
Universidad de la República

Facultad de Ingeniería

Tesis de Grado

# Estudio del estado del arte del SLAM e implementación de una plataforma flexible

*Autor:*
Martin Llofriu

*Autor:*
Federico Andrade

*Tutor:*
MSc Ing. Gonzalo Tejera

*Tutor:*
Ing. Facundo Benavides

# Resumen

La creciente demanda de soluciones robóticas ha estimulado el desarrollo de robots capaces de desempeñarse de forma autónoma en terrenos diversos y, en ocasiones, desconocidos para los seres humanos. En este contexto, resulta fundamental que el robot sea capaz de navegar de forma autónoma en estos entornos. Para lograr estos cometidos el agente robot debe ser capaz de responder a la siguientes preguntas:

*¿Dónde estoy?*

*¿Dónde estuve?*

*¿Por dónde voy?*

Para responder a estas preguntas es crítico tener un mapa del entorno y ubicarse en relación a este. La academia e industria acuden a las técnicas de SLAM como solución estándar al problema de obtención de un mapa y localización en un entorno desconocido.

En el presente trabajo se describe el sistema robótico elaborado, capaz de realizar SLAM en el contexto de la solución de un problema que implica cubrir completamente un entorno desconocido para el robot.

Para implementarlo, se realizó, en primer lugar, una investigación sobre el estado del arte del SLAM. Se llevó a cabo un estudio del marco teórico del problema y se relevaron diferentes bibliotecas de investigación de SLAM de código abierto. Respecto a este ítem se incluye una descripción general del problema del SLAM junto a los enfoques y técnicas más destacadas de la literatura estudiada.

Luego se describe la implementación del sistema robótico que resuelve el problema planteado. Para esto se implementó un sistema de SLAM basado en FastSLAM y calibrado mediante el uso de algoritmos evolutivos. Este sistema de SLAM brinda un sistema de localización confiable sobre el que implementar la solución de cubrimiento. Se incluyen en este documento una descripción del algoritmo de SLAM implementado, de la arquitectura del sistema y del proceso de calibración de parámetros utilizando algoritmos evolutivos.

Finalmente, se realizaron pruebas de rendimiento comparando contra otro sistema de cubrimiento implementado sobre un sistema de localización basado en integración de movimientos (odometría).

# Índice general

# Índice de figuras

# Índice de cuadros

# Índice de algoritmos

# Introducción

## Objetivos

El objetivo de este documento es introducir el trabajo realizado en el marco del proyecto de grado pgSLAM[17]. Se espera que a partir de la lectura del mismo el lector comprenda en forma general el problema de Localización y Construcción de Mapas en Simultáneo (del inglés Simultaneous Localization and Mapping, en adelante SLAM) y la naturaleza de los principales algoritmos y técnicas aplicados en esta área de investigación. Gran parte de estas técnicas fueron utilizados por los autores para desarrollar un sistema de SLAM basado en marcas.

Además, se pretende explicar al lector el problema sobre el cual se trabajó, y exponer con alto nivel de detalle todas las componentes que forman parte de la solución implementada y los resultados obtenidos.

## Organización del documento

El documento está compuesto por siete capítulos, que van desde una introducción al SLAM hasta el desarrollo de implementación de un sistema de SLAM y un caso de estudio que presenta una aplicación a un problema concreto.

En el primer capítulo se realiza una introducción al problema general del SLAM, se exponen las principales aplicaciones del mismo y, en último lugar, los principales desafíos involucrados.

Más adelante, en el segundo capítulo, se presentan los fundamentos de las soluciones al problema del SLAM relevadas y utilizadas en la solución desarrollada. Dos de los diferentes enfoques existentes en la literatura sobre el tema son desarrollados en este capítulo, bioinspirados y probabilísticos.

En el tercer capítulo el lector encontrará el desarrollo de un sistema de SLAM, sus principales características, componentes y calibración.

Una aplicación del SLAM se puede encontrar en el cuarto capítulo, donde se expone una propuesta concreta de un problema real simplificado. El resto del capítulo se dedica a presentar una solución específica al problema utilizando el sistema de SLAM implementado. La solución se subdivide en tres partes: mecánica, hardware y software, siendo el último el más relevante para los autores.

Luego, en el quinto capítulo se presentan las características más relevantes de la parametrización del sistema de SLAM desarrollado.

En el sexto capítulo se conduce al lector a través del proceso de pruebas realizado sobre el sistema de SLAM implementado.

Finalmente, en el sexto capítulo se presentan conclusiones y trabajos futuros.

# Capítulo 1

# El problema del SLAM

## 1.1. Introducción

Este capítulo comienza presentando la definición de robot autónomo, las motivaciones que llevan a los investigadores a trabajar en el problema del SLAM y finalmente se introduce el problema del SLAM propiamente dicho. Sobre el final de este capítulo se exponen las principales dificultades que tiene este problema en la robótica y algunas de sus clasificaciones.

## 1.2. Robots Autónomos

Existen diversas definiciones para el significado de robot autónomo o de funcionamiento autónomo. A continuación citamos dos definiciones de autores destacados en el área:

- Funcionar autónomamente implica que un robot puede operar, autocontenido[1], en variadas condiciones y sin necesidad de supervisión humana. Que un robot sea autónomo significa que puede adaptarse a los cambios en el ambiente (p.e. que se apaguen las luces) o a problemas en sí mismo (p.e. si se rompe alguna de sus partes) sin dejar de perseguir su objetivo [35].

- Un sistema es autónomo en la medida en que su comportamiento está determinado por su propia experiencia (en el largo o mediano plazo)[36]. El autor aclara también que sería demasiado estricto pedir una autonomía completa desde el inicio de la ejecución de un robot ya que el único conocimiento disponible es el incluido por el desarrollador.

De las definiciones se desprende que la capacidad de navegar en ambientes desconocidos forma una parte importante de lo que significa un robot autónomo móvil. En este sentido, uno de los principales problemas con los que se tiene que enfrentar un investigador o desarrollador de robots móviles, es el problema conjunto de la localización y armado de mapas del entorno. En la figura 1.1 se observa un robot autónomo creado por la NASA, que se encuentra en un ambiente desconocido y sin información precisa sobre su ubicación, sin embargo, la principal tarea de este robot es la navegación, y para realizar dicha actividad en forma eficiente y eficaz, se necesita conocer el mapa del entorno y la ubicación del robot en dicho mapa.

[1] Un robot autocontenido es aquel que posee todo lo necesario para desempeñar sus tareas en el propio robot. Es decir, que no depende de elementos externos que realicen tareas, por ejemplo, realizar cálculos fuera del robot.

Figura 1.1: Imagen del robot Oportunity de la NASA en el planeta Marte.
Imagen extraída de [37].

## 1.3. Motivación del Problema del SLAM

Existen ocasiones en las que se conoce de antemano el entorno en el cual se moverá el robot y es posible proporcionarle un mapa del mismo. Puede tomarse como ejemplo un agente robótico que debe limpiar un apartamento completo, utilizando un mapa de este para llevar cuenta de las habitaciones visitadas.

Por otro lado, existen situaciones en las que un agente robótico debe moverse en un entorno desconocido, pero cuenta con información precisa sobre su ubicación. Para conocer su ubicación el agente puede utilizar, por ejemplo, visión global (como ocurre en las competencias robóticas como el sumo[2] o fútbol de robots[3]) o un sistema GPS.

Sin embargo, existen casos aún más complejos que los mencionados, en los que no se conoce el entorno y tampoco se tiene información sobre la ubicación exacta del robot. En este caso, el robot deberá generar un mapa del entorno y mantener su ubicación en el mismo de forma simultánea. Esta tarea resulta compleja debido a que para poder localizarse de forma precisa se necesita un mapa, y por otro lado, para poder crear un mapa es menester estar localizado en forma exacta. Esta es el problema que estudia el SLAM, localización y armado de mapas en simultáneo. Algunos ejemplos, en los cuales es puede ser útil realizar SLAM, son:

- Exploración espacial
- Rescate en zonas afectadas por catástrofes
- Tareas domésticas
- Conducir un automóvil
- Tareas agropecuarias

[2]Competencia robótica inspirada en el deporte de mismo nombre, que consiste en que dos robots se saquen uno al otro de un dohyo. Por ejemplo http://www.fing.edu.uy/inco/eventos/sumo.uy/
[3]Versión robótica del fútbol. Por ejemplo www.robocup.org/

## 1.4. El problema del SLAM

Tomando en cuenta lo expuesto, es fácil deducir que el problema del SLAM aplica cuando el robot no tiene acceso a un mapa del entorno ni conoce tampoco su posición en el mismo. Un ejemplo de estas situaciones puede observare en la figura 1.2, donde un robot es utilizado para tareas de rescate luego de un accidente, en una zona donde el humano no puede acceder debido a los elevados niveles de radiactividad. En este ejemplo, el ambiente puede haber cambiado a causa del accidente y no reflejar lo señalado en los mapas disponibles. Adicionalmente la localización del robot no puede ser determinada con sistemas de GPS debido a que se trata de un entorno cerrado.

Luego, el robot solo dispone de la información proporcionada por las medidas obtenidas de los sensores y la noción del movimiento propio. El agente robótico intentará obtener un mapa del entorno y simultáneamente localizarse en relación a dicho mapa[37].

Figura 1.2: Robot *Packbot* de la empresa IRobot
luego del terremoto en Japón, para realizar tareas de asistencia en las plantas nucleares [5].

## 1.5. Dificultades que presenta el SLAM

A continuación se presentan los principales desafíos que tiene el problema del SLAM.

### 1.5.1. Manejo de incertidumbre

Como se mencionó anteriormente, determinar la posición exacta del robot localmente requiere una buena estimación del mapa en el cual el robot se mueve. Por otro lado, la determinación del mapa del entorno requiere conocimiento de la posición exacta del agente. Dado que el agente no posee inicialmente ninguno de estos datos, y dado que la incertidumbre de su posición aumenta con su movimiento, el algoritmo de SLAM debe ser capaz de manejar cierto error en los datos que son computados. Esta incertidumbre debe ser manejada de forma tal que el error en las estimaciones no crezca constantemente (de manera de evitar una divergencia en la estimación de la posición del robot y del mapa).

### 1.5.2. Sensores

Los sensores son dispositivos utilizados para obtener información del estado del entorno, o del robot mismo. En la figura 1.3 se observa un sensor láser que tiene la capacidad de realizar barridos de 180° y medidas a


Figura 1.3: Sensor láser comúnmente utilizado para hacer SLAM. Imagen extraída de [9].

distancias hasta de 100 metros[4]. Los sensores son limitados en lo que pueden percibir y no son totalmente precisos en sus medidas. Éstas restricciones vienen dadas por muchos factores. El rango y la resolución de un sensor están sujetos a limitaciones físicas. Un claro ejemplo son las cámaras, cuya calidad de imagen está condicionada por la calidad del lente, los filtros utilizados, la cantidad de megapixeles, el preprocesamiento de los valores obtenidos, entre otras limitaciones.

Además, las medidas realizadas por los sensores están sujetos a ruido estocástico, lo que perturba las medidas de formas impredecibles.

Luego, de lo mencionado anteriormente se desprende que los datos obtenidos desde un sensor deben ser manipulados con cuidado para extraer información útil.

### 1.5.3. Movimiento del robot

Mientras el robot se mueve en el entorno, se envían órdenes de movimiento a los actuadores, que comprenden información del cambio en la posición del robot. Esta información luego se procesará para actualizar la estimación de la posición del robot. Un problema puede ocurrir cuando, por ejemplo, se le envía una orden de movimiento a los motores y una de las ruedas (traccionada por el motor que recibió la orden) del robot queda en el aire o resbala, lo que implica que el robot no cambió su posición, o lo hizo pero en menor medida de lo esperado. En este sentido la estimación de la posición del robot será errónea si se basa solamente en esta información.

### 1.5.4. Cerrar ciclos

Cerrar ciclos refiere a la situación en que el robot debe poder reconocer cuando pasa por un lugar que ya ha sido visitado. Realizar esta tarea con éxito se vuelve primordial en los mapas que poseen algún cruce para poder armar el mapa correctamente. En la figura 1.4 se puede apreciar un escenario típico donde se presenta el problema del cierre de ciclos.

[4]Dato obtenido de www.sick.com

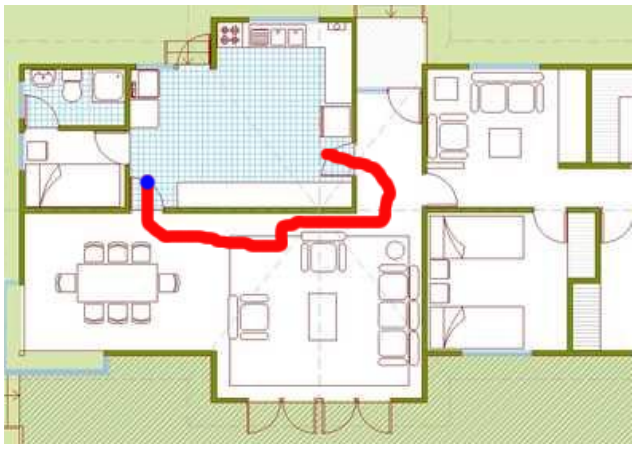


Figura 1.4: La figura muestra el recorrido que hizo un robot en un mapa.
Comienza en la cocina (donde se encuentra el punto azul). Éste sale de la habitación, recorre una sala y entra nuevamente en la cocina. Para que el algoritmo de SLAM arme el mapa correctamente deberá reconocer que la cocina en la que entró en segundo lugar es la misma de donde partió. Imagen extraída de[8].

### 1.5.5. Asociación incorrecta

El algoritmo de SLAM debe ser lo suficientemente preciso como para no confundir dos lugares diferentes de forma que lo lleve a creer que son el mismo. En caso de que el robot estime que dos lugares diferentes con características similares son efectivamente el mismo, cerrará un ciclo donde no lo hay y generará un mapa incorrecto del entorno.

### 1.5.6. Capacidad de cómputo

Una gran limitación que encuentran los investigadores y desarrolladores a la hora de resolver un problema de SLAM es el procesamiento de los datos. Los algoritmos de SLAM suelen realizar tareas de procesamiento intensas debido a la densidad de la representación del mapa y a la complejidad de los cálculos involucrados en la estimación de la posición. Esto muchas veces hace difícil el procesamiento dentro del robot y obliga a extraer los datos sensados para ser procesados afuera de este.

## 1.6. Clasificaciones de SLAM

A continuación se incluyen las clasificaciones más utilizadas en la literatura sobre el problema del SLAM.

### 1.6.1. Sensores utilizados

Los algoritmos de SLAM pueden ser clasificados según el tipo de sensor que utilizan para obtener información del entorno. Estos sensores pueden ser:

- Sensores de barrido láser: realizan varias medidas por segundo utilizando un láser sobre una plataforma rotativa. Cada medida brinda información sobre orientación (relativa al robot) y distancia del objeto sensado.

- Sensores de ultrasonido e infrarrojos: estos sensores miden la distancia al objeto más cercano. A diferencia de los sensores de barrido láser, los sensores ultrasónicos e infrarrojos no brindan información

precisa sobre la orientación relativa del objeto sensado.

- Cámaras de video: Estos sensores permiten obtener imágenes del entorno y procesarlas para conocer la ubicación de los elementos de interés.

### 1.6.2. *Offline* vs. *Online*

En el caso del SLAM *online* se procesa la información en el mismo robot mientras este navega en el entorno. Por otro lado, en el *offline* se realiza SLAM sobre un conjunto de datos que previamente fueron recuperados con algún robot, tanto de la medida de sus sensores como la información de movimiento robot.

### 1.6.3. Topológico vs. Métrico

Algunas técnicas de armado de mapas solamente mantienen la descripción de ciertas características del entorno y la relación entre ellas. Estos métodos son conocidos como topológicos. Por otro lado, los métodos métricos proveen información de distancias entre los lugares. En los últimos años, los métodos topológicos han pasado de moda a pesar de la amplia evidencia de que los humanos utilizan a menudo información topológica, y que éstos pueden resolver el problema de SLAM[37].

### 1.6.4. Activo vs. Pasivo

En los algoritmos de SLAM pasivos, es otra entidad quien se encarga de controlar el robot, mientras que el algoritmo de SLAM es puramente observador. La gran mayoría de los algoritmos de SLAM son de este tipo[37]. En el caso de los algoritmos de SLAM activo, el robot explora de forma activa el entorno en busca de conseguir un mapa más preciso en el menor tiempo posible. Existen técnicas híbridas donde el algoritmo de SLAM sólo controla la dirección de los sensores y otra entidad se encarga de la dirección del movimiento del robot.

### 1.6.5. Estático vs. Dinámico

En el caso del SLAM estático se asume que el entorno es no cambia con el paso del tiempo. En el caso del SLAM dinámico el algoritmo debe estar preparado para contemplar variaciones en el entorno de un momento a otro. La mayoría de la literatura asume entornos estáticos[37].

### 1.6.6. Volumétrico vs. Basado en marcas

En SLAM volumétrico, el mapa es representado a una resolución que permite una reconstrucción fotográfica del entorno. En este caso el costo computacional del procesamiento de la información es alto. Por otro lado, en el SLAM basado en marcas se extraen características de las medidas de los sensores de forma de armar el mapa en base a marcas dispersas (ver sección 2.3). Las técnicas que utilizan SLAM basado en marcas suelen ser mas eficientes ya que se descarta gran parte de la información de los sensores[37].

# Capítulo 2

# Fundamentos teóricos de la soluciones de SLAM

## 2.1. Introducción

El problema del SLAM puede dividirse en varios módulos o subproblemas, siendo cada uno de estos un campo de investigación en sí. Este capítulo se organiza de la siguiente manera: se dedica una sección a la revisión de los dos principales enfoques de la solución del SLAM, bioinspirados y probabilísticos. Luego se dedican las siguientes secciones a la discusión de los subproblemas más importantes y sus soluciones más comunes en los sistemas SLAM probabilísticos relevados.

## 2.2. Enfoques

Existen dos grandes enfoques para la solución del problema del SLAM, bioinspirados y probabilísticos. Estos enfoques difieren principalmente en la forma en que procesan la información de entrada, de modo de encontrar una buena estimación de la ubicación del robot y mapa del entorno.

Actualmente, el enfoque probabilístico domina el campo y ha logrado implementaciones que escalan a ambientes grandes y complejos[32]. Sin embargo, la gran capacidad de navegar que poseen mamíferos como las ratas, simios y seres humanos han motivado la investigación y desarrollo de sistemas que imitan los mecanismos biológicos de navegación de estos animales.

### 2.2.1. Bioinspirados

Desde la década de 1970, el entendimiento del funcionamiento del cerebro mamífero vinculado a las actividades de navegación y confección de mapas del entorno ha ido aumentando. Sin embargo, descubrimientos a partir del año 2005 en adelante han cambiado la forma en que se conciben los mecanismos mentales utilizados por los mamíferos para estas actividades. Estos descubrimientos corresponden al hallazgo de células especializadas en las actividades de navegación y reconocimiento del entorno, principalmente ubicadas en el hipocampo. Estas células evidencian la existencia de una representación interna del entorno.

Los sistemas de SLAM bioinspirados buscan reproducir en modelos computacionales los circuitos neuronales que dan sustento a esta representación espacial del cerebro.

Para una descripción más detallada del enfoque bioinspirado pueden consultarse documentos anteriores del proyecto[28].

### 2.2.2. SLAM Probabilísticos

En el contexto del SLAM existen diversas fuentes de incertidumbre, es decir, factores que incrementan la dificultad de estimar la ubicación y armar un mapa del entorno correcto. Algunos de estos factores son:

- Ruido en la información sensada
- Imprecisión en el desplazamiento
- Simetrías en el entorno
- Observabilidad parcial
- Dinamismo del entorno

En pos de adaptarse a esta incertidumbre, gran parte de las soluciones del SLAM plantean la estimación de la posición del robot y el mapa del entorno como una distribución de probabilidad.

Estos métodos se concentran en encontrar la distribución de probabilidad de la posición del robot y del mapa del entorno a través del tiempo. Para ilustrar este concepto de estimación como una distribución de probabilidad, resulta conveniente analizar la figura 2.1. En esta figura se puede observar un robot que navega en un entorno unidimensional con tres puertas como únicos objetos distinguibles. En la parte inferior se despliega la estimación de la posición del robot como una distribución de probabilidad, en este caso una distribución gaussiana, donde se le asigna a cada intervalo del eje $x$ una probabilidad de que el robot se encuentre en dicho intervalo.

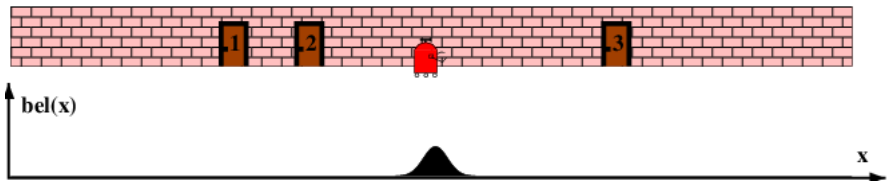



Figura 2.1: Un agente se mueve en un entorno con tres objetos distinguibles, las puertas. La estimación de su ubicación se representada como una distribución de probabilidad, como la graficada en la parte inferior de la imágen.
Imagen extraída de [39].

Luego, se definen tres variables estocásticas involucradas en el modelo de SLAM probabilístico, que se explican a continuación.

#### 2.2.2.1. Variables

**Estado oculto** La variable $x_i$ modela el valor a estimar en el instante de tiempo $i$. Para el caso del SLAM, esta variable puede representar la posición del robot, el mapa, o ambos. En ocasiones, esta variable se separa en la posición del robot $x_i$ y el mapa $m_i$, en otras $x_i$ involucra ambos conceptos. Esta variable suele recibir el nombre de "estado oculto" debido a que su valor no es directamente observable (no puede ser directamente sensada) y, por eso, debe estimarse en función de las restantes variables observables.

**Observación** La variable $z_i$ modela la observación realizada en el instante $i$. Esta observación puede constar de una medida, un conjunto de medidas o una observación de alto nivel de abstracción, como puede ser "el objeto distinguible 1 está a 3 metros de distancia y a 30 grados hacia la izquierda".

**Movimiento** La variable $u_i$ modela la información del movimiento realizado desde el instante $i-1$ al instante $i$.

Algunos ejemplos son los comando enviados al robot, como la velocidad (lineal o angular) del robot, la velocidad enviada a cada motor individualmente, o la distancia a recorrer por el robot. Estos comandos brindan información preliminar sobre el movimiento efectivamente realizado por el robot.

Otra fuente de información que es modelada utilizando esta variable es la información de sensores propioseptivos como encoders o sensores de inercia, que brindan nociones del movimiento propio al robot, ya que estos sensores también proporcionan información preliminar sobre el movimiento propio del robot.

Si bien en este caso, la información proviene de sensores, suele diferenciarse de las medidas $z_i$ debido a que las últimas refieren a medidas sobre el entorno y no a información introspectiva. En general, cuando se hable de información sensada, se estará haciendo referencia al sensado del entorno $z_i$.

Esta variable recibe varios nombres en la literatura, como por ejemplo información de controles o información de odometría. En este documento se utilizarán ambas indistintamente.

**El tiempo** Todas las variables estocásticas toman sus valores en diferentes instantes de tiempo. Se considera la variable adicional de tiempo $t$, que suele tomarse como un conjunto discreto de instantes, que suelen coincidir con los instantes en los que el robot recibe información de odometría o sensado. Los subíndices en las variables estocásticas $x_i$, $z_i$ y $u_i$ corresponden al instante de tiempo al que están asociadas estas variables.

#### 2.2.2.2. Formulación del problema

Tomando en cuenta las variables arriba definidas, la distribución de probabilidad a estimar para solucionar el problema del SLAM puede formularse como:

$$p(x_{1:t}|z_{1:t}, u_{1:t})$$

Es decir, la distribución de probabilidad de estado $x_{1:t}$ ($x_1..x_t$) que se adapta mejor a las observaciones $z_{1:t}$ y controles $u_{1:t}$.

Alternativamente se puede encontrar en la literatura esta probabilidad representada como

$$bel(x_{1:t}|z_{1:t}, u_{1:t})$$

para hacer énfasis en el hecho de que es la creencia (*belief*) del robot sobre el estado del sistema. Se utilizan en este texto ambas notaciones indiferentemente.

En conclusión, resolver el problema de SLAM implica estimar la distribución de probabilidad para la variable que representa el estado del sistema $x_i$ en cada instante de tiempo discreto del 1 al $t$, es decir la variable $x_{1:t}$.

#### 2.2.2.3. Redes de Bayes

Para realizar la estimación de $x_i$ se modela a esta variable de acuerdo a la red bayesiana incluida en la figura 2.2. En esta red se observa la relación de dependencia entre las variables involucradas en el problema

de SLAM, donde una flecha de una variable $A$ a una $B$ indica que la primera incide en la segunda, es decir que $p(B|A)$ y $p(B)$ no son necesariamente iguales ($A$ y $B$ no son independientes). Las variables del estado $x_i$ afectan solamente a la observación correspondiente $z_i$ y al estado siguiente $x_{i+1}$. Los controles $u_i$ afectan solamente al estado correspondiente $x_i$. Finalmente, como es coherente, todas las observaciones $z_i$ están influenciadas también por el mapa real del entorno $m$.

Las flechas de dependencia entre variables puede verse como un flujo de información. El valor que toma una variable precedente afecta al posible valor que toma la subsiguiente, y de esta forma la información de esta variable es propagada por la red. Si se observa la estructura de la red de la figura 2.2, puede observarse que toda la información de variables del pasado ($u_{1:i-1}$, $z_{1:i-1}$, $x_{1:i-1}$) afecta a las variables del futuro ($u_{i+1:t}$, $z_{i+1:t}$, $x_{i+1:t}$), solo a través de la estimación del estado en el presente $x_i$. Si el valor de $x_i$ es conocido, la información del pasado pierde relevancia, pues no afecta al posible valor que puede tomar esta variable, porque el mismo está ya determinado (no hay especulación). Esto implica que observaciones $z_i$ y controles $u_i$ anteriores a este instante no afectan el estado del sistema en instantes posteriores, conocido el estado del sistema $x_i$. Esto es llamado "Asunción de Markov" (Markov Assumption) en la literatura y puede verse violada por algunos factores en la práctica[39], por ejemplo por componentes no modelados correspondientes a la dinámica del sistema a modelar $x_i$. Al existir componentes dinámicos no modelados (p. e. personas) la independencia condicional entre medidas $z_i$ deja de cumplirse, debido a que el sensado de una persona en un momento $i-1$ implica que es probable que se sense nuevamente la persona en el instante $i$, aún cuando el estado $x_i$ es conocido, porque este no modela la persona sensada.

También se infiere otro hecho interesante de la figura 2.2. Existe un vínculo que relaciona las medidas $z_i$, a través del tiempo. Este vínculo se realiza a través de la trayectoria del robot, es decir $x_{1:t}$. En la práctica, esto implica que las diferentes partes del mapa sensadas se van correlacionando por medio de la trayectoria del robot. Esta correlación torna computacionalmente más costosa la aplicación del algoritmo, debido a que es necesario actualizar todo el mapa luego de realizar cambios en una localidad. Sin embargo, esta correlación es, de hecho, lo que permite que la estimación del mapa converja, es decir que su incertidumbre disminuya por debajo de un valor definido, a pesar de la incertidumbre en la trayectoria. Esto se debe a que la correlación hace que la incertidumbre sobre la distancia relativa entre partes del mapa disminuya[18].

#### 2.2.2.4. Filtros de Bayes

Como se verá más adelante en la sección 2.6, algunas soluciones del SLAM se basan en el procesamiento dato a dato de la información. Este procesamiento tiene como marco teórico los Filtros[1] de Bayes, por lo que resulta pertinente explicar este tema.

Los Filtros de Bayes son algoritmos genéricos que permiten estimar el estado del sistema oculto $x_t$ en función del estado anterior $x_{t-1}$, en este caso tomando la información de sensado $z_t$ y odometría $u_t$ más recientes.

Para explicar estos algoritmos conviene extender la notación $bel(x_{1:t}|z_{1:t}, u_{1:t})$, agregando un nuevo término:

$$\overline{bel}(x_{1:t}|z_{1:t-1}, u_{1:t})$$

[1] El término de filtro tiene sus orígenes en la literatura del procesamiento de señales con ruido . Estas herramientas intentan solucionar el problema de estimar una señal original x(t), en línea, cuando sólo es posible observar
$y(t) = x(t) + r(t)$
donde r(t) es una señal de ruido no conocida. Muchas soluciones a este problema buscan encontrar el x*(t) óptimo tal de maximizar la probabilidad
$p(x(t)|y(0) \dots y(h))$

que designa la creencia del estado del sistema incluyendo solo la información de odometría en el tiempo $t$ y no la de sensado.

Mediante manipulación matemática, cuyo desarrollo puede consultarse en [39], es posible llegar a una definición recursiva de *bel* en función de $\overline{bel}$ y viceversa:

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}).bel(x_{t-1})dx_{t-1} \tag{2.1}$$

$$bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t) \tag{2.2}$$

De las ecuaciones 2.1 y 2.2 puede deducirse que es posible obtener la distribución de probabilidad del estado en el tiempo $t$ ($bel(x_t)$) a partir del estado en el tiempo $t-1(bel(x_{t-1}))$ utilizando como paso intermedio el cálculo de $\overline{bel}(x_t)$. Esto resulta interesante pues permite mantener una distribución de probabilidad de estado oculto, procesando la información de sensado y odometría de a una a la vez.

