UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN INGENIERÍA ELÉCTRICA

FACULTAD DE INGENIERÍA

CAMPO DE CONOCIMIENTO: PROCESAMIENTO DIGITAL DE SEÑALES

LOCALIZACIÓN DE UN ROBOT MÓVIL UTILIZANDO MODELOS OCULTOS DE MARKOV

TESIS

QUE PARA OPTAR POR EL GRADO DE

MAESTRO EN INGENIERÍA ELÉCTRICA

PRESENTA:

OSCAR FUENTES CASARRUBIAS

Director de tesis:

Dr. Jesús Savage Carmona

Facultad de Ingeniería

Ciudad Universitaria , Cd.Mx. Junio 2018

Agradecimientos

Agradezco a mi familia y a mi novia antes que nada, siempre me apoyaron y cargaron conmigo las cargas de esta aventura académica. A la UNAM , por ser una institución de las pocas que funcionan en un país tan disfuncional. A la Facultad de Ingeniería y el posgrado, al Conacyt por su apoyo indispensable. A los profesores , y en particular al Dr. Savage y el laboratorio de Bio Robótica. A turtlebot

Resumen

El presente trabajo muestra un sistema de localización y mapeo de un robot móvil autónomo. Se analizan técnicas de procesamiento digital de señales tradicionales, y se las compara con algunas alternativas de tipo machine learning y big data. Se propone un sistema basado en Modelos Ocultos de Markov (HMM) para resolver el problema de SLAM, con un robot omnidireccional equipado con un sensor láser tipo RangeFinder. Se presentan algunas alternativas en la implementación de los mismos y se plantean cuestionamientos respecto a los efectos de algunos parámetros del sistema en su desempeño final.

Con intención de comparar se comenta brevemente los métodos utilizados hoy en día con uno de los sistemas de comunicación más empleados en el ambiente de la robótica ROS.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

> Supuestamente el cerebro humano es algo parecido a una libreta que se adquiere en la papelería: muy poco mecanismo y muchas hojas en blanco.
>
> Alan Turing

Los robots han impactado de forma considerable a la humanidad desde el inicio de la revolución industrial, realizando labores repetitivas de una forma mucho más eficiente (y económica) que los obreros humanos. Ya sean robots exploradores que pueden operar en condiciones no aptas para los humanos, o brazos mecánicos capaces de fuerza y precisión. La idea de una máquina inteligente capaz de realizar tareas por nosotros, siempre ha seducido la imaginación humana. Pero no fue sino hasta principios del siglo pasado que el termino robot adquirió el significado que conocemos hasta nuestros días. Originalmente acuñado por el escritor Carel Kapek [2] en su obra de teatro R.U.R. Robots Universales Rossum, la palabra robota significaba obrero forzado en la lengua eslava.Posteriormente en la década de los 80's, la robótica es definida como la ciencia que estudia conexión inteligente entre percepción y acción.

Percepción del entorno recolectando información por medio de sensores. Esta información es procesada (inteligencia), mediante operaciones computacionales, que finalmente permiten realizar una acción.

Esta acción puede ser de un tipo estática (como por ejemplo brazos actuadores en la industria) o, proveyendo al sistema con algún método de locomoción, móvil.

En general se consideran tres grandes grupos o tipos de robots:

1. Robots Fijos.
2. Robots Médicos.
3. Robots Móviles.

En la figura 1.1 se muestra un brazo robótico típico, la figura 1.2 muestra prótesis inteligentes, es decir,robots médicos. Y finalmente la figura 1.3 muestra a Justina, el robot de servicio utilizado en el laboratorio de bio-robótica en la facultad de ingeniería de la UNAM

Figura 1.1: Brazo Robótico para la Industria.

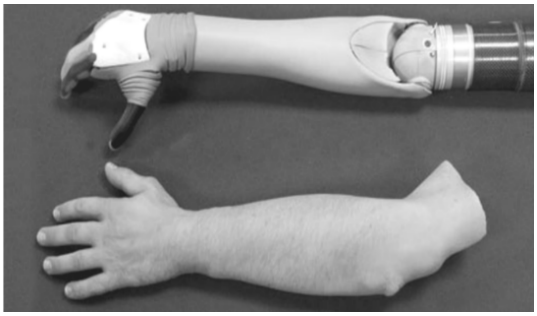


Figura 1.2: Robot Médico.

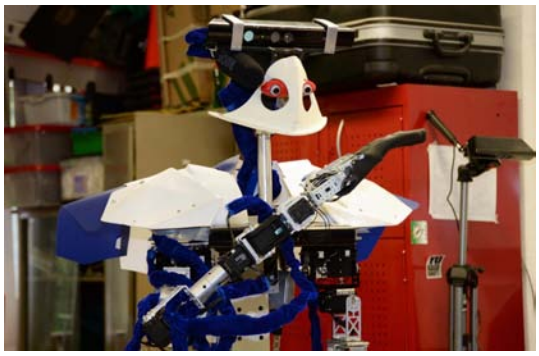


Figura 1.3: Robot de servicio Justina.

1.1. Justificación

La localización y mapeo simultáneo, o SLAM por sus siglas en inglés (Simultaneous Localization and Mapping), es una tarea obligada para cualquier agente móvil al que pueda llamarse verdaderamente autónomo. La capacidad de encontrarse en una posición desconocida, en un lugar desconocido, y construir un mapa dado un conjunto de observaciones y luego usar este mapa de manera simultánea para navegar, haría al robot autónomo [15].

El método más comúnmente utilizado para solucionar los llamados algoritmos SLAM utiliza el filtro de Kalman[24]. La principal ventaja es que provee formas iterativas de resolver el problema de navegación. Existen muchos otros métodos menos estudiados para realizar esta tarea de localización y navegación, usando diferentes enfoques y tipos de observaciones, principalmente basados en visión computacional.

El objetivo de este trabajo es el de mostrar la efectividad de los Modelos Ocultos de Markov (HMM) para localizar a un robot en un ambiente desconocido utilizando un sensor de tipo RangeFinder como generador de las observaciones.Los algoritmos utilizados para "leer" los mapas generados con modelos HMM son más económicos que los filtros de partículas. Además existen procesadores especializados (DSP) como la familia de DSP's C5x deTexasInstruments capaces de correr estos algoritmos a gran velocidad y en paralelo [11].

1.2. Definición del Problema

Un robot móvil, como su nombre indica, es aquel que es capaz de navegar el espacio donde se encuentra. Debe considerarse que el entorno donde un robot de servicio puede trabajar no necesariamente es un ambiente estructurado (a diferencia de los robots industriales), por lo que métodos de autonomía son indispensables para que en un futuro un robot pueda interactuar, de forma segura,con humanos en los ambientes donde típicamente habitamos. Esta autonomía implica trabajar con altos niveles de incertidumbre, que emana de los ambientes a navegar, los objetos con que interactúa, y el ruido inherente a los procesos de medición y lectura de los sensores.Una conducta reactiva que trabaja directamente sobre los sensores no resulta suficiente para la navegación, pues los pequeños errores en cada lectura (normalmente despreciables) se acumulan en cada medición, incrementando la incertidumbre de la posición. Por eso necesidad de una inteligencia artificial que permita extraer información relevante de los sensores,para navegar el entorno y almacenar esta información en un mapa.

Existen varias formas de representar el mapa, o modelo. Las figuras 1.4 , 1.5 , 1.6 y 1.7 muestran algunos ejemplos de estos.

- Mapas Topológicos

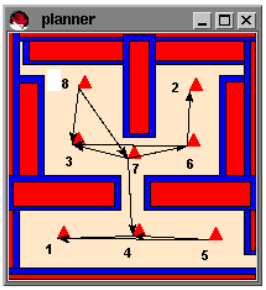
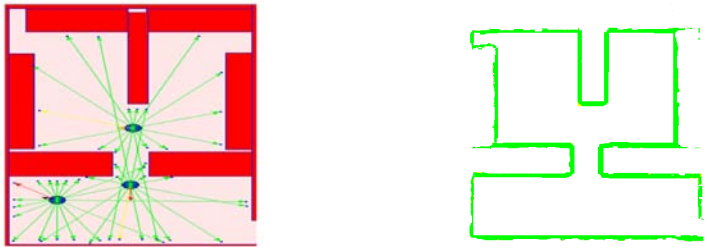


Figura 1.4: Mapa topológico.

- Mapas Raw
- Rejilla o celdas
- Simbólico



(a) Sensores (b) Mapa de lecturas sin procesar

Figura 1.5: Mapas de lecturas sin procesar(RAW)

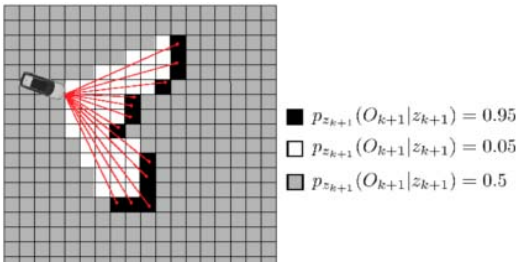


Figura 1.6: Rejilla de Ocupación.[16]

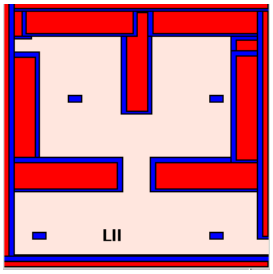


Figura 1.7: Mapa de representación simbólica.

Una vez que se cuenta con un mapa apropiado de la región, ahora se requiere navegarlo. Un robot de servicio, dada una coordenada de origen (o estado inicial del robot) debe encontrar el punto al que debe desplazarse, así como todos los puntos intermedios que componen la ruta de navegación, para realizar la tarea encomendada (SLAM) [15] [14] utilizando información a-priori del ambiente a navegar.

1.3. Hipótesis

Las observaciones generadas por un sensor de un robot dada su posición cumplen con la propiedad de Markov [5]. Por lo que es posible generar un modelo estocástico capaz de describir el fenómeno observado.

Utilizando los algoritmos propios para el estudio de los Modelos Ocultos de Markov(HMM) es posible extraer la información necesaria para navegar, de manera autónoma, un entorno.

1.4. Objetivo

Implementar una solución al problema de SLAM en los robots móviles autónomos, mediante el uso de Modelos Ocultos de Markov(HMM).

Con este fin se divide la tarea en varias partes.

- Generar un mapa, o modelo(HMM). Un modelo estocástico capaz de relacionar las observaciones con variables de localización.
- Proponer un sistema para navegar dichos modelos HMM.
- Probar el método en un robot real (TurtleBot)

1.5. Metodología Propuesta

El agente móvil navegará de forma autónoma un ambiente, almacenando el conjunto de observaciones. El robot navega mientras las observaciones obtenidas son almacenadas, cuantizadas y procesadas para construir un modelo estocástico del ambiente (HMM Modelo Oculto de Markov). La figura 1.8 muestra un diagrama de un ejemplo de un HMM. Las variables en la figura son: X para el estado a y b son probabilidades de transición y emisión respectivamente y y las observaciones. Éste permite calcular de forma eficiente (mediante el algoritmo de Viterbi) la probabilidad de que el robot se encuentre en una región en particular del mundo mapeado, en otras palabras los últimos n estados del sistema.

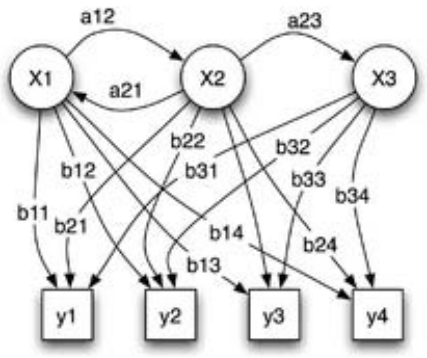


Figura 1.8: Modelo Oculto de Markov (HMM).

En este trabajo se muestra la capacidad de un robot móvil de ubicarse dentro de un entorno dado que se tiene un Modelo (HMM) del mundo en que navega. Así como el método de generación de dicho modelo dado un conjunto de observaciones previas del ambiente a modelar.

Se presenta un sistema que va desde el muestreo de un láser (range finder), para generar vectores de entrenamiento, cuantización vectorial, que mapea una secuencia de observaciones continuas en una digital. La construcción de un modelo probabilístico para finalmente localizar en tiempo real el estado más probable dentro del Modelo Oculto de Markov generado dadas las últimas n observaciones. De una forma similar que el filtro de Kalman, estima la posición de acuerdo a las últimas muestras. Para el caso estudiado, el estado del sistema es la información de navegación.

1.6. Estructura de la Tesis

El presente trabajo está dividido en capítulos:

- *Capítulo 1. Introducción.* Donde se justifica la relevancia de esta investigación. Se hace un planteamiento general de la problemática a resolver. Se establecen hipótesis y objetivos para finalmente se proponer una metodología de solución.

- *Capítulo 2. Marco Teórico* Se plantean los antecedentes teóricos utilizados en el problema estudiado, así como las soluciones más utilizadas al mismo.

