



Universidad Católica
San Pablo

**FACULTAD DE INGENIERÍA Y
COMPUTACIÓN**

**DEPARTAMENTO DE CIENCIA DE LA
COMPUTACIÓN**

**Escuela Profesional de Ciencia de la
Computación**

**Una Estructura de Datos para la Consulta
Visual Interactiva por Similitud de Grandes
Volúmenes de Datos Multidimensionales Mixtos
Georeferenciados**

Tesis

Presentada por el bachiller

**** Nombre completo del tesista ****

Para Optar por el Título profesional de:

Licenciado en Ciencia de la Computación

Asesor: ** Dr./Mg. Nombre completo del Asesor **

**** Arequipa, Mes Año ****

Aquí deberás colocar a quien va dedicada tu tesis por ejemplo: A Dios, por todo lo que me ha dado, a todos los profesores por sus enseñanzas y algunos amigos.

Abreviaturas

CMM *Capability Maturity Model*

Agradecimientos

Aquí deberás colocar a quien y porque agradeces. Ejemplo:

En primer lugar deseo agradecer a Dios por haberme guiado a lo largo de estos cinco años de estudio.

Agradezco a mis padres por el apoyo brindado para forjarme como un profesional.

Agradezco a la universidad, mi *alma matter*, por haberme cobijado y brindado la formación que ahora me permitirá ayudar a construir una mejor sociedad.

Agradezco de forma muy especial a mi asesor Prof. Dr./Mag. nombre 1 por haberme guiado en esta tesis. ...

Deseo agradecer de forma especial a mis profesores: nombre 1, nombre 2, nombre 3 porque fueron ejemplos que deseo seguir en mi vida profesional.

Deseo agradecer al personal administrativo de la universidad: nombre 1, nombre 2, nombre 3. Muchas gracias por la atención brindada y porque siempre estuvieron dispuestas a ayudarnos.

Resumen

Aquí deberás colocar hasta 300 palabras como máximo, describiendo el problema que intentas resolver, la justificación, aportes o soluciones que planteas, qué tanto los haz alcanzado en calidad de resultados obtenidos.

Abstract

Here you should enter up to 300 words maximum, describing the problem you are trying to solve, the justification, contributions or solutions you are proposing, how much you have achieved them in terms of obtained outcomes.

Índice general

1. Introducción	2
1.1. Motivación y Contexto	2
1.2. Planteamiento del Problema	2
1.3. Objetivos	3
1.3.1. Objetivos Específicos	3
1.4. Organización de la tesis	3
2. Nombre del Capítulo II (Usualmente Marco Teórico)	4
2.1. Sección 1 del Capítulo II	4
2.1.1. Sub Sección	4
2.2. Recomendaciones generales de escritura	5
3. Propuesta	6
4. Propuesta	7
4.1. Motivación y limitaciones de los modelos generativos	7
4.2. Definición y componentes de RAG	7
4.3. Evolución y taxonomía de RAG	8
4.3.1. Mejoras y optimizaciones	8
4.4. RAG en entornos offline	9
4.5. RAG como servicio (RAGaaS)	9
4.6. Marcos de desarrollo y herramientas de código abierto	10

4.6.1. Haystack	10
4.6.2. LangChain	10
4.6.3. LlamaIndex	10
4.6.4. RAGFlow	11
4.6.5. txtAI	11
4.6.6. Cognita	11
4.6.7. Dify	11
4.6.8. Elección de framework	12
4.7. Modelos de embeddings, bases vectoriales y LLMs	12
4.8. Propuesta de implementación local	12
4.9. Conclusiones	13
5. Pruebas y Resultados	14
6. Conclusiones y Trabajos Futuros	15
6.1. Problemas encontrados	15
6.2. Recomendaciones	15
6.3. Trabajos futuros	15
Bibliografía	16

