4
Testes e Análise dos resultados

4.1
Hardware

O rastreamento em áreas maiores que uma mesa, em geral, precisam de um equipamento portátil, e de preferência sem fio. No entanto, a portabilidade do sistema não será abordada nesse trabalho, pois já existem diversos outros trabalhos tratando desse tema, como por exemplo, sistemas cliente-servidor usando PDAs ou Celulares[18], sistemas que usam laptops robustos[32] ou carrinhos equipados com um computador de mesa[29].

Para os testes, utilizamos um computador de quatro núcleos de 2.4Ghz com 3Gb de memória volátil(RAM). Entretanto, utilizamos apenas dois processos paralelos visto que a maioria dos laptops tem apenas dois núcleos. A câmera utilizada foi a Fire-I da fabricante Unibrain na resolução de 320x240 pixels a 30 fps. A lente usada tem distância focal de 2.1 mm. A câmera foi anexada a um monitor portátil com o intuito de permitir que o usuário do sistema veja em tempo real a imagem da câmera anexada às suas costas, como pode ser visto na figura 4.1

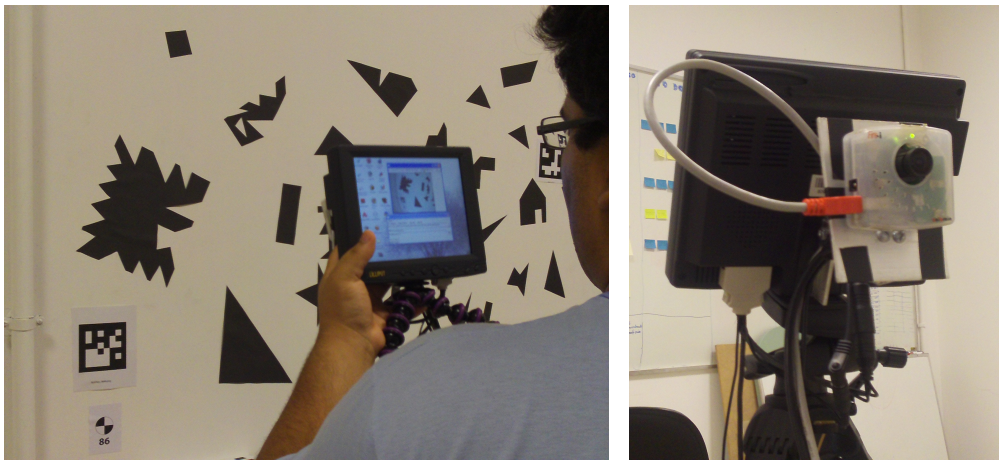


Figura 4.1: Equipamento utilizado.

4.2 Implementação

A implementação da abordagem proposta é bastante complexa por incluir diversos algoritmos diferentes. Nessa seção vamos descrevemos os principais módulos do programa, mas vamos apenas citar as bibliotecas que precisaram ser adicionadas por dependência indireta. O nome dos módulos estão em inglês uma vez que o software foi escrito nessa língua tornando necessário preservar os nomes originais para viabilizar o uso dessa dissertação como base teórica para implementação. Por fins de didática, vamos chamar de módulo o conjunto de uma ou mais classes C++ que juntas desempenham um determinado papel na implementação.

4.2.1 Arquiterura

A arquitetura do programa foi pensada no sentido de garantir um acoplamento mínimo entre os diferentes algoritmos usados nesse trabalho, viabilizando a implementação e teste dos módulos individualmente. Além disso, essa arquitetura permite trocar os rastreadores e o algoritmo de SLAM utilizado caso seja desejado. Outra propriedade requerida é o controle do acesso aos dados, que precisa ser gerenciado por conta do funcionamento em dois processos em paralelo. Na figura 4.2 estão demonstrados os principais módulos do programa escrito e os dados que eles trocam.