- *Capítulo 3. Modelos de Markov.* Se definen formalmente los Modelos Ocultos de Markov, tema central en este trabajo.
- *Capítulo 4. Sistema Propuesto* Se describe el sistema propuesto y los procesos involucrados en su funcionamiento.
- *Capítulo 5 Experimentos* Se describen resultados obtenidos,tanto en simulación como en el robot real.
- *Apéndices*

Capítulo 2

Marco Teórico

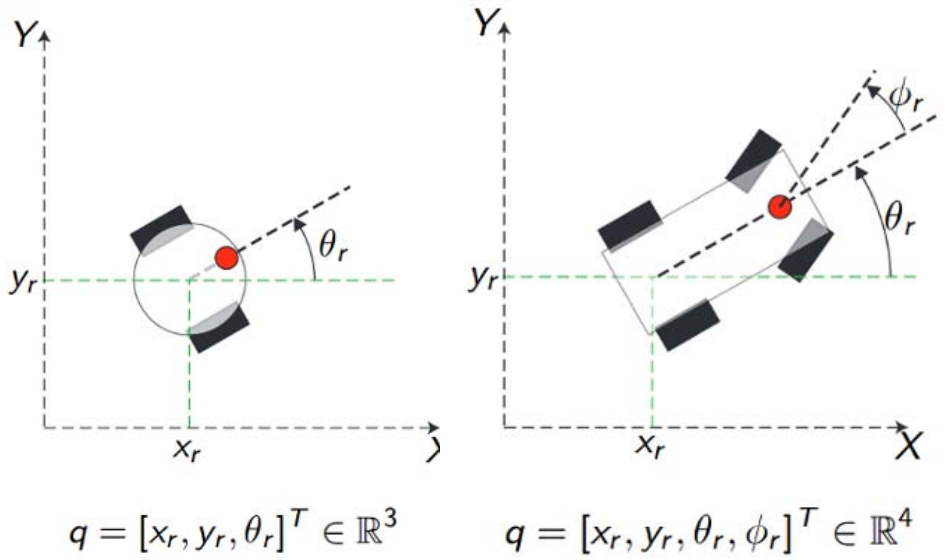
> La pregunta de si un computador puede pensar no es más interesante que la pregunta de si un submarino puede nadar.
>
> Edsger Dijkstra

2.1. Robot Móvil

Un robot móvil es aquel agente equipado con algún tipo de sensor e inteligencia. Es además capaz de desplazarse de un punto a otro en un entorno. Típicamente un software controla los actuadores responsables del movimiento, mientras interpreta lecturas físicas del entorno captadas por algún sensor.

Infinidad de modelos de locomoción han sido estudiados, con mayor o menor complejidad, cada uno apto para el tipo de tarea a desempeñar. La rueda sin lugar a dudas es la solución más simple. Existen ya modelos de estos métodos. El espacio de parámetros que se define por el estado interno del modelo del robot varía con cada configuración, y su dimensionalidad también, un par de ejemplos pueden observarse en la figura 2.1.Esta es la principal diferencia entre móviles y por ende una forma adecuada de clasificación.

Un robot móvil puede ser considerado autónomo o asistido, dependiendo de su capacidad de navegación y localización en el entorno donde se encuentra. El cam-



(a) modelo omnidireccional (b) modelo Ackerman

Figura 2.1: Algunos Modelos y sus espacios paramétricos.

po de la robótica móvil está más activo que nunca, y las técnicas de inteligencia artificial siguen mejorando los algoritmos de control y navegación. Sin embargo, el esquema general consta de dos etapas; un planeador de acciones (alto nivel) que define objetivos, destinos trayectoria según la tarea a realizar y otro de movimientos que convierte información cruda de los sensores y de control en movimiento (instrucciones de control a los actuadores).

El problema de navegación entonces es, dada una observación y modelo de robot, qué señal de control debe generarse para llegar al punto del mapa que el controlador de tareas solicita.

2.2. Sensores

En este trabajo se utilizan sensores del tipo Láser range finder que es un dispositivo capaz de medir la distancia utilizando pulsos láser y la información obtenida al comparar el pulso enviado con el recibido, después de reflejarse en una superficie.

En particular se utilizó un sensor modelo Hokuyo URG-04LX-UG01, mostrado en la figura 2.2, capaz de medir las distancias en un arco de 240 grados, con una

resolución angular de 0.35° y espacial de 5.6 metros. Peso menor a los 200 gramos, lo que lo hace ideal para aplicaciones de robótica móvil. El fabricante estima un error de medición de ± 0.3mm.

Los vectores de observaciones $\vec{o}$ se forman por un número finito de estas distancias en cada tiempo t, el vector será de dimensión conocida, dada por el número de mediciones recolectadas del sensor.

Figura 2.2: Sensor Hokuyo laser range finder.

2.3. Cuantización Vectorial-Algoritmo Linde Buzo Gray

"La cuantización vectorial es un sistema que mapea una secuencia de observaciones (continua o discreta), en una secuencia digital, apta para almacenamiento o transmisión en un canal digital" [8]

El sistema funciona dividiendo un conjunto de vectores de observaciones $\vec{o}$, en subconjuntos de aproximadamente el mismo número de integrantes. Cada conjunto es representado con el centroide del mismo. Aunque existen varios algoritmos para implementarlo, a continuación se detalla el algoritmo usado: Linde Buzo Gray (LGB) [13].

El algoritmo LGB, es un método iterativo que resuelve ambos criterios de optimización de la cuantización vectorial. Funciona primero calculando un codebook inicial, formado por el promedio de todos los vectores a cuantizar, C_0 Se divide en dos centroides C_1 y C_2 al perturbar el centroide dado un valor ϵ cercano a cero. Y se procede iterativamente, después de duplica ese par de centroides hasta obtener el número deseado de centroides. La figura 2.3 muestra un conjunto de coordenadas aleatorias que son cuantizadas utilizando 16 centroides encontrados con este algoritmo.

1. Centroide Inicial

$$\vec{C_1} = \frac{1}{N}\sum_{j=1}^{N}\vec{x_j} \tag{2.1}$$

Donde N es el número de vectores.

2. Perturbar para generar dos centroides nuevos

$$\vec{C_{i+1}} = \vec{C_i}(1+\epsilon) \tag{2.2}$$

$$\vec{C_{i+2}} = \vec{C_i}(1-\epsilon) \tag{2.3}$$

3. Agrupar los vectores en la región con el centroide más cercano.

$$d_1 = D(\vec{x_j}, \vec{C_{i+1}}) \tag{2.4}$$

$$d_2 = D(\vec{x_j}, \vec{C_{i+2}}) \tag{2.5}$$

siendo D la distancia entre $\vec{x}$ y $\vec{C}$

Si

$$d_1 < d_2 \vec{x_j} \rightarrow C_{i+1} \tag{2.6}$$

Si

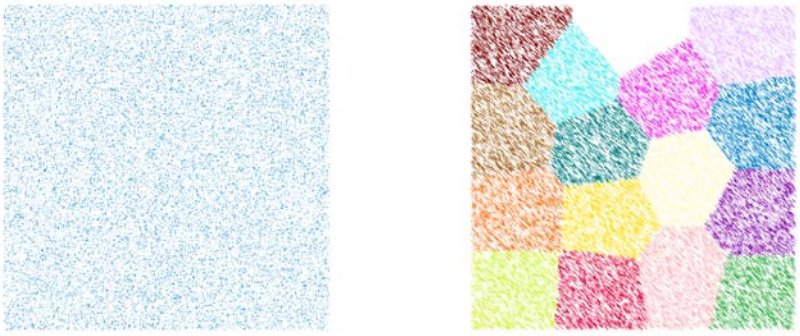
$$d_1 > d_2 \vec{x_j} \rightarrow C_{i+2} \tag{2.7}$$

$$D_g^t = D_g^t min(d_1, d_2) \tag{2.8}$$

iterar mientras

$$D_g^t - D_g^{t-1} \geq \epsilon \tag{2.9}$$

recalcular centroides con los vectores agrupados en cada región.



(a) Conjunto de puntos de entrada. (b) 16 centroides.

Figura 2.3: Ejemplo cuantización vectores 2D utilizando LGB

2.4. Filtro de Kalman

El filtro de Kalman, es un algoritmo recursivo propuesto por Rudolf Kalman en 1960 [20]. Este algoritmo ha sido ampliamente utilizado en diferentes aplicaciones relacionadas a la navegación, desde misiones espaciales hasta vehículos de minería o submarinos. Por lo que toda una familia de algoritmos de control, navegación y localización está basada en él. En el campo de control es tan ampliamente utilizado que toda una rama de control está basada en este tipo de sistemas.

En su forma más simple este algoritmo permite estimar, de manera recursiva, el estado interno de un sistema dinámico lineal a partir de una serie de observaciones sujetas a ruido. Si bien se le llama filtro, es más correcto llamarle estimador. Dado el conjunto de las últimas n observaciones se busca estimar el estado actual.

El sistema dinámico que representa el agente móvil se modela como sigue:

$$x_n = Ax_{n-1} + Bu_n + w_{n-1} \tag{2.10}$$

donde x representa el estado del sistema, para el caso de la navegación.

$$x_n = \begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} \tag{2.11}$$

A es el modelo de transición según el último estado, B representa el modelo de control, u_n es la señal de control y w ruido del proceso, que se supone gaussiano.

$$o_n = Hx_n + v_n \tag{2.12}$$

La observación o_n relaciona el modelo de observaciones H, con el estado x_n y un ruido de observación v_n que igualmente se supone gaussiano.

El algoritmo de Kalman funciona en dos partes; predicción y corrección.

- Predicción:

$$\hat{x}_n^- = A\hat{x}_{n-1} + Bu_n \tag{2.13}$$

$$P_n^- = AP_{n-1}A^T + Q \tag{2.14}$$

  Donde P es la matriz de la covarianza de los errores a-posteriori de la estimación. Q y R las matrices de covarianza del ruido de observaciones y proceso respectivamente. El cálculo de estas matrices no es trivial, y existen enfoques varios para el mismo. La utilizada en la mayoría del software disponible es la de ALS (auto covariance least-squares).

- Corrección:

$$K_n = P_n^- H^T (HP_n^- H^T + R)^{-1} \tag{2.15}$$

$$\hat{x}_n = \hat{x}_n^- + K_n(o_n - H\hat{x}_n^-) \tag{2.16}$$

$$P_n = (I - K_n H)P_n^- \tag{2.17}$$

Para la aplicación de SLAM se utiliza la versión extendida del filtro EKF (extended Kalman filter). Que funciona de una forma similar, con la salvedad que puede tratar con modelos no lineales, al precio de no resultar la estimación óptima. En una forma parecida a las series de Taylor, es posible "linearizar" la estimación alrededor de la observación actual utilizando derivadas parciales (muy útiles para modelar dinámica del móvil).

Se asume entonces que los estados x_n permanecen iguales pero ahora el sistema no es lineal, razón por la que igualmente se considera que el proceso es no lineal, y es expresado de la forma siguiente:

$o_n = \mathrm{h}(x_k,\, o_k)$.

w y v, siguen representando el ruido de proceso y medición. De la misma forma que en el filtro de Kalman, el EKF tiene ecuaciones de predicción y de corrección:

- Predicción:

$$\hat{x}_n^- = f(\hat{x}_{n-1}, u_n, 0) \tag{2.18}$$

$$P_n^- = A_n P_{n-1} A_n^T + W_n Q_{n-1} Q_n^T \tag{2.19}$$

  Y conservando las propiedades del filtro de Kalman estas ecuaciones proyectan el estado y la covarianza del paso anterior (n-1) al tiempo actual n.

- Correción:

$$K_n = P_n^- H_n^T (H_n P_n^- H_n^T + V_n R_n H_n^T)^{-1} \tag{2.20}$$

$$\hat{x}_n = \hat{x}_n^- + K_n(o_n - h(\hat{x}_n^-, 0)) \tag{2.21}$$

$$P_n = (I - K_n H_n) P_n^- \tag{2.22}$$

2.5. Navegación

La navegación es el problema que busca llevar un agente móvil de un punto a otro, según las instrucciones de un planeador de acciones. Diversos enfoques se han sugerido para resolver el problema, y pueden clasificarse según su funcionamiento en dos grandes grupos, navegación con mapa y sin mapa. A los que utilizan mapa se les llama de planificación global, a los que no, de planificación local.

Un esquema de navegación local tiene una conducta del tipo reactivo, su contra parte global resuelve el problema de navegación relacionando de alguna forma la información contenida en el mapa y buscando con técnicas de inteligencia artificial, una ruta secuencial que permite llegar de un punto a otro.

La navegación local resuelve el problema de navegación priorizando la tarea de evasión de obstáculos y de dirigirse en alguna dirección. Existen muchas implementaciones, todas siguiendo un esquema muy similar al que se ve en la figura 2.4. Algunos ejemplos son:

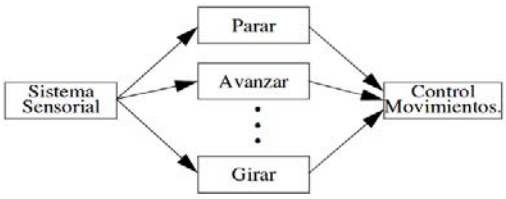


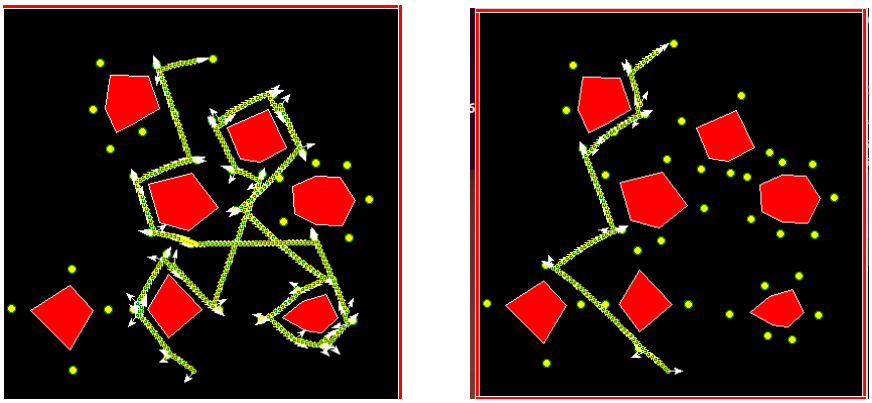
Figura 2.4: Navegación Reactiva

- Campos Potenciales
- Navegación con Landmarks

Se mencionan ahora algunos ejemplos de navegación global, basada en grafos, que hace una búsqueda inteligente del mismo para encontrar una ruta de navegación. Existen varios algoritmos de inteligencia artificial capaz de elegir una ruta, en caso de Dijsktra, se encuentra la mejor ruta según algún parámetro de costo, en este caso la distancia entre nodos. En la Figura 2.5 se comparan los algoritmos Depth First y Dijkstra.

