



Trabalho de Conclusão de Curso

Um Estudo de Caso para Análise e Avaliação do Algoritmo ORB-SLAM

João Victor Torres Borges
borgesjvt@gmail.com

Orientador:
Prof. Dr. Leonardo Viana Pereira

Maceió, Março de 2018

João Victor Torres Borges

Um Estudo de Caso para Análise e Avaliação do Algoritmo ORB-SLAM

Monografia apresentada como requisito parcial para
obtenção do grau de Bacharel em Engenharia de
Computação do Instituto de Computação da Univer-
sidade Federal de Alagoas.

Orientador:

Prof. Dr. Leonardo Viana Pereira

Maceió, Março de 2018

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação do Instituto de Computação da Universidade Federal de Alagoas, aprovada pela comissão examinadora que abaixo assina.

Prof. Dr. Leonardo Viana Pereira - Orientador
Instituto de Computação
Universidade Federal de Alagoas

Prof. Dr. Heitor Soares Ramos Filho - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Dr. Heitor Judiss Savino - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Agradecimentos

Agradeço a Deus, por ter me dado disposição e paciência. Agradeço aos meus pais, também pela paciência e por todo apoio incondicional. Agradeço aos meus primos Matheus Torres e Maurício Torres e aos meus amigos Israel Loureiro e Arthur Barbosa por me ajudarem diretamente em etapas desse TCC. E agradeço aos demais amigos, familiares e professores que em algum momento contribuíram para este trabalho e para minha formação pessoal e profissional.

“Cada sonho que você deixa pra trás, é um pedaço do seu futuro que deixa de existir.”

– , Steve Jobs

“Quando você elimina o impossível, o que restar, não importa o quão improvável, deve ser a verdade.”

– , Sherlock Holmes

*“Atrás do quê os pilotos de Formula 1 estão correndo?”
“Atrás da grana, meu amigo. Atrás da grana.”*

– Nelson Piquet, Tricampeão de Fórmula 1

Resumo

Em robótica móvel e navegação, SLAM (do acrônimo, Simultaneous Localization and Mapping) é o problema de construir o mapa de um ambiente desconhecido ao mesmo tempo em que você se localiza dentro deste mapa. Este problema é considerado fundamental para a navegação de sistemas autônomos e pode ser resolvido através de diversas estratégias combinando diversos sensores e algoritmos para atuar e atender em ambientes e restrições de desempenho variados. Neste trabalho, apresenta-se uma revisão do estado da arte acerca dessas estratégias para o problema do SLAM, com ênfase no SLAM visual monocular.

O objetivo deste trabalho é implementar uma estratégia de SLAM monocular, especificamente, o algoritmo ORB-SLAM, pontuando os desafios práticos e as características do algoritmo através de um estudo de caso. Como resultados deste trabalho temos: primeiro, o próprio estudo de caso, onde analisamos o mapa construído pelo algoritmo e avaliamos o comportamento do robô na realização das tarefas definidas, concluindo por fim como promissores; e segundo, o projeto de um robô móvel desenvolvido paralelamente a fim de dar suporte ao experimento executado no estudo de caso.

O estudo de caso apresentado compõe uma etapa fundamental na execução de um projeto de maior abrangência, onde é pretendida a incorporação de diferentes sensores a serem integrados através de estratégias de fusão de dados e, dessa forma, atingir um comportamento satisfatório em robustez e acurácia, não apenas em ambientes controlados, mas também em cenários de maior complexidade remetentes à navegação.

Palavras-chave: Robótica Móvel, SLAM, Fusão de Dados, SLAM Visual, Navegação

Abstract

In mobile robotics and navigation, SLAM (Simultaneous Localization and Mapping) is the problem of building the map of a unknown environment at the same time as it is located into this map. This problem is considered fundamental for the autonomous navigation systems and can be solved through several strategies by combining a number of sensors and algorithms to act in several environments and comply with a number of performance restrictions. In this work, we present a review of the state-of-the-art regarding these strategies to the SLAM's problem, with emphasis to monocular visual SLAM.

The goal of this work is to implement a monocular SLAM strategy, specifically, the ORB-SLAM algorithm, by pointing out the practical challenges and the algorithm's features through a case of study. As results of this work we have presented: first, the case of study itself, where we analysis the constructed map by the algorithm and evaluate the robot's behavior in achieving the defined tasks, concluding in the end as promising; and second, the project of a mobile robot developed in parallel in order to give support to the experiment executed in the case of study.

The case of study shown here is one fundamental step in the execution of a bigger project scope, where is pretended the incorporation of different sensors to be integrated through data fusion strategies and, in this way, to accomplish a satisfactory behavior in terms of robustness and accuracy, not just for simple and controlled environments but also most complex scenarios assigned to navigation.

Keywords: Mobile Robotics, SLAM, Sensor Fusion, Visual SLAM, Navigation

Conteúdo

Lista de Figuras	vi
Lista de Tabelas	vii
1 Introdução	1
2 Fundamentação Teórica	4
2.1 Elementos do vSLAM	4
2.2 Avaliação de Métodos vSLAM	5
2.3 ORB-SLAM	8
2.3.1 Rastreamento	8
2.3.2 Mapeamento Local	9
2.3.3 Fechamento de Malha	10
3 Metodologia	12
3.1 Arquitetura do Sistema	12
3.2 Ferramentas Utilizadas	13
3.3 Calibração da Câmera	13
3.4 Avaliação do Experimento	14
4 Resultados e Discussões	15
4.1 Plataforma Robótica	15
4.1.1 Projeto Mecânico	15
4.1.2 Projeto Elétrico	15
4.1.3 Projeto de Software	16
4.2 Estudo de Caso	20
5 Considerações Finais	24
Referências bibliográficas	26

Lista de Figuras

2.1	Comparativo do processo de execução entre os métodos direto e baseado em características. [Retirada da apresentação de Jakob Engel, Semi-Dense Direct SLAM]	6
2.2	Comparativo de desempenho e robustez entre os métodos direto e baseado em características. [Retirada da apresentação de Jakob Engel, Semi-Dense Direct SLAM]	7
2.3	Visão Geral do Sistema ORB-SLAM. [Retirada de ORB-SLAM]	8
2.4	Diferença entre a posição real e estimada devido ao erro acumulativo de desvio. [Retirada de http://cogrob.ensta-paristech.fr/loopclosure.html]	11
3.1	Diagrama da arquitetura do sistema	12
3.2	Cenários para execução do experimento	14
4.1	Plataforma robótica	16
4.2	Esboço mecânico das peças	17
4.3	Esboço elétrico dos componentes	17
4.4	Esquemático	17
4.5	Identificação de pontos característicos no frame atual	21
4.6	Mapa e localização gerado pelo ORB-SLAM no cenário 1	22
4.7	Mapa e localização gerado pelo ORB-SLAM no cenário 1 publicado no ambiente ROS	22
4.8	Mapa e localização gerado pelo ORB-SLAM no cenário 2 publicado no ambiente ROS	23
4.9	Snapshot do experimento em tempo real	23

Lista de Tabelas

3.1	Parâmetros de calibração da câmera	13
-----	--	----

1

Introdução

A crescente demanda por soluções de automação e sistemas autônomos em conjunto à ampla oferta de recursos proporcionada pela tecnologia contemporânea sustentam o alto interesse de empresas e pesquisadores nesta área e suas adjacências. Este panorama é causa da destinação de largas somas de investimentos que, dessa forma, possibilitam a investigação de uma diversificada gama de aplicações onde, inserido nesta, encontra-se definido o escopo das pesquisas abordadas neste trabalho.

A robótica e os sistemas autônomos vem transformando diversos setores como manufatura, agricultura, inspeção, exploração, aviação, saúde e entretenimento. Dada a vasta gama de setores e a variabilidade de aplicações, a multinacional Boston Consultant Group ([Wolfgang et al., 2017](#)) classificou estas diversas aplicações nos seguintes grupos: militar, industrial, comercial e bens de consumo, estimando um volume de investimentos de 87 bilhões de dólares até o ano de 2025 no mercado da robótica.

