Universidad Nacional Autónoma de México

Facultad de Estudios Superiores Acatlán

ALGORITMOS ESTOCÁSTICOS DE LOCALIZACIÓN Y SU APLICACIÓN EN EL SISTEMA MULTIAGENTE DE FÚTBOL ROBÓTICO POKTAPOK.

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Licenciado en matemáticas aplicadas y computación

PRESENTA:

Luis Gómez Aussenac

Asesor:

José Sebastián Bejos Mendoza

Santa Cruz Acatlán, Edo. de México 2015

Agradecimientos

Quiero agradecer y dedicar este trabajo principalmente a mis padres, Hipólito y Laura, sin su apoyo, esfuerzo y sacrificio hubiera sido imposible lograr la culminación de mi carrera, de este trabajo y de esta etapa en mi vida. A mi hermana Daniela, quien me motiva para ser un ejemplo a seguir para ella.

Agradecer a todos los profesores que colaboraron en mi desarrollo académico, a la Facultad de Estudios Superiores Acatlán y a la UNAM en general, por permitir desarrollarme en sus aulas como estudiante universitario. Al profesor José Sebastián Bejos Mendoza, mi maestro y asesor, quien confió en mí, brindándome su apoyo y conocimientos en todo momento para poder llevar a cabo mi trabajo de investigación dentro del Laboratorio de Algoritmos para la Robótica.

También dar las gracias a mis compañeros del laboratorio, quienes día a día me mostraban que con esfuerzo y dedicación se logran las metas, por muy difíciles que parezcan. Principalmente a mis amigos Abel, Gilberto y Luis, quienes estuvieron en todo momento para resolver o ampliar mis dudas y así perfeccionar mi trabajo.

No puedo dejar a un lado a mis abuelos, quienes con su confianza y apoyo económico hicieron posible este reto. A mis abuelitas que con sus palabras de cariño y sus bendiciones lograron mi bienestar y seguridad en mis trayectos a la universidad. A todos mis tíos, en especial a mi tío Eduardo y mi tía Olga, que apoyaron de alguna u otra manera con mis viáticos.

Finalmente agradecer a mi novia, Melisa, que en este último año ha sido mi apoyo y motivación para seguir adelante y cumplir mis metas.

Índice general

Resumen

En este trabajo se explican a detalle filtros gaussianos y no paramétricos, los cuales son técnicas probabilísticas para hacer frente a la incertidumbre dentro de un sistema en presencia de ruido. Estas técnicas son utilizadas para diseñar un algoritmo de localización en el sistema multiagente de fútbol robótico PokTaPok. El algoritmo desarrollado en este trabajo, está basado en la localización Monte Carlo, la cual se deriva de un filtro no paramétrico conocido como filtro de partículas. En el análisis del algoritmo propuesto se encontró que este no genera un error en la localización que se comporte como ruido blanco, sin embargo se genera un error con desviación estándar y media menor a 0.1 y -0.01, respectivamente.

Introducción

La robótica es un concepto el cual consiste en aplicar la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones, trabajos o tareas específicas. Dentro de la robótica existen diversas ramas, por ejemplo la robótica móvil, la cual se basa en robots autónomos. Este tipo de robots son capaces de realizar tareas específicas sin necesidad de recibir instrucciones en tiempo real o ser controlados, se desenvuelven de manera automática en su entorno. En esta rama existen problemáticas, entre las cuales destacan: la navegación, la toma de decisiones, la localización, etc.

El problema del que se hablará a lo largo de este trabajo es el problema de la localización, el cual consiste en conocer la ubicación o posición del robot en el ambiente donde se encuentra. Este es un problema indispensable y necesario de resolver para cualquier sistema robótico, debido a que sin una buena localización, no es posible que el robot realice adecuadamente sus tareas asignadas. Por lo tanto se considera el primer problema a resolver en un robot autónomo.

Conocer la posición de un robot, dentro de su entorno, es un problema a causa de un concepto llamado ruido o incertidumbre, el cual se ocasiona porque los sensores y motores de los robots no son 100 % precisos en todas las ocasiones. Un robot al percibir un objeto dentro de su entorno, no tiene la certeza de percibir adecuadamente, también al realiza alguna acción, no es completamente seguro de haber realizado el movimiento correctamente. Es probable que exista un error. Los errores de los sensores y motores de los robots, pueden ser ocasionados por diversos factores: falta de batería, descomposturas, mal funcionamiento, entornos complejos que impidan el movimiento adecuado del robot, etc.

Existen técnicas probabilísticas utilizadas para dar solución al problema de localización, en este trabajo se explican y desarrollan algunas de ellas. Además se explica cómo se implementó y adaptó un algoritmo de localización para el sistema PokTaPok, basado en un algoritmo no paramétrico conocido como localización Monte Carlo o Filtro de Partículas.

El sistema PokTaPok es un proyecto desarrollado en el Laboratorio de Algoritmos para la Robótica de la FES Acatlán, para participar en el torneo de robótica *RoboCup* dentro de la liga de simulación 2D. Esta categoría, es una competencia que se desarrolla de manera virtual, simulando un partido de fútbol entre dos sistemas multiagentes, teniendo como objetivo que se desarrollen técnicas y algoritmos enfocados a la inteligencia artificial para vencer al rival. La competencia considera las mismas reglas que la FIFA durante el partido.

Capítulo 1

Preliminares

Para poder interpretar los algoritmos y fórmulas utilizadas en este trabajo y en el contexto de la robótica, es necesario dar un breve resumen sobre definiciones de probabilidad y conceptos de robótica, entre los cuales destacan los teoremas de probabilidad total y condicional, que son los conceptos de mayor peso en el caso de la localización en robots.

Las siguientes definiciones y conceptos que se muestran a continuación, fueron tomados de [14] y [25].

1.1. Conceptos básicos de probabilidad

Experimento. Un *experimento* es realizar alguna acción con la finalidad de obtener información de interés. Existen dos tipos de experimentos: *aleatorios* y *deterministas*. Un ejemplo de estos sería: lanzar una moneda n número de veces y contar la cantidad de "soles" que cayeron, siendo este un experimento aleatorio debido a que no se tiene la certeza de lo que se obtendrá; calentar una olla con agua sería una experimento determinista porque el resultado es que el agua se calentará y por consecuencia hervirá. De esta manera un experimento determinista es aquel cuyo resultado es previsible, mientras en un experimento aleatorio no se puede predecir el resultado que se obtiene cada vez que se repita. Pero siempre se puede analizar el comportamiento o tener en cuenta todos los resultados posibles.

Espacio muestral. Sea un experimento aleatorio ε, se define el *espacio muestral* $\mathcal{S}$ como el conjunto de todos los resultados posibles distintos de ε.Se debe tomar en cuenta que el espacio muestral puede ser distinto para el mismo experimento dependiendo de la información que se quiera obtener.

Suceso. Si un espacio muestral $\mathcal{S}$ es finito o infinito numerable, todo subconjunto se puede considerar como un *suceso*.

Sucesos mutuamente excluyentes.Dos sucesos, A y B, son *mutuamente excluyentes* si no pueden ocurrir juntos. Esto se expresa como $A \cap B = \emptyset$; es decir, la intersección de A y B es el conjunto vacío.

Frecuencia relativa. Sea un experimento ε, el cual se repite n veces y sean A y B dos sucesos asociados con ε. Sean n_A y n_B el número respectivo de veces que el suceso A y el suceso B ocurrieron en las n repeticiones, $f_A = \frac{n_A}{n}$ se llama la *frecuencia relativa* del suceso A en las n repeticiones de ε. La frecuencia relativa f_A tiene las siguientes propiedades:

1. $0 \leq f_A \leq 1$.
2. $f_A = 1$ si y sólo si A ocurre cada vez en las n repeticiones.
3. $f_A = 0$ si y sólo si A nunca ocurre en las n repeticiones.
4. Si A y B son dos sucesos que se excluyen mutuamente y si $f_{A\cup B}$ es la frecuencia relativa asociada al suceso $A \cup B$, entonces $f_{A\cup B} = f_A + f_B$.
5. f_A, basada en las n repeticiones del experimento y considerada para una función de n, converge en cierto sentido probabilístico a $P(A)$ cuando $n \to \infty$

Probabilidad. Sea ε un experimento y $\mathcal{S}$ un espacio muestral asociado con ε. Con cada suceso A se asocia un número real, designado por $P(A)$ y llamado la *probabilidad* de que A satisfaga las siguientes propiedades:

1. $0 \leq P(A) \leq 1$
2. $P(\mathcal{S}) = 1$
3. Si A y B son sucesos que se excluyen mutuamente, $P(A \cup B) = P(A) + P(B)$.

Esta definición de probabilidad es conocida como definición axiomática.
Una definición clásica y la cual puede comprenderse mejor puede ser:

La *probabilidad* de que ocurra un suceso A, se calcula como el cociente entre la cantidad de casos posibles favorables m de dicho suceso y el total de resultados igualmente posibles n del espacio muestral.

$$P(A) = \frac{m}{n}$$

Esto bajo la suposición de que todos los experimentos del suceso A son igualmente probables.

Partición. Los sucesos $B_1, B_2, \ldots, B_k$ representan una *partición* del espacio muestral $\mathcal{S}$ si:

1. $B_i \cap B_j = \emptyset$ para todo $i \neq j$.
2. $\bigcup_{i=1}^{k} B_i = S$.
3. $P(B_i) > 0$ para todo i.

En otras palabras en una partición de $\mathcal{S}$ en sucesos B_i, si se realiza un experimento ε, ocurre uno y solo uno de los sucesos B_i.

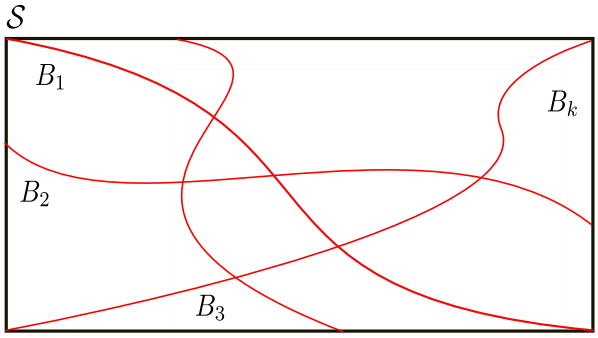


Figura 1.1: Partición de un espacio muestral $\mathcal{S}$

Probabilidad conjunta. La probabilidad conjunta unicamente nos expresa la probabilidad de que un evento de X tome un valor x y un evento de Y un valor y, es decir:

$$p(x, y) = p(X = x, Y = y)$$

Probabilidad condicional. Sean los sucesos A y B, se define la *probabilidad condicional* del suceso B respecto al A, con una $P(A) > 0$, o la probabilidad condicional del suceso A respecto al B, con $P(B) > 0$, como:

$$P(B \mid A) = \frac{P(A \cap B)}{P(A)} \quad \text{o} \quad P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

La expresión anterior se lee como la probabilidad de B dado que ocurrió A. Para tener una idea más clara sobre como se obtiene esta expresión, se retomará el concepto de frecuencia relativa. Suponga un experimento ε se ha repetido n veces. Sean n_A, n_B y $n_{A\cap B}$ el número respectivo de veces que los sucesos A, B y $A \cap B$ han ocurrido en las n repeticiones. La frecuencia relativa del suceso $A \cap B$ se expresa como $f_{A\cap B} = \frac{n_{A\cap B}}{n}$ y la frecuencia relativa del suceso A como $f_A = \frac{n_A}{n}$, por lo tanto si quiere expresar la frecuencia relativa con la cual el suceso B se repite dado que el suceso A es un hecho, se expresa como: $\frac{f_{A\cap B}}{f_A}$. Se sabe que una de las propiedades de la frecuencia relativa es que cuando n tiende al infinito la frecuencia relativa está próxima a la probabilidad del suceso, por lo tanto:

$$\frac{\frac{n_{A\cap B}}{n}}{\frac{n_A}{n}} = \frac{f_{A\cap B}}{f_A} = \frac{P(A \cap B)}{P(A)} = P(B \mid A)$$

Una consecuencia importante de la definición de probabilidad condicional se obtiene escribiéndola de la siguiente manera:

$$P(A \cap B) = P(B \mid A)P(A) \quad \text{equivalente a:} \quad P(A \cap B) = P(A \mid B)P(B)$$

A esta expresión se le conoce con el nombre de *teorema de multiplicación de probabilidades*. El cual se puede aplicar para el cálculo de la probabilidad de ocurrencia simultánea de dos sucesos A y B, es decir: $p(A, B) = p(A \cap B)$

Sucesos independientes. En algunos experimentos aleatorios, la realización de un suceso B no afecta a la probabilidad de un suceso A, es decir, $P(A \mid B) = P(A)$. Esto es la noción de sucesos independientes, la cual se define como sigue: Sean dos sucesos A y B se dicen independientes si y solo si:

$$P(A \cap B) = P(A) \cdot P(B)$$

Sin tomar en cuenta la restricción de que $P(A)$ o $P(B)$ deben ser mayores que cero.

Variable aleatoria. Al realizar algunos experimentos los resultados no precisamente son valores numéricos, por ejemplo: basta con alguna clasificación, como el espacio muestral defectuoso y no defectuoso. Sin embargo en ocasiones es necesario medir algo y anotarlo como un número. Siguiendo el ejemplo anterior, se puede asignar un valor de 1 a defectuoso y 0 a no defectuoso. En otras palabras, en muchas ocasiones se desea asignar un número real x a cada uno de los elementos s del espacio muestral $\mathcal{S}$. Es decir, $x = X(s)$ es el valor de una función X del espacio muestral a los números reales.

Formalmente hablando:

Sea un experimento ε y $\mathcal{S}$ el espacio muestral asociado con el experimento. Una función X que asigna a cada uno de los elementos $s \in \mathcal{S}$, un número real $X(s)$, se llama *variable aleatoria*.

La probabilidad de una variable aleatoria se escribe de la siguiente forma:

$$P(X(s) = x)$$

Variable aleatoria discreta. Sea X una variable aleatoria. Si el número de valores posibles de X es finito o infinito numerable, se llama a X una *variable aleatoria discreta*. Es decir, los valores posibles de X son $x_1, x_2, \ldots, x_n, \ldots$, donde la lista termina para el caso finito y la lista continúa indefinidamente para el caso infinito numerable.

Variable aleatoria continua. Se dice que X es una *variable aleatoria continua* si existe una función f, llamada función de densidad de probabilidad de X, que satisface las siguientes condiciones:

1. $f(x) \geq 0$ para todo x
2. $\int_{-\infty}^{\infty} f(x)dx = 1$
3. Para valores a,b, tal que $-\infty < a < b < +\infty$,
   se tiene $P(a \leq X \leq b) = \int_a^b f(x)dx$

En pocas palabras una variable aleatoria continua puede tomar infinidad de valores.

Función de distribución. Toda variable aleatoria tiene asociada una función de distribución. La función de distribución de una variable aleatoria X es la función: $F(x) : \mathbb{R} \to [0, 1]$, definida como: $F(x) = P(X \leq x)$. Esta función también es conocida como *función de probabilidad acumulada*.
La definición formal se puede escribir como: Una función $F(x) : \mathbb{R} \to [0, 1]$ es llamada función de distribución si cumple las siguientes propiedades:

- $\lim_{x \to +\infty} F(x) = 1$
- $\lim_{x \to -\infty} F(x) = 0$
- Si $x_1 \leq x_2$ entonces: $F(x_1) \leq F(x_2)$

.

Proceso estocástico. Un proceso estocástico es una colección de variables aleatorias $\{x_t : t \in T\}$ parametrizadas por un conjunto $T = \{0, 1, 2, ...\}$ o bien $T = [0, \infty)$, considerados como tiempos, llamado espacio parametral, y con valores en un conjunto $\mathcal{S}$ llamado espacio de estados.
Los posibles espacios de estados que se consideran están dentro del conjunto $\mathbb{R}$.
En ocasiones se define de manera informal a un proceso estocástico como *función aleatoria*.

Cadena de Markov. Suponiendo que la variable aleatoria x_t de un proceso estocástico se conoce, las variables aleatorias anteriores a x_t no tienen influencia en la variable aleatoria futura x_{t+1}. Esto se puede expresar de la siguiente manera: Para cualesquiera variables $x_0, x_1, \ldots, x_{n-1}$ (pasadas), x_n (presente) y x_{n+1} (futura), se cumple la igualdad:

$$p(X_{n+1} = x_{n+1} \mid X_0 = x_0, \ldots, X_n = x_n) = p(X_{n+1} = x_{n+1} \mid X_n = x_n)$$

de esta forma la probabilidad del evento futuro $(X_{n+1} = x_{n+1})$ sólo depende del evento inmediato anterior $(X_n = x_n)$. La información de los eventos pasados es irrelevante.

1.1.1. Probabilidad total

Sea A algún suceso con respecto a $\mathcal{S}$ y sea $B_1, B_2, \ldots, B_k$ una partición de un espacio muestral $\mathcal{S}$. Por lo tanto, se puede escribir:

$$A = A \cap B_1 \cup A \cap B_2 \cup \cdots \cup A \cap B_k$$

Sin importar que existan intersecciones $A \cap B_i$ que sean vacíos, la igualdad anterior se cumple. Lo importante es que al ser una partición de $\mathcal{S}$, los sucesos B_i son mutuamente excluyentes y por lo tanto se aplica la propiedad de la probabilidad de intersección entre eventos mutuamente excluyentes, la cual nos da como resultado:

$$P(A) = P(A \cap B_1) + P(A \cap B_2) + \cdots + P(A \cap B_k)$$

Cada término $P(A \cap B_j)$ se puede expresar como $P(A \mid B_j)P(B_j)$ (tomado de la definición de probabilidad condicional), dando como resultado el llamado *Teorema de la probabilidad total*:

$$P(A) = P(A \mid B_1)P(B_1) + P(A \mid B_2)P(B_2) + \cdots + P(A \mid B_k)P(B_k)$$

o bien

$$P(A) = \sum_{i=1}^{k} P(A \mid B_i)P(B_i) \quad \text{caso discreto} \tag{1.1}$$

$$P(A) = \int_B P(A \mid B)P(B)dB \quad \text{caso continuo} \tag{1.2}$$

1.1.2. Regla o teorema de Bayes

Existe un teorema muy importante, el cual se conoce como *teorema de Bayes* o de la probabilidad inversa. Consiste en que se tiene una partición de un espacio muestral $\mathcal{S}$ de B_k sucesos. La probabilidad de que un suceso B_i ocurra dado que el suceso A ha ocurrido se expresa como $P(B_i \mid A) = \frac{P(A \cap B_i)}{P(A)}$, donde esta expresión se puede escribir como:

$$P(B_i \mid A) = \frac{P(A \mid B_i)P(B_i)}{\sum_{j=1}^{k} P(A \mid B_j)P(B_j)} \quad \text{caso discreto} \tag{1.3}$$

$$P(B_i \mid A) = \frac{P(A \mid B_i)P(B_i)}{\int_B P(A \mid B)P(B)dB} \quad \text{caso continuo} \tag{1.4}$$

A las probabilidades de B_i se les conoce como probabilidades a *priori*. Las $P(B_i \mid A)$ son llamadas probabilidades a *posteriori* y a las $P(A \mid B_i)$ *verosimilitudes*.

En muchas ocasiones y por cuestiones prácticas, la formula del teorema de Bayes se escribe como:

$$P(B_i \mid A) = \eta P(A \mid B_i)P(B_i)$$

Para cualquiera de las probabilidades a posteriori el denominador es el mismo, por lo tanto se puede expresar como una constante η, a la cual, en ocasiones se le da el nombre de *normalizador* y expresa lo siguiente: $\eta = [P(A)]^{-1}$.

1.2. Interacción del agente con su entorno

A lo largo de este trabajo se utilizará el concepto de agente debido a que se habla de la aplicación de algoritmos a un sistema multiagente y por lo tanto será de mucha utilidad tener una definición sencilla pero concreta de qué es un agente.

Un agente es un ente que simula capacidades motrices y sensoriales, el cual es capaz de tomar decisiones con base en su entorno, en otras palabras, es todo aquello que puede percibir su entorno mediante sensores, procesar la información y desenvolverse en el ambiente por medio de acciones.

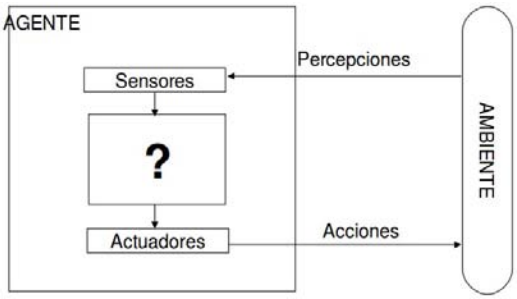


Figura 1.2: Diagrama de un agente con sus respectivas características. El signo de interrogación representa los procesos que realiza, los cuales varían dependiendo del tipo de agente.

Para el desarrollo de algoritmos capaces de resolver tareas específicas de un robot, es necesario interpretar su entorno de una manera abstracta; para el caso de este trabajo el entorno del agente. La información o datos que el agente puede percibir y la manera en como interacciona con los objetos que lo rodean, deben de ser considerados para la toma de decisiones. La información se percibe a través de los ***sensores*** del agente y la interacción con el entorno se realiza a través de sus ***actuadores*** (controles o motores). Este tipo de conceptos se utilizan a lo largo de este texto junto con los siguientes:

Se llamará ***estado*** del agente a la colección de información que percibe el agente de su entorno, de los objetos y características propias del agente, por ejemplo: colores, identificadores, ubicación, velocidades, distancia y dirección de los objetos que lo rodean, la temperatura del ambiente, sonidos, incluso si alguno de sus sensores funciona o no, etc.

En otras palabras el estado del agente contiene toda la información posible sobre su entorno y él mismo, la cual en ocasiones se modifica con el paso del tiempo y ocasiona un cambio en el entorno y afecta las decisiones del agente. En el contexto de este trabajo se denotará el estado en el tiempo t como x_t, y se llamarán variables de estado.

Dentro de la información que contiene el estado del agente se encuentra la ubicación dentro de su entorno, la cual se llamará ***pose*** del agente. El número de elementos que se contemplan en la pose del agente depende de todas las posibles posiciones que pueda tomar (espacio de configuración). El *vector pose* más común y sencillo es el que contiene la coordenada de un plano cartesiano y un ángulo, los cuales representan la posición y la orientación.

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

Existen objetos especiales conocidos con el nombre de ***referencias***. Este nombre se le da a los objetos dentro del entorno del agente que no cambian su ubicación y de los cuales el agente tiene conocimiento sobre su coordenada.

Se llamará a x_t ***estado completo*** si este contiene la mayor información sobre el entorno y el agente, es decir, no existe un estado, conjunto de estados, actuadores o percepciones pasadas que proporcionen información que afecte la toma de decisiones del agente. Esto es importante porque si se considera un estado completo a x_t, es posible pronosticar de la mejor manera al estado x_{t+1}. De esta manera, este proceso se puede considerar como una *cadena de Markov*, x_{t+1} solo depende del estado inmediato anterior.

Para el problema que se tratará en este trabajo, se supondrá que los estados son completos, además el tiempo será discreto ($t = 0, 1, 2, \ldots$), de tal manera que todas las interacciones que realice el agente con su entorno se asumirá que se realizan en el intervalo de tiempo $(t-1, t]$.

Se utilizará la variable u_t para denotar los controles o bien las acciones que realice el agente, las cuales pueden ser: velocidad de rotación, de traslación, distancia de desplazamiento, cantidad de impulso eléctrico, etc., dependiendo del tipo de robot o agente es como se establece el contenido de la llamada ***variable de control*** u_t. Se denota a la secuencia de acciones o controles realizados por el agente como $u_{t_1:t_2}$ para $t_1 \leq t_2$, que significa:

$$u_{t_1:t_2} = u_{t_1}, u_{t_1+1}, u_{t_1+2}, \ldots, u_{t_2} \tag{1.5}$$

Se llamará ***variable de percepción*** al vector z_t, el cual representará la percepción del agente en el tiempo t. En la mayoría de las ocasiones esta variable contiene la distancia, color, identificador, etc. del objeto percibido. La cantidad de objetos que perciba el agente será la dimensión del vector z_t. El conjunto de todas las percepciones entre el tiempo t_1 hasta t_2, para $t_1 \leq t_2$ se denotan:

$$z_{t_1:t_2} = z_{t_1}, z_{t_1+1}, z_{t_1+2}, \ldots, z_{t_2} \tag{1.6}$$

Como se mencionó anteriormente, el ruido en la robótica es el principal factor que ocasiona los problemas que se derivan en el accionar de los robots.
Gracias a que se conoce que existe este factor, la idea principal o la más lógica para contemplar los datos que se obtienen de los sensores y controles del agente es mediante distribuciones de probabilidad.

1.2.1. Leyes generativas probabilísticas

La información que contiene el estado del agente cambia a través del tiempo y se sabe que el estado x_t se genera estocásticamente como una cadena de Markov por el estado x_{t-1}, es decir, el estado en el tiempo t depende unicamente del estado en el tiempo $t-1$.

La idea principal para predecir x_t es que está condicionado por todos los estados pasados, controles y percepciones, por lo tanto se puede expresar la probabilidad de x_t como: $p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t})$. Por conveniencia se asume que el agente ejecuta primero el control u_t y después toma una percepción z_t, por lo tanto esta expresión no considera la percepción en el tiempo t.

Bajo la suposición de que los estados son completos, el estado x_{t-1} resume a la perfección todos los controles y percepciones anteriores hasta el tiempo t, entonces todas las variables de la expresión anterior, excepto u_t, se pueden omitir si se conoce x_{t-1}. De este modo se reduce:

$$p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t \mid x_{t-1}, u_t) \tag{1.7}$$

De igual forma es indispensable tener el modelo por el cual está gobernado o regido la percepción del agente, el cual quedaría expresado de la siguiente manera:

$$p(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t \mid x_t) \tag{1.8}$$

La expresión 1.8 se deduce porque se conoce el estado x_t y por lo tanto se tiene la información sobre todos los controles y percepciones hasta el tiempo t.

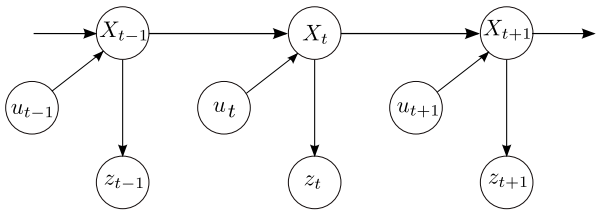


Figura 1.3: Proceso Estocástico

A la probabilidad $p(x_t \mid x_{t-1}, u_t)$ se le conoce como *probabilidad del estado de transición*. Esta probabilidad representa cómo el estado x_t cambia a través del tiempo en función al control del agente u_t.

La probabilidad $p(z_t \mid x_t)$ es llamada *probabilidad de percepción*. Esta especifica como la percepción del agente depende unicamente del estado x_t.

En resumen: El estado en el tiempo t es estocásticamente dependiente del estado en el tiempo $t-1$ y del control u_t y la percepción z_t depende estocásticamente del estado en el tiempo t.

1.2.2. Distribuciones creencia

Se trabajará con un concepto conocido como ***creencias*** o ***distribuciones creencias***, las cuales expresarán la probabilidad del estado x_t. Las creencias se representan a través de distribuciones de probabilidad condicional. Una distribución creencia asigna una probabilidad (o valor de densidad) a cada posible estado con respecto al estado verdadero. Se denotará una creencia sobre una variable de estado x_t como $bel(x_t)$ y se define de la siguiente manera:

$$bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t}) \tag{1.9}$$

Esta es una probabilidad sobre el estado x_t en el tiempo t, condicionada a todos los datos pasados percibidos $z_{1:t}$ y a todos los controles pasados $u_{1:t}$. Esto se puede leer como: "*la probabilidad del estado x_t dado que ha percibido $z_{1:t}$ y a realizado $u_{1:t}$* ".

En la expresión 1.9, se puede observar que está incorporada la percepción z_t. Ocasionalmente, será de utilidad el cálculo de una creencia antes de incorporar z_t, justo después de ejecutar el control u_t. Esta será denotada de la siguiente manera:

$$\overline{bel}(x_t) = p(x_t \mid z_{1:t-1}, u_{1:t}) \tag{1.10}$$

La expresión 1.10, donde no se contempla z_t, recibe el nombre de *creencia estimada*. En el contexto de filtros probabilísticos (del cual se habla en 1.3), se conoce como *predicción* ($\overline{bel}(x_t)$). Y una vez que se incorpora z_t, *corrección* ($bel(x_t)$).

1.3. Filtro de Bayes

1.3.1. Algoritmo del filtro de Bayes

Existe un algoritmo general para calcular creencias, el cual se llama *filtro de Bayes*. Su nombre se debe a que está basando en el teorema de Bayes. Este algoritmo calcula la distribución creencia (bel) a partir de datos proporcionados por percepciones y controles. El algoritmo de filtro de Bayes es el siguiente:

Algoritmo 1 Filtro de Bayes($bel(x_{t-1}), u_t, z_t$)

1: **for all** x_t **do**
2: $\quad \overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$
3: $\quad bel(x_t) = \eta p(z_t \mid x_t) \overline{bel}(x_t)$
4: **end for**
5: **return** $bel(x_t)$

El algoritmo 1 describe el filtro de Bayes, el cual es recursivo, esto es, la creencia en el tiempo t es calculada a partir de la creencia en el tiempo $t-1$. Las entradas del algoritmo son la creencia bel_{t-1}, junto con el control u_t y la percepción z_t. La salida del algoritmo es la creencia $bel(x_t)$.

El algoritmo de filtro de Bayes, y en general todos los algoritmos conocidos como filtros, se puede dividir en dos etapas importantes **predicción** y **corrección**.

En la línea 2, se calcula la creencia estimada $\overline{bel}(x_t)$ sobre el estado x_t a partir de la creencia previa sobre el estado x_{t-1} y el control u_t. El $\overline{bel}(x_t)$ que se asigna al estado x_t se obtiene de la integral (suma) del producto de dos distribuciones: la previa asignada a x_{t-1}, y la probabilidad que el control u_t que induce en la transición de x_{t-1} a x_t. A esta etapa se le conoce como *etapa de predicción.*

En la línea 3, se lleva a cabo la *etapa de corrección*, donde se multiplica la creencia $\overline{bel}(x_t)$ por la probabilidad de percepción $p(z_t \mid x_t)$. El algoritmo realiza esto para cada posible estado x_t. Como se verá más adelante en la derivación del algoritmo, no siempre el producto resultante es una probabilidad (Puede no integrar a 1). Entonces, el resultado es normalizado, es por ello que aparece la constante de "normalización" η (sección 1.1.2).

Finalmente en la línea 5 el algoritmo regresa la creencia final $bel(x_t)$.

El algoritmo al ser recursivo requiere de un estado inicial x_0 en el tiempo $t = 0$. Si se conoce con certeza el valor del estado inicial, se puede comenzar colocando

toda la probabilidad sobre ese valor correcto x_0 y asignar probabilidad de cero en cualquier otro estado. Si no se tiene conocimiento sobre el estado inicial x_0 y por lo tanto también se desconoce la creencia inicial $bel(x_0)$, se puede empezar con una distribución uniforme sobre el dominio de x_0. En la sección de anexos en la página 87 se explica a detalle la justificación del algoritmo y el porqué funciona, a esto se le llama derivación.

1.3.2. Ejemplo

Para un caso práctico del uso del algoritmo 1, se redacta el siguiente problema:
Se desea saber si una habitación con un foco, está oscura o iluminada. Dentro de la habitación hay un botón, que al presionarlo se enciende o apaga el foco. Se tiene un robot, que tiene un sensor para detectar si el foco se encuentra encendido o apagado y además el robot cuenta con un brazo mecánico, con el cual puede oprimir el botón para encender o apagar el foco. Si el robot se introduce a la habitación y únicamente se sabe lo que percibe y lo que realiza, sin conocer el estado en el que se encuentra la habitación (oscura o iluminada), ¿cómo saber el estado de la habitación?

Se pueden definir las variables de la siguiente manera:

$$\text{estado} := x_t = \{\text{iluminada}, \text{oscura}\}$$

$$\text{percepción} := z_t = \{\text{encendido}, \text{apagado}\}$$

$$\text{acción} := u_t = \{\text{presionar}, \text{nada}\}$$

Además se conocen las probabilidades de percepción y transición.

Probabilidades de percepción:

$p(z_t = encendido \mid x_t = iluminada) = .9$
$p(z_t = encendido \mid x_t = oscura) = .1$
$p(z_t = apagado \mid x_t = iluminada) = .15$
$p(z_t = apagado \mid x_t = oscura) = .85$

Probabilidades de transición:

$p(x_t = iluminada \mid x_{t-1} = iluminada, u_t = presionar) = .1$
$p(x_t = iluminada \mid x_{t-1} = oscura, u_t = presionar) = .9$
$p(x_t = iluminada \mid x_{t-1} = iluminada, u_t = nada) = .95$
$p(x_t = iluminada \mid x_{t-1} = oscura, u_t = nada) = .05$

$p(x_t = oscura \mid x_{t-1} = iluminada, u_t = presionar) = .9$
$p(x_t = oscura \mid x_{t-1} = oscura, u_t = presionar) = .1$
$p(x_t = oscura \mid x_{t-1} = iluminada, u_t = nada) = .05$
$p(x_t = oscura \mid x_{t-1} = oscura, u_t = nada) = .95$

En este ejemplo se asume que las probabilidades se calcularon de forma experimental para conocer la precisión, o bien, el error de los sensores y controles del robot. En la sección 1.5 se analizarán modelos para calcular estas probabilidades mediante métodos específicos.