- Beam Search

- Depth First
- Dijkstra



(a) Depth First (b) Dijkstra.

Figura 2.5: Navegación de nodos en un mapa topológico

2.6. SLAM

Simultaneous Localization and Mapping o Slam, es el problema de estimar en tiempo real la estructura de un ambiente real, percibido mediante al menos un sensor mientras el agente móvil se desplaza, al mismo tiempo que navega, en el mapa generado. La tarea de localización y mapeo, como se ha mencionado ya, es de suma importancia, y campo de estudio fértil. Por mucho tiempo, la solución más comúnmente aceptada como tal, es el slam basado en el filtro de Kalman, EKF-SLAM, el filtro *extendido* EKF para ser precisos, propuesto por Smith, Self y Chessman en 1987 [20].

SLAM está formado por tres operaciones básicas:

- Movimiento. El robot se mueve, y genera nuevas observaciones, al hacerlo, el desplazamiento es sujeto a ruido, por lo que la incertidumbre de la posición aumenta. Se propone un modelo que describa este movimiento,llamado modelo de movimiento.

- El robot encuentra nuevos puntos de interés, *landmarks* en inglés. Debido a errores en el sensor, la posición estimada de de estos puntos de interés (landmarks) es incierta. Sumando a la incertidumbre de la posición del robot. De forma similar se necesita un modelo que determine las posiciones de los landmarks. Llamado modelo de observación inversa.

- El robot encuentra puntos de interés mapeados previamente, con ellos, corrige su posición y la de los landmarks reduciendo ambas incertidumbres.El modelo que predice las observación dado que se tiene la posición del robot y del landmark se llama modelo de observación directa.

Estos tres modelos más un estimador bastan para resolver el problema de slam. El filtro EKF y sus variantes son ejemplos de dichos estimadores.

- Modelo de Movimiento

  Se supone un modelo de un robot R, que se desplaza gracias a una señal de control u con un ruido de proceso w.La función entonces actualiza el estado del robot R.

$$R \leftarrow f(R, u, w) \tag{2.23}$$

  .

- Modelo Observación Directa; el modelo de la medición $\vec{o}_k$, cuando el robot R encuentra el landmark késimo L_k mapeado con los sensores S.

$$o_k = h(R, S, L_k) \tag{2.24}$$

  .

- Modelo Observación Inversa Se calcula el estado de un nuevo landmark.

$$L_k = g(R, S, o_k) \tag{2.25}$$

2.7. EKF-SLAM

Cuando el estimador propuesto para resolver el problema de slam es un Kalman Extendido EKF, entonces se habla de un EKF SLAM. En él el mapa es formado con un vector que va apilando el estado del robot R con los landmarks encontrado L_k.

$$x_n = \begin{bmatrix} R \\ L_1 \\ L_2 \\ . \\ . \\ L_k \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \\ L_1 \\ L_2 \\ . \\ . \\ L_k \end{bmatrix} \tag{2.26}$$

En EKF-SLAM el mapa es modelado con una variable gaussiana, definiéndola con la matriz de medias y covarianzas del vector estado, denotados respectivamente, x_n y P. El objetivo del EKF-SLAM entonces no es más que mantener actualizado el mapa [x_n, P].

$$P = \begin{bmatrix} P_{RR} & P_{RM} \\ P_{MR} & P_{MM} \end{bmatrix} \tag{2.27}$$

Puede apreciarse por qué el algoritmo es tan caro cuando hay un numero grande de landmarks, pues hay que mantener actualizadas matrices de covarianzas de tamaño k en cada tiempo n.

2.8. Landmarks o puntos de interés

Se ha comentado de la necesidad de encontrar puntos claves en el ambiente que el robot va sensando. Un buen landmark debe ser fácil de reobservar desde cualquier punto que se le vea. Es deseable también que sean fácilmente distinguibles, pues como se ha comentado en párrafos anteriores, un buen punto de interés es capaz de disminuir la incertidumbre de las estimaciones.

Particular atención merecen los landmarks encontrados por el método de picos.

Que usando valores extremos encuentra estos puntos de interés en señales de tipo láser. Los landmarks se localizan buscando valores en los que las mediciones entre puntos cercanos cambien mucho entre una lectura y la siguiente. Un ejemplo de lecturas láser y los puntos detectados se muestra en la figura 2.6.

Figura 2.6: Landmarks a partir de lecturas láser.

2.9. ROS

Robot Operating System, es un sistema de comunicación entre módulos muy utilizado para controlar robots. En un nivel general ROS es un conjunto de librerías y aplicaciones de licencia abierta, para el desarrollo de aplicaciones robóticas. El sistema es capaz de comunicar varios dispositivos y programas, en un ambiente bastante cómodo. Basado en mensajes y tópicos. Cualquier dispositivo o código puede ser un nodo. Que publica o recibe mensajes.

Una de las muchas ventajas que pretende explotarse con este sistema es la estandarización en la estructura de los datos de comunicación, como sensores, controladores de movimiento, incluso mapas y puntos de interés. Otra gran ventaja es la capacidad de pasar de simulación a un agente real. Como se hizo con ayuda del Turtlebot.

Un ejemplo de una definición, que resulta obligado para este trabajo es el de el mensaje que genera la lectura de un sensor láser.

```
1
2 std_msgs/Header header
3 float32 angle_min
```

```
4  float32 angle_max
5  float32 angle_increment
6  float32 time_increment
7  float32 scan_time
8  float32 range_min
9  float32 range_max
10 float32[] ranges
11 float32[] intensities
```

Todos los valores se explican claramente en la hoja de especificaciones del sensor utilizado, y se genera un servidor de mensajes que es el que irá construyendo la biblioteca de observaciones $\vec{o}_n$. En terminología de ROS, un cliente implementado en Python, estará suscrito al tema sensor láser, realiza cálculos y publica los mensajes propios de navegación y localización utilizados para SLAM.

Capítulo 3

Modelos de Markov

> Es ridículo vivir cien años y sólo ser capaces de recordar treinta millones de bytes. O sea, menos de lo que cabe en un CD. La condición humana se hace más obsoleta con cada minuto que pasa
>
> Marvin Minsky

3.1. Propiedad de Markov y cadenas de Markov

Las cadenas de Markov son un modelo estocástico que cumple con la propiedad de falta de memoria conocida como propiedad de Markov. La propiedad de Markov dice formalmente que [5] la probabilidad de transición de un estado a otro del sistema depende enteramente del estado actual, y no de los estados anteriores que lo llevaron a él. Es decir, el sistema no tiene memoria.

$$P(S_{n+1} = s_{n+1} \mid S_n = s_n, S_{n-1} = s_{n-1}, ..., S_1 = s_1) = P(S_{n+1} = s_{n+1} \mid S_n = s_n) \tag{3.1}$$

Una secuencia de variables aleatorias en el espacio de estados. La transición entre estados se modela estadísticamente y registrando las probabilidades de transición de

un estado, dado que se está en un estado previo[10].

Formalmente una cadena o modelo de Markov es un autómata probabilístico. Un diagrama de transición de estados como el descrito puede observarse en la figura 3.1.

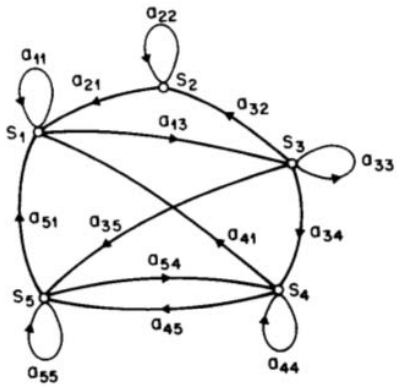


Figura 3.1: Modelo de Markov.

3.2. Procesos de Markov

Propuesto en 1960 por Ronald Howard [9], es una forma de modelar procesos que dependen en parte de una variable aleatoria y en parte de una decisión, o proceso de control. Los MDP's (acrónimo de Markov decision process) han probado ser útiles estudiando problemas de optimización resueltos mediante algoritmos de aprendizaje supervisado y programación dinámica.

Un proceso de decisiones de Markov es un proceso de control estocástico discreto en el tiempo. En cada tiempo n el proceso se encuentra en un estado S. En él, un planeador de acciones decidirá que acción a (disponible en el estado del sistema actual) debe realizarse. Dado que esta acción y el estado S son condicionalmente independientes de estados y acciones anteriores, se dice que cumplen con la propiedad de Markov. En el caso que sólo hay una posible acción en cada estado, el problema se reduce a una cadena de Markov.

Las transiciones de los estados son afectadas por una señal de recompensa, una función que compara el estado futuro con el actual $R(S_k, S_{k-1})$, con la idea de favorecer transiciones a estados deseables. El problema se resuelve por simulación.

Generando recorridos de estados de forma aleatoria y observando las recompensas generadas en cada uno.

La idea es entonces encontrar un conjunto secuencial de acciones, llamado política de control, $\pi = \vec{a}$, tal que maximice las recompensas acumuladas. Una vez que esta secuencia de decisiones es obtenida, entonces el problema se reduce nuevamente a una cadena de Markov.

3.3. Modelos Ocultos de Markov

El Modelo Oculto de Markov HMM por sus siglas en inglés, es un modelo estocástico de un número finito de estados, cada uno de los cuales se encuentra relacionado con un conjunto de probabilidades, llamadas de emisión(Figura 3.2). Las transiciones entre estos estados son gobernadas por otro conjunto de probabilidades, el de transición.

El fenómeno sólo es observado a través de las emisiones, el estado del modelo no se observa directamente, por eso *ocultos* [19].

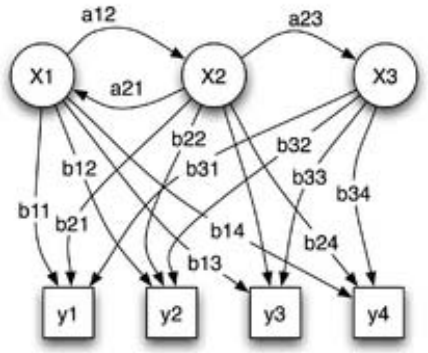


Figura 3.2: Modelo Oculto de Markov.

El modelo pasa por un cambio estados (siendo posible que la transición sea permanecer en el mismo estado) en un período finito de tiempo. De acuerdo al conjunto de probabilidades de transición. Definiendo cada uno de los momentos como t=1,2,... y el estado actual en el tiempo t como q_t.

Para el caso particular de una cadena de Markov, discreta y de primer orden, es posible una descripción probabilística completa conociendo la descripción que corresponde a la probabilidad del estado actual y el anterior. Por lo que dada la naturaleza discreta del alfabeto de las posibles observaciones el Modelo Oculto de Markov está conformado por los siguientes elementos.

3.3.1. Elementos de HMM

1. Numero de estados N.

2. Tamaño de alfabeto de observaciones posibles M.

3. Una matriz de transición dada por: $a_{ij} = P[q_t{=}S_j \mid q_{t-1}{=}S_i]$.

   considerando algunas limitantes estocásticas

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{3.2}$$

$$a_{ij} \geq 0 \tag{3.3}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & ... & a_{1N} \\ a_{21} & a_{22} & a_{23} & ... & a_{2N} \\ a_{31} & a_{32} & a_{33} & ... & a_{3N} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ a_{N1} & a_{N1} & a_{N1} & ... & a_{NN} \end{bmatrix} \tag{3.4}$$

4. La probabilidad de el símbolo emitido en el estado j:

$$b_j(k) = P[V_k \mid q_t = S_j] \tag{3.5}$$

$$1 \leq j \leq N, 1 \leq k \leq M \tag{3.6}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & ... & b_{1M} \\ b_{21} & b_{22} & b_{23} & ... & b_{2M} \\ b_{31} & b_{32} & b_{33} & ... & b_{3M} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ b_{N1} & b_{N1} & b_{N1} & ... & b_{NM} \end{bmatrix} \tag{3.7}$$

5. Probabilidad de estado inicial π_{i}

$$\pi_i = P[q_1 = S_i], 1 \leq i \leq N. \tag{3.8}$$

Entonces el Modelo queda completamente definido por las matrices A, B y π.

$$\lambda = (A, B, \pi) \tag{3.9}$$

3.3.2. Los Tres Problemas Básicos utilizando Modelos Ocultos de Markov

En un Modelo Oculto de Markov hay tres problemas básicos que interesa resolver.

Problema 1

Dado que se conoce una secuencia de observaciones o=$\mathrm{o_1o_2...o_T}$ y un modelo λ se busca la P(o| λ), es decir, la probabilidad de que se generó la secuencia de observaciones o dado el modelo λ. Esta probabilidad se calcula de forma inductiva usando el algoritmo de adelanto que se detalla más adelante.

Problema 2

Dado el vector de observaciones o=$\mathrm{o_1o_2...o_T}$ y el modelo λ, cuál es la secuencia de estados más probable que generó las observaciones. Esta probabilidad puede ser calculada eficientemente con el algoritmo de Viterbi, como se verá a continuación.

P(Q| o, λ).

Problema 3

El problema 3 es el de encontrar un modelo λ que maximice la probabilidad de generar un conjunto de observaciones dado dicho modelo λ ; es decir, P(o| λ). Este problema se resuelve estadísticamente, contando el número de veces que se transiciona de un estado a otro y las emisiones generadas en cada estado. El método utilizado es el algoritmo de Baum-Welch que igualmente se detalla en las secciones siguientes.

3.3.3. Algoritmo de Adelanto

Como se mencionó antes, el algo ritmo de Adelanto (Forward Algorithm) es utilizado para resolver el primer problema de los detallados previamente. Dada una secuencia de observaciones *o*, qué modelo λ es el que más probablemente generó estas observaciones ClaseElegida=argmax[P(o| λ_{clase})].

