Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

SLAM visual em ambientes híbridos: Estudo de caso com robô Pioneer

Gabriel M. de Miranda

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Carla M. C.e C. Koike

Brasília
2018

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

SLAM visual em ambientes híbridos: Estudo de caso com robô Pioneer

Gabriel M. de Miranda

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Carla M. C.e C. Koike (Orientador)
CIC/UnB

Prof. Dr. Flávio de Barros Vidal
CIC/UnB

Prof. Dr. Díbio Leandro Borges
CIC/UnB

Prof. Dr. Ricardo Pezzoul Jacobi
Coordenador do Curso de Engenharia da Computação

Brasília, 29 de junho de 2018

Dedicatória

Á minha família, pelo carinho, apoio e paciência.

Agradecimentos

Agradeço à minha família, minha mãe, Kátia, pelo amor imensurável por todos estes anos, à minha irmã, Marina, minha melhor amiga e confidente, à meu padrasto, Carlos, pela alegria e acolhimento.

Aos meus tios e primos, pelo amor, companheirismo e almoços de domingo.

Aos amigos que fiz no curso, Guilherme, Matheus, Amanda, Gabriel, Otávio, Ana, Henrique. No projeto CLARA, Pedro e Dayanne. Na equipe UnBeatables, em especial Raphael, Bruno, Jhonantans e Henrique. Nos estágios, Vinicius, Pedro, Yan, Diego, Mateus.

Aos professores, em especial Díbio, Mariana e Carla, por compartilharem seu conhecimento comigo e me mostrarem duas grandes áreas, a Visão Computacional e a Robótica.

Agradeço em especial à Carla, minha orientadora, pela paciência e apoio durante minha caminhada como pesquisador.

Agradeço à UnB, pela oportunidade de cursar um dos melhores cursos do país em engenharia e computação e permitir que me tornasse o profissional que sou hoje.

Resumo

O problema de *Simultaneous Localization and Mapping (SLAM)* consiste em um robô obter um mapa do ambiente em que se encontra ao mesmo tempo em que se localiza neste ambiente. Tal mapa permite que um robô realize diversas atividades por meio de planejamento de rotas e localização. Problemas e aplicações recentes de grande interesse, tais como busca e salvamento em áreas de desastre, exploração de minas abandonadas, carros autônomos e robôs para limpeza doméstica podem ser resolvidos ou estão diretamente ligados ao *SLAM*. Diante da relevância do problema, este trabalho propõe a realização de *SLAM visual* no 1º andar do prédio de computação da UnB, CiC/Est, utilizando o robô Pioneer 3-AT e sua câmera monocular. Para isto, o algoritmo ORB SLAM foi escolhido e adaptado para interagir com o sistema de comunicação distribuída desenvolvido localmente entre o computador embarcado do Pioneer e o computador de base, um notebook pessoal. Doze operações pelo prédio foram realizadas e analisadas de forma qualitativa, tendo por base o fechamento de *loops*, relocalização e distorções na escala do mapa gerado, sendo cada mapa obtido analisado por meio da sobreposição da nuvem de pontos gerada ao *ground truth* real do prédio. Por fim, as principais dificuldades encontradas são discutidas. Para trabalhos futuros, seria interessante unir um algoritmo de exploração inteligente capaz de auxiliar o vSLAM na geração dos mapas, tendo em vista os problemas explorados e discutidos nas análises. Outras aplicações interessantes seriam utilizar o mapa gerado para permitir que um robô realize a limpeza do prédio ou entregue documentos nas salas dos docentes.

Palavras-chave: vSLAM, VO, SLAM, Pioneer 3-AT, Visão Computacional, Robótica

Abstract

The SLAM problem consists of a robot getting a map of the environment in which it is at the same time that it is located in this environment. Such a map allows a robot to perform various activities through route planning and location. Recent issues and applications of big interest, such as search and rescue in disaster areas, exploration of abandoned mines, self-driving cars and domestic cleaning robots can be solved or are directly connected to SLAM. In view of the relevance of the problem, this work proposes the realization of visual SLAM in the first floor of the CiC/ Est building at UnB, , using the Pioneer 3-AT robot and its monocular camera. To do this, the ORB SLAM algorithm was chosen and adapted to interact with the distributed communication system developed locally between the Pioneer embedded computer and a base personal computer. Twelve operations around the building were performed and analyzed in a qualitative way, based on loops, relocalization and distortions in the scale of the generated map, each map being analyzed and compared by overlapping the cloud of points generated in the *ground truth* building. In the end, the main difficulties encountered are discussed. For future work, it would be interesting to combine an intelligent exploration algorithm capable of supporting vSLAM in map generation, in view of the problems explored and discussed in the analyzes. Other interesting applications would be to use the map generated to allow a robot to clean the building or deliver documents in the professors' rooms.

Keywords: vSLAM, VO, SLAM, Pioneer 3-AT, Computer Vision, Robotics

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

BA Bundle adjustment.

BoW Bag of Words.

CIC Departamento de Ciência da Computação.

CiC/Est Departamento de Ciência da Computação e Estatística.

CML Concurrent Mapping and Localization.

DoF Degrees of Freedom.

DSO Direct Sparse Odometry.

DTAM Direct Tracking and Mapping.

EKF Extended Kalman Filter.

fps frames por segundo.

GBA Global Bundle adjustment.

GPS Global Positioning System.

GPU Graphical Processing Unit.

GUI Graphical User Interface.

ICP Iterative Closest Point.

IP Internet Protocol.

JPEG Joint Photographic Experts Group.

KLTT Kanade-Lucas-Tomasi Tracking.

LAICO LAboratório de sistemas Integrados e COncorrentes.

LBA Local Bundle adjustment.

LSD SLAM Large Scale Direct monocular SLAM.

MBps Megabytes por segundo.

PGO Pose Graph Otimization.

PnP Perspective-n-Point.

PTAM Parallel Tracking and Mapping.

RANSAC RANdom SAmple Consensus.

RMSE Root Mean Square Error.

ROS Robot Operating System.

Rviz ROS visualization.

SDK Software Development Kit.

SFM Structure From Motion.

SLAM Simultaneous Localization and Mapping.

SVO Semi-direct VO.

TCP Transmission Control Protocol.

UnB Universidade de Brasília.

VNC Virtual Network Computing.

VO Visual Odometry.

vSLAM visual SLAM.

WBA Windowed Bundle adjustment.

Capítulo 1

Introdução

O problema de Simultaneous Localization and Mapping (SLAM) consiste em um robô obter uma representação do mapa de um ambiente ao mesmo tempo em que se localiza globalmente no ambiente. Para isto, o robô deve utilizar sensores de percepção, como *lasers* e câmeras. A grande dificuldade do SLAM está em sua dualidade; para poder se localizar, o robô precisa de um mapa consistente que o permita identificar lugares que já passou; e para poder mapear, ele precisa ter uma boa estimativa de onde está. Dentre as motivações para abordar este tema está o fato de que, com um mapa do ambiente, o robô pode realizar planejamento de rotas, localizar-se com precisão numa região ou até mesmo prover uma visualização intuitiva do ambiente para um humano. Problemas como busca e salvamento em áreas de desastre, exploração de minas abandonadas, carros autônomos e robôs para limpeza doméstica podem ser resolvidos ou estão relacionados com o SLAM.

Quando se leva em consideração características como o tipo do mapa, ambiente, precisão, performance, é possível afirmar que SLAM ainda não é um problema completamente resolvido. Alguns dos problemas ainda em aberto são a exploração inteligente do ambiente de forma a dar suporte ao mapeamento e a obtenção de mapas de alto nível, capaz de dar valor semântico ao que está sendo observado, como distinguir uma cadeira ou uma mesa.

Na literatura clássica de SLAM, a abordagem de filtros probabilísticos foi bastante explorada. É possível dizer que, para um ambiente *indoor*, com mapeamento *2D* e uso de sensores *laser*, o mapeamento com localização simultâneos já foi resolvido. Dentre os principais algoritmos desta era, estão o EKF SLAM, baseado no filtro de Kalman extendido, o FastSLAM, baseado no filtro a partículas Rao-Blackwellized, e o GraphSLAM, que em vez de filtro utiliza grafos.

Na era de percepção robusta que nos encontramos agora, filtros probabilísticos estão sendo substituídos por métodos de *Bundle adjustment (BA)*, com utilização de *keyframes*, e aproximando-se muito de problemas conhecidos em Visão Computacional, *Visual*

Odometry (VO) e *Structure From Motion (SFM)*. O enfoque passou a ser na utilização de câmeras como único sensor de percepção do ambiente, sendo agora chamado *visual SLAM (vSLAM)*, e na geração de mapas *3D*. Enquanto alguns problemas da era clássica foram resolvidos, outros surgiram, como inconsistências na rotação pura, inicialização do mapa e ambiguidade da escala. As etapas dos principais algoritmos de *vSLAM* tem uma estrutura básica, que são associação de dados, inicialização, rastreamento, geração e manutenção do mapa, recuperação a falhas e detecção de *loop*, que em geral ocorrem de forma paralela. Os principais métodos de *vSLAM* monocular são PTAM, LSD SLAM e ORB SLAM.

O objetivo deste trabalho é realizar o mapeamento e localização simultâneos utilizando o robô Pioneer 3-AT no prédio CiC/Est de computação da Universidade de Brasília (UnB). Para isto, foi escolhido como base o algoritmo ORB SLAM, considerado o mais completo na categoria *features* monocular, e o sistema operacional Robot Operating System (ROS). Foi montado um sistema de comunicação distribuída a cabo *Ethernet* capaz de fornecer taxa de *frames* em tempo real a partir da câmera do robô para o *notebook* pessoal. O robô foi operado pelo prédio por meio de um *joystick* usb comum. O algoritmo ORB SLAM foi adaptado para interagir com o sistema nesse contexto.

Para avaliar os resultados obtidos, doze operações foram realizadas pelo prédio buscando horas do dia, tempo climático e trajetos diferentes e foram divididas por tipo da operação, se todo o 1º andar do prédio foi usado, por tipo do mapa gerado e se ficou completo, incompleto ou irreconhecível. Uma análise qualitativa foi realizada, que discutiu se cada operação fechou *loops*, realizou relocalização ou teve problemas de escala, sendo que ao fim o mapa gerado foi comparado com um *ground truth* do prédio. Por fim, as principais dificuldades envolvidas foram abordadas.

Esta monografia está dividida da seguinte forma: o Capítulo 2 apresenta os fundamentos, com revisão bibliográfica, principais algoritmos e categorias de SLAM, com enfoque no *ORB SLAM*. O Capítulo 3 discorre das ferramentas utilizadas, do sistema implementado e realiza uma breve descrição do prédio. O Capítulo 4 mostra a metodologia utilizada para permitir que o *SLAM* fosse realizado no prédio. O Capítulo 5 mostra os resultados obtidos das operações e as principais dificuldades encontradas. O capítulo 6 apresenta a conclusão e trabalhos futuros.

Capítulo 2

SLAM

2.1 Introdução

Ter um mapa de um ambiente permite que muitas tarefas em robótica possam ser realizadas, tais como planejamento de rotas, localização em um ambiente ou mesmo prover uma visualização intuitiva de um local para um operador humano [2]. Ter um mapa de landmarks[1] distinguíveis permite a uma robô resetar seu erro de localização ao revisitar áreas já conhecidas. Problemas atuais, tais como busca e salvamento em áreas de desastre [9] e exploração de minas abandonadas [10], tarefas em ambientes nocivos para os seres humanos, podem ser tratados por um robô provido de um mapa. Outros, como carros que dirigem de forma autônoma [11] e robôs que realizam limpeza doméstica [12], trariam maior segurança e conforto para a sociedade. Tais assuntos foram citados na literatura, em diferentes níveis de aprofundamento, como problemas a serem tratados com *SLAM*.

O objetivo de realizar *Simultaneous Localization and Mapping (SLAM)* é obter um mapa consistente do ambiente em que o robô se encontra. Para construir um mapa, o robô precisa obter estímulos externos, que o faz por meio de sensores capazes de simular o sensoriamento humano. Além disto, para mapear estes estímulos no local correto, o robô precisa ter uma boa estimativa de seu estado ou ***pose*** - posição e orientação - no ambiente, o que torna o mapeamento também um problema de localização. Daqui provêm a dualidade a que *SLAM* está exposto: para que o robô possa se localizar, precisa de um mapa consistente que o permita identificar lugares já visitados, sendo que para obter um mapa consistente precisa ter uma boa estimativa da posição e orientação em que se encontra a cada instante de tempo. Somado a isto, determinar correspondências espaciais entre medidas de sensores diferentes - associação de dados - e reconhecer se um local já foi visitado anteriormente no ambiente - fechamento de *loop* - constituem alguns dos

[1] Formas criadas pela associação de *features* que possuem uma assinatura que as identifica unicamente em meio a outras *landmarks*[8]

maiores desafios que devem ser solucionados no *Concurrent Mapping and Localization (CML)*[2][8, 13, 14]. Sendo assim, dadas as medidas de sensores e comandos de controle de movimentação de um robô este deve realizar mapeamento e localização simultaneamente.

Apesar de ser uma área de pesquisa bem estabelecida e com maturidade, pode-se afirmar que o *SLAM* ainda não foi resolvido ao considerar as diversas combinações possíveis envolvendo o tipo de robô, o ambiente em que encontra e a performance da execução [2]. Alguns dos fatores envolvendo o tipo do robô são sua dinâmica de movimento, a acurácia e frequência de captura dos sensores e seus recursos computacionais. Para o ambiente, considera-se o tipo do mapa desejado, se poderão ser utilizadas *landmarks* inseridas artificialmente, se existe a presença de muitos elementos dinâmicos além do robô e se apresenta muita simetria, este último podendo confundir a percepção dos sensores. Em questões de performance, considera-se a acurácia na estimativa do estado do robô, a acurácia métrica do mapa gerado e seu tipo de representação, a taxa de sucesso em que o mínimo de acurácia é satisfeita para situações diferentes, a performance computacional do sistema, o tempo máximo de operação e o tamanho da área mapeada.

A lista a seguir aborda questões ainda não resolvidas ou que foram pouco exploradas.

- ***SLAM* ativo** (do termo em inglês *Active SLAM*). Consiste em minimizar a incerteza do mapa e da localização por meio de ações de controle autônomas realizadas pelo robô. É considerado um problema de exploração.
- Recuperação a falhas ou escolha automática de parâmetros, que permita adaptar entradas do sistema de acordo com o ambiente e retornar a um estado válido após uma falha.
- Mapeamento de alto-nível, que dá valor semântico aos objetos do mapa e não apenas uma reconstrução geométrica.
- Uso consciente dos recursos computacionais, que permita adaptar o custo dos algoritmos de acordo com a performance do sistema.
- Percepção inteligente, que seja capaz de selecionar apenas informações relevantes e descartar as de baixo interesse.

2.2 SLAM clássico

A literatura clássica [2] obteve resultados satisfatórios ao tratar o *SLAM* como um problema probabilístico, criando modelos tanto para a movimentação do robô, realizada por meio do controle de seus atuadores, quanto para as medições realizadas por seus sensores.

[2]Referente ao problema de SLAM

É possível considerar que mapear um ambiente *indoor*[3] 2D com um robô equipado com odômetro e sensor *laser*, com boa acurácia de localização e baixa taxa de falha está em grande parte resolvido [2].

2.2.1 Abordagens

Na forma probabilística existem duas maneiras principais de abordar o SLAM. Na primeira delas busca-se estimar a probabilidade posterior 2.1 utilizando-se filtros, sendo x_t a pose do robô no momento atual, m o mapa, $z_{1:t}$ e $u_{1:t}$ as medidas do sensor e controles de movimento do início da execução ao momento atual. Dados o mapa e o estado mais recente do robô, a probabilidade é estimada a medida em que os dados dos sensores se tornam disponíveis. Tal abordagem é conhecida como *online SLAM* ou *proactive SLAM*, sendo que os filtros utilizados englobam os filtros paramétricos, tal como o filtro de Kalman extendido no *EKF-SLAM* e os filtros a partículas, tal como o Rao-Blackwellized no *FastSLAM*[13, 15, 14, 8].

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{2.1}$$

Na segunda abordagem estima-se a probabilidade posterior 2.2 por meio de suavizadores, sendo $x_{1:t}$ as poses do robô do início ao fim da execução, $z_{1:t}$ e $u_{1:t}$ as medidas do sensor e controles de movimento do início ao fim da execução. Dados o mapa e caminho completo percorrido pelo robô, a probabilidade é estimada dados todos os controles e medidas do sensor obtidos da operação do algoritmo. É conhecido como *full SLAM lazy SLAM*, sendo os estimadores algoritmos de otimização baseados em grafos, tal como o utilizado no *GraphSLAM*.

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{2.2}$$

2.2.2 Modelo de movimento

O modelo de movimento descreve a probabilidade de mudança de estado, 2.3, sendo o estado atual, x_t, calculado a partir do estado anterior, x_{t-1}, e do controle atual, u_t. Em geral, um ruído gaussiano é introduzido com o objetivo de modelar o erro inerente do movimento, que pode ser dado por comandos de **velocidade** ou por medidas de sensores com informações de posição e orientação, como o odômetro e o *Global Positioning System (GPS)* [16]. O odômetro obtém a pose por meio de *encoders* de rotação que medem o movimento rotacional das rodas, sendo o ruído dado por efeitos de deslize e irregularidades

[3] Ambiente fechado

do piso. O *GPS* obtém o posicionamento via satélites e está sujeito a leituras distorcidas devido à baixa potência do sinal. O odômetro é bastante utilizado e apresenta resultados melhores que o modelo de velocidade [8].

$$p(x_t|x_{t-1}, u_t) \tag{2.3}$$

2.2.3 Modelo de percepção

O modelo de percepção descreve a probabilidade, 2.4, de realizar uma leitura, z_t, dado o estado atual do robô, x_t, e o mapa com todas as leituras já realizadas, m. Em geral, um ruído gaussiano é utilizado para modelar os erros que podem ocorrer nas leituras dos sensores, que podem ser **acústicos**, como o sonar e ultrasom, ***laser*** e de **visão** [16]. O modelo pode utilizar medidas cruas, como a distância do robô ao objeto mais próximo, gerando uma nuvem de pontos. Neste caso, o mapa é representado por uma **grade de ocupação**, que armazena as probabilidades de um *grid* estar ocupado ou não. Outra alternativa é usar medidas com extração de *landmarks*, sendo o mapa representado por seus parâmetros de assinatura, diminuindo o espaço de armazenamento e facilitando a representação [13].

$$p(z_t|x_t, m) \tag{2.4}$$

2.2.4 Principais algoritmos

EKF-SLAM

O *SLAM* baseado no filtro de Kalman extendido [17] é uma técnica para estimação de estado baseada no filtro de Bayes que busca predizer funções não-lineares com aproximações lineares obtidas da expansão da série de Taylor de 1º ordem. Considera que a probabilidade posterior tem uma distribuição gaussiana, assim como todos os estados estimados. Para o cálculo da posterior, Equação 2.1, é encontrada sua **média** e **covariância** por meio de dois passos realizados em cada espaço de tempo

1. Predição

   Calcula o novo estado do robô a partir do controle e do estado anterior e também uma matriz de covariância que modela a incerteza desse novo estado.

2. Correção

   Resolve as associações de dados entre as medidas dos sensores e corrige o novo estado e sua matriz de covariância com as informações da medida.

Suas principais vantagens são sua facilidade de implementação e produção de uma solução exata para a posterior. Como desvantagens, temos seu grande custo computacional, que restringe o tamanho do mapa para apenas algumas centenas de *landmarks* [8]. Outras desvantagens são a aproximação linear dos modelos não-lineares, que pode resultar em inconsistências na solução e a incapacidade de lidar com distribuições multi-modais [13, 15, 14, 8].

FastSLAM

Baseado no filtro a partículas Rao-Blackwellized, utiliza partículas para representar a posterior, Equação 2.2 [18]. O *FastSLAM* resolve tanto o *online SLAM* quanto o *full SLAM*, pois encontra tanto o caminho completo quanto uma *pose* por tempo. Além disso, cada partícula contém uma estimativa da *pose* do robô, assim como um *set* de *EKFs* independentes, com média e covariância, que rastreiam os estados das *features*[4] no mapa, sendo o problema de mapeamento fatorado em diversos subproblemas. Consiste de três etapas principais

1. Amostragem

   São recuperadas M partículas e suas *features* a partir da distribuição posterior, Equação 2.2. Para cada partícula, uma nova *pose* é amostrada usando o modelo de movimento, Equação 2.3.

2. Atualização

   Features são observadas do mapa, sendo as novas adicionadas ao vetor de estados das partículas e as já observadas atualizadas a fim de calcular o **fator de importância** de cada partícula.

3. Re-amostragem

   A posterior é encontrada adicionando M partículas escolhidas arbitrariamente à sua distribuição.

As vantagens do FastSLAM são que pode ser implementado em tempo logarítmico com o número de *features*, possui maior robustez ante problemas de associação de dados e é capaz de modelar funções não-lineares. Sobre as desvantagens, sofre de perda temporal de acurácia devido ao fato que a re-amostragem causa depleção do histórico e a marginalização do mapa causa dependência neste mesmo histórico de poses e medidas obtidos. Além disso, existe a dificuldade em escolher um número ótimo de partículas que balanceie o ganho de acurácia com a perda de performance [13, 15, 14, 8].

[4]Características de grande gradiente na imagem, como cantos e linhas. Uma landmark comporta uma ou mais features[8]

GraphSLAM

O *GraphSLAM* [19] representa a informação por meio de um grafo esparso de restrições, diferente do *EKF-SLAM* que representa com um vetor de estados e uma matriz de covariância. Os mapas podem alcançar grandes proporções, sendo que inicialmente apenas informações são acumuladas no grafo, sem resolvê-lo. A posterior é calculada em cima da trajetória completa do robô, sendo o mapa estimado a partir de um *set* de dados de tamanho fixo, que permite que técnicas de linearização e associação de dados sejam mais eficazes que no caso *online*. Em geral, os mapas gerados são mais exatos que os do *EKF-SLAM* [8]. Restrições suaves não-lineares, das medidas dos sensores e de odometria, que formam o grafo são extraídas e linearizadas para formarem uma matriz e um vetor de informação. Pode ser dividido em duas etapas

1. Construção do grafo

   Os dados de medidas dos sensores e controles são utilizados para construção do grafo. Os nós são as poses do robô e as *features* do mapa. Cada aresta corresponde a um evento de movimento, que liga dois nós de *pose* do robô, ou um evento de leitura de sensores, que liga um nó de *pose* a um nó de *feature* do mapa. As arestas correspondem às restrições que ligam medidas dos sensores ao estado do robô.

2. Inferência

   O mapa e a trajetória são então estimados a partir da matriz de informação, que requer que um sistema de equações lineares seja solucionado. Otimizações de grafos e álgebra linear são utilizados para tornar a solução viável.

A principal vantagem é que provê um mapa consistente, permitindo relinearizar o modelo por meio de poses passadas. A desvantagem é que necessita armazenar toda a trajetória percorrida pelo robô.

2.3 Visual SLAM

Algoritmos de *visual SLAM (vSLAM)* têm ganhado muita popularidade nos últimos anos [4]. Dentre os principais motivos, por câmeras estarem presentes em diversos dispositivos, seja em *smartphones*, tablets e inúmeros robôs, além de terem custo, tamanho, peso e consumo de energia reduzidos e por serem utilizadas em várias aplicações que se complementam, como visão computacional, realidade aumentada e robótica [4]. Outra vantagem a ser considerada é que a odometria com informação visual (termo em inglês *Visual Odometry (VO)*) e inercial apresenta diversas vantagens em relação ao odômetro convencional,

já que não sofrem com deslize de rodas e irregularidades do piso [20]. Nesta configuração, o erro na estimativa da *pose*[5] fica em torno de 0.5% do tamanho da trajetória [2].

Enquanto a era clássica deu principal enfoque na resolução probabilística de *SLAM* com a utilização de **filtros** a era de percepção robusta [2], que nos encontramos atualmente, deu principal enfoque a resolução por métodos de *global Bundle adjustment (BA)* [21] com utilização de *keyframes*[6], que se aproxima bastante do problema de *Structure From Motion (SFM)*, da área de visão computacional. Em [22] é mostrada a superioridade em acurácia da técnica de *BA* em relação ao filtro, principalmente devido ao fato de permitir a inclusão de grande quantidade de *features* no mapa em tempo linear. A técnica permite a otimização das *poses* e do mapa gerado, e apesar de sua complexidade crescer com o número de *frames* [20], permite processamento em tempo real levando em consideração a propriedade do *vSLAM* que permite que rastreamento e mapeamento sejam realizados em paralelo [3].

Os principais problemas abertos são o **movimento rotacional puro**, a **inicialização do mapa**, a estimação dos **parâmetros intrínsecos da câmera** e a **ambiguidade da escala** do sistema de coordenadas.

O problema de rotação pura acontece no caso monocular, em que não é possível fazer o mapeamento pois o algoritmo de **triangulação**[7] utilizado necessita de diferença nas posições dos frames de entrada. Soluções propostas para este problema envolvem utilizar modelos de projeção diferentes para cada caso, como rastreamento baseado em **homografia**[8] para rotações puras e rastreamento em **6 *Degrees of Freedom (DoF)***[9] para outros movimentos.

Para a inicialização do mapa, é exigido um movimento da câmera que permita uma ampla ***baseline***[10], o que não é ideal pois restringe a forma de operação do movimento. Algumas soluções propostas são o rastreio inicial de marcadores artificiais, em que um objeto conhecido é usado como referência e seu formato é usado para refinar o mapa.

A maioria dos *vSLAM* assumem que os parâmetros intrínsecos da câmera[11] já são conhecidos, o que exige que o usuário execute uma ferramenta para obtê-los. Formas de solucionar isto envolvem obter estes parâmetros durante a execução do algoritmo, que convergem no processo de estimação.

