Pontifícia Universidade Católica
do Rio de Janeiro

Lucas Pinto Teixeira

Local SLAM

Localização de Câmera e Mapeamento Local de Ambientes Simultâneos

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática da PUC–Rio

Orientador : Prof. Marcelo Gattass
Co–Orientador: Prof. Alberto Barbosa Raposo

Rio de Janeiro
março de 2010

Pontifícia Universidade Católica
do Rio de Janeiro

Lucas Pinto Teixeira

Local SLAM

Localização de Câmera e Mapeamento Local de Ambientes Simultâneos

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC–Rio. Aprovada pela comissão examinadora abaixo assinada.

Prof. Marcelo Gattass
Orientador
Departamento de Informática — PUC–Rio

Prof. Alberto Barbosa Raposo
Co–Orientador
Departamento de Informática — PUC–Rio

Prof. Waldemar Celes Filho
Departamento de Informática — PUC-Rio

Doutor Manuel Eduardo Loaiza Fernandez
Grupo de Tecnologia em Computação Gráfica — PUC-Rio

Prof. Raul Queiroz Feitosa
Departamento de Eng. Elétrica — PUC-Rio

Prof. Jóse Eugenio Leal
Coordenador do Centro Técnico Científico — PUC–Rio

Rio de Janeiro, 25 de março de 2010

Lucas Pinto Teixeira

Graduou-se em Engenharia de Computação na Pontifícia Universidade Católica do Rio de Janeiro, onde deu continuidade a seus estudos no curso de mestrado do departamento de Informática. Durante seu tempo na PUC-Rio, atuou em diversos projetos no laboratório Tecgraf, no departamento de Informática.

Ficha Catalográfica

Teixeira, Lucas P.

Local SLAM / Lucas Pinto Teixeira; orientador: Marcelo Gattass; co–orientador: Alberto Barbosa Raposo. — Rio de Janeiro : PUC–Rio, Departamento de Informática, 2010.

v., 66 f: il. ; 29,7 cm

1. Dissertação (Mestrado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas.

1. Informática – Tese. 2. SLAM. 3. Rastreamento. 4. Visão Computacional. 5. Reconhecimento de Padrões. I. Gattass, Marcelo. II. Raposo, Alberto B.. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

Agradecimentos

Aos meus orientadores, pelo apoio, pelos comentários e pela confiança em mim depositada.

À CAPES, FAPERJ e à PUC-Rio, pelo apoio financeiro, sem o qual este trabalho não seria possível.

À Fabíola pelo apoio, ajuda, compreensão e amor em todos os momentos.

Aos meus pais, avos, irmã e familia pelo carinho, amor e apoio incondicional em tudo.

Aos meus amigos pelo apoio e ajuda ao longo do processo.

Ao Michael Loesler, pelo ajuda na parte de geomensura.

Ao Daniel Pustka, por tem ensinado tudo sobre a competição de rastreamento.

Resumo

Teixeira, Lucas P.; Gattass, Marcelo; Raposo, Alberto B.. **Local SLAM**. Rio de Janeiro, 2010. 66p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Atualmente, sistemas de visão computacional em computadores portáteis estão se tornando uma importante ferramenta de uso pessoal. Sistemas de visão para localização de objetos é uma área de pesquisa muito ativa. Essa dissertação propõe um algoritmo para localizar posições no espaço e objetos em ambientes não instrumentados com o uso de uma câmera web e um computador pessoal. Para isso, são usados dois algoritmos de rastreamento de marcadores para reinicializar frequentemente um algoritmo de Visual Simultaneous Localisation and Mapping. Essa dissertação também apresenta uma implementação e um conjunto de testes para validar o algoritmo proposto.

Palavras–chave

SLAM. Rastreamento. Visão Computacional. Reconhecimento de Padrões.

Abstract

Teixeira, Lucas P.; Gattass, Marcelo; Raposo, Alberto B.. **Local SLAM**. Rio de Janeiro, 2010. 66p. MsC Thesis — Departament of Computer Science, Pontifícia Universidade Católica do Rio de Janeiro.

Nowadays, vision systems in portable computers are becoming an important tool for personal use. Vision systems for object localization are an active area of research. This dissertation proposes an algorithm to locate position and objects in a regular environment with the use of a simple webcam and a personal computer. To that end, we use two algorithms of marker tracking to reboot often a Visual Simultaneous Localisation and Mapping algorithm. This dissertation also presents an implementation and a set of tests that validate the proposed algorithm.

Keywords

SLAM. Tracking. Computer Vision. Pattern Recognition.

Sumário

Lista de figuras

Lista de tabelas

O primeiro dever da inteligência é desconfiar dela mesma.

Albert Einstein, .

1
Introdução

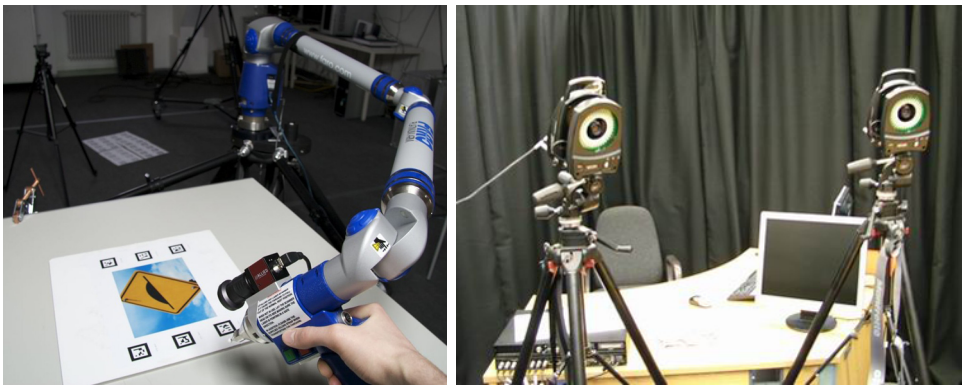
A visão é provavelmente o sentido humano mais utilizado, principalmente na interação com o ambiente. Sendo assim, a ideia de alimentar um computador de informações provenientes de uma imagem vem instigando a comunidade científica há muito tempo. Esse campo de pesquisa é chamado de *visão computacional*(VC). Em paralelo a esse campo de estudo, as técnicas de *realidade virtual*(RV) fornecem meios para substituir as informações sensoriais humanas por informações geradas por computador, normalmente imagens e sons, a fim de transportar a imaginação do usuário para uma realidade totalmente artificial. Nesse trabalho usamos técnicas de visão computacional e realidade virtual para criar uma *realidade aumentada*(RA). O conceito de realidade aumentada mais aceito hoje em dia é o de Azuma [2], no qual RA é: combinação da realidade com elementos virtuais, interativa em tempo real e registrada em três dimensões. Sendo que, o registro é o alinhamento entre o mundo real e o virtual.

A realidade aumentada de forma geral depende fortemente da capacidade de estimar a posição da câmera dentro do ambiente, pois é através dessa estimativa que será possível escolher e apresentar as informações virtuais corretamente. Essa posição pode ser estimada por diversas técnicas. Entre elas, existem equipamentos mecânicos como, por exemplo, o braço articulado da FARO (figura 1.1(a)) e também sistemas óticos baseados em luz infravermelha como, por exemplo, os sistemas de captura de movimento da Vicon (figura 1.1(b)). Esses dois tipos de equipamento são muito precisos, chegando à precisão milimétrica. No entanto, eles possuem uma limitação de amplitude de movimento e preços na faixa das dezenas de milhares de dólares, o que os torna inviáveis para o público em geral.

Para viabilizar o acesso à realidade aumentada, a comunidade científica foca seus esforços nas técnicas puramente óticas, pois as câmeras de computador, comumente chamadas de webcams, estão amplamente difundidas pelos usuários de computador. Da mesma forma, as câmeras de celulares, também bastante disseminadas, podem ser usadas para realidade aumentada. Sendo assim, para fazer uma aplicação de RA que seja acessível às pessoas comuns

hoje em dia, ela precisa ser baseada em visão computacional.