Este cálculo se define como la suma de la probabilidad de todos los posibles caminos (o secuencias) de estados, que pudieron generarlo; es decir:

$$p(o \mid \lambda_{clase}) = \sum_{j=1}^{S} \left(\pi_{s1} \prod_{t=1}^{i=T} a_{st.st+1} b_{st-st+1.oi} \right) \tag{3.10}$$

La complejidad computacional de este cálculo es del orden de $N^T(o(N^T))$. Lo que hace poco viable su implementación directa. [3].

El Algoritmo de Adelanto permite calcular esta probabilidad de forma iterativa.

1. Inicialización
$$\alpha_1(i) = \pi_i.para 1 \leq i \leq N \tag{3.11}$$

2. Para t=1,2,....T:
$$\alpha_{t+1}(i) = \sum_{i=1}^{N} \left(\alpha_t(i) a_{i,j}\right) b_j(o_{t+1}). \tag{3.12}$$

3. Término
$$p(o \mid \lambda) = \sum_{i \in estadosterminales} \alpha_{T+1}(i) = \sum_{i=1}^{N} p(o_1, o_2, ..., o_T, q_t = i \mid \lambda). \tag{3.13}$$

El proceso puede verse de forma gráfica en la figura 3.3.

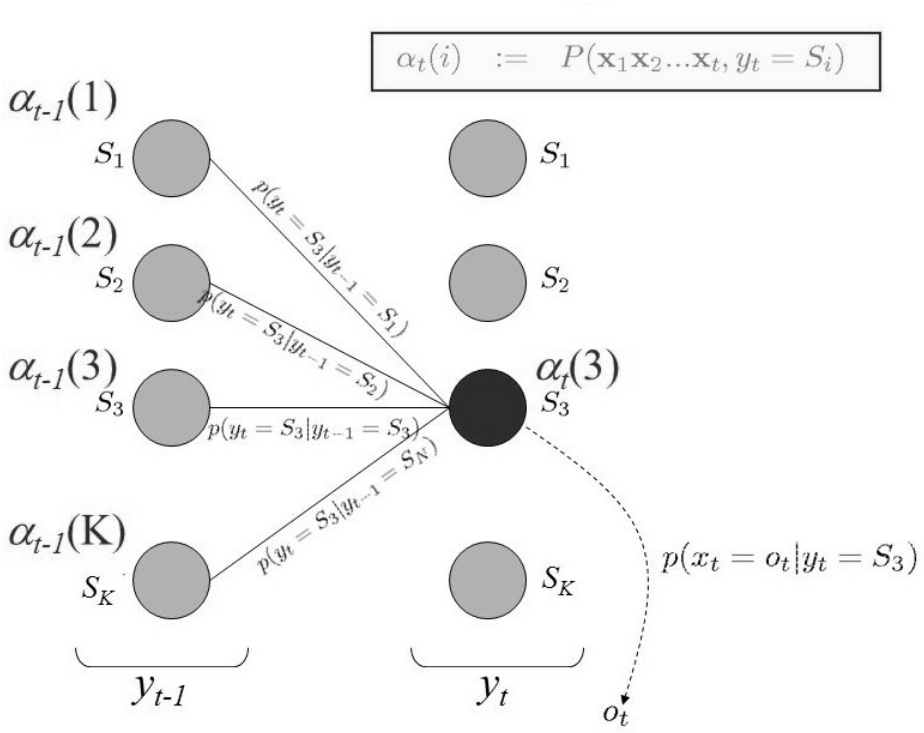


Figura 3.3: Lattice Algoritmo de Adelanto.

El algoritmo se inicializa el valor correspondiente de π_i. Y se observa que el modelo puede permanecer en el mismo estado (siguiendo la línea horizontal) o cambiara otro estado (y tomar la ruta diagonal). En cada caso la probabilidad de la ruta es igual al producto del punto de la rejilla anterior $\alpha_t(i)$, la probabilidad de transición dada por $a_{i,j}$ y la probabilidad de la observación $b_{i,j.o_t}$. La probabilidad de cada punto del lattice α_{t+1} es la suma de todas los arcos entrantes. Por lo tanto la probabilidad total de cada punto de la rejilla es la probabilidad de llegar del estado inicial al estado del tiempo t. La probabilidad final es la suma de todas las posibles probabilidades de salida.Este esquema es conocido también como un decoder de adelanto o decoder de Viterbi.

Es de esta forma como se resuelve el primer problema, dado una secuencia de observaciones, cuál de los modelos λ a comparar es el que tiene la máxima probabilidad de generar la secuencia. Como se observa en la figura 3.4.

Una nota importante tiene que ver con el orden de magnitud de este cálculo, ya

que todas las probabilidades son menores a uno, el orden de α disminuye rápidamente en cada iteración, por lo que es conveniente agregar un factor de escala que corrija este problema para evitar que el cálculo salga del rango dinámico del sistema. En la implementación del algoritmo se agregó un término de orden 4 para escalar la probabilidad en cada paso.

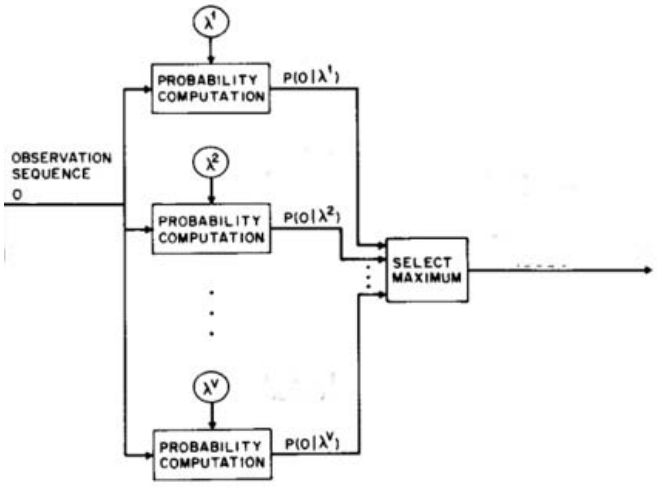


Figura 3.4: [19]Diagrama de Bloques para elegir el modelo de qué región es el más probable dada la secuencia de observaciones.

3.3.4. Algoritmo de Viterbi

El algoritmo de Viterbi sirve para resolver el segundo problema planteado en 3.3.2. El de calcular la secuencia de estados más probable dado que se conoce una secuencia y de observaciones y el modelo λ.

Propuesto originalmente en 1967,el algoritmo calcula la probabilidad Máximo a Posteriori (MAP) de una secuencia de estados estimada, dado que se conoce una secuencia de observaciones las probabilidades de transición y emisión (modelo).Considerando la cadena de Markov y las posibles emisiones mostradas en la figura 3.5. En la figura 3.6 se observa como en cada tiempo k se analizan las posibles secuencias , y cómo se van eliminando las menos probables, hasta que al final se decide por una secuencia.

El algoritmo funciona almacenando las secuencias que generan la ruta más corta

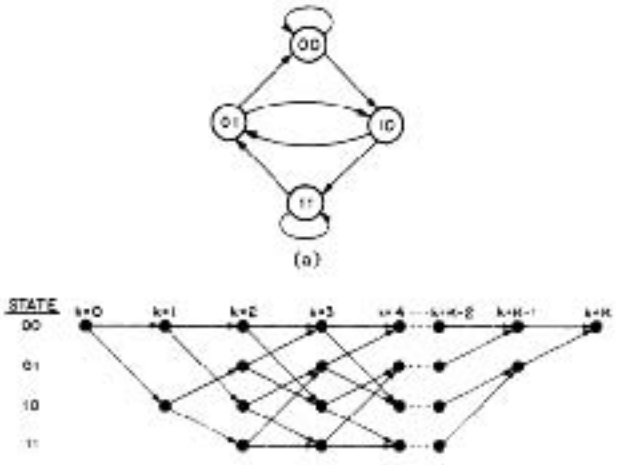


Figura 3.5: [7] a)Diagrama de estados b)Trellis de las transiciones

hasta el tiempo k. Igualmente se va almacenando la distancia de cada una de estas secuencias sobrevivientes en cada iteración. El algoritmo genera una serie de estados S_T con máxima probabilidad de emitir una secuencia de observaciones o.

Se almacenan por un lado la secuencia de estados más probable hasta el tiempo j, y por otra parte se almacena la probabilidad que esta secuencia de estados haya generado las observaciones o. La secuencia de estados resultante es obtenida en orden inverso, por lo que el resultado debe ser invertido.

1. Inicio

$$P_1 = \pi_i \cdot b_i(o_1) \tag{3.14}$$

2. Iteración

$$P_t(j) = \max_{1<i<N} [P_{t-1}(i) \cdot a_{ij}] \cdot b_j(o_T). \tag{3.15}$$

$$S_t(j) = arg \max_{1<i<N} [P_{t-1}(i) \cdot a_{ij}] \tag{3.16}$$

3. Fin

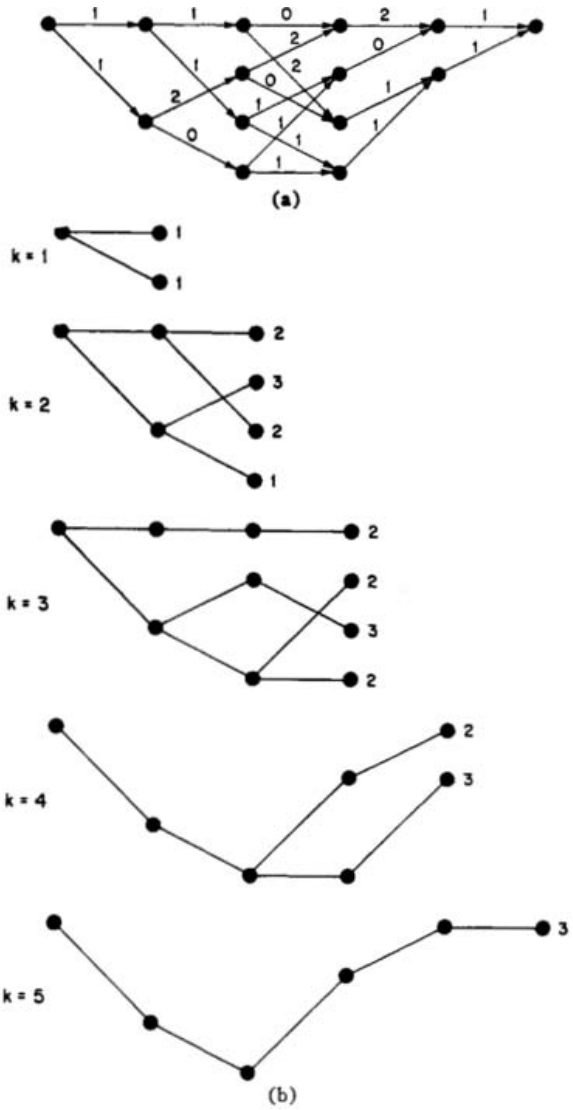


Figura 3.6: [7] Algoritmo de Viterbi

$$P_t(j) = \max_{1<i<N} P_T(i) \tag{3.17}$$

$$S_t(j) = arg \max_{1<i<N} P_T(i) \tag{3.18}$$

3.3.5. Entrenamiento

El tercer problema consiste en ajustar el modelo dado que se tiene un conjunto de observaciones con su respectiva secuencia de estados.Este no es un problema trivial, y se resuelve por diferentes métodos sin que se tenga uno cómo óptimo aún. [19]

Se utilizan algoritmos de gradientes [12], y los llamados EM (Expectation Modification) [4]. Sin embargo, el algoritmo utilizado es el propuesto por Baum Welch, que funciona combinando el algoritmo de adelanto con su contra parte de atraso,y se detalla a continuación. Para estimar los parámetros de un modelo, deben conocerse el numero esperado de transiciones del estado i y el numero esperado de transiciones de i al estado j. Y dado que los valores son calculados de todas formas para obtener la información de los puntos anteriores, no añade demasiados cálculos al proceso.

Se define

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j|o,\lambda) \tag{3.19}$$

Esta probabilidad puede ser escrita usando los valores que se obtuvieron iterativamente con los algoritmos de adelanto y de atraso.

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{T+1})\beta_{t+1}(j)}{P(o|\lambda)} \tag{3.20}$$

$$= \frac{\alpha_t(i)a_{ij}b_j(o_{T+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(o_{T+1})\beta_{t+1}(j)} \tag{3.21}$$

Se define γ_t(i), como la probabilidad de estar en el estado S_i dada la secuencia de observaciones o y el modelo λ y está dada por :

$$\gamma_t(i) = \sum_{j=1}^{N}\xi_t(i,j) \tag{3.22}$$

$$\sum_{t=1}^{T-1}\gamma_t(i) = \text{Numero de transiciones desde } S_i \tag{3.23}$$

$$\sum_{t=1}^{T-1}\xi_t(i,j) = \text{Numero de transiciones de } S_i \text{ a } S_j \tag{3.24}$$

Finalmente se calculan las matrices de transición y transferencia que formarán el nuevo modelo a calcular :

$$\hat{\pi}_i \text{Numero de veces que el estado } S_i \text{ se encuentra en t=1} = \gamma_1(i) \tag{3.25}$$

$$\hat{a_{i,j}} = \frac{\text{Numero de transiciones del estado } S_i \text{ al estado } S_j}{\text{Número de Transiciones desde } S_i} \tag{3.26}$$

$$= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(j)} \tag{3.27}$$

además

$$\hat{b_i(k)} = \frac{\text{Numero de Veces que se observa el símbolo } V_k \text{ en el estado j}}{\text{Número de Veces en el estado j}} \tag{3.28}$$

$$= \frac{\sum_{t=1}^{T} 1_{o_t=v_k} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{3.29}$$

Capítulo 4

Desarrollo

> Es un error capital teorizar antes de que uno tenga datos, insensatamente uno empieza ajustar los hechos para acomodar a las teorías, en vez de ajustar las teorías para cuadrar con los datos.
>
> Sherlock Holmes

4.1. Virbot

Las simulaciones fueron realizadas en el sistema Virbot, una plataforma desarrollada en la UNAM capaz de simular un agente móvil y un arreglo de sensores láser. En él se prueban primero varias conductas reactivas de navegación local y de evasión de obstáculos utilizando campos potenciales. También se generó un modelo global que escoge una trayectoria de nodos óptima utilizando el algoritmo de Dijkstra, dado que se tiene información previa en forma de un mapa topológico. A continuación se detallan los módulos utilizados.