El estado en que se encuentra la habitación es desconocido, por lo tanto es conveniente establecer una probabilidad igual a cada estado posible, $p(x_0 = oscura) = .5$ y $p(x_0 = iluminada) = .5$. Esto representa el estado inicial (el estado en el tiempo $t = 0$). Para conocer el estado de la habitación en el tiempo $t = 1$, es decir, x_1, es necesario saber la acción que realiza el robot y la percepción. Si se tiene una acción $u_1 = presionar$ y una percepción $z_1 = encendido$. Estos tres datos son los parámetros de entrada del algoritmo 1 ($bel(x_{t-1}), u_t, z_t$). Por simplicidad se cambiarán las palabras completas por la letra con la que inicia, por ejemplo: iluminada → i , oscura → o , etc.

Ejecutando el algoritmo 1

- Para $t = 1$ y $x_1 = i$ con $u_1 = p$ $z_1 = a$

$$\begin{aligned}\overline{bel}(x_1 = i) =& p(x_1 = i \mid u_1 = p, x_0 = i)p(x_0 = i) + \\ & p(x_1 = i \mid u_1 = p, x_0 = o)p(x_0 = o)\end{aligned}$$

$$\begin{aligned}\overline{bel}(x_1 = i) =& (.1)(.5) + (.9)(.5) \\ =& .5\end{aligned}$$

$$\begin{aligned}bel(x_1 = i) =& p(z_1 = a \mid x_1 = i)\overline{bel}(x_1 = i) \\ =& (.15)(.5) = .075\end{aligned}$$

- Para $t = 1$ y $x_1 = o$ con $u_1 = p$ $z_1 = a$

$$\begin{aligned}\overline{bel}(x_1 = o) =& p(x_1 = o \mid u_1 = p, x_0 = i)p(x_0 = i) + \\ & p(x_1 = o \mid u_1 = p, x_0 = o)p(x_0 = o)\end{aligned}$$

$$\begin{aligned}\overline{bel}(x_1 = i) =& (.9)(.5) + (.1)(.5) \\ =& .5\end{aligned}$$

$$\begin{aligned}bel(x_1 = o) =& p(z_1 = a \mid x_1 = o)\overline{bel}(x_1 = o) \\ =& (.85)(.5) = .425\end{aligned}$$

Se puede ver que la creencia mayor pertenece al estado $x_1 = oscura$. El detalle que hay es que la suma de los valores resultantes de la creencia de cada estado, no dan uno. Esto es porque en el ejemplo anterior no se aplicó el normalizador η. Si se realiza el cálculo de η y se multiplican los valores, se tiene lo siguiente:

$$\begin{aligned}\eta = [p(z_1)]^{-1} &= [\sum^{n} p(z_1 \mid x_0^{[n]})p(x_0^{[n]})]^{-1} \\ &= [p(z_1 \mid x_0 = i)p(x_0 = i) + p(z_1 \mid x_0 = o)p(x_0 = o)]^{-1} \\ &= [(.15)(.5) + (.85)(.5)]^{-1} = [.5]^{-1}\end{aligned}$$

Al multiplicar los resultados de $bel(x_1 = i)$ y $bel(x_1 = o)$ por el normalizador, se obtiene lo siguiente:

$$\begin{aligned}\eta bel(x_1 = i) = [.5]^{-1}(.075) &= .15 \\ &+ \\ \eta bel(x_1 = o) = [.5]^{-1}(.425) &= .85 \\ &|| \\ &1\end{aligned}$$

Las probabilidades finales para cada estado posible x_t suman uno. Y el estado $x_1 = oscura$ es el que tiene mayor probabilidad.

1.4. Liga de fútbol de simulación 2D

La liga de *fútbol de simulación 2D* es una categoría del torneo de robótica *RoboCup*. Esta categoría está enfocada en la aplicación de inteligencia artificial y estrategia en equipo. Consiste en simular un partido de fútbol virtual entre dos equipos de 11 jugadores (agentes) cada uno, los cuales son autónomos. Durante

el juego se aplican las mismas reglas de la FIFA.
La categoría consta de dos partes importantes, el servidor y el monitor.

Figura 1.4: Partido de la liga de simulación 2D.

1.4.1. Servidor soccerserver

El *soccerserver* es un sistema que permite que dos equipos virtuales compitan en un juego de fútbol simulado dentro de una computadora. El servidor *soccerserver* se puede descargar para diversos sistemas operativos como se muestra en [3]. Debido a que el partido se lleva a cabo en un estilo *cliente-servidor*, no existen restricciones de lenguaje de programación para la construcción de los equipos. El único requisito es que las herramientas utilizadas para el desarrollo de un equipo soporten la comunicación *cliente-servidor* a través de un *socket UDP/IP*. Cada agente representa un jugador de un equipo, que a su vez representa un cliente en el servidor, un proceso por separado, el cual se conecta por un puerto especifico. Después que un cliente (jugador, agente) se conecta al servidor todos los mensajes entre el cliente y el servidor se empiezan a enviar a través del puerto. Un equipo puede tener máximo hasta 12 clientes, es decir, 11 jugadores (10 de campo y un portero) y un entrenador (*coach*). Los jugadores envían solicitudes al servidor con respecto a las acciones que desean realizar, como: correr, patear el balón, girar, etc. El servidor recibe estos mensajes, se encarga de las peticiones y actualiza el entorno como consecuencia de las acciones realizadas. También el servidor se encarga de proveer a todos los jugadores información sensorial, por ejemplo: información visual sobre la posición de los objetos en el campo o bien datos sobre los recursos que tiene el jugador, como energía, velocidad, etc. Es importante mencionar que el servidor es un sistema en tiempo real que trabaja en intervalos de tiempo discretos, que se les llama ciclos. Cada ciclo tiene una duración especifica, que en este caso es de 10 milisegundos.

El servidor también es el encargado de realizar todos los cálculos matemáticos, es decir, todos los resultantes de las fuerzas físicas que simulan el movimiento y sus efectos como colisión entre objetos, fuerza del viento, fricción con el pasto, etc. De igual manera es el encargado de jugar el papel de arbitro durante el partido, detectando y marcando las faltas, sancionando a los jugadores, marcando los tiros de esquina, los fuera de lugar, etc. Aplica todas las reglas durante el partido. Es importante resaltar que se siguen las mimas reglas que se aplican en un partido oficial de la FIFA.

El partido consta de dos tiempos con duración de 3000 ciclos cada uno (6000 ciclos por partido). Si existe un empate el servidor también es capaz de simular tiempos extra y penales.

Algo muy importante que simula el servidor son objetos conocidos como *referencias*. Para este caso los objetos *referencia* son unas *banderas* y las líneas del campo de juego. Estos son objetos fijos que contienen un identificador único y tienen coordenadas específicas que se conocen por los desarrolladores de los equipos. Las coordenadas dentro del campo son similares a las de un plano cartesiano, tomando como el origen $(0, 0)$ el centro del campo, para el lado izquierdo los negativos en el eje x, y con la diferencia en el eje y, donde los negativos son hacia arriba. El campo de juego que simula el servidor y los objetos *referencia* se pueden observar en la figura 1.5.

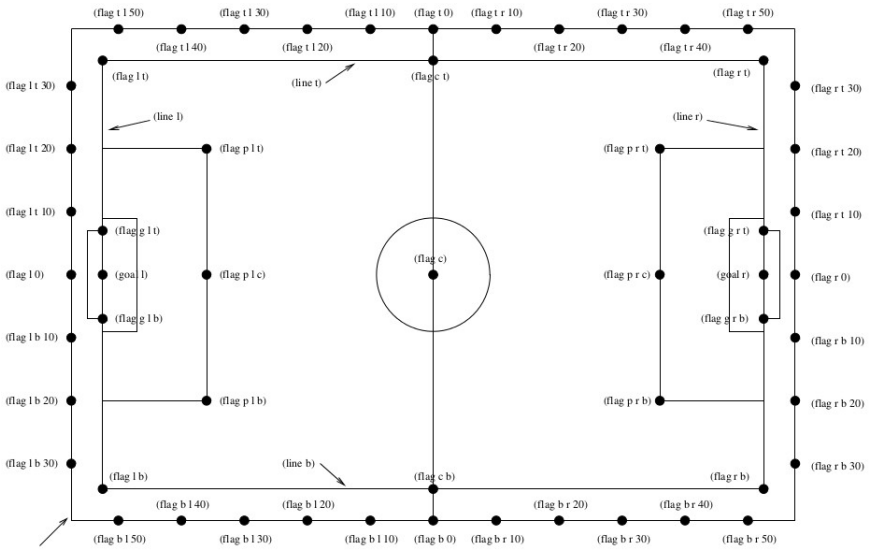


Figura 1.5: Posición y nombre de las banderas en el campo de juego simulado por el servidor.

1.4.2. Monitor soccermonitor

El *soccermonitor* es una herramienta de visualización que ayuda al usuario a ver lo que está pasando en el servidor durante el juego, en otras palabras, el monitor toma los datos numéricos que calcula el servidor y los convierte en una visualización amigable a la vista de las personas.

El monitor incluye en su visualización aspectos como el campo de juego, el marcador, el ciclo en el que se encuentra el partido, nombres de los equipos, y por supuesto todas las posiciones de los objetos en la cancha, como los agentes y el balón en tiempo real durante cada ciclo.

Esta herramienta de visualización también contiene características extras como son: aplicar acercamientos a cualquier zona del campo durante el juego, observar la energía restante de los jugadores, observar el cono de visión de algún jugador, etc.

Figura 1.6: Monitor soccermonitor con sus características.

1.4.3. Agente 2D

La simulación de cada jugador se lleva a cabo a través de un *agente2D*, el cual se compone de ciertos elementos que simulan las capacidades reales de un ser humano, por ejemplo: un cono de visión, el cual representa la capacidad visual de una persona. También se compone de un sensor auditivo, que le ayuda

a recibir mensajes. Un sensor de cuerpo que se representa por un área circular al rededor de él, incluso el agente simula la capacidad de girar el cuello, únicamente girando el cono de visión sin mover el cuerpo.

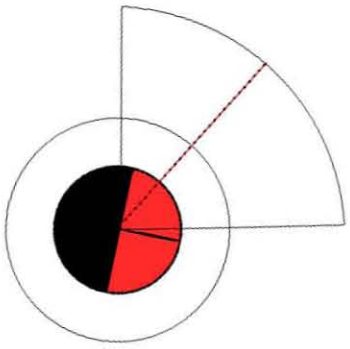


Figura 1.7: Representación de un agente 2D.

Los sensores que componen al agente, llegan en forma de mensaje desde el servidor hacia el cliente. Los tres mensajes que representan a los sensores del agente son los siguientes:

sense body.- hace referencia a un sensor corporal, el cual proporciona información al agente sobre el estado del juego y características propias de él mismo. El servidor manda la información de este sensor de la siguiente manera:

(**sense body** time view mode stamina speed head angle kickCount dashCount sayCount turnNeckCount catchCount moveCount changeViewCount)

donde

- time → ciclo en el que se encuentra la simulación.
- view mode → es la calidad y apertura del cono de visión.
- stamina → energía restante del agente.
- speed → magnitud y dirección del vector de velocidad.
- head angle → ángulo que hay entre la dirección del cuerpo y el cuello.
- *Count → son contadores de cada uno de los comandos, respectivamente.

- *see*.- representa al sensor visual, el cual contiene la información sobre los objetos que se encuentren dentro del cono de visión del agente. Entre más cerca se encuentre un objeto del agente y se encuentre dentro del cono de visión, la información que el servidor le proporciona al agente sobre ese objeto es mayor y de mejor calidad. El mensaje proveniente del servidor es de la siguiente forma:

  (**see** time objInfo)

  donde *time* es ciclo donde se recibe la información de la simulación, y objInfo varia dependiendo del objeto y de la distancia a la que se encuentre. Este parámetro puede ser de la siguiente manera:

  (objName distance direction dirChange bodyFacingDir bodyHeadDir)
  (objName distance direction distChange dirChange)
  (objName distance direction)
  (objName direction)

  donde

  - distance → distancia al objeto.
  - direction → dirección a la que se encuentra el objeto.
  - dirChange → dirección con respecto al cuello.
  - bodyFacingDir → dirección del cuerpo del agente.
  - bodyHeadDir → dirección del cuello del agente.

- *hear*.- es el sensor por el cual el agente es capaz recibir cadenas de caracteres, las cuales se interpretan como mensajes que el jugador puede escuchar.

La simulación de un jugador no se lleva a cabo únicamente mediante la simulación de las capacidades sensoriales de una persona, sino también a través de las capacidades motrices. Es por esto que el agente cuenta con comandos, los cuales utiliza como solicitudes para el servidor de las acciones que quiere realizar. Los comandos que el agente puede realizar son los siguientes:

- *catch*.- Comando único del portero, el cual se utiliza para atrapar el balón cuando se encuentra dentro un área donde el balón puede ser atrapado. Necesita como parámetro un ángulo, que representa la dirección de donde el agente quiere atrapar el balón.

- *change_ view.*- Sirve para cambiar la apertura y la calidad del cono de visión del agente. Recibe como parámetros la apertura (reducida, normal, amplia) y la calidad (buena, poca).
- *dash.*- Es el comando que se utiliza para que el agente avance. Recibe como parámetro un poder, que es el poder de impulso que se le da al agente para que se desplace.
- *kick.*- Se utiliza para patear el balón, recibe como parámetros un poder y un ángulo, que significa la potencia con la cual el agente desea patear el balón y en que dirección.
- *move.*- Este comando es especial porque se puede utilizar únicamente cuando el partido no ha comenzado o en el medio tiempo. Sirve para desplazar el agente a cierta posición especifica en el campo. El servidor lo utiliza para desplazar a los agentes cuando se comete alguna falta y no pueden estar cerca del balón. Tiene como parámetros una coordenada (x, y), la cual representa la posición en el campo a donde se desplaza el agente.
- *say.*- Recibe como parámetro una cadena de caracteres la cual representa un mensaje que el agente quiere comunicar.
- *turn.*- Se implementa cuando el agente desea girar. Su parámetro es una cantidad de grados, que representa los grados que el agente girará.
- *turn_ neck.*- Es similar al comando turn, sólo que este sirve para girar el cuello cierta cantidad de grados. El giro se realiza con respecto al ángulo del cuerpo.

La liga de *fútbol de simulación 2D*, cuenta con muchas más características y especificaciones, las cuales pueden ser consultadas en [3].

1.5. Modelos de movimiento y percepción

Como se vio en el filtro de Bayes (sección 1.3) existen dos etapas en las cuales se puede dividir el algoritmo. La primer etapa, predicción, la cual se basa en dar una hipótesis del estado. Ocurre unicamente contemplando los controles del agente. Para ello es necesario tener un modelo de movimiento para dicho agente, el cual está dado por las acciones o movimientos que realiza, es decir los controles u_t.

La segunda etapa, corrección, en la cual se ajusta la hipótesis que se obtuvo con ayuda de la percepción del agente, z_t. Para esta etapa de corrección se necesita tener conocimiento sobre un modelo de percepción.

1.5.1. Modelo de velocidad de movimiento

Este modelo es para obtener las probabilidades de transición, similares a las mostradas en la sección 1.3.2 página 13. En lugar de obtenerlos de manera experimental. El modelo de velocidad de movimiento asume que el control del agente se basa en velocidades. Una velocidad de traslación y una de rotación, las cuales nos representan la distancia recorrida y los grados que se han girado después de cierto intervalo de tiempo, respectivamente.

Este modelo es uno de los más comunes en el mundo de la robótica, porque infinidad de robots se controlan en base a estas dos velocidades, que también se les puede llamar velocidad de desplazamiento y de giro.

Para explicar este modelo se denotará la velocidad de traslación en el tiempo t como v_t y la velocidad de rotación como w_t, recordando que son la distancia desplazada y los grados que se giraron en un tiempo fijo Δt. Así se puede denotar un vector velocidad u_t, también llamado vector de control, como:

$$u_t = \begin{pmatrix} v_t \\ w_t \end{pmatrix} \tag{1.11}$$

Arbitrariamente, una velocidad de rotación positiva o un giro con grados positivos es en sentido de las manecillas del reloj, y una velocidad de traslación positiva o distancia recorrida positiva, corresponde a un movimiento hacia delante.

El modelo de velocidad de movimiento sirve para calcular la probabilidad de transición, $p\left(x_t \mid x_{t-1}, u_t\right)$. Un algoritmo simple para calcular esta probabilidad se muestra a continuación:

Algoritmo 2 motion_model_velocity(x_t, u_t, x_{t-1})

1: $\mu = \frac{1}{2}\frac{(x-x')\cos\theta+(y-y')\sin\theta}{(y-y')\cos\theta-(x-x')\sin\theta}$
2: $x^* = \frac{x+x'}{2} + \mu\left(y-y'\right)$
3: $y^* = \frac{y+y'}{2} + \mu\left(x'-x\right)$
4: $r^* = \sqrt{\left(x-x^*\right)^2 + \left(y-y^*\right)^2}$
5: $\Delta\theta = \text{atan2}\left(y'-y^*, x'-x^*\right) - \text{atan2}\left(y-y^*, x-x^*\right)$
6: $\hat{v} = \frac{\Delta\theta}{\Delta t} r^*$
7: $\hat{w} = \frac{\Delta\theta}{\Delta t}$
8: $\hat{\gamma} = \frac{\theta'-\theta}{\Delta t} - \hat{w}$
9: **return** $\textbf{prob}(v-\hat{v}, \alpha_1 v^2 + \alpha_2 w^2) \cdot \textbf{prob}(w-\hat{w}, \alpha_3 v^2 + \alpha_4 w^2) \cdot \textbf{prob}(\hat{\gamma}, \alpha_5 v^2 + \alpha_6 w^2)$

1.5.2. Descripción del algoritmo

El algoritmo recibe como entrada una pose inicial del robot o agente en el tiempo $t-1$ que se denota como: $x_{t-1} = (\ x \quad y \quad \theta\ )^T$, un vector de control o de acciones en el tiempo t, en este caso es un vector velocidad, denotado como: $u_t = (\ v \quad w\ )^T$ y una pose futura o hipótesis en el tiempo t que es denotada por $x_t = (\ x' \quad y' \quad \theta'\ )^T$. La salida del algoritmo representa la probabilidad condicional del estado de transición $p\left(x_t \mid x_{t-1}, u_t\right)$ que se puede interpretar como la probabilidad del robot o agente de estar en x_t después de haber realizado los controles o acciones u_t cuando se encontraba en x_{t-1}, asumiendo que los controles se llevan a cabo en un tiempo fijo Δt. Los parámetros $\alpha_1, \ldots, \alpha_6$ son errores específicos del movimiento del robot o agente. La función **prob**(x,b^2), da como resultado el valor de densidad de una variable aleatoria x de una distribución centrada cero y varianza b^2. La derivación del algoritmo 2 se encuentra en la parte de anexos en la página 87.

1.5.3. Algoritmo de muestreo

Para los algoritmos no paramétricos, como se vera en el capitulo 3, es necesario crear una serie de estados o poses hipótesis x_t y no una probabilidad de transición, lo cual basta con muestrear con base en el modelo de movimiento, posibles poses del agente. El algoritmo de muestreo (3) genera poses ($x_t = (\ x \quad y \quad \theta\ )$) del agente.

Algoritmo 3 sample_motion_model_velocity(u_t, x_{t-1})

1: $\hat{v} = v$+**sample**$(\alpha_1 v^2 + \alpha_2 w^2)$
2: $\hat{w} = w$+**sample**$(\alpha_3 v^2 + \alpha_4 w^2)$
3: $\hat{\gamma}$ =**sample**$(\alpha_5 v^2 + \alpha_6 w^2)$
4: $x' = x - \frac{\hat{v}}{\hat{w}} \sin\theta + \sin\frac{\hat{v}}{\hat{w}} \sin(\theta + \hat{w}\Delta t)$
5: $y' = y + \frac{\hat{v}}{\hat{w}} \cos\theta - \cos\frac{\hat{v}}{\hat{w}} \cos(\theta + \hat{w}\Delta t)$
6: $\theta' = \theta + \hat{w}\Delta t + \hat{\gamma}\Delta t$
7: **return** $x_t = (x', y', \theta')^T$

1.5.4. Descripción del algoritmo

El algoritmo para muestrear poses $x_t = (\ x' \quad y' \quad \theta'\ )^T$ a partir de $x_{t-1} = (\ x \quad y \quad \theta\ )^T$ y un control $u = (\ v \quad w\ )^T$, recibe como entradas una pose x_{t-1} y un vector de control u_t. Regresando como resultado una nueva pose x_t.

En las líneas 1 y 2 se agrega ruido a las velocidades del vector de control, donde **sample**(b^2) genera muestras aleatorias de una distribución centrada en cero y varianza b^2. En la línea 3 se considera la orientación final γ únicamente agregando un valor aleatorio de la distribución dada por la función **sample**. De

la línea 4 a la 6 se utiliza el modelo de velocidad de movimiento, para generar la nueva pose x_t, que es el resultado del algoritmo.

1.6. Modelo de percepción extracción de características

Este modelo de extracción de características es para obtener las probabilidades de percepción, similares a las mostradas en la sección 1.3.2 página 13. En lugar de obtenerlas de manera experimental, se obtienen con este modelo.
Los agentes cuentan con sensores que les permiten percibir ciertos aspectos de los objetos que se encuentran en su entorno o mapa. Los sensores más comunes, son los sensores visuales, con los cuales los agentes son capaces de medir distancia y grados a los que se encuentran los objetos, percibir colores, tamaño, etc.

Dentro del entorno de un agente también existen objetos fijos (referencias), esto quiere decir, que se conoce la posición de estos objetos en el mapa; en el mundo de la robótica también son llamados *landmarks*. Estos objetos son muy útiles para los agentes, porque al saber sus coordenadas en el mapa, pueden ayudar a la localización de los agentes mediante métodos trigonométricos y geométricos, o incluso para la navegación del mismo.

1.6.1. Extracción de características

El modelo más común dentro de la percepción, es el modelo de extracción de características. Este modelo asume que el sensor puede medir la distancia al objeto, el ángulo relativo con respecto al cuerpo del agente y una cierta singularidad o característica especifica del objeto. Para caso práctico se supondrá que la singularidad es un valor numérico, por ejemplo: el promedio del color, pero de igual forma puede ser un valor entero que clasifique el tipo de referencia que se observa o un simple identificador.

Para formalizar esta idea, se denotará el extractor de características como una función f y z_t será el vector de medición o percepción, el cual contiene los objetos que fueron percibidos en el tiempo t. Por lo tanto la acción de extraer características en el tiempo t se denotará como $f(z_t)$. La distancia al objeto la será denotada como r, el ángulo relativo como ϕ y su singularidad como s, por lo tanto al aplicar la extracción de características en el tiempo t se tendrá:

$$f(z_t) = \{f_t^1, f_t^2, \ldots\} = \{ \begin{pmatrix} r_t^1 \\ \phi_t^1 \\ s_t^1 \end{pmatrix}, \begin{pmatrix} r_t^2 \\ \phi_t^2 \\ s_t^2 \end{pmatrix}, \ldots \} \tag{1.12}$$

Se debe de tener en cuenta, que el número de objetos que se percibe en cada intervalo de tiempo es variable.

Algo sobresaliente, es que la mayoría de los algoritmos de robótica probabilística asumen la existencia de independencia condicional entre características. La cual se aplica si el ruido en cada percepción $\begin{pmatrix} r_t^i & \phi_t^i & s_t^i \end{pmatrix}^T$ es independiente del ruido de otra percepción $\begin{pmatrix} r_t^j & \phi_t^j & s_t^j \end{pmatrix}^T$, para toda $i \neq j$.

Para continuar con el desarrollo del modelo, se va definir el mapa o entorno como una lista de referencias, $m = \{m_1, m_2, \ldots\}$, donde cada referencia puede poseer un identificador y su ubicación. La ubicación de una referencia será denotada como $m_{i,x}$ y $m_{i,y}$, que simplemente son coordenadas que expresan su ubicación en el mapa.

El vector de percepción o medición, libre de ruido, es fácil de especificar por reglas geométricas estándar. Para modelar el ruido en el vector de percepción, se agrega ruido gaussiano en cada uno de los componentes como se muestra en la ecuación 1.13:

$$\begin{pmatrix} r_t^i \\ \phi_t^i \\ s_t^i \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ \text{atan2}(m_{j,y} - y, m_{j,x} - x) - \theta \\ s_j \end{pmatrix} + \begin{pmatrix} \varepsilon_{\sigma_r^2} \\ \varepsilon_{\sigma_\phi^2} \\ \varepsilon_{\sigma_s^2} \end{pmatrix} \tag{1.13}$$

Donde ε_{σ_r}, $\varepsilon_{\sigma_\phi}$ y ε_{σ_s} son variables de error gaussiano con desviación estándar σ_r, σ_ϕ y σ_s, respectivamente.

En este tipo de modelo existe un problema conocido como *problema de asociación de datos*. Esto ocurre cuando las referencias no pueden ser identificadas; entonces existe cierto grado de error con respecto a la singularidad de la referencia observada. Para desarrollar el modelo de una manera más robusta y evitar este tipo de problemas, es necesario introducir una variable de correspondencia entre la característica f_t^i y la referencia m_j en el mapa. Esta variable la denotará como c_t^i con $c_t^i = \{1, \ldots, N+1\}$; N siendo el número de referencias en el mapa m. Si $c_t^i = j \leq N$, entonces la i-ésima característica observada en el tiempo t corresponde a la j-ésima referencia en el mapa. En otras palabras, c_t^i es el identificador de la característica observada. La única excepción ocurre cuando $c_t^i = N + 1$,en este caso, la característica observada no

corresponde a ninguna característica de mapa m.

El siguiente algoritmo sirve para calcular la probabilidad de una característica f_t^i con correspondencia conocida $c_t^i \leq N$.

Algoritmo 4 landmark_model_known_correspondence(f_t^i, c_t^i, x_t, m)

1: $j = c_t^i$
2: $\hat{r} = \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2}$
3: $\phi = \text{atan2}(m_{j,y} - y, m_{j,x} - x) - \theta$
4: $q =$**prob**$(r_t^i - \hat{r}, \sigma_r) \cdot$ **prob**$(\phi_t^i - \hat{\phi}, \sigma_\phi) \cdot$ **prob**$(s_t^i - s_j, \sigma_s)$
5: **return** q

1.6.2. Descripción del algoritmo

El algoritmo 4 recibe como entradas el vector de características definido como $f_t^i = (\ r_t^i \quad \phi_t^i \quad s_t^i\ )^T$, el identificador verdadero de la característica c_t^i, la pose del agente $x_t = (\ x \quad y \quad \theta\ )^T$ y el mapa m. Su salida es la probabilidad $p(f_t^i \mid c_t^i, m, x_t)$, que es la probabilidad de medición o percepción, que se interpreta como la probabilidad de percibir la característica f_t^i dado que tiene un identificador c_t^i en el mapa m y el agente se encuentra en x_t.
En las líneas 2 y 3 se calcula la verdadera distancia y orientación relativa de la referencia. La probabilidad de medir o percibir esa distancia y orientación son calculadas en la línea 4, donde se asume independencia en el ruido, y como se ve el algoritmo implementa la ecuación 1.13.

Capítulo 2

Filtros gaussianos

En este capítulo se hablará sobre una clase de estimadores para estados, conocidos como *filtros gaussianos*. Se les llama estimadores porque aproximan variables de estado y filtros gaussianos por utilizar la función de distribución normal o gaussiana.

Las técnicas gaussianas utilizadas para filtros tienen la idea básica que las *creencias* son representados por distribuciones normales multivariadas, donde su función de densidad se expresa como se muestra en la expresión 2.1.

$$p(x) = det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{2.1}$$

Esta densidad sobre la variable x tiene dos parámetros: La media μ y la covarianza Σ. La media μ es un vector que posee la misma dimensión que la variable x. La covarianza Σ es una matriz cuadrada, simétrica y positiva-semidefinida, que posee el mismo número de renglones que la dimensión de la variable x. Por lo tanto el número de elementos de la matriz de covarianza es igual al cuadrado del número de elementos de la variable x.

El hecho de que la distribución esté representada por una gaussiana tiene implicaciones importantes. Una de ellas es que las gaussianas son unimodales, es decir, son funciones que tienen un máximo único. Con ello se puede asumir que la probabilidad de x estará con un margen de error pequeño alrededor del estado real.

Un concepto que se tomará en este texto, es la parametrización de una gaussiana a través su media y covarianza; a esto se le llama *parametrización de momentos*. Esto es debido a que la media y la covarianza son el primer y segundo momento de una distribución de probabilidad, todos los demás momentos en una distribución normal o gaussiana son cero.

2.1. Filtro de Kalman

El filtro de Kalman (***KF***) es una técnica estudiada para implementar el filtro de Bayes y estimar variables de estado. La restricción de esta técnica radica en que es para sistemas lineales, los cuales se definirán más adelante (ecuaciones 2.2 y 2.5). También implementa el cálculo de creencias para estados continuos, es decir, no se puede aplicar para espacios discretos o híbridos.

El filtro de Kalman representa las creencias por la parametrización de momentos; en el tiempo t, la creencia es representada por la media μ_t y la covarianza por Σ_t. Se dirá que la distribución objetivo es gaussiana si las siguientes tres propiedades se cumplen, junto con es supuesto de estado completo y generado estocásticamente por el estado inmediato anterior.

1. La probabilidad de transición $p(x_t \mid u_t, x_{t-1})$ debe ser una función lineal en sus argumentos con ruido Gaussiano agregado. Esto quiere decir que:

   $$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \tag{2.2}$$

   Donde x_t y x_{t-1} son vectores de estado, y u_t es el vector de control en el tiempo t. Que se expresan de la siguiente forma:

   $$x_t = \begin{pmatrix} x_{1,t} \\ x_{2,t} \\ \vdots \\ x_{n,t} \end{pmatrix} \quad u_t = \begin{pmatrix} u_{1,t} \\ u_{2,t} \\ \vdots \\ u_{m,t} \end{pmatrix} \tag{2.3}$$

   En el estado de transición, que se define como una función lineal en la ecuación 2.2, A_t y B_t son matrices. A_t es una matriz cuadrada de $n \times n$, donde n es la dimensión del vector de estado x_t. La matriz B_t es de tamaño $n \times m$, con m siendo la dimensión del vector de control u_t. Al multiplicar el vector de estado y de control con las matrices A_t y B_t, respectivamente, se puede ver que la función de probabilidad de transición de estado es lineal en sus argumentos. Así, filtros de Kalman asume un sistema lineal.

   La variable ε_t en la ecuación 2.2 es un vector aleatorio Gaussiano que modela la incertidumbre introducida por la transición de estado. Este es de la misma dimensión que el vector de estado. Su media es cero y su covarianza la se denota como R_t.

   La ecuación 2.2 define el estado de transición, la probabilidad de estado de transición queda definido por la conexión entre la ecuación (2.2) y la

definición de la distribución normal multivariada (2.1). Así la media del estado siguiente estará dada por $A_t x_{t-1} + B_t u_t$ y la covarianza por R_t:

(2.4)

$$p(x_t \mid u_t, x_{t-1}) = det(2\pi R_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x - A_t x_{t-1} - B_t u_t)\right\}$$

2. La probabilidad de percepción $p(z_t \mid x_x)$, al igual que la probabilidad de transición, debe de ser lineal en sus argumentos y con ruido gaussiano agregado. De esta manera se puede expresar el vector de percepción como se muestra en la ecuación 2.5.

(2.5)
$$z_t = C_t x_t + \delta_t$$

Donde C_t es una matriz de tamaño $k \times n$, donde k es la dimensión del vector de percepción z_t. El vector δ_t describe el ruido de percepción. La distribución de δ_t es gaussiana multivariante con media cero y covarianza Q_t. La probabilidad de percepción entonces es dada por la siguiente distribución multivariante, descrita en la ecuación 2.6:

(2.6)
$$p(z_t \mid x_t) = det(2\pi Q_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)\right\}$$

3. Y por último, la creencia inicial $bel(x_0)$ debe ser distribuida normalmente. Se denotará la media de esta creencia como μ_0 y la covarianza como Σ_0, y así se tiene como resultado la ecuación 2.7, para la creencia inicial:

(2.7)
$$bel(x_0) = p(x_0) = det(2\pi\Sigma_0)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_0 - \mu_0)\Sigma^{-1}(x_0 - \mu_0)\right\}$$

Estas tres suposiciones son suficientes para asegurar que $bel(x_t)$ posterior es siempre gaussiano, para cualquier punto en el tiempo t. La prueba de esto se realizará en la derivación del filtro de Kalman, la cual se encuentra en el apartado de anexos en la página 87.

2.1.1. Algoritmo del filtro de Kalman

El algoritmo del filtro de Kalman o KF como también es conocido se muestra a continuación:

Algoritmo 5 Kalman_filter$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

1: $\overline{\mu}_t = A_t\mu_{t-1} + B_t u_t$
2: $\overline{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$
3: $K_t = \overline{\Sigma}_t C_t^T (C_t\overline{\Sigma}_t C_t^T + Q_t)^{-1}$
4: $\mu_t = \overline{\mu} + K_t(z_t - C_t\overline{\mu_t})$
5: $\Sigma_t = (I - K_t C_t)\overline{\Sigma}_t$
6: **return** μ_t, Σ_t

El algoritmo representa la creencia $bel(x_t)$ en el tiempo t por la media μ_t y covarianza Σ_t. Recibe como entradas la creencia en el tiempo $t-1$, representado por su media y covarianza correspondiente, μ_{t-1} y Σ_{t-1}, también recibe el vector de control u_t y el vector de percepción z_t. La salida del algoritmo es la creencia en el tiempo t, que de igual manera es representado por la media μ_t y covarianza Σ_t.