Esta dissertação apresenta um algoritmo para estimação de posição de câmera baseado em visão computacional no intuito de ser viável financeiramente e ao mesmo tempo atender aos requisitos quanto ao tipo de ambiente, amplitude de movimentação de câmera e precisão. Na próxima seção será determinado o conjunto de aplicações em que é esperado que o algoritmo funcione a fim de definir esses requisitos. Em seguida será dada uma ideia geral do algoritmo proposto.



1.1(a): Braço articulado da FARO. Fonte: [25]

1.1(b): Sistema Vicon. Fonte: Laboratorio de Robótica de Briston

Figura 1.1: Sistemas de rastreamento

1.1 Motivação e Objetivo

A motivação desse trabalho é o desafio de localizar um objeto dentro de um ambiente muito grande. Alguns exemplos desse problema são: localizar uma determinada peça dentro de um estoque, onde são armazenadas milhares de diferentes peças; conseguir encontrar um determinado mantimento dentro de um supermercado ou um determinado livro em uma biblioteca[27]. A biblioteca do Trinity College Dublin com 200.000 livros na mesma sala(figura 1.2(a)) e um dos estoques do site de venda Amazon(figura 1.2(b)) do tamanho de 10 campos de futebol são ótimos exemplos de como localizar um objeto específico pode ser muito díficil. Inclusive já existe uma área da Realidade Aumentada estudando como apresentar as informações de forma ao usuário consiga rapidamente recolher(pick) objetos dentro de estoques. Essa área é intitulada Pick-by-Vision [32]

Então, sabendo que todo objeto ocupa um lugar no espaço, ao invés de procurar pelo objeto pode-se procurar pela sua localização no espaço. Sendo assim, o objetivo desse trabalho é *localizar uma posição em um espaço*

1.2(a): Câmara principal da biblioteca do Trinity College Dublin. Fonte: Flickr

1.2(b): Estoque do site de vendas eletrônica Amazon. Fonte: The Guardian, fotógrafo Gareth Phillips

Figura 1.2: Exemplos de ambientes onde é dificil localizar um objeto

tridimensional (3D) previamente conhecido, como por exemplo uma biblioteca ou museu específicos.

Para um usuário localizar uma posição 3D específica usando realidade aumentada são necessárias duas informações. A primeira, conhecer essa posição 3D em coordenadas absolutas, enquanto a segunda é conhecer a posição absoluta do usuário no espaço e a direção para onde ele está olhando. Com esses dois dados é possível traçar uma rota que leve o usuário à posição requerida. Como os objetos são considerados estáticos, então as suas posições em coordenadas absolutas podem ser calculadas em pré-processamento. O mesmo não acontece com a posição e orientação do usuário, pois este está em movimento. Sendo assim é necessário rastrear a sua posição absoluta no espaço 3D e orientação. Como neste trabalho vamos utilizar métodos de visão computacional para resolver o problema e viabilizar a realidade aumentada, a posição do usuário é dada pela posição de uma câmera acoplada a ele.

1.2 Abordagem

Para estimar a posição da câmera acoplada ao usuário em coordenadas absolutas propomos uma técnica de rastreamento híbrida baseada em Visual SLAM[14](Simultaneous Localisation and Mapping) e marcadores, tanto os naturais quanto os fiduciais. Os marcadores fiduciais são posicionados em lugares em que não façam oclusão ou atrapalhem a estética do ambiente. Eles são usados para inicializar o sistema de rastreamento. Como marcadores fiduciais do tipo código de barra 2D fornecem um número identificador para o marcador e a posição relativa da câmera ao marcador, o algoritmo proposto

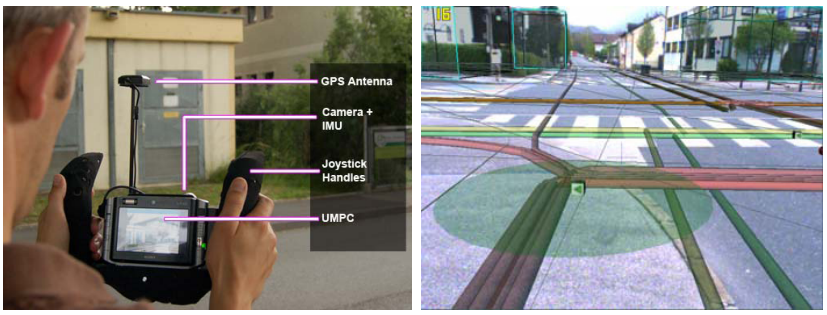
usa esse número para consultar uma base de dados e descobrir a posição absoluta do marcador e consecutivamente a posição da câmera. A detecção de marcadores naturais é mais difícil e cara, pois é necessário comparar a imagem da câmera atual com cada um dos possíveis marcadores presentes no ambiente. Essa comparação com um grande número de marcadores naturais é inviável em tempo interativo, então nós usamos a última posição conhecida da câmera para inferir um pequeno grupo de marcadores possíveis, assim viabilizando a detecção de marcadores naturais em tempo interativo(cerca de 3 detecções por segundo). Quando o marcador é identificado, acontece o mesmo processo dos marcadores fiduciais resultando na posição da câmera em coordenadas absolutas. Com esses marcadores conseguimos descobrir a posição da câmera apenas nos momentos em que a câmera esteja olhando para um marcador. Para não ser necessário ter marcadores em todos os lugares do ambiente, propomos usar um algoritmo de Visual SLAM para estimar a posição da câmera nos momentos em que não haja marcadores disponíveis. Chamamos esta abordagem de Local SLAM, visto que, enquanto os algoritmos de SLAM armazenam todos os pontos mapeados, o algoritmo de SLAM proposto só armazena dados de mapeamento localmente. Os detalhes sobre os algoritmos citados serão mais bem detalhados na seção de trabalhos relacionados.

1.3 Organização

Esse trabalho é organizado da seguinte forma. No capítulo 2 são apresentados os trabalhos relacionados. Em seguida,no capítulo 3, o algoritmo Local SLAM é explicado. No capítulo 4 são explicadas a implementação, as condições de testes e os testes propriamente ditos. Por fim, as conclusões e trabalhos futuros são expostos.

2
Trabalhos Relacionados

Ao se pensar em encontrar uma posição no espaço 3D, uma ideia seria utilizar o sistema de posicionamento global (GPS). No entanto, o GPS comum tem alguns inconvenientes, como por exemplo, não funcionar dentro de edificações e ter um erro de posicionamento de cerca de 10 a 15 metros. Em [30] é apresentado um sistema para localização precisa de dutos embaixo das ruas, onde se utiliza além de um GPS diferencial (DGPS), sensores inerciais e câmeras (figura 2.1). O GPS diferencial apresenta uma maior precisão em relação ao GPS comum, chegando a alcançar um erro entre 1 e 3 metros. Mesmo assim, o DGPS não foi suficiente para localizar a câmera em relação aos dutos, sendo necessária para isso a adição de sensores inerciais e algoritmos de visão computacional. O trabalho[19] utiliza o mesmo tipo de solução para fazer realidade aumentada em ambientes ao ar livre.



2.1(a): Equipamento para visualização

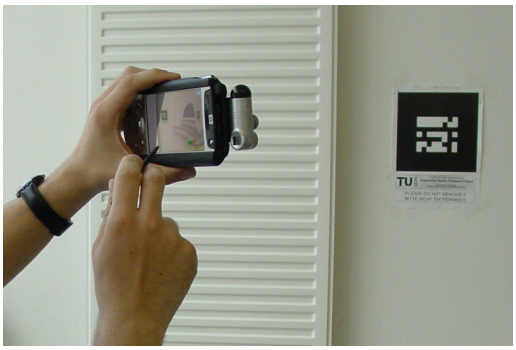
2.1(b): Imagem visualizada pelo usuário.