Este sistema de simulación funciona en cuatro capas principales y es similar a la arquitectura del agente INTERRAC[23](Figura 4.1).

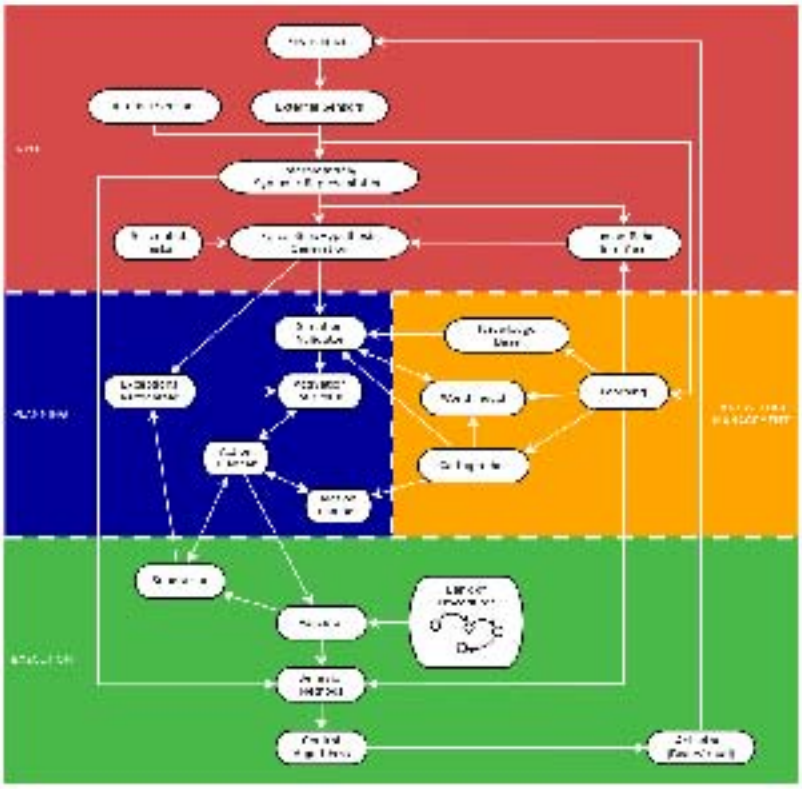


Figura 4.1: Diagrama de Virbot

Capa de Entradas

En esta capa se encuentran los sensores internos y externos del robot, así cómo el simulador de los mismos. Estos valores son cuantizados y representados y procesados.

Planeador

- Planeador de Acciones: Aquí se calculan las actividades físicas requeridas para lograr un objetivo.

- Planeador de Movimientos: Este modulo recibe un conjunto de puntos que forman la ruta necesaria para lograr el objetivo del planeador de acciones. Es aquí que el algoritmo de Dijkstra genera la trayectoria óptima.

Capa de Conocimiento

El submódulo de interés para este trabajo es el Cartógrafo. Módulo responsable de generar los diferentes tipos de mapas que representan el entorno. Y de estimar la posición dentro de los mismos, es decir navegar. Compara los valores calculados en la capa de entradas, y con la ayuda de los mapas almacenados da y recibe información al Planeador.

4.2. Robot

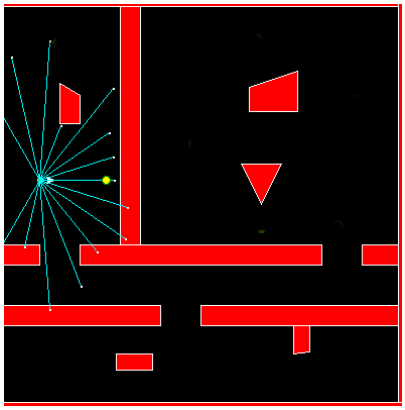
El robot a utilizar es un Turtlebot,mostrado en la figura 4.2. Un robot omnidireccional equipado con un sensor láser Hokuyo Hokuyo URG-04LX-UG01, del tipo range finder. Capaz de medir distancias en un arco de 270° al frente del robot. Este conjunto de distancias forman un vector de observaciones $\vec{o}$. Las mismas son leídas y transmitidas en un ambiente ROS, donde son procesadas. También es posible acceder a los mensajes de control del robot. En el software, el robot es capaz de una conducta autónoma de navegación aleatoria. Convirtiendo de forma reactiva lecturas de sensores en instrucciones de movimiento.

Figura 4.2: Turtlebot

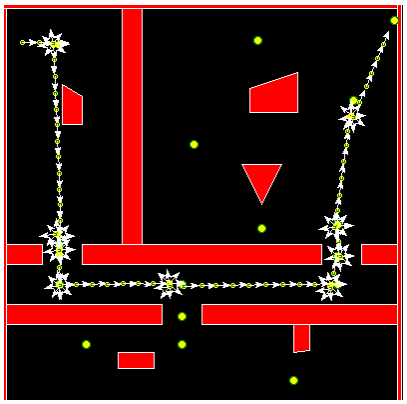
4.3. Proceso

Se supone primero que el robot tiene alguna información previa sobre el ambiente, que le permite navegar de forma autónoma utilizando un mapa topológico. Formado por una cuantización del espacio libre, generando nodos de navegación artificiales en la ubicación de cada uno de los centroides generados. Las observaciones son obtenidas

por un láser montado sobre el robot con el se toman 15 lecturas distribuidas uniformemente en un arco de 270° frente al robot y se forma un vector $\vec{o}$ compuesto por cada una de estas mediciones a cada tiempo n. Las observaciones $\vec{o_n}$ son cuantizadas utilizando los centroides calculados con el algoritmo LBG. Convirtiendo la secuencia de observaciones $\vec{o_n}$ en la secuencia de símbolos observados v_n. Esta secuencia será la variable observable del modelo HMM.



(a) 15 lecturas laser del sensor.



(b) Navegación mapa topológico

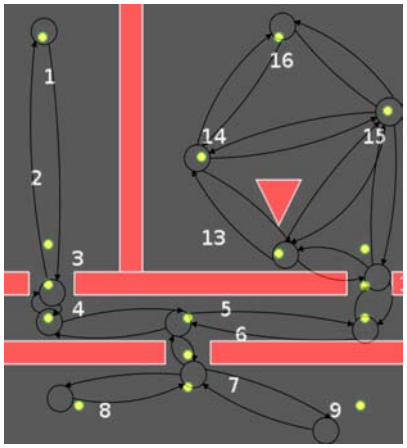


Figura 4.4: Cada estado del modelo HMM corresponde a un nodo del mapa topológico.

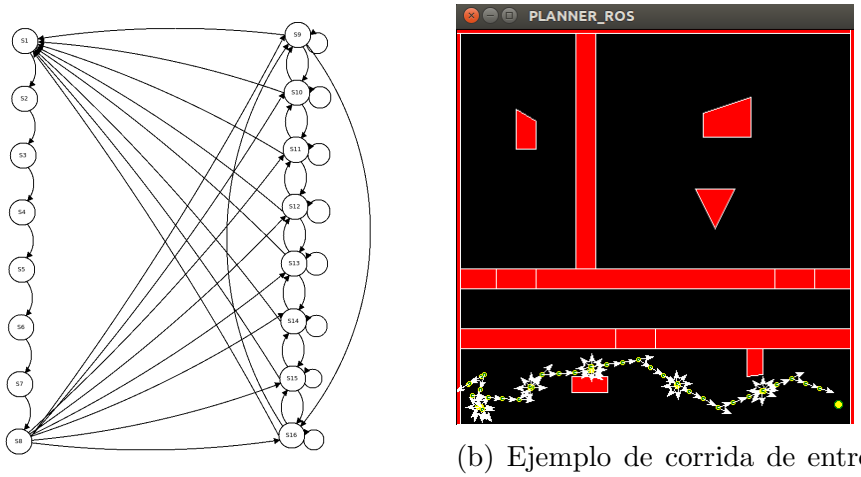
Las instrucciones utilizadas para navegar el mapa topológico permiten generar un modelo HMM relacionando las observaciones con el nodo, que se supone plenamente localizado y reconocible. El modelo HMM entonces estimaría cual es el nodo más cer-

cano dadas las últimas$n$observaciones. Donde$n$es el tamaño de búfer de Viterbi. En la figura 4.4 pueden observarse el arreglo de sensores, una corrida de entrenamiento y finalmente una imagen superpuesta del modelo de transiciones propuesto.

Otro enfoque toma en cuenta la situación en que el robot no tiene información previa del entorno, y la navegación por el mismo se realiza de forma reactiva. El entrenamiento se realiza almacenando las lecturas cuantizadas a cada tiempo n. Igual que en el caso anterior las observaciones son generadas por un láser, se toman 15 lecturas distribuidas uniformemente en un arco de 270° frente al robot y se forma un vector $\vec{o}_n$ compuesto por cada una de estas mediciones a cada tiempo n. Se genera así la secuencia de observaciones(cuantizada) v_k que será la variable observable de los modelos HMM. La diferencia en este enfoque es que la variable oculta será relacionada con la orientación del robot. Con el objetivo de generar un modelo diferente en cada región del entorno. El robot transita reactivamente la región, donde se evita que salga de alguna forma externa(cerrando la puerta),reestimando la orientación correcta cada determinado número de pasos libres. Esta conducta de entrenamiento puede ser observada en el diagrama de la figura 4.4. Donde se presenta el diagrama de transiciones y un ejemplo de una corrida de entrenamiento en una región con las puertas cerradas. El estado 1 y 8 corresponden a orientación 0° cuando se encuentra en el giro de reorientación. Si el 0° se registra en conducta reactiva corresponderá a los estados 9 y 16. Cada estado corresponde a una variación en la orientación de 45° .

La localización se realiza primero utilizando el algoritmo de adelanto que permite estimar cual Modelo λ es el que más probablemente generó estas observaciones v_n. En la figura 4.7 se muestra el caso en el que hay cuatro regiones (con su cuatro modelos HMM correspondientes). Es interesante comentar que existen transiciones no permitidas, por ejemplo, para pasar de la región 1 a la 4. Forzosamente habrá que cursar la región 2 primero.

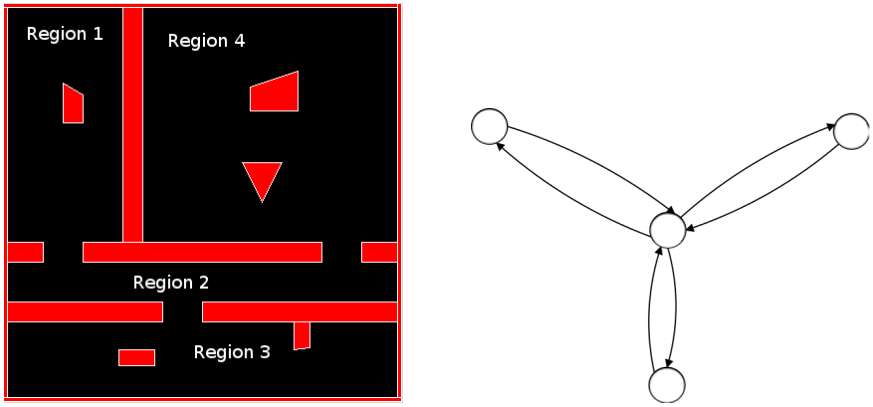
En otras palabras, cuál es la secuencia de estados x_n (representando orientación del móvil) del modelo λ más probable dada la secuencia de símbolos v_n que representan observaciones $\vec{o}_n$. La Figura 4.8 muestra un diagrama de bloques del sistema propuesto.



(a) Transición de estados.

(b) Ejemplo de corrida de entrenamiento.

Figura 4.5: Variable Oculta Orientación



(a) El entorno se divide en 4 regiones.

(b) Se generan 4 modelos, uno por región

Figura 4.6: Regiones y modelos .

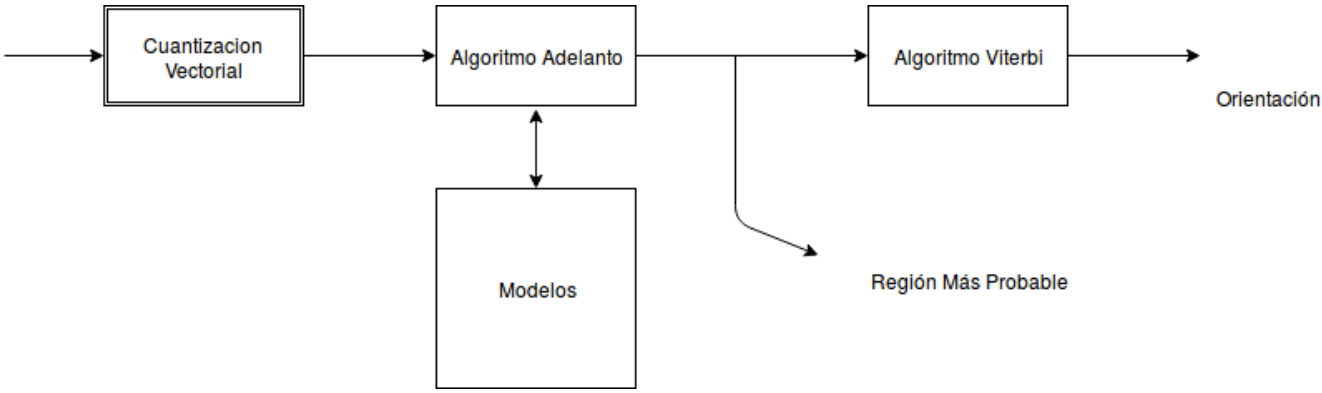


Figura 4.7: Diagrama Sistema Propuesto.