El problema vinculado a la utilización de los Filtros de Bayes como solución de SLAM radica en que las operaciones de integración en 2.1 y multiplicación en 2.2 no son fácilmente resolubles. Se profundizará más en este punto en las subsecciones 2.6.2 y 2.6.3, ya que los Filtros de Bayes son el marco teórico que sostiene lo allí presentado.

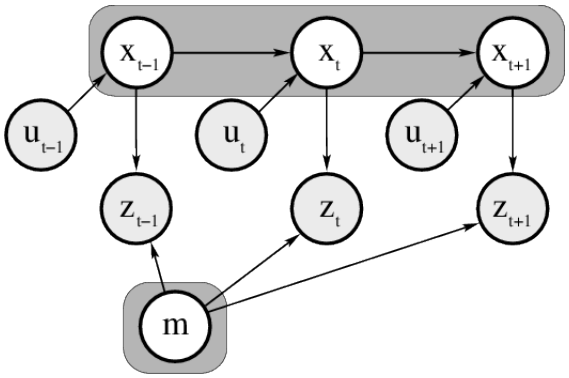



Figura 2.2: La red bayesiana que modela el problema de estimación probabilística del SLAM.
En gris oscuro se muestran las variables ocultas a estimar, compuestas por la posición $x_i$ y el mapa $m$. Los círculos en gris claro corresponden a las variables cuyo valor puede ser observado. La figura ha sido extraída de [37].

## 2.3. Representación del mapa

Un algoritmo de SLAM puede llevar una representación o mapa de su ambiente de diversas maneras. Cada uno de estos tipos de mapas tienen sus ventajas y desventajas asociadas. Al momento de elegir una representación de mapa a implementar, se deben tener en cuenta las siguientes preguntas:

- ¿Es necesario mantener en el mapa nociones métricas?
- ¿Cuál es el uso que se le dará al mapa?
- ¿El entorno es dinámico?
- ¿De qué sensores se dispone?

- ¿De cuánto poder de cómputo se dispone?

Se desarrolla, de las representaciones relevadas, solamente la representación de mapa basada en marcas, ya que fue la utilizada en la solución implementada.

### 2.3.1. Mapa basado en marcas

Una representación ampliamente utilizada en los sistemas relevados es la de un mapa compuesto por un conjunto de marcas[24, 21, 16, 26, 23, 38, 19].

Una marca es una o más características de interés perceptualmente distinguibles, ubicadas en un objeto o un lugar de interés [35].

La mayoría de los sistemas basados en marcas usan una cámara como sensor principal. Sin embargo pueden distinguirse marcas utilizando otros tipos de sensores. Por ejemplo, con múltiples medidas de un sonar o un sensor láser es posible identificar esquinas, puertas, columnas y ventanas que pueden servir como marcas. Se relevó, incluso, un sistema[22] que utiliza las medidas de un sensor láser para obtener perfiles de los árboles presentes en el entorno y utilizarlos como marcas.

Un mapa de marcas consta de un conjunto de coordenadas que indican la posición de cada marca en el entorno de trabajo.

Los mapas de marcas no contienen información detallada o volumétrica del entorno. Esto hace que la representación sea compacta. Por otro lado, debido a esta falta de información, no es posible realizar tareas de planificación de movimientos utilizando este tipo de mapas.

Las marcas pueden ser perfectamente identificables, es decir que es posible saber de cuál marca se trata con tan solo observarla. También existen sistemas donde esta correspondencia no puede ser determinada con seguridad. En los casos en los que las marcas no son perfectamente distinguibles, el algoritmo que utilice este mapa deberá tomar en cuenta esta fuente de incertidumbre.

En el contexto del SLAM, los mapas de marcas suelen representarse, en vez de con un conjunto de coordenadas, con las distribuciones de probabilidad de la ubicación de cada marca. De esta forma es posible representar la incertidumbre en la posición de las marcas, que puede ser actualizada a medida que se navega por el entorno. Tomando como ejemplo el sistema implementado, para cada marca se almacenan los dos parámetros de una gaussiana correspondiente, la media $\mu$ y la varianza $\sigma^2$. Esto puede representarse gráficamente, para el caso de un entorno de trabajo de dos dimensiones, como un conjunto de elipses, donde el centro indica la media de la gaussiana y los ejes los vectores propios de su matriz de covarianza, ver figura 2.3.

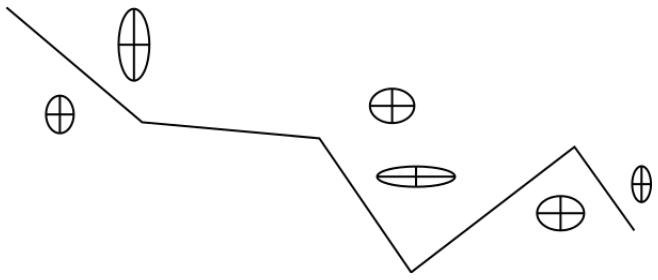


Figura 2.3: Representación gráfica de un conjunto de marcas. Cada marca es representada por una elipse. La linea marca la trayectoria del robot.

## 2.4. Sensado

Cuando un robot sensa el ambiente, recibe información del entorno. Tanto la actualización del mapa como la localización del robot requieren relacionar la información sensada con el mapa actual. Para esto, se definen distribuciones de probabilidad de lo sensado en función de cierta configuración del entorno, o cierto mapa, y la posición del robot. Estas distribuciones de probabilidad llevan el nombre de *modelo de sensado* y pueden escribirse como,

$$p(z_i|x_i, m)$$

La representación de estos modelos como distribuciones de probabilidad permite contemplar las fuentes de ruido que hacen que el sensor no sea un instrumento completamente fiable. De esta manera, los posibles errores cometidos por el sensor son modelados naturalmente en la solución del SLAM, tomándolos como parte de la incertidumbre a manejar en el proceso de estimación del estado del sistema $x_i$.

Resulta importante recalcar que el modelo de sensado se fija para un sistema de SLAM dado y forma parte de su calibración.

Naturalmente, el modelo de sensado dependerá del sensor y representación del mapa utilizado. A continuación se describe uno de los modelos relevados, que fue el utilizado en el prototipo final. Se pueden encontrar otros modelos en el documento de relevamiento del estado del arte[28].

### 2.4.1. *Pinhole Model*

Este modelo es utilizado por las soluciones de SLAM que utilizan cámaras para la extracción de marcas (ver sección 2.3). Este modelo se basa en el el funcionamiento básico de una cámara tradicional, una caja con un orificio diminuto por donde entra la luz que es interpretada como una imagen, ver figura 2.4.

El modelo Pinhole permite relacionar parámetros de la imagen proyectada y parámetros intrínsecos de la cámara con las medidas y posición de los objetos observados. Por ejemplo, realizando un análisis geométrico basado en triángulos homomórficos es posible derivar la relación:

$$d/D = f/Z$$

donde $d$ corresponde al tamaño del objeto en la imagen, $D$ al tamaño real, $f$ a la distancia entre el orificio y el plano de proyección (este parámetro es conocido como *focal lenght*) y $Z$ es la distancia al objeto, ver figura 2.4.

También permite inferir el ángulo relativo de una marca con respecto al robot, aplicando derivaciones similares, basándose en la hipótesis de que la cámara consta de un orificio pequeño y un plano de proyección.

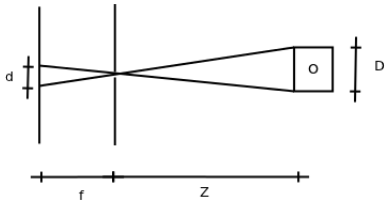



Figura 2.4: Modelo de la cámara *pinhole* con los parámetros f, Z, d y D asociados.

Este modelo es utilizado ampliamente por su simplicidad, aún siendo un modelo básico. Algunas implementaciones agregan a este modelo la noción de distorsión causada por los lentes[16].

## 2.5. Movimiento

Para mantener un estimado de la ubicación del robot es importante mantener una noción del movimiento propio del robot. Como se observó anteriormente, el movimiento del robot no es un proceso determinista, debido a que, por ejemplo, si las ruedas resbalan sobre el suelo, el robot se desplazará menos de lo esperado. Por esto, el modelado del movimiento propio del robot se realiza, al igual que en los modelos de sensado, con distribuciones de probabilidad. Se define el *modelo de movimiento* o modelo de transición como:

$$p(x_t|x_{t-1}, u_t)$$

Es decir, la probabilidad de que el sistema se encuentre en determinado estado $x_t$, dado que se encontraba en el estado $x_{t-1}$ y aplica los controles $u_t$. Es importante recordar que la variable $x_t$ puede modelar la posición del robot únicamente, o la posición del robot y el mapa conjuntamente.

Este modelo de transición también modelará como cambia el mapa a través del tiempo, incluso modelando la evolución de la posición de objetos dinámicos en el ambiente. Sin embargo, la mayoría de las implementaciones relevadas consideran el ambiente estático y su modelo de transición se remite a la porción de $x_t$ relacionada con la posición del robot.

Resulta importante recalcar que el modelo de movimiento se fija para un sistema de SLAM dado y forma parte de su calibración.

A continuación se incluye una descripción del modelo de movimiento denominado *modelo de odometría*, utilizado en la implementación final. Se puede encontrar más información al respecto en [28].

### 2.5.1. Modelo de odometría

Este modelo plantea tomar los datos de odometría para calcular el movimiento inmediatamente anterior del robot. Los datos de odometría suelen obtenerse de la integración de la información de los *encoders* del robot, sin embargo, es posible obtenerla de otras fuentes como:

- Análisis de imágenes consecutivas para detectar cambios que permitan estimar el movimiento propio[31]
- Técnicas de *scanmatching* que permiten estimar el movimiento propio respecto a un mapa local[20, 33]
- Integración de datos provenientes de sensores de inercia

Este modelo descompone el movimiento realizado en tres componentes (ver figura 2.5):

- Una rotación inicial de ángulo $\delta_{rot1}$
- Una traslación de largo $\delta_{tras}$
- Una rotación final de ángulo $\delta_{rot2}$

Para representar las estocasticidad del movimiento real del robot, se le agregan a este modelo tres variables que permiten contemplar un posible error en el modelo:

- Una variable $\hat{\delta_{rot1}}$ que contempla un error en la primera rotación

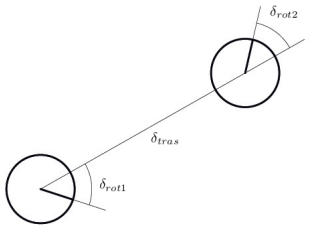



Figura 2.5: La descomposición del movimiento en tres submovimientos.
Imagen extraída de [39].

- Una variable $\hat{\delta_{tras}}$ que contempla un error en la traslación
- Una variable $\hat{\delta_{rot2}}$ que contempla un error en la rotación final

Luego, si tomamos como ejemplo el submovimiento de traslación, este será modelado como un movimiento de magnitud lineal $\delta_{tras} + \hat{\delta_{tras}}$, donde $\delta_{tras}$ es el movimiento reportado y $\hat{\delta_{tras}}$ es el error cometido, que es desconocido.

## 2.6. Procesamiento de la información

El procesamiento de la información es el proceso central en un algoritmo de SLAM. Este procesamiento es el que convierte la información sensorial y de odometría entrante en un estimado de la posición del robot y el mapa del entorno.

### 2.6.1. Taxonomías del procesamiento de información de SLAM

Existen diversas taxonomías para clasificar a este módulo de SLAM, a continuación se detallan algunas de ellas.

#### 2.6.1.1. Full vs. Online

Existen dos problemas diferentes a resolver en SLAM. El primero consiste en estimar la trayectoria y mapa más verosímil dado un conjunto de observaciones $z_{1:t}$ y un conjunto de controles $u_{1:t}$. Este problema es conocido como Full SLAM y puede formularse como:

$$p(x_{1:t}|z_{1:t}, u_{1:t}) \tag{2.3}$$

Otro planteo del SLAM probabilista propone procesar la información entrante una a una, hallando la posición más verosímil en cada instante. Este problema es conocido como Online SLAM y puede formularse como:

$$p(x_t|z_t, u_t, x_{t-1}) \tag{2.4}$$

Debido a la similitud del problema de Online SLAM con el problema de filtros de información, algunas soluciones al problema de SLAM toman el nombre de filtros (p.e. Filtros de Kalman, Filtros de Partículas). Estas soluciones del Online SLAM se basan en el Filtro de Bayes explicado en la sección 2.2.2.

Al observar las ecuaciones 2.3 y 2.4 puede verse que la diferencia entre ellas radica en que Full SLAM toma en cuenta toda la información disponible al momento, mientras que Online SLAM calcula los cambios en el estado oculto $x_t$ de a un paso, tomando sólo la última información disponible. En términos matemáticos, esto puede expresarse como que la diferencia radica en que Online SLAM realiza una marginalización de toda la información anterior de sensado en el estado oculto del sistema al momento $t$.

El problema de Full SLAM suele consumir más recursos. Por otro lado, el problema de Online SLAM es más complejo debido a que debe lograrse una buena marginalización de la información. Aún así, el problema de Online SLAM goza de mayor popularidad debido a que puede ser ejecutado dentro de un robot operando en tiempo real. Estas ideas son desarrolladas en mayor detalle en [28].

En la práctica también existen algoritmos híbridos, combinando Full y Online SLAM, por ejemplo [26].

Finalmente, resulta importante notar que esta taxonomía es independiente a la clasificación realizada en la sección 1.6.2, que refiere al momento en que se realiza el procesamiento de la información, y no a la forma de procesamiento y la cantidad de datos tomados en cuenta simultáneamente.

#### 2.6.1.2. Paramétrico vs. No Paramétrico

Cuando se estima la distribución de probabilidad del estado oculto se busca encontrar una función de densidad de probabilidad, usualmente sobre un espacio de tres o más dimensiones. La memoria necesaria para el almacenamiento de esta distribución y el tiempo necesario para su actualización, crecen con la resolución con la que se desea modelar a esta distribución. Es decir, cuanto mayor es la precisión con la que se modela la distribución de probabilidad, mayor es el poder de cómputo necesario para mantener ese mod Por lo tanto, es necesario realizar simplificaciones al modelo, para que la solución sea computable en tiempo razonables con una cantidad de memoria razonable.

La simplificación más fuerte que suele hacerse es utilizar distribuciones de probabilidad conocidas que dependan de un conjunto pequeño de parámetros, conocidas como distribuciones paramétricas. El ejemplo más común es la utilización de una distribución gaussiana que puede representarse utilizando la media $\mu$ y la varianza $\sigma^2$, como se verá más adelante en la subsección Filtros de Kalman. Al utilizar estas distribuciones paramétricas, el algoritmo solo debe trabajar con este conjunto reducido de parámetros para la actualización de su estimado, a medida que llega nueva información.

La gran desventaja que presenta la utilización de distribuciones paramétricas para representar el estado del sistema radica en que sólo es posible representar un conjunto reducido de funciones de densidad posibles. Por ejemplo, en el caso de las distribuciones gaussianas, se observa que estas distribuciones son unimodales, es decir, que tienen un solo máximo local. En la práctica, esto implica que el estimador solo puede representar una única creencia sobre el estado del sistema.

Otros sistemas, como los Filtros de Partículas (ver sección 2.6.3), son capaces de representar cualquier función de densidad de probabilidad y, por lo tanto, mantener varios máximos locales en su distribución de probabilidad. Esto equivale a mantener varios estimados completamente diferentes del estado del sistema, de forma concurrente. Como contrapartida, la actualización de estas distribuciones a partir de la información entrante suele ser más costosa.

Existen sistemas que combinan ambos tipos de distribución, utilizando distribuciones unimodales para algunas variables y multimodales para otras[33].

### 2.6.2. Filtros de Kalman

Los Filtros de Kalman[27] son una solución al problema de Online SLAM. La principal característica de esta solución es el uso de distribuciones gaussianas como estimador. Esta solución se basa en el hecho que al sumar o multiplicar una variable de distribución gaussiana con un número (o matriz en el caso multivariable) se obtiene otra variable de distribución gaussiana. Luego, el algoritmo propone que la evolución del sistema puede modelarse mediante dos ecuaciones lineales, que permiten mantener el estimado del estado del sistema como una distribución gaussiana.

#### 2.6.2.1. Modelado

El modelado de la evolución del sistema se realiza mediante dos ecuaciones, la ecuación de transición y la ecuación de observación. A continuación se describen ambas ecuaciones.

**Ecuación de transición** La primera ecuación actualiza el estado del estimado del sistema en función de los cambios inherentes del sistema, los cambios que ocurren independientemente de los factores modelados, y otra en función de la información de odometría:

$$x_t = F.x_{t-1} + B.u_t + r_t \tag{2.5}$$

donde:

- $x_t$ refiere al estado del sistema a estimar
- $F$ consta de una matriz que representa el cambio inherente al sistema. Este cambio podría representar, por ejemplo, la evolución de la posición de un objeto del entorno que viaja a velocidad constante
- $u_t$ representa la información de odometría
- $B$ consta de una matriz que representa la transformación de la información de odometría en los cambios producidos en el sistema. Esta matriz representa, de hecho, el concepto de modelo de movimiento del sistema explicado en la sección 2.5
- $r_t$ es un factor de ruido que representa la naturaleza estocástica de los otros términos de la ecuación. Este ruido es también gaussiano

**Ecuación de observación** La segunda ecuación propone una relación entre el estado estimado del sistema y la observación a realizar por el robot en un tiempo $t$. La ecuación puede expresarse como:

$$z_t = H.x_t + w_t \tag{2.6}$$

donde:

- $z_t$ corresponde a la información de sensado
- $H$ consta de una matriz que transforma el estado del sistema en la observación a realizar. Por ejemplo esta matriz podría transformar las coordenadas de una marca en las coordenadas del campo visual del robot (proyección). Esta matriz representa, de hecho, el concepto de modelo de sensado del sistema explicado en la sección 2.4

- $w_t$ modela fuentes de ruido como puede ser el ruido introducido por los sensores del robot. Este ruido es también gaussiano

**Cálculo del estado del sistema** Bajo las condiciones mencionadas, la solución que minimiza el error cuadrático esperado[2] del estado del sistema puede calcularse de forma cerrada utilizando un Filtro de Kalman[40]. Los Filtros de Bayes comentados en la sección 2.2.2.4 permiten explicar esta idea de cálculo cerrado. Si se observan las ecuaciones del Filtro de Bayes:

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}).bel(x_{t-1})dx_{t-1} \tag{2.7}$$

$$bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t) \tag{2.8}$$

se puede observar que si:

- La distribución de probabilidad de $bel(x_0)$ es una distribución gaussiana,
- El modelo de movimiento $p(x_t|u_t x_{t-1})$ es lineal en $x_{t-1}$, y
- El modelo de sensado $p(z_t|x_t)$ también es lineal

entonces, las integrales y multiplicaciones pueden calcularse de forma cerrada, es decir, utilizando fórmulas que transforma una gaussiana en otra, manipulando los parámetros de media y varianza.

Por lo tanto, los cálculos de actualización del estado del sistema involucran únicamente operaciones de álgebra lineal para actualizar los estimadores de la posición del sistema (media $\mu$ y varianza $\sigma^2$).

En contraste, sin las hipótesis de trabajo de modelos lineales y distribuciones gaussianas, las integrales del Filtro de Bayes se vuelven irresolubles (*intractable*), incluso al tratar problemas simples de pocas dimensiones.

Por más información sobre los Filtros de Kalman puede consultarse [40, 39].

**Restricciones del modelo** El algoritmo de Kalman propone la evolución de un sistema, modelado por distribuciones gaussianas, regido por dos ecuaciones lineales. La linealidad de estas ecuaciones representa una restricción importante. Supongamos que nuestras observaciones constan del sensado de la distancia a una determinada marca. La función que toma como entrada las coordenadas del robot y la marca, y retorna la distancia, es decir el modelo de sensado correspondiente a este caso, no es una función lineal. Esto se debe a que la norma euclidiana depende de forma no lineal de sus parámetros ($dist = \sqrt{x^2 + y^2}$). Por lo tanto, al utilizar este sistema se cometerían sistemáticamente errores por utilizar una función lineal (la más aproximada posible) para la representación de la función de distancia, que no es lineal.

Además, la utilización de una distribución gaussiana como estimador del estado del sistema implica que sólo es posible mantener un único máximo local en la creencia del estado de este sistema. Esto viola las hipótesis de trabajo necesarias para la marginalización de la información realizada por los filtros, pudiendo en algunos casos hacer que la estimación del estado oculto diverja.

[2] Algunos problemas de optimización suelen plantearse como la minimización del error del sistema al cuadrado, donde el error corresponde al valor estimado $\bar{x}$ menos el valor real $x$. De otra forma:

$$x_{sol} = \underset{\bar{x}}{\text{argmin}}((\bar{x} - x)^2)$$

#### 2.6.2.2. Extensiones a la propuesta

Para eliminar la restricción de modelos de sensado y transición lineales, se diseñaron extensiones a la propuesta original. En esta variante, conocida como Filtro de Kalman Extendido, se reemplazan los modelos de sensado y odometría para contemplar funciones no lineales. Las ecuaciones 2.5 y 2.6 quedan:

$$x_t = f(x_{t-1}, u_t) + r_t$$

$$z_t = h(x_t) + w_t$$

donde $f$ y $h$ son los modelos de transición y sensado no lineales, respectivamente.

Esta modificación permite la utilización de Filtros de Kalman con modelos no lineales. Se logra utilizando un algoritmo similar al original, pero linealizando los modelos de datos utilizando series de Taylor en cada iteración.

Sin embargo, el nuevo algoritmo no corresponde a la solución cerrada del problema, es decir que la solución no es exacta y el algoritmo puede diverger por problemas vinculados a las linealizaciones realizadas en los modelos.

#### 2.6.2.3. Costo computacional

El costo computacional de la actualización del Filtro de Kalman se puede dividir en dos componentes:

- Un costo asociado a la inversión de una matriz relacionada con la matriz de sensado $H$. Utilizando algoritmos eficientes de inversión, esta operación puede realizarse en $O(k^{2,4})$, donde $k$ es la dimensión del vector de la medida $z_i$
- Un costo asociado a la multiplicación de dos matrices de dimensión $n \times n$, donde $n$ es la dimensión del estado oculto, en la práctica aproximadamente proporcional a la cantidad de marcas. Esta operación tiene un costo de $O(n^2)$

En la práctica, el segundo costo suele dominar al primero, debido a que la dimensionalidad del vector de sensado suele ser mucho más pequeño que la cantidad de marcas. Por lo tanto el algoritmo suele tener un costo computacional de $O(n^2)$.

### 2.6.3. Filtros de Partículas

Los filtros de partículas intentan aproximar la distribución de probabilidad del estado del sistema $x_t$ utilizando Métodos de Montecarlo. Para esto, mantienen un conjunto de partículas que son $N$ muestras (*samples*) de la distribución de probabilidad del estado del sistema $x_i$. Cada una de estas partículas mantiene un estado concreto del sistema, es decir una ubicación del robot y un mapa. El conjunto de estos estados concretos representa la distribución de probabilidad del estado del sistema.

Al igual que el Filtro de Kalman, actualizan esta distribución a medida que se encuentra disponible nueva información de sensado u odometría. Es decir que los filtros de partículas pertenecen a la categoría de Online SLAM.

#### 2.6.3.1. Actualización de la distribución de probabilidad

La dinámica de actualización de la función de densidad del filtro de partículas es similar al del Filtro de Kalman (ver sección 2.6.2). Esta actualización consta de dos pasos, el de predicción y el de actualización

propiamente dicho.

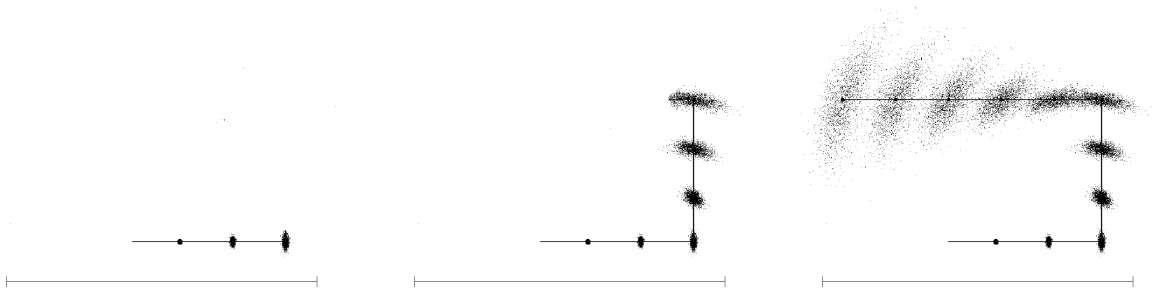
**Paso de predicción (*predict*)** El paso de predicción busca modificar la función de densidad para reflejar un movimiento realizado por el robot. Para esto, se modifica la posición de cada una de las partículas según:

$$x_t = f(x_{t-1}, u_t)$$

donde $f$ representa el modelo de movimiento a utilizar.

En la figura 2.6 se puede observar el efecto de aplicar sucesivamente el paso de predicción a un conjunto de partículas. Las imágenes muestran el efecto de sucesivos movimientos en línea recta seguidos de rotaciones.

Esta imagen ilustra como este paso del algoritmo aumenta la dispersión de las partículas, y por consiguiente la varianza de la distribución. Esto se debe a que el modelo de movimiento incluye una componente de ruido para modelar la naturaleza estocástica del movimiento del robot. Este ruido se va acumulando en cada paso de predicción y aumenta la incertidumbre de la estimación.



(a) Movimiento en línea recta

(b) Una rotación de 90 grados y un movimiento en línea recta.

(c) Otra rotación de 90 grados y un movimiento en línea recta.

Figura 2.6: Paso de predicción aplicado sucesivas veces.
Imágenes extraídas de [39]

**Paso de actualización (*update*)** Este paso busca ajustar la función de densidad a la última información de sensado recibida. Para esto se le asigna a cada partícula un peso $w_i$ proporcional a la verosimilitud de la información de sensado acorde al estado actual del sistema. Es decir

$$w_i \sim p(z_t|x_t)$$

Este peso indica, en resumen, cuán verosímil es el modelo representado por la partícula en función de la última información de sensado.

Luego, se realiza un proceso denominado remuestreo (*resampling*) en el que se extraen con reposición $N$ partículas del conjunto de partículas actuales. Siendo la probabilidad de seleccionar una partícula proporcional a su peso $w_i$.

El paso de actualización disminuye la incertidumbre de la estimación. En otras palabras, disminuye la dispersión de las partículas.

#### 2.6.3.2. El algoritmo

Reuniendo ambos pasos y colocándolos en una iteración infinita, obtenemos el algoritmo de filtro de partículas, según el algoritmo 2.1.

**Algoritmo 2.1** Algoritmo de filtro de partículas.

```
particulas = random(posicionInicialMedia, varianzaInicial)
por siempre:
        para cada particula p en particulas:
                p = modeloMov(p, u_t)
                peso = modeloSensado(p, z_t)

        particulas = resample(particulas)
```

Las partículas son inicializadas en torno a una posición inicial media arbitraria y una varianza que representa la incertidumbre inicial de la ubicación del robot.

#### 2.6.3.3. Costo computacional

El costo computacional del algoritmo de partículas es proporcional a la cantidad partículas utilizadas $M$, es decir $O(M)$.

En general se asume que el número de partículas necesario crece exponencialmente con la dimensión del estado a estimar $x_i$. Esta idea se deriva intuitivamente de la combinación de:

- Cada partícula representa un punto en el mapa para el estado oculto, y
- La cantidad de puntos posibles del mapa crece exponencialmente con la dimensión de este.

En conclusión, el costo computacional del algoritmo crece exponencialmente con la dimensión del espacio de búsqueda.

# Capítulo 3

# Desarrollo de un sistema de SLAM

## 3.1. Introducción

En este capítulo se presenta la solución de SLAM desarrollada. Se describen los modelos de sensado y movimiento y el filtro de partículas. Adicionalmente, se incluye una sección que describe la arquitectura del sistema implementado, y cómo esta se adecúa a los requerimientos no funcionales establecidos.

## 3.2. Algoritmo de SLAM

El Algoritmo de SLAM está compuesto principalmente por tres componentes: el núcleo encargado de realizar la estimación del estado oculto, el modelo de sensado y modelo de movimiento. En esta sección se presentan las características de cada una de estas tres componentes. Además, se hace una breve reseña de la forma en que se estima una posición concreta en base a la distribución mantenida.

### 3.2.1. Núcleo del SLAM

A continuación se describen cuatro componentes principales del sistema implementado.

#### 3.2.1.1. Partículas

El sistema de partículas encapsula toda la funcionalidad referente a los procesos de actualización basado en la observación *(update),* generación de nuevas partículas *(resampling)* y estimación del movimiento *(predict*), esenciales para los filtros de partículas[39].

Cada partícula mantiene un estimado de su posición utilizando tres coordenadas $(x, y, \theta)$ y una estimación del mapa, en particular cada una mantiene $K$ Filtros de Kalman (ver sección 2.6.2, en adelante FK) que estiman la posición de las $K$ marcas existentes.

La arquitectura de software implementada para el sistema de partículas se detalla en la sección 3.3.4.1.