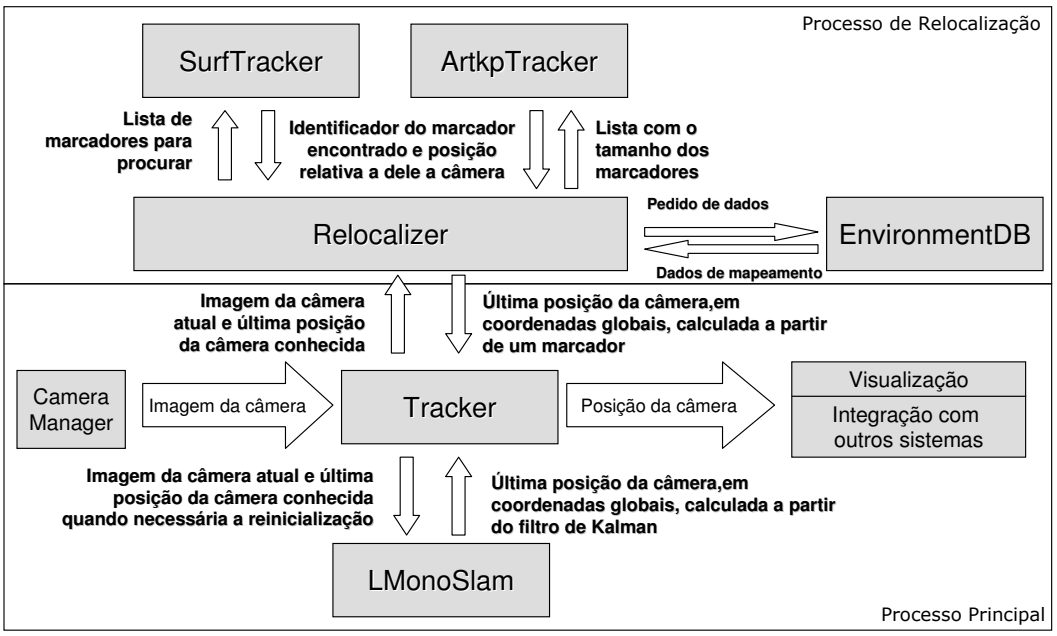


Figura 4.2: Arquitetura resumida da implementação do algoritmo proposto.

4.2.2 EnvironmentDB

Este módulo será explicado com um pouco mais de detalhes por não haver nenhuma referência sobre ele nas seções anteriores. Sua função é armazenar, carregar e salvar os dados de mapeamento do ambiente. Os dados por ele gerenciados são fornecidos pelo usuário por uma interface escrita em IUP que permite adicionar os dados medidos sobre os marcadores. O armazenamento em arquivo é feito em XML. Foi usada a biblioteca escrita por Marcin Kalicinski, RapidXml, como analisador(parser) e escritor(writer). Essa biblioteca é considerada uma das mais rápidas bibliotecas de XML no momento. Um dos motivos é que ela não tem validação do formato do arquivo, mas como usamos um editor especialmente escrito para o arquivo adotado, essa validação não é necessária. No entanto o principal motivo para termos escrito um editor foi a necessidade de marcar as coordenadas mapeadas nos marcadores naturais.

O módulo EnvironmentDB só responde a pedidos do módulo Relocalizer sobre os marcadores em tempo real uma vez que os dados de mapeamento não são alterados em tempo de execução do rastreamento. Ele também é responsável por calcular a posição global dos marcadores baseado nos pontos amostrados pela estação total. Esse módulo aceita dois tipos de pedido de dados. O primeiro é a lista de marcadores naturais possíveis de estar na cena. O Relocalizer informa a ultima posição conhecida da câmera e o instante de tempo que essa posição se refere. O EnviromentDB responde com um objeto NaturalMarkerCandidates, que é um vetor com o nome dos marcadores e os a lista de pontos chave de cada um. O segundo tipo de pedido é uma requisição de dados sobre um marcador específico. Como dado de entrada é fornecido o nome do marcador. No caso do marcador natural, é o nome que foi fornecido dentro do NaturalMarkerCandidates e no caso dos ficiduais, é o número que é extraído da matriz de quadrados pretos e brancos presentes. Como dado de saída é fornecido uma instância do objeto EnvMarker. Esse objeto é uma abstração de todos os marcadores e cria uma interface única para acesso aos marcadores. Ele fornece a posição do marcador em coordenadas globais, a matriz de transformação que leva do sistema de coordenadas do marcador para o global e a lista de pontos que são bons para serem rastreados pelo SLAM.

4.2.3 ArtkpTracker

Este módulo é responsável pela interação com o rastreador de marcadores fiduciais. A biblioteca utilizada como rastreador foi a ARToolKitPlus[37].

Usamos a biblioteca sem nenhuma modificação interna. Ela fornece uma API suficientemente vasta para diversos usos. O nosso processamento foi feito usando a detecção de marcadores na qual a biblioteca não conhece a priori nenhum marcador específico, ela apenas procura por algum marcador na imagem. Então, após ela detectar algum marcador, é informado a ela qual o tamanho daquele marcador e requisitado que ela faça a calibração de câmera. O resultado dessa calibração é formatado como uma matriz ModelView do OpenGL.