Figura 2.1: Sistema de localização de dutos sob o chão. Fonte: [30]

Sistemas de rastreamento mecânicos e por luz infravermelha foram descartados nessa dissertação por serem muito caros e de amplitude demasiadamente limitada, como explicado na seção 1. No caso do infravermelho temos o agravante dele não permitir nenhuma fonte de luz infravermelha externa, como uma lâmpada de luz quente ou até mesmo uma janela por onde possa entrar luz natural. O rastreamento infravermelho vem sendo usado para pesquisa de sistemas de localização no âmbito industrial enquanto não há uma solução com a mesma precisão e facilidade de uso [32, 29, 35].

As soluções baseadas em visão computacional sem iluminação infravermelha são as opções realmente viáveis a fim de atingir nossos objetivos. Existem vários sistemas de localização usando visão. Cada um deles tem especificações diferentes quanto ao tipo de preparação do ambiente necessário, a amplitude de rastreamento e a precisão do rastreamento. Na tese de doutorado de Klein[23], ele faz um resumo bastante extenso sobre os todos os principais trabalhos de rastreamento visual. Abaixo será feito um resumo de alguns dos sistemas mais fortemente relacionados a essa dissertação:

Wagner and Schmalstieg[38] apresentam um sistema de localização dentro de um edifício espalhando marcadores fiduciais (figura 2.2). No início da utilização, o usuário seleciona no PDA(Personal digital assistant - Computador de mão) para qual sala ele deseja ir e então aponta o PDA com uma câmera acoplada para um marcador. Esses marcadores ficam distribuídos por todas as paredes em média a cada metro. A imagem capturada pelo PDA é processada no próprio PDA ou enviada para um servidor. O resultado desse processamento é a posição 3D do PDA em relação ao edifício e a direção para a sala requisitada. Então o sistema apresenta uma seta indicando para onde o usuário deve seguir para conseguir chegar ao destino desejado. O custo de processamento desse sistema é sua principal qualidade. O processamento de um único marcador na cena é barato, pois são marcadores fiduciais. Os marcadores usados nesse trabalho são similares a um código de barra 2D, codificando em si um número. Com esse número é fácil consultar uma base de dados para saber qual a posição 3D do marcador e então derivar a posição do PDA. Em contrapartida, a interferência no ambiente e a inexistência de rastreamento do usuário entre dois marcadores inviabilizam o seu uso para localizar objetos pequenos. Além de na maioria dos lugares ser impossível colocar tantos marcadores como é necessário nesse sistema.

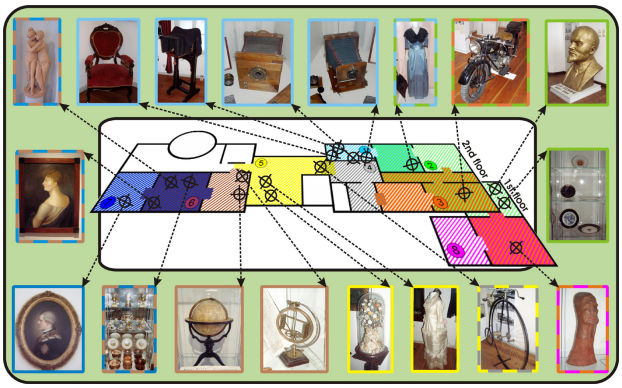


2.2(a): Sistema PDA e marcadores para localização

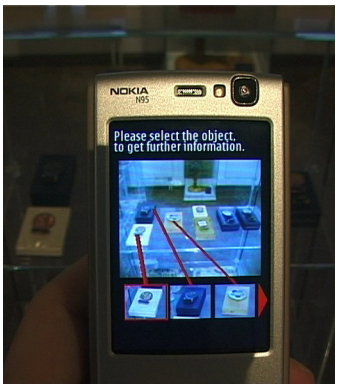
2.2(b): Aplicação de RA sendo apresentada ao usuário

Figura 2.2: Sistema de RA de Wagner e Schmalstieg. Fonte: [38]

Föckler et al[18, 9] apresentam um sistema que é capaz de auxiliar o visitante de um museu(figura 2.3(a)) na forma de um guia virtual. O sistema mostra informações em texto e em vídeo na tela de um celular após o usuário apontar a câmera do celular para o objeto do museu, como mostrado na figura 2.3(b). O sistema funciona por reconhecimento de padrões e RFID. Na fase de preparação do ambiente, fotos de cada objeto são usados para treinar uma rede neural que será utilizada para reconhecer os objetos posteriormente. Os Emissores de RFID são usados para definir regiões do museu e, com isso o software de reconhecimento é capaz de eliminar da lista de candidatos os objetos de salas que não estão próximas a um determinado emissor, melhorando o desempenho e a precisão. A melhor qualidade desse sistema é que ele não é invasivo ao ambiente, uma vez que os marcadores são os próprios objetos do museu. Esse tipo de marcador recebe o nome de marcador natural. Os emissores de RFID(Radio-Frequency IDentification) também são pequenos, do tamanho de duas pilhas AA. No teste feito pelo autor, em um museu de dois andares com 13 salas e 155 objetos, foram necessários apenas 8 emissores. O problema desse sistema é que ele foi projetado para dar informações quando o usuário está em frente ao objeto. O máximo que essa abordagem pode fornecer é uma posição grosseira do usuário no momento em que ele se encontra em frente a um objeto, pois é apenas um sistema de identificação de objetos, não um rastreador.



2.3(a): Mapa dos objetos dentro do museu passiveis de serem reconhecidos pelo sistema



2.3(b): Aplicação de RA sendo apresentada ao usuário

Figura 2.3: Sistema de RA Phone Guide. Fonte: [18]

Davison [14] apresenta um sistema de SLAM (Simultaneous Localisation and Mapping). Sistemas de SLAM são capazes de, a partir de uma posição conhecida, se mover no espaço e manter sua posição conhecida ao mesmo tempo em que geram um mapa do ambiente ao redor. Esses sistemas funcionam de

forma incremental. A partir de uma posição conhecida, ele mapeia novos pontos que estejam no campo de visão. Com esses novos pontos, que abrangem uma área maior que a inicial, a câmera é deslocada. Nesse novo ponto de vista novamente existem pontos conhecidos e pontos desconhecidos que são mapeados e assim recursivamente. A implementação de Davison foi revolucionária, pois foi a primeira a atingir tempo real de processamento em um computador de mesa comum e a utilizar imagens provenientes de uma única câmera como dado de entrada. Esse sistema rastreia os pontos característicos usando a correlação cruzada e um pequeno retalho 2D ao redor dele. O sistema é inicializado a partir de um retângulo preto sobre fundo branco. Esse objeto geométrico fornece os 4 pontos com posições conhecidas, sendo que o centro do sistema de coordenadas é o centro do retângulo. Dando as coordenadas iniciais desses quatro pontos no sistema métrico, o mapeamento também será métrico. Em seguida o sistema procura pontos na imagem que se sobressaiam segundo o detector de Shi-Tomasi[33]. Ele escolhe o ponto mais nítido e tenta adicioná-lo no mapa. Essa adição é bem sucedida caso a câmera se movimente o suficiente para calcular a profundidade do ponto em relação à câmera por triangulação estéreo. O estado do sistema é armazenado em um filtro de Kalman estendido. Esse filtro é o responsável por armazenar a posição da câmera, a velocidade da câmera, as posições das características e a incerteza associada a cada variável do estado. O tempo real desse sistema é atingido graças ao uso de uma técnica chamada busca ativa(Active Search). Normalmente, a busca é feita de forma passiva, ou seja, o pareamento é feito da seguinte maneira: os pontos característicos no quadro atual são procurados no quadro anterior. Essa busca é feita delimitando, no quadro anterior, uma área de busca de tamanho fixo ao redor da posição do ponto no quadro atual. Caso no quadro anterior tenha, dentro da área de busca, um ponto com características semelhantes, então eles são considerados correspondentes. Já na técnica de busca ativa, o ponto do quadro anterior, que no nosso caso pertence ao mapa e está armazenado no filtro de Kalman, é buscado no quadro atual. Além disso, a busca é inteligente, pois a área de busca é definida pelo preditor do filtro de Kalman e pela incerteza associada a cada ponto característico. Quanto maior a incerteza, maior a área de busca. Como o movimento dos pontos característicos que são candidatos a entrar no mapa não pode ser descrito por incertezas gaussianas, pois sua profundidade em relação a câmera não é conhecida, Davison usa um filtro de partícula para predizer a posição desses pontos no próximo quadro. Apenas quando esse filtro converge para um comportamento parecido com o da função densidade de probabilidade(pdf) de uma gaussiana da profundidade, em outras palavras, quando o filtro de partícula apresenta um comportamento parecido

