UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA ELÉCTRICA – SISTEMAS ELECTRÓNICOS

RECONOCIMIENTO DE OBJETOS Y MANEJO DE UN BRAZO ROBÓTICO MEDIANTE UN PROCESADOR SITARA

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
ING. MAURICIO ONTIVEROS SALGADO

TUTOR PRINCIPAL
DR. JUAN MARIO PEÑA CABRERA, IIMAS-UNAM

MÉXICO, D. F. ENERO 2016

JURADO ASIGNADO:

Presidente:	DRA. NAVARRETE MONTESINOS MARGARITA
Secretario:	M.I. ALVAREZ CASTILLO JESÚS
Vocal:	DR. PEÑA CABRERA JUAN MARIO
1 er. Suplente:	DR. DE LA ROSA NIEVES SAÚL
2 d o. Suplente:	DR. PRADO MOLINA JORGE

Lugar o lugares donde se realizó la tesis: INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y EN SISTEMAS, UNIVERSIDAD NACIONAL AUTÓNOMA DE MEXICO

TUTOR DE TESIS:

DR. JUAN MARIO PEÑA CABRERA

FIRMA

A mis padres, Margarita y Benjamin, que han estado siempre a mi lado

A todos los que luchan por tener una sociedad justa y equitativa

Agradecimientos:

- A Dios por protegerme durante todo mi camino y darme serenidad en los momentos complicados

- A la Universidad Nacional Autónoma de México, por permitirme crecer en lo humano y profesional

- Al Dr. Juan Mario Peña Cabrera por todo su apoyo y atención en el desarrollo de este proyecto

- Al M. en C. Arturo Ocampo Álvarez por guiarme en mi vida profesional

- A los profesores del posgrado por su enseñanza

- A mis compañeros y amigos: Ricardo, Fernando, Miguel y Adrian por compartir momentos importantes

- A la Coordinación de Estudios de Posgrado (CEP) y al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado

Contenido

Índice de figuras

Resumen

En el presente trabajo se describe la integración del proceso de reconocimiento de objetos y el manejo de un brazo robótico mediante un procesador Sitara. Este dispositivo está basado en una arquitectura ARM Cortex-A8 la cual básicamente contiene las funcionalidades de un microcontrolador de alto rendimiento y una microcomputadora que ejecuta un sistema operativo.

En un espacio de trabajo definido, se presentan cuatro objetos rígidos de manufactura, cada uno con diferente forma. Estos son capturados por una cámara de video y procesados para su reconocimiento. Posteriormente por medio del brazo robótico son clasificados de acuerdo a su forma en diferentes contenedores.

El reconocimiento de objetos se realiza implementando el algoritmo para la obtención de la función frontera BOF (Boundary Object Function), la cual se usa para determinar la silueta y coordenadas de un objeto. Al obtener la BOF se conforma un vector descriptor el cual representa a cada uno de los objetos, reduciendo así la cantidad de información de la imagen.

Haciendo uso de bibliotecas de código abierto, dicho vector es alimentado a una Máquina de Vectores Soporte (SVM por sus siglas en inglés) con propósitos de entrenamiento y aprendizaje del sistema para que este clasifique los objetos.

Con la integración de los procesos de reconocimiento de objetos y del manejo del brazo robótico por medio del mismo dispositivo electrónico se obtiene un sistema económico en recursos de hardware y software, por lo que consigue un circuito compacto. La implementación de la BOF en el procesador, permite caracterizar prácticamente cualquier tipo de objeto.

Abstract

In this thesis the integration of object recognition process and the manipulation of a robotic arm by Sitara processor are described. This processor is based on an ARM Cortex-A8 architecture which basically contains the features of a high-performance microcontroller and a microcomputer that running an operating system.

In a defined work space, four different manufacturing rigid objects are placed. These are captured by a video camera and these are processed for recognition. After through the robotic arm these are classified according to their shape in different containers.

Object recognition is performed using the algorithm for obtaining the function BOF (Boundary Object Function), which is used to determine the shape and coordinates of an object. When the function is obtained, a descriptor vector is formed which represent each one of the objects, therefore the information is reduced in a image.

Using open source libraries, this vector is fed to a Support Vector Machine (SVM) for training purposes and learning system for classify the objects.

By integrating the processes of object recognition and control of the robotic arm by the same electronic device it is obtained an economic system in hardware and software resources, while reducing space and design time. The implementation of the BOF in the processor, allows characterize any type of object

Capítulo 1

Introducción

Una de las líneas de trabajo en el Departamento de Ingeniería de Sistemas Computacionales y Automatización del IIMAS-UNAM es el reconocimiento de objetos por medio de Visión Artificial.

Un sistema de visión tiene por objetivo reconocer y localizar objetos mediante el procesamiento de imágenes, teniendo como meta extraer características para su descripción e interpretación. Estos sistemas son dotados de computadoras cada vez más rápidas y potentes con lo cual deben ser capaces de ver y percibir objetos, quizá lo más parecido como lo hace el ser humano.

La visión artificial busca emular el funcionamiento de la visión humana [1], esto se hace cada vez más necesario al utilizar sistemas robóticos complejos en donde la programación y la guía del movimiento son cada vez más difíciles, debido a la gran variedad de aplicaciones en que se utilizan [2]. Los algoritmos empleados en la visión por computadora se caracterizan por tener operaciones complejas y repetitivas que implican una variedad de interacciones de datos [3].

En el análisis de imágenes, algunas aplicaciones necesitan distinguir un objeto de su entorno detectando su frontera [4]. El proceso de segmentación permite encontrar objetos localizando sus contornos o regiones de interés [5,6].

Una técnica puesta en prueba en el DISCA del IIMAS-UNAM que es utilizada para el reconocimiento de piezas de manufactura, es la función frontera BOF (Boundary Object Function) la cual determina la silueta de un objeto definiendo la distancia desde su centroide hasta los puntos de su perímetro [7], caracterizándolo a partir de las imágenes digitalizadas que se obtengan, para posteriormente tener una referencia que es comparada con otros objetos similares y hacer el reconocimiento de patrones respectivo [1].

En un sistema de visión se producen modelos (basados en datos de entrenamiento) los cuales predicen valores, lo que permite al sistema incrementar su conocimiento, obteniendo así un proceso eficiente en el reconocimiento de imágenes [8].Una técnica de aprendizaje son las Máquinas de Vectores Soporte las cuales se emplean para el entrenamiento y clasificación de datos [9].

Un robot auto-adaptable que usa una técnica de entrenamiento, es capaz de aprender habilidades manuales en línea y así disminuir la incertidumbre en el trabajo, la cual podría causar que una operación sea fallida [10].

En los procesos automatizados hay un interés por el desarrollo de celdas de manufactura aplicando el reconocimiento de objetos [11]. Actualmente, en los desarrollos a base de hardware se buscan implementar computadoras de dimensiones reducidas, de bajo costo y con un diseño específico. Un sistema electrónico que cumple con estas condiciones es la tarjeta BeagleBone Black, la cual es una computadora con dimensiones reducidas que contiene integrado un procesador Sitara ARM Cortex A8 [12, 13].

La tarjeta BeagleBone cuenta con un procesador Sitara el cual representa una solución para el procesamiento de imágenes ya que ejecuta instrucciones a 1Ghz de velocidad. También tiene una memoria RAM de 512MB, lo cual la convierte en una tarjeta embebida ideal para aplicaciones multimedia. Además presenta una gran cantidad de periféricos de propósito general para su comunicación con el exterior [14].

1.1 Planteamiento del problema

Un sistema flexible de manufactura consiste básicamente en una estación de trabajo, manejo y transporte automatizado de piezas y un sistema de control normalmente implementado por medio de un procesador. En estos sistemas automatizados se busca siempre la mayor eficiencia, reducción de costos y recursos materiales.

Un problema que se encuentra en las celdas de manufactura, es el uso de computadoras con software caro para el procesamiento de señales y el entrenamiento del sistema. Además, para que éstas puedan comunicarse con el exterior y tener el control sobre una maquinaria específica, necesitan de una interface electrónica la cual puede estar compuesta principalmente por un sistema embebido como lo es un microcontrolador o un FPGA, lo que implica obtener un sistema costoso.

1.2 Justificación

En el DISCA del IIMAS-UNAM existe una línea de investigación en robótica en la que se busca desarrollar una celda de manufactura flexible experimental con un modo sensorial de visión.

Actualmente, el mayor avance de la investigación se ha dado en la rama de la visión artificial, mediante el uso de una computadora de escritorio y con un software especializado (MATLAB), se realiza la adquisición de imágenes y el reconocimiento de la posición y forma de objetos. Además, por medio de las paqueterías de dicho software, ha sido implementada una red neuronal para el aprendizaje del sistema. En cuanto al control del brazo robótico, se han utilizado dispositivos embebidos (un microcontrolador o un FPGA).

De acuerdo a las necesidades del Departamento, se pretende tener un sistema electrónico compacto para reducir costos y espacio, por lo que se presenta la opción de integrar los procesos ya puestos en prueba, mediante un solo dispositivo electrónico.

Para este trabajo nos centramos en este tipo de dispositivos porque se encuentran entre los punteros del mercado. Además de sus altas prestaciones y capacidades avanzadas en soportar gráficos y de incluir opciones flexibles de administración de energía. Como resultado de ello, surgieron una serie de propuestas que terminó desembocando en el proyecto que actualmente se presenta.

1.3 Propuesta de solución

Una propuesta para el control de una celda de manufactura es integrar el proceso de adquisición, procesamiento y clasificación de imágenes y la manipulación de un brazo robótico por medio de una sola tarjeta electrónica. Las ventajas más importantes que se obtienen son: un sistema compacto, reducción de costos y una manera fácil de modificar el código para el reconocimiento de objetos y al mismo tiempo las señales de control para el sistema mecánico ya que se realiza en la misma plataforma.

La solución que se presenta para no ocupar software que requiere de una licencia para su empleo, es la implementación de bibliotecas de código abierto en la tarjeta de desarrollo, lo que conlleva a disminuir costos y tener poco uso de memoria de almacenamiento

1.4 Objetivos

El objetivo principal de este trabajo es la integración del proceso de reconocimiento de imágenes y la manipulación del brazo robótico usando un procesador Sitara.

Otro objetivo es la implementación del algoritmo para el cálculo de la función frontera denominada BOF (Boundary Object Function) en el procesador para el reconocimiento de objetos.

El tercer objetivo es implementar un clasificador mediante una Máquina de Vectores Soporte para el entrenamiento, aprendizaje y reconocimiento de formas de los objetos.

1.5 Metodología

Como primer paso se realiza una revisión bibliográfica mediante la lectura de artículos científicos y tesis de grado de maestría que describen las etapas del procesamiento de señales y la implementación de la función frontera de objetos (BOF). Posteriormente se hace una exploración referente a los procesadores que existen en el mercado para elegir el más conveniente, tomando en cuenta su capacidad de procesamiento y almacenamiento de datos.

Al obtener los elementos con los que se va a trabajar, se inicia por delimitar una región de interés y se define una posición estática de la cámara de video, la cual solo podrá adquirir imágenes dentro de dicha región.

En la tarjeta BeagleBone Black se realiza la adquisición de una imagen con la ayuda de las bibliotecas de código abierto OpenCV. Posteriormente, se hace un procesamiento en el cual se convierte a la imagen en escala de grises y enseguida se transforma en un formato de ceros y unos. En donde los unos representan al objeto y los ceros el fondo.

Después, se desarrolla el algoritmo para la obtención de la función frontera de objetos (BOF) la cual se representa por medio de un descriptor. Con el uso de las bibliotecas LIBSVM se ocupa un algoritmo para el entrenamiento de dicho descriptor utilizando una Máquina de Vectores Soporte.

En seguida, se implementa un código para la manipulación y guía del brazo robótico. Al tener los procesos de reconocimiento de objetos, entrenamiento del sistema y control del brazo; se integra un solo código dentro la tarjeta BeagleBone Black.

Finalmente se realizan pruebas del funcionamiento del sistema y se recopilan los resultados de las predicciones realizadas. Estos datos son documentados en la información final de este trabajo.

Capítulo 2

Fundamentos Básicos

2.1 Visión Artificial

La ciencia de la informática avanza hoy en día hacia la creación de computadoras cada vez más rápidas, más expertas y más autónomas.

Proporcionar capacidades sensoriales a las computadoras es una tarea sumamente complicada. A pesar de los inconvenientes, existe un interés especial por dotar a los ordenadores de uno de los cinco sentidos del hombre, el cual se refiere a la habilidad de ver [1].

La visión artificial describe la estructura y propiedades de un mundo tridimensional a partir de una imagen bidimensional de ese espacio. Se puede decir que la Visión Artificial es un campo de la Inteligencia Artificial que mediante la utilización de técnicas adecuadas, permite la obtención, procesamiento y análisis de cualquier tipo de información especial obtenida a través de imágenes digitales [3].

La visión tanto para un ser humano como para una computadora, consta principalmente de dos fases: captar una imagen e interpretarla. Por lo que la Visión Artificial lo que hace es interpretar las imágenes, distinguir los objetos de una escena, extraer información de ellos e interpretar esa información en un código digital [15].

2.1.1 Etapas de la Visión Artificial

La primera etapa en el proceso de la Visión Artificial es adquirir una imagen digital, para ello se necesita de una cámara de video y una computadora que tenga la capacidad de digitalizar la señal producida [16,17].

Una vez que la imagen digitalizada ha sido adquirida, el siguiente paso consiste en el preprocesamiento con la finalidad de mejorar la imagen para que el objetivo final tenga mayores posibilidades de éxito.

Enseguida, la imagen se somete a una segmentación; la cual tiene la finalidad de dividirla en objetos específicos, los cuales pueden ser caracterizados y clasificados [18].

La salida del proceso de segmentación es un código que contiene información de la frontera o de las coordenadas de un elemento especial. La representación de la frontera en un objeto es apropiada cuando el objetivo se centra en las características de la forma externa, como pueden ser las esquinas o contornos [19].

La última etapa es el reconocimiento y la interpretación del objeto. El reconocimiento es un proceso en el cual se asigna un descriptor al elemento para poder caracterizarlo. La interpretación consiste en diferenciar un conjunto de objetos reconocidos.

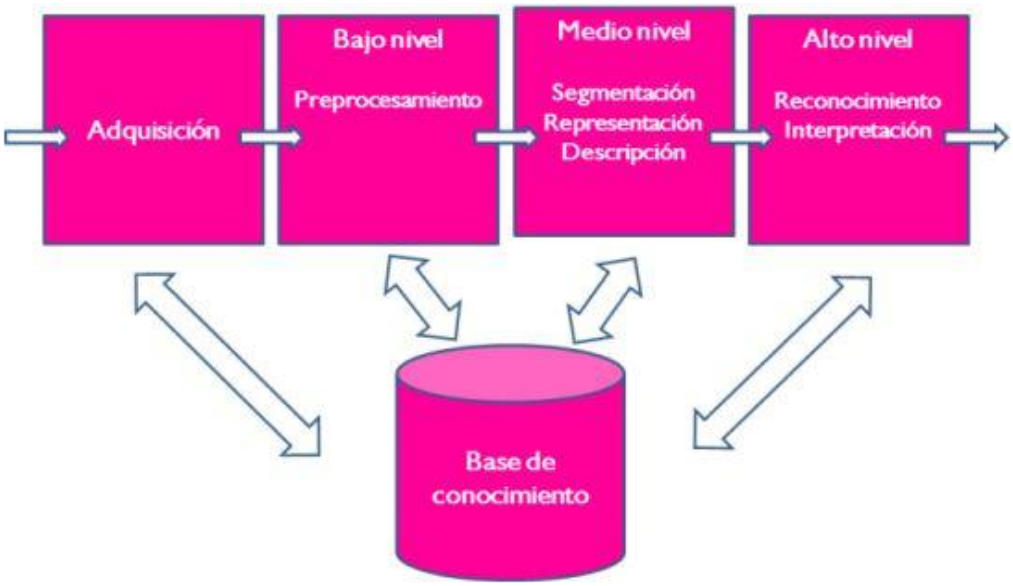


Figura 2.1 Etapas de la Visión Artificial

2.1.2 Representación de una imagen

Una imagen es una representación de una realidad, la cual proporciona información sobre colores, brillo, formas, etcétera. Estas son estáticas o en movimiento.

La digitalización se obtiene a través de una cámara fotográfica o de video, un escáner o directamente desde una computadora por medio de una cámara y utilizando algún programa para el tratamiento de imágenes. La información digital que generan los dispositivos electrónicos es almacenada en una memoria.

Una imagen digital es la representación bidimensional de pixeles en la que cada pixel adquiere valores de color codificados.

Las imágenes acromáticas son una composición en escala de grises, en la que a cada punto de una imagen se le asocia información relativa al brillo y se representa como una superficie, en la cual la altura de cada punto indica su nivel de brillo [20].

2.1.3 Extracción de características en una imagen

La extracción de características consiste en reducir la cantidad de información que representa a un objeto, obteniendo así un vector que interpreta de la mejor manera al objeto real.

Para que un objeto sea interpretado por un procesador es necesario tener un modelo del mismo, por lo que este se representa por medio de un descriptor. Los elementos del vector deben ser independientes de su tamaño, localización u orientación y deben ser suficientes para discriminarse entre sí. Su interés consiste en la evaluación de alguna característica [21].

Por lo regular se busca un conjunto mínimo de características que permitan determinar de maneara particular a que clase pertenecen los objetos con los que se está trabajando. La elección errónea de las características hace que el sistema sea costoso y lento o bien que sea imposible construir un clasificador para resolver un problema mediante un descriptor.

2.1.4 Procesamiento de una imagen

El procesamiento digital de imágenes cobra mayor importancia debido a sus múltiples aplicaciones en los diversos campos como: la medicina, radiología, en el campo de la defensa militar, en la industria para el control de calidad y otras aplicaciones que requieren de un supervisado [22].

Una imagen se define como:

$$I = f(x, y)$$

Para procesar una imagen en una computadora, esta se debe discretizar espacialmente. La digitalización de la coordenada espacial (x, y) se llama muestreo de la imagen.

2.1.5 Histograma de una imagen

El histograma es una curva donde se representa la frecuencia con que aparece cada nivel de gris. Generalmente se suele representar en el eje x el nivel de gris (por ejemplo de 0 a 255) y en el eje $y$la cantidad de pixeles representados por cada uno de los niveles [23].

Una imagen digital monocromática es una matriz en donde un pixel en la fila i y columna j tienen niveles de grises $f(i, j)$ que toma valores entre 0 (negro) y blanco (1).

Los niveles de grises se representan en un histograma, el cual permite determinar si la digitalización de la imagen se realizó correctamente. Existen técnicas de procesamiento que se basan en la transformación del histograma, o lo que es lo mismo, de los niveles de grises. La mejora de una imagen depende en ocasiones de la transformación del nivel de brillo de cada pixel.

La siguiente figura muestra una imagen de 8x8 pixeles:

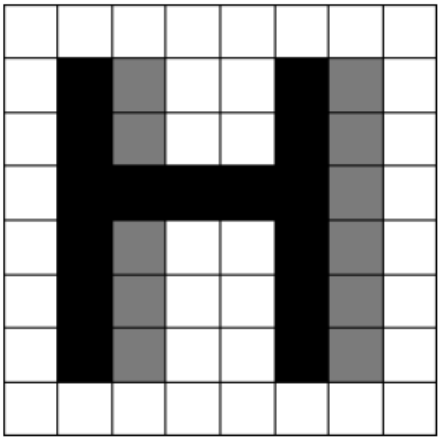


Figura 2.2 Imagen de 8x8 pixeles

Para la imagen anterior, se utilizan 2 bits para codificar el brillo de cada pixel. Los niveles de gris se enumeran del 0 al 3, correspondiendo un nivel de brillo mayor a los números más altos. Por lo que el histograma de la imagen anterior se representa de la siguiente manera:

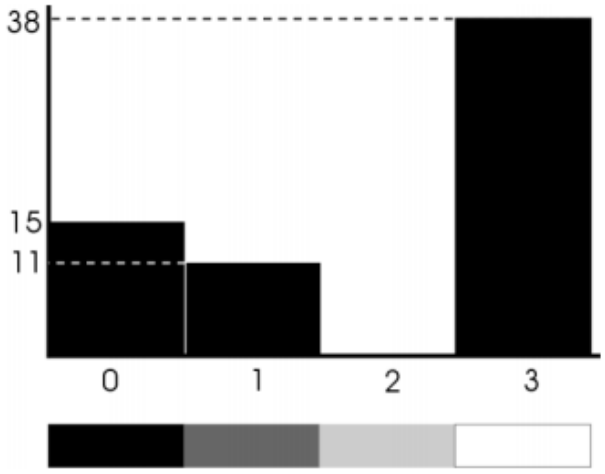


Figura 2.3 Histograma de la figura 2.2

En la imagen de la figura 2.3, se observa que cada barra representa el número de pixeles de la imagen equivalente a un nivel. Por tanto, la imagen tiene 15 pixeles

con nivel 0, 11 con nivel 1 y 38 pixeles con nivel 3. No se tienen pixeles con nivel 2.