En las líneas 1 y 2, se calcula la predicción de $\hat{\mu}_t$ y $\hat{\Sigma}_t$ que representan la creencia estimada $\hat{bel}(x_t)$, que es antes de considerar el vector de percepción z_t. Únicamente se utiliza el vector de control u_t para predecir esta creencia. La media es actualizada en la línea 1 utilizando la ecuación de la función de estado de transición (2.2), con la media μ_{t-1} sustituida por el vector estado del tiempo $t-1$, x_{t-1}. La actualización de la covarianza en la línea 2, considera el hecho que el estado depende de los estados anteriores a través de la matriz lineal A_t, a esta primera parte del algoritmo se le conoce como predicción.

La creencia estimada $\hat{bel}(x_t)$ es transformada en la creencia $bel(x_t)$, de las líneas 3 a 5, mediante la incorporación del vector de percepción z_t, a esta parte del algoritmo se le conoce como ajuste, porque utiliza lo que el robot percibió para ajustar el estado x_t.

La variable K_t, que se calcula en la línea 3, se le conoce como *ganancia de Kalman* (*Kalman gain*), esta variable especifica el grado en el que la percepción (z_t) se incorpora en la nueva estimación del estado. Esta variable se analizará más a detalle en la derivación del algoritmo (apartado de anexos, página 87). En la línea 4 se manipula la media, ajustándose en proporción a la ganancia de Kalman y la desviación de la percepción. Finalmente en la línea 5 se realiza el ajuste de la covarianza del nuevo estado x_t, regresando en la línea 6 los nuevos parámetros μ_t y Σ_t de la creencia en el tiempo t, $bel(x_t)$.

2.2. Filtro de Kalman extendido

Como se vio en en el filtro de Kalman (sección 2.1), es necesario considerar ciertas restricciones, o mejor dicho ciertos supuestos. Como es el caso de que el estado es función lineal del estado previo o que las percepciones son funciones lineales del estado. Este tipo de suposiciones son muy importantes para la demostración del algoritmo.

En la vida real muy pocos robots o sistemas robóticos se rigen por un sistema lineal como el que se presentó en el filtro de Kalman. En pocas palabras los estados de transición y percepción rara la vez son lineales en la práctica. Es por esto que existe una extensión del algoritmo del filtro de Kalman, el cual se le conoce como *filtro extendido de Kalman* o **EKF**. Este algoritmo es muy similar al algoritmo del filtro de Kalman, la diferencia radica en que no supone linealidad en los estados de transición y percepción.

La suposición que contempla el EKF es que el estado de transición y percepción son regidos por funciones no lineales, que en este caso se llamarán g y h.

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t \tag{2.8}$$

$$z_t = h(x_t) + \delta_t \tag{2.9}$$

La desventaja de este supuesto, es que con funciones no lineales g y h las creencias ya no son gaussianos (normales).

Debido a la importancia de tener creencias gaussianos, la idea principal del EKF es la de *linealizar* las funciones g y h.

2.2.1. Linealización a través de la expansión de Taylor

Existen diversas técnicas para linealizar una función. En este caso EKF utiliza el método llamado *expansión de Taylor* o de primer orden, el cual consiste en construir una aproximación lineal de una función g mediante el uso de la serie de Taylor, la cual se define de la siguiente manera:

$$\begin{aligned} f(x) &\approx f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots \\ &= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \end{aligned} \tag{2.10}$$

La serie de Taylor aproxima el valor de una función $f(x)$ en el punto a con una suma infinita de términos. Al querer linealizar la función g, es decir, aproximar esa función mediante una función lineal, no es necesario utilizar por completo la serie de Taylor, basta con los dos primeros términos para obtener una función lineal, porque al usar más de dos términos de la serie, se tendría de nuevo una función no lineal. Por esta razón también se le llama método de primer orden, porque la variable se encuentra elevada a la potencia uno.
Como se puede ver, para aplicar esta técnica se necesita la primera derivada de la función g, la cual se expresa de la siguiente manera:

$$g'(u_t, x_{t-1}) = \frac{\partial(u_t, x_{t-1})}{\partial x_{t-1}} \tag{2.11}$$

De este modo, tanto la función g como su derivada o pendiente, dependen de los argumentos de la función g, es decir, de los valores de u_t y x_{t-1}. Es necesario un valor de x en el tiempo $t-1$. Una elección lógica para el valor de este argumento es considerar el estado más probable, que para el caso de las gaussianas es la media.

Utilizando la serie de Taylor para aproximar la función g y considerando el posible valor más probable, la media μ_{t-1}, se tendrá la siguiente expresión 2.12:

$$\begin{aligned} g(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + \underbrace{g'(u_t, \mu_{t-1})}_{=G_t}(x_{t-1} - \mu_{t-1}) \\ &= g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1} \end{aligned} \tag{2.12}$$

Con la expresión anterior se tiene una función lineal que aproxima a la función no lineal g, por lo tanto se podrá expresar la probabilidad del estado de transición por una aproximación de una función de densidad de una gaussiana, que se expresa de la siguiente manera:

$$\begin{aligned} p(x_t \mid u_t, x_{t-1}) \\ &\approx \det(2\pi R_t)^{-\frac{1}{2}} exp\{-\frac{1}{2}[x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]^T \\ &\quad R_t^{-1}[x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]\} \end{aligned} \tag{2.13}$$

Hay que notar que G_t es una matriz cuadrada de $n \times n$, con n como la dimensión del vector de estado x_t. A esta matriz también se le conoce con el nombre de *Jacobiano*.

El EKF implementa la misma técnica para linealizar la función de percepción h. La única diferencia es que la expansión de Taylor utiliza el valor de $\overline{\mu}$ para aproximar el valor de la función en ese punto. Entonces la aproximación de la función h queda de la siguiente manera:

$$h(x_t) \approx h(\overline{\mu}_t) + \underbrace{h'(\overline{\mu}_t)}_{=H_t}(x_t - \overline{\mu}_t) \\ = h(\overline{\mu}_t) + H_t(x_t - \overline{\mu}_t) \tag{2.14}$$

Una vez teniendo la aproximación de la función h, se puede escribir la aproximación de la función de densidad de probabilidad del estado de percepción, que queda de la siguiente manera:

$$p(z_t \mid x_t) \approx \det(2\pi Q_t)^{-\frac{1}{2}} exp\{-\frac{1}{2}[z_t - h(\overline{\mu}_t) - H_t(x_t - \overline{\mu}_t)]^T \\ Q_t^{-1}[z_t - h(\overline{\mu}_t) - H_t(x_t - \overline{\mu}_t)]\} \tag{2.15}$$

2.2.2. Algoritmo del filtro de Kalman extendido

El algoritmo EKF se muestra a continuación:

Algoritmo 6 Extended_Kalman_filter$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

1: $\overline{\mu}_t = g(u_t, \mu_{t-1})$
2: $\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
3: $K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1}$
4: $\mu_t = \overline{\mu} + K_t(z_t - h_t \overline{\mu_t})$
5: $\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t$
6: **return** μ_t, Σ_t

Este algoritmo es muy similar al algoritmo KF, la diferencia más importante radica en que la suposición de linealidad de los estados se cambia por la generalización de estados no lineales. También el EKF utiliza las matrices jacobianas en lugar de las correspondientes matrices A_t, B_t y C_t de los sistemas lineales que se ocupaban en el KF. El jacobiano G_t corresponde a las matrices A_t y B_t, y el jacobiano H_t corresponde a la matriz C_t.

Si se desea ver la derivación del algoritmo 6, se encuentra en la sección de anexos en la página 87.

2.3. Localización de Markov

Existen variantes de los filtros de Bayes que se utilizan para resolver de manera probabilística los problemas de localización. La aplicación más simple o fácil de los filtros de Bayes para el problema de localización es llamada *localización de Markov* o *localización markoviana*. El algoritmo básico de la localización markoviana se muestra a continuación:

Algoritmo 7 Markov_localization$(bel(x_{t-1}), u_t, z_t, m)$

1: **for all** x_t **do**
2: $\quad \overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}, m) bel(x_{t-1}) dx_{t-1}$
3: $\quad bel(x_t) = \eta p(z_t \mid x_t, m) \overline{bel}(x_t)$
4: **end for**
5: **return** $bel(x_t)$

La única diferencia, entre el algoritmo de *filtro de Bayes* y *localización markoviana*, es que necesita un mapa m como entrada. Este mapa tiene un papel importante en el modelo de percepción $p(z_t \mid x_t, m)$, en la línea 3 del algoritmo. En algunas ocasiones y dependiendo del problema se incluye en el modelo de movimiento $p(x_t \mid u_t, x_{t-1}, m)$, línea 2.

Este algoritmo trabaja de igual manera que el filtro de Bayes, convirtiendo una creencia del tiempo $t-1$ en una creencia en el tiempo t de manera recursiva. La creencia inicial, $bel(x_0)$, es inicializada dependiendo del problema de localización que se tenga, es decir, si se conoce con exactitud la posición inicial del robot, $\overline{x}_0$ denotando la pose inicial del robot, entonces

$$bel(x_0) = \begin{cases} 1 & \text{si } x_0 = \overline{x}_0 \\ 0 & \text{c.o.c.} \end{cases} \tag{2.16}$$

En otras ocasiones la pose inicial del robot no es conocida exactamente, entonces la creencia inicial se representa o se inicializa con una distribución gaussiana centrada en $\overline{x}_0$

$$bel(x_0) = \underbrace{\det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_0 - \overline{x}_0)^T \Sigma^{-1}(x_0 - \overline{x}_0)\right\}}_{\sim \mathcal{N}(x_0; \overline{x}_0, \Sigma)} \tag{2.17}$$

donde Σ es la covarianza, que representa la incertidumbre de la pose inicial.

En este tipo de problemas de localización, es muy común no saber la pose inicial del robot. Cuando esto ocurre se asigna una distribución uniforme sobre todo el espacio de las posibles poses sobre el mapa, esto es

$$bel(x_0) = \frac{1}{|X|} \tag{2.18}$$

donde $|X|$ representa el volumen del espacio de todas las poses dentro del mapa.

2.3.1. Algoritmo de Localización EKF

Una vez conocidos los algoritmos básicos basados en el filtro de Kalman y el algoritmo de localización markoviana, se puede presentar un algoritmo para la localización de un agente móvil. A este algoritmo se le conoce con el nombre de *localización con el filtro de Kalman extendido* o bien *localización EKF*.

Localización EKF representa las *creencias* $bel(x_t)$ a través su media μ_t y su covarianza Σ_t. Este algoritmo está basado en el algoritmo básico EKF (6), enfocado al problema de localización en robótica. Este es un caso especial de localización markoviana.

La localización EKF asume que el mapa es representado por una colección de características. En cualquier punto en el tiempo t, el robot llega a observar vectores de distancias y orientaciones de los elementos cercanos: $z_t = \{z_t^1, z_t^2, \ldots\}$. Para este algoritmo se suponen que todas las características son singularmente identificables. La identidad o autenticidad de un elemento es expresada por un conjunto de *variables de correspondencia*, denotadas por c_t^i, una para cada vector de características z_t^i. El algoritmo supone que las correspondencias son conocidas, es decir, el algoritmo asume que se conoce el mapa m donde se encuentran las *referencias*.

El algoritmo recibe como entrada la media y covarianza en el tiempo $t-1$, que se expresan como μ_{t-1} y Σ_{t-1}; también las acciones y percepciones del robot en el tiempo t, que son u_t y z_t; al igual que el mapa m y las variables de correspondencia c_t. La salida del algoritmo corresponde a la media y covarianza actualizada en el tiempo t, es decir, μ_t y Σ_t, junto con la probabilidad de las características observadas, p_{z_t}. Una cosa importante que hay que considerar para este algoritmo, es que no toma el caso especial de $w_t = 0$, que es el mismo caso especial del modelo de velocidad de movimiento. El algoritmo de localización EKF se presenta a continuación en el algoritmo 8.

Algoritmo 8 EKF_ Localization_ Known_ Correspondences

1: **EKF_ Localization** $(\mu_{t-1}, \sigma_{t-1}, u_t, z_t, c_t, m)$

2: $\theta = \mu_{t-1,\theta}$

3: $G_t = \begin{pmatrix} 1 & 0 & -\frac{v_t}{w_t}\cos\theta + \frac{v_t}{w_t}\cos(\theta + w_t\Delta t) \\ 0 & 1 & -\frac{v_t}{w_t}\sin\theta + \frac{v_t}{w_t}\sin(\theta + w_t\Delta t) \\ 0 & 0 & 1 \end{pmatrix}$

4: $V_t = \begin{pmatrix} \frac{-\sin\theta + \sin(\theta + w_t\Delta t)}{w_t} & \frac{v_t(\sin\theta - \sin(\theta + w_t\Delta t))}{w_t^2} + \frac{v_t\cos(\theta + w_t\Delta t)\Delta t}{w_t} \\ \frac{\cos\theta - \cos(\theta + w_t\Delta t)}{w_t} & -\frac{v_t(\cos\theta - \cos(\theta + w_t\Delta t))}{w_t^2} + \frac{v_t\sin(\theta + w_t\Delta t)\Delta t}{w_t} \\ 0 & \Delta t \end{pmatrix}$

5: $M_t = \begin{pmatrix} \alpha_1 v_t^2 + \alpha_2 w_t^2 & 0 \\ 0 & \alpha_3 v_t^2 + \alpha_4 w_t^2 \end{pmatrix}$

6: $\overline{\mu}_t = \mu_{t-1} + \begin{pmatrix} -\frac{v_t}{w_t}\sin\theta + \frac{v_t}{w_t}\sin(\theta + w_t\Delta t) \\ \frac{v_t}{w_t}\cos\theta - \frac{v_t}{w_t}\cos(\theta + w_t\Delta t) \\ w_t\Delta t \end{pmatrix}$

7: $\overline{\Sigma}_t = G_t\Sigma_{t-1}G_t^T + V_t M_{t-1} V_t^T$

8: $Q_t = \begin{pmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\phi^2 & 0 \\ 0 & 0 & \sigma_s^2 \end{pmatrix}$

9: **for all observed features** $z_t^i = (r_t^i \quad \phi_t^i \quad s_t^i)$ **do**

10: $j = c_t^i$

11: $q = \left(m_{j,x} - \overline{\mu}_{t,x}\right)^2 + \left(m_{j,y} - \overline{\mu}_{t,y}\right)^2$

12: $\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \arctan 2\left(m_{j,y} - \overline{\mu}_{t,y}, m_{j,x} - \overline{\mu}_{t,x}\right) - \overline{\mu}_{t,\theta} \\ m_{j,s} \end{pmatrix}$

13: $H_t^i = \begin{pmatrix} -\frac{m_{j,x} - \overline{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \overline{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y} - \overline{\mu}_{t,y}}{q} & -\frac{m_{j,x} - \overline{\mu}_{t,x}}{q} & -1 \\ 0 & 0 & 0 \end{pmatrix}$

14: $S_t^i = H_t^i \overline{\Sigma}_t \left[H_t^i\right]^T + Q_t$

15: $K_t^i = \overline{\Sigma}_t \left[H_t^i\right]^T \left[S_t^i\right]^{-1}$

16: $\overline{\mu}_t = \overline{\mu} + K_t^i\left(z_t^i - \hat{z}_t^i\right)$

17: $\overline{\Sigma}_t = \left(I - K_t^i H_t^i\right)\overline{\Sigma}_t$

18: **end for**

19: $\mu_t = \overline{\mu}_t$

20: $\Sigma_t = \overline{\Sigma}_t$

21: **return** μ_t, Σ_t

Como se sabe una de las entradas del algoritmo es la media en el tiempo $t-1$, μ_{t-1}. La cual representa la pose del robot. Se considera la pose del robot como el vector (x y θ).En la línea 2 del algoritmo se asigna la orientación del robot, a la variable θ. En las líneas 3 y 4 se calculan los Jacobianos que son

necesarios para linealizar el modelo de movimiento. En la línea 5 se determina la matriz de covarianza del ruido del movimiento ocasionado por los controles o accionar del robot. Se asigna a la variable $\overline{\mu}_t$ la pose predicha después de haber realizado el movimiento, línea 6. La línea 7 contiene la matriz correspondiente a la incertidumbre de los sensores, es decir, las varianza para la distancia, el ángulo y la referencia del objeto percibido. La actualización a través de la medición o etapa de corrección se realiza de las líneas 8 a la 21. La parte importante es el ciclo a través de todas las características i observadas en el tiempo t. En la línea se asigna a j la variable de correspondencia de la característica i-ésima en el vector de percepción z_t. En las líneas 11 y 12 se calcula el vector aproximado de percepción $\hat{z}_t^i$ y en la línea 8 se calcula el Jacobiano H_t^i correspondiente al modelo de percepción. Utilizando este Jacobiano, el algoritmo determina S_t^i, que es la incertidumbre correspondiente para la medición predicha $\hat{z}_t^i$. En la línea 15 se calcula la ganancia de Kalman, K_t^i. En las líneas 16 y 17 los estimadores de la media y covarianza, $\overline{\mu}_t$ y $\overline{\Sigma}_t$, son actualizados por cada característica. En las líneas 19 y 20 se asignan los nuevos parámetros ya actualizados para la creencia en el tiempo t y por último en la línea 21 se regresan la media y covarianza actualizados. de percepción.

La derivación del algoritmo 8 se encuentra en la sección de anexos, página 87.

Capítulo 3

Filtros no paramétricos

Existe una alternativa de filtros, a los cuales se les conoce con el nombre de filtros no paramétricos. Estos no dependen de una función de distribución definida, sino que aproximan las funciones de distribución posteriores a través de un número finito de datos o parámetros, donde cada una de ellas representa una parte o región en el espacio de estados. Algunos filtros de Bayes no paramétricos dependen de una descomposición del espacio de estados, en el cual cada uno de dichos valores corresponde a la probabilidad acumulada de la densidad posterior en una subregión compacta del espacio de estados. En otros casos, el espacio de estados es aproximado por muestras aleatorias tomadas de la distribución posterior. En este tipo de filtros la aproximación sobre la función de distribución posterior será más precisa con un mayor número de parámetros, es decir, las técnicas no paramétricas tienden a converger a la distribución posterior correcta cuando el número de parámetros tiende al infinito.

Los filtros no paramétricos son muy útiles para representar creencias multimodales complejas y no necesitan los supuestos de linealidad como los filtros gaussianos. Por estas razones son las técnicas más utilizadas en el mundo de la robótica, debido a que contemplan situaciones más realistas.

3.1. Filtro de partículas

El filtro de partículas es un filtro no paramétrico, el cual aproxima la distribución posterior por un número finito de parámetros, a los cuales se llamarán *partículas*.
La idea principal del filtro de partículas es representar la posterior $bel(x_t)$ por un conjunto de muestras aleatorias obtenidas de la misma distribución posterior. Tal representación es aproximada, y no paramétrica, por lo cual

puede representar un espacio más amplio de diversas funciones o distribuciones. Otra ventaja de la representación basada en muestreo es su capacidad para el modelado de transformaciones no lineales de las variables aleatorias, y por supuesto el modelado de funciones no unimodales.

Las muestras o en este caso partículas, como se mencionó al principio de este capítulo, se denotarán como se muestra en 3.1.

$$\mathcal{X}_t := x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]} \tag{3.1}$$

Cada partícula $x_t^{[m]}$, donde toda $1 \leq m \leq M$ y M siendo el número total de partículas en el conjunto $\mathcal{X}_t$, es una hipótesis sobre el posible estado en el mundo real en el tiempo t.

La idea principal del filtro de partículas es aproximar la creencia $bel(x_t)$ por un conjunto de partículas $\mathcal{X}_t$, donde la probabilidad de una partícula hipótesis $x_t^{[i]}$ de ser incluida en el conjunto $\mathcal{X}_t$ es proporcional a su filtro de Bayes posterior $bel(x_t)$. Expresado de otra manera, cada partícula tendrá una probabilidad que tiende a la creencia:

$$x_t^{[m]} \sim p(x_t \mid z_{1:t}, u_{1:t}) \tag{3.2}$$

Como resultado de esto se puede ver que la subregión más densa del espacio de estados, está poblada o repleta de varias muestras, por lo tanto es más probable que el estado verdadero se encuentre en esa región. Para lograr una mejor aproximación de la distribución posterior, es necesario un valor grande de M, el cual cuando $M \to \infty$ converge hacia la distribución posterior.

3.1.1. Algoritmo básico

El algoritmo básico del filtro de partículas se presenta a continuación:

Algoritmo 9 Particle_filter$(\mathcal{X}_{t-1}, u_t, z_t)$

1: $\overline{\mathcal{X}}_t = \mathcal{X} = 0$
2: **for** $m = 1$ **to** M **do**
3: **sample** $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
4: $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
5: $\overline{\mathcal{X}}_t = \overline{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6: **end for**
7: **for** $m = 1$ **to** M **do**
8: tomar i con probabilidad $\propto w_t^{[i]}$
9: agregar $x_t^{[i]}$ a $\mathcal{X}_t$
10: **end for**
11: **return** $\mathcal{X}_t$

El algoritmo recibe como entradas el conjunto de partículas $\mathcal{X}_{t-1}$, junto con el más reciente control u_t y la más reciente percepción z_t. Básicamente el algoritmo construye un conjunto temporal de partículas $\overline{\mathcal{X}}_t$, el cual se puede interpretar como la creencia estimada $\overline{bel}(x_t)$. Este se construye a partir del conjunto de partículas del tiempo $t-1$ y con ayuda del control que se realiza en el tiempo t. La muestra resultante lleva un indice m, el cual indica que fue generada por la m-ésima partícula del conjunto $\mathcal{X}_{t-1}$. Este paso que se muestra en la línea 3 del algoritmo, representado por la probabilidad del estado de transición $p(x_t \mid u_t, x_{t-1})$, indica que después de M iteraciones, el conjunto de partículas obtenido es la representación de $\overline{bel}(x_t)$.

En la línea 4 del algoritmo se calcula un factor para cada partícula $x_t^{[m]}$ que se conoce con el nombre de *factor de importancia*, denotado como $w_t^{[m]}$. Básicamente los factores de importancia incorporan a cada partícula la percepción z_t, y se agregan al conjunto $\overline{\mathcal{X}}_t$, como se ve en la línea 5. La importancia de este factor es que representa la percepción z_t para cada una de las partículas $x_t^{[m]}$. El factor está dado por $w_t^{[m]} = p(z_t \mid x_t^{[m]})$, que se conoce como la probabilidad de percepción. Es muy común llamar al factor $w_t^{[m]}$ como *peso* o *ponderación* de la partícula.

El paso importante en el algoritmo del filtro de partículas es de las líneas 7 a 10. En estas líneas se implementa un paso del algoritmo que se conoce con el nombre de *remuestreo* o *muestreo de importancia*. El algoritmo extrae con reemplazo M partículas del conjunto temporal $\overline{\mathcal{X}}_t$. La probabilidad de extraer cada partícula esta dada por la importancia de su peso, es decir, entre más

peso tenga una partícula, es más probable que sea extraída. Debido a que es un muestreo con reemplazo, algunas partículas se repetirán dentro del nuevo conjunto $\mathcal{X}_t$, y las partículas con menor peso serán descartadas. Resumiendo esta etapa: Mientras antes del paso de remuestreo las partículas tenían una distribución de acuerdo a $\overline{bel}(x_t)$ (considerando únicamente el control u_t), después del remuestreo las partículas tendrán (aproximadamente) una distribución de acuerdo a la posterior $bel(x_t) = \eta p(z_t \mid x_t^{[m]})\overline{bel}(x_t)$, ya considerando la percepción z_t.

3.1.2. Importancia del *remuestreo*

Es importante discutir más a detalle el paso del remuestreo, para comprender de mejor manera el algoritmo 9 .
El problema que se trata de resolver con el filtro de partículas es encontrar una función de densidad de probabilidad f desconocida. De la cual se conoce únicamente muestras dadas a partir de una función de densidad de probabilidad diferente, g. Por ejemplo, se quiere saber la esperanza de que $x \in A$. Se puede expresar esta probabilidad como una esperanza sobre g. Donde I es una función indicadora, que toma el valor de 1 si su argumento es verdadero y 0 en cualquier otro caso.

$$
\begin{aligned}
E_f[I(x \in A)] &= \int f(x) I(x \in A) dx \\
&= \int \underbrace{\frac{f(x)}{g(x)}}_{=w(x)} g(x) I(x \in A) dx \\
&= \int g(x) w(x) I(x \in A) dx \\
&= E_g[w(x) I(x \in A)]
\end{aligned} \tag{3.3}
$$

En la expresión anterior $w(x) = \frac{f(x)}{g(x)}$, el cual es un factor de peso o ponderación que explica el desajuste o desfase entre f y g. Las cuales son funciones de densidad de alguna distribución de probabilidad. A la función f se le llamará *distribución objetivo* y a la función g *función propuesta*.

Obtener un muestreo directamente de la función f es imposible. Lo que se realiza es generar muestras (partículas) de la función de distribución propuesta g, la cual sí se conoce. Sin embargo el conjunto de partículas resultante es distribuido de acuerdo a g, no a f. En particular, para cualquier intervalo $A \subseteq dom(X)$ el recuento empírico (suma) de las partículas que caen dentro de A converge a la integral de g sobre A, esto es:

$$\frac{1}{M}\sum_{m=1}^{M} I(x^{[m]} \in A) \rightarrow \int_A g(x)dx \tag{3.4}$$

Para compensar esta diferencia entre f y g, las partículas $x^{[m]}$ son ponderadas por el cociente

$$w^{[m]} = \frac{f(x^{[m]})}{g(x^{[m]})} \tag{3.5}$$

Utilizando esta igualdad se puede expresar la integral de la ecuación (3.4) como:

$$\left[\sum_{m=1}^{M} w^{[m]}\right]^{-1} \sum_{m=1}^{M} I(x^{[m]} \in A) w^{[m]} \rightarrow \int_A f(x)dx \tag{3.6}$$

donde el primer término sirve como el normalizador para todos los pesos importantes. En otras palabras, aunque se han generado las partículas de la densidad g, las partículas apropiadamente ponderadas convergen a la densidad f. Esta aproximación converge a la deseada $E_f[I(x \in A)]$ para conjuntos arbitrarios A.

En el filtro de partículas la densidad f corresponde a la creencia objetivo $bel(x_t)$. Bajo la suposición de que las partículas $\mathcal{X}_{t-1}$ son distribuidas de acuerdo a $bel(x_{t-1})$ la distribución g corresponde al resultado de la distribución

$$p(x_t \mid u_t, x_{t-1}) bel(x_{t-1}) \tag{3.7}$$

Una vez vista la importancia del muestreo y cómo influyen los pesos o ponderaciones, ayudará a seguir con la derivación. Se pueden imaginar las partículas como muestras de secuencias de estados

$$x_{0:t}^{[m]} = x_0^{[m]}, x_1^{[m]}, \ldots, x_t^{[m]} \tag{3.8}$$

El algoritmo únicamente cambia agregando a la partícula $x_t^{[m]}$ la secuencia de muestras de estados que fueron generadas, $x_{0:t-1}^{[m]}$. Este "nuevo" filtro calcula la posterior sobre todas las secuencias de estado:

$$bel(x_{0:t}) = p(x_{0:t} \mid u_{1:t}, z_{1:t}) \tag{3.9}$$

en lugar de la creencia $bel(x_t) = p(x_t \mid u_{1:t}, z_{1:t})$. Ciertamente, el espacio de todas las secuencias de estados es enorme y abarcarlo con partículas no es una buena idea. Sin embargo esta definición es buena para la derivación del algoritmo.

La distribución posterior $bel(x_{0:t})$ es obtenida analógicamente de la derivación del $bel(x_t)$ que se derivó en el filtro de Bayes (algoritmo 1).

Utilizando el teorema de Bayes:

$$p(x_{0:t} \mid z_{1:t}, u_{1:t}) = \eta p(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t}) p(x_{0:t} \mid z_{1:t-1}, u_{1:t})$$

Supuestos de Markov y estado completo:

$$\begin{aligned} &= \eta p(z_t \mid x_t) p(x_{0:t} \mid z_{1:t-1}, u_{1:t}) \\ &= \eta p(z_t \mid x_t) p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t}) \\ &= \eta p(z_t \mid x_t) p(x_t \mid x_{t-1}, u_t) p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1}) \end{aligned} \tag{3.10}$$

Asumiendo que las partículas iniciales son obtenidas por el muestreo $p(x_0)$. También se asume que el conjunto de partículas del tiempo $t-1$ es distribuido de acuerdo a $bel(x_{0:t-1})$. Para la m-ésima partícula $x_{0:t-1}^{[m]}$ en este conjunto, la muestra $x_t^{[m]}$ generada en el paso 4 del algoritmo es generado por la distribución propuesta:

$$p(x_t \mid x_{t-1}, u_t) bel(x_{0:t-1}) = p(x_t \mid x_{t-1}, u_t) p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1}) \tag{3.11}$$

recordando el valor del peso como:

$$\begin{aligned} w_t^{[m]} &= \frac{\text{distribución objetivo}}{\text{distribución propuesta}} \\ &= \frac{\eta p(z_t \mid x_t) p(x_t \mid x_{t-1}, u_t) p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1})}{p(x_t \mid x_{t-1}, u_t) p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1})} \\ &= \eta p(z_t \mid x_t) \end{aligned} \tag{3.12}$$

La constante η no juega un papel importante en esta ecuación, debido a que en el remuestreo se asignan partículas con probabilidad *proporcional* a la importancia de los pesos. Por el remuestreo de partículas con probabilidad proporcional a $w_t^{[m]}$, las partículas resultantes en efecto se distribuyen de acuerdo con el producto de la propuesta y los pesos importantes $w_t^{[m]}$:

$$\eta w_t^{[m]} p(x_t \mid x_{t-1}, u_t) p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1}) = bel(x_{0:t}) \tag{3.13}$$

Esta derivación es únicamente correcta para $M \to \infty$.

3.2. Localización Monte Carlo

En esta parte se describirá el algoritmo de localización más popular en la robótica, el cual representa la creencia $bel(x_t)$ a través de partículas. Este algoritmo es llamado *Localización Monte Carlo* o *MCL*. De igual forma se basa en la idea de todos los filtros, siendo recursivo y dividiéndose en dos partes, aproximación y corrección.
Esta localización se basa en el filtro de partículas. Se trata de un algoritmo muy fácil de implementar, además de ser robusto y resolver infinidad de problemas de localización. Esto debido a que se basa en un filtro no paramétrico, y por lo tanto no es necesario que el modelo de movimiento cumpla ciertas restricciones y además puede representar creencias no unimodales.

3.2.1. Algoritmo MCL

El algoritmo de localización Monte Carlo es muy similar al algoritmo de filtro de partículas. Únicamente cambia en que recibe como entrada un mapa m y sustituyendo apropiadamente modelos de movimiento y percepción. En este algoritmo se representa la creencia $bel(x_t)$ por un conjunto de M partículas $\mathcal{X}_t = \{x_t^{[1]}, x_t^2, \ldots, x_t^M\}$. Recibe como entradas el conjunto de las partículas en el tiempo $t-1$, el control y la percepción del robot en el tiempo t y un mapa m. La salida del algoritmo es el conjunto de partículas en el tiempo t. El algoritmo MCL se presenta a continuación:

Algoritmo 10 MCL($\mathcal{X}_{t-1}, u_t, z_t, m$)

1: $\overline{\mathcal{X}}_t = \mathcal{X} = 0$
2: **for** $m = 1$ **to** M **do**
3: $\quad x_t^{[m]} =$ **sample_motion_model**$(u_t, x_{t-1}^{[m]})$
4: $\quad w_t^{[m]} =$ **measurement_model**$(z_t, x_t^{[m]}, m)$
5: $\quad \overline{\mathcal{X}}_t = \overline{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6: **end for**
7: **for** $m = 1$ **to** M **do**
8: $\quad$ tomar i con probabilidad $\propto w_t^{[i]}$
9: $\quad$ agregar $x_t^{[i]}$ to $\mathcal{X}_t$
10: **end for**
11: **return** $\mathcal{X}_t$

La derivación del algoritmo es idéntica a la derivación del filtro de partículas. Lo que hay que tener en cuenta es que MCL utiliza el algoritmo de muestreo, para representar las partículas. Y se asignan pesos a las mismas mediante el modelo de percepción de extracción de características.

Capítulo 4

Algoritmo de localización para el sistema multiagente PokTaPok

Es muy interesante e importante resolver la localización de agentes en ambientes con ruido porque sin una buena localización los agentes no pueden desarrollar adecuadamente sus tareas. Para el sistema multiagente PokTaPok la autolocalización de los agentes dentro del campo de juego significa conocer el vector pose $(\ x \quad y \quad \theta \ )^T$ en el tiempo t. Recordando que el *soccerserver* agrega ruido en todo momento a las acciones y a las percepciones de los agentes, por lo tanto para comenzar a resolver el problema de la localización del sistema PokTaPok, primero se analizarán los modelos de movimiento y visión que el *soccerserver* utiliza y cómo es que agrega ruido a estos modelos. Se plantea y se explica el algoritmo desarrollado para resolver el problema de localización del sistema multiagente PokTaPok.

