UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

FILIPE WALL MUTZ

Um Sistema para Mapeamento de Grandes Regiões usando GraphSLAM

VITÓRIA

2014

FILIPE WALL MUTZ

Um Sistema para Mapeamento de Grandes Regiões usando GraphSLAM

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

VITÓRIA

2014

FILIPE WALL MUTZ

Um Sistema para Mapeamento de Grandes Regiões usando GraphSLAM

COMISSÃO EXAMINADORA

Prof. Dr. Alberto Ferreira de Souza
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Thiago Oliveira-Santos
Universidade Federal do Espírito Santo
Coorientador

Profa. Dra. Claudine Badue
Universidade Federal do Espírito Santo

Prof. Dr. Fernando A. Auat Cheein
Universidad Técnica Federico Santa María

Vitória, 30 de julho de 2014.

AGRADECIMENTOS

Várias pessoas foram muito importantes para a realização deste trabalho. Eu vou me esforçar para agradecer a todos individualmente, mas se eu me esquecer de alguém, por favor, não se sinta triste.

Agradeço em primeiro lugar a Deus por permitir que eu aprendesse a não precisar de muitas coisas para ser feliz e por ter projetado uma vida tão movimentada e cheia de surpresas para mim. Para minha alegria, a minha vida tem sido uma grande aventura até aqui, cheia de momentos bons e momentos ruins, mas principalmente cheia de momentos.

Agradeço à minha família por serem uma constate nesse mundo tão variável. Sou grato por todo o suporte e todo o apoio sem o qual seria impossível alcançar qualquer objetivo.

Agradeço ao meu orientador e amigo Alberto Ferreira De Souza por me dar ótimos conselhos tanto sobre o trabalho quanto sobre a vida. Agradeço de forma muito especial também ao meu co-orientador Thiago Oliveira-Santos por estar sempre presente e com uma disposição infindável para trabalhar. A sua expertise em registro de nuvens de pontos foi fundamental para que este trabalho fosse um sucesso.

Agradeço também ao meu grande amigo Lucas de Paula Veronese por me ajudar em tudo. O seu apoio e incentivo foram muito importantes para mim. Agradeço aos meus amigos Lauro José Lyrio Júnior, Mariella Berger, Avelino Forechi, Edilson de Aguiar, Fernando A. Auat Cheein, Elias Silva de Oliveira e Tiago Alves de Oliveira por alegrarem os meus dias e por serem mananciais de boas ideias e bons conselhos. Vocês são a minha família também.

Por fim, agradeço ao Dr. Rainer Kümmerle e ao Dr. Giorgio Grisetti, desenvolvedores do framework G2O, por estarem sempre de prontidão para responder perguntas e debater ideias.

RESUMO

Um dos grandes desafios de robótica nos últimos anos tem sido o desenvolvimento de veículos robóticos com alto nível de autonomia, capazes de navegar por longos períodos de tempo sem a intervenção de um usuário humano. Existe uma grande expectativa de que, no futuro, os automóveis autônomos possam ser usados para prover acessibilidade para deficientes, reduzir o tempo e o custo para transportar cargas, e oferecer conforto para pessoas que não podem ou simplesmente não querem dirigir. Além disso, o uso de veículos autônomos tem potencial para aumentar significativamente a segurança no trânsito, uma vez que os robôs possuem uma capacidade ampliada de observar o mundo e não estão sujeitos a estados indesejados, como cansaço, nervosismo, embriaguez, pressa ou estresse.

Neste trabalho, foi estudado o problema de criação de mapas de grandes regiões para a localização e navegação de automóveis autônomos. Foram estudadas várias abordagens para solução do problema e foi desenvolvido um sistema de mapeamento de grandes regiões (*Large-scale Environment Mapping System* – LEMS) usando o algoritmo GraphSLAM baseado em poses. Os experimentos realizados mostraram que o LEMS foi capaz de mapear com uma boa qualidade diferentes regiões com tamanhos e características específicos.

Durante o desenvolvimento do LEMS, foi observado que os erros de odometria eram não-gaussianos e sujeitos a *bias*. Por isto, foi criada uma ferramenta para encontrar automaticamente o *bias* usando otimização por enxame de partículas. Os experimentos realizados mostraram que ao integrar o *bias* à odometria, o *dead-reckoning* foi capaz de aproximar com uma boa precisão o trajeto medido pelo GPS.

ABSTRACT

One of the big challenges of robotics in the last years has been the development of robotic vehicles with high level of autonomy able of navigating for long periods of time without human intervention. There is an high expectation that, in future, autonomous vehicles can be used to provide accessibility to injured people, reduce time and cost of transportation systems, and offer comfort to people that do not want (or cannot) to drive. Besides of that, the use of autonomous vehicles has the potential to highly increase the safety in traffic due to the increased capacity of robots to observe the world, and due to the fact that robots are not subject to dangerous states, such as tiredness, anger, drunkenness, hurry or stress.

The focus of this work was the problem of map building in big environments in order to make autonomous vehicles able to localize themselves and navigate. It was studied the well succeeded solutions found in literature and it was developed a Large-scale Environment Mapping System (called LEMS) using pose-based GraphSLAM algorithm. Experiments revealed that the LEMS was able to build high-quality maps of different regions with specific sizes and features.

During the development of LEMS, it was observed that the odometer data errors are non-Gaussians and subject to biases. Because of that, it was created a tool to automatically find the biases using particle swarm optimization. Experiments revealed that integrating the biases to odometer data, the dead-reckoning was able to reproduce with a good precision the path measured by GPS.

LISTA DE FIGURAS

Sumário

1 INTRODUÇÃO

Um dos grandes desafios de robótica nos últimos anos tem sido o desenvolvimento de robôs com alto nível de autonomia, capazes de realizar atividades por longos períodos de tempo sem a intervenção de um usuário humano. Entre as principais aplicações desta tecnologia podem ser destacados: os robôs autônomos para exploração espacial [1,2], robôs domésticos [3,4,5,6] e, em especial, os automóveis autônomos [7,8,9,10]. Existe uma grande expectativa de que, no futuro, os automóveis autônomos possam ser usados para prover acessibilidade para deficientes, reduzir o tempo e o custo para transportar cargas, e oferecer conforto para pessoas que não podem ou simplesmente não querem dirigir. Além disso, o uso de veículos autônomos tem potencial para aumentar a segurança no trânsito visto que robôs possuem uma capacidade ampliada de observar o mundo e não estão sujeitos a estados indesejados, como cansaço, nervosismo, embriaguez, pressa ou estresse. Buehler et. al. afirma que sensoriamento e controle automatizado constituem importantes tecnologias que, se aplicadas aos futuros veículos, poderão salvar milhares de vidas que seriam perdidas em acidentes de trânsito [8].

Visando acelerar o desenvolvimento da tecnologia de veículos autônomos, a Agência de Projetos de Pesquisa Avançados do Departamento de Defesa Americano (*Defense Advanced Research Projects Agency* – DARPA) organizou três competições em que universidades e empresas foram desafiadas a criar soluções para os problemas que inviabilizavam o uso prático de veículos autônomos [11]:

1. DARPA Grand Challenge 2004: nesta competição, os carros tinham que percorrer 240 km autonomamente. O evento teve 106 inscritos, mas apenas 15 times competiram no evento final. O melhor veículo conseguiu completar cinco por cento do percurso antes de falhar [11].
2. DARPA Grand Challenge 2005: nesta competição, os carros tinham que percorrer 212 km de estrada de chão autonomamente. O evento teve 197 inscrições e cada um dos 23 times que chegaram à final passaram por uma

série de desafios complexos, que comprovaram que seus carros eram melhores que os carros finalistas da corrida de 2004. Cinco times conseguiram completar o percurso [11].

3. DARPA Urban Challenge 2007: esta competição envolvia um percurso de 96 km em área urbana, que tinha que ser completado em menos de 6 horas. As regras incluíam obedecer todos os sinais de trânsito e negociar com outros carros para entrar em rotatórias [8]. Seis equipes completaram o percurso.

Depois das competições organizadas pela DARPA, progressos na tecnologia de veículos autônomos continuaram acontecendo por outras organizações, como:

- Google *driveless car*: veículo autônomo baseado no Toyota Prius desenvolvido pela empresa Google. Foi o primeiro carro autônomo licenciado em Nevada [12]. Google anunciou que o veículo já percorreu 500000 km autonomamente sem acidentes [13].
- *MadeInGermany*: veículo autônomo baseado no Volkswagen Passat desenvolvido pelo laboratório AutoNOMOS [14]. Foi certificado para dirigir autonomamente no estado de Berlim [15].
- Recentemente, montadoras de veículos também mostraram resultados em suas pesquisas de veículos autônomos. A Mercedes-Benz apresentou uma versão autônoma do sedã S 500 e demonstrou a tecnologia em um percurso urbano [16]. A Volvo apresentou um carro capaz de estacionar e dirigir autonomamente. O veículo está equipado com um sistema para detectar animais, pedestres, faixas de rodovias e barreiras, assim como um sistema para comunicação entre carros [17]. A Nissan também está pesquisando a tecnologia de veículos autônomos e pretende introduzi-la em 2020 [18].

Entre os principais desafios no desenvolvimento de veículos autônomos estão a criação de uma representação do mundo interna ao robô (um mapa) e o posterior uso desta representação interna para inferir a localização do veículo no mundo e permitir a sua navegação. Várias abordagens foram propostas para a solução destes problemas [19,20,21,22,23,24], sendo que a mais bem sucedida foi a divisão do problema em duas etapas: a construção do mapa (usualmente *offline*) e,

posteriormente, a localização e navegação de forma *online* [9]. A ênfase deste trabalho é a fase de mapeamento, em especial, o desenvolvimento de um algoritmo de localização e mapeamento simultâneos (*Simultaneous Localization and Mapping* – SLAM) [19] capaz de mapear de forma autônoma grandes regiões.

1.1 *Simultaneous Localization and Mapping* (SLAM)

O problema de SLAM pode ser definido como a capacidade de um robô criar uma representação interna de um ambiente previamente desconhecido e, simultaneamente, se localizar neste ambiente mesmo que ele seja inicializado em uma posição desconhecida *a priori*. Para realizar essa tarefa, os robôs são equipados com sensores que podem ser de dois tipos: sensores que medem variações no estado do robô (odômetro, acelerômetros, GPS, etc.) e sensores que medem mudanças no ambiente (câmeras, lasers, radares, sonares, etc.). Na abordagem probabilística de SLAM, assume-se que todas as medidas dos sensores são sujeitas a erros e busca-se encontrar o conjunto de poses e o mapa com máxima *likelihood*, dado o conjunto de medidas realizadas pelos sensores e os seus respectivos erros.

Os algoritmos de SLAM probabilísticos podem ser divididos em duas categorias: os algoritmos de SLAM completo (em inglês*, Full-SLAM*) e os algoritmos de SLAM *online* [25]. No *Full-SLAM*, todos os dados capturados pelos sensores são usados para calcular o mapa e as poses com maior probabilidade. Por lidar com grandes massas de dados, os algoritmos de *Full-SLAM* são, por natureza, *offline* e não são apropriados para aplicações executadas em tempo real. No SLAM *online*, por outro lado, a hipótese de *Markov* é usada para simplificar o problema de SLAM de forma que ele seja solucionável inclusive por aplicações que funcionam em tempo real. Um processo estocástico segue a hipótese de *Markov* (e neste caso é chamado de processo *Markoviano*) se a distribuição de probabilidade dos estados futuros condicionada aos valores passados (e presentes) depende apenas do estado atual e não do histórico de estados passados [26]. No contexto de SLAM, o uso da hipótese

de *Markov* permite que os dados capturados em instantes passados sejam descartados, uma vez que, pela hipótese, eles não influenciam a correção do erro corrente do robô. Embora o problema de SLAM não seja de fato um processo *Markoviano*, experimentos mostraram que a adoção da hipótese não causa perdas significativas de qualidade em aplicações de pequena escala [25].

Embora os algoritmos de SLAM *online* tenham desempenhado um papel importante e sido usados em várias aplicações, eles possuem um importante inconveniente: os algoritmos de SLAM *online* não possuem uma forma natural de resolver o problema de fechamento de loop, uma vez que, pela hipótese de *Markov*, os dados não podem ser revisitados [22]. Este problema se manifesta como um desalinhamento entre os mapas criados pelo robô ao visitar a mesma região duas ou mais vezes. Este desalinhamento é causado por erros acumulados ao longo do tempo e, usualmente, é resolvido com a adição de relações probabilísticas entre as poses do robô nas várias visitas à mesma região. Essas relações vão de encontro à hipótese de *Markov* e, por essa razão, quando o problema de fechamento de loop precisa ser tratado explicitamente, geralmente são usados algoritmos de *Full-SLAM*. Mapear grandes regiões envolve invariavelmente tratar o problema de fechamento de loop e, por essa razão, neste trabalho foi utilizado um algoritmo de *Full-SLAM*.

O GraphSLAM é um algoritmo de *Full-SLAM* em que as variáveis do modelo probabilístico são representadas por vértices de um grafo e os dados capturados pelos sensores são representados por arestas que relacionam os vértices do grafo. Esse grafo pode ser mapeado em uma função de densidade de probabilidade dependente dos vértices e estes podem ser calculados usando estimadores de máxima *likelihood* (em inglês, *Maximum-Likelihood Estimation* – MLE) [22].

As primeiras abordagens de GraphSLAM usavam um estado composto contendo as poses do robô e as variáveis do mapa. Nestas abordagens, o resultado do processo de estimação era o conjunto de poses e o mapa com maior *likelihood*, dadas as medidas realizadas pelos sensores. Entre os tipos de mapa mais utilizados estão os mapas de *landmarks* e os mapas de *grid*. Os mapas de *landmarks* são compostos por vários pontos característicos encontrados no ambiente chamados *landmarks*. Já

nos mapas de *grid*, o ambiente é particionado em várias áreas de mesma dimensão (usualmente chamadas células do *grid*) que armazenam características locais capturadas pelos sensores. Os mapas de *grid* mais usados são os *grids* de ocupação e os mapas de imagem do solo. Nos *grids* de ocupação, cada posição *x-y* do *grid* armazena a probabilidade daquela posição ser um obstáculo. Já nos mapas de imagens do solo, cada coordenada *x-y* do *grid* possui uma cor que representa alguma característica do solo, seja a cor, a refletividade infravermelha, a altura ou qualquer outra característica.

Nas abordagens de GraphSLAM baseadas em poses e mapa, diferentes tipos de vértices são usados para modelar as variáveis que representam as poses e as variáveis que representam o mapa. Quando são usados mapas de *landmarks*, os vértices do mapa usualmente representam as posições das *landmarks* no mundo. Já quando são usados mapas de *grid*, os vértices usualmente representam os valores de cada posição *x-y* do mapa, seja na forma de probabilidade, seja na forma da coloração dos pixels. O leitor pode observar sem dificuldade que, independente do tipo de mapa utilizado, ao navegar por percursos longos, o número de vértices do mapa irá crescer rapidamente e que este número será várias ordens de grandeza maior que o número de vértices que representam as poses do robô. Este crescimento ilimitado das variáveis relacionadas ao mapa torna esta abordagem de GraphSLAM ineficiente e não aplicável para o mapeamento de grandes regiões.

Visando solucionar este problema, Levinson et. al. propôs que o mapa fosse marginalizado do processo de estimação e que as dependências entre as poses e o mapa fossem transformadas em dependências apenas entre poses [9]. Ao marginalizar o mapa, o número de variáveis do modelo foi reduzido significativamente, de forma que o problema se tornou solucionável inclusive para o mapeamento de grandes regiões. É importante dizer, entretanto, que a função de distribuição de probabilidade obtida após a marginalização é apenas uma aproximação da função original. Contudo, nos experimentos realizados pelos autores o erro causado pela aproximação não causou perda de qualidade significativa. Após a estimativa das poses, os autores realizaram uma segunda etapa

de processamento para obtenção do mapa. Como as poses do robô eram conhecidas com uma boa precisão, a construção do mapa se tornou trivial e consistiu, basicamente, da projeção dos dados dos sensores do sistema de referência do robô para o sistema de referência do mundo.

1.2 Integração de *bias*

As abordagens probabilísticas para a solução do problema de SLAM usualmente assumem que os erros existentes nas medidas dos sensores são gaussianos e poucos foram os trabalhos que estudaram como incorporar dados que possuem erros não-gaussianos e/ou com média diferente de zero (medidas com *bias*) [27,28,29,30,31]. Entre as diversas razões para a existências de erros não-gaussianos e *bias* em SLAM, podem ser citadas: descontinuidades e não-linearidades nos modelos de medida e no modelo de movimento, o uso de modelagens incorretas, sensores com *bias*, erros de calibração, entre outros [31]. Em aplicações de pequena escala, várias soluções paliativas podem ser usadas para contornar o problema, como a adição de processos de estabilização, filtragem e linearização dos dados [32,21]. Entretanto, o efeito acumulativo das medidas com *bias* pode ocasionar erros significativos em aplicações *outdoor* e de grande escala. Esses efeitos podem se manifestar como imprecisões e incoerências no mapa e como divergências no processo de inferência [28].

Dos poucos trabalhos que apresentaram soluções para os problemas de erros não-gaussianos e/ou com média diferente de zero, a abordagem mais utilizada foi a adição de variáveis ao modelo probabilístico para representar *bias* nas medidas dos sensores e *offsets* nos parâmetros do robô [27,28,29,30]. Os *bias* nas medidas dos sensores são erros sistemáticos capazes de desviar a média do erro de zero. Exemplos de *bias* são medidas feitas pelos sensores sistematicamente menores (ou maiores) que os valores verdadeiros (como odômetros que medem velocidades maiores que as reais e lasers que medem distâncias menores que as reais). Os *offsets* nos parâmetros, por outro lado, são erros nas medidas que, devido aos

efeitos acumulativos, podem causar incoerências no modelo probabilístico com o passar do tempo. Os *offsets* mais comuns são medidas incorretas das posições dos sensores em relação ao robô (como a posição de um laser em relação ao eixo traseiro de um automóvel robótico ou a posição de uma câmera em relação ao centro de um robô direcional).

Nos trabalhos citados, os autores assumiram que, para cada pose do robô, existia um conjunto associado de variáveis de *bias* e *offsets*. Com isso, durante o processo de estimação, eram calculados os valores com máxima *likelihood*, tanto das poses quanto das demais variáveis, dadas as medidas realizadas pelos sensores. Neste trabalho, foi desenvolvida uma nova abordagem para o cálculo de erros sistemáticos usando algoritmos evolutivos. Em especial, foi estudado o problema de cálculo do *bias* existente no sensor de odometria do robô. Diferente dos trabalhos citados, nos quais os autores assumiram que o *bias* existente no modelo pode variar a qualquer instante de tempo, neste trabalho foi feita a hipótese de que estes erros são características dos sensores e que, por essa razão, tendem a ser constantes ao longo de grandes intervalos de tempo. Ao fazer esta simplificação, o cálculo do *bias* pôde ser realizado em uma etapa *offline* de pré-processamento.

Ao integrar o *bias* aos dados vindos do sensor de odometria, pôde ser observado que o *dead-reckoning* reproduziu com uma boa precisão o percurso medido pelo sistema de posicionamento global (*Global Positioning System* – GPS). Para padronizar a nomenclatura, neste trabalho, o termo odometria será utilizado para descrever os dados gerados pelo odômetro (velocidade e ângulo do volante) e o termo *dead-reckoning* será usado para descrever a predição do movimento realizado pelo robô através da aplicação dos dados de odometria ao modelo de movimento.

Para avaliar os algoritmos propostos neste trabalho, foi utilizada a plataforma robótica *Intelligent Autonomous Robot Automobile* (IARA), apresentada na Figura 1. IARA foi construída usando um automóvel de passeio Ford Escape Hybrid adaptado com sensores, mecanismos de controle e um conjunto de computadores instalados em seu porta-malas. Foram realizados experimentos com bases de dados de diferentes tamanhos e características. Os resultados demonstraram que tanto o

algoritmo de cálculo do *bias* da odometria quanto o algoritmo de GraphSLAM alcançaram resultados satisfatórios.

Assim como os experimentos de calibração do *bias*, os testes realizados com o sistema de mapeamento usando GraphSLAM também se mostraram satisfatórios. O sistema foi capaz de construir mapas de grandes regiões com uma boa qualidade, inclusive na presença de fechamentos de loop após longos trajetos.

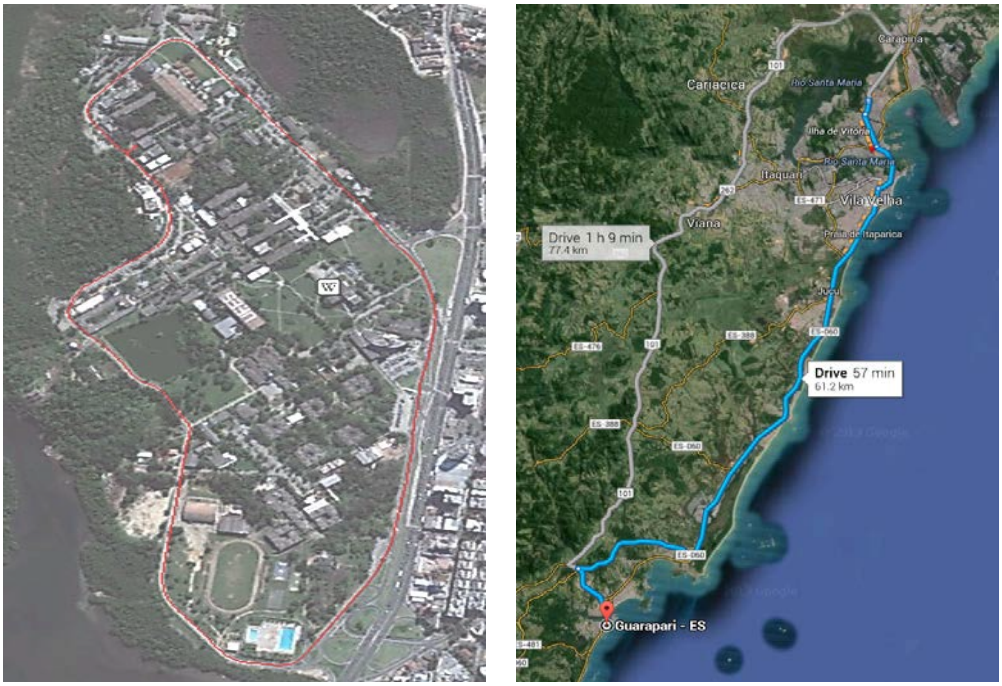
1.3 Motivação

Este trabalho está inserido em uma linha de pesquisa do Laboratório de Computação de Alto Desempenho (LCAD) do Departamento de Informática (DI) da Universidade Federal do Espírito Santo (UFES). Um dos objetivos do LCAD é a navegação completamente autônoma da IARA em dois percursos: a Volta da UFES e a Ida a Guarapari.

Figura 1 – IARA - *Intelligent Autonomous Robotic Automobile*.

Na Volta da UFES, o objetivo é desenvolver pesquisas que confiram à IARA as habilidades necessárias para realizar uma volta completa ao redor do campus de Goiabeiras da UFES. Este campus é o principal da UFES e possui um anel viário com 3.570 metros (Figura 2 (a)). Já na Ida a Guarapari, o objetivo é desenvolver pesquisas que viabilizem o aperfeiçoamento da IARA de modo a torná-la capaz de realizar uma viagem partindo da UFES com destino à cidade de Guarapari autonomamente. O percurso total da UFES à cidade de Guarapari possui 58,5 Km (Figura 2 (b)) e, neste caminho, todas as leis de trânsito terão que ser consideradas pelos algoritmos que comandarão a IARA.

A principal motivação deste trabalho é contribuir para o alcance dos objetivos citados por meio do estudo e desenvolvimento de um subsistema de mapeamento capaz de criar mapas de grandes regiões com qualidade suficiente que permita a IARA se localizar em todo o percurso, tanto na Volta da UFES quanto na Ida a Guarapari.



(a) (b)

Figura 2 - (a) Caminho da Volta da UFES. (b) Caminho da Ida a Guarapari.

1.4 Objetivo

O objetivo deste trabalho é desenvolver um sistema de mapeamento para o veículo autônomo IARA capaz de mapear grandes regiões e resolver o problema de fechamento de loop. O sistema de mapeamento deve funcionar em diferentes ambientes (dentro da universidade, em vias urbanas, estradas, etc.) e ser escalável, de forma a manter a qualidade mesmo que o robô navegue por grandes percursos.

Além disso, um objetivo secundário deste trabalho é estudar novas formas de calcular o *bias* e integrá-lo aos sensores de odometria de forma a conseguir um *dead-reckoning* com precisão comparável ao GPS.

1.5 Contribuições

A principal contribuição do trabalho foi o desenvolvimento de um sistema de mapeamento para o veículo autônomo IARA usando o algoritmo GraphSLAM baseado em poses. Experimentos mostraram que o algoritmo desenvolvido foi capaz de mapear grandes áreas e manter a qualidade do mapa, mesmo quando regiões foram visitadas mais de uma vez (problema de fechamento de loop).

Outra contribuição importante deste trabalho foi o desenvolvimento de uma nova metodologia para cálculo do *bias* da odometria usando algoritmos evolutivos. Esta nova metodologia é uma simplificação dos métodos convencionais e se mostrou eficiente e eficaz, alcançando um *dead-reckoning* com precisão próxima ao GPS.

1.6 Organização do Texto

Esta dissertação está dividida da seguinte forma:

Capítulo 2: Trabalhos Relacionados. Neste capítulo é realizada uma revisão dos trabalhos relacionados à este trabalho. São apresentados os principais métodos

para construção de mapas encontrados na literatura e são discutidas as vantagens e desvantagens de cada método.

Capítulo 3: GraphSLAM baseado em Poses. Neste capítulo é realizada uma revisão detalhada de como o problema de SLAM pode ser modelado probabilisticamente usando o GraphSLAM baseado em poses.

Capítulo 4: Sistema de Mapeamento de Grandes Regiões. Neste capítulo é apresentado o desenvolvimento principal deste trabalho: o sistema de mapeamento de grandes regiões.

Capítulo 5: Metodologia e Materiais. Neste capítulo são descritos os materiais utilizados neste trabalho e são apresentados detalhes de implementação do sistema de mapeamento de grandes regiões.

Capítulo 6: Experimentos e Resultados. Neste capítulo são apresentados a metodologia experimental e os resultados obtidos neste trabalho.

Capítulo 7: Discussão. Neste capítulo, é realizada uma discussão e análise crítica dos resultados alcançados neste trabalho e de como estes resultados se comparam aos demais trabalhos existentes na literatura.

Capítulo 8: Conclusão. Neste capítulo, são apresentados um sumário do trabalho, as conclusões obtidas e possíveis direções para trabalhos futuros.

2 TRABALHOS RELACIONADOS

O problema de Localização e Mapeamento Simultâneos – *Simultaneous Localization and Mapping* (SLAM) – tem como objetivo permitir que um robô móvel posicionado em um ambiente previamente desconhecido, passe a construir um mapa coerente do ambiente e, simultaneamente, se localize neste mapa. SLAM tem sido aplicado em diferentes domínios de robótica, em robôs terrestres tanto em ambientes internos quanto em ambientes externos, em veículos autônomos subaquáticos (*Autonomous Underwater Vehicles* – AUV) e em veículos aéreos não tripulados (*Unmanned Aerial Vehicles* – UAV). Do ponto de vista teórico e conceitual, o problema de SLAM já pode ser considerado um problema resolvido. Do ponto de vista prático, entretanto, ainda existem várias questões em aberto que precisam ser estudadas, principalmente no que diz respeito à autonomia de longo prazo, segurança e usabilidade.

Em [19], o autor afirma que a discussão do problema de SLAM se iniciou no ano de 1986, na Conferência de Robótica e Automação da IEEE (*IEEE Robotics and Automation Conference*), em São Francisco, Califórnia. A década de 80 foi marcada pelo início da utilização de métodos probabilísticos em aplicações de robótica e inteligência artificial. Durante a conferência, vários artigos foram publicados tendo como tema localização e mapeamento em plataformas robóticas. Nas apresentações, várias discussões relevantes aconteceram e chegou-se à conclusão de que mapeamento probabilístico era um problema fundamental em robótica. Estas discussões aqueceram a área, e nos anos seguintes, várias publicações foram feitas abordando o tema.