2.1.6 Binarización de una imagen

La binarización consiste en comparar los niveles de grises presentes en una imagen con un valor (umbral). Por ejemplo, si el nivel de gris de la imagen es menor que el valor predeterminado, al pixel de la imagen binarizada se le asigna el valor de 0 (negro), y si es mayor, se le asigna un 1 (blanco) [24]. Por lo que se obtiene una imagen en blanco y negro, o dicho de otra manera, una imagen binarizada. En la siguiente figura se muestra un ejemplo:

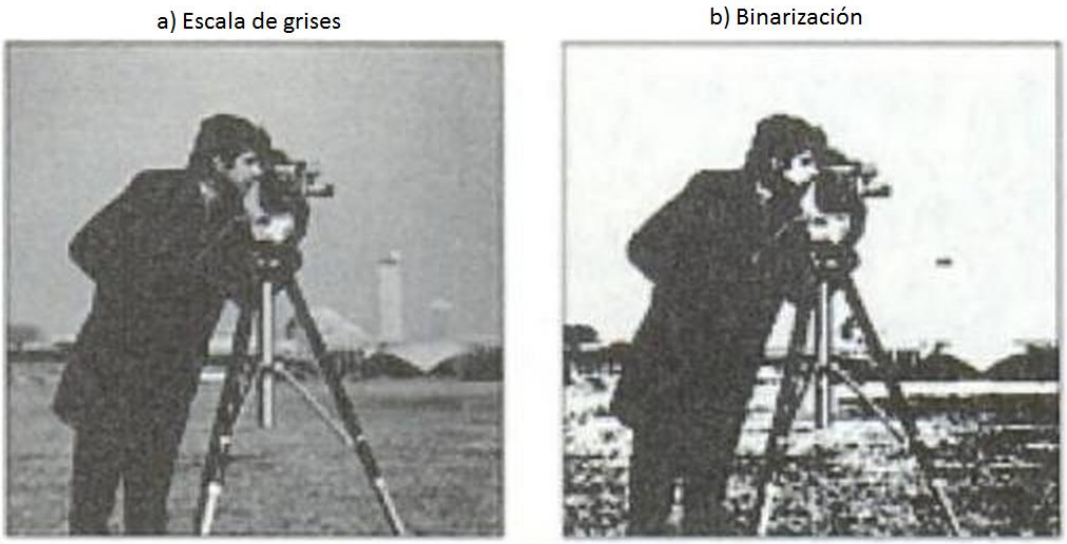


Figura 2.4. a) Imagen en escala de grises b) Imagen binarizada

Generalmente se utiliza un umbral de 128 si se trabaja con 256 niveles de gris, pero en algunas aplicaciones se requiere de tomar otro valor como parámetro.

2.1.7 Características del objeto a medir

Un objeto al ser identificado se caracteriza de acuerdo a su forma representándola por medio de un vector. Las características métricas se basan en mediciones de distancia entre dos puntos V_1 y V_2 [25]. Para calcularla utilizamos la distancia Euclidiana, la cual es deducida a partir del Teorema de Pitágoras:

$$d(V_1, V_2) = \sqrt{(V_{2x} - V_{1x})^2 + (V_{2y} - V_{1y})^2}$$

En donde, las coordenadas del punto V_1 son $(V_{1x} - V_{1y})$ y las coordenadas del punto V_2 son $(V_{2x} - V_{2y})$.

Al mismo tiempo, al obtener el contorno de una imagen conocemos su perímetro el cual está definido por:

$$P = \sum_{(m)} Pixeles \sum_{(n)} Pixeles\,(m,n) \in Contorno$$

De igual manera, se obtiene el área, la cual se calcula sumando la cantidad de pixeles que pertenecen a la forma:

$$A = \sum_{(m)} Pixeles \sum_{(n)} Pixeles\,(m,n) \in Forma$$

Una característica fundamental en la detección de objetos, es la obtención del centroide, el cual se define como el centro geométrico de un objeto.

$$X_c = \frac{\sum_{(x,y)} j}{A}, \qquad Y = \frac{\sum_{(x,y)} i}{A}$$

En donde A es el área expresada en pixeles.

2.3 Robots manipuladores

El campo de la robótica es una disciplina científica que incluye la investigación y desarrollo de una clase particular de sistemas mecánicos llamados robots manipuladores, diseñados para ejecutar diversas aplicaciones científicas, industriales, comerciales y domésticas [26].

Actualmente, una gran parte de los robots industriales son esencialmente brazos articulados. Según la definición del Instituto de Robótica de América, un robot industrial es un manipulador programable multifuncional diseñado para mover piezas, herramientas o dispositivos, mediante movimientos variados programados para la ejecución de diversas tareas.

Los robots manipuladores han tenido una excelente aceptación dentro de la industria, por lo que se han convertido en elementos fundamentales de los procesos de automatización, logrando así, una considerable reducción de costos y un incremento en la productividad. Dentro de las empresas más importantes que diseñan y construyen robots industriales se encuentran: FANUC Robotics, ABB Group, KUKA Robotics, MOTOMAN y EPSON, las cuales cuentan con una amplia variedad para diferentes aplicaciones.

2.3.1 Sistema mecánico

El sistema mecánico de un manipulador está compuesto por diversas articulaciones, en las cuales normalmente se distingue entre el brazo y el órgano terminal o efector, que puede ser intercambiable.

Las articulaciones están formadas normalmente por servomotores que permiten la conexión y movimiento entre dos eslabones consecutivos del robot. Dependiendo del tipo de desplazamiento que realicen las articulaciones, estas se definen como rotacionales o lineales. Las primeras se miden en radianes o grados, mientras que las segundas generalmente se miden en metros [27].

Al aumentar el número de articulaciones, se aporta mayor maniobra sobre el robot; pero se dificulta el control del mismo, por lo que se obtiene menor precisión debido a la acumulación de errores. La mayoría de robots industriales tienen menos de seis grados de libertad de rotación o traslación; este es el número mínimo que se requiere en general para tener una orientación en el espacio.

Físicamente la mayoría de los robots manipuladores tienen cierta similitud con la anotomía de las extremidades superiores del cuerpo humano. Algunas referencias de las articulaciones que componen a un manipulador son: cintura, hombro, brazo, codo y muñeca. Estos elementos los podemos reflejar en la siguiente imagen:

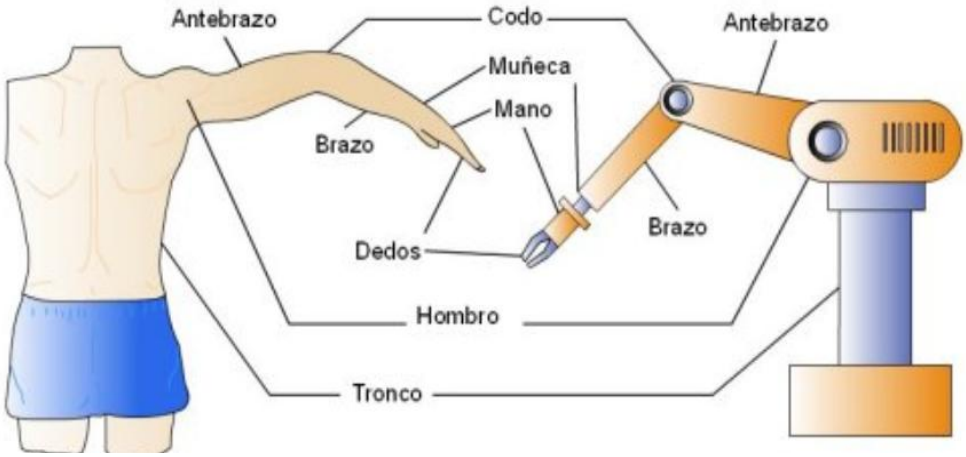


Figura 2.5 Esquema Antropomórfico

2.3.2 Características de los robots

Los robots son de diferentes diseños al igual que los programas que los controlan, todo depende de la función que se desea ejecutar. Lo que relaciona a un robot con respecto a otros, son las características que poseen, entre las se citan las más importantes:

- Grados de libertad: Es el número determinado por la cantidad de articulaciones del manipulador. Debido a que las articulaciones empleadas son de rotación y traslación, con un único grado de libertad cada una, el número de grados de libertad del robot suele coincidir con el número de articulaciones que lo componen.

- Capacidad de carga: Es el peso que transporta la pinza del manipulador. Normalmente este dato lo proporcionan los fabricantes, incluyendo el peso de la propia pinza. Es importante tomar en cuenta el peso de todas las articulaciones e instrumentos periféricos ya que el manipulador debe soportar la combinación de todas las cargas.

- Área de trabajo: Esta definida por las dimensiones de los elementos del robot y los grados de libertad. Es una característica fundamental debido a que define el recorrido del manipulador y la ubicación de una pieza en específico.

- Velocidad: Es la máxima velocidad alcanzable por las articulaciones. En ocasiones una velocidad de trabajo elevada, aumenta con éxito el rendimiento del manipulador.

- Precisión: La precisión es una medida que hace referencia a que tan cerca puede llegar el robot a un punto dado dentro de un espacio de trabajo. La precisión depende de la repetibilidad

- Repetibilidad: Es la capacidad para volver a la misma posición a la que fue dirigido en las mismas condiciones.

2.3.3 Programación de robots

Con el arribo de procesadores de bajo costo y poderosos, la tendencia en la industria ha estado orientada a la programación de robots mediante códigos escritos en software especializados. Antes del desarrollo de microcomputadoras en la industria, los controladores de robots eran similares a los secuenciadores simples utilizados en la automatización fija.

Un programa de control debe integrar el movimiento de objetos en un espacio tridimensional, por lo que es evidente que cualquier lenguaje de programación orientado al control de robots necesita describir y relacionar dichos movimientos. Por lo que una acción indispensable en la programación de robots es permitir la descripción de los movimientos que se espera que se ejecuten [28].

Los robots manipuladores se diferencían de la automatización fija al ser flexibles, lo cual podemos deducir que son sistemas programables. No solo podemos programar el movimiento de las articulaciones de los robots, sino que a través del uso de sensores, los manipuladores pueden ser adaptables a las necesidades de una tarea en específico.

En una celda de manufactura se incluyen uno o más robots y sistemas transportadores de piezas, por lo que la programación de un manipulador es solo una parte de un proceso de automatización. Las celdas de manufactura se interconectan en redes distribuidas en una fábrica, de manera que un procesador central tiene el control. Por tal motivo, la programación de robots se considera normalmente el problema más grande, debido a la variedad de máquinas interconectadas en una celda [26].

En una estación de trabajo la integración de un sistema de visión permite referenciar al manipulador en un espacio por medio de las coordenadas de un objeto de interés [27].

2.3.4 Brazo robótico Labot Pro 5

Es un brazo manipulador antropomórfico fabricado por la empresa mexicana Robótica CRYA el cual está diseñado para propósitos de enseñanza. Cuenta con seis grados de libertad las cuales dependen de sus articulaciones que son: base, hombro, codo, muñeca, mano y efector. Cada articulación es controlada por servomotores los cuales brindan fuerza y estabilidad. El Labot Pro 5 se muestra en la siguiente imagen [29]:

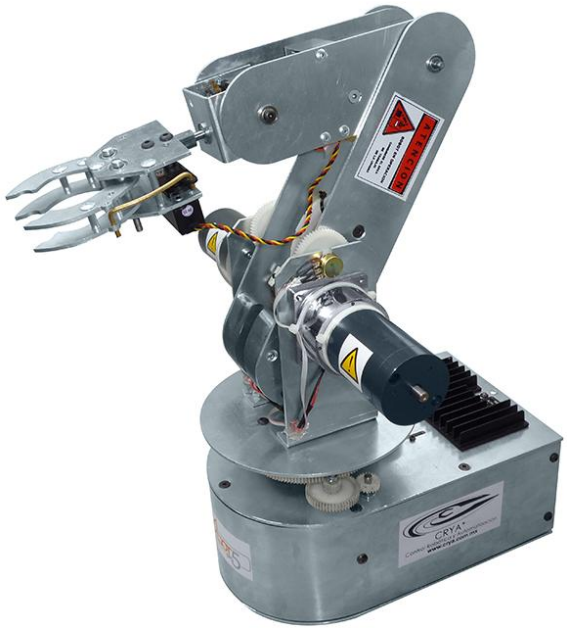
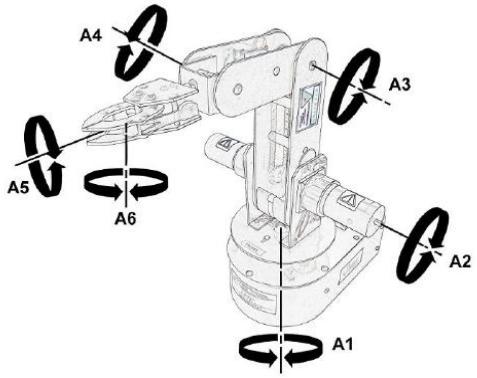


Figura 2.6 Manipulador robótico antropomórfico Labot Pro 5

Hay dos aspectos fundamentales que deben ser considerados cuando se opera un manipulador robótico: la dirección de rotación de los ejes y su espacio de trabajo. Conocer a fondo ambos, es fundamental para evitar alguna operación inadecuada.

Antes de calcular el espacio de trabajo, es conveniente tener definida la dirección de rotación y los intervalos de los ejes del manipulador, los cuales se muestran a continuación por medio de la siguiente figura:



Eje	Movimiento
A1	360°
A2	180°
A3	180°
A4	180°
A5	90°
A6	60°

Figura 2.7 Rotación y su intervalo de barrido del Labot Pro 5

Para saber en qué rango opera el manipulador, es indispensable conocer qué puntos se alcanza y en cuales se daña la estructura del brazo robótico. Para impedir algún movimiento erróneo se estableció un área de trabajo definida, en la cual el manipulador opera sin problemas. La siguiente imagen muestra el espacio en el que opera el brazo:

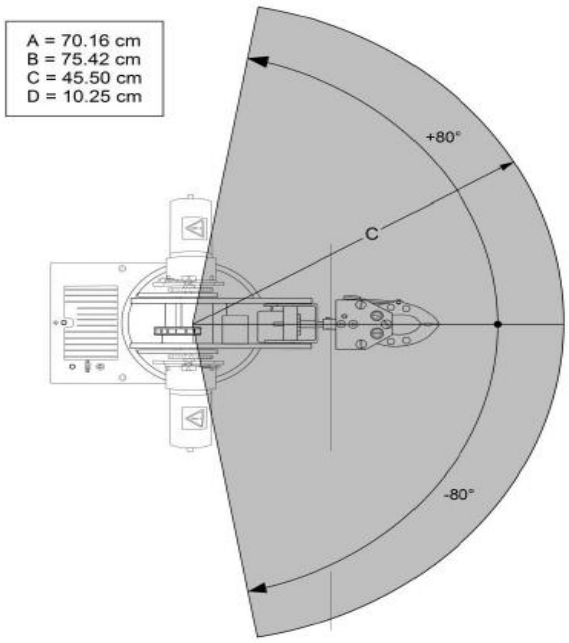


Figura 2.8 Espacio de trabajo del Labot Pro 5

2.4 Máquinas de Vectores Soporte para identificación de sistemas

Las Máquinas de Vectores Soporte o *Support Vector Machines (SVM* son técnicas que permiten extraer el conocimiento de bases de datos para utilizarlo en un sistema de entrenamiento.

El objetivo de las SVM es idear una manera eficiente de encontrar hiperplanos que separen un espacio de características [30]. Son un método de aprendizaje relativamente nuevo utilizado para la clasificación binaria.

Las SVM se han utilizado con gran éxito en campos de la informática como son la recuperación de información, la categorización de textos, el reconocimiento de escritura o la clasificación de imágenes.

La clasificación de patrones consta de etiquetar algún objeto dentro de una de las familias llamadas clases, a partir de un conjunto de datos asociados a cada elemento. La clasificación se realiza por medio de un software informático el cual

se desarrolla de tal manera que los objetos (x)sean identificados unos de otros conprecisión [31].

Existen dos enfoques principales de clasificación: la supervisada y no supervisada. En la primera se tiene un conjunto de datos de prueba, cada uno de estos consiste de medidas sobre un conjunto de variables y asociado a cada dato una etiqueta que define la clase del objeto. La segunda se refiere con frecuencia al agrupamiento de datos que se distinguen unos de otros a partir de sus características.

Las redes neuronales y las SVM son clasificadores de aprendizaje supervisado las cuales emplean un conjunto de entrada-salida, estos adquieren una función de decisión que asocia a un nuevo dato [32].

2.4.1 Clasificación SVM

El modelo más simple de las Máquinas de Vectores Soporte, fue el primero en ser introducido y es llamado “clasificador de margen máximo”. Este modelo trabaja únicamente con datos que son linealmente separables en un espacio, por tanto, no puede ser utilizado en muchas situaciones del mundo real [30].

Para clasificar con la técnica SVM, se comienza realizando una etapa de aprendizaje. Consiste en encontrar el hiperplano$h(x) = w^T xi + b = 0$ que mejor separe un conjunto de datos $X \in \mathcal{R}^d$ según la clase $Y \in \{-1,1\}$ a la que pertenecen. Dicho hiperplano corresponde con el que maximiza la distancia al punto más próximo de cada clase, por tanto, estará a la misma distancia de los ejemplos más cercanos entre ellos de cada categoría. Según la teoría de Vapnik, el separador lineal que maximiza el margen (2 veces la distancia al punto más próximo de cada clase) es el que nos da la mayor capacidad de generalización, es decir, la capacidad de distinguir características comunes de los datos de cada clase que permitan clasificar imágenes que no sean las del conjunto de entrenamiento.

Geométricamente, el hiperplano$h(x) = 0$ debe tener una dirección y una posición tales que se maximice su distancia al punto más próximo de cada clase. A los datos que se utilizan para hallar la frontera de decisión (hiperplano), se les conoce como vectores de entrenamiento o de aprendizaje [32].

Los parámetros de este modelo se obtienen a partir de los patrones de entrenamiento. La solución del problema de optimización debe satisfacer las

siguientes restricciones: los puntos más cercanos al hiperplano, denominados vectores de soporte, cumplen lo descrito en *Ec. (1)* y *Ec. (2)*, mientras que el resto de los patrones, cumplen lo reflejado en *Ec. (3)* y *Ec. (4)*, ya que no puede haber datos del conjunto de aprendizaje dentro del margen:

$$h(x) = 1\,, \quad si\ \ y_i = 1 \qquad (1)$$

$$h(x) = -1\,, \quad si\ \ y_i = -1 \qquad (2)$$

$$h(x) > 1\,, \quad si\ \ y_i = 1 \qquad (3)$$

$$h(x) < 1\,, \quad si\ \ y_i = -1 \qquad (4)$$

La siguiente figura muestra como una SVM separa un conjunto de datos:

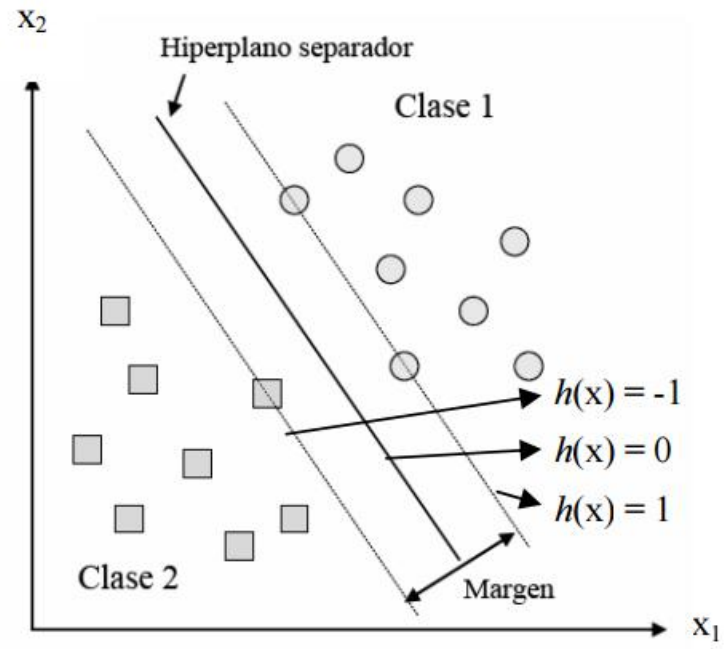


Figura 2.9Separación de dos tipos de clases mediante la técnica SVM

La distancia $dist(h,x)$ de un punto a un hiperplano es:

$$dist(h,x) = \frac{|h(x)|}{||\omega||}$$

Como se ha visto, los puntos más cercanos al hiperplano cumplen$|h(x)| = 1$ya que su distancia sería:

$$dist(h,x) = \frac{1}{||\omega||}$$

Para hallar la frontera de decisión que mejor separa un conjunto de puntos según las SVM (es decir, encontrar los valores de$w\ y\ b$), se resuelve un problema de optimización que consiste en maximizar la distancia $dist(h,x)$entre el hiperplano y el punto de entrenamiento más próximo:

$$\text{Max } \frac{1}{||\omega||}$$

$$\text{Sujeto a: } y_i(\omega^T x_i + b) \geq 1 \quad i = 1, \dots, n$$

Desde un punto de vista intuitivo, cuanto mayor sea el margen, mejor será la clasificación de las clases. Puede asumirse que todos los datos de entrenamiento que se encuentran por encima de la línea $+1$ satisfacen:

$$\frac{\omega \cdot x_{(+1)} + b}{\|\omega\|} \geq \tau$$

En donde τ es la distancia mínima del hiperplano de separación a los datos de entrenamiento más cercanos [SVM-AP].

