al siguiente paso.

Con la primera vez que se realiza el reconocimiento de objetos, se calcula la siguiente región que se visitara. Se invoca a la función siguiente la cual regresa un valor entero de la coordenada que resulto como distancia mínima a la posición actual, en caso de que no exista la posibilidad de visitar otra región la función escribe la palabra fin en el archivo para que el sistema finalice todos los procesos. En caso contrario, se ejecuta el ciclo hasta recorrer todas las regiones.

Si cuando se checa el archivo de conexión se encontró la palabra “fin” se despliega un mensaje en pantalla informando que el recorrido de búsqueda finalizó, así como también el número de objetos encontrados, el ciclo termina en donde estaba y el robot se va a la posición en home (referencia inicial del sistema de posicionamiento establecida por los 3

sensores de posicionamiento). Se habilitan los botones Búsqueda y Desconectar para esperar a que el usuario determine la siguiente acción a realizar. Las funciones reconoce y siguiente son las únicas que pueden escribir en el archivo de conexión. Por lógica tenemos que ejecutar estas funciones en el orden en que se menciona antes de buscar información en nuestro archivo de texto de modo contrario podemos tener errores al momento de la implementación. En la Figura 39 se muestra el diagrama de bloques de este botón.

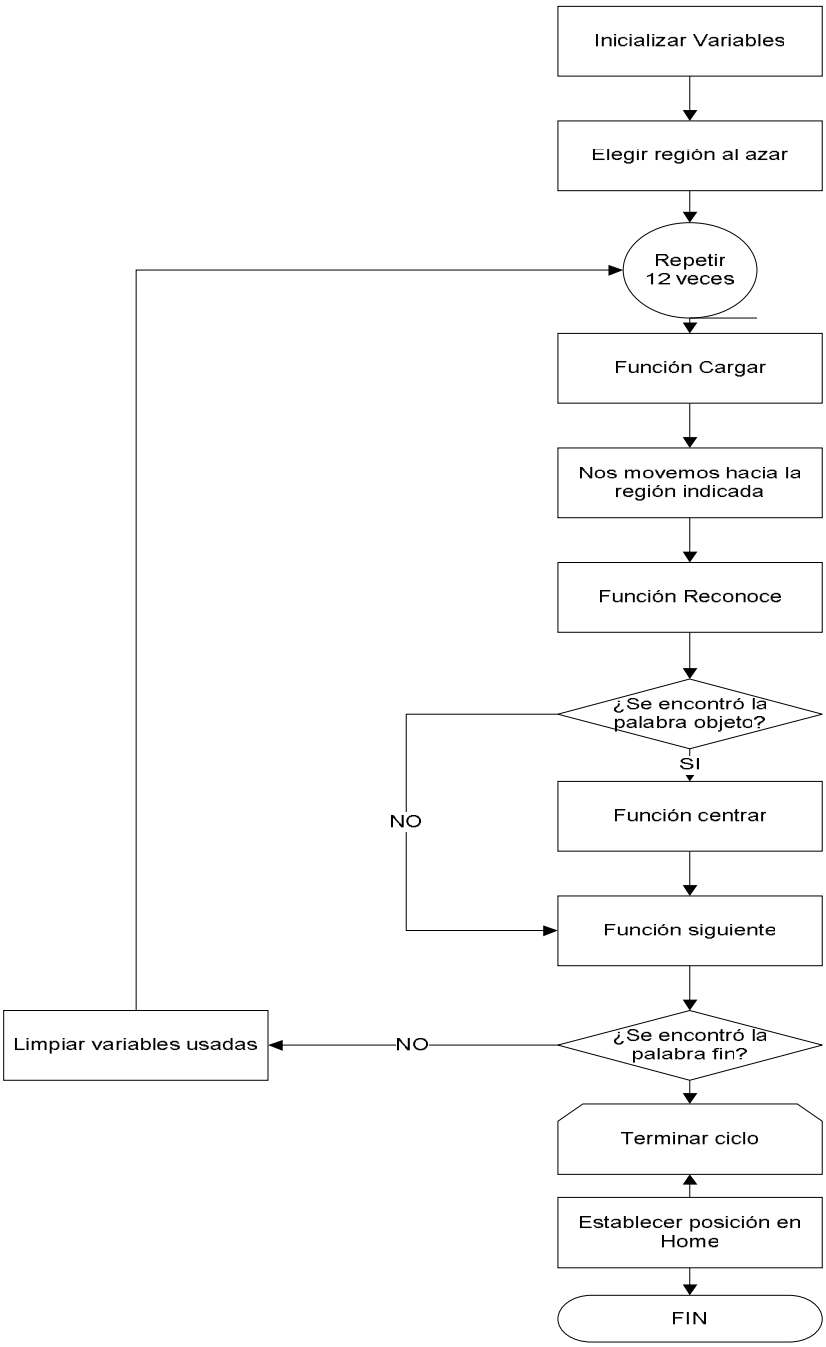


Figura 39. Diagrama de flujo del botón buscar.

Conclusiones

Como fue presentado el algoritmo SIFT es el que se encarga de obtener las características distintivas en la imagen resultó y resultó fundamental para el desarrollo de este proyecto. Específicamente, fue importante en el desarrollo del módulo de visión ya que se encarga de extraer características distintivas para comparar las dos imágenes: la capturada contra la almacenada en la base de datos. El conjunto de dichas características son los Keypoints, puntos de interés que son utilizados para reconocer unívocamente a un objeto. Este algoritmo para obtener los suficientes Keypoints depende de la condiciones de iluminación que se presenten al momento de capturar la escena así como de las propiedades de los objetos a identificar. Por lo que la escena tiene que ser preparada para evitar estas condiciones.