com o de uma gaussiana, o ponto candidato passa a pertencer ao mapa. Esse é o ponto no qual a profundidade é suficientemente conhecida para ser inserida no mapa. Caso essa convergência não aconteça em um número determinado de quadros, o ponto característico é descartado da lista de candidatos.

As qualidades desse sistema são muitas. Em princípio, qualquer posição 3D, se posicionada em relação ao retângulo preto que inicializa o sistema, pode ser alcançada. Isso significa que nenhum tipo de modificação seria necessária no ambiente além de fixar um único marcador em uma parede. Mas esse sistema também tem problemas. O tamanho do estado do filtro de Kalman cresce quadraticamente em relação ao número de pontos característicos no mapa, o que, segundo o próprio autor, deixa de ser computável em tempo real na faixa de 400 pontos. Eade[1] propôs uma estratégia usando um grafo espacial que indica para cada ponto quais outros pontos precisam ter suas covariâncias atualizadas. Então esse problema pode ser significativamente reduzido. Um grave problema do sistema de Davison é o erro acumulado no mapa. Como os novos pontos são calculados a partir de pontos que foram anteriormente estimados, por menor que seja o erro em cada medida, para pontos distantes dos quatro pontos que inicializaram existe um erro acumulado potencialmente grande. Infelizmente, nesse sistema, é quase impossível fazer uma volta fechada, em torno do usuário segurando a câmera, em que ao voltar à posição de inicialização do sistema ele consiga identificar os pontos iniciais como sendo, no mapa, aqueles que foram iniciados na primeira volta.

Esse problema é conhecido como 'loop-closing' (fechamento de volta) e é bem discutido na comunidade científica, como pode ser visto nos trabalhos [13, 39, 15]. Todas essas técnicas de volta fechada têm como princípio identificar o momento em que a volta se fechou para corrigir a posição de todos os pontos do mapa e, com isso os erros de acumulação seriam eliminados. No entanto essa ideia tem um problema em sua essência. Enquanto a volta não se fechar os pontos continuam errados, além de a correção dos pontos não acontecer em tempo real. No nosso caso, de localização de posições 3D no espaço, podemos ter uma fase de preparação, apesar de não ser a ideia inicial do SLAM. No entanto, para podermos ter os benefícios da correção do mapa através de uma volta fechada é necessário fazer, na fase de preparação do ambiente, o mapa com quantas voltas fechadas sejam necessárias e, com isso, não usar mais apenas os quatro pontos inicialmente conhecidos, mas todos os que já foram calculados na fase de preparação. Essa ideia funcionaria se considerarmos que o ambiente não sofre nenhuma mudança significativa de iluminação ao longo do tempo. Como isso não é verdade, então a ideia de deixar um mapa completo armazenado não é viável, pois a iluminação vai mudar ao longo do tempo e

o pareamento, pixel a pixel (block-matching), dos retalhos armazenados no estado do filtro de Kalman com os capturados em tempo real com a câmera, não funcionará, já que esse método não é invariante à mudança de iluminação.

O último problema dessa abordagem para a localização de uma posição 3D no espaço é a inicialização. A implementação de Davison considera que o movimento sempre será inicializado com a câmera apontando para o retângulo preto que é o centro do sistema. Esse tipo de requisito torna o algoritmo de rastreamento muito limitado, dado que para qualquer problema em que o rastreamento se perca, é necessário retornar à posição do retângulo preto.

Castle et al [11, 12] propõem uma modificação no sistema de Davison. Eles sugerem adicionar uma base de dados com imagens (texturas) que estão presentes na cena, mas sem necessidade de conhecer a posição das imagens no espaço. Com isso, em paralelo ao processo do SLAM de Davison, é mantido um processo que procura por texturas conhecidas. Quando a textura é localizada, os vértices da caixa envolvente são adicionados ao mapa. Com essa abordagem ele é capaz de reinicializar constantemente os pontos da textura, ajudando na estabilidade do rastreamento. Essa modificação mantém os problemas do sistema de Davison, mas melhora a estabilidade e a tolerância às falhas. Além disso, o sistema pode ser reinicializado por uma das texturas mapeadas. Quanto ao erro acumulado, como as texturas são mapeadas pelo processo de SLAM de Davison, elas também possuem erro de posicionamento. Outro problema apresentado por essa abordagem é que o processo de localização de texturas utilizado, o SIFT [26], é bastante caro computacionalmente. Segundo testes realizados por Castle et al., um banco de dados de apenas 16 objetos já possui cerca de 30.000 pontos chave para serem comparados com os pontos recuperados da imagem da câmera. Assim, podemos inferir que, mesmo com uma organização em árvore, esse banco de dados não pode crescer muito, pois corre o risco de tornar a detecção de texturas incomputável em tempo aceitável, além de agravar o problema da ambiguidade gerado pelo excesso de texturas parecidas.

Klein e Murray [24] apresentaram um sistema chamado de 'Parallel Tracking and Mapping for Small AR Workspaces' (PTAM). Eles argumentam que uma pessoa segurando uma câmera é muito diferente de um robô se movendo. Sendo assim, um filtro de Kalman estendido não é suficiente para modelar o problema. Esse algoritmo separou em dois processos independentes a localização (rastreamento) e o mapeamento. Ambos os processos são modelados com técnicas bastante conhecidas em outras áreas da visão computacional. O mapeamento passa a ser tratado como um problema de Structure-from-Motion (SfM) a partir de quadros chave. O algoritmo escolhe alguns quadros

suficientemente distantes e então alimenta o algoritmo de SfM. Esse algoritmo faz uma otimização local entre os quadros mais próximos, além da otimização global, mais cara, do mapa inteiro. Essa otimização global tem os mesmos efeitos das otimizações utilizadas no fechamento de volta (loop-closing). Com isso o mapa tem uma qualidade bastante superior a dos algoritmos baseados em filtro de Kalman, principalmente pela falta de otimização, como demonstrado nesse trabalho de Klein e Murray. Em contrapartida essa otimização é muito cara. Ela que é a maior responsável pelo nome 'Small AR Workspaces', pois, segundo recomendações do autor, o sistema só funciona bem com um mapa de até cerca de 6.000 pontos e cerca de 150 quadros chave. Com esses valores a otimização já leva cerca de 7 segundos para ser executada. O valor de 6.000 pontos pode parecer muito comparado aos 400 do algoritmo de Davison, mas não é. Enquanto em uma volta completa em torno do usuário, usando o algoritmo de Davison, o mapa tem cerca de 100 pontos, no algoritmo de Klein ele precisa de 11.000 pontos, o que o próprio autor afirma estar fora do escopo de 'pequena área de trabalho'.