#### 3.2.1.2. Mapa y Filtro de Kalman

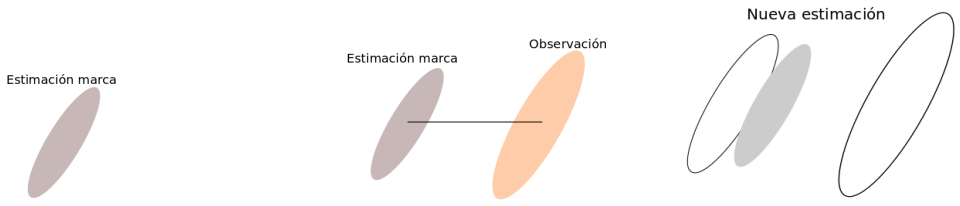
Como se explicó anteriormente, el sistema de SLAM mantiene un modelo interno del entorno en el que se mueve utilizando un mapa formado por marcas. En este modelo, cada marca es un FK, que mantiene una distribución gaussiana de dos dimensiones. Formalmente se modela cada marca como:

- Un vector con la posición $(x, y)$ de la marca
- Una matriz de covarianza asociada al error de la estimación de la posición de la marca

La matriz de covarianza es inicializada con la matriz de covarianza del error de observación (ver sección 3.2.3).

Como la estimación de las marcas está condicionada por la estimación del recorrido del robot, cada partícula tiene un FK para cada marca existente en el entorno. Esto difiere del enfoque tradicional de FK en el cual se mantiene una matriz de $K * K$ (siendo $K$ el número de marcas) donde luego de una nueva observación, el orden del costo de actualización de la matriz es $O(K^2)$. Luego, en nuestro caso, para $M$ partículas y $K$ marcas habrá $M * K$ FK, por lo cual el costo de actualización es $O(MK)$. Esto es posible debido a que la estimación de las marcas depende solamente del recorrido del robot y no hay dependencia entre las marcas[34].

Cada actualización del mapa (observación de una nueva marca o de una marca ya existente en el sistema) se realiza a través de un FK que se encarga de combinar la estimación actual de la marca con la nueva observación, ponderando según la varianza de cada gaussiana. En caso de que la marca observada no tenga una estimación anterior, se la da de alta en el sistema con un bajo nivel de confianza (un valor alto en la covarianza). En la figura 3.1 se puede observar como actúa el FK implementado frente a una nueva observación.




(a) Ejemplo de estimación gaussiana de una marca.

(b) Nueva observación de la misma marca realizada por el robot. El óvalo de observación es más grande debido a que la observación posee una incertidumbre mayor a la estimación actual.

(c) Integración de la observación con la estimación de la marca. La media de la gaussiana resultante se encuentra más cerca de la gaussiana de menor covarianza. La covarianza resultante es menor que la más chica de las gaussianas combinadas.

Figura 3.1: Etapa de actualización de una marca a través del FK.

El FK logra que, tras sucesivas observaciones de la misma marca, la covarianza disminuya, es decir que aumenta la confianza en la estimación de la marca.

#### 3.2.1.3. Proceso de *Resampling*

Este proceso (como se mencionó en 2.6.3.1) permite ajustar la estimación a la última observación realizada por el robot. Para ello se dispone de un método en la clase presentada en la subsección 3.3.4.1 destinado a realizar el *resampling*. Este algoritmo tiene una relevancia importante ya que es el encargado de "decidir" que partículas se mantienen (sobreviven) y cuales desaparecen (mueren), acción que puede tener resultados muy negativos en caso de ser realizada en forma incorrecta. En la implementación presentada, esta operación es ejecutada cada vez que el modelo de sensado obtiene, con cierto nivel de confianza, una nueva medida de una marca. En 3.1 se presenta un seudocódigo del algoritmo de *resampling* implementado.

**Algoritmo 3.1** Seudocódigo del proceso de resampling.

```
pesoTotal = suma del peso de todas las partículas
si pesoTotal == 0
        para cada partícula p del conjunto
                p.peso = 1
        fin (no realizar el remuestreo)
sino
        normalizar el peso
        calcular la varianza de los pesos
        si varianza > parámetro de resampling
                nuevasPartículas = vacío
                hasta completar la cantidad máxima de partículas
                        sortear una partícula en proporción al peso
                        realizar una copia
                        agregar(copia, nuevasPartículas)
        partículas = nuevasPartículas
fin
```

Este seudocódigo comienza realizando un control de que el peso de las partículas sea mayor que cero. En caso contrario no se realiza el resampling y se le asigna peso 1 a todas las partículas. Esto se debe a que es necesario mantener un valor superior a cero en el peso para llevar a cabo los cálculos con partículas sin error, sobretodo divisiones.

Luego, se obtiene la varianza de los pesos normalizados y se compara con el parámetro de resampling. Esto se debe a que no siempre es útil realizar el paso de resampling. El paso de *resampling* sólo debe llevarse a cabo cuando se espera que esto implique un cambio sustancial en la estimación de la posición[20]. En general esto ocurre cuando las partículas están muy dispersas, y los pesos asociados a la última observación presenta una gran varianza.

Si se cumple la condición de resampling, se realiza un sorteo, en el cual se eligen $N$ partículas con reposición y probabilidad proporcional al peso. Esto último se debe a que las partículas con mayor peso son las que se adaptan mejor la última observación realizada por el robot.

#### 3.2.1.4. Proceso de *predict*

La implementación del proceso de predict se realiza iterando sobre las partículas, aplicando a cada una un cambio determinado por el modelo de movimiento y la información de odometría recibida.

En el seudocódigo 3.2 se observa las operaciones realizadas sobre cada partícula. Se incluye como último argumento de las funciones avanzar y rotar, dos números al azar obtenidos de distribuciones gaussianas. Estas distribuciones dependen de los parámetros del modelo de desplazamiento.

**Algoritmo 3.2** Seudocódigo del proceso de predict.

```
para cada partícula p
        avanzar(p,u_t, randomGauss(parametro desplazamiento))
        rotar(p, u_t, randomGauss(parametro rotación))
```

### 3.2.2. Estimación de la posición

Para realizar la estimación de la posición, el algoritmo implementado realiza una suma ponderada de la posición de cada partícula. Es decir, que la posición estimada sería:

$$x_{pos} = \frac{\sum_1^N w_i x_i}{N}$$

Este modelo representa una simplificación del utilizado realmente. El modelo real realiza un promedio por separado de las componentes del vector $x_i$. Esto se debe a que la componente de orientación del robot debe ser promediada de una forma particular, debido a que toma valores en un espacio con una discontinuidad en el valor 180 (*manifold*).

El promedio de la orientación se realiza promediando vectores de dos dimensiones, que forman un ángulo $\alpha$ con el eje de las abscisas y tienen un módulo igual al peso de la partícula correspondiente. Esto es:

$$v_{rot} = \frac{\sum_1^N w_i(cos(rot_i), sin(rot_i))}{N}$$

$$rot = atan(v_{rot})$$

donde *atan* toma un vector y retorna el ángulo que este forma con el eje $x$.

Esta práctica no tiene fundamento teórico debido a que los pesos $w_i$ refieren a la verosimilitud de una partícula dada al momento de realizado el último sensado. Este peso es asignado para realizar el paso de *resampling* que convierte la distribución propuesta en la distribución correspondiente a la ubicación del robot y el mapa. Luego de realizado el paso de *resampling*, este peso $w_i$ pierde validez.

Sin embargo, dado que la representación de la distribución de probabilidad se realiza con un conjunto de elementos discretos, los autores entienden que la utilización de esta práctica mejora el estimado de la media de la distribución real.

### 3.2.3. Modelo de sensado

El modelo de sensado supone conocidas la distancia y ángulo relativos a una marca sensada. Luego, puede tomarse la posición estimada de la marca como valor sensado. Es decir que se considera un metasensor que proporciona las coordenadas absolutas de la marca, calculándolas a partir de los datos de distancia y ángulo relativos, y la estimación de la posición absoluta del robot.

Entonces, el modelo de sensado, es decir la función que establece la probabilidad de sensar un dato, dado el mapa y la posición del robot, se estableció como una distribución gaussiana en torno a la estimación actual de la marca.

Adicionalmente, dado que la posición de las marcas se estima también utilizando una distribución gaussiana, se estableció como modelo de sensado la convolución de la gaussiana de sensado con la de estimación de la marca. Haciendo esto, se toman en cuenta los parámetros de varianza de la estimación de la marca y no solo la media como dato.

La utilización de esta distribución gaussiana como modelo de sensado agrega un parámetro al sistema, correspondiente a la varianza de esta. Su media no representa un parámetro, porque vale cero, debido a que la gaussiana siempre se concentra en la ubicación de la marca.

### 3.2.4. Modelo de movimiento

Se implementó un modelo de movimiento similar al presentado en la sección 2.5, que contempla dos fuentes de ruido:

- Ruido gaussiano relacionado al desplazamiento lineal
- Ruido gaussiano relacionado a la rotación del robot

Ambas fuentes de ruido dependen proporcionalmente de la magnitud del movimiento realizado.

Cada fuente de ruido tiene un parámetro asociado. Este parámetro indica la varianza de la distribución gaussiana. Cuanto mayor sea el parámetro, mayor será la varianza del ruido generado en el desplazamiento. Este parámetro fue relevado de forma empírica, esto implica que:

- El sistema será más robusto ante errores cometidos en el desplazamiento ya que en el modelo se tienen en cuenta los errores "mecánicos" cometidos por el robot al desplazarse en el escenario
- La dispersión de las partículas tenderá a aumentar proporcionalmente al valor de los parámetros mencionados

El modelo refleja el error real cometido por el robot en sus movimientos.

El tercer parámetro asociado a la rotación final no fue utilizado, debido a que se supone que los movimientos realizados por el robot siempre constan de rotaciones o traslaciones, pero nunca describen arcos.

## 3.3. Arquitectura

### 3.3.1. Introducción

En esta subsección se desarrollan en profundidad las características más relevantes de la arquitectura del sistema robótico implementado. Se presenta una breve descripción de los módulos más importantes y finalmente se explica como se organiza la arquitectura.

### 3.3.2. Requerimientos de la arquitectura

Como se puede inferir del problema a resolver, el sistema tiene el requerimiento principal de realizar localización continua sobre un mapa generado por el propio sistema en forma simultánea, basándose en la información resultante del sensado de las marcas dispuestas en el entorno.

Además, se establecieron requerimientos no funcionales para el sistema, que se detallan a continuación:

- generar una plataforma flexible
- ejecutar el sistema embebido en el robot
- desacoplamiento de módulos
- fácil depuración
- experimentos reproducibles

Estos puntos se desarrollan a continuación.

#### 3.3.2.1. Generar una plataforma flexible

Se espera que la arquitectura implementada permita reutilizar el sistema de SLAM generado, permitiendo intercambiar los módulos (ej. módulo de sensado o movimiento) sin alterar el resto del sistema.

#### 3.3.2.2. Ejecución embebida en el robot

Dado que el sistema a implementar pretende ser base para el desarrollo de un robot autónomo, este se debe ejecutar por completo en el robot.

#### 3.3.2.3. Desacoplamiento

Teniendo en cuenta que el sistema de SLAM es parte de un sistema de navegación (que es una actividad deliberativa[35]), consideramos importante desacoplar el sistema generado de los sistemas de operación básicos del robot. Esto permitiría la implementación de arquitecturas regidas por la organización del software que propone el paradigma híbrido.

Específicamente, se desea que el sistema sea capaz de mantener un estimado de la posición independientemente de la actividad desarrollada por el robot, sin que este tenga que modificar su comportamiento para que el sistema de SLAM funcione correctamente, ni que el comportamiento más rudimentario del robot precise del estimado proporcionado por el sistema de localización para funcionar.

#### 3.3.2.4. Depuración

Para contemplar la complejidad del proceso de depuración del sistema de localización a implementar, se desea que la arquitectura permita el envío de información de depuración a una PC utilizando protocolos inalámbricos.

A través de este protocolo, se realiza el envío de los comandos que son procesados y ejecutados en el robot (movimientos realizados y detección de marcas) a una PC.

#### 3.3.2.5. Facilitar la investigación

Dado que el desarrollo forma parte de un proyecto de investigación, la arquitectura del sistema desarrollado debe facilitar la investigación de las técnicas utilizadas en el SLAM.

Se busca:

- Facilitar la reproducción de experimentos para comprender el funcionamiento del sistema con mayor sencillez.
- Simplificar los procesos de optimización del sistema, como la parametrización y estudio del uso de diferentes opciones utilizadas en la literatura del SLAM.

### 3.3.3. Diagrama de arquitectura

Para mayor comprensión del sistema a desarrollar, se incluye un diagrama de componentes de la arquitectura en la figura 3.2.

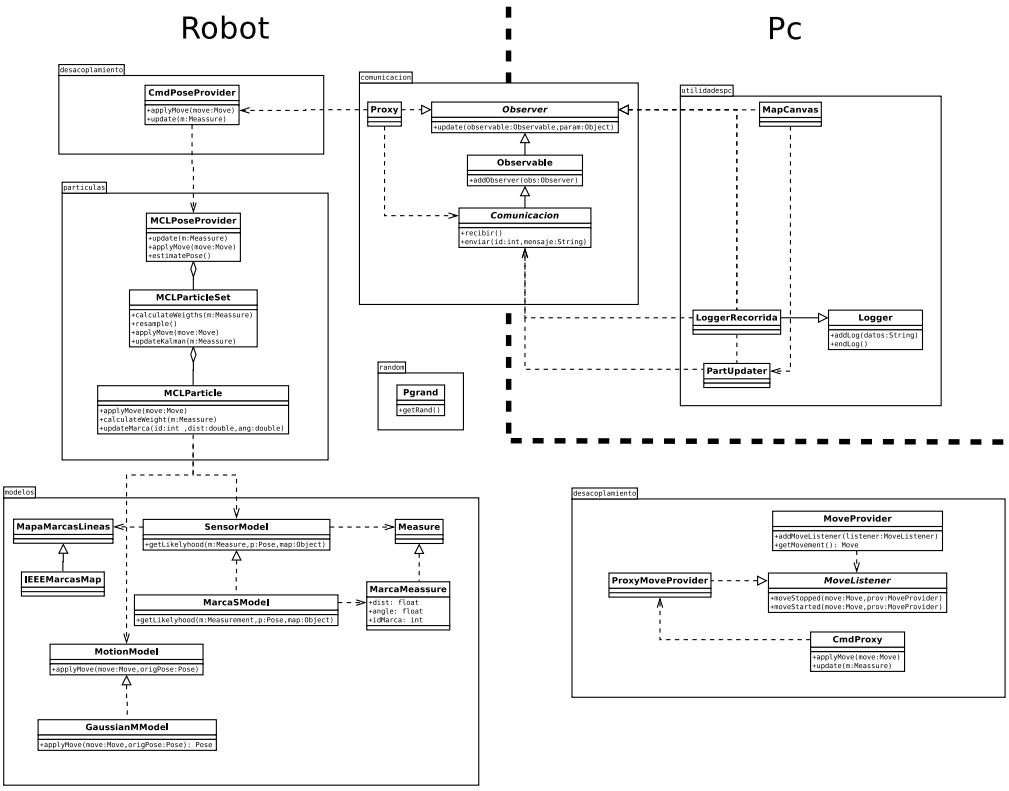



Figura 3.2: Diagrama de componentes del sistema.

### 3.3.4. Subsistemas

El sistema se divide en los siguientes subsistemas, correspondientes a las componentes presentadas en el diagrama de clases:

- Partículas
- Modelos
- Desacoplamiento
- Comunicación
- Random

Todos estos subsistemas se explican en detalle a continuación.

#### 3.3.4.1. Partículas

Este subsistema pertenece al núcleo del SLAM y mantiene la estimación de la posición del robot y del mapa. Está compuesto de tres clases con tareas bien diferenciadas: MCLPoseProvider, MCLParticleSet y MCLParticle, como se puede observar en la figura 3.3.

La lógica relacionada con los modelos de sensado y movimiento es delegado a otro subsistema, haciendo al filtro más genérico y reutilizable.

Este subsistema se implementó tomando como base la implementación de partículas de LeJOS[11]. En la figura 3.3 se aprecian las tres clases que conforman el filtro de partículas con las operaciones más relevantes de cada una de ellas.

A continuación se describen las clases que componen el sistema de Partículas.

***MCLPoseProvider*** Esta clase es la interfaz del sistema. Centraliza las funcionalidades de interacción, como el ingreso de información de odometría y sensado. También centraliza los cálculos de posiciones a partir de las partículas actuales.

***MCLParticleSet*** Esta clase administra el conjunto de partículas. Centraliza los procesos de iteración sobre las partículas, necesarios en los pasos de *update* y *predict*. También se encarga de serializar[1] el conjunto de partículas para ser transmitido, funcionalidad implementada acorde al requerimiento 3.3.2.4 para enviar las partículas como información de depuración.

***MCLParticle*** Esta clase mantiene una referencia a la posición estimada, la estimación del mapa y los modelos de sensado y movimiento. Encapsula la funcionalidad de las operaciones necesarias por partícula para las operaciones de *update*, *predict* y *updateMarca* (utilizada para la actualización del mapa).

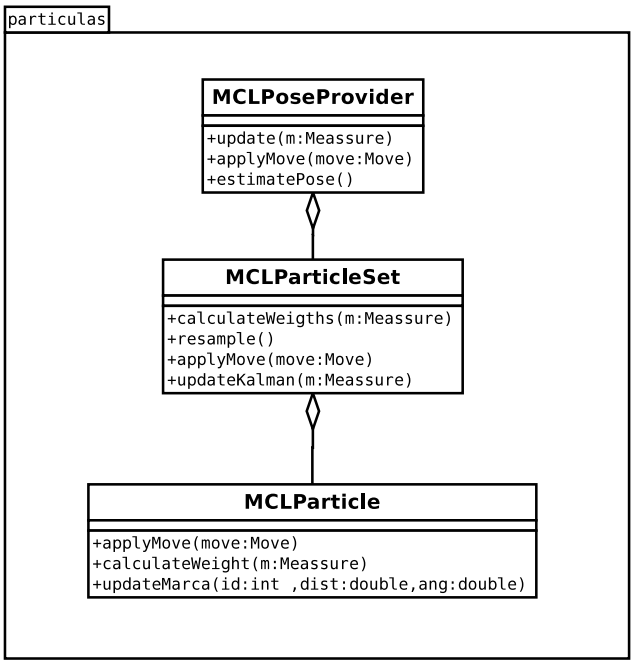



Figura 3.3: Subsistema de partículas.

#### 3.3.4.2. Modelos

Este subsistema encapsula la lógica de los modelos de sensado y movimiento. Se implementó de forma genérica para que pueda ser reutilizado en el sistema acorde al requerimiento 3.3.2.1.

[1] El proceso de serialización permite convertir una estructura de datos en un arreglo de bytes para ser transmitido por un canal de comunicación.

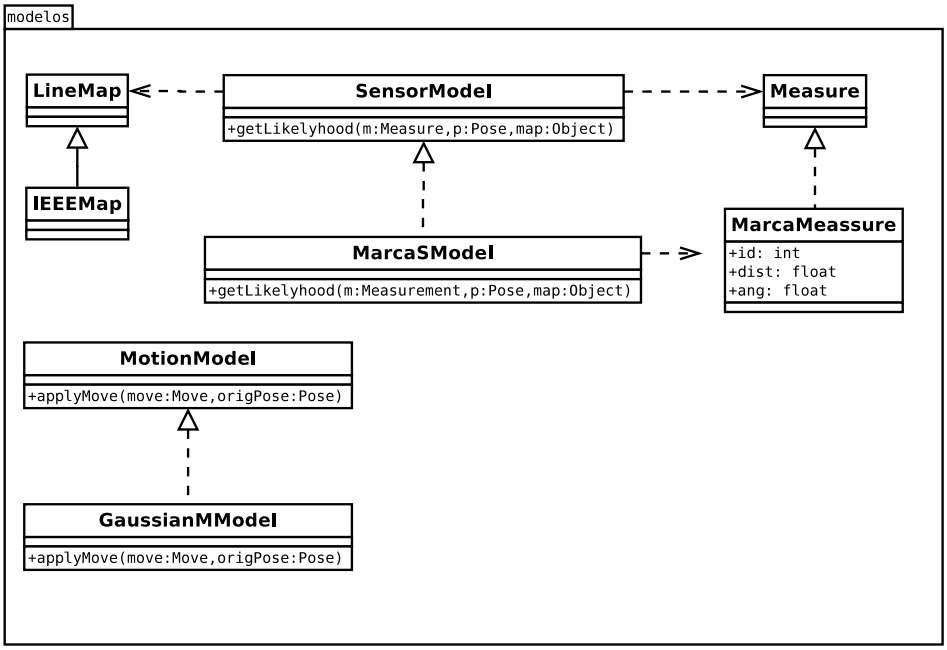



Figura 3.4: Subsistema de modelos de sensado y movimiento.

Además del subsistema genérico de modelos, se implementaron dos modelos específicos para la solución de SLAM implementada:

***GaussianMModel*** Un modelo de movimiento que toma como entrada la posición y el movimiento realizado por el robot, devolviendo como salida una muestra de la distribución de probabilidad de la nueva posición del robot. Este tema fue tratado en la sección 3.2.4.

***MarcaSModel*** Un modelo de sensado que utiliza la información de la marca sensada, la posición del robot y el mapa, para determinar la verosimilitud de la posición estimada con respecto a lo sensado. Este tema fue tratado en la sección 3.2.3.

#### 3.3.4.3. Desacoplamiento

Este subsistema se encarga de las funcionalidades de desacoplamiento entre el módulo de partículas y el de comportamiento del robot.

Para esto se implementó un sistema de comandos (Patrón de diseño *Command*) que son ejecutados de forma asincrónica (ver figura 3.5). Estos comandos se corresponden a los pasos de *update* y *predict*. Cuando el robot sensa información o termina un movimiento, avisa al sistema de desacoplamiento de esta información, quien encola los comandos correspondientes, para su posterior procesamiento.

Las clases de MoveProvider se encargan de reportar movimientos de forma asincrónica, que son serializados por la clase CmdProxy y enviados a la clase CmdProvider.

#### 3.3.4.4. Comunicación

El subsistema de comunicación fue implementado para cumplir con el requerimiento 3.3.2.4. Como se puede ver en la figura 3.6, el sistema utiliza patrones *Observer* para detectar cambios en los objetos de

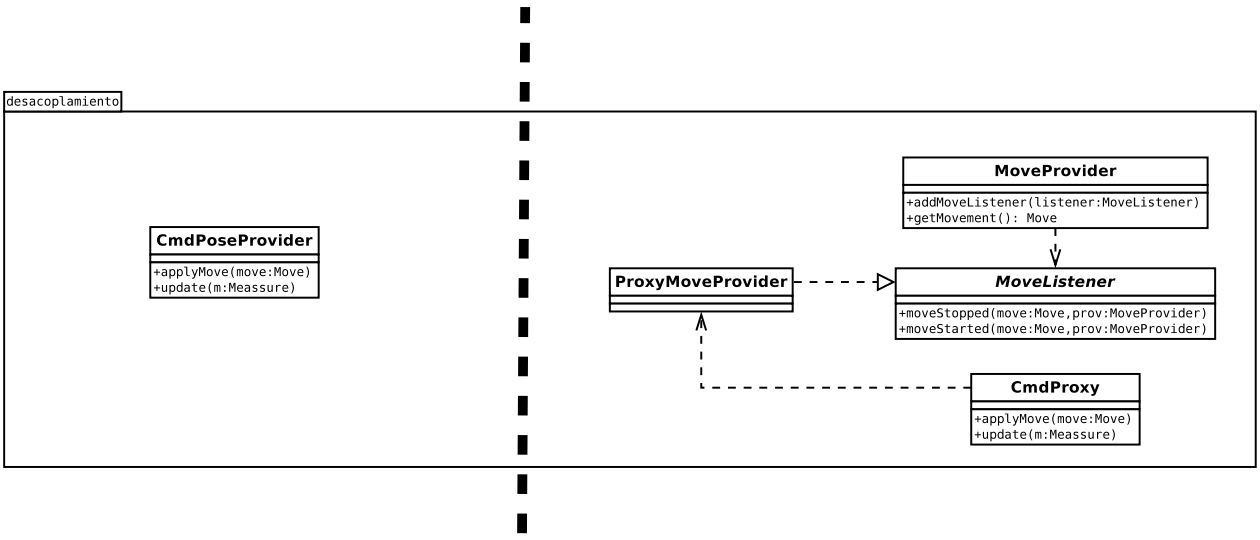



Figura 3.5: Subsistema de desacoplamiento.

interés, serializarlos y enviarlos hacia la PC.

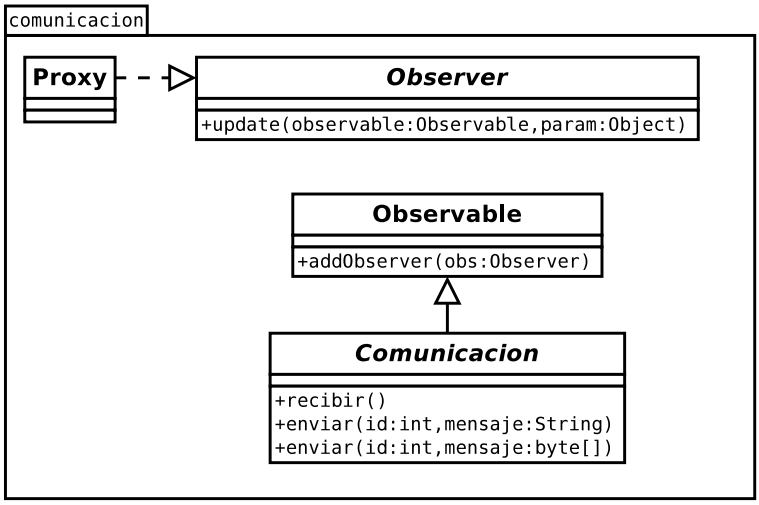



Figura 3.6: Subsistema de comunicación.

Se implementó el sistema buscando centralizar todas las operaciones comunes al envío y recepción de información en una misma clase, que luego es heredada por cada clase específica, para la comunicación entre componentes arbitrarios del sistema.

Para serializar los objetos se implementó una operación que trabaja sobre un arreglo de bytes, en pos de un uso eficiente del flujo de datos sobre el USB.

**Utilidades PC** Este subsistema recibe datos del robot y los utiliza para diferentes funciones (ver figura 3.7):

- Desplegar las partículas actuales sobre el mapa del entorno (*MapCanvas* y *PartUpdater*). En la figura 3.8 es posible observar un conjunto de partículas graficadas (rojo) sobre el mapa del entorno. La cruz verde representa la última posición obtenida del sistema de visión.

- Guardar los datos de los comandos del *CmdPoseProvider* para uso posterior en los procesos de parametrización (*LoggerRecorrida*).

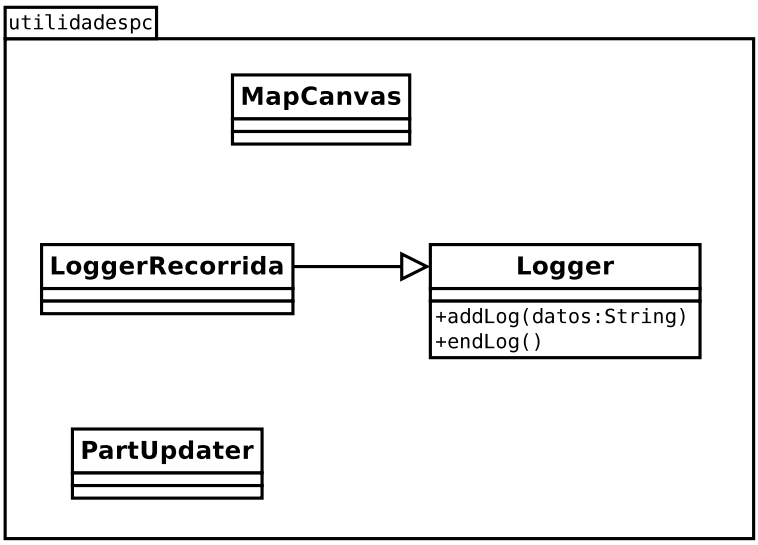



Figura 3.7: Subsistema de utilidades de la PC.

#### 3.3.4.5. *Random*

Este subsistema se implementó acorde al requerimiento 3.3.2.5. Posee una única clase y tiene como cometido centralizar la generación de números aleatorios. Mediante esta centralización se torna más fácil el proceso de reproducción de experimentos, utilizando una única semilla (*seed*) centralizada y conocida. Esta semilla es utilizada en la generación y evaluación de las partículas.