Não existe um consenso universal de quando surgiu a robótica, qual foi o primeiro robô, e nem sobre a definição do que é a robótica; definição essa que já foi reformulada várias vezes e vem evoluindo ao longo da história. O que se sabe hoje, no entanto, é que a robótica e os sistemas autônomos envolvem capacidades de sensoriamento e ação no mundo físico com objetivos a serem cumpridos de forma autônoma ([Mataric, 2007](#)).

A capacidade de coletar informações sensoriais, processá-las e atuar no meio físico a fim de cumprir esses objetivos está fortemente associada ao avanço científico em três grandes áreas; Inteligência Artificial, Teoria de Controle e Visão Computacional, que unidas constituem a base fundamental para o desenvolvimento da robótica. Uma vez que um robô pode ser modelado como um sistema dinâmico não-linear, modelos robóticos tem sido usados como bons estudos de caso para exemplificar conceitos gerais de projeto e análise de teoria de controle, tais como controle robusto e adaptativo, da mesma forma que sistemas robóticos tem se beneficiado dos avanços dos novos algoritmos de controle ([de Wit et al., 1996](#)).

Este benefício mútuo entre a robótica e a teoria de controle se estende para as áreas de inteligência artificial e visão computacional, onde se pode destacar contribuições no âmbito da robótica colaborativa, sistemas multi-agentes e fusão de dados (Murphy, 2000).

Todos esses tópicos subjacentes as linhas de pesquisa citadas estão relacionados aos aspectos técnicos que envolvem desde o sensoriamento dos dados e atuação no meio físico até o processo de tomada de decisão do robô para realizar sua tarefa. Entre estes aspectos técnicos, podemos citar vários desafios mais específicos como o controle de trajetória de um manipulador robótico, o reconhecimento de objetos ou pessoas e os sistemas de navegação e exploração de robôs móveis.

No escopo da robótica móvel, especificamente no problema de navegação, o objetivo é partir de um ponto de origem onde o robô está localizado até um ponto de destino em que se pretende chegar. A fim de atingir este objetivo é necessário responder três perguntas: onde eu estou? onde eu quero chegar? e como eu chego lá? Estas perguntas podem ser traduzidas em sub-problemas categorizados como problemas de localização, detecção e desvio de obstáculos, mapeamento e planejamento de trajetória (Siegwart et al., 2011). Os subproblemas de localização e mapeamento podem ser resolvidos no que se chama de SLAM, e é aqui onde se aplica os esforços do que foi realizado neste trabalho.

SLAM (do acrônimo, Simultaneous Localization and Mapping) é o problema de construir o mapa de um ambiente desconhecido ao mesmo tempo em que você se localiza dentro deste mapa, permitindo assim saber onde você está em relação a um determinado referencial e o que está em sua volta. Este problema é considerado fundamental para a navegação de sistemas autônomos e pode ser resolvido através de diversas estratégias combinando sensores e algoritmos distintos para atuar e atender em ambientes e restrições de desempenho variados.

A fim de discutir sobre a complexidade dessas estratégias e avaliar sua maturidade para a resolução de problemas, Cadena et al. (2016) avaliou o SLAM como uma combinação dos seguintes aspectos:

Robô: caracterizado pelo conjunto de sensores utilizados (e.g., LiDAR, câmeras, ultrassom), tipo de movimento (dinâmica, velocidade máxima) e recursos computacionais (memória, processamento, taxa de amostragem);

Ambiente: o ambiente é estático ou dinâmico, planar ou 3D, entre outras influências como luminosidade, textura e simetria;

Restrições de desempenho: acurácia na estimação da localização e da construção do mapa, latência, tempo máximo de operação;

onde cada configuração de robô/ambiente/restrições apresenta custos, capacidades e limitações específicas. Por exemplo, o mapeamento 2D de um ambiente fechado usando um

robô de duas rodas com um sensor de profundidade pode ser considerado amplamente resolvido, satisfazendo uma acurácia de (<10 cm) ([KUKA Robotics Corporation](#)). Por outro lado, existem outras configurações de robô/ambiente/restrições que precisam de mais estudo.

Dada a importância do SLAM e as diversas possibilidades de configurações, o objetivo deste trabalho é uma revisão bibliográfica aprofundada no SLAM visual monocular, especificamente, o algoritmo ORB-SLAM, pontuando os desafios práticos e as características do algoritmo através de um estudo de caso. A dificuldade técnica do SLAM visual torna a área um grande desafio. Contudo, o uso da visão computacional para resolver o problema do SLAM agrega fatores importantes como baixo custo e a extração de informações 3D, tornando-o um campo popular de pesquisa.

Este trabalho se divide como segue: No Capítulo 2 é discutido o estado da arte dos algoritmos de SLAM visual; Capítulo 3 define a estrutura do estudo de caso e a arquitetura de solução propostas; Capítulo 4 apresenta os resultados obtidos; e, finalmente, o Capítulo 5 apresenta as considerações finais, concluindo este trabalho.

2

Fundamentação Teórica

Em cenários onde são utilizadas câmeras de vídeo como principal fonte para obtenção de dados de entrada, é vantajoso tirar proveito das facilidades de aquisição e utilização deste recurso no desenvolvimento de aplicações SLAM, compondo uma vertente de pesquisa mais específica conhecida na literatura como SLAM Visual (vSLAM) (Taketomi et al., 2017). Este é o foco desenvolvido neste Trabalho de Conclusão.

É importante destacar que os algoritmos de vSLAM são utilizados de forma interdisciplinar em áreas como visão computacional, robótica e realidade aumentada (Taketomi et al., 2017).

Um dos fatores de relevância dentro deste contexto é a dificuldade técnica do vSLAM, normalmente superior ao observado quando são empregados outros tipos sensores devido ao campo limitado de visão da câmera monocular. É possível observar esta diferença em relação ao comportamento de um sonar ou LiDAR, por exemplo.

Por outro lado, a extração de informação 3D, a capacidade de detectar contexto no ambiente (por exemplo, reconhecer faces ou objetos) sem a adição de novos sensores, o uso em aplicações aéreas, jogos, *wearables* e o baixo custo, tornam o vSLAM um campo popular de pesquisa dentre as diversas estratégias de sensores e algoritmos para SLAM.

Neste capítulo abordamos a taxonomia do vSLAM e apresentamos os principais algoritmos encontrados na literatura.

2.1 Elementos do vSLAM

De acordo com Taketomi et al. (2017) existem cinco módulos que podem compor um algoritmo vSLAM dos quais os três primeiros são imprescindíveis a qualquer algoritmo da categoria:

Inicialização: Para iniciar o algoritmo, é preciso definir um sistema de coordenadas para estimar a posição da câmera e a reconstrução do ambiente a ser explorado. Portanto,

na inicialização, o sistema de coordenadas global é definido e uma parte do ambiente é reconstruído como mapa inicial. Em seguida, as etapas de rastreamento e mapeamento são executadas de forma concorrente e durante todo período de execução do algoritmo.

Rastreamento e mapeamento: O mapa reconstruído é rastreado na imagem para estimar a posição da câmera em relação ao mapa por intermédio de correspondências 2D-3D. Este processo se dá a partir da extração de pontos característicos para calcular as respectivas distâncias.

Relocalização: Quando a câmera se move muito rápido ou sofre alguma perturbação, a aplicação se torna propensa à ocorrência de falhas no rastreamento. Nesse caso, é necessário computar a posição da câmera em relação ao mapa novamente e assim recuperar sua localização. Sistemas vSLAM que não possuem relocalização não são confiáveis em cenários práticos, uma vez que o sistema seria interrompido em caso de perda do rastreamento.