O processo de rastreamento é uma versão modificada do KLT [34]. No KLT o pareamento é feito por busca passiva, como explicado anteriormente. A modificação de Klein é que não são usados os pontos detectados no quadro anterior. Ao invés disso, ele usa a posição da câmera estimada no quadro anterior e projeta cerca de 50 pontos escolhidos aleatoriamente do mapa no plano da imagem. Depois de conseguir identificar no plano 2D qual foi o movimento de alguns desses 50 pontos, ele usa a posição tridimensional dos pontos para estimar a nova posição da câmera. Como Klein cria um mapa denso e utiliza, no rastreamento, cerca de 1.000 pontos reprojetados para realizar a otimização da posição da câmera, ele consegue um rastreamento muito estável. Esse algoritmo pode funcionar como localizador de pontos 3D, mas apenas em regiões muito pequenas, como, por exemplo, a área de uma mesa de trabalho, o que não atende os casos descritos na seção de motivação. Outro problema apresentado é a falta de invariância a mudanças de iluminação, da mesma forma que o algoritmo de Davison. Apesar de ter cerca de mil pontos sendo rastreados, uma mudança na iluminação do ambiente faz com que praticamente todos os pontos sejam perdidos.

Castle et al [10] fez uma modificação em cima do algoritmo PTAM que possibilitou criar múltiplos mapas e navegar entre eles. Com isso ele fez um algoritmo bastante preciso em pequenas ilhas de interesse. Além disso, ele é capaz de indicar onde estão as ilhas ao redor. Ele supõe que o usuário é suficientemente capaz de caminhar até a próxima ilha de interesse. De fato esse sistema atende ao nosso problema de localização de posições 3D, mas

apresenta o mesmo problema de iluminação do PTAM.

2.1 Posicionamento desse trabalho

Neste trabalho apresentamos uma versão modificada do algoritmo de Davison [14]. A primeira modificação é a substituição da inicialização que utiliza um retângulo preto por uma inicialização usando um marcador fiducial aos moldes do algoritmo de Wagner [38], ou seja, para cada marcador fiducial temos sua posição no espaço. Com isso é possível inicializar o sistema de posições diferentes do centro do sistema de coordenadas.

A segunda modificação é a introdução de um detector de texturas naturais em paralelo, análogo à Castle et al [12]. Mas, diferentemente de Castle, vamos mapear a posição exata da textura na fase de preparação do ambiente e, com isso eliminar o erro inerente ao mapa do SLAM de Davison sem otimização. Além disso, como as posições das texturas naturais são conhecidas, podemos prever a posição delas no ambiente e, com isso podemos eliminar da pesquisa de reconhecimento de padrões as texturas que estão fora da visibilidade da câmera.

A terceira modificação é a constante reinicialização. Sabendo que o erro de um algoritmo de SLAM é diretamente proporcional à distância do marcador de inicialização, então vamos reinicializar o sistema toda vez que conseguirmos encontrar um marcador natural ou fiducial com sucesso. Com isso, reduzimos o número de pontos característicos no filtro de Kalman, o que garante bastante amplitude de operação.

Como os marcadores fiduciais são invariantes à mudança de iluminação e existem detectores de textura natural invariantes à mudança de iluminação global, podemos fazer um mapa que só terá que ser alterado caso os marcadores sejam movidos de lugar. Outra qualidade dessa abordagem é que como inicializamos com marcadores fiduciais e, além deles marcadores naturais que estão no campo visível, então temos uma quantidade reduzida de texturas para procurar. Essa ideia é análoga à de Föckler et al[18] no uso do RFID para reduzir o campo de busca e consequentemente o tempo de processamento e as ambiguidades. Além disso, como temos, além dos marcadores fiduciais, os marcadores naturais, podemos usar menos marcadores fiduciais, o que é importante, pois esses muitas vezes provocam a oclusão de partes importantes do ambiente, além de interferir no seu aspecto.

3
Local SLAM

Os algoritmos de SLAM (Simultaneous Localisation and Mapping) surgiram da necessidade de se enviar robôs por ambientes desconhecidos. Os primeiros algoritmos usavam dados de odometria e sensores de distância para ir mapeando o que tinha a frente do robô(figura 3.1), e depois para saber onde ele estava naquele mapa. Há poucos anos foi possível através de computadores e algoritmos melhores fazer tanto o mapeamento quanto a localização usando apenas uma câmera. Essa família de algoritmos foi chamada de Visual SLAM. Em Visual SLAM, a câmera acoplada ao robô tem a sua posição rastreada ao longo do tempo. Logo os pesquisadores perceberam que esse conceito era idêntico ao rastreamento de câmera para realidade aumentada. No entanto, um problema sério surgiu e até hoje está mal resolvido, o erro acumulado, como foi explicado na seção de trabalhos relacionados no trabalho de Davison[14]. O SLAM é usado pelos robôs apenas para evitar obstáculos desconhecidos e ter uma localização razoável do robô no espaço. Já as aplicações de realidade aumentada que propomos acontecem em ambiente conhecido e é essa informação que usamos para propor o conceito de Local SLAM. Como explicado no capítulo de trabalhos relacionados, não podemos esperar que um mapa feito previamente vá funcionar em um segundo momento, por isso propomos usar um algoritmo de Visual SLAM apenas localmente, entre dois marcadores previamente mapeados. Assim que localizado um novo marcador a posição da câmera é reinicializada, o mapa do SLAM é apagado e um novo começa a ser gerado.

Nessa seção é detalhado o algoritmo proposto. Primeiramente, vamos explicar qual tipo de marcador fiducial é usado. Em seguida como rastreamos marcadores naturais. Então explicamos como fazer o mapeamento do ambiente. Em seguida explicamos as modificações feitas num algoritmo de Visual SLAM para suportar o Local SLAM. Em seguida explicamos quando decidimos reinicializar o algoritmo de SLAM. Por fim é descrito o processamento geral do algoritmo.

Figura 3.1: Uma montagem clássica de robôs para usar técnicas de SLAM. Fonte: [28]

3.1 Marcadores fiduciais

Hoje em dia, a forma mais popular e simples de identificar objetos e fazer RA é colocar marcadores fiduciais ao redor do objeto. Os arcabouços mais utilizados para marcar, reconhecer e rastrear objetos são o ARToolkit [22], ARTag [17] e muitos outros derivados deles, como por exemplo o ARToolKitPlus[37] e o Studierstube Tracker[31]. Nessas bibliotecas foram adicionadas extensões e otimizações à detecção e ao rastreamento. Além disso, essas bibliotecas são fáceis de usar e são extremamente eficientes em computadores de mesa. Elas conseguem de forma geral fazer o processamento de uma imagem 640x480 em menos de 5ms dependendo do número de marcadores na imagem.

No nosso caso preferimos utilizar a biblioteca ARToolKitPlus(figura 3.2) por ela usar uma matriz 2D com um código do tipo BCH que é capaz de se autovalidar, o que seria análogo a um dígito de segurança. Usando uma matriz 6x6, a biblioteca é capaz de codificar cerca de 4.000 números, mais do que suficiente para nosso algoritmo.

Essa biblioteca fornece a posição da câmera em relação ao marcador. Consideramos posição de câmera como sendo o par (posição do ponto focal da câmera, direção para onde ela está olhando). A fim de que a posição da câmera seja dada em coordenadas métricas, temos uma base de dados com o exato tamanho de cada marcador, assim após identificado o código do marcador, ele é consultado numa base de dados para saber qual o tamanho em metros daquele marcador. Além disso, a biblioteca também fornece as coordenadas no plano da imagem dos quatro vértices do quadrado preto que envolve o marcador.

Como essa biblioteca é vastamente documentada e utilizada, não vamos entrar em maiores detalhes sobre seu funcionamento.