Entonces, las muestras de entrenamiento por debajo de la línea -1 deben satisfacer:

$$\frac{-\omega \cdot x_{(-1)} + b}{\|\omega\|} \geq \tau$$

Compactando las dos expresiones anteriores en una sola, queda la desigualdad:

$$\frac{y_k(\omega \cdot x_{(k)} + b)}{\|\omega\|} \geq \tau, \quad k = 1,2,\dots,m$$

Donde $y_k \in \{-1,1\}$ y m es el numero de datos de entrenamiento.

El problema de encontrar el hiperplano óptimo se reduce a encontrar el ω que maximiza el margen τ. Suponiendo que tenemos un conjunto de entrenamiento linealmente separable con m muestras:

$$(x_1, y_1), \dots, (x_m, y_m). x \in \mathbb{R}^n, \qquad y_k \in \{+1|-1\}\, k = 1, \dots m$$

La frontera de separación de las dos clases va a ser una función lineal que va a permitir separar los datos de entrenamiento:

$$D_{(x)} = \omega \cdot x + b$$

Todas las muestras de conjunto de entrenamiento cumplen:

$$y_k(\omega \cdot x_k + b) \geq 1, \qquad k = 1, \dots, m$$

La siguiente figura muestra las distancias que deben existir entre la frontera y un conjunto de datos:

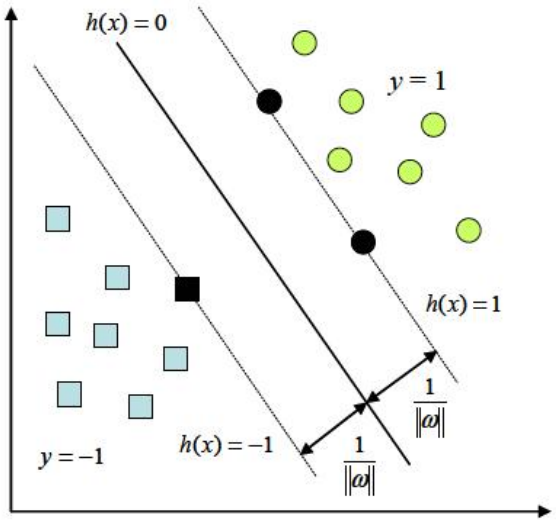


Figura 2.10Frontera de decisión

Lo planteado hasta el momento se cumple sólo si los patrones de entrenamiento son linealmente separables, ya que de lo contrario, el modelo de optimización tendrá un espacio de soluciones vacío. Sin embargo, en la práctica, es poco frecuente encontrar clases linealmente separables, y cuando no tengan esta característica, existe una alternativa para resolver el problema: utilizar una transformación de los datos a un nuevo espacio de entrada.

2.4.2 Función Kernel

La función Kernel es fundamental en la técnica SVM dado a que esta función logra separar los datos de manera que la maquina identifica adecuadamente los datos de validación [32].

Como ya se comentó anteriormente, ocurre que los puntos no sean linealmente separables, como se muestra en la siguiente figura:

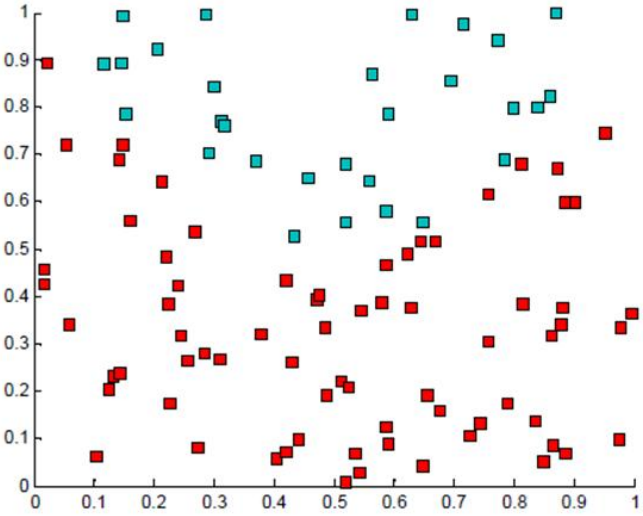


Figura 2.11Datos linealmente no separables

Por lo que existe la posibilidad de transformar los datos a un espacio de mayor dimensión (espacio de características) en el que los puntos si pueden ser separados por el híperplano. Para ello, se utiliza una función ϕ, tal que:

$$\phi\colon \mathcal{R}^D \to 3$$

$$x \to \phi(x)$$

En donde la frontera de decisión resultante en el espacio de entrada ya no es lineal. La siguiente imagen representa la transformación de datos:

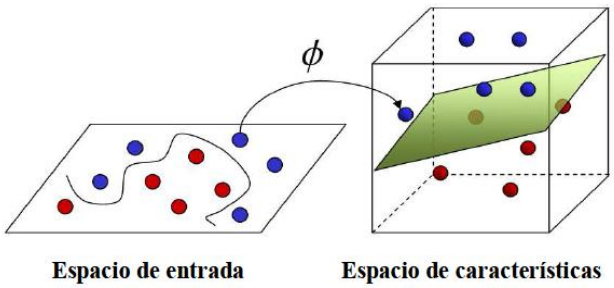


Figura 2.12Transformación de datos

Las funciones núcleo o kernels son usadas para realizar la transformación de datos y representan el producto vectorial de dos patrones en el espacio de características. El kernel asociado al espacio de características es

$$K(x, x') =< x, x' >^2$$

Para obtener la frontera de decisión para la SVM no lineal, se sustituye el producto vectorial del espacio de entrada x_i^T, x por el del espacio de características que se corresponde a la función núcleo:

$$h(x) = \sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + b$$

Las siguientes funciones son los kernels más usados:

Polinomial: $K(x, x') = (x \cdot x' + 1)^d$

Función de Base Radial (RBF): $K(x, x') = \exp(-\|x - x'\|^2 / 2\sigma)$

2.4.3 Kernel RBF

La Función de Base Radial (RBF por sus siglas en inglés) o Kernel RBF, es empleada como algoritmo de aprendizaje en las Máquinas de Vectores Soporte. En esta función los valores de salida dependen de la distancia de un argumento (vector de entrada) a un vector almacenado, propio de la función [31].

Una función radialmente simétrica utilizada en las SVM es la Gaussiana, que viene dada por la siguiente expresión:

$$K(x, x') = \exp(-\|u\|^2 / 2\sigma)$$

Donde u es la distancia Euclidiana entre valores de entrada x y el vector centro μ, es decir:

$$u = \|x - \mu\|$$

La salida de un Kernel RBF viene dada por:

$$y = \sum_{i=1}^{m} W_i \varphi \left(\| x - \mu_i \| \right)$$

Capítulo 3

Obtención de la Función Frontera de Objetos

3.1 Frontera de un objeto

La detección de la frontera de un objeto es parte de un proceso de segmentación o aislamiento, que consiste en la identificación de elementos dentro de una imagen [7].

La frontera es simplemente el contorno de un objeto que lo diferencía del fondo de la imagen. Esto se debe a los cambios en los niveles de grises que ocurren en una ubicación en específico. En cuanto más grande sea el cambio de nivel, más fácil es detectar la frontera, sin embargo, para un sistema computacional, cualquier cambio de nivel puede ser detectado rápidamente.

La representación de un contorno se hace mediante polilíneas las cuales son líneas continuas compuestas de varios segmentos que describen una frontera. Cada segmento X se especifica mediante un punto inicial y hasta un punto final. Por lo que la concatenación de dichos puntos describe un contorno:

$$X_1, X_2, X_3 X_4 \dots \dots X_n, X_1$$

En donde X_i corresponde a las coordenadas x, y de cada punto del contorno. En la siguiente figura se muestra una representación de un contorno mediante polilíneas.

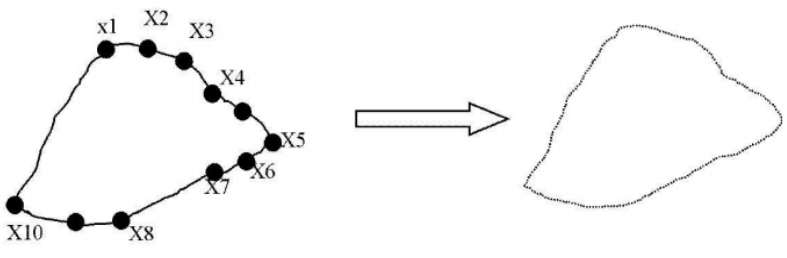


Figura 3.1 Representación de un contorno mediante polilíneas

3.1.1 Detección de contornos

Una manera de representar las fronteras de los objetos en una imagen digital consiste en utilizar el código de cadena de Freeman, el cual es una secuencia de segmentos conectados consecutivamente, de longitud y orientación especifica que relacionan pixeles adyacentes.

La suma de segmentos se compone en entornos de cuatro u ocho pixeles vecinos. Por tanto cuando se usa un entorno de *4* vecinos tendremos cuatro orientaciones delimitadas por los números *0, 1, 2 y 3*. Si se ocupa un entorno de *8* pixeles, entonces tendremos orientaciones representadas por los números *0, 1, 2, 3, 4, 5, 6 y 7*[22].

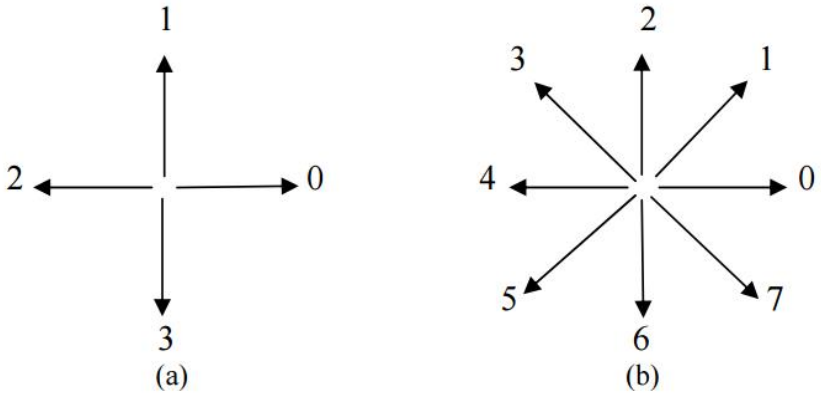


Figura 3.2 Entornos de pixeles. (a) Cuatro orientaciones (b) Ocho orientaciones

El código de cadena para representar un contorno, genera una secuencia de números que representan las orientaciones de los segmentos conectados consecutivamente, se parte de un punto del contorno y se sigue el sentido de las agujas del reloj. Normalmente se comienza por el pixel superior izquierdo de la frontera.

Por ejemplo, en la figura 3.3 (a) se representa un contorno con cuatro orientaciones, el cual está representado por el código 00333333232212111011 01, y la figura 3.3 (b) con ocho orientaciones tiene el código 076666553321212.

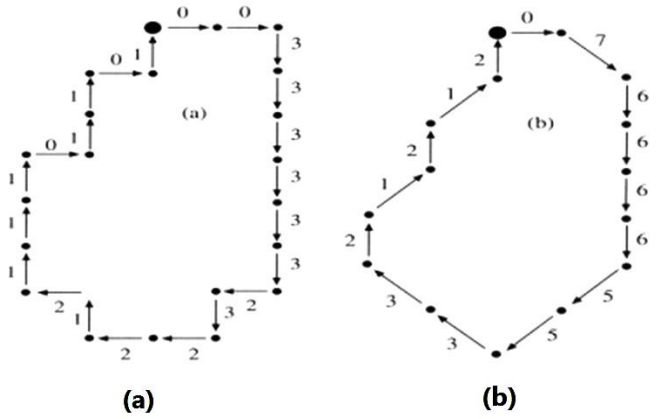


Figura 3.3(a) Código de cadena con 4 orientaciones (b) Código de cadena con 8 orientaciones

En cuanto a objetos que tienen regiones no conectadas o consecutivas, estos se representan por más de una cadena y se debe especificar si esta corresponde a la parte exterior del objeto o a una parte interior (agujero).

3.1.2 Ventajas y desventajas del código cadena

Los códigos cadena tienen diversas ventajas, entre las cuales encontramos las siguientes:

- Se obtiene el área y perímetro de un objeto de forma eficiente y rápida.
- Es una representación invariante frente a la rotación y traslación de un objeto en espacio. Esta propiedad facilita la comparación entre objetos.
- Es una representación compacta de un objeto en binario.
- Suministra una comprensión fácil de la descripción del contorno ya que cada cadena se codifica solo con una serie de datos binarios.

Las desventajas de utilizar el código cadena son:

- Cuando se tienen objetos de dimensiones grandes, el código cadena tiende a ser largo, por lo que cualquier perturbación pequeña a lo largo del contorno causa cambios en el código, lo que representaría una mala secuencia.

La imagen se representa por medio de un conjunto de mxn valores discretos que corresponden a los tonos de gris de cada uno de los pixeles obtenidos por el proceso de muestro, expresados por la siguiente matriz:

$$M_{m,n} = \begin{bmatrix} X_{1,1}X_{1,2} & \cdots X_{1,n} \\ X_{2,1}X_{2,2} & \cdots X_{2,n} \\ \vdots \quad \vdots & \ddots \quad \vdots \\ X_{m,1} \quad X_{m,2} & \dots \quad X_{m,n} \end{bmatrix}$$

Donde n es el número de pixeles en una columna y m el de una fila. Un pixel específico en su forma básica, es identificado por las coordenadas en el intervalomxn que representa la imagen.

Normalmente las imágenes contienen información inusual y en la mayoría de las aplicaciones solo se analizan zonas concretas que son conocidas como regiones de interés o *Region of Interest (ROI)*, las cuales definen como submatrices. [21].

3.2 Cálculo para obtener la BOF

Al obtener el contorno de un objeto, este se divide en un número determinado de puntos que se encontraran a la misma distancia uno de otro de manera consecutiva.

El centroide y las coordenadas de los puntos frontera, son obtenidos también como un vector de donde se extraen características de las formas de los objetos. Se obtiene una función de frontera del objeto (BOF) calculando las distancias de los bordes al centroide [8]. Por ejemplo, para el caso de un objeto en donde su contorno se divide en 16 elementos, se tiene:

$$[D_1, D_2, D_3 \dots \quad \dots D_n] \quad para\ n = 16$$

Con estos parámetros podemos formar la Función Frontera de Objetos BOF:

$$BOF_n = \sqrt{(P_X - C_X)^2 + (P_Y - C_Y)^2}$$

En donde:

BOF_n = Es uno de los valores de la función BOF

P_x , P_y= Son las coordenadas de un elemento del contorno

C_x , C_y = Son las coordenadas del centroide

Todas las magnitudes de estas distancias conforman la BOF la cual está representada por un vector.

Esta función es útil cuando se pretende diferenciar las formas de las figuras resultantes de objetos en tres dimensiones en imágenes de 2D. La BOF describe una pieza específica y varía de acuerdo a la forma de cada pieza [1].

3.2.1 Función Frontera de un cuadrado

Para obtener la BOF en un cuadrado, se define primero un sistema de coordenadas y su origen. Se obtiene su centroide y posteriormente el contorno para así dividirlo en secciones iguales, por ejemplo dieciséis.

El primer elemento del contorno será la coordenada que se encuentre en la parte superior izquierda del objeto, y en el sentido de las manecillas del reloj, los demás puntos serán distribuidos de manera equitativa dentro de las mismas coordenadas del contorno.

Conociendo todas las coordenadas del cuadrado, se obtienen las 16 distancias desde el centroide hasta cada uno de los puntos del objeto. Al final se conforma un vector de 16 elementos que nos permite caracterizar la figura geométrica.

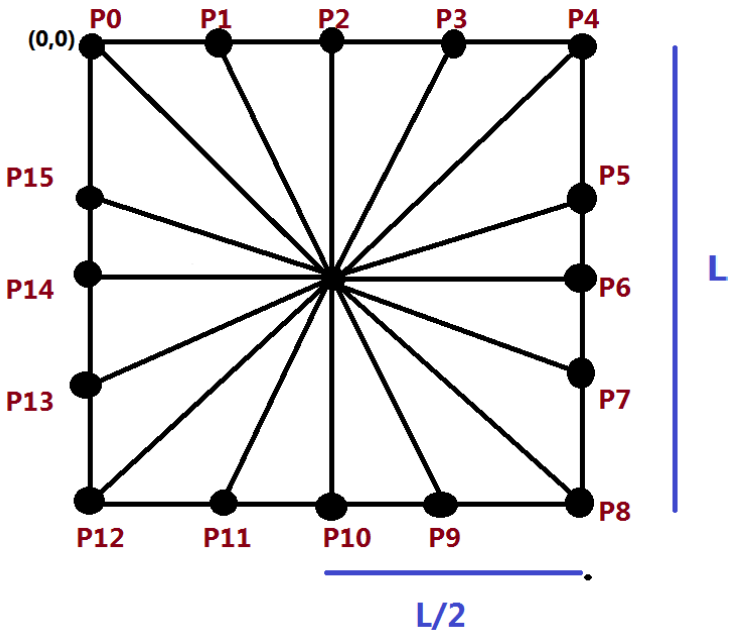


Figura 3.4 Descripción gráfica de la BOF de un cuadrado

Si el origen se encuentra en la parte superior izquierda con las coordenadas (0,0), para la ejemplificación de la BOF del cuadrado, tomaremos valores positivos hacia la derecha en el eje de las X y valores positivos hacia abajo en el eje de las Y.

De acuerdo a la figura, las coordenadas del centroide son:

$$C(C_X\ ;\ C_Y) = \left(\frac{L}{2}\ ;\ \frac{L}{2}\right)$$

Las coordenadas de los puntos del contorno son:

$$P_0 = (P_{0X}, P_{0Y}) = (0,0)$$

$$P_1 = (P_{1X}, P_{1Y}) = (L/4\ ,0)$$

$$P_2 = (P_{2X}, P_{2Y}) = (L/2,0)$$

$P_3 = (P_{3X}, P_{3Y}) = (3L/4,0)$

$P_4 = (P_{4X}, P_{4Y}) = (L, 0)$

$P_5 = (P_{5X}, P_{5Y}) = (L, L/4)$

$P_6 = (P_{6X}, P_{6Y}) = (L, L/2)$

$P_7 = (P_{7X}, P_{7Y}) = (L, 3l/4)$

$P_8 = (P_{8X}, P_{8Y}) = (L, L)$

$P_9 = (P_{9X}, P_{9Y}) = (3L/4, L)$

$P_{10} = (P_{10X}, P_{10Y}) = (L/2, L)$

$P_{11} = (P_{11X}, P_{11Y}) = (L/4, L)$

$P_{12} = (P_{12X}, P_{12Y}) = (0, L)$

$P_{13} = (P_{13X}, P_{13Y}) = (0,3L/4)$

$P_{14} = (P_{14X}, P_{14Y}) = (0, L/2)$

$P_{15} = (P_{15X}, P_{15Y}) = (0, L/4)$

Con cada una de las coordenadas obtenidas, se procede a calcular las distancias hacia el centroide, en donde las definimos como: D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, D_{10}, D_{11}, D_{12}, D_{13}, D_{14}, D_{15}

$D_0 = \sqrt{(C_x - P_{0x})^2 + (C_Y - P_{0Y})^2} = \sqrt{(L/2 - 0)^2 + (L/2 - 0)^2} = 0.707L$

$D_1 = \sqrt{(C_x - P_{1x})^2 + (C_Y - P_{1Y})^2} = \sqrt{(L/2 - L/4)^2 + (L/2 - 0)^2} = 0.559L$

$D_2 = \sqrt{(C_x - P_{2x})^2 + (C_Y - P_{2Y})^2} = \sqrt{(L/2 - L/2)^2 + (L/2 - 0)^2} = 0.5L$

$D_3 = \sqrt{(C_x - P_{3x})^2 + (C_Y - P_{3Y})^2} = \sqrt{(L/2 - 3L/4)^2 + (L/2 - 0)^2} = 0.559L$

$D_4 = \sqrt{(C_x - P_{4x})^2 + (C_Y - P_{4Y})^2} = \sqrt{(L/2 - L)^2 + (L/2 - 0)^2} = 0.707L$

Sí algunas distancias se repiten, se simplifica de la siguiente manera:

$$D_0 = D_4 = D_8 = D_{12} = 0.707L$$
$$D_1 = D_3 = D_5 = D_7 = D_9 = D_{11} = D_{13} = D_{15} = 0.559L$$
$$D_2 = D_6 = D_{10} = D_{14} = 0.5L$$

De tal manera que el vector que describe al cuadrado se conforma de la siguiente manera:

$$BOF = [D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, D_{10}, D_{11}, D_{12}, D_{13}, D_{14}, D_{15}]$$

La grafica que representa la Función Frontera de un cuadrado es la que se muestra a continuación:

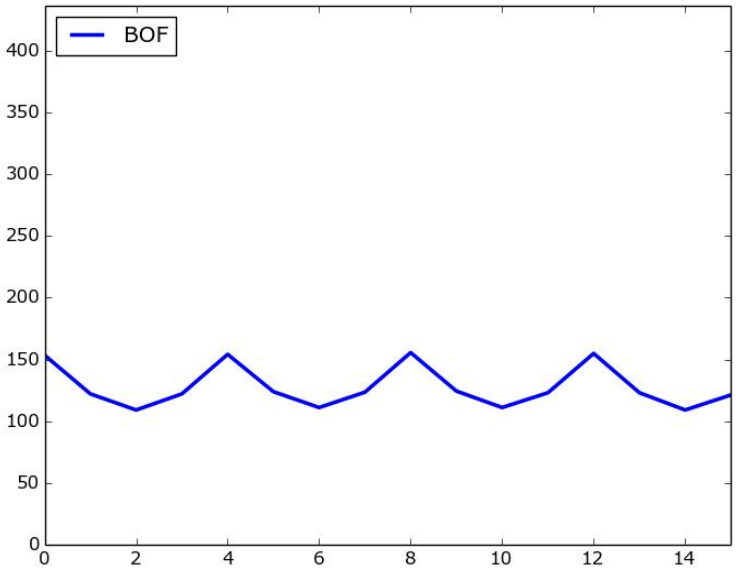


Figura 3.5 Función característica de un cuadrado de lado L=300

3.2.2 Función Frontera de un círculo

En el caso de la función frontera para un círculo, se siguen los mismos pasos que en la obtención de la BOF para un cuadrado; es decir, se obtiene el contorno del objeto y este se divide en 16 puntos distribuidos de manera equitativa a lo largo de su perímetro. Posteriormente obtenemos el centroide y las distancias hacia cada uno de los puntos para formar el vector descriptivo.