Por otra parte, se consiguió que el módulo desarrollado para el reconocimiento de objeto correspondiera con los paradigmas propuestos en inteligencia artificial. Este módulo se desarrolló como software inteligente capaz de reconocer una imagen en particular dentro de un conjunto almacenado en una base de datos. Esta parte es la que simula el proceso de visión realizado por los mamíferos desde el momento de obtener la luz (captura de la escena), pasarla al cerebro para decodificar la información (obtención de características) y finalmente asociar la escena con las experiencias que se obtienen del mundo (comparación con base de datos). Simulando este proceso es posible reconocer imágenes incluso con ocultación en los objetos.

Se diseñó una interfaz intuitiva procurando ocultar todas las capas de procesamiento. Cada una de ellas fue realizada ya sea en Visual C++ o en MATLAB. Los cuales se comunican por medio de mensaje para satisfacer con algunos de los requisitos para la interacción e interoperabilidad de aplicaciones de software. Por supuesto, el software fue construido de tal manera que puede ser modificado fácilmente dada la estructura empleada en su desarrollo. Como todo proyecto, aún con los resultados obtenidos es posible mejorar los dos módulos de programación que componen el sistema. La experimentación realizada se efectuó solamente para dos objetos dentro de una región y es posible identificar solo uno de los elementos omitiendo las piezas restantes, esto es una limitante cuando se encuentra un objeto. También, la interfaz visual puede ser modificada para mostrar la captura de la cámara en tiempo real o agregar componentes que hagan más atractiva la interacción con el usuario final.

En general los objetivos propuestos en este proyecto terminal se cumplieron. Una limitación del movimiento en el eje Z del robot cartesiano es que está acotado. Es en este eje en donde se encuentra la pinza de sujeción por lo que la forma en que un objeto es reconocido y agrupado en su respectiva clase se realizó centrando el eje Z en dicho objeto y así establecer su ubicación.

El sistema de posicionamiento automático del robot cartesiano puede ser extendido y complementado para el reconocimiento de más y distintos objetos. Este trabajo constituye la base para que en un futuro, el sistema pueda comprender estas características.