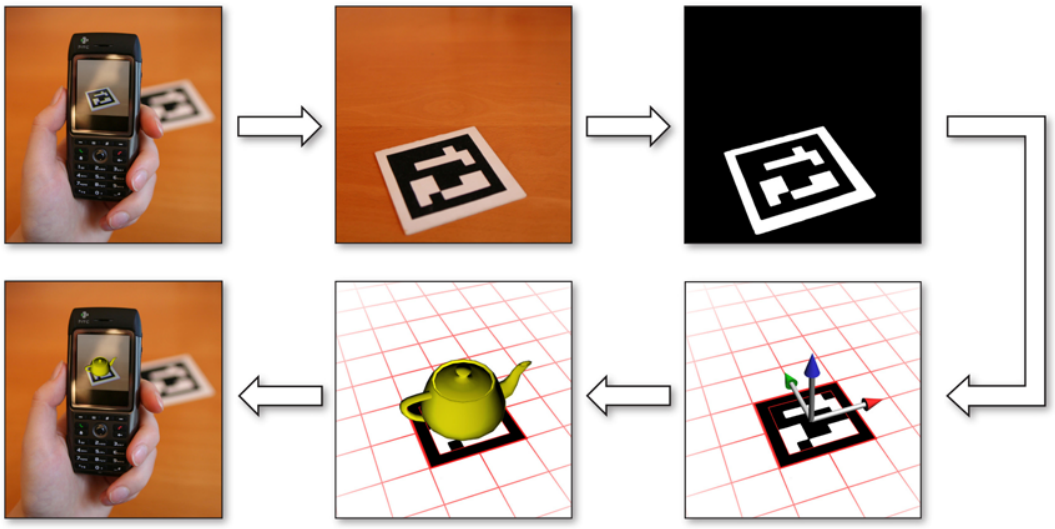


Figura 3.2: Fluxo de processamento de uma aplicação de RA usando marcadores fiduciais. Fonte: [37]

3.2 Marcadores Naturais

Para marcadores naturais não é utilizada uma biblioteca de rastreamento pronta. Neste trabalho é utilizado um reconhecedor de texturas juntamente com um algoritmo de calibração de câmera baseado em pares de pontos que conhecemos sua posição tanto no plano da imagem de textura quanto na imagem capturada pela câmera. Esses pares de pontos são fornecidos pelo algoritmo reconhecedor. Nas próximas seções serão brevemente descritos os processos de reconhecimento de texturas e de calibração de câmera utilizados nesse trabalho.

3.2.1 Reconhecedor de texturas

Atualmente, os dois algoritmos de reconhecimento de textura naturais mais usados são o SIFT [26] e o SURF [6]. No reconhecimento, o SIFT apresenta uma precisão superior à do SURF na maioria dos artigos publicados [3, 21]. Por outro lado, o SURF se mostrou cerca de quatro vezes mais rápido, segundo seu próprio autor e também segundo Bauer et al [3], onde é mostrado um gráfico do que podemos chamar de custo benefício, fig 3.3. O gráfico mostra o número de pontos detectados e pareados corretamente dividido pelo tempo gasto. Nesse gráfico pode-se perceber que o SURF é significativamente superior ao SIFT em todos os casos de teste apresentados. Os casos de teste são representados por variações de atributos da imagem (rotação, ruído branco,

escala, iluminação e perspectiva). Sendo assim, neste trabalho optamos por usar o SURF.

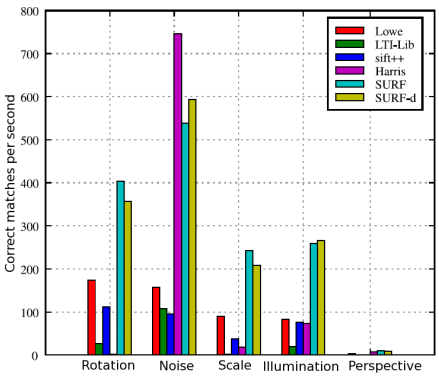


Figura 3.3: Número de pontos reconhecidos corretamente por segundo do SURF em diversas situações. Fonte: [3]

O algoritmo SURF[6, 4], acrônimo para *Speeded Up Robust Features*, é composto por um detector de pontos característicos e um descritor de pontos, que serão usados posteriormente para o reconhecimento das texturas. Os pontos característicos, comumente chamados na literatura de *features*, são pontos na imagem que podem ser localizados automaticamente de diferentes pontos de vista. Os algoritmos que fazem a detecção automática são os detectores de pontos característicos. Para cada ponto característico detectado o SURF usa o descritor de pontos para calcular um identificador do ponto baseado na textura ao redor dele. Em seguida vamos dar um breve resumo do detector, do descritor propostos no algoritmo SURF e do reconhecedor propriamente dito.

Detector de pontos característicos

O detector de pontos característicos foi chamado de *Fast-Hessian Detector*. Como o próprio nome diz, é um detector baseado na matriz Hessiana. A formulação dele é dada a seguir e, para facilitar a compreensão desse detector será seguida a nomenclatura e notação do artigo original. Dado um ponto $\mathbf{x} = (x, y)^T$ na imagem I, a matriz Hessiana $\mathcal{H}(\mathbf{x}, \sigma)$ em x na escala σ é definido como

$$\mathcal{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix}, \tag{3-1}$$

onde $L_{xx}(\mathbf{x}, \sigma)$ é a derivada de segunda ordem da convolução gaussiana de parâmetro σ ($\frac{\partial^2}{\partial x^2} g(\sigma)$) no ponto $\mathbf{x}$, por similaridade temos $L_{yy}(\mathbf{x}, \sigma)$ e $L_{xy}(\mathbf{x}, \sigma)$

O nome do detector é hessiana rápida, na tradução literal, porque o algoritmo faz simplificações no cálculo da matriz hessiana. A convolução gaussiana de 9 x 9 na verdade é reduzida a filtros caixa, como desmonstrado na figura 3.4. Esses filtros caixa podem ser calculados rapidamente com o uso de imagem integral como foi definido por Viola e Jones[36]. Com isso também foi resolvido o problema das várias escalas, já que essa aproximação da hessiana pode ser calculada em diferentes escalas apenas aumentando ou diminuindo o tamanho do filtro caixa sem necessidade de efetivamente fazer as reduções da imagem original. Mais detalhes e imagens explicativas são apresentados no artigo original do SURF[6].

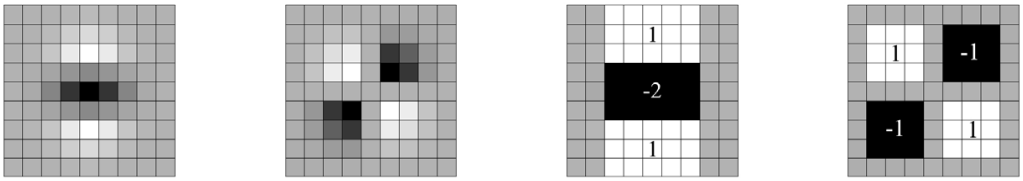


Figura 3.4: A esquerda derivada de segunda ordem da convolução gaussiana(L_{yy} e L_{xy})) e a direita as correspondentes aproximações usando filtros caixa(D_{yy} e D_{xy})). Fonte: [6]

Após calcular a hessiana para toda a imagem em diferentes escalas, em geral a escala original e mais 3 reduções, é calculado o determinante da hessiana rápida em cada ponto pela fórmula 3-2

$$det(\mathcal{H}_{rápida})(\mathbf{x},\sigma) = D_{xx}(\mathbf{x},\sigma)D_{yy}(\mathbf{x},\sigma) - (0.9D_{xy}(\mathbf{x},\sigma))^2, \qquad (3\text{-}2)$$

onde $D_{xx}(\mathbf{x},\sigma)$ corresponde ao filtro caixa correspondente a $L_{xx}(\mathbf{x},\sigma)$, por similaridade temos $D_{yy}(\mathbf{x},\sigma)$ e $D_{xy}(\mathbf{x},\sigma)$

Finalmente, é feita uma supressão de não máximos desses determinantes na vizinhança 3 x 3 x 3 para encontrar os melhores pontos característicos.