Índice de tablas

Índice de figuras

2.1. SLAM puramente topológico basado en grafos	5
---	---

Capítulo 1

Introducción

Este es el primer capítulo de la tesis. Se inicia con el desarrollo de la introducción de la tesis. Es importante que el texto utilice la tabla de abreviaturas correctamente. En el archivo `abreviaturas.tex` contiene la tabla de abreviaturas. Para citar alguna de ellas debes usar los comandos

`backslashactu-sigla-aqui`. Si es la primera vez que utilizas la sigla ella se expandirá por completo. Por ejemplo, el comando

`backslashacCMM` va a producir: *Capability Maturity Model* (CMM). Si más adelante repites el mismo comando sólo aparecerá la sigla CMM. Para explorar mucho más este comando es necesario leer su manual disponible en: [http : //www.ctan.org/tex – archive/macros/latex/contrib/acronym/](http://www.ctan.org/tex-archive/macros/latex/contrib/acronym/)

1.1. Motivación y Contexto

En esta sección se va desde aspectos generales a aspectos específicos (como un embudo). No se olvide que es la primera parte que tiene contacto con el lector y que hará que este se interese en el tema a investigar.

El objetivo de esta sección es llevar al lector hacia el tema que se va a tratar en forma específica y dejar la puerta abierta a otras investigaciones

1.2. Planteamiento del Problema

En esta sección se realiza el planteamiento del problema que queremos resolver con la tesis. Sea muy puntual y no ocupe más de un párrafo en especificar cual es el problema que desea atacar.

1.3. Objetivos

En esta sección se coloca el, o los objetivos generales de la tesis (Máximo dos). Si necesita detallar estos objetivos, remítase a la sección de objetivos específicos
downarrow.

1.3.1. Objetivos Específicos

En esta sección se coloca los objetivos específicos de la tesis, que serán aquellos que contesten a las preguntas de investigación, usualmente son 3 ó 4 objetivos específicos.

1.4. Organización de la tesis

Capítulo 2

Nombre del Capítulo II (Usualmente Marco Teórico)

Cada capítulo deberá contener una breve introducción que describe en forma rápida el contenido del mismo. En este capítulo va el marco teórico. (pueden ser dos capítulos de marco teórico)

2.1. Sección 1 del Capítulo II

Un capítulo puede contener n secciones.

Con respecto a las referencias bibliográficas, se hace de la siguiente manera [Mateos et al., 2000], se debe de redactar el texto de forma tal que si se retiraran las referencias, la redacción no debe perjudicarse. Por ejemplo

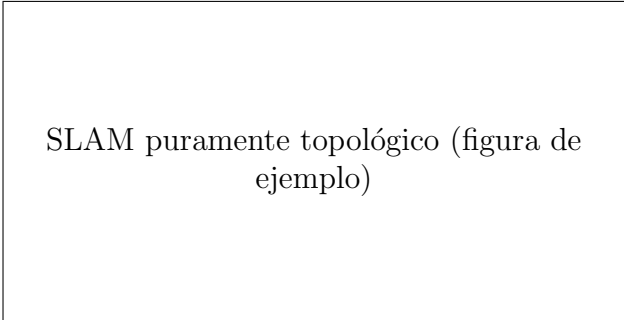
Localización y Mapeo Simultaneos SLAM puramente topológico

Se puede hacer un sistema SLAM con sólo reconocimiento de lugares, generando un SLAM topológico con representación basada en grafos [Choset and Nagatani, 2001]. Se almacena un registro de lugares visitados y cómo se llega de uno a otro (conexiones) sin tener información geométrica explícita.

Este tipo de SLAM es más adecuado para planificación y navegación. La figura 2.1 ilustra este tipo de SLAM

2.1.1. Sub Sección

Una sección puede contener n sub secciones.[Galante, 2001]

El diagrama muestra un rectángulo vacío con el texto "SLAM puramente topológico (figura de ejemplo)" centrado dentro. Este rectángulo representa un ejemplo de un grafo de SLAM puramente topológico.