Para simplificar la representación de la BOF del círculo, suponemos que el centroide se encuentra en las coordenadas $(0,0)$. La siguiente figura muestra de forma gráfica las coordenadas de cada uno de los puntos y del centroide:

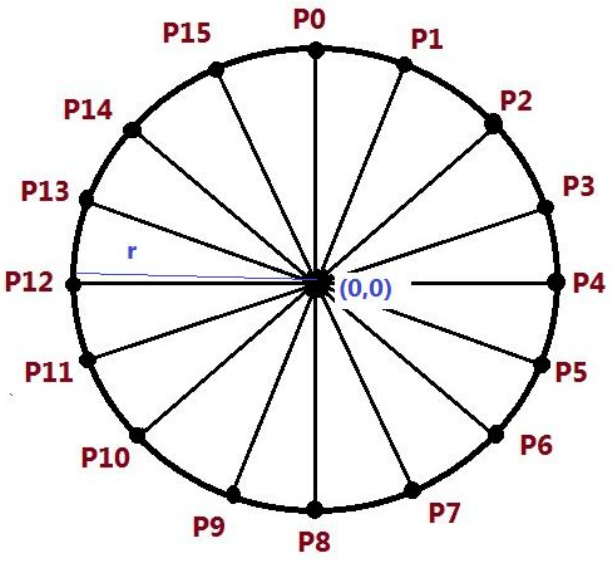


Figura 3.6 Descripcion gráfica de la BOF de un circulo

Por tanto, todas las distancias del centroide hasta cada uno de los puntos del contorno son equivalentes al Radio del círculo, en donde:

$$D_0 = D_1 = D_2 = D_3 = D_4 = D_5 = D_6 = D_7 = D_8 = D_9 = D_{10} = D_{11} = D_{12} = D_{13} = D_{14} = D_{15} = R$$

La función frontera del círculo se compone por cada una de las distancias obtenidas:

$$BOF = [D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, D_{10}, D_{11}, D_{12}, D_{13}, D_{14}, D_{15}]$$

Por lo que la grafica que representa al círculo es la siguiente:

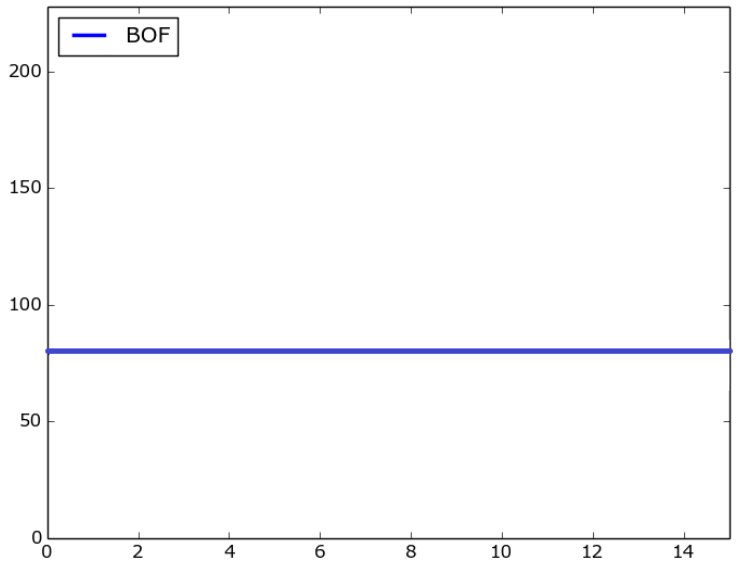


Figura 3.7 Función característica de un circulo de radio R=75

Capítulo 4

Procesador ARM

4.1 Introducción a los procesadores ARM

Los procesadores tipo ARM (Advanced RISC Machine) fueron diseñados para permitir implementaciones de tamaño reducido y alto rendimiento. Estos dispositivos han evolucionado rápidamente en los últimos años [33].

Estos procesadores están licenciados por la compañía Britania ARM Holdings, que realizó su primer prototipo en el año 1985, desde entonces, el diseño se ha actualizado de forma permanente de tal manera que en 2013 se convirtió en la arquitectura de procesadores más exitosa del mundo. El éxito de estos dispositivos ha sido considerable, pues en el año 2009 sumaban alrededor del 90% de todos los procesadores tipo RISC (Reduced Instruction Set Computer) de 32 bits.

Los microprocesadores ARM de 32 bits son los dispositivos electrónicos más utilizados en sistemas móviles y embebidos en el presente. Actualmente los encontramos en aparatos electrónicos como: tablets, consolas de videojuegos, calculadoras científicas o reproductores multimedia.

4.1.1 Arquitectura ARM

ARM es una arquitectura RISC de originalmente 32 bits, actualmente existen procesadores ARM de 64 bits. La simplicidad de esta arquitectura la hace ideal para aplicaciones de baja potencia.

Una arquitectura general ARM se despliega en la siguiente figura:

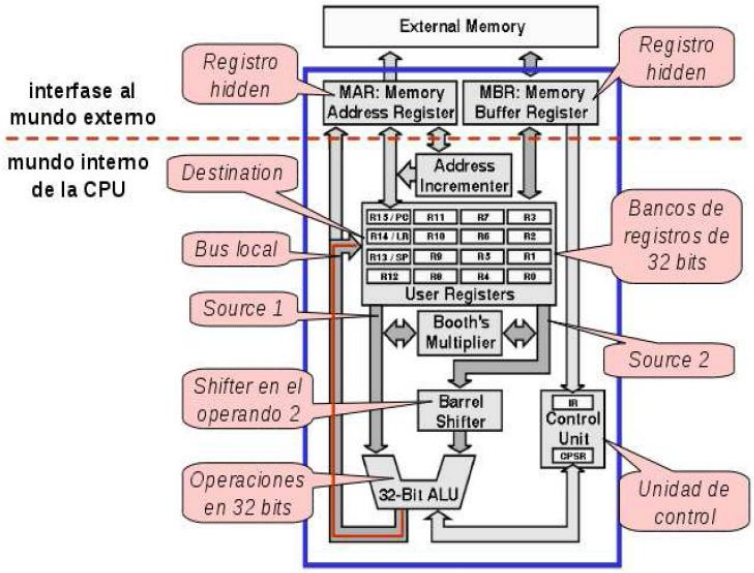


Figura 4.1 Arquitectura general ARM

En donde podemos encontrar elementos como:

- Un conjunto de registros R_0 *a* R_{15} que pueden ser apuntadores a memoria o acumuladores de 32 bits
- Una unidad ALU (Arithmetic Logic Unit) de 32 bits
- Bloque de corrimiento que opera de forma paralela a la ALU
- Un bloque multiplicador
- Una interface para la transferencia de datos de memoria de registros MAR (Memory Address Register) y MBR (Memory Buffer Register).
- Una unidad de control

Algunas de las características que ARM incorporó de la arquitectura RISC son:

- Arquitectura de carga (load-store): También llamada arquitectura almacenamiento en donde las instrucciones que acceden a la memoria están separadas de las instrucciones que procesan datos ya que en este último caso los datos necesariamente están en registros.

- Formato fijo de palabra de instrucciones: Se refiere a un campo de instrucciones de longitud fija para simplificar la decodificación de las instrucciones.

- Máquina de tres direcciones: Es un número f de bits que representa al código de una operación, n bits para especificar la dirección del primer operando y n bits para especificar la dirección del segundo operando y por ultimo n bits para especificar la dirección del resultado o el destino. Por ejemplo para sumar dos números en lenguaje ensamblador, el resultado es:

$$ADD\ d,\ S_1,\ S_2 \qquad ;d = S_1 + S_2$$

En la siguiente figura se muestran los registros de una máquina de tres niveles:

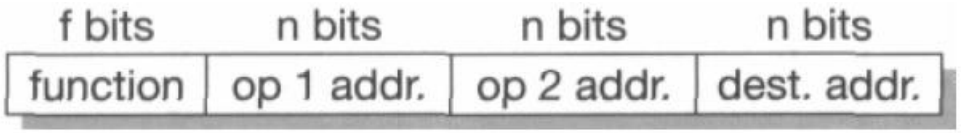


Figura 4.2 Instrucciones de 3 direcciones

En una máquina de 3 direcciones de ARM, cada dirección se especifica por registros que se han cargado previamente con el contenido de sus direcciones de memoria correspondientes.

A su vez ARM incorporó algunas características novedosas, como lo son:

- Conjunto de instrucciones ortogonal: En este tipo de repertorio no hay restricción en los registros usados ya que la mayoría son registros de propósito general.

- Control sobre la ALU: Se posee un control en cada instrucción de procesamiento de datos para maximizar su desempeño.

- Ejecución de instrucciones rápida: Todas las instrucciones se ejecutan en un ciclo de reloj o menos.

- Bajo consumo: Son arquitecturas que consumen baja potencia, en donde la tensión de alimentación llega a ser de 1.8 V y una frecuencia eficaz de trabajo durante el tiempo de ejecución de las instrucciones. Esto se logró gracias a la reducción de transistores y el tamaño del silicio ocupado.

- Eficiencia para operar con lenguaje “C”: Esta arquitectura fue concebida para trabajar en un lenguaje de alto nivel con una densidad de código superior a los microprocesadores de 8 bits.

Las mejoras que se hicieron sobre la arquitectura RISC permiten a los procesadores ARM adquirir un buen equilibrio entre el alto rendimiento, bajo consumo de energía y poco uso de silicio para su fabricación lo que los hace ser económicos y potentes [34].

4.1.2 Arquitectura pipeline

De acuerdo a la estructura de una arquitectura pipeline, ésta optimiza los recursos del hardware y el rendimiento del procesador. Consiste en comenzar a procesar una nueva instrucción antes de que haya concluido el procesamiento de la actual.

Una de las características del alto desempeño de los procesadores ARM es la arquitectura tipo *pipeline*. Por medio de la siguiente figura se representa la ejecución de instrucciones en este tipo de arquitectura:

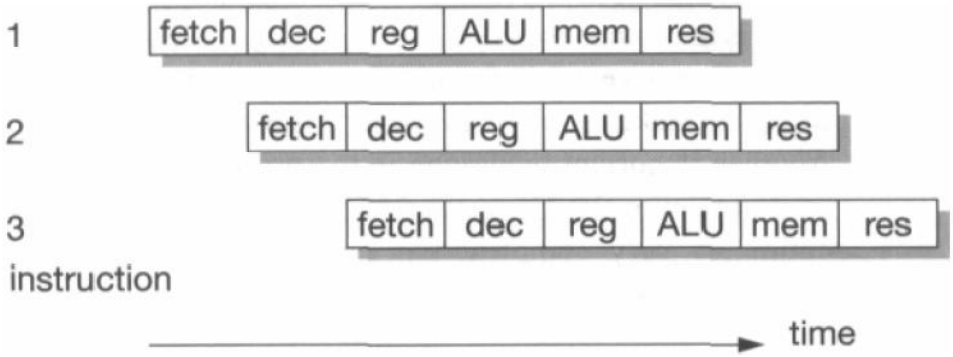


Figura 4.3 Ejecución de instrucciones en una arquitectura *pipeline*

El procesador realiza simultáneamente la ejecución de la primera instrucción y la búsqueda del código de la siguiente, de esta manera se ejecuta una instrucción en un solo ciclo de reloj. Por lo que el tiempo de ejecución deberá ser seis veces más veloz que el que corresponde a las instrucciones no superpuestas [35].

En la siguiente figura se tiene una arquitectura *pipeline* de tres etapas, en donde, como se observa, cada instrucción se ejecuta en cada una de ellas:

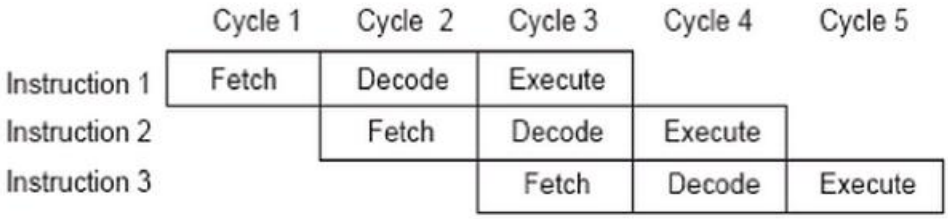


Figura 4.4 *Pipeline* de 3 etapas

La ejecución se interpreta de la siguiente manera:

- *Fetch:* se lee la instrucción de la memoria y se coloca en el *pipeline*
- *Decode:* decodificar: se decodifica la instrucción
- *Execute:* se ejecuta la instrucción

El *pipeline* es lineal, lo que significa que el procesador busca el código de la operación de la instrucción 3, mientras se decodifica el código de la operación leída anteriormente en la instrucción 2 y se ejecuta la instrucción 1 leída y decodificada previamente.

4.1.3 Arquitectura Thumb

La densidad de código es una medida de cuanta memoria necesita un sistema embebido para contener instrucciones, ya que frecuentemente en estos dispositivos hay limitaciones en el tamaño de la memoria.

Hay aplicaciones en donde es indispensable la densidad del código, por lo que ARM incorporó un mecanismo llamado arquitectura Thumb, la cual consiste en un repertorio de instrucciones en forma comprimida a 16 bits de un repertorio de 32 bits original [36].

El set de instrucciones Thumb usado para obtener alta densidad de código en los procesadores ARM utiliza una arquitectura de dos direcciones. Se puede llevar a cabo un nuevo ajuste en el número de bits que se requiere para almacenar una instrucción haciendo que el registro de destino coincida con alguno de los registros fuentes.

Al tener la mitad del ancho de palabra, se consigue disminuir la cantidad de código y mejorar la densidad. Realmente las instrucciones son menos poderosas que las

de 32 bits, de manera que se requiere más de una instrucción de 16 bits para equiparar a una de 32.

4.1.4 Versiones de la familia ARM

ARM ha conservado su arquitectura inicial, sin embargo ha introducido algunos cambios significativos [36]:

- Versión 1: VLSI Technology fabrico en silicio el ARM1 el 26 de abril de 1985 con 25,000 transistores, un bus de datos de 32 bits, 26 bits de dirección y 16 registros.

- Versión 2: También fue fabricado por VLSI Technology en 1986 y disponía de un bus de datos de 32 bits, un bus de dirección de 26 bits y 27 registros de 32 bits.

- Versión 3: Introdujo el direccionamiento de 32 bits y variantes como Thumb y un multiplicador.

- Versión 4: Corresponde a una arquitectura ARMv4T la cual introdujo el conjunto de instrucciones Thumb de 16 bits comprimidos en el año 1996. El ARM7 es el procesador embebido a 32 bits más utilizado de la historia.

- Versión 5: En esta versión ARM agrego instrucciones para el procesamiento digital de señales (DSP), y aplicaciones multimedia. También se agrega la letra “E” como ARMv5TE, el cual significa que soporta codificador Java, esta surgió en 1999.

- Versión 6: La versión v6 mejoro los accesos a memoria, acelero los procesos en aplicaciones inalámbricas y multimedia. La familia de procesadores ARM11 está basada en el ARMv6 el cual incluye un sistema de instrucciones tipo SIMD (Single InstructionMultiple Data) la cual mejora el desempeño hasta en dos o cuatro veces las aplicaciones multimedia.

- Versión 7: ARM diseño las versiones Cortex con los perfiles ARMv7-A, ARMv7-M y ARMv7-R, en donde:

    - v7A: aplicaciones NEON (acelera señales multimedia), para plataformas abiertas tal como sistemas operativos. Se encuentran en dispositivos como tablets, celulares, smart-TV, etc.
    - v7R: aplicaciones de tiempo real, se encuentran en aplicaciones como la robótica, impresoras, control electrónico de motores, etc.
    - v7M: aplicaciones para Microcontroladores de bajo consumo, se encuentran en lavadoras, microondas, mandos, nodos inalámbricos, etc.

- Versión 8: fue lanzada a finales de 2011, la cual representa todo un cambio en la arquitectura ARM en la cual se añaden 64 bits llamada “A Arch64”.

4.2 Procesador ARM Sitara

El procesador SITARA es fabricado por la empresa Texas Instruments, los cuales son microprocesadores de alto rendimiento basados en ARM9 y ARM Cortex A8 que soportan velocidades desde 200Mhz hasta 1 Ghz. La gama de estos dispositivos incluye combinaciones exclusivas de periféricos y aceleradores para reducir los costos de los sistemas y expandir las opciones de conectividad.

Con este tipo de dispositivos se pueden realizar tareas de alto desempeño como aplicaciones gráficas multimedia, procesamiento de señales, control de robots, etc. Básicamente tenemos las funcionalidades de una computadora en donde podemos ejecutar un sistema operativo y de un microcontrolador de 32 bits en un mismo dispositivo [38].