4.1. Modelo de movimiento del soccerserver

El movimiento de los objetos dentro del servidor se establece mediante tres comandos, el comando *dash*, *turn* y *kick*. El comando *dash* se utiliza para desplazar al agente hacia adelante o atrás. El comando *turn* sirve para que el agente gire y el comando *kick* es para que el agente pueda "patear" el balón y desplazarlo hacia determinada dirección. A continuación, en las secciones 4.1.1 y 4.1.2 se describen a detalle los modelos de los comandos *dash* y *turn* que son los comandos de interés para entender como se desplazan los agentes.

4.1.1. Modelo del comando dash

El comando *dash* proporciona un impulso o aceleración al agente, este comando recibe como parámetro un valor entre $[minpower, maxpower]$ que representa el "poder" del impulso con valores de -100 para *minpower* y 100

para *maxpower*. El modelo para el desplazamiento de un objeto es muy simple, únicamente se agrega un vector de movimiento a la posición del objeto. La velocidad se incrementa con el comando *dash* para el caso de los agentes y con el comando *kick* para el balón y a través del tiempo la velocidad sufre un decaimiento.

En cada ciclo el movimiento de los objetos móviles (agentes, balón) es calculado de acuerdo con las siguientes fórmulas:

$$(v_x^0, v_y^0) = (0,0) : \text{velocidad inicial} \tag{4.1}$$

$$(a_x^0, a_y^0) = (0,0) : \text{aceleración inicial} \tag{4.2}$$

$$(u_x^{t+1}, u_x^{t+1}) = (v_x^{t+1}, v_x^{t+1}) + (a_x^t, a_x^t) + (\tilde{r}_1, \tilde{r}_2) : \text{velocidad} \tag{4.3}$$

$$(p_x^{t+1}, p_y^{t+1}) = (p_x^t, p_y^t) + (u_x^{t+1}, u_y^{t+1}) : \text{movimiento} \tag{4.4}$$

$$(v_x^{t+1}, v_y^{t+1}) = \mathbf{Decay} \times (u_x^{t+1}, u_y^{t+1}) : \text{decaimiento} \tag{4.5}$$

donde (p_x^t, p_y^t), (v_x^t, v_y^t) y (a_x^t, a_y^t) denotan la posición, la velocidad y la aceleración del objeto en el ciclo t, respectivamente. El vector $(\tilde{r}_1, \tilde{r}_2)$ representa el ruido que agrega el servidor al movimiento del objeto, donde los valores $\tilde{r}_i$ son tomados de una distribución uniforme sobre el rango $[-r_{max}, r_{max}]$ y el valor de r_{max} depende de la velocidad del objeto como se muestra en la ecuación 4.6. El parámetro **Decay** representa la tasa de decaimiento de velocidad del objeto, para los agentes toma el valor de **player_decay**= 0.4 y para el balón **ball_decay**= 0.94, estos valores de los parámetros son fijos dentro del servidor.

$$r_{max} = \mathbf{Rand} \cdot ||(v_x^t, v_y^t) + (a_x^t, a_y^t)|| \tag{4.6}$$

En la ecuación 4.6 el parámetro **Rand** es un valor fijo el cual toma el valor de **player_rand**=0.1 o **ball_rand**= 0.05, para el jugador y el balón respectivamente [3].

Con el conocimiento de la ecuación 4.6, se puede concluir que al desplazar el agente con el comando *dash*, se crea un área rectangular y perpendicular a los ejes, de posibles posiciones del agente, debido al ruido uniforme que se agrega al vector de movimiento. Esto es: Sea $P_0 = (x_0, y_0)$ la posición en un plano cartesiano. Si se suma a la posición P_0 un vector de movimiento con ruido aleatorio uniforme agregado a la coordenada x y y, se genera una región rectangular y perpendicular a los ejes de posibles posiciones, como se muestra en la figura 4.1.

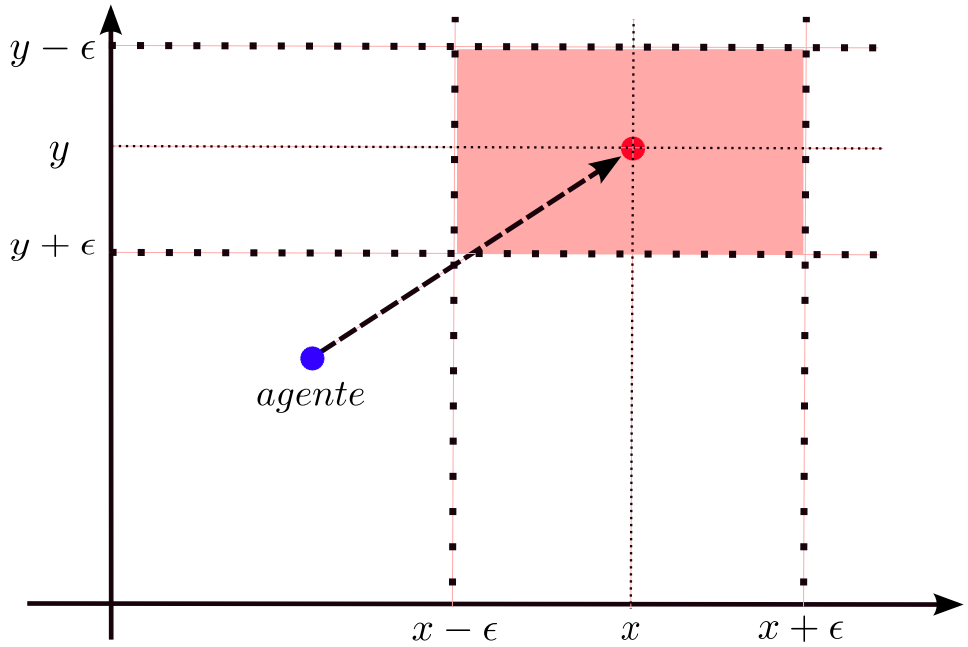


Figura 4.1: Área de posibles posiciones del agente después de un desplazamiento.

Algoritmo de aceleración

El modelo de movimiento que simula el servidor tiene un algoritmo para crear un vector aceleración mediante el parámetro *power* que recibe el comando *dash*. El algoritmo 11 muestra como se crea el vector de aceleración.

Algoritmo 11 Dash_Model_Soccer_Server_2D(power)

```
1: power = NormalizarDashPower(power,maxpower,minpower)
2: back_dash = power < 0.0
3: if back_dash == true then
4:    power_need = power* − 2.0
5: else
6:    power_need = power
7: end if
8: power_need = min(power_need,stamina+extraStamina)
9: if back_dash == true then
10:   power = power_need/2.0
11: else
12:   power = power_need
13: end if
14: efective_dash_power = effort*power*dashPowerRate
15: acel = fromPolar( efective_dash_power , body_angle )
16: return acel
```

Para explicar el algoritmo 11 es necesario saber que el servidor considera un modelo de *stamina*, el cual representa la "energía" que tiene el agente. La energía disminuye dependiendo de la cantidad de poder que utilice el agente para desplazarse. Para mayor referencia acerca del modelo de *stamina* que utiliza el servidor, consultar [3] sección 4.5.2.

En la línea 1 del algoritmo 11 se verifica que *power* se encuentre dentro de su dominio [*minpower* , *maxpower*]. En la línea 2 se utiliza una variable *booleana* para saber si el desplazamiento del agente es hacia atrás o hacia delante. En caso de ser hacia atrás, *power* se multiplica por -2 indicando que se necesita el doble de *stamina* para un desplazamiento en esa dirección. En la línea 8 la variable *power_need* es la cantidad mínima entre *power_need* y la *stamina*, que representa la cantidad de poder que se puede utilizar en el desplazamiento dependiendo de la *stamina* que se tenga. En la línea 9, si el desplazamiento es hacia atrás, el poder se divide entre dos para volver al poder inicial que se requería en el desplazamiento. En la línea 14 se calcula la distancia o magnitud del desplazamiento, para que en la línea 15 se convierta el vector de coordenadas polares, compuesto por la magnitud de desplazamiento y dirección del cuerpo del agente, a coordenadas cartesianas y se tenga el vector aceleración en la variable *acel*, resultado del algoritmo 11.

4.1.2. Modelo del comando turn

El comando *turn* recibe como parámetro la cantidad de grados que se desea girar el agente. El dominio del parámetro se encuentra entre los valores del servidor *minmoment* y *maxmoment*, que se muestran en la tabla 4.1.

El servidor agrega ruido al comando *turn* como se muestra en la expresión 4.7 y además delimita la cantidad de grados que el agente puede girar dependiendo de su velocidad.

$$ang = \frac{(1.0 + \tilde{r}) \cdot ang'}{1.0 + \text{inertia} \cdot \text{player\_speed}} \tag{4.7}$$

En la expresión anterior (4.7), $\tilde{r}$ es un número aleatorio tomado de una distribución uniforme sobre el intervalo [*-player_rand* , *player_rand*], *ang* es el valor de los grados con ruido, ang' es el valor del parámetro del comando **turn**, *inertia* es un parámetro del servidor el cual denota la inercia del agente y *player_speed* es la velocidad que lleva el agente. En la figura 4.2 se expresa cómo después de un giro del agente existe un intervalo de posibles direcciones que pudo haber obtenido el agente. Los valores de los parámetros mencionados se encuentran en la tabla 4.1.

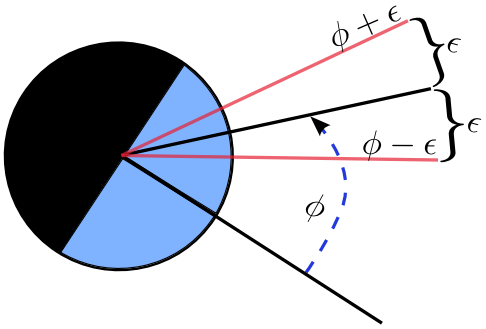


Figura 4.2: Posibles ángulos después de un comando turn.

Un detalle a considerar es que la máxima velocidad de un agente es *player_speed_max*= 1.05 y utilizando la ecuación 4.7, el giro máximo de un agente a su máxima velocidad es de ±28.8 grados, aún sin considerar el ruido. Sin embargo, una de las características del modelo de movimiento del servidor para los agentes, es que un agente no puede realizar un comando **dash** y **turn** en el mismo ciclo. Por ello cuando un agente realiza un giro (**turn**), la velocidad máxima a la cual puede encontrarse es de 0.42, por la ecuación de decaimiento de la velocidad ($player\_speed\_max \cdot player\_decay = 1.05 \cdot 0.4 = 0.42$). Teniendo en cuenta esto, la efectividad de un giro del agente es de ±58.06 grados, sin considerar el ruido.

Parámetro	Valor
minmoment	-180
maxmoment	180
player_rand	0.1
inertia	5.0
player_speed_max	1.05

Tabla 4.1: Parámetros del servidor.

4.2. Modelo de visión del soccerserver

Las percepciones del agente que simula el servidor son a través de tres mensajes que el servidor envía al agente a través del mensaje *aural*, *body* y *see*. El mensaje que se utiliza para la localización es el mensaje *see*. Este mensaje representa un sensor visual para el agente. Este mensaje *see* proporciona información al agente para conocer distancias, ángulos y características de los objetos en el campo.

El servidor modela el sensor visual mediante un área llamada *cono de visión*. Este cono representa el área de visibilidad que tiene el agente, en pocas palabras,

el agente recibe información únicamente de los objetos que se encuentren dentro del cono. Los objetos que el agente puede percibir son: balón, agentes, líneas y banderas.

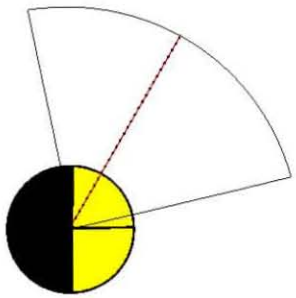


Figura 4.3: Representación de un agente 2D y su cono de visión.

El área visible de un agente depende de dos parámetros, el parámetro *sense_step* que determina el paso del tiempo entre cada recepción de mensajes *see*, este parámetro tiene un valor por *default* de 150 mili-segundos y el parámetro *visible_angle* que representa los grados de apertura del cono de visión que son 90 grados por *default*.

Cuando el objeto visible es un agente, la información que se incluye en el mensaje *see* tiene cierta probabilidad de ser agregada al mensaje. Esta probabilidad depende de la distancia a la que se encuentre el agente que es visible. Un ejemplo para entender cómo y con qué probabilidad el servidor proporciona la información sobre los agentes visibles es el siguiente: Sea *dist* la distancia a la que se encuentra un agente dentro del cono de visión, entonces:

- Si $dist \leq unum\_far\_length$, el número identificador del agente y nombre del equipo se incluyen en el mensaje *see* (son visibles).
- Si $unum\_far\_length < dist < unum\_too\_far\_length$, entonces el nombre del equipo es visible, pero la probabilidad de ver el número identificador decrece linealmente de 1 a 0 cuando *dist* aumenta.
- Si $dist \geq unum\_too\_far\_length$, entonces el número identificador no es visible.
- Si $dist \leq team\_far\_length$, entonces le nombre del equipo es visible.
- Si $team\_far\_length < dist < team\_too\_far\_length$, entonces la probabilidad de que el nombre del equipo sea visible decrece linealmente de 1 a 0 cuando *dist* aumenta.
- Si $dist \geq team\_too\_far\_length$, el nombre del equipo no es visible.

Los parámetros *unum_far_length*, *unum_too_far_length*, *team_far_length* y *team_too_far_length* están definidos dentro del servidor con valores de 20, 40, 40 y 60 respectivamente [3].

Otra característica muy importante del modelo de percepción, es la manera en la que el servidor agrega ruido y provoca incertidumbre a la información que proporciona sobre la distancia y ángulo hacia los objetos visibles. La distancia hacia un objeto es modificada para agregarle ruido como se muestra en 4.8.

$$d' = Quantize(exp(Quantize(ln(d), StepValue)), 0.1) \tag{4.8}$$

En la ecuación 4.8, d es la distancia que existe entre el agente y el objeto, d' es la distancia con ruido, *StepValue* es un parámetro del servidor, el cual toma el valor de *quantize_step*= 0.1 cuando el objeto es un agente o el balón y un valor de *quantize_step_l*= 0.01 para los objetos *referencia*. La distancia d' es la distancia que se incluye en el mensaje *see*, d la conoce únicamente el servidor. Además en 4.8 la función *Quantize* se define como se muestra en 4.9.

$$Quantize(A, B) = rint(A/B) \cdot B \tag{4.9}$$

Donde la función *rint* es una función que redondea al valor entero más cercano.

Un ejemplo de cómo el servidor genera ruido en la distancia hacia un objeto referencia con la ecuación 4.8 es: suponer que una distancia entre un agente y un objeto referencia es de 99, el servidor utiliza la ecuación 4.8 como se muestra en 4.10

$$\begin{aligned} d' &= Quantize\Big(exp\Big(Quantize\Big(ln(99), .01\Big)\Big), .1\Big) \\ d' &= rint\Big(exp\Big(rint\Big(ln(99)/.01\Big) * .01\Big)/.1\Big) * .1 \\ d' &= 99.5 \end{aligned} \tag{4.10}$$

La distancia que el servidor enviaría a través del mensaje *see* al agente sería de 99.5 como se muestra al final de la ecuación 4.10. Con este tipo de error se puede obtener el intervalo donde se encuentra la distancia "correcta" al objeto referencia "visto" por el agente. Dada la distancia d' se puede determinar el valor mínimo y máximo entre los cuales se encuentra la distancia d (distancia correcta al objeto referencia). En la ecuación 4.9 el valor B siempre será constante, entonces se

asume que $Quantize(A, B) = d'$ y despejando de la ecuación 4.9 se puede calcular los valores mínimo y máximo del argumento d, como se muestran en 4.11 y 4.12.

$$d_{min} = \left(rint\left(\frac{d'}{B}\right) - 0.5 \right) * B \tag{4.11}$$

$$d_{max} = \left(rint\left(\frac{d'}{B}\right) + 0.5 \right) * B \tag{4.12}$$

Si se utilizan 4.11 y 4.12, junto con el ejemplo anterior donde el servidor envía a través del mensaje *see* el valor de 99.5 como la distancia d', se puede obtener el intervalo donde se encuentra la distancia d como se muestra a continuación en las ecuaciones 4.13 y 4.14.

$$d_{min} = exp\left(\left(rint\Big(ln\big(((rint(99.5/.1) - .5) * .1) \big)/.01 \Big) - .5 \right) * 0.1 \right) \tag{4.13}$$

$$d_{max} = exp\left(\left(rint\Big(ln\big(((rint(99.5/.1) + .5) * .1) \big)/.01 \Big) + .5 \right) * 0.1 \right) \tag{4.14}$$

$$d_{min} = 98.98813555 \tag{4.15}$$

$$d_{max} = 99.98298285 \tag{4.16}$$

El intervalo donde se encuentra la distancia "real" a un objeto referencia, cuando el mensaje *see* tiene una distancia de 99.5 es $[98.98813555, 99.98298285]$, como se muestra en 4.15 y 4.16.

La dirección a la que se encuentra un objeto con respecto al agente es modificada por el servidor como se muestra en la ecuación 4.17.

$$g' = Quantize(g, 1.0) \tag{4.17}$$

La ecuación 4.17 es la manera en la que el servidor agrega ruido a la percepción de la dirección donde se encuentra un objeto. La variable g representa los grados reales a los que se encuentra el objeto percibido por el agente (valor que conoce el servidor) y la variable g' son los grados con ruido (valor que llega en el mensaje *see*). Con un proceso similar, al que se hizo en la ecuación 4.8, sobre la ecuación 4.17, también se puede conocer el intervalo de los ángulos entre el que se encuentra el objeto percibido, como se muestra en las ecuaciones 4.18 y 4.19.

$$g_{min} = \Big(rint\big(g'/1.0\big) - .5 \Big) * 1.0 \tag{4.18}$$

$$g_{max} = \Big(rint\big(g'/1.0\big) + .5 \Big) * 1.0 \tag{4.19}$$

Debido a que el mensaje *see* incluye la dirección hacia los objetos redondeando al entero más próximo, el intervalo entre el que se encuentra la dirección "real" al objeto se muestra en 4.20.

$$[(g' - 1), (g' + 1)] \tag{4.20}$$

El tipo de ruido en la distancia y la dirección hacia los objetos percibidos permite crear un área de un segmento de cono de posibles posiciones donde se encuentra el agente, donde el intervalo de los grados es la apertura del cono y el intervalo de la distancia representa la distancia del vértice al primer corte del cono y la distancia del vértice hacia el segundo corte del cono. Al área del segmento de cono de posibles posiciones del agente generada a causa de la percepción de la *i-ésima* bandera se le llamará Γ^i. Como se muestra en la figura 4.4.

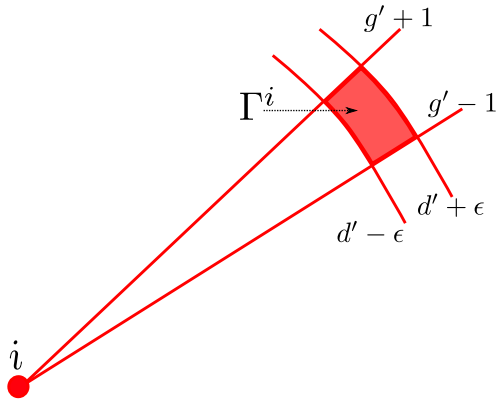


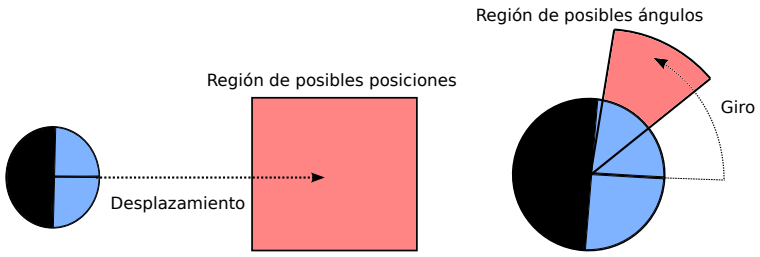
Figura 4.4: Área Γ^i. Posibles posiciones del agente que coinciden con el intervalo de error de la percepción de la bandera i

En resumen, en la sección 4.1 se describió los modelos que el servidor utiliza para agregar ruido y limitar los movimientos del agente, mientras que en la sección 4.2 se habla de cómo el servidor agrega ruido a las percepciones del agente. Considerando estas características del ruido que el servidor agrega a los modelos de percepción y movimiento de los agentes y con base a la teoría vista a lo largo de este trabajo, en la sección 4.3 se plantea un algoritmo para localizar a los agentes del sistema PokTaPok.

4.3. Planteamiento del algoritmo de localización PokTaPok

El ruido agregado en los modelos de movimiento y percepción del servidor es ruido uniforme, como ya se menciono anteriormente el valor "correcto" se encuentra dentro de un intervalo. El ruido uniforme asigna la misma probabilidad

a todos los posibles valores dentro de un intervalo. Los comandos de acción del agente (dash,turn) con ruido uniforme se pueden interpretar gráficamente como se muestra en la figura 4.5. La figura del lado izquierdo (figura 4.5a) muestra el ruido en el desplazamiento del comando *dash*, mientras que la figura del lado derecho (figura 4.5b) muestra el ruido en el giro del comando *turn*.



(a) Ruido de desplazamiento (b) Ruido de giro

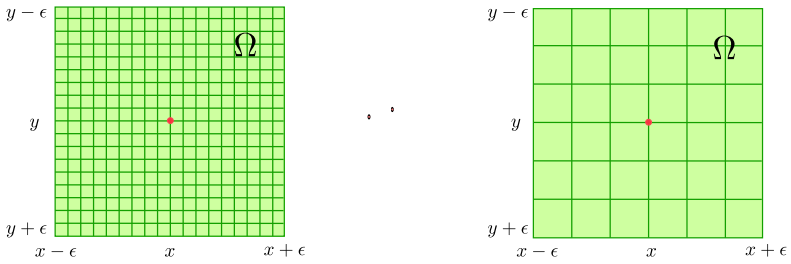
Figura 4.5: Las áreas rojas representas la región de posibles ubicaciones y la región de posible orientación del agente después de realizar un *dash* o *turn*.

Un ejemplo del ruido uniforme para el caso del modelo de percepción visual sería, si agente recibe el dato d como la distancia que hay hacia un objeto (dato erróneo por causa del ruido), se puede saber los valores entre los cuales se encuentra la distancia correcta, $[d - \varepsilon, d + \varepsilon]$.

El ruido uniforme también existe en la dirección percibida a la que se encuentran los objetos, proporcionada en grados. Si el servidor envía al agente el dato ϕ como la cantidad de grados a los que se encuentra algún objeto, el dato no es correcto pero se puede saber el intervalo donde está el valor correcto, $[\phi - \varepsilon, \phi + \varepsilon]$.

Una vez recordado el ruido en los movimientos y en la percepción del agente, se puede comenzar el planteamiento del algoritmo de localización para el sistema multiagente PokTaPok. De principio se interpretará la región de posibles posiciones que se genera después de un comando de movimiento mediante una región rectangular dentro del campo, esto es debido a que se tendrá un intervalo para la coordenada x y un intervalo para la coordenada y, por causa del ruido uniforme. El problema de tener regiones con distribución uniforme es que todos los puntos que se encuentran dentro de la región tienen la misma probabilidad de ser la posición "correcta" a donde se desplazó el agente y además en la región rectangular existen infinidad de puntos. Por esta razón se generará una "discretización" de la región. Una manera simple y común de discretizar es generando una cuadrícula o malla ("*grid*") dentro de la región, donde cada uno de los puntos o vértices de la malla se consideran como una posible ubicación. A esta región cuadriculada generada después de que el agente realiza una acción se le llamará región Ω.

Una característica importante de todo *grid* es la *granularidad*. Cuando tiene una granularidad pequeña la distancia entre las líneas que crean la malla es menor y una granularidad grande hace que la distancia entre las líneas que crean la malla sea mayor. Esta característica sirve para considerar una mayor o menor cantidad de posibles posiciones, respectivamente. Dentro del la región Ω cada vértice de la malla representa una posición, entonces entre menor sea la granularidad, más posiciones se están considerando. La desventaja de trabajar con granularidad pequeña es que se utilizan más datos y más memoria, lo que se traduce a una mayor complejidad computacional en el algoritmo.



(a) Región Ω con granularidad pequeña.

(b) Región Ω con granularidad grande.

Figura 4.6: Ejemplos de la región Ω con los tipos de granularidad.

Otro aspecto que se toma en cuenta para el planteamiento del algoritmo de localización PokTaPok es la región Γ que se crea por cada bandera percibida (figura 4.4). Con las regiones Γ que se generarán al percibir las banderas visibles, se obtendrá una nueva región dentro de la región Ω, la nueva región se llamará γ y será la intersección de todas las regiones Γ generadas. γ es la región factible de posiciones debido a que todas las percepciones hacia las banderas visibles coinciden en esa región. Un ejemplo gráfico se muestra en la figura 4.7.

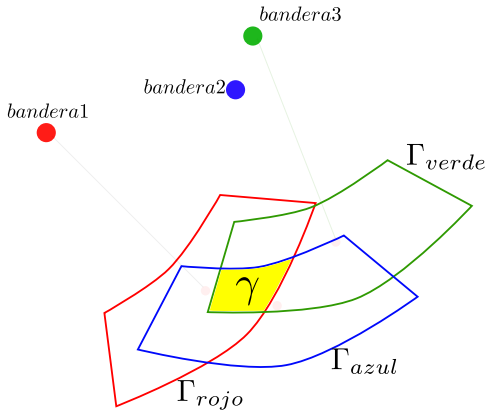


Figura 4.7: El área amarilla representa la región γ, que es la intersección de todas las regiones Γ generadas por la percepción de las banderas.

4.3.1. Descripción de las etapas en el algoritmo

Como se vio a lo largo de este trabajo, los algoritmos de localización se dividen en dos etapas "predicción" y "corrección". Para el caso del algoritmo de localización PokTaPok se tomarán tres etapas, una tercera para el calculo de la orientación del agente.

La etapa de "predicción" se lleva a cabo cuando el agente realiza un comando de acción y se crea una región Ω. Se debe recordar que la región Ω está definida como una región rectangular en forma de *grid*, donde cada vértice de la malla representa una posición en el campo. Estas posiciones dentro de este trabajo serán llamadas *partículas*.

La etapa de "corrección" se lleva a cabo en la percepción del agente, recibiendo la información de la distancia y ángulo hacia las banderas percibidas. Con la información de n banderas percibidas se generan n regiones Γ y se desea encontrar la región factible de dónde se encuentra el agente. Esta región será la intersección de todas las regiones Γ la cual se llamará región γ. Como se vio anteriormente en la figura 4.7.

El cálculo de la región γ en el algoritmo de localización PokTaPok se realiza de manera inteligente, evitando el cálculo explícito de la intersección de las regiones Γ.

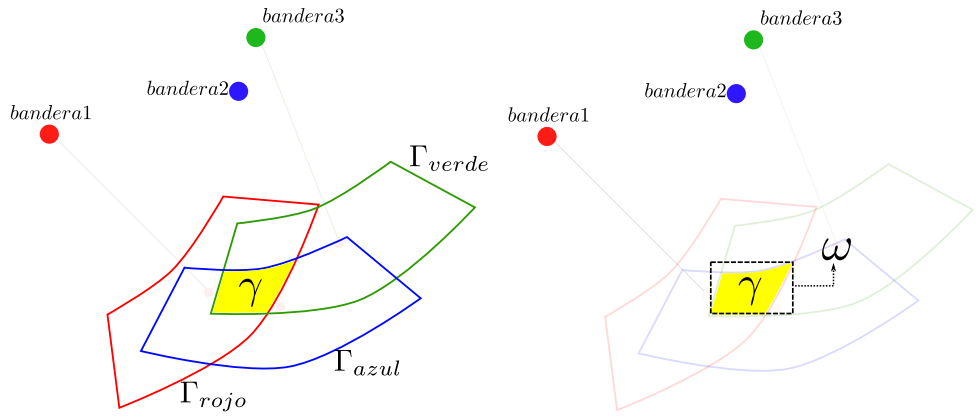
El cálculo de la intersección de polígonos simples tiene complejidad lineal en tiempo y espacio. Sin embargo las regiones Γ no son polígonos simples y para realizar el cálculo de su intersección, es necesario hacer uso de la teoría de las sucesiones de *Davenport-Schinzel* [21].

En el algoritmo PokTaPok simplemente se hace un recorrido sobre cada partícula dentro de la región Ω (cada vértice de la malla) y se pondera una partícula con un $+1$ cada vez que pertenece a una región Γ, es decir, cada vez que una partícula se encuentra dentro del intervalo de error de percepción de una bandera se pondera con una unidad, este tipo de ponderación es para indicar que esa partícula pertenece a una región factible. Entre mayor sea la ponderación de una partícula, la factibilidad de esa partícula es mejor.

Un ejemplo es si la percepción de una bandera en el tiempo t arrojó como distancia del agente hacia a una bandera la cantidad d, se sabe que la distancia real hacia la bandera está entre $[d - \varepsilon, d + \varepsilon]$, por lo tanto todas las partículas que se encuentren a una distancia dentro del intervalo son ponderadas y así para cada una de las banderas percibidas cada vez que una partícula sea factible, debido a que se encuentra dentro del intervalo de error de la percepción se

ponderará con una unidad. Al finalizar el recorrido, se sabe que las partículas con la máxima ponderación representan la intersección de todas las regiones Γ, debido a que coincidieron con el intervalo de error de percepción de todas las banderas, a la intersección de todas las regiones Γ se le llamará γ.

Es importante considerar que la región γ que se crea con el razonamiento anterior no es una región rectangular, pero para el algoritmo PokTaPok se toma en cuenta una región rectangular mínima que contiene todas las partículas con la máxima ponderación. Esta región rectangular final donde se encuentran todas las partículas con la máxima ponderación se llamará ω. Véase la figura 4.8.



(a) Intersección de áreas. (b) Acotación de área final.

Figura 4.8: Etapa de corrección.

La región γ (región rectangular mínima que encierra todas las partículas con la máxima ponderación) se define de la siguiente manera:

Sean n partículas con la máxima ponderación, se llamará A_k al punto de la $k-ésima$ partícula, con coordenadas (x_i, y_j), donde $1 \leq i, j, k \leq n$ y sean $P_1(x_a, y_a)$, $P_2(x_b, y_a)$, $P_3(x_a, y_b)$ y $P_4(x_b, y_b)$ las coordenadas de las cuatro esquinas de la región rectangular ω, entonces:

$$x_a \leq x_1, x_2, x_3, \ldots, x_n \leq x_b$$

y

$$y_a \leq y_1, y_2, y_3, \ldots, y_n \leq y_b$$

por lo tanto cualquier combinación de (x_i, y_j) se encuentra dentro de la región ω.

No se debe de dejar de lado, que el resultado sigue siendo una región en la cual existen infinidad de puntos, entonces se considera el centro de la región

resultante como la posición final única del agente.

En la tercera y última etapa se calcula la orientación del agente utilizando la percepción de todas las banderas vistas por el agente. Este proceso se explica a detalle en la sección 4.3.2 en la etapa de orientación.

4.3.2. Pseudo código de las etapas en el algoritmo

El algoritmo de localización que se implementó para el sistema multiagente PokTaPok consta de tres etapas: predicción, corrección y orientación, como se menciono anteriormente. El algoritmo de la etapa de predicción recibe como entradas la acción realizada por el robot en el tiempo t (u_t). La entrada u_t es un objeto que indica si se realizó el comando *dash* o *turn* y con que valor de parámetro de entrada. También recibe un arreglo llamado *"Partículas"* del tiempo $t-1$. El arreglo *"Partículas"* contiene cuatro elementos tipo *"partícula"*. Los elementos tipo *"partícula"* contienen una tupla de tres elementos $(\ x \quad y \quad \theta\ )^T$ que representa una pose del agente dentro del campo. Se utilizan cuatro elementos *"partícula"* para representar las cuatro esquinas de la región rectangular Ω que se crea en la etapa de predicción (figura 4.9).

Existe un caso particular para la pose del agente, donde $t = 0$ (tiempo inicial). En este caso los cuatro elementos *"partícula"* son la misma pose y por lo tanto la región rectangular es solo un punto. El algoritmo de la etapa de predicción da como resultado el arreglo *"Partículas"* del tiempo t, que contiene las coordenadas de las cuatro esquinas de la región rectangular Ω.

Figura 4.9: Representación de la región Ω con los elementos del arreglo *Partículas*. Los superíndices representan la posición dentro del arreglo *Partículas* y los subíndices el tiempo, en este caso tiempo $t-1$.

