UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA ELÉCTRICA – PROCESAMIENTO DIGITAL DE SEÑALES

SISTEMA DE LOCALIZACIÓN EN ESPACIOS INTERIORES PARA UN ROBOT MÓVIL UTILIZANDO NUBES DE PUNTOS Y MODELOS OCULTOS DE MARKOV

TESIS

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

PRESENTA:

CARLOS ADRIÁN SARMIENTO GUTIÉRREZ

TUTOR PRINCIPAL:
DR. JESÚS SAVAGE CARMONA
FACULTAD DE INGENIERÍA

MÉXICO, CD. MX. , ENERO 2017

JURADO ASIGNADO:

Presidente: Dr. Francisco García Ugalde
Secretario: M. I. Larry Escobar Salguero
Vocal: Dr. Jesús Savage Carmona
1er Vocal: Dra. María Elena Martínez Pérez
2do Vocal: Dr. Boris Escalante Ramírez
Lugar donde se realizó la tesis: Laboratorio de Bioróbotica. Ciudad Universitaria.

TUTOR DE TESIS:

Dr. Jesús Savage Carmona

FIRMA

Agradecimientos

... A mis padres... Por todo.

... A mi hermano Christian y su esposa Jacqueline, por su apoyo y cariño.

... A los camaradas y amigos, por el apoyo incondicional: Víctor, César, Fernando, Raymundo y Telésforo.

... A mis compañeros de viaje: Alberto, Vivían, Luis, Yoloxóchitl, Erick y Nadia.

... A Azucena, por su inagotable optimismo en los momentos difíciles.

... Al Dr. Jesús Savage Carmona por su tiempo y consejos.

Finalmente agradezco a CONACYT por los recursos económicos otorgados para la realización de estos estudios de maestría.

También agradezco a la DGAPA-UNAM por el apoyo proporcionado para la realización de esta tesis, a través del proyecto PAPIIT IG100915 "Desarrollo de técnicas de la robótica aplicadas a las artes escénicas y visuales".

Resumen

En la última década la robótica ha tenido un gran desarrollo, por una parte debido a que resulta ser una área de investigación muy interesante con muchas aplicaciones útiles, y porque el mercado de dispositivos electrónicos ha favorecido su desarrollo debido al aumento de la capacidad de cómputo y la disminución de precios en los procesos de fabricación. En particular, los sistemas de visión por computadora se han vuelto asequibles para cualquier persona y ahora es común verlos integrados en sistemas de entretenimiento o dispositivos móviles con fines lúdicos. En consecuencia ha sido posible utilizar estos sistemas de visión con fines de investigación sin tener un gran costo de inversión; un ejemplo es el sensor Kinect©, el cual en principio fue diseñado como interfaz de interacción humana en una consola de videojuegos, pero este ha logrado integrarse en robots móviles como parte de los sistemas de percepción del entorno.

En este trabajo de tesis se exponen los mecanismos para utilizar modelos estadísticos y la información proveniente del sensor Kinect con el fin de asociar el lugar en el espacio donde fue capturada. El objetivo es el de dotar a un robot móvil destinado a labores de servicio, con procedimientos para su localización en entornos interiores usando información tridimensional. También se integra la información de color usando un modelo basado en la percepción cromática del ser humano.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En la última década los robots han comenzado a ser aplicados fuera del ramo industrial, donde han sido extensamente utilizados y ahora se pretende que desarrollen trabajos más comunes y menos específicos como fue en su origen la soldadura o el ensamblaje. La expansión de su campo de aplicación a dado lugar a una nueva rama de la robótica llamada *robótica de servicio* [1], esto debido a que las tareas más genéricas son mucho más complicadas de resolver que las tareas repetitivas en la industria. La federación internacional de robótica plantea el concepto *robot de servicio* como:

"Un robot que opera de manera automática o semiautomática para realizar servicios útiles al bienestar de los humanos o a su equipamiento, excluyendo las operaciones de fabricación."

En la actualidad hay pocos ejemplos de robots de servicio completamente funcionales, pero existen lugares donde se realiza investigación alrededor del mundo dedicados a la robótica de servicio, como son universidades en Estados Unidos, Europa, y algunos laboratorios especializados en la automatización de sistemas. En un futuro se espera que se desarrolle un mercado de consumo personal para este tipo de robots.

Los robot de servicio son todos aquellos diseñados con capacidades de convivir con personas y de ejecutar tareas comunes que realizan las mismas. Algunos tipos de robots de servicio son los robots domésticos, robots de vigilancia, robots con aplicaciones médicas entre otros.

Un robot doméstico debe realizar sus tareas sin intervención de las personas excepto durante su programación [2]. Uno de los primeros y más simples sistemas en el mercado que cumple con estas características es el robot de limpieza. Éste robot utiliza distintos algoritmos para aspirar todos los espacios interiores en una casa, rodear muebles y realizar su propia recarga de energía cuando lo requiere.

Otras funciones requeridas en el comportamiento de un robot doméstico son la mani-

pulación de objetos, como platos, vasos y utensilios que se encuentran en el hogar, también el ordenar espacios y estantes. Dichos robots aún no existen de manera comercial, aunque en algunos centros de investigación se está trabajando en su desarrollo como es el Laboratorio de Bioróbotica en la UNAM, a cargo del Dr. Jesús Savage Carmona.

1.1. Objetivos

Diseñar e implementar un sistema de localización para un robot móvil de servicio basado en información visual de color y de forma, utilizando modelos estadísticos llamados modelos ocultos de Markov.

En particular se logrará:

1. Capturar, procesar y describir la información de forma y color en una escena capturada por el robot.

2. Obtener una representación estadística del espacio visitado por el robot para realizar la función de localización en sus labores.

3. Medir el desempeño del sistema usando dos técnicas de clasificación: modelos ocultos de Markov y redes neuronales artificiales.

1.2. Hipótesis

Usando información visual y de forma en tres dimensiones capturada con el sensor Kinect junto con la asociación de modelos ocultos de Markov a dicha información, se mejorará el sistema de localización en un robot móvil de servicio aumentando su robustez y autonomía.

1.3. Justificación

La presente investigación surge de la necesidad de crear robots de servicio más eficientes, que se asemejen en su comportamiento a un ser humano, es decir, que sean capaces de tomar desiciones y de desplazarse en un entorno nuevo con la información que reciben de su sistema de visión y que cuenten con la capacidad de recordar características de los lugares que han visitado. Además la constante mejora en sistemas de imágenes digitales y su bajo precio hace posible el llevar a cabo experimentos rápidamente y de calidad.

Los robots de servicio personales en la actualidad son una área de desarrollo en continuo crecimiento, por lo que la mejora de los sistemas incorporados en estos robots, eventualmente permitirá su aplicación real en tareas cotidianas. La investigación en esta área de conocimiento resulta muy atractiva e incluso es aplicable a otras ramas de la ciencia fuera de la robótica de servicio.

1.4. Estructura de la tesis

En el capítulo dos se hace una breve presentación de los robots móviles y de servicio. En el capítulo tres se hace un resumen de los conceptos fundamentales de procesamiento digital de imágenes que se usan como base de la visión por computadora.

El capítulo cuatro expone los principales métodos de visión tridimensional utilizados en esta tesis, como son la detección y descripción de características de forma únicas capturadas por el sensor Kinect. En el capítulo cinco se desarrollan los conceptos de la cuantización vectorial y agrupamiento de vectores que permiten estructurar la información de profundidad de manera no supervisada.

Dentro del capítulo seis se desarrollan los métodos de clasificación de información visual utilizados en este trabajo, como son los modelos ocultos de Markov, y en comparativa con un método ampliamente usado en la actualidad; las redes neuronales artificiales.

En el capítulo siete se presenta la metodología y las herramientas de software usadas para desarrollar el sistema de localización y los detalles de los métodos de detección y clasificación utilizados. El capítulo ocho presenta los distintos resultados obtenidos de la validación de los métodos planteados.

Por último, en el capítulo nueve se exponen las conclusiones de la investigación y las posibles mejoras al sistema de localización propuesto.

Capítulo 2

Antecedentes

Los sistemas robóticos pueden emplearse para una gran variedad de tareas [3] que van desde servicios de asistencia en cirugías, tareas de montaje y soldadura de automóviles o navegación y cartografía de un entorno urbano. Una habilidad que se requiere en estos sistemas es llevar a cabo las tareas *autónomamente*, es decir, que el robot pueda cumplir con sus objetivos sin intervención de un ser humano.

Los robots móviles son mayormente máquinas autónomas, con sensores y computadoras en su interior. Debido a su capacidad de desplazarse por sí mismos y comprender su entorno, se utilizan en labores riesgosas o en las que es muy costosa la intervención de un ser humano y los sistemas de soporte de vida para el mismo. Este es el caso de la exploración en el espacio exterior o en otros planetas como son los robots enviados a Marte, donde resulta increíblemente difícil el envío de personal para trabajos de investigación [4].

Los robot móviles tienen una arquitectura general en sus sistemas de funcionamiento. Esta puede ser divida en la parte física del robot y el conjunto de programas que lo controlan.

Los componentes físicos se pueden clasificar en los siguientes subsistemas:

- Locomoción: Estos son todos los elementos que realizan el movimiento del robot como actuadores y mecanismos.
- Sensado: Son los elementos que miden las características del entorno y el estado del mismo robot.
- Comunicación: Todos los componentes que permiten la interacción del robot con su entorno y con los usuarios. Ejemplos son el sistema de conexión a internet o los sistemas de generación de voz sintética.

Los programas hechos para controlar al robot se pueden subdividir en dos grandes clases:

- Comportamientos: Son todos aquellos procedimientos que estiman una reacción solo basados en las mediciones actuales de los sensores y el estado del robot.

- Planeación: Son un conjunto de procedimientos generados automáticamente y que cumplen un objetivo específico en base a una descripción previa de las condiciones para dicha tarea. Toman en cuenta el estado actual del robot y la configuración externa de su entorno.

  Algunos sistemas de planeación son los sistemas que controlan la lista de tareas a ejecutar o los sistemas que calculan la trayectoria más adecuada para llegar a un lugar.

2.1. El problema de localización y navegación

En el libro [5] se define que la navegación está compuesta por dos procesos: la *búsqueda del camino* y la *locomoción*. La búsqueda de un camino resuelve problemas de orientación y toma de decisiones, se procura elegir la ruta más apropiada hacia un lugar con características bien definidas. La locomoción coordina los movimientos usando información del sistema sensorial y del sistema de locomoción.

En el diseño de robots móviles se han utilizado múltiples clases de sensores pero en particular se han desarrollado sistemas basados en visión artificial usando cámaras digitales de distintos tipos, como son a colores, en escala de grises, cámaras de visión nocturna entre otras, según la aplicación para la que se haya diseñado el robot.

Los sistemas de visión resultan adecuados ya que la percepción visual ofrece mucha información útil del entorno y aumenta el alcance de las aplicaciones. Un ejemplo es el que un robot puede ser capaz de navegar, detectar y manipular objetos, reconocer personas y lugares, con tan solo la información proveniente de su sistema de visión.

Se define el *problema de localización* como la estimación de la posición de un robot móvil, dicha posición puede ser relativa respecto de las mediciones, de manera global si se tiene un sistema de referencia como un mapa del entorno, o de manera estadística generando una probabilidad de estar en un lugar determinado. El problema de localización es la parte fundamental de los sistemas de navegación en un robot autónomo. Debido a la naturaleza dinámica del ambiente estos sistemas deben ser robustos a perturbaciones,

variaciones del entorno y oclusiones, además ser capaces de actualizar los registros de las locaciones que han visitado.

Los sistemas de localización pueden clasificarse como sistemas globales o sistemas locales.

- La *localización local* estima la posición del robot basada en las lecturas de los sensores y se hace una predicción en el momento, pero se limita a rastrear la posición hasta el punto donde el robot comenzó a operar.

- La *localización global* es más complicada, está aproxima la posición actual del robot respecto al entorno completo usando solo las observaciones actuales.

  Un problema fundamental de la localización global es la de inicializar los sistemas de localización al encender el robot sin que este sepa donde ha estado anteriormente, usando solo la información que recibe de sus sensores.

Por otra parte, el concepto de *navegación* se define como la planeación de las acciones para lograr llegar un lugar determinado en el ambiente. Para esto es fundamental el realizar una primera estimación de la localización, establecer el destino, crear un plan de movimiento y ejecutarlo, todo de manera autónoma y considerando cambios en el entorno. El problema de navegación es complicado de resolver ya que involucra muchos tipos de subsistemas como son sistemas sensoriales, reconocimiento de objetos, sistemas de control de movimiento, odometría, mapeo, sistemas de evasión de obstáculos, etc. Estos subsistemas deben estar integrados compartiendo la capacidad de cómputo y el sistema de distribución de energía del robot.

Los sistemas de navegación pueden clasificarse en dos grandes clases: sistemas de navegación en espacion interiores y sistemas para navegación en espacios exteriores. Cada uno a su vez, tiene subclases que se listan a continuacion:

- Navegación en interiores.

  - Navegacion basada en mapas. Son los sistemas de navegación con una descripción previa del entorno nombrada comúnmente como un mapa. Este mapa puede ser una representación geométrica como un conjunto de polígonos, o un grupo de celdas ocupadas y libres, o un mapa que representa la relación en el espacio de las características que se han observado, conocido como mapa topológico.

  - Navegación basada en construcción de mapas. En este tipo de navegación el robot es capaz de dezplazarse por el entorno mientras construye una representación del mismo. Una técnica popular es SLAM (siglas del inglés *Simultaneous Localization and Mapping: Localización y mapeo simultáneo*) [6].
  - Navegación sin mapas. En este tipo de navegación el robot no cuenta con un mapa, solo tiene disponible la información sensorial en el momento. Algunas técnicas son el flujo óptico [7] y la búsqueda de correspondencias basada en apariencias [8].

- Navegación en exteriores.
  - Navegación en ambientes estructurados. Esta es la navegación en ambientes abiertos pero con algún tipo de estructura como la pintura para indicar la separación de los carriles en las carreteras o la apariencia de un sendero en caminos de terracería. Esta técnica es popularmente conocida como *seguir el camino*.
  - Navegación en ambientes no estructurados. Estos sistemas de navegación son los más demandantes ya que no existe algún tipo de estructura constante que el robot pueda rastrear, por lo que debe explorar el entorno para construir un mapa con las mediciones de algún sistema de odometría y utilizar los datos que ha recolectado por sí mismo. La odometría también limitará los errores durante su desplazamiento.

2.2. Sistemas de navegación basados en visión por computadora

La tecnología de visión por computadora es por mucho la selección óptima para sistemas de navegación en robots móviles, debido a la relación de la información proporcionada contra los costos del sistema. Actualmente se venden cámaras de calidad aceptable a un precio mucho menor que los escáneres láser, de sonar y de radar. Los sensores de visión ayudan en el diseño de sistemas económicamente viables con menos limitaciones que sensores más simples. Por otra parte los sensores de visión pueden proporcionar información que no está disponible para otros sensores, por ejemplo proporcionan información semántica de una escena a través de la comprensión de su aspecto visual.

Los sistemas de navegación basados en visión por computadora pueden tener muchos componentes interactuando de forma compleja pero tradicionalmente pueden organizarse en cinco clases:

- Adquisición. Sistema de lectura y transferencia de los datos de los sensores.

- Mapas. Un sistema de construcción de representaciones del entorno.
- Extracción de características. Son todos los procedimientos para extraer información de las imágenes adquiridas como es el color, la textura, forma, etc.
- Reconocimiento de características. Es el subsistema que reconoce marcas distintivas del entorno capturadas en las imágenes y es capaz de encontrarlas en presencia de distorsiones o variaciones del entorno.
- Autolocalización. Es la etapa que estima la posición actual del robot en función de las marcas o características detectadas y su ubicación previa.

La navegación se divide a menudo en cuatro problemas principales:

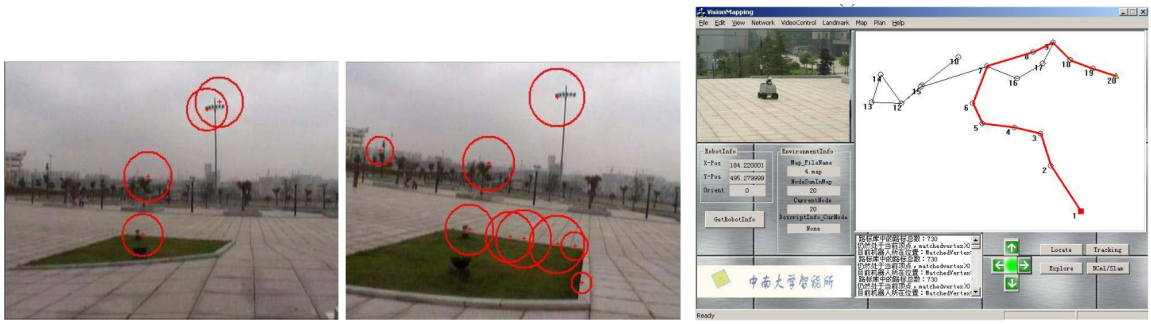
- Percepción del mundo. Es la medición del entorno y la conversión de dichas mediciones en características únicas.
- Planeación de trayectorias. Es la creación de secuencias de objetivos que el robot debe cumplir.
- Generación de trayectorias. Es la ejecución de cada objetivo para obtener una ubicación final.
- Rastreo de trayectoria. Es el sistema de control para mantener al robot en la trayectoria hacia su destino.

Ejemplos de sistemas de navegación en robots móviles basados en visión se estudian en [9] dónde se propone la selección de descriptores SIFT (siglas del ingles *Scale-Invariant Feature Transform: Transformación de caracteristicas invariantes a escala*) para construir mapas topológicos basados en modelos ocultos de Markov, en espacios interiores y exteriores (ver figura 2.1).

También en [10] se aborda la misma problemática en ambientes interiores dinámicos usando modelos ocultos de Markov, un ejemplo se muestra en la figura 2.2.

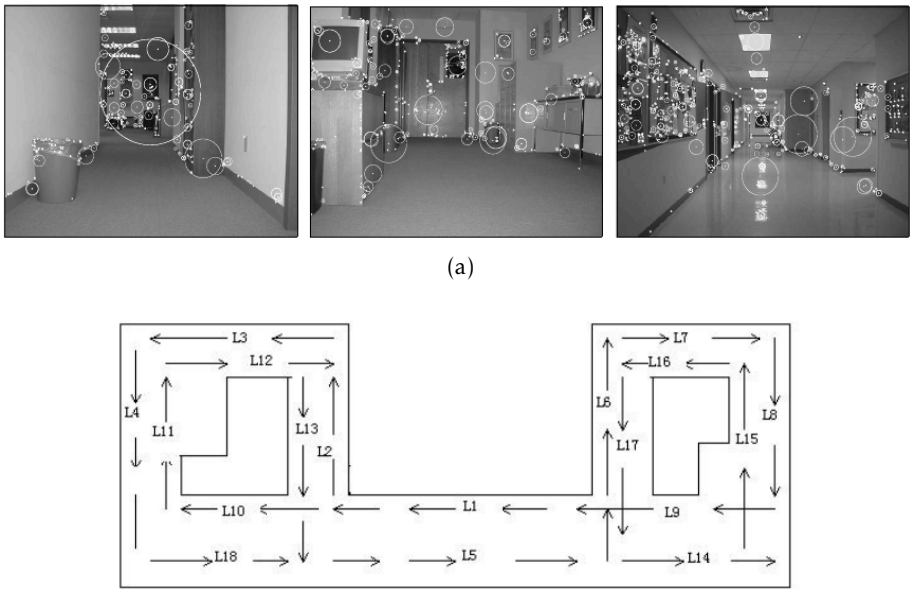
En [11] se plantea construir mapas topológicos del terreno usando información tridimensional (ver figura 2.3).

En [12] se utilizan características visuales para reconocer estructuras de corredores en entornos interiores, un ejemplo se muestra en la figura 2.4.



(a) (b)

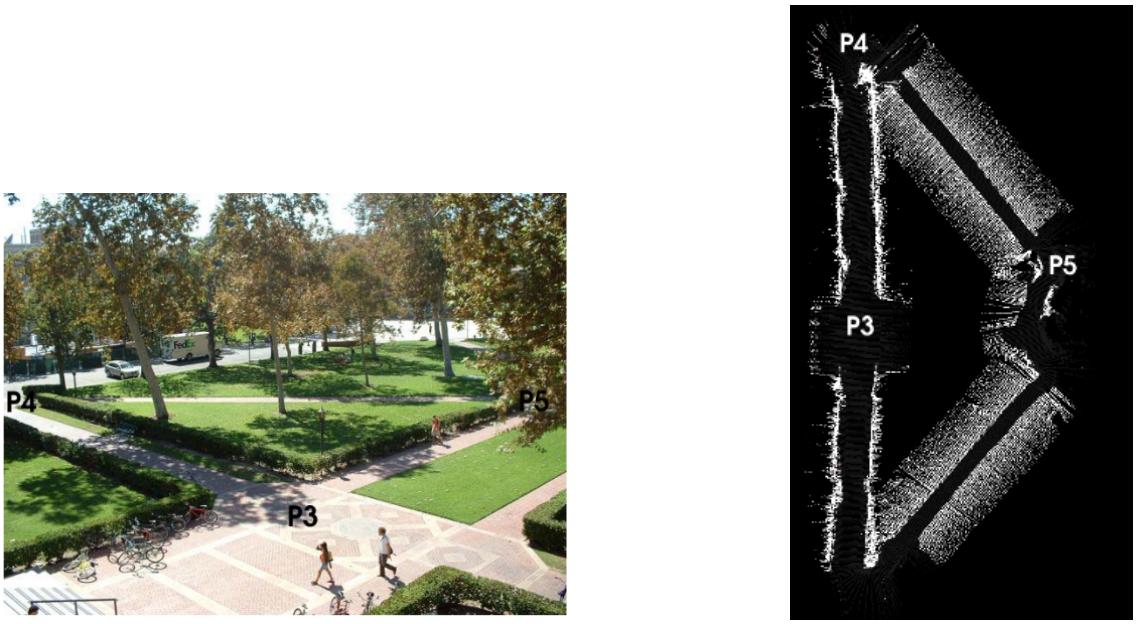
Figura 2.1: Ejemplo de sistema de navegación basado en el descriptor SIFT y mapas topológicos. (a) Ejemplo de seleccion de puntos. El diametro de los circulos rojos indican la escala seleccionada. (b) Construcción del mapa topológico. *Imágen (a) y (b) tomada de [9].*



(a)

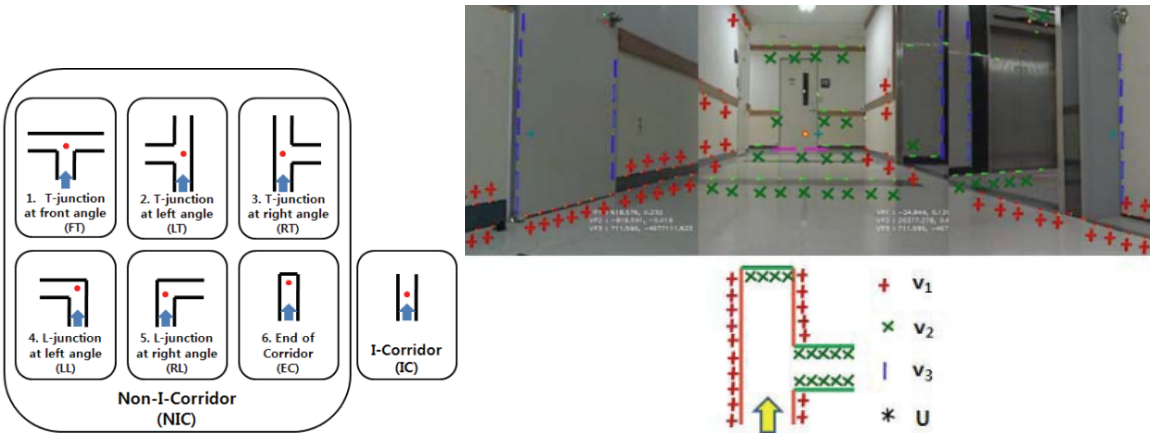
(b)

Figura 2.2: Ejemplo de sistema de navegación en entornos interiores. Las etiquetas L1 a L18 corresponden a las distintas regiones donde se tomaron las imágenes. (a) Ejemplo de seleccion de puntos basado en SIFT. Los circulos blancos indican la escala seleccionada. (b) Construcción del mapa topológico. *Imágen (a) y (b) tomada de [10].*



(a) (b)

Figura 2.3: Ejemplo de sistema de navegación basado en información tridimensional. Las etiquetas P3 a P5 identifican las distintas regiones en el mapa. (a) Ejemplo de seleccion de regiones. (b) Construcción del mapa tridimensional. *Imágen (a) y (b) tomada de [11].*



(a) (b)

Figura 2.4: Ejemplo de sistema de navegación basado en la forma de un pasillo. Las etiquetas ***x***, +, |, identifican las distintas formas y su secuencia genera una estructura que se analiza con modelos ocultos de Markov. (a) Ejemplo de seleccion de formas. (b) Construcción del mapa. *Imágen (a) y (b) tomada de [12].*

Capítulo 3

Representación de imágenes en sistemas digitales

Una imágen natural observada con una cámara, un telescopio, un microscopio o cualquier otro tipo de instrumento óptico presenta una variación de colores y sombras continua, imágenes de este tipo se llaman *imágenes analógicas*. Para que una imágen analógica, ya sea en blanco y negro, en escala de grises o a color, pueda ser manipulada usando una computadora, primero debe convertirse a un formato adecuado. Este formato es la *imágen digital* correspondiente.

La transformación de una imágen analógica a otra discreta se llama digitalización y es el primer paso en cualquier aplicación de procesamiento digital de imágenes.

3.1. Imágen digital

El término *imágen digital monocromática* o simplemente imágen, es definido por González en [13] como una función bidimensional de la intensidad de luz $f(x,y)$, donde x e y indican las coordenadas espaciales y el valor de f en cualquier punto (x,y) es proporcional a la luminosidad de la imágen en dicho punto. Una *imágen digital* es una función $f(x,y)$ que ha sido discretizada tanto en coordenadas espaciales como en luminosidad (ver figura 3.1). Se puede considerar como una matriz cuyos índices de renglón y columna identifican un punto en la imágen y el valor en ese elemento identifica el nivel de luz. Los elementos de estos arreglos son llamados *elementos de imágen* o *píxeles*. También se usa el término *nivel de gris* para referirse a imágenes monocromáticas. Por otra parte, las imágenes a color están formadas por la combinación de imágenes individuales, represen-

tadas en un espacio de color específico. Por ejemplo, uno de los espacios de color más usados es el llamado RGB (siglas del inglés *Red Green Blue: Rojo Verde Azul*), donde una imágen se representa como un conjunto de tres imágenes R, G y B. La imágen R representa la intensidad de color rojo, la imágen G la intensidad de color verde y la imágen B la intensidad de color azul, la combinación de las tres imágenes genera el color. Las técnicas para procesar imágenes con niveles de grises son también utilizadas para imágenes de color.

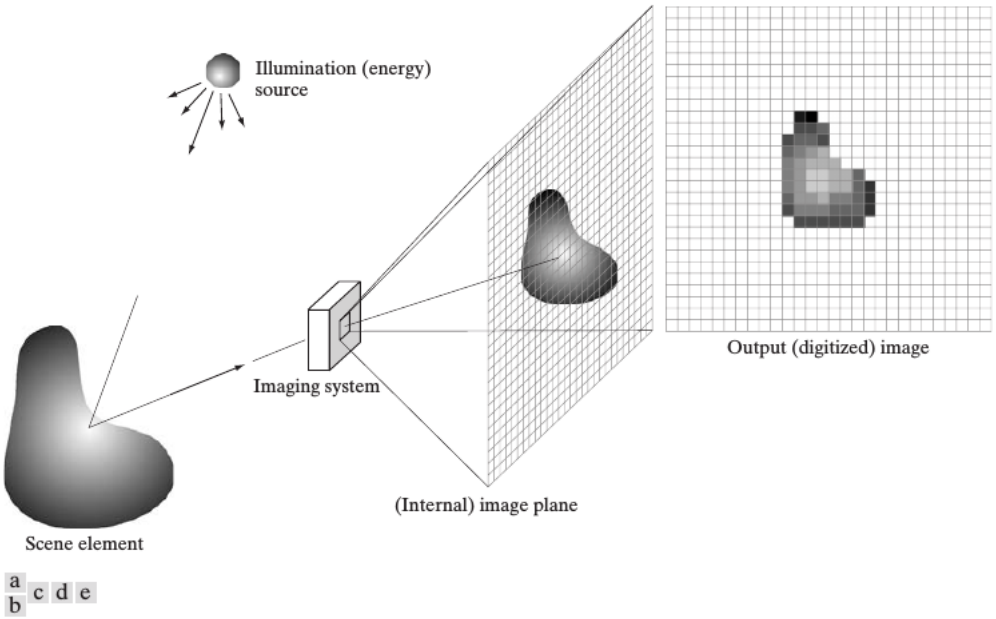


Figura 3.1: Ejemplo de digitalización de una imágen. (a) Fuente de iluminación. (b) Objeto en la escena. (c) Sistema de imágenes. (d) Proyección de la escena en el plano de la imágen. (e) Imágen digitalizada. *Imagen tomada de [13].*

La forma de representar una imágen es una matriz con M renglones por N columnas. Se usa la convención de que el eje vertical se incrementa hacia abajo de la imágen y el eje horizontal hacia la derecha, ambos a partir del primer píxel ubicado en la esquina superior izquierda, este será el origen de la imágen $(0,0)$ como se muestra a continuación:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ x(1,0) & f(1,1) & \dots & f(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix}$$

3.2. Nubes de puntos

Un conjunto de puntos en tres dimensiones $\{p_1, p_2, .., p_n\}$ desconectados entre sí, con cada punto expresado como $p_i = (x_i, y_i, z_i)$, se le llama *nube de puntos* (del inglés *point cloud*). La falta de conectividad entre los puntos hace parecer que están flotando al gráficarlos como se observa en la figura 3.2. Estos puntos están representados en un sistema de coordenadas, donde el origen esta en el centro óptico del lente en el dispositivo de captura.