La arquitectura Cortex de ARM provienen de la versión 7 (v7) y se dividen en 3 ramas [34]:

A: Para aplicaciones abiertas

M: Integradas en Microcontroladores

R: Para aplicaciones embebidas en tiempo real

La siguiente imagen muestra la evolución de de las arquitecturas ARM Cortex:

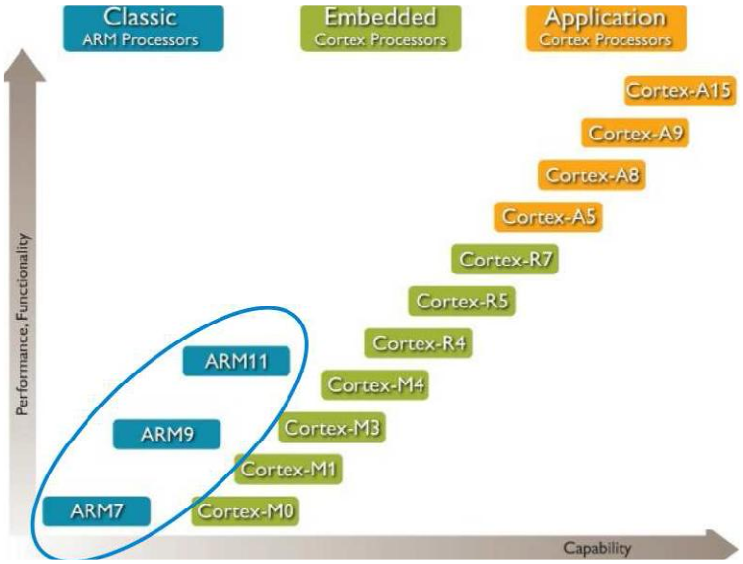


Figura 4.5 Familias de procesadores ARM Cortex

La arquitectura interna de un procesador Sitara la podemos representar por medio de la siguiente figura:

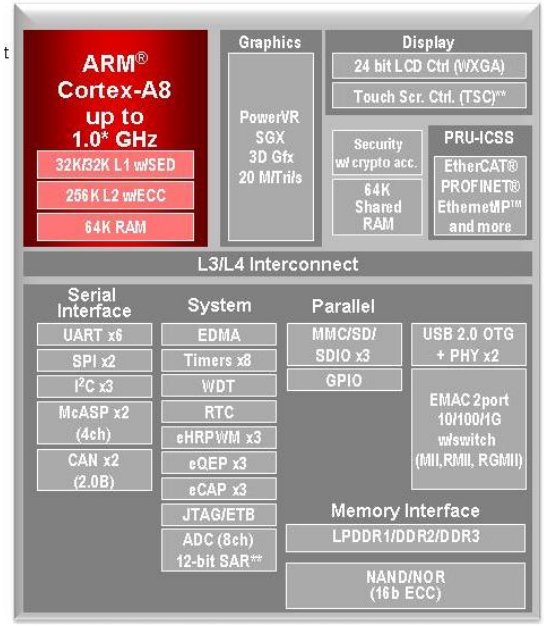


Figura 4.6 Procesador Sitara AM335X

4.3 Tarjeta de desarrollo BeagleBone Black

Es una tarjeta de desarrollo para sistemas embebidos, la cual incluye un procesador Sitara AM335Xy una memoria de almacenamiento de datos de 4GB en cual contiene instalado el sistema operativo Linux

Es una herramienta de código abierto, bajo costo y expandible con otras tarjetas de desarrollo. Su introducción al mercado fue en abril del 2013 por *BeagleBoard.org Foundation*, una comunidad de desarrolladores patrocinados por Texas Instruments. La siguiente imagen muestra la tarjeta [37]:

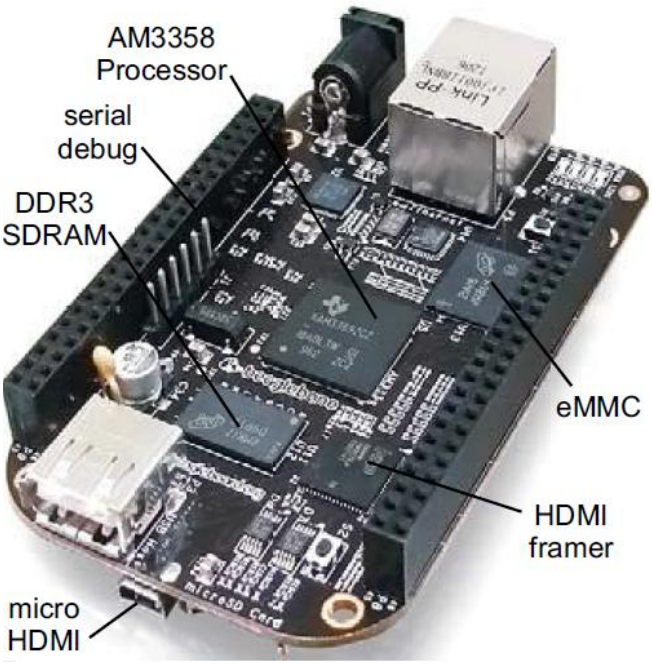


Figura 4.7 Tarjeta de Desarrollo de código abierto

Esta tarjeta contiene un Sitara AM3358 ARM el cual opera a una frecuencia de 1Ghz, una memoria RAM DDR3 de 512MB y el nuevo modelo (Rev C) incluye una memoria de almacenamiento de 4GB en donde tiene preinstalado el sistema operativo *Debian Linux*. Cuenta también con un puerto USB, un puerto Ethernet y un conector micro HDMI para comunicarse con una fuente de video digital.

El diagrama a bloques de la BBB se representa en la siguiente imagen:

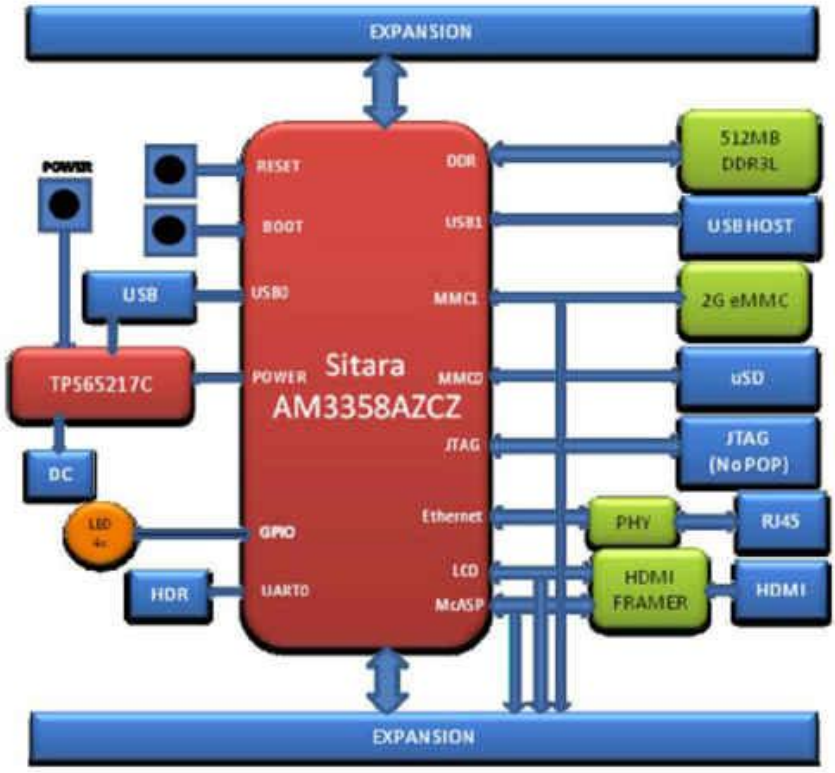


Figura 4.8 Diagrama a bloques de la BBB

La tarjeta BeagleBone Black cuenta con 4 puertos UART, 2 módulos SPI, 8 módulos PWM, 2 periféricos CAN, un convertidor ADC de 12 bits con 7 canales de entrada. Estos módulos se encuentran distribuidos en los 92 pines disponibles en dos tiras de pines “headers” P8 y P9.

Este dispositivo electrónico no intenta reemplazar a una computadora de escritorio o a una Laptop, si no que intenta ser usada exclusivamente como una computadora embebida para un proyecto específico y tener la flexibilidad para moverse en conjunto con dicho proyecto cuando el usuario desee cambiar su área de trabajo.

Normalmente en las aplicaciones de robótica en donde se realiza el procesamiento de imágenes, es indispensable ocupar una PC o Laptop para el procesamiento, así como desarrollar algoritmos que permitan reconocer objetos. Pero para el control de un robot, se necesita de un dispositivo embebido que guie sus movimientos, lo cual implica una comunicación entre la computadora y un microcontrolador o FPGA (por ejemplo). Esto conlleva a que el sistema sea

costoso debido a la cantidad de dispositivos empleados y a que este pierda tiempo entre la interacción de elementos electrónicos [38].

En la figura 4.9 se muestra un robot con un sistema *Kinect* para la detección de objetos, una computadora portátil para el procesamiento de datos y un sistema embebido para el control del mecanismo, lo que implica que se tenga un sistema costoso, pesado y voluminoso:

Figura 4.9Plataforma de un robot Parallax que utiliza una laptop y un sistema embebido

Capítulo 5

Desarrollo del sistema clasificador de objetos

5.1 Funcionamiento del sistema

El sistema se compone principalmente de una tarjeta de desarrollo BeagleBone Black integrada con un procesador ARM Sitara el cual tiene la función de adquirir y procesar imágenes y al mismo tiempo manipular el funcionamiento del brazo robótico.

La figura 5.1 muestra el diagrama esquemático del sistema:

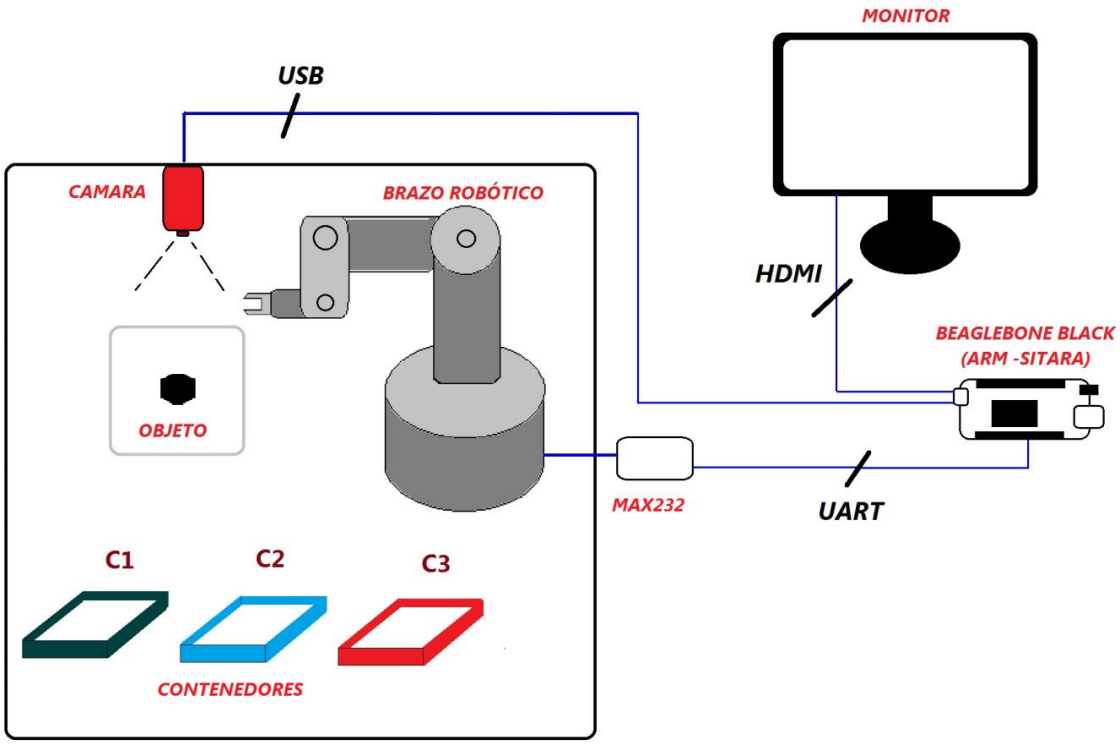


Figura 5.1. Diagrama esquemático del sistema clasificador de objetos

La tarjeta BBB (Rev C) tiene instalado el sistema operativo Linux con la versión de distribución Debian, en el cual han sido instaladas las paquearías de código abierto de OpenCV las cuales son usadas en aplicaciones de visión artificial.

Con estas bibliotecas y la conexión de una cámara de video hacia la tarjeta BeagleBone Black (por medio de un cable USB) se realiza la adquisición de una imagen la cual es almacenada en la memoria interna de dicha tarjeta.

El preprocesamiento de la imagen consta de convertirla en escala de grises para obtener un mapa de bits y después binarizarla y así reducir la cantidad de información almacenada en memoria y tener un procesamiento rápido y eficiente de un objeto en específico.

En el procesamiento de la imagen se obtiene el contorno, centroide y las coordenadas en pixeles de un objeto. Con esta información se desarrolla la Función Frontera y se caracteriza al objeto por medio de un vector, el cual tiene información de su área y perímetro. Cada pieza es representada por medio de un descriptor diferente y estos son clasificados mediante una Máquina de Vectores Soporte para poder diferenciar entre una y otra.

Gráficamente es representada la captura en imagen de cada elemento. En este proceso se visualiza cada etapa del procesamiento de la imagen para así monitorear cualquier error que se presente.

Con las coordenadas obtenidas del objeto dentro del área trabajo, se procede a controlar las articulaciones del brazo robótico para posicionarse cerca de una pieza, sujetarla y transportarla al contenedor de donde pertenece.

5.2 Procesamiento de la imagen en el procesador Sitara

La distribución precargada que tiene el procesador Sitara es *Debian*, la cual es ideal para operar en este dispositivo ya que es de fácil configuración, rápida, de poco uso de memoria y en la cual se utiliza un sin número de aplicaciones de software libre. La versión instalada consume aproximadamente 600MB de los 4GB disponibles en la memoria de almacenamiento de la tarjeta BBB por lo que se tiene suficiente espacio para ser aprovechado en otros programas o bibliotecas [39].

El lenguaje utilizado para definir el funcionamiento del procesador es “Python”. Se decidió utilizar este lenguaje de programación de alto nivel ya que es un software

libre, poderoso y de fácil interpretación. “Python” permite escribir programas legibles y más compactos si se compara con lenguajes como “C”, “C++” o “Java2 ya que no es necesario declarar variables ni argumentos, tampoco es necesario abrir y cerrar llaves ya que la agrupación de instrucciones se hace por sangría. Este lenguaje puede ser interpretado por GNU/LINUX [40].

Para la ayuda en el reconocimiento de objetos se utiliza OpenCV (Open Source Computer Vision), el cual es una biblioteca de código abierto empleada en aplicaciones de visión artificial. Inicialmente fue desarrollada por Intel e incluye cientos de algoritmos de visión por computadora y es multiplataforma lo cual posibilita su instalación y funcionamiento en Mac OSX, Windows y Linux [41].

En cuanto al entrenamiento y clasificación de los objetos caracterizados se utiliza una biblioteca interpretada en “Python” llamada “LIBSVM” la cual cuenta con algoritmos para el diseño y entrenamiento de clasificadores SVM[42].

5.2.1 Adquisición de la imagen

La primera etapa consiste en capturar la imagen de una pieza mediante una cámara de video (Logitech C170) conectada a través de un cable USB a la tarjeta.

La adquisición de la imagen se realiza con la ayuda de la biblioteca de OpenCV con la función *cap.read().* Inicialmente el sistema captura video (secuencia de imágenes) hasta que un objeto es puesto en el área de trabajo y detectado por medio de un sensor de presencia infrarrojo el cual manda una señal a la tarjeta BeagleBone Black y esta después de 500 ms captura la imagen del objeto y la almacena en la memoria Flash de 4GB.

La figura 5.2 representa la detección del objeto, la adquisición y almacenamiento de la imagen:

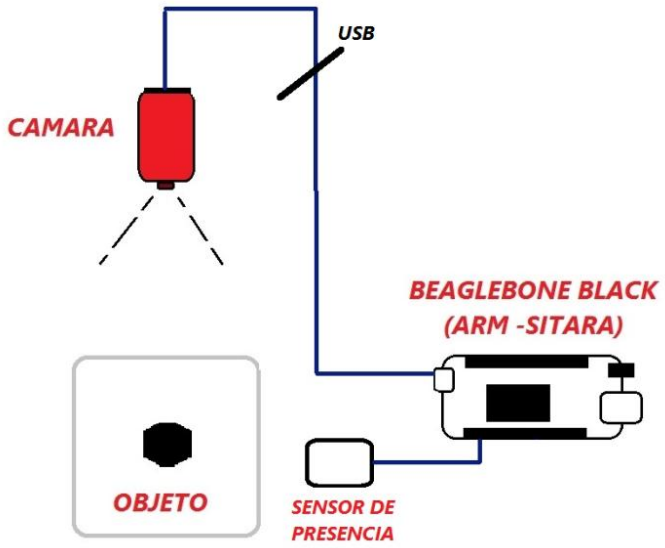


Figura 5.2 Adquisición y almacenamiento de la imagen

Para la adquisición de una imagen se utiliza el siguiente código:

```
1  while(True):

2      ret, frame = cap.read()
3      cv2.imshow('imagen',frame)
4      if GPIO.input("P8_11"):
           cv2.imwrite('captura.png',frame)
           break

5  cap.release()
   cv2.destroyAllWindows()
```

En donde se describe cada uno de los pasos:

1. Mediante la estructura *while*, el sistema estará capturando una secuencia de imágenes hasta que se cumpla una condición, la cual depende de la señal del sensor de presencia.

2. Las imágenes se adquieren con la función *cap.read()* y se guardan en la variable *frame*
3. Cada imagen adquirida es desplegada en un monitor de pantalla.
4. Si hay una señal en estado alto (proporcionada por el sensor de presencia) la imagen es capturada y almacenada en memoria.
5. La imagen desplegada en el monitor se cierra, hay una "limpieza de pantalla".

5.2.2 Escala de grises de la imagen

Posteriormente a la adquisición de la imagen, esta se convierte a escala de grises mediante la función *cv2.cvtColor(img,cv2.COLOR_BGR2GRAY).*

El objetivo es quitar el color de la imagen para reducir la cantidad de información y operar de manera rápida y eficiente. Inicialmente la imagen se encuentra en un formato RGB (Red, Green, Blue) el cual se compone mediante 24 bits, un byte por color. La conversión a escala de grises implica trabajar con 8 bits únicamente, por lo que solo se representa la luminancia o el brillo de cada pixel.

El código para la obtención de la escala de grises del objeto es el siguiente:

```
1 img = cv2.imread('captura.png',0)
2 imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

En donde:

1. La imagen que fue almacenada en la memoria previamente, es leída por el sistema.
2. Se convierte la imagen del formato RGB a escala de grises.

5.2.3 Binarización de la imagen

Ya que la imagen se encuentra en su escala de grises se procede a binarizarla con ayuda de la función *cv2.THRESH_BINARY,* la cual permite convertir de un rango de 0 a 255 (8 bits) a uno de 0 y 1 (1 bit).

La binarización en una imagen es una representación de dos colores, que por lo general son el blanco (1) y el negro (0). Este proceso es sencillo, solo se indica a cada pixel que tono de color debe tener en base a un parámetro que se lo conoce como valor de umbral. Por lo tanto la binarización consiste en saturar a una escala de grises igual a 0 o a 255.

Este método sirve para separar objetos que queremos analizar en una imagen. La separación es basada de acuerdo a la variación de intensidad entre un objeto y el fondo, a este proceso se le conoce como segmentación.

Para el caso de esta aplicación, la binarización se ocupa para segmentar una pieza con respecto a su fondo o su área de trabajo, por lo que, eligiendo un umbral correcto, se eliminan sombras indeseadas.

El código es el siguiente:

```
1 img,200,255,cv2.THRESH_BINARY
2 cv2.imshow("Binarizada",images[0])
3 cv2.imwrite('captura.png',images[0])
```

En donde:

1. La imagen que previamente estaba en escala de grises, ahora se binariza con un umbral de 200.
2. La imagen binarizada es desplegada en un monitor de pantalla
3. Se guarda en memoria la nueva imagen

5.3 Código para obtener la Función Frontera

El primer paso para obtener la BOF es detectar el contorno de un objeto, y esto se realiza con la función *findContours,* la cual devuelve un conjunto de puntos que representan la frontera de cada uno de los elementos que se encuentren en la imagen.

Esta función encuentra el contorno de la imagen propia y de los elementos u objetos que se encuentran dentro de ella. Estos contornos son enumerados de acuerdo a su tamaño, por lo que el propio perímetro de la imagen siempre será el contorno más grande y al cual se va a descartar en cada captura debido a que los contornos de interés son los que se encuentran dentro de la imagen.

En la aplicación desarrollada, solo se presenta una pieza a la vez en la región de trabajo, por lo que al capturar la imagen solo tendremos un objeto y un contorno a detectar. Sin embargo, si se presentara más de una pieza, el sistema reconocerá los contornos de todas estas y los enumerará de acuerdo al tamaño de su perímetro. El código que se utiliza para obtener el contorno del objeto es el siguiente:

```
1  contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,
   cv2.CHAIN_APPROX_NONE)

2  for i in range( ):
       lista.append(len(contours[i]))
3  lista.remove(max(lista))
   print "elemento_mayor2:", lista.index(max(lista))
4  cnt_index=lista.index(max(lista))
   best_cnt= cnt_index +1
   print "best_cnt:", best_cnt
   print "# de contornos:",  len(contours)
5  cntx = contours[best_cnt]
```

En donde:

1. Encuentra los contornos de una imagen binaria y guarda cada una de las coordenadas de los puntos que lo conforman.
2. Forma un vector llamado “lista” con la cantidad de contornos que detecta en la imagen.
3. Remueve el contorno mayor, el cual es el perímetro de la propia imagen ya que no es de utilidad.

4. El segundo mayor contorno ahora es el más grande y se agrupa en el vector “lista”.
5. El contorno más grande de la imagen se representa por la variable *cntx*.

Ya que se tiene la información del contorno del objeto, se determina el área, perímetro y centroide con las siguientes funciones:

```
1  punto = puntos_cntx/32.0
2  area = cv2.contourArea(cntx)
   print "Area pixeles:  ",area

3  perimeter = cv2.arcLength(cntx,True)
   print "Perimetro: ", perimeter

4  cx = int(M['m10']/M['m00'])
   cy = int(M['m01']/M['m00'])
   print "Centroide Cx", cx
   print "Centroide Cy", cy
```

Donde:

1. El contorno del objeto se divide en 32 puntos
2. Se obtiene el área en pixeles
3. Se obtiene el perímetro en pixeles
4. Se obtienen las coordenadas del centroide

Con las piezas capturadas en imagen, se obtiene su representación en escala de grises y enseguida una representación binaria.

Por último se encuentran las distancias desde el centroide hasta cada uno de los 32 puntos de la frontera del objeto y finalmente se obtiene la BOF representada por medio de un vector:

```
1   D0= ( (cx-x0)**2   + (cy-y0)**2 )**0.5
    D1= ( (cx-x1)**2   + (cy-y1)**2 )**0.5
    D2= ( (cx-x2)**2   + (cy-y2)**2 )**0.5
    .
    .
    .
    D31= ( (cx-x3)**2   + (cy-y3)**2 )**0.5

2   BOF=[D0, D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15, D16,
    D17,  D18, D19, D20, D21, D22, D23, D24, D25, D26, D27, D28, D29, D30, D31]
```

Lo cual se interpreta:

1. Se obtienen las distancias del centroide hasta cada uno de los 32 puntos del contorno
2. Se forma la Función Frontera de Objetos (BOF) representada por medio de un vector.

El vector de 32 elementos que representa a cada una de las dos piezas capturadas son:

- Pieza “a”:

$$BOF = [138, 130, 122, 122, 129, 141, 125, 103, 67, 40, 21, 41, 72, 106, 138, 172,$$
$$205, 236, 256, 230, 204,175, 143, 118, 86, 56, 31, 17, 36, 62, 93, 119]$$

- Pieza “b”:

$$BOF = [84, 84, 83, 83, 84, 85, 86, 86, 87, 87,87, 86, 87, 87, 85,84, 84, 85, 85,$$
$$85, 85, 85, 85, 85, 86, 86, 86, 86, 85, 85, 84, 83]$$

5.3.1 Entrenamiento de los vectores descriptivos por medio de la SVM

Los vectores de 32 elementos obtenidos, son alimentados a una Máquina de Vectores Soporte con el propósito de entrenar al sistema para el reconocimiento de piezas.

Debido a que los objetos aparecen en cualquier posición dentro del area de trabajo, se decidió hacer un recorrido de todos los elementos del vector (BOF) de tal manera que siempre se obtendrá un mismo vector descriptivo independiente del ángulo en el que la pieza se presente. Primero se encuentra el elemento mas grande del BOF y posteriormente se realiza un corrimiento a la izquierda de manera ordenada de los 31 elementos restantes, quedando de esta manera, el elemento mayor como primer índice del vector.

El vector descriptivo que se obtiene es muy similar para cada posición en la que se presente el objeto, por lo que se deduce que en este sistema, prácticamente no importa la ubicación y el ángulo en el que aparezcan las piezas, siempre se tendrá un vector descriptivo parecido en cada captura de imagen.

El extracto del código para recorrer los elementos del vector, se muestra a continuación:

```
1  index_m = BOF.index(max(BOF))
2  t=0
   BOF2=BOF[:]
3  BOF, BOF2

   for i in BOF:

4   BOF[t]=BOF2[index_m]
    index_m= index_m -1
    t=t-1
5   if t==31:
     break
```

En donde los números representan:

1. Se obtiene el índice mayor del vector BOF

2. Se inicializa una variable con valor igual a “0”. Esta sirve para llevar el conteo del recorrido de los 31 elementos.
3. Se pasa el valor de BOF a BOF2.
4. Se recorren todos los índices del vector
5. Si se recorrieron los 31 elementos, salimos de la estructura “for”.

Ya que se obtienen descriptores similares que representan a una pieza específica, se procede a capturar varias muestras de un mismo objeto, con diferente posición y ángulo dentro del área de trabajo para someter al sistema en una etapa de entrenamiento.

Cada una de las muestras es guardada en una base de datos y todos los elementos del vector BOF son etiquetados por el sistema con un -1 para identificarse de otros elementos que no conforman a dicho vector. Portanto, los elementos etiquetados con -1 pertenecen a una clase, la cual es la que identifica a nuestra pieza y todos los elementos etiquetados con $+1$ pertenecen a una clase la cual no es nuestra pieza de interés.

Las muestras de cada una de las piezas son almacenadas en bases de datos diferentes.

5.3.2 Clasificación de datos por medio de la SVM

Para el entrenamiento y clasificación de las muestras se utiliza una Máquina de Vectores Soporte con una función Kernel RBF implementada por medio de las bibliotecas LIBSVM.

El código para el entrenamiento y predicción de una clase de objeto, es la siguiente:

```
1  f = np.loadtxt("pieza1.data")
2  svm = mlpy.LibSvm(svm_type='c_svc', kernel_type)
3  svm.learn(x, y)
4  ynew = svm.pred(xnew).reshape(xx.shape)
```

En donde:

1. Se lee la información de una base de datos, la cual contiene las muestras de 10 vectores que representan al mismo objeto.
2. Se elige una clasificación SVM con un Kernel tipo RBF
3. Se entrena a un conjunto de elementos los cuales pertenecen a las muestras de la base de datos.
4. El modelo predice una clase y retorna un $+1$ o un -1 según sea el caso.

Al tener la clasificación de las dos clases, se puede predecir fácilmente a cual pertenecen los elementos de un nuevo vector.

En el caso de esta aplicación, para predecir una pieza específica, se cuenta el número de elementos que tienen etiqueta -1 y se promedia entre el número total de elementos del descriptor (31). Si el promedio es igual o mayor al 87.5% se considera que la pieza ha sido predicha de forma correcta.

5.4 Manipulación del brazo robótico Labot Pro 5

La manipulación del robot Labot Pro 5 se realiza a través del módulo serial *UART1* de la tarjeta BeagleBone Black. La siguiente tabla muestra la relación del motor que se va a mover (articulación) con respecto al comando en hexadecimal que se debe transmitir seriamente al sistema Labot Pro 5:

Articulación	**Comando en hexadecimal**
Base	A
Hombro	B
Codo	C
Muñeca	D
Mano	E
Pinza	F

El intervalo de valores de desplazamiento de cada motor es un valor numérico que va de 0500 a 2500, en donde 0500 es un extremo del movimiento de una articulación y 2500 el otro extremo.

Por ejemplo si se desea posicionar la base a la mitad de su intervalo, debemos enviar la cadena de caracteres “A1500”.

5.4.1 Espacio de trabajo del Labot Pro5

El espacio de trabajo en cual opera el brazo robótico de delimita a un área con dimensiones de 30cm de largo por 28cm de ancho, debido a que el manipulador presenta errores de precisión y repetibilidad, por lo que en el área determinada opera de manera exitosa. Tal como se muestra en la siguiente figura:

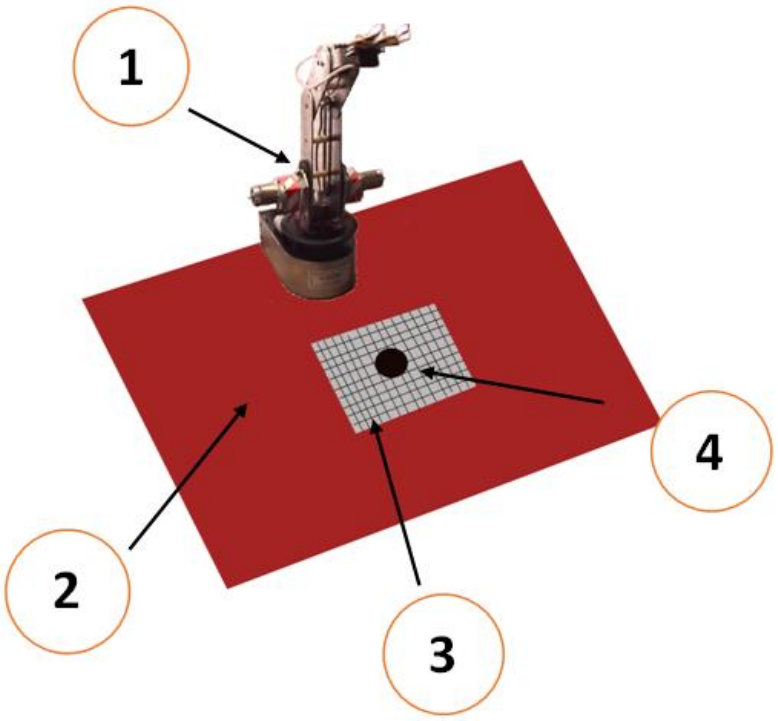


Figura 5.3 Espacio de operación

En la figura se muestran los siguientes elementos:

1. Brazo manipulador
2. Superficie fija
3. Plano de referencia (Área de trabajo)
4. Objeto

La cámara de video está situada de tal manera que la imagen coincido con el área de trabajo, la cual está delimitada a 5x4 cuadrantes, por lo que se puede identificar las coordenadas (x, y) de la posición del objeto.

La siguiente figura representa la detección de un objeto en un espacio de trabajo de 5x4 cuadrantes:

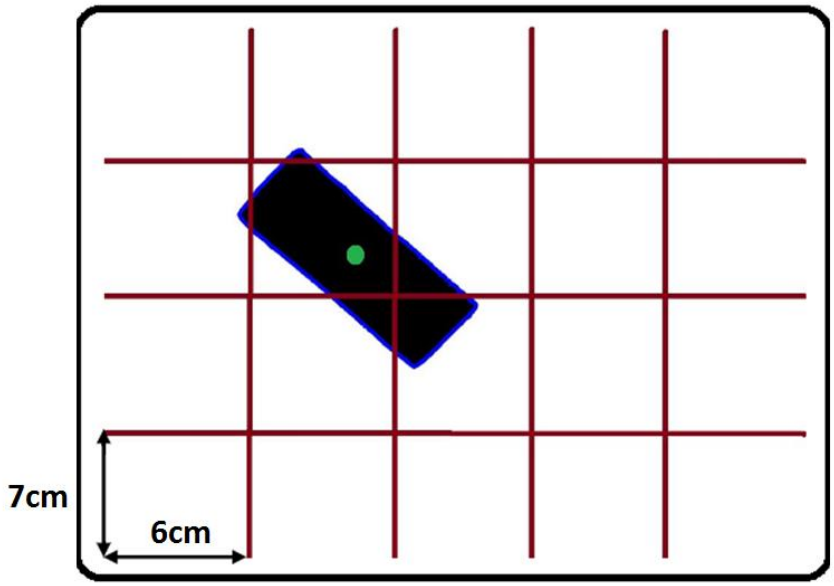


Figura 5.4Área de 5x4 cuadrantes

5.4.2 Código para manipular el brazo robótico

Primeramente se le envían comandos seriales al brazo manipulador de tal manera que este permanezca siempre en una posición inicial delimitada. Esta configuración siempre debe de ejecutarse antes de realizar cualquier movimiento que permita tomar una pieza.

Al haber realizado previamente el reconocimiento del objeto, podemos interpretar en que coordenada se encuentra el centroide del mismo, por lo que de esta manera sabremos que códigos enviar al brazo manipulador en alguno de los 20 cuadrantes del área de trabajo.

Para cada par de coordenadas (x_n, y_n) le corresponden valores $A_n, B_n, C_n, D_n, E_n \; y \; F_n$ que serán transmitidos al manipulador a través de la comunicación serial.

La siguiente figura representa la manera en la que el sistema interpreta el área de trabajo:

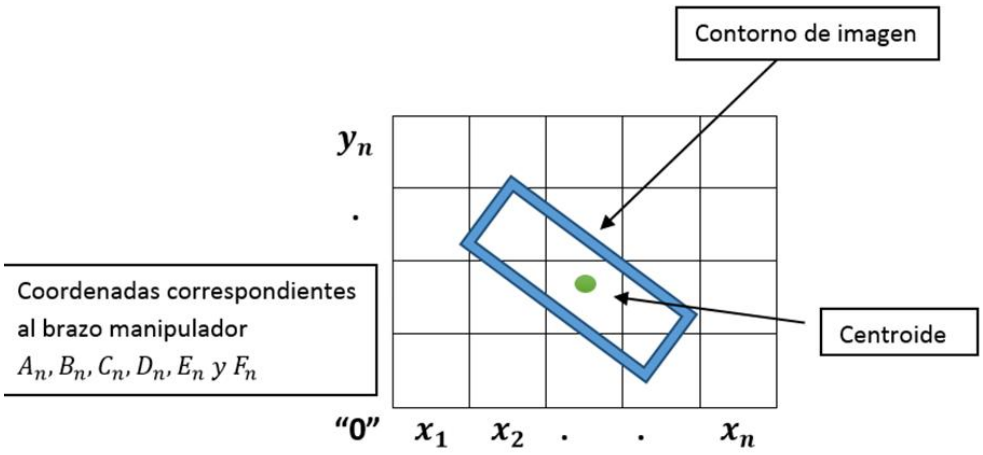


Figura 5.5 Interpretación del área de trabajo

Previamente a la recolección del objeto, se determina una altura A_0 que corresponde al ancho de la pieza que ha sido predicha. En la siguiente imagen se presenta esta referencia:

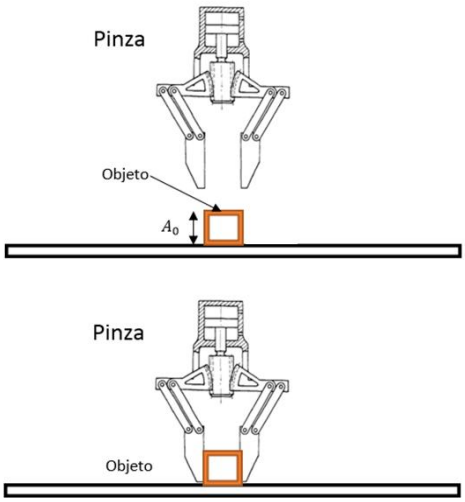


Figura 5.6Distancia entre la pinza del manipulador y el objeto

En el proceso del reconocimiento de objetos se obtienen las coordenadas del contorno y centroide de cada una de las piezas. Para saber el ángulo en el cual la "mano" del robot debe posicionarse para recoger el objeto, se obtiene la distancia más grande del centroide al contorno. En base a esa distancia y con respecto al eje de las "x" se conoce el ángulo en el que se encuentra posicionado dicho objeto, por lo que se establece el código para el giro de la "mano" del robot.

En cuanto al ancho que debe abrir y cerrar la pinza del robot para recoger el objeto, se miden las distancias más cortas desde el centroide hasta el contorno de la pieza, tal como se muestra en la siguiente figura:

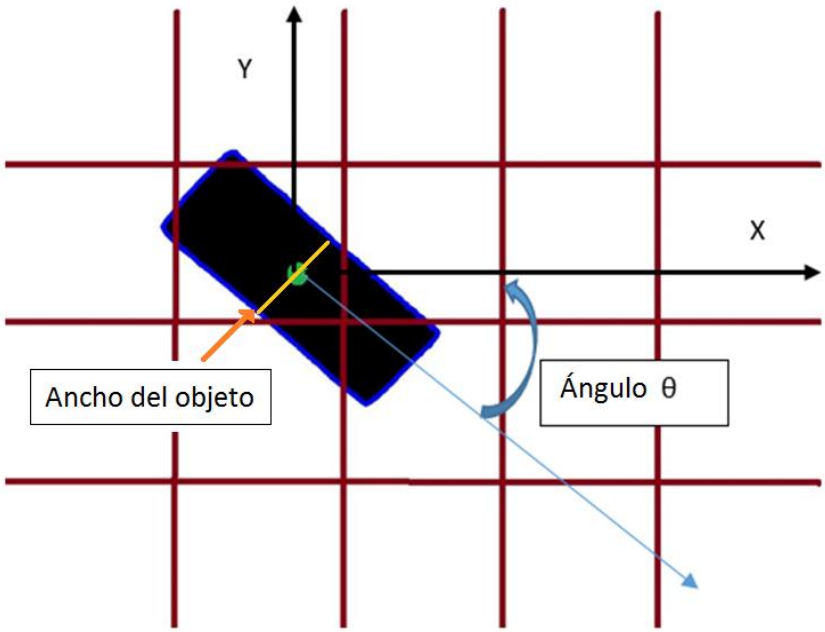


Figura 5.7 Delimitación del ancho y ángulo de la posición del objeto

Capítulo 6

Resultados experimentales

6.1 Obtención de la BOF en piezas capturadas en imagen

En la siguiente figura se muestran los cuatro objetos capturados con una cámara de video (Logitech C170). Estas son las piezas electas para que el sistema las clasifique de acuerdo a su forma:

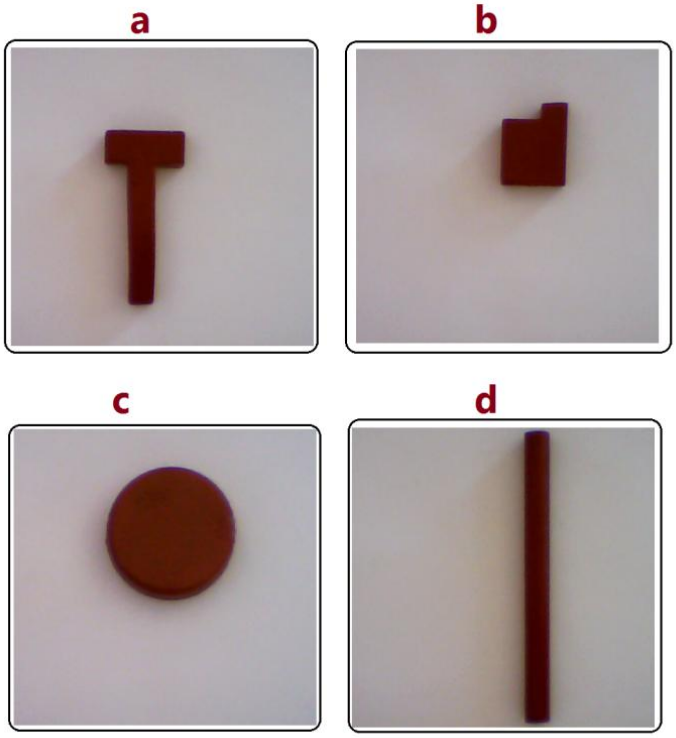


Figura 6.1 Imagen original de las piezas a clasificar

La conversión a escala de grises de cada una de las piezas se muestra en la siguiente figura:

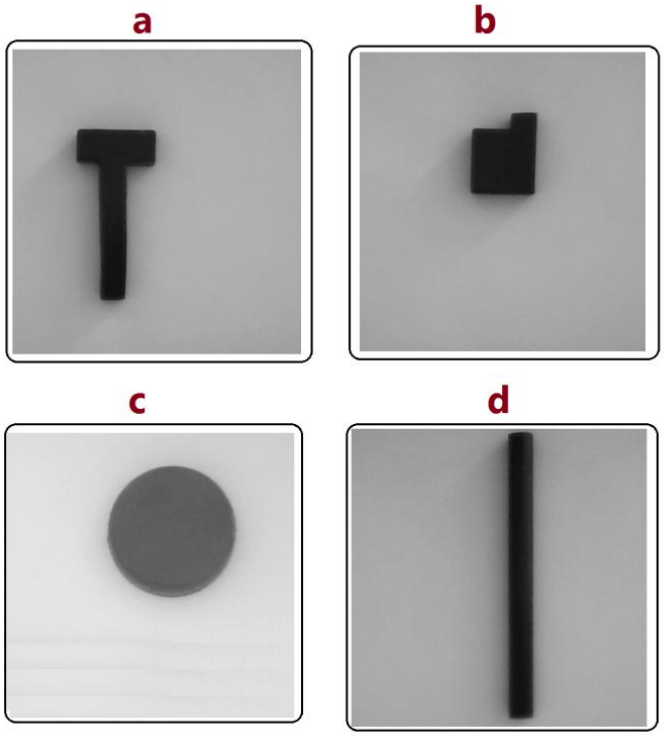


Figura 6.2 Imágenes convertidas a escala de grises

El siguiente paso fue la transformación de la imagen en escala de grises a una imagen binaria que también se almacena en memoria.

De acuerdo a las pruebas realizadas en el DISCA del IIMAS-UNAM, la técnica de binarización fue utilizada para descartar sombras en la imagen, las cuales provocan ruido y la detección de contornos erróneos.

Las figura 6.3 muestra las 4 piezas binarizadas con un umbral de 75:

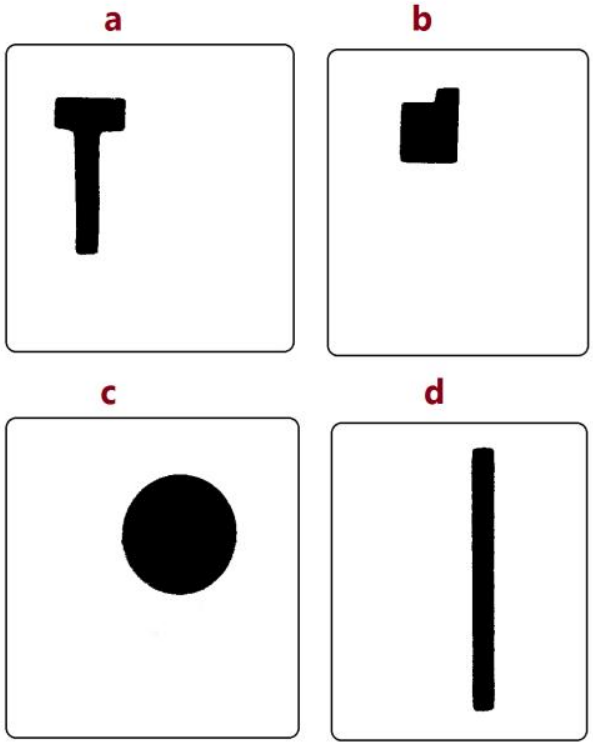


Figura 6.3 Imágenes binarizadas con umbral =75

En la imagen anterior se observa que las sombras se eliminan por lo que es más fácil obtener el contorno del objeto.

A continuación se muestra gráficamente la obtención de la BOF de cada una de las piezas:

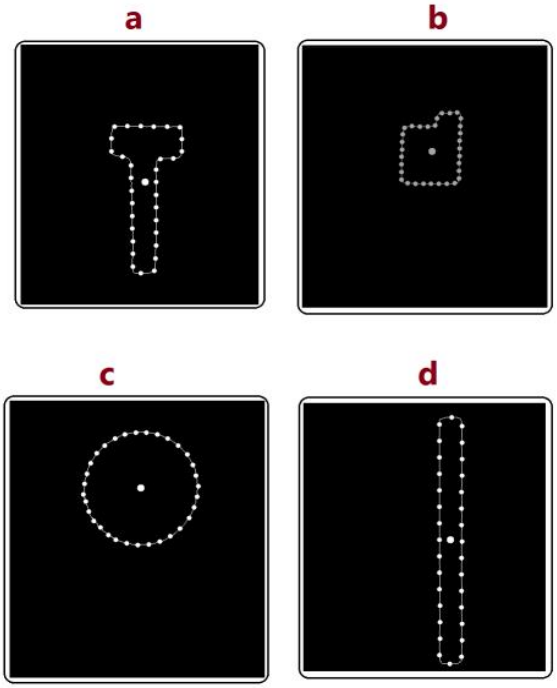


Figura 6.4Obtención del contorno y centroide de las piezas

Prácticamente a cualquier tipo de objeto se le puede obtener su BOF; en el *Anexo I* se presentan dos objetos con siluetas distintas.

Los valores de los vectores descriptivos que representan a cada una de las cuatro piezas se muestran a continuación, así como sus gráficas:

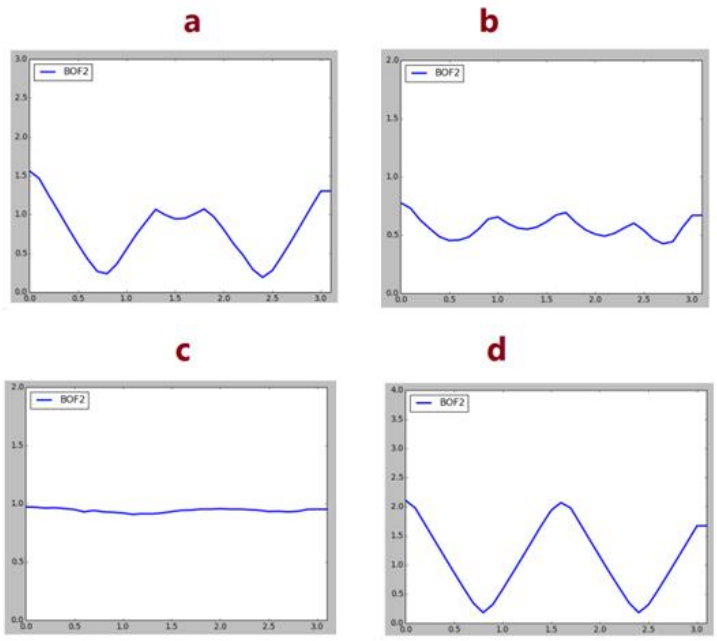


Figura 6.5Gráfica de las BOF de cuatro objetos diferentes

Los elementos del vector graficados y los que se muestran numéricamente, han sido recorridos empezando por el elemento más grande:

$$BOF_a = [158,156,134,110,87,65,44,27,22,40,60,78,96,103,95,$$
$$93,96,100,105,99,82,65,47,37,18,20,40,57,73,97,120,144$$

$$BOF_b = [73,73,73,63,54,48,44,44,48,55,64,61,55,53,52,55,$$
$$61,67,59,52,49,47,49,53,54,55,48,43,43,52,65,70]$$

$$BOF_c = [89,88,88,87,87,86,86,84,84,85,85,86,85,86,87,87,87$$
$$88,87,88,86,85,83,84,85,86,87,87,87,87,88,88]$$

$$BOF_d = [211,190,162,143,125,100,71,45,26,14,33,53,77,105,125,149$$

$$176,203,206,177,146,116,87,57,28,19,41,71,102,132,162,193]$$

Estos vectores son los que se guardan en una base de datos para el entrenamiento de la SVM.

6.2 Resultados de la clasificación SVM

Las pruebas de aprendizaje y clasificación de la Máquina de Vectores Soporte con módulo Kernel tipo RBF, se realizaron con los vectores descriptivos que se obtuvieron de cada una de las piezas.

Para estos procesos primero se entrenó al sistema con 10 vectores por objeto. En donde los elementos de la BOF se etiquetaron mediante una clase negativa $y = -1$ y todos los demás elementos que no pertenecen a dicha clase se representan con etiquetas positivas $y = 1$. En la figura 6.6 se observa que la frontera azul representa a la pieza que queremos predecir:

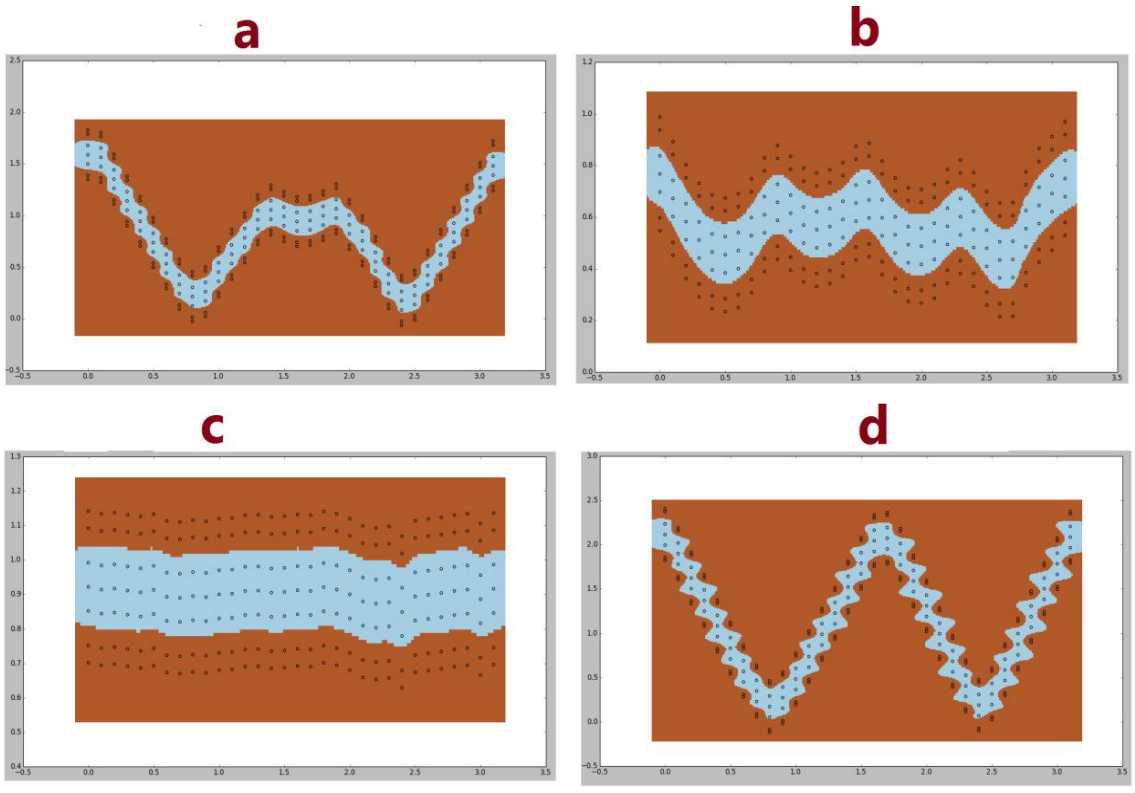


Figura 6.6 Entrenamiento y clasificación de las cuatro piezas (a, b, c y d)

En las gráficas anteriores se muestra una SVM entrenada con un módulo Kernel RBF. Cuando un vector de entrada nuevo se alimenta al clasificador, sus elementos deberán coincidir en un 87.5% o más, con los elementos de entrenamiento para que sean considerados dentro de la clase que se busca.

En la siguiente tabla se muestra el porcentaje de predicción del sistema para cada una de las piezas. Los objetos fueron capturados con diferentes ángulos y posiciones dentro del área de trabajo, a continuación se muestra el resultado de 25 capturas por pieza:

Predicciones (%)				
Posición	***Pieza_a***	***Pieza_b***	***Pieza_c***	***Pieza_d***
1	90.6	100	100	90.6
2	96.9	100	100	87.5
3	96.9	100	100	93.8
4	100	96.9	100	90.6
5	90.6	96.9	100	81.3
6	100	90.6	100	90.6
7	100	100	100	93.8
8	96.9	87.5	96.9	90.6
9	100	96.9	100	81.3
10	100	100	100	84.4
11	100	96.9	100	90.6
12	90.6	90.6	100	93.6
13	100	90.6	100	87.5
14	100	100	100	84.4
15	100	90.6	100	87.5
16	100	100	96.9	90.6
17	96.9	96.9	96.9	90.6
18	96.9	100	100	81.3
19	100	93.8	100	90.6
20	100	93.8	100	90.6
21	93.8	90.6	96.9	81.3
22	96.9	90.6	96.9	100
23	93.8	96.9	100	90.6
24	100	96.9	96.9	81.3
25	93.8	96.9	100	90.6

En la siguiente tabla se muestra el promedio total de predicción por pieza de las 25 muestras y el número de veces en cual el sistema pudo predecir de manera eficaz las piezas, es decir cuando el porcentaje fue mayor al 87.5%:

Promedio total de las predicciones			
Pieza_a	***Pieza_b***	***Pieza_c***	***Pieza_d***
97.4%	95.8%	99.3%	88.6%
Numero de predicciones eficientes			
25	25	25	18

6.2.1 Clasificación con porcentaje de predicción de 100%

A continuación se muestran algunos ejemplos de las predicciones y clasificaciones de las piezas en donde la gráfica en color verde muestra el nuevo vector BOF a ser comparado, la frontera azul representa la clase que fue entrenada previamente y el espacio en color café representa una clase la cual no es el objeto:

- Pieza “a”:

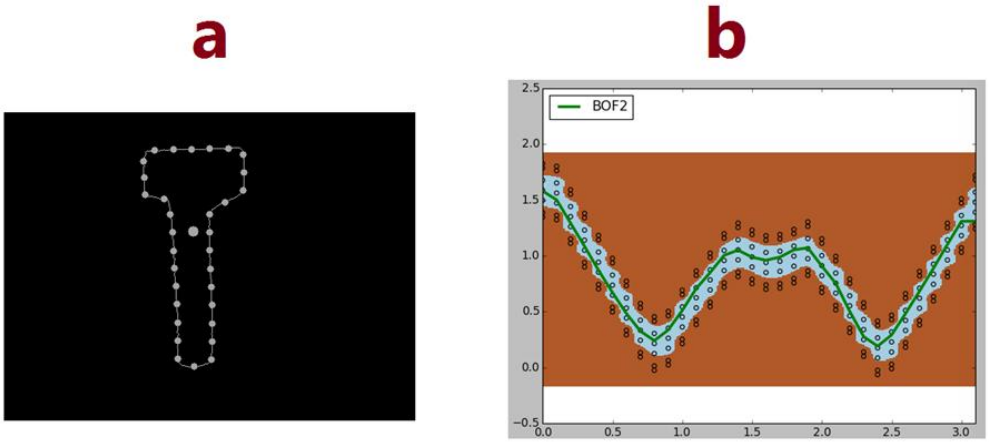


Figura 6.7 Predicción en un 100% de la figura “a”

- Pieza “b”:

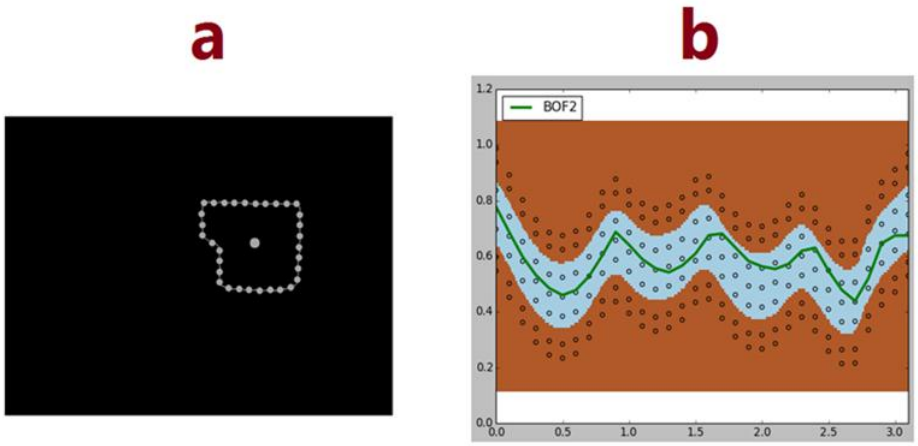


Figura 6.8 Predicción en un 100% de la figura “b”

- Pieza “c”:

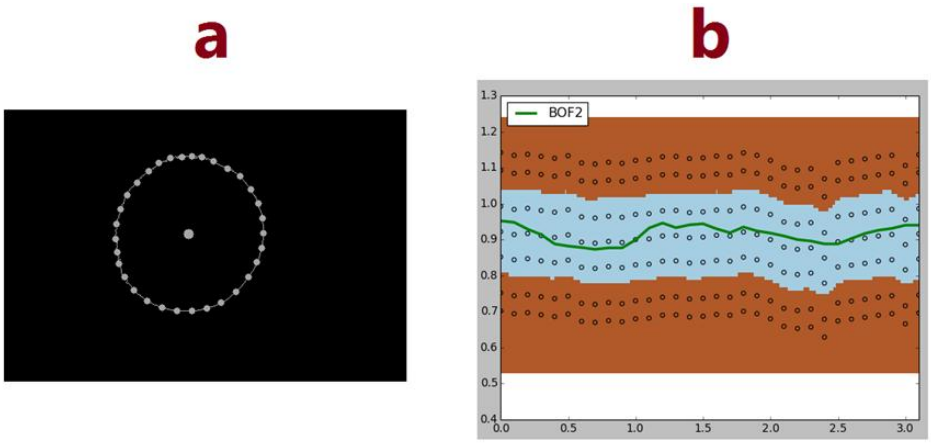


Figura 6.9 Predicción en un 100% de la figura “c”

- Pieza “d”:

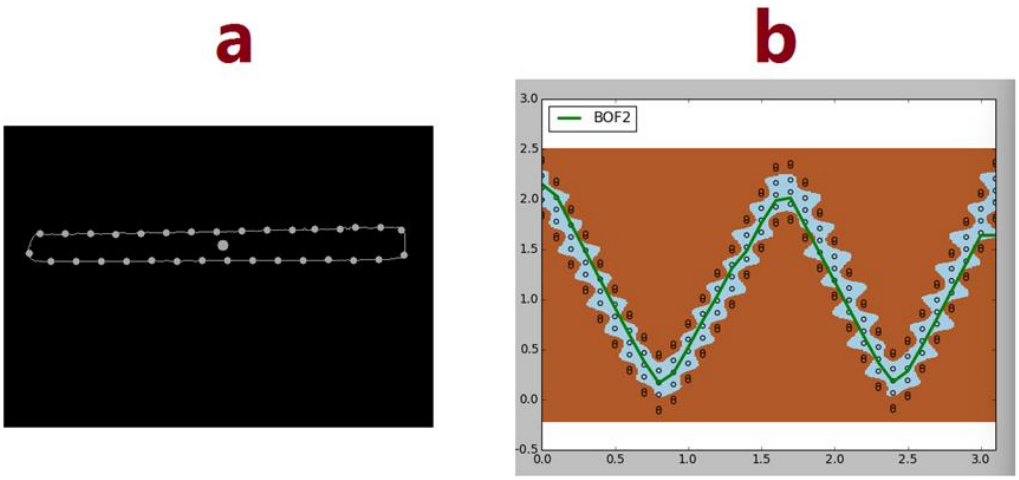


Figura 6.10 Predicción en un 100% de la figura “d”

6.2.2 Clasificación con porcentaje de predicción menor de 100%

El entrenar al sistema para que prediga algún objeto que queremos reconocer, es de suma importancia en la obtención de la función frontera, debido a que difícilmente se tendrá un mismo vector para ser comparado. Un factor que afectó al cambio de valores de la función frontera, fue el horario en el que se trabajaba, pues debido a la luminosidad, en algunas ocasiones se generaban sombras que alteraban al descriptor. Sin embargo, se predijo en la mayoría de los casos la figura deseada, aunque con un porcentaje menor al 100% tal como se muestra a continuación:

- Pieza “a” con predicción del 87.5%:

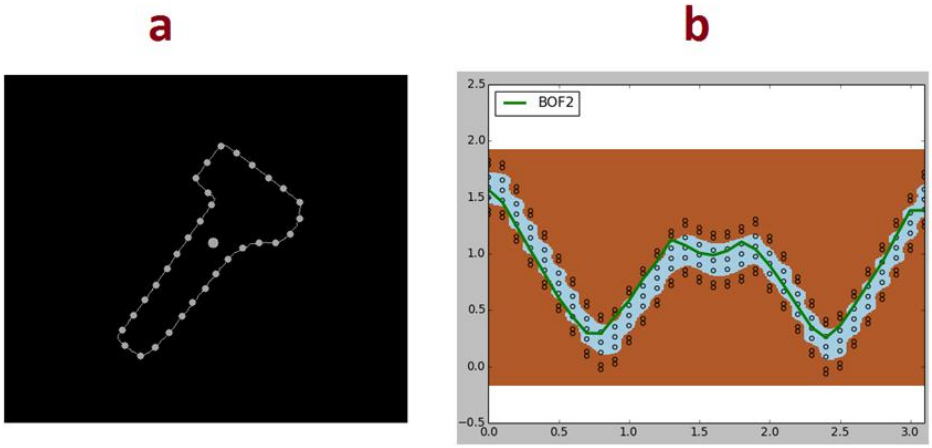


Figura 6.11Predicción de la figura “a”

- Pieza “b” con predicción del 87.5%:

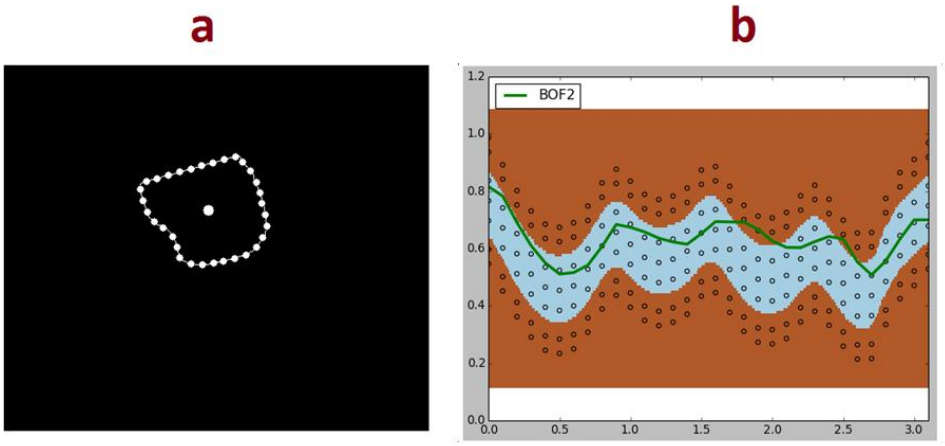


Figura 6.14 Predicción de la figura “b”

- Pieza “c” con predicción del 87.5%:

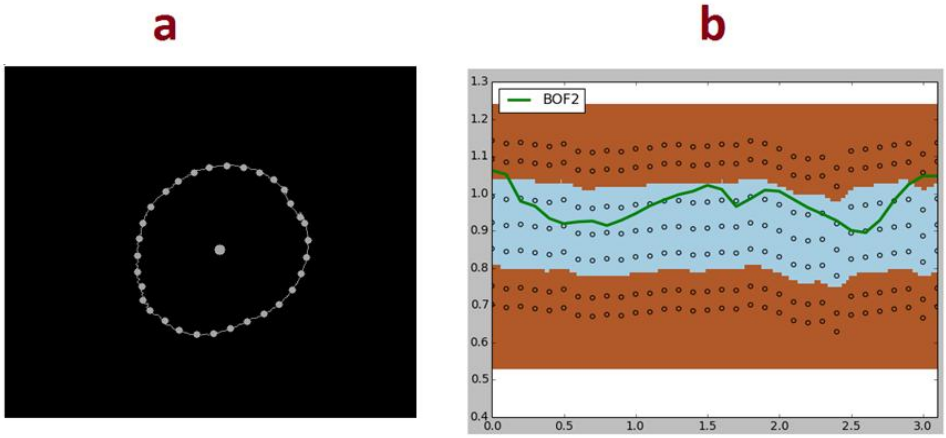


Figura 6.13 Predicción de la figura “c”

- Pieza “d” con predicción del 90.6%:

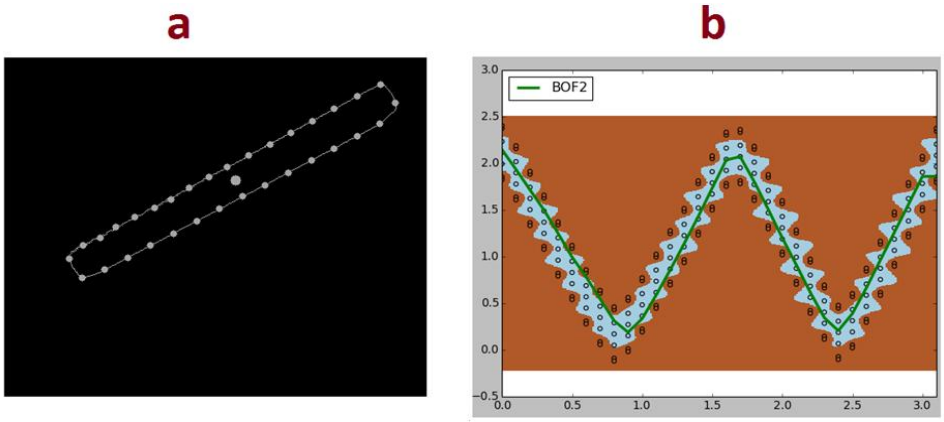