SLAM puramente topológico (figura de ejemplo)

Figura 2.1: SLAM puramente topológico basado en grafos

Sub sub sección

Una sub sección puede contener n sub sub secciones.

2.2. Recomendaciones generales de escritura

Un trabajo de esta naturaleza debe tener en consideración varios aspectos generales:

- Ir de lo genérico a lo específico. Siempre hay que considerar que el lector podría ser alguien no muy familiar con el tema y la lectura debe serle atractiva.
- No redactar frases muy largas. Si las frases tienen más de 2 líneas continuas es probable que la lectura sea dificultosa.

Cada capítulo excepto el primero debe contener, al finalizar, una sección de consideraciones que enlacen el presente capítulo con el siguiente.

Capítulo 3

Propuesta

En este capítulo se desarrolla toda la propuesta realizada a través de la investigación. Sigue la misma estructura del capítulo anterior.

El título del capítulo es flexible de acuerdo a cada tesis. Algunos títulos sugeridos podrían ser:

- Algoritmo X: nuestra propuesta.
- Modelo MRLO

Este título debe de ser definido junto a su asesor de tesis. Consúltelo en su sala de clase.

Capítulo 4

Propuesta

En este capítulo se presenta un análisis exhaustivo del estado del arte en *Retrieval-Augmented Generation* (RAG) y se propone una arquitectura adaptada a las necesidades del proyecto. El objetivo es ofrecer un panorama actualizado de las tendencias más recientes (2023–2025) y sentar las bases para diseñar un sistema de consulta visual interactiva que combine recuperación eficiente y generación de lenguaje natural. Se describen los motivos que justifican el uso de RAG, su evolución, las variantes existentes (offline y como servicio), los marcos de desarrollo de código abierto y los modelos base que se emplean actualmente. Finalmente, se proponen lineamientos para una implementación local que elimine las dependencias de servicios en la nube y se adapte a un entorno de un único usuario.

4.1. Motivación y limitaciones de los modelos generativos

Los modelos de lenguaje de gran escala (*LLM*) han demostrado capacidades notables en tareas de generación y comprensión de texto. Sin embargo, estos modelos presentan limitaciones inherentes: su conocimiento está “congelado” en la fecha de corte del corpus de entrenamiento y no incorporan datos privados o recientes. Además, generan salidas probabilísticas que pueden contener errores u “alucinaciones” [? line:20]. Un análisis de Pinecone señala que los modelos de fundación carecen de profundidad en dominios especializados, no incluyen datos privados y no proporcionan citas de sus fuentes, lo que erosiona la confianza de los usuarios[916670030940373†L32-L92]. Estas limitaciones justifican la búsqueda de mecanismos que conecten a los LLM con fuentes de información externas y actualizadas.

4.2. Definición y componentes de RAG

Retrieval-Augmented Generation es una técnica que combina la búsqueda de información con la generación de lenguaje natural. Según el tutorial de SingleStore [? line:24],

RAG es un modelo que integra los procesos de recuperación y generación para mejorar la precisión de los LLM, mitigando las alucinaciones al incorporar datos externos relevantes. La guía de Pinecone resume este enfoque en cuatro componentes: (1) ingestión de datos de autoridad en una base vectorial; (2) recuperación de información relevante mediante búsquedas densa y dispersa; (3) aumento del mensaje combinando la consulta del usuario con la información recuperada; y (4) generación de la respuesta por el LLM utilizando el contexto aportado[916670030940373†L118-L133]. Este flujo permite proporcionar respuestas actualizadas, trazables y contextualizadas.

4.3. Evolución y taxonomía de RAG

Una revisión reciente [?, line:28] analiza la evolución de RAG y propone una taxonomía basada en la localización de las innovaciones. Se distinguen arquitecturas centradas en el recuperador, centradas en el generador y sistemas híbridos. Los sistemas *retriever-centric* mejoran la búsqueda mediante técnicas de redensificación, filtrado y búsqueda híbrida; los *generator-centric* ajustan la decodificación y la integración del contexto; mientras que los híbridos coordinan ambas etapas para equilibrar precisión y flexibilidad[589007710703999†L74-L93]. El estudio también revisa avances como la re-clasificación dinámica de resultados, el uso de filtros de diversidad y los mecanismos de seguridad ante entradas adversarias[589007710703999†L960-L981].