[5] No vSLAM, a pose é normalmente definida como os parâmetros extrínsecos da câmera com transformações lineares (de rotação e translação) realizadas no sistema de coordenadas global definido na inicialização[3]

[6] Frames com grande disparidade entre si para permitir triangulação acurada[3]

[7] Usado para estimar a posição 3D no mapa a partir da posição 2D na imagem[3]

[8] Transformação que relaciona um conjunto de pontos entre duas imagens[23]

[9] Graus de liberdade da câmera, que são translações em x,y,z e rotações em roll, pitch e yaw. Alguns algoritmos utilizam a escala do mapa como sétima coordenada[1]

[10] Distância entre câmeras no tempo (caso monocular) ou no espaço (caso estéreo)[20]

[11] Incluem as distâncias focais e coordenadas 2D do centro de projeção[14]

O problema de ambiguidade de escala ocorre no caso monocular. As soluções para estimar a escala absoluta do sistema de coordenadas envolvem a detecção de partes do corpo do usuário, como mãos e rosto, e também o uso de sensores inerciais, como acelerômetro. Em alguns algoritmos, técnicas de *BA* global e otimização de poses em 7 *DoF* eliminam o problema [3].

2.3.1 Tecnologias relacionadas

Tanto o *vSLAM*, quanto *VO* e *SFM* estimam o movimento da câmera e a estrutura 3D de um ambiente desconhecido.

VO

Enquanto o VO se preocupa em fornecer consistência local da trajetória e do mapa, o vSLAM busca obter consistência global. Apesar de compartilharem etapas semelhantes, vSLAM possui escopo expandido e **detecção de loop**.

SFM

Diferente do SLAM visual, o SFM frequentemente opera com um *set* de imagens fora de ordem com nenhuma ou pouca restrição de tempo, podendo operar em escala global de reconstrução.

2.3.2 Framework básico

De maneira geral, tanto vSLAM quanto VO possuem os módulos básicos de **associação de dados**, **inicialização**, **rastreamento**, **geração do mapa**, **manutenção do mapa** e **recuperação a falhas**. Além destes, o módulo de **detecção de loop** está presente apenas no vSLAM.

Associação de dados

Nesta etapa, ocorre a detecção e casamento de *features* ou de toda a imagem. Diversos detectores e descritores de features estão disponíveis nas publicações acadêmicas, tais como o **SIFT** e o **ORB** [24, 25, 26]. Além disto, três tipos de associações podem ser realizadas entre os planos da imagem (2D) e do mapa virtual (3D), que são **2D-2D**, **3D-2D** e **3D-3D**.

Inicialização

Na inicialização é definido o sistema de coordenadas global, que permite a estimação da ***pose* da câmera** e reconstrução 3D iniciais. Para isto, a mesma cena deve ser observada por dois pontos de vista distintos separados por uma *baseline*, sendo que apenas a associação de dados entre elas é utilizada e conhecida. Alguns métodos utilizados são por cálculo das matrizes de **homografia**, **essencial** ou por **inicialização de profundidade randômica**.

Rastreamento

Também é chamado de estimação da pose. O mapa e a associação de dados entre dois frames são conhecidos, a pose é estimada com base na pose do *frame* anterior. Um **modelo de velocidade** constante pode ser usado, entretando falha com movimentos bruscos. Para algoritmos com alta taxa de quadros, é possível assumir que não há mudança de pose entre *frames* consecutivos. A *pose* pode ser usada para limitar a região de busca e casamento de *features* nos frames subsequentes. A *pose* é estimada **minimizando erros** fotométricos ou erros de reprojeção das *features*.

Geração do mapa

Nesta etapa, o mapa é expandido calculando a estrutura 3D do ambiente virtual a partir do ambiente real. Pode ser realizado por **triangulação** em dois pontos de vistas distintos, por estimação de profundidade com **filtros** ou por **mapa topológico**.

Manutenção do mapa

Na manutenção, o mapa é otimizado usando ou ***Global Bundle adjustment (GBA)*** ou ***Pose Graph Otimization (PGO)***. O primeiro é considerado melhor pois otimiza tanto as *poses* dos *keyframes* quanto a estrutura 3D, entretanto é mais lento[4]. No caso de VO, esta otimização considera apenas os keyframes mais recentes, sendo realizado ***Windowed Bundle adjustment (WBA)*** [20].

Recuperação a falhas

Também conhecida como relocalização, é necessária quando o rastreamento se perde, que pode acontecer devido a movimentos bruscos não previstos no modelo de movimento, em casos de rotação pura ou regiões com pouca ou nenhuma *feature*. Enquanto vSLAM procura se relocalizar considerando todos os keyframes, VO considera apenas os mais recentes.

Detecção de Loop

Por ser um problema de otimização, a estimação das poses está sujeita a erros que vão se acumulando a medida que novos keyframes são adicionados. Loops são estimados para corrigir e minimizar estes erros, sendo utilizados métodos como *BA* e *PGO*.

2.3.3 Categorias

Os algoritmos de *vSLAM* se dividem em diversas categorias distintas, quais sejam na **metodologia**, no tipo de **processamento na imagem**, no tipo de **mapa** gerado, na **configuração das câmeras** utilizada e na **licença de código**.

A Tabela 2.1 ilustra alguns dos principais algoritmos de *vSLAM* e *VO* existentes, assim como suas categorias. Alguns deles são abordados em diferentes níveis de detalhes, sendo que prioridade será dada aos que utilizam *keyframes* e são monoculares.

Metodologia

Sobre a metodologia, podem utilizar **filtro** ou ***keyframe***. No filtro, são utilizados os filtros probabilísticos da literatura clássica, como o filtro de Kalman e filtro a partículas. Já nos que utilizam *keyframes*, técnicas de visão computacional e otimização de grafos são usadas.

Como foi dito anteriormente, a técnica por *BA* e *keyframe* tem sido preferida por apresentar melhores resultados. A principal desvantagem do filtro é a incapacidade de gerir grandes quantidades de *features*, com crescimento quadrático.

Processamento na imagem

Podem ser baseados em ***features* (indireto)**, na **aparência (direto)** ou **híbrido (semi-direto)**. No baseado em *features*, pontos estratégicos da imagem são rastreados e mapeados num ambiente 3D virtual. No método direto, a imagem inteira é utilizada para estimação da pose e do mapa. No semi-direto, o rastreio é realizado com o casamento de *features* e o mapeamento é denso. O método indireto faz cálculo de erros de reprojeção, enquanto o método direto utiliza erros fotométricos.

A principal desvantagem do uso de *features* é que necessita que o ambiente seja texturizado, caso contrário o rastreamento se perde. Já o método direto é de grande custo computacional, de forma que as técnicas atuais utilizam *Graphical Processing Unit (GPU)* para o processamento. Além disso, não são robustos a oclusão [20].

Mapa

Sobre o tipo de mapa gerado, pode ser **esparso**, **denso** ou **semi-denso**. No esparso, o mapa 3D é composto apenas da reconstrução das *features*. No denso, toda a imagem é reconstruída. No semi-denso, áreas de maior gradiente são reconstruídas, dando apenas a forma do ambiente. Neste caso, localmente é métrico enquanto que de forma global é topológico.

A desvantagem do mapa esparso é que sua representação foca apenas na ideia geral do ambiente, quando é tratado como um mapa topológico. A desvantagem do mapa denso é o custo necessário para produzi-lo.

Configuração das câmeras

Sobre a configuração das câmeras, podem ser **monocular**, **estéreo** ou **RGB-D**. A monocular utiliza apenas uma câmera. No estéreo, duas ou mais câmeras são utilizadas com um pequeno deslocamento entre elas. No RGB-D, uma câmera monocular vem associada a um sensor infra-vermelho que capta informações de profundidade na imagem.

A câmera monocular sofre com problemas de escala no eixo de coordenadas e de rotações puras, sendo suas principais desvantagens. A câmera RGB-D têm seu alcance de profundidade limitado entre um e quatro metros. Além disso, sofre com a grande quantidade de dados obtidos na captura e só pode ser utilizada em ambientes *indoor*. No caso estéreo, quando a distância a cena é bem maior que a *baseline*, este degenera para o caso monocular.

Licença de código

Podem ser ***open-source*** ou ***closed-source***. No primeiro, o código é disponibilizado publicamente, enquando o segundo é privado.

2.3.4 Principais algoritmos

MonoSLAM (2007)

Utiliza uma abordagem probabilística que modela os estados da câmera (posição, orientação, velocidades linear e angular) em 6 *DoF* e das features (posição) e suas incertezas. Os estados e a atualização do mapa são atualizados com um EKF e utiliza um modelo de movimento para movimentos de câmera suaves, com velocidades linear e angular constantes, não lidando bem com acelerações bruscas.

A inicialização considera um conhecimento prévio de um objeto na cena, com *features*, tamanhos e aparência conhecidas, permitindo assinalar uma escala precisa. Usa um

tablado com quadrados e uma distância pré-definida, permitindo inicialização de profundidade e do sistema de coordenadas global.

Utiliza detecção de features Shi e Tomasi e descritor SIFT. Projeta as features 3D-2D, produzindo um template de busca das features em frames subsequentes. A profundidade é detectada via triangulação com múltiplas vistas. *Features* são descartadas quando possuem baixa acurácia e possuem uma etapa de estimação de orientação pela normal da superfície onde se encontram para atribuir maior robustez ao método.

Uma vantagem é que lida melhor com rotações puras em relação a outros métodos baseados em keyframes.

A principal desvantagem é o custo computacional que aumenta em proporção com o tamanho do mapa, sendo que o vetor de estados aumenta quadraticamente com o número de *features*. Devido a isto, suporta um número fixo de *features* (em torno de 100) para ser executado em tempo real (30Hz).

PTAM (2007)

Foi o primeiro a separar o rastreamento e o mapeamento em duas *threads* separadas. Foi pensado para pequenos ambientes de AR e manipulação pela mão. O rastreamento ocorre em tempo real enquanto o mapeamento ocorre com um custo computacional (BA) em intervalos de tempo maiores.

O mapa é inicializado com o algoritmo de *5 pontos*[12] e RANSAC[28], que estima a matriz essencial e triangula o mapa base, refinado com *BA*. É assumido movimento do usuário entre frames iniciais de 10 cm para estimar a escala. Não é necessária a entrada de profundidade explícita, porém um movimento de translação lento e suave deve ser feito paralelo à cena observada pelo usuário, que pode ser considerada frágil e de difícil execução.

O rastreamento é realizado a cada frame, sendo a estimativa da pose dada por modelo de velocidade. É utilizado detector FAST com Shi-Tomasi[29]. A busca por *features* é realizada em regiões e imagens específicas com níveis em pirâmede. A pose é representada como uma **transformação SE(3)** em seis parâmetros e calculada por meio de projeções 2D-3D com o casamento de texturas. Atualizações de poses são calculadas de forma iterativa minimizando erros de reprojeção. A qualidade do rastreamento é medida em cada frame, decidindo se será um keyframe ou não.

O mapeamento é realizado com a **triangulação** de *features* de *keyframes* com busca epipolar, podendo conter milhares de features.

[12]Dados 5 pontos comuns entre duas imagens e os parâmetros instrínsecos estima o movimento relativo da câmera por meio das matrizes Essencial(E), de rotação(R) e vetor de translação (t)[27]

Na otimização do mapa, é usado *LBA* com os cinco *keyframes* mais recentes e GBA entre todos os *keyframes*. Na recuperação a falhas, o frame é comparado com o banco de *keyframes* usando uma versão menor e borrada da imagem com *Randomized Tree-based Feature Classifier*.

A principal vantagem é que permite que o mapa seja representado por um grande número de features, sendo o algoritmo base para todos os outros baseados em keyframes subsequentes.

As desvantagens são a dependência com ambientes texturizados e a difícil interpretação do mapa por um operador humano, características das *features*.

DTAM (2011)

Sistema de rastreamento e reconstrução que utiliza cada pixel da imagem de uma câmera RGB. Por ser totalmente denso, consegue operar em tempo real em GPUs. Sistema monocular que cria superfície 3D densa e usa esta estrutura para rastreamento também denso.

A inicialização do algoritmo ocorre com um método estéreo de features até a aquisição do primeiro *keyframe*, a partir do qual se torna totalmente denso.

O rastreamento da *pose* ocorre alinhando cada frame com o modelo denso de textura utilizando o *Iterative Lucas-Kanade*[30]. O 6 DoF da câmera é rastreado em cada *frame*. A imagem de entrada é comparada com imagens sintéticas geradas do mapa reconstruído. A *pose* da câmera com relação ao mundo é representada por uma matriz pertencente ao SE3[13].

O mapeamento é feito usando múltiplas *baselines* com estéreo e é otimizado considerando a continuidade do espaço. O modelo denso é composto de *keyframes* sobrepostos, sendo informações fotométricas coletadas e resolvidas para obter mapas de profundidade. Um framework de minimização da energia global é usado para o mapeamento, onde a energia é a soma dos erros fotométricos.

O mapa é refinado e expandido dados os mapas de profundidade de texturas das *keyframes*.

Uma vantagem é que o modelo denso é resistente a oclusão e operação multi-escala.

Uma desvantagem é a perda de performance com borrão de movimento e a incapacidade de obter tempo real em CPU.

[13]Referente a álgebra de Lie. Uma transformação rígida arbitrária no SE(3) pode ser separada em uma translação com uma rotação rígida[31]

SLAM++ (2013)

É um algoritmo que propõe produzir um mapa semântico a nível de objetos utilizando câmeras RGB-D. Uma base de dados online armazena objetos 3D conhecidos. Usado com GPU. O algoritmo inclui fechamento de loop, relocalização e detecção no movimento de objetos.

Utiliza reconhecimento de objetos 3D em tempo real e rastreamento 6DoF criando um grafo de objetos. Usa refinamento de *poses* com o algoritmo *Iterative Closest Point (ICP)*.

Mundo representado por um grafo, em que cada nó armazena a pose SE(3) - rotação e translação em relação a um frame fixo - da câmera e objetos. Um método de malhas e votos é usado para reconhecer objetos na cena, sendo usados *Point-Pair Features (PPFs)*. Durante a execução, o mapeamento de pontos estimados é casado com os objetos *online* e são substituídos, reduzindo assim o problema de grande quantidade de dados. A estrutura 3D é estimada com a combinação de múltiplos mapas de profundidade.

O grafo de objetos é refinado continuamente por Pose Graph Otimization.

Uma vantagem é que faz reconstrução 3D densa com grande compressão na representação e gera uma cena a nível de objeto, podendo permitir interação com o ambiente.

SVO (2014)

Feito para *MAVs*, robôs aéreos de pequeno porte com câmera voltada para baixo. O algoritmo necessita rodar com alta taxa de quadros para alcançar boa acurácia. Opera diretamente em intensidades de pixels. É considerado uma versão esparsa do *DTAM* e *LSD SLAM*. No Semi-direct VO (SVO), o rastreamento é feito com casamento de *features*, enquanto o mapeamento utiliza um método direto.

Na inicialização, é assumida cena planar local e estimada a homografia, não sendo necessária entrada dupla de usuário e utilizando o primeiro *keyframe*, sendo as *features* FAST rastreadas com *Kanade-Lucas-Tomasi Tracking*. Monitora a mediana da *baseline* das *features* rastreadas entre o primeiro *keyframe* e o atual. Quando o valor está acima de um limiar, *parallax*[14] suficiente foi obtido e então estima a homografia, que é decomposta em *poses* e *landmarks* e são trianguladas e usadas para estimar a profundidade inicial. *BA* é utilizado entre os dois *keyframes*. Entretanto, sofre do mesmo problema do PTAM com a movimentação do usuário inicial.

Realiza rastreamento de *features* com rastreamento e mapeamento paralelos e seleção de *keyframes*, sendo a abordagem direta para precisão e melhor desempenho em cenas com baixa textura. Utiliza correspondência de *features* usando o movimento direto e

[14]Diferença relativa entre pontos coincidentes em imagens diferentes[32]

não com extração e casamento. Apenas usa extração de *features* quando um *keyframe* é selecionado para inicializar novos pontos 3D. Utiliza *features* FAST. O rastreamento com alta taxa de frames permite que o filtro convirja mesmo em cenas com textura repetitiva. Erros fotométricos são minimizados em pontos próximos às *features*. Assume mesma *pose* entre *frames* consecutivos, minimizando erros fotométricos entre eles. Requer pequena diferença entre *frames* consecutivos e alta taxa de quadros.

Um método probabilístico é usado para mapeamento. Quando não há *keyframes* sendo processados, o mapa é atualizado com um filtro, que converge quando a distribuição da profundidade estimada por uma semente vai de uma uniforme para uma distribuição gaussiana. Um filtro bayesiano que modela *outliers* é usado para estimar a profundidade dos locais das *features*. Quando a incerteza do filtro se torna pequena o bastante, um novo ponto 3D é inserido no mapa e imediatamente usado para estimação do movimento. Filtros de profundidade são inicializados no canto FAST com maior valor Shi-Tomasi na célula da imagem (30x30) que já teve correspondência 2D-3D.

Na manutenção do mapa, é mantida uma quantidade fixa de *keyframes* sendo os mais distantes removidos, mantendo alta performance em distâncias largas. Na recuperação a falhas, é tentado alinhar o *frame* com o banco de *keyframes*.

A principal vantagem é velocidade já que não extrai *features* a cada frame e acurácia com correspondência de *features* em *subpixel*. O ganho de velocidade se dá pois a extração e casamento de *features* não é necessária para estimar movimentos sendo que um método direto, baseado em intensidades, é usado.

LSD SLAM (2014)

Método monocular direto, sem uso de *feature*. Roda em tempo real em CPUs. Composto de três componentes principais, rastreamento, estimação do mapa de profundidade e otimização do mapa, três *threads* paralelas que começam após a inicialização. Estima erros fotométricos. Rastreia e mapeia trajetórias de mais de 500 metros, com grandes variações de escala e rotações. Aplica uma abordagem direta probabilística para estimar mapas semi-densos usados no alinhamento das imagens.

Na inicialização, uma profundidade randômica é escolhida para a cena do primeiro ponto de vista, que é então refinada por medidas dos frames subsequentes. Não requer geometria de duas vistas. Um único frame é utilizado, sendo pixels de interesse inicializados no sistema com uma distribuição de profundidade randômica e alta variância. A profundidade é refinada numa abordagem de **filtro**. Existe uma fase intermediária de rastreamento em que o mapa não é confiável. Na inicialização do sistema, é suficiente inicializar o primeiro keyframe com um mapa de profundidade randômico e grande variância.

O rastreamento estima a *pose* atual do *frame* anterior com uma tranformação *SE(3)* usando otimizações *Gauss-Newton*. Estimação da pose baseada no alinhamento da imagem. Rastreamento direto com detecção de *drift* de escala. Cada *keyframe* é composto de uma imagem, um mapa inverso de profundidade e sua variância. *Frames* não usados como *keyframes* são usados para refiná-los. Usa um método direto para alinhar dois *keyframes* em Sim(3), detectando e incorporando *drift* de escala. Abordagem probabilística que incorporta ruído nos mapas de profundidade para o rastreamento.

Mapeamento reconstruído como grafo de poses de *keyframes* com mapas de profundidade semi-densos obtidos de filtros probabilísticos. É semi-denso pois o mapa de profundidade é definido somente na vizinhança de pixels em regiões de alto gradiente, sendo a reconstrução limitada a estas áreas. Para resolver o problema de *drift* de escala, torna o inverso da média da profundidade sempre um, que incorpora a diferença de escala entre *keyframes*. Desta forma, valores randômicos são escolhidos como profundidade inicial e são otimizados pela consistência fotométrica.

Na otimização do mapa, roda uma terceira *thread* utilizando PGO para manter a consistência global, sendo este representado por *keyframes* conectados por restrições de *pose*. As arestas são transformações 3D de similaridade, incorporando detecção de mudança de escala no ambiente. Aplica o PGO em Sim(3) (ou SE3)[15], que representa a escala do sistema, permitindo correções de erros na escala e detecção de *loop* em tempo real.

Na recuperação a falhas, é tentado alinhar o novo *frame* com *keyframes* selecionados randomicamente. Detecção de *loops* são realizadas nos dez *keyframes* mais próximos. Realiza detecção de *loop* e grafo otimizado de pose em 7 *DoF*.

As principais vantagens são a criação de um mapa semi-denso de informação em tempo real usando CPU, a robustez ente rotações da câmera e correção de *drifts* na escala do mapa pelo filtro probabilístico.

DT SLAM (2014)

Sistema SLAM em tempo real que rastreia incrementalmente *features* 2D individuais e estima a *pose* pelo casamento entre elas, independente do comprimento da *baseline*. Três *threads* paralelas, rastreamento, mapeamento e BA. É assumido alta taxa de *frames*, o bastante de forma que a mudança de *pose* entre os *frames* não seja tão grande. Robusto a rotações puras, sendo que funde tanto a translação quanto a rotação no BA.

Não tem fase explícita de inicialização, que vem integrada no módulo de rastreamento como um método de estimação da matriz essencial.

[15]Para transformações 3D rígidas, representa (t1, t2, t3, R1, R2, R3). Para transformações 3D de similaridade, adiciona a escala, (t1, t2, t3, R1, R2, R3, s)[31]

Utiliza features FAST. Possui a habilidade de estimar *poses* de *features* 2D e 3D, o que o torna robusto a movimentos de rotação pura. A triangulação das *features* em pontos 3D é adiada até que *keyframes* com *baseline* suficiente esteja disponível. São estimadas três tipos de poses, rotação pura, matriz essecial e 6DoF. Se *features* 3D suficientes forem observadas, pose completa é estimada (rotação, direção de translação e escala). Se apenas *features* 2D são observadas com *parallax* suficiente, a matriz essencial é estimada (aqui a rotação e direção do movimento são conhecidos, mas a escala não). Se *features* 2D sem *parallax* suficiente, assume rotação pura sem mudança do centro da câmera. Utiliza RANSAC para estimação da pose, sendo obtida minimizando erros de *features* casadas em *frames* diferentes. Procura minimizar os erros de reprojeções 3D-2D e casamentos 2D-2D. Para obter a matriz essecial é usado o algoritmo de 5 pontos.

Gerencia múltiplos mapas. Junção *online* de regiões do mapa desconectadas. Devido ao problema de rotações puras não restringirem a escala do mapa, foi proposto um método que rastreia e mapeia tanto *features* trianguladas (3D) quanto não-trianguladas (2D). Pelo mapa conter *features* não-trianguladas, *keyframes* de maior contribuição podem ser adicionados no mapa. Gerencia múltiplos mapas de escala indefinida com fusão quando há casamento de pontos 3D suficientes. Consegue reinicializar mapas locais, sendo a fusão dos diversos submapas capaz de gerar um mapa de escala uniforme. O mapa contém tanto *features* 2D quanto *landmarks* 3D, em que a triangulação de *features* 2D em *landmarks* 3D ocorre por duas vistas quando há *parallax* suficiente.

Uma terceira *thread* otimiza o mapa com GBA esparso. Um loop é detectado quando casamentos de *features* entre frames não relacionados são suficientes.

A grande vantagem é que gerencia o problema inerente de escala indefinida ao criar múltiplos submapas e uní-los quando *links* são descobertos, são robustos a rotações puras.

DPPTAM (2015)

Método direto que estima a reconstrução densa da cena em tempo real usando CPU. Utiliza minimização de erros fotométricos em diferentes visões. É assumido que regiões de cor homogêneas pertencem a regiões aproximadamente planares. Dividido em três *threads*. A primeira realiza o rastreamento, a segunda a extração de *keyframes* e mapeamento semi-denso, e a terceira superpixels 3D e mapeamento denso. Método similar ao LSD SLAM.

No rastreamento, modelo de velocidade constante é usado. Se os erros fotométricos forem grandes, utiliza a *pose* do último *frame*.

A criação do mapa minimiza o erro fotométrico entre *pixels* de grande gradiente no último *keyframe*. Ao fim, são geradas dez hipóteses de profundidade para cada dos *pixels* utilizados. Testes escolhem a melhor solução. Modela o ambiente com pontos 3D para áreas de grande gradiente e planos 3D para áreas de baixo gradiente. Assim, é assumido

que áreas da imagem com baixo gradiente fotométrico são praticamente planares. O mapa semi-denso é usado tanto para o rastreamento quanto para estimar as superfícies planares.

Não aplica manutenção do mapa, mas mapas densos explorando propriedades planares do ambiente.

Em relação ao LSD SLAM, utiliza uma terceira *thread* para reconstrução densa usando super-pixels segmentados em cenas planares *indoor*.

DSO (2016)

Modelo direto probabilístico com minimização de erros fotométricos. Método esparso e direto. Otimiza a probabilidade total para todos os parâmetros do modelo, como *poses* de câmeras, parametros intrínsecos, parâmetros geométricos (valores de profundidade inversas). Toma vantagem da calibração fotométrica da câmera, incluindo atenuação das lentes, correção gama e tempos de exposição. Roda em tempo real em CPU.

Otimização contínua dos erros fotométricos numa janela de *frames* recentes, usando um modelo fotométrico calibrado. Usa Windowed Bundle adjustment (WBA). Otimiza o erro total numa janela deslizante usando *Gauss-Newto*n. Variáveis antigas são removidas por marginalização do complemento de *Schur*.

Combina os benefícios da abordagem direta de utilizar reconstrução com todos os pontos em vez de só cantos e a abordagem esparsa com otimização dos modelos dos parâmetros. A imagem de entrada é dividida em vários blocos, sendo pontos de alta intensidade selecionados como candidatos para reconstrução. Utiliza calibração de câmera tanto geométrica quanto fotométrica, permitindo maior acurácia na estimação. A otimização para minimização de erros acumulativos locais remove fatores de erros de perspectivas geométricas e fotométricas.

Como o *DSO* só considera a consistência local do mapa, é considerado *VO*.

2.3.5 ORB SLAM (2015)

O SLAM visual escolhido para este trabalho foi o ORB SLAM. Os principais motivos são listados a seguir:

- Considerado o mais completo na categoria de *feature* monocular[3].
- Boa performance, capaz de rodar em tempo real em CPUs, diferente da maior parte dos métodos diretos.
- Código aberto, com muitos colaboradores[16].

[16] `https://github.com/raulmur/ORB_SLAM2`

- Dentre os três considerados monoculares, de código aberto, com manutenção do mapa e relocalização a nível global e com fechamento de *loop*, LSD SLAM, DT SLAM e ORB SLAM, é o mais recente e que apresentou melhores resultados empiricamente, além de, em seu artigo, ter obtido resultados superiores ao LSD SLAM.

Estrutura básica

O sistema ORB SLAM é composto de três *threads* em paralelo, que são rastreamento, mapeamento local e fechamento de *loop*.

- Inicialização

  Totalmente automática, é realizada com seleção do modelo por matriz de homografia ou fundamental, dependendo se a cena é planar ou não.