En su forma más simple, una nube de puntos solo contiene información de posición de la superficie que ha sido capturada con el sensor. En representaciones extendidas, puede incluirse información de las componentes del vector normal o el color del objeto capturado, para cada punto, esto se expresa como una tupla $p_i = (x_i, y_i, z_i, \vec{n}_{xi}, \vec{n}_{yi}, \vec{n}_{zi}, R_i, G_i, B_i)$. Debido a que las nubes de puntos contienen información en un sistema de referencia, se usan como modelo del mundo en sistemas navegación de robots, en representaciones previas antes de construir un modelo basado en mallas, como se hace en el diseño asistido por computadora, también para procesos de verificación de calidad en la industria o incluso para trabajos de restauración de obras de arte para crear registros históricos [14] [15].

(a) (b)

Figura 3.2: Ejemplo de una nube de puntos. (a) Imágen de una escena real. (b) Nube de puntos asociada. *Imágen (a) tomada del conjunto de datos en [16].*

3.3. Sensor Kinect

El sensor Microsoft© Kinect© fue lanzado al mercado junto con la consola Xbox 360 el 4 de Noviembre del año 2010 en Estados Unidos, basado en el hardware desarrollado por la empresa israelí PrimeSense©. Kinect es un sensor que permitía la interacción con la consola Xbox, mediante la captura, rastreo e interpretación de movimientos de cuerpo

completo, reconocimiento de gestos y comandos por voz. En la semana de su lanzamiento al mercado comenzó una *carrera por hacer ingeniería inversa en el dispositivo* para poder utilizarlo junto con cualquier computadora según anunció Kyle Machulis, un hacker en Berkeley, California [17]. Adafruit Industries© ofrece esa misma semana un premio a quien logré comunicar el dispositivo con una computadora.

El día 10 de Noviembre del 2010, Hector Martin libera el código del proyecto *libfreenect*, disponible en Github [18]. Este proyecto de ingeniería inversa alcanza su completa funcionalidad en Marzo del año 2011. Debido a lo anterior, el dispositivo Kinect libera su potencial para su uso en sistemas de visión artificial en distintas áreas de investigación, debido a su bajo costo (150 dólares en 2010) y su portabilidad entre sistemas operativos. En respuesta Microsoft libera su kit de desarrollo de software para uso no comercial en Febrero del año 2011, pero solo para su sistema operativo Windows©.

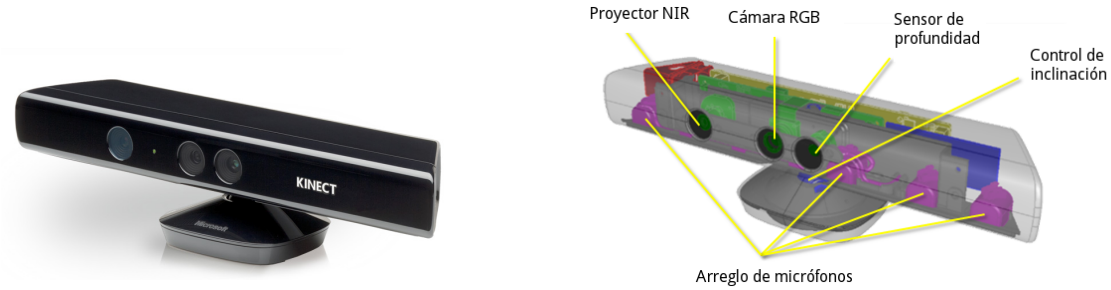
Una de las primeras empresas en explotar el potencial del dispositivo fue Willow Garage© en Palo Alto, California; una empresa en el negocio de sistemas de visión y robótica fundada desde el año 2008. En Diciembre del año 2010, Willow Garage junto con PrimeSense liberan los controladores como software libre llamado OpenNI (del inglés *Open Natural Interaction*), a travéz de la creación de la organización sin fines de lucro del mismo nombre, con el fin de certificar e impulsar el uso de la interacción natural con los usuarios. Además se incluyó el software llamado NITE, como interfaz para el reconocimiento de acciones y gestos. Actualmente Willow Garage es el principal proveedor de soporte para el desarrollo de las bibliotecas OpenCV [19], ROS [20] y PCL [21] extensamente utilizadas en sistemas de visión por computadora para robots, todas ellas con soporte incluido para interactuar con el sensor Kinect.

PrimeSense es comprada por la empresa Apple© en el año 2014. En la actualidad la primera versión de Kinect y la reciente versión lanzada al mercado en el año 2014, cuentan con soporte por la biblioteca libfreenect distribuida como software libre.

3.3.1. Especificaciones del sensor Kinect

El sensor Kinect es un sensor de imágenes con uso altamente extendido entre la comunidad que desarrolla aplicaciones de visión por computadora. Este sensor usa una variación de visión estéreo llamada *visión estéreo con luz estructurada* que requiere la proyecćion de un patrón conocido para estimar la profundidad. El sensor integra una cámara de color RGB, un proyector láser en el espectro NIR (siglas en inglés para *Near Infra-Red*: *cercano al infrarrojo*), un sensor de profundidad, un acelerómetro, un arreglo de 4 micrófonos y un

motor que controla la inclinación de todo el arreglo de sensores como se muestra en la figura 3.3.



(a) (b)

Figura 3.3: Estructura del sensor Kinect. (a) Vista exterior. (b) Estructura interna.

Este sensor cuenta con las especificaciones descritas en la tabla 3.1 para resolución y velocidad de transferencia. Otros parámetros como ángulo de visión, acelerómetro y sistema de captura de audio se describen en la tabla 3.2.

Cámara	Resolución (píxeles)	Tasa de transferencia (fps)
RGB	1280x1024	10
	640x480	30
Profundidad	640x480	30

Tabla 3.1: Especificaciónes de transferencia y resolución.

Característica	Especificación
Ángulo de visión	43° vertical 57° horizontal
Formato de audio	16 kHz, 24 bits, PCM
Entradas de audio	4 micrófonos con un ADC de 24 bits por canal
Acelerómetro	Acelerómetro con modos 2G/4G/8G. 1° de resolución
Rango de distancia	[0.4m , 4m] nominal

Tabla 3.2: Especificaciones adicionales.

3.4. Visión estéreo

La visión estéreo esta basada en el principio de triangulación de características visuales a partir de un par de imágenes, esto permite construir un modelo para estimar la profundidad.

Dadas dos imágenes, una izquierda y una derecha, de las cuales se conoce que fueron obtenidas de un sistema de cámaras, montadas sobre el mismo plano horizontal (ver figura 3.4), con distancia b entre ellas. Es posible determinar la distancia a cada punto capturado en cada píxel, relacionando el desplazamiento horizontal de una característica localizada en la imágen izquierda y la misma característica encontrada en la imágen derecha (a este proceso se le llama *búsqueda de correspondencias*). Además mediante el proceso de calibración de la cámaras es posible conocer los parámetros intrínsecos (ver [22] y [23] para técnicas de calibración estéreo), entre ellos, la distancia focal f.

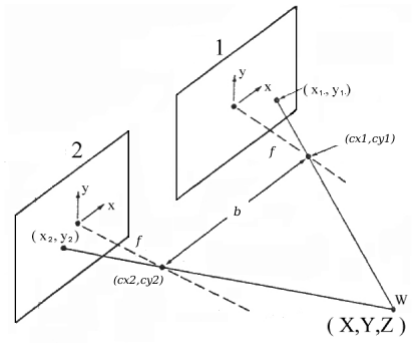


Figura 3.4: Configuración espacial de un sistema de visión estéreo.

Así, se define la disparidad $m(x,y)$ entre correspondencias, como la diferencia entre la posición (x,y) del centro de una característica visual en la imágen izquierda y la posición de la misma (x',y'), en la imágen derecha. Esto se define mediante la ecuación 3.1.

$$m(x,y) = (x,y) - (x',y') = \frac{bf}{Z} \tag{3.1}$$

Donde Z es la profundidad asociada al centro de dicha característica en el sistema de coordenadas del mundo W como se muestra en la figura 3.4. Dado que se desconoce la coordenada Z pero es posible obtener la disparidad para todos los puntos de la imágen, basta con despejar Z en la ecuación 3.1 como se muestra a continuación:

$$Z = \frac{bf}{m(x,y)}$$

La disparidad $m(x,y)$ usualmente se expresa en unidades de píxeles y la distancia focal f en unidades de píxeles sobre distancia, por lo que debe convertirse a unidades de píxeles de la forma $f_x = \frac{f_x^u}{s_{px}}$ donde s_{px} es el tamaño del píxel en la coordenada x y f_x^u es la distancia focal en el eje x en unidades métricas. Para el eje y se tiene $f_y = \frac{f_y^u}{s_{py}}$.

Al calibrar la cámara se obtienen dos distancias focales, f_x y f_y, por lo que f se obtiene como función de ambos valores $f = F(f_x, f_y)$ interpolando o de manera sencilla, promediando.

Al conjunto de todas las disparidades en cada píxel se le llama *mapa de disparidad* y es también una imágen digital. La proyección del mapa de disparidad en el sistema de referencia del mundo W, solo en su amplitud se llama *mapa de profundidad*, y al proyectar las coordenadas horizontal, vertical y la profundidad del mapa de disparidad se obtiene una *nube de puntos*, que es un conjunto de puntos tridimensionales. Esta proyección se realiza según las ecuaciones 3.2, 3.3 y 3.4.

$$X = \frac{(u - c_x)\ Z}{f} \tag{3.2}$$

$$Y = \frac{(v - c_y)\ Z}{f} \tag{3.3}$$

$$Z = \frac{(u,v)}{K} \tag{3.4}$$

Los valores u y v son las coordenadas horizontal y vertical en la imágen del mapa de disparidad en unidades de píxeles, K es una constante de normalización en el sistema métrico. Por ejemplo, si $(u,v) = 2000$ y $K = 2000$, entonces es $Z = 1$ según la ecuación 3.4 y representa 1 metro de distancia por cada 2000 unidades del mapa de profundidad. Esta constante se obtiene en el proceso de calibración. En este caso f debe estar expresada en píxeles.

Los valores de c_x y c_y tienen unidades de píxeles y son los centros ópticos del lente estimados en el proceso de calibración.

3.4.1. Visión estéreo con luz estructurada

En el método de *visión estéreo con luz estructurada*, se sustituye el uso de un par o múltiples cámaras por una sola de ellas, y se agrega un proyector que ilumina la escena (ver figura 3.5), por lo que se le clasifica como una técnica de *visión estéreo activa*. El proyector genera una secuencia de patrones conocidos que iluminan la escena y los objetos que se encuentran en dicha escena deforman el patrón de manera que, es posible analizar dicha distorsión para estimar y extraer la profundidad a la que se encuentran los objetos.

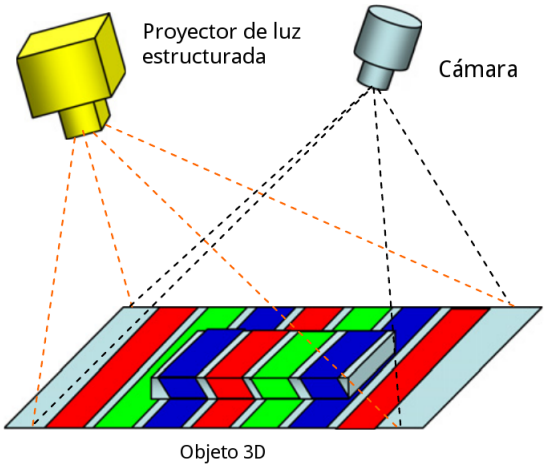


Figura 3.5: Ejemplo de un sistema de visión estéreo basado en luz estructurada.

Distintas técnicas para diseñar el patrón de un sensor de luz estructurada se revisan en [24] y [25].

El sensor Kinect utiliza una técnica refinada para crear el patrón de luz estructurada, usando un láser de espectro cercano al infrarrojo y un patrón obtenido a partir de una lente astigmática y un conjunto de difusores para separar y replicar el haz de luz (ver figura 3.6a). El patrón obtenido es semejante al ruido speckel (ver figura 3.6b). Este patrón es único y por consecuencia se almacena en la memoria no volátil del dispositivo durante su fabricación. La distancia a un objeto será aproximada mediante la disparidad del patrón y la misma región en una nueva captura de una imágen.

En [26] se analiza un modelo empírico del ruido asociado a la primera versión del sensor Kinect (versión 2010). Se concluye que la principales distorsiones son: los efectos de borde observados en las imágenes del mapa de disparidad y la incertidumbre en la distancia estimada en el eje Z, además la inclinación de los objetos capturados. El modelo

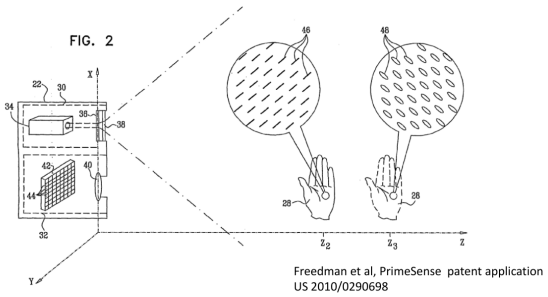
del ruido es el descrito en la ecuaciones 3.5 a la 3.7.

$$\sigma_L(\theta)_{pixels} = 0.8 + 0.035 \frac{\theta}{\frac{\pi}{2} - \theta} \tag{3.5}$$

$$\sigma_L(\theta)_{metros} = \sigma_L(\theta)_{pixels} \, Z \, \frac{p_x}{f_x} \tag{3.6}$$

$$\sigma_Z(Z,\theta) = 0.0012 + 0.0019\,(Z-0.4)^2 + 0.0001 \frac{\theta^2}{\sqrt{Z}\,(\frac{\pi}{2} - \theta)^2} \tag{3.7}$$

Con p_x el tamaño del píxel, f_x la distancia focal del sensor Kinect en eje X y θ el ángulo del objeto respecto el eje Y (ver figura 3.7). Se supone que la desviación estándar σ_L es perpendicular al eje Z. σ_Z es la desviación estándar de la distancia en el eje Z.



(a)

(b)

Figura 3.6: Estructura de la luz generada por el proyector del sensor Kinect. (a) Sistema de proyección. (b) Ejemplo del patrón proyectado en una escena. *Imágen (a) tomada de [27]. Imágen (b) tomada de [28].*

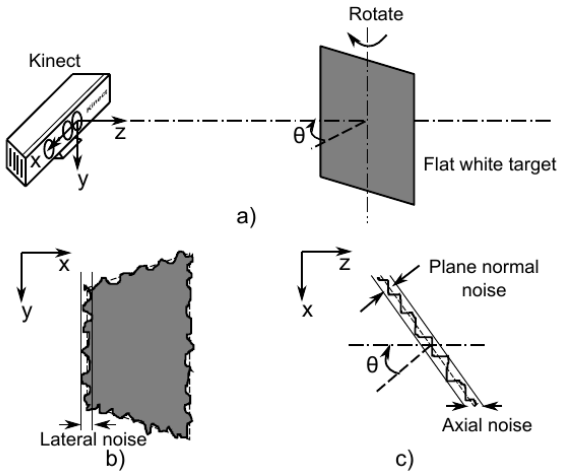


Figura 3.7: Esquema del modelo de ruido para el sensor Kinect. *Imágen tomada de [26].*

3.5. Filtrado espacial en imágenes digitales

En el procesamiento digital de imágenes, existen muchas técnicas para la mejora, transformación o representación de la información. Se llama *filtrado espacial* a las técnicas que se aplican en el dominio donde se representan las imágenes, es decir, el espacio matricial. Se adopta el término *filtrado* usado en técnicas que se aplican a señales en el dominio de la frecuencia en el procesamiento digital de señales.

Un filtro espacial se constituye por dos elementos básicos: el primero es un conjunto de puntos cercanos a un *punto central* que se llaman *vecindario*, éste es una matriz cuadrada o rectangular que contiene al punto central. El segundo elemento es definir una operación o función sobre el punto central y su vecindario, esta operación generará un nuevo punto con propiedades distintas, pero igual posición en la imágen con respecto a la original. Al aplicar la operación de manera individual a todos los puntos en la imágen, se obtiene una imágen de salida que se nombra *imágen filtrada*.

Si la operación de filtrado aplicada a la imágen cumple con la propiedad de superposición se dice que el filtro es *lineal* [29]. Si no, es *no lineal*. En particular los filtros espaciales lineales tienen una contraparte en el dominio de la frecuencia, por lo que pueden aplicarse según se requiera, en uno u otro dominio y obtener el mismo resultado.

3.5.1. Filtrado espacial lineal

El filtrado espacial lineal se obtiene mediante el reemplazo de todos los píxeles de la imágen $f(x,y)$ con alguna función lineal de los píxeles vecinos en el punto (x,y), esta función se asume independiente de dicha posición. Usualmente la operación de reemplazo se modela como una correlación o una convolución y se le asocia una matriz $w_{m\times n}$ con $m,n \in \{3,5,7,...\}$. El resultado es la suma de productos de los coeficientes de dicha matriz con una submatriz de la imágen. Esta operación esta definida en la ecuación 3.8.

$$g(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} f(x+i,y+j)\ w(i,j) \qquad a = \frac{m-1}{2}\ ,\ b = \frac{n-1}{2} \tag{3.8}$$

La submatriz de $f(x,y)$ y la matriz w se muestran a continuación:

$$\begin{bmatrix} f(x-a,y-b) & \dots & f(x,y-b) & \dots & f(x+a,y-b) \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ f(x-a,y) & \dots & f(x,y) & \dots & f(x+a,y) \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ f(x-a,y+b) & \dots & f(x,y+b) & \dots & f(x+a,y+b) \end{bmatrix} \quad , \quad \begin{bmatrix} w(-a,-b) & \dots & w(0,-b) & \dots & w(a,-b) \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ w(-a,0) & \dots & w(0,0) & \dots & w(a,0) \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ w(-a,b) & \dots & w(0,b) & \dots & w(a,b) \end{bmatrix}$$

El filtro se aplica coincidiendo el punto (x,y) de la imágen con el coeficiente $w(0,0)$ en el filtro de la siguiente manera:

$$\begin{bmatrix} f(x-a,y-b)w(-a,-b) & \dots & f(x,y-b)w(0,-b) & \dots & f(x+a,y-b)w(a,-b) \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ f(x-a,y)w(-a,0) & \dots & \boldsymbol{f(x,y)w(0,0)} & \dots & f(x+a,y)w(a,0) \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ f(x-a,y+b)w(-a,b) & \dots & f(x,y+b)w(0,b) & \dots & f(x+a,y+b)w(a,b) \end{bmatrix}$$

La imágen filtrada es el resultado de la repetición de la operación anterior para todos los puntos (x,y) en la imágen.

3.5.2. Filtrado espacial no lineal

El filtrado no lineal se refiere a la funcion que procesa los píxeles en la imágen y que no satisface el principio de superposición, principalmente se basan en estadísticos calcu-

lados en regiones locales de la imágen. Su importancia deriva en los siguientes hechos: los filtros espaciales lineales muestran un bajo rendimiento cuando el ruido no es aditivo o cuando este no se distribuye como una función Gaussiana, y en consecuencia no pueden obtenerse mediante la convolución o correlación; hay mayor complejidad del cálculo de los coeficientes pero aumenta su desempeño debido a su mayor selectividad para rechazar el ruido.

Algunos ejemplos de filtros espaciales no lineales son: el filtro de mediana muy utilizado por su simplicidad o el filtro bilateral, preservador de bordes; filtros basados en máximos y mínimos, entre muchas otras variantes de filtros no lineales [30].

3.5.2.1. Filtrado bilateral

El filtro bilateral se define como una media ponderada de los píxeles cercanos, de una manera muy similar a una convolución Gaussiana. La diferencia es que el filtro bilateral tiene en cuenta la diferencia de valor con los vecinos para suavizar la imágen a la vez que preserva los bordes. La idea clave del filtro bilateral es que, para que un píxel pueda influir en otro píxel, se debe ocupar un lugar cercano a este, pero también tener un valor de intensidad similar (ver figura 3.8). La formalización de esta idea se remonta en la literatura de Tomasi y Manduchi [31].

El filtro bilateral para una imágen $I(u)$ en la coordenada $u = (x, y)$ se define como:

$$\hat{I}(u) = \frac{\sum_{p \in N(u)} W_s(\| p - u \|) W_r(| I(u) - I(p) |) I(p)}{\sum_{p \in N(u)} W_s(\| p - u \|) W_r(| I(u) - I(p) |)} \tag{3.9}$$

La función de suavizado espacial W_s es un filtro Gaussiano $W_s(x) = e^{\frac{-x^2}{2\sigma_s^2}}$ con desviación estandar σ_s. La función de suavisado de intensidad se define como un filtro Gaussiano con la forma $W_r(x) = e^{\frac{-x^2}{2\sigma_r^2}}$ y desviación estandar σ_r. Además $N(u)$ es el vecindario del punto u definido como el conjunto:

$$N(u) = \{p_i \ : \| u - p_i \| < \ \rho = 2\sigma_s\} \tag{3.10}$$

La función de distancia para calcular la semejanza entre $I(p)$ e $I(u)$ usualmente es la distancia euclidiana, aunque dependerá del espacio de color y del desempeño del filtro.

El parámetro σ_s especifica la influencia de los píxeles p sobre el píxel central u, σ_r disminuye la influencia de los puntos cercanos cuando tienen intensidad diferente a u. La

combinación de ambos componentes se asegura de que sólo los píxeles similares contribuyen al resultado final (ver figura 3.9).

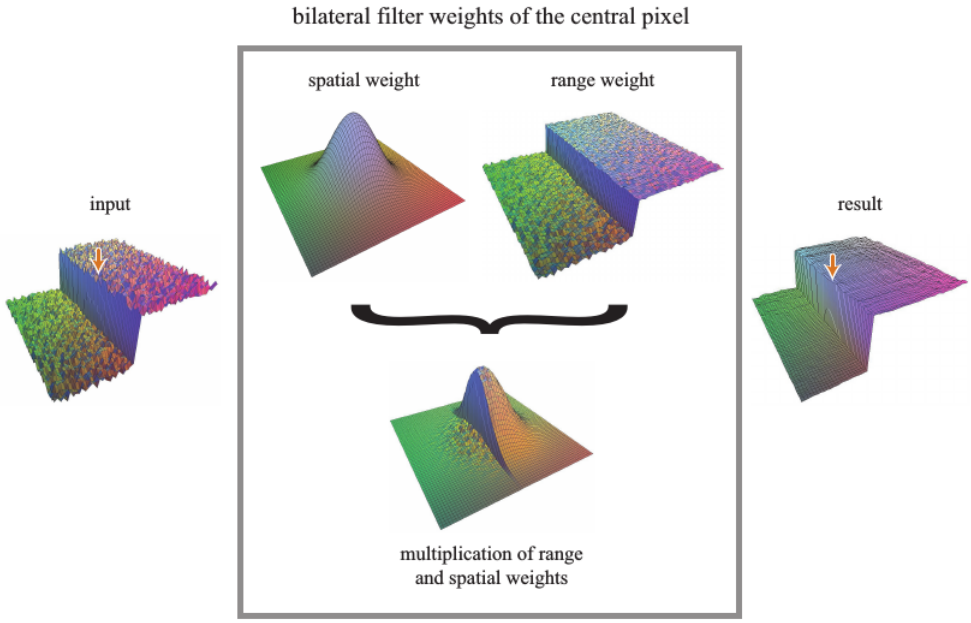


Figura 3.8: Estructura del filtro bilateral. Los pesos del filtro bilateral están representados para el píxel central (punto debajo de la flecha roja). *Imagen tomada de [32].*

En [33] se desarrolla como implementar el filtro bilateral usando una convolución tridimensional en el espacio $S \otimes R$ con S el subespacio en el dominio de la imágen y R el subespacio de intensidad. En [34] se abordan aproximaciones a la estructura original para implementar un filtro separable en dos dimensiones.

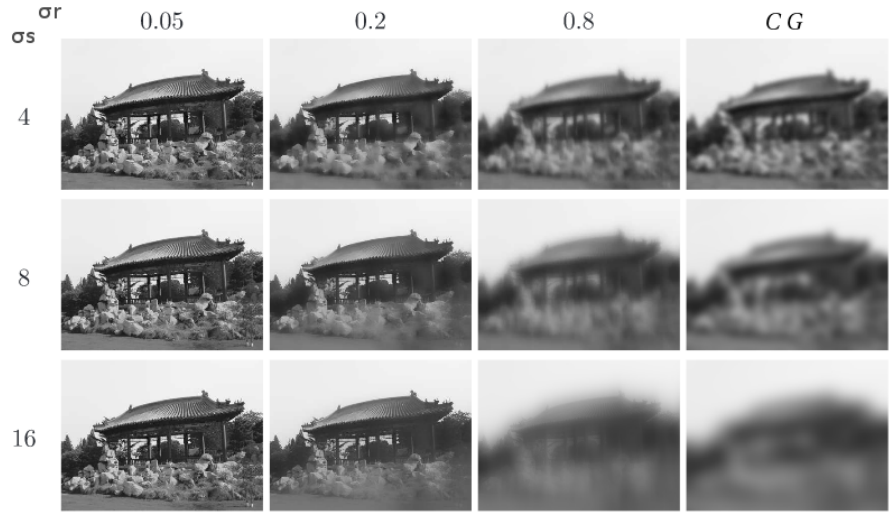


Figura 3.9: Ejemplo de salida del filtro bilateral. Los parámetros de intensidad σ_r y de distancia σ_s del filtro bilateral proporcionan un control más versátil de la convolución Gaussiana (CG). Por ejemplo, note los contornos de la azotea para valores pequeños y moderados de σ_r, y que la nitidez es independiente de los valores de σ_s. *Imágen tomada de [32].*

3.6. Extracción de características visuales

En una imágen digital, la mayoría de los píxeles individualmente no presentan datos que aporten información para identificar dicha imágen o identificar características visuales de interés, esto es debido a que los valores de intensidad o color observados resultan a menudo, muy semejantes o iguales entre sí. Por este problema es necesario el construir un método de detección de características usando conjuntos de píxeles, estos conjuntos deben ser únicos y por lo tanto describen a dicha imágen sin ambigüedad al ser comparados entre distintas imágenes.

Un detector automático de características visuales debe cumplir con los principales requerimientos listados a continuación:

- Las características extraídas deben ser robustas y estables. Por ejemplo: a cambios de iluminación, rotaciones, traslaciones, ruido, etc.
- Se debe extraer la mayor información posible de la imágen. Por ejemplo: color, geometría, textura, etc.
- La localización de dichas características debe ser lo más exacta posible.

- La búsqueda de características únicas debe ser eficiente; se debe evitar un costo excesivo de capacidad de computo.

Los requerimientos anteriores generalmente se implementan en dos etapas independientes. La primera etapa solo realiza la búsqueda de las características según la aplicación del sistema de visión, a este proceso se le conoce como *etapa de detección*, y los métodos usados para este proceso son conocidos como *detectores de puntos característicos*. La segunda etapa es la representación dichas características de manera que se tenga una representación única y lo más simple posible. A esta etapa se le nombra como *etapa de descripción*. Los métodos diseñados para esta etapa se conocen como *descriptores de puntos característicos*.