Capítulo 5

Experimentos

> Los datos son el petróleo del siglo XXI. El despliegue de sensores y el incremento de la capacidad del procesamiento, son claves en la transformación de muchos sectores y en la creación de un mundo más medible y programable.
>
> Cesar Alierta

5.1. Simulación

El robot móvil fue simulado con un sensor que tiene un rango de visión de 270° al frente del robot y un muestreo de quince puntos equidistantes sobre este arco. Se forma un vector de observaciones con estas 15 muestras para cada instante n. El robot es de tipo omnidireccional y es simulado bajo el ambiente ROS y una interfaz gráfica de polígonos VirBOT. El robot cuenta con varios tipos de conducta, que funcionan de forma reactiva con los sensores, sin necesidad de mapas.

El primer paso es obtener vectores de entrenamiento, a partir de los cuales sea posible obtener un modelo HMM. La cantidad de vectores de entrenamiento utilizados es un factor importante en el desempeño.Y algunas consideraciones de sobreentrenamiento, ciertas para casi todos los algoritmos de machine learning, deben tomarse en

Entrenamiento	*Número de Símbolos*	*Bufer Viterbi*	*% error*
4000	32	36	33 %
8000	32	36	29 %
4000	256	36	25 %
8000	256	36	18 %
4000	512	36	23 %
8000	512	36	12 %
4000	32	64	33 %
8000	32	64	29 %
4000	256	64	24 %
8000	256	64	15 %
4000	512	64	17 %
8000	512	64	09 %

Cuadro 5.1: Tabla errores clasificación en simulación.

cuenta. Este conjunto de vectores de entrenamiento serán representados por símbolos, obtenidos con el algoritmo LGB.

Una vez que se tiene el modelo HMM del entorno se realizan corridas de prueba, y se comprueba la efectividad del modelo para estimar correctamente la secuencia de estados relacionada. Que son observados directamente fuera del sistema para realizar la comparación.

Fue necesario construir una nueva máquina de estados, capaz de reestimar la orientación del robot de forma reactiva, utilizando un Landmark virtual simulado. Que hace las veces de un " norte" para el modelo y para el agente móvil. La navegación reactiva utiliza las mediciones para encontrar una pared, y seguirla, con la idea de rodearla, las esquinas de una habitación funcionan como landmarks, que al volver a encontrar generan una trayectoria en espiral para buscar navegar todos los puntos de la región. Un parámetro importante a determinar es cada cuantos pasos resulta eficiente reestimar la orientación del móvil. Se propone un sistema que incremente esta cantidad de pasos libres a medida que la incertidumbre disminuye.

Los experimentos que se muestran utilizan diversos tamaños de centroides detallados en las tablas, la cantidad de últimas muestras corresponde al tamaño del búfer de Viterbi.

Con base en estos resultados se elige realizar las pruebas con más vectores de entrenamiento(se utilizaron 20000), 256 símbolos en el cuantizador vectorial y un tamaño de búfer de 36 muestras. Primero el entorno fue modelado en un sólo HMM. Dado que se cuenta con un sistema de navegación capaz de leer un modelo topológico obtenido previamente con la cuantización del espacio libre. El robot realiza varias tareas, traducidas en secuencias de nodos por el planeador e interpretado en movimientos(sujetos a ruido) por el planeador de movimientos. El modelo HMM estima la secuencia de estados internos más probable, dada una secuencia de lecturas de longitud 36(bufer de Viterbi) . La variable oculta X entonces estará relacionada con los nodos. X=Numero de nodo del mapa topológico. El sistema probó efectividad y robustez estimando correctamente la secuencia con el algoritmo de Viterbi. Se observa la tendencia de los estados estimados a tener errores cuando el búfer no se encuentra lleno.

El problema se encontró al entrenar el robot real, pues los valores de odometría utilizados para encontrar los nodos y poder hacer la navegación topológica, no eran suficientemente precisos. Adicionalmente el algoritmo de Viterbi no tiene restricciones suficientes para garantizar un alto grado de éxito en la estimación. Dados los resultados exitosos de la simulación suponen que la dependencia de un sistema adecuado para navegar un mapa topológico hacen de este enfoque para la implementación de SLAM inviable.

Por esta razón se propone otro experimento, donde no es necesaria ninguna información previa y donde mínima supervisión de entrenamiento es requerida. Se propone dividir el mundo en cuatro regiones (cuartos). Con el propósito de relacionar la variable oculta del HMM con la orientación. El entrenamiento fue realizado dejando al robot en un comportamiento totalmente reactivo, vagando en un cuarto a la vez. Mientras de forma externa se impide salir de la región, cerrando la puerta por ejemplo. Los vectores de observación que se generan se almacenan, sabiendo que provienen del cuarto (región o modelo λ) en donde se permite vagar libremente al robot.

El experimento relaciona un Modelo Oculto de Markov a cada región. El mundo es compuesto por varios Modelos Ocultos de Markov interconectados, uno por cada región ; el estado de cada una de las regiones o estados, será la orientación del robot

dentro de dicha región.

De esta manera el mundo es modelado a su vez por varios sub-modelos, y la localización del robot, es calculada encontrando cuál modelo λ es más probable dado un conjunto de observaciones. Esta probabilidad puede ser calculada con el algoritmo de adelanto. Así, en conjunción con el algoritmo Viterbi puede calcularse la secuencia de estados más probable (la orientación), además del modelo que más probablemente las puede explicar(la región). Una gran ventaja de este método es que el mapa global es compuesto por mapas más pequeños, y nuevas partes pueden agregarse o modificarse sin necesidad de actualizar todos los modelos que conforman el mapa global.

Una vez obtenidos los modelos se realizan corridas de prueba, estimando en cada punto cual es la región más probable. Externamente se observa la ubicación del móvil para comprobar los valores estimados. En la figura 5.1 se muestran los resultados de estimación de la región, la cantidad de muestras de las corridas de prueba en la región contra las estimaciones del modelo HMM que más probablemente generó las observaciones. Si bien la información comprueba que el sistema funciona, deben realizarse muchas más corridas de prueba para poder obtener información estadística del desempeño.

Ya que tienen 4 Modelos HMM las lecturas cuantizadas entran al algoritmo de adelanto, así se escoge que modelo HMM utilizar para estimar el estado interno utilizando el algoritmo de Viterbi. De forma similar se controla externamente la orientación real para compararla con la estimación. Esta comparación puede verse gráficamente en la figura 5.2; ahí se muestra la comparación de las posiciones estimadas en una corrida real y los valores reales (orientaciones) a cada tiempo$n$.

Es muy importante destacar que cuando el búfer es de una dimensión muy alta puede causar que el sistema salga del rango dinámico en el algoritmo de adelanto. Hay que notar que los giros, la secuencia obligada del 1 al 8 generan gran certidumbre para el algoritmo de Viterbi permitiendo alinear la secuencia con la estimada de una forma mucho más eficiente, mientras el móvil reestima su orientación por medios reactivos.

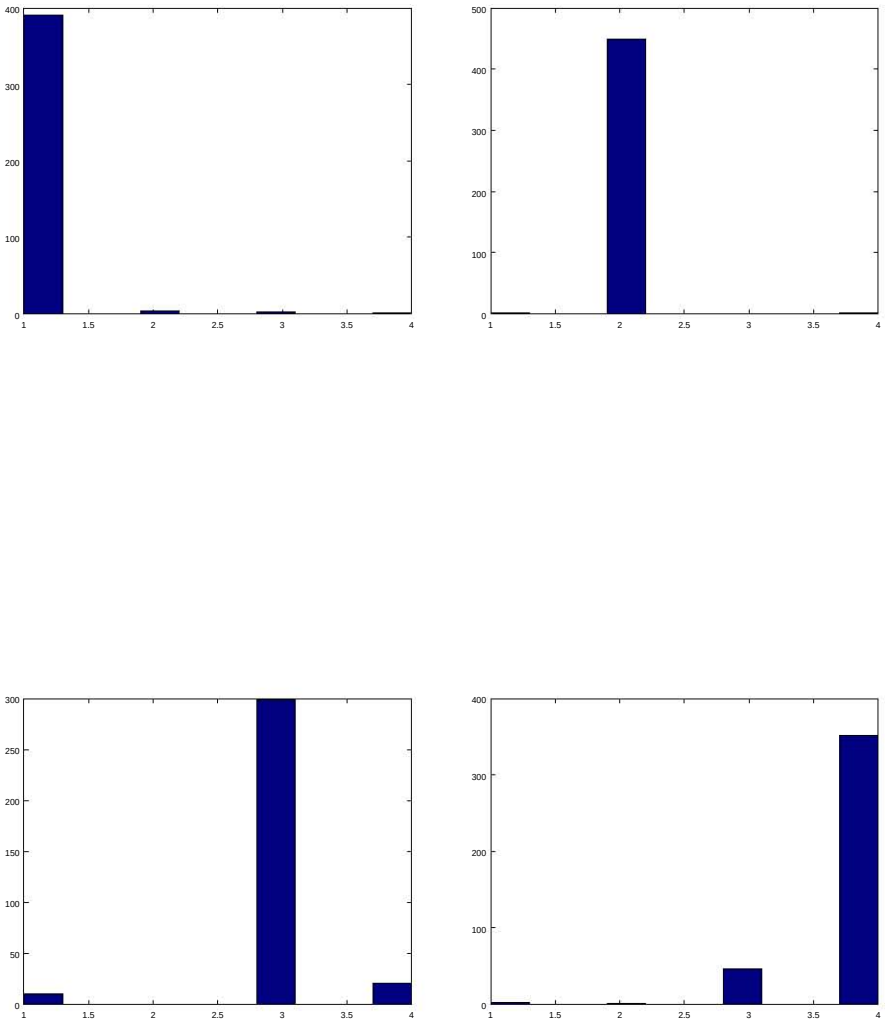


Figura 5.1: Estimación de regiones con algoritmo de adelanto.Bufer Viterbi 64 lecturas 512 centroides

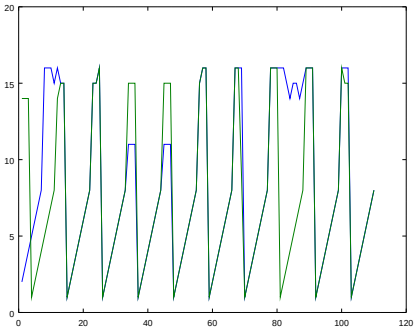


Figura 5.2: Algoritmo de Viterbi empatando secuencia estimada con real

5.2. Experimentos con robot real.

El método se implementa ahora en un robot móvil, TurtleBot figura 5.3, equipado con un sensor láser Hokuyo. Se muestra a continuación la secuencia de estados estimados.

Si bien el sistema que modela y estima los modelos es exactamente el mismo que el de las simulaciones, se hicieron pequeñas adaptaciones en el ángulo de sensores y conductas de entrenamiento.

Figura 5.3: Turtlebot entrenando

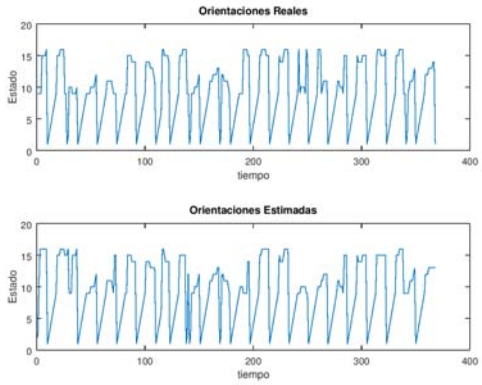


Figura 5.4: Estados reales y estados estimados.

Los resultados muestran que el sistema es robusto, aunado al ruido comentado, el entorno de entrenamiento fue sujeto a tránsito humano y muchos cambios del entorno no considerados en el modelo y aún así el sistema funcionó adecuadamente.

El cuadro 5.2 muestra el desempeño del robot real con la misma conducta de las simulaciones anteriores. Utilizando diversos tamaños de vectores de entrenamiento, numero de símbolos en el cuantizador vectorial, el tamaño del búfer de las últimas muestras y finalmente el porcentaje de error (medido simplemente como acierto o no en la localización y orientación del móvil). La resolución utilizada no permitió capturar información suficiente para discriminar entre algunas regiones, como puede apreciarse en la figura 5.5. y las orientaciones (o estados) mostrados en la figura 5.4. También debe comentarse que la obtención de la estimación no se hizo en tiempo real para el robot, y las corridas de prueba debieron almacenarse y procesarse fuera de línea. Por eso la información contenida en el cuadro 5.2 debe tomarse como una muestra del funcionamiento del estimador en una corrida particular (que pudo favorecer una región). Al igual que con la simulación un número mucho mayor de corridas de prueba es necesario para medir el desempeño del sistema, sin embargo se comprueban las premisas propuestas. Similarmente la clasificación reportada en la figura 5.5 es calculada al dejar fuera de los vectores de entrenamiento un 10 % de las observaciones almacenadas para utilizarlas como observaciones de prueba.