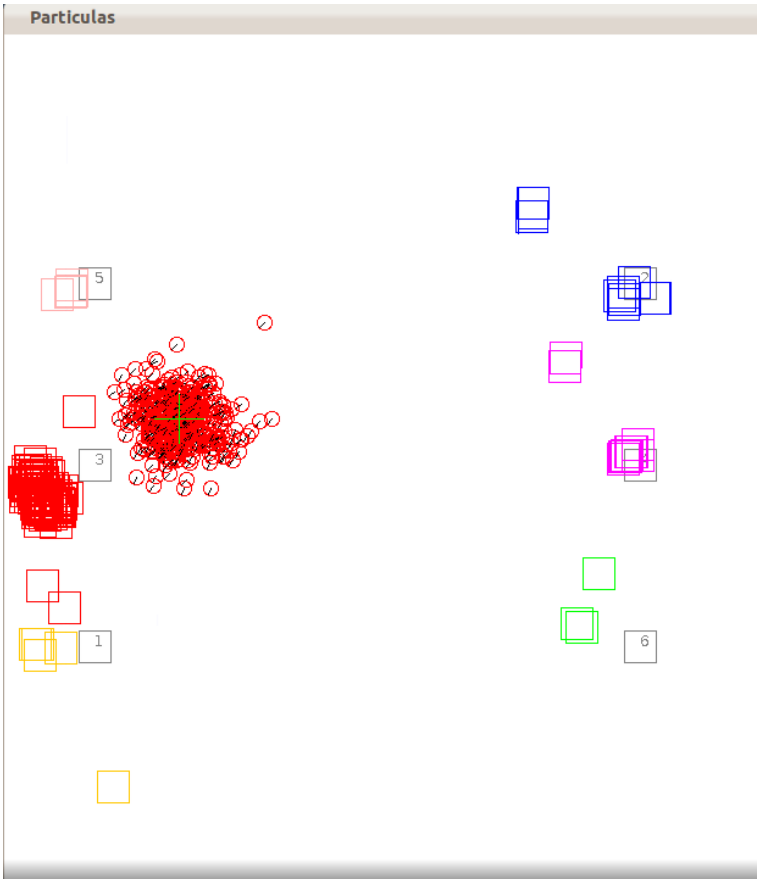



Figura 3.8: Interfaz de una aplicación del subsistema de utilidades de la PC, utilizada para depuración.

# Capítulo 4

# Caso de estudio

## 4.1. Introducción

En este capítulo se presenta un problema abordado como caso de estudio y una solución robótica que utiliza un sistema de SLAM sobre un entorno controlado. Se describe el hardware, diseño mecánico y el software de control de este sistema.

Además, en este capítulo el lector encontrará una sección dedicada a estudiar el caso de una ejecución de la solución implementada, destinada a facilitar la comprensión de todos los elementos presentados y su interacción.

Finalmente se incluye una sección de análisis de los métodos de calibración del sistema de SLAM considerados.

## 4.2. El problema

Se plantea el problema de recorrer de forma exhaustiva un terreno rectangular, de dimensiones desconocidas para el robot. Se busca solucionar el problema general de encontrar objetos que están distribuidos en el entorno en forma aleatoria y que pueden ser sensados únicamente a corta distancia. A partir de esta propuesta, el robot debe recorrer el entorno realizando el mayor cubrimiento posible del mismo. Además, el robot debe evitar salir de los límites del entorno, delimitados por paredes, acción por la cual será penalizado.

Este problema aparece con frecuencia en la industria. Se pueden citar ejemplos como la limpieza completa de un cuarto o la recorrida de una plantación para cosecha, entre otras actividades.

### 4.2.1. Entorno

El entorno en el cual se mueve el robot es un rectángulo de 2,0m. de ancho por 2,4m. de largo, de superficie lisa, delimitado por paredes blancas de 60cm. de alto. En el mismo, se encuentran distribuidas seis marcas perfectamente distinguibles por el robot. Estas marcas fueron colocadas cerca de los límites del ambiente de trabajo con el fin de facilitar la navegación del robot. En la figura 4.1 se puede observar una vista aérea esquemática del entorno donde ejecutará el robot.

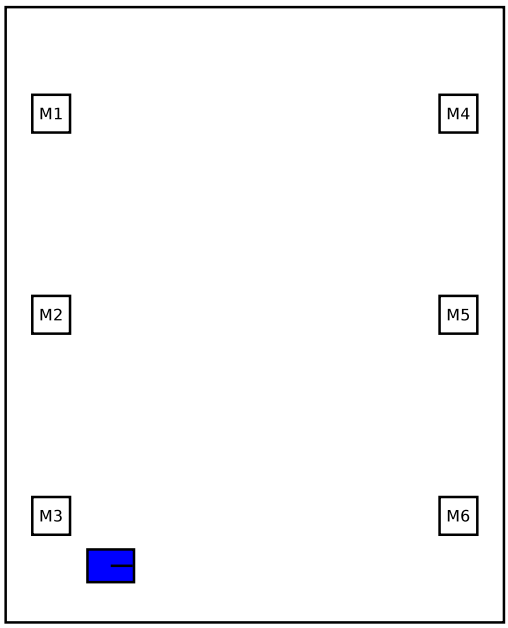



Figura 4.1: Entorno de pruebas del robot.
Los cuadrados etiquetados M1..M6 representan a las marcas. El rectángulo azul representa al robot en su posición inicial y la línea en su interior apunta en la orientación del robot. El rectángulo exterior representa las paredes que delimitan el entorno.

### 4.2.2. Marcas

Las marcas son prismas de base cuadrada utilizadas como referencias para el robot. Cada marca mide 15cm. de alto por 5cm. de lado. Posee tres colores diferentes, violeta, naranja y verde, como se explicó en la sección 3.2.3, y se diferencian entre sí por la combinación vertical de dichos colores. Cada color ocupa 1/3 de la altura de la marca.

Luego de varios experimentos combinando tipos de luz y materiales para las marcas, se determinó que para disminuir el brillo y permitir un mejor sensado, la creación de marcas de goma eva resultaba la mejor opción.

### 4.2.3. Iluminación

Se determinó que la combinación de luz fluorescente y luz incandescente resultaba la mejor opción para el mejor funcionamiento de la cámara (ver subsubsección 4.3.3.1).

Se utilizaron tres focos incandescentes de 500 watts, que proporcionaban el nivel de luz necesario para que la detección de las marcas no presentara mayores problemas. La utilización de goma eva para forrar las marcas disminuyó el efecto negativo de quemado y enrojecimiento de los colores producido por la luz incandescente de alta intensidad.

Adicionalmente a los tres focos incandescentes, se utilizó la iluminación presente en el laboratorio, de fuentes fluorescentes. Esta iluminación adicional contribuyó a la eliminación de sombras en el entorno. En combinación con la presencia de luz incandescente intensa, la luz fluorescente no presentaba las desventajas mencionadas anteriormente.

### 4.2.4. Límites

Se colocaron paredes blancas de 60cm. de alto de *Carton Plast* rodeando el entorno para mejorar el rendimiento de la cámara[1] aislando objetos externos al entorno de trabajo. Se busca disminuir la probabilidad de detectar objetos que no sean marcas (falsos positivos), y que la presencia de objetos extraños pueda afectar la proporción de falsos negativos.

## 4.3. Hardware

El sistema robótico construido se compone de cinco tipos de componentes de hardware diferentes, estos son:

- 1 Placa de entrada/salida basada en microcontrolador (en adelante, *ladrillo*) LEGO NXT[1]
- 1 Placa computadora FoxBoard G20[4]
- 2 Sensores de ultrasonido
- 1 Cámara
- 3 Motores

Las características y funciones más relevantes de los componentes utilizados se describen a continuación.

### 4.3.1. Ladrillo LEGO

En el ladrillo LEGO se ejecuta el sistema de navegación. Además, los sensores y actuadores utilizados por el agente robótico se encuentran conectados a este, como se describe en la sección 4.4. Los aspectos más sobresalientes del ladrillo LEGO son:

- Procesador: Atmel® 32-bit ARM® processor, AT91SAM7S256
- Frecuencia de reloj: 48MHz
- RAM: 64KB
- FLASH: 256KB
- Comunicación: Bluetooth y USB
- Puertos entrada (sensores): 4
- Puertos salida (actuadores): 3

### 4.3.2. Placa Computadora FoxBoard G20

Una placa computadora es una computadora completa en un sólo circuito impreso. La placa computadora Fox Board G20 cuenta con un microprocesador, memoria flash, memoria RAM, puertos de entrada/salida, ethernet, entre otras prestaciones. Es similar a una placa madre de un computador estándar, con la salvedad de que suelen fabricarse en tamaños reducidos. Las principales características de dicha placa son:

[1] Se utiliza una cámara como sensor para detectar las marcas. Esto se explica en la sección 4.3.3.1

- Procesador: Atmel AT91SAM9G20 ARM9 CPU
- Frecuencia de reloj: 400Mhz
- RAM: 64MB
- Flash: 8MB
- USB *host*: 2
- USB *device*: 1
- Ethernet conector
- Puertos de entrada/salida: 80

### 4.3.3. Sensores y actuadores

El robot cuenta con tres sensores y tres actuadores. Dos de los sensores son ultrasónicos y son utilizados para detectar la proximidad a las marcas y a eventuales obstáculos. El tercer sensor, una cámara, conforma la base del sistema de visión. Por otro lado, dos de los tres actuadores brindan movilidad al robot, mientras que el tercero se ocupa de la rotación de la cámara. Estos componentes se describen a continuación.

#### 4.3.3.1. Cámara

La cámara utilizada es una NXTCam-v3[13], que pertenece al grupo de sensores avanzados del Kit LEGO NXT[1]. La cámara posee la capacidad de realizar un preprocesamiento de imagen en el propio firmware. Este firmware reporta sólo los objetos binarios más grandes (en inglés Binary Large Objects, en adelante BLOBS). Esto ocurre debido a que la cámara ordena los BLOBS por tamaño y existe un máximo de 8 BLOBS reportados al mismo tiempo.

Las características más destacadas de este sensor son:

- Capacidad de distinguir hasta 8 objetos definidos por el usuario de diferentes colores
- Lentes construidos con filtro infrarrojo, que permiten mayor tolerancia a los cambios de luz
- Funciona a 30fps

#### 4.3.3.2. Ultrasonido

El sensor de ultrasonido se utiliza para medir distancias. Este sensor emite una señal de audio de alta frecuencia y cuenta el tiempo que demora la misma en volver al sensor. Existe una correspondencia entre el tiempo medido y la distancia al objeto más cercano. La especificaciones del sensor son:

- Es capaz de medir en un rango de 0 a 255cm.
- Precisión +/- 3cm.
- Mide distancias en centímetros o pulgadas

#### 4.3.3.3. Motores

En nuestro caso se utilizan motores del Kit LEGO NXT. Estos motores son del tipo servo que permiten rotación continúa. Además, cuentan con un sensor interno que permite conocer el ángulo relativo que se movió el motor desde su posición inicial.

## 4.4. Diseño mecánico

Esta sección está destinada a presentar al lector los componentes utilizados en la construcción del robot y la estructura del mismo.

### 4.4.1. Componentes

A continuación, se listan en forma completa, todos los elementos físicos que forman parte del sistema robótico:

- 1 Ladrillo del Kit LEGO NXT v2.0
- 1 Placa computadora FoxBoard G20
- 2 Sensores de ultrasonido LEGO
- 2 Ruedas LEGO con llantas de plástico y cubierta de goma
- 1 *Rueda loca* de plástico
- Piezas (bastones, uniones, engranajes) del Kit LEGO NXT v2.0
- 3 Motores del Kit LEGO NXT v2.0
- 1 Dispositivo USB para conexión wireless

### 4.4.2. Estructura

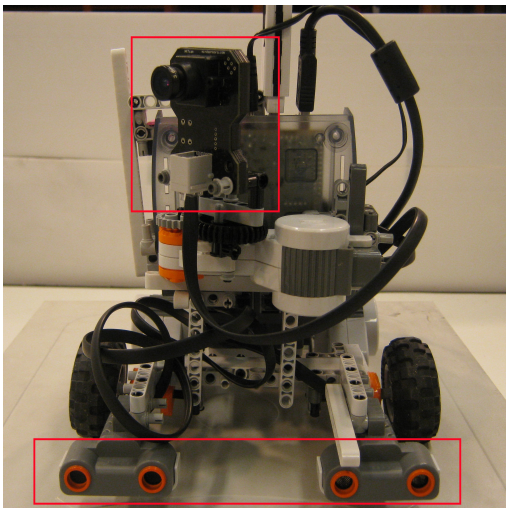
El robot fue diseñado con forma de carrito, utilizando piezas del Kit LEGO NXT v2.0. El mismo, está contenido en un prisma de 26cm. de largo por 20cm. de ancho por 24cm. de alto. El robot posee un eje principal sobre el que están montados dos motores, y a cada uno de estos se acopla una rueda de goma. Una tercera rueda compone el tercer vértice del triángulo de apoyo. Esta última rueda no está acoplada a ningún motor, ejerciendo solamente la función de apoyo (Rueda Loca).

En la figura 4.2 se pueden observar cuatro vistas del robot desde diferentes ángulos. En ella se observa la disposición de los elementos que componen al robot presentados en 4.4.1. Los elementos más pesados del robot (ladrillo, batería, motores y placa computadora) fueron colocados lo más centrado y cerca del eje posible, de forma de llevar el centro de masa hacia el eje de las ruedas motorizadas. Esto permite al robot realizar rotaciones sobre si mismo, en lugar de describir arcos.

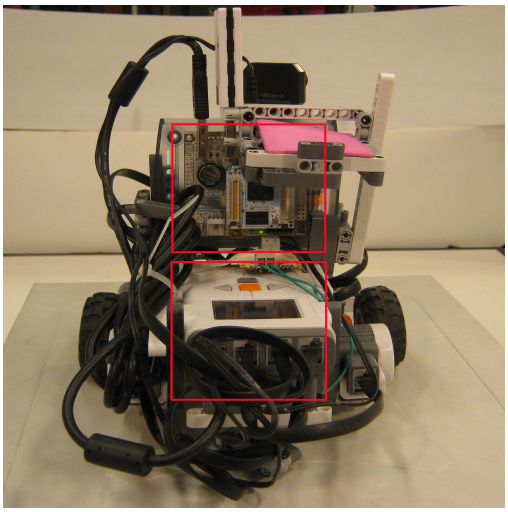
Al frente del robot se colocó una cámara montada sobre un eje motorizado que permite su rotación, de forma de tener un mayor ángulo de visión sin la necesidad de realizar giros con el robot. La cámara fue colocada lo más alto posible en concordancia con el modelo de visión utilizado, según se detalla en la sección 3.2.3.

Finalmente, el robot posee sensores ultrasónicos colocados en la parte frontal inferior del mismo para detectar obstáculos como las marcas o los límites del entorno.

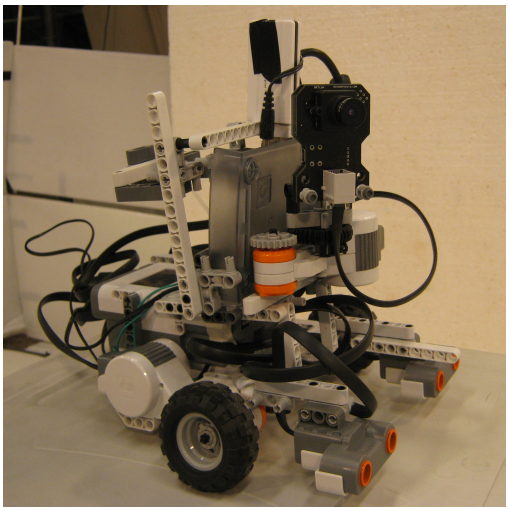
Se consideró que el tamaño del robot debería ser el menor posible para poder aprovechar más el tamaño del escenario de trabajo y lo suficientemente robusto para cargar con el hardware necesario para implementar SLAM.



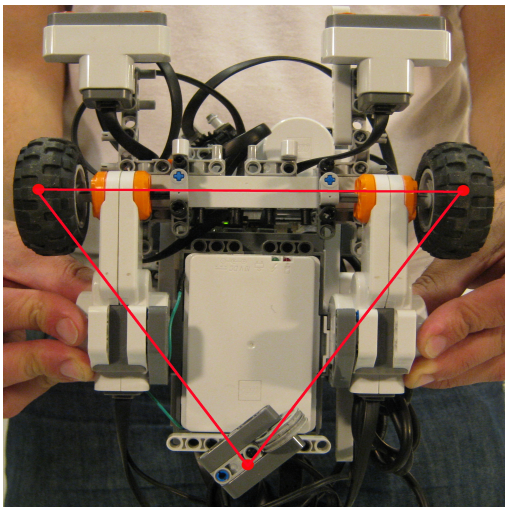
(a) Vista frontal del robot. En el recuadro superior se observa la cámara utilizada por el sistema de visión, mientras que en inferior se aprecian los dos sensores ultrasónicos.



(b) Vista trasera del robot donde se destaca la placa computadora (recuadro superior) y el ladrillo LEGO (recuadro inferior).



(c) Vista panorámica del robot.



(d) Vista inferior del robot donde se observa la disposición triangular de las ruedas.

Figura 4.2: Diferentes vistas del robot.

## 4.5. Software

A continuación se describen los mecanismos de instanciación del sistema de SLAM descrito en el capítulo anterior. Más adelante se describe el sistema de recorrido implementado para solucionar el problema de

cubrimiento.

### 4.5.1. Instanciación de los modelos

Para utilizar el sistema de SLAM en la solución del problema de cubrimiento, se implementaron componentes adicionales de software que proporcionan la información de sensado y odometría necesarias a los modelos correspondientes (sensado y movimiento).

#### 4.5.1.1. Información de sensado

Se implementó una componente de software destinada a la detección de marcas, y a la estimación de la distancia y ángulo relativos de las mismas.

Éste se divide en dos capas diferenciadas:

- La detección de las marcas
- La estimación de la ubicación de la marca

A continuación se describen las principales características de cada capa.

**Detección de las marcas** La cámara de LEGO presenta una gran componente de ruido en las mediciones que realiza. Además, la calidad de la imagen no permite un calibración precisa de los colores. En la figura 4.3 se observa una captura de la cámara obtenida utilizando el software de calibración.

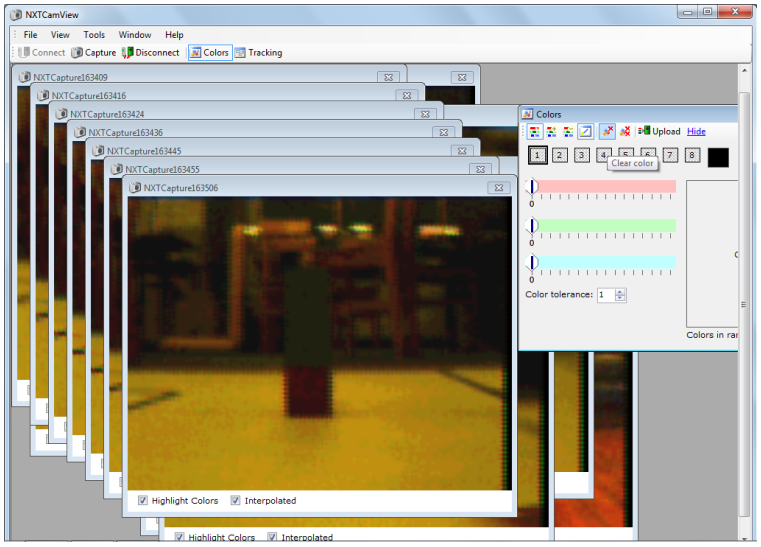



Figura 4.3: Captura de pantalla del software de calibración de la cámara.

Los problemas mencionados hacen difícil la detección de marcas (falsos negativos) debido al ruido, y también aumentan la probabilidad de detectar objetos que no corresponden (falsos positivos) debido a la imprecisión de la configuración de los colores.

Para solucionar estos problemas se diseñaron las marcas para que fueran fácilmente detectables por la cámara y que permitieran filtrar los falsos positivos.

Experimentalmente se determinó que a mayor cantidad de colores configurados para la detección en la cámara, mayor es la cantidad de falsos negativos en el reconocimiento de un objeto. Esto se debe a que la

calidad de la imagen proporcionada no permite distinguir bien un color de otro, y los colores configurados no pueden solaparse, lo que lleva al usuario a configurar los colores con muy poco margen de tolerancia. Luego, pequeños cambios en la iluminación pueden afectar el reconocimiento de los objetos.

**Diseño de las marcas** En consecuencia, se diseñaron las marcas para que se compusieran de tres objetos distinguibles, de tres colores diferentes. Estos colores son comunes a todas las marcas, de modo que solo es necesario configurar tres colores.

Cada marca consta de una pila de tres cubos, donde el orden de los colores la diferencia perfectamente del resto. El conocimiento *a priori* de esta información permite filtrar falsos positivos con mayor eficacia. Al buscar tres objetos de colores diferentes, perfectamente alineados, de forma rectangular y tamaño similar disminuye la probabilidad de detectar un falso positivo.

Por otro lado, la oscilación de la cámara y la baja calidad de la imagen, provoca, a veces, errores en la detección de los objetos, variando su tamaño en alguna dimensión, su posición, o ambos.

**Modelo bayesiano** Para integrar la información *a priori* de la forma de las marcas y contemplar las anomalías en el sensado de la cámara, se implementó un modelo inverso de sensado, como el utilizado en los sensores de distancia[35], que utiliza la regla de Bayes de la probabilidad condicional.

La idea de base es tomar uno a uno los objetos detectados como uno de los tres objetos de una potencial marca $A$. Luego, se establecen las distribuciones de probabilidades de los dos objetos restantes. Finalmente, se itera sobre el producto cartesiano de las particiones de objetos ordenadas por color, hallando la probabilidad de sensar cada conjunto, dado que la marca se corresponde a la marca $A$.

Los conjuntos de objetos que superan un umbral preestablecido de probabilidad son considerados marcas sensadas y son pasados a la capa siguiente. En caso de que sea más de una marca la que pasa a la siguiente capa, se actualiza cada observación de forma individual.

**Estimación de la ubicación de las marcas** Una vez detectada una marca, se estima la distancia y ángulo relativo al robot.

Esto se hace con modelos polinómicos aprendidos en base a datos de entrenamiento, según se hace en [30]. Esta solución se basa en los modelos de cámara Pinhole presentados en la sección 2.4. Para evitar la estimación de los parámetros intrínsecos de la cámara y presentar una robustez mayor ante la distorsión generada por los lentes, se utilizaron polinomios aprendidos en base a muestras de datos reales, en lugar de realizar los cálculos teóricos del modelo Pinhole.

Para la estimación de la distancia se utiliza la coordenada $y$ del píxel más bajo de la marca, que presenta un estimador más estable de la distancia que la medición del tamaño de los objetos[25].

La estimación del ángulo se realiza tomando como dato la coordenada $x$ del centro de la marca.

Para la determinación de los polinomios que mejor ajustan a los datos, se utilizó la función *polyfit* de Octave.

#### 4.5.1.2. Información de odometría

La información de odometría fue obtenida la clase DifferentialPilot implementada en LeJOS. Este sistema permite registrarse como observador de eventos de odometría.

De esta forma, el sistema de SLAM es avisado cada vez que existe nueva información de los movimientos realizados por el robot.

En la figura 4.4 se puede observar el componente de desacoplamiento modificado, con el agregado de esta instancia particular de *MoveProvider*.

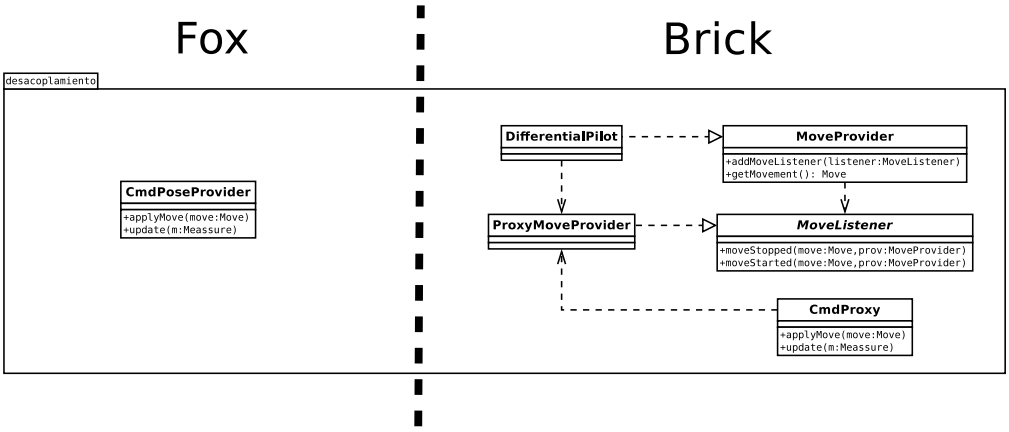



Figura 4.4: Subsistema de desacoplamiento.

### 4.5.2. Despliegue de subsistemas de SLAM

La arquitectura del sistema es distribuida y se despliega en tres plataformas de cómputo diferentes: un ladrillo LEGO, una placa Fox Board G20 y una PC de propósito general. Esta última se usa sólo a propósitos de monitoreo, el SLAM se realiza completamente en el hardware del robot.

#### 4.5.2.1. La PC

La inclusión de la PC como parte del sistema responde a:

1. El requerimiento 3.3.2.4: la PC cumple la función de receptora de los datos de depuración enviados.

2. El requerimiento 3.3.2.5: la implementación de los sistemas desplegados en la PC permiten monitorear e interpretar los datos producidos en los procesos de parametrización y los experimentos para la comprensión del sistema.

#### 4.5.2.2. Arquitectura distribuida

Debido a las limitantes de procesamiento del ladrillo de LEGO, se utiliza también la placa Fox Board G20, en la que se realizan las tareas de mayor carga, en términos computacionales. A continuación se incluye una descripción de cómo se distribuyen las tareas:

- Un ladrillo LEGO dedicado a las tareas de control (exploración) del robot y sensado del ambiente.

- Una placa computadora Fox Board G20 dedicada a la ejecución del sistema de SLAM implementado.

Ambos componentes de hardware ejecutan en paralelo y se comunican utilizando el protocolo que se mencionó anteriormente en 3.3.4.4.

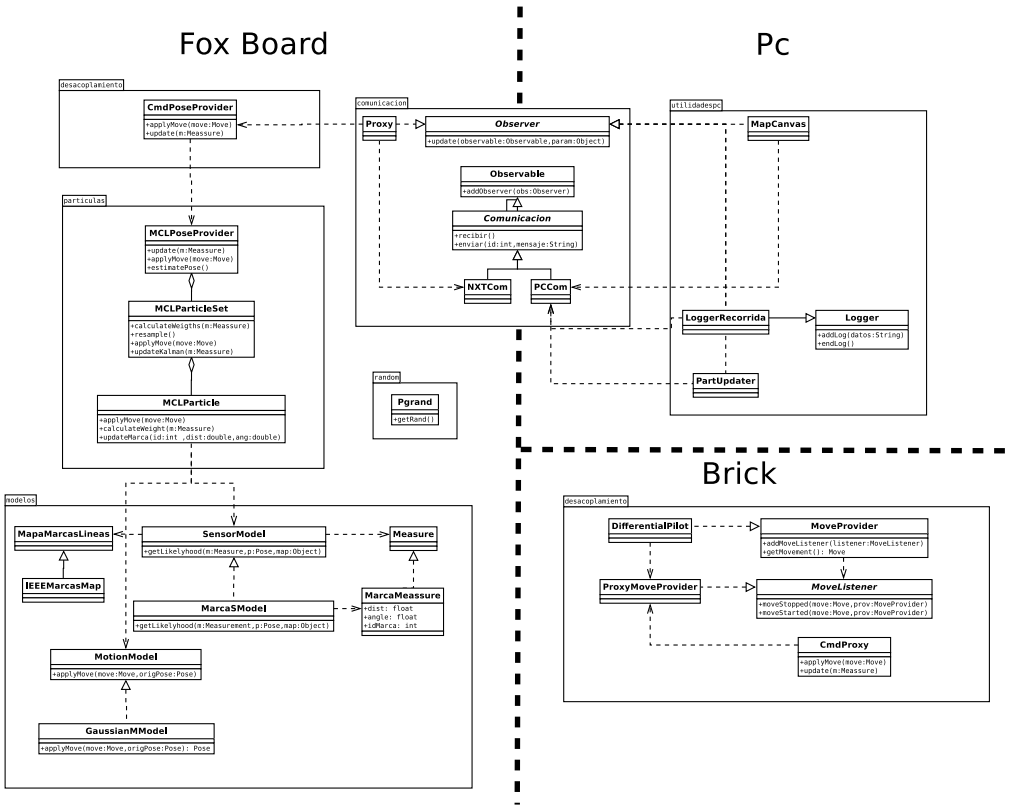



Figura 4.5: Diagrama de despliegue de las clases de la arquitectura.

#### 4.5.2.3. Configuración de despliegue

A continuación se incluye la descripción del despliegue de cada subsistema en cada unidad de hardware:

- Ladrillo Lego: Aquí se despliegan las funciones de exploración, ajenas al subsistema de desacoplamiento
- Fox Board G20: Aquí se despliegan los subsistemas de desacoplamiento, partículas, modelos, comunicación y el de random
- PC: Aquí se despliegan los subsistemas de comunicación (la parte de la PC) y utilidades PC

En la figura 4.5 se puede apreciar el despliegue de los componentes de la arquitectura de SLAM instanciada.

**Desacoplamiento** Debido a que el sistema de partículas se ejecuta en la placa computadora Fox Board G20 y el ladrillo LEGO es quien posee los sensores e información de odometría, estos comandos son serializados y enviados a la Fox Board G20.

La clase *CmdProxy* recibe información por dos canales:

- Mediante la implementación de un patrón *Observer*. La clase *CmdProxy* escucha los eventos generados por las clases *DifferentialPilot* del sistema de LeJOS. Estos eventos se corresponden con el inicio y fin de un movimiento del robot.
- Mediante llamadas explícitas del comportamiento del robot a operaciones de *update*, realizadas por el módulo de detección de marcas.