Descritor de pontos

O descritor de pontos do SURF é composto principalmente por: $\mathbf{x}$,σ,um vetor de orientação em $\mathbb{R}^2$, e uma matriz de 64 posições na forma 4 x 4 x 4 com informações sobre os vizinhos do ponto. Os dois primeiros atributos já podem ser calculados na fase de detecção. A orientação é o vetor calculado a partir da resposta da Haar wavelet na direção x e na direção y. Da mesma forma que a gaussiana, as Haar wavelets podem ser computadas rapidamente por filtros

caixa. O cálculo desse vetor torna o descritor invariante à rotação, ou seja, é possível rotacionar a imagem que o ponto descrito continua sendo identificado. Por fim, a matriz com as informações sobre os vizinhos é calculada. Essa matriz é calculada a partir do retalho de tamanho 20 x 20 pixels centrado no ponto **x** na imagem I reduzida pelo fator σ e orientado pelo vetor de orientação. Esse retalho é dividido em 4 x 4 sub-retalhos cada um com 5x5 pixels como mostra a figura 3.5. Cada um dos 5 x 5 pixels tem computada sua resposta da Haar wavelet na direção x' e y', que são as direções x e y corrigidas pelo vetor orientação e vamos chamá-las de dx e dy. O filtro caixa para a resposta da Haar wavelet é do tamanho 2 x 2. Para cada sub-retalho é feito o somatório em cada um dos seus 25 pixels de dx e dy e de seus valores normalizados $|dx|$ e $|dy|$. Com isso cada um dos sub-retalhos passa a ter como atributos $\sum dx$,$\sum |dx|$,$\sum dy$,$\sum |dy|$ que são a terceira dimensão de tamanho 4 da matriz de 64 posições que descreve a vizinhança do ponto. A invariância a mudança uniforme de iluminação é obtida graças ao uso do Haar wavelet que mede a diferença de iluminação entre os vizinhos ao invés do valor propriamente. A invariância a mudança de contraste pode ser obtida se normalizarmos a matriz de 64 posições para ter soma dos elementos unitária. Para saber se dois pontos são parecidos basta calcular a distância entre as duas matrizes. Essa distância é dada considerando cada matriz como um ponto em $\mathbb{R}^{64}$ e calculando a distância euclidiana no espaço do descritor entre os pontos, ou seja, $dist = \sqrt{\sum_{i=0}^{64}(A_i - B_i)^2}$

Reconhecimento de texturas

O reconhecimento de texturas do SURF, diferente do reconhecimento de marcadores fiduciais, precisa ser feito por comparação das possíveis texturas. Portanto, é necessário em pré-processamento fazer uma base de dados das texturas que serão procuradas. Cada textura é processada pelo detector de pontos característicos. Em seguida, é calculado o descritor de cada um desses pontos. O ponto característico com um descritor associado recebe o nome de pontos-chave, ou na literatura como *keypoints*. O processo de identificação da textura na imagem capturada pela câmera durante o processamento é igual. Os pontos característicos são detectados nela e em seguida descritos. Para procurar se existe alguma textura pertencente ao nosso banco de dados, precisamos comparar cada ponto chave da imagem da câmera com todos os pontos chave de todas as texturas, uma por uma. Em cada uma computamos os pares de pontos da imagem da câmera e da textura procurada que tenham a distância

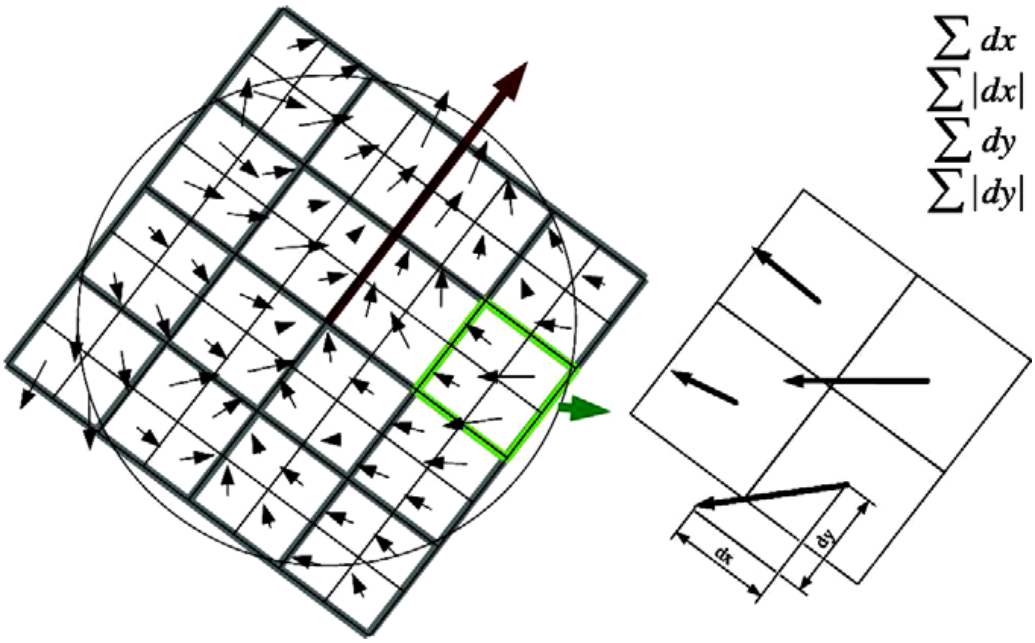


Figura 3.5: Para construir o descritor, o retalho orientado é subdividido numa grade de 4x4 sub-retalhos. Dentro de cada um as respostas ao wavelet foram calculadas de 5 x 5 amostras(por questões ilustrativas, só tem 2x2). Fonte: [6]

euclidiana no espaço do descritor razoavelmente pequena. O número de pares de pontos com cada textura i da base de dados é N_i. Normalmente para saber qual textura está aparecendo na imagem da câmera, considerando que só possa ter uma de cada vez, é a textura que tem N_i maior, mas isso pode nos levar para falsas detecções, pois o pareamento dos pontos foi considerando um valor maior que zero para a distância entre os descritores, pois é impossível esperar distância zero com o descritor sendo um vetor de números reais, levando a um inescapável erro numérico por menor que seja, além de que a imagem da câmera é a representação discreta da cena vista pela câmera, que é um espaço contínuo. Sendo assim utilizamos o critério S_r que balanceia o N_i com a distância entre os pontos-chave, esse método foi proposto em [5]. O critério S_r é definido como

$$S_r = \underset{i}{\operatorname{argmax}} \left(\frac{N_i}{\sqrt{\sum_{j=i}^{N_i} dist_{ij}^2}} \right), \tag{3-3}$$

onde i representa a i-ésima textura do banco de dados e j o j-ésimo keypoint da textura i. Para aceitar como válido o reconhecimento, S_r precisa ser maior que 0.8 vezes do que o segundo maior S_r e $N_i maior que 8$.

3.2.2 Calibração de câmera

Calibração de câmera é o procedimento de descobrir os parâmetros intrínsecos e extrínsecos de uma câmera. Os parâmetros intrínsecos são os inerentes ao conjunto sensor de captura (CCD) e lente. Eles só variam em três situações: caso a câmera mude, a lente mude, ou a distância focal mude. Os parâmetros são o centro focal no plano da imagem, a distância focal, o tamanho da imagem e os coeficientes de distorção radial. Os parâmetros extrínsecos são a rotação e a translação que definem a posição da câmera em relação ao sistema de coordenadas de referência.

Os parâmetros intrínsecos não foram abordados nesse trabalho. Para encontrá-los, usamos um método manual auxiliado por um kit de ferramenta(toolkit) de calibração de parâmetros intrínsecos no MATLAB, o *Camera Calibration Toolbox for Matlab*[7] feito por Bouguet. O algoritmo utilizado por ele não influenciou os detalhes de projeto do nosso algoritmo. O mais importante é que ele capaz de calcular e verificar a qualidade dos parâmetros intrínsecos com bastante precisão a partir dos dados de entrada do operador, sendo assim supomos que o operador ajustou os dados de entrada até que a precisão dos parâmetros intrínsecos tenham ficado num limiar aceitável para o tipo de aplicação. Vale lembrar que os parâmetros intrínsecos calculados são os mesmos para todos os algoritmos de calibração desse trabalho, ou seja, o algoritmo explicado nesta seção, o algoritmo de calibração interno do ArToolKitPlus e o algoritmo de SLAM.