Otimização global: Normalmente o mapa acumula um erro de estimação de acordo com o movimento da câmera. A otimização global tem a função de suprimir esse erro levando em conta a consistência do mapa com um todo. A técnica utilizada como referência para eliminar esse erro global é o fechamento de malha, que uma vez identificado, permite estimar o erro acumulado durante o movimento da câmera. Um exemplo de fechamento de malha seria um volta por um quarteirão, onde ao revisitar os pontos já conhecidos (pontos de início do percurso), é possível obter consistência geométrica no mapa do quarteirão de forma global.

2.2 Avaliação de Métodos vSLAM

Os algoritmos de vSLAM são classificados em duas categorias, de acordo com a abordagem adotada pelo algoritmo, que são o método baseado em características e o método direto.

O método baseado em características processa a imagem para encontrar pontos como quinas e bordas (ponto de alta frequência). Há um vasto grupo de algoritmos que podem realizar esta etapa como, por exemplo, ORB, SIFT, SURF ([Lowe \(2004\)](#); [Bay et al. \(2008\)](#); [Rublee et al. \(2011\)](#)) entre outros. Partindo desse princípio, regiões da imagem que contém mais textura (heterogêneas) são mais interessantes do que regiões mais homogêneas. Outra questão relacionada a este método é a armazenagem dos pontos característicos processados, o que pode se tornar altamente custoso. No entanto, desde que esse método elimina todos os outros pontos da imagem que não são relevantes geometricamente, isso o torna mais eficiente em termos de processamento do que o método de algoritmos diretos.

Existem dois tipos de métodos baseados em características: o método baseado em filtros, muito comum em outras categorias de SLAM e nos primeiros algoritmos de SLAM visual (Davison, 2003), onde são utilizadas técnicas como filtro de Kalman (KF), filtro de Kalman Extendido (EKF) e filtro de partículas (PF); e os baseados em Bundle Adjustment (BA), algoritmo que monta um problema de otimização sobre uma estrutura 3D e através da triangulação e refinamento dos pontos característicos detectados descreve uma cena geométrica.

Em contraste com o método baseado em características, o método direto utiliza uma imagem como entrada sem nenhuma abstração. Esse método é baseado no princípio da fotogrametria no qual a consistência fotogramétrica é utilizada como medida de erro em contrapartida com a consistência geométrica que utiliza as distâncias entre pontos característicos. Na figura 2.1 mostramos um comparativo entre o processo de execução dos métodos.

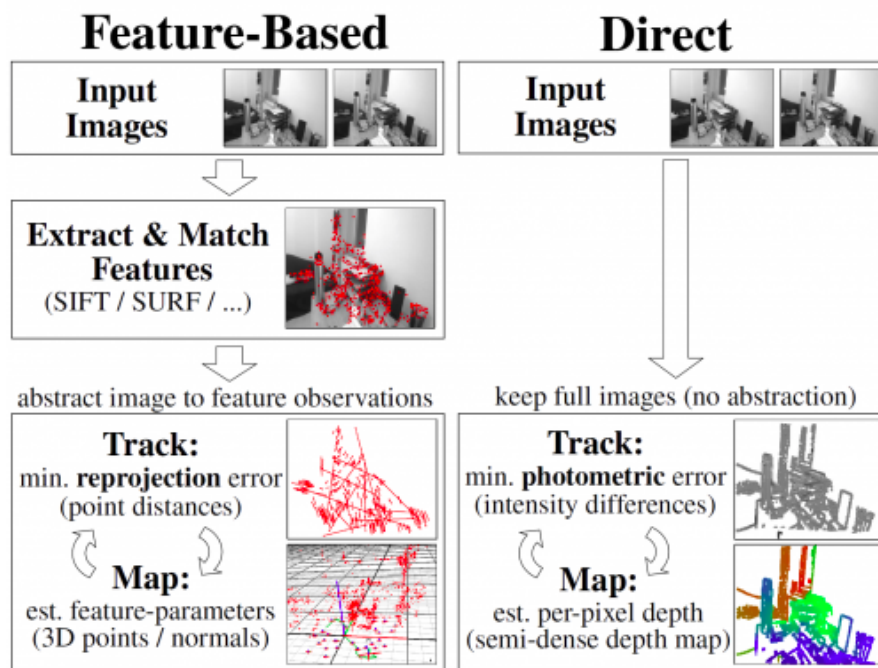


Figura 2.1: Comparativo do processo de execução entre os métodos direto e baseado em características. [Retirada da apresentação de Jakob Engel, *Semi-Dense Direct SLAM*]

No trabalho de Huletski et al. (2015) o autor aponta as seguintes métricas para estimar o desempenho e a qualidade dos principais algoritmos vSLAM:

- A acurácia da localização é a principal propriedade do algoritmo SLAM. Normalmente, utiliza-se o erro médio quadrático (RMS) entre as posições do robô e o ground truth.
- Tempo de processamento do conjunto de dados ou uso médio da CPU como medida de esforço computacional.
- Picos de consumo de memória.


	
Feature-Based	Direct
can only use & reconstruct corners	can use & reconstruct whole image
faster	slower (but good for parallelism)
flexible: outliers can be removed retroactively.	inflexible: difficult to remove outliers retroactively.
robust to inconsistencies in the model/system (rolling shutter).	not robust to inconsistencies in the model/system (rolling shutter).
decisions (KP detection) based on less complete information.	decision (linearization point) based on more complete information.
no need for good initialization.	needs good initialization.
~20+ years of intensive research	~4 years of research (+5years 25 years ago)
Jakob Engel	Semi-Dense Direct SLAM
	24

Figura 2.2: Comparativo de desempenho e robustez entre os métodos direto e baseado em características. *[Retirada da apresentação de Jakob Engel, Semi-Dense Direct SLAM]*

- Tempo de processamento por frame ou frames por segundo (FPS). Este parâmetro é importante para aplicações em tempo real visto que o robô precisa reagir à mudanças no ambiente com restrições no tempo de resposta (latência). Um exemplo comum são robôs aéreos.
- Robustez pode ser definida como a habilidade do algoritmo não degradar a acurácia da localização ao longo do tempo assim como atuar em cenários diversos como ambientes indoor e outdoor. Outra perspectiva para robustez é a capacidade do algoritmo produzir resultados semelhantes para as mesmas entradas.

Na figura 2.2 podemos ver um comparativo dos métodos em função das principais métricas.

A comunidade científica vem intensificando suas pesquisas nestes métodos durante os últimos anos, especialmente o método direto, e acredita-se que no futuro, os algoritmos de visual SLAM venham a agregar vantagens das duas classes. Dentre os algoritmos avaliados em [Cadena et al. \(2016\)](#); [Taketomi et al. \(2017\)](#) encontram-se o LSD-SLAM e o ORB-SLAM, considerados pela comunidade os principais algoritmos vSLAM do estado da arte do qual o ORB-SLAM será descrito em mais detalhes a seguir. Os algoritmos foram avaliados na base de dados [Sturm et al. \(2012\)](#) construída na Universidade Técnica de Munich (TUM), que é utilizado nos mais diversos trabalhos da área contendo cenários em vários patamares de dificuldade. Nestes trabalhos ([Huletski et al., 2015](#); [Taketomi et al., 2017](#)) conclui-se que o

vSLAM ainda gera resultados imprecisos e bastante suscetíveis a ruído. Com o objetivo de resolver esses problemas, estratégias de fusão de dados vêm sendo amplamente abordadas com o intuito de melhorar a acurácia e robustez dos algoritmos SLAM.

2.3 ORB-SLAM

O ORB-SLAM é considerado o principal algoritmo de vSLAM da atualidade (Taketomi et al., 2017) sendo o primeiro a realizar todas as tarefas necessárias da classe, descritas na seção 2.1 deste trabalho. Foi desenvolvido inicialmente para câmera monocular (Mur-Artal et al., 2015) e posteriormente estendido para câmeras estereográficas e RGB-D [ORB-SLAM2, Raul, 2017]. Dada sua importância, este foi o algoritmo escolhido para ser implementado nos experimentos e será descrito em mais detalhes adiante.