Etapa de predicción

Algoritmo 12 Localization_PokTaPok_Prediction $(Partículas_{t-1}, u_t)$

1: **for** $i = 0$ **to** 3 **do**
2: $mov =$ **modelo_movimiento**$(Partículas^i_{t-1},\ u_t\ )$
3: $r_{max} =$ player_rand$* \mid\mid mov \mid\mid$
4: $r1 = -r_{max} \qquad r2 = r_{max}$
5: $mov\_x\_min = mov.x + r1 \qquad mov\_y\_min = mov.y + r1$
6: $mov\_x\_max = mov.x + r2 \qquad mov\_y\_max = mov.y + r2$
7: **switch** (i)
8: **case** 0**:**
9: $Partículas^i_{t-1}.x = Partículas^i_{t-1}.x + mov\_x\_min$
10: $Partículas^i_{t-1}.y = Partículas^i_{t-1}.y + mov\_y\_min$
11: **case** 1**:**
12: $Partículas^i_{t-1}.x = Partículas^i_{t-1}.x + mov\_x\_max$
13: $Partículas^i_{t-1}.y = Partículas^i_{t-1}.y + mov\_y\_min$
14: **case** 2**:**
15: $Partículas^i_{t-1}.x = Partículas^i_{t-1}.x + mov\_x\_min$
16: $Partículas^i_{t-1}.y = Partículas^i_{t-1}.y + mov\_y\_max$
17: **case** 3**:**
18: $Partículas^i_{t-1}.x = Partículas^i_{t-1}.x + mov\_x\_max$
19: $Partículas^i_{t-1}.y = Partículas^i_{t-1}.y + mov\_y\_max$
20: **end switch**
21: **end for**
22: **return** $Partículas_{t-1}$

El algoritmo de la etapa de predicción cuenta con una estructura de repetición que va de 0 a 3 para realizar el proceso a cada elemento del vector *Partículas* que representa las cuatro esquinas de la región Ω, como se ha mencionado anteriormente.

En la línea 2 se calcula un vector de movimiento con la función **modelo_ movimiento** que se explica más adelante en el algoritmo 13. En la línea 3 se calcula el parámetro de error máximo que está en función de la magnitud del movimiento que realiza el agente. En la línea 4 se calculan el valor negativo y positivo del error. En la línea 5 se calcula la coordenada mínima posible, mientras que en la línea 6 se calcula la coordenada máxima posible. A partir de la línea 7 hasta la 20, se asignan las coordenadas correspondientes a cada elemento del arreglo *Partículas*, es decir, al elemento $Partículas^0$ le corresponde la esquina superior izquierda de la nueva región Ω creada, al elemento $Partículas^1$ la esquina derecha superior de la región Ω y así para los 4 elementos, como se mostró en la figura 4.9.

Al final el algoritmo regresa el arreglo *Partículas* modificado en el tiempo $t-1$, el cual contiene la región rectangular creada después de que el agente realiza un comando de acción.

En la figura 4.10 se ejemplifica el proceso que se realiza en la etapa de predicción, aplicando el comando de acción a cada esquina de la región rectangular. Este proceso genera cuatro subregiones rectangulares, una por cada esquina, o bien, por cada elemento del arreglo $Partículas_{t-1}$ se crea una subregión rectangular. Una vez obtenidas las cuatro subregiones, se toman los extremos, es decir, los valores máximos y mínimos para x y y, creando la región rectangular final.

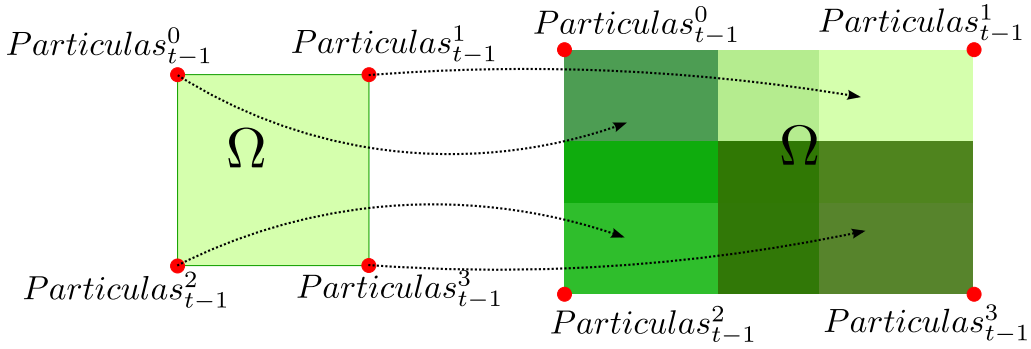


Figura 4.10: Región Ω en el tiempo $t-1$ y la región Ω modificada después de un desplazamiento en el tiempo $t-1$, generada al aplicar el algoritmo de predicción.

En el algoritmo 13 que se muestra a continuación, se describe la función *modelo_movimiento*, utilizada en la etapa de predicción. La función *modelo_movimiento* recibe el i-ésimo elemento del arreglo *Partícula* en el tiempo t junto con el comando de acción u_t que realiza el agente, dando como resultado un vector movimiento.

Algoritmo 13 modelo_movimiento ($Partículas^{i}_{t-1}$, u_t)

1: **if** $u_t.dash == 0.0$ **then**
2: **if** $u_t.turn == 0.0$ **then**
3: **if** $body\_dis\_vel == 0.0$ **then**
4: $mov.x = mov.y = 0.0$
5: **else**
6: $mov = \mathbf{fromPolar}(body\_dis\_vel, \mathbf{rad}(Partículas^{i}_{t-1}.\theta + body\_dir\_vel))$
7: **end if**
8: **else**
9: **if** body_dir_vel $== 0.0$ **then**
10: $mov.x = mov.y = 0.0$
11: $Partículas^{i}_{t-1}.\theta = \mathbf{Pr}(\frac{(1.0-\tilde{r})\cdot ang'}{1.0}, \frac{(1.0+\tilde{r})\cdot ang'}{1.0})$
12: **else**
13: $mov = \mathbf{fromPolar}(body\_dis\_vel, \mathbf{rad}(Partículas^{i}_{t-1}.\theta + body\_dir\_vel))$
14: $Partículas^{i}_{t-1}.\theta = \mathbf{Pr}(\frac{(1.0-\tilde{r})\cdot ang'}{1.0+\text{inertia}\cdot\text{player\_speed}}, \frac{(1.0+\tilde{r})\cdot ang'}{1.0+\text{inertia}\cdot\text{player\_speed}})$
15: **end if**
16: **end if**
17: **else**
18: $mov = \mathbf{fromPolar}(body\_dis\_vel, \mathbf{rad}(Partículas^{i}_{t-1}.\theta + body\_dir\_vel))$
19: acel = Dash_Model_Soccer_Server_2D(power)
20: $mov.x = acel.x + mov.x \qquad mov.y = acel.y + mov.y$
21: **end if**
22: **return** mov,$Partículas_{t-1}$

El vector de movimiento que da como resultado el algoritmo 13, se calcula dependiendo del tipo de comando de acción que realiza el agente y la velocidad heredada del tiempo $t-1$.

Existen diversos casos que se consideran para calcular el vector de movimiento. Para el caso donde el agente no realice ningún comando de acción y la velocidad heredada del tiempo $t-1$ sea igual a cero, se genera un vector de movimiento nulo (línea 4). Si el agente tiene cierta velocidad heredada del tiempo $t-1$, se calcula un vector de movimiento considerando la velocidad, la orientación del agente y la dirección de la velocidad (línea 6) y utilizando la función *fromPolar*, la cual convierte una coordenada polar a una coordenada cartesiana, como se indica en la ecuación 4.133 de la página 112.

Para los dos casos posibles donde el agente puede realizar el comando *turn* se calcula la posible orientación del agente (líneas 11 y 14) utilizando el modelo del comando *turn* (ecuación 4.7) y obteniendo el promedio entre el ángulo mínimo y máximo posibles, este procedimiento se define en la ecuación 4.134 en la página 112.

Si el comando *turn* se realizo con cierta velocidad también se calcula el desplazamiento posible con base a la velocidad (línea 13). Los dos casos restantes para calcular el vector de movimiento son cuando el agente realiza un comando *dash* con velocidad o sin velocidad heredada del tiempo anterior. Ambos casos se simplifican y se reducen a las operaciones que se muestran en las líneas 18 (obtención del desplazamiento con base en la velocidad) y línea 19 (modelo de comando *dash*, algoritmo 11). Por último el algoritmo genera el vector de movimiento final, sumando el desplazamiento a causa de la velocidad (mov) y el desplazamiento a causa del comando *dash* ($acel$), como se muestra en la línea 20. El algoritmo regresa el vector de desplazamiento y el arreglo *Partículas*.

Etapa de corrección

La etapa de corrección del algoritmo de localización PokTaPok se lleva a cabo después de haber generado la región Ω, con las posibles posiciones del agente. Durante la etapa de corrección en la región Ω se descartar algunas posiciones mediante la percepción de las banderas que percibe el agente. El pseudo código de la etapa de corrección se muestra a continuación en el algoritmo 14.

El algoritmo de la etapa de corrección recibe como entradas el arreglo *Partículas* en el tiempo $t-1$ (representando la región Ω), el vector z_t que contiene las banderas percibidas por el agente en el tiempo t y una variable g que representa la granularidad, es decir, el espacio entre la malla de Ω.

Algoritmo 14 Localization_PokTaPok_Correction ($Partículas_{t-1}$,z_t,g)

1: $eX = \lceil (\frac{\text{Partículas}^1_{t-1}.x - \text{Partículas}^0_{t-1}.x}{g}) \rceil \quad eY = \lceil (\frac{\text{Partículas}^2_{t-1}.y - \text{Partículas}^0_{t-1}.y}{g}) \rceil$
2: $grid[2][eX * eY - 1]; \quad peso[eX * eY - 1]; \quad k = 0; \quad pesoMax = 0;$
3: $xmin = ymin = \infty; \quad xmax = ymax = -\infty;$
4: **for** $i = 0$ **to** $eY - 1$ **do**
5: **for** $j = 0$ **to** $eX - 1$ **do**
6: $grid[0][k] = \text{Partículas}^0_t.x + (g * j); grid[1][k] = \text{Partículas}^0_t.y + (g * i); peso[k] = 0.0; k++$
7: **end for**
8: **end for**
9: **for** $i = 0$ **to** $k - 1$ **do**
10: **for** $j = 1$ **to** $size(z_t)$ **do**
11: $peso[i] = peso[i]$+**modelo_percepcion**$(z^j_t, grid[0][i], grid[1][i], \text{Partículas}^0_t.\theta)$
12: **end for**
13: **if** $peso[i] > pesoMax$ **then**
14: $pesoMax = peso[i]$
15: **end if**
16: **end for**
17: **for** $i = 0$ **to** $k - 1$ **do**
18: **if** $peso[i] = pesoMax$ **then**
19: **if** $grid[0][i] < xmin$ **then**
20: $xmin = grid[0][i]$
21: **end if**
22: **if** $grid[1][i] < ymin$ **then**
23: $ymin = grid[1][i]$
24: **end if**
25: **if** $grid[0][i] > xmax$ **then**
26: $xmax = grid[0][i]$
27: **end if**
28: **if** $grid[1][i] > ymax$ **then**
29: $ymax = grid[1][i]$
30: **end if**
31: **end if**
32: **end for**
33: $Partículas^0_t.x = xmin; \quad Partículas^0_t.y = ymin$
34: $Partículas^1_t.x = xmax; \quad Partículas^1_t.y = ymin$
35: $Partículas^2_t.x = xmin; \quad Partículas^2_t.y = ymax$
36: $Partículas^3_t.x = xmax; \quad Partículas^3_t.y = ymax$
37: **return** $Partículas_t$

En la línea 1 del algoritmo 14 se divide el largo y el alto de Ω entre la granularidad, para obtener las variables eX y eY, respectivamente. Estas dos variables representan el número de partículas que caben horizontalmente y

verticalmente en Ω. En la línea 2 se definen variables y arreglos que se utilizan en el algoritmo, el arreglo bidimensional *grid* es de dos renglones y cantidad total de partículas dentro de Ω como el número de columnas, esto es para representar la coordenada (x, y) de cada partícula, los elementos $grid[0][i]$ representan la coordenada x y los elementos $grid[1][i]$ representan la coordenada y, el arreglo unidimensional *peso* tiene un tamaño igual a la cantidad total de partículas dentro de Ω, este arreglo sirve para guardar la ponderación de cada partícula, la variable *pesoMax* representará la ponderación máxima, por ello se inicializa en cero, la variable k almacena la cantidad total de partículas en Ω.

El algoritmo también contiene cuatro variables en la línea tres con valores adecuados para realizar búsquedas de máximos y mínimos en líneas posteriores del algoritmo. A partir de las línea 4 a la 8 hay dos estructuras de repetición anidadas que inicializan los arreglos *grid* y *peso*. En las líneas 9 a 16 se realiza una estructura de repetición que se itera para cada una de las partículas, la siguiente estructura de repetición de las líneas 10 a 12 se realiza por cada bandera percibida, es decir, el tamaño del vector z_t.

En el algoritmo el proceso de ponderación de cada partícula es con base en el modelo de percepción que está representado por la función **modelo_percepcion**, que se define más adelante en el algoritmo 15. Con la función *modelo_percepción* se obtendrá la ponderación para cada partícula, es decir, se estarían generando las regiones Γ de la *i-ésima* partícula a causa de la percepción de las banderas. Después de haber terminado la ponderación de la *i-ésima* partícula se verifica en la línea 13 si la ponderación correspondiente a esa partícula es el máximo, si es así se cambia el valor de la variable *pesoMax*. En la última estructura de repetición de la línea 17 a la 32 se verifican todas las partículas, si la *i-ésima* partícula tiene una ponderación igual al peso máximo. Es importante ver, que si se juntarán todas las partículas con el peso máximo se estaría generando la región γ, que es la intersección de todas las regiones Γ. Considerando las partículas con máxima ponderación se pregunta si la coordenada de la partícula con ponderación máxima es algún extremo de la nueva región Ω, es decir, si su coordenada es la mínima o máxima en x ó, mínima o máxima en y, estos valores se guardan en las variables $xmin$, $xmax$, $ymin$ y $ymax$ que representan las coordenadas de las cuatro esquinas de la nueva región rectangular Ω. En las líneas 33 a la 36 se asignan los nuevos valores al arreglo Partículas, al elemento $Partículas_t^0$ se le asigna la coordenada correspondiente a la esquina superior izquierda, a $Partículas_t^1$ la coordenada correspondiente a la esquina superior derecha, a $Partículas_t^2$ la esquina inferior izquierda y a $Partículas_t^3$ la esquina inferior derecha (véase la figura 4.10 en la página 62).

En el algoritmo 15, se describe la función *modelo_percepcion*, con la cual se pondera cada partícula. Se debe recordar que la ponderación será mayor si

la partícula se encuentra dentro de la mayor cantidad de regiones γ. Como se menciono anteriormente, cada región γ se crea por cada bandera en percibida.

Algoritmo 15 modelo_percepcion $(z_t^j, grid[0][i], grid[1][i], \text{Partículas}_t^0.\theta)$

1: $ponderacion = 0$
2: $disX = z_t^j.x - grid[0][i]$
3: $disY = z_t^j.y - grid[1][i]$
4: $vAprox = \sqrt{disX^2 + disY^2}$
5: $v\_min = (rint(\frac{z_t^j.dist}{0.1}) - 0.5) * 0.1$ ecuación 4.11
6: $v\_max = (rint(\frac{z_t^j.dist}{0.1}) + 0.5) * 0.1$ ecuación 4.12
7: **if** $vAprox >= v\_min$ **AND** $vAprox <= v\_max$ **then**
8: $ponderacion++$
9: **end if**
10: $gAprox =$ **atan2**$(disY, disX)$-**rad**$(Partículas_t^0.\theta)$
11: $g\_min = (rint(\frac{z_t^j.ang}{1.0}) - 0.5) * 1.0$ ecuación 4.18
12: $g\_max = (rint(\frac{z_t^j.ang}{1.0}) + 0.5) * 1.0$ ecuación 4.19
13: **if** $gAprox >= g\_min$ **AND** $gAprox <= g\_max$ **then**
14: $ponderacion++$
15: **end if**
16: **return** $ponderacion$

El algoritmo de la función *modelo_percepcion*, que se muestra en el algoritmo 15, recibe como entradas la *j-ésima* bandera percibida en el tiempo t, es decir z_t^j. También recibe la coordenada (x, y) de la *i-ésima* partícula $(grid[0][i], grid[1][i])$ y recibe la orientación del agente, $Partículas_t^0.\theta$. En la línea 1 se inicializa la variable ponderación en cero, donde se guardará la ponderación correspondiente a la *i-ésima* partícula. En la línea 2 se calcula la diferencia entre la coordenada x de la *i-ésima* partícula y la *j-ésima* bandera. En la línea 3 se calcula la diferencia entre la coordenada y de la *i-ésima* partícula y la *j-ésima* bandera. En la línea 4 se calcula la distancia euclidiana que hay entre la *i-ésima* partícula y la *j-ésima* bandera.

Las ecuaciones 4.11 y 4.12 se utilizan en las líneas 5 y 6 para calcular los valores mínimo y máximo del intervalo de error de la distancia hacia la *j-ésima* bandera. De las líneas 7 a 9 se verifica si la distancia entre la *i-ésima* partícula y la *j-ésima* bandera está entre el intervalo de error de la distancia percibida, si se encuentra dentro del intervalo, se aumenta en uno la ponderación de la *i-ésima* partícula. En la línea 10 se calcula el ángulo entre la *i-ésima* partícula y la *j-ésima* bandera, utilizando la funciones **atan2(b,a)**(ecuación 4.21 que se encuentra definida en la etapa de orientación), **rad(ϕ)** (ecuación 4.131) y **grad(ϕ)** (ecuación 4.132). Estas funciones se definen en las ecuaciones indicadas,

que se encuentran en la página 112.

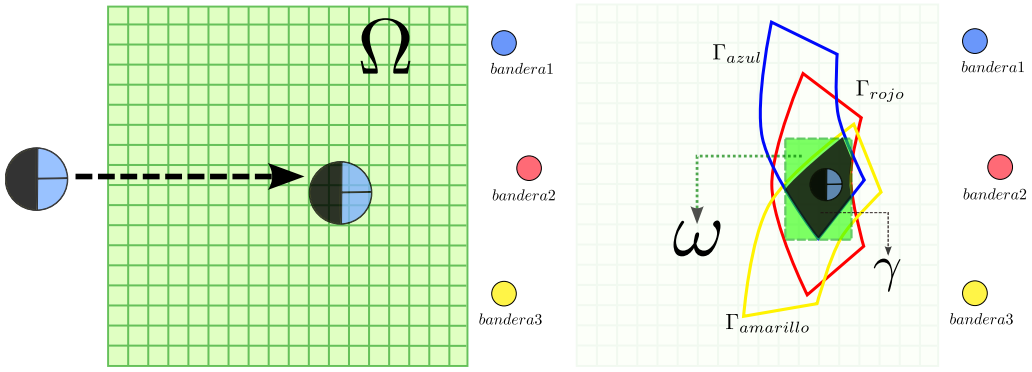
Las ecuaciones 4.18 y 4.19 se utilizan en las líneas 11 y 12 para calcular los valores mínimo y máximo del intervalo de error del ángulo hacia la *j-ésima* bandera. De las líneas 13 a 15 se verifica si el ángulo entre la *i-ésima* partícula y la *j-ésima* bandera está entre el intervalo de error del ángulo percibido, si se encuentra dentro del intervalo, se aumenta en uno la ponderación de la *i-ésima* partícula.

Al finalizar, el algoritmo regresa la variable *ponderación*, que representa la ponderación de la *i-ésima* partícula con respecto a la *j-ésima* bandera. Como se mencionó anteriormente, este proceso se realiza a cada partícula con todas las banderas percibidas.

Cuando se concluyen la etapa de predicción y corrección se obtiene la región ω, la cual es una región rectangular de posibles posiciones del agente, es importante considerar que la región ω se obtiene descartando partículas de la región Ω y por lo tanto es una región de menor tamaño, que al final es lo que se busca obtener, la región más pequeña de posibles posiciones del agente para poder elegir una posición única con el menor error posible. Para obtener una posición única del agente se obtiene el punto medio de la región ω.

El cálculo para obtener el punto medio de la región ω no se encuentra dentro del algoritmo 14 debido a que el algoritmo de localización PokTaPok no necesita una posición única, se debe recordar que el algoritmo recibe como entrada el arreglo *Partículas* (región Ω) y no es indispensable dentro del algoritmo tener una posición única, pero debido a que la mayoría de las tareas del agente necesitan una posición se calcula el punto medio de la región, obteniendo así los elementos $(\ x \quad y \ )$ de la pose del agente.

Todo el procedimiento de las etapas predicción y aproximación, mostradas en los algoritmos 12 y 14, se pueden resumir en la siguiente figura (4.11).



(a) Generando el *grid* (b) Reduciendo el *grid*

Figura 4.11: En la figura 4.11a se muestra cómo la región Ω se genera después de un desplazamiento del agente. En la figura 4.11b se observa la intersección entre tres regiones Γ construyendo la región γ (región de color negro), y se ve también la región ω (región color verde), la cual es el rectángulo más pequeño que contiene a la región γ.

Etapa de orientación

Para completar el vector pose final del agente, $(\ x \quad y \quad \theta\ )^T$, es necesario calcular la orientación θ, es decir, la dirección en la cual se encuentra el agente. Para poder calcular el elemento θ se emplea el algoritmo 16. Este algoritmo utiliza la función *atan2(y,x)*, la cual se define como se muestra en 4.21.

$$atan2(y,x) = \begin{cases} atan(y,x) & \text{si } x > 0 \\ sign(y)(\pi - atan(|\ \frac{y}{x}\ |)) & \text{si } x < 0 \\ 0 & \text{si } x = y = 0 \\ sign(y)\frac{\pi}{2} & \text{si } x = 0, y \neq x \end{cases} \tag{4.21}$$

En concreto la función *atan2(y,x)* calcula el valor del arco tangente de y/x en un rango de $[-\pi, \pi]$, utiliza funciones como *sign(a)*, la cual obtiene el signo del parámetro a, y también la función **atan(b)** que se encarga de calcular el arco tangente del parámetro b. La función *atan2(y,x)* es una variante de la función *atan(b)* que permite ajustar el ángulo resultante de manera inmediata para los cuadrantes en los que se puedan encontrar x y y. Una manera de ejemplificar la utilidad de la función *atan2(y,x)* es, si se necesita encontrar el ángulo hacia el punto $(-2, -2)$, como se muestra en la figura 4.12, el ángulo debe de ser de $\frac{5\pi}{4}$ o equivalente.

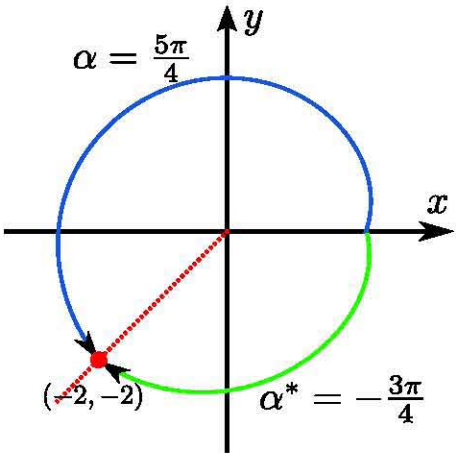


Figura 4.12: Figura que representa el uso de la función atan2(y,x).

Si se utiliza únicamente el arco tangente para calcular este ángulo el resultado sería

$$atan\left(\frac{y}{x}\right) = atan\left(\frac{-2}{-2}\right) = \frac{\pi}{4}$$

lo cual es un resultado erróneo. Si se aplica la definición de *atan2(y,x)* el resultado sería

$$atan2(-2,-2) = sing(-2)\left(\pi - atan\left(\left|\frac{-2}{-2}\right|\right)\right) = -\frac{3\pi}{4} \sim \frac{5\pi}{4}$$

este resultado es la prueba de que la función *atan2(y,x)* no requiere ajustar dependiendo el cuadrante donde se encuentre el punto. Además *atan2(y,x)* contempla el caso de indeterminación cuando $y = 0$ y el caso simple cuando $x = 0$, esto gracias a que es una función por partes.

Los cuatro casos de la función por partes *atan2(y,x)* se pueden ejemplificar con la figura 4.13.

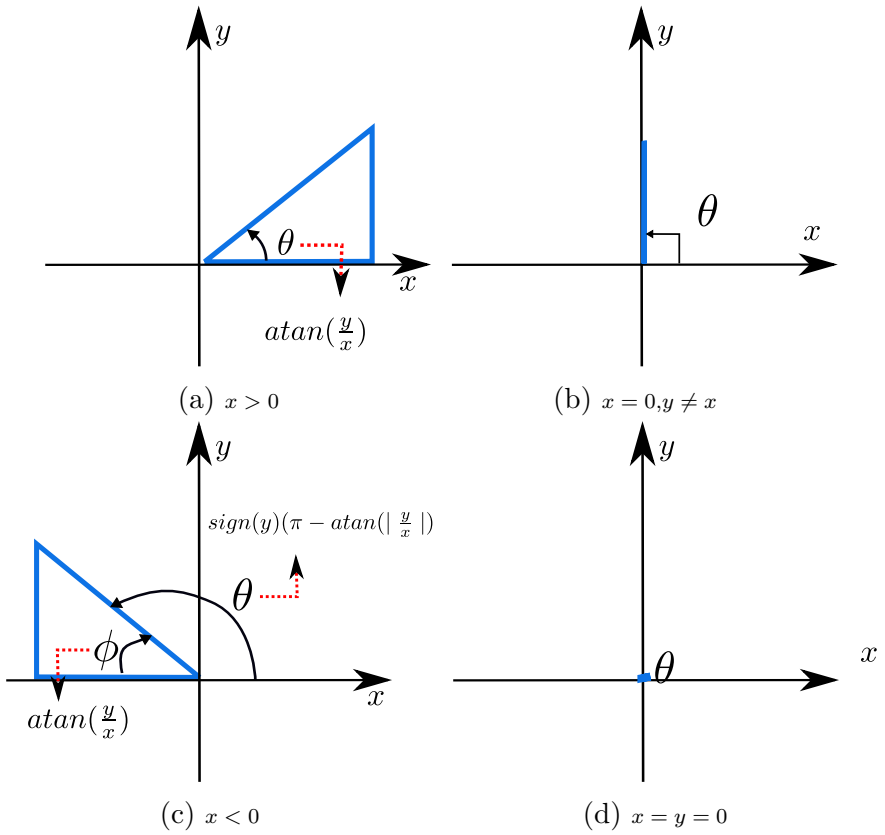


Figura 4.13: Representación gráfica de los cuatro casos de la función atan2(y,x).

Una vez explicada con detalle la función *atan2(y,x)* se puede ver que para obtener la orientación θ del agente se puede utilizar la ecuación 4.22.

$$\theta = norm(atan2(b_y - y, b_x - x) - b_\phi) \tag{4.22}$$

En la expresión 4.22 la función **norm(ϕ)** regresa un ángulo equivalente a su argumento ϕ en el intervalo $[-180, 180]$, esta función se define en el algoritmo 17 en la sección 4.4 112. La variable b denota la bandera, donde (b_x, b_y) expresan las coordenadas de la bandera dentro del campo de juego y b_ϕ expresa la dirección o ángulo al cual se encuentra la bandera con respecto al agente, es decir, este valor se optiene de la percepción del agente a través de su cono de visión. Las variables x y y representan la posición del agente en el campo. Un ejemplo gráfico de la ecuación 4.22 se puede observar en la figura 4.14.

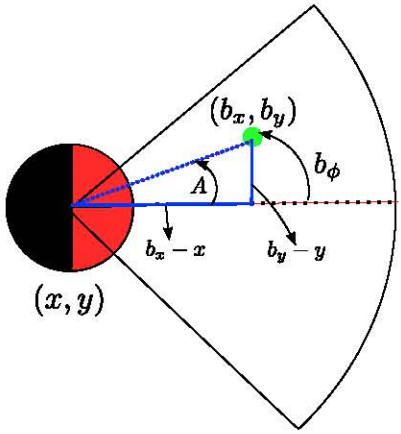


Figura 4.14: Las variables de la ecuación 4.22 de manera gráfica, con un agente percibiendo una bandera.

Es importante recordar, como se ha mencionado a lo largo de este trabajo, la existencia de incertidumbre por causa del ruido. Por lo tanto para hacer más certero el cálculo de θ es necesario emplear la percepción de todas las banderas. El método para calcular la orientación del agente calcula la orientación que se obtiene con cada bandera percibida por el agente en el instante t y se realiza un promedio de las orientaciones obtenidas. Esta mejora para calcular la orientación del agente utilizando todas las banderas percibidas parece muy simple, porque sólo se trata de obtener un promedio, pero no es posible calcular un promedio de ángulos sumando y dividiendo entre el total. Por ello se utiliza el algoritmo 16, basado en una simple fórmula que calcula el promedio con base en los senos y cosenos de cada ángulo.

Algoritmo 16 Orientation_PokTaPok $(Partículas_t, z_t)$

1: $sumaSin = sumaCos = 0$
2: **for** $i = 1$ **to** $size(z_t)$ **do**
3: $\quad orientacion = norm(\text{grad}(atan2(z_t^i.y - y\ ,\ z_t^i.x - x)) - z_t^i.\phi)$
4: $\quad sumaSin = sumaSin + \sin(rad(orientacion))$
5: $\quad sumaCos = sumaCos + \cos(rad(orientacion))$
6: **end for**
7: $sumaSin = sumaSin/size(z_t)$
8: $sumaCos = sumaCos/size(z_t)$
9: $Partículas_t.\theta = norm(grad(atan2(sumaSin, sumaCos)))-neckDir$
10: **return** $Partículas_t$

En el algoritmo 16, la función **size(v)** (4.4) da como resultado el tamaño de su argumento v, en este caso da el total de banderas percibidas en el instante t. La función **rad(ϕ)** (4.131) convierte su parámetro ϕ a radianes. La función **norm(ϕ)** (algoritmo 17) calcula el ángulo equivalente a su argumento a en el intervalo $[-180, 180]$ y la función **grad(ϕ)** (4.132) convierte su argumento ϕ a

grados.

Las entradas del algoritmo son el arreglo Partículas y la percepción z_t. Es importante recordar que el arreglo Partículas representa la región Ω y se recibe únicamente para completar la pose del agente agregando la orientación θ a los cuatro elementos del arreglo Partículas. La percepción z_t es un arreglo que contiene n banderas percibidas con su respectiva información obtenida a través del cono de visión del agente como la distancia y la dirección, ambos datos con ruido, y también el vector z_t contiene los datos correspondientes a un objeto referencia como etiqueta o identificador y coordenadas o ubicación en el campo, ver figura 1.5.

En la línea 1 el algoritmo comienza inicializando dos variables que se utilizan para acumular. De la línea 2 a la 6 hay una estructura de repetición que se realiza por cada bandera percibida, dentro de la estructura de repetición se calcula la orientación del agente con respecto a la *i-ésima* bandera percibida utilizando la ecuación 4.22, línea 3 del algoritmo. En las dos líneas siguientes se acumulan los valores de las funciones trigonométricas seno y coseno de la orientación obtenida en la línea 3. Este proceso se realiza por cada bandera percibida, acumulando cada valor de la función seno y coseno de cada orientación calculada. En la línea 7 se calcula el promedio de la función seno y en la línea 8 el promedio de la función coseno. Por último en la línea 9 se calcula orientación del agente utilizando la función *atan2(y,x)* y *normalizando* el resultado dentro de un intervalo de $[-180, 180]$. La variable *neckDir* representa los grados que existen de diferencia entre el cono de visión y la dirección del agente. Este valor el servidor lo proporciona sin error.

El algoritmo da como resultado el arreglo Partículas modificado en sus elementos Partícula.θ. Es importante recordar que la orientación obtenida se asigna a los cuatro elementos del arreglo Partículas en sus elementos θ.

Para concluir esta sección que habla sobre el algoritmo para calcular la orientación del agente, el algoritmo se puede expresar en una sola expresión considerando que las funciones *seno* y *coseno* son funciones cíclicas con un contra dominio en el intervalo de $[-1.0, 1.0]$, por lo tanto al considerar los *senos* y *cosenos* de todos los ángulos, se está tomando en cuenta ángulos equivalentes. Al final lo que se realiza es el promedio de los *senos* y *cosenos*, que son convertidos en ángulos.

Básicamente la función que realiza el algoritmo 16 es la que se muestra en la expresión 4.23.

$$\theta = norm(atan2(\frac{1}{n}\sum_{i=1}^{n}\sin(\phi_i), \frac{1}{n}\sum_{i=1}^{n}\cos(\phi_i))) - neckDir \tag{4.23}$$

Concluidas las tres etapas del algoritmo de localización PokTaPok se tiene el vector pose del agente en el tiempo t, $(\ x \quad y \quad \theta\ )^T$. Con este vector se tiene la ubicación y dirección del agente dentro del campo.

4.3.3. Complejidad del algoritmo

El algoritmo que se desarrolló para el sistema PokTaPok se dividió en tres etapas: predicción, corrección y orientación. Predicción y corrección se utilizan para calcular la coordenada (x, y) del agente dentro del campo de juego, mientras que la etapa de orientación se utiliza para saber la dirección θ del agente.

Para obtener la complejidad del algoritmo se calcula la complejidad de manera independiente de cada etapa. En la etapa de predicción se tiene una complejidad lineal, debido a que realiza únicamente operaciones constantes. La etapa de corrección contiene estructuras de repetición donde dos de ellas se encuentran anidadas, la estructura exterior se realiza p veces y la interior k veces, donde p representa el número de partículas y b representa la cantidad de banderas percibidas por el agente, con esto se tiene pb iteraciones. Las siguientes estructuras de repetición se realizan p y cuatro veces, respectivamente, por lo que se concluye que esta etapa tiene una complejidad de: $O(pb + p)$.

En la complejidad de la etapa de corrección es muy importante enfatizar acerca del valor de p, que representa el número de partículas. El número de partículas depende de la granularidad con la que se desee trabajar, así como también del tamaño de la región Ω.