El preprint también enumera los modelos base empleados en los experimentos de referencia, entre ellos *LLaMA 2*, *LLaMA 3*, *GPT-3.5/4*, *Vicuna*, *Mistral*, *Mixtral*, *Gemini* y *Gemma*[589007710703999†L1016-L1020]. Esta diversidad muestra que la técnica es agnóstica al modelo subyacente y permite combinar LLM comerciales o de código abierto con recuperadores densos y dispersos.

4.3.1. Mejoras y optimizaciones

Las investigaciones recientes proponen varias mejoras para incrementar la eficacia del pipeline RAG. Entre las más destacadas se encuentran:

- **Búsqueda híbrida y filtrado de contexto:** combinar vectores densos con índices dispersos (BM25) para capturar tanto la similitud semántica como coincidencias exactas de términos[916670030940373†L238-L250]. Asimismo, se aplican filtros de diversidad o estrategias de *maximal marginal relevance* para reducir la redundancia y mejorar la cobertura.
- **Reordenamiento mediante re-rankers:** después de recuperar los primeros k documentos, se emplean modelos *cross-encoder* (por ejemplo, BGE Reranker o modelos similares) que asignan un puntaje de similitud más preciso a cada par consulta-documento. Pinecone señala que el re-ordenamiento se efectúa tras combinar los resultados de búsqueda densa y dispersa para devolver los fragmentos más relevantes[916670030940373†L246-L251].

- **Optimización de la generación:** se exploran estrategias como la generación condicional con mezcla de expertos, la reincorporación iterativa de evidencias o la restricción de la decodificación para reducir alucinaciones. El estudio de Sharma [?, line:39] discute técnicas de control de alucinación, adaptabilidad a dominios y seguridad frente a ataques adversarios[589007710703999†L960-L981].

4.4. RAG en entornos offline

La mayoría de los sistemas RAG se ofrece como servicios en la nube, pero existe un creciente interés por implementaciones locales que preserven la privacidad y funcionen sin conexión a internet. El blog de Alex Ho describe una experiencia de *RAG offline* utilizando Ollama, un entorno que permite ejecutar modelos de lenguaje localmente mediante GPU[246311036369843†L87-L96]. El autor narra cómo empleó *Llama 2* y *Llama 3* en su portátil y desarrolló un pequeño CLI que responde preguntas sobre notas Markdown almacenadas en su computadora[246311036369843†L104-L110]. Para preparar la base de datos personalizada, el flujo consiste en: leer los archivos, dividir el texto en fragmentos de longitud fija (con solapamiento para no perder contexto), generar embeddings con un modelo local (por ejemplo `nomic-embed-text`) y almacenarlos junto con metadatos en una base vectorial como Chroma[246311036369843†L204-L225]. Una vez indexados los vectores, el usuario puede generar consultas, las cuales se embeben y comparan con el índice; los fragmentos recuperados se envían a un modelo local para elaborar la respuesta. Este enfoque evita el coste de las API comerciales y ofrece control total sobre los datos.