Algunos detectores comúnmente usados en imágenes son el detector de esquinas Harris y Shi-Tomasi, FAST (siglas del inglés *Features from Accelerated Segment Test: Características de prueba acelerada segmentada*), SURF (siglas del inglés *Speeded Up Robust Features: Características robustas aceleradas*) [35]. Y algunos descriptores populares son SIFT (siglas del inglés *Scale-Invariant Feature Transform: Transformación de caracteristicas invariantes a escala*), BRIEF (siglas del inglés *Binary Robust Independent Elementary Features: Características binarias elementales, independientes y robustas*), ORB (siglas del inglés *Oriented FAST and Rotated BRIEF*) [36].

Capítulo 4

Elementos de visión tridimensional

A lo largo de este capítulo de desarrollan los principales métodos utilizados en este trabajo de tesis para la percepción de infomación visual tridimensional. Se expone la herramienta de filtrado utilizada, los detectores y descriptores para nubes de puntos y la representación de color que se integra en la descripción de los datos de profundidad. Por último, se describe una herramienta de ordenamiento en nubes de puntos e imágenes llamada *orden en Z* que es muy utilizada por su eficiencia para generar estructuras de datos de almacenamiento en memoria.

4.1. Filtrado bilateral en nubes de puntos

La mejora continua en dispositivos de captura basados en láser ha hecho frecuente el uso de nubes de puntos en sistemas de visión para control de calidad en la industria, también en aplicaciones de construcción de modelos tridimensionales como se hace en la mecánica, ingeniería civil y diseño arquitectónico, y por supuesto en la robótica que fue la primera en explotar el potencial de estos dispositivos.

Debido a imperfecciones en las mediciones, atribuidas a causas externas al dispositivo como vibraciones o cambios en la reflexión del haz de luz, siempre se añade ruido a los datos recolectados, por lo que es necesario un módulo de preprocesamiento para reducir dicha distorsión. Las técnicas más habituales de reducción de ruido involucran una o varias etapas de filtrado antes de cualquier proceso de reconstrucción o representación de superficies. En [37] se adapta el algoritmo para mallas desarrollado en [38], para aplicarlo a nubes de puntos. Este proceso consta de dos etapas: la primera es filtrar las normales estimadas para los datos; la segunda es usar las normales suavizadas para filtrar la nube de puntos. El proceso de suavizado de normales se define en las ecuaciones 4.1 a la 4.3:

$$\mathbf{n_i} = \frac{\sum_{p_j \in N(p_i)} W_s(\| p_j - p_i \|) W_r(d_{ij})}{\sum_{p_j \in N(p_i)} W_s(\| p_j - p_i \|) W_r(d_{ij})} \hat{\mathbf{n}}_\mathbf{j} \tag{4.1}$$

con vecindario definido como:

$$N(p_i) = \left\{ p_j \ : \| p_j - p_i \| < \ \rho = 2\sigma_s \right\} \tag{4.2}$$

donde los puntos p_j tienen normal unitaria $\hat{\mathbf{n}}_\mathbf{j}$ y d_{ij} es la diferencia de intensidades entre dos vectores normales $\hat{\mathbf{n}}_\mathbf{i}$ y $\hat{\mathbf{n}}_\mathbf{j}$. La diferencia de intensidades se define como:

$$d_{ij} = \hat{\mathbf{n}}_\mathbf{i} \ \cdot \ (\hat{\mathbf{n}}_\mathbf{i} - \hat{\mathbf{n}}_\mathbf{j}) \tag{4.3}$$

Las normales obtenidas después del proceso de filtrado se usan para suavizar la nube de puntos, usando las ecuaciones 4.4 a la 4.6.

$$\Delta \mathbf{p_i} = \frac{\sum_{p_j \in N(p_i)} W_s(\| p_j - p_i \|) W_r(d_{ij})}{\sum_{p_j \in N(p_i)} W_s(\| p_j - p_i \|) W_r(d_{ij})} \mathbf{x_j} \tag{4.4}$$

$$\mathbf{x_j} = (((\mathbf{n_i} \cdot \mathbf{n_j})\mathbf{n_j}) \cdot \mathbf{n_i})\mathbf{n_i}) \tag{4.5}$$

$$\mathbf{p_i} = \mathbf{p_i} + \Delta \mathbf{p_i} \tag{4.6}$$

Donde el vector $\mathbf{p_i}$ es el vector de posición del punto p_i. El vector $\mathbf{x_j}$ tiene magnitud proporcional al desplazamiento hecho por la proyección en el nuevo plano. Finalmente los puntos originales se desplazan en dirección $\Delta \mathbf{p_i}$ para obtener la superficie con reducción de ruido.

Existen muchas variaciones del filtrado bilateral para mallas triangulares y nubes de puntos, una de ellas se muestra en [39] muy semejante a la presentada en la ecuación 4.4, además se expone como este tipo de filtro tiene mejor desempeño frente a filtros de media, mediana y MLS (siglas en inglés para *Moving Least Squares*: *minimos cuadrados móviles*; una técnica de reconstrucción local en nubes de puntos). Igualmente en [37] se muestran resultados que confirman la efectividad del filtrado bilateral para reducción de ruido en nubes de puntos.

4.2. Descriptor de puntos característicos ISS

El detector y descriptor ISS (siglas del inglés *Intrinsic Shape Signature*: *firma de forma intrínseca*) es un método para caracterizar regiones locales en una nube de puntos. Zhong define en [40] una firma de forma intrínseca $S_i = \{F_i, f_i\}$, ésta consiste en un marco de referencia intrínseco F_i en una región determinada en una nube de puntos, y un vector de características de forma f_i. Esto permite la extracción de información invariante al ángulo de visión de manera rápida y altamente discriminatoria, lo que codifica la forma de un objeto.

Primeramente se define un marco de referencia intrínseco F_i en un punto de origen p_i, en el cual se construye una esfera de soporte con radio r_{frame}, usando análisis de valores propios en la matriz de dispersión C del punto como sigue:

1. Se cálcula el peso w_i para cada punto p_i inversamente relacionado al número de puntos en su vecindario esférico, con radio $r_{density}$:

$$w_i = \frac{1}{|\{p_j : | p_j - p_i | < r_{density}\}|} \tag{4.7}$$

   Este peso es usado para compensar el muestreo no uniforme de puntos, por lo que los puntos dispersos contribuyen más que las regiones densamente muestreadas. Esto aporta robustez a la variación de la densidad de puntos que depende de la distancia del sensor de captura.

2. Se calcula la matriz de dispersión ponderada $\mathbf{C}(\mathbf{p_i})$ para $\mathbf{p_i}$ usando todos los puntos $\mathbf{p_j}$ dentro de la distancia r_{frame}:

$$C(p_i) = \frac{\sum_{|p_j - p_i| < r_{frame}} w_j (\mathbf{p_j} - \mathbf{p_i})(\mathbf{p_j} - \mathbf{p}_i)^T}{\sum_{|p_j - p_i| < r_{frame}} w_j} \tag{4.8}$$

3. Se calculan los valores propios $\{\lambda_i^1, \lambda_i^2, \lambda_i^3\}$ y se ordenan de manera decreciente junto con sus vectores propios $\mathbf{e_i^1}, \mathbf{e_i^2}, \mathbf{e_i^3}$.

4. Usando $\mathbf{p_i}$ como origen, y $\mathbf{e_i^1}$, $\mathbf{e_i^2}$, y su producto cruz $\mathbf{e_i^1} \otimes \mathbf{e_i^2}$ como los ejes x', y' y z' respectivamente, se define un sistema de coordenadas en tres dimensiones $F_i = \{p_i, \{\mathbf{e_i^1}, \mathbf{e_i^2}, \mathbf{e_i^3}\}\}$, el cual se llamará *sistema de referencia intrínseco*. Existe ambigüedad en el sistema de referencia intrínseco para distinguir entre *adentro* y *afuera* de la superficie (ver figura 4.1), por lo que puede forzarse a uno de los vectores de dicho sistema de

referencia para tener un producto punto positivo con respecto al vector del sistema de referencia del sensor, apuntando al frente del mismo, esto orienta la superficie local y reduce las variantes de la firma intrínseca de cuatro solo dos casos.

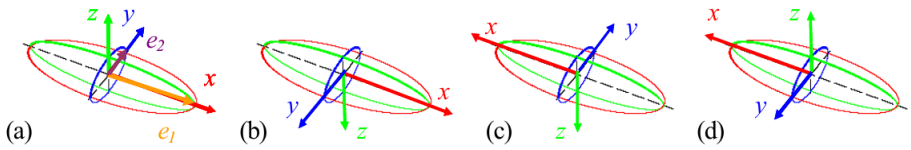


Figura 4.1: Sistema de referencia local ISS. (a) a (d) muestran las posibles variaciones obtenidas durante la descomposición en valores propios. *Imágen tomada de [40].*

Para seleccionar los puntos candidatos para crear su vector de características de forma, se usan dos umbrales: el primer umbral γ_{12}, es para comparar la razón entre el primer y segundo valor propio; el segundo umbral γ_{23}, para comparar la razón entre el segundo y tercer valor propio. Los puntos que cumplan con ambas condiciones son usados como puntos distintivos del objeto y posteriormente se construirá el descriptor de características de forma y con ello, la firma de forma intrínseca.

En la segunda parte de construcción del descriptor, se usa una rejilla esférica con K celdas estimada recursivamente como se describe en [41] (ver figura 4.2). Los pesos calculados en la ecuación 4.7 se suman para cada punto que este dentro de cada celda, esta suma será el descriptor de características de forma $f_i = (f_{i0}, f_{i1}, ..., f_{iK-1})$.

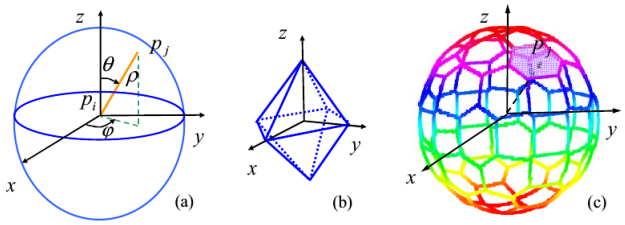


Figura 4.2: Estructura del sistema de referencia para ISS. (a) Sistema de referencia local para el punto p_i. (b) Primera iteración para la construcción de la rejilla. (c) Rejilla de soporte final. *Imágen tomada de [40].*

Por último, la firma de forma intrínseca será la concatenación del sistema de referencia local con el descriptor de forma expresado como:

$$S_i = \{p_i, \{\mathbf{e_i^1}, \mathbf{e_i^2}, \mathbf{e_i^3}\}, \{f_{i0}, f_{i1}, ..., f_{iK-1}\}\}$$

Cabe resaltar que este detector y descriptor es uno de los más robustos para la detección de puntos característicos junto con SIFT3D debido a su alta repetitividad y robustez a rotaciones según se muestra en [42].

4.3. Descriptor de puntos característicos SHOT

El descriptor SHOT (siglas del inglés *Signature of Histograms of Orientations*: *firma de histogramas de orientaciones*) es un descriptor de puntos característicos propuesto y probado por Tombari y Salti en [43]. Este descriptor está basado en un sistema de referencia local construido en una vecindad esférica de radio R, tomando como centro el punto característico $\mathbf{p}$ (ver figura 4.3). El soporte esférico se requiere para construir un histograma respecto al sistema de coordenadas local haciendo el descriptor robusto a escala y para aumentar su repetitividad en presencia de ruido.

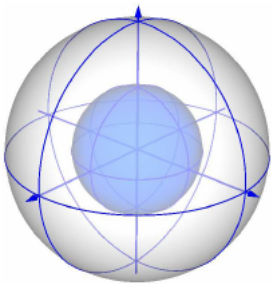


Figura 4.3: Rejilla espacial del descriptor SHOT para la construcción de la firma de histogramas (Solo se muestran 4 divisiones del azimuth). *Imagen tomada de [43].*

Los vectores del sistema local de referencia se obtienen por la descomposición en valores propios o en valores singulares de la matriz M, expresada en la ecuación 4.9. Se pondera con mayor influencia a los puntos cercanos al punto característico.

$$M = \frac{1}{\sum_{i:d_i \leq R} (R - d_i)} \sum_{i:d_i \leq R} (R - d_i)\ (\mathbf{p}_i - \mathbf{p})\ (\mathbf{p}_i - \mathbf{p})^T \tag{4.9}$$

$$\text{con } d_i = \| \mathbf{p_i} - \mathbf{p} \|$$

Los vectores propios calculados proveen un sistema de referencia local pero pueden existir ambigüedades en los signos de sus valores propios asociados. En [44] se aborda el problema. Básicamente se propone reorientar cada vector singular o vector propio to-

mando en cuenta el signo de la mayoría de los vectores en un vecindario.

Los vectores propios unitarios ordenados decrecientemente según sus valores propios, se referirán como $\mathbf{x}^+$, $\mathbf{y}^+$ y $\mathbf{z}^+$ respectivamente con $\mathbf{x}^-$, $\mathbf{y}^-$ y $\mathbf{z}^-$ para las direcciones opuestas. Sea $M(k)$ el subconjunto con k puntos dentro de la esfera de soporte, que tienen distancia más cercana a la distancia media del punto característico d_m, es decir:

$$M(k) \doteq \{i \ : \mid m - i \mid \leq k, \quad m = arg(media(d_j))\} \tag{4.10}$$

La corrección se define para el vector $\mathbf{x}$ como:

$$S_x^+ \doteq \{i : d_i \leq R \ \wedge \ (\mathbf{p_i} - \mathbf{p}) \ \cdot \mathbf{x}^+ \geq 0\} \tag{4.11}$$

$$S_x^- \doteq \{i : d_i \leq R \ \wedge \ (\mathbf{p_i} - \mathbf{p}) \ \cdot \mathbf{x}^- > 0\} \tag{4.12}$$

$$\tilde{S}_x^+ \doteq \{i : i \in M(k) \ \wedge \ (\mathbf{p_i} - \mathbf{p}) \ \cdot \mathbf{x}^+ \geq 0\} \tag{4.13}$$

$$\tilde{S}_x^- \doteq \{i : i \in M(k) \ \wedge \ (\mathbf{p_i} - \mathbf{p}) \ \cdot \mathbf{x}^- > 0\} \tag{4.14}$$

$$\mathbf{x} = \begin{cases} \mathbf{x}^+, & \mid S_x^+ \mid > \mid S_x^- \mid \\ \mathbf{x}^-, & \mid S_x^+ \mid < \mid S_x^- \mid \\ \mathbf{x}^+, & \mid S_x^+ \mid = \mid S_x^- \mid \ \wedge \ \mid S_x^+ \mid > \mid \tilde{S}_x^- \mid \\ \mathbf{x}^-, & \mid S_x^+ \mid = \mid S_x^- \mid \ \wedge \ \mid S_x^+ \mid < \mid \tilde{S}_x^- \mid \end{cases} \tag{4.15}$$

Para eliminar ambigüedades en la descomposición en vectores propios, cuando los puntos tienen signo $\mid S_x^+ \mid = \mid S_x^- \mid$, se considera un número impar de k puntos en $M(k)$ generando los subconjuntos $\mid \tilde{S}_x^+ \mid$ y $\mid \tilde{S}_x^- \mid$ y reorientando a ese signo. El mismo procedimiento se aplica a los vectores $\mathbf{z}$ y posteriormente se obtiene $\mathbf{y}$ con el producto cruz $\mathbf{z} \times \mathbf{x}$.

Para construir los histogramas locales, se calcula el coseno del ángulo θ_q, entre la normal de cada punto $\mathbf{n_q}$ y el vector $\mathbf{z_k}$ de la forma $cos(\theta_q) = \mathbf{z_k} \cdot \mathbf{n_q}$. Después se hacen histogramas para la rejilla espacial que se muestra en la figura 4.2. Experimentalmente se concluye el uso de 32 secciones: 8 para el azimuth (sectores de 45 grados), 2 para la distancia radial y 2 para la elevación (el ecuador divide las regiones). Adicionalmente cada histograma tiene 11 clases y se acumulan los valores $(1 - d_i) + (1 - d_j) + (1 - d_k) + (1 - d_l)$ que se obtienen por cuadri interpolación; d_i para el $cos(\theta_q)$, d_j para el azimuth, d_k para la elevación

y d_l para la distancia radial.

Por último se normaliza el descriptor a magnitud unitaria.

El descriptor SHOT forma un vector de tamaño $32 \times 11 = 352$ dimensiones. En la implementación se almacena también una matriz de tamaño 3×3 con los vectores propios $\mathbf{x_k}, \mathbf{y_k}, \mathbf{z_k}$.

4.4. Espacio de color CIE lab

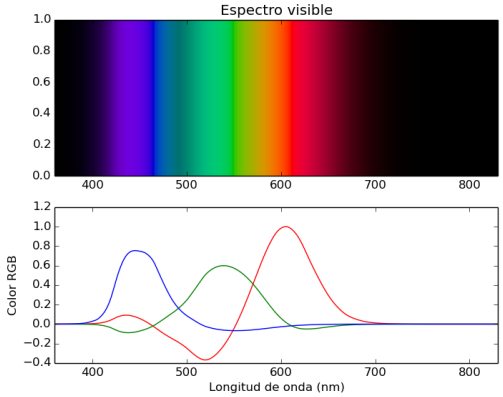
Un método para representar el color de un objeto usando algún tipo de representación númerica es comúnmente conocido como *espacio de color*. La Comisión Internacional de Iluminación (CIE, del francés *Commission Internationale de lÉclairage*), una organización sin fines de lucro que ha definido distintos espacios de color, entre ellos el *CIE XYZ*, *CIE L*C*h*, y *CIE L*a*b**, para estandarizar la representación del color en dispositivos de despliegue de manera objetiva.

El espacio de color CIE L*a*b*, también referido como *CIE Lab*, es actualmente una de las representaciones de color utilizadas para evaluar la representación del color. Este espacio de color es usado por la relación entre sus valores de color con la percepción visual en los humanos. Académicos y fabricantes lo aplican continuamente para identificar fallas o expresar precisamente sus resultados en el espacio Lab o en otros espacios comparados con este.

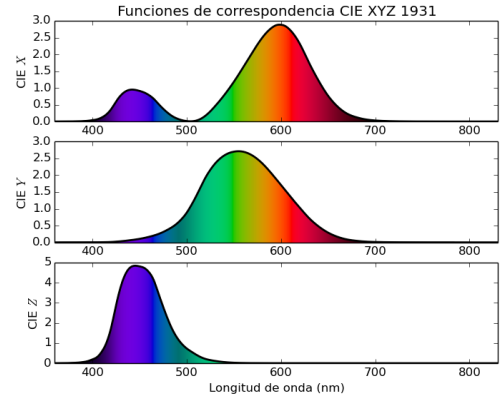
El espacio de color CIE Lab codifica la respuesta triestímulo promedio del ojo humano a la luz con frecuencias de onda altas (colores azules), frecuencias medias (colores verdes) y frecuencias bajas (colores rojos) (ver figura 4.4). Por lo que un sistema de despliegue de colores que utilice esta representación será invariante entre dispositivos y en promedio distintas personas observaran la misma representación del color.

El mapa de cromaticidad construido a partir de la aproximación de la respuesta triestímulo se muestra en la figura 4.5, este espacio de color se conoce como CIE XYZ y se basa en tres colores primarios falsos X, Y, Z basados en los espectros aproximados visibles al ojo humano, posteriormente se convierte a CIE Lab mediante las ecuaciones 4.16 a la 4.20.

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.755160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{4.16}$$



(a)



(b)

Figura 4.4: Aproximación triestímulo CIE 1931. (a) Respuesta promedio triestímulo del ojo humano. (b) Aproxiamción CIE a la respuesta del ojo humano.

$$X \leftarrow \frac{X}{X_n}, \text{ con } X_n = 0.950456$$

$$Z \leftarrow \frac{Z}{Z_n}, \text{ con } Z_n = 1.088754$$

$$L = \begin{cases} 116\ Y^{\frac{1}{3}} - 16\ , \text{ para } Y > \ 0.008856 \\ 903.3\ Y\ , \text{ para } Y \leq \ 0.008856 \end{cases} \tag{4.17}$$

$$a = 500\ [f(X) - f(Y)]\ +\ \Delta \tag{4.18}$$

$$b = 200\ [f(Y) - f(Z)]\ +\ \Delta \tag{4.19}$$

$$f(t) = \begin{cases} t^{\frac{1}{3}}\ ,\ \text{para}\ t\ >\ 0.008856 \\ 7.787t + \frac{16}{116}\ ,\ \text{para}\ t\ \leq\ 0.008856 \end{cases} \tag{4.20}$$

$$\Delta = \begin{cases} 128\ ,\ \text{para imágenes de 8 bits} \\ 0\ ,\ \text{para imágenes en punto flotante} \end{cases}$$

$$0\ \leq\ L\ \leq\ 100\ ,\quad -127\ \leq\ a\ \leq\ 127\ ,\quad -127\ \leq\ b\ \leq\ 127$$

En caso de que la imágen de salida se desee en representación de 8 bits, se normalizan los valores de salida L, a, b de la forma:

$$L \leftarrow L\ \frac{255}{100}$$
$$a \leftarrow a\ +\ 128$$
$$b \leftarrow b\ +\ 128$$

Para imágenes representadas en punto flotante los valores se dejan sin cambios. El espacio de color obtenido tiene una forma de esfera con origen en el punto $(0, 0, 0)$ donde el azimuth codifica el valor de la cromaticidad, y la latitud la luminosidad como se muestra en la figura 4.6.

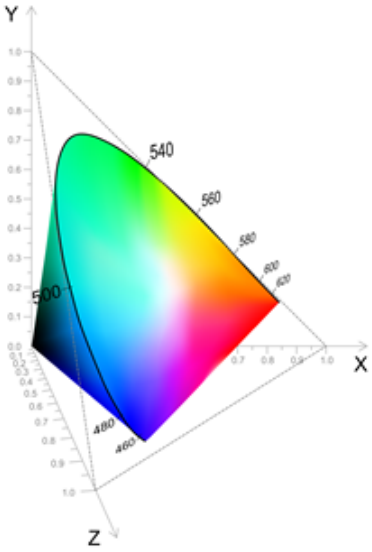


Figura 4.5: Espacio de color CIE XYZ.

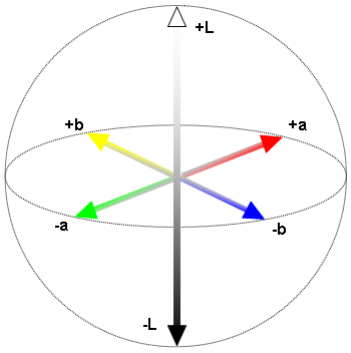


Figura 4.6: Espacio de color CIE Lab.

4.5. Descriptor de puntos característicos Color SHOT

El descriptor de forma y textura Color SHOT o brevemente CSHOT es la extensión natural de agregar la información RGB que usualmente acompaña a una nube de puntos, la imágen de color es tomada en el mismo instante que se captura la nube de puntos por el sensor Kinect, en una correspondencia $1:1$ en tiempo. Para añadir robustez se mapea el color RGB a un dominio absoluto llamado CIE Lab como se describió en la sección 4.4, en el cual la información de color es independiente del dispositivo de captura o de despliegue, generando una descripción de mayor calidad.

Igual que el descriptor SHOT, se construyen histogramas de color CIE Lab sobre la esfera de soporte añadiendo un paso al algoritmo original de cómputo mediante la operación:

$$\gamma = \| \ Lab_{\mathbf{q}} - Lab_{\mathbf{p}} \|_1 \tag{4.21}$$

donde $\mathbf{q}$ es el color CIE Lab de un punto dentro de la esfera de soporte y p el punto característico que se desea describir.

En [45] Alexandre expone distintas métricas para comparar distancias, en el caso del espacio de color CIE Lab, Tombari y Salti en [46] descubren empíricamente que la norma L_1 es más efectiva que la distancia ΔE_{00} propuesta en la revisión *CIE lab2000* [47]. Este descriptor utiliza la misma rejilla espacial de 32 secciones para organizar los histogramas de color como en el descriptor SHOT, pero los histogramas ahora cuentan con 31 clases normalizadas. Por último se concatenan los histogramas SHOT y los CIE Lab como se muestra en la figura 4.7.

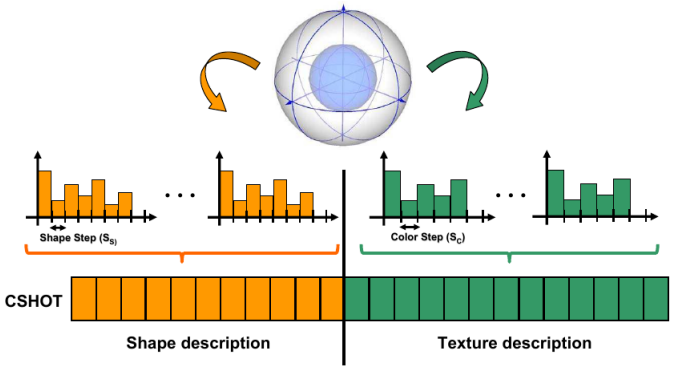
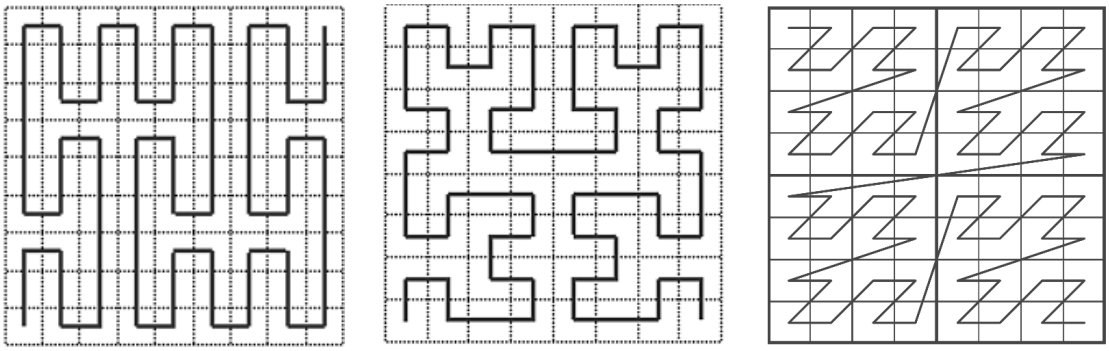


Figura 4.7: Ensamble del descriptor Color SHOT. *Imagen tomada [46].*

Al final se obtiene un descriptor de longitud: (32 regiones × 11 clases de forma) + (32 regiones × 31 clases de color) = 1344 dimensiones, mas la matriz de 3×3 del sistema de referencia. Este descriptor incluye la información de textura y color, además de ser robusto a ruido, invariante a escala y rotaciones.

4.6. Curvas de llenado de espacio

Las *curvas de llenado de espacio*, de dimensión n son curvas paramétricas continuas en intervalos cerrados en $\mathbb{R}^n$ definidas dentro de un *soporte* n dimensional (por ejemplo, el soporte puede ser el conjunto de puntos que delimitan una malla cuadrada en dimensión dos o los puntos que delimitan secciones cúbicas en dimensión tres, etc). En dichos soportes se hacen sucesivas divisiones que la curva irá uniendo, una a una en un orden específico, de modo que cada subdivisión y su vecindario son también cercanas en la curva. A cada proceso de división se le llama *orden* p. Al acumular iteraciones la curva irá recorriendo celdas cada vez más pequeñas del soporte original, hasta que en el límite se obtiene una curva que recorre todos los puntos en el soporte [48]. Ejemplos de estas curvas son la curva de Peano, de Hilbert y de Morton (*orden Z*) mostradas respectivamente en las figuras 4.8a, 4.8b y 4.8c.



(a) Ejemplo de curva de Peano. (b) Ejemplo de curva de Hilbert. (c) Ejemplo de curva de Morton.

Figura 4.8: Ejemplo de curvas de llenado de espacio. (a) Curva de Hilbert con p=3. (b) Curva de Peano con p=3. (c) Curva de Morton con p=3.

Estas curvas se obtienen iterativamente, por lo que al aumentar las veces que se itera sobre los puntos del soporte, tienden a llenar el espacio delimitado por el mismo (ver figura 4.9). Su principal característica es que asignan un lugar único en la curva a cada región en el soporte de manera ordenada y creciente, también preservan el orden local de los puntos por lo que son ideales para construir estructuras de datos de acceso a memoria

organizando los datos en bloques.