Esta información es serializada y enviada a la clase *CmdPoseProvider* utilizando los mecanismos de comunicación implementados.

Para transmitir esta información, se implementaron dos clases que instancian la clase comunicación, NXTCom y PCCom, como puede verse en la figura 4.6.

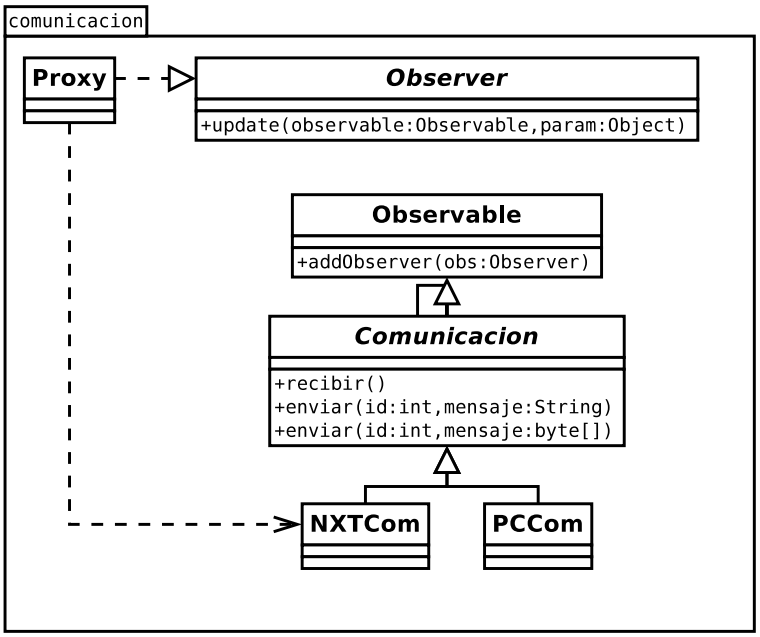



Figura 4.6: Subsistema de comunicación.

Finalmente, los comandos son recibidos por *CmdPoseProvider* y encolados para su ejecución, cuando el subsistema de partículas esté pronto para recibir nueva información.

### 4.5.3. Sistema de Recorrido

En alto nivel, el algoritmo de recorrida busca realizar el cubrimiento total del escenario.

Se plantea una solución basada en comportamientos que utiliza la implementación de subsumption de LeJOS[12]. Se definieron un conjunto de comportamientos que toman el control del robot en diferentes situaciones.

Todos los comportamientos comparten una misma representación, que consta de:

- Una serie de puntos a visitar (*waypoints*)
- La posición brindada por el sistema de SLAM
- Una variable de estado que indica el sentido actual de recorrida, que puede tomar dos valores "yendo" y "volviendo"

#### 4.5.3.1. Comportamientos

A continuación se incluye una descripción de los comportamientos implementados en la solución.

- *WayPointFollower*: navega por los *waypoints* presentes en la lista actual de *waypoints*. Este sistema utiliza la información de ubicación proporcionada por el sistema de SLAM para navegar a cada *waypoint*.

- *WallStopper*: Detiene el robot al detectar un obstáculo utilizando dos sensores de distancia colocados en su parte delantera. Elimina todos los *waypoints* actuales y crea un nuevo *waypoint* que provoca que el robot navegue paralelo a la pared una distancia determinada. El sentido de navegación depende de la variable de sentido de la recorrida.

- *WayPointCreator*: cuando la lista de *waypoints* está vacía, genera un número de nuevos *waypoints* mayor que el ancho del escenario, que hacen que el robot navegue paralelo al eje de las $x$ del mapa, manteniendo la coordenada $y$.

- *Vision*: Detiene el robot para observar una marca presente en el rango de visión y estimar su distancia. Luego, esta información es utilizada para actualizar el sistema de SLAM.

#### 4.5.3.2. Análisis de la ejecución esperada

A modo de ilustrar la conducta de los comportamientos implementados, se incluye el análisis paso a paso de una ejecución acorde a lo que se espera:

1. El robot se encuentra en el punto de partida. Dado que no existen *waypoints*, el comportamiento *WayPointCreator* toma el control y genera una serie de *waypoints* como se muestran en la figura 4.7a.

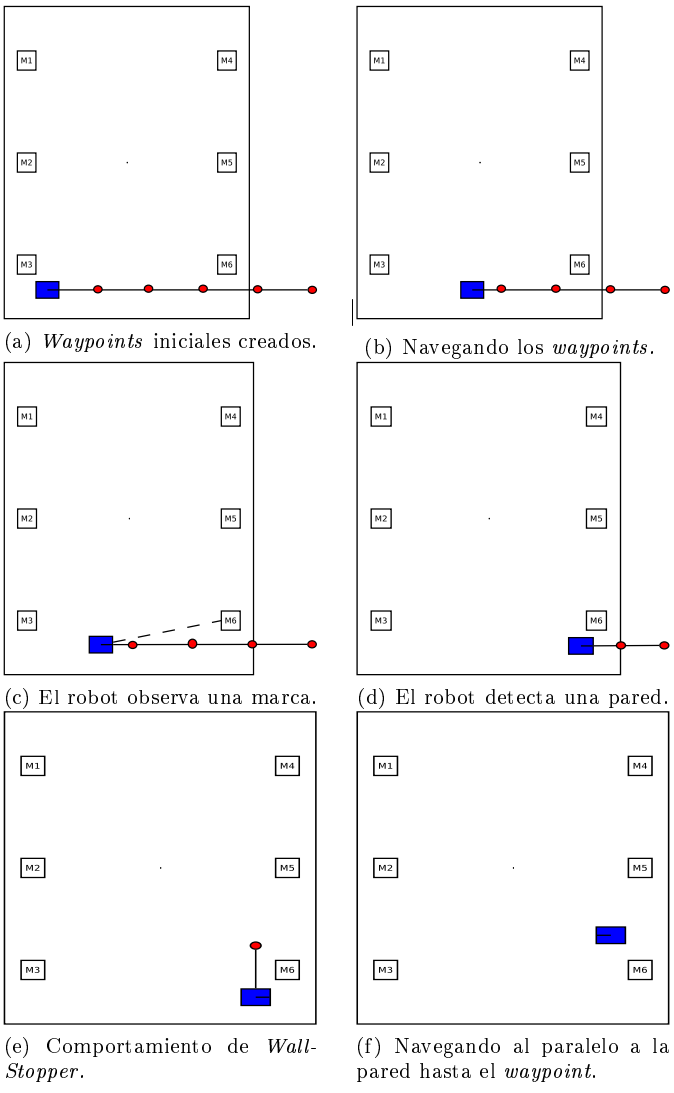
2. Luego, el comportamiento *WayPointFollower* toma la ejecución y navega uno a uno los *waypoints* existentes. Ver figura 4.7b. Es importante notar que luego de llegar a cada *waypoint,* el navegador utiliza la ubicación proporcionada por el sistema de SLAM para decidir como llegar al nuevo *waypoint.*

3. Camino hacia un *waypoint,* el robot detecta una marca y se detiene para observarla mejor y determinar la distancia a la que se encuentra, ver figura 4.7c. Esto es ejecutado por el comportamiento *Vision* que toma el control en el momento en que se observa la marca. La información sensada es enviada al sistema de SLAM. Luego, este comportamiento devuelve el control y es el comportamiento WayPointFollower el que genera que se continúe con el trayecto preestablecido por WayPointCreator.

4. Cuando el robot llega a la proximidad de una pared o una marca, ver figura 4.7d, el comportamiento *WallStopper* elimina los *waypoints* restantes y genera un nuevo *waypoint* acorde a la figura 4.7e.

5. Una vez más, el comportamiento *WayPointFollower* toma la ejecución y navega hasta el *waypoint* generado por el comportamiento *WallStopper,* ver figura 4.7f.

6. Finalmente, dado que no hay *waypoints* a navegar, el comportamiento *WayPointCreator* toma el control nuevamente y el ciclo vuelve a comenzar, ver figura 4.8.




(a) *Waypoints* iniciales creados.

(b) Navegando los *waypoints*.

(c) El robot observa una marca.

(d) El robot detecta una pared.

(e) Comportamiento de *Wall-Stopper*.

(f) Navegando al paralelo a la pared hasta el *waypoint*.

Figura 4.7: Ejecución esperada.

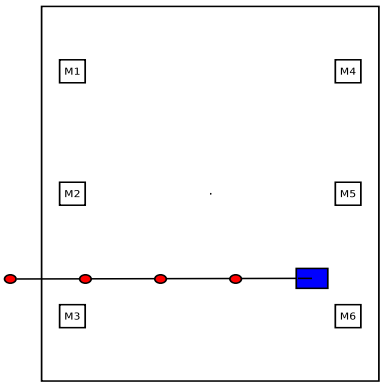



Figura 4.8: Nueva generación de *waypoints*.

### 4.5.4. Ejemplo de una corrida

A continuación se describe una ejecución del sistema de SLAM implementado, en la cual el lector puede ver como los sistemas y subsistemas mencionados anteriormente interactúan entre sí.

1. Cuando el sistema inicia, se da de alta un conjunto de partículas, en una posición inicial establecida y con el mismo ángulo. En la figura 4.9 se puede ver que todas las partículas (círculo rojo) están concentradas en el mismo punto (por eso se ve como una sola). El círculo azul corresponde a la partícula promedio. La mancha gris corresponde al terreno cubierto por el robot. Al principio, el sistema no posee ningún mapa del entorno. Luego, el robot comienza a resolver la navegación como se explicó en la sección 4.5.3.2.

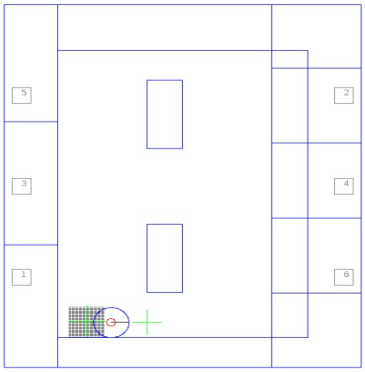



Figura 4.9: Estado inicial del sistema.

2. Luego que el robot comienza a moverse y completa un movimiento de avance o giro, el ladrillo le envía a la Fox Board (MCLPoseProvider) un comando llamado applyMove para que este último aplique el nuevo movimiento sobre las partículas (paso de predicción). Esta invocación se realiza mediante el sistema de desacoplamiento, donde la clase Proxy en el ladrillo envía un mensaje *Command* a CmdPoseProvider en la Fox, que encola el comando para ser procesado cuando el sistema de partículas esté disponible.

3. Para cada partícula, utilizando el modelo de movimiento como se explicó en 3.2.4, se estima su nueva posición $(x, y, \theta)$, acción que se puede ver reflejada en la figura 4.10. El peso de las partículas se mantiene inalterado, pero el sistema en general pierde confianza ya que las partículas están más dispersas. La dispersión de las partículas ocurre sólo en la dirección de avance del robot debido a que este sólo realizó movimientos de desplazamiento y no de rotación, por lo que no existe incertidumbre con respecto a la rotación, y las partículas se mueven todas en el mismo sentido.

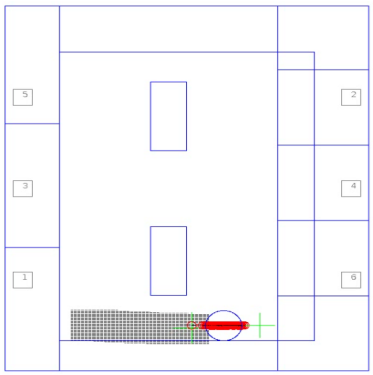


Figura 4.10: Aplicación de applyMove (*predict*) al completar un movimiento.
Además, se puede apreciar como se dispersaron las partículas en el plano horizontal debido al error en el desplazamiento.

4. El robot continúa moviéndose y enviando el comandoapplyMove cada vez que completa un movimiento. Esto se mantiene así hasta que el robot observa una marca.

5. Una vez que el robot observa una marca, se envía un comando update al MCLPoseProvider. Esto desencadena tres procesos: la actualización del mapa en cada partícula, la actualización del peso de cada partícula y la evaluación de la necesidad de hacer resampling . Los tres procesos se describen con mayor detalle a continuación:

   *a*) El proceso de integrar la nueva observación al mapa, implica que para cada partícula se actualice el FK correspondiente a la estimación de dicha marca. En este caso la partícula genera un nuevo FK correspondiente a la estimación de la marca observada, debido a que esta marca no existía en su mapa.
   Debido a que las partículas presentaban un alto grado de dispersión, el conjunto de estimaciones de la marca amarilla también presenta un alto grado de dispersión. Esto se puede ver en la figura 4.11.

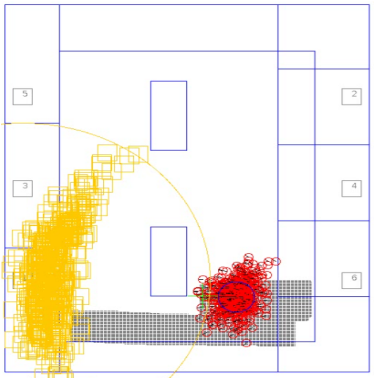


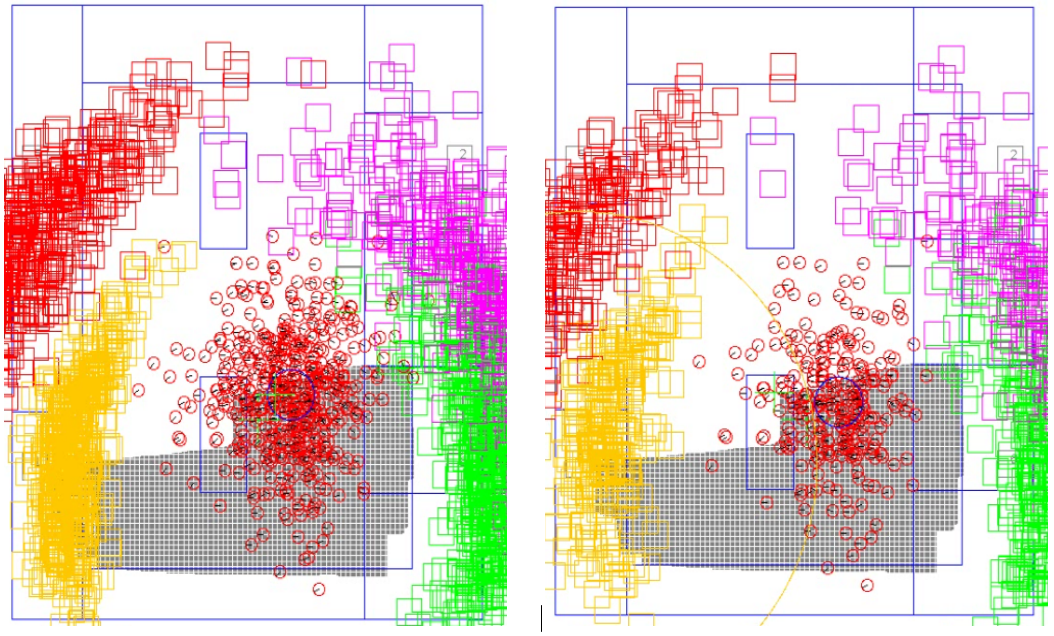
Figura 4.11: Integración de una nueva observación al sistema.
Se puede apreciar la distribución de las partículas (círculos rojos) en el mapa (acorde al error en el avance y la rotación). Además, en amarillo, se muestra la estimación de la marca observada según cada partícula. El círculo amarillo representa la distancia estimada en torno a la marca identificada.

   *b*) Luego, se calculan los pesos de cada partícula teniendo en cuenta la nueva observación y el mapa

mantenido por cada una.

*c*) Como esta marca es observada por primera vez, no proporciona nueva información sobre la verosimilitud de una partícula u otra, por lo que la distribución de los pesos será totalmente uniforme, es decir que tendrán varianza cero. Por lo tanto, no se realizará el paso de resampling.

Más adelante, luego de dar de alta otras marcas, se observa nuevamente la marca amarilla y se realiza el resampling. En la figura 4.12 podemos apreciar el resultado del proceso de resampling.
Se puede observar un ligero cambio en la posición promedio estimada luego del resampling.



(a) Distribución de los respectivos mapas generados por cada partícula.

(b) Se puede observar que las partículas con mayor verosimilitud se mantuvieron en el sistema y además como disminuye la incertidumbre en la posición y en el mapa.

Figura 4.12: Proceso de resampling en una ejecución.

## 4.6. Calibración de parámetros

El rendimiento del sistema de SLAM implementado depende de los valores de ciertos parámetros del algoritmo. A continuación detallamos la naturaleza de estos parámetros.

### 4.6.1. Parámetros

**Parámetros del Modelo de Movimiento** Como se explicó en la sección 3.2.4, el modelo de movimiento depende de dos parámetros correspondientes a las varianzas de las distribuciones que modelan el ruido de desplazamiento.

Estos parámetros deben establecerse de modo que reflejen los errores cometidos por el robot al moverse. Al aumentar el valor de estos parámetros, se contempla un mayor error potencialmente cometido por el robot. Sin embargo, si este parámetro es demasiado alto, la dispersión de las partículas aumenta rápidamente y nunca se logra un estimado preciso de la posición.

**Parámetros del Modelo de Sensado** En la sección 3.2.3 se explicó que el modelo de sensado depende de un solo parámetro. Este debe ser configurado para reflejar la confianza en la estimación de la posición de la marca sensada.

Al aumentar el valor de este parámetro, la confianza en los valores sensados disminuye, aumentando la robustez del sistema ante medidas erróneas. Sin embargo, si estos valores son demasiado altos, la información aportada al sistema de SLAM por cada valor sensado disminuye, debido a que este sistema no confía en estas medidas realizadas. Esto torna imposible la convergencia del sistema en base a la eliminación de partículas inverosímiles.

**Parámetro de resampling** Como se comentó en 3.2.1.3, se implementó un parámetro que controla la frecuencia con la que se realiza el paso de *resampling*.

**Observaciones** Es importante destacar que los parámetros correspondientes a los modelos de sensado y movimiento suelen ser más relajados de lo que deberían ser en teoría. Es decir, que suelen representar más incertidumbre sobre las medidas y movimientos realizados que la incertidumbre real. Esto se hace de esta forma para evitar que la región de alta probabilidad del modelo de sensado coincida con una zona sin partículas, lo que causaría anomalías en el proceso de *resampling*[39].

### 4.6.2. Métodos de calibración

Tomando en cuenta que existen cuatro parámetros a configurar, para los cuales se decidió elegir 5 valores para cada uno, existen $5^4$ combinaciones de parámetros a probar. Además, dado que el rendimiento del sistema en una ejecución depende del azar, deben realizarse varias ejecuciones por combinación de parámetros para evaluar estadísticamente el rendimiento del sistema utilizando una configuración dada. Se eligió 5 como un valor de corridas razonable tanto para no sesgar el experimento como para no perder mucho tiempo en la ejecución de las pruebas.

Tomando estos números como cotas inferiores, se deben realizar al menos $5 * 5^4 = 3125$ ejecuciones para determinar el mejor conjunto de parámetros para el sistema de SLAM. Se estimaron empíricamente unos diez minutos destinados a realizar cada una de las ejecuciones. Luego, se tienen que invertir 31250 minutos, es decir 52 horas para realizar todas las ejecuciones.

Como este valor resultaba demasiado grande, se decidió acortar los tiempos corriendo la calibración en simulaciones, utilizando un conjunto de datos reales, recabados utilizando nuestro robot previamente.

### 4.6.3. Conjuntos de datos

La utilización de datos (*datasets*) es una práctica usual en la evaluación de un algoritmo de SLAM. Pueden encontrarse sitios dedicados a la publicación de estos conjuntos de datos, como por ejemplo [2].

Los conjuntos de datos constan de una lista de datos de sensado y movimiento realizados por un robot real, que constituyen toda la entrada de un algoritmo de SLAM pasivo. Utilizando estos datos, puede ejecutarse un algoritmo repetidas veces sin tener que recurrir a la plataforma robótica real.

Opcionalmente, el conjunto de datos puede incluir información de la ubicación real del robot, a fin de evaluar el desempeño del sistema simulado.

**Obtención de los conjuntos de datos** Se recabaron 12 conjuntos de datos correspondientes a diferentes ejecuciones de un algoritmo de exploración ejecutado por el robot en el ambiente de pruebas.

El ambiente de pruebas constaba del mismo escenario descrito en la sección 4.2, con la salvedad que las marcas estaban espacialmente dispuestas de forma diferente. Además, el entorno contaba con líneas dispuestas sobre el suelo utilizadas para delimitar el entorno de navegación, como se puede apreciar en la figura 4.13. En este ambiente, el robot navegaba en el rectángulo interno del entorno, sin atravesar líneas (se utilizó un sensor de escala de grises apuntando hacia el suelo).

Durante la ejecución, el robot realiza tareas de exploración al azar, mientras se envían datos de movimiento y sensado a una PC.

Al mismo tiempo, esta PC se comunica con un sistema de localización global basado en cámaras, utilizado en fútbol de robots, de nombre Doraemon[3]. El sistema de localización global detecta un parche colocado en la parte superior del robot y brinda coordenadas de la ubicación del robot en los momentos que se sensa una línea en el suelo.

La información de la posición real del robot se incluyó en los conjuntos de datos.

### 4.6.4. Métrica de rendimiento

Utilizando los conjuntos de datos, puede simularse la ejecución del sistema de SLAM para medir el rendimiento de un determinado conjunto de parámetros.

Sin embargo, para esto deben definirse métricas que permitan comparar el resultado de dos ejecuciones sobre los conjuntos de datos.

Se identificaron varias métricas posibles:

- La suma del error cometido por el sistema
- El error promedio cometido por el sistema
- El error máximo cometido por el sistema
- Indicador RMS (*root mean squared*) de la estimación de la coordenada $x$ o $y$ utilizado en la literatura de SLAM
- La calidad del mapa construido

En este contexto se utiliza el término error para referir a la distancia euclidiana entre la posición estimada por el sistema de localización y la brindada por Doraemon.

Se consideró que el error máximo podía verse sesgado por errores en el sistema de visión global o por inconsistencias en parte del conjunto de datos.

Respecto al indicador RMS, se consideró que no incluía mayor información que el error promedio y estos podían intercambiarse indiferentemente.

Se consideró que la suma del error era equivalente al error promedio, con la ventaja que el error promedio brinda información más intuitiva sobre el rendimiento del sistema.

La calidad del mapa construido fue descartada por ser difícil de evaluar de manera objetiva, sobre todo en los casos en el que el mapa presentaba transformaciones lineales con respecto al real, fenómeno que se detalla en la sección 6.2.

Finalmente se escogió el error promedio como métrica a utilizar.

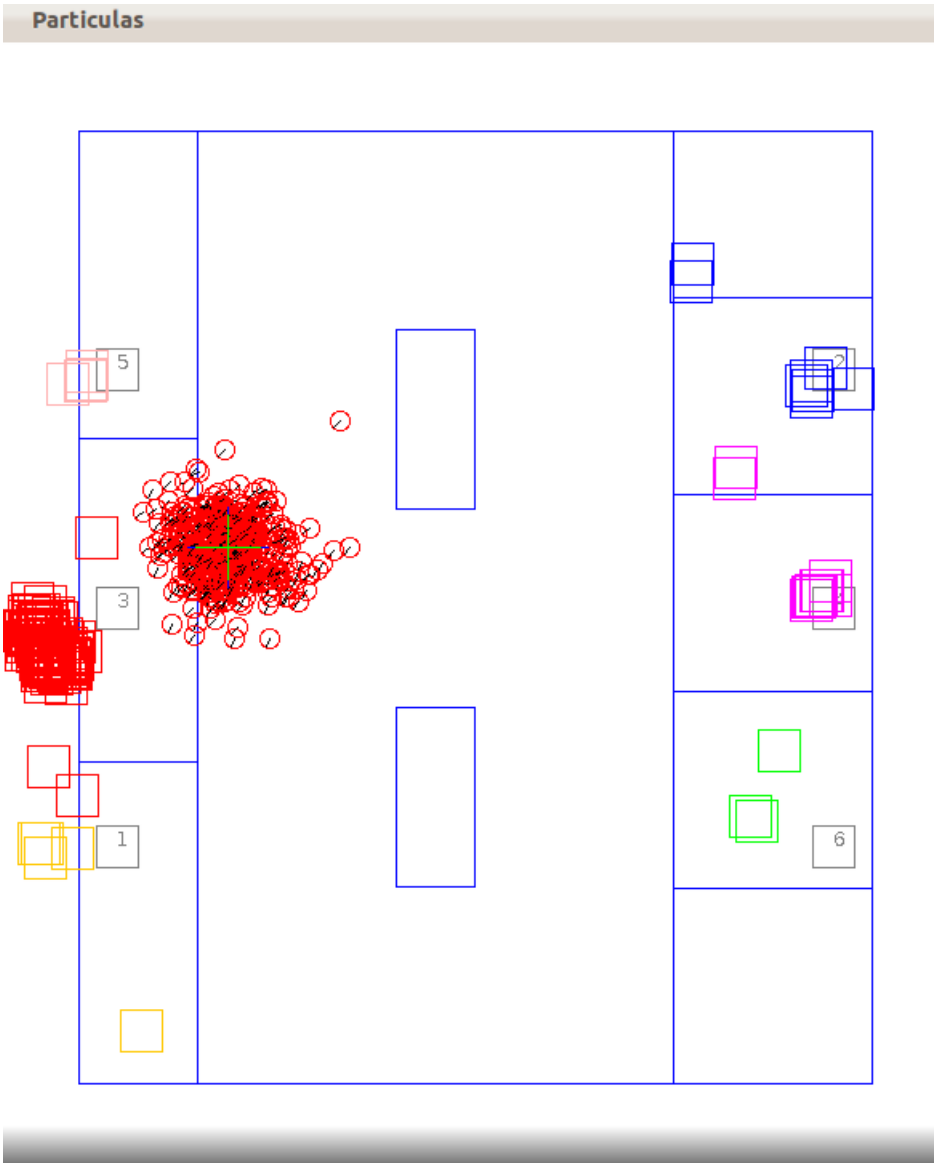



Figura 4.13: Ambiente de pruebas.

### 4.6.5. Definición del conjunto de parámetros

Una vez obtenido el conjunto de datos y definida la métrica a utilizar, se debe definir el método de generación de conjunto de parámetros a utilizar.

**Producto cartesiano** La primera aproximación constaba en la definición de un conjunto de valores para cada parámetro, para luego generar un conjunto de valores resultante del producto cartesiano entre los conjuntos de valores definidos.

Este método resultó ineficiente debido a que probaba combinaciones que resultaban poco probables de ser óptimas. Además, para obtener tiempos de ejecución razonables para el proceso de parametrización, el conjunto de valores a probar debía ser reducido, corriendo el riesgo de dejar fuera buenas combinaciones.

**Algoritmo evolutivo** Se decidió utilizar un algoritmo evolutivo para realizar una búsqueda más inteligente sobre el espacio de parámetros.

Se incluye en el capítulo 5 el detalle del algoritmo evolutivo utilizado para la calibración y los resultados arrojados.

# Capítulo 5

# Algoritmo Genético de Calibración

En este capítulo se presenta un breve resumen del marco teórico de algoritmos evolutivos, y luego una descripción en detalle del algoritmo genético utilizado para la calibración del sistema de SLAM. Se detalla la plataforma utilizada, la configuración de los individuos, los operadores utilizados, la función de *fitness*, la generación de la población inicial, la condición de parada y el proceso de calibración del algoritmo evolutivo genético. Finalmente, se expone un análisis de los resultados obtenidos.

## 5.1. Marco teórico de Algoritmos Evolutivos

### 5.1.1. Introducción

El uso de métodos exactos garantiza obtener óptimos globales para un problema dado, pero eso se torna muy difícil a la hora de resolver problemas que crecen en forma exponencial con el tamaño de los datos. Por otro lado, heurísticas específicas suelen ser muy rápidas, aunque las soluciones obtenidas no son de tanta calidad, y tampoco son reutilizables para otro problemas. Luego, existen las metaheurísticas que pretenden ser un punto medio entre ambas propuestas. Las metaheurísticas son técnicas genéricas que encuentran "buenas" soluciones (incluso el óptimo global) en tiempo de ejecución moderado. También, se las puede ver como una estrategia de alto nivel, que posee cierta estructura y que especifica como se deben aplicar ciertas operaciones para explorar espacios complejos de muchas dimensiones. Dentro de las metaheurísticas podemos encontrar a los algoritmos evolutivos.

Los algoritmos evolutivos consisten en mantener y evolucionar un conjunto de individuos (soluciones tentativas), imitando el proceso biológico de evolución. Estos realizan un proceso iterativo y estocástico sobre un conjunto de individuos, denominado población, donde cada uno representa una posible solución al problema. Para medir la calidad de cada individuo se utiliza una función de *fitness*. Los algoritmos evolutivos realizan dos tipos de tareas: exploración y explotación. En el caso de la exploración se trata de combinar individuos de la población actual utilizando diferentes técnicas para generar nuevos individuos "distintos", y eventualmente mejores, que puedan estar más cerca de la solución óptima. Por otro lado, durante el proceso de explotación se intenta mejorar a los mejores individuos de la especie. Parte de la evolución es determinada por selección natural, de forma similar a la supervivencia del más fuerte. Luego, algunos individuos son mejores que otros y esos tienen mayor probabilidad de sobrevivir y propagar su material genético[29, 15].