Os primeiros avanços significativos alcançados na área de SLAM são encontrados em [33] e [34]. Nestes trabalhos, os autores demostraram a existência de um alto grau de correlação entre as várias *landmarks* (pontos característicos encontrados no ambiente) visualizadas por um robô durante a realização de um percurso. Eles mostraram ainda que esta correlação cresce à medida que sucessivas observações das *landmarks* são feitas. Em [35], os autores mostraram que se um robô se move

por um ambiente enquanto faz observações de *landmarks*, as posições dessas *landmarks* são correlacionadas umas com as outras pelo erro na estimativa da posição do robô. Essa descoberta mostrou que, para encontrar uma solução conjunta para os problemas de localização e mapeamento, os algoritmos probabilísticos deveriam manter um estado composto de grande dimensão, contendo as poses do robô e todas as *landmarks* existentes no mapa. Em [33], o autor mostrou que, uma vez que os problemas de localização e mapeamento fossem tratados como um problema de estimação único, o resultado seria convergente e que, com isso, seria possível encontrar o conjunto de poses e mapa com erros mínimos.

Este capítulo traz uma revisão das propostas existentes na literatura para solução do problema de SLAM e, para cada uma das propostas, é realizada uma análise contendo as vantagens e desvantagens de cada método. As seções 2.1, 2.2 e 2.3 sintetizam as abordagens mais famosas: o *Extended Kalman Filter* SLAM (EKF-SLAM), o FastSLAM e o GraphSLAM. A seção 2.4 traz uma revisão sobre mapas de *grid* e a seção 2.5 realiza uma análise comparativa das técnicas de mapeamento usando mapas de *landmarks* e mapas de *grid*. Por fim, a seção 2.6 apresenta uma discussão sobre outro tema abordado neste trabalho: como incorporar dados com erros não-gaussianos ao SLAM.

2.1 *Extended Kalman Filter* SLAM (EKF-SLAM)

Em [36], o autor apresentou a primeira solução real para o problema de SLAM. No trabalho, o autor formalizou conceitos como relações espaciais, variáveis espaciais, operadores de composição de relações espaciais e formas de representar probabilisticamente o erro destas relações. O autor apresentou também os conceitos de modelo de movimento, modelo de medida e o conceito de mapa probabilístico como um conjunto de objetos do mundo e as suas respectivas posições. Por fim, todos estes conceitos foram usados em um *Extended Kalman Filter* para inferir as poses do robô e dos objetos do mapa com o mínimo de erro possível.

A base para o algoritmo de SLAM usando *Extended Kalman Filter* (EKF-SLAM) e demais algoritmos de SLAM baseados em filtros de Kalman (em inglês, *Kalman Filters*), como o *Unscented Kalman Filter* SLAM (UKF-SLAM) [37], o *Iterated Sigma Point Kalman Filter* SLAM (ISPKF-SLAM) [38] e outros, é usar a probabilidade dos movimentos realizados pelo robô e a probabilidade das medidas realizadas pelos seus sensores para melhorar as estimativas da pose, do mapa e dos respectivos erros [19], [39] e [40]. Exemplos bem sucedidos de aplicações que usaram EKF-SLAM ou uma de suas variantes podem ser encontrados em [41], [42], [43] e [20].

O algoritmo EKF-SLAM foi utilizado por muito tempo como a solução padrão para o problema de SLAM. Entretanto, ele possui uma série de desvantagens que impedem a sua utilização em aplicações práticas, principalmente em ambientes *outdoor* e/ou de grande escala. A adição de novas poses e a visualização de novas *landmarks* envolvem a adição e atualização de campos na matriz de estado e na matriz de covariância do modelo probabilístico. É fácil notar que, como estas matrizes precisam ser mantidas em memória, o crescimento ilimitado do número de variáveis rapidamente torna essa computação impraticável. Além disso, o algoritmo de EKF-SLAM padrão é especialmente frágil à associação incorreta de *landmarks* (problema de *matching*), um problema que continua em aberto e é especialmente desafiador em ambientes com *landmarks* que mudam dependendo do ponto de vista [44]. Por fim, o algoritmo de EKF-SLAM realiza a linearização de modelos não-lineares de movimento e de observação. Com isto, a presença de descontinuidades ou não-linearidades significativas nos modelos inevitavelmente irá gerar inconsistências, muitas vezes dramáticas, e a divergência dos algoritmos de estimação [45]. Vários trabalhos foram desenvolvidos para tentar solucionar os problemas presentes no EKF-SLAM e tornar a sua aplicação mais prática, como [38], [46] e [23]. Contudo, a grande maioria dos trabalhos continua a propor o uso de EKF-SLAM apenas em ambientes de pequena escala, *indoor* ou com condições controladas.

2.2 FastSLAM

O algoritmo FastSLAM, introduzido por Montemerlo et. al. [47], marcou uma mudança conceitual importante nas novas abordagens de SLAM probabilístico. Em [19], o autor afirma que a criação do FastSLAM teve como principais influências os experimentos de SLAM probabilísticos realizados anteriormente por Murphy [48] e Thrun, com outros autores [49]. Até então, os trabalhos publicados buscaram melhorar o desempenho dos algoritmos de SLAM baseados em filtros de Kalman, mas mantendo a hipótese essencial de erros gaussianos e modelos de movimento e observação lineares. O FastSLAM, baseado nos métodos de amostragem de Monte Carlo (em inglês, *Monte Carlo Sampling*), também conhecidos como Filtros de Partículas, foi a primeira abordagem a representar explicitamente modelos de movimento não-lineares e poses com distribuições não-gaussianas.

Além dos novos modelos probabilísticos, o FastSLAM apresentou uma melhoria significativa de desempenho em relação aos algoritmos de SLAM baseados em *Kalman Filters*. A complexidade computacional destes algoritmos é justificada pelo fato de que a matriz de covariância mantida pelo *Kalman Filter* possui um número de elementos igual ao quadrado do número de *landmarks*, todos os quais precisam ser atualizados sempre que uma única *landmark* é atualizada [50,51,52]. A grande contribuição do FastSLAM foi a decomposição do problema de SLAM em um problema de localização e vários problemas de estimativa das *landmarks*, sendo que cada um destes problemas possui dependência com a estimativa das poses do robô [47]. O FastSLAM usa a técnica de Rao–Blackwellization para representar o problema de SLAM como um filtro de partículas. Neste filtro, cada partícula representa um possível caminho realizado pelo robô e as suas estimativas pessoais das posições das *landmarks*. Diferente do EKF-SLAM, entretanto, no FastSLAM cada *landmark* é representada por uma média e uma covariância e, pela hipótese de independência nas medidas das *landmarks*, não é armazenada nenhuma relação compartilhada entre as distribuições de probabilidade das *landmarks*. Com esta representação, sempre que uma *landmark* é visualizada, apenas a sua média e matriz de covariância existente em cada partícula precisa ser atualizada. Com isso, o

procedimento de atualização de medidas, que antes era quadrático no número de *landmarks*, passou a ser proporcional ao número de *landmarks* multiplicado pelo número de partículas [47]. Como o número de partículas é constante e ordens de grandeza menor do que o número de *landmarks*, o desempenho alcançado pelo FastSLAM foi sem precedentes.

O FastSLAM foi um trabalho marcante na área de robótica móvel. Entretanto, algumas críticas podem ser feitas a ele. Do ponto de vista teórico, a hipótese de independência das *landmarks* cria uma forte dependência entre as distribuições de probabilidade das *landmarks* e a distribuição de probabilidade do caminho percorrido pelo robô. Por essa razão, a exclusão de partículas pouco prováveis durante a fase de *resampling* pode levar à perda de importantes informações estatísticas que poderiam, posteriormente, ser usadas para corrigir as poses e o mapa [53]. Esta inabilidade de esquecer o passado pode causar degenerações e inconsistências tanto na estimativa do mapa quanto na estimativa do caminho percorrido pelo robô.

Do ponto de vista prático, a qualidade do FastSLAM, assim como de todos os algoritmos baseados em *landmarks*, está intimamente ligada ao método de associação de *landmarks*. Dessa forma, a ocorrência de sucessivos *matchings* incorretos pode causar impactos significativos nas estimativas do mapa e das poses do robô. Outro ponto importante que precisa ser citado é o fato de que o FastSLAM não possui ferramentas para lidar explicitamente com o problema de fechamento de loop. Dessa forma, ao utilizá-lo para mapear grandes regiões, erros acumulados podem causar inconsistências e degenerações em áreas visitadas mais de uma vez.

2.3 GraphSLAM

Thrun e Montemerlo afirmam em [22] que uma desvantagem chave dos algoritmos de SLAM *online* é o fato deles usarem os dados dos sensores e, então, descartá-los. Ao fazer isto, se torna impossível revisitar os dados no momento de construção do mapa para, por exemplo, tratar o problema de fechamento de loop. Técnicas *offline*,

introduzidas em [54] e vários artigos subsequentes [55,24,56,57], mostraram que é possível alcançar um incremento significativo de qualidade através da memorização dos dados e do adiamento da criação do mapa até que todos os dados tenham sido processados. Em [55], os autores mostraram que a probabilidade *a posteriori* do SLAM naturalmente forma um grafo esparso e que este grafo tem a forma de um somatório de restrições quadráticas não-lineares. Os autores mostraram ainda que estas restrições podem ser otimizadas de forma a obter o mapa e o conjunto de poses com máxima *likelihood*. Estas descobertas e os vários outros estudos desenvolvidos a partir de então culminaram na criação do GraphSLAM, o algoritmo estado-da-arte para solução do problema de SLAM.

Diferente das abordagens de SLAM *online* citadas até agora, o GraphSLAM, proposto por Thrun e Montemerlo em [22], é um algoritmo de *Full-SLAM* capaz de usar todos os dados dos sensores para estimar o caminho percorrido pelo robô. No GraphSLAM, a probabilidade *a posteriori* do SLAM é representada por um grafo, em que os nós representam as variáveis do modelo (poses e *landmarks*, no trabalho original) e as arestas representam as medidas dos sensores e os respectivos erros. Como em aplicações *outdoor* o número de *landmarks* tende a ser muito grande, os autores propuseram o uso de técnicas de eliminação de variáveis para marginalizar o mapa e reduzir o grafo ao conjunto de poses e as suas relações. Com isto, foi obtido um problema com dimensionalidade significativamente reduzida e técnicas convencionais de otimização, como o gradiente conjugado, puderam ser usadas para encontrar os valores dos vértices que maximizavam a *likelihood* do modelo.

As principais vantagens do GraphSLAM em relação às outras soluções apresentadas é a possibilidade de solução do problema de fechamento de loop, através da adição de ligações entre as poses do robô durante duas ou mais visitas à mesma região, e a possibilidade de localização global através da inserção de dados de GPS na forma de arestas unitárias no grafo [9].

2.4 Mapas de *Grid*

Embora a maior parte dos algoritmos de SLAM citados faça uso de mapas de *landmarks*, estes mapas não são os únicos existentes na literatura. Os Mapas de *Grid* (em inglês, *Grid Maps*) formam outra família de métodos que também foram vastamente utilizados para construção de mapas em algoritmos de SLAM. Nos mapas de *grid*, o espaço é particionado em pequenas regiões de mesma dimensão, onde são armazenadas características locais do mundo. Entre os mapas de grid os mais famosos estão os *grids* de ocupação (em inglês, *Occupancy Grids*) [58] e os mapas de remissão luminosa (em inglês, *Remission Maps*) [59].

Nos *grids* de ocupação, cada célula contém a probabilidade de existir um obstáculo naquela região. Para construir um *grid* de ocupação, o robô deve ser equipado com sensores capazes de medir distâncias, como lasers, câmeras estereoscópicas, etc. Assumindo que as poses do robô são conhecidas, o algoritmo para construção de um *grid* de ocupação é muito simples. Cada vez que o sensor realiza uma medida, as células que possuem uma distância até o robô igual àquela medida pelos sensores recebem um incremento na probabilidade de serem obstáculos. Já as células localizadas entre o robô e o provável obstáculo recebem um decremento na probabilidade de estarem ocupadas (ou seja, recebem um incremento na probabilidade de serem células livres) [25]. Este decremento é justificado pelo fato de que, se alguma delas fosse um obstáculo, a distância medida pelo sensor seria menor. Os *grids* de ocupação são mapas convenientes para a navegação e o planejamento de caminhos e foram muito utilizados em aplicações de robótica [58,60,61].

Os mapas de remissão, por outro lado, possuem em cada célula do *grid* uma cor que representa a remissão luminosa dos materiais existentes naquela região. A remissão luminosa é um valor que representa a quantidade de luz refletida ou dispersada por um determinado material. Para construir um mapa de remissão, o robô precisa ser equipado com emissores e receptores de luminosidade, como lasers ou outras fontes de infravermelho. O procedimento para criação de mapas de

remissão, dado que o caminho percorrido pelo robô é conhecido, consiste em preencher cada célula tocada pelo sensor com a respectiva remissão luminosa. Se uma célula for tocada mais de uma vez, a remissão luminosa média é armazenada. Entre os trabalhos que utilizaram mapas de remissão podem ser destacados [30] e [9].

Em [25], o autor propôs uma forma de utilizar o FastSLAM para construir mapas de *grid*. Neste trabalho, cada partícula armazenava uma versão do mapa e, ao final do SLAM, o mapa da partícula mais provável era retornado. Esta abordagem possui um alto custo computacional, visto que os mapas de todas as partículas precisam ser atualizados quando novos dados dos sensores são capturados e que os mapas precisam ser copiados durante a fase de *resampling*.

Em [9], o autor propõe adicionar os valores de cada célula do *grid* ao conjunto de vértices do GraphSLAM. Com isto, durante a fase de estimação, não só as poses, mas também as variáveis do mapa seriam calculadas. É fácil notar, entretanto, que esta abordagem é inviável, pois implicaria na adição de milhares de novas variáveis ao modelo probabilístico. No mesmo trabalho, o autor propõe uma nova versão do GraphSLAM, o GraphSLAM baseado em poses, no qual a construção do mapa seria realizada em uma etapa posterior à otimização das poses. Este novo método se mostrou eficiente e eficaz e, por essa razão, foi escolhido como tema central de estudo neste trabalho.

Embora existam técnicas para correção de desalinhamentos e propagação de erros no mapa em algoritmos de SLAM *online* [25], dependendo do tamanho do mapa, o uso destas técnicas pode envolver o consumo de intervalos significativos de tempo. Intervalos durante os quais o robô estaria ocupado realizando processamentos e não estaria apto a navegar ou a reagir a modificações no ambiente. Tendo isto em mente, a menos que o robô fosse a única entidade no ambiente, a fase de correção dos desalinhamentos e propagação dos erros deveria ser realizada com o robô imóvel e a não observação dessa restrição poderia colocar em risco as pessoas e demais seres posicionados no mesmo ambiente do robô.

Por ser naturalmente *offline*, o GraphSLAM baseado em poses é capaz de encontrar o conjunto de poses do robô com máxima *likelihood*, inclusive levando em consideração problemas de fechamento de loop, sem adicionar os riscos que existiriam em aplicações que precisam construir o mapa de forma *online*.

2.5 Mapas de *Landmarks* vs. Mapas de *Grid*

Embora vários trabalhos tenham sido desenvolvidos usando tanto mapas de *landmarks* quanto mapas de *grid*, ambas as abordagens possuem vantagens e desvantagens que precisam ser consideradas.

Quando se deseja mapear grandes regiões, tanto os mapas de *landmarks* quanto os mapas de *grid* apresentam restrições importantes. O uso de *landmarks* envolve a atualização das poses e do mapa sempre que uma nova observação é realizada. Com isto, quanto mais *landmarks* são visualizadas em um trajeto, maior será o custo computacional para atualização do modelo. O uso de mapas de *grid*, por outro lado, envolve a atualização apenas das células tocadas pelos sensores. Entretanto, o número de células existentes no mapa costuma ser significativamente maior que o número de *landmarks*, e, por essa razão, a sua utilização envolve um maior consumo de memória. Uma solução para ambos os problemas encontrada na literatura é a utilização de submapas [62,63,21]. Esta técnica propõe o mapeamento de pequenas regiões separadamente e, em determinado momento, a utilização de algoritmos de encaixe (frequentemente *offline*) para alinhar as regiões umas às outras.

Uma crítica realizada aos algoritmos de SLAM baseados em *landmarks* é o fato de que a sua qualidade é intimamente relacionada à qualidade do método de associação de dados. Se associações incorretas de *landmarks* são feitas com frequência, existe uma grande probabilidade de divergência do algoritmo de SLAM.

Além disto, algoritmos de SLAM baseados em *landmarks* possuem uma inabilidade nativa de realizar localização global, enquanto os algoritmos de SLAM que fazem

uso de mapas de *grid* possuem mecanismos naturais para a realização desta tarefa. A forma mais ingênua seria, por exemplo, supor que o robô pode estar localizado em qualquer uma das células do *grid* e, assim que novos dados fossem capturados, as localizações menos prováveis fossem eliminadas. Esse processo poderia ser repetido até que apenas uma possibilidade restasse [25]. Uma proposta alternativa pode ser usada quando o robô está equipado com um sistema de posicionamento global (*Global Positioning System* – GPS). Neste caso, o conjunto de partículas poderia ser inicializado nas células ao redor da posição medida pelo GPS e, como na abordagem anterior, as partículas menos prováveis poderiam ser continuamente excluídas até que apenas uma possibilidade restasse.

Por fim, vale citar que, dentre os tipos de mapas apresentados, apenas os *grids* de ocupação são adequados para planejamento de caminho e navegação. Os mapas de *landmarks* e os mapas de remissão luminosa são adequados apenas para a localização, visto que não são capazes de representar quais regiões contém obstáculos e quais regiões são livres.

2.6 Calibração de *Bias* dos Sensores

Grande parte dos algoritmos de SLAM probabilísticos assume a hipótese de que as medidas realizadas pelos sensores possuem um erro gaussiano com média zero. Embora, na prática, esta hipótese possa ser frequentemente adotada, o uso de sensores com erros altamente não-gaussianos e, em especial, com erros sistemáticos (comumente chamados de *bias*) pode causar a divergência do estimador probabilístico e levar a criação de mapas inconsistentes. Em [28], o autor propõe que, a cada instante de tempo seja adicionado um *bias* instantâneo ao conjunto de variáveis de um EKF. Semelhantemente, em [27], o autor adicionou um *bias* instantâneo aos vértices de um GraphSLAM. Em ambos os casos, os autores assumiram que o *bias* poderia variar (inclusive bruscamente) em qualquer instante de tempo. Neste trabalho, foi estudada uma abordagem alternativa que supõe a

existência de um *bias* característico do sensor que tende a mudar pouco ao longo do tempo.

3 GRAPHSLAM BASEADO EM POSES

O GraphSLAM baseado em poses foi um importante desenvolvimento em direção à criação de algoritmos de SLAM capazes de mapear grandes regiões. Este Capítulo está organizado da seguinte forma: a seção 3.1 descreve o GraphSLAM baseado em poses, a seção 3.2 apresenta o modelo probabilístico geral do GraphSLAM e a seção 3.3 descreve como integrar as *likelihoods* dos sensores usados neste trabalho ao modelo geral do GraphSLAM.

3.1 Introdução

O GraphSLAM baseado em poses (a partir daqui chamado apenas de GraphSLAM) trouxe uma importante inovação em relação às propostas anteriores: a divisão do problema de mapeamento em duas etapas, a otimização das poses e a construção do mapa. Na primeira fase, os dados dos sensores são usados para encontrar o conjunto de poses que representam com maior *likelihood* o caminho percorrido pelo robô. Na segunda fase, os dados são percorridos novamente e as poses otimizadas são usadas para construir o mapa.

No GraphSLAM, o problema de SLAM é representado como um grafo em que os vértices representam as variáveis probabilísticas que serão estimadas e as arestas representam observações realizadas pelos sensores e os respectivos erros. Neste trabalho, os vértices do grafo são as poses 2D do robô, compostas pela posição *x-y* e pela orientação θ, e as arestas do grafo são as observações realizadas por vários sensores. Os erros dos dados capturados pelos sensores são representados na forma de matrizes de covariância. As arestas podem possuir diferentes cardinalidades, de acordo com o número de vértices que elas restringem. Para exemplificar, uma aresta de odometria tem cardinalidade dois, pois ela relaciona duas poses, a de origem e a de destino. Da mesma forma, uma aresta de GPS tem cardinalidade um, pois ela restringe apenas uma pose do robô. O procedimento para construção do grafo é direto e consiste apenas em adicionar as poses do robô como

vértices e as observações dos sensores como arestas ligando os vértices apropriados. O grafo construído pode, então, ser mapeado em uma função objetivo e um algoritmo de otimização pode ser usado para encontrar os valores das poses que maximizam a *likelihood* do modelo [22,9].

3.2 Modelo Probabilístico

Nesta seção é apresentado o modelo probabilístico usado pelo GraphSLAM para descrever o problema de SLAM. Seja $X = \{X_0, X_1, \ \ldots, X_n\}$ o conjunto de poses do robô, onde *n* é o número total de poses e X_T é a pose do robô no instante *T*. Seja ainda $M = \{M_0^0, M_1^0, \ldots, M_k^0, \ldots, M_{k-1}^n, M_k^n\}$ o conjunto de observações realizadas por um ou mais sensores instalados no robô, onde M_p^T representa a observação feita pelo sensor *p* no instante de tempo *T*. No GraphSLAM, o conjunto *X* representa os vértices do grafo (as variáveis que deseja-se estimar) e o conjunto *M* representa o conjunto de arestas (as variáveis constantes ao longo do processo de estimação).

Dadas estas definições, o problema a ser resolvido pelo GraphSLAM pode ser modelado como um problema de estimativa de máxima *likelihood* (em inglês, *Maximum-Likelihood Estimation* – MLE). O MLE é um método probabilístico que tem como objetivo encontrar os valores dos parâmetros que maximizam a *likelihood* de um determinado modelo. No caso do GraphSLAM, o objetivo do MLE é encontrar os valores das poses que maximizam a *likelihood* das poses dadas as observações realizadas pelos sensores. Matematicamente, este problema pode ser definido como:

$$X^* = argmax_X \, L(X \mid M), \tag{1}$$

onde *L(.)* é uma função de *likelihood* e *X** são as poses que formam o conjunto solução do problema.

A título de informação vale lembrar que usa-se o termo *likelihood* para designar a função probabilística dos parâmetros dado o conjunto observações, enquanto usa-se o termo probabilidade para designar a função probabilística das observações dados os parâmetros do modelo. A *likelihood* e a probabilidade se relacionam por:

$$L(X \mid M) = P(M \mid X). \tag{2}$$

De volta ao problema inicial, o modelo probabilístico do GraphSLAM pode ser expandido em relação ao conjunto de medidas de forma a se obter:

$$X^* = argmax_X \, L(X \mid M) = argmax_X \, L(X \mid M_0^0, \ldots, M_k^n). \tag{3}$$

Assumindo que as medidas realizadas pelos sensores são independentes, a *likelihood* se torna:

$$X^* = argmax_X \, L(X \mid M) = argmax_X \, L(X \mid M_0^0) \times \ldots \times L(X \mid M_k^n). \tag{4}$$

Como as observações dos sensores só geram dependência nas poses restringidas por elas, a equação pode ser escrita como:

$$X^* = argmax_X \, L(X \mid M) = argmax_X \, L\left(X_{M_0^0} \mid M_0^0\right) \times \ldots \times L\left(X_{M_k^n} \mid M_k^n\right), \tag{5}$$

onde $X_{M_p^T}$ é o conjunto de poses que possuem dependência com a medida M_p^T.

Embora o modelo acima esteja matematicamente correto, ele dificilmente poderia ser usado em uma implementação computacional. Como todas as *likelihoods* pertencem ao intervalo [0, 1] e, frequentemente, são menores que um, a multiplicação delas rapidamente tenderia a zero. Entretanto, a representação de números de ponto flutuante usada pelos computadores é limitada e, por essa razão, com poucas arestas o problema já se tornaria intratável. Por esta razão, usualmente

escolhe-se otimizar o negativo do log da *likelihood* ao invés da *likelihood* per si [1]. Aplicando o negativo do log ao modelo do GraphSLAM obtém-se:

$$X^* = argmax_X\, L(X \mid M) = argmin_X - \log\big(L\,(X \mid M)\big). \quad (6)$$

Expandindo a likelihood, a equação se torna:

$$X^* = argmin_X - \log\Big(L\big(X_{M_0^0}|M_0^0\big) \times \ldots \times L\big(X_{M_k^n}|M_k^n\big)\Big). \quad (7)$$

Por fim, aplicando a transformação do log da multiplicação na soma dos logs, o problema de estimação passa a ser definido como:

$$X^* = argmin_X - \Big[\log\Big(L\big(X_{M_0^0}|M_0^0\big)\Big) + \ldots + \log\Big(L\big(X_{M_k^n}|M_k^n\big)\Big)\Big]. \quad (8)$$

3.3 *Likelihood* das Observações

O modelo do GraphSLAM definido acima é genérico e pode ser usado para integrar as probabilidades de quaisquer arranjos de sensores. Entretanto, neste trabalho, serão abordados apenas dois tipos de sensores: os sensores de posicionamento global e os sensores de deslocamento.

Os sensores de posicionamento global são aqueles capazes de medir a pose do robô (ou parte dela) em um sistema de referência global. Por restringirem apenas uma pose, estes sensores são representados como uma aresta unária no grafo. Entre os sensores de posicionamento global mais usados estão o *Global Positioning System* (GPS) e o magnetômetro.

Os sensores de deslocamento são aqueles capazes de medir a relação espacial entre duas poses. No GraphSLAM, os sensores de deslocamento geram uma

[1] Como o logaritmo é uma função estritamente crescente, os valores que maximizam/minimizam uma determinada função $f(x)$ são os mesmos que otimizam a função $log(f(x))$. Ao inverter o sinal do *log*, entretanto, as variáveis que antes minimizam o *log* passam a maximizar o negativo do *log*. Da mesma forma, as variáveis que antes maximizavam o *log* passam a minimizar o negativo do *log*.

restrição entre duas poses, uma de origem e uma de destino. Por essa razão, os seus dados são representados como uma aresta ligando dois vértices do grafo. Exemplos de sensores de deslocamento são a odometria cinemática, a odometria visual e os algoritmos de *matching*. Algoritmos de *matching* são algoritmos que tem como objetivo calcular a transformada capaz de alinhar dados de diferentes sensores ou dados do mesmo sensor capturados em diferentes instantes de tempo. Entre os algoritmos de *matching* mais utilizados, podem ser destacados o *Generalized Iterative Closest Point* (GICP), usado para medir o deslocamento entre duas nuvens de pontos 3D [64], algoritmos de *Feature Matching* como SIFT e SURF, que usam pontos característicos para medir o deslocamento entre duas imagens [65], algoritmos de *Scan Matching*, usados para medir o deslocamento entre dois feixes de laser [66], e algoritmos *Map Matching*, usados para medir o deslocamento entre dois mapas de *grid* [67].

Definindo como M_G o subconjunto de M que contém as observações realizadas por sensores de posicionamento global e M_D como o subconjunto de M que contém as observações realizadas por sensores de deslocamento, a equação (8) pode ser reescrita como:

$$X^* = argmin_X - \sum_{Y \in M_G} \log\big(L(X_Y|Y)\big) - \sum_{\delta \in M_D} \log\Big(L\big(X_{from}, X_{to}\big|\delta\big)\Big), \quad (9)$$

onde X_Y é a pose robô no instante em que a medida Y foi realizada, e X_{from} e X_{to} são, respectivamente, as poses de origem e destino do deslocamento δ medido pelo sensor.

As medidas realizadas por sensores de posicionamento global com erros gaussianos podem ser modeladas por:

$$s = X_Y - Y, \quad (10)$$

onde X_Y é a pose do robô, Y é a observação realizada pelo sensor e s é um erro gaussiano com média zero e matriz covariância R. A *likelihood* desta observação é definida como:

$$L(X_Y|Y) = \frac{1}{\sqrt{(2\pi)^d|R|}} \exp\left(-\frac{1}{2}(X_Y - Y)^T R^{-1} (X_Y - Y)\right), \quad (11)$$

onde *d* é a dimensão das variáveis X_Y e *Y*.

Escrevendo na forma de *log-likelihood*, a restrição se torna:

$$\log\big(L(X_Y|Y)\big) = \log\left(\frac{1}{\sqrt{(2\pi)^d|R|}}\right) - \frac{1}{2}(X_Y - Y)^T R^{-1} (X_Y - Y). \quad (12)$$

Note que o primeiro termo da subtração e a constante multiplicativa $-1/2$ do segundo termo não são dependentes do valor de X_Y e, por esse motivo, podem ser ignoradas durante a otimização. Com isto, a restrição gerada pelos sensores de posicionamento global pode ser aproximada por:

$$\log\big(L(X_Y|Y)\big) \approx (X_Y - Y)^T R^{-1} (X_Y - Y). \quad (13)$$

As observações realizadas por sensores de deslocamento com erros gaussianos podem ser modeladas por:

$$r = X_{to} - (X_{from} + \delta), \quad (14)$$

onde X_{from} é a pose de origem, X_{to} é a pose de destino, δ é o deslocamento entre X_{from} e X_{to} medido pelo sensor e *r* é um erro gaussiano com média 0 e matriz covariância *Q*. A *likelihood* destas observações pode ser definida como:

$$L\big(X_{from}, X_{to}\big|\delta\big) = \frac{1}{\sqrt{(2\pi)^d|Q|}} \exp\left(-\frac{1}{2}\Big(X_{to} - \big(X_{from} + \delta\big)\Big)^T Q^{-1} \Big(X_{to} - \big(X_{from} + \delta\big)\Big)\right), \quad (15)$$

onde d é a dimensão das variáveis X_{from}, X_{to} e δ.