Una parte muy importante en la complejidad es el análisis de la peor situación o caso en el que se ejecute un algoritmo. El algoritmo PokTaPok tendría el peor de sus casos si la región Ω creciera indefinidamente, debido a que su complejidad depende de sus partículas y al crecer más la región aumenta el número de partículas.

La región Ω crece indefinidamente cuando no existe o no se puede efectuar la etapa de corrección y la región crece en cada intervalo de tiempo, como se muestra en la figura 4.15.

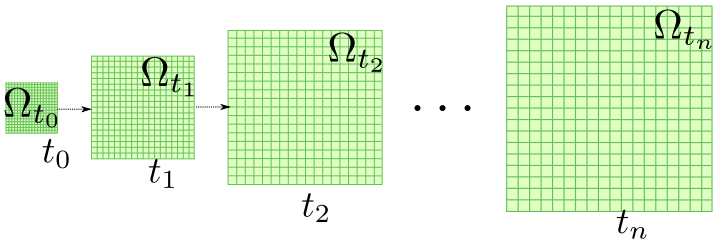


Figura 4.15: La región Ω crece indefinidamente a través del tiempo. No existe la etapa de corrección.

Para detallar el caso donde Ω crece indefinidamente y saber la manera en la que crece esta región se supondrá lo siguiente. Sea la región Ω inicial en el tiempo $t = 0$ únicamente un punto en el plano, una partícula y para el caso $t = 1$ cuatro partículas. Suponga que la región crece indefinidamente y proporcionalmente el tamaño de una partícula en cada lado de la región rectangular (para este caso cuadricular) a través del tiempo, de la manera en que se ejemplifica en la figura 4.16.

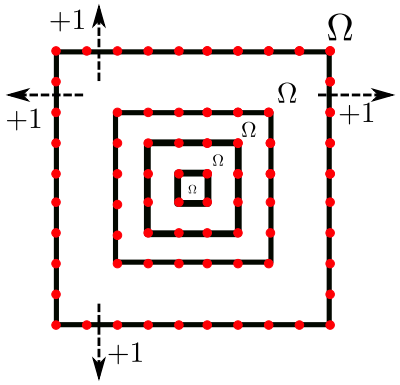


Figura 4.16: La región Ω crece "una partícula" en cada lado. Las partículas son representadas con los puntos rojos y las líneas negras representan la región Ω a través del tiempo.

Como se puede ver en la figura 4.16, el número de partículas en la región Ω crece como se indica en la relación de recurrencia 4.24 que se muestra a continuación.

$$
\begin{aligned}
t_0 &= 1 \\
t_1 &= 4 \\
t_i &= \left(\sqrt{t_{i-1}} + 2\right)^2
\end{aligned}
\tag{4.24}
$$

En la tabla 4.2 se muestran algunos resultados de la relación de recurrencia descrita anteriormente (4.24), junto con los resultados de las sucesiones para n^2 y n^3. Se observa que la sucesión de la relación de recurrencia está acotada por n^2 y n^3.

n^2	1	4	9	16	25	36
ecuación 4.24	1	4	16	36	64	100
n^3	1	8	27	64	125	216

Tabla 4.2: Tabla con algunos valores de la relación de recurrencia que describe el crecimiento de la región Ω en una partícula por cada lado de la región y las sucesiones n^2 y n^3.

En el ejemplo anterior se vio como se comporta el crecimiento del número de partículas en Ω cuando dicha región crece una partícula por lado, es decir, en dos partículas horizontalmente y dos partículas verticalmente. Este crecimiento se ve reflejando en la constante $+2$ de la relación de recurrencia 4.24. Pero una forma más realista de analizar el crecimiento del número de partículas sería asumir que la cantidad de partículas aumenta en $+M$, en lugar de $+2$. Dado que $M = \frac{r_{max}}{k}$, donde r_{max} representa el desplazamiento máximo que puede realizar un agente y k representa la granularidad. Por lo tanto la relación de recurrencia para el crecimiento del número de partículas en la región Ω sería descrita como se muestra en 4.25.

(4.25)
$$\begin{aligned} t_0 &= 1 \\ t_1 &= 4 \\ t_i &= \left(\sqrt{t_{i-1}} + \frac{r_{max}}{k}\right)^2 \end{aligned}$$

Sin embargo el análisis anterior supone que el agente no percibió ninguna bandera. Este caso no sucede durante un partido dentro del servidor *soccerserver*, únicamente ocurre si el agente se encuentra fuera del campo de juego y viendo en dirección opuesta al campo, lo cual no tiene sentido.

Para considerar el peor caso posible que ocurra durante un partido en el servidor *soccerserver*, se asume que la etapa de corrección es realizada en todo tiempo t_i y se analiza la situación en la cual la reducción de la región Ω es mínima, es decir, será la reducción de mayor área. Véase la figura 4.17.

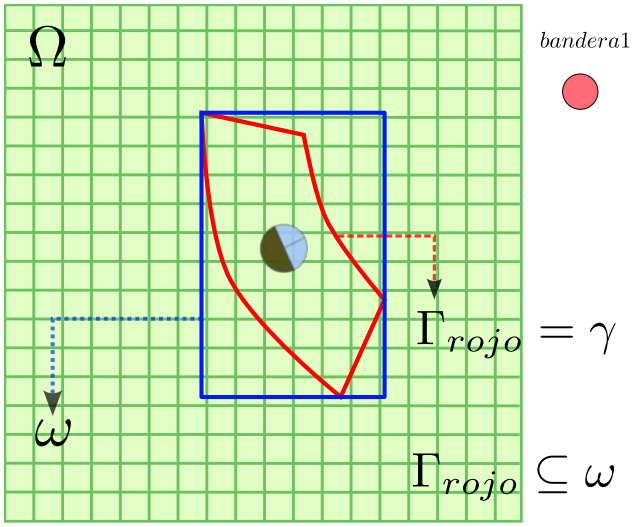
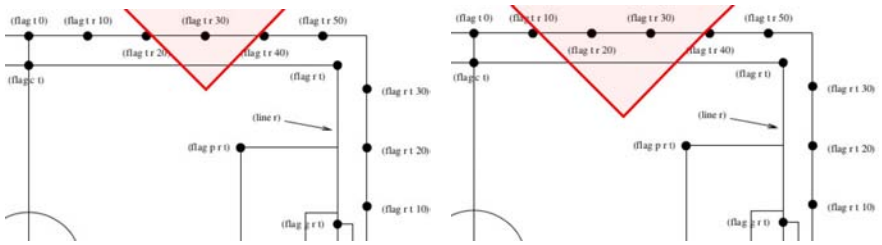


Figura 4.17: Etapa de corrección percibiendo una sola bandera y obteniendo la *"reducción máxima"*.

La reducción con el área máxima puede ocurrir considerando ciertos casos particulares. Uno de los casos sería que el agente únicamente percibiera una sola bandera, con la cual se genera una región Γ y por lo tanto en la etapa de corrección no existirían intersecciones, entonces $\Gamma \subset \omega$, es el caso que se muestra en la figura 4.17. Otro caso que se puede considerar como el peor caso, es donde el agente perciba dos banderas y la intersección de ambas regiones Γ generadas sea mayor a la región Γ más grande que se pueda generar al percibir solamente una bandera. Esto puede ocurrir debido al tipo de error que existe en la percepción de las banderas, entre más banderas se perciben existen más intersecciones, pero puede ser que vean varias banderas lejanas ocasionando regiones Γ grandes y por ello la intersección podría ser aún mayor que una sola región Γ, debido a que bandera percibida puede estar cerca del agente. Es claro ver que existen varias combinaciones entre las banderas en el campo (figura 1.5) y posiciones del agente en el campo para buscar un peor caso, pero utilizando el sentido común, se pueden considerar un par de casos como los peores. Los dos posibles peores casos se muestran en la figura 4.18.



(a) Peor caso percibiendo solo una bandera. (b) Peor caso percibiendo dos banderas.

Figura 4.18: Representación de los dos posibles peores casos. Las áreas de color rojo representan el cono de visión de un agente y cómo sería la forma de percibir una sola bandera o únicamente dos banderas.

Con fines prácticos, se analizará el caso donde el agente percibe únicamente una bandera lo más lejos posible, véase figura 4.18a. Las dimensiones de la región ω con área máxima se puede calcular con base en el modelo de percepción (sección 4.2) y considerando el análisis del peor caso, debido a que el tamaño depende de que tan lejos se encuentre la bandera como se mostrará a continuación con el siguiente análisis de la figura 4.19.

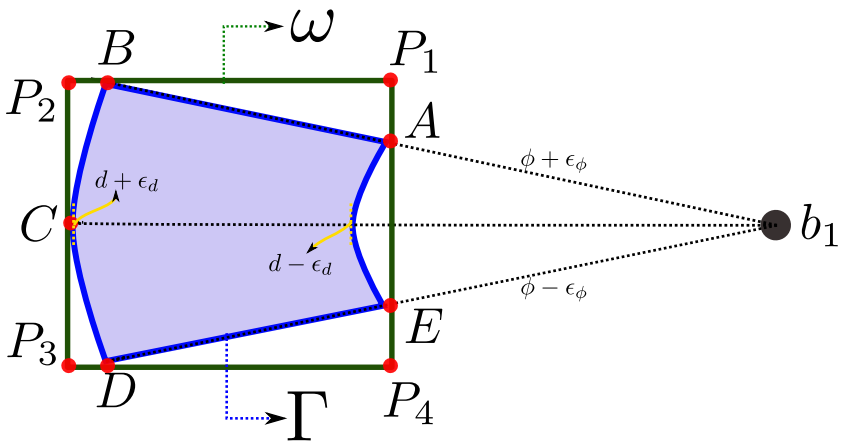


Figura 4.19: Puntos que definen las regiones Γ, generada por una bandera b_1, y la región ω máxima.

Considerando el dibujo 4.19, se pueden obtener las coordenadas polares con respecto a la bandera percibida de los puntos de la región Γ (A,B,C,D,E). Véase la tabla 4.3.

	radio	*ángulo*
A	$d - \epsilon_d$	$\phi + \epsilon_\phi$
B	$d + \epsilon_d$	$\phi + \epsilon_\phi$
C	$d + \epsilon_d$	ϕ
D	$d + \epsilon_d$	$\phi - \epsilon_\phi$
E	$d - \epsilon_d$	$\phi - \epsilon_\phi$

Tabla 4.3: Tabla con las coordenadas polares de los puntos de la región Γ.

Utilizando las coordenadas polares con respecto a la bandera b_1 de la región Γ de la figura 4.19, se calculan las coordenadas cartesianas, las cuales se muestran en la tabla 4.4.

	x	y
A	$(d - \epsilon_d)\cos(\phi + \epsilon_\phi)$	$(d - \epsilon_d)\sin(\phi + \epsilon_\phi)$
B	$(d + \epsilon_d)\cos(\phi + \epsilon_\phi)$	$(d + \epsilon_d)\sin(\phi + \epsilon_\phi)$
C	$(d + \epsilon_d)\cos(\phi_\phi)$	0
D	$(d + \epsilon_d)\cos(\phi - \epsilon_\phi)$	$(d + \epsilon_d)\sin(\phi - \epsilon_\phi)$
E	$(d - \epsilon_d)\cos(\phi - \epsilon_\phi)$	$(d - \epsilon_d)\operatorname{sen}(\phi - \epsilon_\phi)$

Tabla 4.4: Tabla con las coordenadas cartesianas de los puntos de la región Γ.

Una vez obtenidas las coordenadas cartesianas de la región Γ se pueden calcular las coordenadas cartesianas de la región ω, en la figura 4.19 representada por los puntos P_1,P_2,P_3 y P_4. Las coordenadas se muestran en la tabla 4.5. Y conociendo las coordenadas cartesianas se puede calcular el área de la región ω.

	x	y
P_1	$(d - \epsilon_d)\cos(\phi + \epsilon_\phi)$	$(d + \epsilon_d)\sin(\phi + \epsilon_\phi)$
P_2	$(d + \epsilon_d)\cos(\phi)$	$(d + \epsilon_d)\sin(\phi + \epsilon_\phi)$
P_3	$(d + \epsilon_d)\cos(\phi)$	$(d + \epsilon_d)\sin(\phi - \epsilon_\phi)$
P_4	$(d - \epsilon_d)\cos(\phi - \epsilon_\phi)$	$(d + \epsilon_d)\sin(\phi - \epsilon_\phi)$

Tabla 4.5: Tabla con las coordenadas cartesianas de los puntos de la región ω.

Sí las coordenadas obtenidas toman como referencia la bandera b_1, por lo tanto $\phi = 0$ y también se sabe que el error en el ángulo percibido es de un grado, $\epsilon_\phi = 1$ (sección 4.2), por lo tanto las coordenadas cartesianas de la región ω se simplifican como se muestra en la tabla 4.6.

	x	y
P_1	$(d-\epsilon_d)\cos(1)$	$(d+\epsilon_d)\sin(1)$
P_2	$d+\epsilon_d$	$(d+\epsilon_d)\sin(1)$
P_3	$d+\epsilon_d$	$(d+\epsilon_d)\sin(-1)$
P_4	$(d-\epsilon_d)\cos(-1)$	$(d+\epsilon_d)\sin(-1)$

Tabla 4.6: Tabla con las coordenadas cartesianas de los puntos de la región ω simplificadas.

En la tabla 4.6, las dimensiones de la región ω dependen únicamente de la distancia d y el error ϵ_d, donde el error ϵ_d aumenta si la distancia d, distancia de percepción a la bandera, aumenta (sección 4.2). La distancia d se puede obtener con ayuda de la figura 4.18 y utilizando leyes de senos se obtiene que la distancia máxima a la que un agente puede percibir solo una bandera es $d = 14.14$. Si el agente se encuentra a esa distancia el intervalo considerando el error sería $[9.92443601, 10.12492291]$, las dimensiones de la región ω serían de 0.20048690 por lado. Entonces si en el algoritmo de localización PokTaPok se utilizó una granularidad de 0.05 y considerando el peor caso donde el agente percibió solo una bandera, el número de iteraciones para este caso es de 25 iteraciones, considerando un redondeo hacia arriba para tomar en cuenta los decimales y no truncar las dimensiones de la región ω.

La complejidad de la etapa de orientación es $O(b)$, es decir, se itera el número de banderas percibidas. Por último para obtener la complejidad total del algoritmo de localización PokTaPok se agrupan los resultados de las tres etapas, dando una complejidad:

$$O(pb + p + b)$$

.

Entonces complejidad del algoritmo de localización PokTaPok está en función de las partículas (región Ω y granularidad) y las banderas percibidas.

4.4. Resultados

Una vez implementado el algoritmo de localización para el sistema multiagente PokTaPok, se ejecutaron diez partidos en el servidor *soccerserver* y se obtuvieron las coordenadas de la ubicación de los agentes durante cada ciclo en cada uno de los partidos. Los datos obtenidos se guardaron en archivos de texto plano para poder compararlos con la localización real de los agentes en cada ciclo, la cual se obtiene de un archivo generado por el servidor *soccerserver* en cada partido (log del servidor). El *log del servidor* contiene toda la información sobre la simulación de todo el partido, incluyendo la posición exacta de cada

agente. Con la información obtenida del *log del servidor* se calculó el error en la localización durante cada ciclo del juego de cada agente.

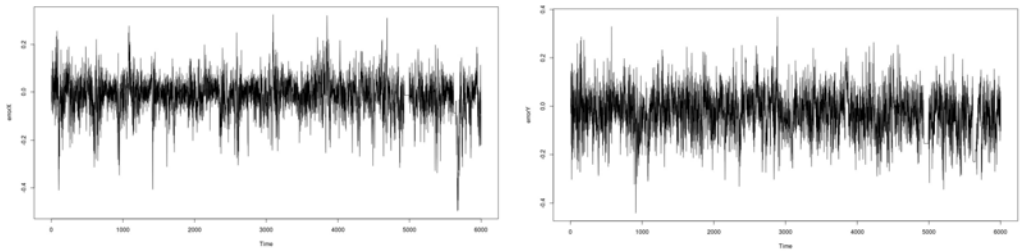
Se utilizaron hojas de calculo en el software *LibreOffice Calc* para trabajar los datos de la localización y obtener el error. En la tabla 4.7 se muestra un ejemplo de como se obtuvo la serie de errores de un agente.

	Algoritmo		Servidor		Error	
Ciclo	x	y	x	y	x	y
⋮	⋮	⋮	⋮	⋮	⋮	⋮
861	34.5	20.5	34.9	19.9	0.4	-0.6
862	35.7	21.6	35.2	21.2	-0.5	-0.4
863	36.3	20.9	36.5	21.4	0.2	0.5
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Tabla 4.7: Ejemplo de una sección de la hoja de cálculo donde se trabajaron con los datos de la localización.

Una vez obtenido el error, de decidió realizar un análisis para tener una métrica sobre la precisión de la localización del agente. A continuación se muestran los resultados de un solo agente en un solo partido, pero los resultados en los demás partidos fueron similares.

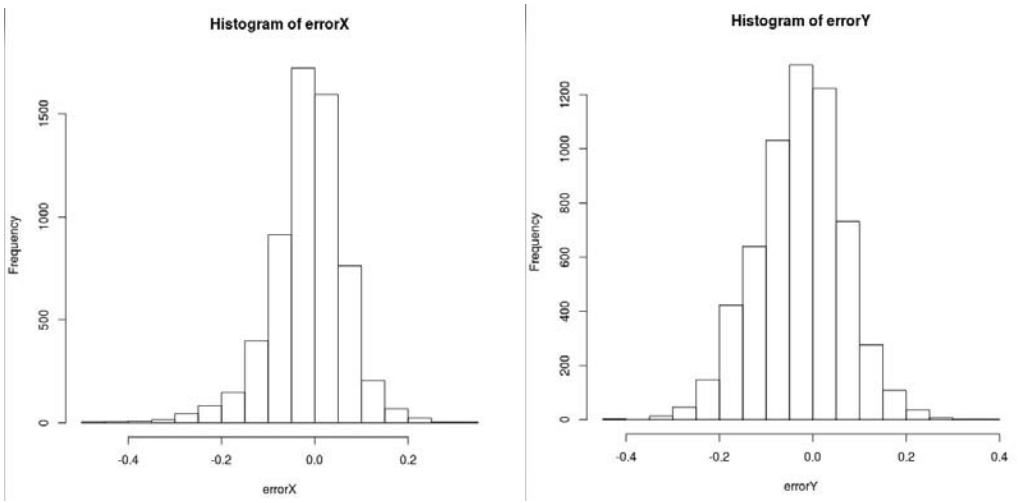
En principio se calculó el promedio y la desviación estándar del error resultante al comparar la información de la localización obtenida con el algoritmo y la localización proporcionada por el servidor, dando como resultado valores muy cercanos a cero. La serie de errores de un agente se muestran a continuación en la figura 4.20, al igual que el histograma de los errores en la figura 4.21, donde la media del error fue de -0.0150120122 para la coordenada x y -0.0271009405 para la coordenada y, y desviaciones estándar de 0.0808804963 y 0.0925032946, respectivamente. En el cálculo de la media y la desviación estándar de los agentes restantes los resultados fueron similares.



(a) Error en la coordenada x (b) Error en la coordenada y

Figura 4.20: Gráficos de error a través del tiempo de un agente.

En la siguiente figura (4.21) se muestran los histogramas correspondientes a los errores de la localización en la coordenada x y y. Ambos gráficos se realizaron con ayuda del software *R*.



(a) Histograma de los errores en la coordenada x (b) Histograma de los errores en la coordenada y

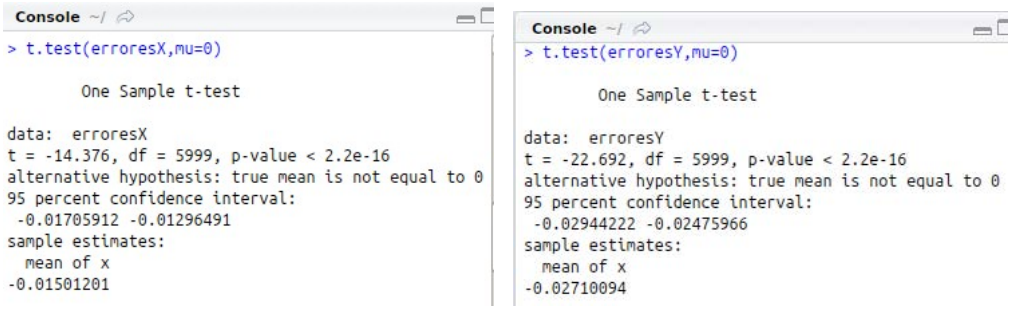
Figura 4.21: Gráficos del agente

Después de ver los excelentes resultados con respecto a la desviación estándar y la media del error, y basándose en la forma de los histogramas de los errores, se decidió hacer un análisis para saber si el error se comportaba como ruido blanco gaussiano, sin embargo no fue el caso.

La manera "óptima" en la cual se debe comportar una serie de errores es como ruido blanco gaussiano, debido a que tiene media cero, desviación estándar constante y la mayoría de los datos se encontrarían cercanos a cero. Por esta razón fue que se decidió realizar las pruebas necesarias para verificar si los datos obtenidos se comportaban como ruido blanco gaussiano. Para que una serie de datos se comporte como ruido blanco gaussiano se deben de cumplir que los datos deben de tener media cero, que los datos no tengan correlación, que su varianza sea constante y que se comporten como una distribución normal. Para verificar estos aspectos se realizaron las siguientes pruebas con ayuda del software R.

La primer prueba que se realizó fue *test-t*. Esta prueba verifica si una serie de datos tiene media cero.

- **Test-t:**



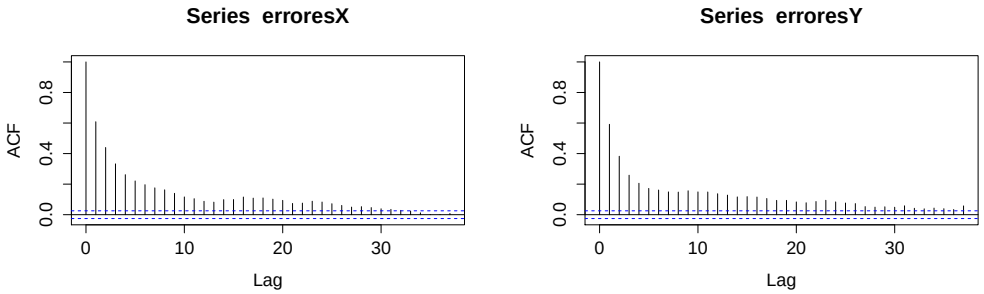
(a) t-test para error en x (b) t-test para error en y

Figura 4.22: Pruebas de media cero (t-test)

El resultado indica con un 95 % de confiabilidad que los datos no tienen media cero. Esto se puede ver a través de *p-value*, el cual tiene un valor muy pequeño y por lo tanto se rechaza la hipótesis nula.

La segunda prueba que se realizó a los datos fue el *periodograma*. Con el periodograma se corrobora si los datos están correlacionandos. Esta prueba consiste en un gráfico el cual muestra la correlación que existe entre la serie de errores, las barras del gráfico no deben sobrepasar los límites para que no exista correlación entre los datos.

- **Periodograma:**



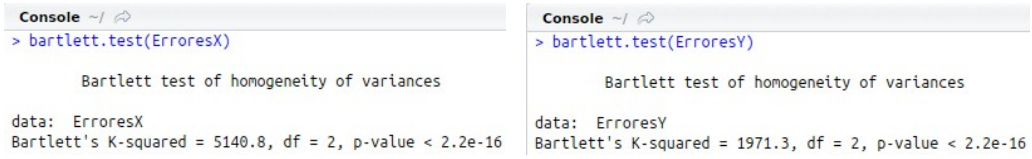
(a) Periodograma error en x (b) Periodograma error en y

Figura 4.23: Periodogramas de los errores en x y y.

La hipótesis nula para el caso del periodograma es que los datos no están correlacionados y con el resultado obtenido se tienen pruebas suficientes para rechazar la hipótesis nula, es decir, existe correlación entre la serie de errores.

La tercer prueba es el test de Bartlett, la cual sirve para verificar si se tienen varianzas constantes. Para esta prueba es necesario ingresar más de un conjunto de datos, por lo cual se decidió ingresar la serie de errores de tres agentes y verificar si los tres conjuntos tienen varianzas constantes.

- **Test de Bartlett:**



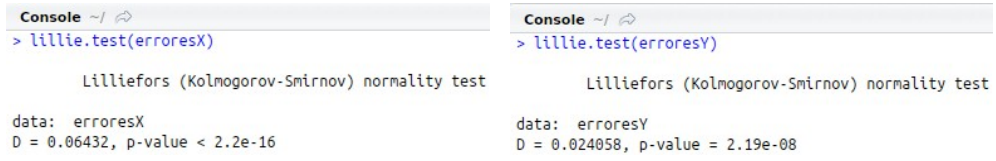
(a) Prueba de Bartlett para el error en x (b) Prueba de Bartlett para el error en y

Figura 4.24: Pruebas de Bartlett

Después de aplicar la prueba de Bartlett, se puede ver en el *p-value* que el valor es muy pequeño en ambos casos, por lo tanto se rechaza la hipótesis nula que dice que los datos tienen una varianza constante, entonces no hay pruebas suficientes para aprobar esta hipótesis.

Al finalizar estas pruebas se puede concluir que los errores no se comportan como ruido blanco, no cumplen las condiciones necesarias. Finalmente se realiza la prueba para verificar si los errores siguen una distribución normal. Para ello se realiza la prueba de *Kolmogorov-Smirnov*. Los resultados de esta prueba se muestran en la figura 4.25.

- **Kolmogorov-Smirnov:**



(a) Prueba de Kolmogorov-Smirnov para los errores en x (b) Prueba de Kolmogorov-Smirnov para los errores en y

Figura 4.25: Pruebas Kolmogorov-Smirnov

Los resultados de la prueba de *Kolmogorov-Smirnov* indica que se rechaza la hipótesis nula, es decir, se rechaza que los datos se comporten de manera normal.

Al finalizar las pruebas se concluye que los errores no se comportan como ruido blanco gaussiano, pero el promedio y desviación estándar de los errores muestran excelentes resultados del algoritmo de localización PokTaPok.

Conclusiones

Dentro del contexto de la robótica, es indispensable el uso de técnicas probabilísticas debido a la cantidad de incertidumbre que existe y estas técnicas es una manera muy eficiente de combatirla. Es muy importante enfatizar que la incertidumbre genera ruido y el ruido ocasiona errores en las acciones que se quieran realizar.

El objetivo fundamental de esta tesis fue abordar el problema de la localización de robots móviles y proponer una técnica adecuada para el desarrollo de un algoritmo de localización contemplando la incertidumbre, para al sistema multiagente PokTaPok.

Se logró desarrollar un algoritmo con el cual se localiza a los jugadores del sistema de fútbol robótico PokTaPok dentro del campo de juego. El algoritmo se basa en una técnica no paramétrica conocida como filtro de partículas. Cuando esta técnica se aplica al problema de localización, se le conoce con el nombre de localización Monte Carlo.

Al analizar el algoritmo de localización PokTaPok se tuvo una complejidad: $O(p(b+1)+b)$, donde p es la cantidad de partículas y b es la cantidad de banderas percibidas por el agente. Como se mencionó anteriormente, p depende de la región Ω, que es una región rectangular de posibles posiciones, donde cada posición es un vertice de la cuadrícula generada dentro de la región y a su vez esta cuadrícula depende de la granularidad con la que se decida trabajar.

El sistema PokTaPok participó en el Torneo Mexicano de Robótica 2013 en Puebla, donde se venció al equipo del Tecnológico de Monterrey. Un año después clasificó a la RoboCup en Brasil.

Como trabajo futuro sería interesante saber que complejidad tiene la intersección de las regiones γ utilizando series de Davenport-Schinzel [21] y calcular el error en lugar de la región rectangular ω, con la que se trabajo a lo largo de esta tesis. También sería interesante trabajar el algoritmo de localización en sistemas donde el ambiente no este controlado, donde exista incertidumbre

real y no solo simulada, lo cual tendría como consecuencia trabajar la misma idea del algoritmo de localización PokTaPok bajo otras distribuciones de error.

Anexos

Derivaciones

Derivación del algoritmo 1

Para realizar la derivación del algoritmo del filtro de Bayes (algoritmo 1) se utilizarán las definiciones de teorema de Bayes (1.3) y probabilidad total. Si se aplica el teorema de Bayes a la definición de creencia $bel(x_t)$, se tiene lo siguiente:

$$p(x_t \mid z_{1:t}, u_{1:t}) = \frac{p(z_t \mid x_t, z_{1:t-1}, u_{1:t}) p(x_t \mid z_{1:t-1}, u_{1:t})}{p(z_t \mid z_{1:t-1}, u_{1:t})} \tag{4.26}$$
$$= \eta p(z_t \mid x_t, z_{1:t-1}, u_{1:t}) p(x_t \mid z_{1:t-1}, u_{1:t})$$

La simplificación de la ecuación anterior se puede ver de manera más simple:

$$p(A \mid B, C) = \frac{p(B \mid A, C) p(A \mid C)}{p(B \mid C)}$$

asignando las variables de la siguiente forma:

$$A = x_t \quad B = z_t \quad C = z_{1:t-1}, u_{1:t}$$

y η reemplaza al denominador que es constante en la regla de Bayes, que es conocido como constante de normalización.

Considerando el estado x_t como completo. No es necesario las percepciones, ni controles para calcular la probabilidad de z_t. Por esta razón, se puede expresar:

$$p(z_t \mid x_t, z_{1:t}, u_{1:t}) = p(z_t \mid x_t) \tag{4.27}$$

Colocando la expresión (4.27) en la ecuación (4.26) se puede escribir de la siguiente manera:

$$p(x_t \mid z_{1:t}, u_{1:t}) = \eta p(z_t \mid x_t) p(x_t \mid z_{1:t-1}, u_{1:t}) \tag{4.28}$$

Para simplificar más la expresión (4.28) se utiliza la definición de creencia estimada $\overline{bel}(x_t)$ y de la creencia $bel(x_t)$, entonces tiene:

$$bel(x_t) = \eta p(z_t \mid x_t)\overline{bel}(x_t) \tag{4.29}$$

como se puede ver, esta ecuación es la implementada en la línea 3 del algoritmo del filtro de Bayes (1).

En la línea 2 del algoritmo 1 se encuentra la creencia estimada $\overline{bel}(x_t)$, de la cual su definición es:

$$\overline{bel}(x_t) = p(x_t \mid z_{1:t-1}, u_{1:t}) \tag{4.30}$$

Con esta expresión se utiliza la definición del teorema de probabilidad total, y decir que la probabilidad de x_t es la probabilidad de conjunción con todos los posibles x_{t-1} dado que han ocurrido los controles y percepciones pasadas.

$$p(x_t \mid z_{1:t-1}, u_{1:t}) = \sum_{x_{t-1}} p(x_t, x_{t-1} \mid z_{0:t-1}, u_{0:t}) \tag{4.31}$$

Con la definición de probabilidad condicional se puede escribir la expresión como:

$$p(x_t \mid z_{0:t-1}, u_{t_{0:t}}) = \sum_{x_{t-1}} p(x_t \mid x_{t-1}, z_{0:t-1}, u_{0:t}) p(x_{t-1} \mid z_{0:t-1}, u_{0:t}) \tag{4.32}$$

como se esta considerando estado como completo la expresión se reduce a:

$$p(x_t \mid z_{0:t-1}, u_{t_{0:t}}) = \sum_{x_{t-1}} p(x_t \mid x_{t-1}, u_t) p(x_{t-1} \mid z_{0:t-1}, u_{0:t}) \tag{4.33}$$

En el último factor, la variable u_t, no influye en la probabilidad de x_{t-1}, por lo cual se elimina. Entonces el factor queda como $p(x_{t-1} \mid z_{0:t-1}, u_{0:t-1})$. Y por último se aplica las definiciones de $bel(x_{t-1})$ y $\overline{bel}(x_t)$.

$$\overline{bel}(x_t) = \sum_{x_{t-1}} p(x_t \mid x_{t-1}, u_t) bel(x_{t-1}) \tag{4.34}$$

Si el estado fuera continuo:

$$\overline{bel}(x_t) = \int_{x_{t-1}} p(x_t \mid x_{t-1}, u_t) bel(x_{t-1}) \delta x_{t-1} \tag{4.35}$$

que es la línea 2 del algoritmo 1 .

La estructura de repetición en el algoritmo 1 se aplica para calcular la probabilidad de todos los posibles estados x_t.

En resumen, los tres requisitos indispensables, para la implementación del algoritmo del filtro de Bayes son: La creencia inicial $p(x_0)$ o bien $bel(x_0)$, la probabilidad de percepción $p(z_t \mid x_t)$ y la

probabilidad del estado de transición $p(x_t \mid u_t, x_{t-1})$. En la sección 1.5 se describieron algunos modelos con los cuales se puede conocer este tipo de densidades, que son necesarias para la aplicación del algoritmo 1.

Derivación del algoritmo 2

La interpretación gráfica que se le da al modelo de velocidad de movimiento y que ayuda a entender la derivación del algoritmo es la suposición de que el robot o agente se desplaza en trayectorias de segmentos de círculo.