Apéndice

```
#include "StdAfx.h"
#include "Proyecto.h"
#include "ProyectoDlg.h"
#include <afxole.h>
#include <afxdisp.h>
#include <windows.h>

void CProyectoDlg::Load(int almacen[], int caso)
{
  switch(caso)
  {

    /* Carga los valores iniciales para cada una de las 12 regiones a visitar
          almacenamos los valores de los puntos (x,y) y los 4 vecinos de la region*/
        case 1:
     almacen[0]=-18; // X
            almacen[1]=16;  // Y
            almacen[2]=0;   // Arriba
            almacen[3]=4;   // Abajo
            almacen[4]=2;   // Der
            almacen[5]=0;   // Izq

        break;

        case 2:
            almacen[0]=-18;
            almacen[1]=0;
            almacen[2]=0;
            almacen[3]=5;
            almacen[4]=3;
            almacen[5]=1;

        break;

        case 3:
            almacen[0]=-18;
            almacen[1]=-16;
            almacen[2]=0;
```

```
        almacen[3]=6;
        almacen[4]=0;
        almacen[5]=2;

break;

case 4:
        almacen[0]=-6;
        almacen[1]=16;
        almacen[2]=1;
        almacen[3]=7;
        almacen[4]=5;
        almacen[5]=0;

break;

case 5:
        almacen[0]=-6;
        almacen[1]=0;
        almacen[2]=2;
        almacen[3]=8;
        almacen[4]=6;
        almacen[5]=4;

break;

case 6:
        almacen[0]=-6;
        almacen[1]=-16;
        almacen[2]=3;
        almacen[3]=9;
        almacen[4]=0;
        almacen[5]=5;

break;

case 7:
        almacen[0]=6;
        almacen[1]=16;
        almacen[2]=4;
```

```
        almacen[3]=10;
        almacen[4]=8;
        almacen[5]=0;

break;

case 8:
        almacen[0]=6;
        almacen[1]=0;
        almacen[2]=5;
        almacen[3]=11;
        almacen[4]=9;
        almacen[5]=7;

break;

case 9:
        almacen[0]=6;
        almacen[1]=-16;
        almacen[2]=6;
        almacen[3]=12;
        almacen[4]=0;
        almacen[5]=8;

break;

case 10:
        almacen[0]=18;
        almacen[1]=16;
        almacen[2]=7;
        almacen[3]=0;
        almacen[4]=11;
        almacen[5]=0;

break;

case 11:
        almacen[0]=18;
        almacen[1]=0;
        almacen[2]=8;
```

```
                almacen[3]=0;
                almacen[4]=12;
                almacen[5]=10;

            break;

            case 12:
                almacen[0]=18;
                almacen[1]=-16;
                almacen[2]=9;
                almacen[3]=0;
                almacen[4]=0;
                almacen[5]=11;

            break;

            default:
                    break;
   };

}

void CProyectoDlg::reconoce()
{
   /*Llamamos al modulo creado en Matlab para el reconocimiento de imagenes
     se ejecuta de manera silenciosa y detiene todos los procesos hasta terminar
     con su rutina          */
          SHELLEXECUTEINFO ShExecInfo = {0};
          ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);
          ShExecInfo.fMask = SEE_MASK_NOCLOSEPROCESS;
          ShExecInfo.hwnd = NULL;
          ShExecInfo.lpVerb = NULL;
          ShExecInfo.lpFile = "~\\reconoce.exe";
          ShExecInfo.lpParameters = "";
          ShExecInfo.lpDirectory = NULL;
          ShExecInfo.nShow =SW_HIDE;
          ShExecInfo.hInstApp = NULL;
          ShellExecuteEx(&ShExecInfo);
          WaitForSingleObject(ShExecInfo.hProcess,INFINITE);
}
```

```
int CProyectoDlg::siguiente(int almacen[], int num)
{
   /* Funcion que calcula la proxima region a visitar, se obtiene
            el arreglo con la coordenada y los vecionos de la region actual
            asi como el entero de region ya visitada  */

  int coordenada, j, aux1, menor;

  // Este arreglo maneja las regiones visitadas se marca con un 2 para llevar
  // el control de los segmentos del area de exploracion
   v[num]=2;

  //Limpiamos el arreglo donde almacenamos las distancias a los vecinos
 for(j=0; j==3; j++)
  dist[j]=55;


   // Calculamos la ditancia region arriba
  if(almacen[2]!=0 && v[almacen[2]]==false)
  {
          Load(calculos, almacen[2]);

          if (calculos[1]!=0)
                  calculos[1]=-calculos[1];

          dist[0]= abs(almacen[0] - calculos[0] + almacen[1] + calculos[1]);
  }

  else