Escrevendo na forma de *log-likelihood*, a restrição se torna:

$$\log\Big(L\big(X_{from}, X_{to}\big|\delta\big)\Big) = \log\left(\frac{1}{\sqrt{(2\pi)^d|Q|}}\right) - \frac{1}{2}\Big(X_{to} - \big(X_{from} + \delta\big)\Big)^T\ Q^{-1}\ \Big(X_{to} - \big(X_{from} + \delta\big)\Big). \tag{16}$$

Note que, novamente, o primeiro termo da subtração e a constante multiplicativa $-1/2$ do segundo termo não são dependentes dos valores de X_{from} e X_{to} e, por esse motivo, podem ignoradas durante a otimização. Ao fazer isso, a restrição gerada pelos sensores de deslocamento pode ser aproximada por:

$$\log\Big(L\big(X_{from}, X_{to}\big|\delta\big)\Big) \approx \Big(X_{to} - \big(X_{from} + \delta\big)\Big)^T\ Q^{-1}\ \Big(X_{to} - \big(X_{from} + \delta\big)\Big). \tag{17}$$

Substituindo as fórmulas (13) e (17) na equação (9) o problema do GraphSLAM passa a ser definido por:

$$\begin{aligned} X^* = argmin_X & \\ & -\textstyle\sum_{Y\in M_G}(X_Y - Y)^T\ R_Y^{-1}\ (X_Y - Y) \\ & -\textstyle\sum_{\delta\in M_D}\Big(X_{to} - \big(X_{from} + \delta\big)\Big)^T\ Q_\delta^{-1}\ \Big(X_{to} - \big(X_{from} + \delta\big)\Big). \end{aligned} \tag{18}$$

onde R_Y é a covariância da medida *Y* e Q_δ é a covariância da observação δ.

A título de informação, é válido mencionar que a inversa da matriz de covariância recebe o nome de matriz de informação e pode ser vista como a influência da restrição na função objetivo. Esta afirmação é justificada observando que se uma medida possui uma covariância alta (erro grande), ela terá uma informação baixa (certeza pequena) e, da mesma forma, se uma medida possui uma covariância baixa (erro pequeno), ela terá um alto nível de informação (certeza grande). Como a matriz de informação é um fator multiplicativo das restrições, ela naturalmente irá dar

um peso maior para restrições mais precisas e um peso menor para restrições menos precisas.

Além disso, vale notar que tanto as restrições geradas pelos sensores de deslocamento quanto aquelas obtidas pelo uso de sensores de posicionamento global são restrições quadráticas não-lineares. Devido a esta característica, métodos baseados em gradiente, com o gradiente conjugado, representam uma opção natural para a solução do problema de otimização [9].

4 SISTEMA DE MAPEAMENTO DE GRANDES REGIÕES

Este capítulo descreve em detalhes o Sistema de Mapeamento de Grandes Regiões desenvolvido neste trabalho. A seção 4.1 traz uma visão geral do sistema e apresenta em alto nível as subatividades que compoem o sistema. Em seguida, cada uma das subatividades é detalhada nas seções seguintes, 4.2, 4.3, 4.4 e 4.5.

4.1 Visão Geral

A Figura 3 traz um diagrama de fluxo do Sistema de Mapeamento de Grandes Regiões (*Large-scale Environment Mapping System* – LEMS). Inicialmente, os dados, capturados pelos sensores (ou provenientes de um arquivo de *log*), passam por uma fase de pré-processamento, na qual os dados são calibrados e sincronizados. Em seguida, acontece um teste de fechamento de loop e, se for detectado que alguma região já foi visitada, é executado um procedimento para medir o deslocamento entre as poses da primeira visita e as poses medidas nas visitas seguintes. Os dados resultantes servem como entrada para o GraphSLAM, que utiliza essas informações para calcular as poses mais prováveis. Por fim, os dados são percorridos novamente e as poses otimizadas são utilizadas para construir um mapa de *grid*.

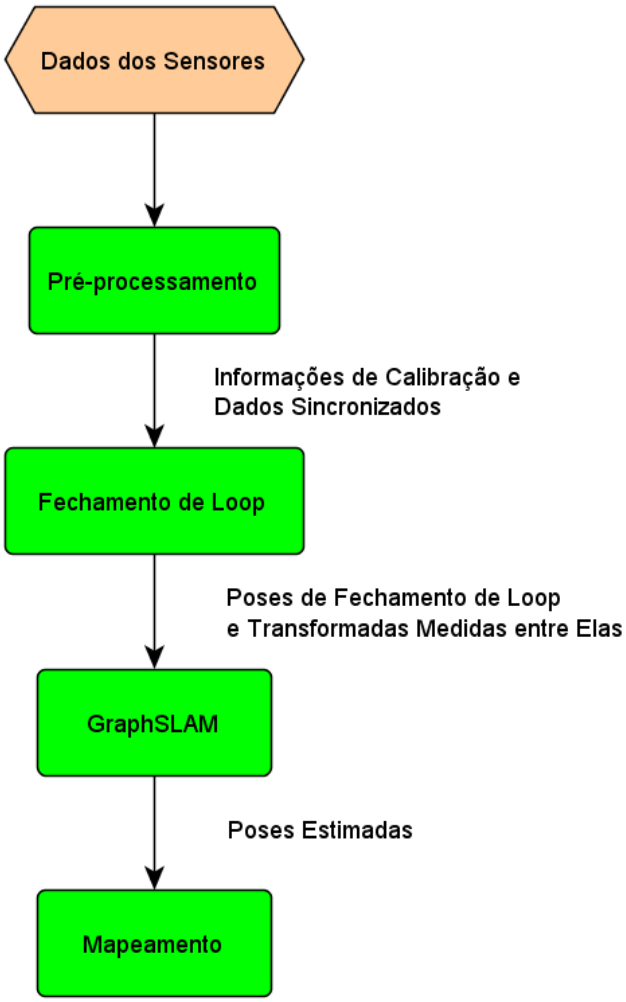


Figura 3: Diagrama de Fluxo do LEMS

4.2 Pré-Processamento

Um diagrama de fluxo da fase de pré-processamento é apresentado na Figura 4. A fase de pré-processamento possui dois objetivos: calibrar os dados dos sensores e sincronizá-los. Os sensores utilizados neste trabalho foram um sistema de posicionamento global (*Global Positioning System* – GPS), uma unidade de medidas inerciais (*Inertial Measurement Unit* – IMU), um odômetro e um sensor laser LiDAR

(*Light Detection And Ranging*) 3D *Velodyne* (a Seção 5.1 apresenta mais detalhes sobre os sensores e como eles foram montados na plataforma robótica).

No decorrer deste trabalho, foi observado que o odômetro da plataforma robótica possuía um *bias* multiplicativo na medida da velocidade e um *bias* aditivo e outro multiplicativo na medida do ângulo do volante. Embora na literatura os *biases* sejam frequentemente considerados infinitesimais e ignorados, neste trabalho a presença destes *biases* causou efeitos desastrosos, entre os quais a não-convergência do GraphSLAM. Por essa razão, foi criada uma ferramenta para encontrar automaticamente o *bias* e integrá-lo às medidas de odometria.

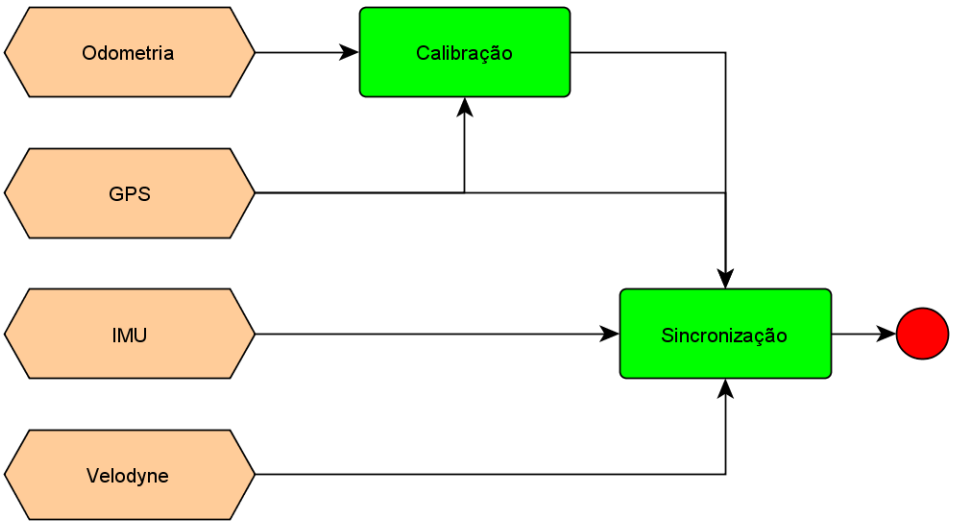


Figura 4: Diagrama de Fluxo da fase de pré-processamento

Para encontrar estes *biases,* foi desenvolvido um algoritmo de otimização que teve como objetivo encontrar os parâmetros que fizessem as poses obtidas pelo *dead-reckoning* se aproximarem o máximo possível daquelas medidas pelo GPS. Como o GPS não possui informações angulares, o ângulo inicial do robô foi adicionado como uma variável extra do problema. Embora os dados da IMU possam ser utilizados para prover o ângulo inicial, o grande erro da medida (até 30 graus, no nosso caso) poderia fazer com que o algoritmo de otimização não encontrasse uma solução.

Uma observação que precisa ser citada é que não foi adicionado um *bias* aditivo para a velocidade para evitar que, quando o robô estivesse parado, fosse medida uma velocidade diferente de zero.

Formalizando o algoritmo desenvolvido, seja V_T a velocidade do veículo e φ_T o ângulo do volante no instante de tempo *T*. Defina ainda B_V^* como o *bias* multiplicativo da velocidade e B_φ^+ e B_φ^* como os *bias* aditivo e multiplicativo do ângulo do volante, respectivamente. Então a velocidade corrigida V_T' e o ângulo do volante corrigido φ_T' são dados por:

$$V_T' = V_T * B_V^*. \tag{19}$$

$$\varphi_T' = \varphi_T * B_\varphi^* + B_\varphi^+. \tag{20}$$

A função objetivo do algoritmo a ser minimizada foi definida como a média dos erros quadráticos (em inglês, *Mean of Squared Errors* – MSE) entre o *dead-reckoning*, usando a velocidade e o ângulo do volante corrigidos, e as medidas realizadas pelo GPS em cada instante de tempo:

$$e = \frac{1}{N}\sum_T (Y_T - G(V_T',\, \varphi_T')\,)^{\,2}, \tag{21}$$

onde N é o número de poses utilizadas na otimização, *G(.)* é o modelo de movimento do veículo e Y_T é a medida do GPS instante *T*.

Para realizar a otimização foi usado o algoritmo de Otimização por Enxame de Partículas (do inglês, *Particle Swarm Optimization* – PSO) [68]. O PSO é um algoritmo evolutivo de otimização em que cada possível solução é representada por um indivíduo (também comumente chamado de partícula). Inicialmente, é criada uma população composta por um número pré-definido de indivíduos com valores aleatórios de características. A cada iteração, é calculado o valor da função objetivo (no contexto de algoritmos evolutivos, chamada de função de *fitness*) para cada indivíduo e, em seguida, a população se move em direção às regiões mais

promissoras. Estas instruções são repetidas por um certo número de iterações ou até que os movimentos das partículas sejam suficientemente pequenos.

Para identificar regiões promissoras, o PSO faz um uso de várias topologias, dentre as quais as famosas são a topologia do Melhor Global (em inglês, *Global Best* – GBEST) [69] e a topologia do Melhor Local (em inglês, *Lobal Best* – LBEST) [70]. Na topologia GBEST, o movimento dos indivíduos acontece partindo das melhores posições individuais (termo técnico: *Particle Best* – PBEST) em direção à melhor posição de toda a população. Já na topologia LBEST, o movimento das partículas acontece da posição PBEST em direção à melhor posição do melhor vizinho de cada partícula. Na topologia GBEST, as regiões mais promissoras estão nas redondezas da melhor partícula encontrada, enquanto na topologia LBEST, existem várias regiões promissoras locais, circundando os melhores indivíduos de cada vizinhança. Por esta razão, a topologia GBEST usualmente é utilizada em problemas que possuem um único máximo global, enquanto a topologia LBEST é utilizada em problemas multimodais. Neste trabalho, foi adotada a topologia GBEST.

Para atualizar as partículas, na topologia GBEST, as partículas são submetidas a uma aceleração aleatória com direção e sentido dado pelo vetor que liga a posição do PBEST e a posição do GBEST. Essa aceleração se manifesta como uma mudança na velocidade da partícula (inicialmente definida como zero) e é usada para atualizar os seus valores. Definindo como I_K a k-ésima partícula, V_K a sua velocidade, P_K sua melhor posição (PBEST) e G_B como a posição da melhor partícula de toda a população (GBEST), a regra de atualização da velocidade V_K e da posição de P_K é dada por:

$$V_K = V_K + C_1 R_1 (P_K - I_K) + C_2 R_2 (G_B - I_K)\ e \tag{22}$$

$$I_K = I_K + V_K, \tag{23}$$

onde C_1 e C_2 são parâmetros pré-definidos do sistema, cuja soma deve ser maior que 4 e R_1 e R_2 são número aleatórios entre 0 e 1.

Para realizar a calibração dos *biases* usando o PSO, são criados aleatoriamente várias partículas I_K, cada uma representando um possível conjunto de valores para os *biases* e para o ângulo inicial do robô. Para cada uma delas, é calculado o *dead-reckoning* e as poses encontradas juntamente com os dados do GPS são utilizados para calcular a função objetivo. A partícula com o *dead-reckoning* mais próximo ao caminho medido pelo GPS e, logo, com menor valor da função objetivo, é definida como GBEST G_B. Após a definição do GBEST, todas as partículas I_K são aceleradas em direção ao GBEST usando a Equação (22) para calcular a velocidade V_K e, em seguida, a Equação (23) para calcular a nova posição I_K da partícula dada a velocidade V_K. Durante o movimento, algum indivíduo pode encontrar um *dead-reckoning* melhor que o anterior e se tornar o novo GBEST. Este procedimento é repetido até que seja detectada a convergência do algoritmo. Neste trabalho, o único critério de convergência utilizado foi o número de iterações do algoritmo. Os experimentos realizados mostraram que, com a integração dos *biases* à odometria, o caminho medido pelo GPS pôde ser reproduzido com uma boa precisão pelo *dead-reckoning*.

Após a correção das medidas de odometria, os dados de todos os sensores foram sincronizados de acordo com o tempo do sensor mais lento, o Velodyne. Para realizar a sincronização, os dados mais frequentes, provenientes do GPS, da IMU e da odometria, foram armazenados em uma lista de mensagens até que uma mensagem do Velodyne fosse recebida. Assim que uma mensagem do Velodyne era recebida, a lista de mensagens dos outros sensores era percorrida e os dados com os instantes de captura mais próximos ao do Velodyne eram retornados. Durante a sincronização, os dados de GPS e IMU foram integrados para formar uma pose completa, sendo a posição x-y dada pelo GPS e ângulo dado pela IMU.

4.3 Fechamento de Loop

Após a fase de pré-processamento, os dados dos sensores são enviadas para a segunda etapa do LEMS: a detecção de fechamentos de loop e medida do deslocamento entre as poses.

A Figura 5 traz um diagrama de fluxo da fase de fechamento de loop. Para detectar se uma determinada região já foi visitada, a posição de GPS mais recente é comparada com todas as outras medidas anteriores. Se for encontrada alguma posição com uma distância menor que um determinado valor (5 metros, neste trabalho) e uma diferença temporal[2] maior que um dado limiar (2 minutos, neste trabalho), assume-se que foi detectado um fechamento de loop. Neste caso, o par de posições, a atual e a anterior, é transmitido para a etapa seguinte de medida do deslocamento entre as poses. A distância temporal é importante para prevenir que fechamentos de loop sejam detectados quando o carro está parado ou trafegando com baixas velocidades.

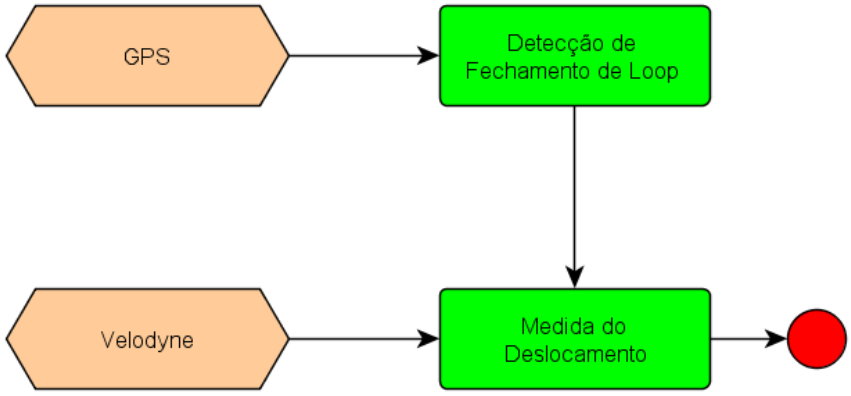


Figura 5: Diagrama de fluxo da fase de Fechamento de Loop

Se duas ou mais posições satisfizerem às condições fechamento de loop, apenas a posição com menor distância é retornada. Se todas as posições fossem usadas, o

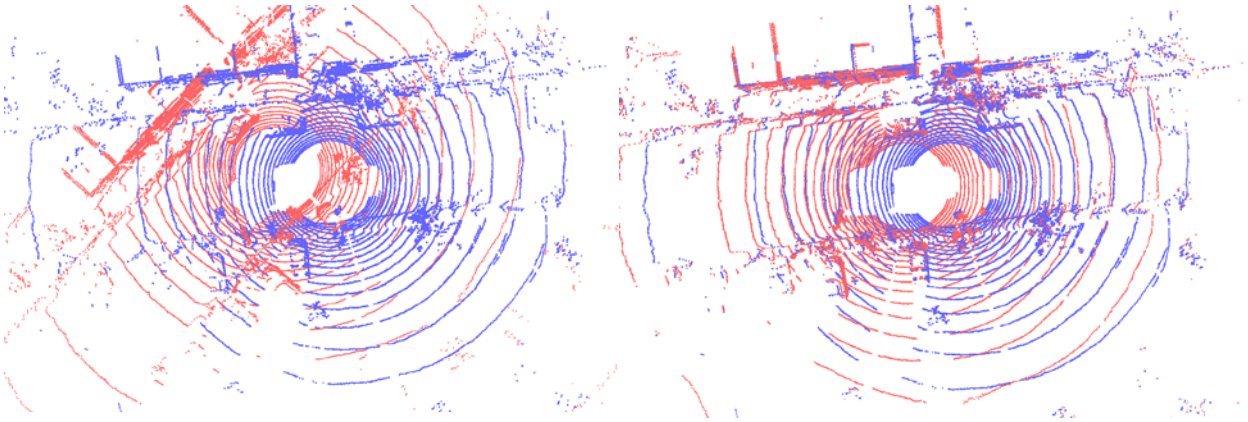
[2] A distância temporal é definida como a diferença absoluta de tempo entre duas mensagens

sistema poderia, estatisticamente, alcançar uma maior precisão nas medidas de deslocamento. Entretanto, o preço a ser pago por isso seria um aumento considerável no custo computacional.

É válido citar que o uso do GPS para detectar fechamentos de loop foi motivado pelo fato de que o erro do GPS, diferente dos demais sensores, é limitado por um valor máximo e não é acumulativo ao longo da trajetória percorrida pelo robô.

Os pares de poses resultantes da detecção de fechamento de loop passam para a segunda etapa do procedimento onde são calculados os deslocamentos entre cada par de poses. Para medir o deslocamento entre as poses, as nuvens de pontos do laser foram projetadas nas respectivas posições de GPS (os dados da IMU foram usados para adicionar informação angular) e um algoritmo de registro de nuvens de pontos foi usado alinhar as nuvens de pontos.

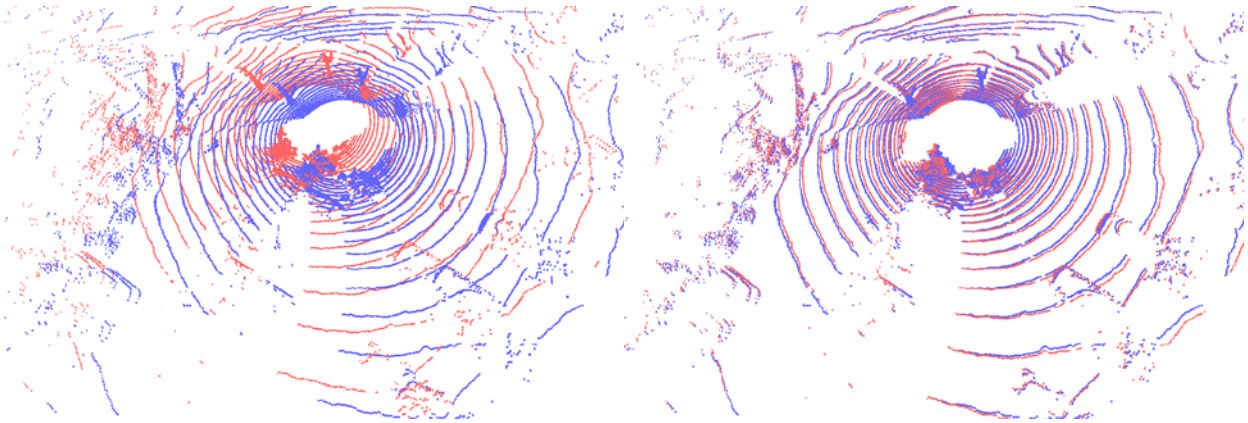
O algoritmo de registro de nuvens de pontos utilizado foi o *Generalized Iterative Closest Point* (GICP) [64]. O GICP é um algoritmo de *matching* probabilístico do tipo *point-to-plane* especialmente desenvolvido para lidar com nuvens de pontos 3D. A Figura 6 e a Figura 7 ilustram exemplos de execução do algoritmo GICP. Em ambos os casos, a nuvem de pontos em azul corresponde à primeira visita do veículo à região e a nuvem de pontos em vermelho corresponde à segunda visita. Nos dois exemplos, a Figura (a) apresenta as nuvens de pontos antes do alinhamento e a Figura (b) mostra as nuvens de pontos após a correção do GICP. Para alinhar as nuvens de pontos, o GICP busca planos na nuvem capturada durante a primeira visita do robô a uma região e, em seguida, calcula o deslocamento que melhor ajusta a nuvem de pontos da segunda visita a estes planos [64].



(a) Nuvens de pontos antes da aplicação do GICP

(b) Nuvens de pontos após a aplicação do GICP

Figura 6: Exemplo de aplicação do algoritmo GICP



(a) Nuvens de pontos antes da aplicação do GICP

(b) Nuvens de pontos após a aplicação do GICP

Figura 7: Exemplo de aplicação do algoritmo GICP

4.4 GraphSLAM

O diagrama de fluxo do algoritmo de GraphSLAM é apresentado na Figura 8. O GraphSLAM é, basicamente, dividido em duas etapas: a construção do grafo e a otimização das poses. Na primeira etapa, os dados dos sensores são utilizados para inicializar os vértices e adicionar as arestas no grafo. Na segunda etapa, um

algoritmo de otimização é usado para encontrar os valores mais prováveis para as poses.

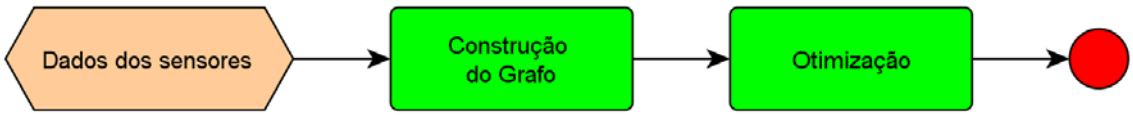


Figura 8: Diagrama de Fluxo do GraphSLAM

A Figura 9 apresenta o papel dos dados na construção do grafo. A cor e o tipo de linha dos *boxes* foram trocados para enfatizar o fato que essa figura *não* é um diagrama de fluxo, mas apenas uma ilustração dos papéis que os dados possuem na construção do grafo. No GraphSLAM, os vértices representam as variáveis que deseja-se estimar (poses do robô) e as arestas representam as observações realizadas pelos sensores (GPS/IMU, odometria e fechamento de loop) e as respectivas matrizes de covariância.

A Figura 10 traz uma ilustração do grafo construído durante o GraphSLAM. Os círculos com bordas contínuas em preto representam as poses, as arestas vermelhas tracejadas representam as relações criadas pelos dados de odometria, as arestas unitárias com traço contínuo em verde correspondem aos dados de GPS/IMU e as arestas azuis com linhas do tipo traço e ponto representam as relações de fechamento de loop.

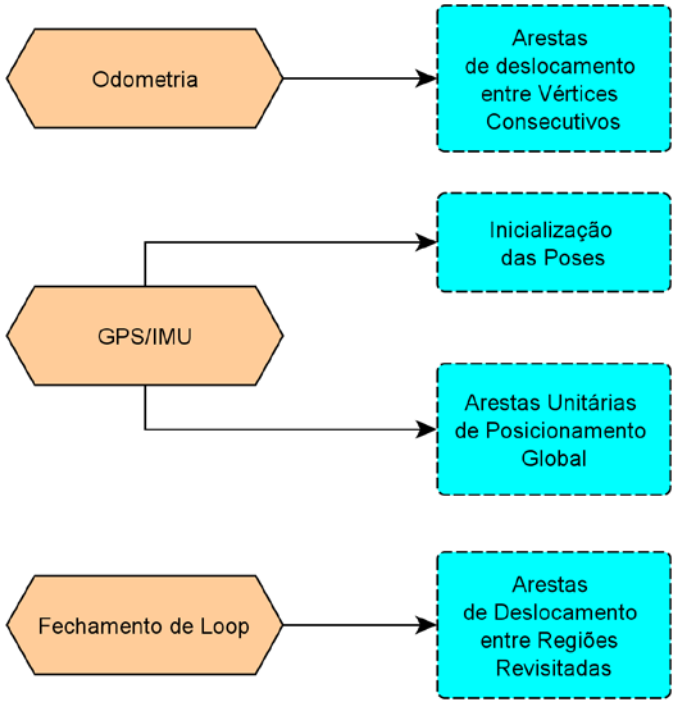


Figura 9: Papéis dos sensores no GraphSLAM

Os dados de GPS/IMU foram usados para inicializar os vértices e para adicionar restrições de posicionamento global. Estas restrições são representadas por arestas unitárias no grafo e são usadas para limitar o espaço de busca do algoritmo de otimização. Os dados de GPS/IMU são adequados tanto para a inicialização quanto para a adição de arestas globais porque o erro de ambos os sensores é limitado independente do tamanho do caminho percorrido.

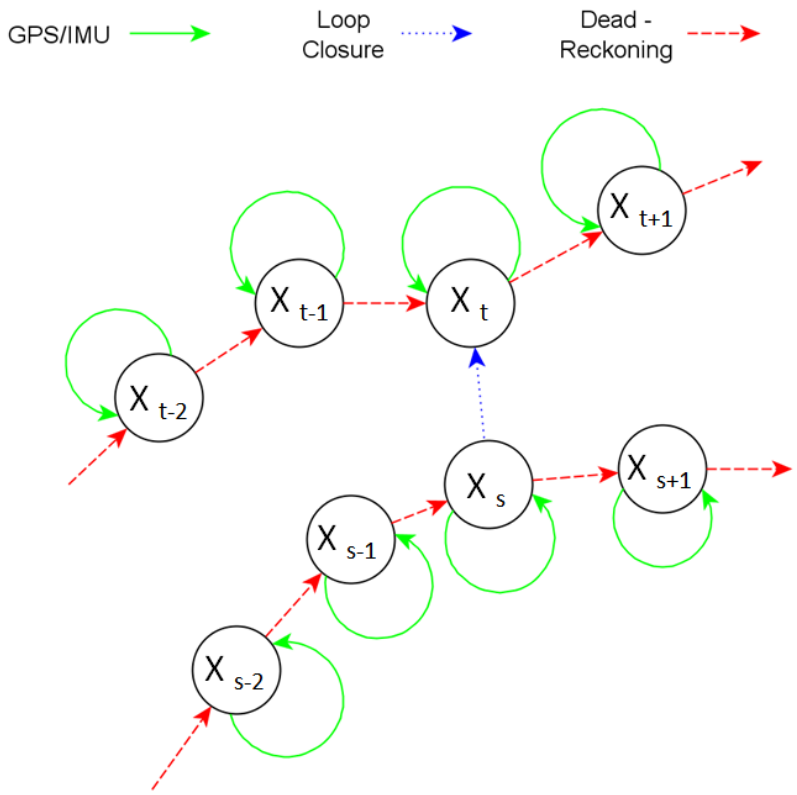


Figura 10: Visualização do grafo construído durante o GraphSLAM

Os dados da odometria aplicados ao modelo de movimento Ackermann [71] foram usados para estimar o movimento do robô de um instante de tempo para o seguinte. Este movimento foi representado no grafo como uma aresta de um vértice para o seguinte. Já os dados de fechamento de loop, foram representados como arestas de deslocamento entre poses do robô em diferentes visitas a uma mesma região.