4.2.4 SurfTracker

Este módulo é responsável pelo rastreamento dos marcadores naturais. A parte de detecção dos pontos chave foi feita usando uma biblioteca chamada OpenSURF[16]. O cálculo de homografia usando DLT e RANSAC foi feita com o auxilio do OpenCV[8]. Os outros cálculos foram implementados como descrito nesse trabalho. Optamos por também apresentar o resultado dessa calibração no formato de uma matriz ModelView do OpenGL com objetivo de padronizar a saída de dados com o ArtkpTracker. Esse formato é o mais comum para bibliotecas de rastreamento destinadas à realidade aumentada, pois permite desenhar objetos sobre a imagem da câmera usando o OpenGL.

4.2.5 Relocalizer

Este módulo é responsável pelo Relocalizador explicado na seção 3.6.1. Ele encadeia os rastreadores e transforma os sistemas de coordenada dos marcadores em coordenadas globais usando os dados provenientes do EnvironmentDB. Ele contém a função principal do processo de relocalização. A biblioteca para criar os processos paralelos foi a Pthread for windows. Essa mesma biblioteca fornece uma API de semáforos para controlar o acesso nos espaços de memória compartilhada com o processo principal.

Apesar de termos uniformizado a forma de passar os parâmetros extrínsecos da câmera usando uma matriz ModelView do OpenGL é preciso converter esses parâmetros para o formato usado pelo algoritmo de SLAM de Davison. Na figura 4.3 são apresentados os três sistemas de coordenadas presentes que o módulo Relocalizer gerencia e garante que as informações cheguem a cada módulo no sistema de coordenada esperado.

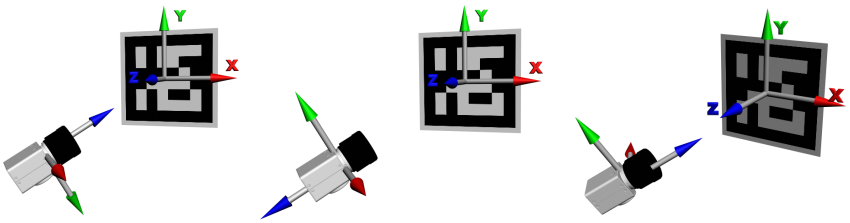


Figura 4.3: Sistema de coordenada do ArToolKitPlus, no meio, do SurfTracking e do OpenGL e, ao final, do SLAM de Davison

4.2.6 LMonoSlam

LMonoSlam é uma extensão da API fornecida pela biblioteca que o próprio Davison disponibiliza para seu algoritmo. LMonoSlam adicionada a capacidade de se reinicializar a partir de um marcador além de algumas outras funções de gerência do mapa, como por exemplo apagar todas os retalhos conhecidos do mapa. O algoritmo principal desse módulo foi descrito na seção 3.4, algoritmo 2

4.2.7 Tracker

Por fim, esse módulo é a interface de toda a implementação do Local SLAM com o resto do programa. Ele recebe imagens do Camera Manager e processa e entrega a posição da câmera naquele quadro caso o rastreamento usando Local SLAM tenha tido sucesso. Esse módulo também engloba a parte de heurística de reinicialização. Ele que decide quando usar os dados provenientes do Relocalizer para reinicializar o LMonoSlam

4.3 Estação Total

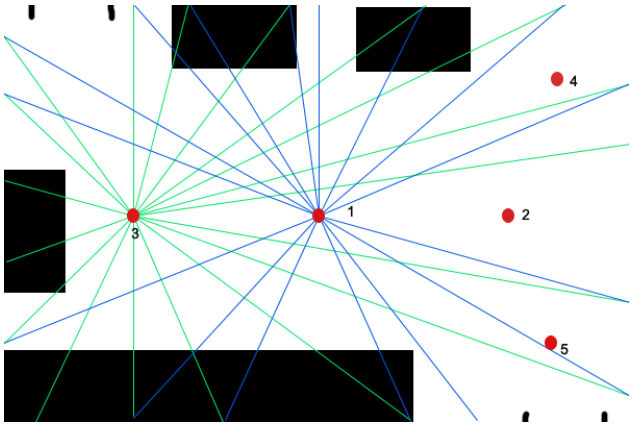
A estação total é na verdade um instrumento de Engenharia Cartográfica(Surveying). Ela foi projetada originalmente para medir prédios, grandes propriedades de terra ou mesmo cidades. Então decidimos testar a precisão real desse equipamento para fazer mapeamento 3D de uma sala, por conta de ser uma aplicação tão atípica, inclusive sem nenhuma referência bibliográfica. A estação total usada é SET630RK da marca Sokkia. Tal modelo apresenta a precisão de rotação de 6” de grau e o de distâncias de 5 mm para pequenas distâncias. Esses valores de precisão não representam muito para o nosso uso, os valores que queremos conhecer são os erros em relação ao x, y e z do sistema de coordenada gerado pela estação total. Nós supomos que esse erro deve