         dist[0]=100;



        // Calculamos la ditancia region abajo
   if(almacen[3]!=0 && v[almacen[3]]==false)
  {
          Load(calculos, almacen[3]);
          if (calculos[1]!=0)
```

```
                calculos[1]=-calculos[1];

        dist[1]=  abs(almacen[0] - calculos[0] + almacen[1] + calculos[1]);
}
 else
        dist[1]=100;



        // Calculamos la ditancia region derecha
if(almacen[4]!=0 && v[almacen[4]]==false)
{
        Load(calculos, almacen[4]);
        if (calculos[1]!=0)
                calculos[1]=-calculos[1];

        dist[2]=  abs(almacen[0] - calculos[0] + almacen[1] + calculos[1]);
}
        else
        dist[2]=100;



        // Calculamos la ditancia region izquierda
 if(almacen[5]!=0 && v[almacen[5]]==false)
{
        Load(calculos, almacen[5]);
        if (calculos[1]!=0)
                calculos[1]=-calculos[1];

        dist[3]=  abs(almacen[0] - calculos[0] + almacen[1] + calculos[1]);
}
 else
        dist[3]=100;



        //si todas las distancias son 100, buscamos en visitados la region no visitada
        if(dist[0]==100 && dist[1]==100 && dist[2]==100 && dist[3]==100)
        {
                if(v[1]!=2 || v[2]!=2 || v[3]!=2 || v[4]!=2 ||
```

```
                v[5]!=2 || v[6]!=2 ||     v[7]!=2 || v[8]!=2 ||
                v[9]!=2 || v[10]!=2 || v[11]!=2 || v[12]!=2)
        {


        for(j=12; j>=1; j--)
        {
                if(v[j]==0)
                {

   coordenada=j;
                        break;}

        }
        }
        else
        {  //Si todos estan visitados
                fp1 = fopen("conexion.txt", "w+");
                fprintf(fp1, "fin");
                fclose(fp1);
                coordenada=0;
        }

}
    //Si no calculamos la distancia minima para ir a la siguiente coordenada

   else
{

        aux1=0;
        menor=dist[0];


        for(j=1; j<=3; j++)
        {
                if(menor >= dist[j])
                {menor=dist[j];
                 aux1=j;}

        }
```

```
                coordenada=almacen[aux1+2];

        }

        //Regresamos la siguiente region a explorar
        return coordenada;
}


void CProyectoDlg::OnBuscar()
{
        int dist1, dist2, n=0, d, xi=0,yi=0, x,y,l,m;

        //Limpiamos registro que almacena los pasos necesarios para centrar
        //la region despues de encontrar un objeto
        for(d=0;d<=3;d++){
                coordenadas[d]=0;}


 //pasos necesarios para moverse un centimetro
 calibracionx=34246;
 calibraciony=78740;

        //Limpiamos el archivo de conexion
  fp1 = fopen("conexion.txt", "w+");
        Sleep(100);
        fclose(fp1);

        //Deshabilitamos los botones de busqueda y desconectar
                        b_busqueda.EnableWindow(FALSE) ;
                        b_desconectar.EnableWindow(FALSE) ;

  //Elegimos la primer coordenada al azar
        ZeroMemory(linea, 1);
        seg=time(NULL);
        srand(seg);
        coordenada = rand () % 12 + 1;

        verdadero=true;
```

```
   //Limpiamos el registro que controla las regiones visitadas
         for(d=0; d<=12; d++)
          v[d]=0;

   //Repetimos el ciclo 12 veces
         for(d=0; d<=11; d++)
         //while (verdadero)
         {
         //Llamamos a la funcion Load
          Load(valores, coordenada);

          dist1= xi + valores[0];
          dist2= yi + valores[1];

    x= (coordenadas[0]/12)*(coordenadas[2]);
          y= (coordenadas[1]/18)*(coordenadas[3]);

         //Limpiamos la cadena que usamos para establecer los comandos hacia el
controlador
          ZeroMemory(cadena, 128);
          wsprintf(cadena, "scld%d,%d:", calibracionx + x, calibraciony + y);
         //Calibramos la mesa en cm
          k4.Write (cadena);

   //Establecemos la distancia a recorrer
         ZeroMemory(cadena, 128);
    wsprintf(cadena, "d%d,%d:", dist1, dist2);
         k4.Write (cadena);
         k4.Write ("go11:");


         //Dependiendo la distancia damos tiempo al drive de hacer su recorrido
          if(dist1>13||dist1<-13||dist2>17||dist2<-17)
          Sleep(10000);
          else
    Sleep(4000);


                for(d=0;d<=3;d++){