  Computa a *pose* relativa entre dois *frames* para triangular um *set* inicial de pontos do mapa. É independente da cena (planar ou não) e de operação humana (selecionar duas cenas com *parallax* suficiente).

  Dois modelos são calculados em paralelo: a homografia (DLT normalizado) assumindo cena planar e a matriz fundamental (algoritmo 8-pontos) assumindo cena não-planar. Uma heurística para seleção entre eles é utilizada.

  Após a seleção do modelo, as hipóteses de movimento são recuperadas e a mais forte é usada para realizar a reconstrução inicial. Ao fim, *Global Bundle adjustment* é realizado para refinar esta reconstrução.

- Rastreamento

  É realizado a cada *frame*.

  O rastreamento no frame atual é guiado pelo do frame anterior usando um modelo de velocidade constante. Se não forem encontrados casamentos, a busca por casamentos no mapa é mais ampla.

  Quando o rastreamento é perdido, transforma o frame em BoW e entra em modo de relocalização global. Para obter a *pose* após relocalização, projeta os *keypoints* de cada *keyframe* candidato nas *features* do *frame* atual para casamento, utilizando RANdom SAmple Consensus (RANSAC) e Perspective-n-Point (PnP). Após estimar a *pose*, projeta o mapa local no *frame* e tenta casar *keypoints* e recuperar a escala.

  O rastreamento cuida também da decisão se o *frame* atual deve se tornar um *keyframe*, devendo ter tido uma mudança visual mínima, ter tido uma boa relocalização e um bom rastreamento.

- Mapeamento local

  Processa os novos *keyframes* adicionados no rastreamento. Realiza Local Bundle adjustment (LBA) para reconstrução do mapa local, que otimiza tanto o *keyframe* que acabou de ser adicionado, os diretamente conectados no grafo de covisibilidade e todos os *keypoints* deles.

  A cada *keyframe*, suas *features* são casadas com as *features* dos *keyframes* diretamente conectados do grafo de covisibilidade para tentar realizar a triangulação. Para aceitar novos pontos, deve haver profundidade positiva em ambas vistas, *parallax*, erro de reprojeção e consistência de escala.

  De tempos em tempos, uma política de corte de *keypoints* remove os menos significantes baseado nas informações do rastreamento.

  Uma política de corte remove também *keyframes* redundantes, em que 90% de seus *keypoints* foram vistos nos três últimos *keyframes* na mesma escala.

- Fechamento de *loop*

  A cada novo *keyframe*, é tentado fechar um *loop* no mapa. Quando é fechado, ocorre alinhamento e fusão dos *keypoints*, e Pose Graph Otimization (PGO) é realizado no Grafo Essencial para a consistência global e correção dos erros de escala. Este alinhamento ocorre com transformações de similaridade, que é propagado para todos os vizinhos do grafo.

  Assim como na relocalização, é utilizado o algoritmo DBoW2, porém com informações do grafo de covisibilidade.

- Relocalização

  Baseada nas *features* ORB, invariante a pontos de vista e iluminação. Quando ocorre perda de rastreamento, a relocalização global é realizada. Uma vez que se localiza novamente, o mapa local visível de keyframes em que a câmera se encontra é ativado. Os pontos do mapa local visível ficam em vermelho no mapa. Casamentos de pontos no mapa local são realizados com reprojeção, permitindo que a *pose da câmera* atual seja otimizada com base no mapa local construído anteriormente.

  Para realizar a relocalização, é utilizado o módulo DBoW2[17], que realiza reconhecimento de lugares com casamentos de *features* usando Bag of Words (BoW), ou vocabulário visual, obtido de forma *offline*. Os candidatos são os *keyframes* do grafo de covisibilidade com mais de 75% de casamento.

[17] `https://github.com/dorian3d/DBoW2`

Componentes

Os principais componentes do ORB SLAM são listados abaixo.

- *Features ORB*

  As mesmas *features* são utilizadas para o rastreamento, mapeamento, relocalização e fechamento de *loop*, tornando o sistema simples e rápido.

- *Keypoints*

  São os pontos triangulados do mapa. Cada um é composto de posição 3D global, um vetor direção de onde foi visto (média de todos os keyframes que viram), o descritor/assinatura ORB, distância mínima e máxima que pode ser observado de acordo com o ORB.

- *Keyframe*

  Frames que representam a localização da câmera e que compõe os grafos. É composto de *pose*, transformação entre os sistemas do mundo e da câmera, parâmetros intrínsecos da câmera, coordenadas sem distorção de todas as *features* ORB vistas no *keyframe*.

- Grafo de covisibilidade

  Grafo que permite que rastreamento e mapeamento sejam realizados numa área local, independente do tamanho global do mapa. Grande fator que permite operação em larga escala e em tempo real[33].

  Grafo com pesos não-direcionado. Cada nó é um *keyframe*. Uma aresta existe se ambos *keyframes* compartilham *keypoints*, ou pontos no mapa, sendo o peso da aresta o número de pontos compartilhados.

- Grafo Essencial

  Um subgrafo esparso do grafo de Covisibilidade. Contém todos os nós (*keyframes*), mas somente algumas arestas. Quando um *loop* é fechado, é realizado PGO[18] para corrigir o grafo.

  É composto a partir de uma *Spanning Tree*, atualizada sempre com inclusão e exclusão de *keyframes*, das arestas de maior peso do grafo de covisibilidade e dos *links* de fechamento do *loop*.

  A acurácia do PGO é tão boa que o Global Bundle adjustment (GBA) realizado em seguida pouco corrige o grafo.

[18] `https://github.com/RainerKuemmerle/g2o`

A Figura 2.1 a seguir, tirada de [1], sumariza as etapas e componentes do ORB SLAM citadas aqui.

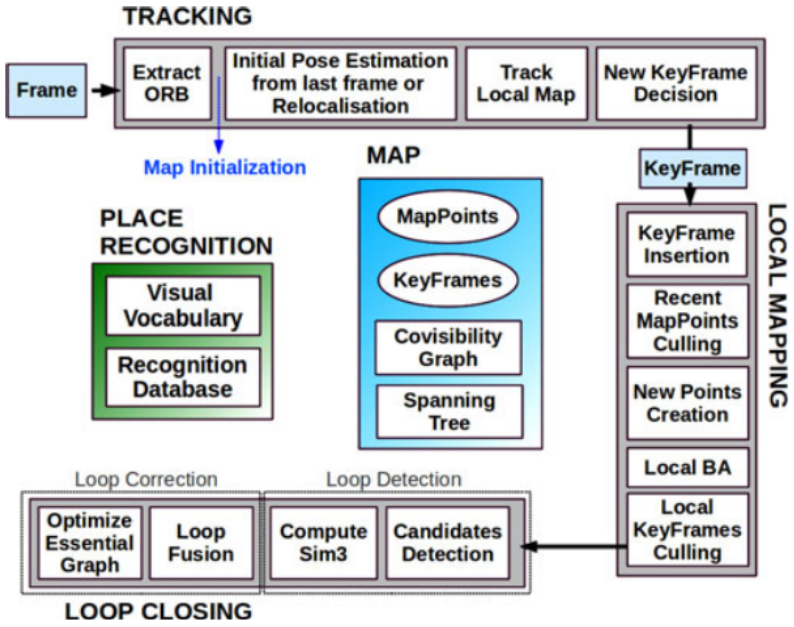


Figura 2.1: Etapas do ORB SLAM. Fonte: [1].

	M	PI	MG	CC	LC	MM	RF	DL
MonoSLAM[34](2007)	filtro	features	esparso	monocular	open	desconhecido	desconhecido	desconhecido
PTAM[35](2007)	keyframe	features	esparso	monocular	open	global	global	não
DTAM[36] (2011)	keyframe	aparência	denso	monocular/estéreo	closed	global	global	não
SLAM++[37] (2013)	keyframe	desconhecido	denso	RGB-D	closed	global	global	sim
SVO[38] (2014/2016)	keyframe	híbrido	esparso	monocular	open	local	local	não
LSD SLAM[39] (2014)	keyframe	aparência	semi-denso	monocular	open	global	global	sim
DT SLAM [40] (2014)	keyframe	features	esparso	monocular	open	global	global	sim
ORB SLAM [1] (2015)	keyframe	features	esparso	monocular	open	global	global	sim
DPPTAM [41] (2015)	keyframe	aparência	denso	monocular	open	sem	local	não
DSO [42] (2016)	keyframe	aparência	esparso	monocular	open	local	local	não

Tabela 2.1: Principais métodos de vSLAM e VO - M (Metodologia), PI (Processamento na Imagem), MG (Mapa Gerado), CC(Configuração das Câmeras), LC(Licença do Código), MM(Manutenção do Mapa), RF(Recuperação a Falhas) e DL(Detecção de Loop). Fonte: [2, 3, 4]

Capítulo 3

Ferramentas e Ambiente

3.1 ROS

Para tornar possível a implementação proposta neste trabalho de conclusão, é necessário o uso de algum *framework* que possa abstrair toda a dificuldade de trabalhar com um hardware embarcado, no caso o Pioneer, tanto no acesso a *drivers* de *encoders*, câmera e outros sensores, como para facilitar o controle e a percepção do robô.

Escrever *software* para robôs é difícil principalmente devido aos diferentes tipos de *hardware* existentes, tornando o reúso de código uma tarefa não-trivial [43]. Além do código de alto nível necessário para tarefas como percepção e raciocínio para tomada de decisões, é necessário código a nível de *drivers* específicos do *hardware*, o que exige muito do programador, com várias linhas de código para realizar uma tarefa simples.

Diante disto, *frameworks* são criados com o objetivo de facilitar a interação com estes *hardwares*, aumentando a produtividade do programador comum e o reúso de código. Um destes sistemas é o *Robot Operating System (ROS)*, que obteve uma grande popularidade ao longo dos anos no ramo, principalmente devido a abordagem e arquitetura únicas utilizadas por ele.

O *ROS* possui ênfase em integração de larga escala, *software* robusto, de propósito geral e colaborativo, possuindo uma comunidade de pesquisadores grande e ativa. Um de seus pontos fortes é a comunicação ***peer-to-peer***, em que um número arbitrário de processos, que podem rodar em diferentes sistemas hospedeiros, são conectados em tempo de execução numa topologia de grafo. Outro ponto interessante é o gerenciamento baseado em **ferramentas**, com um *design* de *microkernel* em que diversas pequenas ferramentas constroem e executam várias componentes do sistema, como navegar na árvore de código, visualizar a topologia de conexão *p2p* e outros. Além disso, podemos citar como vantagens do ROS a neutralidade em relação a **linguagens de programação**, com suporte atual em C++, Python, Octave e Lisp, a **modularidade e reusabilidade**, sendo todos os

códigos de *drivers* e algoritmos obtidos de diversas bibliotecas *open-source* autônomas, promovendo um sistema enxuto, e código **livre e *open-source***, disponível publicamente e sem custos para a comunidade.

3.1.1 Arquitetura

A arquitetura do *ROS* é dividida em três níveis de conceitos, que são o **grafo de computação**, o **sistema de arquivos** e o de **comunidade**, [44]. O sistema de arquivos descreve a estrutura básica de um pacote, que é a unidade básica de um projeto no *ROS*. Já o nível de comunidade descreve as formas e ferramentas utilizadas para compartilhar conhecimento e algoritmos entre os desenvolvedores, cujo detalhamento foge do escopo deste trabalho.

Grafo de computação

Neste nível são descritos todos os componentes que se relacionam com a rede *p2p* de processos.

- Nós

  São os próprios processos, programas em execução. São escritos por meio de uma biblioteca cliente do *ROS*, como roscpp ou rospy.

- Master

  Provê o registro de nomes e serviços de tabela de consulta para todos os outros nós. É ele quem torna possível toda a comunicação no *ROS*, com nós, serviços, mensagens e outros.

- Servidor de Parâmetros

  Permite armazenar e alterar parâmetros de configuração dos nós do sistema. Com ele, é possível alterar variáveis que definem o comportamento de um nó enquanto este está em execução.

- Mensagens

  Nós se comunicam uns com os outros por meio de mensagens. Possuem uma estrutura bem definida, com um cabeçalho e *payload*. Existem diversos tipos padrão no *ROS*, sendo possível a criação de tipos customizados pelo usuário.

- Tópicos

Funciona como um barramento de comunicação para as mensagens, sendo que as identifica unicamente por um nome. Quando um nó envia dados por meio de mensagens, ele publica estas mensagens no tópico que as identifica. Para um outro nó receber estas mensagens, ele se inscreve neste tópico cujas mensagens de interesse estão sendo publicadas. Podem haver múltiplos publicadores e inscritos num dado tópico, sendo que um único nó pode publicar e se inscrever em diversos tópicos diferentes. Tópicos são usados para trocar grandes fluxos de dados, sendo necessário uma frequência informada pelo nó tanto no envio quanto recebimento de mensagens.

- Serviços

  Serviços são definidos pelo par de mensagens de **pedido** e **resposta**, sendo usado para troca de informações rápidas quando há um único servidor e um ou mais clientes (um-para-muitos), diferente dos tópicos (muitos-para-muitos). Aqui, um nó expõe um serviço e funciona como um servidor de envio de respostas, enquanto nós clientes se comunicam com ele por meio de pedidos.

- Bags

  Bags são um formato de arquivo em disco usados para armazenar fluxos de mensagens trocadas entre os nós, tais como dados de sensores de *laser* ou câmera. São extremamente úteis pois permitem simular e compartilhar dados obtidos por diferentes robôs para serem executados e testados de forma *offline*, sem a necessidade de um robô físico publicando mensagens todas as vezes que se for testar um algoritmo.

Pacote

É a unidade principal de organização de *software* no *ROS*. Representa um diretório com os arquivos e subdiretórios a seguir.

- include/nome_do_pacote

  Contém os arquivos de cabeçalhos das bibliotecas utilizadas.

- msg/

  Contém os arquivos de mensagens personalizadas do usuário.

- scripts/

  Contém os arquivos de *script* de qualquer linguagem.

- src/

  Contém os arquivos fontes dos nós.

- srv/

  Contém os arquivos de definição de serviços.

- CMakeLists.txt

  Arquivo que define as diretivas de compilação dos nós do projeto, assim como suas dependências externas.

- package.xml

  Arquivo de metadados, contendo nome, versão, descrição e dependências externas de nós do pacote.

3.1.2 Drivers

Para permitir a comunicação com os *drivers* de robôs da família Pioneer, o *ROS* disponibiliza dois nós de interface, sendo o p2os_driver e o ROSARIA.

p2os_driver

Provê *drivers* para robôs que usam a interface Pioneer P2OS/ARCOS, DX ou AT. Diferente do ROSARIA, possui as próprias implementações dos protocolos. Está a mais tempo em disponibilidade, sendo originariamente desenvolvido para robôs Pioneer2. Devido ao maior tempo de uso na comunidade e sua estabilidade, foi escolhido para ser usado no projeto. O parâmetro utilizado de maior relevância é o **pulse**, usado para controlar a frequência de checagem de comandos e desabilitar os motores. Se inscreve no tópico /**cmd_vel**, recebendo mensagens geometry_msgs/Twist de velocidade e publica nos tópicos /**sonar**, p2os_driver/SonarArray com informações das medidas dos sonares frontais e traseiros, /**pose**, nav_msgs/Odometry com informações de odometria, /**tf**, tf/tfMessage com informações de transformações de coordenadas e /**battery_state**, p2os_driver/BatteryState com informações do estado e carga da bateria.

ROSARIA

Fornece interface para a maioria dos robôs móveis fabricados na empresa Adept Mobilerobots, o que inclui Pioneer2, Pioneer3, AmigoBot, PeopleBot e outros, que suportam a biblioteca *open-source* ARIA. Tal biblioteca é chamada diretamente pelo ROSARIA, o que o torna mais complexo de configurar e usar, porém mais portável para diferentes tipos de configurações e robôs. Por possuir uma maior complexidade, possui mais parâmetros, serviços e tópicos relacionados que o p2os_driver. Também se inscreve no /**cmd_vel**, recebendo mensagens geometry_msgs/Twist. Publica nos tópicos /**pose**, nav_msgs/Odometry, /**sonar**,/sensor_msgs/PointCloud, **motors_state**,

std_msgs/Bool com informações do estado dos motores, **<lasername>_laserscan**, sensor_msgs/LaserScan com informações de medidas de laser. Disponibiliza o serviço /**enable_motors**, std_srvs/Empty para habilitar os motores via *software*.

3.1.3 Utilitários

A seguir alguns pacotes e nós bastante utilizados no ROS.

rosbag

Pacote que provê comandos para trabalhar com arquivos ***bag***, tais como leitura e escrita. Os principais comandos são **record**, para gravar o fluxo de mensagens de um determinado tópico em arquivo bag, **info**, para fornecer informações do conteúdo do arquivo, tais como duração, tamanho, quantidade de mensagens capturadas, tipos das mensagens e tópicos e **play**, para executar um arquivo *bag*, simulando a publicação de mensagens do robô real, sendo possível alterar a frequência para ambientes de teste.

camera_calibration

Nó que permite calibrar câmeras monocular ou estéreo por meio de um tabuleiro de xadrez. Com o procedimento, é possível obter os parâmetros intrínsecos, extrinsecos e distorções presenter nas lentes da câmera, possibilitando uma correção precisa por meio de transformações de retificação. Os parâmetros de entrada do nó são o tamanho do tabuleiro, ou quantos pontos de intercessão entre os quadrados brancos e pretos na horizontal e vertical, o tamanho do quadrado e o fluxo de imagens.

teleop_twist_keyboard

Nó que traduz comandos do teclado em mensagens geometry_msgs/Twist de velocidade, sendo possível aumentar e diminuir as velocidades linear e angular do robô em tempo real. Abordagem parecida foi implementada neste trabalho, porém em vez do teclado foi usado um *joystick*.

image_view

Nó para vizualização de mensagens em formato de imagem. O nó captura as mensagens e reproduz um vídeo na frequência da publicação.

3.1.4 Mensagens

Abaixo, é descrito o formato das principais mensagens utilizadas no ROS.

- geometry_msgs/Twist

  Expressa velocidade em coordenadas linear e angular. Cada componente é modelada como um vetor de 3 posições, com velocidade linear nas componentes x, y e z e velocidade angular em roll, pitch, yaw.

- nav_msgs/Odometry

  Estimativa de *pose* no espaço aberto utilizando odometria. A posição é dada por um ponto tridimensional, enquanto a orientação é um *quaternion*. Vem acompanhado de uma matriz de covarância representando o erro da medida.

- sensor_msgs/Image

  Representa uma imagem obtida de uma câmera, sendo que contém os campos altura e largura da imagem, codificação, o tamanho da linha em *bytes* e a matriz de dados da imagem.

- sensor_msgs/LaserScan

  Encapsula uma varredura de leitura de *laser*. Seus campos são o ângulo mínimo e máximo em radianos, a distância angular entre as medidas, o tempo entre as medidas e entre varreduras, as distâncias mínimas e máximas lidos pelo *laser*, um vetor com as distâncias obtidas de cada medida e um vetor das intensidades por medida.

3.1.5 Simuladores

Os simuladores podem ser utilizados para testar aplicações antes de serem utilizadas em operações e ambientes reais. Tal etapa é fundamental para prover robustez e confiabilidade ao código realizado.

Rviz

ROS visualization (Rviz) é a ferramenta *ROS* de visualização 3D [45]. Ao usuário é fornecida uma interface para visualisar o que o robô sente com os seus sensores e suas ações, facilitando enormemente o *debug* da aplicação. Com o Rviz é possível mostrar dados de *laser*, sonar, câmeras monocular e estéreo em formas de nuvens de pontos ou imagens de profundidade.

Gazebo

Gazebo é um ambiente de simulação para o *ROS*, desenvolvido pela Willow Garage [45]. Ele suporta diversas funcionalidades, tais como o *design* de modelos de robôs - em arquivos

URDF- , a simulação de ambientes internos e externos, a simulação de sensores e da física com *engines* de alta performance.

3.2 Pioneer-3AT

3.2.1 Robô

O Pioneer 3-AT é uma plataforma robótica de quatro rodas e quatro motores anti-derrapantes altamente versátil, compatível a nível de *software* com todos os robôs da empresa MobileRobots[1] e usado tanto em ambientes *indoor* quanto *outdoor*. Em sua versão 3-AT, vem de fábrica com uma bateria, um botão de parada de emergência, *encoders* nas rodas, um microcontrolador com o *firmware* ARCOS e a plataforma de desenvolvimento de software Pioneer SDK. Opcionalmente, vêm com um computador embarcado e diversos acessórios, tais como giroscópio e garras robóticas. O nosso robô, apelidado de CACIC, encontra-se no prédio CiC/Est, no laboratório LAICO e pode ser visto na Figura 3.1.

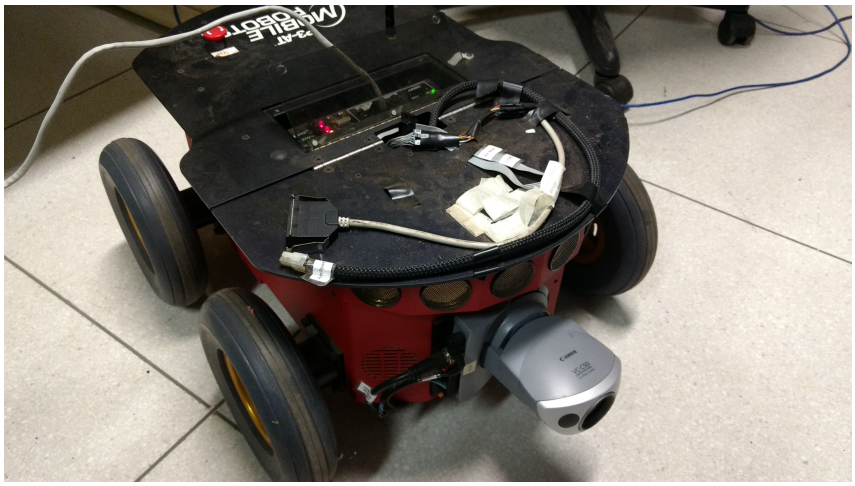


Figura 3.1: Pioneer 3-AT disponível no LAICO, chamado CACIC. Fonte: Autor.

3.2.2 SDK

O Pioneer *Software Development Kit (SDK)* é uma coleção de bibliotecas e aplicações padrão nos robôs de plataforma da MobileRobots. Dentre seus componentes estão a **ARIA**, biblioteca C++ para controle, suporte a programação em rede com *sockets* e cliente-servidor e a acessórios, a **MobileSim**, para simulação, a **MobileEyes**, um cliente

[1]Inicialmente chamada ActivMedia Robotics, teve o nome trocado para MobileRobots Inc e mais tarde comprada pela Adept como Adept Mobilerobots

de *Graphical User Interface (GUI)* para operação remota e monitoramento do robô, a ***Mapper 3-Basic***, para criar e editar mapas e a **SONARNL**, que provê localização e navegação utilizando sonares.

3.2.3 Carregamento das baterias

Em uso contínuo, suporta de duas a quatro horas de trabalho, podendo ser carregado por até três baterias por vez. O tempo de carregamento pode ser de 12 horas em modo padrão ou 2.4 horas em modo de alta capacidade, com fontes de energia de 5V e 1.5A e 12V e 2.5.

3.2.4 Painel de controle do usuário

No painel de usuário, encontra-se um *buzzer* MIDI programável, um indicador de potência, um indicador de carga de bateria, interruptores de energia auxiliares, um botão de *reset* do sistema e um botão para habilitar/desabilitar motores.

3.2.5 Acessórios

Diversos são os acessórios opcionais no Pioneer. Dentre eles, estão disponíveis no CACIC o sensor *laser* Hokuyo UTM-30LX[2] e a câmera de rede monocular Canon VC-C50i[3], principal componente usado neste trabalho. Dentre suas especificações, possui 460 linhas de TV na horizontal e 350 na vertical, que convertendo para pixels resulta em torno de 640 de largura por 480 de altura, que foi a resolução utilizada. A Figura 3.2 ilustra a câmera utilizada pelo CACIC.

[2] `https://www.hokuyo-aut.jp/search/single.php?serial=169`
[3] `https://www.canon.ie/for_work/business-products/network-cameras/vc-c50i/`

Figura 3.2: Modelo da câmera disponível no CACIC e utilizada no trabalho, Canon VC-C50i. Fonte: Canon.

3.2.6 Computador embarcado

Seu computador embarcado opcional é totalmente integrado, preinstalado e configurado pelo usuário, podendo utilizar tanto sistemas Ubuntu Linux ou Windows 7, com *drivers* e *software* prontos para uso imediato.

3.3 Sistema computacional utilizado

Para o presente trabalho, foi usado tanto o computador embarcado do CACIC, chamado aqui de **embarcado** ou **local**, quanto o notebook de uso pessoal, chamado de **base**, terminologia esta utilizada pelo ROS e adotada aqui.

O sistema instalado no CACIC é o Ubuntu 12.04.5 LTS, de codenome *Precise*. Já o notebook de uso pessoal tem instalado o Ubuntu 14.04.5 LTS, de codenome *Trusty*. Tais informações foram obtidas com o comando **lsb_release -a**. O sistema ROS do local e do base são Hydro [4] e Indigo [5], respectivamente. Ambos foram obtidos com **rosversion -d**.

As informações de *hardware* de ambos os sistemas foram fornecidas com o software **HardInfo** [6], obtido com o comando **sudo apt-get install hardinfo**, e são mostradas na Tabela 3.1. Os dados de espaço de disco livre são após obter a maior parte dos resultados em fluxo de vídeo*.

Além das informações do sistema em geral, foi de extrema importância obter informações dos adaptadores de rede, wireless e usb, mostradas na Tabela 3.2. Para obter

[4] `http://wiki.ros.org/hydro`
[5] `http://wiki.ros.org/indigo`
[6] `https://github.com/lpereira/hardinfo`

Tabela 3.1: Informações do computador embarcado no CACIC (local) e do notebook pessoal (base) obtidas com o software HardInfo. Fonte: Autor.