Após a construção do grafo, o GraphSLAM passa para a sua segunda etapa: a otimização das poses. Nesta etapa, um algoritmo de otimização é executado para encontrar os valores das poses mais prováveis dadas as observações realizadas pelos sensores. A função objetivo a ser minimizada é o negativo da *log-likelihood* dado por:

$$J = R_O + R_G + R_L, \tag{24}$$

onde R_O é o somatório das restrições de odometria, R_G é o somatório das restrições de GPS/IMU e R_L é o somatório das restrições de fechamento de loop.

Como a odometria e o fechamento de loop geram restrições de deslocamento e o GPS/IMU geram restrições de posicionamento global, na forma de *log-likelihood*, a função objetivo se torna:

$$J = \sum_{T} (X_T - (X_{T-1} + \delta_T))^T \; Q_T^{-1} \; (X_T - (X_{T-1} + \delta_T)) + \sum_{T} (X_T - Y_T)^T \; R_T^{-1} \; (X_T - Y_T) + \sum_{X_A, X_B \in Loops} \left(X_B - \left(X_A + \delta_{X_A}^{X_B}\right)\right)^T \; S_{X_A, X_B}^{-1} \; \left(X_B - \left(X_A + \delta_{X_A}^{X_B}\right)\right), \tag{25}$$

onde X_T é a pose do instante T, δ_T é o deslocamento entre as poses X_T e X_{T-1}, Y_T é a medida de GPS/IMU no instante T, $\delta_{X_A}^{X_B}$ é o deslocamento entre as poses de fechamento de loop X_A e X_B, calculado usando o GICP, e Q_T^{-1}, R_T^{-1} e S_{X_A, X_B}^{-1} são as inversas das matrizes de covariância da odometria, do GPS/IMU e do fechamento de loop, respectivamente.

Vale ressaltar que os valores que irão mudar durante a otimização serão as poses (na fórmula, X_T, X_{T-1}, X_A e X_B) e os valores que ficarão fixos são as observações dos sensores (na fórmula, δ_T, Y_T, $\delta_{X_A}^{X_B}$ e as matrizes de covariância (na fórmula, Q_T, R_T e S_{X_A, X_B}).

4.5 Mapeamento

Uma vez que as poses do robô são conhecidas, o algoritmo para criação dos mapas de *grid* se torna muito simples e consiste apenas de percorrer novamente os dados dos sensores e usar as poses para projetar os dados no mapa. A ênfase deste trabalho foi a criação de mapas do tipo *grid* de ocupação.

Nos *grids* de ocupação, o ambiente visitado pelo robô é subdividido em um quadriculado composto por células de mesma dimensão. Cada uma dessas células armazena a probabilidade de existir um obstáculo dentro dos seus limites. Usualmente, os algoritmos que usam *grids* de ocupação representam a probabilidade de ocupação usando *log-odds* para evitar instabilidades numéricas nos casos de probabilidades próximas de um ou de zero. O *log-odds* (*L*) pode ser obtido a partir da probabilidade (*P*) usando a seguinte relação:

$$L = \log\left(\frac{P}{1-P}\right). \tag{26}$$

Da mesma forma, a probabilidade pode ser facilmente recuperada a partir do *log-odds* usando a relação inversa:

$$P = 1 - \frac{1}{1+e^{L}}. \tag{27}$$

O primeiro passo para a construção de um *grid* de ocupação é a inicialização das células do mapa com uma probabilidade a *priori* neutra L_0. Esta probabilidade informa que as células podem tanto estar ocupadas quanto serem livres. A partir daí, sempre que novos dados dos sensores são recebidos, as células no campo perceptual do robô são atualizadas com um incremento ou decremento na probabilidade de estarem ocupadas, segundo a seguinte equação:

$$L_T = L_{T-1} + \Delta L - L_0, \tag{28}$$

onde L_T é o *log-odds* de uma determinada célula no instante *T*, L_{T-1} é o *log-odds* da célula no instante *T–1* e ΔL é o incremento ou decremento do *log-odds* de acordo com a medida realizada pelo sensor.

O incremento ou decremento do *log-odds* é calculado de acordo com um valor que indica se um obstáculo foi detectado ou não pelo sensor. Para detectar se uma determinada célula está ocupada ou não, foram utilizados os dados capturados pelo Velodyne. Estes dados consistem de um feixe de 32 lasers verticais que giram

horizontalmente de forma a criar uma nuvem de pontos 3D, em coordenadas esféricas, do ambiente ao redor do robô (Figura 11). O Velodyne possui um raio de atuação de até 70 metros e é capaz de medir distâncias com um erro médio de 2 centímetros.

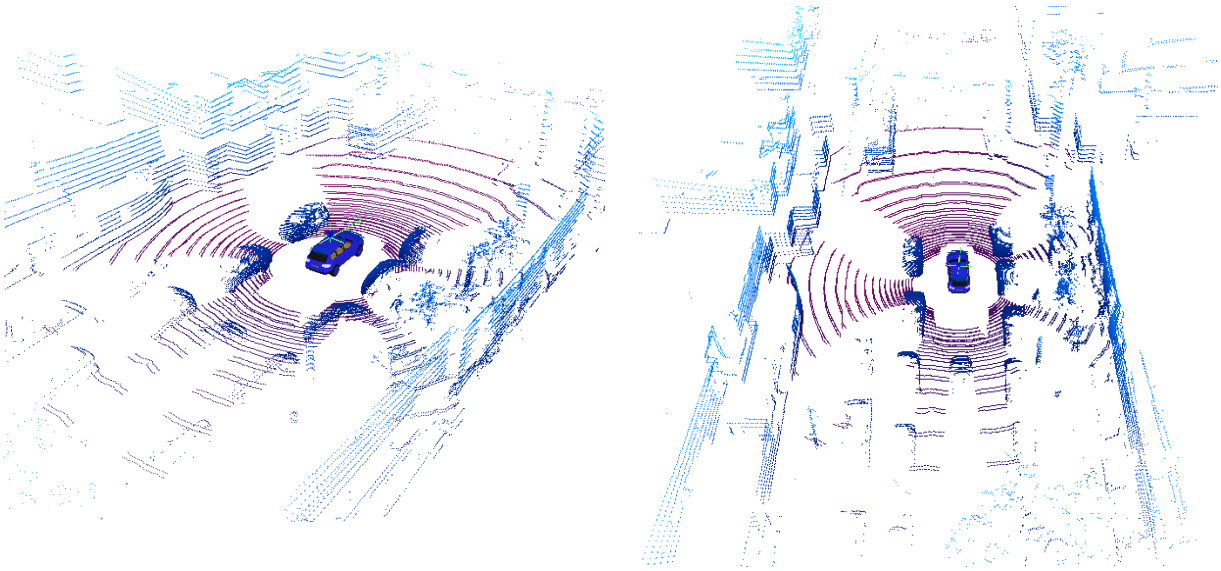


Figura 11: Nuvens de pontos do Velodyne

A detecção de obstáculos é feita usando o feixe de lasers verticais do Velodyne e repetida para cada ângulo horizontal. Para detectar obstáculos, as distâncias medidas por cada laser, começando pelo mais baixo, são comparadas com aquelas medidas realizadas pelo sensor localizado imediatamente acima. Se o robô estiver navegando em um plano onde não existam obstáculos, ambos os lasers irão medir a distância do sensor até o plano. Se existirem obstáculos, entretanto, as medidas realizadas serão menores que a distância do sensor até o plano (Figura 12). Para calcular o indício de que os raios do Velodyne tenham tocado um obstáculo, os dois raios dos lasers são projetados no chão e a diferença entre o tamanho dos raios é medida. Se o robô estiver navegando em um plano, esta diferença será igual a um determinado valor esperado. Se existir um obstáculo no campo perceptivo do robô, a diferença medida será menor que o valor esperado.

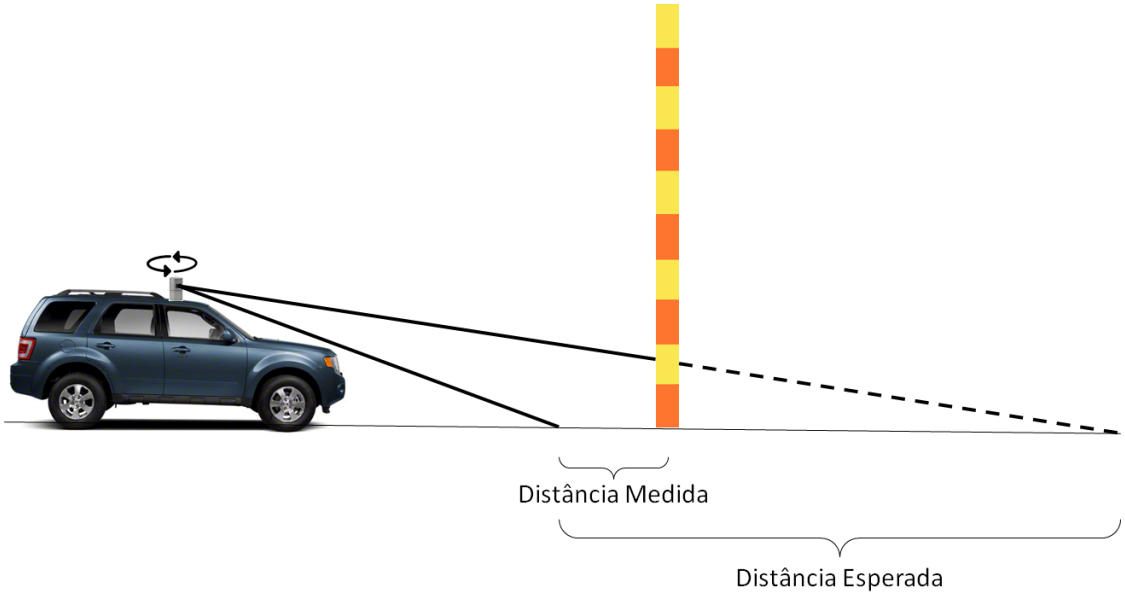


Figura 12: Exemplo de detecção de obstáculo

Formalizando a ideia, seja D_E a diferença esperada entre o tamanho de dois raios verticais do Velodyne projetados no plano quando não existem obstáculos e D_M a diferença medida entre estas projeções. Então, o valor (entre 0 e 1) do indício de que um obstáculo tenha sido detectado pode ser calculada por:

$$P_O = \frac{D_E - D_M}{D_E}, \tag{29}$$

onde D_E é a diferença esperada entre dois raios do Velodyne projetados no chão e D_M é a diferença medida entre estes raios. Se o indício calculado for maior que um determinado limiar, a célula tocada pelo laser recebe um incremento na probabilidade de ocupação.

O cálculo de D_M é direto e consiste apenas em projetar os dois raios do Velodyne no plano e subtrair os seus respectivos tamanhos. O cálculo de D_E, por outro lado, é mais sofisticado e será descrito em detalhes. Para auxiliar a explicação, a Figura 13 traz um modelo geométrico que descreve como dois raios do Velodyne tocariam o

chão se o veículo estivesse navegando em um plano sem obstáculos. Na figura, *H* representa a altura do sensor em relação ao plano (altura do veículo + distância do teto do veículo até o centro do sensor), α e β correspondem aos ângulos verticais dos raios do Velodyne, *B* é o tamanho do primeiro raio projetado no chão, *X* é o tamanho esperado da projeção do segundo raio na ausência de obstáculos e D_E é a diferença entre os tamanhos dos dois raios projetados no chão. A distância medida pelo laser superior é dada por R_1 e os ângulos ω e γ são ângulos auxiliares utilizados nos cálculos. As variáveis conhecidas *a priori* são *H*, α, β e R_1 e a variável que deseja-se encontrar é D_E.

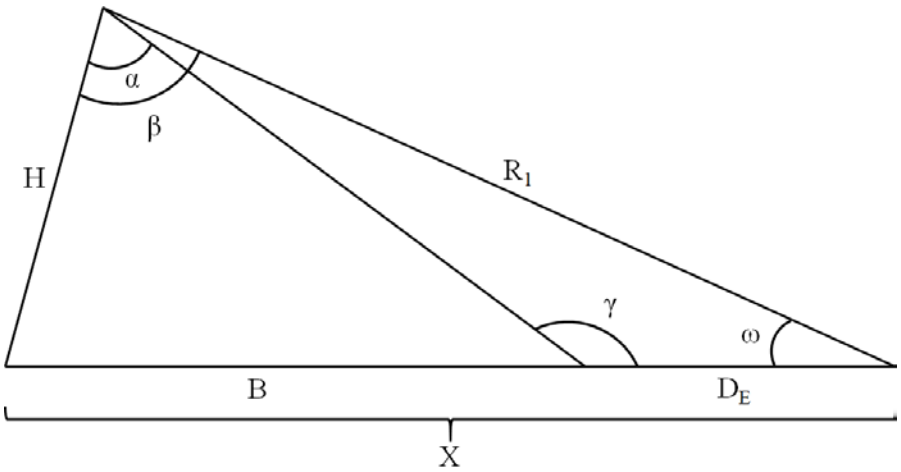


Figura 13: Modelo geométrico usado para o cálculo do valor esperado entre dois raios verticais do Velodyne projetados no chão quando o automóvel está em um plano

O primeiro passo para calcular o valor de D_E é a aplicação da lei dos senos usando o triângulo interno localizado mais à direita:

$$\frac{\sin(\gamma)}{R_1} = \frac{\sin(\beta - \alpha)}{D_E}. \tag{30}$$

Isolando D_E, obtém-se:

$$D_E = \frac{R_1 \sin(\beta - \alpha)}{\sin(\gamma)}. \quad (31)$$

Dessa forma, para solucionar o problema, basta se calcular o valor de γ. Para isto, observe que:

$$\gamma = \pi - \omega - (\beta - \alpha). \quad (32)$$

Ao fazer isto, o problema se resume a calcular o valor de ω. Mas ω pode ser calculado usando mais uma vez a lei dos senos:

$$\frac{\sin(\beta)}{X} = \frac{\sin(\omega)}{H}. \quad (33)$$

Isolando ω, obtém-se:

$$\omega = \arcsin\left(\frac{H\,sin(\beta)}{X}\right). \quad (34)$$

Finalmente, para calcular o valor *X*, basta aplicar a lei dos cossenos usando o triângulo mais externo:

$$X^2 = H^2 + R_1^2 - 2HR_1 \cos(\beta). \quad (35)$$

Logo:

$$X = \sqrt{H^2 + R_1^2 - 2HR_1 \cos(\beta)}. \quad (36)$$

Com isto, substituindo a Equação (36) em (34), em seguida a Equação (34) em (32) e, por fim, a Equação (32) em (31), a distância esperada entre os dois raios consecutivos do Velodyne passa a ser dada por:

$$D_E = \frac{R_1 \sin(\beta - \alpha)}{\sin\left(\pi - (\beta - \alpha) - \arcsin\left(\frac{H\,sin(\beta)}{\sqrt{H^2 + R_1^2 - 2HR_1\cos(\beta)}}\right)\right)}. \quad (37)$$

Após aplicar esta técnica de detecção dos obstáculos em todos os lasers de um feixe vertical, o obstáculo mais próximo do robô é selecionado e todas as células localizadas entre o robô e este obstáculo recebem um incremento na probabilidade de serem livres. Este incremento é justificado pelo fato de que se existisse um obstáculo em alguma dessas células, algum dos raios iria medir uma distância menor até o robô e um obstáculo mais próximo seria detectado.

Por fim, para permitir o mapeamento de grandes regiões, foi utilizada uma técnica de submapeamento. Nesta abordagem, o ambiente mapeado pelo robô foi subdividido em vários blocos de 50x50 metros, todos com uma resolução de 20 centímetros (62.500 células por bloco). Para evitar o consumo excessivo de memória, quando os blocos são carregados para a memória do computador, eles são carregados em grupos de nove, formando um bloco maior de 150x150 metros. Um gerenciador de submapas é responsável por manter o robô sempre no bloco central e, quando ele sai desta região, novos blocos são carregados para a memória, integrados aos que já existem e aqueles que não são mais necessários são salvos em disco e descartados da memória.

A Figura 14 traz uma ilustração do procedimento de troca de submapas. Assuma que inicialmente o robô está localizado no submapa 4 (bloco central) e começa a se deslocar para cima em direção ao bloco 1 (Figura 14 (a)). Assim que o robô ultrapassa a divisa do bloco central, três novos blocos são carregados (blocos 9, 10 e 11) e incorporados ao mapa mantido em memória (Figura 14 (b)). Ao fazer isto, o bloco 1 passa a ser o bloco central e os blocos 6, 7 e 8 são salvos em disco e descartados da memória (Figura 14 (c)). Por fim, os blocos são renomeados e voltam ao estado mostrado na Figura 14 (a). Caso seja necessário carregar algum bloco de 50x50 metros que ainda não tenha sido salvo em disco, são criados novos

blocos, com todas as células inicializadas com L_0, para compor o mapa de 150x150 metros.

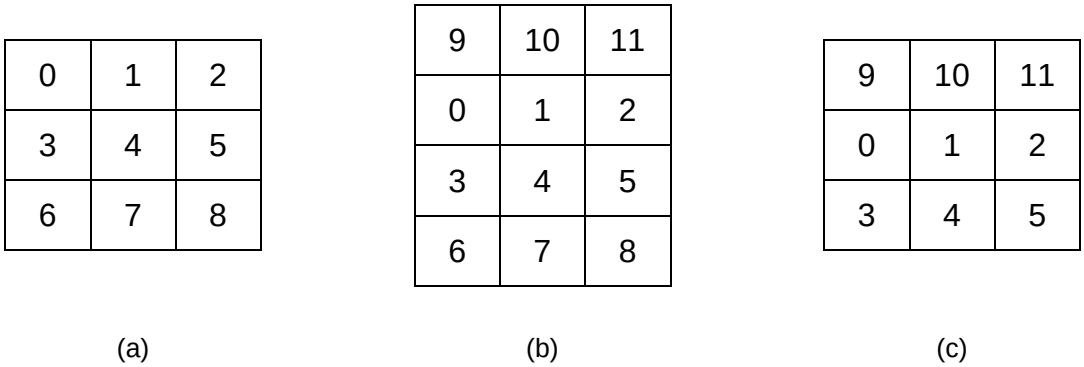


Figura 14: Ilustração do sistema de gerenciamento de submapas

Por fim, é importante dizer que, neste trabalho, não foram utilizados mecanismos para lidar com obstáculos móveis. Por esta razão, algumas células do mapa que, no mundo real, são livres podem ter sido marcadas como obstáculos se estiverem momentaneamente ocupadas por veículos, pessoas ou outras entidades em movimento. Em algumas situações, quando as células são vistas mais de uma vez, o próprio algoritmo de mapeamento foi capaz de marcar as células novamente como livres. Em outras, entretanto, a célula permaneceu incorretamente marcada como ocupada e, nestas situações, um *expert* humano precisou manualmente marcar algumas células do mapa como livres.

5 METODOLOGIA E MATERIAIS

Este capítulo apresenta a metodologia e os materiais utilizados neste trabalho. A seção 5.1 descreve a plataforma robótica IARA e os seus componentes. A seção 5.2 introduz o *framework* CARMEN, responsável pela comunicação entre os módulos de *software* e as seções 5.3 e 5.4 citam as bibliotecas utilizadas para implementação do GraphSLAM e do GICP, respectivamente. Por fim, a seção 5.5 descreve em detalhes a implementação do LEMS e sua integração ao *framework* CARMEN.

5.1 A Plataforma Robótica IARA

Para testar os algoritmos apresentados neste trabalho, foi utilizada a plataforma robótica IARA (*Intelligent Autonomous Robotic Automobile*). IARA foi construída com base no automóvel de passeio Ford Escape Hybrid (Figura 15 e Figura 16), com diversas adaptações, como a instalação de sensores e a instalação de mecanismos de controle do acelerador, freio e posição do volante por meio dos computadores instalados no porta-malas. A tecnologia eletrônica de acionamento dos atuadores (volante, acelerador, freio, entre outros) do automóvel foi desenvolvida pela empresa *Torc Robotics* [72].

Figura 15: Plataforma IARA

Figura 16: Recursos computacionais da IARA

A Figura 17 mostra os sensores disponíveis na IARA que incluem câmeras estéreo *Bumblebee XB3* da *Point Grey* (Colúmbia Britânica, Canadá) (http://ww2.ptgrey.com/stereo-vision/bumblebee-xb3), um *Light Detection And Ranging* (LiDAR) HDL-32E da Velodyne (Califórnia, Estados Unidos) (http://velodynelidar.com/lidar/hdlproducts/hdl32e.aspx), e um *Attitude and Heading Reference System* (AHRS) MTi da Xsens (Califórnia, Estados Unidos) (http://www.xsens.com/products/mti/). A Figura 18 traz uma ilustração de como os sensores foram instalados na plataforma robótica IARA.

(a) (b) (c)

(d)

Figura 17: (a) Câmera estéreo *Bumblebee* XB3 da *Point Grey*; (b) LIDAR HDL-32E da *Velodyne*; (c) AHRS/GPS MTi-G da *Xsens*; (d) Computador *Dell Precision* R5500.

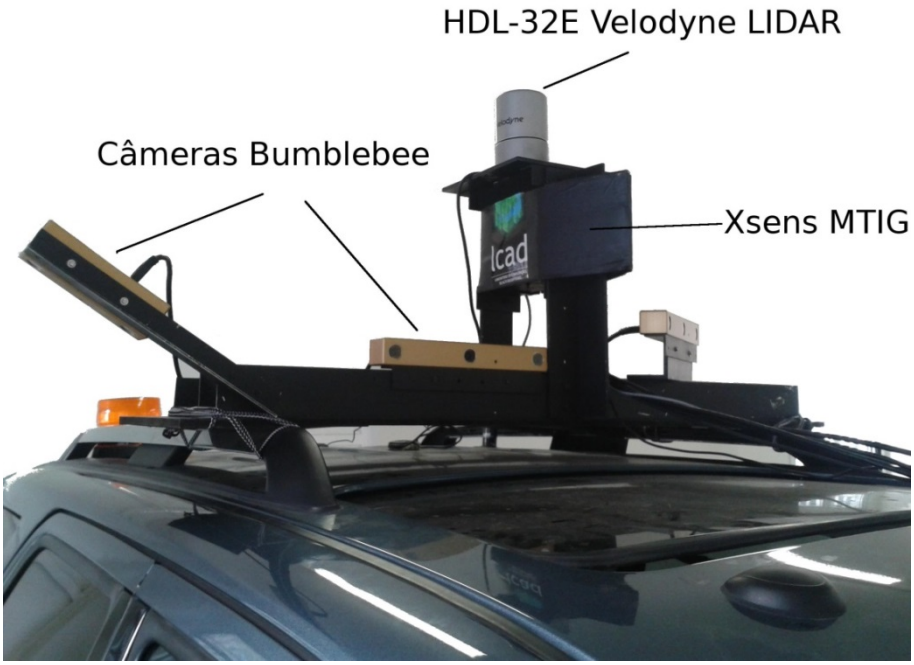


Figura 18 – Sensores da IARA

Para processar todos os dados dos sensores, IARA também conta com seis computadores Dell Precision R5500 (2 Processadores Intel Xeon 2.13 GHZ, 12 GB de memória DDR3 1333MHZ, 2 HDs SSD 120GB em RAID0, 2 Placas de Rede 1GB, Placa de Vídeo Quadro 600, Placa de Vídeo Tesla C2050). Todos os computadores utilizaram sistema operacional Ubuntu 12.04 com *kernel real-time*. O uso de HDs SSD foi importante para permitir o armazenamento de grandes quantidades de dados sem perda de informação. De forma similar, o uso de sistemas operacionais com *kernel real-time* foi importante para garantir uma alta precisão nas medidas de tempo das mensagens.

5.2 O *Framework* CARMEN

Os algoritmos apresentados neste trabalho foram desenvolvidos na linguagem C/C++ como um módulo do *framework* CARMEN. O *framework* CARMEN é um conjunto de softwares para controle de robôs. Ele foi projetado para fornecer uma

interface consistente e um conjunto de primitivas básicas para pesquisa em robótica em uma ampla variedade de plataformas de robôs comerciais. O objetivo do *framework* é diminuir a barreira entre o desenvolvimento de novos códigos tanto para robôs simulados quanto para robôs reais, e também facilitar o compartilhamento de algoritmos entre diferentes instituições [73].

O *framework* CARMEN utiliza uma arquitetura orientada a serviços composta por diversos softwares modulares. Cada módulo representa uma habilidade do robô independente das demais. Entre os módulos existentes no CARMEN estão: o módulo de localização, o módulo de planejamento de movimento e o módulo de GraphSLAM. A comunicação entre os módulos é feita através de um protocolo de comunicação chamado de *Inter Process Communication* (IPC) [74], utilizando principalmente o paradigma de comunicação *Publish-Subscribe*.

A implementação do padrão *Publish-Subscribe* disponibilizada pelo IPC possui diversas vantagens, como: flexibilidade, eficiência e capacidade de trocar mensagens com estruturas de dados complexas, incluindo listas e vetores de tamanho variável. Além disso, um atrativo do IPC é a possibilidade de comunicação entre processos sendo executados em diferentes computadores por meio do protocolo TCP/IP.

A Figura 19 apresenta o modelo de comunicação *Publish-Subscribe* disponibilizado pelo IPC. Neste modelo, a gerência da comunicação é realizada por um processo chamado Central. O Central é responsável por receber as mensagens publicadas e repassá-las para os módulos interessados. Os módulos que publicam mensagens são chamados de *Publishers*, enquanto que os módulos que recebem mensagens são chamados de *Subscribers*. Vale notar que um módulo pode desempenhar um papel de *Publisher* e de *Subscriber* simultaneamente. Um módulo se torna um *Subscriber* assim que ele demonstra interesse em receber uma determinada mensagem. Esse interesse é representado por uma assinatura da mensagem enviada ao Central. Já para se tornar um *Publisher*, um módulo precisa publicar uma mensagem. O envio de uma nova mensagem acontece através de um pedido de publicação enviado ao Central. No IPC, a recepção de mensagens é feita de forma

assíncrona, ou seja, cada S*ubscriber* possui uma função de *callback* associada a cada mensagem assinada e sempre que uma nova mensagem é recebida, a respectiva função de *callback* é executada.

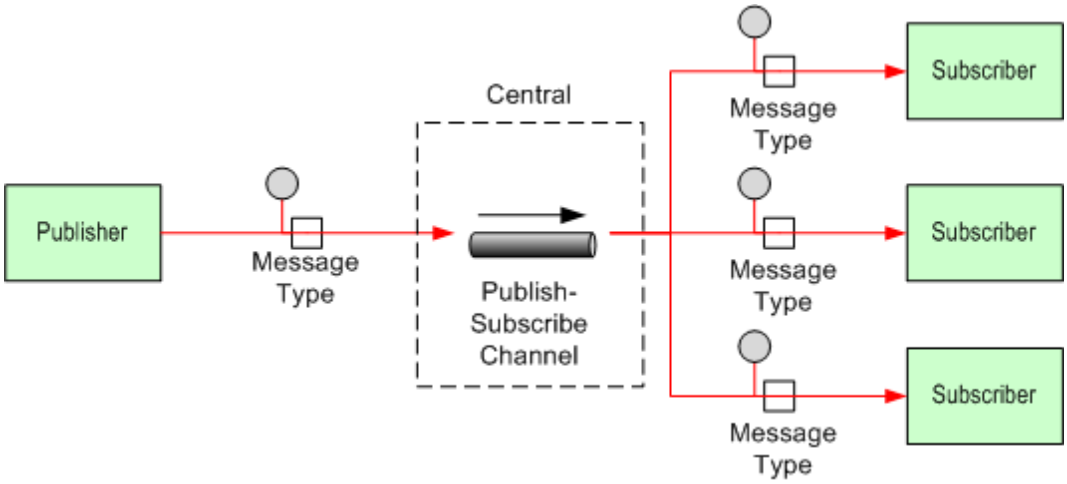


Figura 19: Modelo de comunicação *Publish-Subscribe* [75]

Entre as diversas vantagens da arquitetura orientada a serviços e da comunicação pelo IPC, podem ser citadas:

- Escalabilidade: os módulos não precisam ser executados em um mesmo processador. Módulos críticos podem ser executados em um computador separado e a comunicação acontece pela rede via IPC.
- Confiabilidade: a falha de um módulo não faz com que os outros módulos parem de funcionar. A única exceção a essa regra é o módulo Central.