Figura 6.14. Predicción de la figura “d”

Como se observa en las figuras anteriores, los objetos presentan una alteración en sus formas y esto se debe a que el sistema detecto una parte de sus sombras producidas como perteneciente al contorno de la pieza.

Como se comentó anteriormente, esta alteración se debe a la luminosidad que se presenta en el día, sin embargo las piezas fueron predichas de manera exitosa.

6.3 Resultado de la manipulación del brazo robótico

El área de trabajo en el cual opera el manipulador es relativamente pequeña, de tal manera que no necesita realizar grandes desplazamientos, por lo que en este espacio el robot funciona de manera adecuada.

Para referenciar al manipulador con respecto al lugar al cual debe ser guiado, se le debe indicar las coordenadas del centroide y de las distancias más cortas de este hacia alguno los dos puntos opuestos entre sí. Esto nos permite conocer el cuadrante al cual el manipulador debe ser guiado

Las coordenadas del centroide fueron encontradas en el cuadrante que le correspondía de manera exitosa en todos los casos, así como las coordenadas del contorno.

En la siguiente imagen se muestra al manipulador cuando es referenciado al centroide de la pieza “a”:

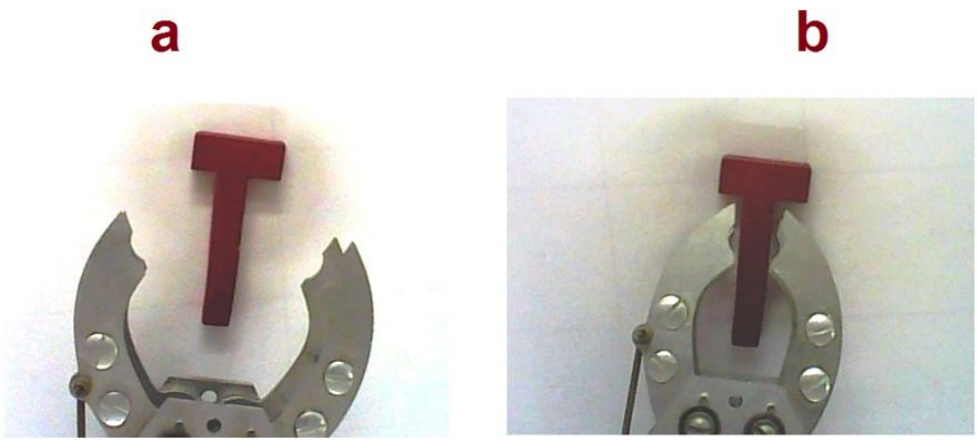


Figura 6.15 Brazo robótico sujetando la pieza "a"

Debido al diseño del brazo manipulador, no se pudo referenciar el ángulo en el cual estaba posicionada una pieza, porque la articulación de la mano y pinza no permiten agarrar un objeto que no esté en un ángulo aproximado a 90° con respecto al eje de las "x". Sin embargo, el ángulo y la función frontera se obtuvieron correctamente.

Las siguientes imágenes muestran cuando el manipulador sujeta la pieza “a” y la trasporta al contenedor correspondiente:

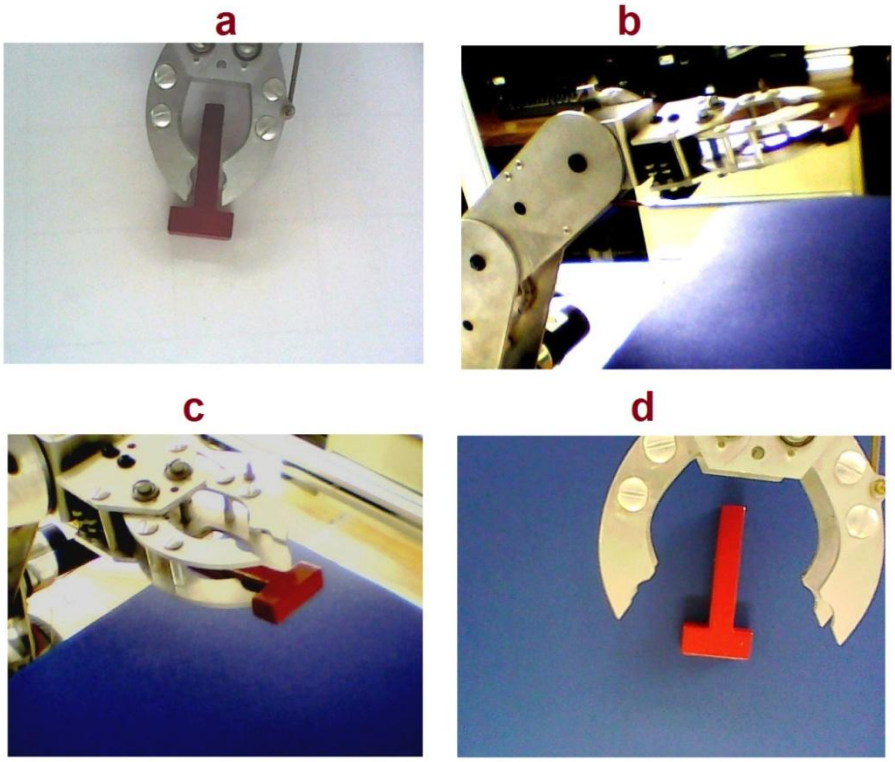


Figura 6.16 Brazo robótico transportando la pieza "a"

Capítulo 7

Conclusiones

De acuerdo a los objetivos planteados inicialmente, se comprueba que se puede integrar la etapa de adquisición y procesamiento de imágenes tanto como la manipulación de un brazo robótico en un solo procesador.

Se logró implementar la Función Frontera de Objetos (BOF) de una manera eficiente, por lo que prácticamente a cualquier tipo de pieza se le pudo obtener su vector descriptivo.

Conforme a las pruebas realizadas en el laboratorio, la captura en imagen de las piezas fue con diferente ángulo y ubicación dentro del área de trabajo y de igual forma la luminosidad, por lo que fue necesario implementar una etapa de entrenamiento para que el sistema aprendiera a detectar los objetos bajo estas condiciones de variación. De acuerdo a los resultados presentados, se concluye que con 10 muestras por pieza, es suficiente para lograr una clasificación exitosa entre elementos usando una Máquina de Vectores Soporte.

Para la manipulación del brazo robótico, se hizo una calibración, la cual consiste en referenciar a este en una matriz de 5x4 cuadrantes, por lo que no se utilizó algún algoritmo especial, sin embargo, se logró cumplir con el objetivo de que este sujete las piezas y las transporte a su debido contenedor.

Se concluye que con un sistema embebido de nueva generación como el implementado en este trabajo, se podría tener control sobre una celda de manufactura, obteniendo un sistema reducido de componentes y de espacio de trabajo, lo que refleja una disminución en los costos del sistema.

7.1 Trabajo futuro

Debido a que se desea desarrollar una celda de manufactura experimental, siempre se podrá tener mejoras, por lo que se plantea que la etapa de entrenamiento y aprendizaje sea en una sola muestra, colocando 4 cámaras en el área de trabajo, de tal forma que estas capturen en imagen diferentes ángulos de la pieza deseada. Para obtener un solo vector de las 4 imágenes, se desarrollaría

un algoritmo que permita concatenar la información extraída. Con respecto a la luminosidad, se puede tener el control sobre un sistema compuesto por *leds* para mantener siempre la misma claridad en el objeto.

En cuanto al circuito digital, se sugiere utilizar un procesador de cuatro núcleos (incorporado recientemente en las tarjetas Rasberry Pi 2 y BeagleBoard-X15) y emplear la programación en paralelo para designarles tareas específicas a cada uno de estos, y así optimizar el funcionamiento del sistema.

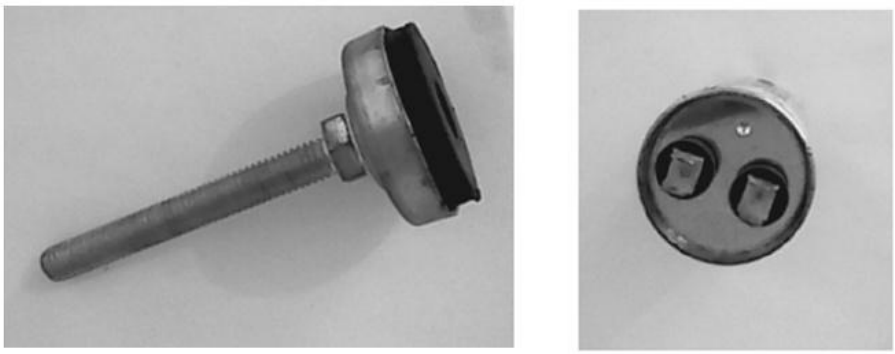
El vector descriptivo obtenido en este trabajo contiene datos del contorno de un objeto, sin embargo, este puede ser complementado integrando información de las piezas capturadas como es la profundidad, textura y color, para obtener una caracterización más detallada del objeto y por ende mayor precisión al momento de clasificar.

Anexo I

Se presentan dos objetos distintos a los cuales se les caracterizo con la función frontera. Debido a su peso, no se utilizaron en este proyecto, ya que el brazo robótico no los soporta, sin embargo se demuestra que se logro adquirir su vector descriptivo.

a b

Dos piezas capturadas en imagen y almacenadas en la BBB

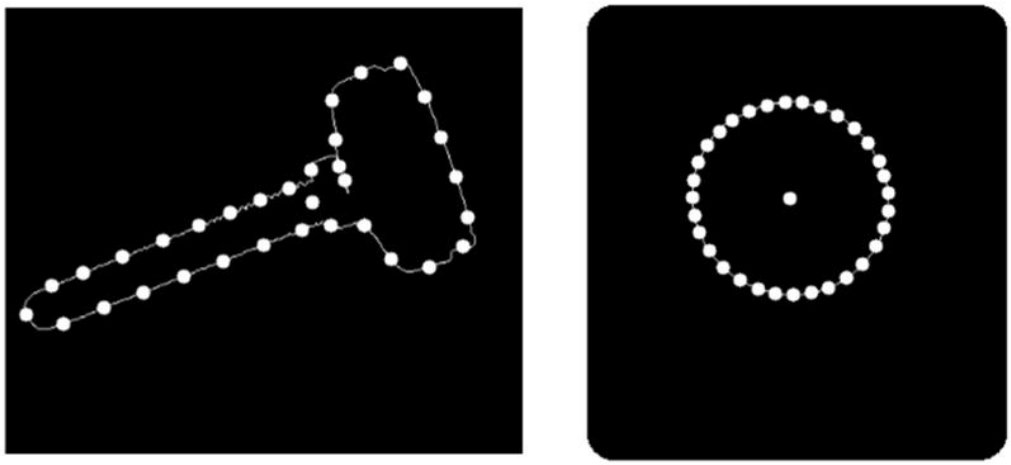


a b

Dos piezas transformadas a escala de grises

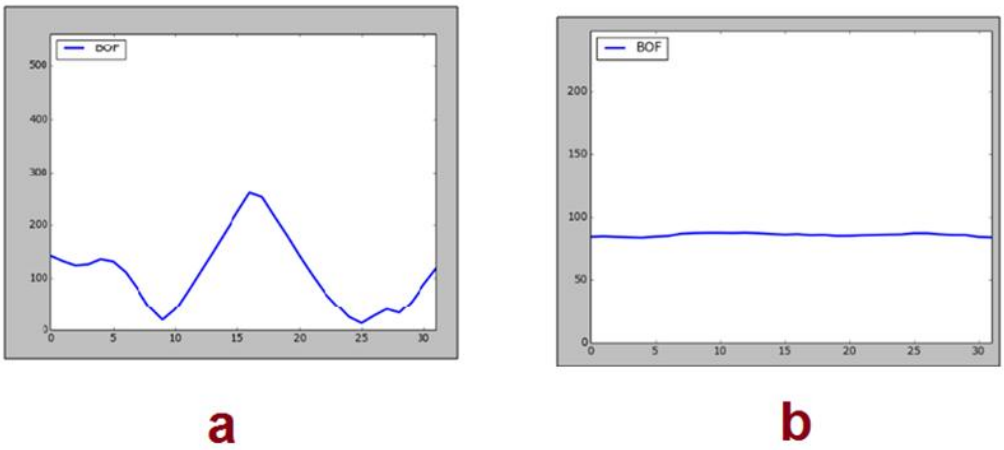
a b

Imagen de dos piezas binarizadas



a b

Detección del contorno de dos piezas por separado, obtención de sus coordenadas y detección de sus centroides.



a b

Gráfica de la BOF de dos piezas por separado

Referencias bibliográficas:

[1] Peña, M.et al, *A Learning Approach for On Line Object Recognition Tasks*, Proceedings of the Fifth Mexican International Conference in Computer Science (ENC'04), 2004.

[2] Jungbeck, M.K. Madrid, M, *Optimal Neural Network Output Feedback Control for Robot Manipulators*. Second Workshop on Robot Motion and Control, October 18-20, 2001, pp 85-90

[3] Nalini K. Ratha, Anil K. Jain, *Computer Vision Algorithms on Reconfigurable Logic Arrays*, IEEE, Vol. 10 (1), 1999, pp 29-43

[4] F.H.Y.Chan, F.K. Lam, P.W.F. Poon, H. Zhu, K. H. Chan. *Object boundary location by region and contour deformation*, Vision, image and Signal Processing, IEE Proceedings, Vol 143 (6), Dec 1996, pp 353-360

[5] S. Siddiqui, A. El-Boustani, W. Kinsner, *MODELLING MULTIFRACTAL OBJECT BOUNDARIES USING ITERATED FUNCTION SYSTEM*,IEEE, Vol 3, 2004, pp 1431-1435

[6] A. Toshev, B. Taskar, K. Daniilidis, *Object Detection via Boundary Structure Segmentation*. Computer Vision and Pattern Recognition, IEEE Conference, 2010, pp 950-957

[7] M. Peña Cabrera, et al. *Real Time Object Recognition Methodology*, IEEE Conference, pp 439-444

[8] Peña, M. et al, *Un Proceso de Aprendizaje para Reconocimiento de Objetos en Línea en Tareas Robotizadas*. Sistemas, Cibernética e Informática.Vol 1 (2).pp 85-91

[9] Nikolas P. Galatsanos.et al, *A Support Vector Machine Approach for Detection of Microcalcifications*, IEEE, Vol 21 (12), 2002, pp. 1552-1563

[10] I. Lopez-Juarez,M. Howarth,*Learning Manipulative Skills with ART*, IEEE International Conference on Intelligent Robots and System, pp 578-583

[11] K. Kim, J. Kim, S. Kang, J. Kim, J. Lee,*Object Recognition for Cell Manufacturing System*, 9th International Conference on Ubiquitous Robots and Ambient Intelligent (URAI) Daejeon Korea, Nov.2012, pp 512-514

[12] G. Pineda-Garcia. et al, *A Prototype Helping Device for the Visually Impaired Using an Optical to Mechanical Transducer*, IEEE, Nov 2013, pp 1-4

[13] M.Ferdowsi.et al, *New Monitoring Approach for Distribution Systems*. Instrumentation and Measurement Technology Conference (I2MTC) IEEE, May 2014, pp 1506-1511

[14] P. Poudel, M. Shirvaikar, *Optimization of Computer Vision Algorithms for Real Time Platforms*. 42nd South Eastern Symposium on System Theory University of Texas at Tyler, TX, USA, IEEE Conference Publications, March 7-9, 2010, pp 51-55

[15] M. Tim Jones, *Artificial Intelligence A System Approach*, Infinity Science Press LLC, Hingham Massachusetts, 2008, pp 1-7

[16] Sparsh Mittal, Saket Gupta, S. Dasgupta, *Fpga: An efficient and promising platform for real-time image processing applications*, Proceedings of the National Conference on Research and Development in Hardware & Systems, 2008

[17] M. Brady L.A. Gerhardt and H. F. Davidson. *Robotics and Artificial Intelligence*. Proccedings of the NATO Advanced Study Institute on Robootics and Artificial Intelligence held Castelvecchio Pascoli (Barga). Italy, June 26-July 8. 1983

[18] G. Stockman y L. Shapiro, *Computer Vision*, Pretince-Hall, 2001

[19] M. Sonka, V. Hlavac y R. Boyle. *Image Processing, Analysis and Machine Vision*, Pretince Hall, 1998.

[20] Raúl Benítez, Gerard Escudero y Samir Kanaan, David Masip Rodó. *Inteligencia Artificial Avanzada*. UOC, 2013

[21] Arbelaez, P. y Cohen, L. *A metric approach to vector-valued image segmentation*. International Journal of Computer Vision, 2006

[22] Bhabatosh Chada and Dwijesh Dutta Majumder, *Digital Image Processing and Analysis*. Pretince-Hall of India, 2006

[23] Ana González Marcos, Francisco Javier Martínez de Pisón Ascacíbar, Alpha Verónica Pernía Espinoza, Fernando Alba Elías, Manuel Castejón Limas, Joaquín Ordieres Meré y Eliseo Vergara González (Integrantes del Grupo de Investigación EDMANS). *Técnicas y algoritmos básicos de visión artificial*. Universidad de La Rioja, Servicio de Publicaciones, 2006

[24] José Jaime Esqueda Elizondo, Luis Enrique Palafox Maestre. *Fundamentos de procesamiento de imágenes.* Universidad Autónoma de Baja California, 2005

[25] José F. Pertusa Grau. *Técnicas de análisis de imagen. Aplicaciones en Biología*. Universidad de Valencia, 2010.

[26] Niku, Saeed B. *Introduction to Robotics: Analysis, Control, Applications*. 2a ed. John Wiley and Sons, 2011

[27] Rivera Dueñas, Juan C. *Apuntes de Control de Robots Industriales*. Facultad de Ingeniería, UNAM.

[28] LopezGarcia, Hilario y Gonzalez Librán, Rafael. *Programación de Robots Industriales, Control Remoto del Robot ASE IRB2000*. Universidad de Oviedo, 1996.

[29] Robótica CRYA. Especificaciones CRYA LABOT PRO 5. Robótica CRYA

[30] Nello Cristianini, John Shawe-Taylor. *An Introduction Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000

[31] Shigeo, Abe. *Support Vector Machines for Pattern Classification*. Springer, 2005

[32] Jabbour, Georges; Marquez, Renny y Ruiz, Lisbeth (Facultad de Ingeniería ULA, Venzuela); Maldonado, Luciano (Instituto de Estadística Aplicada y Computación, IEAC). *Off-line signature recognition using support vector machines*. RevistaCiencia e Ingenieria, 2010.

[33] Joseph Yiu. *The definitive guide to the ARM Cortex –M0*.Elsevier, 2011.

[34] A.P. Godseand D.AGodse. *Microcontroller and Embedded Systems*.Technical Publications, 2007.

[35] Joseph Yiu. *The definitive guide to the ARM Cortex –M3*.Elsevier, 2007.

[36] Sergio R. Caprile. *Desarrollo con microcontroladores ARM Cortex-M3*. Puntolibro. 2012.

[37] Derek Molloy. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*.John Wiley and Sons, 2015.

[38] Richard Grimmett. *BeagleBone Robotic Projects*. Packt, 2013

[39] Brian McLaughlin. *The BeagleBone Black Primer*. Que, 2015

[40] Lentin Joseph. *Learning Robotics Using Python*. Packt, 2015

[41] Joseph Howse. *OpenCV Computer Vision with Python*.Packt, 2013

[42] Gary Bradski and Adrian Kaebler. *Learning OpenCV. Computer Vision with the Opencv Library*. O´Reilly 2008