Pioneer	
Processador	Intel(R) Pentium(R) M processor 1.80GHz
Cache	2048kb
Frequência	1793.37MHz
Memoria	470MB (256MB used)
Sistema	Ubuntu 12.04.5 LTS
Resolução	640x480 pixels
Disco	36.6 Gib de 53.1 GiB livres
Notebook	
Processador	4x Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
Cache	3072kb
Frequência	2340.00MHz
Memoria	8083MB (4946MB used)
Sistema	Ubuntu 14.04.5 LTS
Resolução	1366x768 pixels
Disco	89.4 Gib de 409.9 GiB livres

informações de rede foram utilizados os comandos **lspci | grep -i net** e **sudo lshw -C network**. Já para obter as informações de USB foi usado **lsusb**. Para o *Wireless*, vale notar que o suporte oferecido pelo Pioneer é **a**, **b** e **g**, enquanto o do notebook é **b**, **g** e **n**. No *Ethernet*, o local suporte apenas *Fast*, enquanto o base *Fast* e *Gigabit*. No *USB*, versões 1.1 e 2.0 contra 2.0 e 3.0.

3.4 Edifício CiC/Est

O Departamento de Ciência da Computação (CIC) está localizado no Campus Universitário Darcy Ribeiro, edifício CIC/EST, compartilhado com o Departamento de Estatística. Na Figura 3.3 é mostrada a planta baixa simplificada do primeiro andar do CiC/Est. Em verde, temos as áreas abertas, R1 a R6 são as salas de reunião. Os demais espaços sem identificação são as salas dos professores, de estudos e afins. Por motivos de segurança do operador e do robô, foi escolhido o primeiro andar para a realização dos experimentos por possuir uma divisão entre as áreas fechada e aberta.

Tabela 3.2: Informações de rede e usb dos computadores local e base. Fonte: Autor.

Pioneer	
Wireless	Qualcomm Atheros AR5413/AR5414
	Wireless Network Adapter [802.11abg] (rev 01)
Ethernet	Intel Corporation 8255xER/82551IT
	Fast Ethernet Controller (rev 10)
USB	2x Linux Foundation 1.1 root hub
	1x Linux Foundation 2.0 root hub
Notebook	
Wireless	Qualcomm Atheros QCA9565/AR9565
	Wireless Network Adapter [802.11bgn] (rev 01)
Ethernet	Realtek Semiconductor Co., Ltd. RTL8101/2/6E PCI Express
	Fast/Gigabit Ethernet controller (rev 07)
USB	2x Linux Foundation 2.0 root hub
	1x Linux Foundation 3.0 root hub

Figura 3.3: Planta baixa simplificada do 1º andar do CiC/Est. Fonte: Autor.

Capítulo 4

Metodologia

4.1 Fluxo de trabalho

O sistema computacional utilizado neste trabalho é composto do computador embarcado do Pioneer, assim como o sistema do próprio robô, e do notebook de uso pessoal, cuja especificação técnica de ambos está apresentada no capítulo anterior.

Como pode ser visto na Figura 4.1, o computador base é responsável por executar o nó de inicialização do ROS e os algoritmos de: ORB SLAM, de captura dos sinais de movimento do *joystick* e de salvamento em disco das mensagens referentes ao fluxo de imagens da câmera, que chegam a partir do robô por meio do cabo *Ethernet*. O computador embarcado executa seu nó de comunicação com o ROS e o algoritmo de obtenção das imagens da câmera, além de receber as mensagens provenientes de comandos de joystick do computador de base pelo cabo *Ethernet* e traduzir estas mensagens para comandos de controle dos atuadores das rodas.

Diversos foram os motivos para este sistema com dois computadores compartilhando informações em vez de utilizar apenas o computador local. Dentre eles:

1. A tela do notebook permite ver em tempo real o mapa sendo gerado. Caso contrário, seria necessário um monitor e uma fonte de energia que acompanhasse o robô em movimento.

2. A necessidade de salvar o fluxo de vídeo para que os resultados aqui apresentados pudessem ser gerados novamente. Após vários testes, foi observado que salvar o fluxo de vídeo no sistema local causava lentidão e travamento do sistema, impedindo o processamento em tempo real necessário para o projeto.

3. A execução do algoritmo ORB SLAM também apresentou lentidão e travamento do sistema no computador local.

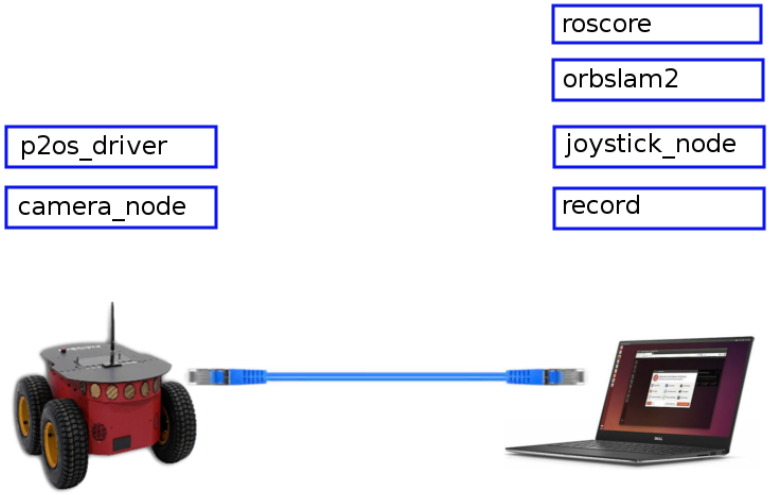


Figura 4.1: Ambiente de trabalho com compartilhamento de informações entre o Pioneer e o notebook pessoal. Fonte: Autor.

Diante dos itens acima explicitados, o sistema foi balanceado de forma a deixar a maior carga de processamento no computador de base, enquanto o computador embarcado do pioneer apenas ficou responsável por prover o fluxo de vídeo do sistema e responder aos comandos de controle do *joystick*. Apesar desta configuração resolver grande parte dos problemas e tornar possível o andamento do projeto, a sobrecarga ficou na comunicação entre os dois computadores. É necessário transferir o fluxo de imagens do local para o base numa taxa próximo de 30 fps para assegurar o processamento em tempo real, além de transferir os comandos do *joystick* do base para o local de forma a minimizar o atraso entre o comando e resposta do robô.

A Figura 4.2 ilustra o fluxo de trabalho temporal com os códigos utilizados para preparar o sistema para a operação de captura dos dados e execução do *vSLAM*. A seta direcionada na vertical indica o espaço temporal, que começa no topo superior e acaba na ponta da seta. Cada seta horizontal indica a utilização de um terminal, sendo a mais curta utilizada no computador base e a mais longa no computador embarcado do Pioneer. Os textos em negrito são ações a serem tomadas independentes do sistema de operação, seja base ou local. A seta pontilhada indica um comando opcional: no exemplo usado para checar a taxa de frames obtidos da câmera. Assume-se que as configurações de rede e de calibração de câmera, que devem ser realizadas uma única vez, já foram executadas. Tais comandos serão citados/explicados nas seções que seguem.

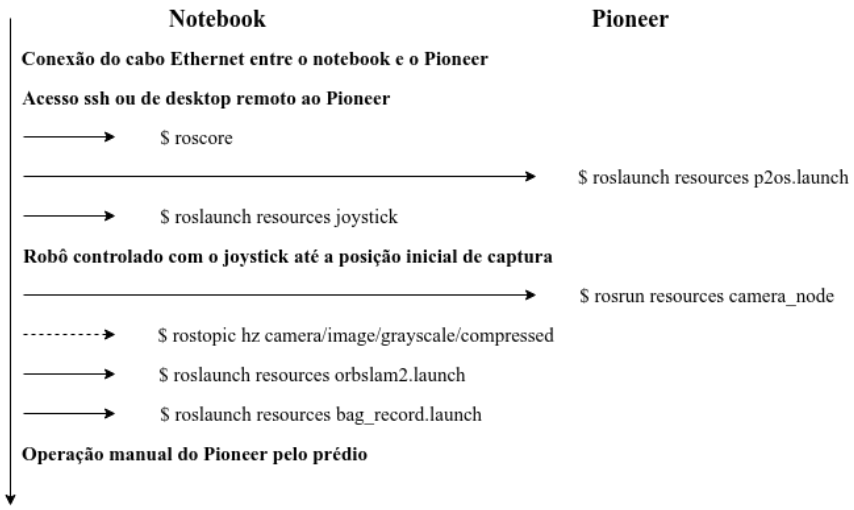


Figura 4.2: Fluxo de trabalho temporal com os comandos em linhas de terminal utilizados. Fonte: Autor.

4.2 Comunicação

4.2.1 Avaliação dos meios para troca de informação

Como foi dito anteriormente, o sistema de comunicação para balanceamento da carga de execução entre o notebook e o robô resolveu apenas parte dos problemas. Tornou-se necessário um meio eficiente de troca de informações capaz de entregar um fluxo de vídeo em tempo real do local para a base e troca de informações do *joystick* no sentido inverso.

Para decidir o melhor meio de troca de informações, foi realizado um estudo dos principais disponíveis no mercado: o cabo Ethernet, a rede Wireless e o cabo com porta USB. Além deste estudo, foi estimado, na Equação 4.1 a seguir, a quantidade de Megabytes por segundo (MBps) necessários para transferir um fluxo de vídeo em tempo real, 30 fps a cores, desconsiderando o tamanho do cabeçalho do protocolo de rede utilizado. Considerando que cada pixel precisa de 3 bytes, um para cada canal, que cada imagem tem 640 de largura por 480 de altura e que é desejável 30 *frames por segundo*, é necessário a transferência de, no mínimo, 27,64 Megabytes por segundo.

$$3B/pixel * 640 * 480 * 30fps = 27648000Bps = 27,65MBps \tag{4.1}$$

A tabela Tabela 4.1 ilustra os principais meios de comunicação, assim como suas versões, e suas velocidades máximas de transferência. Diante da tabela, e tendo por base as tecnologias suportadas por cada plataforma como mostrado na Tabela 3.2, as melhores opções para troca de dados seriam o USB 2.0, seguido do Fast Ethernet e do

Tabela 4.1: Principais meios de comunicação com suas velocidades máximas de transferência permitidas em Megabytes por segundo (MBps). Fonte: [5, 6, 7].

Ethernet	
Fast	12,5MBps
Gigabit	125MBps
Wifi 802.11	
a	6,75MBps
b	1,375MBps
g	6,75MBps
n	37,5MBps
USB	
1.1	1,5MBps
2.0	60MBps
3.0	640MBps

Wifi 802.11g, de forma que apenas o USB seria capaz de transferir os 27,65 MBps como estimado na Equação 4.1. Apesar do computador de base suportar as tecnologias mais velozes e recentes, como o USB 3.0, Gigabit Ethernet e Wifi 802.11n, ficava restrito a utilizar apenas as tecnologias suportadas também pelo computador local, embarcado no Pioneer.

Embora a opção USB 2.0 ter sido a única viável de entregar dados na velocidade requerida, foi preferido utilizar a tecnologia *Fast Ethernet* para comunicação do fluxo de dados e encontrar uma forma de diminuir a taxa necessária para o fluxo de imagens. Alguns dos motivos para preferir a abordagem *Ethernet* em razão da USB 2.0 e Wifi 802.11g foram:

1. Dificuldade/impossibilidade de transferir dados de rede pela porta USB, requisito necessário para a comunicação distribuída do sistema ROS, que usa por padrão o *Transmission Control Protocol (TCP)*. Além disso, era necessário um cabo USB macho-macho unido por um circuito que permitisse a transferência, caso contrário poderia danificar ambos os sistemas.

2. A facilidade de configurar uma rede local Ethernet em face da rede local Wifi, além da rede *Ethernet* ser mais rápida. Além disto, a rede local *Ethernet* possui mais estabilidade por não depender da distância do dispositivo, ao contrário do que acontece com o roteador Wireless. Uma forma de contornar isto seria usar o roteador acoplado no robô, porém seria necessário uma fonte de energia para o roteador, assim como aconteceria no caso do monitor para visualizar o mapa em tempo real.

4.2.2 Compressão e fluxo em níveis de cinza

Como pode ser visto pela Figura 4.3, a taxa de frames recebidos via cabo *Ethernet* sem nenhuma medida para diminuir a taxa MBps requerida apresentou um valor estável por volta dos 13 fps.

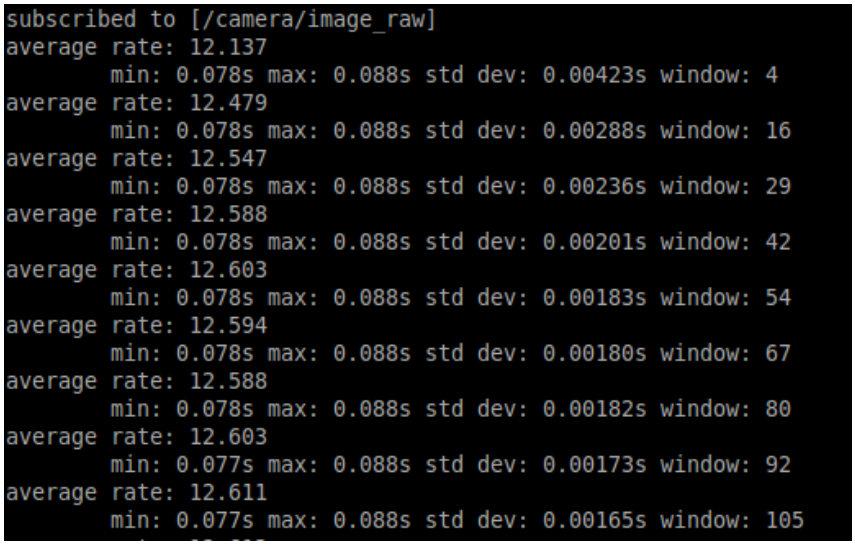


Figura 4.3: Taxa de quadros recebidos via Ethernet transferindo imagens cruas. Comando ***rostopic hz /camera/image_raw***. Fonte: Autor.

Transferência de imagens em níveis de cinza

Para relaxar a necessidade de transferir 27,65 MBps e assim melhorar a taxa de *frames* de entrada do ORB SLAM, foi proposta a transferência de imagens em níveis de cinza, já usada pelo algoritmo ORBSLAM, que em seu interior faz a conversão de imagens em cores para tons de cinza.

Para realizar este passo, o nó ***camera_node*** executado pelo Pioneer foi adaptado de forma a utilizar o método ***cvtColor*** da biblioteca ***OpenCV*** para transformar imagens coloridas para tons de cinza. Além disto, foi usado o pacote ***cv_bridge*** do ROS para encapsular esta nova imagem como uma mensagem de rede e adicionar em seu cabeçalho que se trata de uma imagem com apenas um canal de cor. Para manter a coerência da nomenclatura dos tópicos do ROS, esta nova mensagem foi renomeada de /camera/image_raw para /camera/image/grayscale.

Diante desta modificação, a Equação 4.1 pode ser reescrita como Equação 4.2, diminuindo a taxa de dados em três vezes. Com isto, a taxa estimada ficou abaixo da taxa máxima de transferência permitida no cabo *Ethernet*. Mesmo assim, o fps observado ainda não ficou estável em 30 fps. Isso se deve ao fato de que a estimativa não leva

em consideração os tamanhos dos cabeçalhos das mensagens, as mensagens de controle e fatores externos de processamento.

$$1B/pixel * 640 * 480 * 30fps = 9216000Bps = 9,22MBps \tag{4.2}$$

Transferência de imagens comprimidas

Para suavizar ainda mais a dependência com a taxa de transferência, as imagens já em níveis de cinza foram comprimidas em JPEG.

Esta compressão foi realizada por meio do pacote ***image_transport*** do ROS utilizando o plugin ***compressed_image_transport***. Este plugin permite definir o tipo da compressão, que pode ser JPEG ou PNG, sendo JPEG o padrão, além de definir o quanto da qualidade original da imagem deve ser mantida em razão do ganho em armazenamento, no caso tamanho de transferência, sendo o padrão de 80%. Mantendo a qualidade da image no valor padrão de 80%, e fazendo uma estimativa que o tamanho da imagem diminui em cerca de 25%[46], a Equação 4.2 pode ser reescrita como Equação 4.3, que resulta num valor de aproximadamente metade da taxa de transferência máxima permitida pelo cabo *Ethernet*. Novamente, para manter a coerência dos nomes nos ROS, o novo tópico foi chamado de /camera/image/grayscale/compressed.

Com esta última adaptação, é possível observar na Figura 4.4 que o *fps* recebido finalmente se manteve estável por volta de 30 *fps*.

$$0.75(1B/pixel * 640 * 480) * 30fps = 6912000Bps = 6,91MBps \tag{4.3}$$

4.2.3 Configurações de comunicação distribuída

A plataforma *ROS* permite o processamento distribuído em um sistema de computadores conectados, permitindo assim que a complexidade de projetos possa ser dividida entre *hardwares* diferentes de acordo com o seu poder de processamento. Como dito anteriormente, o computador embarcado do Pioneer3AT usado em laboratório mostrou-se ineficiente na execução completa dos algoritmos necessários. A solução encontrada para o problema foi utilizar o sistema distribuído disponibilizado pelo *ROS* entre o computador de base, notebook pessoal e o computador local, embarcado no Pioneer, que funciona por meio do compartilhamento do servidor ***Master***. Para compartilhar o *Master* e, por consequência, os mesmos Tópicos, Parâmetros e Mensagens é necessário que a máquina cliente conheça a máquina servidora na rede. Foi decidido utilizar a máquina de base como servidora do *Master* e a máquina embarcada no Pioneer como a cliente, deixando a maior carga de processamento no computador pessoal.

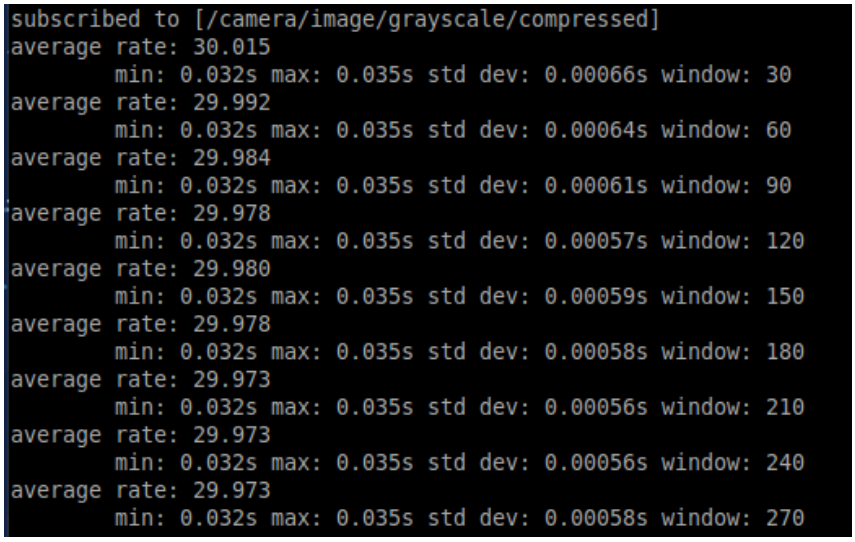


Figura 4.4: Taxa de quadros recebidos via Ethernet transferindo imagens em níveis de cinza e comprimidas. Comando ***rostopic hz /camera/image/grayscale /compressed***. Fonte: Autor.

1. Tornar máquina servidora conhecida pelo Pioneer

   Para a máquina cliente conhecer a servidora na rede é necessário adicionar uma linha no arquivo /**etc**/**hosts** com as definições de *IP*, nome de hospedeiro e *alias* do computador de base. O motivo de se ter escolhido **10.0.0.1** como o *IP* será explicado na subseção a seguir.

$$10.0.0.1\, gabrielPC\, gabrielPC \tag{4.4}$$

2. Definir servidor do Master no cliente

   Sendo a máquina servidora conhecida pela cliente, esta deve definir a servidora como aquela que irá iniciar o servidor *Master*. Para isto, foi necessário exportar na máquina cliente a variável de ambiente **ROS_MASTER_URI** com o valor da máquina responsável por lançar o *Master* e a porta de comunicação, sendo a porta 11311 padrão de utilização no *ROS*.

$$export\, ROS\_MASTER\_URI = http://gabrielPC : 11311 \tag{4.5}$$

4.2.4 Conexão local com cabo ethernet

Para facilitar a operação com o robô, foi criada uma rede local à cabo *Ethernet* com o serviço de *desktop* remoto utilizando o software Remmina Remote Desktop Client[1], padrão em sistemas Linux.

Rede local Ethernet

Para permitir a comunicação entre as duas máquinas foi utilizada uma rede local, eliminando todo o tráfego desnecessário de hospedeiros externos. A rede foi configurada de forma manual em ambos os computadores, sendo utilizado **Ipv4**, máscara de rede **255.255.255.0** e *gateway* **0.0.0.0**. Os *IPs* foram definidos de forma fixa, sendo **10.0.0.1** para a máquina base e **10.0.0.2** para a máquina local. A rede foi chamada de **Direct to Pioneer** e **Direct to Notebook**, respectivamente, nas máquinas base e local.

Acesso remoto ao Pioneer

Para facilitar o acesso ao computador embarcado do Pioneer, foi realizado acesso remoto a partir do computador de base.

1. Permitindo acesso remoto no Pioneer

   Por meio do software *Desktop Sharing*, no computador local, foi ativada a opção de compartilhamento de desktop via senha, escolhida como ***pioneer***.

2. Realizando o acesso

   No computador de base, por meio do software *Remmina Remote Desktop Client*, foi realizado o acesso remoto utilizando o protocolo *Virtual Network Computing (VNC)* e definindo o *IP* de acesso como **10.0.0.2** e a senha ***pioneer***.

A partir deste momento, todo o sistema embarcado no Pioneer está disponível para acesso remoto no computador de base.

4.3 Calibração da câmera

O algoritmo ORB SLAM utilizado como base, assim como os atuais algoritmos de vSLAM existentes, necessitam como entrada os parâmetros de calibração da câmera utilizada, que são as distâncias focais das lentes, **fx** e **fx**, os centros ópticos expressos em coordenadas de *pixels*, **cx** e **cy** e os coeficientes de distorção das lentes, **k1**, **k2**, **p1**, **p2** e **k3**. Estes

[1] `https://www.remmina.org/wp/`

coeficientes são utilizados pelo algoritmo para fazer as transformações de coordenadas e retificações dos pontos chaves da imagem da câmera para o mapa 3D sendo criado.

Para obter estes parâmetros foi utilizado o pacote **camera_calibration**[2] disponibilizado pelo ROS. O método de calibração necessita de um tabuleiro de xadrez com dimensões conhecidas, usadas como entrada do nó, que são o número de pontos de intercessão entre os quadrados negros na horizontal e na vertical, além do tamanho de cada quadrado em metros. Assim, o algoritmo de calibração procura identificar estes pontos de intercessão no tabuleiro dados estes parâmetros de entrada.

Para que a calibração seja bem sucedida, é necessário apresentar o tabuleiro em diversas configurações de posição, escala e inclinação no campo de visão do fluxo de vídeo. Após obter dados suficientes de configurações do tabuleiro, os cálculos para realizar a calibração da câmera podem ser realizados e os parâmetros intrínsecos, as distâncias focais e centros ópticos, e extrínsecos, coeficientes de distorção, podem ser obtidos.

O tabuleiro utilizado foi obtido de um link oficial da biblioteca Opencv[3], impresso e fixado num suporte rígido de papelão para evitar que dobrasse durante o procedimento. A Figura 4.5 mostra o tabuleiro utilizado no processo de calibração da câmera do Pioneer.

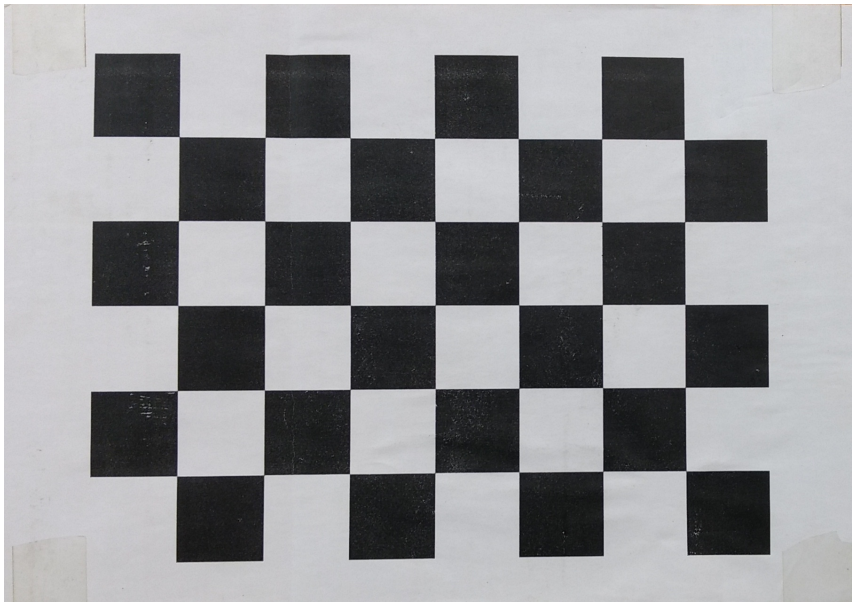


Figura 4.5: Tabuleiro utilizado para calibrar a câmera do Pioneer. As dimensões utilizadas foram de 7x5 intercessões e 0.108 metros por quadrado. Fonte: Autor.

A Figura 4.6 ilustra o procedimento para realizar a calibração do Pioneer. Vale notar que o fluxo de vídeo utilizado possui as mesmas características do utilizado durante o

[2] `http://wiki.ros.org/camera_calibration`
[3] `https://docs.opencv.org/2.4.9/_downloads/pattern.png`

Tabela 4.2: Parâmetros obtidos como resultado da calibração da câmera do Pioneer. Fonte: Autor.

fx	828.357771
fy	857.633796
cx	311.337125
cy	226.366846
k1	-0.207202
k2	0.694100
p1	-0.006168
p2	-0.002871
k3	0.000000

procedimento do ORB SLAM para obter os parâmetros mais acurados possíveis, ou seja, em níveis de cinza e comprimido em JPEG.