                coordenadas[d]=0;}
```

```
      //Ejecutamos el modulo de reconocimiento de objetos
       reconoce();

       //Abrimos el archivo de conexion en modo de lectura
       fp1 = fopen("conexion.txt", "r");
       fscanf(fp1,"%s", linea);

 //Si el primer caracter es "o" (de objeto)

       if (linea[0]=='o')
       {
              //centrar();
              //Sleep(1500);

        fscanf(fp1, "%s", palabra);
        eje1=atoi(palabra);

        fscanf(fp1, "%s", palabra);
        eje2=atoi(palabra);

        //Obtenemos el centro del objeto y llamamos a la funcion centar
              centrar(eje1, eje2);
              Sleep(1500);
              n=n+1;


       }
//cerramos el archivo de conexion
       fclose(fp1);

       //Aqui funcion que calcula siguente coordenada
       coordenada= siguiente(valores, coordenada);


       xi= -valores[0];

       if(valores[1]!=0)
        yi= -valores[1];
       else
```

```
    yi=0;


        //Lee archivo en busca de la palabra fin
        fp1 = fopen("conexion.txt", "r");
        fscanf(fp1,"%s", linea);


       //si es fin cerramos todo y mandamos un mensaje
        if (linea[0]=='f')
        {
                ZeroMemory(cadena, 128);
                wsprintf(cadena, "Recorrido Finalizado objetos encontrados: %d", n);
                MessageBox(cadena, "Valor", 0);
               verdadero=false;
               break;
        }




     //Volvemos a calibrar nuestro sistema
               ZeroMemory(cadena, 128);
               wsprintf(cadena, "scld%d,%d:", calibracionx, calibraciony);
               k4.Write (cadena);
               linea[0]='p';
               fclose(fp1);
        }


        //Al finalizar habilitamos los botones de busqueda y desconectar
        // Y establecemos el sistema en home
        Sleep(4);
        k4.Write("hom11xx:");
        b_busqueda.EnableWindow(TRUE) ;
        b_desconectar.EnableWindow(TRUE);
}


void CProyectoDlg::centrar(int eje1, int eje2)
```

```
{
 int  pasosx=0, pasosy=0;

 //pasos necesarios para moverse un pixel
 pixelx=604;
 pixely=1408;

 /*Dividimos la imagen en 4 partes iguales dependiendo la region del centro de la imagen
   nos movemos el sentido y direccion hacia el punto  */
          if(eje2<=384 && eje1<=512)
          {
                    pasosy= 384 - eje1;
                    pasosy= pasosy * pixely;

                    pasosx= 512 - eje2;
                    pasosx=pasosx * pixelx;

                    coordenadas[0]=pasosx;
                    coordenadas[1]=pasosy;
                    coordenadas[2]=-1;  //Sentido movimiento x
                    coordenadas[3]=1;   //Sentido movimiento y

                    ZeroMemory(cadena, 128);
                    wsprintf(cadena, "scld%d,%d:", pasosx, pasosy);
                    k4.Write (cadena);

                    k4.Write ("d-1,1,18:");
                    k4.Write ("go111:");
                    Sleep(6000);
                    k4.Write ("d0,0,-18:");
              k4.Write ("go001:");
                    Sleep(4000);

          }
          //--------------------------------------------------
          else if(eje2<=384 && eje1>=513)
          {
                    pasosy= 384 - eje1;
                    pasosy= pasosy * pixely;
```

```
            pasosx= eje2 - 512;
            pasosx=pasosx * pixelx;

            coordenadas[0]=pasosx;
            coordenadas[1]=pasosy;
            coordenadas[2]=-1;
            coordenadas[3]=-1;

            ZeroMemory(cadena, 128);
            wsprintf(cadena, "scld%d,%d:", pasosx, pasosy);
            k4.Write (cadena);

            k4.Write (cadena);

            k4.Write ("d-1,-1,18:");
            k4.Write ("go111:");
            Sleep(6000);
            k4.Write ("d1,1,-18:");
      k4.Write ("go001:");
            Sleep(5000);
}
//-------------------------------------------------
else if(eje2>=385 && eje1<=512)
{
             pasosy= eje1 - 384;
            pasosy= pasosy * pixely;

            pasosx= 512 - eje2;
            pasosx=pasosx * pixelx;

           coordenadas[0]=pasosx;
            coordenadas[1]=pasosy;
            coordenadas[2]=1;