Podemos dividir o algoritmo ORB-SLAM em três partes que são executadas simultaneamente em threads paralelas: rastreamento, mapeamento local e fechamento de malha. A figura 2.3 apresenta uma visão geral dos módulos do sistema.

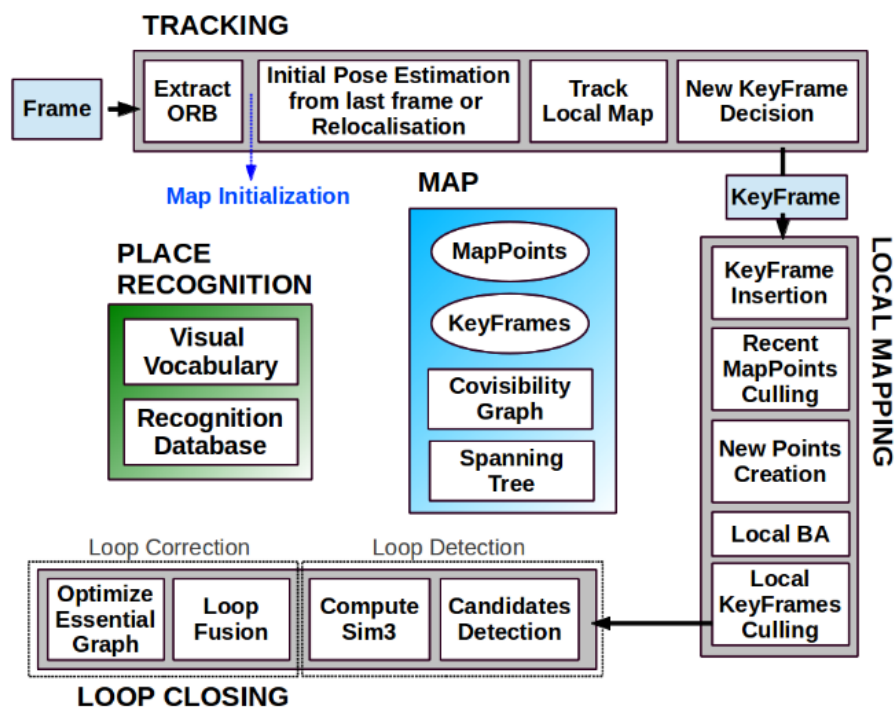


Figura 2.3: Visão Geral do Sistema ORB-SLAM. [Retirada de ORB-SLAM]

2.3.1 Rastreamento

O rastreamento é responsável pela localização da câmera em cada frame e pela decisão de quando e qual novo frame será inserido no vetor de frames chaves.

Extração de características: O algoritmo inicia executando este módulo, responsável pela extração de características dos frames de entrada. Existem diversos algoritmos capazes de realizar esta tarefa (Lowe (2004); Bay et al. (2008); Rublee et al. (2011)), no entanto, devido as restrições de execução em tempo real, visando um FPS de até 33ms, e buscando generalizar a capacidade de reconhecimento para diversas aplicações, o extrator de características escolhido foi o ORB, que apresenta boa performance e robustez a rotação e escala.

Estimação da pose ou relocalização: Uma vez detectados os pontos, usa-se um modelo de velocidade constante para prever a região desses pontos na próxima iteração. Se houver o casamento de pontos entre os frames, a pose da câmera será otimizada, caso contrário, o rastreamento é perdido, e o algoritmo entra no modo de relocalização. Uma vez no modo de relocalização, com o intuito de se relocalizar no mapa, os frames de entrada são convertidos em bags-of-words(palavras-chave), que posteriormente são cruzados com a base de dados contendo todos os keyframes localizados anteriormente.

Rastreamento de mapa local: Uma vez que temos a pose da câmera e um conjunto inicial de pontos correspondentes, podemos projetar o mapa e buscar mais correspondências entre pontos. Para eliminar a complexidade em grandes mapas, utiliza-se apenas um mapa local.

Decisão do novo frame chave: O último passo é decidir se o frame de entrada atual é apropriado para se tornar um frame chave. Para inserir um frame chave as seguintes condições precisam ser satisfeitas de forma a eliminar frames redundantes intensificar a robustez à movimentos bruscos: Mais de 20 frames devem ter passado desde a última relocalização global. O mapa local deve estar ocioso ou mais de 20 frames devem ter passado desde a última inserção de frame chave. O frame atual deve rastrear pelo menos 50 pontos. O frame atual deve rastrear pelo menos 90% dos pontos do frame de referência.

2.3.2 Mapeamento Local

As etapas do mapeamento local são executadas toda vez que um novo keyframe é inserido.

- **Inserção do Keyframe:** O primeiro passo ao inserir um novo keyframe é adicionado ao grafo de covisibilidade, atualizando as arestas que correlacionam os frames. Em seguida, atualiza-se a árvore de custo mínimo ligando o novo keyframe à um keyframe já existente, de acordo com o número de pontos em comum compartilhados por eles. Depois disso, encontra-se a representação do keyframe em bag-of-words para triangular novos pontos.

- **Eliminação de Map Points recentes:** Para os map points serem retidos permanentemente no mapa, os pontos devem passar por um teste de restrição para assegurar que não foram erroneamente estimados devido à associação de ruído. As duas condições do teste são: o rastreamento deve encontrar os pontos em mais de 25% dos frames em que ele foi previsto; e os pontos devem ser observados em pelo menos 3 keyframes.
- **Criação de novos Map Points:** Novos pontos são criados pela triangulação do ORB entre os keyframes conectados no grafo de covisibilidade.
- **O BA local otimiza o keyframe atual,** todos os keyframes associados a ele no grafo de covisibilidade e todos os map points vistos por esses keyframes. Observações que sejam identificadas como outliers serão descartadas da otimização.
- **Eliminação local de keyframes:** Afim de reduzir a complexidade e manter a reconstrução compacta keyframes redundantes são detectados e eliminados. Esta etapa é importante pois o custo computacional em termos de processamento e memória sofrem grande impacto na execução do BA, sendo crucial essa podagem para viabilizar a aplicação em tempo real. A política de eliminação adotada é descartar os keyframes cujo 90% dos map points estejam presentes em pelo menos outros 3 keyframes.

2.3.3 Fechamento de Malha

Ao longo do rastreamento e mapeamento erros de deslocamento são acumulados e eventualmente o mapa global fica desalinhado em relação ao seu percurso original. Podemos observar esse erro na figura 2.4

A thread do fechamento de malha computa a semelhança entre o vetor bag-of-words do último keyframe processado e todos os seus vizinhos e em seguida consulta a base de dados de reconhecimento, descartando todos os keyframes inferiores a um determinado limiar. Os keyframes restantes que estão conectados ao grafo de covisibilidade são considerados possíveis candidatos. Depois, calculamos uma Transformação de Semelhança do atual keyframe que nos informa sobre o erro acumulado no loop e ao mesmo tempo nos confirma se o loop existe de fato. Uma vez detectado, o próximo passo é fundir os mapas de pontos duplicados e inserir novas arestas no grafo de covisibilidade. A transformação de semelhança corrige o keyframe atual e propaga para todos os keyframes vizinhos. Por fim, para efetivamente fechar o loop, computa-se uma otimização global sobre o grafo essencial, distribuindo o erro do loop ao longo de todo o grafo.