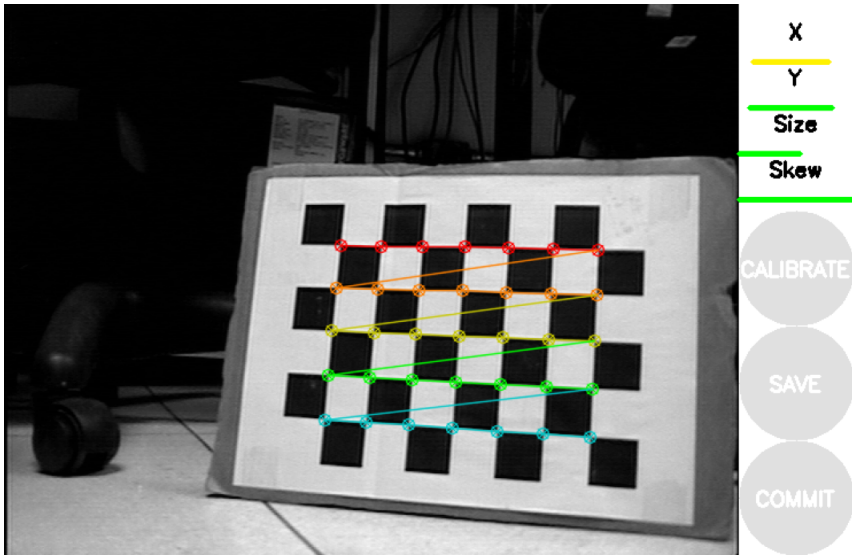


Figura 4.6: Procedimento usado para calibração da câmera do Pioneer utilizando o pacote camera_calibration do ROS. Várias configurações de posição, escala e inclinação do tabuleiro (indicadas pelas barras X,Y,*Size* e *Skew*) são necessárias para obter precisão no cálculo dos parâmetros. Fonte: Autor.

A Tabela 4.2 mostra os parâmetros obtidos da calibração e usados como entrada para o algoritmo ORB SLAM.

4.4 Controle

Para controlar o Pioneer durante a execução do ORB SLAM foi utilizado um *joystick* simples de computador com conexão USB da marca de eletrônicos ***dazz***[4], como pode ser visto em Figura 4.7. Vale notar que alguns controles USB não são reconhecidos pelo ROS. Ao tentar utilizar um *joystick* similar de outra marca, os movimentos dos eixos do controle não foram reconhecidos pelo sistema e portanto não puderam ser mapeados.

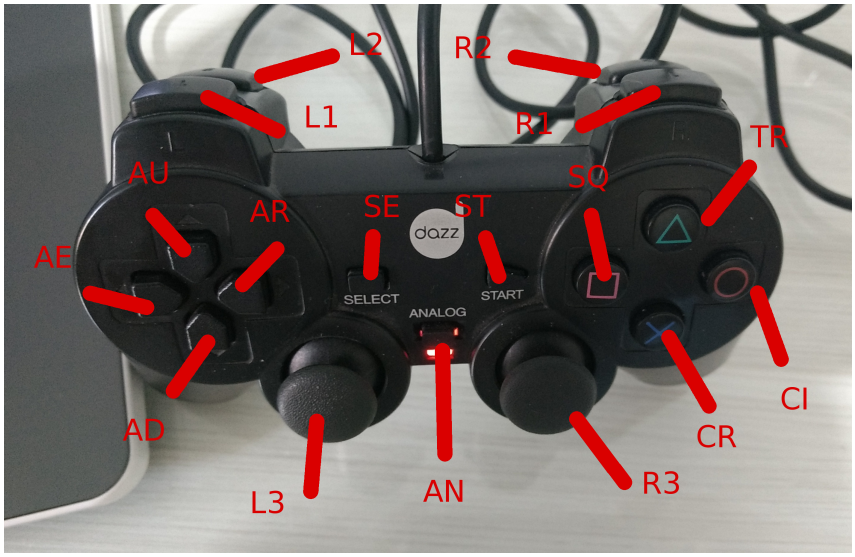


Figura 4.7: Joystick utilizado para controlar o Pioneer. Cada botão e eixo foi nomeado para facilitar o entendimento. Fonte: Autor.

Para permitir o interfaceamento entre o *joystick* e o ROS foi utilizado o pacote ***joy*** [5]. O nó ***joy_node*** deste pacote tem como parâmetros de entrada o dispositivo de entrada de sinais, */dev/input/js0*, a zona morta dos eixos, escolhido como *0.3* e frequência de reenvio de mensagens, *20* e publica mensagens recebidas do controle no formato de dois vetores, um chamado *axes*, que contém seis posições com valores *float* que vão de -1.0 a 1.0, e indicam movimentações nos dois eixos móveis do joystick, **R3** e **L3**, e nas quatro setas a esquerda, **AU**, **AR**, **AD**, **AL**, e o outro chamado *buttons*, que contém doze posições podendo ser 0 ou 1, e indicam o pressionamento de todos os botões do joystick. As Tabelas 4.3 a 4.4 a seguir mostram as associações entre os vetores, botões e eixos. Para todos os casos, o valor 0 indica sem ação e é o valor padrão.

[4] `http://www.dazz.net.br/Produto/621322/controle-dual-shock`
[5] `http://wiki.ros.org/joy`

Tabela 4.3: Associação entre botões e índices do vetor *buttons*. Fonte: Autor.

Botão	**Índice**
TR	0
CI	1
CR	2
SQ	3
L1	4
R1	5
L2	6
R2	7
SE	8
ST	9
L3	10
R3	11

Tabela 4.4: Associação entre eixos/setas e índices do vetor *axes*. Fonte: Autor.

Botão	**Ação**	**Índice**	**Valor**
L3	esquerda	0	(0.0, 1.0]
L3	direita	0	(0.0, -1.0]
L3	cima	1	(0.0, 1.0]
L3	baixo	1	(0.0, -1.0]
R3	esquerda	2	(0.0, 1.0]
R3	direita	2	(0.0, -1.0]
R3	cima	3	(0.0, 1.0]
R3	baixo	3	(0.0, -1.0]
AL	pressionado	4	1.0
AR	pressionado	4	-1.0
AU	pressionado	5	1.0
AD	pressionado	5	-1.0

Tabela 4.5: Mapeamento do *joystick* para botões de ativação. Fonte: Autor.

Botão	**Movimento**	**Velocidade máxima (m/s)**
L2 (comum)	linear	0.1
L2 (comum)	angular	0.1
R3 (turbo)	linear	0.7
R3 (turbo)	angular	0.2

Tabela 4.6: Mapeamento do *joystick* para controle do robô. Fonte: Autor.

Botão	**Ação**	**Direção**
L3	esquerda	vel angular esquerda
L3	direita	vel angular direita
L3	cima	vel linear cima
L3	baixo	vel linear baixo

4.4.1 Associação de comandos de movimento

Por meio do tópico ***cmd_vel*** do ROS, que entende mensagens do tipo ***geometry_msgs/Twist***, é possível controlar as rodas do Pioneer por meio de comandos de velocidade lineares e angulares. A mensagem é composta de dois vetores de três posições, ***linear*** e ***angular***, que controlam as velocidades linear nos eixos x,y e z e angular, também nos eixos x,y e z, ou *roll*, *pitch* e *yaw*, respectivamente. Como o Pioneer tem liberdade de movimento de translação apenas no eixo x, frente e trás, e de rotação apenas no eixo z, *yaw*, foram usadas apenas estas posições dos vetores.

Foi criado o nó ***joystick_node*** com o objetivo de realizar o mapeamento entre os comandos do *joystick* recebidos pelo nó *joy_node* e os comandos de velocidade entendidos pelo Pioneer.

Para manter a segurança do operador e do robô durante a operação, foram utilizados dois botões de ativação do movimento, chamados aqui de **comum** e **turbo**. Ao pressionar o botão de ativação comum, tanto a velocidade de translação do robô quanto de rotação são no máximo 0.1 metros por segundo. No caso do botão turbo, a velocidade de translação é de 0.7 m/s, enquanto a de rotação é de 0.2 m/s. Tais valores são explicitados na Tabela 4.5. O botão comum foi escolhido como o **L2**, enquanto o turbo o **R2**. Tais valores de velocidade podem ser ajustados em arquivo de configuração que acompanha o projeto e foram escolhidos de forma empírica afim de proporcionar maior conforto na operação do robô.

A Tabela 4.6 mostra o mapeamento usado entre os comandos do *joystick* e de controle de velocidade do robô. É interessante notar que foi usado apenas o eixo **L3** para todas as formas de movimento. O cálculo da velocidade em cada direção é a velocidade máxima de ativação vezes o valor do eixo L3.

Tabela 4.7: Limites de movimentação em cada eixo da câmera. Fonte: Autor.

Eixo	**Limites (graus)**
pan	-97 a 97
tilt	-29 a 87
zoom	0 a 1960

Tabela 4.8: Mapeamento de botões do *joystick* para controle da câmera. Fonte: Autor.

Botão	**Eixo**	**Sentido**
TR	tilt	positivo
CI	pan	positivo
CR	tilt	negativo
SQ	pan	negativo
L1	zoom	negativo
R1	zoom	positivo

4.4.2 Associação adicional de movimento da câmera

Na fase inicial de operação do robô, foi codificada a opção de controle da movimentação da câmera com o *joystick*. O nó ***p2os_driver*** ouve mensagens do tipo *p2os_driver/PTZState* no tópico /ptz_control e publica continuamente mensagens do mesmo tipo no tópico /ptz_state. Tais tópicos e mensagens indicam o estado desejado e o estado atual da câmera nas coordenadas *pan*, eixo de rotação da base, *tilt*, eixo que prende a câmera na base e *zoom*, que é o zoom próprio da imagem na câmera. A Tabela 4.7 mostra os limites máximos e mínimos utilizados pela câmera do Pioneer.

Cada sentido de movimentação dos eixos da câmera foram mapeados para os botões do *joystick*, sendo que foram escolhidos os passos de **30** unidades para o *tilt*, **30** unidades para o *pan* e **100** unidades para o *zoom* por vez, podendo serem reconfigurados pelo mesmo arquivo de configuração do movimento das rodas do robô. A Tabela 4.8 mostra o mapeamento utilizado para movimentar a câmera.

Embora tenha sido codificado tal estrutura, não foi utilizada pelo projeto, já que idealmente o melhor seria não precisar movimentar a câmera.

4.5 Salvamento do fluxo em disco

Para que fosse possível obter os resultados e analisar a operação de obtenção do mapa, era necessário armazenar as informações obtidas da captura da câmera do robô.

Para salvar estas informações, foi utilizado o pacote do ROS chamado ***rosbag*** e seu nó *record*. Este nó permite salvar em disco todas as mensagens sendo publicadas em um tópico, que no caso foi */camera/image/grayscale/compressed*.

4.6 ORB SLAM

Para realizar o slam visual, foi utilizado como base o algoritmo ORB SLAM[1], disponível publicamente como código aberto no repositório GitHub em `https://github.com/raulmur/ORB_SLAM2`, sendo utilizada a versão dois do algoritmo. Para que fosse possível realizar modificações sem alterar o código original, este repositório foi clonado para um repositório de uso pessoal, disponível em `https://github.com/gabrielmirandat/ORB_SLAM2`.

4.6.1 Modificações

As modificações no código do ORB SLAM a seguir foram necessárias para adaptá-lo ao sistema computacional proposto e para fazê-lo funcional no prédio CiC/Est.

Aceitar imagens níveis de cinza comprimidas

Foi necessário alterar o código original para aceitar imagens comprimidas e em níveis de cinza. Inicialmente, o algoritmo aceita imagens coloridas nos canais RGB e internamente realiza a conversão para níveis de cinza.

Para realizar a conversão, novamente foi utilizado o plugin ***compressed_image_transport***[6], em que foi necessário alterar o gerenciador do nó do ORB SLAM comum para o gerenciador disponibilizado pelo *plugin*. Além disso, o tópico esperado para recebimento de mensagens foi alterado de ***/camera/image_raw*** para ***camera/image/grayscale/compressed***.

Correção de problema na etapa de Inicialização

Inicialmente, o algoritmo original apresentava grande dificuldade na etapa de inicialização, vide seção 2.3.2, de forma que a necessidade de profundidade suficiente necessária para inicializar dificilmente era atingida. Isto foi reportado como um *bug* pelos usuários e foi proposta uma correção no fórum oficial do algoritmo, em `https://github.com/raulmur/ORB_SLAM2/issues/59`. Após alterar o código com as soluções propostas no fórum o problema de inicialização foi resolvido.

4.7 Código realizado

De forma a reunir todo o código utilizado na operação do robô, foi criado um repositório na plataforma Github do projeto, disponível em `https://github.com/gabrielmirandat/tcc_ws`. O repositório por si só já é um *workspace* do ROS, de forma que segue toda a

[6] `http://wiki.ros.org/compressed_image_transport`

estrutura mostrada na seção 3.1.1. Os principais componentes do repositório são o código do ORB SLAM e o pacote ***resources***, que contém todos os arquivos *launch* utilizados na operação, arquivos de configuração e os dois nós criados, chamados ***camera_node***, que obtém o fluxo de vídeo comprimido e em níveis de cinza e ***joystick_node***, que realiza o mapeamento do *joystick* para controle do robô.

Capítulo 5

Resultados e Discussão

Para avaliar o desempenho do mapeamento realizado pelo Pioneer no 1º andar do prédio CiC/Est foram realizadas operações utilizando o sistema mostrado no Capítulo 4. Doze vídeos no formato *bag* do ROS foram gravados para permitir futura análise da operação e obtenção do mapa, sendo que foram classificados por dia, hora, duração da operação, trajeto realizado, tempo climático e completude do mapa gerado. Além disto, análises referentes a perdas de rastreamento, fechamento de *loops* e mudança repentina de escala foram realizadas para cada operação. Tais informações podem ser vistas nas Tabelas 5.1 a 5.2. Para o trajeto realizado, o objetivo é passar por todos os nós e arestas do mapa do prédio mostrado na Figura 5.1.

A depender da completude gerada do mapa, foi feita uma comparação qualitativa do mapa obtido com uma planta do prédio usado como *ground truth*. Para realizar uma análise quantitativa assim como a realizada por [1] em que se considera o *Root Mean Square Error (RMSE)* de translação seria necessário ter quantificada uma trajetória do mapa em coordenadas globais para cada trajetória realizada pelo robô como *ground truth* no formato utilizado pela ferramenta de comparação, disponível em `https://vision.in.tum.de/data/datasets/rgbd-dataset/tools#evaluation`. Devido ao tempo curto para realização do trabalho e o conhecimento tardio da ferramenta, em que cada trajetória deveria ser feita tendo por base um *ground truth*, apenas a análise qualitativa foi realizada.

Tabela 5.1: Operações realizadas para análise do mapa e da localização. Operação completa significa que todos os nós foram visitados. Mapa completo significa que houve mapeamento de todas as arestas da operação. Fonte: Autor.

Dia	**Hora**	**Duração**	**Tempo**	**Operação**	**Mapa**
12 de maio	16:33	17:50s	Nublado	Incompleta	Completo
12 de maio	17:04	14:33s	Nublado	Completa	Incompleto
12 de maio	21:13	10:49s	Noite	Incompleta	Incompleto
16 de maio	13:02	27:21s	Ensolarado	Completa	Irreconhecível
19 de maio	10:58	17:42s	Ensolarado	Completa	Incompleto
23 de maio	9:19	24:36s	Ensolarado	Completa	Incompleto
23 de maio	9:50	28:17s	Nublado	Completa	Completo
29 de maio	17:12	21:09s	Nublado	Completa	Completo
29 de maio	17:42	27:27s	Nublado	Completa	Incompleta
30 de maio	14:30	18:30s	Ensolarado	Completo	Completo
2 de junho	10:49	28:04s	Ensolarado	Completo	Completo
2 de junho	11:27	19:04s	Ensolarado	Completo	Completo

Tabela 5.2: Trajetos realizados por operação. Fonte: Autor.

Id	**Trajeto: Sequência de nós no mapa**
12/05-16:33	NMLHABCDEJPONM
12/05-17:04	LMNOPJEDCINOPQRKGFEDCBAHLM
12/05-21:13	RQPONMLHABCDEFGKRQP
16/05-13:02	ABCDEJPONICDEFGKRQPONMLHAB
19/05-10:58	BCDEJPONICDEFGKRQPONMLHABC
23/05-9:19	RQPONICDEJPONMLHABCDEFGKRQ
23/05-9:50	RQPONICDEJPONMLHABCDEFGKRQPO
29/05-17:12	ABCDEJPONICDEFGKRQPONMLHAB
29/05-17:42	LMNOPJEDCINOPQRKGFEDCINMLHABC
30/05-14:30	LMNOPJEDCINOPQRKGFEDCBAHLM
2/06-10:49	MNOPJEDCINOPQRKGFEDCINOPJEDCBAHL
2/06-11:27	GFEJPONICDEJPONMLABCDEJPQRKGFEJPQRKGF

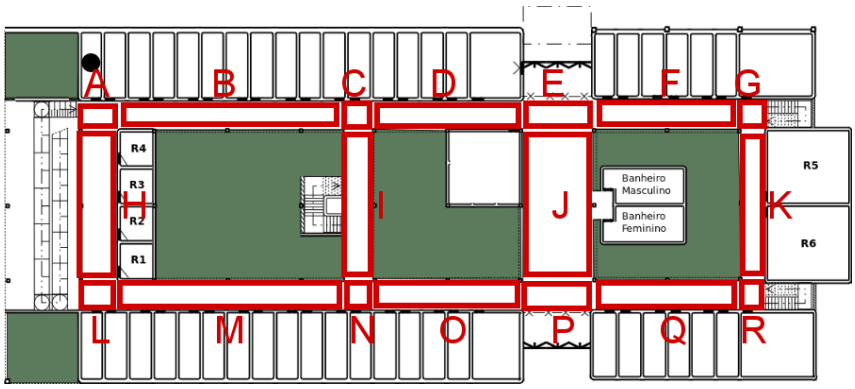


Figura 5.1: Planta baixa simplificada do Edifício Cic/Est com definição de locais para especificar as trajetórias realizadas. Fonte: Autor.

5.1 Operações

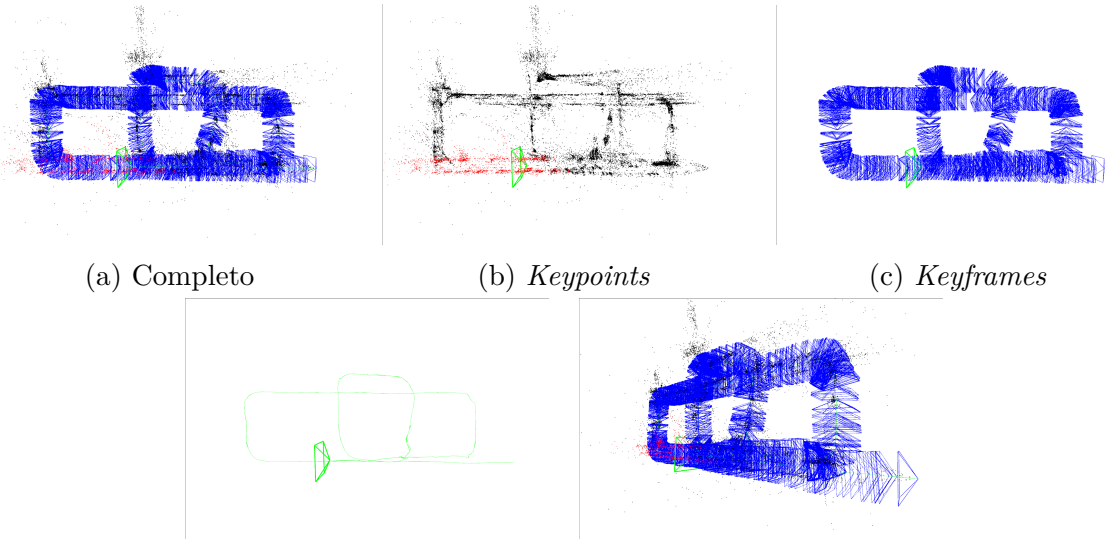
As operações que apresentaram trajeto completo (passando por todos os nós do prédio) e geração de mapa completo (todas as arestas mapeadas) são as tiveram resultados de maior relevância e são mostradas a seguir. Para as imagens da nuvem de pontos, a estrutura verde é a posição atual do robô, os pontos são os *keypoints* que representam o mapa gerado e os pontos em vermelho são os *keypoints* do mapa local atualmente sendo utilizado pelo algoritmo.

5.1.1 Operação completa com mapa completo

23/05-9:50

O trajeto realizado começou de RQ e seguiu por QN, NC, CE, EP, PL, LA, AG, GR, RO. Apresentou perda de escala passando por CD, mas conseguiu recuperar fechando *loop* em PO. Não conseguiu fechar o *loop* na segunda passagem por CD, gerando dois caminhos diferentes no extremo superior. A Figura 5.2 mostra as imagens geradas da operação e a Figura 5.3 mostra o mapeamento realizado sobreposto ao prédio.

Para este caso, o mapa gerado logo após a operação havia conseguido fechar todos os *loops*. A diferença vista acima foi a operação com o vídeo salvo pelo *rosbag*. Esta inconsistência entre os mapas pode ser devida a alguma diferença de performance ao executar o vídeo salvo pelo *rosbag*. A Figura 5.4 mostra imagens tiradas logo após a operação, sendo as imagens obtidas diretamente do cabo *Ethernet*. A Figura 5.5 mostra o mapa com keyframes sobreposto ao prédio.



(a) Completo (b) *Keypoints* (c) *Keyframes*

(d) Grafo (*Spanning Tree*) (e) Outra vista

Figura 5.2: 23/05-9:50 Resultado da operação. Fonte: Autor.

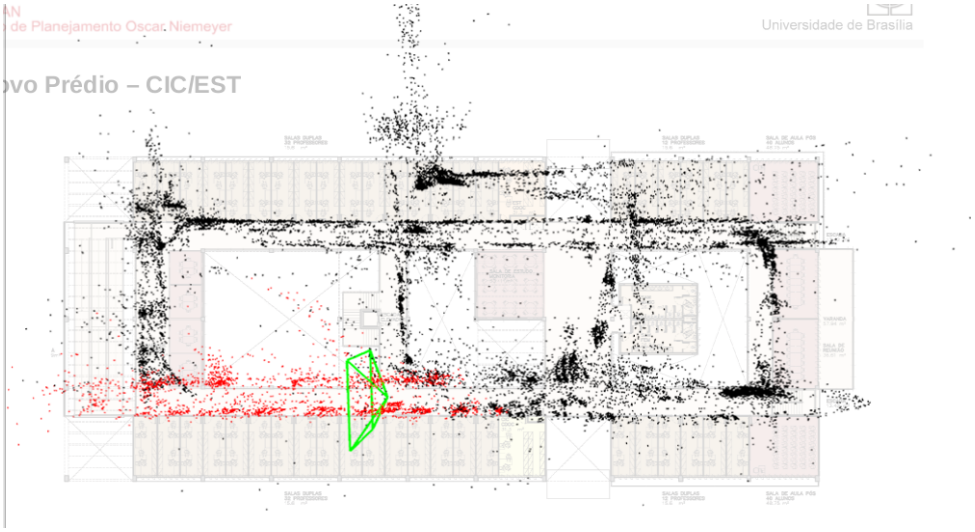
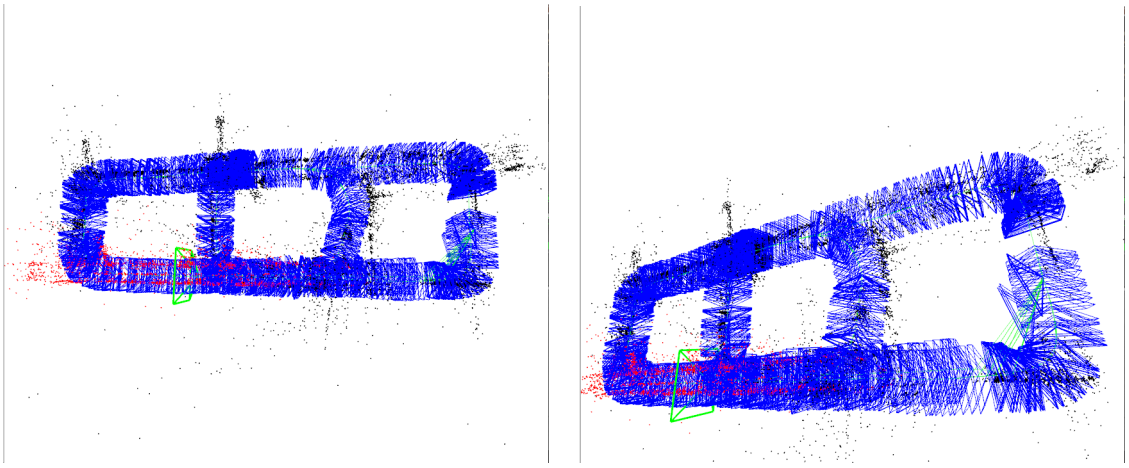


Figura 5.3: 23/05-9:50 Mapa gerado sobreposto ao prédio. Fonte: Autor.



(a) Completo (b) Outra vista

Figura 5.4: 23/05-9:50 Resultado obtido imediatamente após finalizar operação. Fonte: Autor.

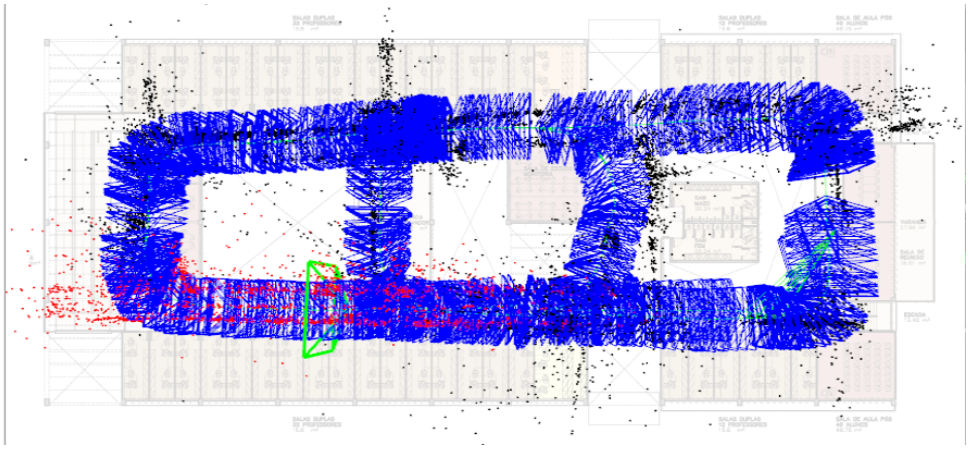
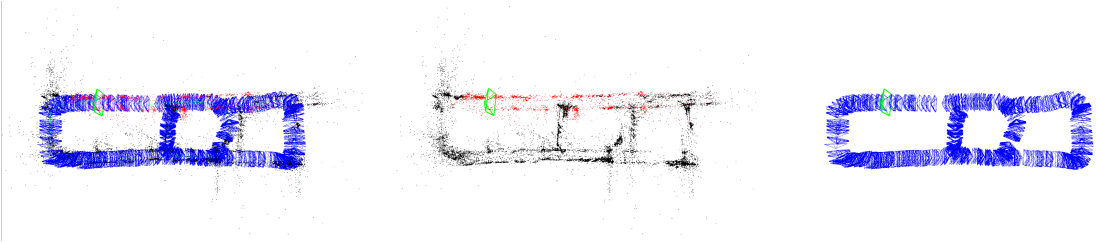


Figura 5.5: 23/05-9:50 Mapa e keyframes gerado sobreposto ao prédio. Fonte: Autor.