O desenvolvimento da versão *open source* do CARMEN foi descontinuado em 2008; contudo, este *framework* garantiu a seus desenvolvedores a vitória na *Defense Advanced Research Projects Agency* (DARPA) *Grand Challege* de 2005 (http://archive.darpa.mil/grandchallenge05) e o segundo lugar (muito embora tenha completado o percurso em primeiro lugar) na DARPA *Urban Challenge* (http://archive.darpa.mil/grandchallenge/index.asp). Assim, embora existam outros *frameworks open source* hoje disponíveis e mantidos, a equipe do LCAD optou por

continuar o desenvolvimento do CARMEN no LCAD. Mais informações sobre o LCAD-CARMEN, estão disponíveis no link http://www.lcad.inf.ufes.br/wiki/index.php/Carmen_Robot_Navigation_Toolkit.

5.3 O *Framework* G2O

O *framework* G2O (acrônimo de *General Graph Optimization)* é um conjunto de ferramentas para otimização de problemas probabilísticos que podem ser representados como um grafo [76]. O *framework* é *open-source*, foi desenvolvido usando a linguagem C++ e teve principal linha guia a eficiência computacional. O G2O se tornou popular logo após o seu lançamento e foi utilizado em vários projetos [77,78,79]. Os criadores do G2O acreditam que a grande popularidade pode ser atribuída à facilidade de uso do *framework*, à ampla gama de aplicações devido ao seu design geral e o bom desempenho em comparação à outros *frameworks* [27]. A facilidade de uso do G2O se deve em grande parte à estrutura de classes básicas incluídas no *framework*. Estas classes são genéricas e, se especializadas, permitem a representação de uma vasta gama de problemas probabilísticos. O G2O possui uma série de exemplos que demonstram a aplicabilidade do *framework* em diversas áreas.

O GraphSLAM desenvolvido neste trabalho foi criado utilizando como base o exemplo SLAM-2D, nativo do G2O. Este exemplo possui classes utilitárias para representar transformadas 2D e as suas operações (inversão, soma, etc.), vértices que representam poses 2D (posição x-y e orientação θ), arestas de deslocamento e classes para lidar com *landmarks* (tanto vértices quanto arestas) que estão fora do escopo deste trabalho. Para implementar o GraphSLAM, o exemplo SLAM-2D foi modificado para dar suporte à arestas de posicionamento global através da extensão da classe de arestas unitárias nativa do G2O.

5.4 PointCloud Library

A implementação do algoritmo GICP utilizada neste trabalho foi proveniente da *PointCloud Library* – PCL (http://pointclouds.org). A PCL é uma biblioteca de código aberto que contém diversos algoritmos para processamento de nuvens de pontos. A biblioteca contém algoritmos estado-da-arte para: filtragem, estimativa de *features*, reconstrução de superfícies, registro e segmentação de nuvens de pontos. Ela recebe suporte da comunidade internacional de pesquisa e de desenvolvedores tanto na área de robótica quanto de percepção artificial [80].

5.5 Implementação

A Figura 20 apresenta um diagrama de fluxo de dados ilustrando os programas desenvolvidos para o LEMS e os dados transmitidos entre estes programas. Os módulos dos sensores (hexágonos na cor salmão) geram dados para os demais módulos. Os módulos de processamento (caixas com bordas arredondadas da cor verde) recebem dados e geram novos dados e os utilitários (caixas com bordas quadradas na cor amarela) são executados de forma *offline*.

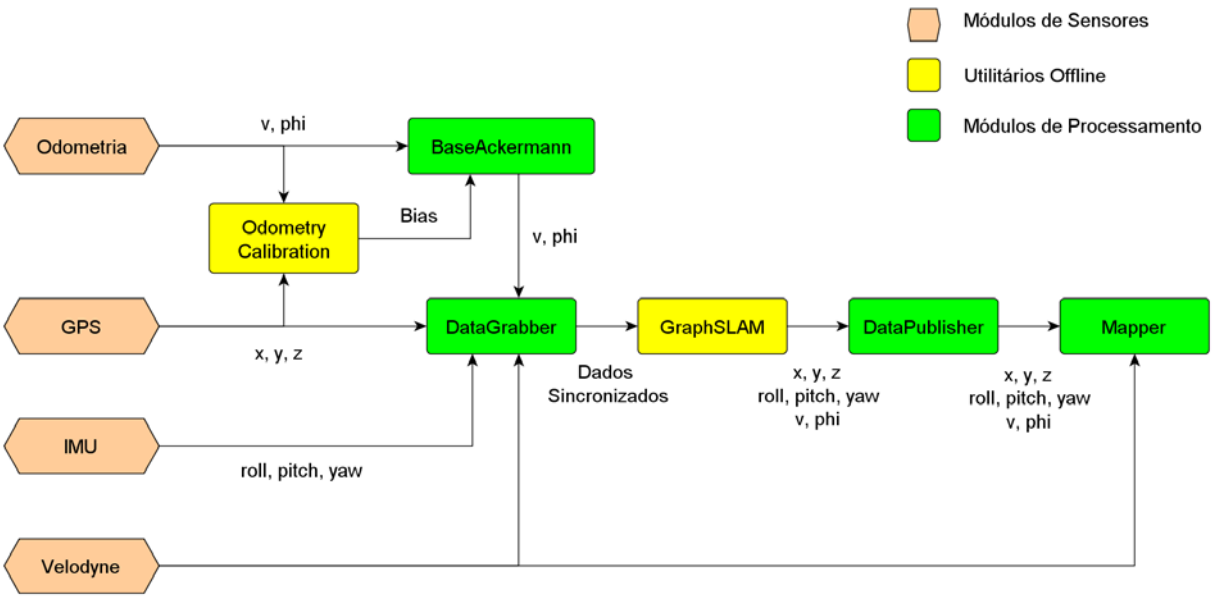


Figura 20: Módulos e Softwares utilitários do LEMS

Os módulos do CARMEN que compõe o LEMS são listados abaixo:

- **DataGrabber:** este módulo recebe como entrada as mensagens dos sensores (odometria, Velodyne, GPS e IMU), realiza a sincronização dos dados e os armazena em um arquivo de saída. As mensagens de odometria contêm a velocidade e o ângulo do volante, as mensagens do Velodyne possuem uma nuvem de pontos 3D, a mensagem do GPS possui uma posição 3D (x, y e z) e a mensagem da IMU contêm os ângulos 3D do carro (*roll, pitch* e *yaw*).
- **DataPublisher:** este módulo lê um arquivo contendo as poses otimizadas pelo GraphSLAM e as publica na forma de uma mensagem integrada contendo uma pose 6D (posição 3D, x, y, z e orientação 3D, *roll*, *pitch* e *yaw*), ângulo do volante e a velocidade do veiculo. Os dados de ângulo do volante e velocidade que são publicados são aqueles provenientes da odometria corrigida.
- **BaseAckermann:** este módulo recebe os dados crus de odometria, integra o *bias* e republica os dados como uma nova mensagem.

- **Mapper:** este módulo recebe como entrada as poses 6D e os dados do Velodyne e gera como saída os mapas de *grid* das regiões visitadas pelo robô.

Além dos módulos do CARMEN, foram criados vários softwares utilitários que podem ser executados de forma *offline*:

- **GraphSLAM:** este utilitário recebe os dados sincronizados de odometria, GPS, IMU e de fechamento de loop, realiza a otimização das poses e as salva em um arquivo de saída.
- **OdometryCalibration:** este software recebe como entrada um arquivo contendo os dados sincronizados de odometria e GPS e gera como saída os valores do *bias*, tanto aditivo quanto multiplicativo, da velocidade e do ângulo do volante.

6 EXPERIMENTOS E RESULTADOS

Este capítulo apresenta a metodologia utilizada para realização dos experimentos e os resultados alcançados com o LEMS. A seção 6.1 descreve a metodologia experimental e as bases de dados utilizadas, enquanto as seções 6.2.1, 6.2.2 e 6.2.3 apresentam os resultados do uso do LEMS para cada conjunto de dados.

6.1 Metodologia Experimental

Para avaliação do LEMS, foram realizados experimentos utilizando três conjuntos de dados com diferentes características e dimensões. O primeiro experimento foi realizado em um estacionamento arborizado localizado no interior do campus da Universidade Federal do Espírito Santo (UFES), Brasil. Por ser um ambiente de pequena escala e com muitas árvores, os erros de GPS deveriam ser os principais desafios do LEMS. O segundo experimento, foi realizado ao redor no anel viário do campus da UFES. Neste experimento, o percurso realizado foi significativamente maior que o anterior e tão arborizado quanto. O objetivo do experimento foi avaliar a capacidade do LEMS de lidar com os erros significativos no *dead-reckoning* e de manter a qualidade das poses mesmo com perdas do sinal de GPS. Por fim, foi realizado um teste de grande escala, em que a plataforma robótica IARA foi guiada por um bairro da cidade de Vitória, capital do estado do Espírito Santo (ES), Brasil. Em todos os percursos existiam áreas de fechamento de loop.

O *software* de calibração da odometria foi testado nos três percursos apresentados acima. Como a captura dos dados foi realizada com uma grande diferença de tempo (meses), assumiu-se que o *bias* teria se modificado de uma data para a outra e, por essa razão, foi realizada uma otimização do *bias* para cada experimento. Intuitivamente, existe uma expectativa de que os valores de calibração não precisariam ser recalculados se os dados fossem capturados em datas próximas (diferença de dias). Entretanto, mais experimentos precisariam ser realizados para confirmar esta hipótese. A Tabela 1 apresenta os parâmetros utilizados pelo

algoritmo PSO, e a Tabela 2 traz o espaço de busca das variáveis. Em todos os experimentos foram utilizados os mesmos parâmetros e os mesmos espaços de busca.

Tabela 1: Parâmetros utilizados no PSO

Parâmetro	**Valor**
Número de Partículas	70
Número de Iterações	200
C1	2.05
C2	2.05

Tabela 2: Espaço de busca dos parâmetros de calibração da odometria

Parâmetro	**Mínimo**	**Máximo**
Bias Multiplicativo da Velocidade (m/s)	0.7	1.3
Bias Aditivo do Ângulo do Volante (rad)	- 0.17	0.17
Bias Multiplicativo do Ângulo do Volante (rad)	0.7	1.3
Ângulo Inicial (rad)	$-\pi$	π

Após a calibração da odometria, o algoritmo GraphSLAM foi utilizado para encontrar os valores das poses com máxima *likelihood* dadas as medidas realizadas pelos sensores (GPS/IMU, *dead-reckoning* e fechamento de loop). Para visualizar os resultados, nas seções seguintes serão apresentados os mapas do tipo *grid* de ocupação para cada um dos percursos realizados. Nestes mapas, as áreas em azul são aquelas que não foram tocadas pelos sensores, as áreas em branco são áreas livres e as áreas em preto são aquelas que possuem uma alta probabilidade de estarem ocupadas. Para garantir ao leitor a lisura dos resultados, nenhuma intervenção manual foi realizada. Todos os mapas apresentados foram construídos

de forma completamente automática e, por essa razão, os ruídos causados pela passagem de objetos móveis continuam presentes.

6.2 Resultados

6.2.1 Mapeamento de um Estacionamento

O primeiro experimento realizado foi o mapeamento de um estacionamento do campus da UFES. Para isto, um motorista humano guiou a plataforma IARA em várias voltas ao redor do estacionamento. O objetivo deste experimento foi avaliar a sensibilidade do LEMS à erros de GPS, mais visíveis em ambientes de pequena escala, e o algoritmo de fechamento de loop. Como foram realizadas várias voltas ao longo do mesmo trajeto, todas as poses, exceto aquelas da primeira volta, eram poses de fechamento de loop. O percurso total percorrido foi de aproximadamente 676 metros ao longo de três voltas ao redor do estacionamento. A Figura 21 traz uma ilustração do percurso realizado.

Figura 21: Percurso realizado durante os experimentos de pequena escala em um estacionamento da UFES

O primeiro teste realizado foi a calibração dos *bias* existentes na odometria. A Figura 22 apresenta o percurso medido usando *dead-reckoning* e o percurso medido pelo GPS antes da calibração da odometria e a Figura 23 apresenta estes percursos após a calibração da odometria. Para possibilitar a comparação dos dois percursos, a primeira posição medida pelo GPS foi adotada como sistema de referência (origem) para os dados de GPS. Ao fazer isso, tanto o *dead-reckoning* quanto o GPS passaram a ter zero como posição inicial. Como pode ser observado, antes da calibração do *bias*, o *dead-reckoning* produzia um caminho muito diferente daquele medido pelo GPS. Após a calibração, por outro lado, as trajetórias se tornaram bem mais próximas, mas, mesmo assim, um ligeiro desvio pode ser observado na parte inferior esquerda da trajetória. Este desvio foi corrigido na fase seguinte usando os dados dos outros sensores e o algoritmo GraphSLAM.

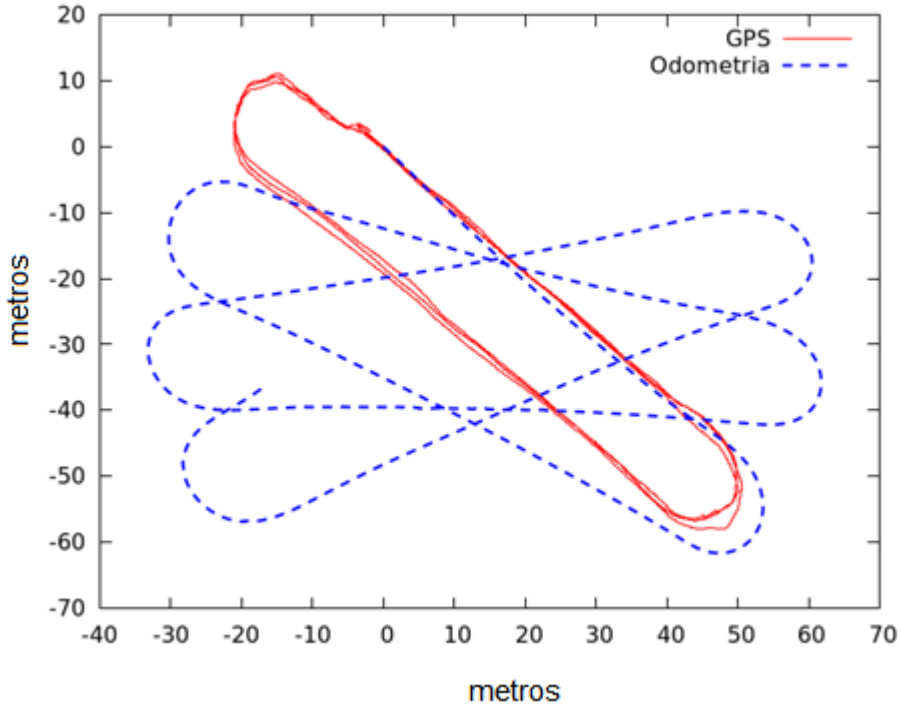


Figura 22: Visualização dos percursos medidos pelo *dead-reckoning* e pelo GPS antes da calibração do sensor de odometria

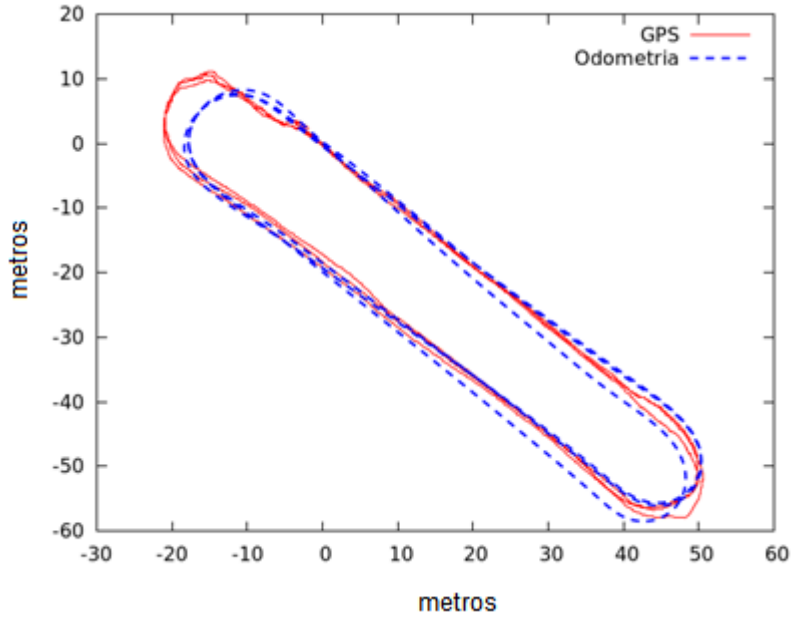


Figura 23: Visualização dos percursos medidos pelo *dead-reckoning* e pelo GPS após a calibração do sensor de odometria

A Figura 24 apresenta a evolução do erro ao longo das iterações do algoritmo de otimização usado para a calibração do *bias* e a Tabela 3 apresenta os valores de *bias* e demais parâmetros obtidos. A média dos erros quadráticos (em inglês, *Mean of Squared Error* – MSE) ao final da otimização foi de 2.34 metros ao quadrado (tirando a raiz tem-se 1.53 metros).

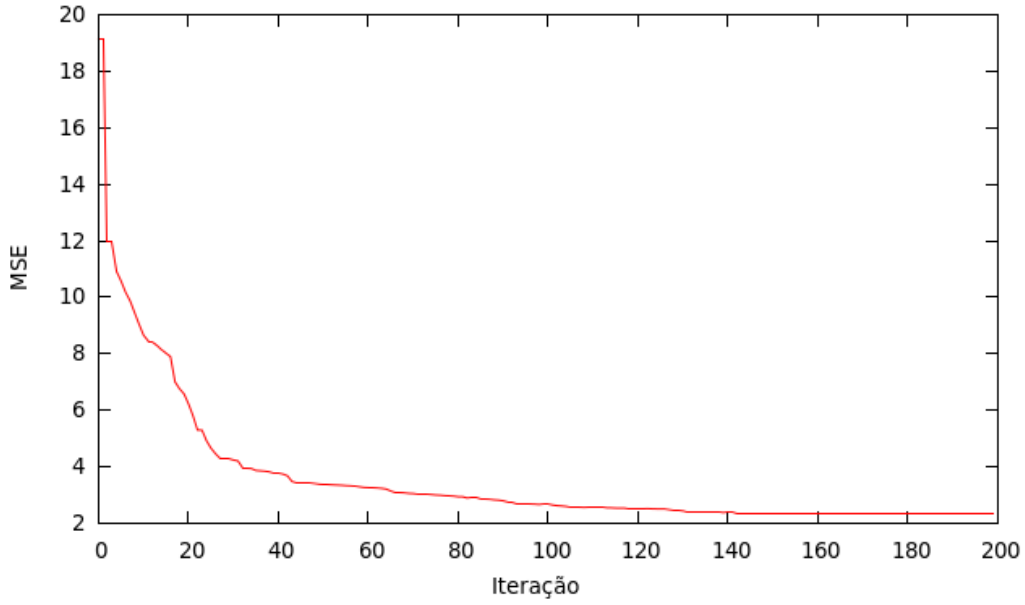


Figura 24: Evolução do erro ao longo da execução do PSO

Tabela 3: Parâmetros de calibração da odometria

Parâmetro	**Valor**
Bias Multiplicativo da Velocidade (m/s)	0.936408
Bias Aditivo do Ângulo do Volante (rad)	-0.002539
Bias Multiplicativo do Ângulo do Volante (rad)	1.104929
Ângulo Inicial (rad)	-0.781037

Após a calibração do *bias* existente na odometria, o *dead-reckoning* e os dados dos demais sensores foram utilizados como entrada para o algoritmo GraphSLAM. Este algoritmo estimou os valores das poses com máxima *likelihood*, dadas as medidas realizadas pelos sensores. A Figura 25 traz uma comparação do percurso medido pelo GPS e o percurso calculado pelo GraphSLAM. As linhas ao redor das medidas de GPS representam o erro do sensor e têm como objetivo mostrar que o caminho calculado pelo GraphSLAM é consistente com as medidas realizadas pelo GPS. Como pode ser observado, o ligeiro desvio na parte inferior esquerda da trajetória, existente no *dead-reckoning* foi corrigido com a utilização do GraphSLAM

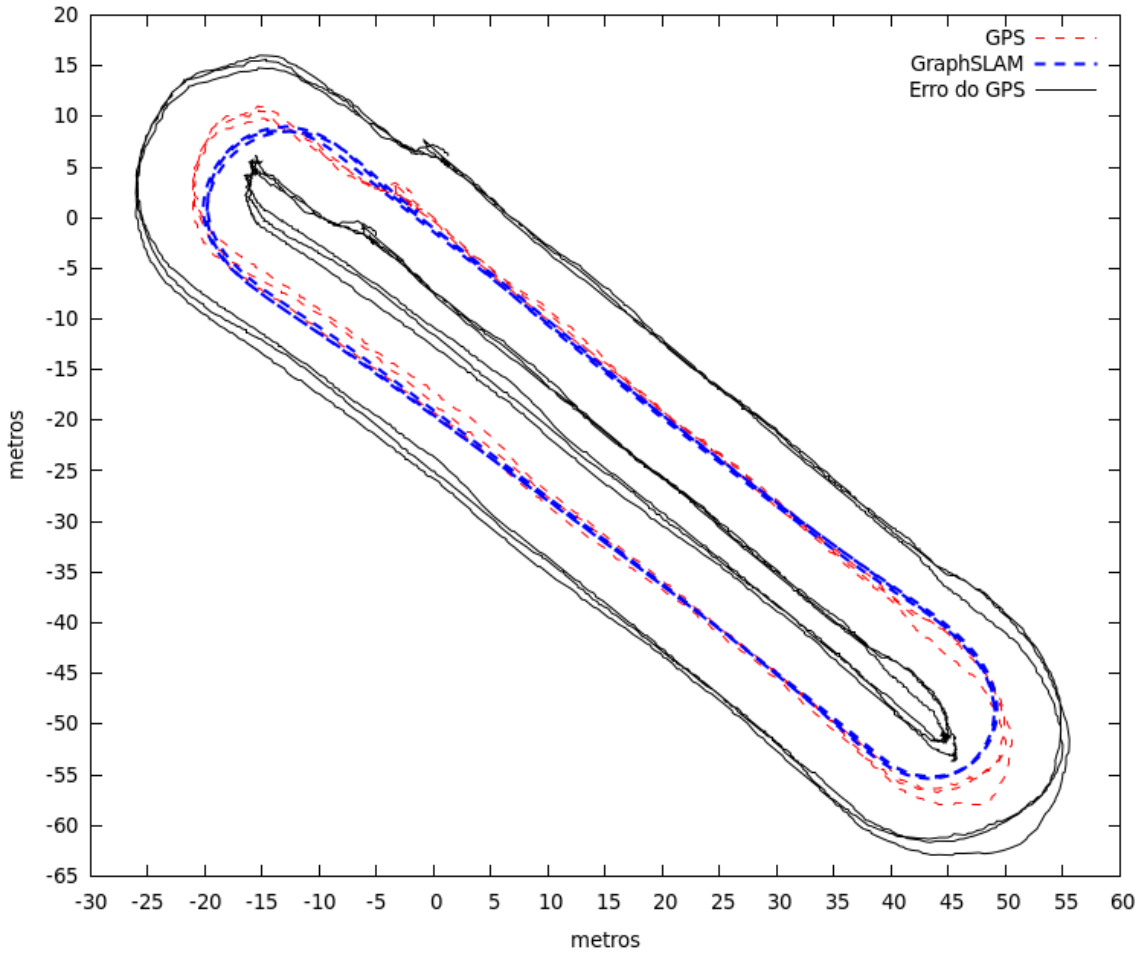


Figura 25: Comparação entre o caminho medido pelo GPS e caminho calculado pelo GraphSLAM

A Figura 26 apresenta o *grid* de ocupação final do estacionamento construído usando o LEMS. A Figura 27 (a) traz uma região do mapa em destaque e as Figura 27 (b) e (c) trazem duas visualizações da nuvem de pontos observada pelo Velodyne no instante em que o trecho da Figura 27 (a) era mapeado. Como pode ser observado, com a utilização do LEMS, foi possível criar um mapa do estacionamento de boa qualidade e, mesmo após a realização de três voltas, o mapa se manteve consistente.

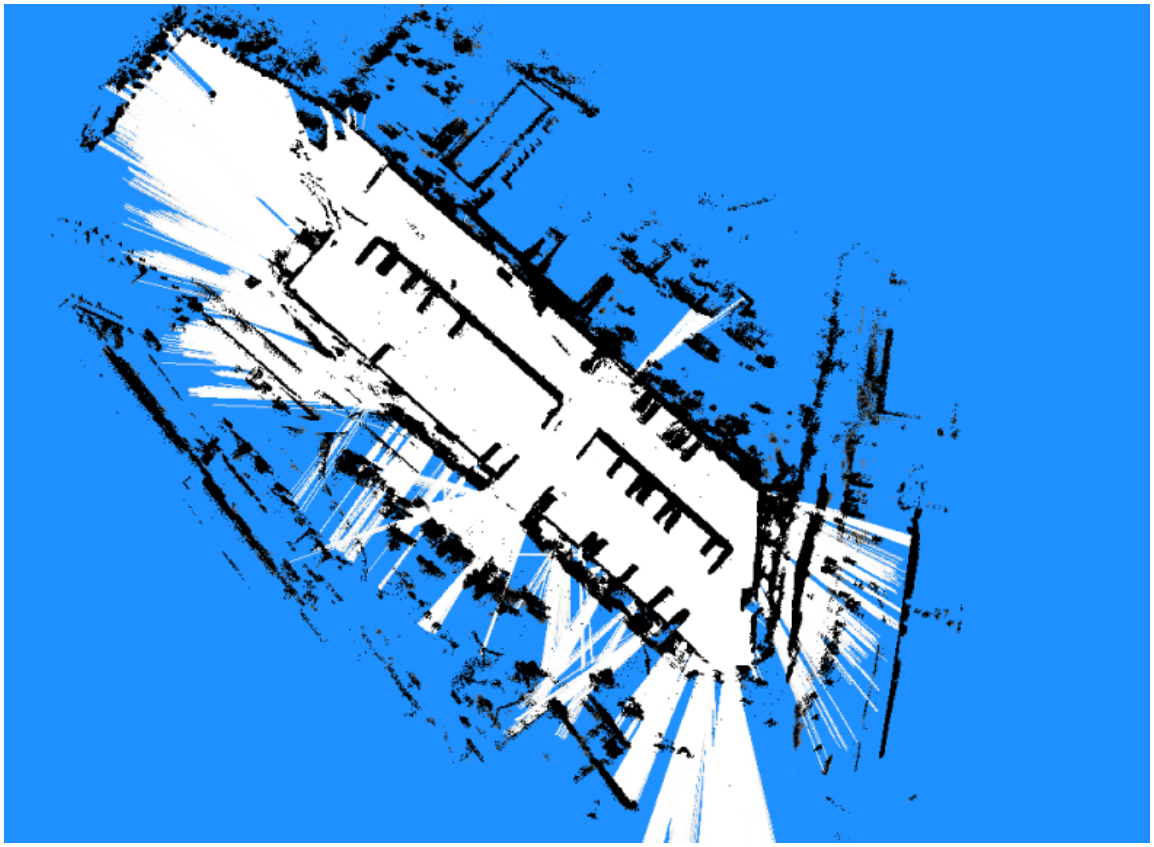
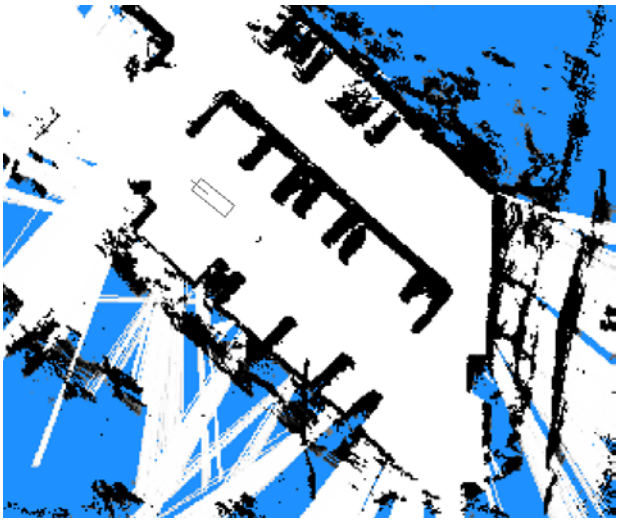
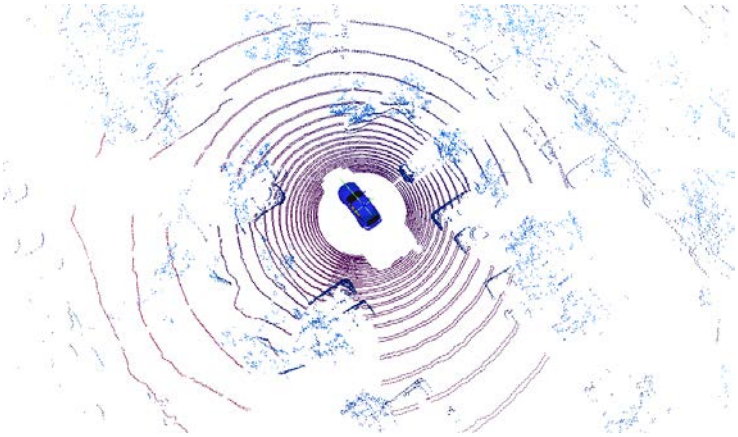


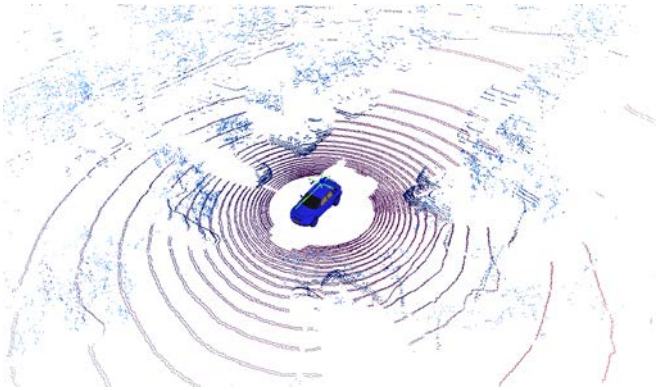
Figura 26: *Grid* de ocupação de um estacionamento da UFES construído usando o LEMS



(a) Região do Mapa em Destaque



(b) Nuvem de Pontos do Velodyne usada para criação do Mapa (Visualização 1)



(c) Nuvem de Pontos do Velodyne usada para criação do Mapa (Visualização 2)

Figura 27: Regiões em destaque do mapa do estacionamento

6.2.2 Mapeamento do Anel Viário da UFES

O segundo experimento realizado foi o mapeamento do anel viário da UFES. Assim como no experimento anterior, um motorista humano guiou a plataforma robótica ao redor de todo o anel viário da UFES. Foi realizada apenas uma volta, com um pequeno trecho de interseção para fechamento de loop. O trajeto percorrido possui aproximadamente 3.7 quilômetros e é ilustrado na Figura 28.