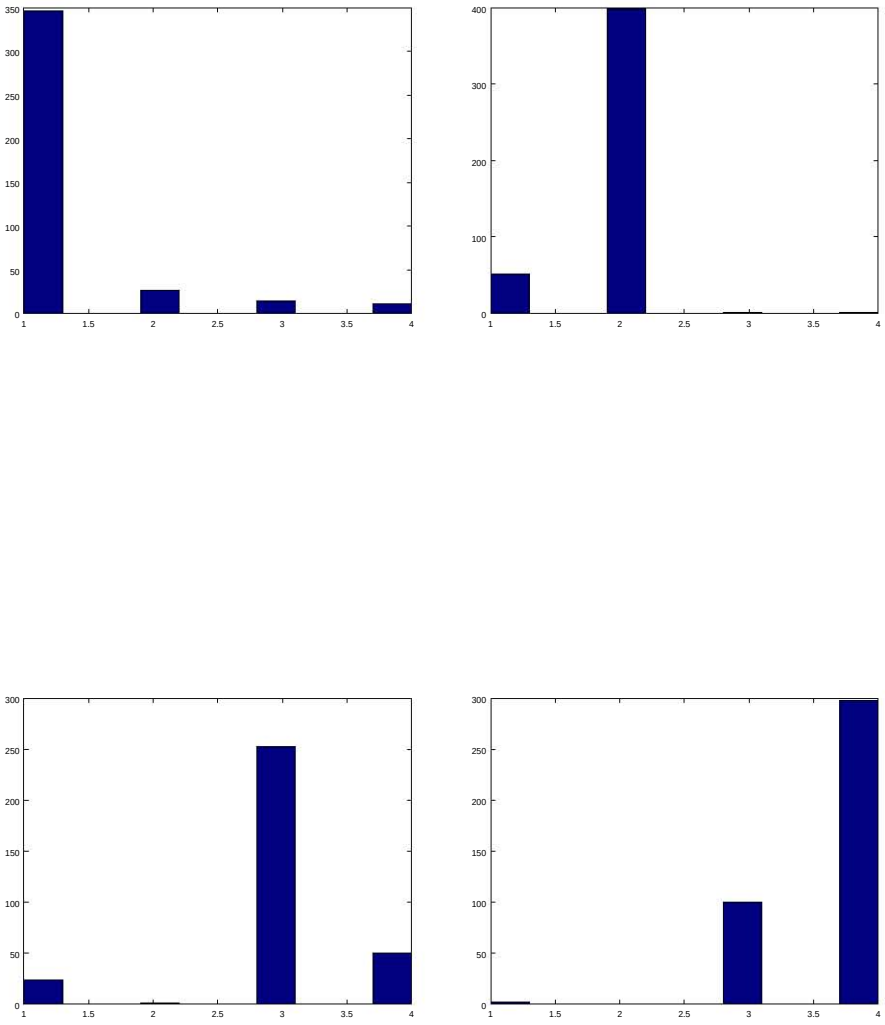


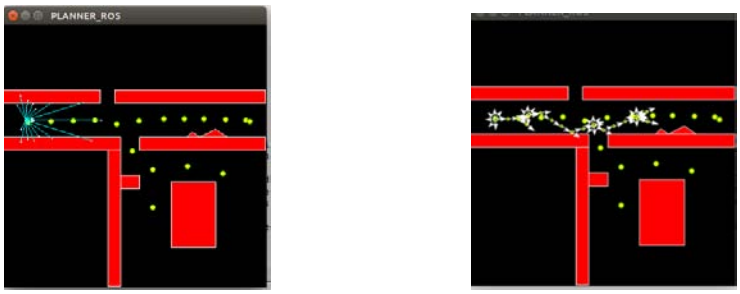
Figura 5.5: Clasificación usando búfer de Viterbi 64 512 centroides, simulación

Entrenamiento	*Número de Símbolos*	*Bufer Viterbi*	*% error*
4000	32	36	56 %
8000	32	36	32 %
4000	256	36	35 %
8000	256	36	16 %
4000	512	36	31 %
8000	512	36	13 %
4000	32	64	45 %
8000	32	64	26 %
4000	256	64	30 %
8000	256	64	15 %
4000	512	64	17 %
8000	512	64	11.3 %

Cuadro 5.2: Tabla errores clasificación robot real.

5.3. Entrenamiento Virtual

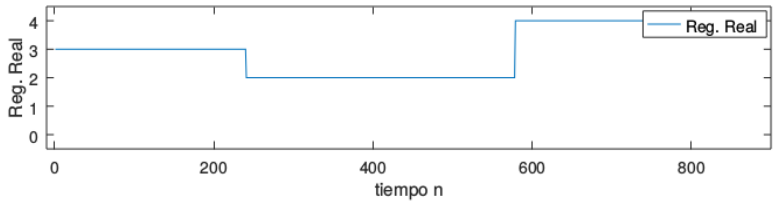
Se propone utilizar el simulador como un entrenador virtual. La ventaja es que en un mapa de celdas de ocupación obtenido previamente con técnicas tradicionales, la estimación del HMM puede generarse con valores ya no de odometría, sino de una localización sin incertidumbre, es decir una odometría virtual completamente definida en el simulador. El robot simulado se entrena en un mapa de polígonos. Y se entrena de forma similar a los experimentos anteriores, la diferencia es que se entrenan 3 modelos HMM a la vez, uno para cada variable de navegación x ,y y θ. Y se genera un modelo por cada región. Las regiones en este experimento no son cuartos físicos, sino regiones cuantizadas del espacio libre. Se forma entonces un modelo por cada región, con la que se estimará la posición. El estado del robot será una vez más, estimado con Viterbi y el modelo HMM elegido. De la misma forma que los experimentos anteriores, se comparan varios parámetros y se forma un mapa global compuesto de 16 modelos HMM, para cada una de las tres variables de localización (x, y, θ), 256 símbolos de observaciones, 20,000 vectores de entrenamiento y bufer de Viterbi de 32 muestras. En la figura 5.6 puede observarse cada una de las "regiones virtuales " utilizadas así como una parte de una corrida de entrenamiento.



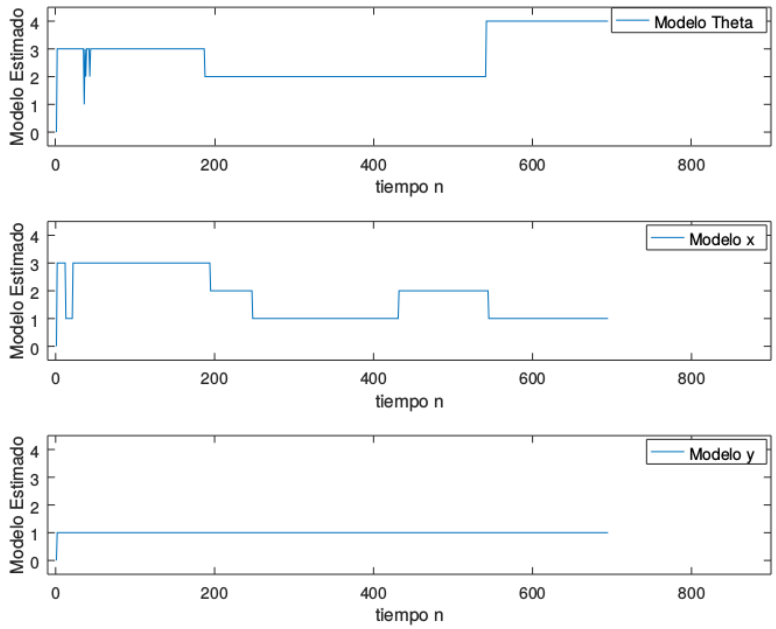
(a) Regiones (b) Ejemplo de corrida.

Figura 5.6: Entrenamiento Virtual.

El sistema se localiza bien, con desempeño consistente a los experimentos anteriores. Es decir, se resuelve satisfactoriamente el problema de localización y navegación, con una conducta y desempeño consistentes con lo observado en los casos anteriores, sin embargo la mejoría de estimación al utilizar tres modelos es apenas apreciable, en contraste con el costo, que se triplica para este planteamiento. Se busca sacar ventaja de la posibilidad de trabajar en paralelo con el uso de DSP's optimizados con este fin. Se propone correr en paralelo un modelo capaz de estimar x, otro y y un tercero θ. Después pueden establecerse relaciones entre las tres salidas para optimizar el comportamiento. En la figura 5.7 se muestra un instante de tiempo de una corrida de prueba. A la salida del algoritmo de adelanto, es decir la estimación de los modelos más probables dadas las observaciones. Se observa que el modelo θ y el modelo x estiman adecuadamente, el modelo y falla. Es decir, para esta secuencia particular de la corrida de prueba, el modelo x y el modelo θ resultan más efectivos. Este no es siempre el caso a lo largo de las corridas de prueba, se observa que en diferentes zonas del mundo parece funcionar mejor uno u otro modelo, el análisis más apropiado de los modelos generados con este experimento parece sugerir un modelo de Markov en tres dimensiones, y permite suponer otro tipo de navegación formando superficies de Markov [17] , mediante una variante en 3D del algoritmo de Viterbi. Fuera del alcance de este trabajo y propuesto como trabajo futuro.



(a) Regiones de una corrida de prueba.



(b) 3 estimadores funcionando en paralelo

Figura 5.7: Comparativo de regiones reales vs 3 modelos de estimadores.

Capítulo 6

Conclusiones y Trabajo Futuro

> Primera ley de los mentat: Un proceso no puede ser entendido deteniéndolo. El entendimiento debe moverse con el flujo del proceso, debe unirse a este y fluir con el mismo
>
> Frank Herbert

6.1. Conclusiones

A lo largo de este trabajo se prueba que los Modelos Ocultos de Markov HMM son capaces de modelar un entorno de forma que permita resolver el problema de navegación. Se encuentra, como era de esperarse, que el proceso de un robot móvil y su desplazamiento puede ser modelado utilizando cadenas de Markov y las herramientas para su estudio.

Como era de esperarse, pues el proceso es un proceso que cumple la propiedad de Markov. Esta propiedad dice que un proceso estocástico es considerado un proceso markoviano(cadena de Markov) si la distribución de probabilidades de los estados futuros depende solamente del estado actual y no de los estados anteriores del sistema para llegar al estado presente.

Tanto en la simulación como en las pruebas con robot real, los modelos mostraron ser bastante robustos a ruido. Proveniente de varias fuentes a la vez, en el caso de las pruebas reales, la suma de errores de desplazamiento del motor, de derrapamiento de las ruedas, de los sensores, además de que al ser un ambiente no controlado las observaciones eran afectadas tanto por gente caminando como pequeños cambios en el ambiente como puertas y sillas, aún así el sistema fue capaz de distinguir con bastante éxito varios ambientes y las últimas orientaciones dentro de ellos.

Al desglosar las partes que componen un algoritmo de EKF SLAM se busca encontrar puntos donde los HMM pudieran ayudar. Las observaciones y su naturaleza son también un punto interesante, pues la tecnología actual tiende al 3D, con los costos de procesamiento relacionados. Se propone como trabajo futuro igualmente, estudiar el comportamiento del sistema con información 3D.

Los porcentajes de estimaciones correctas resultan convincentes, y abre cuestionamientos de como elegir de manera óptima muchos de los valores cuya variación y consecuencia se presentaron en capítulos anteriores. También se abre la puerta a modelos de Markov de mayor dimensionalidad, y otros modelos derivados como los procesos de Markov, que relacionan la señal de control además de los HMM comentados.

Los tiempos de cálculo para las estimaciones hacen viable su implementación en tiempo real.

En un nivel más general, este trabajo me ha permitido conocer algoritmos de aprendizaje automático, machine learning, así como un contacto directo con problemas de Big Data, términos ambos, en voga, y objetivo no sólo de estudio sino de artículos y conversaciones en diversos campos de la industria, el marketing, finanzas, etc. Existe cada vez más paquetería especializada para su análisis, como el caso de Tensor Flow de Google, y una cantidad siempre creciente de información.

Finalmente al estudiar los métodos actuales de resolver el problema de SLAM, no sólo fue posible buscar los puntos donde más probablemente pudieran utilizarse los HMM, también genera una mejor y más profunda comprensión del problema, y la solución aceptada mayormente.

En general se listan algunos objetivos cumplidos, con la idea de poder revisar a trabajo futuro algunos de los no cumplidos.

- El sensor láser utilizado es capaz, y de sobra, de generar lecturas de consistencia y discernibilidad suficientes para probar el método estudiado. La relación costo-calidad-peso es inmejorable una vez que se prueba suficiente para mediciones capaces modelar un entorno no estructurado. En comparación con otros sensores de nubes de puntos como un Kinect, habría que comparar qué tan conveniente resulta la mejora en desempeño de uno y otro contra el precio y complejidad de computo. Se sugiere como trabajo futuro hacer un comparativo tomando en cuenta estos factores.

- La cuantización vectorial reduce exitosamente las muestras sin preprocesar.La cuantización vectorial en general, y el algoritmo LBG en particular resultan suficientes para generar modelos que funcionan correctamente. Si bien son métodos considerados antiguos por algunos, y poco eficientes en comparación con métodos como los árboles de búsqueda, la capacidad de funcionar con datos sin preprocesar hacen del método uno que funciona plenamente para el sistema propuesto. Sin embargo, la inquietud de comparar este método con algunos otros y medir estadísticamente ventajas y desventajas del mismo se sugiere como trabajo futuro.

- Localización y navegación en un ambiente con mapa topológico. El sistema y metodología aquí propuestos funcionan correctamente bajo condiciones de ruido bastante extremas(simuladas y reales). Se identificaron algunos factores que afectan el rendimiento del sistema. Y se identifica la necesidad de una conducta reactiva más eficaz para efectos del modelado. Así mismo se obtuvo información valiosa para comparar con los métodos comentados a continuación como trabajo futuro.

- Localización navegación y modelado de una región de la que no se tienen conocimientos previos. La metodología aquí planteada permite elaborar un clasificador robusto capaz de estimar por regiones las variables de navegación. Se propone un método lo menos dependiente a odometría posible, quedando pendiente como trabajo futuro la elaboración de un método independiente por completo de la odometría.

- Slam, este objetivo falta en cubrirse con cabalidad, pues la complejidad del

tema, y la diversidad de métodos utilizados hacen difícil decidirse por uno u otro en la tarea de implementar una solución basada en los Modelos Ocultos de Markov.