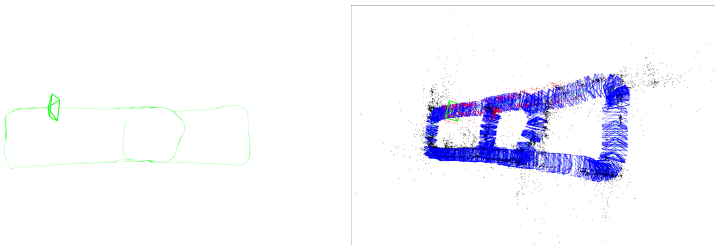
29/05-17:12

O trajeto realizado começou de A e seguiu por E, EP, PN, NC, CG, GR, RL, LA e terminou em B. A operação foi um sucesso, conseguiu fechar os *loops* corretamente na segunda passagem por C, P e A. A Figura 5.6 mostra as imagens geradas da operação.

As Figuras 5.7 a 5.8 mostram a sobreposição da nuvem de pontos assim como os *keyframes* no mapa do CIC. É possível observar que o percurso NC ficou deslocado para a direita, dando a impressão que o retângulo formado por ACNL é mais largo.



(a) Completo (b) *Keypoints* (c) *Keyframes*



(d) Grafo (*Spanning Tree*) (e) Outra vista

Figura 5.6: 29/05-17:12 Resultado da operação. Fonte: Autor.

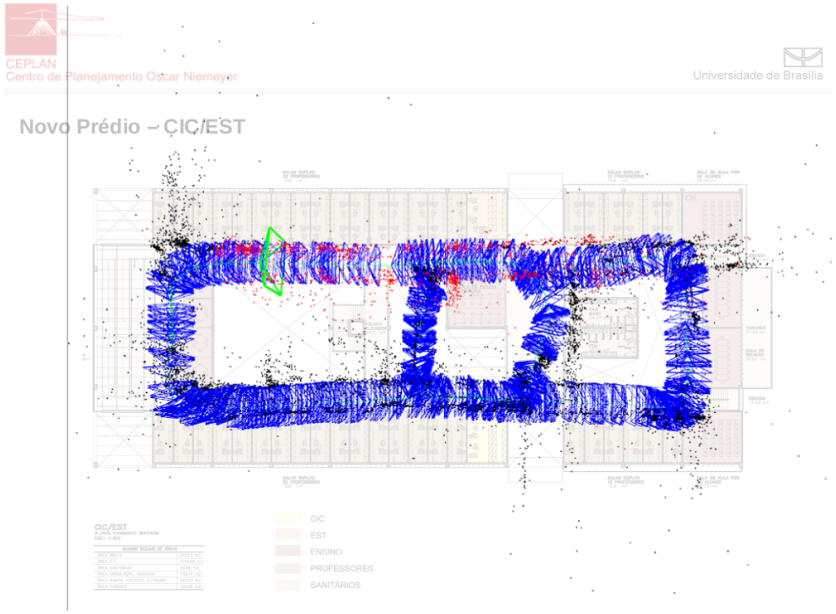


Figura 5.7: 29/05-17:12 Keyframes sobrepostos ao prédio. Fonte: Autor.

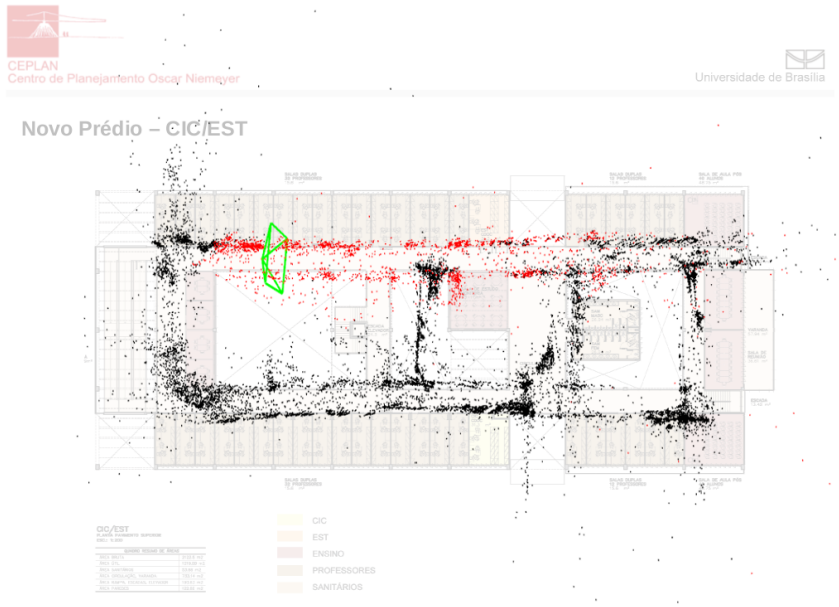
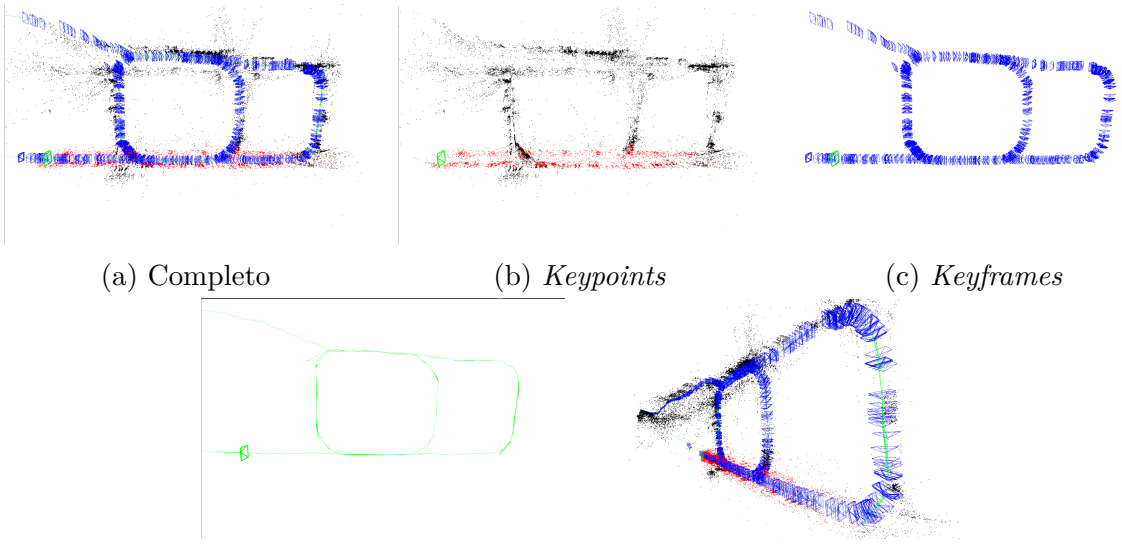


Figura 5.8: 29/05-17:12 Nuvem de pontos sobrepostos ao prédio. Fonte: Autor.

30/05-14:30

O trajeto foi LP, PE, EC, CN, NR, RG, GA, ALM. Neste caso ocorreram problemas de escala nos caminhos LM, logo no início, e em AB, logo no fim, o que gerou uma inconsistência ao fechar o *loop* final. A Figura 5.9 mostra as imagens geradas da operação.

As Figuras 5.10 a 5.11 mostram a sobreposição da nuvem de pontos assim como os *keyframes* no mapa do CIC. É possível observar que não aparece o mapeamento realizado no trajeto BALM. Como pode ser visto na Figura 5.9 (e), o pedaço do mapa referente a esta seção se distanciou muito da realidade, devido ao problema mencionado no parágrafo anterior.



(a) Completo (b) *Keypoints* (c) *Keyframes*

(d) Grafo (*Spanning Tree*) (e) Outra vista

Figura 5.9: 30/05-14:30 Resultado da operação. Fonte: Autor.

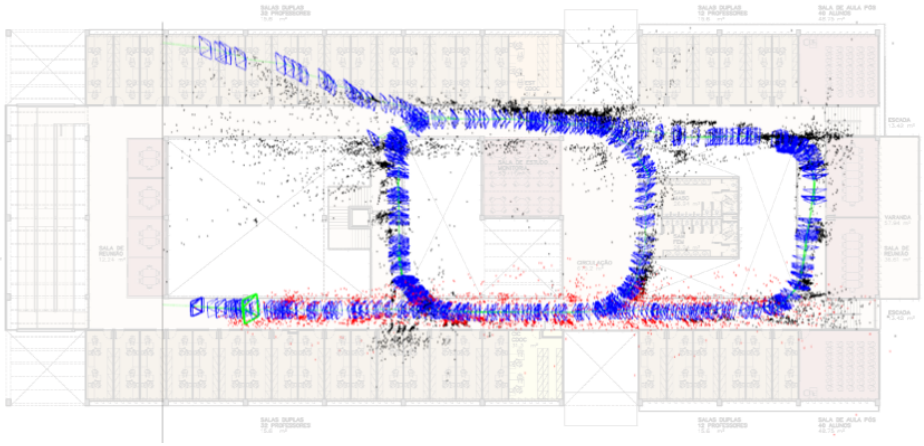


Figura 5.10: 30/05-14:30 Keyframes sobrepostos ao prédio. Fonte: Autor.

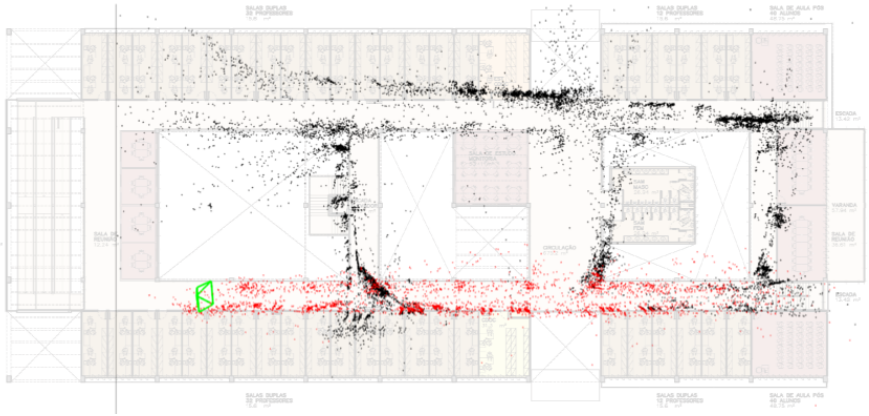
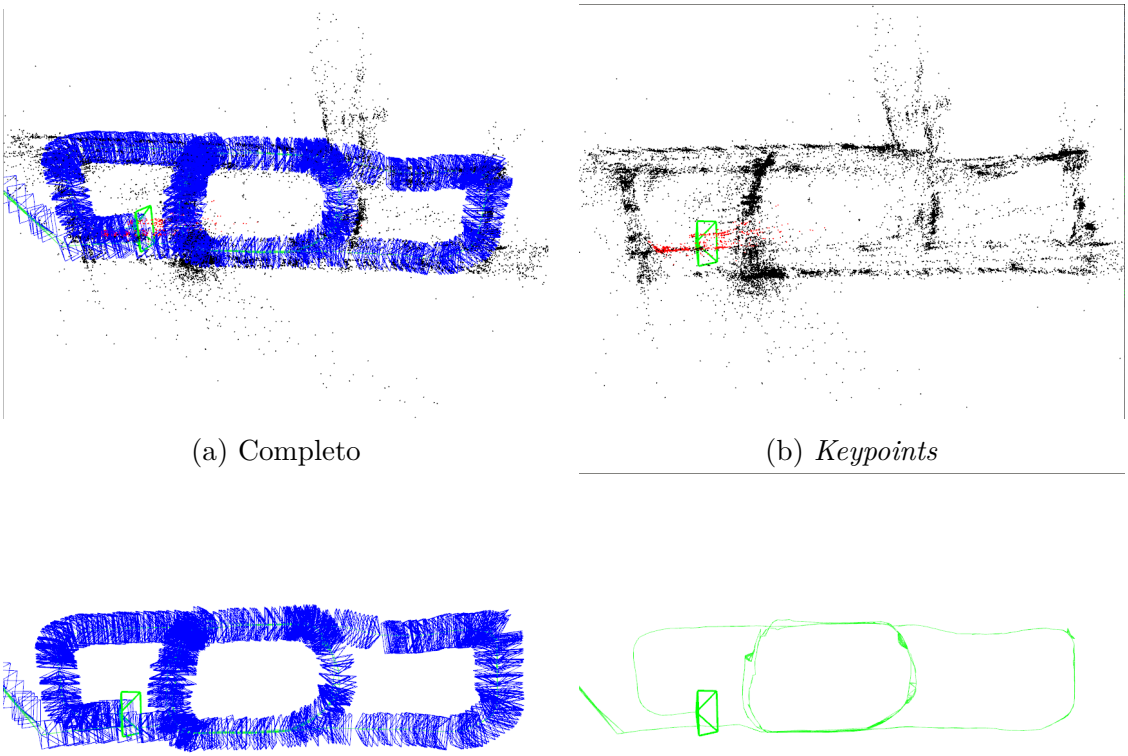


Figura 5.11: 30/05-14:30 Nuvem de pontos sobrepostos ao prédio. Fonte: Autor.

2/06-10:49

O mesmo trajeto da operação anterior foi realizado porém com leves variações. Começando por MP, seguiu por PE, EC, CN, NR, RG, GC, CN, NP, PE, EA, AE, ou seja, deu duas voltas no retângulo central. Ocorreram mudanças de escala em menor grau, não conseguiu fechar o último *loop* voltando por LM. A Figura 5.12 mostra as imagens geradas da operação.

As Figuras 5.13 a 5.14 mostram a sobreposição da nuvem de pontos assim como os *keyframes* no mapa do CIC. É possível observar que ocorreram divergências nas escalas que compõe os retângulos centrais do mapa.



(a) Completo

(b) *Keypoints*

(c) *Keyframes*

(d) Grafo (*Spanning Tree*)

Figura 5.12: 2/06-10:49 Resultado da operação. Fonte: Autor.

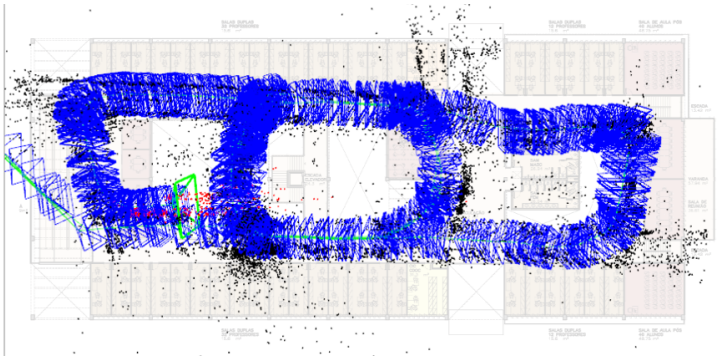


Figura 5.13: 2/06-10:49 Keyframes sobrepostos ao prédio. Fonte: Autor.

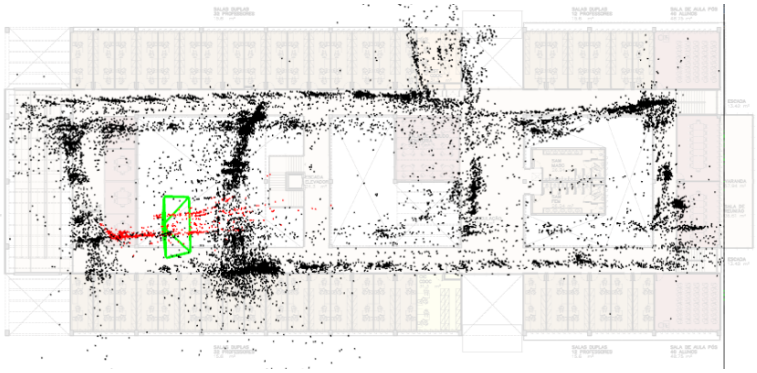
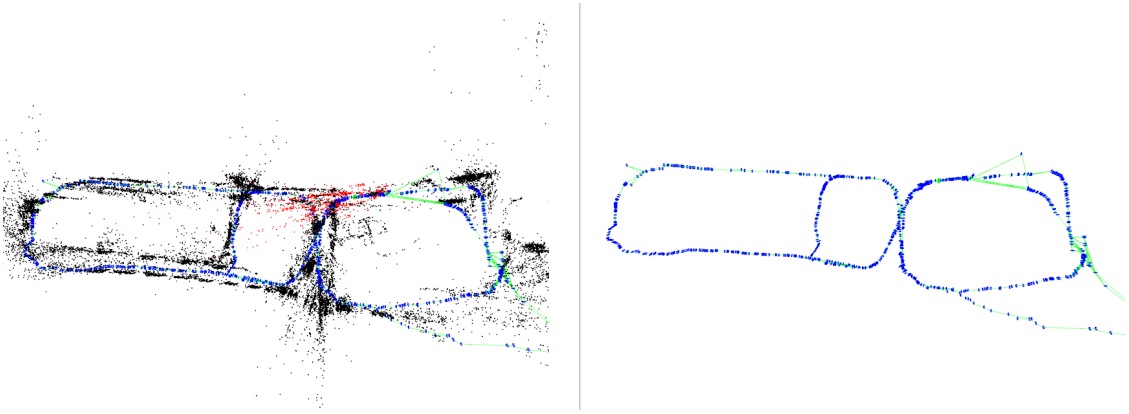


Figura 5.14: 2/06-10:49 Nuvem de pontos sobrepostos ao prédio. Fonte: Autor.

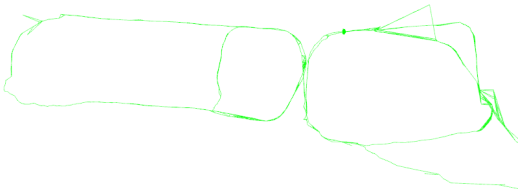
2/06-11:27

Neste caso foi realizada a maior trajetória, sendo GE, EP, PN, NC, CE, EP, PL, LA, AE, EP, PR, RG, GE, EP, PR, RG, GF. Ocorreram problemas de escala nos caminhos ICD, LM e na primeira passagem por PQR. *Loops* foram fechados na segunda passada por JP, EJ, e QR. QR. A Figura 5.15 mostra as imagens geradas da operação.

A Figura 5.16 mostra a sobreposição da nuvem de pontos assim como os *keyframes* no mapa do CIC.



(a) Completo (b) *Keyframes*



(c) Grafo (*Spanning Tree*)

Figura 5.15: 2/06-11:27 Resultado da operação. Fonte: Autor.

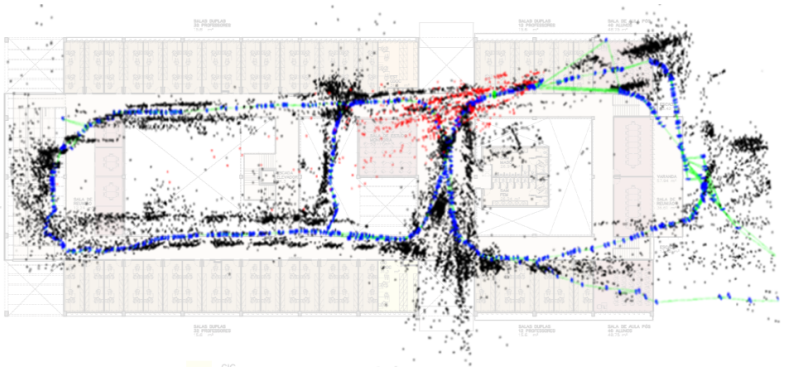


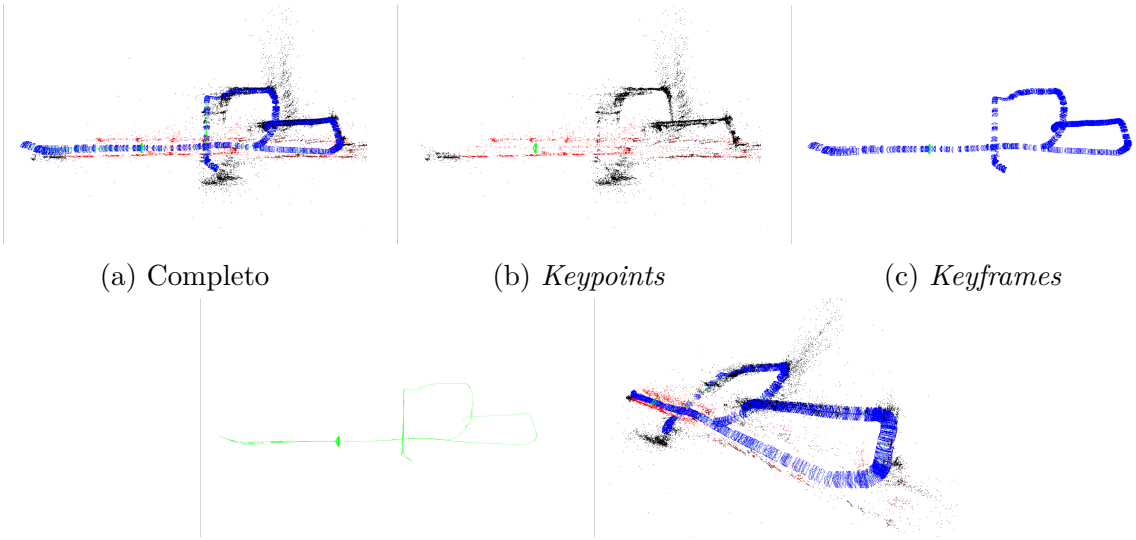
Figura 5.16: 2/06-11:27 Keyframes e nuvem sobrepostos ao prédio. Fonte: Autor.

5.1.2 Operação completa com mapa incompleto

12/05-17:04

Trajeto completo que segue por LP, PE, EC, CN, NR, RG, GA, ALM. A mudança brusca de escala na primeira passagem por ED fez com que o fechamento de *loop* na segunda passada por NO gerasse o erro no extremo inferior do mapa. Posterior erro de escala em RKG e perda de rastreamento em ED fez com que o mapa não fosse muito condizente. A Figura 5.17 mostra as imagens geradas da operação.

As Figuras 5.18 a 5.19 mostram que ao sobrepor o mapa gerado no mapa do CIC grandes problemas de escala são observados. O corredor inicial ficou maior que o esperado, além de o algoritmo ter falhado em fechar um *loop* na segunda passada por ED.



(a) Completo (b) *Keypoints* (c) *Keyframes*

(d) Grafo (*Spanning Tree*) (e) Outra vista

Figura 5.17: 12/05-17:04 Resultado da operação. Fonte: Autor.

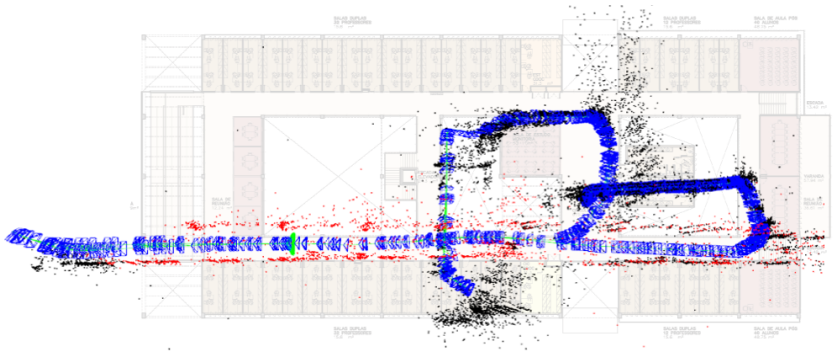


Figura 5.18: 12/05-17:04 Keyframes sobrepostos ao prédio. Fonte: Autor.

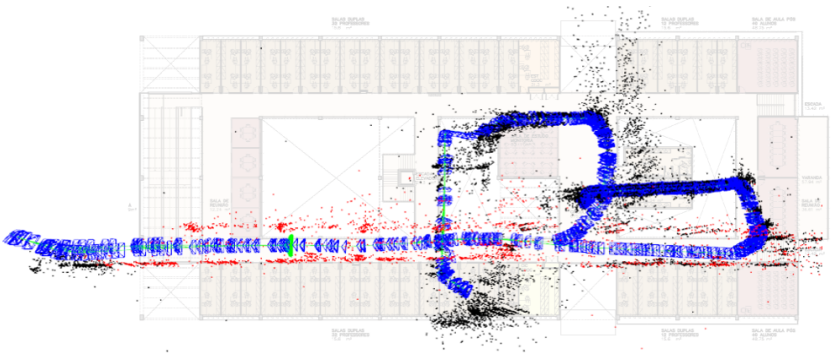
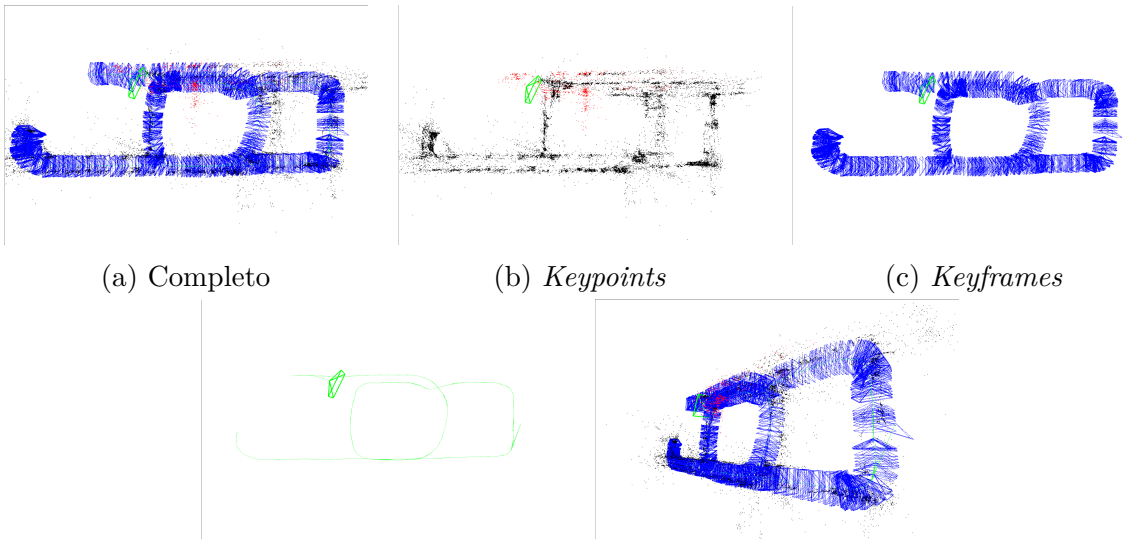


Figura 5.19: 12/05-17:04 Mapa sobreposto ao prédio. Fonte: Autor.