ser bem pequeno, mas como uma das motivações desse trabalho é encontrar livros dentro de uma biblioteca e a largura de um livro varia a menos de um centímetro, entendemos por bem fazer essa avaliação. Nós avaliamos esse erro em dois casos. O primeiro no caso de estar medindo pontos nas paredes ao redor da estação. Esses pontos são usados para reposicionar e necessitam ser conhecidos, para assim, permitir o reposicionamento a estação em diferentes posições da sala. O segundo no caso de a estação total medindo um único objeto pequeno a sua frente.

4.3.1 Erro de mapeamento de uma sala

Esse teste foi executado colando 22 marcadores no formato de cruz(figura 4.4(a)) nas paredes. Como não conhecemos a posição real dos marcadores, medimos todos os marcadores visíveis de 5 diferentes posições da sala, o mapa topográfico desses pontos pode ser visto na figura 4.4(b). Então usamos o programa de código aberto Java Graticule 3D que executa uma espécie de mínimos quadrados e informa um melhor x,y,z e o desvio padrão para cada ponto medido mais de uma vez. Esse programa leva em conta os valores de x, y e z calculado pela estação apenas como uma aproximação inicial. Na pratica, os dados que consideramos são os dados crus da estação, os ângulos e as distâncias, para fazer um ajuste de rede(free-network adjustment). As dimensões da sala de teste são: 7,5 m de comprimento; de 3,6 m de largura e 3,5 m de altura.

4.4(a): Marcador de referência



4.4(b): Mapa topográfico da sala de teste, em preto as mesas, em vermelho as posições da estação total e as linhas coloridas são as medidas feitas a partir da estação, apenas duas estações foram demonstradas

Figura 4.4: Condições de teste

As medidas foram colocadas no Java Graticule 3D(JAG3D) de forma

posições da estação	$\bar{\epsilon_x}$(mm)	$\bar{\epsilon_y}$(mm)	$\bar{\epsilon_z}$(mm)
1	3,363	3,712	4,761
1+2	3,249	3,561	4,580
1+2+3	2,600	2,816	3,879
1+2+3+4	2,336	2,527	3,424
1+2+3+4+5	2,154	1,978	2,932

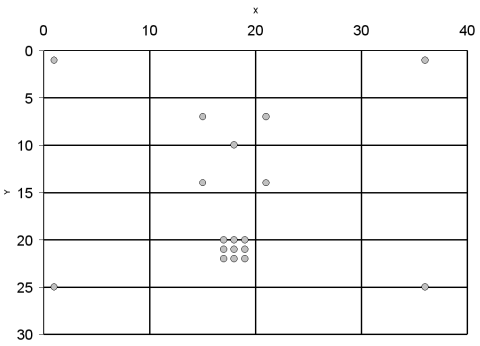
Tabela 4.1: Tabela de resultados de medida para sala

incremental no intuito de entender qual a influência de medidas redundantes para minimizar o erro. Os números das posições estão apresentados na figura 4.4(b). O eixo Z está apontando para o teto da sala, o eixo x no sentido de cima para baixo na figura 4.4(b).Como resultado apresentamos o erro médio ($\bar{\epsilon}$) para cada uma das componentes na tabela 4.1.

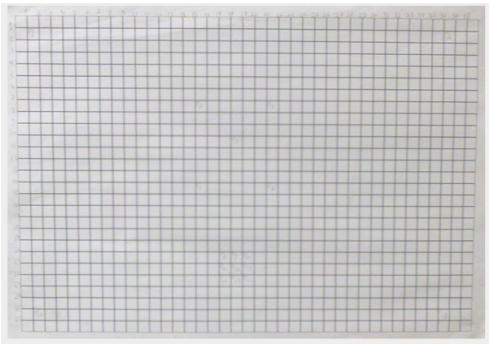
Na tabela 4.1 podemos observar que o erro para cada uma das medidas foi inferior a 5 mm e que ele diminui com a inclusão de mais medidas.