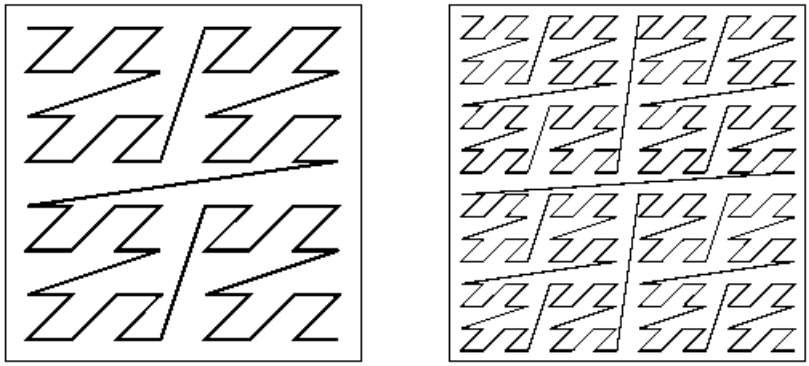


Figura 4.9: Ejemplo de llenado de espacio para una curva de Morton. A la izquierda p=3 y a la derecha p=4.

4.6.1. Orden Z

La curva de Morton es llamada *orden Z* por su parecido con la letra del alfabeto. La secuencia ordenada respecto a la curva se obtiene definiendo la cantidad de bits para representar el número de celdas como coordenadas horizontales y verticales. Después para cada punto que se desee ordenar, se calcula un entrelazamiento de sus coordenadas horizontal y vertical a nivel de bits.

Sea un punto $p = (x, y)$ en dos dimensiones y sus coordenadas en representación en bits $p_{bin}(\{x_n, ..., x_2, x_1\}, \{y_n, ..., y_2, y_1\})$ siendo valores enteros con n bits, su orden Z se obtiene con la operación:

$$\text{orden } Z(p) = \{y_n, x_n, ..., y_2, x_2, y_1, x_1\}$$

La conversión de dicha cadena de bits a un numero de base 10 será el número de celda donde se encuentra el punto. Por ejemplo, para el punto $p = (x, y) = (5, 9)$ y una representación binaria de 4 bits, $p_{bin} = (\mathbf{0101}, 1001)$ su orden Z resulta ser $ordenZ(p) = 1\mathbf{00}1\mathbf{0}0\mathbf{11}_{bin} = 147_{decimal}$. Este procedimiento es válido para cualquier cantidad de dimensiones en los datos respetando siempre que el entrelazado comienza con la primera dimension.

Una aplicación muy popular debido a la eficiencia del procedimiento es usar el orden Z para construir estructuras de datos de búsqueda llamadas árboles cuádruples (del inglés *quadtree*) en las que se construye un árbol de búsqueda donde cada nodo tiene cuatro nodos hijos. Y la jerarquía en el árbol respeta la jerarquía de las celdas del ordenamiento

en Z como se muestra en la figura 4.10.

Existen variaciones del algoritmo de ordenamiento en Z para valores reales [49] aunque ya no es posible calcular los índices mediante operaciones binarias (desplazamiento de bits, operaciones *or* y *and*).

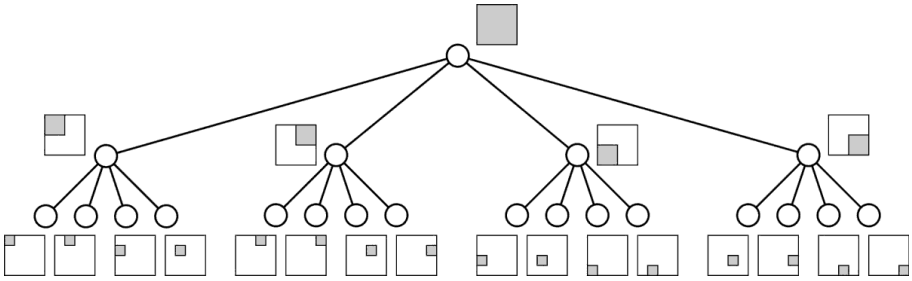


Figura 4.10: Ejemplo de árbol cuadruple construido con el ordenamiento en Z.

Capítulo 5

Técnicas de cuantización vectorial y agrupamiento de vectores

La *cuantización vectorial* es una técnica en la cual, un vector $\mathbf{x}$ con N dimensiones y representado con una tupla de d valores reales, es mapeado a otro vector con dimensión $N' \leq N$ y representado por una tupla $d' \leq d$ con valores reales o discretos. Por lo regular, el mapeo se representa como una transformación $T\{\}$ que al aplicarse al vector $\mathbf{x}$ lo proyecta en subespacio finito $\mathbf{Y}$ con cardinalidad L, en el que se le asocia un vector $\mathbf{y} = T\{\mathbf{x}\}$. Típicamente se le nombra al conjunto de vectores $\mathbf{y_k} \in \mathbf{Y}$ como *libro de códigos*, a $\mathbf{y_k}$ como *vector código* y L el *tamaño del libro* de códigos.

Si la transformación $T\{\}$ se diseña como L particiones con centroides $\mathbf{y_k}$ del espacio N dimensional donde se encuentran los vectores $\mathbf{x}$, se pueden observar particiones del espacio llamadas *celdas de Voronoi* como se muestra en la figura 5.1.

La cuantización vectorial fue estudiada por Linde y Gray en la década de 1980 como una herramienta de compresión de señales de voz, con el objetivo de disminuir el ancho de banda usado en la transmisión de estas señales. También como una herramienta para compresión de datos por Gersho y Gray en la década de 1990.

La mayoría de los algoritmos de cuantización vectorial pertenecen a una clase llamada *algoritmos de agrupamiento*. En estos se agrupa un conjunto de vectores en subconjuntos según una medida de similitud, por lo que se forman clases diferentes si hay distintas propiedades contenidas en los datos. Los vectores dentro de un mismo agrupamiento presentan algún tipo de similitud, correlación o cercanía y vectores en distintos grupos no tienen propiedades en común. Un ejemplo de aplicación de este tipo de métodos es el programa *AUTOCLASS* [50] desarrollado en la NASA, que usa razonamiento bayesiano para, dado un conjunto de datos de entrenamiento, sugerir un conjunto de clases probable. Este progra-

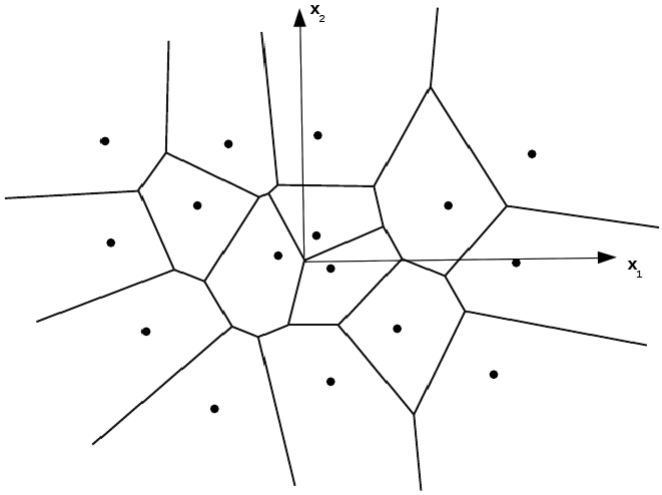


Figura 5.1: Un ejemplo de un espacio 2-dimensional particionado para $L = 16$.

ma encontró nuevas clases de estrellas a partir de sus datos del espectro infrarrojo.

El proceso de construcción de los grupos se hace de manera no supervisada por lo que un programa construye las clases por sí mismo basado en un conjunto de reglas de similitud, una de ellas puede ser la distancia entre vectores. Algunas de las métricas más habituales son la distancia euclidiana, la distancia euclidiana normalizada y la distancia de Mahalanobis. Una revisión de las técnicas y el estado del arte se desarrolla en [51], algunas de las principales son: de agrupamiento jerárquico, que se basan en agrupar los datos jerárquicamente en orden ascendente (usan la conectividad de los datos); de agrupamiento particional (basados en distancia) como son las K medias y redes neuronales auto organizables.

5.1. El método de las K medias

Uno de los métodos más utilizados para la agrupación de vectores es el llamado *K medias*. El objetivo es resolver el problema de encontrar k centroides que minimicen la distancia media entre los vectores de muestra y los centroides más cercanos a estos. Particularmente, dado N número de muestras de entrenamiento $X = \{x_1, ..., x_N\}$ en un subespacio V, se debe encontrar el conjunto de centroides $C = \{c_1, ..., c_k\}$ en V que minimice la función de error definida por:

$$E(C) = \frac{1}{N} \sum_{i=1}^{N} \| x_i - c_i \|^2 \tag{5.1}$$

Donde $c_i = \text{Arg Min}_{j \in \{1,..,k\}} \| x_i - c_j \|^2$. El problema de las K medias es un problema *no polinomial difícil* [52]) por lo que debe balancearse entre minimizar $E(C)$ y tener un bajo tiempo de ejecución.

El algoritmo más común para implementar las K medias es el *algoritmo de LLoyd* [53]. Este algoritmo está basado en que optimizar la generación de los grupos de manera conjunta es difícil, pero optimizar un grupo dado los demás es relativamente fácil. Requiere como valores de entrada los datos de entrenamiento X, la cantidad de centroides k, la cantidad de iteraciones n o un umbral del error ϵ como condiciones de paro. A continuación se muestra el pseudocódigo:

Algoritmo 1 Algoritmo K medias

Entrada: $X = \{x_1, ..., x_N\}$, $k > 0$, $n > 0$, $\epsilon << 1$
Salida: $C = \{c_1, ..., c_k\}$
1: $c_m \leftarrow x_m \in X$ donde $m \in \{1, .., N\}$ elegido aleatoriamente, $fin = 1$
2: $c_i = \text{Arg Min}_{j \in \{1,..,k\}} \| x_i - c_j \|^2$
3: $C = \{c_i\}$
4: $E(C) = \frac{1}{N} \sum_{i=1}^{N} \| x_i - c_i \|^2$
5: $E_{tmp} \leftarrow E(C)$
6: **mientras** $fin < n$ **ó** $E_{tmp} > \epsilon$ **hacer**
7: $c_i = \text{Arg Min}_{j \in \{1,..,k\}} \| x_i - c_j \|^2$
8: $C = \{c_i\}$
9: $E(C) = \frac{1}{N} \sum_{i=1}^{N} \| x_i - c_i \|^2$
10: $E_{tmp} \leftarrow E_{tmp} - E(C)$
11: $fin \leftarrow fin + 1$
12: **fin mientras**
13: **devolver** C

Las implementaciones simples del algoritmo de Lloyd tienen complejidad $O(Nkd)$ donde N es el número de puntos o vectores de dimensión d y k es la cantidad de centroides a calcular. En el algoritmo 1 en la linea 7, se observa que el cuello de botella en el tiempo de ejecución son las asignaciones de cada punto con su respectivo centroide.

Otra dificultad es la llamada *maldición de la dimensionalidad* (del inglés *curse of dimensionality*), un término que se le atribuye a Bellman en su libro de teoría de control del año 1961 [54]. En ese libro describe como resulta imposible optimizar funciones de muchas variables usando búsquedas por *fuerza bruta* en un espacio multidimensional discreto (una búsqueda por fuerza bruta es una *búsqueda combinatoria* o *búsqueda exhaustiva* sobre todos los elementos

de un conjunto). Se acostumbra usar el término maldición de la dimensionalidad para referirse a problemas de análisis de datos con muchas variables.

En particular, Bellman plantea el problema que dado un conjunto fijo de puntos, estos comienzan a *dispersarse* conforme la dimension del espacio donde se encuentran se incrementa. Un ejemplo es el siguiente: considerando 100 puntos con distribución uniforme en el intervalo $[0,1]$ (N=1). Si particionamos el espacio en 10 intervalos iguales, es altamente probable que todos los intervalos tengan cierto número de puntos distinto de cero. Ahora consideremos 100 puntos pero de dimensión igual a 2 (N=2) y construimos celdas de tamaño uniforme en el espacio delimitado por un cuadrado unitario, por lo que ahora tenemos 100 celdas bidimensionales en las que es posible encontrar los 100 puntos. Si continuamos con este proceso en tres dimensiones, obtendremos 1000 celdas en las que es posible encontrar los mismos 100 puntos de dimension tres (N=3). Eventualmente, si aumentamos la cantidad de coordenadas hasta tener un hipercubo unitario de dimensión $N \gg 1$ la probabilidad de encontrar los 100 puntos en dichas hiperceldas será cercana a cero. Conceptualmente esto significa que nuestros datos se han dispersado hasta casi desaparecer en un espacio de alta dimensionalidad.

En [55] se estudian las condiciones de dimensionalidad de los datos en las cuales tiene sentido hacer una búsqueda de los vecinos más cercanos, ahí se muestra como la distancia de un punto a su vecino más cercano y más lejano tiende a cero, para ciertas distribuciones de los datos y conforme aumenta la dimension del espacio. Esto es:

$$\lim_{d\to\infty} \frac{d_{max} - d_{min}}{d_{min}} = 0 \tag{5.2}$$

Más específicamente, en [56] se estudia la *distancia relativa* expresada como $d_{max} - d_{min}$ y se demuestra que para distintas normas (ver ecuación 5.3), la distancia cambia su comportamiento.

$$L_p = \| x - y \|_p = \sqrt[p]{\sum_{i=1}^{N} | x_i - y_i |^p} \quad ,\text{con} \quad d \geq 1 \tag{5.3}$$

- Para la norma L_1 la distancia relativa se incrementa según aumenta la dimension del espacio.
- Para norma L_2 la distancia relativa permanece aproximadamente constante y:
- Para la norma L_d para $d \geq 3$, esta tiende a cero conforme aumenta la dimensionalidad, lo que significa que los puntos en dichos espacios tienden a estar alejados una distancia igual a cero.

Los descrito anteriormente es posible visualizarlo mediante un experimento. Podemos generar 10000 puntos con dimensiones 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 y 2048. Con distribucion $\mathbf{N} \sim (0,1)$ y calcular la distancia mínima promedio y la distancia máxima promedio, ambas usando la distania euclidiana. El resultado se muestra en la figura 5.2. Con esto se confirma el comportamiento descrito anteriormente.

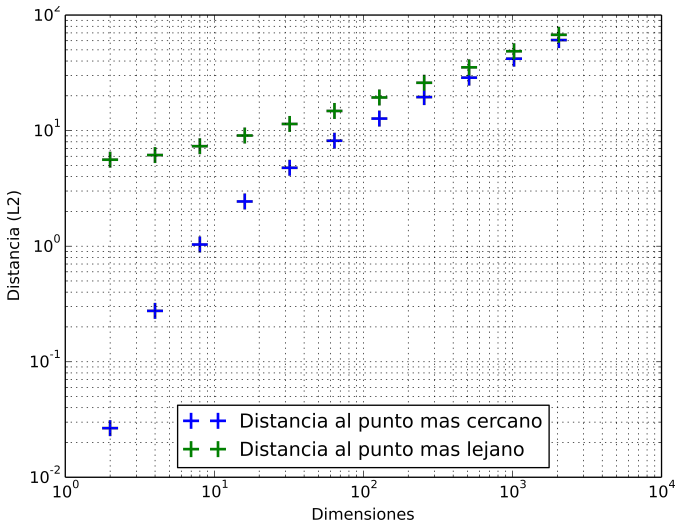


Figura 5.2: Gráfica de la distancia promedio máxima y mínima para 10000 puntos aleatorios comparados con norma L_2.

Para solucionar el problema de la medición de distancias en datos con alta dimensionalidad en [57] se propone usar una redefinición de la norma L_p con $0 < p < 1$ para mejorar el desempeño en algoritmos de agrupamiento y preservar el comportamiento de la distancia euclidiana. La norma se redefine como:

$$L_p = \| x - y \|_p = \sum_{i=1}^{N} | x_i - y_i |^p \tag{5.4}$$

5.1.1. Mejoras al método K medias

5.1.1.1. K medias++

Una de las mejoras que se han propuesto en el estado del arte del método K medias es el algoritmo denominado *K medias++* (del inglés *K-means++*) [58]. En este algoritmo se optimiza la inicialización del algoritmo de Lloyd mediante la selección de los centroides iniciales ponderados por una probabilidad definida como $P(x_i) = \frac{D(x_i)^2}{\sum_{x \in X} D(x)^2}$ donde $D(x_i) =$

Arg $\text{Min}_{c_j \in C} \| x_i - c_j \|_p$. El procedimiento se muestra en el algoritmo 2.

Algoritmo 2 Algoritmo de inicialización K medias++

Entrada: $X = \{x_1, ..., x_N\}$, $k > 0$, $C = \Phi$
Salida: $C = \{c_1, ..., c_k\}$

1: $c_1 \leftarrow x_m \in X$ donde $m \in \{1, .., N\}$ elegido aleatoriamente, $k_{tmp} = 1, C \leftarrow C \cup c_1$
2: **mientras** $k_{tmp} < k$ **hacer**
3: $\quad c_i = x_i \in X$ con la mayor probabilidad $P(x_i) = \frac{D(x_i)^2}{\sum_{x \in X} D(x)^2}$
4: $\quad C \leftarrow C \cup c_i$
5: $\quad k_{tmp} \leftarrow k_{tmp} + 1$
6: **fin mientras**
7: **devolver** C

Al final se ejecuta el algoritmo de Lloyd inicializado con el conjunto de centroides C calculados con el algoritmo 2. Este algoritmo maximiza la varianza entre los centroides escogiendo los puntos más lejanos para iniciar respecto de la distancia media a los centroides ya seleccionados.

5.1.1.2. Algoritmo de Elkan

Una optimización del algoritmo de Lloyd respecto su desempeño es el algoritmo de Elkan [59]. Este algoritmo preserva la *desigualdad de Minkowski* (comúnmente conocida como la desigualdad del triángulo), esto evita hacer comparaciones innecesarias en el proceso de asignaciones de cada punto a su centroide más cercano.

La idea principal se basa en que si un centroide, después de ser actualizado no varía mucho, la mayoría de los cálculos entre puntos y el centroide pueden ser evitados cuando haya una reasignación. Para saber cuales distancias necesitan ser evaluadas, se usa una desigualdad triangular para acotar el límite inferior y superior de las distancias antes de actualizar el mismo. Primero tenemos que:

$$\| x_i - c_{qi} \|_p \leq \frac{\| c - c_{qi} \|_p}{2} \Rightarrow \| x_i - c_{qi} \|_p \leq \| x_i - c \|_p \tag{5.5}$$

Donde c_{qi} es la distancia al centroide asignado al punto x_i, si es menor que la mitad de la distancia a cualquier otro centroide c puede ser evitado cuando se haga una nueva búsqueda para el punto x_i. Esto requiere tener una lista de todas distancias entre centroides. Si la ecuación 5.5 se cumple para todo $c \neq c_{qi}$ el punto x_i no necesita ser actualizado. Esta condición requiere un limite superior U_i.

Si un centroide c es actualizado a c', entonces la nueva distancia del punto x_i a c' está

acotada en su límite inferior L_i y su límite superior U_i como:

$$\| x_i - c \|_p - \| c - c' \|_p \;\leq\; \| x_i - c' \|_p \;\leq\; \| x_i - c' \|_p + \| c + c' \|_p \tag{5.6}$$

Este análisis permite mantener un límite superior entre la distancia de x_i y su centroide actual c_{qi} y un limite inferior entre cualquier otro centroide mediante las reasignaciones:

$$U_i \leftarrow U_i + \| c_{qi} - c'_{qi} \|_p$$

$$L_i \leftarrow L_i - \| c - c' \|_p$$

Y el límite superior e inferior se definen como:

$$U_i = \text{Min}\, L_i(c) \tag{5.7}$$

$$L_i(c) = \| x_i - c' \|_p \tag{5.8}$$

En [60] se estudia como el algoritmo de Elkan resulta la mejor elección para cuantizar datos con alta dimensionalidad para casos $N > 32$. Existen algoritmos más veloces para un número de dimensiones $N \leq 32$. En [61] se hace un estudio de diversos algoritmos y bibliotecas para resolver el problemas de las K medias, se concluye que para $N > 50$ el algoritmo más eficiente es el algoritmo de Elkan y variaciones de este. La principal desventaja de este algoritmo es que requiere de mucha memoria de acceso aleatorio disponible, del orden de Gigabytes dependiendo del conjunto de datos a analizar.

5.1.1.3. Arboles KD aleatorios

Un *árbol KD* (del inglés *K-Dimensional tree*) es una estructura de datos, en la cual se almacenan los datos manteniendo un orden espacial para agilizar búsquedas entre un punto y sus vecinos más cercanos. La letra K hace referencia a las dimensiones del espacio en que se encuentran los datos, así un árbol 2D es un árbol binario para datos bidimensionales y un árbol 3D es un árbol binario para datos en tres dimensiones [62].

El arbol KD es un árbol de decisión binario en el cual se agregan nodos según una regla de clasificación. Para lograrlo se hacen particiones del espacio usando planos en cada dimensión, desde la dimension 1 hasta la K, alternando entre ellas. El primer paso es buscar el punto que corresponde a la mediana de los datos a clasificar en la primera dimensión y se agrega como nodo raíz, esta mediana dividirá todos los datos en dos sub-

conjuntos, uno a la izquierda para los elementos menores a la mediana y uno a la derecha para los elementos mayores o iguales. Después se busca la mediana en la segunda dimensión para el subconjunto izquierdo y la mediana del subconjunto derecho. Por último se agrega a la izquierda del nodo padre la mediana del subconjunto a la izquierda y a su derecha, la mediana del subconjunto derecho. La siguiente iteración desciende en los nodos hijos y particiona el espacio en la tercera dimensión con el mismo procedimiento. Cuando se hayan particionado todas las dimensiones, se comienza con la primera de nuevo. El algoritmo se detiene al haber agregado todos los puntos (ver figura 5.3).

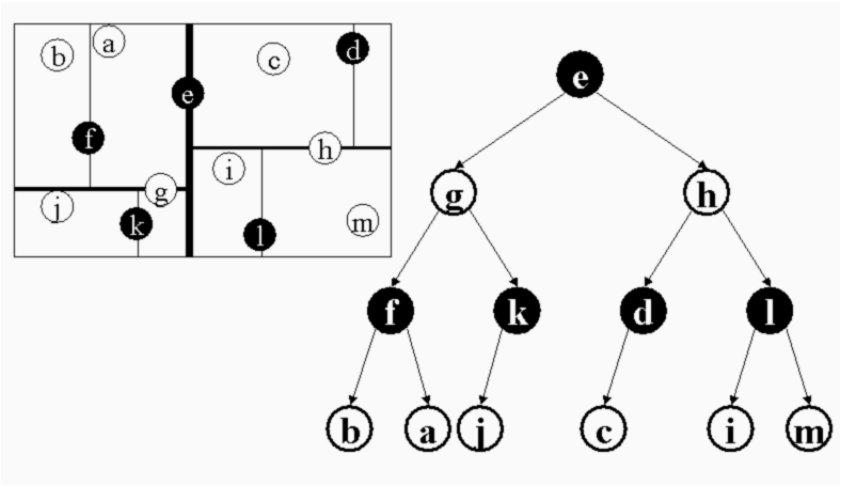


Figura 5.3: Ejemplo de árbol KD, para K=2 y D=2.

Un *árbol KD aleatorio* es una extensión de los arboles KD, con la diferencia de que es una variación que mejora el desempeño en un espacio de altas dimensiones. La mejora radica en particionar el espacio de trabajo N en D dimensiones con la mayor varianza. Después se construyen D árboles KD para cada dimensión seleccionada (ver figura 5.4). El proceso de búsqueda del vecino más cercano a un punto se hace en paralelo por cada árbol utilizando una estructura de datos denominada *cola de proridades* (del inglés *priority queue*) en las que se van agregando los candidatos a vecino más cercano, con una prioridad más alta si se tiene la distancia más cercana al punto (la distancia en un árbol se mide por la cantidad de nodos recorridos para llegar a donde esta el dato almacenado). Al final el mejor candidato será el que tenga la prioridad más alta en la cola ya que resulta ser el más cercano en todos los árboles. Este tipo de método es muy efectivo ya que realiza proyecciones sobre las dimensiones con mayor varianza y mediante un consenso se encuentra el valor del vecino más cercano al punto de prueba.

En [63] se muestra como para búsqueda de correspondencias de descriptores SIFT se logra una precisión del 95%, usando hasta 100 arboles KD aleatorios sobre las 128 dimensiones del descriptor; con 100,000 descriptores SIFT como conjunto de pruebas resulta 100 veces más rápido que con una búsqueda exhaustiva. Para un conjunto de datos generado aleatoriamente con dimensiones entre 1000 y hasta 4000, se logra 90% de precisión con un aumento de velocidad de más de 100 veces respecto de una búsqueda exhaustiva.

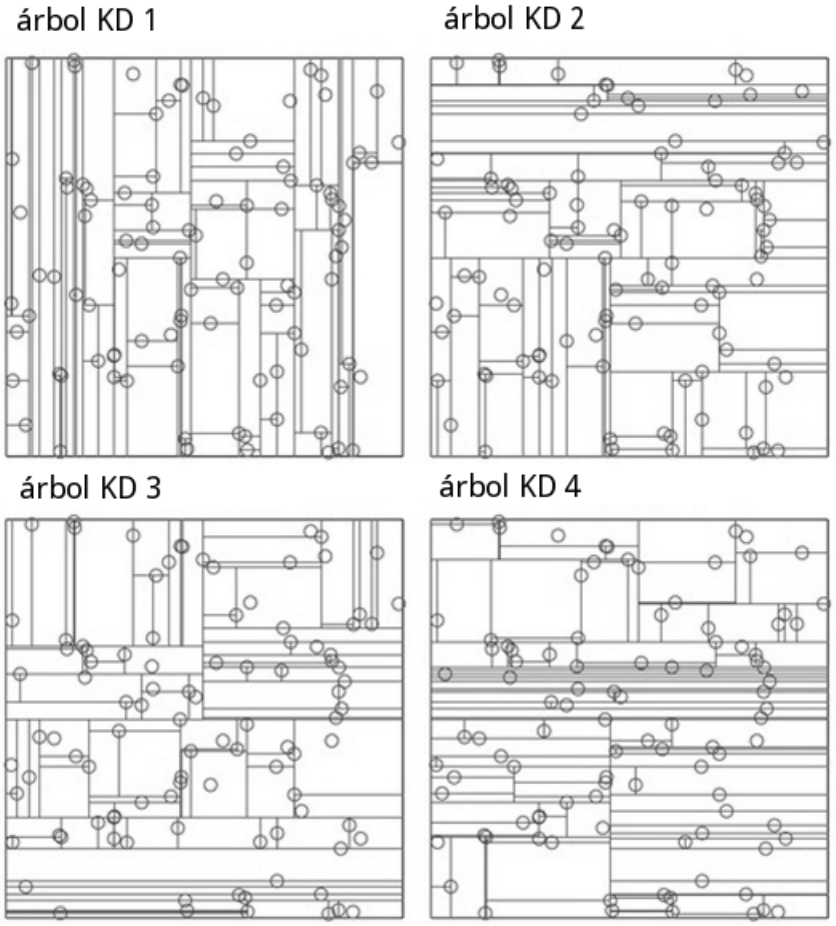


Figura 5.4: Ejemplo de árboles KD aleatorios para D=4 y K=10. Los datos han sido proyectados en un espacio bidimensional para su visualización.

Hasta este capítulo se presentaron todas las herramientas matemáticas para seleccionar y tener la mejor representación posible de los datos de color y forma capturados por el sensor Kinect para su posterior clasificación. Esto se logra mediante el siguiente proceso:

1. Capturar, rectificar y filtrar la información del sensor Kinect. En particular mediante el uso del filtrado bilateral en la nube de puntos.
2. Seleccionar puntos distintivos en la nube de puntos con el detector ISS.
3. Describir los puntos distintivos usando el descriptor Color SHOT.
4. Ordenar los puntos desde el punto de vista del sensor Kinect mediante el orden en Z, esto para conservar la relación espacial entre ellos.
5. Cuantizar dichos descriptores usando el algoritmo K medias con las mejoras K medias++ y el algoritmo de Elkan. El algoritmo de Elkan realiza las búsquedas usando arboles KD aleatorios en lugar de la norma redefinida en la ecuación 5.4.

Debido a que el método de clasificación propuesto son los modelos ocultos de Markov, y este utliza secuencias de símbolos como datos de entrada, los símbolos son obtenidos asociando los índices de los centroides calculados en la cuantización y un nuevo descriptor.

Los modelos ocultos de Markov se presentan en el siguiente capítulo.

Capítulo 6

Métodos de clasificación

En visión por computadora, existen diversos métodos para seleccionar y extraer información que represente de manera única las observaciones o datos capturados por los sensores. El objetivo de obtener esa información y representarla de manera compacta es para comparar entre distintas clases de observaciones y poder tomar decisiones en sistemas automáticos, para esta tarea es necesario diseñar módulos denominados *clasificadores* de características visuales.

En este capítulo se exponen dos principales métodos de clasificación de características visuales. El primero está basado en métodos estadísticos y transforma la información visual en secuencias de símbolos para darle un significado semántico a los datos, este método se conoce como clasificador de modelos ocultos de Markov. El segundo clasifica las características visuales suministradas en forma de vectores y las asocia a una clase predeterminada usando el concepto de neurona artificial, este método es conocido como un clasificador basado en redes neuronales artificiales.