            coordenadas[3]=1;

            ZeroMemory(cadena, 128);
            wsprintf(cadena, "scld%d,%d:", pasosx, pasosy);
            k4.Write (cadena);

            k4.Write ("d1,1,18:");
```

```
                k4.Write ("go111:");
                Sleep(6000);
                k4.Write ("d-1,-1,-18:");
            k4.Write ("go001:");
                Sleep(5000);
        }
        //-------------------------------------------------
        else if (eje2>=385 && eje1>=513)
        {
                pasosy= eje2 - 384;
                pasosy= pasosy * pixely;

                pasosx= eje1 - 512;
                pasosx=pasosx * pixelx;

                coordenadas[0]=pasosx;
                coordenadas[1]=pasosy;
                coordenadas[2]=1;
                coordenadas[3]=-1;

                ZeroMemory(cadena, 128);
                wsprintf(cadena, "scld%d,%d:", pasosx, pasosy);
                k4.Write (cadena);

                k4.Write ("d1,-1,18:");
                k4.Write ("go111:");
                Sleep(6000);
                k4.Write ("d-1,1,-18:");
            k4.Write ("go001:");
                Sleep(5000);
        }

                coordenadas[2]=coordenadas[2]*-1;
                coordenadas[3]=coordenadas[3]*-1;
}

void CProyectoDlg::Hab_botones()
{
    //Establecemos la aceleracion, velocidad y distancia incial
                    k4.Write("a10,10,10:");
```

```
                        k4.Write("v5,v5,v5:");
                        k4.Write("d1,1,-18:");
                        conectado = true;
                        //establcemos calibracion y habilitamos los 3 drives
                        k4.Write("SCLD 34246,78740,78740:");
                        k4.Write("drive111x:");
                        Sleep(500);
                        //Colocamos nuestro sistema en home
                        k4.Write("hom11xx:");
                        k4.Write("go001:");
                        Sleep(500);
                    //Mandamos un mensaje para indicar que la conexion se realizo
satisfactoriamente
                        MessageBox("Conexion correcta","Conectado", MB_OK |
MB_ICONINFORMATION);
                        Sleep(1000);
                        b_busqueda.EnableWindow(TRUE) ;
                        b_conectar.EnableWindow(FALSE) ;
                        b_desconectar.EnableWindow(TRUE) ;

}


void CProyectoDlg::OnIniciar()
{
        CString IPaddr;
   // Establecemos la direccion del controlador maestro
        IPaddr = "192.168.0.15" ;

   //Inicializamos la libreria OLE
        if (indicador==0)
        {AfxOleInit();
         indicador=1;}

  //Creamos la variable objeto para establecer la conexion
        if (k4.CreateDispatch("COM6SRVR.NET"))
        {
                // Deshabilitamos mensaje de servidor ocupado
                COleMessageFilter* pFilter = AfxOleGetMessageFilter();
                pFilter->EnableNotRespondingDialog(FALSE);
```

```
                //Si existe respuesta, mandamos la funcion Hab_botones
                if (respuesta = k4.Connect(IPaddr))
                {
                        Hab_botones();

                }
                else
                {       conectado = false;
                  //Si no desplegamos un mensaje de error
                        AfxMessageBox("No se encontro direccion IP");}

            // Habilitamos "Server Busy"
                pFilter->EnableNotRespondingDialog(TRUE);
                }

        else
                AfxMessageBox("No se creo el objeto COM");
}


void CProyectoDlg::OnTerminar()
{
 // Esta funcion deshabilita los drives habilita el boton conectar
 //Deshabilita los botones buscar y desconectar. Finalmente envia un mensaje
 //al usuario indicando que se realizo la conexion
        if (conectado)
        {
                k4.Write ("drive000x:");
                k4.ReleaseDispatch();
                conectado = false;
                b_busqueda.EnableWindow(FALSE) ;
                b_conectar.EnableWindow(TRUE) ;
                b_desconectar.EnableWindow(FALSE) ;
                MessageBox("Desconectado","Sin Errores", MB_OK |
MB_ICONINFORMATION);

        }
}
```