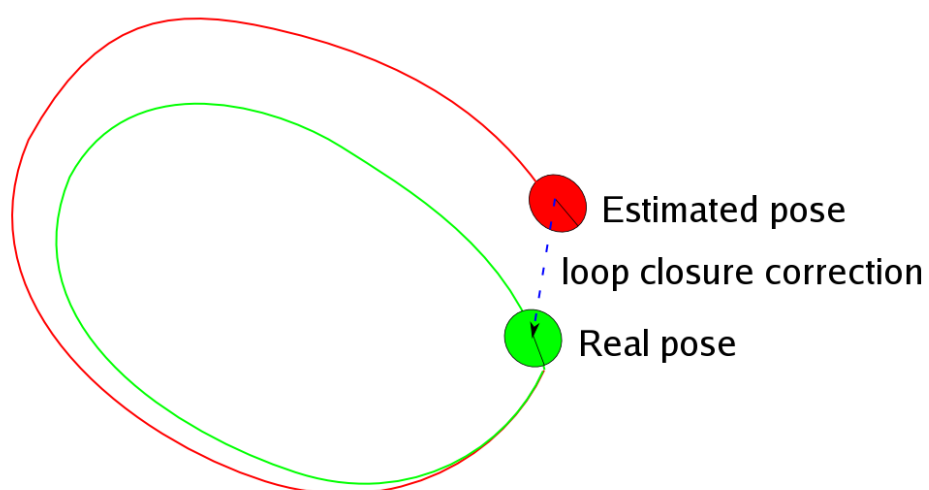


Figura 2.4: Diferença entre a posição real e estimada devido ao erro acumulativo de desvio.
[Retirada de <http://cogrob.ensta-paristech.fr/loopclosure.html>]

3

Metodologia

Neste capítulo mostramos a arquitetura do sistema robótico que foi implementado para executar o experimento de acurácia do algoritmo ORB-SLAM, descrevemos as ferramentas utilizadas e a etapa de calibração da câmera. Por último, detalhamos a etapa de avaliação do experimento.

3.1 Arquitetura do Sistema

A plataforma robótica é formada por três componentes: smartphone, que devido ao seu conjunto de sensores embutidos, como IMU, câmera e GPS, é usado com a função de coletar dados; computador remoto, que devido ao seus recursos computacionais é responsável por processar os dados advindos do smartphone e gerar informações; e o microcontrolador, que recebe comandos do computador remoto para atuar na dinâmica do robô.

A interação entre esses componentes pode ser caracterizada pela figura 3.1.



Figura 3.1: Diagrama da arquitetura do sistema

O sistema segue o seguinte fluxo: o smartphone faz a coleta de dados, neste caso, fornecendo o stream de vídeo através de um servidor web. O aplicativo IP Webcam é responsável por este serviço. No computador remoto temos três processos executando sobre a plataforma ROS. O processo `video_stream` é responsável por ler a sequência de frames e repassar para o processo ORB-SLAM que gera a nuvem de pontos e localiza o robô no mapa. O terceiro processo, cliente TCP, tem a função de controlar as velocidades do robô enviando comandos para o servidor TCP que está executando no microcontrolador.

3.2 Ferramentas Utilizadas

Para a visão do robô, foi utilizada a câmera de um *Smartphone* (Motorola G4 Plus, sistema operacional Android 7.0), por intermédio do aplicativo IP Webcam (<https://play.google.com/store/apps/details?id=com.pas.webcam&hl=en>).

O Microcontrolador em uso é o Raspberry Pi 0W (sistema operacional Raspbian), executando um Python TCP server implementado para esta aplicação.

O Computador Remoto da aplicação está configurado com um processador Core i7, 8GB de Memória RAM e Sistema Operacional Ubuntu 16.04 executando ROS Kinetic dispõe dos seguintes recursos:

- `video_stream_opencv` (http://wiki.ros.org/video_stream_opencv)
- ORB-SLAM (https://github.com/raulmur/ORB_SLAM)
- Controlador TCP client (Implementado para esta aplicação)

3.3 Calibração da Câmera

A calibração da câmera foi realizada de acordo com as instruções de http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration, utilizando um tabuleiro de xadrez 6x7 com 2cm de largura por quadrado e frames de 640x480 pixels. Na tabela a seguir apresenta-se os parâmetros encontrados.

Matriz da câmera	fx	fy	cx	cy
	484.591	503.495	322.319	261.021
Coeficientes de distorção	k1	k2	p1	p2
	0.052375	-0.067473	0.007882	-0.006179

Tabela 3.1: Parâmetros de calibração da câmera

3.4 Avaliação do Experimento

Para realizar o experimento construímos um ambiente a partir de blocos de isopor com bastante textura, de forma a tornar mais preciso os resultados do algoritmo. Dois cenários foram analisados, mostrados na figura 3.2. A validação dos mapas obtidos é feita através da comparação visual dos cenários 1 e 2 com a nuvem de pontos reconstruída. Além disso, procura-se observar comportamentos previstos pelo autor (Mur-Artal et al., 2015) como os efeitos da rotação pura e ambiguidade de escala.



(a) Cenário 1

(b) Cenário 2

Figura 3.2: Cenários para execução do experimento



Resultados e Discussões

Neste capítulo apresentamos o sistema experimental implementado e fazemos uma discussão sobre o que foi observado.

Executa-se o algoritmo ORB-SLAM, utilizando uma plataforma robótica que desenvolvemos para este fim. Os dados gerados pelo algoritmo são publicados em Tópicos do ambiente ROS (Robot Operating System), em conformidade com as práticas de Open Source Robotics. Os resultados do trabalho são descritos em duas partes: plataforma robótica, onde se descreve as etapas de construção do robô separadas em projeto mecânico, projeto elétrico e projeto de software; e estudo de caso, onde utiliza-se esta infra-estrutura para reproduzir o experimento e gerar os dados de localização e mapeamento.

4.1 Plataforma Robótica

Nesta seção descrevemos em detalhes o projeto do robô da figura 4.1.

4.1.1 Projeto Mecânico

O projeto mecânico foi desenvolvido no software [SolidWork Corporation \(2015\)](#), como se pode observar o esboço na figura 4.2, e impresso em uma impressora 3D com 0.4mm de extrusão em material ABS e densidade de 20%.

O perfil das peças no que diz respeito ao espaçamento dos parafusos, posição da bateria, posição das rodas e do celular foi definido baseado em experiências com protótipos anteriores de forma a evitar instabilidade na dinâmica do robô.

4.1.2 Projeto Elétrico

Para alimentar o sistema foi usado um carregador portátil de 5V. A ponte H é responsável por isolar o circuito de força, onde estão conectados os motores, do circuito lógico, onde estão

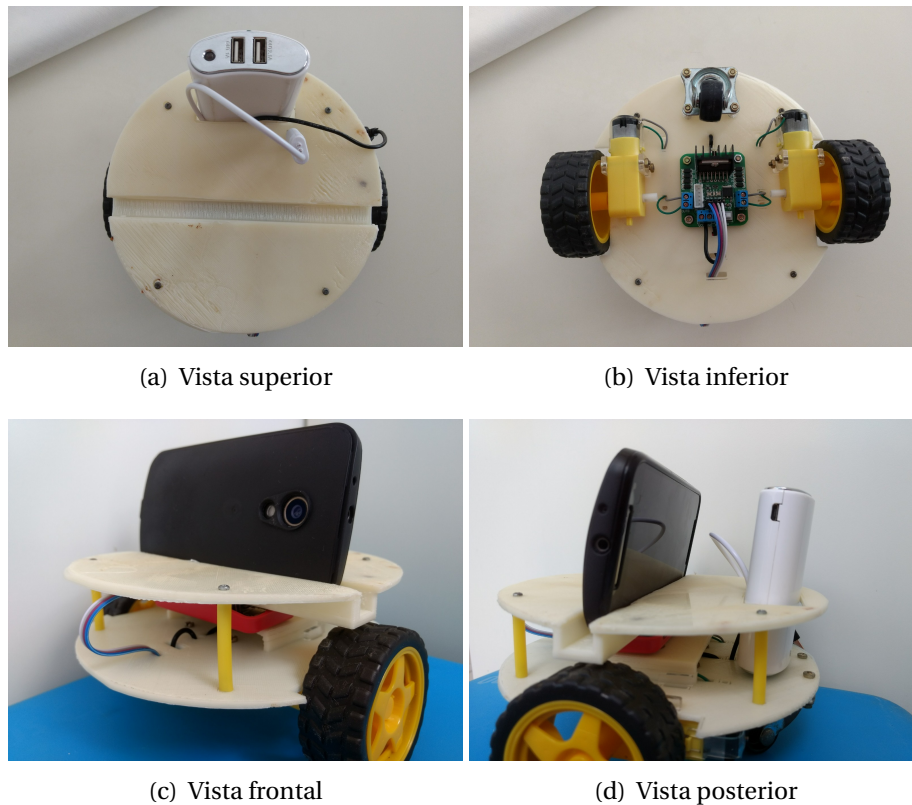


Figura 4.1: Plataforma robótica

conectados os pinos de controle do raspberry pi. No circuito lógico estão sendo usados os pinos pin 7 (GPIO 4), pin 12 (GPIO 18), pin 18 (GPIO 24) e pin 22 (GPIO 25), onde os pinos 12 e 18 tem a função de PWM.