6.1. Modelos ocultos de Markov

Los sistemas del mundo real producen salidas o datos que se pueden tratar como señales, estas pueden ser de naturaleza *discreta* (por ejemplo, las salidas del lanzamiento sucesivo de un dado) o de naturaleza *continua* (por ejemplo, las medidas de la corriente eléctrica en un conductor). Además pueden ser *estacionarias* o *no estacionarias*, según varíen o no sus propiedades estadísticas a través del tiempo. En ese sentido, existe el problema de crear modelos para describir, controlar y reconstruir los procesos generadores de dichas señales. Los modelos de señales se clasifican en dos categorías: *determinísticos* y *estocásticos*. Los modelos determinísticos explotan propiedades conocidas de las señales, mientras que

en los estocásticos se intenta modelar las propiedades estadísticas de la señal.

En el año de 1913, Andréi Andréyevich Márkov introduce un modelo estocástico para analizar la probabilidad de aparición de ciertas vocales en textos del autor Alexander Pushkin's [64], más tarde esta teoría sería desarrollada por Leonard E. Baum en la década de 1960 y usada en la década de 1970 para reconocimiento del habla. Esta formulación se conocería como *modelos ocultos de Markov*.

Un modelo oculto de Markov es un modelo de un *proceso estocástico* que se plantea como un proceso estocástico doble, en el cual las realizaciones de un primer proceso llamado *proceso oculto*, dan origen a un segundo proceso llamado *proceso observado*, los dos procesos estocásticos se logran caracterizar usando sólo el proceso que se puede observar (medir). El principal atributo de los modelos ocultos de Markov es que son modelos que asignan probabilidades de aparición a secuencias de símbolos, se dice que son *procesos generativos* ya que las probabilidades de ocurrencia para cada símbolo, se definen por una serie de pasos que incrementalmente producen la secuencia observada. Este tipo de proceso estocástico se le describe mediante un conjunto de unidades llamadas *estados* que son el mecanismo de emisión de los símbolos. Los estados tienen probabilidades de transición asociadas a los cambios internos del sistema durante la generación de los símbolos.

6.1.1. Procesos de Markov

Cuando una probabilidad condicional de un fenómeno depende únicamente del suceso inmediatamente anterior en el tiempo, cumple con el *principio de Markov de primer orden*, es decir:

$$P[X(t+1) = j \mid X(0) = k_0,\ X(1) = k_1,\ ...,\ X(t) = i] \ = \ P[X(t+1) = j \mid X(t) = i] \ = \ P_{ij} \tag{6.1}$$

Si un suceso depende de otro además del inmediatamente anterior, este es un *proceso de Markov de orden superior*. Por ejemplo, un proceso de segundo orden describe un proceso en el cual el suceso depende de los dos anteriores a él. A los procesos de Markov también se les llama *cadenas de Markov*, y son extensiones del concepto de *autómata finito* [65], con la diferencia de que las salidas del sistema se asocian a probabilidades en lugar de estados determinísticos.

Los Procesos de Markov de primer orden se pueden usar como modelo de un proceso estocástico que tenga las siguientes propiedades:

1. La probabilidad condicional cumple con el principio de Markov.

2. Existencia de un número finito de estados.

3. Las ocurrencias del proceso son en periodos iguales.

4. Las probabilidades P_{ij}, son constantes con respecto al tiempo o período de medición (*proceso homogéneo respecto a la variable tiempo*).

6.1.2. Arquitectura de un modelo oculto de Markov

Un HMM (siglas del inglés *Hidden Markov Model*: *modelo oculto de Markov*) es un proceso estocástico que consta de un proceso de Markov no observado $S = \{S_t\}$ y un proceso observado $O = \{O_t\}$ con estados dependientes de los estados no observados [66]. Un *modelo oculto de Markov, discreto, de primer orden* se define como $\lambda = (V, S, A, B, \Pi)$ donde:

- $V = \{v_1, v_2, ..., v_M\}$ es un alfabeto finito con M símbolos.
- $S = \{S_1, S_2, ..., S_N\}$ es un conjunto finito de N estados, con el estado al tiempo t denotado como q_t.
- $A = [a_{ij}]_{N \times N}$ es una matriz de probabilidades de transición donde el elemento a_{ij} es la probabilidad de transición del estado i al estado j.

  $a_{ij} = P(q_{t+1} = S_j \mid q_t = S_i)$ para todo $1 \leq i, j \leq N$.
- $B = [b_j(k)]_{N \times M}$ es la matriz de probabilidad de emisiónes, esto es que $b_j(k) = P(V_k \ en \ t \mid q_t = S_i)$.
- $\Pi = [\pi_i]_{1 \times N}$ es la matriz de probabilidades iniciales donde $\pi_i = P(q_1 = S_i)$ con $1 \leq i \leq N$.

Además $O = \{O_1, O_2, ..., O_T\}$ es la única secuencia que es posible observar, tiene longitud T y cada O_t es un símbolo de los posibles en V. Por conveniencia se usa la notación compacta $\lambda = (A, B, \Pi)$ para referirse a un modelo oculto de Markov.

Un modelo oculto de Markov se representa como un grafo dirigido de transiciones y emisiones. La arquitectura que permita modelar de la mejor forma posible las propiedades observadas, depende en gran medida de las características del problema. Las arquitecturas más usadas son:

- *Ergódicas* o completamente conectadas en las cuales cada estado del modelo puede ser alcanzado desde cualquier otro estado en un número finito de pasos. Un ejemplo se muestra en la figura 6.1.
- *Izquierda-derecha*, hacia adelante o Bakis, las cuales tienen la propiedad de que en la medida de que el tiempo crece, se avanza en la secuencia de observación asociada O.

En reconocimiento de la voz estas arquitecturas modelan bien los aspectos lineales. Un ejemplo se muestra en la figura 6.2

- *Izquierda-derecha paralelas*, son dos arquitecturas izquierda-derecha conectadas entre sí (ver figura 6.3).

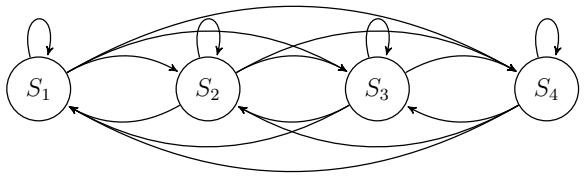


Figura 6.1: Ejemplo de modelo ergódico para cuatro estados.

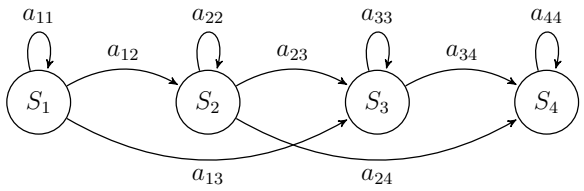


Figura 6.2: Ejemplo de modelo izquierda-derecha con cuatro estados.

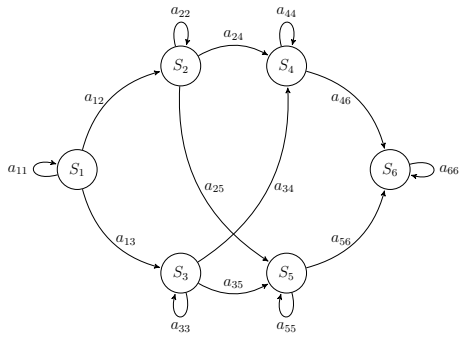


Figura 6.3: Ejemplo de modelo paralelo con seis estados.

6.1.3. Los tres problemas fundamentales

Existen tres problemas fundamentales que deben resolverse antes de poder aplicar los modelos ocultos de Markov en sistemas de reconocimiento de secuencias [66]. A continuación se enumeran:

1. Calcular la probabilidad de una secuencia observada.
   Dada una secuencia de observaciones $O = \{O_1, O_2, ..., O_T\}$ y un modelo $\lambda = (A, B, \Pi)$, calcular $P(O \mid \lambda)$.

2. Calcular la secuencia óptima de estados que genera las observaciones.
   Dada una secuencia de observaciones $O = \{O_1, O_2, ..., O_T\}$ y un modelo $\lambda = (A, B, \Pi)$, encontrar $Q = \{q_1, q_2, ..., q_T\}$ óptima para generar O.

3. El aprendizaje del modelo.
   Ajustar los parámetros A, B y Π para maximizar $P(O \mid \lambda)$.

6.1.3.1. Solución al problema de la evaluación

Se supone que se tiene la secuencia fija de estados $Q = \{q_1, q_2, ..., q_T\}$ con q_1 el estado inicial. Entonces la probabilidad de que la secuencia O sea emitida por Q y un modelo oculto de Markov λ se expresa como

$$P(O \mid Q, \lambda) = \prod_{t=1}^{T} P(O_t \mid q_t, \lambda) \tag{6.2}$$

Asumiendo independencia entre las observaciones tenemos

$$P(O \mid Q, \lambda) = \prod_{t=1}^{T} b_{q_t}(O_t) \tag{6.3}$$

Así, la secuencia de estados Q se expresa como

$$P(Q \mid \lambda) = \pi_{q_1} \prod_{t=2}^{T} a_{q_{t-1} q_t} \tag{6.4}$$

Y la probabilidad conjunta de que O y Q ocurran es

$$P(O, Q \mid \lambda) = P(Q \mid \lambda) P(O \mid Q, \lambda) = \pi_{q_1} \prod_{t=2}^{T} a_{q_{t-1} q_t} \prod_{t=1}^{T} b_{q_t}(O_t) \tag{6.5}$$

Si se consideran todas las secuencias de estados posibles se tiene que

$$P(O \mid \lambda) = \sum_{todo\ Q} \pi_{q_1} b_{q_1 O_1} \prod_{t=2}^{T} a_{q_{t-1} q_t} b_{q_t O_t} \tag{6.6}$$

La ecuación 6.6 se conoce como *evaluación por fuerza bruta* ya que para cada observación hay N posibles estados para transitar y si tenemos $\{1, 2, ..., T\}$ observaciones tenemos $N \times N \times ... \times N$, T veces por lo que la cantidad de operaciones es N^T y aproximadamente $2T$ productos en cada operación. Esta cantidad de operaciones no es factible de realizar, por ejemplo: para 10 estados ocultos y una simple secuencia de 100 símbolos habría que calcular $2 \times 100 \times (10^{100})$ operaciones; el número de átomos de hidrógeno en el universo se estima en cerca de 10^{80} átomos.

Para resolver el problema de la evaluación de manera eficiente se propone el procedimiento conocido como *procedimiento hacia adelante-hacia atrás* (del inglés *Forward-Backward procedure*) definiendo la variable *hacia adelante* $\alpha_t(i)$ como

$$\alpha_t(i) = P(O_1, O_2, ..., O_t, q_t = S_i \mid \lambda) \tag{6.7}$$

Y es posible encontrar una solución recurrente como sigue:

1. Inicialización:

$$\alpha_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \tag{6.8}$$

2. Inducción:

$$\alpha_{t+1}(j) = \left[\sum_{j=1}^{N} \alpha_t(i) a_{ij} \right] b_j(O_{t+1}), \quad 1 \leq t \leq T-1,\ 1 \leq i \leq N \tag{6.9}$$

3. Terminación:

$$P(O \mid \lambda) = \sum_{t=1}^{N} \alpha_t(i) \tag{6.10}$$

Con el procedimiento anterior, en la ecuación 6.9 se observa que se necesitan N productos para N estados posibles y T posibles observaciones, con lo que la cantidad de operaciones es igual a N^2T. Para el caso de $N = 10$ estados ocultos y $T = 100$ símbolos emitidos la cantidad de operaciones es $10^2(100) = 100^3$ operaciones, que es razonablemente computable en un tiempo determinado.

La variable *hacia atrás* no es necesaria para resolver el problema de evaluación, pero resulta de utilidad para desarrollar el problema de aprendizaje de un modelo sobre las observaciones obtenidas. Se define $\beta_t(i)$ como

$$\beta_t(i) = P(O_1, O_2, ..., O_t \mid q_t = S_i, \lambda) \tag{6.11}$$

Y la solución recurrente se define como:

1. Inicialización:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{6.12}$$

2. Inducción:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \quad t = T - i,\ 1 \leq i \leq N \tag{6.13}$$

6.1.3.2. Estimar la secuencia óptima de estados

La ruta más probable en un modelo oculto de Markov es útil para el aprendizaje y para el alineamiento de secuencias con el modelo. La ruta más probable $Q = \{q_1, q_2, ..., q_T\}$ para la secuencia de observación $O = \{O_1, O_2, ..., O_T\}$ puede ser calculada utilizando el algoritmo de Viterbi [66].

Se define el valor

$$\delta_t(i) = \text{Max}_{q_1, q_2, ..., q_{t-1}} [P(q_1, q_2, ..., q_t = i, O_1, O_2, ..., O_t \mid \lambda)] \tag{6.14}$$

La ecuación anterior busca la mejor probabilidad para toda la observación O. Puede deducirse que se debe rastrear la mejor probabilidad en cada trayectoria posible en el tiempo t tomando en cuenta las trayectorias anteriores. Así se tiene que:

$$\delta_{t+1}(i) = \left[\text{Max}_i(\delta_t(i) a_{ij})\right] b_j(O_{t+1}) \tag{6.15}$$

El procedimiento completo se lista a continuación:

1. Inicialización:

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \tag{6.16a}$$

$$\psi_1(i) = 0 \tag{6.16b}$$

2. Recursión:

$$\delta_t(j) = \text{Max}_{1 \leq i \leq N} [(\delta_{t-1}(i) a_{ij}) b_{ij}], \quad 2 \leq t \leq N\ 1 \leq j \leq N \tag{6.17a}$$

$$\psi_t(j) = \text{Arg Max}_{1 \leq i \leq N} [\delta_{t-1} a_{ij}], \quad 2 \leq t \leq N\ 1 \leq j \leq N \tag{6.17b}$$

3. Teminación:

$$P^* = \text{Max}_{1 \leq i \leq N}(\delta_T(i)) \tag{6.18a}$$

$$q_T^* = \text{Arg Max}_{1 \leq i \leq N}(\delta_T(i)) \tag{6.18b}$$

4. Rastreo hacia atrás de la secuencia:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, ..., 1 \tag{6.19}$$

6.1.3.3. El problema de aprendizaje del modelo

El problema más difícil de los los modelos ocultos de Markov es determinar un método para ajustar los parámetros A, B y Π del modelo para satisfacer los criterios de optimización. No se conoce una forma analítica para fijar los parámetros que maximicen la probabilidad de la secuencia de observación. Existen varios algoritmos disponibles para el entrenamiento de un modelo oculto de Markov, entre ellos, Baum-Welch que es un algoritmo iterativo [66].

Se define la variable $\xi_t(i,j)$ como:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j \mid O, \lambda) \tag{6.20}$$

De las definiciones de $\alpha_t(i)$ y $\beta_t(i)$ en las ecuaciones 6.7 y 6.11 se tiene la probabilidad condicional

$$\xi_t(i,j) = \frac{P(q_t = S_i, q_{t+1} = S_j \mid O, \lambda)}{P(O \mid \lambda)} = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(O \mid \lambda)} = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)} \tag{6.21}$$

Se define $\gamma_t(i)$

$$\gamma_t(i) = P(q_t = S_i \mid O, \lambda) \tag{6.22}$$

Y nuevamente se expresa en términos de las variables $\alpha_t(i)$ y $\beta_t(i)$ como

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{6.23}$$

Y $\gamma_t(i)$ se relaciona con $\xi_t(i)$ de la forma

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{6.24}$$

La suma en la ecuación 6.24 puede interpretarse como el número esperado de veces que se pasa por el estado S_i o el número de transiciones de S_i dada O, excluyendo el tiempo $t = T$. Similarmente la suma en la ecuación 6.21, desde $t = 1$ hasta $t = T-1$ se interpreta como el número esperado de transiciones del estado S_i al estado S_j.

De forma breve tenemos:

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{Número esperado de transiciones por } S_i \text{ en } O \tag{6.25}$$

$$\sum_{t=1}^{T-1} \xi_t(i) = \text{Número esperado de transiciones de } S_i \text{ a } S_j \tag{6.26}$$

Así, el procedimiento para actualizar los parámetros de un modelo oculto de Markov se realiza simplemente contando la ocurrencia de eventos y genera el procedimiento de estimación mostrado a continuación:

$$\hat{\pi}_i = \text{Frecuencia en el estado } S_i \text{ en el tiempo } t = 1 = \gamma_1(i) \tag{6.27}$$

$$\hat{a}_{ij} = \frac{\text{Número de transiciones de } S_i \text{ a } S_j}{\text{Número esperado de transiciones por } S_i} = \frac{\sum_{t=1}^{T-1} \xi_t(i)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{6.28}$$

$$\hat{b}_j(k) = \frac{\text{Número de veces en } S_j \text{ observando } v_k}{\text{Número de veces en } S_j} = \frac{\sum_{No.O_t=v_k}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \tag{6.29}$$

Existen adicionalmente tres problemas técnicos en la implementación en un modelo oculto de Markov, todos estos se tratan a detalle en [66] con su respectiva solución. Los tres problemas son:

- El Reescalamiento de las probabilidades en cada iteración durante la ejecución de los algoritmos.
- El tratamiento de observaciones con más de un solo símbolo (vectores de símbolos).
- Inestabilidad en la representación númerica al multiplicar recursivamente valores en el intervalo $[0,1]$ (problema conocido en inglés como *underflow*).

6.2. Redes neuronales artificiales

Las redes neuronales artificiales fueron motivadas debido al estudio del funcionamiento del cerebro. En 1911, Cajál y Ramón introducen el concepto de *neurona* como unidad funcional del cerebro y describen las estructuras que lo constituyen a través de la interacción de estas. Una neurona es una célula con una estructura que le permite la interconexión y comunicación con otras células vecinas. Las células neuronales se componen por zonas receptivas de señales bioeléctricas llamadas *dendritas*, estas conducen la señales hasta el cuerpo celular principal llamado *soma*, y un *axón*, que se extiende fuera de la célula y tiene la función de conducir la señales eléctricas hasta las *sinapsis*, estas se interconectan con muchas otras células [67]. La figura 6.4 muestra un esquema de la estructura de una neurona.

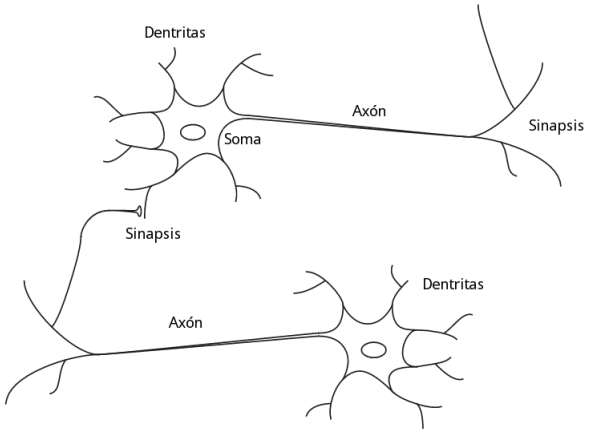


Figura 6.4: Estructura general de una célula neuronal.

En la década de 1940, McCulloch y Pitts desarrollan los primeros modelos de redes neuronales artificiales como resultado de los estudios en el funcionamiento de células neuronales [68]. Las redes neuronales artificiales son una remota aproximación al funcionamiento de una neurona real o del cerebro completo pero se basan en las principales características de ellas. A continuación se mencionan las principales características de su funcionamiento:

- Tienen una estructura altamente no lineal para realizar tareas.
- Trabajan en un grado masivo de paralelismo.
- Aprenden en base a experiencia.

- En conjunto, existe una tolerancia a fallos en células individuales.
- Son energéticamente muy eficientes.

En 1950 Rosenblatt propone una unidad informática de procesamiento [69] basada en los trabajos de McCulloch y Pitts, la cual tiene sinapsis j, con una entrada asociada x_j, ponderada por un peso w_{kj} que se conecta con la neurona k. El peso w_{kj} es de signo positivo si se le asocia como una señal de excitación o negativo como señal de inhibición. Un sumador añade todas las salidas ponderadas de las entradas y alimenta una función de activación φ, esta función se limitada en amplitud en un intervalo de operación o en valores finitos. Típicamente se agrega un valor constante $w_{k0} = b_k$ denominado *sesgo* que permite desplazar el intervalo de activación de la neurona.

Matemáticamente, una neurona artificial k se describe en las ecuaciones 6.30 a la 6.32.

$$v_k = \sum_{j=0}^{p} w_{kj}\ x_j \tag{6.30}$$

$$v_k = \begin{bmatrix} w_{k0} & w_{k1} & \dots & w_{kp} \end{bmatrix} \begin{bmatrix} x_0 = +1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix} = \mathbf{w_k}^T\ \mathbf{x} \tag{6.31}$$

$$y_k = \varphi(v_k) \tag{6.32}$$

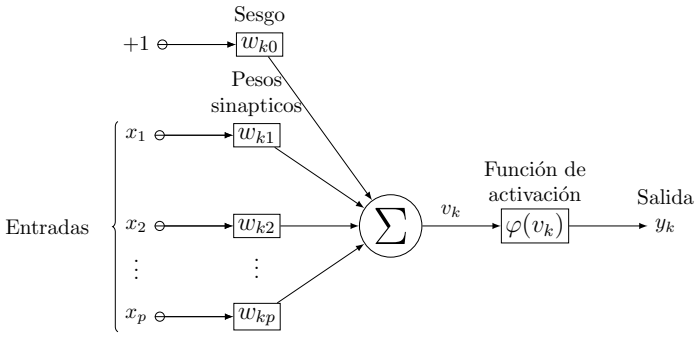


Figura 6.5: Ejemplo de una neurona artificial.

A la estructura mostrada en la figura 6.5 se le conoce como *perceptrón simple*. En 1962 Rosenblatt prueba que es posible entrenar al perceptrón simple y conjuntos de percep-

trones simples, alineados en forma de columna (a este arreglo se le denomina *capa simple*), para construir clasificadores lineales [70].

Para que una neurona pueda realizar una tarea, es decir, que pueda percibir un estimulo externo, se debe asociar la entrada de una señal en sus sinapsis con un valor especifico de salida de la función de activación, mediante un proceso supervisado de aprendizaje. A este proceso se le llama *proceso de aprendizaje del perceptrón simple*. Esto se expresa en las ecuaciones 6.33 y 6.34.

$$w_j(k+1) = w_j(k) + \Delta w_j(k), \quad k = 1, 2, ... \tag{6.33}$$

$$\Delta w_j(k) = \eta(k)[d(k) - y(k)]x_j(x) \tag{6.34}$$

En la ecuación 6.34 se observa que la estimación de los pesos w_j es proporcional al error $e(k) = d(k) - y(k)$, donde $d(k)$ es la salida que se desea obtener. La constante $\eta(k)$ es un parámetro de signo positivo llamado *tasa de aprendizaje* y modifica la velocidad con la que converge el proceso de aprendizaje.

Si se define la función de error en el sentido cuadrático medio tenemos:

$$e(k) = \frac{1}{2}[d(k) - y(k)]^2 \tag{6.35}$$

Se aproxima el valor mínimo de la función $e(k)$ calculando la máxima razón de cambio respecto $y(k)$, es decir $\frac{de}{dy}$. Y la ecuación 6.34 se reescribe como:

$$\Delta w_j(k) = -\eta(k)[d(k) - y(k)]x_j(x) \tag{6.36}$$

Y los pesos se actualizan con la nueva ecuación. A este método se le conoce como *gradiente descendente* y se utiliza para encontrar el mínimo de una función de manera iterativa. Las ecuaciones anteriores son el ejemplo de gradiente descendiente para una función de una variable.

Se acostumbra nombrar a las redes neuronales artificiales de manera breve como *redes neuronales*, y a las neuronas artificiales solamente como *neuronas* en el contexto de la inteligencia artificial y procesamiento automático de información, por lo que en adelante se adopta esta terminología.

6.2.1. Redes neuronales de propagación hacia adelante

Una *red de múltiples capas*, es un conjunto de perceptrones simples organizados en columnas, que se interconectan con las capas sucesivas hasta llegar al final de la red. La primera capa se denomina *capa de entrada*, la última capa de nombra *capa de salida* y las capas intermedias se denominan *capas ocultas* (ver figura 6.6). Si una red neuronal tiene más de una capa oculta se le denomina una *red profunda* (profunda en el sentido de su arquitectura, no confundir con una red de aprendizaje profundo). Una *red neuronal de propagación hacia adelante* es una red de múltiples capas, a las cuales se aplica una señal de entrada que se propaga por todo el arreglo de neuronas hasta llegar a la capa de salida. No presenta realimentación por lo que la salida es solo función de los pesos de la red y la señal de entrada. También se le conoce como *perceptrón multicapa*.

En 1986, Rumelhart y otros [71] proponen usar perceptrones organizados en múltiples capas para resolver problemas no lineales. Estos perceptrones multicapa se *entrenan* mediante el algoritmo conocido como *regla delta generalizada*, también se le conoce como *algoritmo de retropropagación del error*.

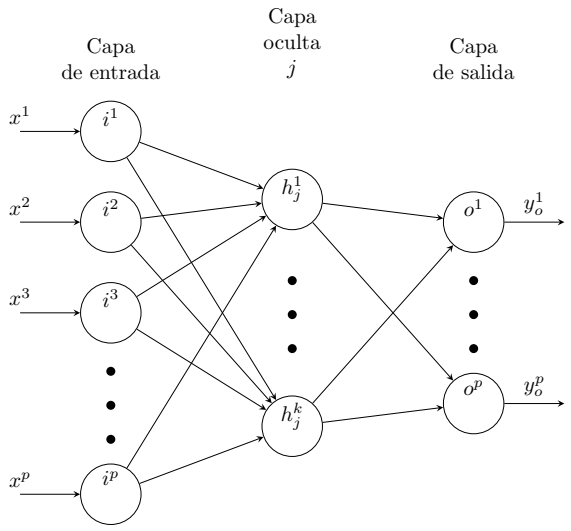


Figura 6.6: Ejemplo de perceptron multicapa.

La regla delta generalizada es un algoritmo que minimiza el error a la salida de una red neuronal multicapa para cada componente de entrada x^p, tomando en cuenta la aportación de cada peso individual de cada neurona, en cada capa, al error de salida. La salida la red para una neurona p en la capa k como:

$$y_k^p = F(s_k^p) \tag{6.37}$$

Con F una función de activación diferenciable, en la cual:

$$s_k^p = \sum_j w_{jk}\ y_j^p\ +\ b_k \tag{6.38}$$

Donde j es el índice de la capa. El error a la salida E para la ultima capa $j = o$ se define como:

$$E = \sum_p E^p = \frac{1}{2}\sum_p (d_o^p - y_o^p)^2 \tag{6.39}$$

Donde d_o^p es valor deseado de salida para la componente de entrada x^p. Para obtener la generalización correcta debemos obtener el negativo de la derivada parcial (método de gradiente descendente) para aproximar el mínimo de la función de error E^p. Definimos:

$$\Delta_p w_{jk} = -\gamma \frac{\partial E^p}{\partial w_{jk}} \tag{6.40}$$

Podemos escribir

$$\frac{\partial E^p}{\partial w_{jk}} = \frac{\partial E^p}{\partial s_k^p}\frac{\partial s_k^p}{\partial w_{jk}} \tag{6.41}$$

Y de la ecuación 6.38 sabemos que

$$\frac{\partial s_k^p}{\partial w_{jk}} = y_j^p \tag{6.42}$$

Se define δ_k^p como:

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p} \tag{6.43}$$

Así, se construye la regla de gradiente descendente

$$\Delta_p w_{jk} = \gamma \delta_k^p y_j^p \tag{6.44}$$

Para calcular δ_k^p aplicamos la regla de la cadena:

$$\gamma\delta_k^p = -\frac{\partial E^p}{\partial s_k^p} = -\frac{\partial E^p}{\partial y_k^p}\frac{\partial y_k^p}{\partial s_k^p} \tag{6.45}$$

De la ecuación 6.37 tenemos

$$\frac{\partial y_k^p}{\partial s_k^p} = F'(s_k^p) \tag{6.46}$$

Ahora si suponemos la capa $k = o$ tenemos que

$$\frac{\partial E^p}{\partial y_o^p} = -(d_o^p - y_o^p) \tag{6.47}$$

Por lo que para cada neurona p en la capa o se tiene que:

$$\delta_o^p = (d_o^p - y_o^p)F'(s_o^p) \tag{6.48}$$

Si es $k = h$, significa que la salida esta en una neurona de las capas ocultas y no estamos en capacidad de poder saber la contribución de dicha neurona al error en la capa de salida. Sin embargo podemos reescribir la función de error E^p en términos de las contribuciones de las entradas de las capas ocultas como $E^p = E^p(s_1^p, s_2^p, ..., s_j^p, ...)$ y podemos aplicar la regla de la cadena.

$$\frac{\partial E^p}{\partial y_h^p} = \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p} \frac{\partial s_o^p}{\partial y_h^p} = \frac{\partial E^p}{\partial s_o^p} \frac{\partial}{\partial y_h^p} \sum_{j=1}^{N_h} w_{ko}\ y_j^p = \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p} w_{ho} = -\sum_{o=1}^{N_o} \delta_o^p w_{ho} \tag{6.49}$$

Y el resultado de la ecuacion 6.49 se sustituye en la ecuación 6.45 y se obtiene

$$\delta_h^p = F'(s_h^p) \sum_{o=1}^{N_o} \delta_o^p\ w_{ho} \tag{6.50}$$

Con las ecuaciones 6.48 y 6.50 tenemos un procedimiento recursivo para estimar los nuevos pesos en las capas ocultas con la ecuación 6.44. La ecuación 6.50 se conoce como la *regla delta generalizada*.