### 5.1.2. Componentes de los algoritmos evolutivos

Los algoritmos evolutivos cuentan con una población inicial que suele ser generada al azar. Por otro lado, existen algunas técnicas para elegir una población inicial que pueda acelerar el proceso de búsqueda de buenas soluciones.

La evolución de la población consta usualmente de la aplicación estocástica de diferentes operadores: mutación, selección y recombinación sobre sus individuos.

Una función de *fitness* cuantifica la calidad de un individuo asignándole un valor que permite la comparación con otros individuos. En otras palabras, el valor de *fitness* representa que tan bueno es el individuo respecto al resto.

Estos algoritmos además cuentan con criterios de parada. Uno de los más comunes es la cantidad de iteraciones, otro suele ser haber encontrado una solución con un error menor que cierta cota.

Los individuos son representados usualmente con cadenas (*strings* binarios, decimales o enteros) o alguna estructura de datos más compleja, como pueden ser árboles.

### 5.1.3. Operadores

Existen tres operadores principales los cuales nos permiten modificar a los individuos de la población: recombinación, mutación y selección.

**Recombinación** Se utiliza para generar nuevos individuos a partir de dos o más existentes. Esta operación suele consistir en dividir dos individuos en una parte elegida en forma aleatoria y combinar las partes resultantes entre sí para formar dos nuevos individuos.

**Mutación** Este operador cambia alguna porción (elegido al azar) del individuo de forma de generar más diversidad y potencialmente, mejores individuos.

**Selección** Corresponde al principio de supervivencia de los individuos más adaptados (similar al esquema de selección natural). Este operador selecciona a los mejores individuos para formar la siguiente generación.

### 5.1.4. Seudocódigo

A continuación presentamos el seudocódigo de un algoritmo evolutivo, en el cual se pueden apreciar todos los conceptos mencionados anteriormente.

**Algoritmo 5.1** Seudocódigo de un Algoritmo Evolutivo.

```
P = GenerarPoblacionInicial()

mientras no alcance la condición de fin do
        P' = SeleccionarPadres(P)
        P' = AplicarOperadoresVariación(P')
        Evaluar(P')
        P = SeleccionarNuevaPoblación(P, P')
fin mientras

retornar la mejor solución encontrada
```

## 5.2. Algoritmo Genético implementado

A continuación se describen las principales características del algoritmo genético utilizado.

### 5.2.1. Plataforma utilizada

Se utilizó la plataforma JGAP[6] basada en Java. Se eligió esta plataforma debido a que:

- Se integraba fácilmente con la solución desarrollada al momento, dado que ambos sistemas fueron implementados utilizando el lenguaje Java.
- Facilidad de uso.
- Conocimiento previo de la plataforma por parte de los integrantes del equipo.

### 5.2.2. Representación

Los individuos del algoritmo evolutivo consisten en cuatro alelos de tipo entero, correspondientes a cada uno de los parámetros que utiliza el sistema de SLAM:

1. Un alelo que corresponde al valor del parámetro de error de desplazamiento multiplicado por cien.
2. Un alelo que corresponde al valor del parámetro de error de rotación multiplicado por cien.
3. Un alelo correspondiente al valor del parámetro de actualización al sensar una marca, correspondiente a la varianza del modelo de sensado.
4. Un alelo correspondiente al parámetro que establece la mínima varianza de pesos requerida para realizar el *resampling* (en adelante, parámetro de *resampling*), multiplicado por mil.

La transformación que se hace a los valores del modelo de desplazamiento y *resampling* (parámetros 1, 2 y 4) se realiza de modo de aumentar la resolución de búsqueda y ajustar las cotas de estos parámetros. Además, se definieron restricciones de validez para los individuos que acotan el espacio de búsqueda de parámetros.

En la tabla 5.1 se presenta la resolución de búsqueda y las cotas inferiores y superiores elegidas para cada parámetro. Estas se eligieron basándose en estimados conservativos, derivados de razonamientos sobre posibles cotas y resoluciones adecuadas. Se entiende que esto no representa un sesgo, pues es posible realizar nuevos experimentos de parametrización en caso de que el valor óptimo de un parámetro se encuentre cerca de las fronteras definidas, así como también experimentos para aumentar la resolución de búsqueda en un entorno de la solución hallada en una primera instancia.

| Parámetro | Cota Inferior | Cota Superior | Resolución |
|---|---|---|---|
| Ruido Desp. | 0 | 1 | 0.01 |
| Ruido Rot. | 0 | 1 | 0.01 |
| Act. Marca | 0 | 10000 | 1 |
| *Resampling* | 0 | 1 | 0.001 |

Cuadro 5.1: Cota superior, cota inferior y resolución elegida para cada parámetro del sistema de SLAM.

### 5.2.3. Función de *fitness*

Se implementó una función de *fitness* que consiste en ejecutar el filtro de partículas implementado, con los parámetros correspondientes al individuo, utilizando los conjuntos de datos recabados (ver sección 4.6).

Luego, se calcula la mediana de los errores promedio cometidos en la estimación de la ubicación para cada conjunto de datos, y se toma como valor de fitness el inverso de este número:

$$fitness = mediana\,(error_i)^{-1}$$

$$error_i = \frac{\sum_j (\bar{x_j}, x_j)}{\eta_i}$$

donde:

- $x_j$ corresponde a la posición brindada por el Doraemon
- $\bar{x_j}$ corresponde a la posición estimada por el filtro de partículas
- $\eta_i$ corresponde al número de comparaciones de $x_j$ y $\bar{x_j}$ realizadas durante cada ejecución del filtro de partículas, con cada conjunto de datos $i$.
- La mediana es calculada ordenando los datos de menor a mayor y se toma el dato de la mitad de la lista. En caso de existir una cantidad par de datos, se toma el promedio entre los dos elementos más cercanos al medio de la lista.

En el caso de que la sumatoria de error fuese igual a cero, se asigna al individuo un valor de fitness alto elegido arbitrariamente. Esta función de *fitness* fue definida acorde a la métrica elegida para evaluar el rendimiento de una ejecución del algoritmo de SLAM, según se detalla en la sección 4.6.4.

Resulta importante notar que la función de *fitness* construida posee una componente randómica, heredada del filtro de partículas que implica que un mismo individuo puede ser evaluado con diferentes valores de rendimiento en distintos momentos.

Por lo tanto, para evaluar un individuo, es decir una calibración dada del sistema de SLAM, el filtro de partículas es ejecutado 10 veces por conjunto de datos, tomando el valor medio del error cometido. Esto se realiza buscando que el azar de una ejecución del algoritmo no favorezca injustamente un juego de parámetros sobre otro.

Además, una vez calculado el valor de *fitness* para un individuo, este es almacenado para evitar calcularlo nuevamente más adelante. Esto, aparte de aumentar la eficiencia del algoritmo, evita que un individuo presente diferentes valores de *fitness* en una misma ejecución del algoritmo evolutivo.

#### 5.2.3.1. Individuos no factibles

No se realiza un tratamiento de individuos correspondientes a soluciones no factibles (valores de alelos fuera de los rangos permitidos) en la implementación de la función de fitness. Esto se debe a que, como se verá en seguida, la generación de la población inicial como los operadores de mutación y cruzamiento no generan este tipo de individuos.

### 5.2.4. Operadores

Se utilizaron operadores de cruzamiento de un punto y mutación. No se utilizaron operadores ad-hoc.

#### 5.2.4.1. Operador de mutación

Se utilizó el operador de mutación implementado en la plataforma JGAP. Este operador de mutación recibe un parámetro que indica la proporción esperada $p_m$ de alelos que debe mutar.

Cuando corresponde, el operador de mutación asigna un valor al azar dentro del rango válido definido para el alelo.

#### 5.2.4.2. Operador de cruzamiento

El operador de cruzamiento utilizado selecciona dos individuos de la población al azar. Luego, se elige un alelo al azar. Este alelo y todos los siguientes son intercambiados entre los individuos elegidos. Los dos nuevos individuos generados son agregados a la población candidata a la selección.

Este operador recibe un parámetro indicando la proporción de la población esperada $p_c$ de individuos a cruzar.

A modo de ejemplo, si se tienen dos individuos a cruzar:

$$A = \{a_1, a_2, a_3, a_4\}$$

$$B = \{b_1, b_2, b_3, b_4\}$$

Se sortea un índice aleatorio entre 0 y 4 no inclusive. Si este número sorteado resulta ser 2, se generan dos nuevos individuos con la configuración siguiente:

$$C = \{a_1, a_2, b_3, b_4\}$$

$$D = \{b_1, b_2, a_3, a_4\}$$

### 5.2.5. Generación de la población inicial

La población inicial se generó totalmente al azar, mas respetando los rangos de validez de los alelos, de modo de no sesgar el proceso de búsqueda.

### 5.2.6. Criterio de parada

Se utilizó como criterio de parada del algoritmo evolutivo la convergencia del *fitness* del mejor individuo de la población.

Para esto, se estableció un número arbitrario de diez ciclos de evolución, luego de los cuales, si la función de *fitness* del mejor individuo permanecía constante, se terminaba la búsqueda. El mejor valor de fitness se consideró constante si no cambiaba en un 1 % luego de las diez iteraciones.

### 5.2.7. Calibración del algoritmo genético

Se observa que el algoritmo genético depende de tres parámetros:

- Probabilidad de cruzamiento
- Probabilidad de mutación

- Tamaño de la población

Para hallar los valores óptimos para estos parámetros, se definieron conjuntos de valores posibles, uno para cada parámetro, acorde a los citados en la literatura. En la tabla 5.2 se pueden observar los valores escogidos para cada uno de los parámetros.

| Prob. Cruzamiento | 0,3 | 0,5 | 0,7 | 0,9 | |
|---|---|---|---|---|---|
| Prob. Mutación | 1/2 | 1/5 | 1/10 | 1/50 | 1/100 |
| Cant. Individuos | 20 | 50 | 100 | 200 | 500 |

Cuadro 5.2: Valores posibles elegidos para la calibración del algoritmo evolutivo.

Luego, se realizaron ejecuciones del algoritmo genético para probar su eficiencia utilizando cada combinación de los valores de los parámetros, resultado de realizar el producto cartesiano entre los conjuntos de valores posibles.

La evaluación de cada conjunto de valores de parámetros consistió en la ejecución del algoritmo genético sobre 2 conjuntos de datos del SLAM de los 12 recabados. Se realizaron 10 ejecuciones del algoritmo genético por conjunto de datos, a fin de evitar que el azar de una ejecución sesgara el proceso de parametrización. Finalmente, de estas 10 ejecuciones, se tomó el valor medio del mejor individuo de cada ejecución, como el individuo obtenido para los parámetros correspondientes.

#### 5.2.7.1. Resultado de la calibración del algoritmo genético

En la tabla 5.3 se pueden observar los parámetros óptimos encontrados.

| Prob. Cruzamiento | Prob. Mutación | Tam. Población |
|---|---|---|
| 0.7 | 1/10 | 100 |

Cuadro 5.3: Conjunto de parámetros óptimo del algoritmo genético.

## 5.3. Resultados arrojados

Luego de calibrado el algoritmo genético, se realizaron 30 ejecuciones para encontrar los parámetros del algoritmo de SLAM que minimizan el error cometido sobre 8 de los 12 juegos de datos. En la tabla 5.4 se pueden observar los valores encontrados por el algoritmo genético. Se incluyen los valores para el mejor individuo, y la media y varianza por parámetro.

| | Mejor Individuo | Media | Varianza |
|---|---|---|---|
| Param. de Traslación | 0,12 | 0,12 | 0,067 |
| Param. de Rotación | 0,10 | 0,125 | 0,035 |
| Param. de Sensado | 7158 | 7920,5 | 1990,32 |
| Param. *Resampling* | 0,071 | 0,45 | 0,30 |

Cuadro 5.4: Valores de los parámetros de SLAM que minimizan el error cometido en el conjunto de datos recabados.

En la tabla 5.5 se incluyen los valores de fitness y error medio obtenido en las ejecuciones. Acorde a la definición de la función de *fitness*, el error incluido en la tabla corresponde al inverso del valor de *fitness*. Se incluye el valor del mejor individuo, así como la media y varianza de los valores de *fitness* y error obtenidos.

| | Mejor Individuo | Media | Varianza |
|---|---|---|---|
| Fitness | 0,005339783 | 0,0049437784 | 0,0001918224 |
| Error | 187,2735290527 | 202,2744750977 | 8,2333059004 |

Cuadro 5.5: Valores de fitness obtenidos.

### 5.3.1. Interpretación

Al interpretar los parámetros arrojados por el sistema se observa que el error de desplazamiento y rotación encontrados como óptimos son relativamente altos. Los parámetros encontrados implican que el óptimo consta de considerar un error mayor o igual al 10 % en cada movimiento realizado.

Los valores del parámetro de sensado encontrados fueron también más altos de lo esperado. Este alto valor refleja la baja confianza depositada en la estimación de la posición de las marcas por parte del sistema de sensado. Este valor óptimo resultó ser mucho más alto del elegido por los autores durante las pruebas iniciales.

El parámetro de *resampling* también constituyó una sorpresa para los autores, debido a que el retornado por el algoritmo evolutivo resulta particularmente bajo. Empíricamente se observó que con un valor como el retornado, el sistema realiza el paso de *resampling* prácticamente en cada iteración, exceptuando algunos casos en los que se realizan sucesivas observaciones de una misma marca desde una misma posición, que aportan relativamente poca información al sistema.

Con respecto a los valores de error obtenidos, los autores los consideran razonables, tomando en cuenta que existen varios factores que no permiten una ubicación perfecta, más allá del software utilizado:

- La naturaleza de la cámara utilizada y el consecuente error cometido en la estimación de la distancia y ángulo a las marcas
- El error cometido por el sistema de visión global
- Presencia de observaciones con id de marca falsos, que viola las hipótesis de trabajo del sistema implementado

Más allá de estos factores de error, como se comenta en la sección 5.4, se observó que los mapas generados por el sistema de SLAM presentan transformaciones de rotación y traslación que afectan al error en la estimación con respecto a un marco absoluto (el sistema de visión global utilizado). Esto introduciría un error constante en la estimación de la ubicación, correspondiente a la transformación entre un marco de referencia y otro.

#### 5.3.1.1. Varianza

Se observó una alta varianza en todos los parámetros encontrados durante las 30 ejecuciones del algoritmo genético.

En la figura 5.1 se puede observar la matriz de correlación de los parámetros obtenidos por ejecución del algoritmo genético (relación lineal entre los parámetros). Allí se observa una correlación leve entre el parámetro de rotación y de *resampling*. Entendemos que esta correlación podría explicar la alta varianza de los parámetros arrojados por el algoritmo genético. Esto se debe a que la solución óptima podría no constar de un punto en el espacio de búsqueda, si no de un subespacio que relaciona el valor del parámetro de *resampling* con el de rotación.

Sin embargo, entendemos que los valores de correlación no son lo suficientemente fuertes para una conclusión definitiva.

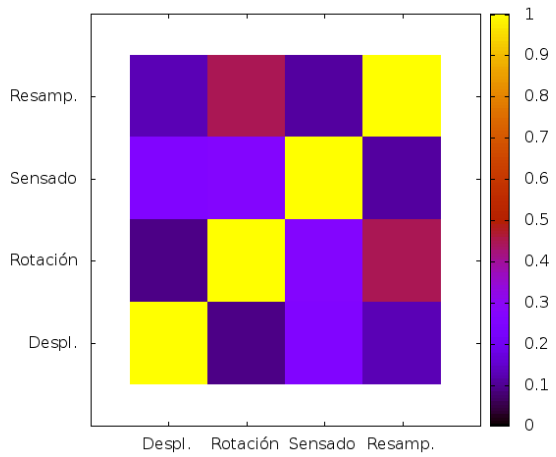



Figura 5.1: Correlación de los parámetros obtenidos por corrida del AG.
Se puede observar una leve correlación del parámetro de rotación y *resampling*.

#### 5.3.1.2. Descomposición de los datos

Si se descompone el conjunto de datos utilizado y se evalúa el rendimiento de la localización sobre cada conjunto de datos, correspondiente a la ejecución del robot por 10 minutos, obtenemos la gráfica de la figura 5.2.

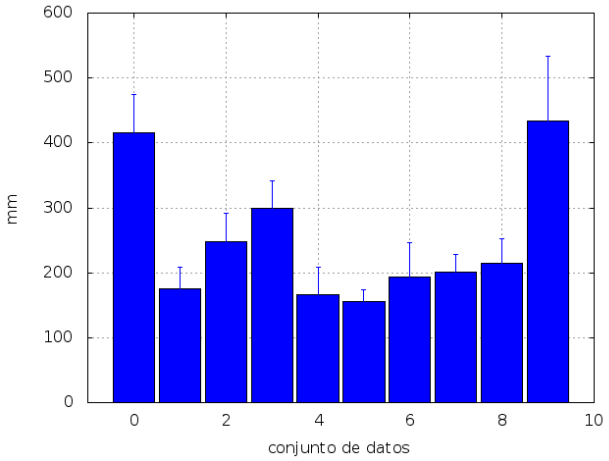



Figura 5.2: Error cometido por ejecución del robot.
Se toma la media y varianza entre todas las ejecuciones del algoritmo evolutivo.

Se observa que los diferentes conjuntos de datos implican diferentes desafíos para el algoritmo. Esto se atribuyó a la presencia de algunos juegos de datos que corresponden a ejecuciones donde el robot comete grandes errores en los movimientos y la recuperación de la localización se torna más compleja, o aquellas en las que existen observaciones con id de marca erróneo.

## 5.4. Validación de los parámetros de SLAM obtenidos

A fin de validar los resultados obtenidos utilizando el algoritmo genético, es decir la parametrización del algoritmo de SLAM, se utilizaron los datos de dos ejecuciones adicionales, que fueron dejadas de lado durante el proceso de calibración y ejecución del evolutivo.

A continuación se incluye una gráfica de error medio y varianza para 10 ejecuciones del algoritmo de SLAM sobre cada juego de datos, ver figura 5.3.

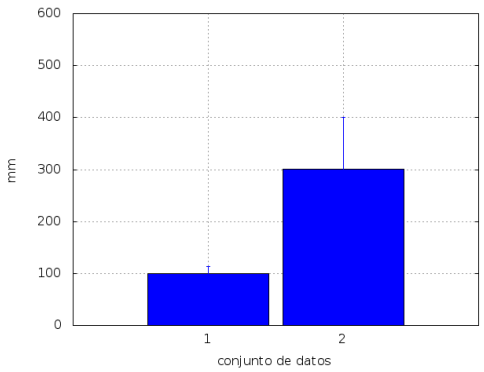



Figura 5.3: Media y varianza del error cometido en los juegos de datos de evaluación.

Se observa un error similar al cometido por el sistema sobre los datos utilizados para la ejecución del algoritmo genético.

### 5.4.1. Mapas generados

Al igual que en la ejecución del algoritmo evolutivo, se observó una diferencia importante en el error cometido entre una ejecución y otra. A fin de explicar esta diferencia, se estudiaron los mapas generados por una ejecución del algoritmo de SLAM parametrizado sobre estos juegos de datos.

Al utilizar el juego de datos 1 se observa que el mapa generado se corresponde bastante con el mapa real del entorno. En la figura 5.4 y 5.5 se observa la evolución del mapa a medida que se procesan los datos de movimiento y sensado para los dos conjuntos de datos utilizados.

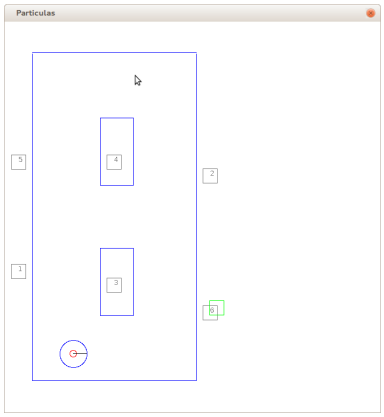
## 5.5. Tiempos de ejecución

A continuación se incluye un reporte de los tiempos de ejecución de la etapa de parametrización y ejecución del algoritmo genético.

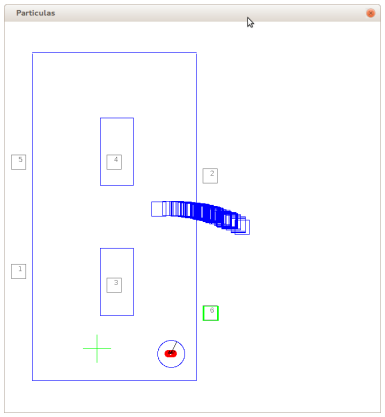
### 5.5.1. Plataforma utilizada

Para la ejecución del algoritmo genético se utilizaron máquinas class0 del cluster de Facultad de Ingeniería[10]. Estas máquinas presentan las siguientes características:

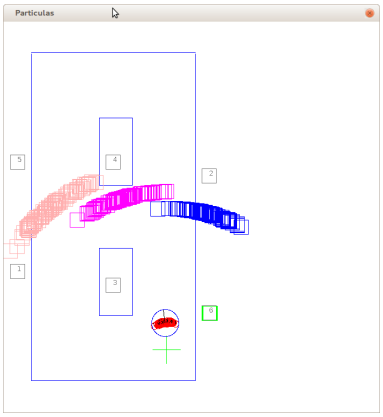
- CPU: Quad core Xeon E5430, 2x6 MB caché, 2.66GHz, 1.333 MHz FSB
- RAM: 8 GB
- Arquitectura de 64 bits




(a) Marca inicial.

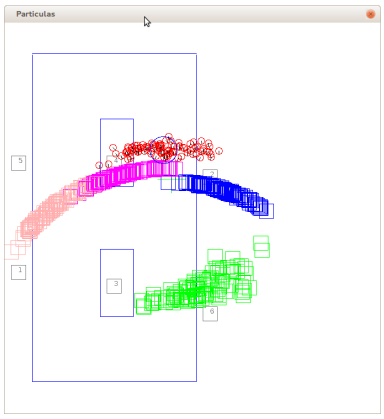



(b) Se añade la segunda marca.

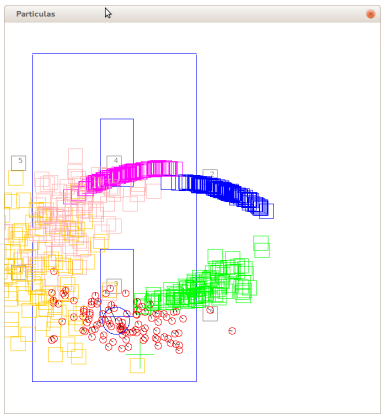



El error en la estim

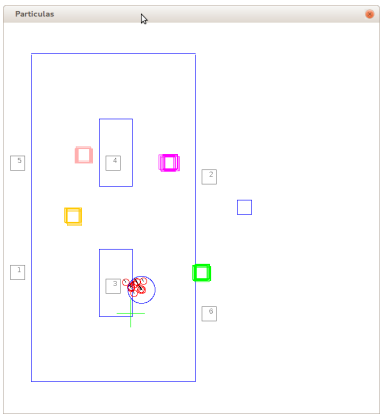
(




(d) El robot observa nuevamente la marca inicial.




(e) Se observan más marcas y la dispersión de la estimación aumenta.




El mapa converge a u

Figura 5.4: Evolución del mapa del conjunto de datos 1.

- SO: Linux
- Java: OpenJDK Runtime Environment (IcedTea6 1.9.10)

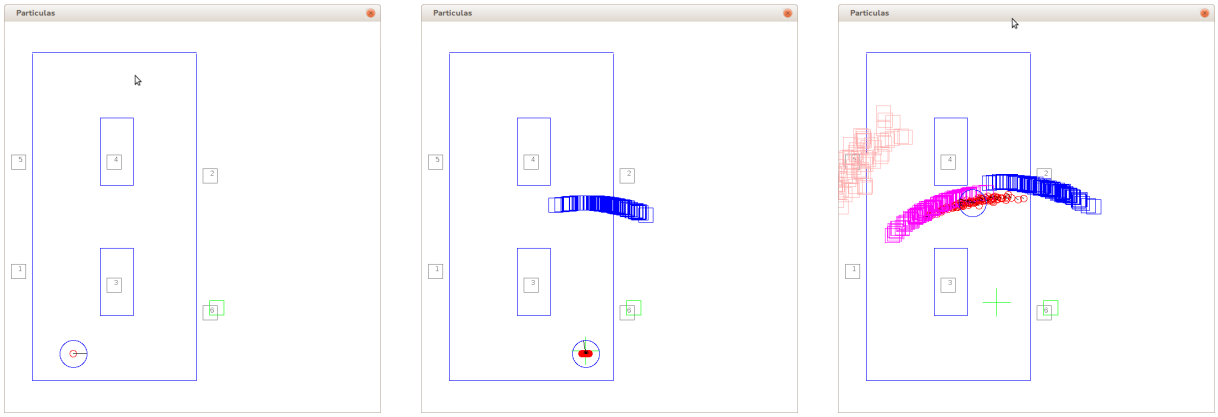
### 5.5.2. Calibración

El proceso de parametrización constó de 100 pruebas del algoritmo para cada juego de parámetros del algoritmo genético, que constó de 10 ejecuciones del algoritmo genético. En la tabla 5.6 se incluye el tiempo promedio y varianza del tiempo de ejecución por prueba.

| Media | Varianza |
|---|---|
| 1844,73 s. | 1396,24 s. |

Cuadro 5.6: Media y varianza del tiempo de ejecución de las pruebas de parametrización en segundos.

Atribuimos la alta varianza a las diferentes poblaciones utilizadas en las diferentes pruebas.




(a) Marca inicial.

(b) Se añade la segunda marca.

(c) Se añaden dos marcas. El error en la estimación aumenta y con este la dispersión de las estimaciones de las marcas.

(d) Ocurre un falso positivo, se observa una marca y se infiere mal su id. Esto perjudica gravemente al algoritmo.

(e) Mapa final.

Figura 5.5: Evolución del mapa del conjunto de datos 2.

### 5.5.3. Ejecuciones

En la tabla 5.7 se presenta el tiempo de ejecución de las 30 ejecuciones del algoritmo evolutivo parametrizado.

| Media | Varianza |
|---|---|
| 7021,03 s. | 400,18 s. |

Cuadro 5.7: Media y varianza del tiempo de ejecución de las corridas del algoritmo evolutivo parametrizado en segundos.

# Capítulo 6

# Pruebas Realizadas

En este capítulo se presentan las pruebas realizadas para evaluar el sistema de recorrido implementado, detallando el entorno de pruebas utilizado, las métricas establecidas, las ejecuciones realizadas, y brindando un análisis de los resultados.

Para evaluar el sistema de SLAM implementado y la solución de recorrido que lo utiliza (en adelante SLAM-R) se propuso realizar la comparación con otro sistema. Este sistema fue implementado por los autores.

El sistema destinado a la comparación consiste de una variante del sistema de recorrido en la que la localización se implementa utilizando un sistema de integración de información de odometría, en adelante Odom-R. En el sistema Odom-R, la información de la ubicación del robot se obtiene sumando la información proporcionada por odometría, es decir que no se utiliza la información de las marcas visuales para corregir el error acumulado.

De este modo, se puede evaluar el sistema de SLAM implementado, comparando los rendimientos de SLAM-R y Odom-R.

En las pruebas se utilizó un sistema de visión global Doraemon (en adelante Doraemon) que provee visión global del entorno donde se mueve el robot. Doraemon se utilizó únicamente con fines de evaluación y la información de posición global nunca fue suministrada al sistema evaluado.

## 6.1. Pruebas de Cubrimiento

Estas pruebas constan en medir el cubrimiento del entorno realizado por ambos sistemas, SLAM-R y Odom-R, a fin de tener un criterio objetivo de comparación.

A continuación se detalla la metodología seguida para las pruebas realizadas.

### 6.1.1. Metodología

#### 6.1.1.1. Punto de Partida

El robot partió en todas las ejecuciones desde la posición $(600, 300)$ medida en milímetros, y tomando como origen el punto más arriba y a la derecha en la figura del entorno descrito, ver figura 6.1.

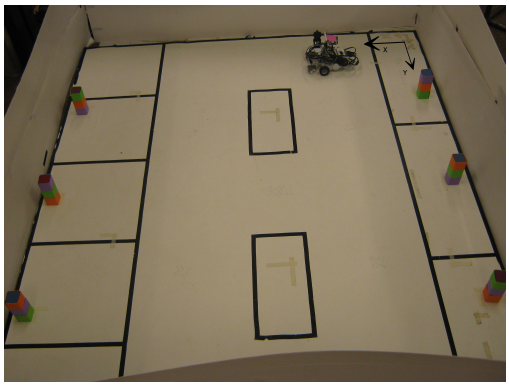


Figura 6.1: Punto de partida del entorno de pruebas.
En la esquina superior derecha se puede observar el marco de referencia. Esta imagen difiere con las capturas de pantalla de los mapas generados en la ubicación del marco de referencia, que se encuentra en la esquina inferior izquierda.

#### 6.1.1.2. Repeticiones

Debido a que el sistema de SLAM utiliza una componente de azar al sortear las partículas, resulta conveniente realizar varias ejecuciones de cada prueba para evaluar al sistema de forma estadística, obteniendo promedios de varias medidas. Se realizaron 5 ejecuciones de cada sistema en cada prueba realizada.