Nas figuras 4.3 e 4.4 observamos as conexões elétricas em duas formas distintas de representação gráfica.

4.1.3 Projeto de Software

No projeto de software apresentamos o algoritmo responsável por prover a interface entre os algoritmos de maior complexidade, que estão sendo processados remotamente, com o corpo físico do robô.

Esses algoritmos mais complexos, responsáveis pela cognição, controle de trajetória e outras tomadas de decisões em geral, irão se reduzir meramente nas saídas V_r e V_l , indicando as velocidades das rodas direita e esquerda, respectivamente.

Uma vez calculada, as velocidades são enviadas via TCP do computador remoto para o sistema embarcado do robô, onde o código fonte a seguir lineariza as velocidades com a potência aplicada nos motores numa escala de -100 a 100. Nesta escala adotamos 100 como potência máxima, 0 para nenhuma força aplicada e o sinal negativo representando o sentido de giro contrário do motor.

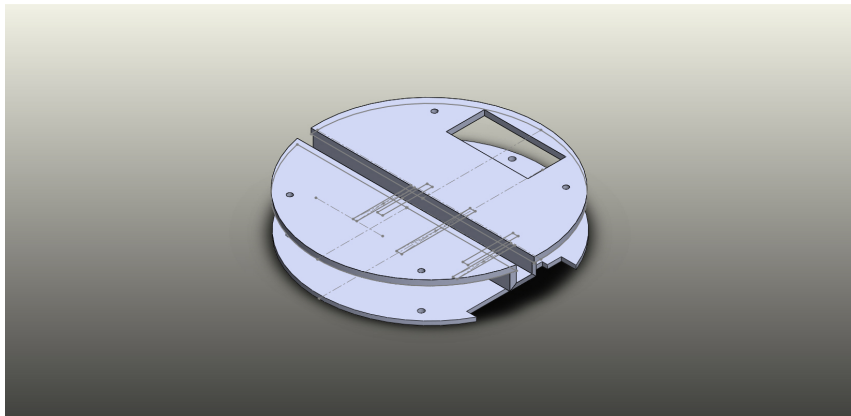


Figura 4.2: Esboço mecânico das peças

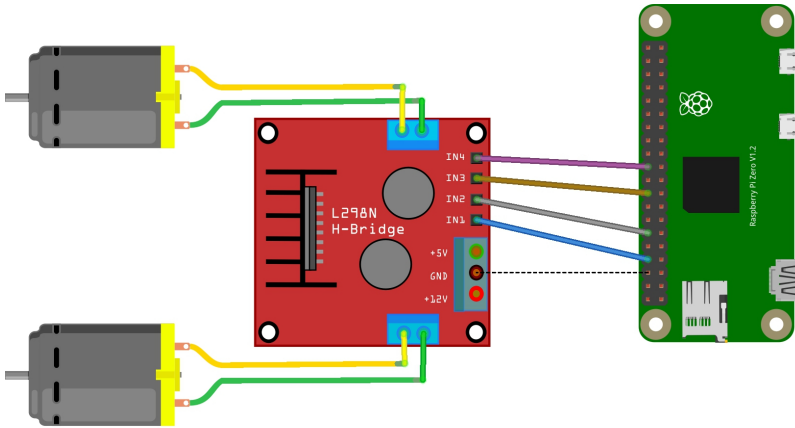


Figura 4.3: Esboço elétrico dos componentes

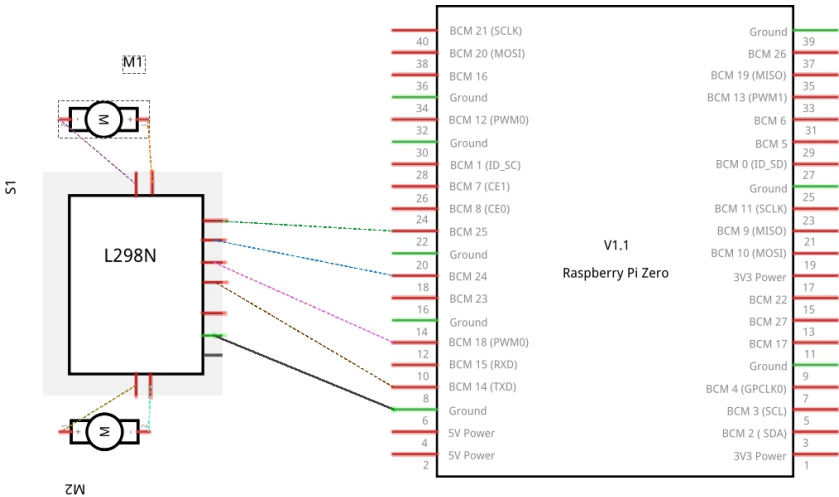


Figura 4.4: Esquemático

```
1 import socket
2 import sys
3 import time
4 import RPi.GPIO as GPIO
```

```
5
6 INA = 12 # marrom
7 INB = 7  # cinza
8 INC = 18 # roxo
9 IND = 22 # laranja
10 GPIO.setmode(GPIO.BOARD)
11 GPIO.setup(INA, GPIO.OUT)
12 GPIO.setup(INB, GPIO.OUT)
13 GPIO.setup(INC, GPIO.OUT)
14 GPIO.setup(IND, GPIO.OUT)
15
16 A = GPIO.PWM(INA, 50) #frequency = 50Hz
17 A.start(0)
18
19 C = GPIO.PWM(INC, 50) #frequency = 50Hz
20 C.start(0)
21
22 def constrain(value, lowerLimit, upperLimit):
23     if value > upperLimit:
24         return upperLimit
25     elif value < lowerLimit:
26         return lowerLimit
27     else:
28         return value
29
30
31 # This function map a value inside a specific range A
32 # into value inside another specific range B
33 def translate(value, leftMin, leftMax, rightMin, rightMax):
34     leftSpan = leftMax - leftMin
35     rightSpan = rightMax - rightMin
36
37     valueScaled = float(value - leftMin) / float(leftSpan)
38     return rightMin + (valueScaled * rightSpan)
39
40 # This function is responsible for control the wheels
41 # velocity from -100 ~ 100
42 def drive(velRightWheel, velLeftWheel):
43     constrain(velLeftWheel, -99, 99)
44     constrain(velRightWheel, -99, 99)
45
46     if velLeftWheel >= 0:
47         A.ChangeDutyCycle(velLeftWheel)
48         GPIO.output(INB, GPIO.LOW)
49     else:
50         GPIO.output(INB, GPIO.HIGH)
51         velLeftWheel = -velLeftWheel
```

```

52     velLeftWheel = translate(velLeftWheel, 0, 100, 100, 0)
53     A.ChangeDutyCycle(velLeftWheel)
54
55     if velRightWheel >= 0:
56         C.ChangeDutyCycle(velRightWheel)
57         GPIO.output(IND, GPIO.LOW)
58     else:
59         GPIO.output(IND, GPIO.HIGH)
60         velRightWheel = -velRightWheel
61         velRightWheel = translate(velRightWheel, 0, 100, 100, 0)
62         C.ChangeDutyCycle(velRightWheel)
63
64 # Create a TCP/IP socket
65 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
66
67 # Bind the socket to the port
68 server_address = ('192.168.15.22', 10000)
69 print >>sys.stderr, 'Starting up on %s port %s' % server_address
70 sock.bind(server_address)
71
72 # Listen for incoming connections
73 sock.listen(1)
74
75 try:
76     while True:
77         # Wait for a connection
78         print >>sys.stderr, 'Waiting for a connection'
79         connection, client_address = sock.accept()
80         #sock.settimeout(0)
81         print >>sys.stderr, 'Connecion from', client_address
82         velocity_r = "0"
83         velocity_l = "0"
84
85         # Receive the data in small chunks and retransmit it
86         while True:
87
88             velocity_r = connection.recv(3)
89             velocity_l = connection.recv(3)
90
91             if velocity_r:
92                 print >>sys.stderr, '———Velocities———'
93                 print >>sys.stderr, '      Vr = "%s"' % velocity_r
94                 print >>sys.stderr, '      Vl = "%s"' % velocity_l
95             else:
96                 print >>sys.stderr, 'No more data from', client_address
97                 drive(0,0)
98                 break