4.5. RAG como servicio (RAGaaS)

Otra tendencia es el *RAG-as-a-Service*. Las empresas de infraestructura ofrecen plataformas que gestionan la ingesta, el indexado, la recuperación y la generación a través de una API. Un artículo de Vlad Koval expone que la promesa de RAGaaS consiste en permitir a los clientes “conectarse” a un servicio que maneja el escalado, las actualizaciones y la observabilidad de la recuperación, de modo que la organización pueda concentrarse en el caso de uso[430126204644756†L53-L62]. Este enfoque acelera la experimentación y resulta atractivo para equipos con recursos limitados[430126204644756†L82-L99]. No obstante, el autor advierte que se sacrifica el control sobre aspectos clave como la segmentación de documentos, la actualización de embeddings o la afinación de los re-rankers[430126204644756†L68-L76]. RAGaaS puede funcionar bien en pruebas piloto o para chatbots de soporte, pero en dominios regulados o cuando los costes de escala son elevados conviene optar por una implementación propia[430126204644756†L106-L114]. La página de Microsoft sobre Azure AI Search muestra un ejemplo de RAGaaS orientado a empresas: el patrón consiste en enviar la consulta del usuario a un servicio de búsqueda que devuelve resultados relevantes, los cuales se inyectan en el prompt del modelo generativo; se destacan la necesidad de estrategias de indexado, ajuste de relevancia, integración con modelos de embeddings y consideraciones de seguridad[316064638127775†L56-L78].

4.6. Marcos de desarrollo y herramientas de código abierto

El ecosistema de RAG cuenta con numerosas bibliotecas y plataformas que facilitan la construcción de aplicaciones. A continuación se resumen los principales marcos de código abierto según el informe de Designveloper [? line:52] y la guía de LangCopilot [? line:52]:

4.6.1. Haystack

Haystack, desarrollada por Deepset, es un marco modular para construir aplicaciones de preguntas y respuestas y flujos RAG. Permite conectar diversas tecnologías como OpenAI, Chroma y Hugging Face gracias a su arquitectura flexible[149667013130509†L319-L343]. Incluye componentes pre-construidos para generadores, recuperadores y embedders, así como un sistema de *pipelines* serializables que se pueden desplegar en Kubernetes. También ofrece una interfaz web (Hayhooks) para exponer pipelines como servicios y herramientas de trazado y evaluación[149667013130509†L344-L352].

4.6.2. LangChain

LangChain es una plataforma composable que facilita el desarrollo de agentes y cadenas de procesamiento para LLMs. Ofrece un IDE visual, plantillas de prompts y múltiples herramientas para resolver las complejidades técnicas[149667013130509†L359-L376]. Sus componentes incluyen modelos de chat, recuperadores, cargadores de documentos, almacenes vectoriales y utilidades de ingeniería de prompts[149667013130509†L366-L371]. Las cadenas son secuencias de componentes reutilizables que pueden combinarse y monitorizarse, lo que habilita flujos complejos y memoria conversacional. Además, se integra con otras herramientas de la familia Lang (LangSmith, LangGraph) que proporcionan depuración, orquestación y despliegue a escala[149667013130509†L383-L387].

4.6.3. LlamaIndex

LlamaIndex (antes *GPT Index*) se centra en la ingesta, indexado y consulta de datos empresariales. Simplifica la creación de asistentes de conocimiento mediante componentes modulares de embedding, carga, indexado y consulta[149667013130509†L393-L401]. Aunque su foco principal es el texto, incorpora soporte emergente para RAG multimodal a través de LlamaExtract y servicios de la plataforma LlamaCloud que permiten extraer datos estructurados de documentos complejos[149667013130509†L401-L415]. También ofrece funciones avanzadas para memoria conversacional, flujos de múltiples pasos y herramientas de evaluación y observabilidad[149667013130509†L407-L410].

4.6.4. RAGFlow

RAGFlow es un motor de RAG orientado a la comprensión profunda de documentos y a la generación de respuestas con citas. Soporta la integración de modelos mediante APIs y permite desplegar modelos locales mediante Ollama o Xinference, brindando flexibilidad entre rendimiento, coste y privacidad[149667013130509†L422-L440]. Dispone de un editor visual de nodos para diseñar flujos, incorpora funcionalidades de construcción de grafos de conocimientos, extracción automática de palabras clave y preguntas, y posibilita la observabilidad detallada de cada paso mediante integración con Langfuse[149667013130509†L430-L439].