6.2. Trabajo Futuro

Con la idea de poder comparar el método propuesto con el método más utilizado hoy en día en ambiente ROS, se debe investigar y comparar los puntos clave con los landmarks aquí sugeridos. La tarea de ahondar en algoritmos de visión computacional, y los métodos de comparación agrupamiento y modelado ahí planteados así como la mejor comprensión de ROS son tarea obligada. Con este fin se dieron los primeros pasos para migrar el sistema a una nueva versión de ROS. Capaz de trabajar con el simulador Gazebo, y de implementar los algoritmos propuestos en este trabajo desde un nodo codificado en Python y las librerías de OPENCV. De forma similar, la paquetería OpenPCL debe ser investigada para poder comparar con modelos basados en Nubes de Puntos.

De igual forma se busca implementar el sistema sustituyendo el láser Range Finder por una cámara de video, un kinect y utilizando algoritmos de visión computacional para formar los vectores de observaciones.

Existen numerosos artículos que tratan de modelos ocultos de Markov de mayores dimensiones, o incluso modelos híbridos, que relacionan a las variables aleatorias un proceso de decisión. Por lo que es muy importante revisar implementaciones con este tipo de modelos "mejorados".

En la actualidad el poder computacional ha crecido, lo que ha llevado a otras formas de resolver el problema. Como son los algoritmos de aprendizaje automático, algoritmos genéticos,etc. Es precisamente esta el área que desea investigarse.

Es de especial interés comparar el método descrito(modelos ocultos de Markov), con algunos de los métodos más utilizados hoy en día, así como buscar la combinación con estos para obtener mejores resultados. Los métodos de redes neuronales convolucionales, debido a la capacidad de analizar en paralelo la información obtenida de las imágenes utilizando el procesador gráfico(GPU).

Modelos híbridos conformados por modelos ocultos de Markov y redes neurona-

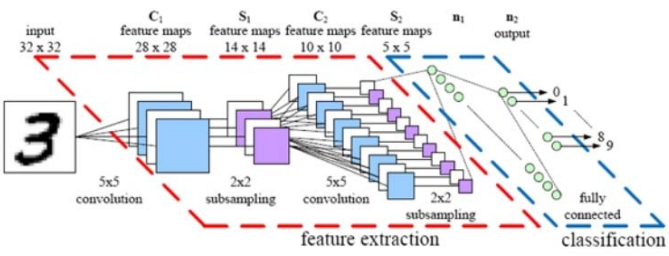


Figura 6.1: Extracción de características utilizando Redes Neuronales Convolucionales

les, han demostrado ser muy eficaces en el reconocimiento de patrones. Y como se comentó anteriormente, es posible hacer un enfoque similar utilizando observaciones obtenidas de la imagen de la escena. En este, la red neuronal funciona como un estimador probabilístico. Y busca estimar las probabilidades de las ecuaciones típicas de los modelos ocultos de Markov P(Q|O). Es decir, la probabilidad de la secuencia de estados dada una secuencia de observaciones. De esta forma las probabilidades de transición son calculadas por un perceptrón multicapa entrenado con este fin, vía el algoritmo de propagación hacia atrás. La figura 6.1 muestra el sistema híbrido modelos ocultos de Markov (HMM) y Redes Neuronales(ANN).

Apéndice A

DSP's

Los DSP's , procesadores de señales digitales, son microprocesadores especializados en el manejo de datos digitales y los algoritmos de procesamiento digital de señales. Los microprocesadores(las computadoras) han probado su eficacia principalmente en dos grandes áreas, el manejo de datos y las operaciones matemáticas. Las computadoras presentes en nuestros escritorios hoy en día contienen microprocesadores capaces para estas dos tareas, pero optimizados por razones de demanda, para el manejo de una de estas áreas. Sería económicamente inviable un producto optimizado para ambas, dado que la mayoría de las computadoras de escritorio tienen más necesidad de bases de datos y procesadores de texto que de implementar transformadas de Fourier y filtros digitales.

La principal diferencia es que las tareas que típicamente desempeña una computadora de escritorio tiene que ver con manejo de datos, y ocasionales operaciones matemáticas. Un procesador de texto, por ejemplo, su desempeño tiene poco que ver con las operaciones matemáticas y mucho que ver con mover datos y revisar banderas. (si esto if (a=b) entonces... $A \rightarrow B$).

En comparación el desempeño en la implementación de un filtro FIR está condicionada por el número de sumas y multiplicaciones requeridas.[21]. Por lo tanto un DSP tiene prioridad en optimizar las instrucciones suma y multiplicación. La capacidad de predecir el tiempo de ejecución es crucial también para este tipo de procesadores[21].

$$y[n] = a_0 + a_1x[n-1] + a_2x[n-2] + a_3x[n-3] + a_4x[n-4] + ...$$

.

La familia de DSP's de Texas Instruments C5x , y en particular el Tex01 y TEx01B, son procesadores optimizados para el usdo de señales digitales DSP's, con apoyo a características que hacen muhísimo más eficiente el calculo del algoritmo de Viterbi. Este procesador de arquitectura tipo Harvard garantiza un ancho de banda constante en la adquisición de datos, imprescindible para sistemas de datos muestreados.

Algunas cualidades de las arquitecturas RISC se encuentran presentes en esta familia de DSP's. en el sentido que las instrucciones relacionadas al calculo del decoder de Viterbi que tienen registros especializados, optimizando su ejecución[18].

Apéndice B

GPU's

Unidades de procesado gráfico,ampliamente utilizadas hoy en día en las computadoras para realizar operaciones sobre imágenes.Se consideran tipo SIMD, single instruction multiple data.Su principal ventaja es que permite la llamada programación en paralelo, aquí la radica la optimización a las imágenes. El ejemplo viene de los GPU's NVIDIA.

Un gpu aplica la misma instrucción a varias partes de la imagen en paralelo, por eso se les conoce como procesadores vectoriales[18].

CUDA es la plataforma de computación en paralelo desarrollada por NVIDIA, que sirve para aprovechar el GPU de la computadora con un paradigma de programación en paralelo.

Figura B.1: GPU NVIDA GFORCE

Apéndice C

Códigos

C.1. Cuantizador Vectorial Lynd Buzo Gray

```
1  function cc2=cuantizador(o,k)
2
3  cc=.99999*mean(o);
4  cc=cat(1,cc,1.00001*mean(o));
5  cont=0;
6  while ( cont<20)
7  aux=zeros(size(cc));
8  hist=zeros(size(cc,1),1);
9  for j=1:size(o,1)
10 d=sum((cc(1,:)-o(j,:)).^2);
11 for n=2:size(cc,1)
12 d=cat(1,d,sum((cc(n,:)-o(j,:)).^2));
13 end
14 [M E]=min(d);
15 cumulo(j)=E;
16 aux(E,:).+=o(j,:);
17 hist(E)++;
18 end
19 cc2=aux./hist;
```

```
20 cont++;
21 if (sum(sum( abs(cc.-cc2)))<.001)
22 cont=29
23 end
24 cc=cc2;
25 end
26 plot (cumulo);
27 while (size(cc,1) <=k)
28 cont=0;
29 cc=cat(1,1.0001.*cc,.999.*cc);
30 while ( cont<20)
31 aux=zeros(size(cc));
32 hist=zeros(size(cc,1),1);
33 for j=1:size(o,1)
34 d=sum((cc(1,:)-o(j,:)).^2);
35 for n=2:size(cc,1)
36 d=cat(1,d,sum((cc(n,:)-o(j,:)).^2));
37 end
38 [M E]=min(d);
39 cumulo(j)=E;
40 aux(E,:).+=o(j,:);
41 hist(E)++;
42 end
43 cc2=aux./hist;
44 cont++;
45 if (sum(sum( abs(cc.-cc2)))<.001)
46 cont=29;
47 end
48 cc=cc2;
49 end
50 end
```

C.2. Forward, Algoritmo de Adelanto.

```
from hmm import HMM
import numpy as np

#the Viterbi algorithm
def viterbi(hmm, initial_dist, emissions):
    probs = hmm.emission_dist(emissions[0]) * initial_dist
    stack = []

    for emission in emissions[1:]:
        trans_probs = hmm.transition_probs * np.row_stack(probs)
        max_col_ixs = np.argmax(trans_probs, axis=0)
        probs = hmm.emission_dist(emission) * trans_probs[max_col_ixs,

        stack.append(max_col_ixs)

    state_seq = [np.argmax(probs)]

    while stack:
        max_col_ixs = stack.pop()
        state_seq.append(max_col_ixs[state_seq[-1]])

    state_seq.reverse()

return state_seq
```

C.3. Viterbi

```
function [delta phi]=viterbi(A,B,sec)
delta=B(:,sec(1));
for T=1:size(sec,1)-1
for j=1:size(delta,1)
[M E]=max( delta(:,T).*A(:,j ));
%%ESPECIAL ATENCIoN AL RANGO DINaMICO
delta (j,T+1)= M    *    B(j,sec(T+1)) *10*T;
phi(j,T+1)=E;
end
end
```

Apéndice D

Glosario

- ALS. Auto Covariance least-squares
- DSP's. Procesadores De Señales Digitales
- EKF. Extended Kalman Filter
- EM. Expectation Modification
- FA. Forward Algoritm
- HMM. Hidden Markov's Models.
- Landmarks. Puntos de interés
- LBG. Lynd Buzo Gray
- MAP. Máximo a Posteriori
- MDP. Markov's Decision Process
- SLAM. Simultaneous Location and Mapping
- Trellis.Rejilla
- VQ. Vector Quantizer, Cuantizador Vectorial

Bibliografía

[1] S. Bernstein. Sur léxtension du théorème limite du calcul des probabilités aux sommes de quantités dépendantes. *Mathematische Annalen*, 97:1–59, 1927. URL http://eudml.org/doc/182666.

[2] Karel Capek. *Rossumovi univerzální roboti*. PRAGA, 1920.

[3] D.B.Paul. Speech recognition using hidden markov models. 1990.

[4] A. P. Dempster, N. M. Laird, y D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[5] G. O. S. Ekhaguere. On notions of markov property. *Journal of Mathematical Physics*, 18(11):2104–2107, 1977. doi:10.1063/1.523189. URL https://doi.org/10.1063/1.523189.

[6] Martin A. Fischler y Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981. ISSN 0001-0782. doi:10.1145/358669.358692. URL http://doi.acm.org/10.1145/358669.358692.

[7] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973. ISSN 0018-9219. doi:10.1109/PROC.1973.9030.

[8] R. Gray. Vector quantization. *IEEE ASSP Magazine*, 1(2):4–29, 1984. ISSN 0740-7467. doi:10.1109/MASSP.1984.1162229.

[9] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[10] Kai Lai Chung. Markov chains with stationary transition probabilities. 1969.

[11] Jeong Hoo Lee, Weon Heum Park, Jong Ha Moon, y M. H. Sunwoo. Efficient dsp architecture for viterbi decoding with small trace back latency. En *The 2004 IEEE Asia-Pacific Conference on Circuits and Systems, 2004. Proceedings.*, tomo 1, págs. 129–132 vol.1. 2004. doi:10.1109/APCCAS.2004.1412709.

[12] S. E. Levinson, L. R. Rabiner, y M. M. Sondhi. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *The Bell System Technical Journal*, 62(4):1035–1074, 1983. ISSN 0005-8580. doi:10.1002/j.1538-7305.1983.tb03114.x.

[13] Y. Linde, A. Buzo, y R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, 1980. ISSN 0090-6778. doi:10.1109/TCOM.1980.1094577.

[14] Michael Montemerlo. Fastslam: A factored solution to the simultaneous localization and mapping problem with unknown data association. 2003.

[15] Raul Mur-Artal, J. M. M. Montiel, y Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *CoRR*, abs/1502.00956, 2015.

[16] Dominik S. Nuß. A random finite set approach for dynamic occupancy grid maps. En *ArXiv*. 2017.

[17] Yongsheng Pan, Won-Ki Jeong, y Ross Whitaker. Markov surfaces: A probabilistic framework for user-assisted three-dimensional image segmentation. *Computer Vision and Image Understanding*, 115(10):1375 – 1383, 2011. ISSN 1077-3142. doi:https://doi.org/10.1016/j.cviu.2011.06.003. URL http://www.sciencedirect.com/science/article/pii/S1077314211001408.

[18] Weon Heum Park, Myung Hoon Sunwoo, y Seong Keun Oh. Efficient dsp architecture for viterbi decoding with small trace back latency. *The 2004 IEEE Asia-Pacific Conference on Circuits and Systems, 2004. Proceedings.*, 1:129–132 vol.1, 2004.

[19] L. Rabiner y B. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3, 1986.

[20] R. Smith, M. Self, y P. Cheeseman. Autonomous robot vehicles. cap. Estimating Uncertain Spatial Relationships in Robotics, págs. 167–193. Springer-Verlag, Berlin, Heidelberg, 1990. ISBN 0-387-97240-4. URL http://dl.acm.org/citation.cfm?id=93002.93291.

[21] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, San Diego, CA, USA, 1997. ISBN 0-9660176-3-3.

[22] L.M. Surhone, M.T. Timpledon, y S.F. Marseken. *Wiener Filter: Norbert Wiener, Noise, Andrey Kolmogorov, Frequency Response, Stochastic Process, Cross-correlation, Deconvolution, Wiener Deconvolution, Expected Value, Quantization Error*. Betascript Publishing, 2010. ISBN 9786130319366. URL https://books.google.com.mx/books?id=hwQYQwAACAAJ.

[23] Michael Wooldridge y Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995.

[24] P. Zarchan y H. Musoff. *Fundamentals of Kalman Filtering: A Practical Approach*. Nº v. 190 en Fundamentals of Kalman filtering: a practical approach. American Institute of Aeronautics and Astronautics, Incorporated. ISBN 978-1-56347-455-2. URL https://books.google.com.mx/books?id=AQxRAAAAMAAJ.