6.2.2. Redes neuronales recurrentes

La redes neuronales recurrentes son redes neuronales que tienen memoria, para ello se realimentan sus salidas dentro de la red misma utilizando bloques de retraso de tiempo, así la red puede tomar en cuenta resultados anteriores para generar una salida. Estas arquitecturas permiten estimar secuencias de datos o señales que varían en el tiempo y hacer una predicción no lineal [72].

Una de la principales arquitecturas de redes neuronales recurrentes son la *red de Jordan* en la cual se realimentan las salidas de la red neuronal a una capa de neuronas llamadas

unidades de estado, estas a su vez reciben una copia de ellas mismas como se muestra en la figura 6.7. Otra arquitectura común es la *red de Elman* en la cual se realimentan salidas de las capas ocultas hacia una capa de neuronas llamada *capa de estado* (ver figura 6.8).

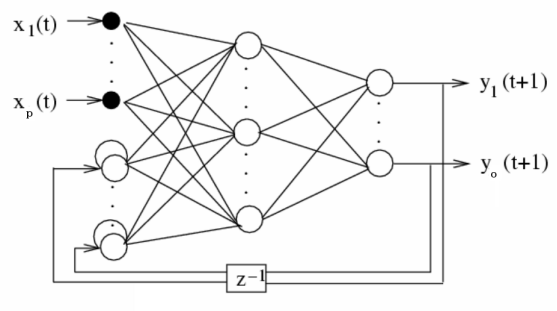


Figura 6.7: Ejemplo de red de Jordan.

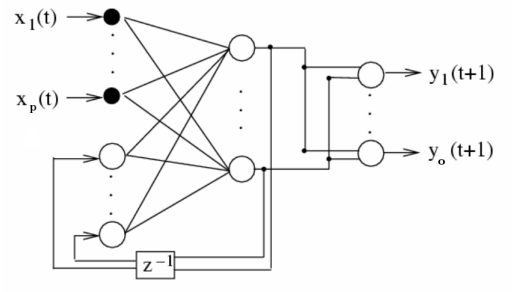


Figura 6.8: Ejemplo de red de Elman.

Una variación de la red de Elman y la red de Jordan es la red de tipo *modelo no lineal auto regresivo exógeno* (NARX, siglas del inglés *Nonlinear AutoRegressive eXogenous model*) que realimenta las salidas actuales además de insertar en la red valores anteriores de las entradas y salidas (*entradas exógenas*), por lo que es un modelo de red neuronal ideal para modelar sistemas dinámicos no lineales. Su estructura básica se muestra en la figura 6.9.

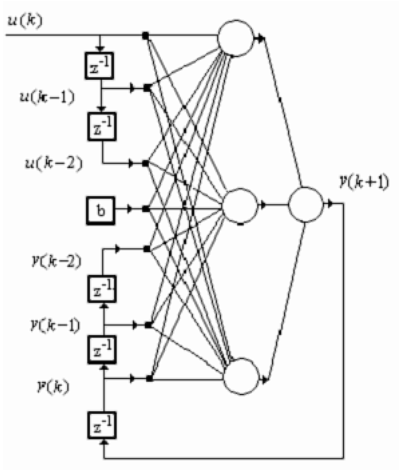
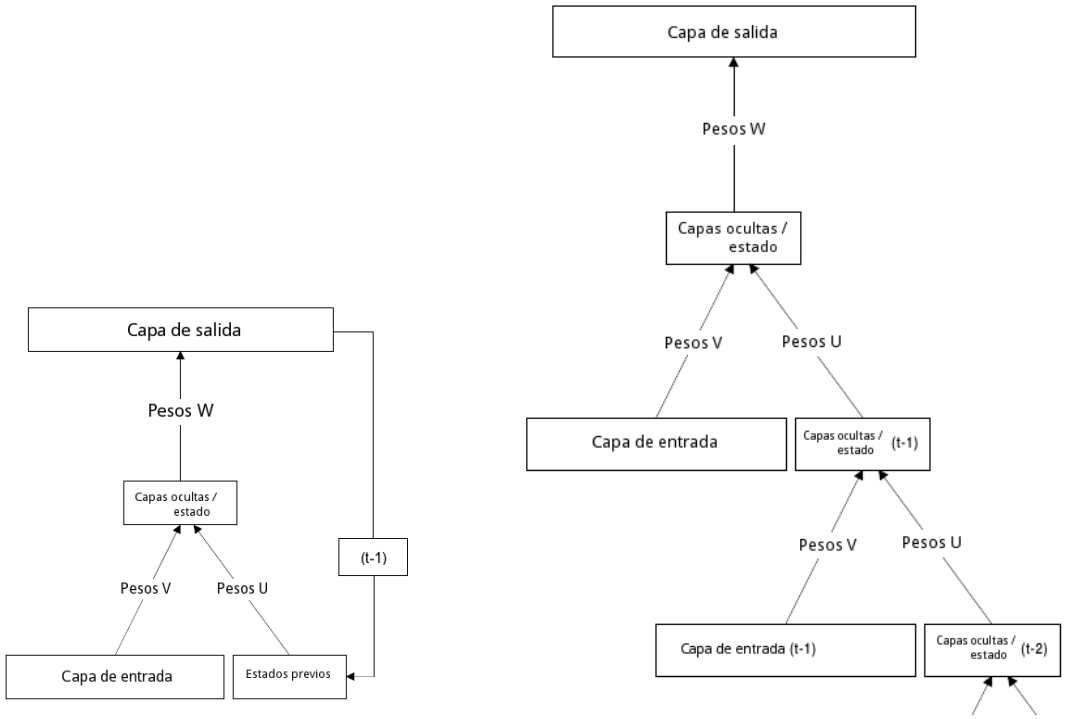


Figura 6.9: Ejemplo de red neuronal NARX.

Las redes neuronales recurrentes se entrenan mediante una variación del algoritmo de retropropagación del error llamado *algoritmo de retropropagación en el tiempo*, que toma en cuenta la inclusión de la variable tiempo en el comportamiento de la red neuronal [73]. El principio de funcionamiento de la retropropagación del error en el tiempo, es desdoblar la red recurrente (ver figura 6.10a) como una red de propagación hacia adelante (ver figura 6.10b) y truncar los retrasos en el tiempo de las entradas y las salidas que regresan, en un intervalo de tiempo definido.



(a) Ejemplo de red recurrente.

(b) Ejemplo de desdoblamiento en el tiempo.

Figura 6.10: Ejemplo de desdoblamiento en el tiempo para aplicar retropropagación del error.

Para una unidad de estado u_{hj} con j la neurona que recibe la señal y h la neurona que envía la señal en la capa anterior, una unidad de tiempo atrás. Puede deducirse con el mismo procedimiento para obtener la ecuación 6.50 ,que el error se propaga en las neuronas de estado como:

$$\delta_{pj}(t-1) = \sum_{h}^{m} \delta_{ph} u_{hj} F'(y_{pj}(t-1)) \tag{6.51}$$

En este capítulo se expusieron todos los elementos teóricos acerca de los modelos ocultos de Markov y redes neuronales artificiales, estos clasificadores se utilizaran para poder seleccionar la nueva información que se recibe a través del sensor Kinect, mediante la comparación de los nuevos descriptores estimados y un modelo previo asociado a una región en el espacio.

Capítulo 7

Implementación del sistema de localización

En este capítulo se presentan todos los detalles la implementación del sistema construido para la localización de un robot móvil de servicio. El sistema de localización tiene cuatro etapas fundamentales que son: la adquisición de información visual y de forma, detección de puntos de interés, descripción de estos puntos y clasificación de la secuencia de ellos mediante un modelo estadístico. La figura 7.1 muestra el diagrama de bloques del sistema.

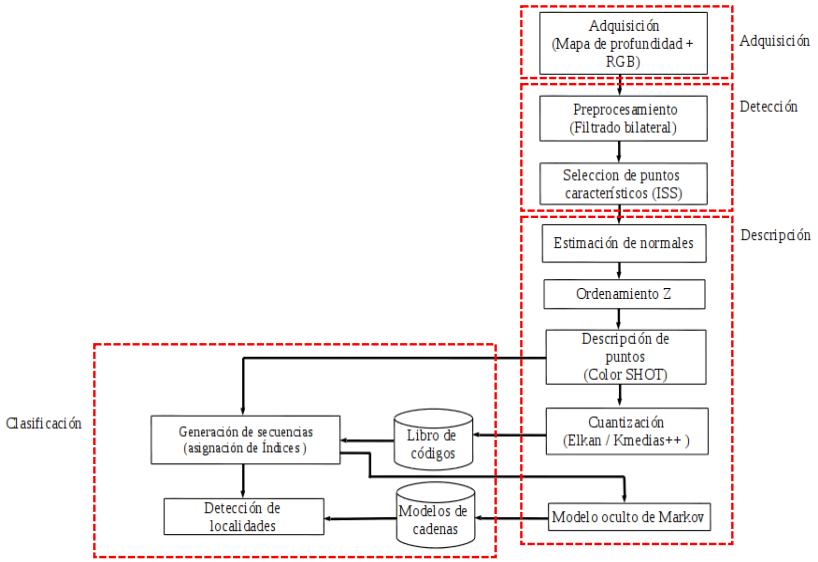


Figura 7.1: Diagrama de bloques del sistema de localización.

7.1. Adquisición

La etapa de adquisición de información visual se basa en la captura de escenas en entornos interiores con el sensor Kinect. Este sensor es un sensor de visión estéreo basado en la técnica de luz estructurada, además cuenta con una cámara tradicional que captura imágenes en color y las codifica en la representación RGB. Se debe calibrar la cámara RGB y el sensor de profundidad antes de utilizar el dispositivo [74], ya que con el uso y la manipulación del mismo surgen variaciones en las mediciones.

El sensor Kinect se conecta a una computadora para realizar la transferencia de los datos a travéz de un puerto USB 2.0. La estructura de datos provista por el dispositivo para transferir la información es un conjunto de cuatro matrices bidimensionales: una para cada color, de tamaño 480 renglones y 640 columnas, cada una con 8 bits de resolución de color; y la profundidad como una matriz de 480 renglones y 640 columnas, con una resolución de 16 bits en cada elemento.

Con los parámetros de calibración previamente obtenidos para la cámara RGB y la de profundidad, se aplica la corrección a cada toma antes de proyectar la información de profundidad y color en el sistema de referencia del sensor y obtener la nube de puntos.

Después se remueve todo punto en cada toma que este a más de 3 metros de distancia de la cámara para eliminar información como las paredes en el fondo de la toma. Esto debido a que puntos más lejanos contienen mayor cantidad de ruido en la coordenada z.

Por último se aplica un proceso de filtrado para atenuar la distorsión debida al ruido. Se aplica una versión de filtrado bilateral para nubes de puntos. Un ejemplo del resultado se muestra en la figura 7.2.

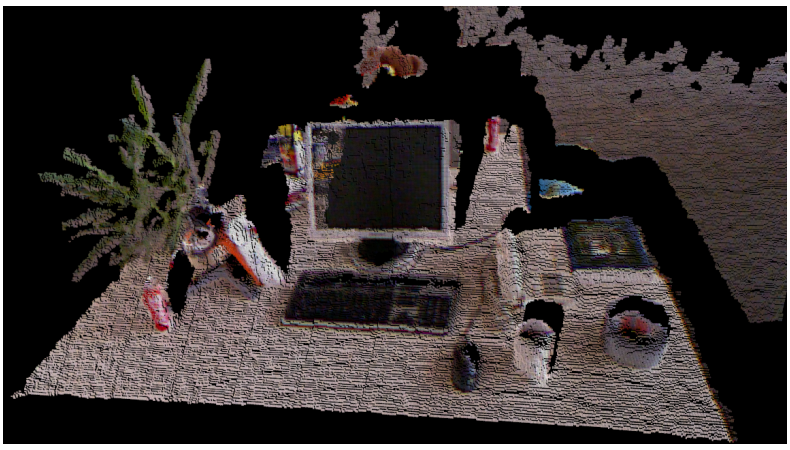


Figura 7.2: Nube de puntos obtenida después del proceso de rectificación y proyección.

7.2. Detección

En esta etapa se localizan puntos característicos altamente repetibles usando el descriptor ISS, solamente hasta la etapa de selección de puntos que implementa. La figura 7.3 muestra algunos puntos seleccionados de una nube de puntos.

Figura 7.3: Ejemplo de puntos característicos detectados en una nube de puntos usando ISS.

7.3. Descripción

Para realizar la descripción de la información se hace una estimación de las normales asociadas a los puntos seleccionados en la etapa de detección [75]. Estas normales se pasan como parámetros al descriptor Color SHOT ya que los requiere para estimar el sistema de referencia local como se explicó en la sección 4.3. Un ejemplo de las normales estimadas para cada punto característico se muestra en la figura 7.4. Antes de realizar la descripción, se ordenan los puntos usando una curva fractal llamada curva de Morton. Esta curva ordena los puntos en una malla, de manera única asignando un código a ellos (la malla de soporte es construída implícitamente por el sensor Kinect al transferir la información como una matriz). Este paso es fundamental para la construcción de un modelo oculto de Markov, ya que permite explotar el ordenamiento de manera local en cada toma y evita la necesidad de alimentar más de una toma contigua al momento de clasificar entre distintos modelos [10].

El ordenamiento se realiza en dos dimensiones explotando la estructura de matriz

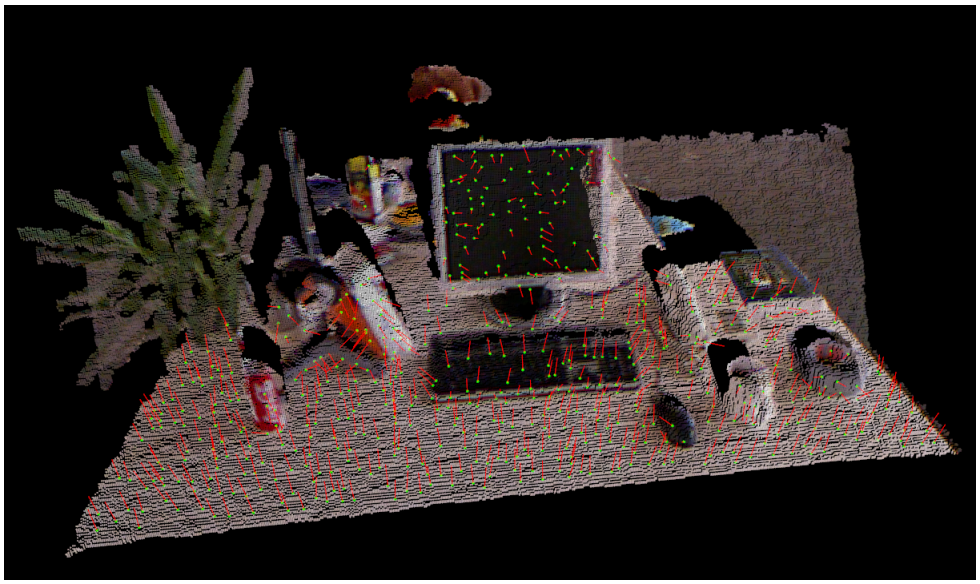


Figura 7.4: Ejemplo de estimación de normales para cada punto característico.

que provee el dispositivo, aunque existen versiones del mismo que no necesitan soporte, es decir, pueden trabajar con cualquier punto en un espacio continuo [49]. El algoritmo bidimensional funciona con operaciones a nivel de bits, por lo que resulta la elección óptima en cuanto a desempeño.

En la etapa de cuantización se usa una variación del algoritmo K medias llamada algoritmo de Elkan, también se usa un método de inicialización óptimo llamado K medias++ que asegura la convergencia y velocidad. Así, es posible cuantizar los descriptores Color SHOT que contienen 1344 dimensiones, algoritmos más simples resultan una mala opción en cuanto a tiempo de ejecución se refiere y también en cuanto a la dimensionalidad del espacio de trabajo. Cabe notar que métodos de reducción de dimensiones son igualmente complejos para esta problemática [76], por lo que se optó por concentrar el tiempo de trabajo en algoritmos más eficientes en lugar de reducir la dimensión de los datos, aunque sería una variante de investigación igualmente válida.

7.4. Clasificación

En la etapa de clasificación, primeramente se organiza la información que proviene del sensor Kinect capturada en forma de vídeo, a este vídeo se le denomina *escena*, la escena se subdivide en intervalos uniformes cada 10 tomas generando un conjunto de datos más

compacto, esto solo para agilizar el proceso de entrenamiento. Enseguida se etiquetan las *localidades* contenidas en la escena, que son subregiones de espacio, usando la búsqueda entre los descriptores más cercanos según un umbral, desde la primera toma capturada hasta la que ya no los presente dentro del intervalo seleccionado (este método se conoce como *búsqueda de correspondencias entre descriptores*), este umbral es ajustado al 1 % [10]. Después se toma el cuadro consecutivo y se repite el procedimiento, lo que genera una gráfica como se muestra en la figura 7.5.

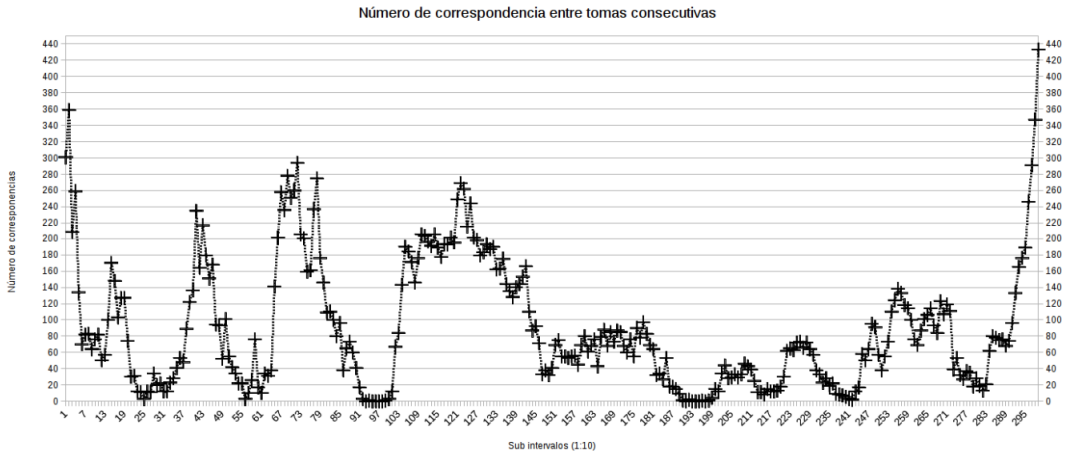


Figura 7.5: Gráfica de correspondencias para la selección de las localidades.

Con el proceso anterior se subdivide el vídeo de la escena en subunidades llamadas localidades, cada una de ellas contiene información única de color y de forma, asociada a los objetos que se muestran en cada toma. Las localidades quedan definidas por los intervalos en el vídeo limitados por los valles en la gráfica de correspondencias entre tomas. La motivación de esta técnica es que el sistema particiona el vídeo sin supervisión después de establecer los umbrales. La búsqueda de correspondencias se hace usando árboles KD aleatorios.

Los descriptores obtenidos de cada toma, en el intervalo de cada localidad, serán usados para el entrenamiento de un modelo oculto de Markov y dos redes neuronales. La transformación de los descriptores en símbolos se hace asociando el índice del vector más cercano en el libro de códigos a cada descriptor y las secuencias de símbolos se construyen con la concatencación de los índices. La búsqueda del vector más cercano en el libro de códigos se realiza usando árboles KD aleatorios.

El libro de códigos se construye con la cuantización de las tomas seleccionadas en intervalos uniformes del vídeo original.

Las secuencias se clasifican en la etapa de pruebas usando el procedimiento hacia adelante (forward procedure) generando una probabilidad, esta probabilidad se evalúa para cada modelo de Markov entrenado. El modelo de Markov que arroja la mayor probabilidad para dicha secuencia de símbolos será elegido como el origen de las observaciones, esto permite localizar en el espacio al robot móvil usando solamente información de su sistema de visión. El diagrama de bloques de dicho clasificador se muestra en la figura 7.6.

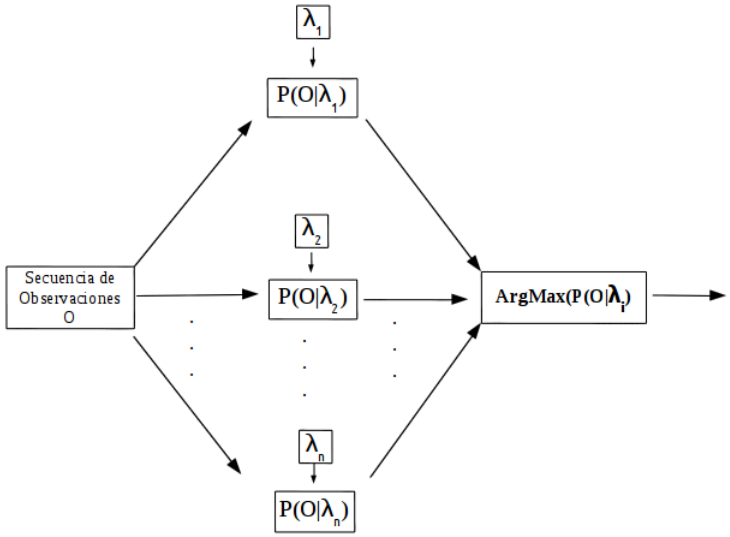


Figura 7.6: Diagrama de bloques del clasificador de localidades.

7.4.1. Perfiles de modelos ocultos de Markov

Para la construcción de los modelos ocultos de Markov, se usa una extensión de los modelos usados en [66], esta adecuación es llamada *perfil de modelo oculto de Markov* (con siglas PHMM, del inglés *Profile Hidden Markov Model*) y es usada para tener una subsecuencia de símbolos como patrón de búsqueda, contenida en las secuencias completas a detectar y clasificar. Debido a este comportamiento se utiliza para analizar secuencias específicas de ADN en largas secuencias arrojadas por máquinas secuenciadoras [77]. Cabe resaltar que siguen siendo cadenas de Markov de primer orden, con una distribución de probabilidad constante en el tiempo.

Para implementar estos modelos es necesario agregar una característica a los algoritmos fundamentales llamada estados silenciosos [78]. Los estados silenciosos permiten

transiciones entre estados sin emitir símbolos por lo que se puede transitar entre símbolos que no se entrenaron para seguir analizando la secuencia completa. También permiten concatenar modelos completos añadiendo un estado silencioso al inicio y final de cada cadena de Markov, lo que evita la necesidad de reentrenar el modelo completo. Un ejemplo de un perfil de modelo oculto de Markov se muestra en la figura 7.7.

La principal diferencia entre los modelos ocultos tradicionales y el perfil de modelo oculto, es que en el perfil de modelo oculto entrena una arquitectura simple y de pequeña longitud, mientras que las arquitecturas comunes requieren secuencias más largas y hay más complejidad para tratar los elementos borrados o añadidos en las secuencias (oclusión).

El fenómeno de inserción y borrado de símbolos es común cuando el ruido es muy grande en las señales capturadas y modela particularmente bien las oclusiones, ya que el truncar la secuencia, remover bloques completos o añadir símbolos extraños en ella, no modifica la verosimilitud del perfil ni entre distintos perfiles. En las arquitecturas tradicionales, durante su entrenamiento, se genera un error de sobre ajuste en caso de tener secuencias muy pequeñas o sub ajuste si el ruido añade símbolos constantemente a la secuencia, por el contrario los perfiles de modelos ocultos manejan de manera simple este fenómeno.

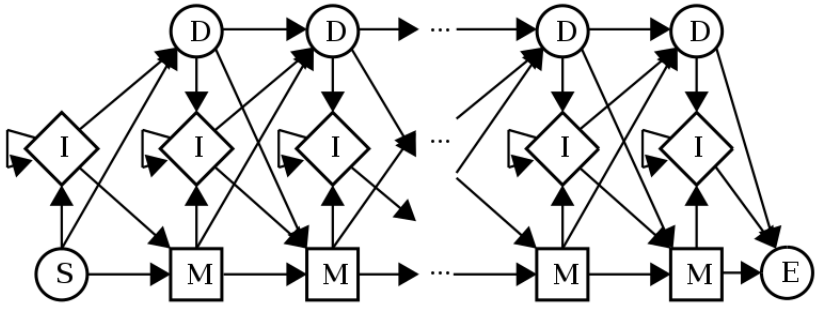


Figura 7.7: Perfil de modelo Oculto de Markov. Los estados con la letra S (start) y E (end) son estados silenciosos para modelar el inicio y final de una secuencia, igualmente los estados D (deleted), los estados con la letra M (matched) son estados con una probabilidad asociada al símbolo específico a buscar, en una posición específica. Los estados con las letras I (insert) modelan la inserción de símbolos en la secuencia patrón (profile).

7.5. Herramientas de desarrollo de software para el sistema de navegación

El software utilizado para la detección y descripción de las nubes de puntos fue la biblioteca PCL 1.8 [21], ya que contiene la mayoría de los algoritmos necesarios para el manejo de este tipo de representación de datos. En particular contiene las funciones en código C/C++ para filtrado bilateral, el detector ISS y el descriptor Color SHOT. Cabe notar que es una versión *Beta* por lo cual no se recomienda su uso si no se tiene experiencia en los algoritmos básicos de visión por computadora (esta biblioteca presenta fallos en la paralelización sobre CPU y GPU de algunos algoritmos como es la proyección para la obtención de la nube de puntos y el descriptor Color SHOT, en otros casos no provee las funciones necesarias como es el caso para el proceso de eliminación de distorsión, igualmente presenta fallos en la estimación de normales en paralelo).

También se usó la biblioteca FLANN (siglas del inglés *Fast Library for Approximate Nearest Neighbors*) [63] para búsquedas de los vecinos más cercanos mediante árboles KD aleatorios. La distorsión se elimina usando funciones integradas en la biblioteca OpenCV [19].

La implementación del ordenamiento en Z es una implementación propia en C/C++.

El algoritmo de cuantización vectorial es autoría de [61] en la biblioteca llamada *eak-means*, suministrada en código C y con una interfaz de usuario en lenguaje Cython/Python, con optimizaciones basadas en OpenBLAS [79].

Para la construcción de los modelos ocultos de Markov se utilizó la biblioteca YAHMM (siglas del inglés *Yet Another HMM*) desarrollada por [77] y distribuida en lenguaje Python.

El programa principal se construyó en lenguaje C++ además de un módulo de evaluación de resultados hecho en lenguaje Python.

Para la comparación de resultados se programaron dos tipos de redes neuronales como clasificadores: una red de propagación hacia adelante y una red recurrente del tipo NARX, esto con la bibliotecas Keras [80] basada en la bilbioteca Theano [81], TensorFlow [82] además de la biblioteca PyBrain [83], todas ellas en lenguaje Python. Estas bibliotecas son de amplio uso en aplicaciones de aprendizaje de máquina y aprendizaje profundo y son distribuidas de manera libre para su uso o modificación.

Ejemplos de todas las implementaciones realizadas se muestran en el apéndice A de manera breve. Detalles sobre el rendimiento y requerimientos están disponibles de manera pública en las referencias de cada biblioteca.

Capítulo 8

Pruebas y resultados

Para las pruebas se eligió un vídeo del repositorio de validación del grupo de visión por computadora en la universidad técnica de Munich [16]. Este conjunto de datos es utilizado para validación de sistemas de navegación de robots móviles. Se elige este conjunto de datos debido a su disponibilidad pública, además de contar con vídeos de espacios interiores, que es el objetivo de este trabajo de tesis. También incluye los parámetros de calibración del sensor utilizado. El vídeo utilizado fue *freiburg2_ xyz_ validation*.