A Figura 29 apresenta o percurso medido usando *dead-reckoning* e o percurso medido pelo GPS antes da calibração da odometria e a Figura 30 apresenta estes percursos após a calibração da odometria. Novamente, a primeira posição medida pelo GPS foi adotada como sistema de referência (origem) para possibilitar a comparação dos trajetos. Pode ser observado que ao integrar o *bias* à odometria, houve um aumento de qualidade significativo no *dead-reckoning*. Entretanto, ainda pode ser visualizado um grande erro de fechamento de loop causado pelo acumulo de erros ao longo de toda a trajetória.

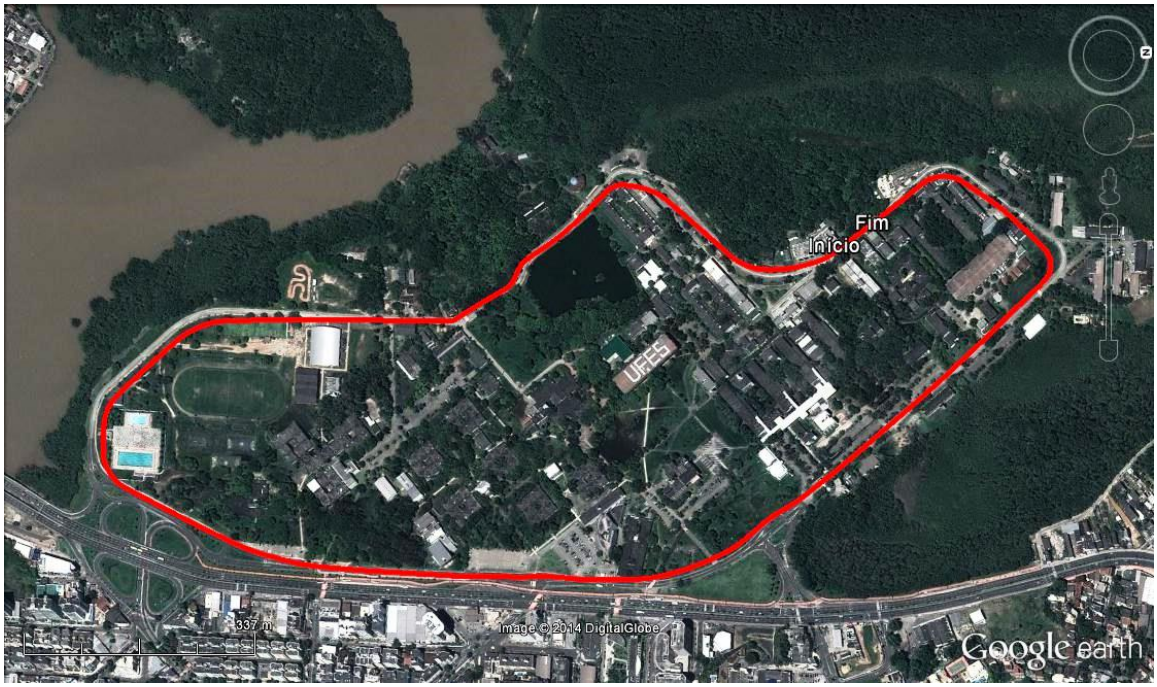


Figura 28: Percurso realizado no anel viário da UFES

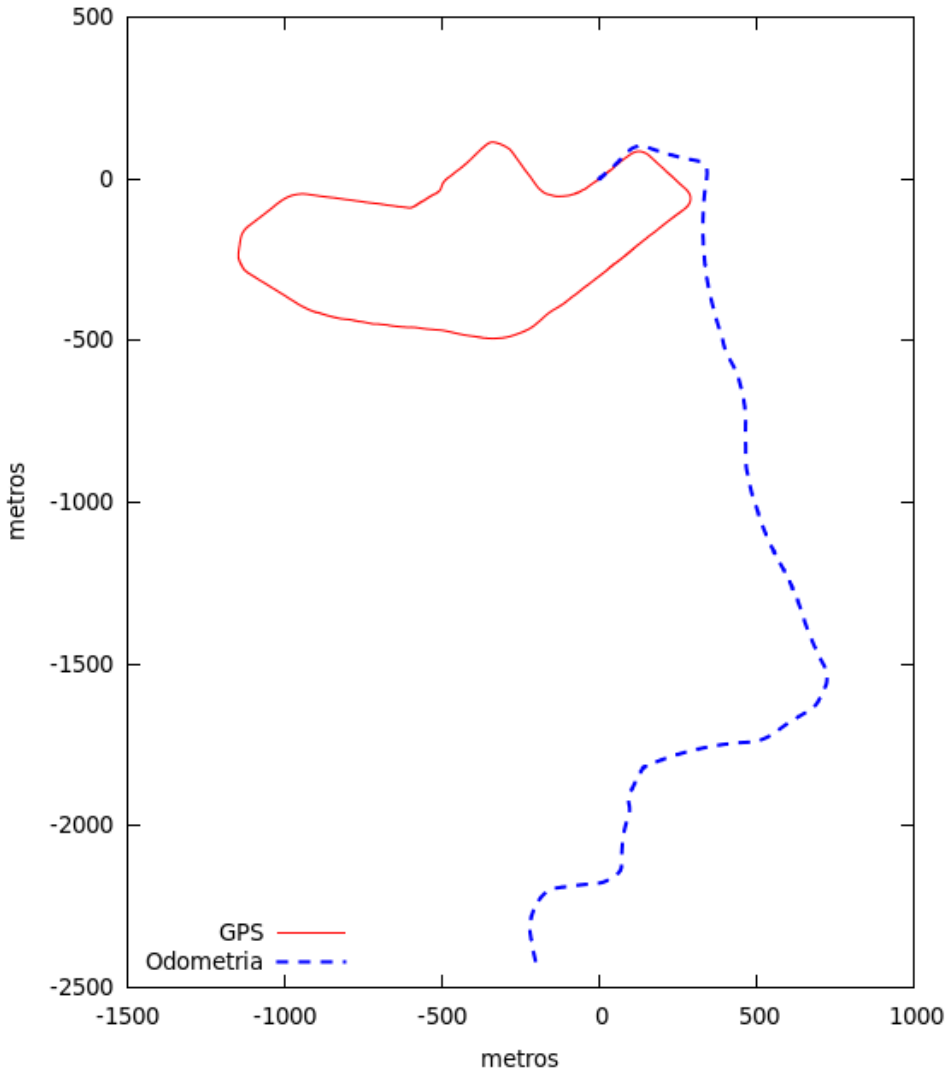


Figura 29: Visualização dos percursos medidos pelo *dead-reckoning* e pelo GPS antes da calibração do sensor de odometria

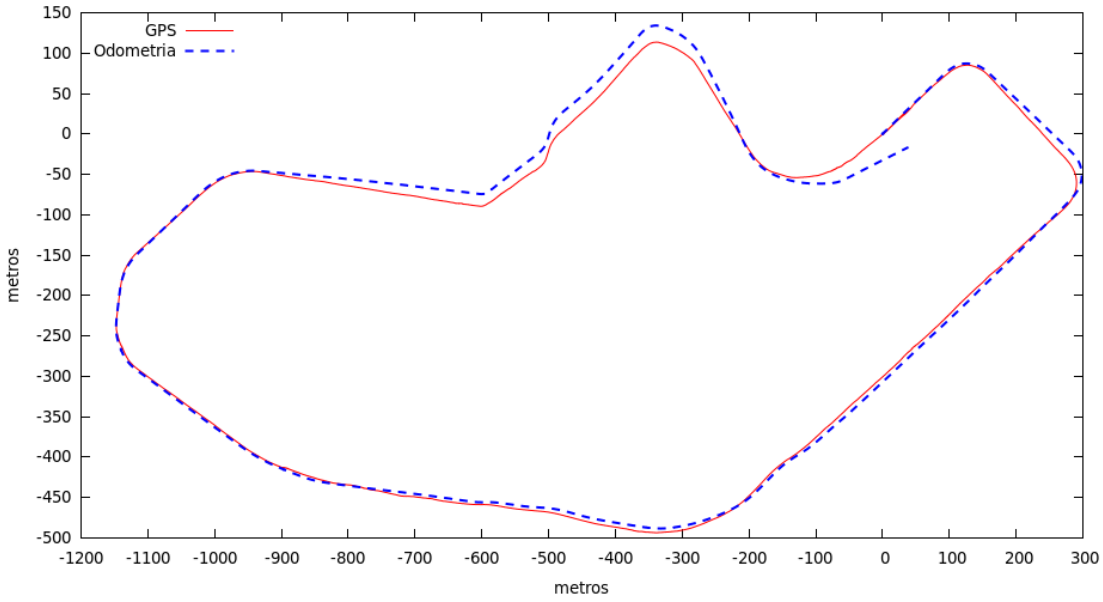


Figura 30: Visualização dos percursos medidos pelo *dead-reckoning* e pelo GPS após a calibração do sensor de odometria

A Figura 31 apresenta a evolução do erro ao longo das iterações do algoritmo de otimização e a apresenta os valores obtidos para os parâmetros. O MSE ao final da otimização foi de 8.88 metros ao quadrado (tirando a raiz tem-se 2.98 metros).

Tabela 4: Parâmetros de calibração da odometria

Parâmetro	**Valor**
Bias Multiplicativo da Velocidade (m/s)	0.984300
Bias Aditivo do Ângulo do Volante (rad)	-0.002788
Bias Multiplicativo do Ângulo do Volante (rad)	1.101828
Ângulo Inicial (rad)	-0.652184

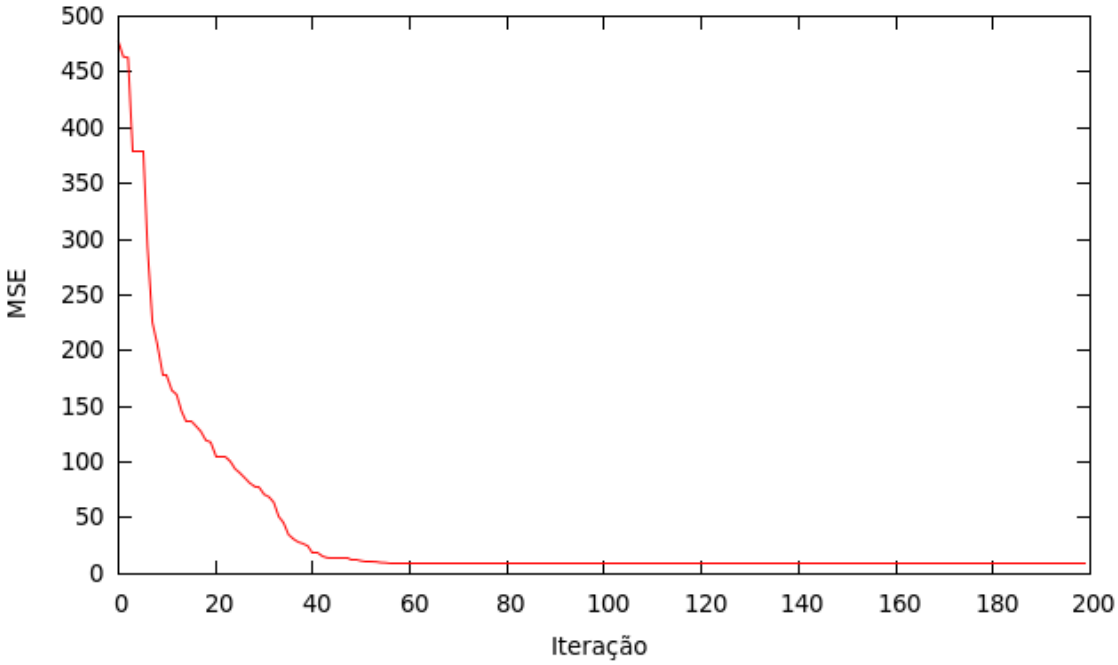


Figura 31: Evolução do erro ao longo da execução do PSO

A Figura 32 apresenta o mapa construído usando os dados de odometria após a integração do *bias*. Como pode ser observado, utilizando somente estes dados foi possível construir um mapa com uma boa qualidade local, mas com erros globais significativos, visíveis pelo erro de fechamento de loop (apresentado em detalhes na Figura 33).

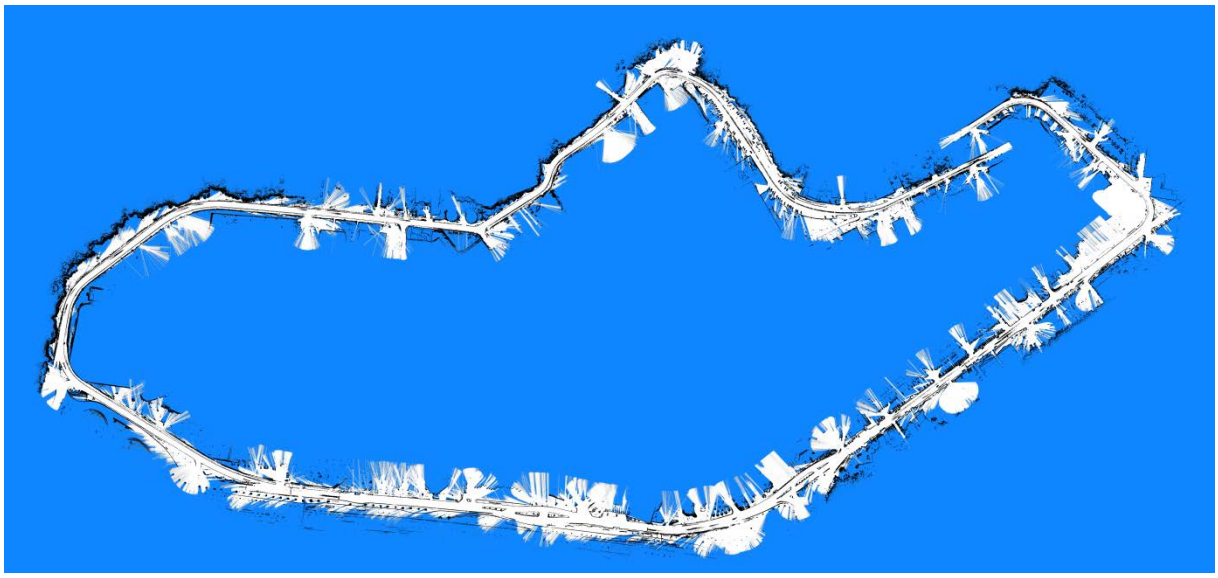


Figura 32: Mapa do anel viário da UFES construído usando apenas a odometria

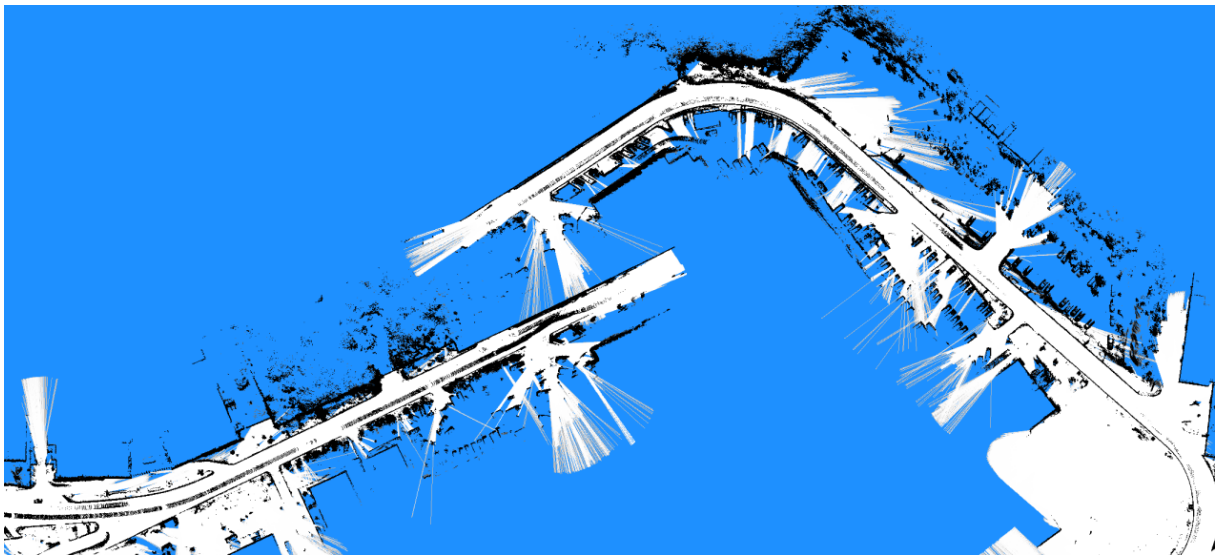


Figura 33: Visualização do problema de fechamento de loop no mapa criado usando apenas a odometria do veículo

Após a calibração da odometria, foi realizada a otimização das poses usando o GraphSLAM. A Figura 34 apresenta uma comparação entre as poses medidas pelo GPS e as poses obtidas pelo GraphSLAM. Como pode ser visto, as poses calculadas pelo GraphSLAM foram consistentes com as medidas do GPS na maior parte do trajeto, mas aconteceram desvios, principalmente nos extremos laterais do

percurso. A Figura 35 apresenta o mapa de obstáculos construído usando o GraphSLAM do anel viário da UFES.

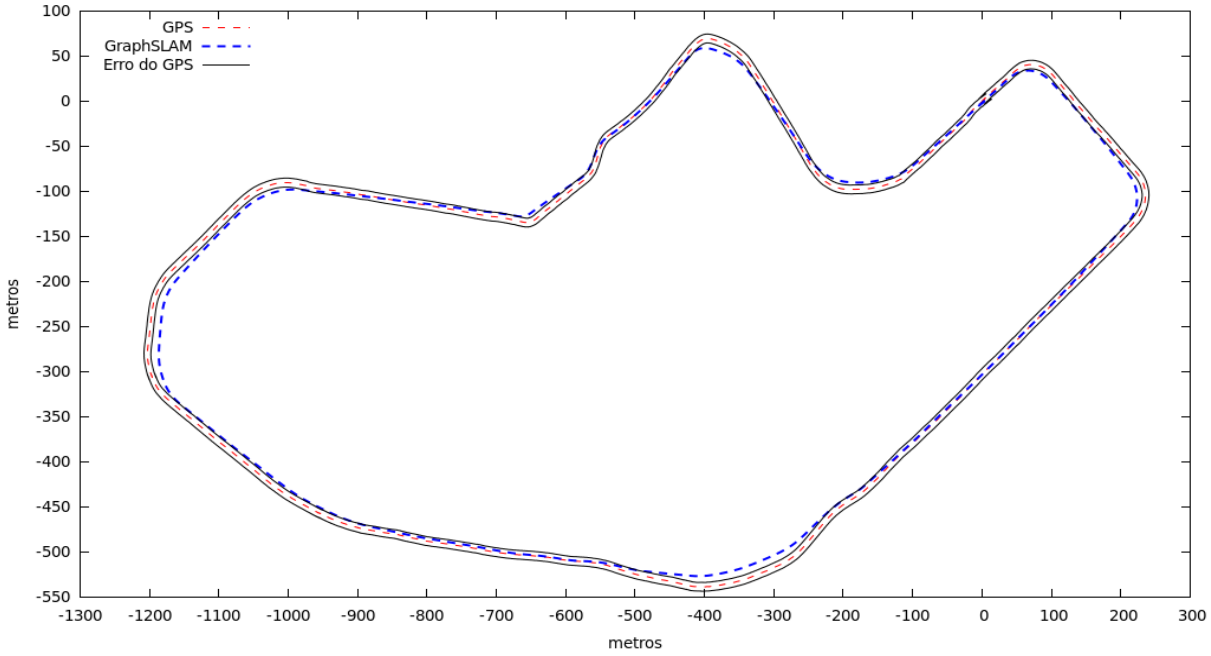


Figura 34: Comparação entre o caminho medido pelo GPS e caminho calculado pelo GraphSLAM

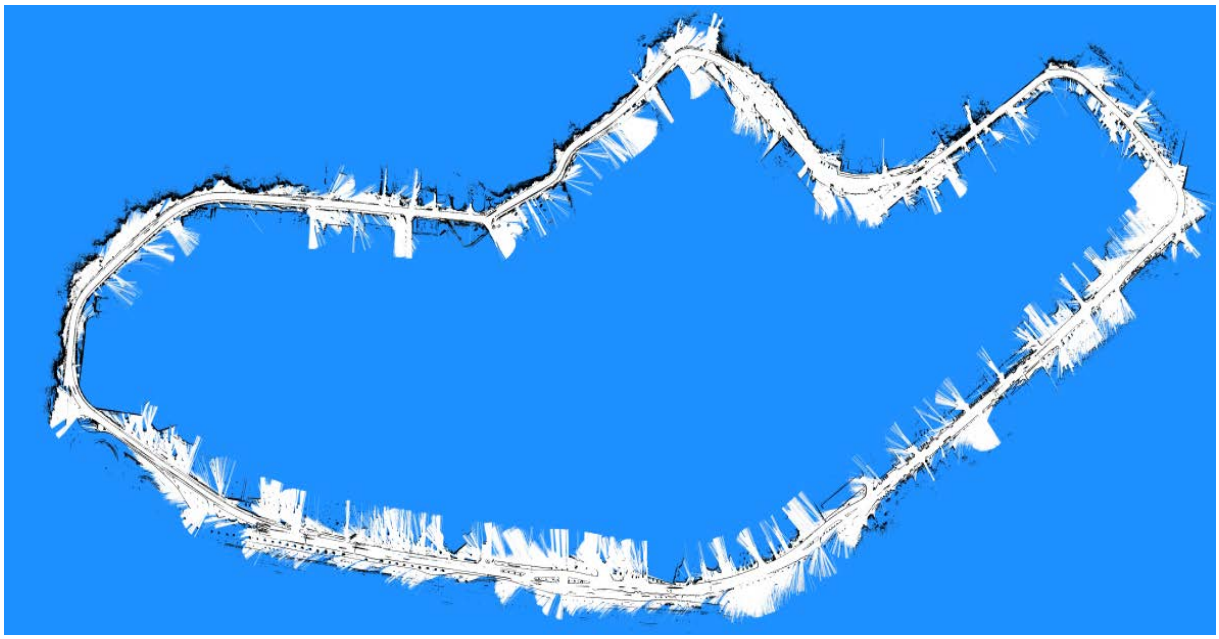


Figura 35: *Grid* de ocupação do anel viário da UFES construído usando o LEMS

A Figura 36 traz em destaque a sequência de construção do mapa durante o fechamento de loop. A ordem temporal das figuras é da esquerda para direita e de cima para baixo. Para permitir ao leitor uma visualização do processo de construção do mapa, a Figura 37 traz uma sequência de imagens capturadas pela câmera e os respectivos estados do mapa. Pode ser observado que as regiões proeminentes existentes nas imagens (como meios-fios e canteiros) de fato foram marcadas como obstáculos no mapa.