Referencias

[1] R. U. Muñoz Morales, C. G. Ordóñez Sánchez, S. E. Ordóñez Sánchez (2006) "Sistema Programable para el Control de Movimiento", Proyecto terminal en Ingeniería Mecánica 14903UAM Azcapotzalco.

[2] Gerardo Aragón Camarasa (2005) "Manipulación del robot cartesiano de la mesa XYZ" Programa en Visual Basic. PDPA, UAM Azcapotzalco.

[3] http://www.juntadeandalucia.es/averroes/~29701428/salud/ojo.htm

[4] E. Kandel. (1997). *Neurociencia y conducta*. Pearson.

[5] Documental National Geographic El cerebro.

[6] Documental El cerebro, el universo dentro de nosotros, percepción.

[7] I. D. García Santillán (2008) "Visión Artificial y Procesamiento Digital de Imágenes usando Matlab", Ibarra-Ecuador.

[8] R. C. Gonzalez and P. Woods, (2002) Digital Image Processing, Addison Wesley,

[9] G. Bradski and A. Kaehler (2008)"Learning Opencv", O´Reilly Media, Inc.

[10] http://www.um.es/geograf/sigmur/teledet/tema06.pdf

[11] F. Durand and J. Dorsey "Fast Bilateral Filtering for the Display of High-Dynamic-Range Images" SIGGRAPH 2002.

[12] David G. Lowe (2004) "Distinctive image features from scale-invariant keypoints"Int. J. of Comp. Vis.**60** (2), 91-110.

[13] R. Szelisky (2010)" Computer Vision: Algorithms and Applications", Springer Verlag.

[14] Lindeberg, T. (1994) Scale-space theory: a basic tool for analysis of structures at different scales. J. of Appl. Stat.**21** (2), 224–270.

[15] M.Brownand D. G. Lowe (2003). "Recognising Panoramas". Proceedings of the ninth IEEE International Conference on Computer Vision. 2. pp. 1218–1225

[16] Edelman, S., Intrator, N. and Poggio, T. (1997) “Complex cells and object recognition”manuscript: http://kybele.psych.cornell.edu/~edelman/archive.html

[17] Manual actuadores lineales serie ER. Parker. 1998

[18] 6k Series Hardware Installation Guide. Parker. 1998