El vídeo seleccionado para la validación es una escena artificial de una oficina, no contiene cambios de la escena durante su captura y presenta un recorrido cerrado. Este vídeo tiene más de 3000 cuadros de imágenes RGB y de profundidad, por lo que se muestrea cada 10 tomas para generar un conjunto de entrenamiento de 300 tomas. Posteriormente se crean localidades espaciales usando búsqueda de correspondencias entre descriptores. La figura 8.1 muestra algunas tomas seleccionadas del conjunto de datos basados en la figura 7.5, en la que se aprecian 6 localidades separadas por los valles en la gráfica.

Para la construcción del cuantizador (libro de códigos), se aplica K medias a las 300 tomas seleccionadas del vídeo original.

Después se construyen los modelos ocultos de Markov con la selección aleatoria del 50% de las 300 tomas preseleccionadas respetando su orden temporal, la localidad a la que pertenecen y sin repetición, esto significa que se asigna un modelo de Markov por localidad por simplicidad, aunque igualmente puede construirse un modelo para todas ellas. La diferencia radica en que en el proceso de detección, si solo se cuenta con un único modelo, hay que buscar la transición entre estados usando el algoritmo de Viterbi; si se usa un modelo por localidad, la probabilidad asociada a una secuencia de observaciones nos localiza en la escena automáticamente comparando entre todas las probabilidades de

cada modelo, sin necesidad de procesar la secuencia de estados.

Para el conjunto de pruebas se toma aleatoriamente el 50% de tomas restantes de las 300 preseleccionadas, además de añadir tomas del vídeo original en caso de que no sea posible elegir sin repetición, respetando el intervalo de selección. Es decir, en el subconjunto de 300 tomas, la localidad uno abarca de la toma 1 a la toma 25, pero respecto al vídeo original se tiene de la toma 1 a la toma 250, por lo que hay 250 tomas excluyendo las seleccionadas en el entrenamiento para validar la localidad uno.

Además, en todas las secuencias generadas para las pruebas se elimina el 10% de los símbolos en ellas de manera aleatoria y se reorganizan para llenar el espacio que quedó vacío. Esto con el fin de simular oclusión y probar la robustez de los perfiles de modelos ocultos de Markov.

Con los modelos ocultos ya entrenados, se procede a la validación de la siguiente manera: se evalúa cada toma de prueba, se compara con respecto a los seis modelos construidos. El modelo que arroja mayor probabilidad es el que emitió la toma. La figura 7.6 muestra la estructura del clasificador. Los resultados se almacenan en un tabla de confusión.

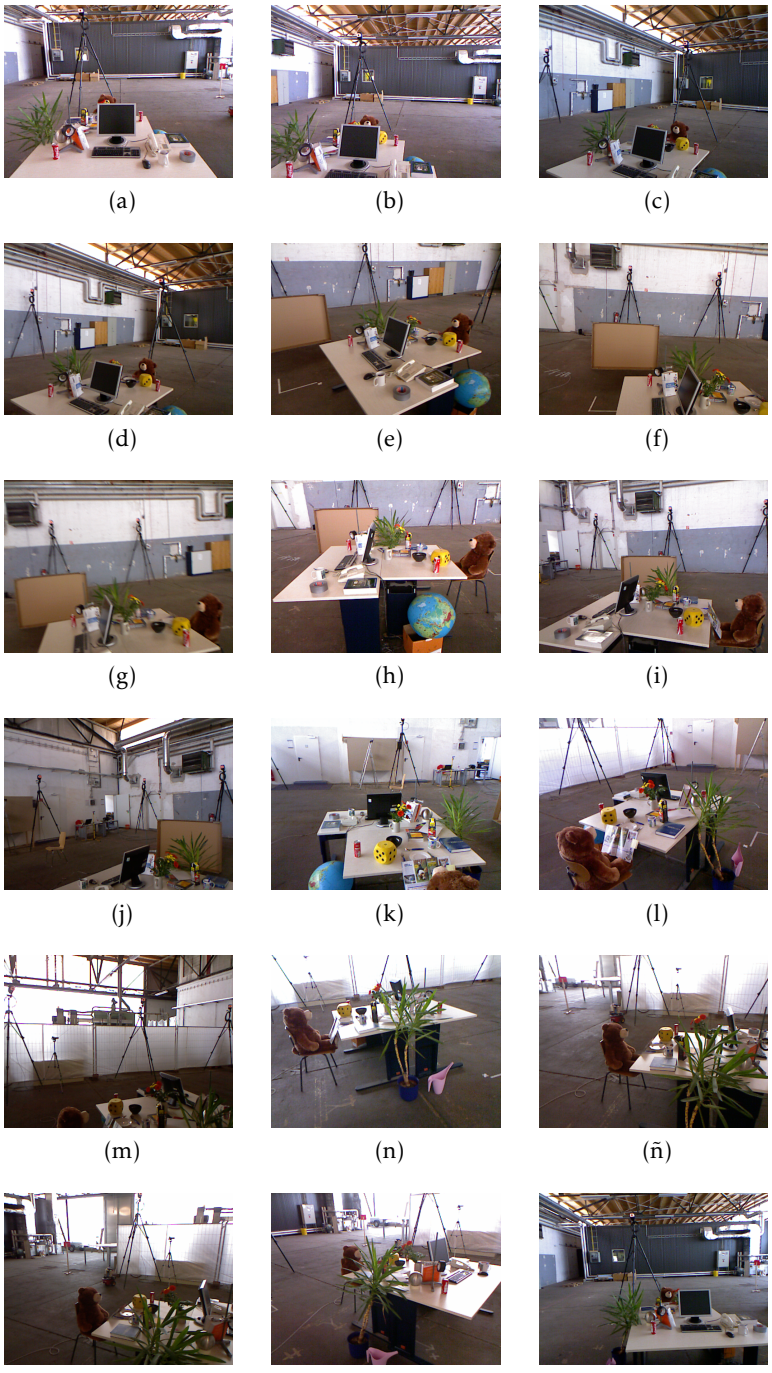


Figura 8.1: Un ejemplo de las localidades seleccionadas por busqueda de correspondencias (solo se muestran tres tomas por localidad). (a),(b),(c) para la localidad 1. (d),(e),(f) para la localidad 2. (g),(h),(i) para la localidad 3. (j),(k),(l) para la localidad 4. (m),(n),(ñ) para la localidad 5. (o),(p),(q) para la localidad 6. *Imágenes pertenecientes al conjunto de datos [16].*

8.1. Resultados para perfiles de modelos ocultos de Markov

Se probaron los modelos para representación de entre 2 hasta 2048 símbolos, en intervalos de potencias de 2 y algunos valores en medio de éstos.

Los perfiles se crean con longitud de 15 estados M con sus correspondientes estados I y D, esta longitud de perfil se determinó empíricamente mediante la alineación de todos los símbolos en las secuencias en bloques con longitudes de 15. Cada perfil representa que se busca como patrón los 15 símbolos en secuencia, más probables por posición, para cada localidad.

Se realizan en total, 10 entrenamientos aleatorios con sus 10 pruebas. A continuación se reporta el número total de tomas clasificadas para las 10 pruebas para cada libro de códigos, desde 2 hasta 2048 símbolos. La forma habitual de reportar los resultados es una *tabla de confusión*. Esta tabla se organiza por renglones para el valor actual de la clase que ha asignado el clasificador a la entrada y en las columnas se coloca la clase esperada, conocida de manera previa. Los valores que se acumulan sobre la diagonal corresponden a los valores clasificados correctamente y los valores asignados en otras columnas sobre el mismo renglón, son los errores en la clasificación.

Por brevedad se presentan los resultados de detección en la tabla 8.1 a la tabla 8.4 para 16, 64, 512 y 1024 símbolos respectivamente.

El porcentaje de aciertos total en la detección para todos los símbolos se muestra en la figura 8.2.

		Valor real									
		L_1	L_2	L_3	L_4	L_5	L_6	R	NR	% R	% NR
Valor actual	L_1	**100**	20	0	0	0	0	100	20	83.3	16.7
	L_2	0	**140**	10	0	0	0	140	10	93.3	6.7
	L_3	10	40	**90**	10	0	20	90	80	52.9	47.1
	L_4	30	80	60	**250**	20	10	250	200	55.6	44.4
	L_5	0	0	40	0	**170**	0	170	40	81	19
	L_6	0	110	60	10	30	**70**	70	210	25	75
							Total	$\frac{\mathbf{820}}{\mathbf{1380}}$	$\frac{\mathbf{560}}{\mathbf{1380}}$	**59.4**	**40.6**

Tabla 8.1: Tabla de confusión para 16 símbolos. L_1 =Localidad 1. L_2 =Localidad 2. L_3 =Localidad 3. L_4 =Localidad 4. L_5 =Localidad 5. L_6 =Localidad 6. R=Secuencias reconocidas. NR=No reconocidas.

		Valor real									
		L_1	L_2	L_3	L_4	L_5	L_6	R	NR	% R	% NR
Valor actual	L_1	**100**	20	0	0	0	0	100	20	83.3	16.7
	L_2	0	**140**	0	10	0	0	140	10	93.3	0.7
	L_3	0	80	**80**	0	10	0	80	90	47	53
	L_4	40	70	0	**330**	10	0	330	120	73.3	26.7
	L_5	0	0	30	0	**160**	20	160	50	76.2	23.8
	L_6	10	100	0	0	30	**140**	140	140	50	50
							Total	$\frac{\mathbf{950}}{\mathbf{1380}}$	$\frac{\mathbf{430}}{\mathbf{1380}}$	**68.8**	**31.2**

Tabla 8.2: Tabla de confusión para 64 símbolos. L_1 =Localidad 1. L_2 =Localidad 2. L_3 =Localidad 3. L_4 =Localidad 4. L_5 =Localidad 5. L_6 =Localidad 6. R=Secuencias reconocidas. NR=No reconocidas.

		Valor real									
		L_1	L_2	L_3	L_4	L_5	L_6	R	NR	% R	% NR
Valor actual	L_1	**110**	10	0	0	0	0	110	10	91.7	8.3
	L_2	0	**140**	0	10	0	0	140	10	93.3	6.7
	L_3	0	20	**150**	0	0	0	150	20	88.2	11.8
	L_4	0	0	10	**440**	0	0	440	10	97.8	2.2
	L_5	0	0	0	0	**210**	0	210	0	100	0
	L_6	0	70	0	10	20	**180**	180	100	64.3	35.7
							Total	$\frac{\mathbf{1230}}{\mathbf{1380}}$	$\frac{\mathbf{150}}{\mathbf{1380}}$	**89.1**	**10.9**

Tabla 8.3: Tabla de confusión para 512 símbolos. L_1 =Localidad 1. L_2 =Localidad 2. L_3 =Localidad 3. L_4 =Localidad 4. L_5 =Localidad 5. L_6 =Localidad 6. R=Secuencias reconocidas. NR=No reconocidas.

		Valor real									
		L_1	L_2	L_3	L_4	L_5	L_6	R	NR	% R	% NR
Valor actual	L_1	**120**	0	0	0	0	0	120	0	100	0
	L_2	0	**150**	0	0	0	0	150	0	100	0
	L_3	0	0	**170**	0	0	0	170	0	100	0
	L_4	0	0	0	**450**	0	0	450	0	100	0
	L_5	0	0	0	0	**210**	0	210	0	100	0
	L_6	0	0	0	0	0	**280**	280	0	100	0
							Total	$\frac{\mathbf{1380}}{\mathbf{1380}}$	$\frac{\mathbf{0}}{\mathbf{1380}}$	**100**	**0**

Tabla 8.4: Tabla de confusión para 1024 símbolos. L_1 =Localidad 1. L_2 =Localidad 2. L_3 =Localidad 3. L_4 =Localidad 4. L_5 =Localidad 5. L_6 =Localidad 6. R=Secuencias reconocidas. NR=No reconocidas.

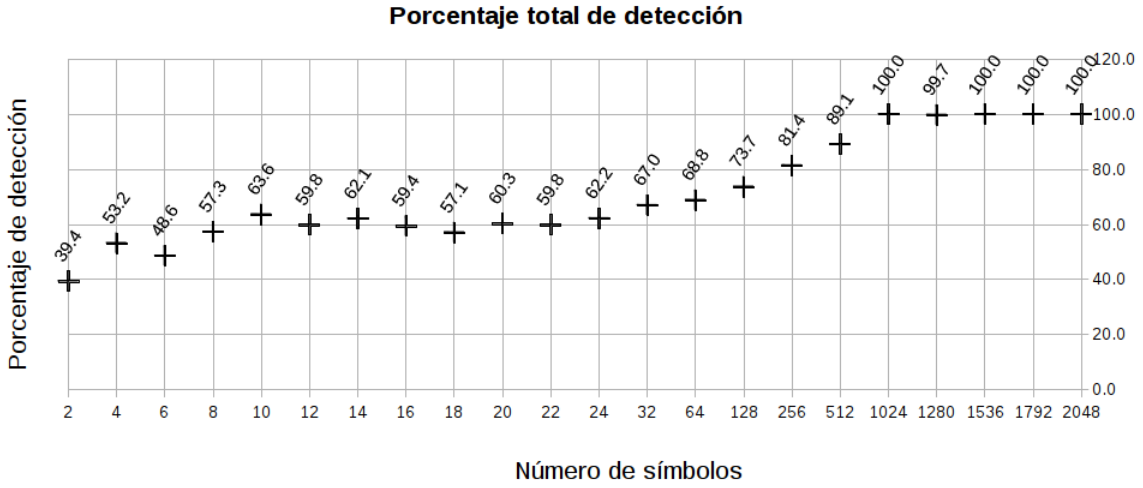


Figura 8.2: Porcentaje de detección total.

Los tiempos promedio de ejecución para el entrenamiento se muestran en la tabla 8.5 hasta la etapa de cálculo de descriptores. Estos tiempos fueron medidos en una computadora portátil con un procesador central i5-4300U, con 8Gb de memoria RAM a 1333 MHz en un sistema operativo Ubuntu 14.04. Todas la pruebas se realizan en ausencia de algún tipo de paralelización u optimización para el juego de instrucciones del procesador en la implementación. Los datos son cargados en la memoria RAM antes de la medición por lo que los tiempos reportados no incluyen el tiempo de acceso al disco duro ni demoras en la asignación de memoria por el sistema operativo.

Etapa del sistema	Tiempo de procesamiento (s)
Eliminación del fondo	$3.70\ x10^{-3}$
Filtrado bilateral	4.94
Estimación de normales	$609\ x10^{-3}$
Detector ISS	1.40
Orden en Z	$5.23\ x10^{-6}$
Descriptor Color SHOT	$87.6\ x10^{-3}$

Tabla 8.5: Tiempos promedio de entrenamiento hasta la etapa de descripción.

Los tiempos promedio de ejecución durante el entrenamiento para la etapa de cuantización se reportan en la tabla 8.6.

Número de símbolos	Tiempo de cuantización (ms)
2	4913
4	9049
6	14733
8	14548
10	18556
12	23869
14	24227
16	28307
18	36505
20	48819
22	30137
24	63450
32	40101
64	82690
128	124773
256	203260
512	366798
1024	656792
1280	738534
1536	896729
1792	1094362
2048	1230155

Tabla 8.6: Tiempos promedio de cuantización durante la etapa de entrenamiento.

Los tiempos promedio de entrenamiento para cada localidad se reportan en la tabla 8.7.

Por último se reporta el tiempo promedio de detección en la etapa de pruebas en la tabla 8.8.

Localidad	Tiempo promedio de entrenamiento (s)
L_1	25.25
L_2	27.32
L_3	44.34
L_4	95.09
L_5	42.78
L_6	21.85

Tabla 8.7: Tiempos promedio de entrenamiento por cada localidad.

Tiempo promedio en la etapa de detección (ms)
109.72

Tabla 8.8: Tiempo promedio de detección.

8.2. Resultados para redes neuronales artificiales

8.2.1. Red neuronal de propagación hacia adelante

Para comparar los resultados obtenidos usando clasificación con los modelos ocultos de Markov se diseñó un par de redes neuronales artificiales, una recurrente y una de propagación hacia adelante. La red neuronal de propagación hacia adelante se compone de 1344 neuronas en su capa de entrada con función de activación lineal con las entradas normalizadas entre valores $[-1,1]$, la capa oculta tiene 50 neuronas con función de activación tangente hiperbólica (ver ecuación 8.1) y seis neuronas de salida con función de activación *softmax* (ver ecuación 8.2) para mapear la salida con las correspondientes seis localidades en la que se separó el conjunto de datos. Los vectores de salida deseados tienen forma de vector. Por ejemplo: para la localidad 1, $\mathbf{d}_1 = (0,0,0,0,0,1)$, para la localidad 2, $\mathbf{d}_2 = (0,0,0,0,1,0)$ hasta la localidad 6 con $\mathbf{d}_6 = (1,0,0,0,0,0)$.

$$y(x) = \frac{1 - e^{-x}}{1 + e^{2x}} \tag{8.1}$$

$$y(x_j) = \frac{e^{x_j}}{\sum_k e^{x_k}}, \quad con \quad k = 1,2,...,N \tag{8.2}$$

Igualmente, se usa el 50% de las tomas preseleccionadas para entrenamiento y el res-

tante 50% para evaluación de manera aleatoria. Esto se realiza 10 veces. Debido a que los modelos ocultos clasifican secuencias de símbolos y la red de propagación hacia adelante clasifica solo descriptores individuales, se alimenta con las secuencias de descriptores sin ser convertidos a símbolos, y se considera que ha clasificado toda la secuencia correctamente solo si todos los descriptores han sido clasificados correctamente. Cada secuencia de descriptores corresponde a los descriptores obtenidos en cada nube de puntos de una toma.

La tabla 8.9 muestra los resultados de detección para la red propuesta.

		Valor real									
		L_1	L_2	L_3	L_4	L_5	L_6	R	NR	% R	%NR
Valor actual	L_1	**120**	0	0	0	0	0	120	0	100	0
	L_2	15	**135**	0	0	0	0	135	15	90	10
	L_3	0	0	**165**	5	0	0	165	5	97	3
	L_4	5	0	5	**430**	5	5	430	20	96	4
	L_5	0	0	0	5	**200**	5	200	10	95.2	4.8
	L_6	5	0	0	0	5	**270**	270	10	96.4	3.6
							Total	$\frac{\mathbf{1320}}{\mathbf{1380}}$	$\frac{\mathbf{60}}{\mathbf{1380}}$	**95.7**	**4.3**

Tabla 8.9: Tabla de confusión para la red neuronal de propagación hacia adelante. L_1 =Localidad 1. L_2 =Localidad 2. L_3 =Localidad 3. L_4 =Localidad 4. L_5 =Localidad 5. L_6 =Localidad 6. R=Secuencias reconocidas. NR=No reconocidas.

El tiempo total de entrenamiento se reporta en la tabla 8.10 y el tiempo de detección por secuencia se muestra en la tabla 8.11.

Tiempo de entrenamiento (s)
1477.56

Tabla 8.10: Tiempo de entrenamiento de la red neuronal de propagación hacia adelante.

Tiempo promedio de detección (ms)
578.36

Tabla 8.11: Tiempo promedio de detección de la red neuronal de propagación hacia adelante.

8.2.2. Red neuronal recurrente tipo NARX

En el caso de la red recurrente, para comparar la estructura de la solución con los modelos ocultos de Markov, se diseña una red del tipo auto regresivo con entradas exógenas para modelar el problema como una señal que varía en el tiempo. Esta red se compone de 1344+6 neuronas en su capa de entrada con funcion de activación lineal entre valores $[-1,1]$, la capa oculta tiene 50 neuronas con función de activación tangente hiperbólica (ver ecuación 8.1) y seis neuronas de salida con función de activación lineal limitada en el intervalo $[-1,1]$, para mapear la salida con las correspondientes seis localidades en las que se separó el conjunto de datos. La entrada y la salida se alimentan con dos retrasos obteniendo un dezplazamiento en tiempo de hasta $(t-2)$ muestras. Esta red se entrena usando retropropagación del error en el tiempo, como se explicó en el capítulo seis.

En este tipo de redes se evalúa el desempeño calculando la función de correlación del error, por lo que para cada entrada aplicada se almacena el error entre el valor deseado y el obtenido en todas las neuronas de salida. Los vectores de salida deseados tienen forma de vector. Por ejemplo, para la localidad 1, $\mathbf{d}_1 = (0,0,0,0,0,1)$, para la localidad 2, $\mathbf{d}_2 = (0,0,0,0,1,0)$ hasta la localidad 6 con $\mathbf{d}_6 = (1,0,0,0,0,0)$. La figura 8.3 muestra la autocorrelación del error.

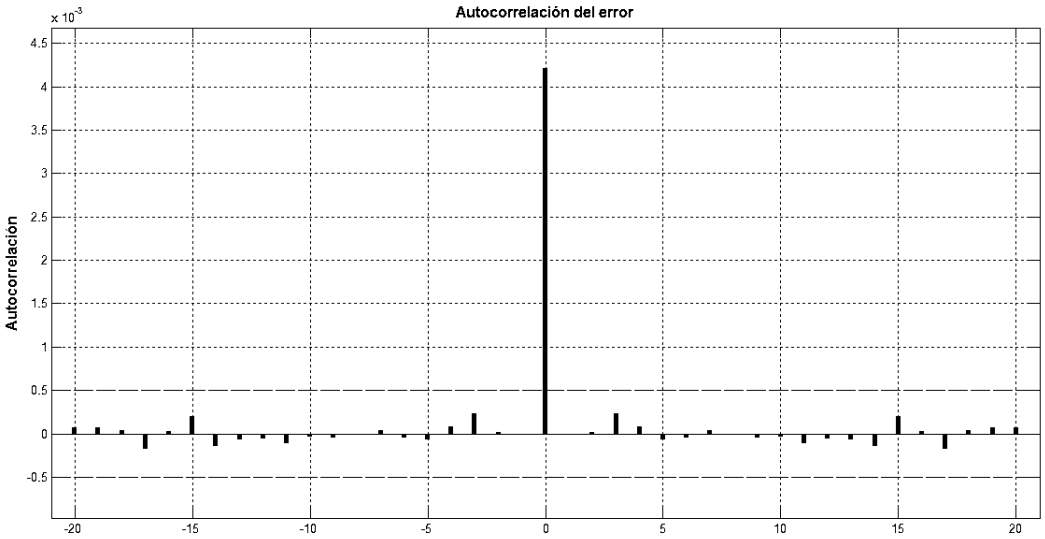


Figura 8.3: Gráfica de autocorrelación del error.

Al tratarse de la función de autocorrelación del error, se espera que se comporte como ruido blanco, es decir, que no haya correlación entre los errores en distintos tiempos y que no se propague este error al retroalimentar la red. De ser así, el error tendrá forma semejante a la autocorrelación del ruido blanco que es una función impulso en el origen.

Para calcular la certeza de que la red recurrente clasifica correctamente, se debe estimar el *indice de confiabilidad* que es el intervalo permitido para los valores de correlación. Para esto primero se calcula la media μ y la deviación estandart σ de los errores obtenidos en el entrenamiento. Despues se supone que el error de distribuye como $\epsilon \sim N(\mu, \sigma^2)$ para aplicar la transfomación que se desarrolla a continuación.

La probabilidad de que el error esté dentro del intervalo $[\mu - n\sigma, \mu + n\sigma]$ esta dado por:

$$P(\mu - n\sigma < \ x \ < \mu + n\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{\mu - n\sigma}^{\mu + n\sigma} e^{\frac{-(x-\mu)^2}{(2\sigma^2)}} dx \tag{8.3}$$

$$P(\mu - n\sigma < \ x \ < \mu + n\sigma) = \frac{2}{\sigma\sqrt{2\pi}} \int_{\mu}^{\mu + n\sigma} e^{\frac{-(x-\mu)^2}{(2\sigma^2)}} dx$$

Si $u = \frac{x-\mu}{\sqrt{2}\sigma}$ entonces $du = \frac{dx}{\sqrt{2}\sigma}$ y al sustituir tenemos:

$$P(\mu - n\sigma < \ x \ < \mu + n\sigma) = \frac{2}{\sigma\sqrt{2\pi}} \sqrt{2}\sigma \int_{0}^{\frac{n}{\sqrt{2}}} e^{-(u)^2} du \tag{8.4}$$

$$P(\mu - n\sigma < \ x \ < \mu + n\sigma) = \frac{2}{\sqrt{\pi}} \int_{0}^{\frac{n}{\sqrt{2}}} e^{-(u)^2} du$$

$$P(\mu - n\sigma < \ x \ < \mu + n\sigma) = erf(\frac{n}{\sqrt{2}})$$

Y al despejar se tiene que:

$$n = \sqrt{2}\ erf^{-1}(P) \tag{8.5}$$

Donde $erf()$ es la función de error y $erf^{-1}()$ es la función de error inversa. Algunos valores para P y n se muestran en la tabla 8.12, al valor de $n\sigma$ se le conoce como índice de confiabilidad.

P	nσ
0.800	1.28155σ
0.900	1.64485σ
0.950	1.95996σ

Tabla 8.12: Tabla para algunos índices de confiabilidad.

Para la red NARX diseñada, la desviación estandar del error σ es aproximadamente $\sigma = 2.551 \times 10^{-4}$ por lo que $n\sigma = 4.90 \times 10^{-4}$ para una probabilidad de 0.95. Este límite se muestra

en la figura 8.3 como 0.5×10^{-3} y -0.5×10^{-3} para un 95% de confiabilidad, excepto en el origen. Esto significa que no hay valores de correlación en el error que sean significativos en la red por lo que el diseño es correcto.

La principal desventaja de este sistema de clasificación es que el algoritmo de retropropagación en el tiempo demora demasiado si las entradas tienen naturaleza vectorial. Los tiempos de entrenamiento y detección se reportan en la tabla 8.13 y la tabla 8.14 respectivamente.

Tiempo de entrenamiento (s)
115226.12

Tabla 8.13: Tiempo de entrenamiento de la red neuronal NARX.

Tiempo promedio de detección (ms)
2294.60

Tabla 8.14: Tiempo promedio de detección de la red neuronal NARX.

8.3. Análisis de resultados

En las columnas %R de la tabla 8.1 a la 8.4 se observa la mejora gradual en el reconocimiento total del clasificador basado en perfiles de modelos ocultos de Markov, esto al incrementar la cantidad de símbolos con los que se crean las secuencias. Esto también se manifiesta en la clasificación errónea de las observaciones que se asignan a las localidades vecinas a la localidad correcta. La explicación a este comportamiento es que el cuantizador forma mejores grupos con características más representativas al aumentar la cantidad de centroides.

El porcentaje de detección total de la red neuronal en la tabla de comparación 8.15, muestra como el clasificador basado en modelos ocultos para el caso de 1024 símbolos, es superior a la red neuronal no recurrente, debido a que ésta no toma en cuenta la secuencia temporal ni espacial de descriptores sino solo sus valores, aunque podría reemplazar al clasificador de modelos ocultos en aplicaciones más simples donde no se requiera de un sistema robusto a oclusión o de mucha presición en la etapa de clasificación.

Porcentaje total de detección		
	No. de símbolos	%
Perfil de modelo oculto	16	59.4
	64	68.8
	512	89.1
	1024	100
Red neuronal de propagación hacia adelante	—-	95.7

Tabla 8.15: Tabla de comparación de porcentajes totales de clasificación.

En cuanto a la red recurrente, es la que tiene el modelo más cercano al proceso de emisión y clasificación de características visuales, ya que las clasifica como una señal en el tiempo, según se procesan una a una las entradas en los módulos de detección y descripción. Lamentablemente este es el metódo de clasificación más caro computacionalmente para entrenar, con más de 32 horas de entrenamiento según se reporta en la tabla de comparación 8.16.

Comparación de tiempos promedio		
	Entrenamiento (s)	Detección (ms)
Perfil de modelo oculto (todas las localidades, 1024 símbolos)	42.77 + 657	109.72
Red neuronal de propagación hacia adelante	1477	578
Red tipo NARX	115226	2294

Tabla 8.16: Tabla de comparación de tiempos promedio.

Con lo anterior, el modelo oculto de Markov resulta en una solución intermedia entre considerar las características visuales sin algún orden espacial y en el tiempo, y el solo considerar una secuencia como una señal temporal. La figura 8.2 muestra que a partir de más de 256 simbolos se comporta como un clasificador relativamente bueno con una presición mayor al 80% con aproximadamente 203 segundos consumidos en la cuantización de los vectores como se ve en la tabla 8.6. Los tiempos de filtrado, estimación de puntos característicos y estimación de normales también resultan una desventaja en cuanto a tiempo de ejecución se refiere.

Capítulo 9

Conclusiones y trabajo futuro

En este trabajo se expone que es posible usar modelos ocultos de Markov como herramienta aplicada a la visión por computadora, en particular para la localización espacial de información visual y de forma, que podrá ser utilizada para sistemas de navegación más robustos al ruido y distorsiones de entrada. El enfoque desarrollado en este trabajo fue uno de los mas difíciles debido a la dimensionalidad del descriptor Color SHOT, pero queda justificado al integrar el color de las imágenes en un sistema de percepción más semejante al del ser humano. Este trabajo es una primera aproximación para el manejo de la oclusión usando secuencias de símbolos mediante los perfiles de modelos ocultos de Markov; y a escala y rotación mediante el detector de puntos característicos ISS y el descriptor Color SHOT.