```

```

99
100         try:
101             if velocity_r == "0.0" and velocity_l == "0.0":
102                 drive(0,0)
103             else:
104                 velocity_r = velocity_r.strip('.')
105                 velocity_l = velocity_l.strip('.')
106                 drive(int(velocity_r.strip('\0')), int(velocity_l.strip('\0')))
107         except ValueError:
108             print '_____'
109
110     except KeyboardInterrupt:
111         pass
112     A.stop()
113     C.stop()
114     GPIO.cleanup()

```

Algoritmo 4.1: Código fonte em Python do sistema embarcado

4.2 Estudo de Caso

Nesta seção descrevemos os detalhes do estudo de caso.

De acordo com a metodologia proposta acima, temos dois cenários, mostrados na figura 3.2. Controlamos remotamente o percurso do robô nesses cenários de maneira a visitar todo o cenário.

Na figura 4.5 mostramos a etapa de detecção de pontos característicos realizada pelo algoritmo ORB-SLAM. Pode-se observar no rodapé da imagem três parâmetros que são KFs, representando o número de keyframes já contabilizados ao longo da execução; MPs, representando o número de mappoints também já contabilizados e o número de Matches, referindo as correspondências entre keyframes e mappoints.

Na figura 4.6 temos o resultado do algoritmo aplicado no cenário 1, com um fechamento de malha. Em azul, temos o histórico do percurso feito pela câmera (ou seja, pelo robô). Em verde, temos as ligações do grafo de visibilidade e a pose atual da câmera. Em preto, temos os pontos do mapa que já foram estabelecidos. Em vermelho, temos os pontos do mapa mais recentes que estão sujeitos a alterações. Podemos dizer que o mapa reconstruído delimita bem o contorno externo do cenário, porém ainda existem pontos cegos no contorno interno, o que pode prejudicar a navegação.

Na figura 4.7 temos os mesmos dados da figura 4.6 no ambiente do ROS. Os dados são acessados através de três tópicos: /ORB_SLAM/points, /ORB_SLAM/current_points e /ORB_SLAM/current_camera.

Na figura 4.8 temos o resultado do algoritmo aplicado no cenário 2. Novamente, po-



Figura 4.5: Identificação de pontos característicos no frame atual

demos dizer que o mapa reconstruído delimita bem o contorno externo do cenário inicialmente, porém comprova-se que os movimentos de rotação caem em singularidades prejudicando totalmente o restante da reconstrução.

Em 4.9 temos um snapshot do experimento, onde mostramos o cenário e as telas do keyframe e mapa executando em tempo real.

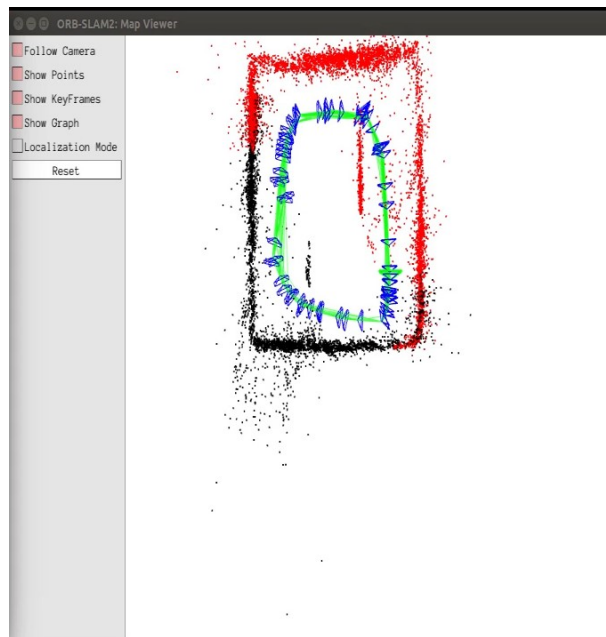


Figura 4.6: Mapa e localização gerado pelo ORB-SLAM no cenário 1

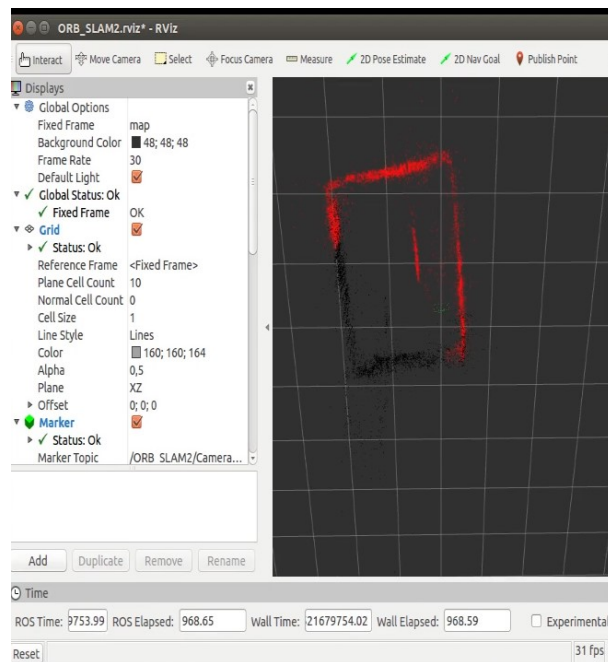


Figura 4.7: Mapa e localização gerado pelo ORB-SLAM no cenário 1 publicado no ambiente ROS

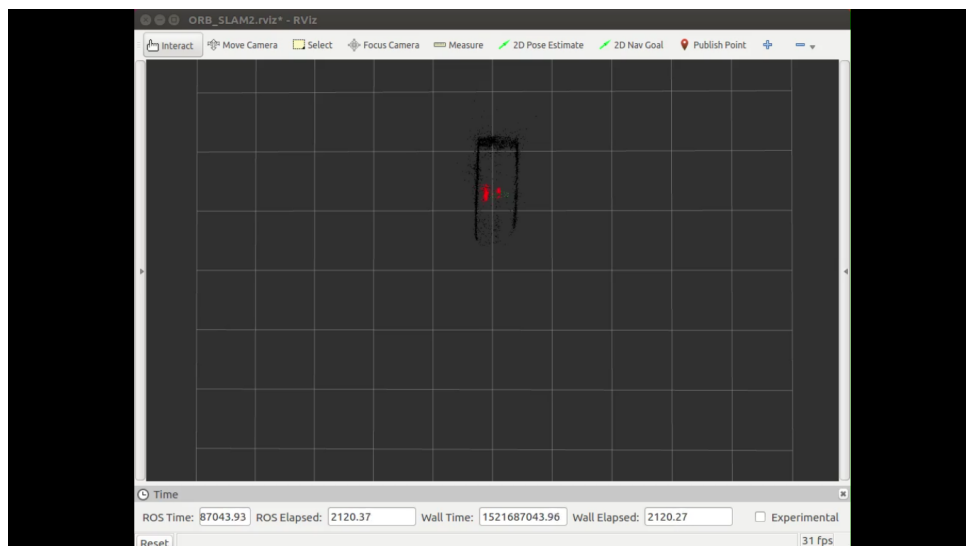


Figura 4.8: Mapa e localização gerado pelo ORB-SLAM no cenário 2 publicado no ambiente ROS