4.3.2
Erro de mapeamento em pequena escala

Para estudar o erro de medida em pequena escala, imprimimos em uma folha A3 um quadriculado com 1cm de separação entre as linhas. Como uma impressora laser comum tem um erro na impressão de cerca de 0,04 mm é aceitável usar esse quadriculado como referência. Na figura 4.5(b) podemos ver uma foto do papel quadriculado usado. Escolhemos 18 pontos no quadriculado para medir de diferentes distâncias, esses pontos podem ser vistos no gráfico da figura 4.5(a). As distâncias foram de 20m,12m e 8m frontalmente ao quadriculado. Além dessa medida, fizemos medições a 9m do papel quadriculado a 45 graus.



4.5(a): Pontos medidos na grade



4.5(b): Foto da folha A3 onde a grade foi impressa e medida

Figura 4.5: Teste para medida do erro do mapeamento em pequena escala

Como nesse teste não temos sistema de coordenada, para aferir a qualidade do resultado vamos comparar a distância de cada ponto para todos os outros 17 pontos do quadriculado. Essa forma de apresentar os dados é a matriz de distâncias. Na figura 4.6 podemos ver o resultado das medições.

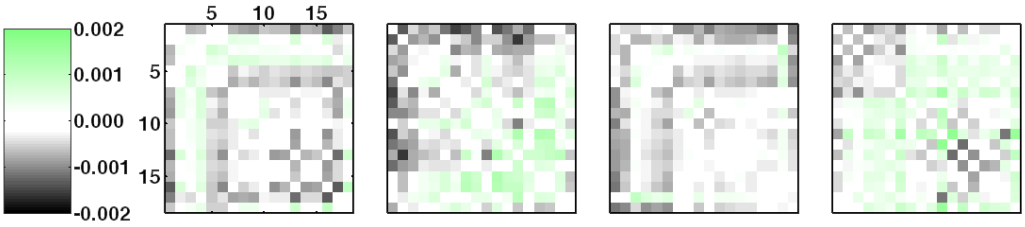


Figura 4.6: Da esquerda para direita as matrizes de distância das medidas à 20m, 12m, 8m e 9m. A unidade do gráfico está em metros.A matriz tem tamanho 18x18 já que foram 18 pontos medidos de cada distância

Podemos observar na figura 4.6 que a maior parte dos valores se manteve próximo a zero(branco). Analisamos também que todos os valores são inferiores a 2 mm, sendo que a maioria não ultrapassa 0.5 mm de erro.

4.4 Precisão na indicação de pontos 3D - Teste Qualitativo

Neste teste queremos avaliar a capacidade do algoritmo proposto indicar uma posição 3D. Para isso colamos sobre uma parede branca uma coleção de figuras abstratas em cartolina preta e marcadores fiduciais a cada 1,5 metros, como pode ser visto na figura 4.7. Os marcadores e algumas figuras foram mapeados usando a estação total. Essas figuras estão marcadas na figura 4.8 com um X. Na parede ortogonal foi medido apenas o triângulo e o trapézio. Em seguida, gravamos um vídeo da posição mais a esquerda da figura 4.8 até o fim da parede. Todos os pontos medidos estão sobre os mesmo planos, exceto os pontos na parede ortogonal. Com isso podemos ter uma boa estimativa sobre se os pontos estão reprojetados na direção correta.

Sobre esse vídeo executamos o algoritmo de Local SLAM com a heurística 'quando possível'(LSLAM-QP) e 'marcador ao centro'(LSLAM-MC). Para termos uma base de comparação, executamos também os rastreadores do ArToolKitPlus original(ARTKP), do SLAM original(SLAM-A) e o algoritmo de Local SLAM sendo reinicializado sempre que o ArToolKitPlus retorne uma calibração válida(LSLAM-ARTKP). O resultado de todas essas execuções estão no vídeo *ex1_parede_grande.avi*(Apendicê A). O resultado do SLAM original não foi incluído, pois ele perde o rastreamento entre o primeiro e

Figura 4.7: Parede de teste em pespectiva

Figura 4.8: Mosaico de fotos da parede maior com as figuras medidas marcadas com um 'X'. Na parede ortogonal foi medido apenas o triângulo e o trapézio.

o segundo marcador e como ele não tem nenhuma política de reinicialização ele não retorna mais o rastreamento.

As conclusões seguintes foram obtidas da análise do vídeo . O resultado do ARTKP e LSLAM-ARTKP é extremamente parecido, a diferença se deve apenas ao fato que os dois algoritmos usam algoritmos de calibração diferentes. Como era de se esperar nenhum dos dois funciona sem marcadores na imagem. O ARTKP , por sua própria formulação, precisa de um marcador e o LSLAM-ARTKP não funciona, pois ele tem oportunidade de gerar um mapa da cena antes que o marcador saia da imagem, já que ele fica reinicializado até o último instante antes do marcador sair da imagem.