Para comenzar la derivación se debe suponer que las velocidades de rotación y traslación son fijas en el intervalo de tiempo $(t-1, t]$, y entonces el robot o agente se mueve en un segmento de círculo con radio:

$$r = \left|\frac{v}{w}\right| \tag{4.36}$$

La expresión anterior se puede observar de la siguiente manera:

$$v = w \cdot r \tag{4.37}$$

Es fácil de interpretar si se recuerda que se está asumiendo que el agente se desplaza en segmentos de círculo, por lo tanto se puede recurrir a la ecuación para calcular el perímetro de un círculo que es:

$$\text{perímetro} = \underbrace{2\pi}_{\text{ángulo}} \cdot \underbrace{radio}_{distancia} \tag{4.38}$$

Utilizando la ecuación 4.38 se puede decir que v es la distancia recorrida sobre el círculo de radio r en un tiempo Δt.
En la ecuación 4.36 existe un caso especial y es cuando w es cero, es decir, no existe una velocidad de rotación, y en este caso la ecuación queda indeterminada por la división entre cero. Para este caso, el desplazamiento del agente deja de ser en segmentos de circunferencias y se desplaza en líneas rectas.

Considerando $x_{t-1} = (\; x \quad y \quad \theta \;)^T$ como la pose inicial del agente y suponiendo velocidades constantes $(\; v \quad w \;)^T$ en un tiempo constante Δt, se observa fácilmente que el centro del círculo por el cual se desplaza el agente, está dado por las ecuaciones:

$$x_c = x - \frac{v}{w} \sin\theta \tag{4.39}$$

$$y_c = y + \frac{v}{w} \cos\theta \tag{4.40}$$

Para obtener estas ecuaciones se utiliza la figura 4.26, que se muestra a continuación:

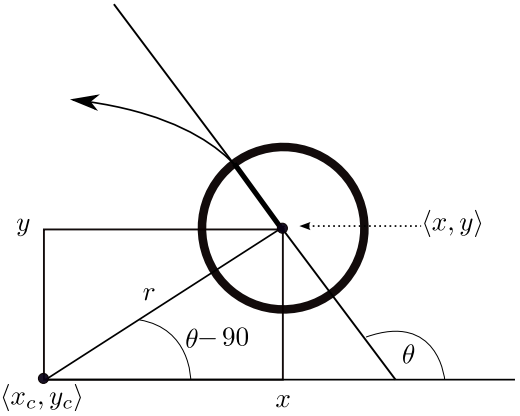


Figura 4.26: Movimiento sin ruido en segmentos de círculos del agente.

Se sabe que r es perpendicular al angulo θ, donde θ es la orientación del agente. Se define $\alpha = \theta - 90$, y con ayuda de las coordenadas polares y el triangulo rectángulo construido en la figura 4.26 se tiene:

$$x_c = x - r\cos\alpha \rightarrow x_c = x - r\cos(\theta - 90)$$
$$y_c = y - r\sin\alpha \rightarrow y_c = y - r\sin(\theta - 90)$$

Utilizando propiedades trigonométricas, se puede escribir las siguiente expresiones:

$$\begin{aligned} \star \cos(\theta - 90) &= \cos\theta\cos\frac{-\pi}{2} - \sin\theta\sin\frac{-\pi}{2} \\ &= -\sin\theta\sin\frac{-\pi}{2} \\ &= -\sin\theta(-1) \\ &= \sin\theta \end{aligned}$$

$$\star \sin(\theta - 90) = \sin\theta \cos\frac{-\pi}{2} + \sin\frac{-\pi}{2}\cos\theta$$
$$= \sin\frac{-\pi}{2}\cos\theta$$
$$= (-1)\cos\theta$$
$$= -\cos\theta$$

Con las igualdades resultantes es como se crean las ecuaciones 4.39 y 4.40, que se utilizan para obtener la pose del agente después de Δt unidades de tiempo, la cual se expresaría de la siguiente manera:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x_c + \frac{v}{w}\sin(\theta + w\Delta t) \\ y_c - \frac{v}{w}\cos(\theta + w\Delta t) \\ \theta + w\Delta t \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{w}\sin\theta + \frac{v}{w}\sin(\theta + w\Delta t) \\ \frac{v}{w}\cos\theta - \frac{v}{w}\cos(\theta + w\Delta t) \\ w\Delta t \end{pmatrix} \tag{4.41}$$

Movimiento real

Estas expresiones fueron obtenidas con trigonometría y considerando que el agente no cambia su velocidad durante el intervalo de tiempo Δt y que además avanza y gira con precisión exacta, es decir, que no tiene ruido. Pero como se dijo en el primer capítulo, todo sistema robótico está sujeto a ruido o incertidumbre, por ello para contemplar el ruido o para poder modelarlo, se asume que las velocidades están dadas por la ecuación 4.42:

$$\begin{pmatrix} \hat{v} \\ \hat{w} \end{pmatrix} = \begin{pmatrix} v \\ w \end{pmatrix} + \begin{pmatrix} \varepsilon_{\alpha_1 v^2 + \alpha_2 w^2} \\ \varepsilon_{\alpha_3 v^2 + \alpha_4 w^2} \end{pmatrix} \tag{4.42}$$

donde ε_{b^2} es una variable de error con media cero y varianza b^2, así con esta expresión se agrega ruido a las velocidades del agente.

Sustituyendo la ecuación 4.42, de las velocidades con ruido agregado, se puede expresar la ecuación 4.42 de la siguiente manera:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}}{\hat{w}}\sin\theta + \frac{\hat{v}}{\hat{w}}\sin(\theta + \hat{w}\Delta t) \\ \frac{\hat{v}}{\hat{w}}\cos\theta - \frac{\hat{v}}{\hat{w}}\cos(\theta + \hat{w}\Delta t) \\ \hat{w}\Delta t \end{pmatrix} \tag{4.43}$$

Orientación final

El radio del segmento circular y la distancia recorrida están influenciados por el control u con ruido, por lo tanto es muy probable que la trayectoria no sea circular. Debido a esto la orientación final también se ve afectada, y es necesario agregar una variable más de ruido para que el modelo de velocidad de movimiento sea más general.
La orientación final estaría dada por la ecuación 4.44:

$$\theta' = \theta - \hat{w}\Delta t' + \hat{\gamma}\Delta t \tag{4.44}$$

donde

$$\hat{\gamma} = \varepsilon_{\alpha_5 v^2 + \alpha_6 w^2} \tag{4.45}$$

Aquí α_5 y α_6 son parámetros específicos del agente que determinan la varianza del ruido de rotación adicional. Así el resultado de modelo de movimiento es como se muestra a continuación en la ecuación 4.46:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}}{\hat{w}}\sin\theta + \frac{\hat{v}}{\hat{w}}\sin(\theta + \hat{w}\Delta t) \\ \frac{\hat{v}}{\hat{w}}\cos\theta - \frac{\hat{v}}{\hat{w}}\cos(\theta + \hat{w}\Delta t) \\ \hat{w}\Delta t + \hat{\gamma}\Delta t \end{pmatrix} \tag{4.46}$$

Cálculo de $p\left(x_t \mid x_{t-1}, u_t\right)$

Utilizando la ecuación 4.46, conociendo el vector de control $\hat{u}_t$ y la pose x_{t-1}, se puede calcular la pose hipótesis o futura x_t del agente.

Para calcular la probabilidad de transición $p\left(x_t \mid x_{t-1}, u_t\right)$, se debe tener conocimiento de x_{t-1}, x_t y u_t, además de suponer que las velocidades del agente son fijas en un intervalo de tiempo Δt y así suponer que el desplazamiento del agente es sobre trayectorias circulares. Se denotará la coordenada del centro del círculo como $(x^*, y^*)^T$, las cuales están dadas por:

$$\begin{pmatrix} x^* \\ y^* \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} -\lambda\sin\theta \\ \lambda\cos\theta \end{pmatrix} = \begin{pmatrix} \frac{x+x'}{2} + \mu(y - y') \\ \frac{y+y'}{2} + \mu(x' - x) \end{pmatrix} \tag{4.47}$$

para valores desconocidos λ, $\mu \in \mathbb{R}$. La primer igualdad de la ecuación 4.47 es resultado del hecho de que el centro del círculo es ortogonal a la orientación inicial del agente, la cual se deduce de la misma manera que las ecuaciones 4.39 y 4.40; la segunda igualdad es el resultado del hecho de que el centro del círculo está sobre

un rayo que pasa por el punto medio entre $x_{t-1} = (\ x \quad y\ )^T$ y $x_t = (\ x' \quad y'\ )^T$ y que es ortogonal a la línea entre estas coordenadas.

Para llegar a obtener la segunda igualdad, se utiliza la ecuación vectorial de la recta la cual está dada por:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + k \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \tag{4.48}$$

Conociendo los dos puntos x_{t-1} y x_t y el punto medio entre ambos, se puede obtener la ecuación de la recta, debido a que se conoce los componentes del vector v_p que son: $v_{p1} = (x' - x)$ y $v_{p2} = (y' - y)$ y el punto medio se puede denotar como: (x_m, y_m), entonces la ecuación quedaría de la siguiente manera:

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} x_m \\ y_m \end{pmatrix} + k \begin{pmatrix} x' - x \\ y' - y \end{pmatrix} \tag{4.49}$$

se sabe que el centro del círculo está sobre el rayo perpendicular a la recta representada por la ecuación 4.49, entonces el vector v de la recta perpendicular es: $v_q = (-v_{p2}, v_{p1})$. Al sustituir estos valores en la ecuación de la recta y sabiendo que el punto medio está dado por: $x_m = \frac{x+x'}{2}$ y $y_m = \frac{y+y'}{2}$, se obtiene la segunda igualdad de la ecuación 4.47.

Usando ambas igualdades de la ecuación 4.47, se obtiene una solución única:

$$x - \lambda \sin\theta = \frac{x + x'}{2} + \mu(y - y') \ldots\ldots (1)$$

$$y + \lambda \cos\theta = \frac{y + y'}{2} + \mu(x' - x) \ldots\ldots (2)$$

de (1):

$$\lambda = \frac{\frac{-x-x'}{2} - \mu(y - y') + x}{\sin\theta}$$

$$\lambda = \frac{-x - x' - 2\mu(y - y') + 2x}{2\sin\theta}$$

$$\lambda = \frac{x - x' - 2\mu(y - y')}{2\sin\theta}$$

en (2):

$$y + \left(\frac{x - x' - 2\mu(y - y')}{2\sin\theta}\right)\cos\theta = \frac{y + y' + 2\mu(x' - x)}{2}$$

$$2y + \left(\frac{x - x' - 2\mu(y - y')}{\sin\theta}\right)\cos\theta = y + y' + 2\mu(x' - x)$$

$$2y\sin\theta + (x - x')\cos\theta - 2\mu(y - y')\cos\theta = (y + y')\sin\theta + 2\mu(x' - x)\sin\theta$$

$$2y\sin\theta + (x-x')\cos\theta - (y+y')\sin\theta = 2\mu(x'-x)\sin\theta + 2\mu(y-y')\cos\theta$$

$$(2y-y-y')\sin\theta + (x-x')\cos\theta = 2\mu[(x'-x)\sin\theta + (y-y')\cos\theta]$$

$$2\mu = \frac{(y-y')\sin\theta + (x-x')\cos\theta}{(x-x')\sin\theta + (y-y')\cos\theta}$$

$$\mu = \frac{1}{2}\cdot\left[\frac{(y-y')\sin\theta + (x-x')\cos\theta}{(x-x')\sin\theta + (y-y')\cos\theta}\right] \tag{4.50}$$

la ecuación 4.50 es la primer línea del algoritmo 2. En la ecuación 4.51, es donde se calculan las coordenadas del centro del círculo, sustituyendo el valor de μ, teniendo como resultado:

$$\begin{pmatrix} x^* \\ y^* \end{pmatrix} = \begin{pmatrix} \frac{x+x'}{2} + \frac{1}{2}\cdot\left[\frac{(y-y')\sin\theta+(x-x')\cos\theta}{(x-x')\sin\theta+(y-y')\cos\theta}\right](y-y') \\ \frac{y+y'}{2} + \frac{1}{2}\cdot\left[\frac{(y-y')\sin\theta+(x-x')\cos\theta}{(x-x')\sin\theta+(y-y')\cos\theta}\right](x'-x) \end{pmatrix} \tag{4.51}$$

La expresión 4.52 calcula el radio del círculo, que se obtiene con la distancia euclidiana

$$r^* = \sqrt{(x-x^*)^2 + (y-y^*)^2} = \sqrt{(x'-x^*)^2 + (y'-y^*)^2} \tag{4.52}$$

además conociendo las coordenadas del centro del círculo, se puede calcular la diferencia o el cambio de dirección u orientación del agente, como se muestra en la ecuación 4.53 :

$$\Delta\theta = \text{atan2}(y'-y^*, x'-x^*) - \text{atan2}(y-y^*, x-x^*) \tag{4.53}$$

como se asume que el agente se desplaza en trayectorias de segmentos de círculo, la distancia que recorre esta dada por:

$$\Delta dist = r^* \cdot \Delta\theta \tag{4.54}$$

teniendo $\Delta dist$ y $\Delta\theta$, es fácil calcular las velocidades $\hat{v}$ y $\hat{w}$:

$$\hat{u}_t = \begin{pmatrix} \hat{v} \\ \hat{w} \end{pmatrix} = \Delta t^{-1} \begin{pmatrix} \Delta dist \\ \Delta\theta \end{pmatrix} \tag{4.55}$$

La velocidad de rotación $\hat{\gamma}$ necesita de la orientación final del agente θ' en $x'y'$ durante Δt , que puede ser determinada con la ecuación 4.44 como:

$$\hat{\gamma} = \Delta t^{-1}(\theta' - \theta) - \hat{w} \tag{4.56}$$

Para finalizar y poder calcular la probabilidad de transición se debe conocer el *error de movimiento*, que está definido de la siguiente manera:

$$v_{err} = v - \hat{v} \tag{4.57}$$

$$w_{err} = w - \hat{w} \tag{4.58}$$

$$\gamma_{err} = \hat{\gamma} \tag{4.59}$$

con base en estos errores se quiere saber qué probabilidad o qué valor de densidad tienen, para ello se define la ecuación 4.60:

$$\varepsilon_{b^2}(a) = \frac{1}{\sqrt{2\pi b^2}} e^{-\frac{a^2}{2b^2}} \tag{4.60}$$

Así la probabilidad de error se expresa de la siguiente manera:

$$\varepsilon_{\alpha_1 v^2 + \alpha_2 w^2}(v_{err}) \tag{4.61}$$

$$\varepsilon_{\alpha_3 v^2 + \alpha_4 w^2}(w_{err}) \tag{4.62}$$

$$\varepsilon_{\alpha_5 v^2 + \alpha_6 w^2}(\gamma_{err}) \tag{4.63}$$

Al final se multiplica cada uno de los resultados para obtener la probabilidad de transición. Como se muestra en la ecuación 4.64.

$$p\left(x_t \mid x_{t-1}, u_t\right) = \varepsilon_{\alpha_1 v^2 + \alpha_2 w^2}(v_{err}) \cdot \varepsilon_{\alpha_3 v^2 + \alpha_4 w^2}(w_{err}) \cdot \varepsilon_{\alpha_5 v^2 + \alpha_6 w^2}(\gamma_{err}) \tag{4.64}$$

Para verificar la credibilidad del algoritmo 2 (modelo de velocidad de movimiento), únicamente se tienen que observar las ecuaciones 4.50, 4.51, 4.52, 4.53, 4.55, y 4.56 y ver que corresponden con las líneas de la 1 a la 8 del modelo de velocidad de movimiento. La línea 9 implementa el calculo de la probabilidad de transición.

Derivación del algoritmo 5

La derivación del filtro de Kalman parte de la definición de la distribución creencia estimada $\overline{bel}(x_t)$, que se calcula en las líneas 1 y 2. La definición de la creencia estimada (ecuación 4.35) se muestra a continuación:

(4.65)
$$\overline{bel}(x_t) = \int \underbrace{p(x_t \mid x_{t-1}, u_t)}_{\sim N(x_t; A_t x_{t-1} + B_t u_t, R_t)} \underbrace{bel(x_{t-1})}_{\sim N(x_t; \mu_{t-1}, \Sigma_{t-1})} dx_{t-1}$$

Como se puede ver el $\overline{bel}(x_{t-1})$ es representado por una media μ_{t-1} y una covarianza Σ_{t-1}. La probabilidad de transición $p(x_t \mid x_{t-1}, u_t)$, se supone en 2.4 como una distribución normal sobre x_t con media $A_t x_{t-1} + B_t u_t$ y covarianza R_t. Y por propiedades de una gaussiana el resultado de la ecuación 4.65 es de nuevo una gaussiana con media $\overline{\mu}_t$ y covarianza $\overline{\Sigma}_t$.
La ecuación 4.65 se puede reescribir de la siguiente manera como se ve en la ecuación 4.66, considerando las distribuciones normales:

(4.66)
$$\begin{aligned}\overline{bel}(x_t) \\ = \eta \int \quad & \exp\{-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R^{-1}(x_t - A_t x_{t-1} - B_t u_t)\} \\ & \exp\{-\frac{1}{2}(x_{t-1} - \mu_{t-1})^T \Sigma^{-1}(x_{t-1} - \mu_{t-1})\} dx_{t-1}\end{aligned}$$

En la expresión 4.66, la variable η representa los valores constantes resultantes de la expresión 4.65. Dejando explícitamente los exponentes de ambas distribuciones normales. Para comodidad, se expresara la ecuación 4.66 como la ecuación siguiente ,4.67.

(4.67)
$$\overline{bel}(x_t) = \eta \int \exp\{-L_t\} dx_{t-1}$$

con

(4.68)
$$\begin{aligned}L_t = & \frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R^{-1}(x_t - A_t x_{t-1} - B_t u_t) \\ & + \frac{1}{2}(x_{t-1} - \mu_{t-1})^T \Sigma^{-1}(x_{t-1} - \mu_{t-1})\end{aligned}$$

L_t es el exponente de la función de distribución de una gaussiana, pero no se conoce el valor de los parámetros, es decir, $\overline{bel}(x_t) \sim N(\overline{\mu}, \overline{\Sigma})$, sin conocer el valor de $\overline{\mu}$ y $\overline{\Sigma}$.

Para conocer los parámetros de una normal o gaussiana, es necesario saber el exponente de la función de densidad de probabilidad, lo cual se obtiene mediante una expresión denominada forma cuadrática. En la definición de $\overline{bel}(x_t)$ (ecuación 4.67) la variable η expresa la parte constante de la función de densidad, por lo tanto todo lo restante es la base y el exponente. Entonces lo único que impide obtener la forma cuadrática de la expresión es la integral, la cual se eliminará realizando una descomposición de L_t en dos funciones, una que dependa de x_{t-1} y otra que no depende de x_{t-1}, como se muestra en la expresión 4.69.

$$L_t = L_t(x_{t-1}, x_t) + L(x_t) \tag{4.69}$$

Con la descomposición 4.69, la definición de la creencia estimada se puede expresar de la siguiente manera (ecuación 4.70):

$$\begin{aligned} \overline{bel}(x_t) &= \eta \int \exp\{-L_t\} dx_{t-1} \\ &= \eta \int \exp\{-L_t(x_{t-1}, x_t) - L_t(x_t)\} dx_{t-1} \\ &= \eta \exp\{-L_t(x_t)\} \int \exp\{-L_t(x_{t-1}, x_t)\} dx_{t-1} \end{aligned} \tag{4.70}$$

La idea clave es elegir $L_t(x_{t-1}, x_t)$ tal que el valor de la integral no dependa de x_t. Si esto se logra, el valor de la integral será constante, y entonces se obtiene la expresión 4.71:

$$\overline{bel}(x_t) = \eta \exp\{-L_t(x_t)\} \tag{4.71}$$

Para empezar con la descomposición de L_t, se busca una función $L_t(x_{t-1}, x_t)$ cuadrática en x_{t-1}, sin importar que dependa de x_t; por el momento no concierne este punto. Para encontrar esta función, se calcula la primera (ecuación 4.72) y segunda derivada (ecuación 4.73) de L_t, que permite conocer los parámetros.

$$\frac{\partial L_t}{\partial x_{t-1}} = -A_t^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t) + \Sigma_{t-1}^{-1}(x_{t-1} - \mu_{t-1}) \tag{4.72}$$

$$\frac{\partial^2 L_t}{\partial x_{t-1}^2} = A_t^T R_t^{-1} A_t + \Sigma_{t-1}^{-1} =: \Psi_t^{-1} \tag{4.73}$$

La segunda derivada muestra directamente la covarianza inversa, a la cual por conveniencia se asigna a una variable Ψ^{-1}.

Para comprender mejor el por qué se obtiene directamente la covarianza se puede considerar que se tiene una expresión de la siguiente manera:

$$\frac{1}{2}(x_t - media)^T Cov^{-1}(x_t - media)$$

Al obtener la segunda derivada de la expresión anterior con respecto a x_t unicamente nos queda Cov^{-1}.

Para la obtención de la media, se tiene que igualar la primera derivada a cero y resolver la ecuación para x_t. Esto se puede interpretar de dos formas; la primera es que se busca el valor para el cual la expresión de la primera derivada sea cero, es decir, si $x_t = media$, la expresión es igual a cero. La segunda forma de interpretar la obtención de la media, es que al ser una expresión cuadrática de una distribución unimodal tiene un máximo, el cual se obtiene realizando la primera derivada, igualándola a cero y resolviendo la ecuación para la variable x_t. Entonces si $\frac{\partial L_t}{\partial x_{t-1}} = 0$ se tiene la expresión 4.74:

$$A^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t) = \Sigma_{t-1}^{-1}(x_{t-1} - \mu_{t-1}) \tag{4.74}$$

Resolviendo para x_{t-1}:

$$\begin{aligned}
&A_t^T R_t^{-1}(x_t - B_t u_t) - A_t^T R_t^{-1} A_t x_{t-1} = \Sigma_{t-1}^{-1} x_{t-1} - \Sigma_{t-1}^{-1}\mu_{t-1} \\
&A_t^T R_t^{-1} A_t x_{t-1} + \Sigma_{t-1}^{-1} x_{t-1} = A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1}\mu_{t-1} \\
&(A_t^T R_t^{-1} A_t + \Sigma_{t-1}^{-1}) x_{t-1} = A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1}\mu_{t-1} \\
&\Psi_t^{-1} x_{t-1} = A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1}\mu_{t-1} \\
&x_{t-1} = \Psi_t[A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1}\mu_{t-1}]
\end{aligned} \tag{4.75}$$

Conociendo ambos parámetros, media y covarianza, se puede expresar la función $L_t(x_{t-1}, x_t)$ como una función cuadrática definida como se muestra en la expresión 4.76

$$\begin{aligned}
L_t(x_{t-1}, x_t) = &\frac{1}{2}\left(x_{t-1} - \Psi_t[A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1}\mu_{t-1}]\right)^T \Psi^{-1} \\
&\left(x_{t-1} - \Psi_t[A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1}\mu_{t-1}]\right)
\end{aligned} \tag{4.76}$$

Se puede notar que la expresión anterior (4.76), claramente es una expresión cuadrática, la cual satisface la forma del exponente de una función de distribución normal. Como se ve en la expresión 4.77.

$$det(2\pi\Psi)^{-\frac{1}{2}} \exp\{-L_t(x_{t-1}, x_t)\} \tag{4.77}$$

La expresión 4.77 es una función de densidad de probabilidad válida para la variable x_{t-1}, porque claramente la función es de la forma que se muestra en 4.78.

$$det(2\pi\Sigma)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\} \tag{4.78}$$

Como se sabe, la integral de una función de densidad de probabilidad es igual a 1. Por lo tanto, tiene:

$$\int det(2\pi\Psi)^{-\frac{1}{2}} \exp\{-L_t(x_{t-1}, x_t)\}dx_{t-1} = 1 \tag{4.79}$$

Entonces se puede decir que:

$$\int \exp\{-L_t(x_{t-1}, x_t)\}dx_{t-1} = det(2\pi\Psi)^{\frac{1}{2}} \tag{4.80}$$

Al tener un valor constante de la integral, se puede contemplar dentro de la variable normalizadora η. Entonces la definición de creencia estimada que se dio en la ecuación 4.71 si es posible.

$$\begin{aligned} \overline{bel}(x_t) &= \eta \exp\{-L_t(x_t)\} \int \exp\{-L_t(x_{t-1}, x_t)\}dx_{t-1} \\ &= \eta \exp\{-L_t(x_t)\} \end{aligned} \tag{4.81}$$

No se debe olvidar que el normalizador η es diferente en ambas líneas de la expresión 4.81

Lo que falta para encontrar la función $L_t(x_t)$, la cual es la diferencia entre L_t y $L_t(x_{t-1}, x_t)$:

$$\begin{aligned} L_t(x_t) =& L_t - L_t(x_{t-1}, x_t) \\ =& \frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R^{-1}(x_t - A_t x_{t-1} - B_t u_t) \\ &+ \frac{1}{2}(x_{t-1} - \mu_{t-1})^T \Sigma^{-1}(x_{t-1} - \mu_{t-1}) \\ &- \frac{1}{2}\left(x_{t-1} - \Psi_t[A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1}\mu_{t-1}]\right)^T \Psi^{-1} \\ &\left(x_{t-1} - \Psi_t[A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1}\mu_{t-1}]\right) \end{aligned} \tag{4.82}$$

Para hacer la diferencia se vuelve a sustituir $\Psi = (A_t^T R_t^{-1} A_t + \Sigma_{t-1}^{-1})^{-1}$ y se multiplicará los términos para desarrollar la expresión y poder realizar la diferencia:

$$
\begin{aligned}
(4.83) \qquad L_t = & \underline{\underline{\frac{1}{2} x_{t-1}^T A_t^T R_t^{-1} A_t x_{t-1}}} \underline{-x_{t-1}^T A_t^T R_t^{-1}(x_t - B_t u_t)} \\
& + \frac{1}{2}(x_t - B_t u_t)^T R_t^{-1}(x_t - B_t u_t) \\
& + \underline{\underline{\frac{1}{2} x_{t-1}^T \Sigma_{t-1}^{-1} x_{t-1}}} \underline{-x_{t-1}^T \Sigma_{t-1}^{-1} \mu_{t-1}} + \frac{1}{2}\mu_{t-1}^T \Sigma_{t-1}^{-1} \mu_{t-1} \\
& \underline{\underline{-\frac{1}{2} x_{t-1}^T (A_t^T R_t^{-1} A_t + \Sigma_{t-1}^{-1}) x_{t-1}}} \\
& \underline{+x_{t-1}^T [A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1} \mu_{t-1}]} \\
& - \frac{1}{2}[A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1} \mu_{t-1}]^T (A_t^T R_t^{-1} A_t + \Sigma_{t-1}^{-1})^{-1} \\
& \quad [A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1} \mu_{t-1}]
\end{aligned}
$$

Como se puede ver los términos que contienen x_{t-1} se cancelan, dando como resultado la función $L(x_t)$ (ecuación 4.84)

$$
\begin{aligned}
(4.84) \qquad L(x_t) = & \frac{1}{2}(x_t - B_t u_t)^T R_t^{-1}(x_t - B_t u_t) + \frac{1}{2}\mu_{t-1}^T \Sigma_{t-1}^{-1} \mu_{t-1} \\
& - \frac{1}{2}[A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1} \mu_{t-1}]^T (A_t^T R_t^{-1} A_t + \Sigma_{t-1}^{-1})^{-1} \\
& \quad [A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1} \mu_{t-1}]
\end{aligned}
$$

La definición de $\overline{bel}(x_t)$ que se dio en la ecuación 4.65, la cual partía de la multiplicación de dos funciones de densidad de probabilidad de una distribución normal, se simplifico contemplando los valores constantes en una variable η y en una función L_t la suma de sus respectivos exponentes. Como L_t se descompuso en dos funciones $L_t(x_{t-1}, x_t)$ y $L_t(x_t)$, ambas funciones deben de ser exponentes de una función de densidad de probabilidad de una normal. Una vez que se comprobó que $L_t(x_{t-1}, x_t)$ sí tiene forma de una expresión cuadrática, $L_t(x_t)$ también debe de ser una expresión cuadrática. Por lo tanto para obtener los parámetros (media,covarianza) de la función $L_t(x_t)$ se debe de obtener la primera derivada, igualarla a cero y resolver la ecuación para así obtener la media, y realizar la segunda derivada para obtener la inversa de la covarianza.

Realizando la primera derivada, la cual se muestra en la expresión 4.85.

$$
\begin{aligned}
(4.85)\qquad \frac{\partial L_t(x_t)}{\partial x_t} =& R_t^{-1}(x_t - B_t u_t) - R_t^{-1} A_t (A_t^T R_t^{-1} A_t + \Sigma_{t-1}^{-1})^{-1} \\
& [A_t^T R_t^{-1}(x_t - B_t u_t) + \Sigma_{t-1}^{-1}\mu_{t-1}] \\
=& [R_t^{-1} - R_t^{-1} A_t (A_t^T R_t^{-1} A_t + \Sigma_{t-1}^{-1})^{-1} A_t^T R_t^{-1}](x_t - B_t u_t) \\
& - R_t^{-1} A_t (A_t^T R_t^{-1} A_t + \Sigma_{t-1}^{-1})^{-1} \Sigma_{t-1}^{-1}\mu_{t-1}
\end{aligned}
$$

Lema 1. ***Lema de inversión.*** *Para cualesquiera matrices cuadradas R y Q y cualquier matriz P con dimensiones apropiadas, lo siguiente es válido:*

$$(R + PQP^T)^{-1} = R^{-1} - R^{-1}P(Q^{-1} + P^T R^{-1} P)^{-1} P^T R^{-1}$$

asumiendo que todas las matrices anteriores se pueden invertir como se indica.
Prueba. *Se define $\Psi = (Q^{-1} + P^T R^{-1} P)^{-1}$. Probando que:*

$$(R^{-1} - R^{-1}P\Psi P^T R^{-1})(R + PQP^T) = I$$

el lema de inversión se puede demostrar.
Lo único que se necesita es desarrollar la expresión:

$$
\begin{aligned}
I =& \underbrace{R^{-1}R}_{=I} + R^{-1}PQP^T - R^{-1}P\Psi P^T \underbrace{R^{-1}R}_{=I} - R^{-1}P\Psi P^T R^{-1} PQP^T \\
=& I + R^{-1}PQP^T - R^{-1}P\Psi P^T - R^{-1}P\Psi P^T R^{-1} PQP^T \\
=& I + R^{-1}P[QP^T - \Psi P^T - \Psi P^T R^{-1} PQP^T] \\
=& I + R^{-1}P[QP^T - \Psi \underbrace{Q^{-1}Q}_{=I} P^T - \Psi P^T R^{-1} PQP^T] \\
=& I + R^{-1}P[QP^T - \Psi \underbrace{Q^{-1}}_{=\Psi^{-1} - P^T R^{-1} P} QP^T - \Psi P^T R^{-1} PQP^T] \\
=& I + R^{-1}P[QP^T - \Psi(\Psi^{-1} - P^T R^{-1} P)QP^T - \Psi P^T R^{-1} PQP^T] \\
=& I + R^{-1}P[QP^T - \underbrace{\Psi\Psi^{-1}}_{=I} QP^T \underbrace{+\Psi P^T R^{-1} PQP^T - \Psi P^T R^{-1} PQP^T}_{=0}] \\
=& I + R^{-1}P[\underbrace{QP^T - QP^T}_{=0}] = I
\end{aligned}
$$

Utilizando el *lema de inversión* se ve que se simplifica el primer término del resultado de la primera derivada, quedando de la siguiente manera:

$$(4.86)\qquad R_t^{-1} - R_t^{-1} A_t (A_t^T R_t^{-1} A_t + \Sigma_{t-1}^{-1})^{-1} A_t^T R_t^{-1} = (R_t + A_t \Sigma_{t-1} A_t^T)^{-1}$$

Por lo tanto la derivada de $L_t(x_t)$ queda como sigue:

$$\begin{aligned}\frac{\partial L_t(x_t)}{\partial x_t} =&(R_t + A_t\Sigma_{t-1}A_t^T)^{-1}(x_t - B_t u_t) \\ &- R_t^{-1}A_t(A_t^T R_t^{-1}A_t + \Sigma_{t-1}^{-1})^{-1}\Sigma_{t-1}^{-1}\mu_{t-1}\end{aligned} \tag{4.87}$$

Una vez obtenida la expresión mínima de la primera derivada, sólo queda igualar a cero la expresión y resolver la ecuación para x_t.
Si se iguala la derivada a cero, se tiene la expresión 4.88:

$$(R_t + A_t\Sigma_{t-1}A_t^T)^{-1}(x_t - B_t u_t) = R_t^{-1}A_t(A_t^T R_t^{-1}A_t + \Sigma_{t-1}^{-1})^{-1}\Sigma_{t-1}^{-1}\mu_{t-1} \tag{4.88}$$

Resolviendo para la variable x_t:

$$\begin{aligned} x_t - B_t u_t &= (R_t + A_t\Sigma_{t-1}A_t^T)R_t^{-1}A_t(A_t^T R_t^{-1}A_t + \Sigma_{t-1}^{-1})^{-1}\Sigma_{t-1}^{-1}\mu_{t-1} \\ x_t - B_t u_t &= R_t R_t^{-1}A_t + A_t\Sigma_{t-1}A_t^T R_t^{-1}A_t(A_t R_t A_t^T + \Sigma_{t-1})\Sigma_{t-1}^{-1}\mu_{t-1} \\ x_t - B_t u_t &= A_t + A_t\Sigma_{t-1}A_t^T R_t^{-1}A_t(A_t R_t A_t^T\Sigma_{t-1}^{-1} + \Sigma_{t-1}\Sigma_{t-1}^{-1})\mu_{t-1} \\ x_t - B_t u_t &= A_t(I + \Sigma_{t-1}A_t^T R_t^{-1}A_t)(A_t R_t A_t^T\Sigma_{t-1}^{-1} + I)\mu_{t-1} \\ x_t &= B_t u_t + A_t\mu_{t-1}\end{aligned} \tag{4.89}$$

Se observa que la media de la creencia estimada $\overline{bel}(x_t)$ se obtiene como se muestra en la primera línea del algoritmo del filtro de Kalman (algoritmo 5).
La línea 2 es la covarianza, la cual se obtiene a partir de la segunda derivada e invirtiendo el resultado, como se muestra en la expresión 4.90.

$$\frac{\partial^2 L_t(x_t)}{\partial x_t^2} = (A_t\Sigma_{t-1}A_t^T + R_t)^{-1} \tag{4.90}$$

Al aplicar la inversa del resultado se obtiene la covarianza de la creencia estimada $\overline{bel}(x_t)$ que se encuentra en la línea 2 del algoritmo del filtro de Kalman (algoritmo 5) .