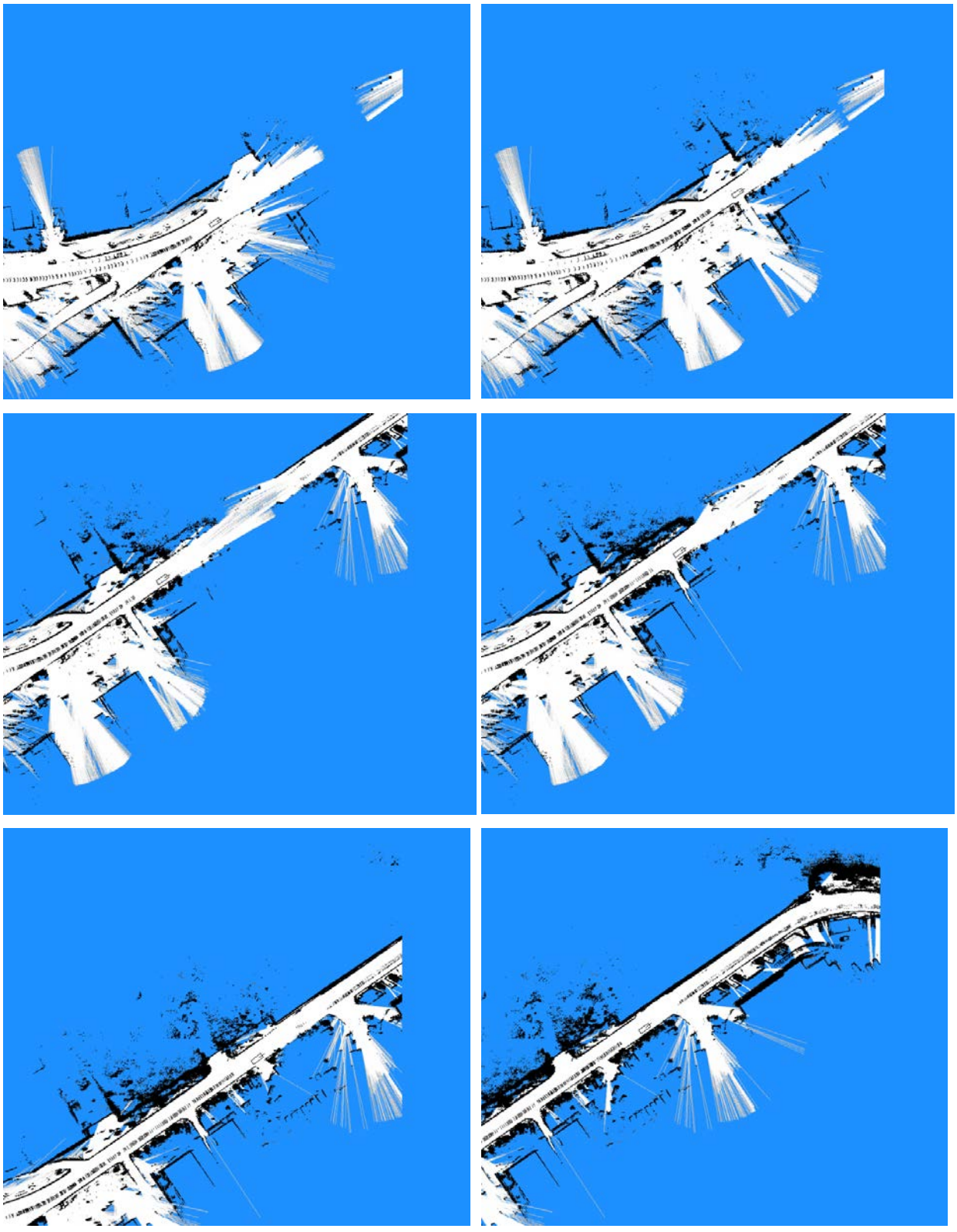


Figura 36: Trecho de fechamento de loop no anel viário da UFES

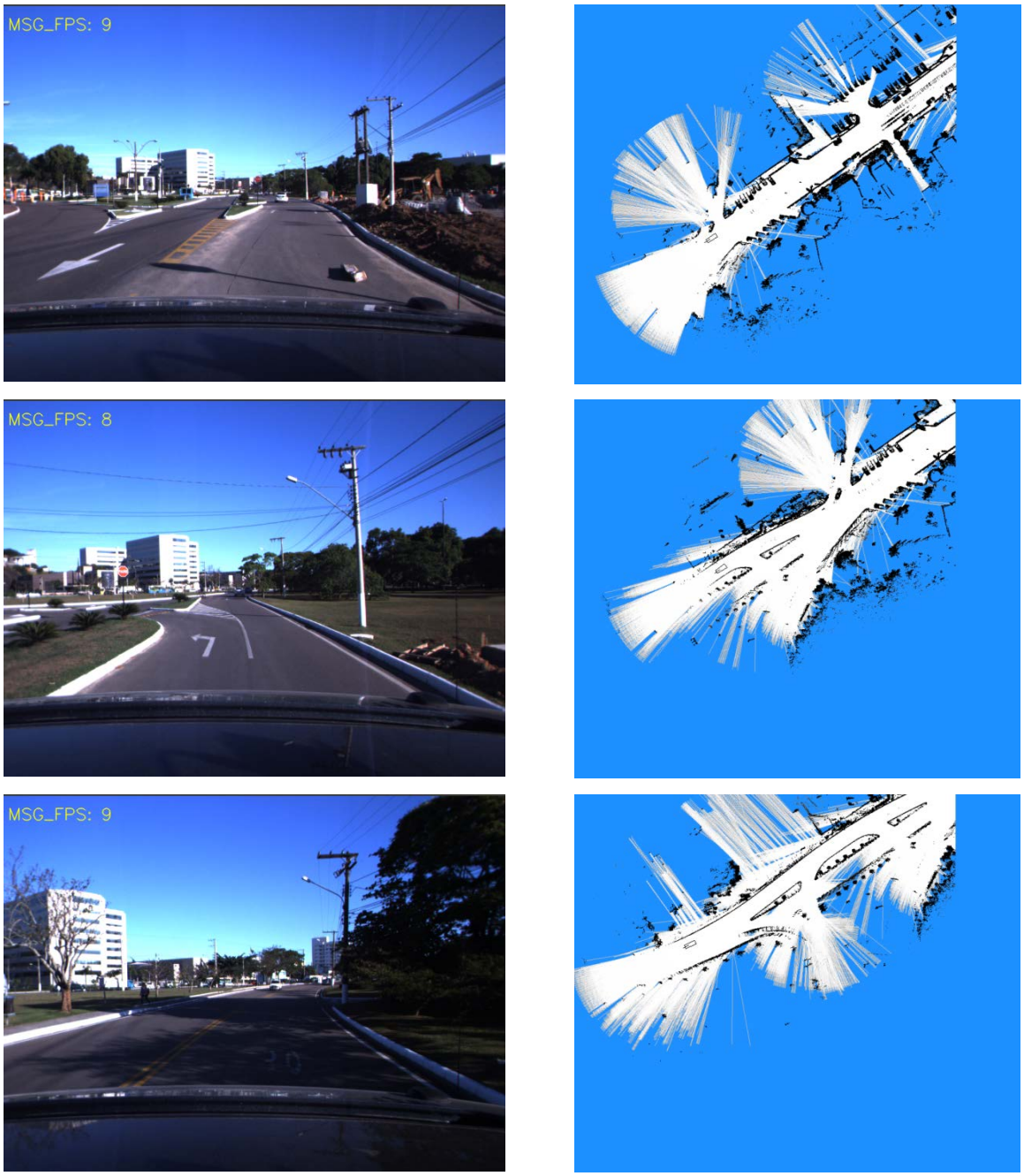


Figura 37: Exemplo de construção do mapa

6.2.3 Mapeamento de um Bairro

O último experimento realizado foi o mapeamento de um bairro da cidade de Vitória, capital do estado do Espírito Santo (ES), Brasil. Um motorista humano guiou a IARA por um trecho urbano de aproximadamente 6.5 quilômetros. A Figura 38 traz uma ilustração do percurso realizado. O trajeto se iniciou e terminou na UFES e, ao longo do trajeto, dois trechos de fechamentos de loop foram identificados.

A Figura 39 apresenta o percurso medido usando *dead-reckoning* e o percurso medido pelo GPS antes da calibração da odometria e a Figura 40 apresenta estes percursos após a calibração. Assim como nos experimentos anteriores, a primeira posição medida pelo GPS foi adotada como sistema de referência para os dados de GPS. Como no anel viário da UFES, a calibração da odometria contribuiu para a estimação de um *dead-reckoning* com uma qualidade significativamente melhor que a inicial. Entretanto, pode ser observado que o *dead-reckoning* ainda possui um importante erro global que precisa ser corrigido (visível pelo erro de fechamento de loop).

Figura 38: Percurso realizado para o mapeamento de um bairro de Vitória (ES)

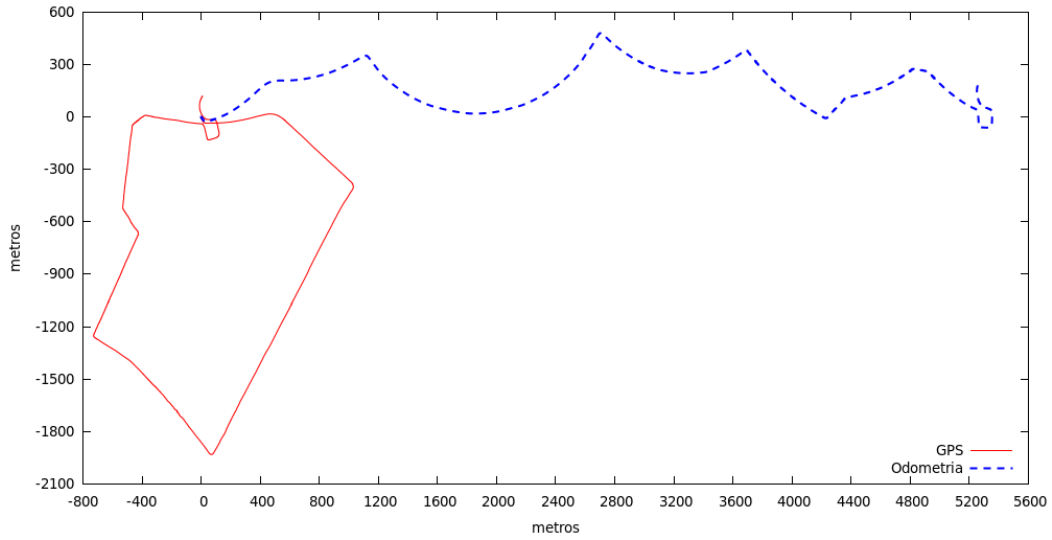


Figura 39: Visualização dos percursos medidos pelo *dead-reckoning* e pelo GPS antes da calibração do sensor de odometria

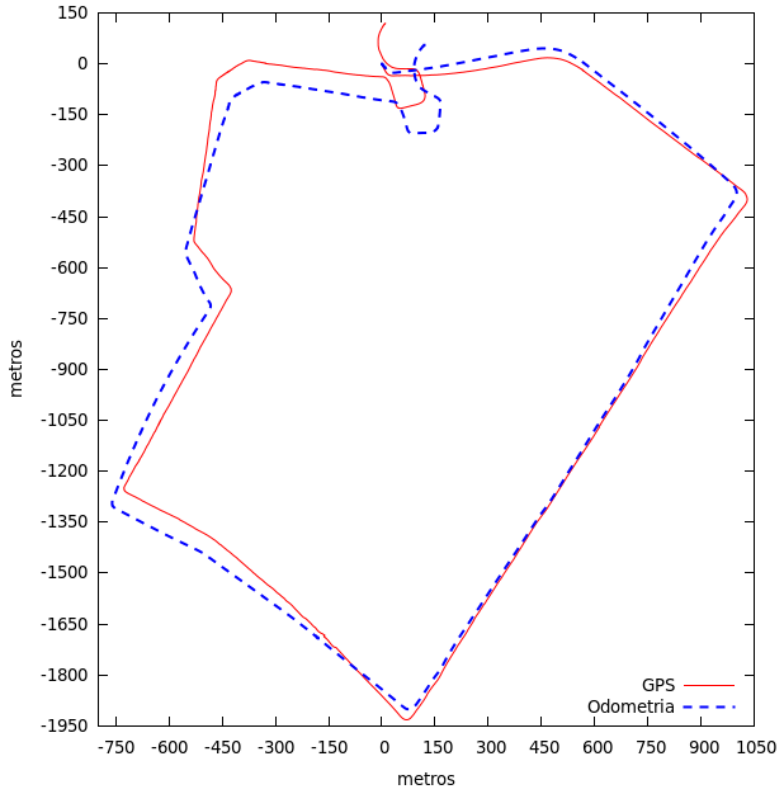


Figura 40: Visualização dos percursos medidos pelo *dead-reckoning* e pelo GPS após a calibração do sensor de odometria

A Figura 41 apresenta a evolução do erro ao longo das iterações do algoritmo de calibração do bias usando PSO e a Tabela 5 apresenta os valores obtidos para os parâmetros. O MSE ao final da otimização foi de 42.56 metros ao quadrado (tirando a raiz tem-se 6.52 metros).

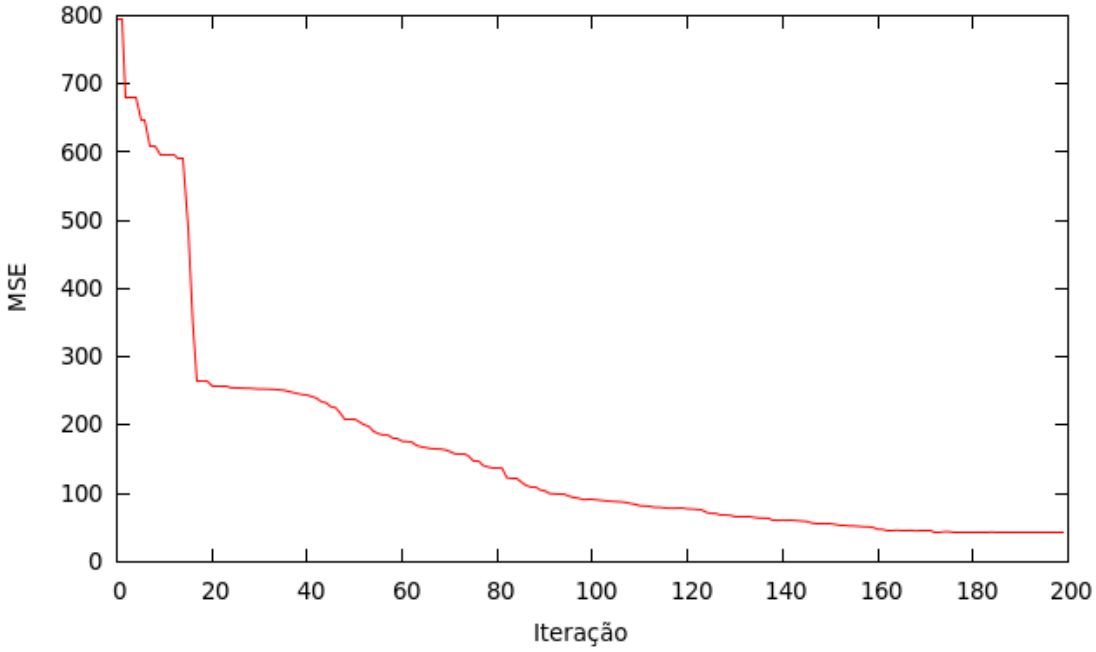


Figura 41: Evolução do erro ao longo da execução do PSO

Tabela 5: Parâmetros de calibração da odometria

Parâmetro	**Valor**
Bias Multiplicativo da Velocidade (m/s)	0.967779
Bias Aditivo do Ângulo do Volante (rad)	-0.002738
Bias Multiplicativo do Ângulo do Volante (rad)	0.996352
Ângulo Inicial (rad)	-0.870175

A Figura 42 apresenta o mapa construído usando apenas o *dead-reckoning* com a odometria calibrada. Assim como no anel viário da UFES, o mapa construído apresentou uma boa qualidade local, mas era globalmente inconsistente. A Figura 43 traz em destaque a região onde deveria existir um fechamento de loop. Devido à esta inconsistência global, os dados de outros sensores e o algoritmo GraphSLAM precisaram ser usados para encontrar os valores das poses com maior precisão.

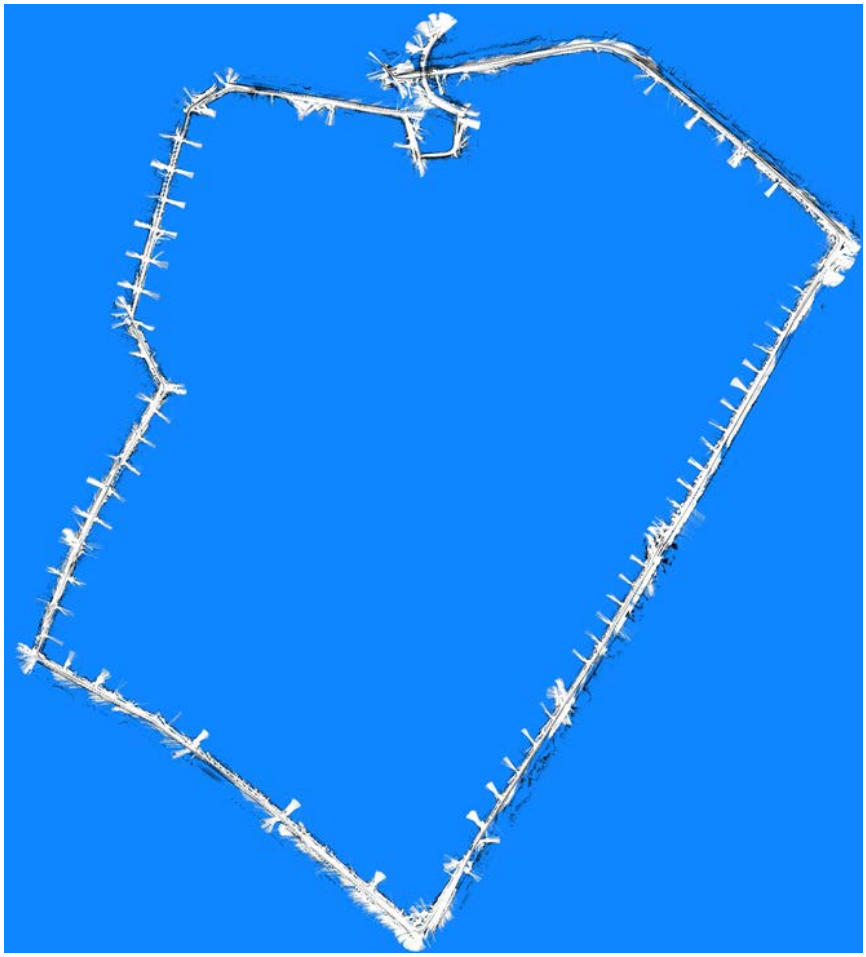


Figura 42: Mapa de um bairro usando apenas a odometria do veículo

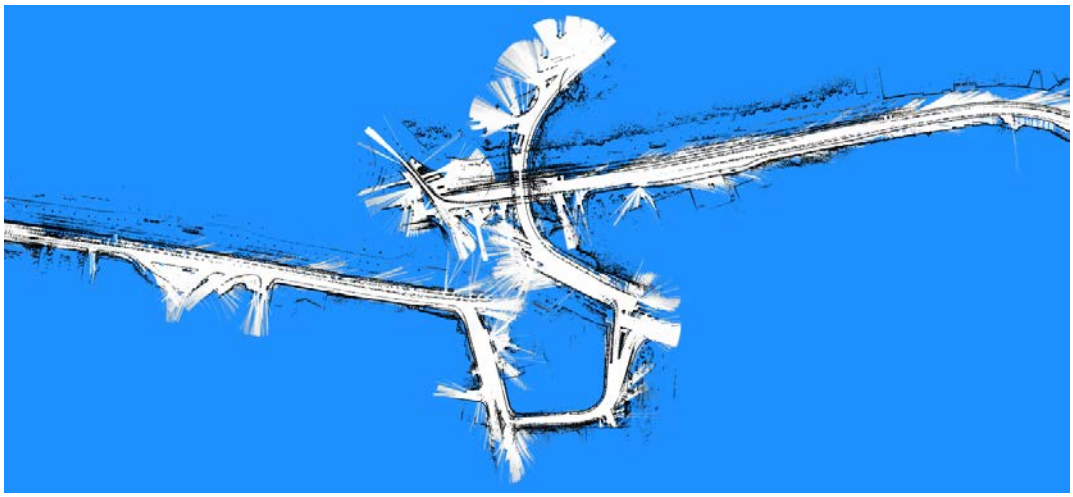


Figura 43: Destaque do fechamento de loop incorreto no mapa de um bairro construído usando apenas odometria

A Figura 44 traz uma comparação entre as poses medidas pelo GPS e as poses calculadas usando o algoritmo GraphSLAM. Como pode ser observado, as poses calculadas pelo GraphSLAM se mantiveram próximas àquelas medidas pelo GPS ao longo de grande parte do percurso. Ligeiros distanciamentos aconteceram apenas na região à extrema direita e na região inferior. A Figura 45 enfatiza a região de fechamento de loop do trajeto, enquanto a Figura 46 traz uma ênfase na região em que as poses do GraphSLAM mais se afastaram das poses do GPS. A Figura 47 apresenta o *grid* de ocupação construído usando as poses otimizadas pelo GraphSLAM. A Figura 48 e a Figura 49 trazem em uma sequência os estados do mapa durante os dois trechos de fechamento de loop. Por ser uma região urbana com grande tráfego de automóveis, várias células do mapa foram marcadas como ocupadas devido à presença de obstáculos móveis. Estas regiões tornam o mapa ruidoso e precisariam ser marcadas manualmente como livres *a posteriori*.

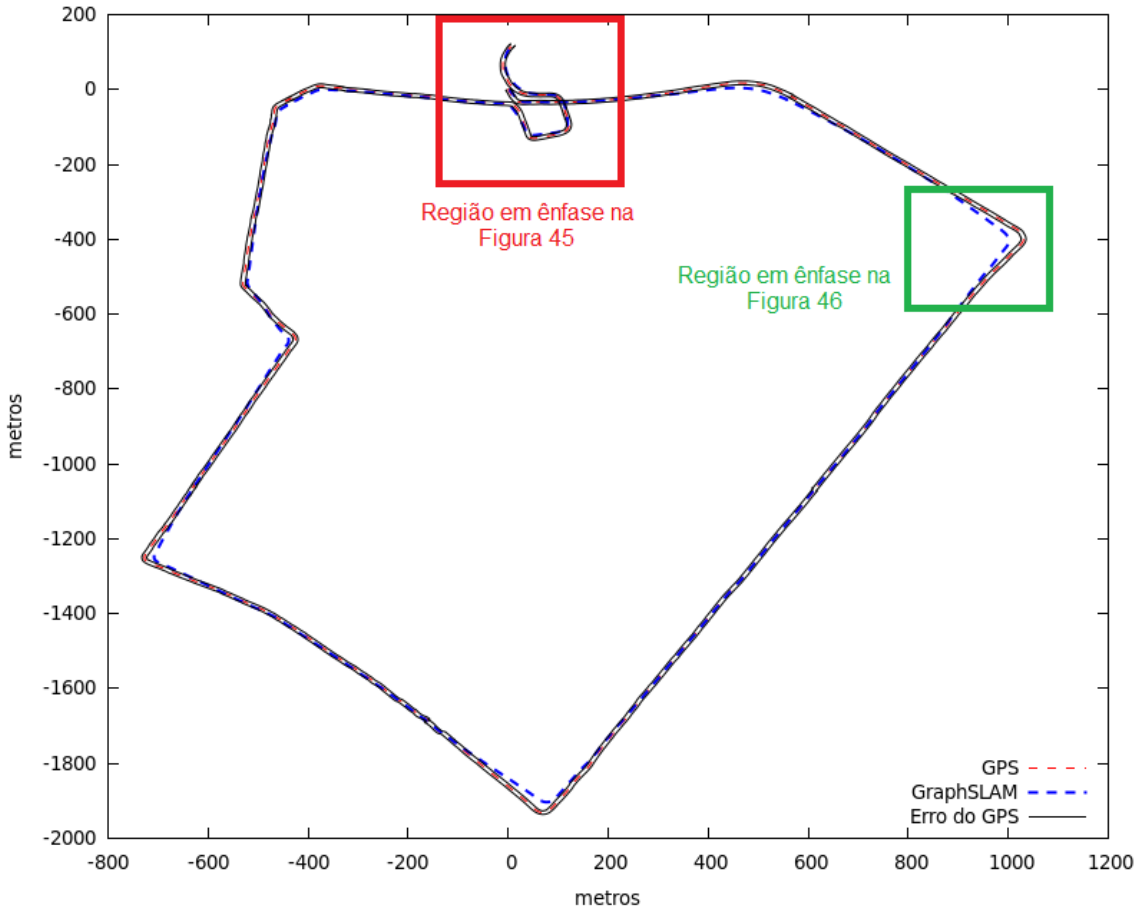


Figura 44: Comparação entre o caminho medido pelo GPS e caminho calculado pelo GraphSLAM

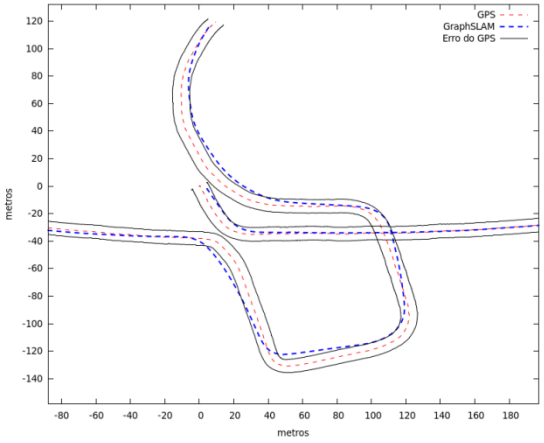


Figura 45: Ênfase na região de fechamento de loop.

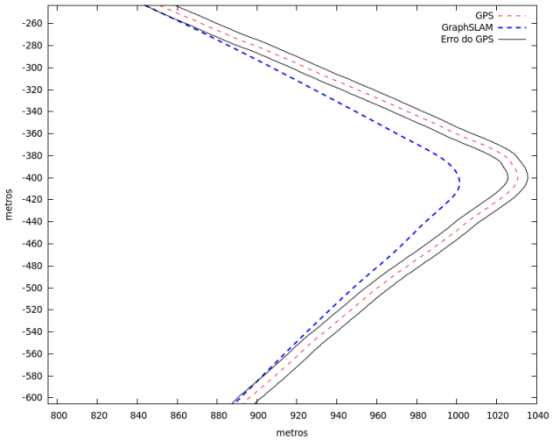


Figura 46: Ênfase em uma região crítica em que o GraphSLAM se afastou do GPS

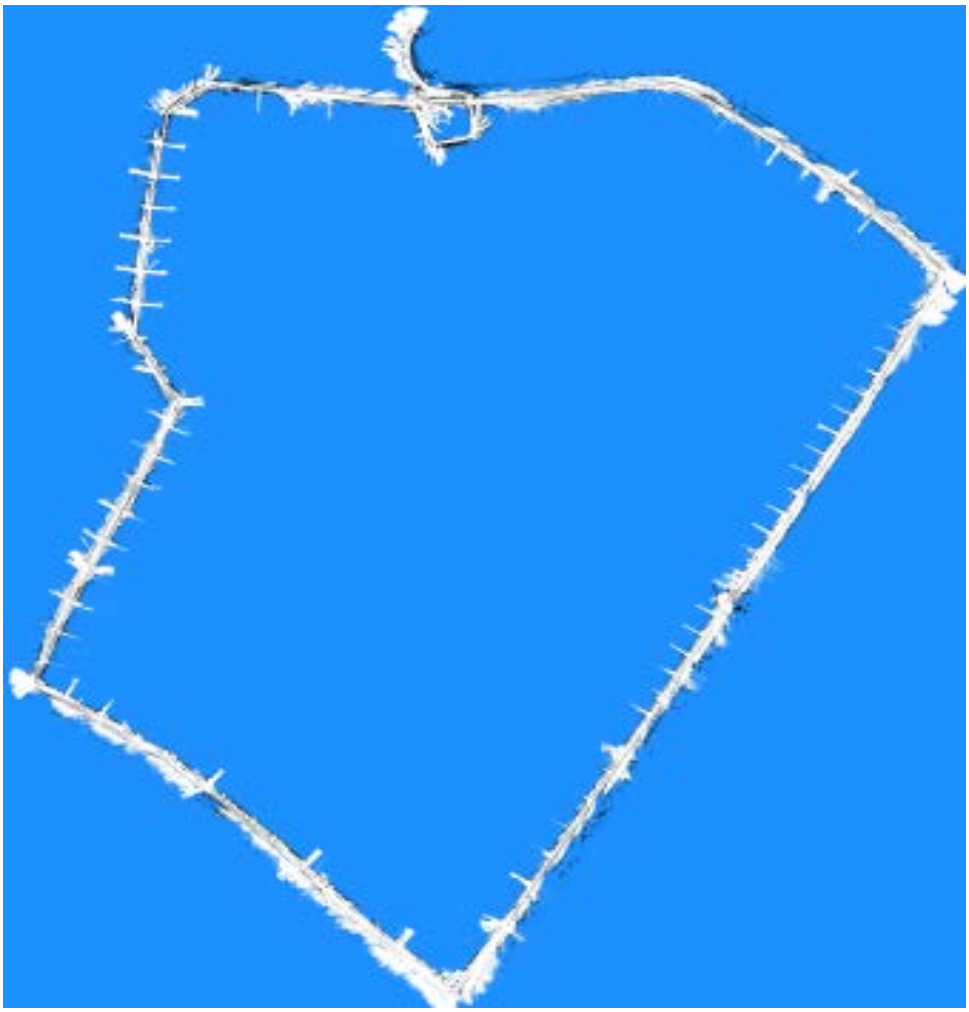


Figura 47: *Grid* de ocupação de um bairro de Vitória, ES, Brasil construído usando o LEMS

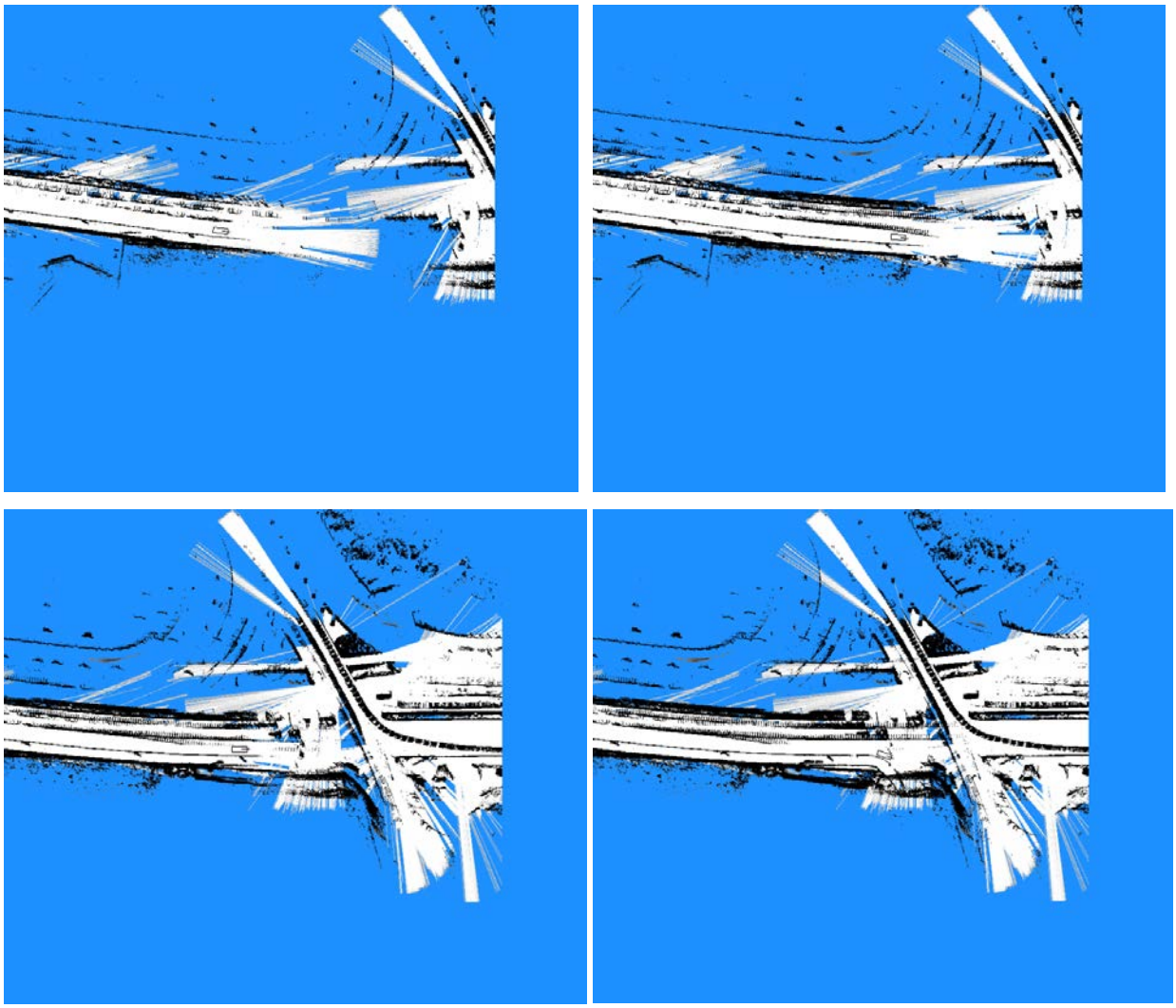


Figura 48: Fechamento de loop no mapeamento de um bairro após a otimização usando GraphSLAM