19/05-10:58

Trajeto completo que quase gerou o mapa completo. Começa de B a E, e segue por EP, PN, NC, CG, GR, RL, LA, terminando em C. Os resultados foram satisfatórios apesar de ter se perdido duas vezes, a primeira em RQ devido a câmera ter virado e a segunda ao revisitar AB e ter perdido o rastreamento devido a movimento brusco pelo sinalizador para pessoas com deficiência. Além disto, a mudança brusca na escala ao passar por LJ. A segunda perda é a principal causa de ter impossibilitado o fechamento de *loop* em AB e permitido a geração do mapa completo. A Figura 5.20 mostra as imagens geradas da operação.

As Figuras 5.21 a 5.22 mostram o resultado da sobreposição.



(a) Completo (b) *Keypoints* (c) *Keyframes*

(d) Grafo (*Spanning Tree*) (e) Outra vista

Figura 5.20: 19/05-10:58 Resultado da operação. Fonte: Autor.

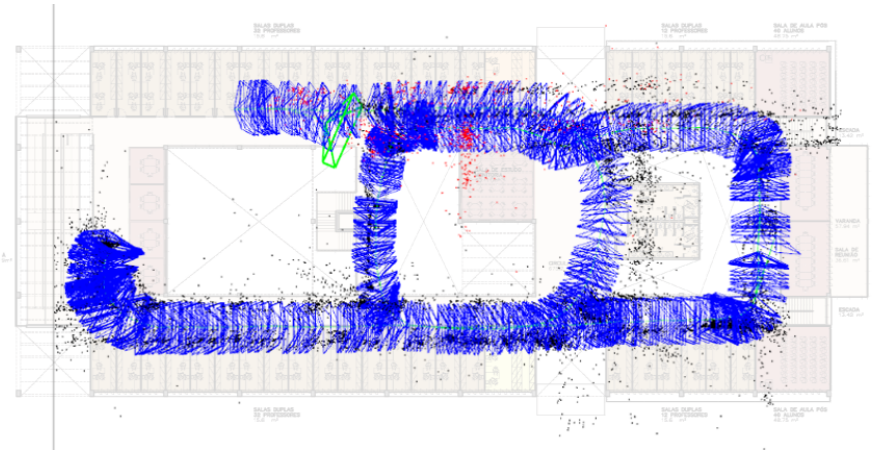


Figura 5.21: 19/05-10:58 Keyframes sobreposto ao prédio. Fonte: Autor.

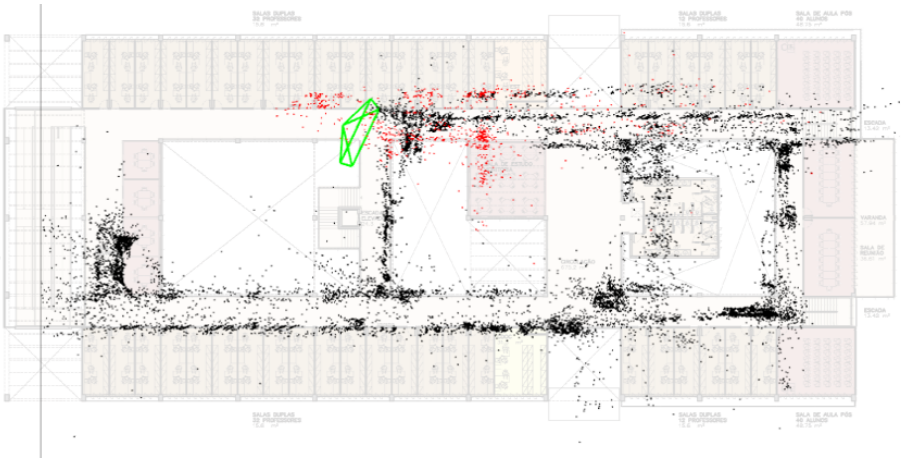
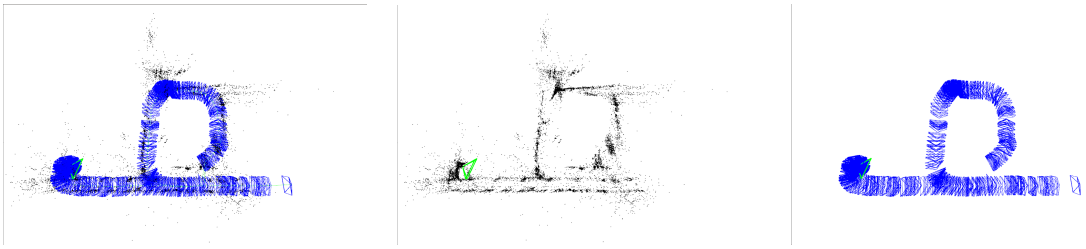


Figura 5.22: 19/05-10:58 Mapa sobreposto ao prédio. Fonte: Autor.

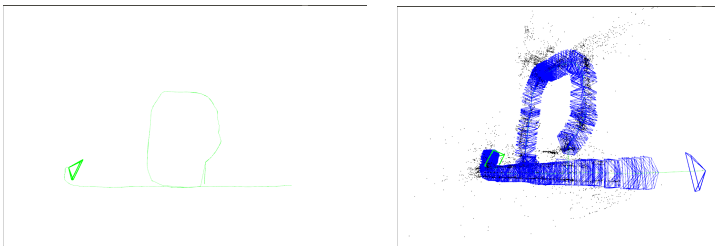
23/05-9:19

O trajeto realizado começou de RQ e seguiu por QN, NC, CE, EP, PL, LA, AG, GR, RQ. O algoritmo conseguiu fechar um *loop* na segunda passada por PO. Devido a uma perda de escala em L e uma perda de rastreamento em AB pelo motivo da passagem de pessoas cobrirem todas as features não foi possível fechar um *loop* na segunda passagem por CD, resultando num mapa incompleto. A Figura 5.23 mostra as imagens geradas da operação.

As Figuras 5.24 a 5.25 mostram o mapeamento realizado sobreposto ao prédio.



(a) Completo (b) *Keypoints* (c) *Keyframes*



(d) Grafo (*Spanning Tree*) (e) Outra vista

Figura 5.23: 23/05-9:19 Resultado da operação. Fonte: Autor.

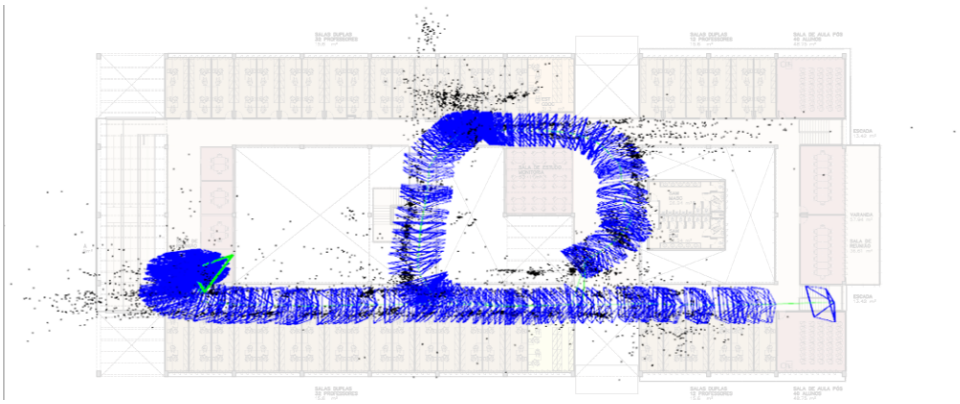


Figura 5.24: 23/05-9:19 Keyframes sobreposto ao prédio. Fonte: Autor.

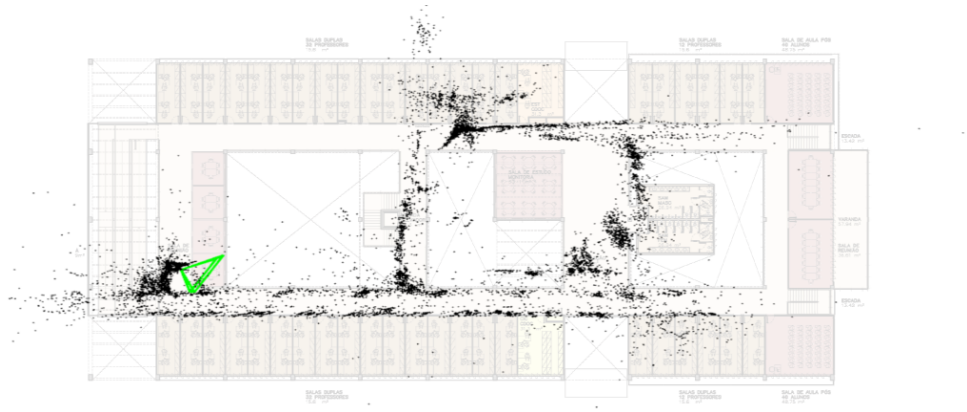
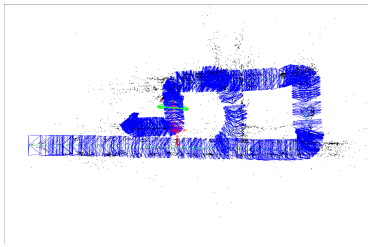


Figura 5.25: 23/05-9:19 Mapa sobreposto ao prédio. Fonte: Autor.

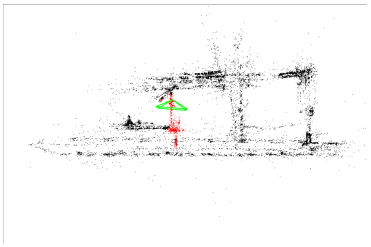
29/05-17:42

O trajeto realizado fez a experiência de tentar fechar um *loop* passando pelo mesmo caminho em sentido contrário. O trajeto realizado começou de L indo a P, PE, EC, CN, CN, RG, GC, CN, NL, LA, AC. Houve uma perda de rastreamento em ED devido a passagem na sinalização para pessoas com deficiência. A partir daí, consegue se relocalizar na segunda passagem por NO. Teve uma mudança de escala na segunda passagem por CI, de forma que não conseguiu fechar um *loop* no sentido NM. Apresentou outra perda em LH por rotação sem tempo de triangular novas *features*. A Figura 5.26 mostra as imagens geradas da operação.

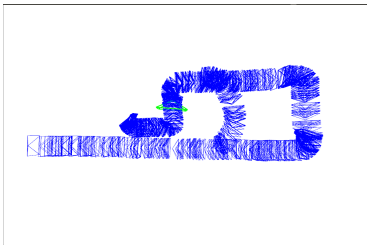
As Figuras 5.27 a 5.28 mostram a sobreposição da nuvem de pontos assim como os *keyframes* no mapa do CIC. Devido a perdas de escala e não ter fechado o loop acreditou estar em novo corredor e não no mesmo.



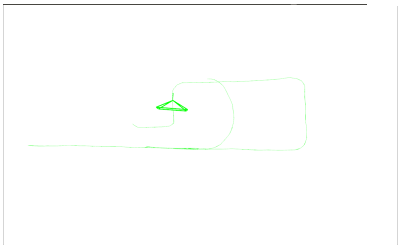
(a) Completo



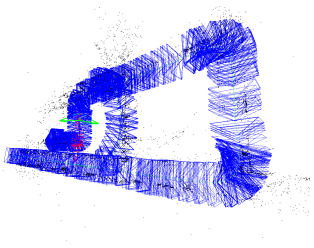
(b) *Keypoints*



(c) *Keyframes*



(d) Grafo (*Spanning Tree*)



(e) Outra vista

Figura 5.26: 29/05-17:42 Resultado da operação. Fonte: Autor.

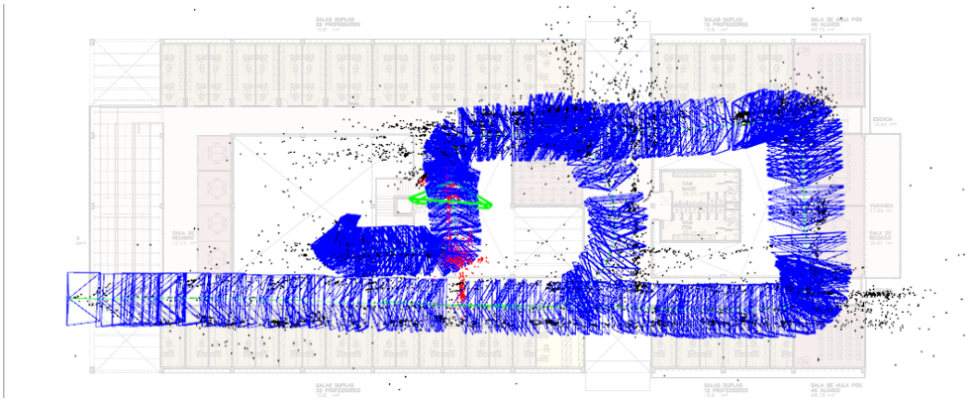


Figura 5.27: 29/05-17:42 Keyframes sobrepostos ao prédio. Fonte: Autor.

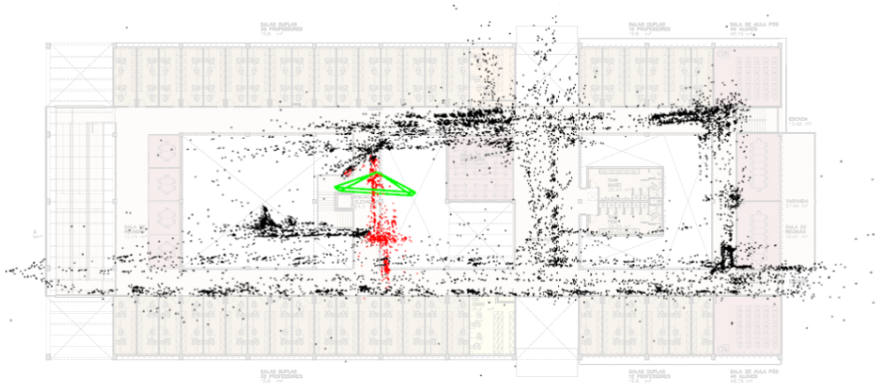
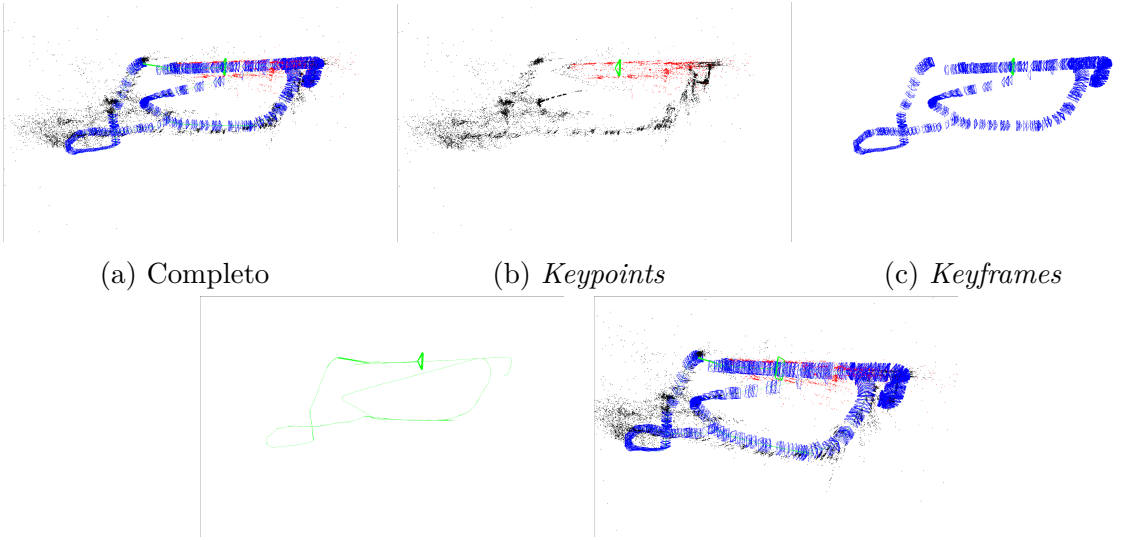


Figura 5.28: 29/05-17:42 Nuvem de pontos sobrepostos ao prédio. Fonte: Autor.

5.1.3 Operação completa com mapa irreconhecível

16/05-13:02

O trajeto mostra um exemplo que apresentou diversos problemas e acabou gerando um mapa irreconhecível. Um grande erro ocorrido na operação foi um *loop* fechado no nó errado, que deveria ocorrer no nó D, porém fechou em A. Além disso, ocorreu uma perda de rastreamento devido ao desfoque repentino da câmera, além de ter tido mudança brusca na escala no nó N. A Figura 5.29 mostra as imagens geradas da operação.



(a) Completo (b) *Keypoints* (c) *Keyframes*

(d) Grafo (*Spanning Tree*) (e) Outra vista

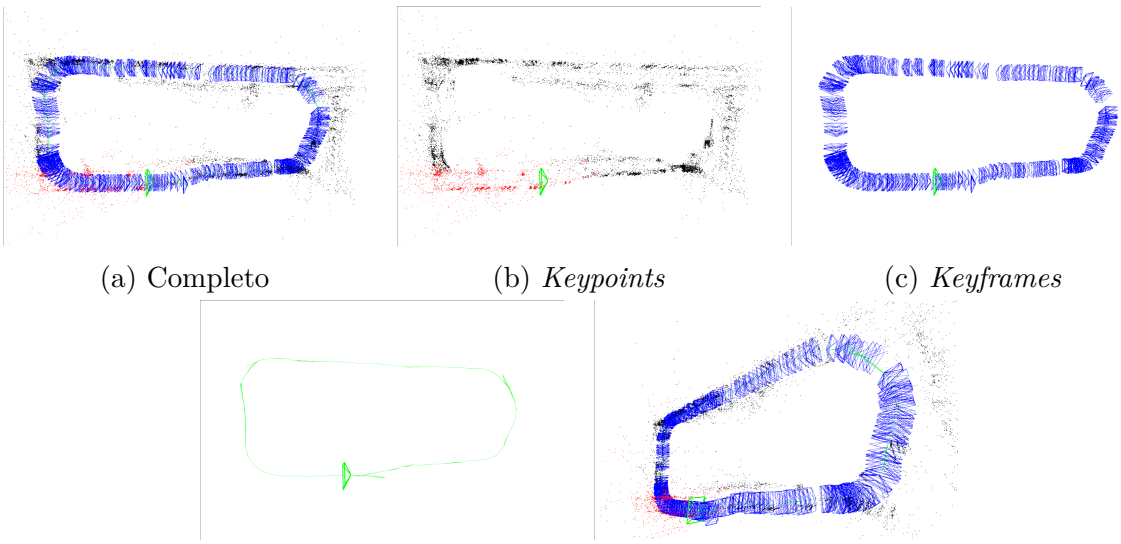
Figura 5.29: 16/05-13:02 Resultado da operação. Fonte: Autor.

5.1.4 Operação incompleta com mapa completo

12/05-16:33

O trajeto percorrido representa apenas uma parte do andar, o retângulo formado por AE, EP, PL, LA. Começando por N, segue por M e dá uma volta até fechar um *loop* na segunda passada por M. Não ocorreram perdas de rastreamento nem problemas de escala. A Figura 5.30 mostra as imagens geradas da operação.

As Figuras 5.31 a 5.32 mostram a sobreposição do mapa.



(a) Completo (b) *Keypoints* (c) *Keyframes*

(d) Grafo (*Spanning Tree*) (e) Outra vista

Figura 5.30: 12/05-16:33 Resultado da operação. Fonte: Autor.

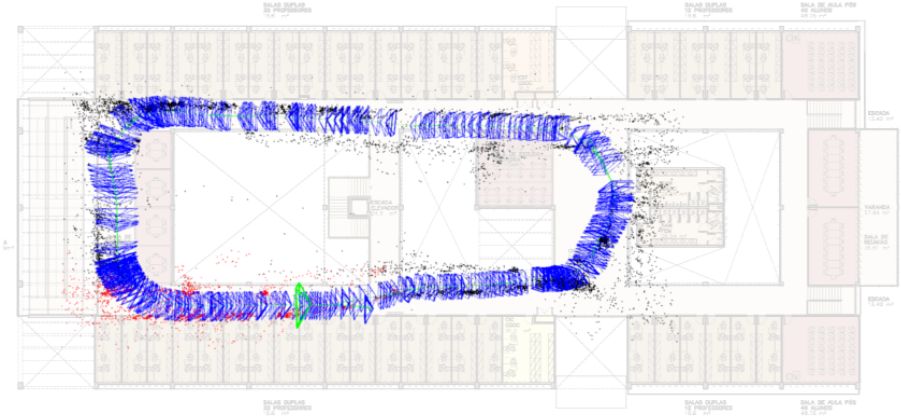


Figura 5.31: 12/05-16:33 Keyframes sobreposto ao prédio. Fonte: Autor.

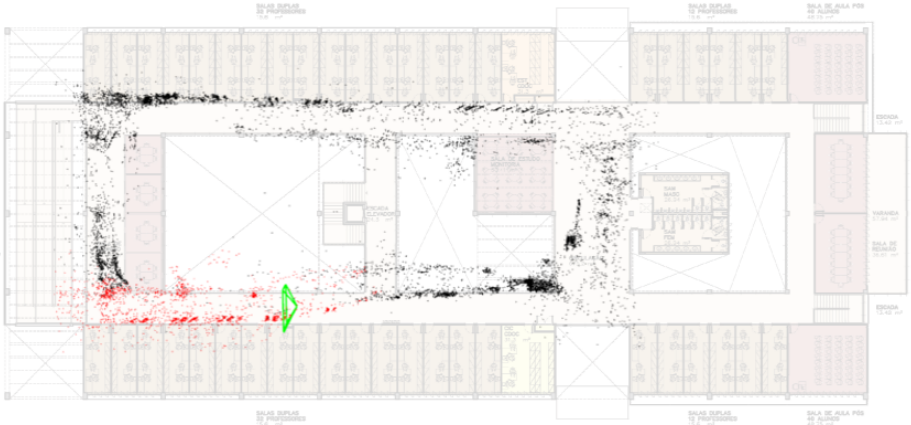


Figura 5.32: 12/05-16:33 Mapa gerado sobreposto ao prédio. Fonte: Autor.

5.1.5 Operação incompleta com mapa incompleto

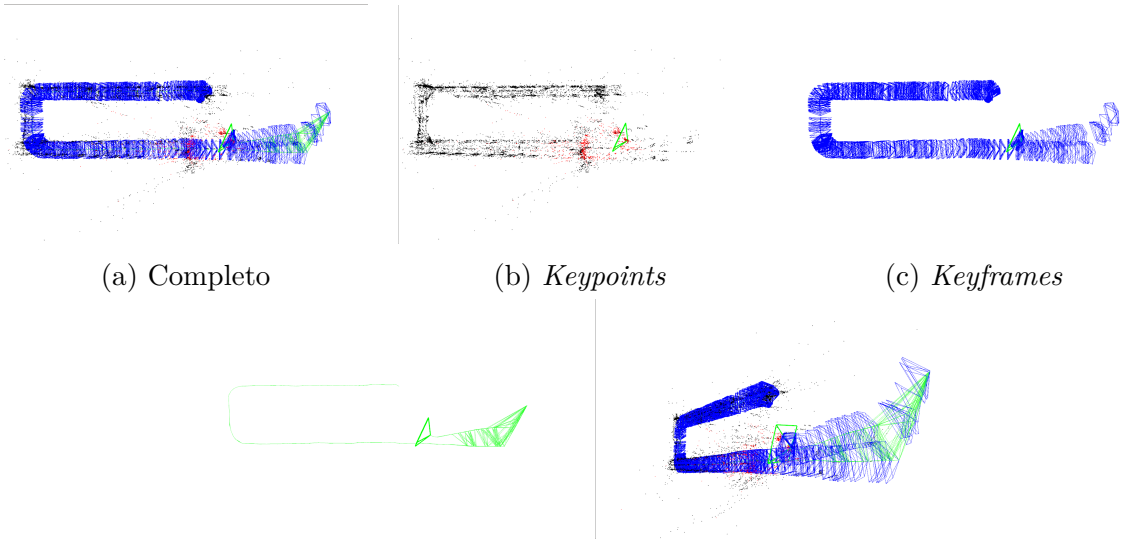
12/05-21:13

O trajeto percorrido representa o retângulo mais externo do prédio, que vai de RL, LA, AG e GR. Não apresentou maiores problemas de escala, porém se perdeu logo antes de fechar o *loop* na segunda passagem por RQ, na curva fechada em GK, devido a parede branca e lisa incapaz de prover *features*.

A Figura 5.33 mostra as imagens geradas da operação.

As Figuras 5.34 a 5.35 mostram que ao sobrepor o mapa gerado no mapa do CIC a escala do percurso vai perdendo precisão com o tempo, de forma que acha que está em EF

enquanto na realidade está virando em G. Existe também uma inconsistência ao fechar o *loop*, em que ruído aparece no extremo direito do mapa.



(a) Completo (b) *Keypoints* (c) *Keyframes*

(d) Grafo (*Spanning Tree*) (e) Outra vista

Figura 5.33: 12/05-21:13 Resultado da operação. Fonte: Autor.