Para continuar con la derivación se retomará las definiciones de distribuciones creencia (sección 1.2.2). En este caso ya se tiene definido la creencia estimada $\overline{bel}(x_t)$, la cual se representa por sus dos parámetros $\overline{\mu}$ y $\overline{\Sigma}$. Ahora se ocupará la definición de creencia $bel(x_t)$, que se presenta en la expresión 4.91.

$$bel(x_t) = \eta \underbrace{p(z_t \mid x_t)}_{\sim N(z_t; C_t x_t, Q_t)} \underbrace{\overline{bel}(x_t)}_{\sim N(x_t; \overline{\mu}_t, \overline{\Sigma}_t)} \tag{4.91}$$

Como se puede ver, la definición consta de una multiplicación entre dos funciones de densidad de una distribución normal. Por lo tanto se ocupará la misma utilizada en la obtención de los parámetros $\overline{\mu}$ y $\overline{\Sigma}$ de la creencia estimada $\overline{bel}(x_t)$, obtener la primera y segunda derivada de la función.

Primero se expresa la definición de $bel(x_t)$ de una manera más compacta, como se muestra en la expresión 4.92.

$$bel(x_t) = \eta \exp\{-J_t\} \tag{4.92}$$

con

$$J_t = \frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t) + \frac{1}{2}(x_t - \overline{\mu}_t)^T \overline{\Sigma}_t^{-1}(x_t - \overline{\mu}_t) \tag{4.93}$$

Como se observar J_t es la suma de los exponentes de funciones de distribución normal y la variable η guarda todos los valores constantes de la misma.

También J_t es cuadrática en x_t y es el exponente de una función de densidad de una distribución normal. Por lo tanto se pueden calcular los parámetros μ (media) y Σ (covarianza) a partir de su primera y segunda derivada, respectivamente, como se muestran en las expresiones 4.94 y 4.95.

$$\frac{\partial J_t}{\partial x_t} = -C_t^T Q_t^{-1}(z_t - C_t x_t) + \overline{\Sigma}_t^{-1}(x_t - \overline{\mu}_t) \tag{4.94}$$

$$\frac{\partial J_t}{\partial x_t} = C_t^T Q_t^{-1} C_t + \overline{\Sigma}_t^{-1} \tag{4.95}$$

Como vio anteriormente (expresión 4.73) la segunda derivada es directamente la inversa de la covarianza, por lo tanto la covarianza de $bel(x_t)$ se encuentra expresa en 4.96.

$$\Sigma_t = (C_t^T Q_t^{-1} C_t + \overline{\Sigma})^{-1} \tag{4.96}$$

Para obtener la media se tiene que igualar a cero la primer derivada y resolver la ecuación para x_t. Igualando a cero se tiene la siguiente expresión 4.97

$$C_t^T Q_t^{-1}(z_t - C_t x_t) = \overline{\Sigma}_t^{-1}(x_t - \overline{\mu}_t) \tag{4.97}$$

Es posible transformar la expresión 4.97 del lado izquierdo de la igualdad como se muestra en 4.98.

$$
\begin{aligned}
(4.98) \quad C_t^T Q_t^{-1}(z_t - C_t x_t) &= C_t^T Q_t^{-1}(z_t - C_t x_t + \underbrace{C_t\overline{\mu}_t - C_t\overline{\mu}_t}_{=0}) \\
&= C_t^T Q_t^{-1}(z_t - C_t\overline{\mu}_t) - C_t^T Q_t^{-1} C_t x_t + C_t^T Q_t^{-1} C_t \overline{\mu}_t \\
&= C_t^T Q_t^{-1}(z_t - C_t\overline{\mu}_t) - C_t^T Q_t^{-1} C_t (x_t + \overline{\mu}_t)
\end{aligned}
$$

por lo tanto, se puede escribir la siguiente expresión 4.99:

(4.99)

$$
\begin{aligned}
C_t^T Q_t^{-1}(z_t - C_t\overline{\mu}_t) - C_t^T Q_t^{-1} C_t (x_t + \overline{\mu}_t) &= \overline{\Sigma}_t^{-1}(x_t - \overline{\mu}_t) \\
C_t^T Q_t^{-1}(z_t - C_t\overline{\mu}_t) &= \overline{\Sigma}_t^{-1}(x_t - \overline{\mu}_t) + C_t^T Q_t^{-1} C_t (x_t + \overline{\mu}_t) \\
C_t^T Q_t^{-1}(z_t - C_t\overline{\mu}_t) &= \underbrace{(\overline{\Sigma}_t^{-1} + C_t^T Q_t^{-1} C_t)}_{=\Sigma_t^{-1}, \text{ec } 3.38}(x_t + \overline{\mu}_t) \\
C_t^T Q_t^{-1}(z_t - C_t\overline{\mu}_t) &= \Sigma_t^{-1}(x_t - \overline{\mu}_t) \\
\Sigma_t C_t^T Q_t^{-1}(z_t - C_t\overline{\mu}_t) &= (x_t - \overline{\mu}_t)
\end{aligned}
$$

Se define la ganancia de Kalman, que se explicará más adelante en la expresión 4.101, como:

$$
(4.100) \qquad K_t = \Sigma_t C_t^T Q_t^{-1}
$$

Una vez definida la ganancia de Kalman, se sustituye su valor en la ecuación 4.103, obteniendo el siguiente resultado que se encuentra en 4.101.

$$
\begin{aligned}
(4.101) \qquad (x_t - \overline{\mu}_t) &= K_t(z_t - C_t\overline{\mu}_t) \\
x_t &= K_t(z_t - C_t\overline{\mu}_t) + \overline{\mu}_t
\end{aligned}
$$

Con esta ecuación se demuestra la línea 4 del algoritmo del filtro de Kalman, la cual nos muestra la media de la creencia $bel(x_t)$.

Ahora se va a demostrar la línea 3 del algoritmo, porque aunque ya está definida la ganancia de Kalman en la ecuación (4.100), se puede ver que no es la misma expresión que se tiene en el algoritmo 5, además de que existe una contradicción en esa definición. Se puede ver que K_t esta en función de Σ_t lo cual no es posible, porque Σ_t se define hasta la línea 6 del algoritmo 5. Por esta razón se va a expresar la ganancia de Kalman en términos de covarianzas que no sean Σ_t.

$$
\begin{aligned}
K_t &= \Sigma_t C_t^T Q_t^{-1} \\
&= \Sigma_t C_t^T Q_t^{-1} \underbrace{(C_t \overline{\Sigma}_t C_t^T + Q_t)(C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1}}_{=I} \\
&= \Sigma_t [C_t^T Q_t^{-1} C_t \overline{\Sigma}_t C_t^T + C_t^T \underbrace{Q_t^{-1} Q_t}_{=I}](C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1} \\
&= \Sigma_t [(C_t^T Q_t^{-1} C_t)\overline{\Sigma}_t C_t^T + \underbrace{\overline{\Sigma}_t^{-1}\overline{\Sigma}_t}_{=I} C_t^T](C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1} \\
&= \Sigma_t [\underbrace{(C_t^T Q_t^{-1} C_t + \overline{\Sigma}_t^{-1})}_{\Sigma_t^{-1}} \overline{\Sigma}_t C_t^T](C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1} \\
&= \Sigma_t [(\Sigma_t^{-1})\overline{\Sigma}_t C_t^T](C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1} \\
&= \underbrace{\Sigma_t \Sigma_t^{-1}}_{=I} \overline{\Sigma}_t C_t^T (C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1} \\
&= \overline{\Sigma}_t C_t^T (C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1}
\end{aligned} \tag{4.102}
$$

Con esta expresión se verifica que la línea 3 del algoritmo del filtro de Kalman es correcta.

Por último falta obtener la línea 5 del algoritmo, la cual nos expresa la covarianza del $bel(x_t)$. Para ello se utilizará nuevamente el lema de inversión para desarrollar la inversa de la covarianza y expresarla en términos de la ganancia de Kalman.

Se puede utilizar el lema de la inversión sobre el resultado de la segunda derivada de J_t, que sería lo siguiente:

$$
(\overline{\Sigma}_t^{-1} + C_t^T Q_t^{-1} C_t)^{-1} = \overline{\Sigma}_t - \overline{\Sigma}_t C_t^T (Q_t + C_t \overline{\Sigma}_t C_t^T)^{-1} C_t \overline{\Sigma}_t \tag{4.103}
$$

Usando esta igualdad se expresa la covarianza de la siguiente manera:

$$
\begin{aligned}
\Sigma_t &= (\overline{\Sigma}_t^{-1} + C_t^T Q_t^{-1} C_t)^{-1} \\
&= \overline{\Sigma}_t - \overline{\Sigma}_t C_t^T (Q_t + C_t \overline{\Sigma}_t C_t^T)^{-1} C_t \overline{\Sigma}_t \\
&= [I - \underbrace{\overline{\Sigma}_t C_t^T (Q_t + C_t \overline{\Sigma}_t C_t^T)^{-1}}_{=K_t} C_t]\overline{\Sigma}_t \\
&= (I - K_t C_t)\overline{\Sigma}_t
\end{aligned} \tag{4.104}
$$

Y con esta última expresión se verifica la línea 5, con la que se obtiene la covarianza. Por último el algoritmo del filtro de Kalman regresa los valores de los parámetros μ y Σ.

Derivación del algoritmo 6

La derivación del EKF es semejante a la del algoritmo del filtro de Kalman, por lo cuál solo se resumirá a continuación.

La etapa de predicción se calcula de la siguiente manera:

$$\overline{bel}(x_t) = \int \underbrace{p(x_t \mid x_{t-1}, u_t)}_{\sim N(x_t; g(u_t,\mu_{t-1})+G_t(x_{t-1}-\mu_{t-1}),R_t)} \underbrace{bel(x_{t-1})}_{\sim N(x_{t-1};\mu_{t-1},\Sigma_{t-1})} \, dx_{t-1} \tag{4.105}$$

Se ve, al igual que en KF, se parte de la definición de la creencia estimada $\overline{bel}(x_t)$. Lo siguiente, al igual que en la derivación del KF, es reducir la expresión y obtener la primera y segunda derivada para la determinación de sus parámetros.

Se denota a L_t como la suma de los exponentes de ambas distribuciones normales:

$$\begin{aligned} L_t =& \frac{1}{2}(x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1}))^T \\ & R_t^{-1}(x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})) \\ & + \frac{1}{2}(x_{t-1} - \mu_{t-1})^T \Sigma_{t-1}^{-1}(x_{t-1} - \mu_{t-1}) \end{aligned} \tag{4.106}$$

Se descompone a L_t en dos funciones $L_t(x_{t-1}, x_t)$ y $L_t(x_t)$, para realizar el mismo procedimiento en la derivación del algoritmo KF.

(4.107)

$$\begin{aligned} & L_t(x_{t-1}, x_t) \\ & \quad = \frac{1}{2}(x_{t-1} - \Phi_t[G_t^T R_t^{-1}(x_t - g(u_t, \mu_{t-1}) + G_t\mu_{t-1}) + \Sigma_{t-1}^{-1}\mu_{t-1}])^T \Phi_t^{-1} \\ & \quad (x_{t-1} - \Phi_t[G_t^T R_t^{-1}(x_t - g(u_t, \mu_{t-1}) + G_t\mu_{t-1}) + \Sigma_{t-1}^{-1}\mu_{t-1}]) \end{aligned}$$

asignando a la variable Φ el valor de: $(G_t^T R_t^{-1} G_t + \Sigma_{t-1}^{-1})$.

Conociendo la función $L_t(x_{t-1}, x_t)$, fácilmente se encuentra el valor de la función L_t, que se da por la siguiente expresión:

$$\begin{aligned} L_t(x_t) =& \frac{1}{2}(x_t - g(u_t, \mu_{t-1}) + G_t\mu_{t-1})^T R_t^{-1}(x_t - g(u_t, \mu_{t-1}) + G_t\mu_{t-1}) \\ & + \frac{1}{2}(x_{t-1} - \mu_{t-1})^T \Sigma_{t-1}^{-1}(x_{t-1} - \mu_{t-1}) \\ & - \frac{1}{2}[G_t^T R_t^{-1}(x_t - g(u_t, \mu_{t-1}) + G_t\mu_{t-1}) + \Sigma_{t-1}^{-1}\mu_{t-1}]^T \\ & \Phi_t[G_t^T R_t^{-1}(x_t - g(u_t, \mu_{t-1}) + G_t\mu_{t-1}) + \Sigma_{t-1}^{-1}\mu_{t-1}] \end{aligned} \tag{4.108}$$

Obteniendo la primer derivada e igualándola a cero, se ve que el resultado es igual a la línea 1 del algoritmo EKF, cuya línea nos dice la media de la creencia estimada $\overline{bel}(x_t)$, $\overline{\mu} = g(u_t, \mu_{t-1})$, y con la segunda derivada, se obtiene directamente la inversa de la covarianza, la cual se calcula en la línea 2 del algoritmo, que es la covarianza de la creencia estimada, $\overline{\Sigma} = G_t \Sigma_{t-1} G_t^T$.

Para verificar la parte de corrección del algoritmo, líneas 4 y 5, se vuelve a utilizar la definición de la creencia $bel(x_t)$.

$$bel(x_t) = \eta \underbrace{p(z_t \mid x_t)}_{\sim N(z_t; h(\overline{\mu}_t) + H_t(x_t - \overline{\mu}_t), Q_t)} \underbrace{\overline{bel}(x_t)}_{\sim N(x_t; \overline{\mu}_t, \overline{\Sigma}_t)} \tag{4.109}$$

Siguiendo la misma idea de la derivación del KF, se reduce la expresión de la definición de la creencia para derivar la suma de los exponentes y obtener los parámetros. Se asigna a la variable J_t dicha suma:

$$J_t = (z_t - h(\overline{\mu}_t) - H_t(x_t - \overline{\mu}_t))^T Q_t^{-1} (z_t - h(\overline{\mu}_t) - H_t(x_t - \overline{\mu}_t)) \tag{4.110}$$

$$+ \frac{1}{2}(x_t - \overline{\mu}_t)^T \overline{\Sigma}_t^{-1} (x_t - \overline{\mu}_t) \tag{4.111}$$

El resultado que se obtiene para los valores de la media y la covarianza de la expresión anterior son:

$$\mu = \overline{\mu}_t + K_t(z_t - h(\overline{\mu}_t)) \tag{4.112}$$

$$\Sigma = (I - K_t H_t)\overline{\Sigma}_t \tag{4.113}$$

con una ganancia de Kalman de:

$$K_t = \Sigma_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1} \tag{4.114}$$

con lo cual el resto del algoritmo EKF queda comprobado.

Derivación del algoritmo 8

La derivación del algoritmo se dividirá en dos partes, la etapa de **predicción** y la de **corrección**. Al estar basado en el EKF, la derivación es similar a la de este algoritmo.

Predicción

En la etapa de predicción que se lleva a cabo de la línea 3 a la 7, se utiliza el modelo de movimiento de velocidades de rotación y traslación, $u_t = \begin{pmatrix} v_t & w_t \end{pmatrix}^T$, para definir la función *no lineal* que representa el estado x_t:

$$\underbrace{\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix}}_{x_t} = \underbrace{\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{w_t}\sin\theta + \frac{v_t}{w_t}\sin(\theta + w_t\Delta t) \\ \frac{v_t}{w_t}\cos\theta - \frac{v_t}{w_t}\cos(\theta + w_t\Delta t) \\ w_t\Delta t \end{pmatrix}}_{g(u_t, x_{t-1})} + \mathcal{N}(0, R_t) \tag{4.115}$$

o bien:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}_t}{\hat{w}_t}\sin\theta + \frac{\hat{v}_t}{\hat{w}_t}\sin(\theta + \hat{w}_t\Delta t) \\ \frac{\hat{v}_t}{\hat{w}_t}\cos\theta - \frac{\hat{v}_t}{\hat{w}_t}\cos(\theta + \hat{w}_t\Delta t) \\ \hat{w}_t\Delta t \end{pmatrix} \tag{4.116}$$

Donde $x_{t-1} = \begin{pmatrix} x & y & \theta \end{pmatrix}^T$ y $x_t = \begin{pmatrix} x' & y' & \theta' \end{pmatrix}^T$ son los vectores de estado en el tiempo $t-1$ y t, respectivamente. La transición que se realiza de la ecuación (4.115) a (4.116) es debido al movimiento real o verdadero que se lleva a cabo por las velocidades de traslación, $\hat{v}_t$, y rotación, $\hat{w}_t$. Como se explicó en el capítulo de modelos de movimiento, el vector de control de movimiento lleva agregado ruido, que en este caso se expresa como gaussiano:

$$\begin{pmatrix} \hat{v}_t \\ \hat{w}_t \end{pmatrix} = \begin{pmatrix} v_t \\ w_t \end{pmatrix} + \begin{pmatrix} \varepsilon_{\alpha_1 v_t^2 + \alpha_2 w_t^2} \\ \varepsilon_{\alpha_3 v_t^2 + \alpha_4 w_t^2} \end{pmatrix} = \begin{pmatrix} v_t \\ w_t \end{pmatrix} + \mathcal{N}(0, M_t) \tag{4.117}$$

Como se sabe para linealizar la función $g(u_t, x_{t-1})$ se aplica expansión de Taylor, por lo tanto:

$$g(u_t, x_{t-1}) \approx g(u_t, x_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \tag{4.118}$$

donde G_t es la derivada de la función $g(u_t, x_{t-1})$, que en este caso es el Jacobiano de la función g con respecto a x_{t-1}

$$G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} = \begin{pmatrix} \frac{\partial x'}{\partial \mu_{t-1,x}} & \frac{\partial x'}{\partial \mu_{t-1,y}} & \frac{\partial x'}{\partial \mu_{t-1,\theta}} \\ \frac{\partial y'}{\partial \mu_{t-1,x}} & \frac{\partial y'}{\partial \mu_{t-1,y}} & \frac{\partial y'}{\partial \mu_{t-1,\theta}} \\ \frac{\partial \theta'}{\partial \mu_{t-1,x}} & \frac{\partial \theta'}{\partial \mu_{t-1,y}} & \frac{\partial \theta'}{\partial \mu_{t-1,\theta}} \end{pmatrix} \tag{4.119}$$

Calculando estas derivadas de la ecuación (4.115) se obtiene la siguiente matriz:

$$G_t = \begin{pmatrix} 1 & 0 & \frac{v_t}{w_t}(-\cos\mu_{t-1,\theta} + \cos(\mu_{t-1,\theta} + w_t\Delta t)) \\ 0 & 1 & \frac{v_t}{w_t}(-\sin\mu_{t-1,\theta} + \sin(\mu_{t-1,\theta} + w_t\Delta t)) \\ 0 & 0 & 1 \end{pmatrix} \tag{4.120}$$

Para derivar la covarianza del ruido del movimiento , $\mathcal{N}(0, R_t)$, es necesario determinar la matriz de covarianza M_t del ruido en el *espacio de los controles*. La cual se determina directamente del modelo de movimiento de la ecuación (4.117):

$$M_t = \begin{pmatrix} \alpha_1 v_t^2 + \alpha_2 w_t^2 & 0 \\ 0 & \alpha_3 v_t^2 + \alpha_4 w_t^2 \end{pmatrix} \tag{4.121}$$

Únicamente es la matriz de covarianza.

Para agregar el ruido de los controles al modelo de movimiento, es necesario realizar una función (mapeo) $\mathbb{R}^2 \to \mathbb{R}^3$. La transformación del espacio de control $\mathbb{R}^2$ al espacio de estados es realizando una aproximación lineal. El Jacobiano necesitado para esta aproximación se denota como V_t, que se obtiene derivando la función g con respecto a los parámetros del movimiento. Esto es:

$$V_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial u_t} = \begin{pmatrix} \frac{\partial x'}{\partial v_t} & \frac{\partial x'}{\partial w_t} \\ \frac{\partial y'}{\partial v_t} & \frac{\partial y'}{\partial w_t} \\ \frac{\partial \theta'}{\partial v_t} & \frac{\partial \theta'}{\partial w_t} \end{pmatrix} \tag{4.122}$$

Al aplicar estas derivadas el resultado nos queda

$$V_t = \begin{pmatrix} \frac{-\sin\theta + \sin(\theta + w_t\Delta t)}{w_t} & \frac{v_t(\sin\theta - \sin(\theta + w_t\Delta t))}{w_t^2} + \frac{v_t\cos(\theta + w_t\Delta t)\Delta t}{w_t} \\ \frac{\cos\theta - \sin(\theta + w_t\Delta t)}{w_t} & -\frac{v_t(\cos\theta - \cos(\theta + w_t\Delta t))}{w_t^2} + \frac{v_t\sin(\theta + w_t\Delta t)\Delta t}{w_t} \\ 0 & \Delta t \end{pmatrix} \tag{4.123}$$

La multiplicación $V_t M_t V_t^T$ proporciona una función o aplicación (mapeo) aproximada entre el ruido de movimiento en el espacio de control y el ruido de movimiento en el espacio de estado.

Con esto, la etapa de predicción concluye y la obtención de los parámetros $\overline{\mu}_t$ y $\overline{\Sigma}$, en las líneas 6 y 7, queda demostrada.

Corrección

Para la etapa de corrección, que se realiza entre las líneas 8 y 20, el algoritmo de localización EKF hace uso de un modelo de percepción linealizado con ruido agregado, que en este caso será ruido gaussiano, esto es:

$$\underbrace{\begin{pmatrix} r_t^i \\ \phi_t^i \\ s_t^i \end{pmatrix}}_{z_t^i} = \underbrace{\begin{pmatrix} \sqrt{(m_{j,x}-x)^2+(m_{j,y}-y)^2)} \\ atan2(m_{j,y}-y, m_{j,x}-x)-\theta \\ m_{j,s} \end{pmatrix}}_{h(x_t,j,m)} + \mathcal{N}(0,Q_t) \tag{4.124}$$

donde $(\; m_{j,x} \quad m_{j,y} \;)^T$ son las coordenadas de la i-ésima referencia detectada en el tiempo t, recordando que el modelo de percepción utilizado supone el conocimiento del mapa y establece correspondencia a través de la variable c_t. Siendo $j = c_t^i$ el identificador de la referencia que corresponde al i-ésimo componente en el vector de percepción. Y la variable $m_{j,s}$ es la "característica" o firma (*signature*).

Para linealizar este modelo, se realiza la misma aproximación mediante la expansión de Taylor. Lo que sería de la siguiente manera:

$$h(x_t, j, m) \approx h(\overline{\mu}_t, j, m) + H_t^i(x_t - \overline{\mu}_t) \tag{4.125}$$

donde la variable H_t^i es el Jacobiano de h con respecto a la localización del robot, utilizando la media predicha $\overline{\mu}_t$. Este Jacobiano es:

$$H_t^i = \frac{\partial h(\overline{\mu}_t, j, m)}{\partial x_t} = \begin{pmatrix} \frac{\partial r_t^i}{\partial \overline{\mu}_{t,x}} & \frac{\partial r_t^i}{\partial \overline{\mu}_{t,y}} & \frac{\partial r_t^i}{\partial \overline{\mu}_{t,\theta}} \\ \frac{\partial \phi_t^i}{\partial \overline{\mu}_{t,x}} & \frac{\partial \phi_t^i}{\partial \overline{\mu}_{t,y}} & \frac{\partial \phi_t^i}{\partial \overline{\mu}_{t,\theta}} \\ \frac{\partial s_t^i}{\partial \overline{\mu}_{t,x}} & \frac{\partial s_t^i}{\partial \overline{\mu}_{t,y}} & \frac{\partial s_t^i}{\partial \overline{\mu}_{t,\theta}} \end{pmatrix} \tag{4.126}$$

Realizando estas derivadas, el resultado queda de la siguiente manera:

$$H_t^i = \begin{pmatrix} -\frac{m_{j,x}-\overline{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y}-\overline{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y}-\overline{\mu}_{t,y}}{q} & -\frac{m_{j,x}-\overline{\mu}_{t,x}}{q} & -1 \\ 0 & 0 & 0 \end{pmatrix} \tag{4.127}$$

donde $q = (m_{j,x} - \overline{\mu}_{t,x})^2 + (m_{j,y} - \overline{\mu}_{t,y})^2$.

La matriz Q_t representa el ruido de percepción agregado en la ecuación (4.124). Esta matriz es la matriz de covarianza:

$$Q_t = \begin{pmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\phi^2 & 0 \\ 0 & 0 & \sigma_s^2 \end{pmatrix} \tag{4.128}$$

Una parte en la cual el algoritmo de localización EKF cambia con respecto al algoritmo del filtro extendido de Kalman es que el proceso de localización recibe múltiples percepciones en el tiempo, en lugar de solo una. El algoritmo de localización EKF depende de la suposición de independencia condicional, lo cual dice que todas las características percibidas son independientes. Dada la pose x_t, la referencia identificada con c_t y el mapa m:

$$p(z_t \mid x_t, c_t, m) = \prod_i p(z_t^i \mid x_t, c_t^i, m) \tag{4.129}$$

Esta suposición permite incrementar progresivamente la información obtenida de múltiples características observadas en el instante t. Esto se ve reflejado en el ciclo de las líneas 9 a 18 del algoritmo. Intuitivamente corresponde a múltiples actualizaciones de percepciones con cero movimiento entre cada una de ellas.

Funciones primitivas

En esta sección se muestran todas las funciones que realizaban subrutinas utilizadas a lo largo de este trabajo dentro de los algoritmos.

- **sign(v)**. Devuelve el signo del valor del argumento v. Y se define como se muestra en la ecuación 4.130.

$$sing(v) = \begin{cases} -1 & \text{si } v < 0 \\ 1 & \text{si } v \geq 0 \end{cases} \tag{4.130}$$

- **rad(ϕ)**. Convierte el valor de su argumento ϕ a un valor equivalente en radienes. Y se define como se muestra en la ecuación 4.131.

$$rad(\phi) = \frac{\phi\pi}{180} \tag{4.131}$$

- **grad(ϕ)**. Convierte el valor de su argumento ϕ a un valor equivalente en grados. Y se define como se muestra en la ecuación 4.132.

$$grad(\phi) = \frac{180\phi}{\pi} \tag{4.132}$$

- **size(v)**. Devuelve el tamaño o cardinalidad del argumento v. Esta función esta definida dentro del lenguaje de programación para calcular la cantidad de elementos que contiene un objeto. En la mayoria de los lenguajes de programción se define como *length(A)*.
- **fromPolar(r,ϕ)**. Convierte una coordenada polar (r, ϕ) a coordenada cartesiana (x, y). Y se define como se muestra en la ecuación 4.133.

$$\begin{aligned} x &= r\cos\phi \\ y &= r\sin\phi \end{aligned} \tag{4.133}$$

- **Pr(ϕ_1,ϕ_2)**. Da como resultado el promedio o ángulo medio entre los dos ángulos recibidos. Y se define como se muestra en la ecuación 4.134.

$$Pr(\phi_1, \phi_2) = norm\left(atan2\left(\frac{\sin\phi_1 + \sin\phi_2}{2}, \frac{\cos\phi_1 + \cos\phi_2}{2}\right)\right) \tag{4.134}$$

- **norm(ϕ)**. Convierte el valor de su argumento ϕ en un valor equivalente en un intervalo entre $[-180^o, 180^o]$. Y se define como se muestra en el algoritmo 17.

Algoritmo 17 norm(ϕ)

1: $r1 = \frac{\phi}{180}$
2: $r1 = \lfloor r1 \rfloor$
3: $r2 = \text{res}\ (r1, 2)$
4: **if** $r2 = 0$ **then**
5: $\quad \phi^* = \text{res}\ (\phi, 180)$
6: **else**
7: $\quad$ **if** $\phi > 0$ **then**
8: $\quad\quad \phi^* = \text{res}\ (\phi, 180) - 180$
9: $\quad$ **else**
10: $\quad\quad \phi^* = \text{res}\ (\phi, 180) + 180$
11: $\quad$ **end if**
12: **end if**
13: **return** ϕ^*

En el algoritmo 17, se utiliza la función **res(a,b)**, la cual se encarga de calcular el residuo de la división de a entre b.

Bibliografía

[1] Baltes, J., Lagoudakis, M. G., Naruse, T., and Shiry, S. *RoboCup 2009: Robot Soccer World Cup XIII*, vol. 5949. Springer Science & Business Media, 2010.

[2] Burkhard, H.-D., Hannebauer, M., and Wendler, J. Belief-desire-intention deliberation in artificial soccer. *AI Magazine 19* (1998), 87.

[3] Chen, M., Dorer, K., Foroughi, E., Heintz, F., Huang, Z., Kapetanakis, S., Kostiadis, K., Kummeneje, J., Murray, J., Noda, I., Obst, O., Riley, Pat andvSteffens, T., Wang, Y., and Yin, X. User manual robocup soccer server for soccer server. http://wwfc.cs.virginia.edu/documentation/manual.pdf, February 2003.

[4] de Boer, R., and Kok, J. The incremental development of a synthetic multi-agent system: The uva trilearn 2001 robotic soccer simulation team. Master's thesis, University of Amsterdam, 2002.

[5] Doucet, A., Godsill, S., and Andrieu, C. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing 10*, 3 (2000), 197–208.

[6] Doucet, A., Smith, A., de Freitas, N., and Gordon, N. *Sequential Monte Carlo Methods in Practice*. Information Science and Statistics. Springer, 2010.

[7] Elfes, A. Using occupancy grids for mobile robot perception and navigation. *Computer 22* (1989), 46–57.

[8] Evensen, G. The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean dynamics 53* (2003), 343–367.

[9] Grimmett, G., and Stirzaker, D. *Probability and Random Processes*. Probability and Random Processes. OUP Oxford, 2001.

[10] Haykin, S. S., Haykin, S. S., and Haykin, S. S. *Kalman filtering and neural networks*. Wiley Online Library, 2001.

[11] Laumond, J. *Robot motion planning and control*. Lecture notes in control and information sciences. Springer, 1998.

[12] Lee, J. H., and Ricker, N. L. Extended kalman filter based nonlinear model predictive control. *Industrial & Engineering Chemistry Research 33*, 6 (1994), 1530–1541.

[13] Matsubara, H., Weitzenfeld, A., and Zhou, C. *Robocup 2008: robot soccer world cup XII*. Springer, 2009.

[14] Meyer, P. *Probabilidad y Aplicaciones Estadísticas*. Iberoamericana, 1999.

[15] Montemerlo, M., Thrun, S., and Siciliano, B. *FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics*, vol. 27. Springer Science & Business Media, 2007.

[16] Motwani, R., and Raghavan, P. *Randomized Algorithms*. Cambridge International Series on Parallel Computation. Cambridge University Press, 1995.

[17] Noda, I. Teamwork in multi-agent system. In *SCIS & ISIS* (2006), vol. 2006, pp. 75–79.

[18] Pliego, F., and PEREZ, L. *Fundamentos de probabilidad*. Ediciones Paraninfo. S.A., 2006.

[19] Roumeliotis, S. I., and Bekey, G. A. Collective localization: A distributed kalman filter approach to localization of groups of mobile robots. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on* (2000), vol. 3, IEEE, pp. 2958–2965.

[20] Rubin, D. B., et al. Using the sir algorithm to simulate posterior distributions. *Bayesian statistics 3*, 1 (1988), 395–402.

[21] Sharir, M., and Agarwal, P. K. *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press, New York, NY, USA, 1996.

[22] Siegwart, R., and Nourbakhsh, I. *Introduction to Autonomous Mobile Robots*. A Bradford book. Bradford Book, 2004.

[23] Stone, P. Intelligent autonomous robotics: A robot soccer case study. *Synthesis Lectures on Artificial Intelligence and Machine Learning 1* (2007), 1–155.

[24] Thrun, S. *Robust Monte Carlo Localization for Mobile Robots*. Research paper. School of Computer Science, Carnegie Mellon University, 2000.

[25] Thrun, S., Burgard, W., Fox, D., et al. *Probabilistic Robotics*, vol. 1. MIT Press Cambridge, 2006.

[26] Visser, U., Ribeiro, F., Ohashi, T., and Dellaert, F. *Robocup 2007: Robot soccer world cup xi*, vol. 5001. Springer Science & Business Media, 2008.

[27] Vlassis, N. A concise introduction to multiagent systems and distributed artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning 1* (2007), 1–71.