Outra conclusão é que o SLAM-A é inviável para rastreamento em ambientes conhecidos, pois ele não conseguiu manter uma reprojeção dos pontos mapeados minimamente suficiente para o usuário se orientar.

Outro ponto que pode ser observado que é o resultado do LSLAM-MC e do LSLAM-QP são bastante parecidos com o do ARTKP quando o marcador está aparente na imagem. No entanto, o LSLAM-MC apresenta erros de

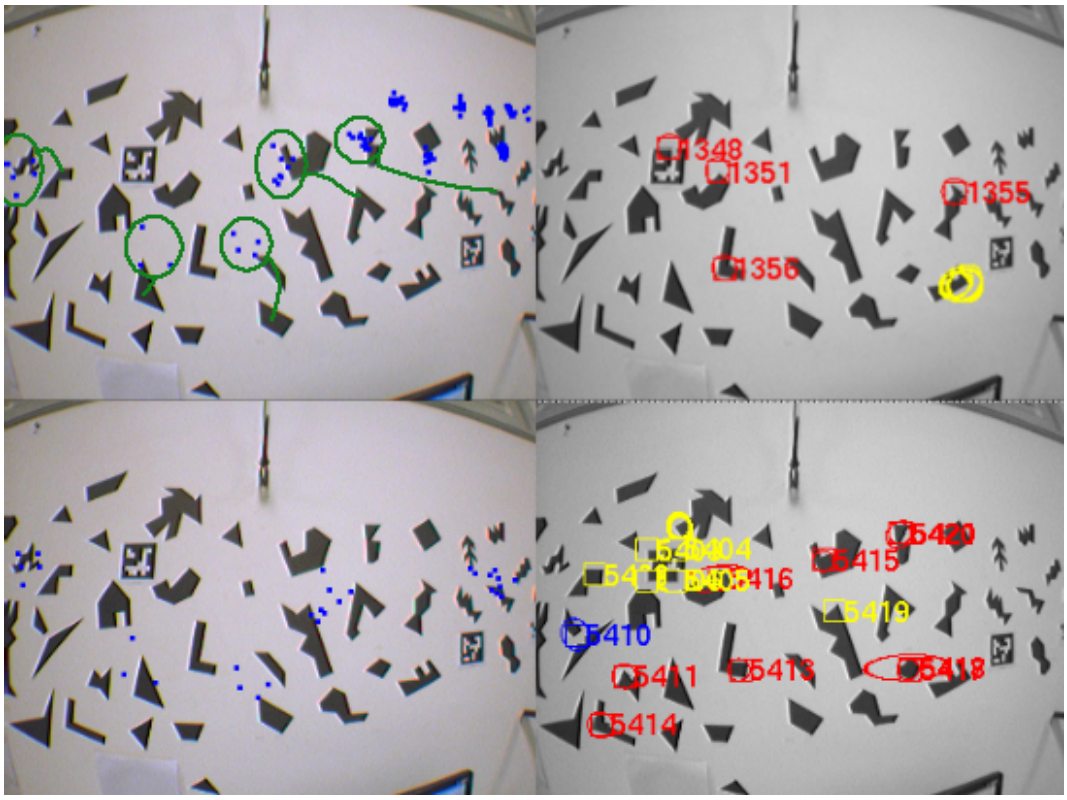


Figura 4.9: Diferença no mapeamento gerado pelas diferentes políticas de reinicialização. Em cima à esquerda, a reprojeção das quinas dos objetos conhecidos na parede pelo algoritmo LSLAM-QP e à direita o mapa de características acompanhas pelo SLAM correspondente. Abaixo, o mesmo para o algoritmo LSLAM-MC.

calibração esporádicos nessa situação por não estar se reinicializando sempre. Em contrapartida, LSLAM-MC apresenta maior robustez ao reprojetar pontos mais distantes do marcador no qual ele foi reinicializado, pois ele consegue mapear pontos em toda volta do marcador. O LSLAM-QP , normalmente, não consegue fazer esse mapeamento em todos os lados do marcador, pois ele só pára de se reinicializar com o marcador muito perto da borda da imagem ou pequeno demais. A figura 4.9 mostra um momento em que a diferença entre os algoritmo fica evidenciada. No apêndice A está explicado como entender os artefatos desenhados sobre a imagem da câmera. Podemos observar que no momento em que o marcador ficou pequeno demais para uma reinicialização, o algoritmo LSLAM-MC, na parte inferior da figura, tinha muito mais pontos para rastrear que o LSLAM-QP. Como resultado podemos ver na reprojeção dos pontos que o erro do LSLAM-QP é muito maior, principalmente nos pontos mais distantes. No caso do LSLAM-QP aparecem na imagem a reprojeção de vários cantos de vários objetos que nem estão na cena, enquanto no LSLAM-

MC isso não acontece, pois o plano da parede ainda está razoavelmente correto.