Si bien 5 ejecuciones por sistema no representan un número óptimo para una comparación estadística, mantuvimos este número para disminuir el tiempo que insumen la preparación y ejecución de las pruebas de modo de hacer la ejecución de los experimentos factible en el tiempo disponible en el marco de un proyecto de grado. Se debe tener en cuenta que, como se verá más adelante, se dedicaron finalmente 100 minutos a cada ejecución.

#### 6.1.1.3. Terminación de las pruebas

Se definieron dos criterios de terminación de las pruebas:

- Abandono del escenario: Si una rueda del robot sale del escenario se considera la prueba como finalizada. Este criterio tiene sentido ya que si el robot sale del ambiente de trabajo le es físicamente difícil de retornar.
- Tiempo de ejecución: Las pruebas finalizaban por tiempo de ejecución. Se realizaron ejecuciones de 20 minutos de duración. Como se verá en la sección 6.1.2, el desenlace de estas ejecuciones motivaron al diseño de un segundo conjunto de pruebas en las que se aumentó el tiempo de ejecución a 100 minutos. Este criterio surge debido a la necesidad de limitar el tiempo de las pruebas, pero teniendo en cuenta el tiempo mínimo necesario que el robot necesita para recorrer todo el escenario.

Ambos criterios fueron utilizados de forma simultánea. Es decir que el cumplimiento de cualquiera de los criterios implicaba la terminación de la ejecución.

#### 6.1.1.4. Métricas de rendimiento

Se establecieron las siguientes métricas para la evaluación de los sistemas de recorrido. Estas métricas se definieron teniendo en cuenta el escenario de trabajo y definiendo una zona en la que se espera que el robot se mueva libremente, a una distancia prudente de las paredes y de las marcas.

**Cubrimiento útil** Se delimitó una zona en el interior del entorno de operación del robot, denominada zona útil. Esta zona corresponde al espacio que el robot debería cubrir en caso de operar de forma óptima.

En base al esquema de comportamientos planteado en la sección 4.5.3, se define la zona útil como un rectángulo interior al entorno de operación, donde cada lado dista 30cm. del lado correspondiente en el límite del entorno, como muestra la figura 6.2.

Debido a que el robot se detiene y cambia su rumbo al ver una pared a 40cm., y el cubrimiento se registra con 20cm. de diámetro, la zona de cubrimiento "no útil" debe ser una banda de 30cm. de ancho.

Sin embargo, la distancia del límite inferior al borde del entorno se redujo a 20cm., debido a que la posición inicial del robot implicaba que estos 10cm. restantes fueran cubiertos en el inicio del experimento, como se vio en la sección 6.1.1.1.

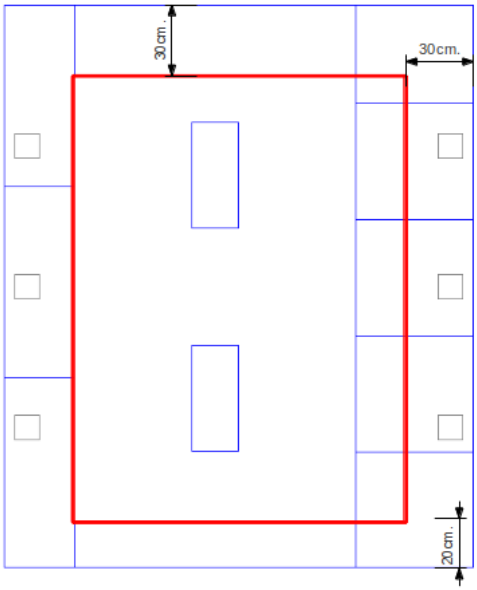



Figura 6.2: Límite de la zona útil (rojo) en el entorno de trabajo (azul).

A su vez, todo el entorno de trabajo fue dividido con una grilla regular cuya precisión es de $0,5$cm. Luego, a medida que el robot navegaba por el entorno, su posición real es obtenida del sistema de visión Doraemon y las celdas vecinas al robot son marcadas como visitadas, donde se entiende vecindad del robot como un cuadrado de 20cm.[1] de lado centrado en el robot.

Luego, se define cubrimiento como la cantidad de celdas marcadas como visitadas por el robot.

En la figura 6.3 se puede observar el cubrimiento de una ejecución graficado en el mapa del entorno.

[1] La utilización de esta constante fue elegida tomando en cuenta el tamaño del robot y margen que podría abarcar una eventual cámara apuntando al piso. Esta decisión se tomó considerando que el robot podría recorrer el entorno buscando elementos que solo pueden ser sensados en un entorno de su ubicación.

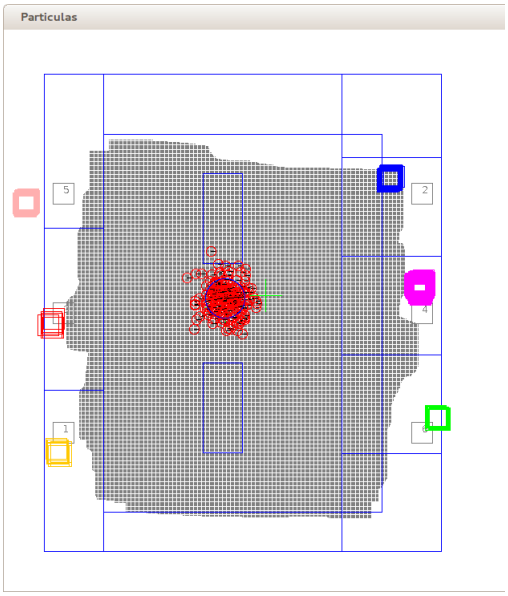



Figura 6.3: Cubrimiento de una ejecución utilizando SLAM.
En gris se muestra la zona cubierta por el robot.

Finalmente, se toma como cubrimiento útil, la cantidad de celdas de la zona útil visitadas. La medida obtenida es normalizada utilizando el número total de celdas de la zona útil, de forma que la métrica representa un porcentaje del cubrimiento útil posible.

**Cubrimiento inútil** Análogamente al cubrimiento útil, esta métrica consta del recuento de celdas visitadas fuera de la zona útil. Al igual que el cubrimiento útil, esta medida es normalizada en base a la cantidad de casillas que componen la zona inútil, por lo que la métrica representa la proporción de zona inútil cubierta por el robot.

Resulta importante mencionar que el cubrimiento inútil no representa el porcentaje de cubrimiento realizado por el robot fuera de la zona útil, si no el porcentaje cubierto de la zona inútil . Esta métrica fue elegida sobre la otra considerando escenarios en los que el cubrimiento fuera de la zona útil pudiera tener una penalización propia, más allá del desperdicio de recursos provocado por recorrer una zona no útil.

**Cubrimiento útil total** Esta métrica cuenta las casillas visitadas dentro de la zona útil, con la salvedad que cuenta varias veces cada casilla visitada. Esta métrica busca medir cuánto cubrimiento útil logró hacer el sistema en toda la corrida, sin importar si este visitaba dos veces la misma zona.

Mientras que la métrica de cubrimiento útil busca contabilizar las casillas visitadas alguna vez, la métrica de cubrimiento útil total busca contabilizar cuántas veces es visitada cada casilla. Es decir que una clasifica las casillas de la zona útil como visitadas o no visitadas, mientras que la otra guarda para cada celda la cantidad de veces que esta fue visitada.

Esta métrica también es normalizada, en base a la cantidad de casillas de la zona útil. Si bien el número obtenido representa una proporción, esta no está directamente relacionada con la cantidad de veces que se recorre completamente el entorno. Esto se debe a que el conteo se realiza con cada posición obtenida del Doraemon, lo que lleva a que usualmente una casilla sea contabilizada más de una vez en una misma pasada del robot. Sin embargo, los autores entienden que el valor es útil para comparar dos cubrimientos cualesquiera,

ya que esté conteo por demás debería ocurrir en la misma proporción para cualquier ejecución de cualquiera de los dos sistemas.

**Eficiencia del recorrido** Esta métrica se define como el cociente entre el cubrimiento útil total y la distancia recorrida por el robot, a fin de medir la relación entre consumo de recursos, por ejemplo batería y tiempo, y el trabajo efectivo realizado. Es decir:

$$EficienciaRecorrido = CubrimientoUtilTotal/DistanciaRecorrida$$

Por lo tanto, esta métrica mide el cubrimiento útil obtenido por unidad de desplazamiento, dando una noción de eficiencia en los recursos invertidos en el desplazamiento del robot.

Dado que esta métrica se basa en el cubrimiento útil total, presenta el mismo problema en cuanto al porcentaje obtenido. Nuevamente, si bien este porcentaje no corresponde a una proporción real de cubrimiento, los autores consideran la métrica apropiada para la comparación.

**Error medio en la estimación de la posición o Error RMS** Esta métrica evalúa directamente el error en la estimación de la posición realizada por los sistemas de localización (SLAM y Odometría), tomando como referencia la posición brindada por el sistema Doraemon.

Para estas pruebas se utilizó un indicador utilizado en la literatura de SLAM denominado RMS, del inglés *root-mean squared*. Esta consta de tomar la raíz cuadrada del promedio del cuadrado de los errores. Es decir:

$$RMS = \sqrt{\frac{1}{n}\sum_{1}^{n}(x_{estimada} - x_{Doraemon})^2}$$

**Ejecución efectuada con éxito o errónea** Esta métrica cualitativa mide el éxito de la ejecución de la prueba. Esta métrica indica si la ejecución terminó debido a errores cometidos por el robot o simplemente debido a que se cumplió el tiempo destinado a la ejecución.

Puede interpretarse esta métrica como una medida de la robustez del sistema o su capacidad de operar en forma continua sin cometer errores que lo saquen del entorno de ejecución.

**Evolución del cubrimiento útil total** Para esta métrica en particular se decidió estudiar su evolución con el paso del tiempo, para determinar las condiciones o el momento en las que un sistema pudiera tornarse más eficiente que otro.

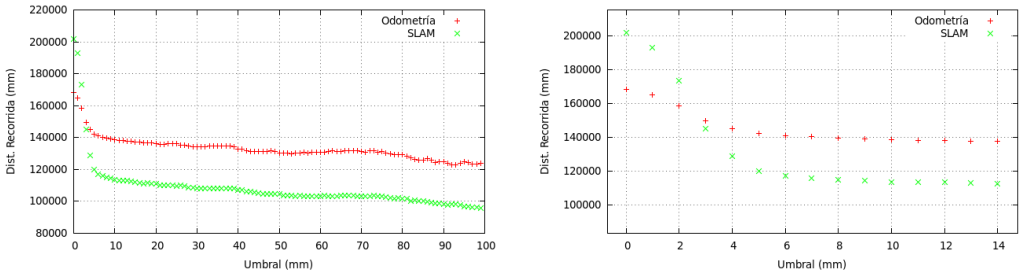
#### 6.1.1.5. Ajustes a los datos

Es importante notar que los datos de posición del sistema de visión global debieron ser filtrados para poder estimar la distancia recorrida. Esto se debió a dos causas:

- Sobreestimación de la distancia recorrida y el cubrimiento realizado
- *Outliers* en la posición entregada por Doraemon

**Sobreestimación** Oscilaciones en la posición entregada por el sistema Doraemon hacían que la simple suma de distancias entre una posición y otra fallara como estimador de la distancia recorrida, debido a que esta aumentaba aún cuando el robot se encontraba detenido.

Dado que ambos sistemas presentan diferentes relaciones entre el tiempo en movimiento y tiempo detenidos, no resulta equitativo ignorar este error. Por lo tanto se realizó un análisis de los datos otorgados por el sistema de visión global para encontrar un umbral mínimo de distancia entre una posición y la siguiente que permita asegurar que el cambio no corresponde a una oscilación del Doraemon. En la figura 6.4 se observa como a partir de un umbral de 10 milímetros, la distancia recorrida por ambos conjuntos de datos permanece relativamente constante. Se atribuye este comportamiento a que a partir de este umbral, las oscilaciones quedan descartadas y los cambios de posición registrados constan de un movimiento real del robot. Luego, el aumento leve del umbral no afecta apreciablemente a la medida de distancia recorrida y esta permanece relativamente constante. Por esto se decidió utilizar 10 milímetros como umbral de filtro de los datos de posición del Doraemon.




(a) (b)

Figura 6.4: Distancia recorrida según el umbral definido.

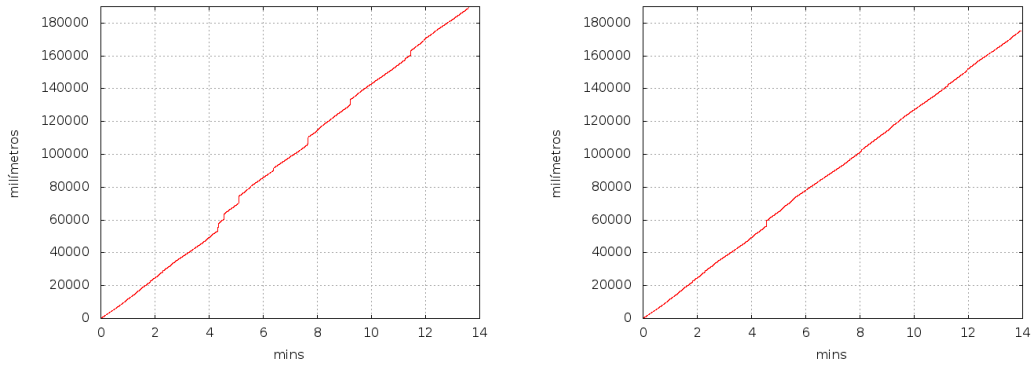
Además, para evitar que durante los periodos de quietud del robot, las mismas casillas sean contabilizadas varias veces para la métrica de cubrimiento útil total, solo se realizó el conteo en los pasos en los que se supera este umbral de distancia mínimo establecido.

***Outliers*** Adicionalmente, durante el análisis de los datos se detectó que los datos proporcionados por Doraemon presentaban *outliers* esporádicos. En función de esto se decidió filtrar los datos utilizando un modelo de velocidad constante. En este modelo, se descartan todos los datos que impliquen que el robot viajó a una distancia mayor a $1m/s$, umbral que resulta sumamente conservador, pero permite filtrar los datos inconsistentes.

En la figura 6.5 se puede observar el incremento en la distancia recorrida para una de las ejecuciones tomada como caso de estudio para el filtrado de los datos. Se incluyen las gráficas sin filtrado y con filtrado para apreciar la diferencia en la calidad de los datos luego de filtrados.

Los *outliers* provocan saltos en la gráfica de suma de distancias, debido a que la posición cambia a un punto distante repentinamente. Se observa en las gráficas que existen varios saltos en los datos sin filtrar y solo un salto en los datos filtrados.

Se observa que si bien existe un salto en la gráfica filtrada, este corresponde con un periodo en el que Doraemon perdió la pista del robot, y cuando lo ubicó nuevamente, el robot había viajado una distancia considerable.




(a) Datos de suma de distancia sin filtrar *outliers*.

(b) Datos de suma de distancia luego de realizar el filtro de *ouliers*.

Figura 6.5: Filtrado de la información para descartar *outliers*.

### 6.1.2. Resultados

Se presentan los resultados correspondientes a las pruebas de cubrimiento con un límite de tiempo de ejecución de 20 minutos, y luego un segundo conjunto de pruebas de 100 minutos de duración por ejecución.

#### 6.1.2.1. Pruebas de cubrimiento de 20 minutos

A continuación se incluye una tabla resumiendo los resultados según las métricas planteadas.

| | Integración de Odometría | | SLAM | |
|---|---|---|---|---|
| Métrica | Media | Varianza | Media | Varianza |
| Cubrimiento Útil ( % total útil promedio) | 92, 24 % | 3,23 % | 86, 82 % | 2,23 % |
| Cubrimiento Inútil ( % total inútil promedio) | 14, 50 % | 3,28 % | 4, 64 % | 1,74 % |
| Cubrimiento Útil Total ( % total útil promedio) | 2035, 92 % | 935,82 % | 1788, 30 % | 46,11 % |
| Eficiencia de Recorrido ( % total útil promedio / metro) | 73, 86 % | 3,48 % | 79, 82 % | 0,72 % |
| Ejecuciones terminadas por error | 4/5 | - | 0/5 | - |

Cuadro 6.1: Resultados de pruebas de 20 minutos.

A continuación se incluye una breve interpretación de los resultados para cada una de las métricas.
En estas pruebas no se realizó una comparación de la estimación de la posición.

**Cubrimiento Útil** Al diseñar los experimentos, se esperaba que la corrección del error realizada por el sistema de SLAM contribuyera a la obtención de un cubrimiento útil más alto por parte de este. Sin embargo, atribuimos los resultados principalmente a dos factores:

- El sistema de SLAM necesita de un tiempo considerable de ejecución para permitir que el mapa converja. En este periodo, se observa como el robot deja "huecos" en el recorrido del entorno debido a sucesivas

correcciones de la posición, que lo llevan a correcciones de la trayectoria, como se muestra en la figura 6.6.

- El sistema de SLAM toma demasiado tiempo para realizar el sensado de las marcas utilizando la cámara. Debido a esto, no logra recorrer tanta distancia como el sistema de odometría.

En consecuencia, se decidió realizar un segundo conjunto de ejecuciones de mayor duración, que permitieran al sistema de SLAM converger a un mapa y proporcionar un estimado estable de la posición al sistema de recorrido.

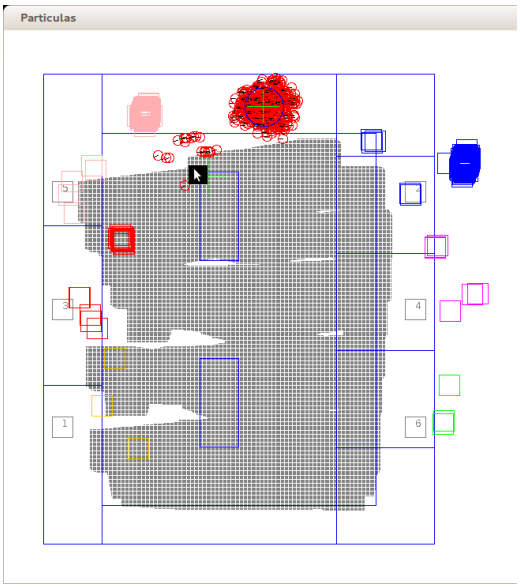



Figura 6.6: Cubrimiento de una recorrida de SLAM.
En gris las casillas cubiertas. Pueden observarse los espacios vacíos dejados durante la primera recorrida, debido a constantes correcciones en la posición.

**Cubrimiento Inútil** Esta métrica refleja los resultados esperados. Debido a que el sistema basado en odometría tiende a acumular los errores cometidos en la estimación del movimiento, es de esperar que se salga de la zona útil de cubrimiento con más frecuencia.

**Cubrimiento Útil Total** A pesar de que es esperable que el cubrimiento útil total sea mayor para el sistema de odometría, dado que el cubrimiento útil es también mayor, observamos que esto ocurrió aún cuando 4 de 5 corridas del sistema de odometría terminaron por errores del robot antes de tiempo. Esto indica que el sistema de odometría es sensiblemente más eficiente que el de SLAM en términos de tiempo, cuando se lo acota a un periodo de 20 minutos.

**Eficiencia de Recorrido** Esta métrica muestra como, a pesar de las diferencias de eficiencia en relación al tiempo (velocidad), ambos sistemas resultaron equivalentes en relación a la eficiencia en función de la distancia recorrida. Atribuimos la leve diferencia en esta métrica al cubrimiento inútil realizado por el sistema de odometría.

**Ejecuciones Terminadas por Error** Esta métrica refleja la robustez proporcionada por el sistema de SLAM, que mantiene al robot dentro de la zona útil, no permitiendo que este salga del entorno del trabajo. El sistema de odometría, en cambio, acumula error en la estimación de la posición, lo que provoca un error creciente en la orientación del robot que causa que este se acerque a las paredes con ángulos demasiado oblicuos, no pudiendo sensarlas, provocando que este salga del entorno.

**Evolución del Cubrimiento Útil** En la figura 6.8 se puede observar cómo el cubrimiento de odometría crece rápidamente hasta alcanzar tempranamente un tope mientras que el de SLAM mantiene un crecimiento constante pero no llega a estacionarse. Se atribuye esto a dos factores principales:

- La demora causada por la observación de las marcas en el sistema de SLAM entorpece el cubrimiento del terreno y lo hace más lento, no llegando a su máximo en el tiempo estipulado

- El sistema de odometría cubre rápidamente la zona útil. El quiebre en el crecimiento observado alrededor del minuto 4 se atribuye a que el sistema de odometría llega rápidamente a hacer una pasada de la zona útil y comienza a recorrer entornos ya visitados

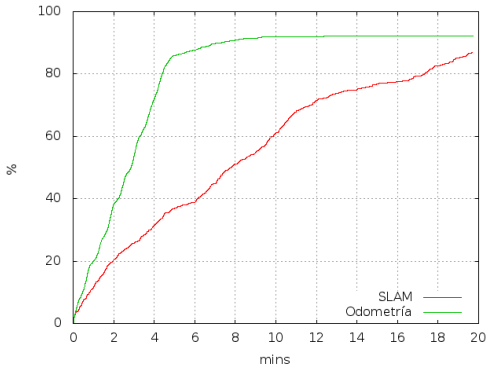



Figura 6.7: Evolución del cubrimiento útil promedio para ambos sistemas.

Además, se estudió la evolución del cubrimiento útil total con el paso del tiempo. En la figura 6.8 se puede observar como, de forma coherente con la evolución del cubrimiento útil, el cubrimiento útil total del sistema de SLAM aumenta de forma constante, mientras que el del sistema de Odometría presenta un quiebre luego de crecer más rápidamente que el de SLAM. Se atribuye este quiebre a la terminación temprana de varias de las ejecuciones. Si bien se proyecta en el largo plazo el cubrimiento útil total del sistema de SLAM superaría al de Odometría, el tiempo adjudicado a estas pruebas no es suficiente para comprobarlo.

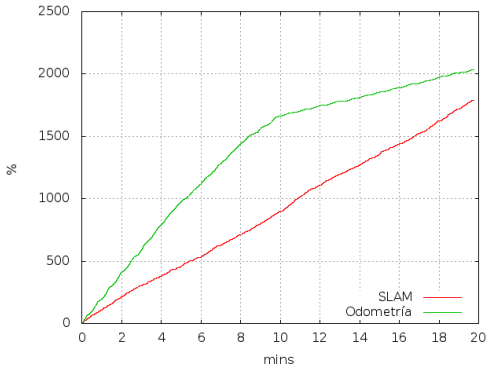



Figura 6.8: Evolución del cubrimiento útil total promedio para los sistemas de SLAM y Odometría.

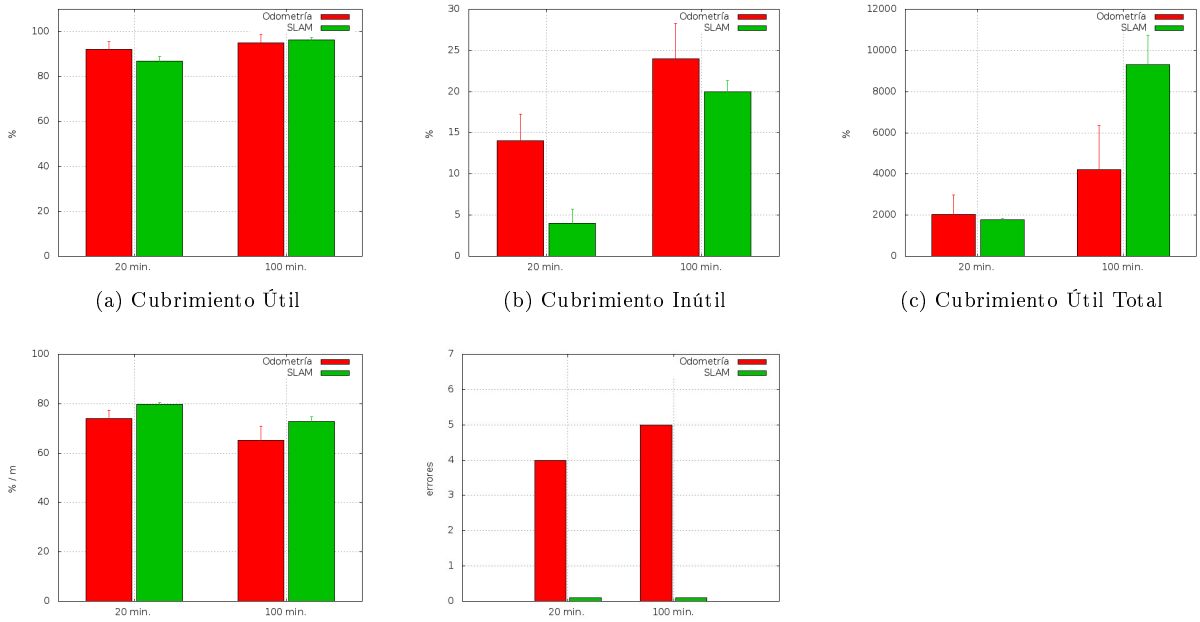
#### 6.1.2.2. Pruebas de cubrimiento de 100 minutos

A continuación se incluye una tabla resumiendo los resultados para las métricas planteadas.

| | Integración de Odometría | | SLAM | |
|---|---|---|---|---|
| Métrica | Media | Varianza | Media | Varianza |
| Cubrimiento Útil ( % total útil prom.) | 95, 13 % | 3,61 % | 96, 25 % | 1,17 % |
| Cubrimiento Inútil ( % total inútil prom.) | 24, 39 % | 4,31 % | 20, 27 % | 1,35 % |
| Cubrimiento Útil Total ( % total útil prom.) | 4218, 16 % | 2146,22 % | 9320, 36 % | 1427,16 % |
| Eficiencia de Recorrido ( % total útil prom. / metro) | 65,22 % | 5,55 % | 72,83 % | 1,87 % |
| Ejecuciones terminadas por error | 5/5 | - | 0/5 | - |
| Error RMS (metros) | 0,64 | 0,26 | 0,42 | 0,18 |

Cuadro 6.2: Resultados de pruebas de 100 minutos.

A fin de facilitar la comparación de resultados con las pruebas de 20 minutos, se incluyen en la figura 6.9 gráficos de comparación de los resultados de ambas pruebas.




(a) Cubrimiento Útil

(b) Cubrimiento Inútil

(c) Cubrimiento Útil Total

(d) Eficiencia de Recorrido

(e) Ejecuciones con error

Figura 6.9: Comparación de resultados entre pruebas de 20 minutos y 100 minutos.

A continuación se incluye una breve interpretación de los resultados de cada una de las métricas, comparando con los resultados obtenidos en las pruebas de 20 minutos.

**Cubrimiento Útil** Con el aumento del límite de tiempo, el sistema de SLAM logró alcanzar el rendimiento del sistema de odometría. Si bien el rendimiento final es aún mayor, se considera que la diferencia es marginal.

**Cubrimiento Inútil** Con respecto a esta métrica se observa un incremento en ambos sistemas, que se adjudica al incremento en el tiempo de ejecución. Se observa que el sistema de SLAM sufrió un incremento netamente mayor, creciendo al cuádruple de su valor anterior, mientras que el incremento del sistema de odometría fue menor al doble. Adjudicamos este hecho a que el incremento en el tiempo de ejecución total del sistema de SLAM fue mucho mayor, y así también el recorrido total.

Se observa, sin embargo, que la relación entre cubrimiento inútil y cubrimiento útil total disminuyó en ambos casos, como se puede observar en la figura 6.10. Además, se observa que la mejor relación entre estas métricas ocurre en el sistema de SLAM ejecutando durante 100 minutos.

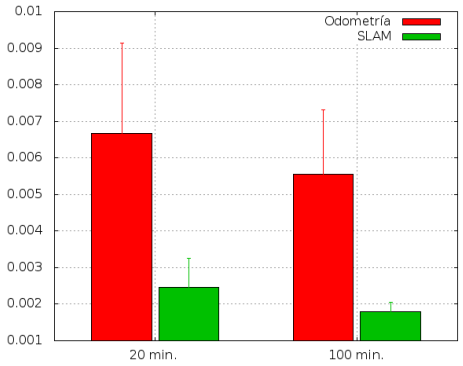


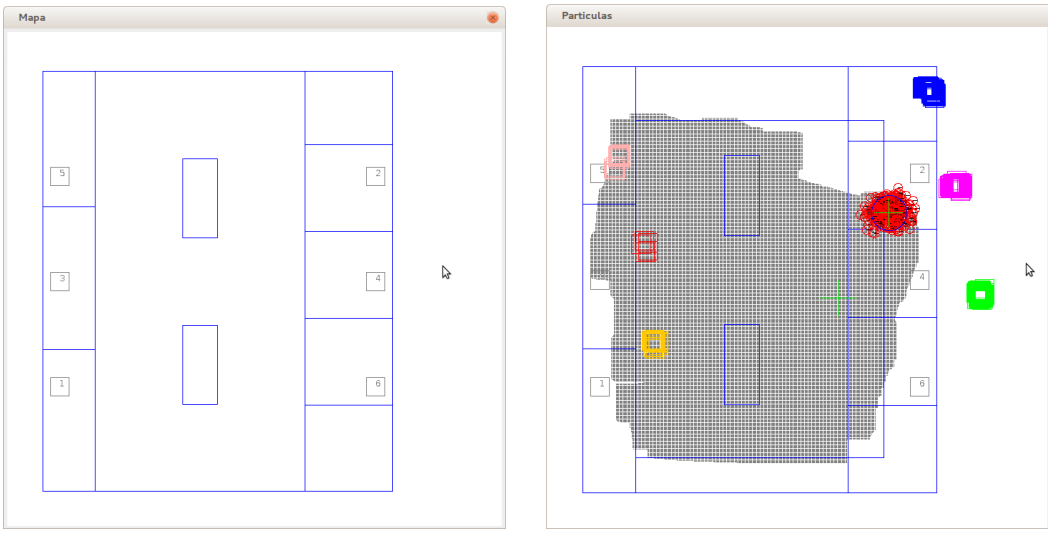

Figura 6.10: Relación entre cubrimiento inútil y cubrimiento útil total.