4.6.5. txtAI

txtAI es un framework que integra búsqueda semántica, orquestación de LLM y flujos de lenguaje. Proporciona una base de embeddings que combina índices densos y dispersos y soporta grafos y estructuras relacionales[149667013130509†L446-L453]. Sus principales ventajas son: compatibilidad con embeddings multimodales (texto, audio, imágenes, video), capacidad para combinar pipelines en flujos complejos y disponibilidad de APIs web y MCP en varios lenguajes. Permite el despliegue local y escalable utilizando contenedores[149667013130509†L446-L465].

4.6.6. Cognita

Cognita es un marco de Truefoundry orientado a crear sistemas RAG escalables. Incluye parsers para diferentes tipos de datos, cargadores de datos desde diversas fuentes, embedders que soportan modelos de OpenAI y Cohere, re-rankers de última generación y un administrador de consultas capaz de gestionar múltiples peticiones y escalar los recursos automáticamente[149667013130509†L472-L489]. Su arquitectura modular facilita la integración de diferentes componentes y garantiza el cumplimiento de normas de seguridad y privacidad.

4.6.7. Dify

Dify es un marco de bajo código que ofrece una interfaz de arrastrar y soltar para desarrollar flujos agentes. Su *Backend-as-a-Service* gestiona la complejidad del desarrollo y transforma los flujos construidos en servicios accesibles mediante MCP[149667013130509†L496-L501]. Permite integrar modelos de OpenAI, Hugging Face, Grok o DeepSeek y ampliar las capacidades mediante *plugins* (Slack, QRCode, xAI, AWS, etc.)[149667013130509†L503-L507]. Además, soporta capacidades multimodales y estrategias inteligentes para orquestar tareas complejas.

4.6.8. Elección de framework

Elegir un marco RAG depende de múltiples factores. Designveloper sugiere considerar la licencia (preferir Apache 2.0 o MIT), el nivel de modularidad, la facilidad de uso (orientado a código o interfaz visual), la capacidad para manejar documentos largos y las integraciones disponibles[149667013130509†L518-L538]. Por ejemplo, Haystack y LangChain destacan en modularidad y soporte para backends diversos, mientras que RAGFlow y Dify ofrecen interfaces visuales orientadas a usuarios no desarrolladores. También es necesario evaluar la ventana de contexto soportada y la capacidad de procesar documentos extensos[149667013130509†L595-L599].

4.7. Modelos de embeddings, bases vectoriales y LLMs

Un sistema RAG requiere seleccionar modelos de embeddings, almacenes de vectores y modelos generativos apropiados. Existen modelos ligeros como *all-MiniLM-L6-v2* de Sentence Transformers o *BGE* que generan vectores de dimensión reducida y permiten la búsqueda eficiente. Para mejorar la precisión se suele añadir un re-ranker *cross-encoder*, como los modelos BGE *reranker*, que calculan la similitud entre la consulta y el documento de forma conjunta[916670030940373†L246-L251]. Las bases vectoriales más utilizadas incluyen FAISS, Chroma, Weaviate y Pinecone. Chroma es útil para implementaciones locales porque almacena los metadatos en SQLite y los índices en su propia estructura[246311036369843†L218-L225]. Para la generación, el estado del arte incluye modelos como LLaMA 2/3, Mistral, Mixtral, Vicuna, Gemini, Gemma y GPT-4, que se pueden ejecutar localmente (mediante Ollama o similares) o a través de APIs comerciales[589007710703999†L1016-L1020].

4.8. Propuesta de implementación local

Con base en la revisión anterior y las necesidades del proyecto, se propone construir un sistema RAG local que elimine la multi-tenencia y las dependencias de servicios en la nube. La arquitectura estaría compuesta por los siguientes módulos:

1. **Ingestión y procesamiento:** se cargan documentos desde el sistema de archivos, se extrae texto (*PDF*, *DOCX*, *HTML*, *TXT*), se dividen en fragmentos de tamaño configurable y se generan embeddings con un modelo local (p.ej., *all-MiniLM* o *nomic-embed-text*). Los vectores y metadatos se almacenan en una base vectorial local (FAISS o Chroma) y en una base relacional ligera como SQLite.
2. **Búsqueda y re-ranqueo:** para cada consulta se genera un embedding, se recuperan los fragmentos más relevantes mediante búsqueda densa e híbrida, se combinan los resultados y se reordenan con un modelo *cross-encoder*. También se pueden aplicar estrategias de diversidad para evitar redundancia.