Outra conclusão é que a precisão absoluta na indicação de pontos 3D dos algoritmos de Local SLAM foi cerca de 5 a 10 cm para pontos próximos ao marcador de inicialização. Para pontos mais distantes, como os pontos do final da parede na figura 4.9 à direita superior, foi possível apenas ter uma indicação grosseira das suas posições, mas sendo impossível identificá-los.

4.5 Precisão na indicação de pontos 3D - Teste Quantitativo

Nesse teste vamos avaliar a qualidade do mapa gerado pelo algoritmo de SLAM. Para esse teste colamos em uma parede branca triângulos e retângulos em cartolina preta. Em seguida medimos a posição 3D de todas as quinas deles. na extremidade inferior do conjunto das figuras colamos um marcador fiducial de 8 cm. Para termos como estimar manualmente a trajetória da câmera ao longo do vídeo fixamos a câmera a um tripé de iluminação que tem uma haste telescópica de 80 cm. Assim, sabemos que a posição inicial é de cerca de 1,05 m de altura e a posição final de 1,85 m de altura. A coordenada em x e y não deveriam se alterar já que o movimento foi feito perpendicular ao chão. Então gravamos um vídeo com a movimentação de baixo para cima e executamos o algoritmo de SLAM sobre ele com uma única reinicialização no início. O resultado dessa execução estão no vídeo *ex2_parede_pequena.avi*. Nesse vídeo além da movimentação para cima ocorrem eventuais vibrações para direita e para esquerda(eixo y) pois a haste permite a câmera rodar sobre o eixo da dela, mas tentamos mante-lá o mais estática possível.

Para avaliação dos dados quantitativos, além do vídeo gerado pela nossa implementação, também salvamos a posição da câmera a cada quadro e o mapa do SLAM ao final do vídeo. Sendo que desprezamos os retalhos do mapa nos quais o próprio algoritmo de SLAM indicava que as medidas eram ruins, ou seja tinham covariança entre o x,y,z com valores altos.

O primeiro gráfico que fizemos relaciona o erro em cada retalho mapeado pelo SLAM com a distância dele até o centro do marcador de onde a inicialização foi feita. Esse gráfico pode ser visto na figura 4.10. O erro a 80 cm do marcador tem cerca de 20 cm. Nesse cenário, teríamos que ter , aproximadamente, 1 marcador a cada 1,60 m para podermos acessar qual ponto do espaço 3D com um erro de 20 cm. Para um cenário comercial de procurar objetos em ambientes ainda são muito marcadores. No entanto 20 cm de erro é um valor aceitável de erro dado que é possível encontrar um caixa ou mesmo a prateleira do item que se esta buscando.

Para entender melhor de onde é o erro na calibração de câmera, dese-

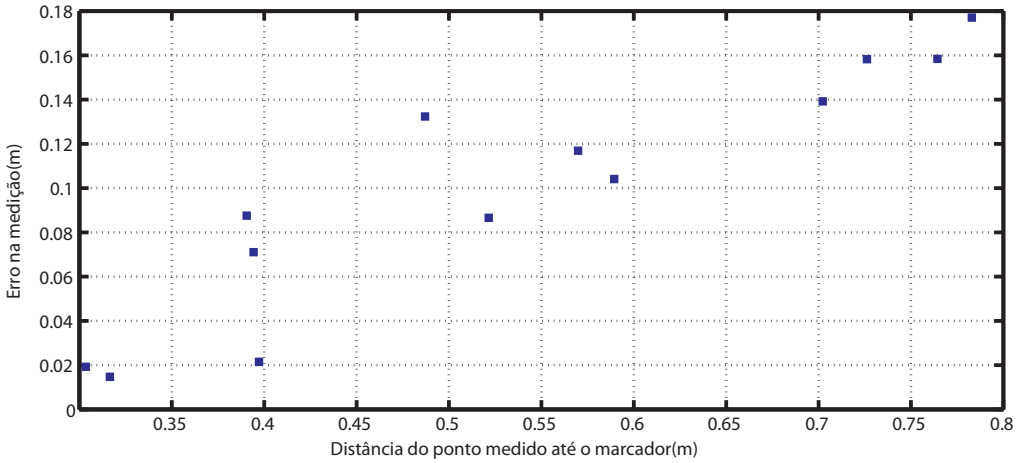
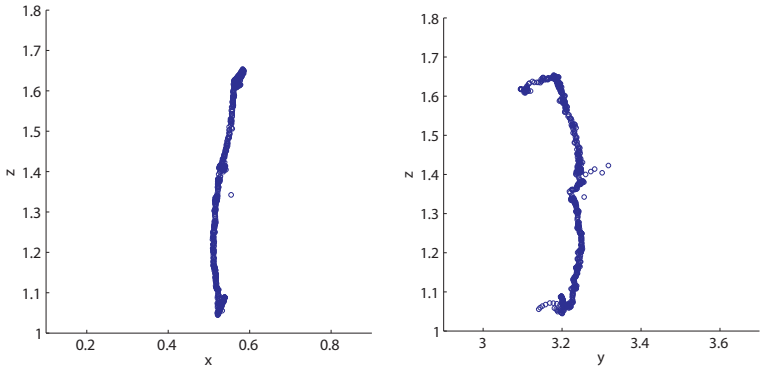