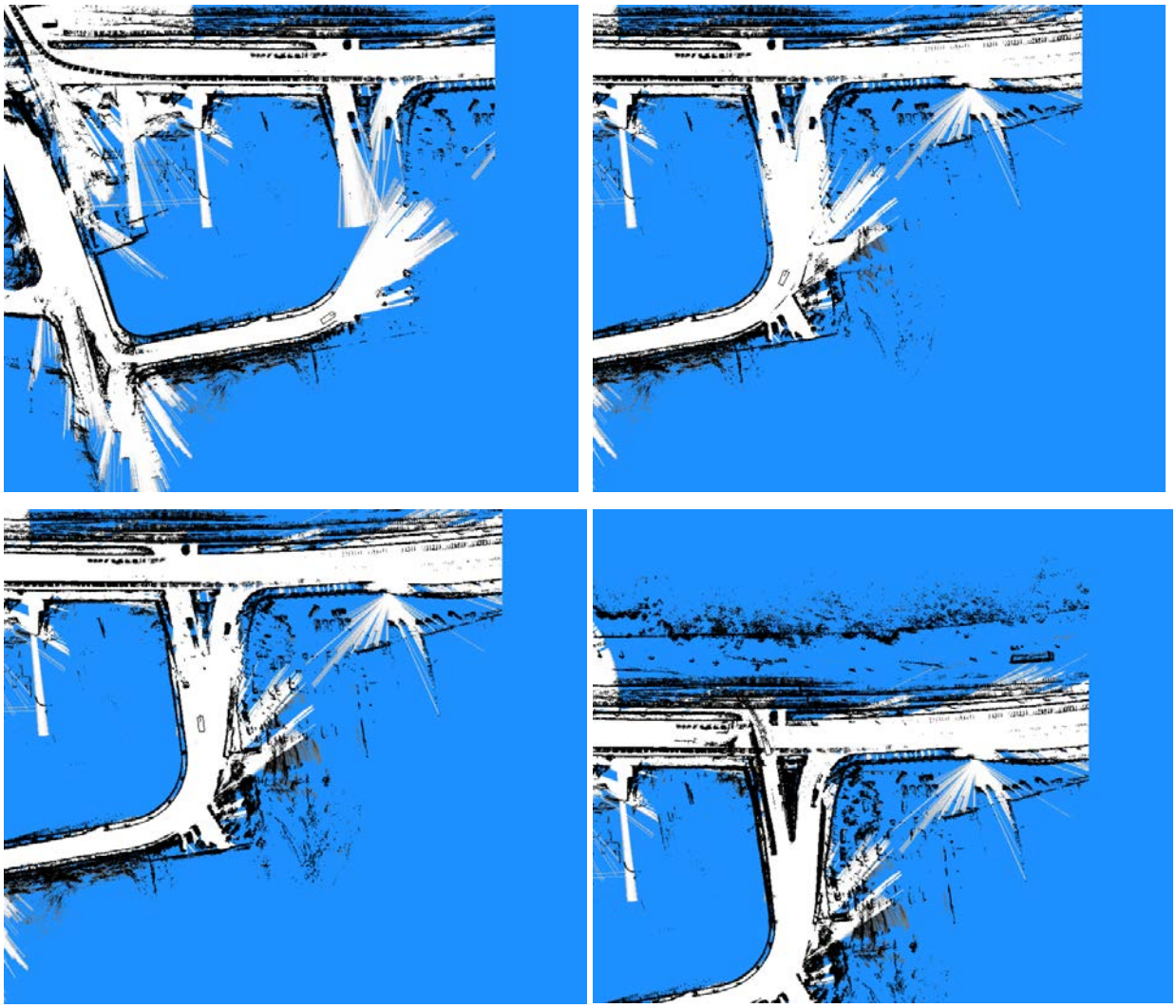


Figura 49: Segundo fechamento de loop no mapeamento de um bairro após a otimização das poses usando o GraphSLAM

7 DISCUSSÃO

As principais contribuições deste trabalho foram a criação de uma nova forma de calcular o *bias* existente nos dados de odometria e o desenvolvimento de um sistema de mapeamento capaz de mapear grandes regiões.

7.1 Análise Crítica

Embora os métodos probabilísticos tradicionais aplicados na solução do problema de SLAM frequentemente assumam que os erros dos sensores são gaussianos, a presença de *bias* e não-linearidades pode causar efeitos desastrosos, como inconsistências no mapa e a divergência do processo de estimação. Tendo isto em mente, pode-se afirmar que o método proposto para a calibração da odometria foi um desenvolvimento fundamental para o sucesso deste trabalho. Os experimentos realizados mostraram que, com a integração do *bias* à odometria, o *dead-reckoning* foi capaz de aproximar com boa precisão o trajeto medido pelo GPS.

O sistema de mapeamento de grandes regiões (*Large-scale Environment Mapping System* – LEMS) desenvolvido neste trabalho pôde ser usado com sucesso para o mapeamento de grandes regiões. Entretanto, a sua forte dependência dos dados de GPS prejudica a sua utilização em ambientes de pequena escala e em regiões em que o sinal de GPS é degradado (florestas, regiões periféricas, etc.). Em ambientes de pequena escala, algoritmos de *matching* (como o GICP) podem ser usados para minimizar os erros de GPS. O uso de algoritmos de *matching* em ambientes de grande escala, entretanto, não é recomendável visto que eles acumulam erros de forma ilimitada. Em regiões em que os dados de GPS são degradados, o resultado do GraphSLAM seria tão bom quanto o *dead-reckoning* do veículo. Para alcançar resultados melhores, outras técnicas de mapeamento teriam que ser utilizadas.

Embora os resultados deste trabalho tenham sido considerados satisfatórios, ainda existem pontos a serem melhorados no LEMS. Entre estes, os pontos que merecem destaque são:

- O algoritmo GICP utiliza a posição de *features* existentes nas nuvens de pontos para calcular a transformada capaz de encaixá-las. A associação incorreta das *features* pode gerar uma estimativa de movimento incorreta e com um erro não-gaussiano. Usualmente, a solução adotada para este problema é a utilização de covariâncias supervalorizadas para que, mesmo que aconteçam erros “caóticos”, eles estejam no raio dado pela covariância. Esta solução, entretanto, tem como efeito colateral a redução do peso das medidas corretas realizadas pelo GICP, devido ao aumento da incerteza. Uma solução mais correta, do ponto de vista teórico, seria a modelagem do erro do GICP usando uma distribuição (talvez não-gaussiana) que, de fato, descrevesse o seu comportamento.
- A etapa com maior custo computacional do LEMS é a execução do GICP para cada pose de fechamento de loop. Entretanto, vale notar que as poses de fechamento de loop são independentes umas das outras e podem ser calculadas em paralelo. O desenvolvimento de um programa capaz de executar o GICP paralelamente para todas as poses de fechamento de loop teria o potencial de tornar o LEMS significativamente mais eficiente.
- Devido à grande dependência dos dados de GPS, o GraphSLAM pode apresentar resultados não satisfatórios quando utilizado para o mapeamento de regiões de pequena escala (nos quais os erros de GPS são mais significativos) e/ou de regiões em que o sinal de GPS é precário. Uma alternativa nestas situações seria a adição de dados de mais sensores com objetivo de, estatisticamente, reduzir a incerteza causada pela falta de dados de GPS ou pela presença de dados com erros excessivamente grandes. Um exemplo de informação extra que poderia ser adicionada ao GraphSLAM seria o uso do algoritmo GICP para calcular o movimento entre duas nuvens de pontos consecutivas do Velodyne.

- Como não foram desenvolvidos tratamentos de objetos móveis, em regiões de tráfego intenso o sistema de mapeamento marcou várias células como ocupadas devido à presença de objetos móveis. Estas células precisaram ser marcadas como livres manualmente por um *expert* humano. Tendo em vista que esta é uma tarefa exaustiva e sujeita a falhas, é importante que sejam desenvolvidas formas automáticas de desconsiderar objetos móveis durante a fase de mapeamento.

8 CONCLUSÃO

Este capítulo apresenta as conclusões e direções para trabalhos futuros. A seção 8.1 apresenta uma síntese deste trabalho, a seção 8.2 traz as conclusões alcançadas e, por fim, a seção 8.3 cita possíveis direções para trabalhos futuros.

8.1 Sumário

O desenvolvimento de mapas de grandes regiões é um tema de pesquisa muito importante para permitir a navegação de automóveis autônomos. A criação de bons mapas é um requisito fundamental para a posterior localização e navegação do veículo de forma segura. A abordagem tradicional para criação de mapas é o uso de modelos probabilístico para Localização e Mapeamento Simultâneos (*Simultaneous Localization and Mapping* – SLAM). Vários algoritmos de SLAM foram propostos, mas poucos são apropriados para o mapeamento de grandes regiões com características heterogêneas.

Nesse trabalho, foi investigado o GraphSLAM baseado em poses na solução do problema de mapeamento para um automóvel autônomo. Neste algoritmo, os vértices representam as poses do robô ao longo de um trajeto e as arestas representam medidas de sensores e os respectivos erros. Para encontrar o conjunto de poses que mais se adaptam às medidas feitas pelos sensores foi usado um otimizador de máxima *likelihood*. Foram realizados experimentos em ambientes com diferentes tamanhos e características. Os resultados mostraram que o sistema desenvolvido é uma boa ferramenta para mapeamento de grandes regiões em que a utilização de GPS é viável.

Além disso, foi apresentado um novo método para estimativa do *bias* existente na odometria do veículo (velocidade e ângulo do volante) usando Otimização por Enxame de Partículas (*Particle Swarm Optimization* – PSO). Experimentos realizados mostraram que, ao integrar o *bias* às medidas de odometria, o *dead-*

reckoning foi capaz de aproximar o caminho percorrido pelo robô com uma precisão significativamente superior ao *dead-reckoning* calculado usando apenas os dados crus dos sensores.

8.2 Conclusões

Neste trabalho, foi desenvolvido um sistema de mapeamento de grandes regiões (*Large-scale Environment Mapping System* – LEMS) que foi integrado à plataforma robótica IARA por meio do *framework* CARMEN. O LEMS se mostrou capaz de criar de forma eficaz mapas de grandes regiões. Para avaliar o LEMS, foram usados conjuntos de dados capturados de regiões com diferentes tamanhos e características. Os experimentos mostraram que o LEMS é um bom algoritmo de mapeamento para regiões de grande porte onde o uso de GPS é viável. Entretanto, o algoritmo se mostrou inapropriado para mapeamento de regiões de pequeno porte, devido à sensibilidade aos erros nas medidas realizadas pelo GPS.

Além do sistema de mapeamento, neste trabalho foi desenvolvido uma nova forma de encontrar o *bias* existente na odometria usando PSO. Os experimentos mostraram que, após a calibração, houve uma melhoria significativa na estimativa do caminho realizado pelo veículo. Ainda existe o que ser melhorado, mas os resultados obtidos podem ser considerados satisfatórios.

8.3 Trabalhos Futuros

Uma direção para trabalhos futuros seria o estudo de novas formas de modelar os erros do algoritmo GICP, uma vez que as suas transformadas frequentemente não possuem erros gaussianos. Para possibilitar o uso do GICP em aplicações que assumem erros gaussianos, usualmente, são utilizadas covariâncias supervalorizadas. Entretanto, esta solução tem como efeito colateral a redução do peso das medidas corretas realizadas pelo GICP devido à incerteza supervalorizada.

Uma solução mais adequada seria o estudo de novas distribuições de probabilidade capazes de modelar de forma mais apropriada o erro existente no GICP.

A inabilidade do LEMS de tratar obstáculos móveis faz com que os mapas de regiões urbanas sejam muito ruidosos e, por essa razão, grandes esforços seriam necessários para realizar a limpeza dos mapas. Para viabilizar a utilização do LEMS para o mapeamento completamente automático de regiões urbanas, deveriam ser desenvolvidos algoritmos para detecção de objetos móveis e marcação das células ocupadas por estes objetos como livres.

Neste trabalho, assumiu-se como hipótese que o tempo é uma grandeza livre de erros, entretanto, devido à arquitetura do sistema, a utilização desta hipótese pode ser uma fonte de erros significativos. Um importante trabalho futuro seria a modelagem do erro existente no tempo, a avaliação de seu impacto e o estudo de como este erro poderia ser adicionado ao modelo probabilístico do GraphSLAM.

Outras direções para trabalhos futuros seriam investigar: a paralelização das execuções do GICP para obter um maior desempenho computacional e o estudo de como usar o LEMS para criar mapas de regiões onde os dados de GPS são imprecisos ou inexistentes (florestas, áreas arborizadas, subsolos, etc.).

9 REFERÊNCIAS BIBLIOGRÁFICAS

1. PEDERSEN, L. **A survey of space robotics**. 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space. 2003.

2. ELFES, A. et al. **Safe and Efficient Robotic Space Exploration with Tele-Supervised Autonomous Robots**. AAAI Spring Symposium: To Boldly Go Where No Human-Robot Team Has Gone Before. 2006.

3. FORLIZZI, J.; DISALVO, C. **Service Robots in the Domestic Environment: a Study of the Roomba Vacuum in the Home.** 1st ACM SIGCHI/SIGART conference on Human-robot interaction. 2006.

4. FIORINI, P.; PRASSLER, E. Cleaning and Household Robots: A Technology Survey. **Autonomous robots**, 2000.

5. SAHIN, H.; GUVENC, L. Household Robotics: Autonomous Devices for Vacuuming and Lawn Mowing. **IEEE Control Systems**, 2007.

6. HICKS II, R. W.; HALL, E. L. Survey of Robot Lawn Mowers. **Intelligent Systems and Smart Manufacturing** , 2000.

7. PAGAC, D.; NEBOT, E. M.; DURRANT-WHYTE, H. An Evidential Approach to Map-building for Autonomous Vehicles. **IEEE Transactions on Robotics and Automation**, 1998.

8. BUEHLER, M.; IAGNEMMA, K.; SINGH, S. (Eds.). **The DARPA Urban Challenge: Autonomous Vehicles in City Traffic**. Springer, 2009.

9. LEVINSON, J.; THRUN, S.; MONTEMERLO, M. Map-Based Precision Vehicle Localization in Urban Environments. **Proceedings of Robotics: Science and Systems**, Atlanta, June 2007.

10. LAHIJANIAN, M. et al. Automatic Deployment of Autonomous Cars in a Robotic Urban-like Environment. **IEEE International Conference on Robotics and Automation**, 2009.

11. BUEHLER, M.; IAGNEMMA, K.; SINGH, S. (Eds.). **The 2005 DARPA Grand Challenge: The Great Robot Race**. Springer, 2007.

12. SMARTER THAN YOU THINK - Google Cars Drive Themselves, in Traffic. **New York Times**, 2010. Disponivel em: <http://www.nytimes.com/2010/10/10/science/10google.html?_r=1&>. Acesso em: 22 Setembro 2013.

13. THE self-driving car logs more miles on new wheels. **Google Official Blog**, 2012. Disponivel em: <http://googleblog.blogspot.hu/2012/08/the-self-driving-car-logs-more-miles-on.html>. Acesso em: 22 Setembro 2013.

14. AUTONOMOS Labs. **AutoNOMOS Labs**. Disponivel em: <http://www.autonomos.inf.fu-berlin.de/>. Acesso em: 22 Setembro 2013.

15. AUTONOMOS Labs. **Autonomous Car Navigates the Streets of Berlin**, 2011. Disponivel em: <http://autonomos.inf.fu-berlin.de/news/press-release-92011>. Acesso em: 22 Setembro 2013.

16. MERCEDES-BENZ mostra carro que anda sozinho. **G1**, 2013. Disponivel em: <http://g1.globo.com/carros/noticia/2013/09/mercedes-benz-mostra-carro-que-anda-sozinho-em-previa-de-frankfurt.html>. Acesso em: 22 Setembro 2013.

17. GIZMAG takes a ride in Volvo's most autonomous car yet. **Gizmag**, 2013. Disponivel em: <http://www.gizmag.com/volvo-autonomous-cars/28161/>. Acesso em: 22 Setembro 2013.

18. NISSAN Says: Leave the Driving to Us. **IEE SPECTRUM**, 2013. Disponivel em: <http://spectrum.ieee.org/tech-talk/green-tech/advanced-cars/nissans-ghosn-

says-leave-the-driving-to-us>. Acesso em: 22 Setembro 2013.

19. DURRANT-WHYTE, H.; BAILEY, T. Simultaneous Localisation and Mapping (SLAM): Part I: The Essential Algorithms. **IEEE ROBOTICS AND AUTOMATION MAGAZINE**, San Francisco, v. 2, p. 2006, 2006.

20. WEINGARTEN, J.; SIEGWART, R. **EKF-based 3D SLAM for Structured Environment Reconstruction**. IEEE/RSJ International Conference on Intelligent Robots and Systems. 2005.

21. WILLIAMS, S. B. **Efficient Solutions to Autonomous Mapping and Navigation Problems**. The University of Sydney. 2001.

22. THRUN, S.; MONTEMERLO, M. The Graph SLAM Algorithm with Applications to Large-scale Mapping of Urban Structures. **The International Journal of Robotics Research**, 2006. 403-429.

23. DISSANAYAKE, G. et al. Map Management for Efficient Simultaneous Localization and Mapping (SLAM). **Autonomous Robots**, Hingham, 1 Maio 2002. 267-286.

24. DUCKETT, T.; MARSLAND, S.; SHAPIRO, J. **Learning Globally Consistent Maps by Relaxation**. IEEE International Conference on In Robotics and Automation. 2000.

25. THRUN, S.; BURGARD, W.; FOX, D. **Probabilistic Robotics**. Cambridge, Massachusetts: MIT Press, 2005.

26. FELLER, W. **An Introduction to Probability Theory and its Applications**. John Wiley & Sons., v. 2, 2008.

27. KÜMMERLE, R. **State Estimation and Optimization for Mobile Robot Navigation**. Universität Freiburg. 2013.

28. PERERA, L. D. et al. **Sensor Bias Correction in Simultaneous Localization and Mapping**. International Conference on Information Fusion. 2003. p. 817-824.

29. KUMMERLE, R.; GRISETTI, G.; BURGARD, W. **Simultaneous Calibration, Localization, and Mapping**. IEEE/RSJ International Conference in Intelligent Robots and Systems. 2011. p. 3716-3721.

30. LEVINSON, J. S. **Automatic Laser Calibration, Mapping, and Localization for Autonomous Vehicles**. Stanford University. 2011.

31. BAILEY, T. **Mobile Robot Localisation and Mapping in Extensive Outdoor Environments**. The University of Sydney. Sydney, p. 1-212. 2002.

32. CLARK, S.; DURRANT-WHYTE, H. **Autonomous Land Vehicle Navigation using Millimeter Wave Radar**. IEEE International Conference in Robotics and Automation. IEEE. 1998. p. 3697-3702.

33. DURRANT-WHYTE, H. Uncertain Geometry in Robotics. **IEEE Journal of Robotics and Automation**, 1988. 23 - 31.

34. SMITH, R. C.; CHEESEMAN, P. On the Representation of Spatial Uncertainty. **International Journal of Robotics Research**, 1987. 56-68.

35. SMITH, R.; SELF, M.; CHEESEMAN, P. Estimating uncertain spatial relationships in robotics. In: COX, I. J.; WILFONG, G. T. **Autonomous Robot Vehicles**. Springer-Verlag New York, Inc., 1990. p. 167-193.

36. SMITH, R.; SELF, M.; CHEESEMAN, P. **Estimating Uncertain Spatial Relationships in Robotics**. International Conference on Robotics and Automation. 1987. p. 850.

37. WAN, E. A.; VAN DER MERWE, R. **The Unscented Kalman filter for Nonlinear Estimation**. IEEE Symposium in Adaptive Systems for Signal Processing, Communications, and Control. 2000. p. 153-158.

38. SIBLEY, G.; SUKHATME, G.; MATTHIES, L. The Iterated Sigma Point Kalman Filter with Applications to Long Range Stereo. **Robotics: Science and Systems**, v. 8, p. 235-244, 2006.

39. DISSANAYAKE, M. W. M. G. et al. A Solution to the Simultaneous Localization and Map Building (SLAM) Problem. **IEEE Transactions on Robotics and Automation**, Jun 2001. 229 - 241.

40. MAYBECK, P. S. **Stochastic Models, Estimation, and Control**. v. 141, 1979.

41. CHEEIN, F. A. A. et al. SLAM Algorithm Applied to Robotics Assistance for Navigation in Unknown Environments. **Journal of neuroengineering and rehabilitation**, 1 February 2010.

42. CHEEIN, F. A. et al. Optimized EIF-SLAM Algorithm for Precision Agriculture Mapping based on Stems Detection. **Computers and Electronics in Agriculture**, 1 Setembro 2011. 195-207.

43. MALLIOS, A. et al. **EKF-SLAM for AUV Navigation under Probabilistic Sonar Scan-matching**. IEEE/RSJ International Conference on Intelligent Robots and Systems. 2010.

44. NEIRA, J.; TARDOS, J. D. Data Association in Stochastic Mapping using the Joint Compatibility Test. **IEEE Transactions on Robotics and Automation**, December 2001. 890 - 897.

45. JULIER, S. J.; UHLMANN, J. K. **A Counter Example to the Theory of Simultaneous Localization and Map Building**. IEEE International Conference on Robotics and Automation. 2001. p. 4238-4243.

46. THRUN, S. et al. Simultaneous Localization and Mapping with Sparse Extended Information Filters. **The International Journal of Robotics Research**, 2004.

47. MONTEMERLO, M. et al. **FastSLAM: A Factored Solution to the**

Simultaneous Localization and Mapping Problem. Proceedings of the AAAI National Conference on Artificial Intelligence. 2002.

48. MURPHY, K. **Bayesian Map Learning in Dynamic Environments**. Advances in Neural Information Processing Systems. 1999. p. 1015--1021.

49. THRUN, S.; BURGARD, W.; FOX, D. **A Real-time Algorithm for Mobile Robot Mapping with Applications to Multi-robot and 3D Mapping**. IEEE International Conference on Robotics and Automation. 2000. p. 321 - 328.

50. GUIVANT, J. E.; NEBOT, E. M. Optimization of the Simultaneous Localization and Map-building Algorithm for Real-time Implementation. **IEEE Transactions on Robotics and Automation**, 2001.

51. LEONARD, J. J.; FEDER, H. J. S. **A Computationally Efficient Method for Large-scale Concurrent Mapping and Localization**. International Symposium in Robotics Research. 2000.

52. NEWMAN, P. **On the Structure and Solution of the Simultaneous Localisation and Map Building Problem**. University of Sydney. 1999.

53. BAILEY, T.; NIETO, J.; NEBOT, E. **Consistency of the FastSLAM Algorithm**. IEEE International Conference In Robotics and Automation. 2006.

54. LU, F.; MILIOS, E. Globally consistent range scan alignment for environment mapping. **Autonomous Robots**, 1997.

55. GOLFARELLI, M.; MAIO, D.; RIZZI, S. **Elastic Correction of Dead-reckoning Errors in Map Building**. IEEE/RSJ International Conference In Intelligent Robots and Systems. 1998.

56. FRESE, U.; HIRZINGER, G. **Simultaneous Localization and Mapping - a Discussion**. IJCAI Workshop on Reasoning with Uncertainty in Robotics. 2001.

57. KONOLIGE, K. **Large-scale Map-making**. National Conference on Artificial Intelligence. MIT Press. 1999.

58. ELFES, A. Using occupancy grids for mobile robot perception and navigation. **Computer**, 1989. 46-57.

59. LEVINSON, J.; THRUN, S. **Robust Vehicle Localization in Urban Environments using Probabilistic Maps**. IEEE International Conference In Robotics and Automation. 2010.

60. THRUN, S.; BÜCKEN, A. **Integrating Grid-based and Topological Maps for Mobile Robot Navigation**. National Conference on Artificial Intelligence. 1996. p. 944-951.

61. MURRAY, D.; LITTLE, J. J. **Using Real-time Stereo Vision for Mobile Robot Navigation**. Autonomous Robots. 2000. p. 161-171.

62. ESTRADA, C.; NEIRA, J.; TARDÓS, J. D. Hierarchical SLAM: Real-time accurate mapping of large environments. **IEEE Transactions on Robotics**, 2005. 588-596.

63. LEONARD, J.; NEWMAN, P. **Consistent, convergent, and constant-time SLAM**. International Joint Conferences on Artificial Intelligence. 2003.

64. SEGAL, A.; HAEHNEL, D.; THRUN, S. Generalized-ICP. **Robotics: Science and Systems**, 2009.

65. BAUER, J.; SUNDERHAUF, N.; PROTZEL, P. **Comparing Several Implementations of Two Recently Published Feature Detectors**. International Conference on Intelligent and Autonomous Systems. Pierre Baudis: Elsevier. 2007. p. 143-148.

66. DIOSI, A.; KLEEMAN, L. **Laser Scan Matching in Polar Coordinates with Application to SLAM**. IEEE/RSJ International Conference In Intelligent Robots

and Systems. IEEE. 2005.

67. BOSSE, M.; ZLOT, R. Map matching and data association for large-scale two-dimensional laser scan-based slam. **The International Journal of Robotics Research**, Salt Lake City, 27 Junho 2008. 667-691.

68. KENNEDY, J.; EBERHART, R. **Particle Swarm Optimization**. IEEE international conference on neural networks. 1995. p. 1942-1948.

69. EBERHART, R. C.; SHI, Y. **Particle Swarm Optimization: Developments, Applications and Resources**. IEEE Congress In Evolutionary Computation. 2001.

70. SUGANTHAN, P. N. **Particle Swarm Optimiser with Neighbourhood Operator**. IEEE Congress In Evolutionary Computation. 1999.

71. KRAMMER, L. **Motion Planning for Car-like Robots**. Vienna University of Technology, Institute of Computer Aided Automation, Automation Systems Group. 2010.

72. **TORC Robotics**. Disponivel em: <http://www.torcrobotics.com/>. Acesso em: 01 jul. 2013.

73. MONTEMERLO, M.; ROY, N.; THRUN, S. **Perspectives on Standardization in Mobile Robot Programming: The Carnegie Mellon Navigation (CARMEN) Toolkit**. International Conference on Intelligent Robots and Systems (IROS). 2003. p. 2436-2441.

74. SIMMONS, R. The inter-process communication (IPC) system. Disponivel em: <http://www.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html>.

75. HOHPE, G.; WOOLF, B. **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions**. 1. ed. Addison-Wesley Professional, 2003.

76. KUMMERLE, R. et al. **g2o: A General Framework for Graph Optimization**. IEEE International Conference In Robotics and Automation. 2011.

77. ENDRES, F. . H. J. et al. **An Evaluation of the RGB-D SLAM System**. IEEE International Conference In Robotics and Automation. : 2012.

78. STRASDAT, H. et al. **Double Window Optimisation for Constant Time Visual SLAM**. IEEE International Conference In Computer Vision. 2011.

79. SUNDERHAUF, N.; PROTZEL, P. **Switchable Constraints for Robust Pose Graph Slam**. IEEE/RSJ International Conference In Intelligent Robots and Systems. 2012.

80. RUSU, R. B.; COUSINS, S. **3d is Here: Point Cloud Library (pcl)**. IEEE International Conference In Robotics and Automation. 2011. p. 1-4.