3. **Generación:** los fragmentos seleccionados se incorporan en el prompt de un modelo de lenguaje local (*Llama 3* o *Mistral*) que produce la respuesta en español. Se incluyen citas que apuntan a las fuentes consultadas.
4. **Interfaz de usuario:** se propone una interfaz web local o una aplicación de línea de comandos que permita subir documentos, iniciar consultas y visualizar las respuestas junto con sus fuentes. La autenticación se simplifica a un único usuario local y no se expone la API fuera del equipo.

Esta solución aprovecha los beneficios de RAG (precisión, trazabilidad y flexibilidad) mientras preserva la privacidad de los datos y reduce los costes operativos. La modularidad de los componentes permite sustituir el modelo de embeddings, la base vectorial o el LLM según las necesidades futuras.

4.9. Conclusiones

La técnica de *Retrieval-Augmented Generation* ha evolucionado rápidamente en los últimos años y se ha convertido en un pilar para construir sistemas conversacionales precisos y verificables. La revisión realizada muestra que los modelos generativos por sí solos sufren de alucinaciones y obsolescencia, mientras que RAG los complementa con recuperación de información actual y contextualizada[916670030940373†L145-L156]. Existen numerosas variantes y optimizaciones que permiten adaptar la técnica a diferentes dominios, desde implementaciones locales con *software* libre hasta plataformas de *RAG-as-a-Service*. Los marcos de código abierto como Haystack, LangChain, LlamaIndex, RAGFlow, txtAI, Cognita y Dify facilitan el desarrollo y despliegue de estas soluciones, cada uno con ventajas y limitaciones específicas[149667013130509†L518-L538]. Finalmente, se propone una arquitectura local que integra estos conceptos y responde a las necesidades de este proyecto, proporcionando una base sólida para la consulta visual interactiva de grandes volúmenes de datos multidimensionales.

Capítulo 5

Pruebas y Resultados

Capítulo 6

Conclusiones y Trabajos Futuros

Las conclusiones de la tesis son una parte muy importante y tiene las siguientes partes.

Escribir las conclusiones de su trabajo por cada uno de los objetivos específicos definidos en el Capítulo 1 y luego escribir la conclusión general de tu trabajo.

6.1. Problemas encontrados

La segunda parte de este capítulo corresponde a los problemas encontrados, esta sección es muy importante para que los siguientes estudiantes o investigadores que hagan algo en esta línea no cometan los mismos errores y tu tesis sea un buen peldaño para avanzar más rápido.

6.2. Recomendaciones

En esta sección el tesista debe reflejar que la tesis ha permitido adquirir nuevos conocimientos que podrían servir para guiar otros trabajos en el futuro.

6.3. Trabajos futuros

En base a los puntos anteriores es recomendable que tu tesis también sugiera trabajos futuros. Esta sección es especialmente útil para otras ideas de tesis.

Todo este capítulo no debe ser más de unas 4 páginas.

Bibliografía

- [Choset and Nagatani, 2001] Choset, H. and Nagatani, K. (2001). Topological simultaneous localization and mapping (slam): toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17(2):125–137.
- [Galante, 2001] Galante, R. (2001). *Evolución de Esquemas en Bancos de Datos Orientados a Objetos con el empleo de versiones*. PhD thesis, Instituto de Informática-UFRGS.
- [Mateos et al., 2000] Mateos, G., García, M., and Ortín, M. (2000). Inclusión de vistas en odmg. *Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2000)*, pages 383–395.