Figura 4.10: Gráfico do erro em cada retalho mapeado pelo SLAM pela distância ao marcador inicial.

nhamos na figura 4.5 o gráfico da trajetória da câmera no plano altura(z) por x e z por y. O gráfico que devia ser uma linha reta variando apenas em z também, varia em x cerca de 7 cm e cerca de 20 cm em y. Esse erro é desprezível na localização de um robô ou uma pessoa numa movimentação dentro de um edifício. Por conta disso que os algoritmos de SLAM são algoritmos de estimação da posição da câmera para navegação, ao invés de serem usados para reconstrução geométrica. O erro é um pouco maior no eixo y devido à liberdade de movimentação da haste, se observarmos no vídeo vemos que nas regiões de maior variação nesse eixo o vídeo apresenta vibrações na câmera.



4.11(a): Plano XZ 4.11(b): Plano YZ

Figura 4.11: Projeção da trajetoria da câmera

4.6 Inicialização com marcadores naturais

Marcadores naturais não podem ser detectados em tempo real. Então, em teoria, movimentos entre a câmera e o marcador poderiam impossibilitar a inicialização. No entanto, o algoritmo de SLAM de Davison tem o recurso de busca ativa, que procura ao redor do lugar onde deveria estar o retalho pela posição correta dele. Esse aspecto do algoritmo será testado nessa seção. Fizemos um vídeo com um marcador fiducial e em sequência com um natural. Tentamos fazer os mesmos tipos de movimentos com a câmera ao redor dos marcadores. O marcador natural é uma caixa de brinquedo com quatro adesivos formando um quadrado de 15cm (figura 4.12), o mesmo tamanho do marcador fiducial. A imagem de referencia da caixa tem cerca de 200 pixels de altura. Como estamos estudando o problema da coerência temporal, com esses adesivos deixamos em condições parecidas os marcadores no quesito sobre os retalhos que serão rastreados pelo SLAM, pois existem retalhos mais fáceis que os outros de serem encontrados por processamento de imagem. Sobre esse vídeo executamos o algoritmo LSLAM com a política de sempre que houver um marcador fiducial ou natural detectado reinicializa o algoritmo de SLAM. Os resultados do rastreamento estão no vídeo *ex3_init_naturais.avi*.

Figura 4.12: Marcadores usados no teste.

Nos resultados vemos que a acurácia dos dois métodos são bastante parecidas quando estão em visão frontal mesmo com movimento. No entanto, o mesmo não acontece em posições mais anguladas ou com a câmera mais afastada. Momentos a posição não permite que o rastreador SURF encontre

a textura da caixa e outros momentos a textura da caixa se confunde com a textura dos adesivos resultando em uma calibração errada. Isso nos permite concluir que por que por mais bem definido sejam os retalhos usados para reinicializar a ambiguidade pode estar presente e isso se torna mais problemático de pontos de vista em diagonal (câmera inclinada ou rotacionada em relação ao plano de suporte), pois as distâncias entre os pontos da imagem diminuem.

Quanto à eficiência na reinicialização, o marcador natural não teve uma piora perceptível em relação ao fiducial. O rastreador SURF teve uma performance de 3 a 4 detecções por segundo. Enquanto o rastreador do ArToolKit consegue ser calculado em todos os quadros fornecidos pela câmera, consumindo menos de 2 ms de processamento. A calibração feita a partir do rastreador SURF nitidamente tem um atraso em relação a imagem atual da câmera, mas a busca ativa do algoritmo de SLAM compensa com sucesso esse atraso desde que não tenha o problema de ambiguidade citado.