**Cubrimiento Útil Total** Esta métrica refleja un gran cambio en los resultados como fruto del incremento del límite de tiempo de ejecución de las pruebas. Se observa que el sistema de SLAM obtuvo un incremento considerable del cubrimiento útil total, superando al sistema de odometría. Se adjudica esta diferencia a que el sistema de SLAM se mantuvo ejecutando los 100 minutos, mientras las pruebas de odometría terminaron todas tempranamente.

**Eficiencia de Recorrido** Los autores entienden que las diferencias en la eficiencia de recorrido son despreciables. Se adjudica la leve disminución de la eficiencia a un leve aumento en la frecuencia de ocurrencia de pequeños errores que provocaron que el robot cubriera pequeñas porciones de zona inútil.

**Ejecuciones terminadas por error** En este caso, se observó que todas las ejecuciones del sistema de Odometría terminaron por error, siendo el promedio de duración de estas $23,56$ minutos. Todas las ejecuciones del sistema de SLAM terminaron a los 100 minutos.

**Error RMS** Se observa una mejor estimación de la posición por parte del sistema de SLAM. Sin embargo, el error RMS obtenido se considera demasiado alto para las dimensiones del entorno navegado. Atribuimos este resultado a un fenómeno observado durante la ejecución de las pruebas. Se observó que en alguna de las ejecuciones, el mapa construido por el sistema de SLAM mantenía un buen estimado de la relación espacial entre las marcas, pero el mapa como un conjunto se encontraba transportado y rotado con respecto al real, como puede observarse en la figura 6.11.




(a) Mapa real del entorno.

Figura 6.11: Mapa real y mapa construido por el sistema de SLAM.
Se observa que el mapa está rotado en sentido anti-horario y transportado levente hacia arriba.

Se observa una rotación entre

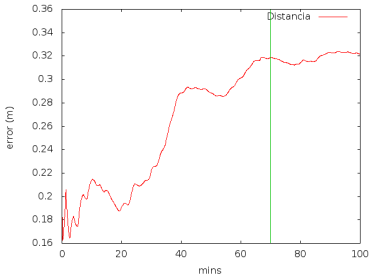
Al existir una diferencia tan grande entre el mapa construido y el mapa real, el estimado de la posición se encontrará siempre a una gran distancia de la posición real, a pesar de ser coherente con el mapa construido.

A fin de profundizar en la causa de este alto error RMS se realizó un estudio del error cometido en la estimación. En las figuras 6.12 y 6.13 se puede observar la evolución del error cometido por la estimación. En la primera se muestra la distancia lineal entre la estimación de la posición y la real. En la segunda se muestra la rotación relativa entre estos dos puntos.

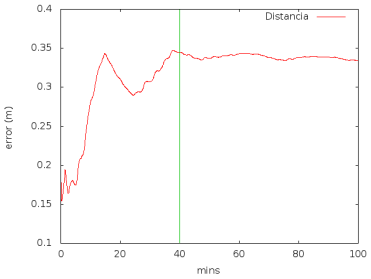
Con estas gráficas se busca determinar si el error cometido en la estimación se debe a la diferencia entre los marcos de referencia del sistema de SLAM y el del Doraemon. De ser así, la estimación de la posición debería converger a una traslación de la posición real, que reflejaría la diferencia entre los sistemas de coordenadas. Por ende, la diferencia de distancia lineal y ángulo relativo debería converger a valores relativamente constantes.

En el caso de las tres primeras ejecuciones, tanto la distancia lineal como la rotación relativa parecen converger a un valor relativamente constante. Esto podría corresponderse a que el error en la estimación está realmente causado por un cambio en el sistema de coordenadas del sistema de SLAM con respecto al original.

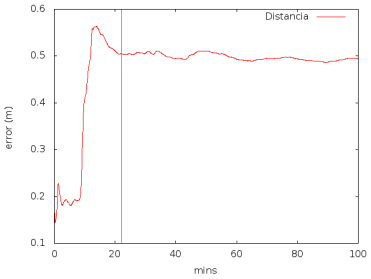
El error de la cuarta ejecución no pareciera corresponderse sólo a una diferencia en el marco de referencia. También podría considerarse así para la quinta ejecución, que parece mostrar una convergencia en la rotación relativa pero no hay ningún signo de estabilización de la distancia lineal. No es posible concluir que el gran error RMS se debe sólo a una traslación del marco de referencia, para estas últimas dos ejecuciones.



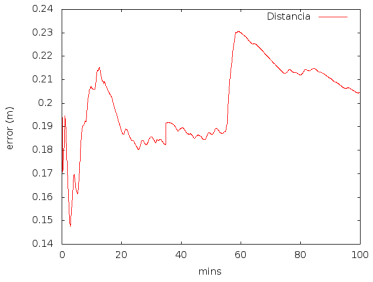

(a) Ejecución 1.



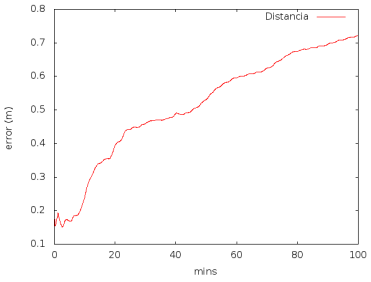

(b) Ejecución 2.




(c) Ejecución 3.

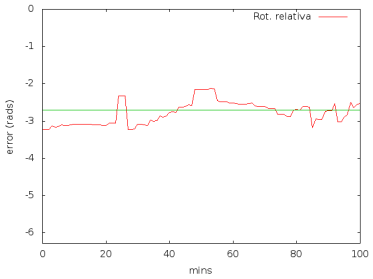



(d) Ejecución 4.

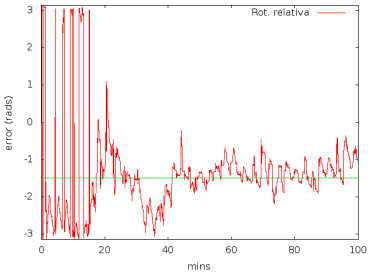



(e) Ejecución 5.

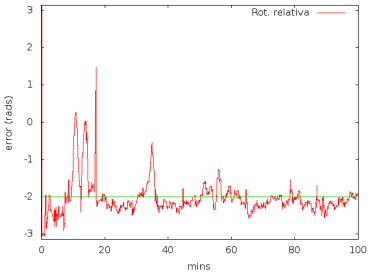
Figura 6.12: Evolución del error promedio de estimación.



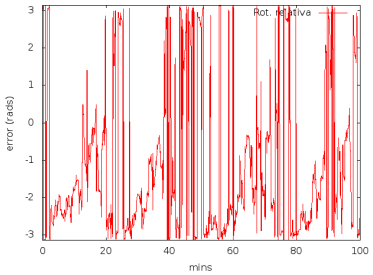

(a) Ejecución 1.



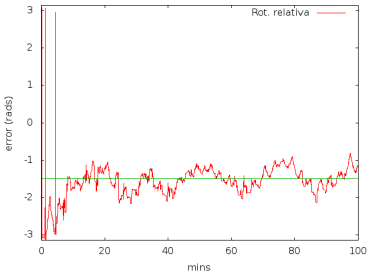

(b) Ejecución 2.




(c) Ejecución 3.




(d) Ejecución 4.




(e) Ejecución 5.

Figura 6.13: Evolución de la rotación relativa entre la estimación de la posición y la real.

Se concluye que este tipo de métrica no resulta adecuada a este contexto, pues compara la estimación de la posición en un sistema de coordenadas con la posición brindada por la visión global, en otro sistema de coordenadas. No resulta posible, sin embargo, concluir que el gran error RMS se corresponde sólo con este

factor.

**Evolución del cubrimiento útil** En las figuras 6.14 y 6.15 se puede observar como se cumplieron las hipótesis planteadas luego de obtenidos los resultados de las pruebas de 20 minutos. Existe un punto de corte en un valor mayor a 20 minutos en el que el sistema de SLAM se vuelve más eficaz en el cubrimiento útil y cubrimiento útil total.

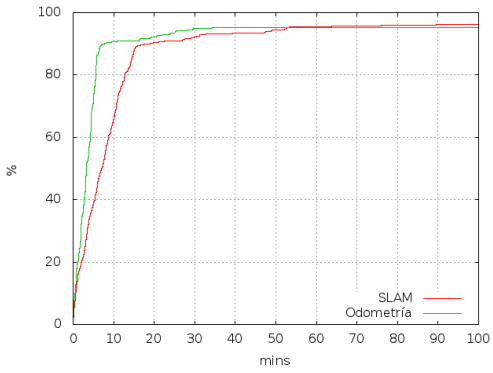



Figura 6.14: Evolución del cubrimiento útil promedio para los sistemas de SLAM y Odometría.

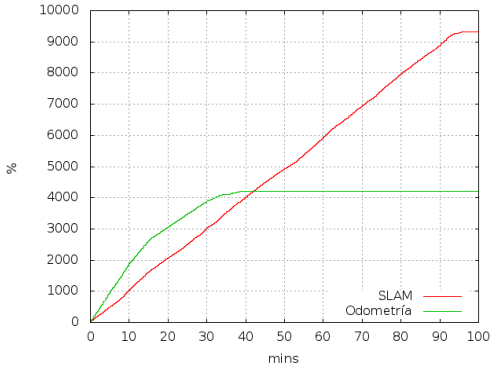



Figura 6.15: Evolución del cubrimiento útil total promedio para los sistemas de SLAM y Odometría.

## 6.2. Pruebas cualitativas de mapa generado

Debido a que durante el desarrollo de las pruebas de cubrimiento se observó que los mapas generados por el sistema de SLAM presentaban distorsiones, se realizaron pruebas adicionales.

Se planteó la hipótesis que debido a que el robot realiza un gran recorrido entre que observa las primeras marcas la primera y la segunda vez, la calidad del mapa generado se ve afectada, porque el espacio de búsqueda puede ser demasiado grande para la cantidad de partículas utilizadas. Este fenómeno es comparable con el problema de cerrado de ciclos del SLAM (en inglés, *loop closure*).

### 6.2.1. Metodología

En consecuencia se planteó realizar un recorrido diseñado *a priori* con el fin de permitir al robot elaborar un mejor mapa. Este recorrido intenta visitar varias veces seguidas las marcas de un subconjunto de las

marcas totales, sin realizar un gran recorrido. De esta forma, la información de relación entre las marcas se propagaría rápidamente sin dejar lugar a la generación de errores por la presencia de estos "ciclos".

Luego de realizado el recorrido, el robot continuaría con su tarea de cubrimiento partiendo del lugar donde se encuentra al finalizarlo.

### 6.2.2. Resultados

Luego de realizar 3 ejecuciones de 100 minutos utilizando este recorrido no se observaron grandes mejoras en el mapa generado.

En la figura 6.16 se puede observar un mapa generado en una de estas ejecuciones. En este mapa se observa también una rotación con respecto al mapa real y una leve traslación en el sentido creciente del eje $y$.

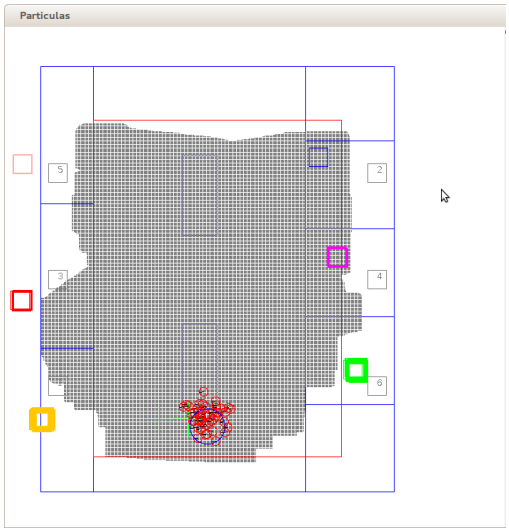



Figura 6.16: Mapa obtenido luego de realizar el recorrido programado *a priori*.

## 6.3. Conclusiones de las pruebas

Basándonos en los resultados de la prueba de 20 minutos es posible concluir que el sistema de odometría resulta más eficiente para realizar un rápido recorrido del terreno. El sistema de SLAM resulta ineficiente en un principio debido a las demoras producidas por las observaciones de la cámara y a la inestabilidad en la posición estimada, antes de que el mapa construido converja.

La información de corrección del error del sistema de SLAM le permite evitar mejor el recorrido de zonas no útiles.

También se observó que el sistema de SLAM presenta un mayor nivel de robustez en estas pruebas, terminando todas las corridas sin salirse del entorno a recorrer.

De las pruebas de 100 minutos se concluye que el sistema de SLAM permite realizar la tarea designada de forma robusta y finalmente con mayor eficacia y precisión.

Del análisis cualitativo de los mapas se concluye que las transformaciones de traslación y rotación que presentan los mapas construidos no se debe a la existencia de grandes ciclos en el recorrido.

En cambio, se adjudica este fenómeno a la diferencia de marcos de referencia entre el mapa generado por SLAM y el marco de referencia absoluto de Doraemon.

# Capítulo 7

# Conclusiones y Trabajos Futuros

En este último capítulo se presentan las conclusiones y trabajos a futuro.

## 7.1. Conclusiones

Se construyó un sistema robótico autónomo que realiza la tarea de *online* SLAM topológico, en el contexto de la resolución de un problema de cubrimiento, en un entorno controlado. Este sistema demuestra ser más robusto que utilizar el sistema de odometría para realizar la localización en el contexto del problema planteado.

El sistema de SLAM diseñado cumple los objetivos de ser una plataforma flexible ya que los modelos de movimiento y modelo de sensado se hicieron como subsistemas independientes que se integran al núcleo del SLAM a través de interfaces, de forma de facilitar mejoras o cambios en los modelos.

Además, se logró un desarrollo fácil de depurar debido a la implementación de un protocolo de comunicación a través de comandos, de forma de reproducir el procesamiento realizado en el robot, pero en una PC externa, lo que permite simular las corridas para su posterior análisis.

Se implementó un sistema de parametrización del sistema de SLAM utilizando algoritmos evolutivos. Esto permitió encontrar los parámetros del sistema que minimizan el error en la estimación. Los parámetros óptimos arrojados por el algoritmo resultaron contradecir los estimados por los autores.

Como logros adicionales se pueden mencionar la adquisición e interiorización de conocimientos de robótica probabilística y SLAM. Los autores entienden que el desarrollo de este campo es vital para el desarrollo de la robótica en Uruguay.

Adicionalmente, dado que se utilizó una plataforma de robótica educativa, los conocimientos adquiridos y el software desarrollado puede ser utilizado para enriquecer los cursos de robótica dictados por la Facultad de Ingeniería.

Además, durante el desarrollo de la solución se corrigieron defectos en algunas clases implementadas de LeJOS, particularmente en la clase de manejo de la cámara. Los autores pretenden colaborar con este software de código abierto a fin de aportar el conocimiento generado durante el proyecto.

## 7.2. Trabajos Futuros

A continuación se incluyen las posibles mejoras al sistema.

### 7.2.1. Rigidez del mapa final

La implementación de Filtros de Kalman para la estimación de cada marca disminuyen la varianza de la gaussiana del estado oculto en cada actualización. Una de las posibles mejoras que se le puede hacer al sistema es que la confianza de las marcas disminuya con el pasar del tiempo, de forma que las nuevas observaciones tengan más peso de lo que tienen actualmente.

### 7.2.2. Modelo de sensado y cámara

En función de los resultados obtenidos, se considera el cambio del modelo de sensado como una tarea prioritaria. En primer lugar se propone cambiar la cámara utilizada, debido a que se considera que su utilización presenta una de las principales limitantes para escalar el sistema a un entorno real.

Se plantea realizar un estudio del error en los modelo de estimación de la distancia y ángulo relativo a las marcas.

Adicionalmente, se propone utilizar bibliotecas robustas y probadas para la estimación de estos datos, como por ejemplo OpenCV[7]. La realización de este ítem debería ir antecedido del cambio de cámara propuesto en la sección 7.2.2.

Finalmente, se considera pertinente evaluar la factibilidad de utilizar sistemas automáticos de detección de marcas, a fin de eliminar la necesidad de colocar marcas artificiales en el entorno.

### 7.2.3. Pruebas adicionales

Se plantea realizar pruebas adicionales que consistan en completar un mapa parcial del entorno integrado a priori. Esta prueba tendría la finalidad de confirmar que el alto error RMS obtenido durante las pruebas realizadas se debe a un cambio en el marco de referencia. Al realizar estas pruebas se forzaría al sistema de SLAM a adoptar el marco de referencia del Doraemon, evitando este desfasaje.

Además, se considera pertinente la realización de pruebas de localización global una vez obtenido el mapa de un entorno.

Finalmente, se sugieren realizar pruebas donde se incremente el error de información proporcionado por el sistema de odometría, para evaluar la robustez de SLAM ante errores en la información de movimiento propioceptivo.

### 7.2.4. Algoritmo genético

Se plantea la posibilidad de probar otras funciones de *fitness* para el algoritmo genético. En particular se considera importante incluir aspectos del mapa generado en la evaluación de la calidad de una calibración.

### 7.2.5. Integración con otros proyectos de grado

Los sistemas generados podrían integrarse con los productos finales de otros proyectos de grado como Salimo (fútbol de robot) y VisRob2.

Se podría implementar un sistema de localización basado en filtro de partículas, que tome datos exportados por el proyecto de visión, con la característica adicional que modele componentes dinámicos de un campo de fútbol como otros jugadores y la pelota.

### 7.2.6. Identificación de marcas libre de errores

Un punto débil del sistema implementado es el sistema de determinación de la identidad de una marca. Si la identificación de las marcas se realiza de forma incorrecta el sistema de SLAM puede diverger.

Existen varias técnicas utilizadas en la literatura para mitigar este riesgo. Existe una solución que realiza la identificación de varias marcas de forma simultanea disminuir la probabilidad de una falsa identificación[14]. Otras soluciones eliminan por completo el concepto de identidad de marca y resuelven esta incertidumbre como parte del problema de estimación.

Se sugiere implementar alguna de estas soluciones al sistema implementado.

### 7.2.7.

# Glosario

Agente: Se entiende por agente como una entidad capaz de percibir y modificar su entorno.

Calibración: Determinación de un conjunto de valores adecuados de parámetros de un sistema para que funcione como se espera.

Carton Plast: Material plástico disponible en láminas, similar al cartón

Covarianza: Matriz que contiene la covarianza entre los elementos de un vector, uno a uno.

Distribución de probabilidad no paramétrica: Distribución de probabilidad que no puede ser representada de forma completa utilizando un conjunto reducido de parámetros.

Distribución de probabilidad paramétrica: Distribución de probabilidad que puede ser representada de forma completa utilizando un conjunto reducido de parámetros.

Distribución multimodal: Distribución de probabilidad que puede presentar más de un máximo local

Distribución unimodal: Distribución de probabilidad que presenta un solo máximo

Encoders: Sensores que permiten conocer la posición del eje de un motor

Estado oculto: Variables a estimar que no pueden ser sensadas directamente. En SLAM estas son las referentes a la posición del robot y el mapa.

Filtro de Bayes: Conjunto de relaciones matemáticas que permiten resolver la estimación de un estado oculto de manera recursiva, procesando la información en línea.

Filtro de Kalman: Filtro bayesiano que considera que todas las variables relacionadas siguen una distribución gaussiana.

Filtro de Partículas: Filtro bayesiano que mantiene una representación de la distribución de probabilidad del estado oculto manteniendo un conjunto grande de muestras de esa distribución.

Full SLAM: Problema de SLAM que consta en determinar la posición y mapa a lo largo del tiempo tomando en cuenta toda la información disponible al momento.

GPS: Sistema que permite conocer la posición de un objeto móvil gracias a la recepción de señales emitidas por una red de satélites.

Luz fluorescente: Es la que proviene de luminaria que cuenta con una lámpara de vapor de mercurio a baja presión y que es utilizada normalmente para la iluminación doméstica e industrial. Su gran ventaja frente a otro tipo de lámparas, como las incandescentes, es su eficiencia energética

Marca: Objeto distinguible en el entorno. Son utilizadas en la navegación como objetos de referencia.

Marginalización: Aplicado a una distribución de dos variables, el término marginalización refiere a descartar la información de una variable Y en una distribución conjunta p(X,Y), usualmente integrando toda la información aportada por Y para llegar a una distribución de una variable p(X)

Media: Parámetro de una distribución de probabilidad correspondiente al valor esperado.

Modelo de movimiento: Distribución de probabilidad que vincula el estado oculto (posición y mapa) en un instante con el siguiente, tomando en cuenta la última información de movimiento propioceptivo disponible.

Modelo de odometría: Modelo de movimiento que se basa en información de odometría para estimar el movimiento, descomponiendolo en una traslación y dos rotaciónes.

Modelo de sensado: Distribución de probabilidad que vincula el estado oculto (posición y mapa) con la información a sensar. Establece la probabilidad de sensar una medida z en función de la posición del robot y el mapa.

Métodos de Montecarlo: Método no determinista de estimación aproximada de valores cuyo cálculo resulta difícil de computar de manera exacta.

NASA: en español, Agencia Nacional de Administración del Espacio

Navegación: Disciplina de la robótica que incluye la localización, la construcción de mapas, la planificación de movimientos y la selección de objetivos.

Octave: Software opensource dedicado al cálculo matricial, entre otros.

Odometría: La odometría se basa en poder obtener el desplazamiento realizado por la rueda asociada a un motor a partir de la medición de la vueltas realizada por el mismo

Online SLAM: Problema de SLAM que consta en determinar la última posición del robot y mapa, utilizando la posición y mapa anteriores y la última información disponible.

Outlier: Observación que se encuentra numéricamente distante del resto de los datos

Paradigma híbrido: Paradigma de programación de agentes robóticos que permite la ejecución concurrente de acciones reactivas y deliberativas

Pinhole Model: Modelo de cámara que permite relacionar la distancia a un objeto y sus tamaños reales y de imágen.

Predict: Paso de un filtro de partículas que aplica la última información de movimiento para actualizar la distribución de probabilidad del estado oculto.

Red bayesiana: Grafo correspondiente a variables estocásticas y su relación de dependencia. Puede representarse gráficamente.

Rueda loca: Rueda que gira libremente

Scanmatching: Refiere a una técnica que consta en encontrar una posición, en un mapa dado, que maximice la verosimilitud de los datos sensados, utilizando normalmente técnicas de hill-climbing

Sensor de inercia: Senor que permite conocer la aceleración instantánea del robot.

SLAM: Localización y construcción de mapas en simultáneo, del inglés Simultaneous Localization And Mapping.

Update: Paso de un filtro de partículas que aplica la última información de sensado para actualizar la distribución de probabilidad del estado oculto.

Varianza: Parámetro de una distribución de probabilidad que puede ser interpretado como una medida de la dispersión de la distribución.

Volumétrico: Representación del entorno que lo modela como un conjunto denso de puntos o celdas.

Waypoints: Puntos distinguibles que establecen una ruta a navegar

# Bibliografía


[1] Kit LEGO MINDSTORM versión 2.0 - http://shop.lego.com/en-us/lego-mindstorms-nxt-2-0-8547. [Visitada en Junio 2012].

[2] Sitio web de Conjunto de Datos de Freiburg - http://kaspar.informatik.uni-freiburg.de/ slamevaluation/-datasets.php. [Visitada en Junio 2012].

[3] Sitio web de Doraemon - sourceforge.net/projects/robocup-video/. [Visitada en Junio 2012].

[4] Sitio web de FoxBoard G20 - http://www.acmesystems.it/index_foxg20. [Visitada en Junio 2012].

[5] Sitio web de iRobot - www.irobot.com. [Visitada en Junio 2012].

[6] Sitio web de JGAP - http://jgap.sourceforge.net/.

[7] Sitio web de OpenCV - opencv.willowgarage.com. [Visitada en Junio 2012].

[8] Sitio web de planos CasaConstruir - http://www.casasconstruir.com/2010/06/arquitectos.html. [Visitada en Junio 2012].

[9] Sitio web de Robots in Search - robotsinsearch.com. [Visitada en Junio 2012].

[10] Sitio web del Cluster de Facultad de Ingeniería - www.fing.edu.uy/cluster/. [Visitada en Junio 2012].

[11] Sitio web del Framework leJOS - http://lejos.sourceforge.net/. [Visitada en Junio 2012].

[12] Sitio web leJOS Subsumption - http://lejos.sourceforge.net/, visita en mayo 2012. [Visitada en Junio 2012].

[13] Sitio Web técnico de LEGO MINDSTORM - http://mindstorms.lego.com/en-us/support/files/default.aspx. [Visitada en Junio 2012].

[14] Tim Bailey. Mobile robot localisation and mapping in extensive outdoor environments, 2002.

[15] Christian Blum y Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003.

[16] Javier Civera y Andrew J Davison. 1-Point RANSAC for EKF Filtering. Application to Real-Time Structure from Motion and Visual Odometry. *Journal of Field Robotics*, 27(5):609–631, 2010.

[17] Facultad de Ingeniería. Sitio web de propuestas a proyectos de grado 2011 http://www.fing.edu.uy/inco/cursos/proygrado/bosquejos2011.html.

[18] Hugh Durrant-Whyte y Tim Bailey. Simultaneous Localization and Mapping (SLAM): Part I The Essential Algorithms. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, 2:2006, 2006.

[19] Udo Frese. Closing a million-landmarks loop. In *In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing.*, pages 5032–5039, 2006.

[20] G. Grisetti, C. Stachniss, y W. Burgard. Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. *Robotics, IEEE Transactions on*, 23(1):34–46, February 2007.

[21] Jose Guivant y Eduardo Nebot. Optimization of the Simultaneous Localization and Map Building Algorithm for Real Time Implementation. *IEEE Transactions on Robotics and Automation*, 17:242–257, 2001.

[22] Jose Guivant y Eduardo Nebot. Optimization of the Simultaneous Localization and Map Building Algorithm for Real Time Implementation. *IEEE Transactions on Robotics and Automation*, 17:242–257, 2001.

[23] Christoph Hertzberg. A Framework for Sparse, Non-Linear Least Squares Problems on Manifolds. Diplomarbeit, Universitat Bremen, 2008.

[24] Shoudong Huang, Zhan Wang, Gamini Dissanayake, y Udo Frese. Iterated SLSJF: A Sparse Local Submap Joining Algorithm with Improved Consistency. In *Proceedings of the Australasian Conference on Robotics and Automation, Canberaa*, 2008.

[25] M. Jamzad, B. S. Sadjad, V. S. Mirrokni, M. Kazemi, H. Chitsaz, A. Heydarnoori, M. T. Hajiaghai, y E. Chiniforooshan. A fast vision system for middle size robots in robocup. In *In 5th International Workshop on RoboCup 2001 (Robot World Cup Soccer Games and Conferences), number 2377 in Lecture Notes in Computer Science*, pages 71–80. Springer, 2002.

[26] M. Kaess, A. Ranganathan, y F. Dellaert. iSAM: Incremental Smoothing and Mapping. *Robotics, IEEE Transactions on*, 24(6):1365 –1378, 2008.

[27] Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[28] M. Llofriu y F. Andrade. Estado del arte del SLAM. 2012.

[29] G. Luque, E. Alba, y B. Dorronsoro. *Parallel Metaheuristics: A New Class of Algorithms*, pages 107–126. Wiley Series on Parallel and Distributed Computing. Wiley, 2005.

[30] Adrian Martinez-Gomez y Alfredo Weitzenfeld. Real time localization in four legged robocup soccer. *VII SBAI/ II IEEE LARS.*, 2005.

[31] M J Milford y G F Wyeth. Mapping a Suburb With a Single Camera Using a Biologically Inspired SLAM System. *IEEE Transactions on Robotics*, 24(5):1038–1053, 2008.

[32] Michael Milford y Gordon Wyeth. Spatial Mapping and Map Exploitation: A Bio-inspired Engineering Perspective. In *Spatial Information Theory*, pages 203–221. Springer Berlin / Heidelberg, 2007.

[33] Michael Montemerlo, Sebastian Thrun, Daphne Koller, y Ben Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002.

[34] Michael Montemerlo, Sebastian Thrun, Daphne Koller, y Ben Wegbreit. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. In *In Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, pages 1151–1156, 2003.

[35] Robin R. Murphy. *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA, 1st edition, 2000.

[36] Stuart Russell y Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, 3 edition, December 2009.

[37] Bruno Siciliano y Oussama Khatib, editores. *Springer Handbook of Robotics*. Springer, 2008.

[38] H. Strasdat, J. M. M. Montiel, y A. Davison. Scale Drift-Aware Large Scale Monocular SLAM. In *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010.

[39] Sebastian Thrun, Wolfram Burgard, y Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[40] Greg Welch y Gary Bishop. An Introduction to the Kalman Filter. Technical report, Chapel Hill, NC, USA, 1995.