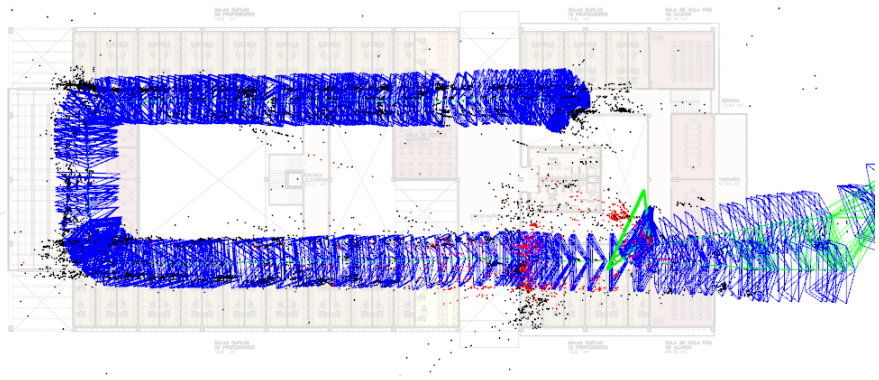


Figura 5.34: 12/05-21:13 Keyframes sobrepostos ao prédio. Fonte: Autor.

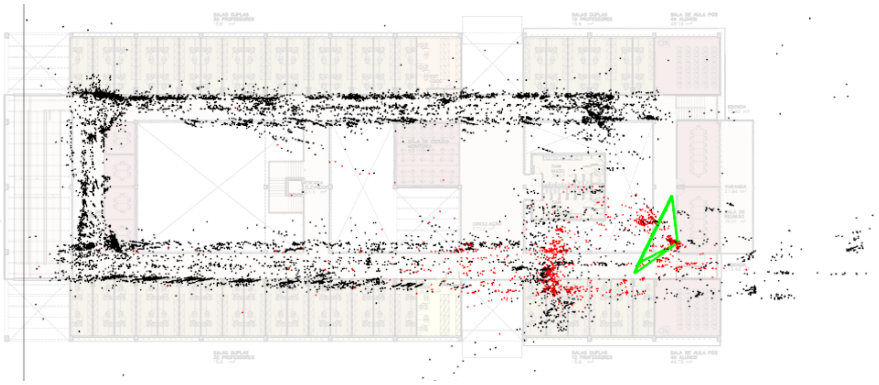


Figura 5.35: 12/05-21:13 Mapa sobreposto ao prédio. Fonte: Autor.

5.2 Dificuldades

As principais dificuldades relacionadas a operação, ao prédio, a câmera e ao algoritmo são sumarizadas a seguir. É possível notar das operações que alguns dos problemas listados na seção 2.3 referentes aos problemas abertos em SLAM visual aconteceram aqui.

5.2.1 Operação do robô

A seguir algumas dificuldades encontradas na operação do robô com o joystick são descritas.

- O operador do robô não deve realizar movimentos de rotação pura. Nos casos em que houve rotação sem translação o robô acabou perdendo o rastreio das *features*, gerando inconsistências no mapa.
- O operador necessita observar constantemente se o robô visualiza features suficientes. Locais com fundos planos e lisos não podem ser mapeados.
- Para qualquer curva realizada, é necessário transladar o robô até que o algoritmo consiga triangular novas *features* sendo observadas. Para isto, o algoritmo precisa observar uma mesma feature de posições globais diferentes.
- Apesar do algoritmo possuir certa robustez em relação a objetos dinâmicos, em alguns casos os pedestres atrapalharam a operação, tanto por fazer parte do mapa como um objeto estático, quando por cobrir totalmente as *features* observadas pelo robô, acarretando na perda de rastreamento das *features*. A Figura 5.36 ilustra um exemplo em que pedestres fizeram parte do mapa.

Figura 5.36: Pedestre que acabou fazendo parte do mapa, operação **23/05-09:19**. Fonte: Autor.

5.2.2 Prédio CiC/Est

A seguir são descritas algumas dificuldades inerentes a realizar SLAM no CiC/Est.

- O edifício possui situações de fundo totalmente liso (sem *features*), sendo que em alguns casos foi necessário alterar o ambiente (com lixeiras dispostas pelo próprio prédio), para permitir que o robô continuasse com o rastreamento. A Figura 5.37 ilustra este caso, em que uma lixeira é colocada para que o robô não se perca ao entrar no corredor K. A aresta K foi a única em que tal modificação foi necessária, tanto ao entrar por G quanto por R.
- Alguns corredores possuem curvas muito estreitas, o que dificulta transladar o necessário para triangular as *features* antes que encontre uma parede.
- As marcações em alto relevo no solo destinadas a pessoas com deficiência visual exigem que o robô ande mais devagar. Em alguns casos, mesmo em baixa velocidade o robô acabou se perdendo devido a vibração da câmera ao cruzar as marcações.
- O fato de o prédio possuir ambientes tanto fechados quanto abertos acarretou grandes mudanças de iluminação, ao passar de um ambiente fechado para um ambiente com vista para fora do prédio. Foi observado um caso em que houve perda de rastreamento devido a mudança da iluminação. A Figura 5.38 ilustra como a imagem escurece ao ter em vista um local de fora do prédio.

- O fato de ser um ambiente híbrido fez também com que o robô acabasse mapeando ambientes fora do prédio, tais como árvores e postes de luz, resultando em *outliers* ao sobrepor o mapa gerado no *ground truth*.

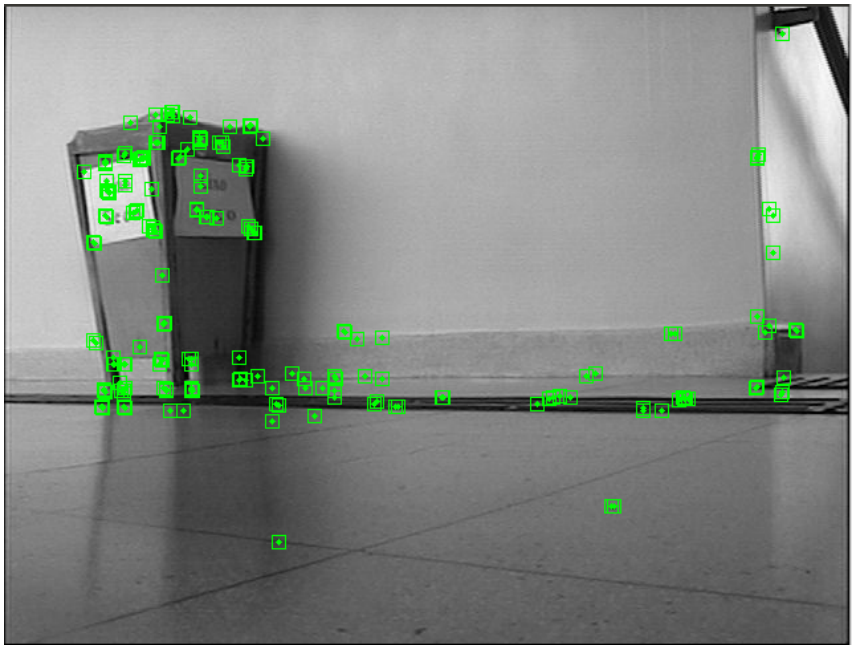


Figura 5.37: Alteração do ambiente em parede totalmente lisa, operação **30/05-14:30**. Fonte: Autor.

5.2.3 Câmera

A principal dificuldade apresentada pela câmera foi seu foco automático, percebido apenas na fase de análise dos resultados. Em um caso específico, o a correção do foco da câmera fez com o que o robô perdesse o rastreamento das *features*, alterando drasticamente a acurácia do mapa gerado. A Figura 5.39 ilustra este caso, em que é possível ver a imagem fora de foco.

5.2.4 Algoritmo ORB SLAM

Os principais problemas observados no algoritmo ORBSLAM nesse trabalho são listados a seguir.

- Em grande parte das operações analisadas anteriormente, ocorreu o problema de perda de escala do mapa, de forma que uma parte do ambiente começa a ser mapeada com um tamanho diferente, em geral menor, do que realmente é. Em alguns casos este problema foi corrigido pelo fechamento de *loop*, porém nem sempre a correção

Figura 5.38: Imagem escura devido a visão para fora do prédio, operação **23/05-09:19**. Fonte: Autor.

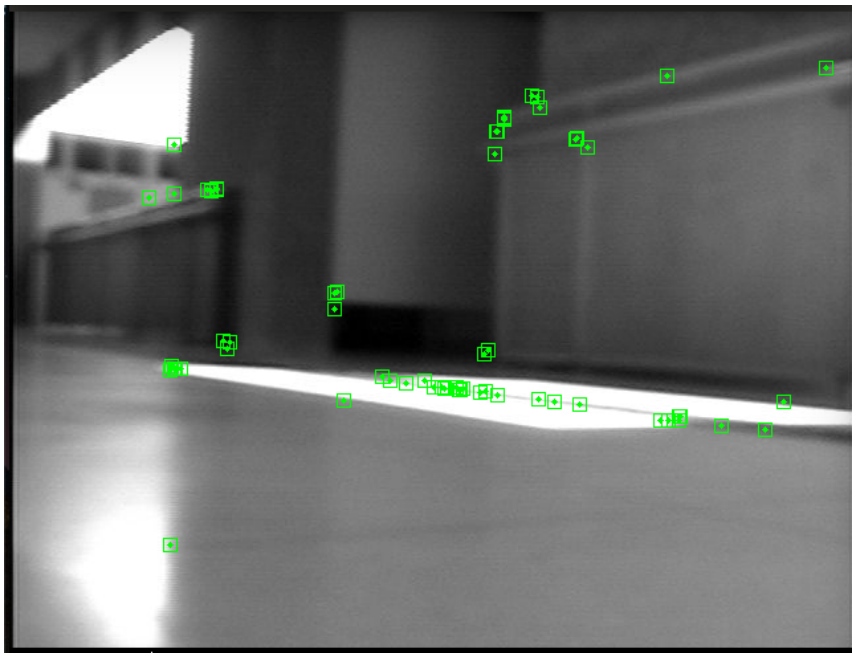


Figura 5.39: Correção automática de foco realizado pela câmera durante a gravação, operação **2/06-10:49**. Fonte: Autor.

era consistente, como pode ser visto na operação **30/05-14:30**. Em outros casos, apesar do erro de escala aparecer em menor grau, a incapacidade de se relocalizar

no mapa fortalece o erro, como pode ser visto na operação **29/05-17:42**.

- Em alguns casos o algoritmo não conseguiu fechar um *loop* ao passar novamente pelo mesmo local. Em outros fechou com inconsistência no mapa, normalmente quando houve discrepância na escala nos corredores imediatamente anteriores. Num terceiro caso, pouco frequente, o fechamento de *loop* ocorreu no nó errado, o que gerou problemas gravíssimos no resultado do mapa, como pode ser visto na operação **16/05-13:02**.

Capítulo 6

Conclusão e Trabalhos Futuros

SLAM se refere ao problema de obter um mapa representativo de um ambiente ao mesmo tempo que localização é realizada no mapa. Ao utilizar um mapa, várias tarefas podem ser realizadas por um robô, como busca e salvamento em áreas de desastre, exploração de minas abandonadas, navegação em carros autônomos, robôs para limpeza doméstica e robôs para serviços de manipulação de objetos. Apesar de bastante abordado, não foi totalmente resolvido para todos os tipos de mapa, ambiente, precisão, performance, e outros fatores. Alguns dos problemas abertos são a exploração inteligente do ambiente, eliminando a necessidade de um operador humano e a capacidade de dar valor semântico aos objetos sendo observados no mapa. Atualmente, muito enfoque é dado à resolução do SLAM utilizando câmeras, chamado de vSLAM. Alguns problemas surgiram da utilização única de câmeras, como inconsistências na rotação pura, inicialização do mapa e ambiguidade da escala. Mesmo assim, a possibilidade de paralelizar o processamento e a utilização de métodos de Visão Computacional como BA proveram resultados melhores do que com a utilização de filtros.

Nesta monografia foi utilizado o algoritmo ORB SLAM como algoritmo base para realizar SLAM visual no prédio CiC/Est utilizando o robô Pioneer 3-AT disponível no LAICO. Configurações de rede, câmera e calibração foram realizados para permitir a operação do sistema. Resultados qualitativos foram obtidos e divididos em 5 categorias, operação completa no prédio com geração de mapa completo, operação completa com mapa incompleto, completa com mapa irreconhecível, incompleta com mapa completo e incompleta com mapa completo. Diante de análises de fechamento de *loop*, relocalização e inconsistência de escala das doze operações, as principais dificuldades relacionadas a operação, ao prédio, a câmera e ao algoritmo foram abordadas. Dentre os problemas de operação está o fato de que toda rotação necessita de translação suficiente até que *features* sejam trianguladas, o que é uma limitação para um usuário comum pilotar o robô durante a exploração do ambiente. Além disto, a passagem de muitos pedestres

pelo prédio acabou por piorar alguns resultados. Sobre o prédio, situações de paredes totalmente lisas, curvas muito estreitas, linhas para pessoas com deficiência no chão e mudança repentina de claridade, devido ao fato de ser um ambiente híbrido, foram fontes de problemas. Da câmera, está o foco automático realizado, não percebido durante a operação. Para o algoritmo, o principal problema foi a ambiguidade de escala, inerente aos algoritmos de vSLAM monocular.

Para trabalhos futuros seria interessante utilizar câmeras estéreo, reduzindo os problemas encontrados de ambiguidade de escala e inicialização, além do erro acumulado ser menor. Alguns dos principais vSLAM monocular, como o LSD SLAM[47] e o ORB SLAM[48], já possuem versões em modo estéreo e RGB-D. Para o problema de claridade, métodos de equalização de histograma podem ser acoplados para diminuir a diferença temporal entre os frames, o que pode ajudar também na consistência dos *keyframes* e facilitar o fechamento de *loop*. Para melhorar o mapeamento, pode ser utilizada a média dos parâmetros da câmera obtidos da calibração, realizando-a diversas vezes, em vez de utilizar os parâmetros obtidos de uma única calibração apenas. Para o problema de paredes totalmente lisas, talvez utilizar uma heurística para decidir realizar o rastreamento e mapeamento utilizando a imagem inteira em vez de apenas features. Outro trabalho futuro seria adicionar um algoritmo de exploração que leve em consideração as dificuldades apresentadas e analisadas aqui, como desacelerar automaticamente em curvas e nas marcações para deficientes visuais no chão. Seria interessante também utilizar o mapa gerado para realizar limpeza no prédio ou um robô serviçal, capaz de realizar entregas aos docentes em suas salas.

Referências

[1] Mur-Artal, Raul, JMM M M M M Montiel e Juan D. Tardos: *Orb-slam: a versatile and accurate monocular slam system*. IEEE Transactions on, 31(5):1147–1163, 2015, ISSN 15523098. http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=7219438{%}0Ahttp://ieeexplore.ieee.org/xpls/abs{%}7B{_}{%}7Dall.jsp?arnumber=7219438. x, 9, 24, 25, 51, 53

[2] Cadena, C, L Carlone, H Carrillo e Y Latif: *Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age*. IEEE Transactions, 32(6):1309–1332, 2016. http://ieeexplore.ieee.org/abstract/document/7747236/. xiii, 3, 4, 5, 9, 25

[3] Taketomi, Takafumi, Hideaki Uchiyama e Sei Ikeda: *Visual SLAM algorithms: a survey from 2010 to 2016*. IPSJ Transactions on Computer Vision and Applications, 9(1):16, 2017, ISSN 1882-6695. http://ipsjcva.springeropen.com/articles/10.1186/s41074-017-0027-2. xiii, 9, 10, 20, 25

[4] Younes, Georges, Daniel Asmar e Elie Shammas: *A survey on non-filter-based monocular Visual SLAM systems*. (2012), 2016. http://arxiv.org/abs/1607.00470. xiii, 8, 11, 25

[5] Mitchell, Bradley: *Wireless standards 802.11a, 802.11b/g/n, and 802.11ac*. https://www.lifewire.com/wireless-standards-802-11a-802-11b-g-n-and-802-11ac-816553, June 2018. Accessed on 2018-06-20. xiii, 40

[6] Mitchell, Bradley: *How fast is ethernet networking?* https://www.lifewire.com/how-fast-is-ethernet-817549, March 2018. Accessed on 2018-06-20. xiii, 40

[7] Fisher, Tim: *Usb: Everything you need to know*. https://www.lifewire.com/universal-serial-bus-usb-2626039, March 2018. Accessed on 2018-06-20. xiii, 40

[8] Bongard, J.: *Probabilistic robotics. sebastian thrun, wolfram burgard, and dieter fox. (2005, mit press.) 647 pages.* Artificial Life, 14(2):227–229, April 2008, ISSN 1064-5462. 3, 4, 5, 6, 7, 8

[9] Kleiner, Alexander, Christian Dornhege e Sun Dali: *Mapping disaster areas jointly: RFID-coordinated SLAM by humans and robots*. Em *SSRR2007 - IEEE International Workshop on Safety, Security and Rescue Robotics Proceedings*, páginas 1–6, 2007, ISBN 9781424415694. 3

[10] Thrun, Sebastian, Scott Thayer, William Whittaker, Christopher Baker, Wolfram Burgard, David Ferguson, Dirk Hähnel, Michael Montemerlo, Aaron Morris, Zachary Omohundro, Charlie Reverte e Warren Whittaker: *Autonomous Exploration and Mapping of Abandoned Mines*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.2789{&}rep=rep1{&}type=pdf. 3

[11] Bresson, Guillaume, Zayed Alsayed, Li Yu e Sebastien Glaser: *Simultaneous Localization And Mapping: A Survey of Current Trends in Autonomous Driving*. IEEE Transactions on Intelligent Vehicles, XX(X):1–1, 2017, ISSN 2379-8904. http://ieeexplore.ieee.org/document/8025618/. 3

[12] Lee, Seongsoo, Sukhan Lee e Seungmin Baek: *Vision-based kidnap recovery with SLAM for home cleaning robots*. Journal of Intelligent and Robotic Systems: Theory and Applications, 67(1):7–24, 2012, ISSN 09210296. 3

[13] Chen, Yuncong: *Algorithms for Simultaneous Localization and Mapping*. 2013. http://cseweb.ucsd.edu/{~}yuc007/documents/re{_}report.pdf. 4, 5, 6, 7

[14] Yousif, Khalid, Alireza Bab-Hadiashar e Reza Hoseinnezhad: *An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics*. Intelligent Industrial Systems, 1(4):289–311, 2015, ISSN 2363-6912. http://link.springer.com/10.1007/s40903-015-0032-7. 4, 5, 7, 9

[15] Durrant-Whyte, Hugh e Tim Bailey: *Simultaneous localization and mapping (SLAM): part I The Essential Algorithms*. Robotics & Automation Magazine, 2:99–110, 2006, ISSN 10709932. 5, 7

[16] Chong, T. J., X. J. Tang, C. H. Leng, M. Yogeswaran, O. E. Ng e Y. Z. Chong: *Sensor Technologies and Simultaneous Localization and Mapping (SLAM)*. Procedia Computer Science, 76(Iris):174–179, 2015, ISSN 18770509. 5, 6

[17] Smith, Randall, Matthew Self e Peter Cheeseman: *Estimating Uncertain Spatial Relationships in Robotics*. Machine Intelligence and Pattern Recognition, 5(C):435–461, 1988, ISSN 09230459. 6

[18] Montemerlo, Michael, Sebastian Thrun, Daphne Koller e Ben Wegbreit: *FastSLAM: A factored solution to the simultaneous localization and mapping problem*. Proc. of 8th National Conference on Artificial Intelligence/14th Conference on Innovative Applications of Artificial Intelligence, 68(2):593–598, 2002, ISSN 10450823. http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:FastSLAM+:+A+Factored+Solution+to+the+Simultaneous+Localization+and+Mapping+Problem{#}0. 7

[19] Lu, F e E Milios: *Globally Consistent Scan Matching For Environment Mapping*. Autonomous Robots, 4(4):333–349, 1997. 8

[20] Scaramuzza, Davide e Friedrich Fraundorfer: *Tutorial: Visual odometry*. IEEE Robotics and Automation Magazine, 18(4):80–92, 2011, ISSN 10709932. 9, 11, 12

[21] Triggs, Bill, Philip Mclauchlan, Richard Hartley e Andrew Fitzgibbon: *Bundle Adjustment — A Modern Synthesis*, volume 34099. 2000, ISBN 3540444807. 9

[22] Strasdat, Hauke, J. M.M. Montiel e Andrew J. Davison: *Visual SLAM: Why filter?* Image and Vision Computing, 30(2):65–77, 2012, ISSN 02628856. `http://dx.doi.org/10.1016/j.imavis.2012.02.009`. 9

[23] Chum, Ondřej, Tomáš Pajdla e Peter Sturm: *The geometric error for homographies.* Computer Vision and Image Understanding, 97(1):86–102, 2005, ISSN 10773142. 9

[24] Hartmann, Jan, Jan Helge Klussendorff e Erik Maehle: *A comparison of feature descriptors for visual SLAM.* 2013 European Conference on Mobile Robots, ECMR 2013 - Conference Proceedings, páginas 56–61, 2013. 10

[25] Lowe, David G: *Distinctive Image Features from Scale-Invariant Keypoints.* International Journal of Computer Vision, 60(2):91–110, 2004, ISSN 0920-5691. 10

[26] Rublee, E, V Rabaud e K Konolige: *ORB : an efficient alternative to SIFT or SURF.* Intl. Conf. Computer Vision, páginas 1–5, 2011. 10

[27] Nisté, David: *An Efficient Solution to the Five-Point Relative Pose Problem.* IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 26, Issue: 6, June 2004), 26(6):756 – 770, 2004. `http://www.ee.oulu.fi/research/imag/courses/Sturm/nister04.pdf`. 14

[28] Fischler, Martin a e Robert C Bolles: *Random Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography.* Communications of the ACM, 24(6):381–395, 1981, ISSN 00010782. 14

[29] Rosten, Edward e Tom Drummond: *Machine learning for high-speed corner detection.* Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3951 LNCS:430–443, 2006, ISSN 16113349. 14

[30] Lucas, Bruce D e Takeo Kanade: *An Iterative Image Registration Technique with an Application to Stereo Vision.* Imaging, 130(x):674–679, 1981, ISSN 17486815. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.49.2019{&}rep=rep1{&}type=pdf`. 15

[31] Eade, Ethan: *Lie groups for 2d and 3d transformations.* `http://ethaneade.com/lie.pdf`, May 2017. Accessed on 2018-06-20. 15, 18

[32] Hartley, Richard e Andrew Zisserman: *Multiple View Geometry in Computer Vision.* Cambridge University Press, New York, NY, USA, 2ª edição, 2003, ISBN 0521540518. 16

[33] Strasdat, Hauke, Andrew J. Davison, J. M.M. Montiel e Kurt Konolige: *Double window optimisation for constant time visual SLAM.* Proceedings of the IEEE International Conference on Computer Vision, páginas 2352–2359, 2011, ISSN 1550-5499. 23

[34] Davison, Andrew J, Ian D Reid, Nicholas D Molton e Olivier Stasse: *MonoSLAM: Real-Time Single Camera {SLAM}.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 29(6):1052–1067, 2007, ISSN 0162-8828. http://dx.doi.org/10.1109/TPAMI.2007.1049. 25

[35] Klein, Georg e David Murray: *Parallel tracking and mapping for small AR workspaces.* 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR, 2007, ISSN 1342-6907. 25

[36] Newcombe, Richard A, Steven J Lovegrove e Andrew J Davison: *DTAM: Dense Tracking and Mapping in Real-Time.* páginas 2320–2327, 2011. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.445.1843{&}rep=rep1{&}type=pdf. 25

[37] Salas-Moreno, Renato F., Richard A. Newcombe, Hauke Strasdat, Paul H.J. Kelly e Andrew J. Davison: *SLAM++: Simultaneous localisation and mapping at the level of objects.* Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, páginas 1352–1359, 2013, ISSN 10636919. 25

[38] Forster, Christian, Matia Pizzoli e Davide Scaramuzza: *SVO: Fast semi-direct monocular visual odometry.* Proceedings - IEEE International Conference on Robotics and Automation, 2014, ISSN 10504729. 25

[39] Engel, Jakob, Thomas Schöps e Daniel Cremers: *LSD-SLAM: Large-Scale Direct monocular SLAM.* Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8690 LNCS(PART 2):834–849, 2014, ISSN 16113349. 25

[40] Daniel, Herrera C., Kihwan Kim, Juho Kannala, Kari Pulli e Janne Heikkilä: *DT-SLAM: Deferred triangulation for robust SLAM.* Proceedings - 2014 International Conference on 3D Vision, 3DV 2014, páginas 609–616, 2015. 25

[41] Concha, Alejo e Javier Civera: *DPPTAM: Dense piecewise planar tracking and mapping from a monocular sequence.* IEEE International Conference on Intelligent Robots and Systems, 2015-Decem:5686–5693, 2015, ISSN 21530866. 25

[42] Engel, Jakob, Vladlen Koltun e Daniel Cremers: *Direct Sparse Odometry.* 2016, ISSN 0162-8828. http://arxiv.org/abs/1607.02565. 25

[43] Quigley, Morgan, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler e Andrew Mg: *ROS: an open-source Robot Operating System.* Icra, 3(Figure 1):5, 2009, ISSN 0165-022X. http://pub1.willowgarage.com/{~}konolige/cs225B/docs/quigley-icra2009-ros.pdf. 26

[44] Fernandez, Enrique, Luis Sanchez Crespo, Anil Mahtani e Aaron Martinez: *Learning ROS for Robotics Programming - Second Edition.* Packt Publishing, 2nd edição, 2015, ISBN 1783987588, 9781783987580. 27

[45] Fairchild, C. e T.L. Harman: *ROS Robotics By Example.* Packt Publishing, 2016, ISBN 9781785286704. https://books.google.com.br/books?id=3_1vDQAAQBAJ. 31

[46] Andrew: *Compression ratio for different jpeg quality values.* https://www.graphicsmill.com/blog/2014/11/06/Compression-ratio-for-different-JPEG-quality-values, November 2014. Accessed on 2018-06-20. 42

[47] Caruso, David, Jakob Engel e Daniel Cremers: *Large-scale direct SLAM for omnidirectional cameras.* IEEE International Conference on Intelligent Robots and Systems, 2015-Decem:141–148, 2015, ISSN 21530866. 80

[48] Mur-Artal, Raul and Tardós, Juan': *ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras.* Analysis, 27(6):988–992, 2005, ISSN 0162-8828. 80

Anexo I

Código

Código utilizado disponível publicamente em `https://github.com/gabrielmirandat/tcc_ws`.