La principal aportación de esta investigación es que usando los perfiles de modelos de ocultos de Markov (PHMM) es posible obtener resultados ligeramente mejores en la clasificación que una red neuronal con una sola capa oculta, ya que los modelos ocultos toman en cuenta la relación espacial y temporal de las observaciones. Esto se observa con toda claridad en la tabla de comparación 8.15 en la cual los resultados de detección van desde el 59.4% hasta el 100%. Los mejores tiempos de entrenamiento y detección igualmente se logran usando modelos ocultos de Markov con 699.77 y 107.72×10^{-3} segundos respectivamente. La desventaja de este enfoque es la selección empírica para los parámetros del modelo.

La selección del tipo de clasificador dependerá de la aplicación particular y de los recursos de cómputo asignados a la misma: si el sistema requiere ser robusto a ruido y oclusión además de que se entrena en la misma plataforma de hardware en la que se prueba, los modelos ocultos de Markov resultan una opción adecuada. Sí es posible entrenar el sistema de clasificación usando recursos externos y en abundancia, como tarjetas gráficas

dedicadas o múltiples procesadores, las redes neuronales son la elección más apropiada ya que la instalación final del sistema será muy compacta y eficiente.

Con todo lo anterior queda demostrado que los modelos ocultos de Markov son igual o más eficientes que los métodos de clasificación basados en redes neuronales con una capa oculta, si se comparan relativamente con igualdad de condiciones de entrenamiento y en pruebas.

9.1. Trabajo futuro

Algunos puntos a mejorar en el presente trabajo están relacionadas con los posibles cambios en cada etapa del sistema, o en las posibles variaciones del método utilizado. Dichas mejoras se listan a continuación:

- Paralelización de la implementación de la biblioteca de modelos ocultos de Markov, además de los detectores y descriptores usados (PCL 1.8 incluye fallos en su implementación en paralelo). La etapa de filtrado bilateral y detección de puntos caracteríticos son las más demandantes en tiempo de cómputo.
- Adecuación del sistema para aplicaciones computacionalmente más ligeras (uso de descriptores menos robustos pero más veloces).
- Implantación del sistema completo en un robot de servicio añadiendo la etapa de navegación.
- Experimentación con diferentes arquitecturas y modelos, como son modelos no homogéneos en el tiempo o mixtos.

Las variaciones en el método descrito pueden ser las siguientes:

- Es posible cambiar de modelo estadístico, usando modelos no paramétricos como son los *modelos ocultos de Markov infinitos* (siglas iHMM, del inglés *infinite Hidden Markov Models*) para la posible implementación de sistemas de reconocimiento no supervisados, como se revisa en [84] siendo el estado del arte revisado hasta Julio del 2016.
- También las redes neuronales pueden entrenarse sin intervención, ya que hay adaptaciones del método llamado *codificación dispersa* (del inglés *sparse coding*) con el fin de extraer características simples de manera autónoma y generar estructuras más complejas, posteriormente se realiza un implementación con una estructura de red neuronal para la clasificación [85].

Apéndice A

Códigos fuente

A.1. Ejemplo del código fuente para la detección de puntos característicos

```
typedef pcl::PointXYZ PointType;
typedef pcl::PointXYZRGB PointTypeColor;
typedef pcl::SHOT1344 ShotColorFeature;
typedef pcl::ReferenceFrame PointRef;

pcl::PointCloud<PointTypeColor>::Ptr cloud(new pcl::PointCloud<PointTypeColor>);
pcl::PointCloud<PointTypeColor>::Ptr cloud_nice(new pcl::PointCloud<PointTypeColor>);
pcl::PointCloud<pcl::PointXYZI>::Ptr cloud_temp(new pcl::PointCloud<pcl::PointXYZI>);
pcl::PointCloud<pcl::PointXYZI>::Ptr cloud_temp2(new pcl::PointCloud<pcl::PointXYZI>);
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_temp3(new pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<PointTypeColor>::Ptr cloud_smooth(new pcl::PointCloud<PointTypeColor>);

if (pcl::io::loadPCDFile<PointTypeColor>(name, *cloud) != 0)
{
   return -1;
}

pcl::PassThrough<PointTypeColor> pass;
pass.setInputCloud (cloud);
pass.setFilterFieldName ("z");
pass.setFilterLimits (0.0, 3.0);//3.0 m
//pass.setFilterLimitsNegative (true);
pass.filter (*cloud_nice);
```

```
cout<<"Distance filter ended"<<endl;

pcl::BilateralFilter<pcl::PointXYZI> bf;
pcl::search::KdTree<pcl::PointXYZI>::Ptr tree_lat (new pcl::search::KdTree<pcl::PointXYZI>);

pcl::PointCloudXYZRGBtoXYZI(*cloud_nice,*cloud_temp);

bf.setInputCloud (cloud_temp);
bf.setSearchMethod (tree_lat);
bf.setHalfSize (0.02f);
bf.setStdDev(0.02f);
bf.filter(*cloud_temp2);

pcl::copyPointCloud(*cloud_temp2,*cloud_temp3);
pcl::copyPointCloud(*cloud_nice,*cloud_smooth);
pcl::copyPointCloud(*cloud_temp3,*cloud_smooth);

//pcl::copyPointCloud(*cloud,*cloud_smooth);

cout<<"Bilateral smoothing ended"<<endl;

// ISS keypoint detector object.
pcl::ISSKeypoint3D<PointTypeColor, PointTypeColor> detector;
pcl::PointCloud<PointTypeColor>::Ptr keypoints(new pcl::PointCloud<PointTypeColor>);
pcl::search::KdTree<PointTypeColor>::Ptr tree_iss(new pcl::search::KdTree<PointTypeColor>);

detector.setInputCloud(cloud_smooth);
detector.setSearchMethod(tree_iss);
double resolution = computeCloudResolution(cloud_smooth);
// Set the radius of the spherical neighborhood used to compute the scatter matrix.
detector.setSalientRadius(6 * resolution);
// Set the radius for the application of the non maxima supression algorithm.
detector.setNonMaxRadius(4 * resolution);

/********With Boundary estimation***/
//  detector.setNormalRadius (4 * resolution);
//   detector.setBorderRadius (4 * resolution);
//   detector.setAngleThreshold (static_cast<float> (M_PI) / 3.0);
detector.setNormalRadius (6 * resolution);
detector.setBorderRadius (4 * resolution);
detector.setAngleThreshold (static_cast<float> (M_PI) / 2.0);
/********************/

// Set the minimum number of neighbors that has to be found while applying the non maxima suppression alg
```

```
detector.setMinNeighbors(5);
// Set the upper bound on the ratio between the second and the first eigenvalue.
detector.setThreshold21(0.975);
// Set the upper bound on the ratio between the third and the second eigenvalue.
detector.setThreshold32(0.975);
// Set the number of prpcessing threads to use. 0 sets it to automatic.
//detector.setNumberOfThreads(8); //Not work. Crashed!!!

detector.compute(*keypoints);

cout<<"ISS ended"<<endl<<" keypoints size: "<<keypoints->size()<<endl;
```

A.2. Ejemplo del código fuente para la descripción de puntos característicos

```
typedef pcl::PointXYZ PointType;
typedef pcl::PointXYZRGB PointTypeColor;
typedef pcl::SHOT1344 ShotColorFeature;
typedef pcl::ReferenceFrame PointRef;

//Compute the normals

pcl::NormalEstimation<PointTypeColor, pcl::Normal> ne;
//ne.setNumberOfThreads(8); //Not work. Crashed!!!
ne.setViewPoint(0,0,0);
ne.setInputCloud (cloud_smooth);
//ne.setSearchSurface(cloud_smooth);

pcl::search::KdTree<PointTypeColor>::Ptr tree2 (new pcl::search::KdTree<PointTypeColor>);
ne.setSearchMethod (tree2);
pcl::PointCloud<pcl::Normal>::Ptr normals_key (new pcl::PointCloud<pcl::Normal>);
ne.setRadiusSearch (0.03);
ne.compute (*normals_key);
std::cout << "Normals ended. "<< std::endl;


pcl::SHOTColorEstimation<PointTypeColor, pcl::Normal, ShotColorFeature,PointRef> shotC_Estimation;
//pcl::SHOTColorEstimation<PointTypeColor, pcl::Normal, ShotColorFeature,PointRef> shotC_Estimation;
pcl::search::KdTree<PointTypeColor>::Ptr tree_shot (new pcl::search::KdTree<PointTypeColor>);

//shotC_Estimation.setNumberOfThreads(8); //Not work. Crashed!!!
shotC_Estimation.setInputCloud(keypoints_z);
```

```
shotC_Estimation.setSearchSurface(cloud_smooth);
shotC_Estimation.setInputNormals(normals_key);

// Use the same KdTree from the normal estimation
shotC_Estimation.setSearchMethod(tree_shot);
pcl::PointCloud<ShotColorFeature>::Ptr shotC_Features(new pcl::PointCloud<ShotColorFeature>);
shotC_Estimation.setRadiusSearch(0.02f);
shotC_Estimation.compute (*shotC_Features);
std::cout << "Color SHOT size: " << shotC_Features->points.size () << std::endl;
std::cout << "Color SHOT estimation ended "<<endl;

pcl::PCDWriter writer;
name=ruta_output+files_local_names.at(i)+"CSHOTserial.pcd";
writer.write(name, *shotC_Features,false);
cout<<"File: "<<name<<" saved"<<endl;

name=ruta_outputkey+files_local_names.at(i)+"keyserial.pcd";
writer.write(name, *keypoints_z,false);
cout<<"File: "<<name<<" saved"<<endl;
```

A.3. Ejemplo de un perfil de modelo oculto de Markov

```
from yahmm import *
model = Model( name="Global Sequence Aligner" )

# Define the distribution for insertions
i_d = DiscreteDistribution( { 'A': 0.25, 'C': 0.25, 'G': 0.25, 'T': 0.25 } )

# Create the insert states
i0 = State( i_d, name="I0" )
i1 = State( i_d, name="I1" )
i2 = State( i_d, name="I2" )
i3 = State( i_d, name="I3" )

# Create the match states
m1 = State( DiscreteDistribution({ "A": 0.95, 'C': 0.01, 'G': 0.01, 'T': 0.02 }) , name="M1" )
m2 = State( DiscreteDistribution({ "A": 0.003, 'C': 0.99, 'G': 0.003, 'T': 0.004 }) , name="M2" )
m3 = State( DiscreteDistribution({ "A": 0.01, 'C': 0.01, 'G': 0.01, 'T': 0.97 }) , name="M3" )

# Create the delete states
d1 = State( None, name="D1" )
d2 = State( None, name="D2" )
d3 = State( None, name="D3" )
```

```
# Add all the states to the model
model.add_states( [i0, i1, i2, i3, m1, m2, m3, d1, d2, d3 ] )

# Create transitions from match states
model.add_transition( model.start, m1, 0.9 )
model.add_transition( model.start, i0, 0.1 )
model.add_transition( m1, m2, 0.9 )
model.add_transition( m1, i1, 0.05 )
model.add_transition( m1, d2, 0.05 )
model.add_transition( m2, m3, 0.9 )
model.add_transition( m2, i2, 0.05 )
model.add_transition( m2, d3, 0.05 )
model.add_transition( m3, model.end, 0.9 )
model.add_transition( m3, i3, 0.1 )

# Create transitions from insert states
model.add_transition( i0, i0, 0.70 )
model.add_transition( i0, d1, 0.15 )
model.add_transition( i0, m1, 0.15 )

model.add_transition( i1, i1, 0.70 )
model.add_transition( i1, d2, 0.15 )
model.add_transition( i1, m2, 0.15 )

model.add_transition( i2, i2, 0.70 )
model.add_transition( i2, d3, 0.15 )
model.add_transition( i2, m3, 0.15 )

model.add_transition( i3, i3, 0.85 )
model.add_transition( i3, model.end, 0.15 )

# Create transitions from delete states
model.add_transition( d1, d2, 0.15 )
model.add_transition( d1, i1, 0.15 )
model.add_transition( d1, m2, 0.70 )

model.add_transition( d2, d3, 0.15 )
model.add_transition( d2, i2, 0.15 )
model.add_transition( d2, m3, 0.70 )

model.add_transition( d3, i3, 0.30 )
model.add_transition( d3, model.end, 0.70 )
```

```
# Call bake to finalize the structure of the model.
model.bake()

for sequence in map( list, ('ACT', 'GGC', 'GAT', 'ACC') ):
    logp, path = model.viterbi( sequence )
    print "Sequence: '{}' -- Log Probability: {} -- Path: {}".format(
        ''.join( sequence ), logp, " ".join( state.name for idx, state in path[1:-1] ) )

#Sequence: 'ACT' -- Log Probability: -0.513244900357 -- Path: M1 M2 M3
#Sequence: 'GGC' -- Log Probability: -11.0481012413 -- Path: I0 I0 D1 M2 D3
#Sequence: 'GAT' -- Log Probability: -9.12551967402 -- Path: I0 M1 D2 M3
#Sequence: 'ACC' -- Log Probability: -5.08795587886 -- Path: M1 M2 M3
```

A.4. Ejemplo de red neuronal de propagación hacia adelante

```
from keras.models import Sequential
from keras.layers import Dense
import numpy
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load dataset
dataset = numpy.loadtxt("freeburg_xyz.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:1344]
Y = dataset[:,1344]
# create model
model = Sequential()
model.add(Dense(50, input_dim=1344, init='uniform', activation='linear'))
model.add(Dense(50, init='uniform', activation='tanh'))
model.add(Dense(6, init='uniform', activation='softmax'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, nb_epoch=150, batch_size=10)
# evaluate the model
scores = model.evaluate(X, Y)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

A.5. Ejemplo de red neuronal NARX

```
from pybrain.structure import RecurrentNetwork
from pybrain.supervised.trainers import BackpropTrainer
from pybrain.datasets import SupervisedDataSet

n = RecurrentNetwork()

n.addInputModule(LinearLayer(1344, name='in'))
n.addInputModule(LinearLayer(1344, name='in_d1'))
n.addInputModule(LinearLayer(1344, name='in_d2'))

n.addModule(TanhLayer(50, name='hidden'))

n.addOutputModule(LinearLayer(6, name='out'))
n.addOutputModule(LinearLayer(6, name='out_d1'))
n.addOutputModule(LinearLayer(6, name='out_d2'))

n.addConnection(FullConnection(n['in'], n['hidden'], name='c1'))
n.addConnection(FullConnection(n['hidden'], n['out'], name='c2'))

#n.addRecurrentConnection(FullConnection(n['out'], n['hidden'], name='c3'))

n.addRecurrentConnection(FullConnection(n['out'], n['out_d1'], name='c4'))
n.addRecurrentConnection(FullConnection(n['out_d1'], n['hidden'], name='c5'))

n.addRecurrentConnection(FullConnection(n['out_d1'], n['out_d2'], name='c6'))
n.addRecurrentConnection(FullConnection(n['out_d2'], n['hidden'], name='c7'))

n.addRecurrentConnection(FullConnection(n['in'], n['in_d1'], name='c8'))
n.addRecurrentConnection(FullConnection(n['in_d1'], n['hidden'], name='c9'))

n.addRecurrentConnection(FullConnection(n['in_d1'], n['in_d2'], name='c10'))
n.addRecurrentConnection(FullConnection(n['in_d2'], n['hidden'], name='c11'))

n.sortModules()

ds = SupervisedDataSet(Inputs,Targets) #inputs must be normalized in [-1,1]

trainer = BackpropTrainer(n, ds)
trainer.train()
```

Bibliografía

[1] J. Engelberger, *Robotics in Service*. MIT Press, 1989.

[2] K. H. Park, H. E. Lee, Y. Kim, and Z. Z. Bien, "A steward robot for human-friendly human-machine interaction in a smart house environment," *IEEE Transactions on Automation Science and Engineering*, vol. 5, pp. 21–25, Jan 2008.

[3] R. Aracil, C. Balaguer, and M. Armada, "Robots de servicio," *Revista Iberoamericana de Automática e Informática Industrial RIAI*, vol. 5, no. 2, pp. 6 – 13, 2008.

[4] P. G. Backes, K. S. Tso, J. S. Norris, and R. Steinke, "Group collaboration for mars rover mission operations," in *Proceedings of the 2002 IEEE International Conference on Robotics and Automation, ICRA 2002, May 11-15, 2002, Washington, DC, USA*, pp. 3148–3154, 2002.

[5] D. Montemello, *The Cambridge Handbook of Visuospatial Thinking*, ch. Navigation. Cambridge University Press, 2005.

[6] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: A survey," *Artif. Intell. Rev.*, vol. 43, pp. 55–81, Jan. 2015.

[7] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, (San Francisco, CA, USA), pp. 674–679, Morgan Kaufmann Publishers Inc., 1981.

[8] S. D. Jones, C. Andresen, and J. L. Crowley, "Appearance based process for visual navigation," in *Intelligent Robots and Systems, 1997. IROS '97., Proceedings of the 1997 IEEE/RSJ International Conference on*, vol. 2, pp. 551–557 vol.2, Sep 1997.

[9] L. Wang, H. Gao, and Z. Cai, "Topological mapping and navigation for mobile robots with landmark evaluation," in *2009 International Conference on Information Engineering and Computer Science*, pp. 1–5, Dec 2009.

[10] J. Kosecka and F. Li, "Vision based topological markov localization," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 2, pp. 1481–1486 Vol.2, April 2004.

[11] D. F. Wolf, G. S. Sukhatme, D. Fox, and W. Burgard, "Autonomous terrain mapping and classification using hidden markov models," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 2026–2031, April 2005.

[12] Y. B. Park and I. H. Suh, "Visual feature extraction for recognition of types of corridor segments under partial occulusion," in *2010 2nd IEEE InternationalConference on Network Infrastructure and Digital Content*, pp. 383–387, Sept 2010.

[13] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[14] J. Otepka, S. Ghuffar, C. Waldhauser, R. Hochreiter, and N. Pfeifer, "Georeferenced point clouds: A survey of features and point cloud management," *ISPRS International Journal of Geo-Information*, vol. 2, no. 4, p. 1038, 2013.

[15] F. Stanco, S. Battiato, and G. Gallo, *Digital Imaging for Cultural Heritage Preservation: Analysis, Restoration, and Reconstruction of Ancient Artworks*. Boca Raton, FL, USA: CRC Press, Inc., 1st ed., 2011.

[16] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[17] J. Giles, "Inside the race to hack the kinect," *New Scientist*, vol. 208, no. 2789, pp. 22 – 23, 2010.

[18] "Openkinect project." https://github.com/OpenKinect. Accesado: 2016-10-20.

[19] G. Bradski *Dr. Dobb's Journal of Software Tools*.

[20] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[21] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), May 9-13 2011.

[22] L. Song, W. Wu, J. Guo, and X. Li, "Survey on camera calibration technique," in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2013 5th International Conference on*, vol. 2, pp. 389–392, Aug 2013.

[23] J. Herraéz, J. L. Denia, P. Navarro, J. Rodriguez, and M. T. Martin, "Stereoscopic vision through epipolarization without orientation parameters," in *2011 18th IEEE International Conference on Image Processing*, pp. 981–984, Sept 2011.

[24] J. Geng, "Structured-light 3d surface imaging: a tutorial," *Adv. Opt. Photon.*, vol. 3, pp. 128–160, Jun 2011.

[25] L. Zhang, B. Curless, and S. M. Seitz, "Rapid shape acquisition using color structured light and multi-pass dynamic programming," in *The 1st IEEE International Symposium on 3D Data Processing, Visualization, and Transmission*, pp. 24–36, June 2002.

[26] C. V. Nguyen, S. Izadi, and D. Lovell, "Modeling kinect sensor noise for improved 3d reconstruction and tracking," in *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission*, pp. 524–530, Oct 2012.

[27] "Primesense patent application us 2010/0290698," 2010.

[28] "Ir pattern." https://bbzippo.wordpress.com/2010/11/28/kinect-in-infrared/. Accesado: 2016-10-20.

[29] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing (3rd Ed.): Principles, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[30] D. J. Natale, M. S. Baran, and R. L. Tutwiler, "Point cloud processing strategies for noise filtering, structural segmentation, and meshing of ground-based 3d flash lidar images," in *2010 IEEE 39th Applied Imagery Pattern Recognition Workshop (AIPR)*, pp. 1–8, Oct 2010.

[31] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, (Washington, DC, USA), pp. 839–, IEEE Computer Society, 1998.

[32] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, "A gentle introduction to bilateral filtering and its applications," in *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, (New York, NY, USA), ACM, 2007.

[33] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," *International Journal of Computer Vision*, vol. 81, no. 1, pp. 24–52, 2009.

[34] T. Q. Pham and L. J. van Vliet, "Separable bilateral filtering for fast video preprocessing," in *2005 IEEE International Conference on Multimedia and Expo*, pp. 4 pp.–, July 2005.

[35] O. Miksik and K. Mikolajczyk, "Evaluation of local detectors and descriptors for fast feature matching," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pp. 2681–2684, Nov 2012.

[36] A. Canclini, M. Cesana, A. Redondi, M. Tagliasacchi, J. Ascenso, and R. Cilla, "Evaluation of low-complexity visual feature detectors and descriptors," in *2013 18th International Conference on Digital Signal Processing (DSP)*, pp. 1–7, July 2013.

[37] B. Moorfield, R. Haeusler, and R. Klette, "Bilateral filtering of 3d point clouds for refined 3d roadside reconstructions," in *Proceedings, Part II, of the 16th International Conference on Computer Analysis of Images and Patterns - Volume 9257*, CAIP 2015, (New York, NY, USA), pp. 394–402, Springer-Verlag New York, Inc., 2015.

[38] H. Fan, Q. Peng, and Y. Yu, "A robust high-resolution details preserving denoising algorithm for meshes," *Science China Information Sciences*, vol. 56, no. 9, pp. 1–12, 2013.

[39] J. Li and R. Wang, "Robust denoising of point-sampled surfaces," *W. Trans. on Comp.*, vol. 8, pp. 153–162, Jan. 2009.

[40] Y. Zhong, "Intrinsic shape signatures: A shape descriptor for 3d object recognition," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pp. 689–696, Sept 2009.

[41] B. K. P. Horn, ed., *Robot Vision*. Cambridge, MA, USA: MIT Press, 1986.

[42] L. Alexandre, "A comparative evaluation of 3d keypoint detectors in a rgb-d object dataset," in *International Conf. on Computer Vision Theory and Applications - VISAPP*, vol. 1, pp. 1–8, January 2014.

[43] F. Tombari, S. Salti, and L. Di Stefano, "Unique signatures of histograms for local surface description," in *Proceedings of the 11th European Conference on Computer Vision Conference on Computer Vision: Part III*, ECCV'10, (Berlin, Heidelberg), pp. 356–369, Springer-Verlag, 2010.

[44] R. Bro, E. Acar, and T. G. Kolda, "Resolving the sign ambiguity in the singular value decomposition," *Journal of Chemometrics*, vol. 22, pp. 135–140, 2008.

[45] L. A. Alexandre, "Set distance functions for 3d object recognition," in *Proceedings, Part I, of the 18th Iberoamerican Congress on Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications - Volume 8258*, CIARP 2013, (New York, NY, USA), pp. 57–64, Springer-Verlag New York, Inc., 2013.

[46] F. Tombari, S. Salti, and L. D. Stefano, "A combined texture-shape descriptor for enhanced 3d feature matching," in *2011 18th IEEE International Conference on Image Processing*, pp. 809–812, Sept 2011.

[47] G. Sharma, W. Wu, and E. N. Dalal, "The CIEDE2000 color-difference formula: implementation notes, supplementary test data, and mathematical observations," *Color research and application*, vol. 30, no. 1, pp. 21–30, 2005.

[48] J. K. Lawder and P. J. H. King, *Using Space-Filling Curves for Multi-dimensional Indexing*, pp. 20–35. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000.

[49] M. Connor and P. Kumar, "Fast construction of k-nearest neighbor graphs for point clouds," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, pp. 599–608, July 2010.

[50] P. Cheeseman, M. Self, J. Kelly, W. Taylor, D. Freeman, and J. Stutz, "Bayesian classification," in *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence*, AAAI'88, pp. 607–611, AAAI Press, 1988.

[51] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, pp. 264–323, Sept. 1999.

[52] G. J. Woeginger, "Exact algorithms for np-hard problems: A survey," *Combinatorial Optimization - Eureka, You Shrink!, LNCS*, pp. 185–207.

[53] S. Lloyd, "Least squares quantization in pcm," *IEEE Trans. Inf. Theor.*, vol. 28, pp. 129–137, Sept. 2006.

[54] R. E. Bellman, *Adaptive control processes - A guided tour*. Princeton, New Jersey, U.S.A.: Princeton University Press, 1961.

[55] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?," in *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings.*, pp. 217–235, 1999.

[56] A. Hinneburg, D. A. Keim, and C. C. Aggarwal, "What is the nearest neighbor in high dimensional spaces?," in *Proc. 26th Int. Conf. on Very Large Databases, Cairo, Egypt*, Sept. 2000.

[57] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, *On the Surprising Behavior of Distance Metrics in High Dimensional Space*, pp. 420–434. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[58] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, (Philadelphia, PA, USA), pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[59] C. Elkan, "Using the triangle inequality to accelerate *k*-means," in *Proc. ICML*, 2003.

[60] G. Hamerly and J. Drake, *Accelerating Lloyd's Algorithm for k-Means Clustering*, pp. 41–78. Cham: Springer International Publishing, 2015.

[61] J. Newling and F. Fleuret, "Fast k-means with accurate bounds," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 936–944, 2016.

[62] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, pp. 209–226, Sept. 1977.

[63] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *In VISAPP International Conference on Computer Vision Theory and Applications*, pp. 331–340, 2009.

[64] D. Jurafsky and J. H. Martin, *Speech and Language Processing (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009.

[65] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[66] L. Rabiner and B. Juang, "An introduction to hidden markov models," *IEEE ASSP Magazine*, vol. 3, pp. 4–16, Jan 1986.

[67] A. Ahmad and R. Wells, "Putting logic in modeling of biological neuron: A new framework," in *Proceedings of the 16th Communications & Networking Symposium*, CNS '13, (San

Diego, CA, USA), pp. 17:1–17:7, Society for Computer Simulation International, 2013.

[68] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[69] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[70] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, vol. 4, pp. 251–257, Mar. 1991.

[71] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research," ch. Learning Representations by Back-propagating Errors, pp. 696–699, Cambridge, MA, USA: MIT Press, 1988.

[72] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015.

[73] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, pp. 1550–1560, Oct 1990.

[74] C. Raposo, J. P. Barreto, and U. Nunes, "Fast and accurate calibration of a kinect sensor," in *Proceedings of the 2013 International Conference on 3D Vision*, 3DV '13, (Washington, DC, USA), pp. 342–349, IEEE Computer Society, 2013.

[75] K. Klasing, D. Althoff, D. Wollherr, and M. Buss, "Comparison of surface normal estimation methods for range sensing applications," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pp. 3206–3211, May 2009.

[76] S. K. Joshi and S. Machchhar, "An evolution and evaluation of dimensionality reduction techniques; a comparative study," in *Computational Intelligence and Computing Research (ICCIC), 2014 IEEE International Conference on*, pp. 1–5, Dec 2014.

[77] J. Schreiber and K. Karplus, "Analysis of nanopore data using hidden markov models," *Bioinformatics*, vol. 31, no. 12, p. 1897, 2015.

[78] R. B. Lyngsø and C. N. Pedersen, "Complexity of comparing hidden markov models," in *Proceedings of the 12th International Symposium on Algorithms and Computation*, ISAAC '01, (Berlin, Heidelberg), pp. 416–428, Springer-Verlag, 2001.

[79] *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, (New York, NY, USA), ACM, 2013.

[80] F. Chollet, "Keras: A Python neural networks library." https://github.com/fchollet/keras, 2015.

[81] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[82] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[83] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber, "PyBrain," *Journal of Machine Learning Research*, vol. 11, pp. 743–746, 2010.

[84] A. Saeedi, M. D. Hoffman, M. J. Johnson, and R. P. Adams, "The segmented ihmm: A simple, efficient hierarchical infinite HMM," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 2682–2691, 2016.

[85] H. Wang, F. Nie, and H. Huang, "Robust and discriminative self-taught learning," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)* (S. Dasgupta and D. Mcallester, eds.), vol. 28, pp. 298–306, JMLR Workshop and Conference Proceedings, May 2013.