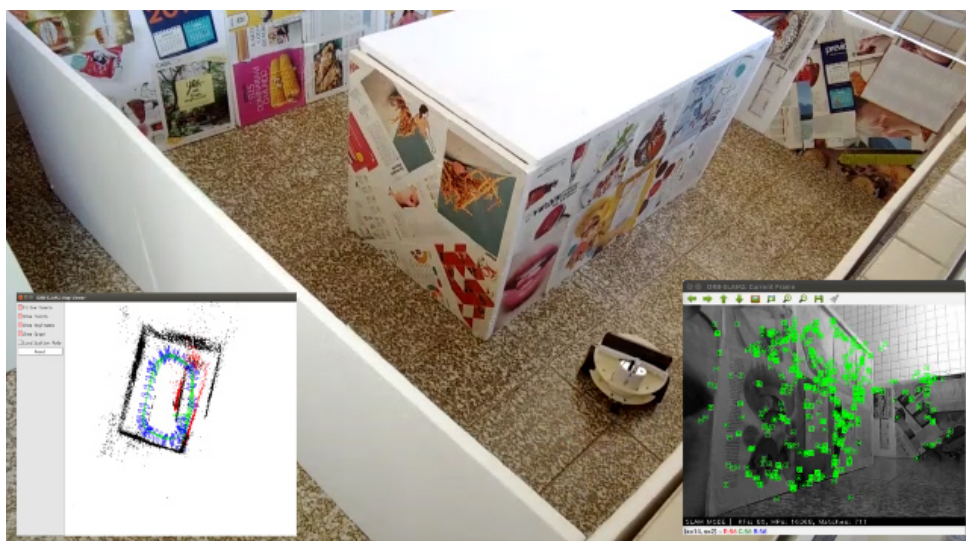


Figura 4.9: Snapshot do experimento em tempo real

5

Considerações Finais

Este trabalho realizou um levantamento do estado da arte dos algoritmos de SLAM visual. Implementamos experimentalmente o algoritmo ORB-SLAM, pontuando os desafios práticos e as características do algoritmo.

Como mostrado, existem vários trabalhos recentes que demonstram o interesse por parte da academia e empresas em resolver problemas relacionados à robótica móvel, mais especificamente, o problema do SLAM. O número de artigos publicados nos últimos anos (2015, 2016, 2017) nos permite classificar a área do vSLAM como ativa e crescente, motivada pelas características agregadas da câmera monocular, da qual podemos citar baixo custo, capacidade de extrair informações 3D e reconhecer contexto no ambiente e aplicações como jogos e *wearable devices*.

O resultado deste trabalho pode ser dividido em duas partes: um estudo de caso para o problema do SLAM, onde foi avaliado o algoritmo ORB-SLAM, acrescido pela contribuição da integração dos dados à plataforma ROS. Em segundo plano, foi projetado e construído um robô de baixo custo para a implementação do estudo de caso, do qual o projeto pode ser facilmente reproduzido por terceiros.

Dentre os desafios encontrados pelo ORB-SLAM e pelos algoritmos de SLAM visual monocular em geral, discutimos a ambiguidade de escala da câmera monocular e o movimento puramente rotacional, a necessidade de ambientes com textura e a susceptibilidade a falhas em ambientes mais complexos. Nos resultados deste trabalho é possível observar os efeitos da escala variável e do movimento rotacional afetando diretamente na qualidade da reconstrução do mapa.

Para resolver esses problemas, uma perspectiva de trabalho futuro é a incorporação de diferentes sensores a serem integrados através de estratégias de fusão de dados e, dessa forma, atingir um comportamento satisfatório em robustez e acurácia, não só para ambientes simples e controlados, mas também ambientes de maior complexidade em relação à navegação.

Entre essas estratégias, destacamos aquelas classificadas como visual-inercial fazendo uso da câmera em conjunto com o IMU, sendo, portanto, uma solução barata e que traz resultados promissores em relação ao estado da arte.

Referências bibliográficas

- Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc J. Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008. DOI [10.1016/j.cviu.2007.09.014](https://doi.org/10.1016/j.cviu.2007.09.014). URL <https://doi.org/10.1016/j.cviu.2007.09.014>.
- Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian D. Reid, and John J. Leonard. Simultaneous localization and mapping: Present, future, and the robust-perception age. *CoRR*, abs/1606.05830, 2016. URL <http://arxiv.org/abs/1606.05830>.
- A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, June 2007. ISSN 0162-8828. DOI [10.1109/TPAMI.2007.1049](https://doi.org/10.1109/TPAMI.2007.1049).
- Andrew J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1403–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1950-4. URL <http://dl.acm.org/citation.cfm?id=946247.946734>.
- Carlos Canudas de Wit, Bruno Siciliano, and Georges Bastin. *Theory of Robot Control*. Communications and Control Engineering. Springer - Verlag London, 1 edition, 1996.
- Arthur Huletski, Dmitriy Kartashov, and Kirill Krinkin. Evaluation of the modern visual slam methods. In *Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT)*, 2015.
- KUKA Robotics Corporation. URL www.kuka.com/en-de/products/mobility/navigation-solution.
- David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.
- Maja J. Mataric. *The Robotics Primer*. Intelligent Robotics and Autonomous Agents. MIT Press, 2007.

- Raúl Mur-Artal and Juan Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33, jun 2017.
- Raúl Mur-Artal, José María Martínez Montiel, and Juan Tardós. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31, oct 2015.
- Robin R. Murphy. *Introduction to AI Robotics*. Intelligent Robotics and Autonomous Agents. A Bradford Book, 2000. ISBN 978-0-262-13383-8.
- Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski. Orb: An efficient alternative to sift or surf. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc J. Van Gool, editors, *ICCV*, pages 2564–2571. IEEE Computer Society, 2011. ISBN 978-1-4577-1101-5. URL <http://dblp.uni-trier.de/db/conf/iccv/iccv2011.html#RubleeRKB11>.
- Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots, Second Edition*. Intelligent robotics and autonomous agents. MIT Press, 2011. ISBN 978-0-262-01535-6.
- Dassault Systèmes SolidWorks Corporation. *Introducing Solidworks 3D Mechanical CAD Software*, 2015. URL http://my.solidworks.com/solidworks/guide/SOLIDWORKS_Introduction_EN.pdf.
- J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.
- Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16, Jun 2017. ISSN 1882-6695. DOI [10.1186/s41074-017-0027-2](https://doi.org/10.1186/s41074-017-0027-2). URL <https://doi.org/10.1186/s41074-017-0027-2>.
- Meldon Wolfgang, Vladimir Lukic, Alison Sander, and Joe Martin. Gaining robotics advantage, June 2017. URL <https://www.bcg.com/en-br/publications/2017/strategy-technology-digital-gaining-robotics-advantage.aspx>.
- Özer Çiftçioğlu and Sevil Sariyildiz. Data sensor fusion for autonomous robotics. In Serdar Kucuk, editor, *Serial and Parallel Robot Manipulators - Kinematics, Dynamics, Control and Optimization*, chapter 19, pages 373–400. InTech, 2012.