Para calcular os parâmetros extrínsecos, é utilizado o método descrito no artigo de Zhang[40] ligeiramente modificado. Esse método é constituído de dois passos, o cálculo da homografia e em seguida o cálculo da rotação e da translação da câmera levando em conta os parâmetros intrínsecos e a homografia calculada. Homografia é uma matriz 3x3 que representa a transformação de perspectiva entre dois planos. A homografia é calcula pelo algoritmo *Direct Linear Transformation (DLT)* [20]. Para usar o DLT é necessário que todos os pares de ponto sejam corretos, mas o SURF não garante absoluta certeza nos pares retornados, então usamos a técnica RANSAC (Random Sample Consensus)[20] minimizando o erro de reprojeção para tentar eliminar os pares incorretos. O uso do DLT e do RANSAC são as modificações feitas no método original de Zhang que usa outro tipo de cálculo de homografia. Esse algoritmo consegue a homografia H a partir de um conjunto de pares de pontos($\mathbf{m} \leftrightarrow \mathbf{m}'$) suficientemente grande. Um par de pontos é constituído por $\mathbf{m} = (u\ v\ 1)^T$,o ponto na imagem, e $\mathbf{m}' = (x\ y\ 1)^T$,o ponto no modelo. O modelo é a textura armazenada previamente, seja fruto do escaneamento ou

mesmo uma foto, já a imagem é a imagem da câmera no momento que se quer saber sua posição 3D. A relação entre eles é descrita como $\mathbf{m} = H\mathbf{m}'$. A técnica RANSAC propõe que use apenas um subgrupo escolhido aleatoriamente dos pares de ponto para o cálculo da homografia e com essa estimativa projete todos os $\mathbf{m}'$ e calcule a distância para $\mathbf{m}$,ou seja,$\xi_i = ||H\mathbf{m_i'} - \mathbf{m_i}||$, o operador $||.||$ é a norma. Caso o $\frac{1}{n}\sum_i \xi_i$, onde n e o número de pares, seja abaixo de um limiar a homografia é considera certa, caso contrário o algoritmo tenta mais algumas vezes o mesmo processo, se todas resultarem acima do limiar ele descarta o reconhecimento feito pelo SURF e o processamento não continua. Com a homografia calculada e os parâmetros intrínsecos conhecidos podemos passar para o cálculo dos parâmetros extrínsecos. Em Zhang[40] é feita a demonstração matemática do método. O apêndice B apresenta as fórmulas necessárias para encontrar os parâmetros extrínsecos já diretamente relacionados aos dados de entrada, ou seja, a homografia e os parâmetros intrínsecos. Para que os parâmetros extrínsecos sejam dados em coordenadas métricas, basta que os pontos no modelo($\mathbf{m}'$) também estejam em coordenadas métricas e os pontos da imagem($\mathbf{m}$) em número de pixels.

3.3 Mapeamento do ambiente

O mapeamento do ambiente é importante para o nosso algoritmo, pois é ele que permite criarmos um sistema de coordenada comum a todos os marcadores. Como foi explicado nas seções acima os rastreadores de marcadores só informam a posição da câmera relativa a um marcador específico. Então, para descobrirmos a posição global da câmera é necessário conhecer também a posição global do marcador e concatenar as duas informações. A outra função do mapeamento é selecionar os pontos que serão acompanhados pelo SLAM quando não estiver sendo utilizado o rastreador de marcador. Para isso precisamos selecionar pontos que sejam facilmente identificados pelo detector de característica interno do algoritmo de SLAM utilizado, no nosso caso o detector é o *Shi e Tomasi*[33]. Além de selecionar os pontos é necessário conhecer a posição deles no sistema de coordenada global, para que o SLAM também calcule sua posição em coordenadas globais. Por fim, o mapeamento precisa calcular o fator métrico que faz com que os rastreadores calculem a posição relativa da câmera ao marcador na escala métrica.

Nesta seção, primeiramente descrevemos o equipamento utilizado para realizar o mapeamento, em seguida explicamos como fazer as medições nos marcadores fiduciais e naturais e como concatenar a posição relativa ao marcador e com a posição global do marcador.

3.3.1 Equipamento de medição

O mapeamento dos marcadores foi feito usando um equipamento de topografia chamado *estação total*(figura 3.6). Esse equipamento é capaz de medir coordenadas 3D com bastante precisão. Ele utiliza três sensores para fazer as medições: um sensor de rotação que mede as rotações do equipamento sobre o eixo colinear com a força da gravidade(pan), outro que mede a rotação vertical (tilt) e, por fim, um sensor laser para medir a distância do ponto que se deseja até o equipamento. Os sensores de rotação tem precisão de 6" de grau e o de distâncias de 3 a 5 mm para pequenas distâncias. Com os dados provenientes desses três sensores e alguns pontos de referência conhecidos o equipamento calcula internamente as coordenadas 3D. Para mapas maiores que necessitam de medidas de várias posições da estação pode ser usado um algoritmo de mínimos quadrados para diminuir o erro de medição e o erro acumulado proveniente da movimentação.

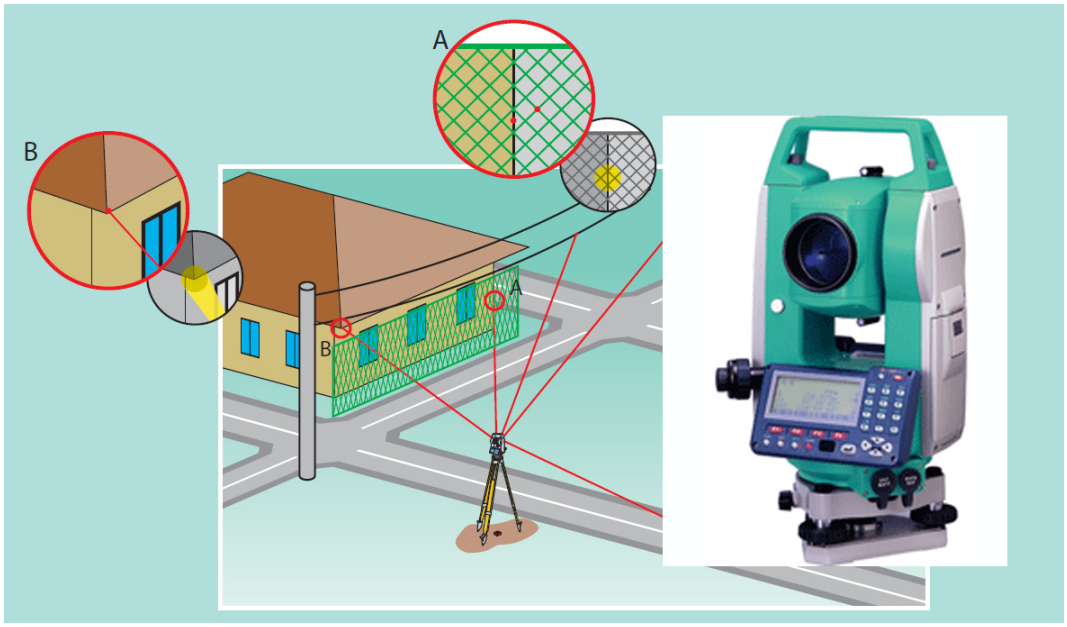


Figura 3.6: Estação Total. Fonte: Sokkia Corp.

3.3.2 Mapeamento dos marcadores fiduciais

Nos marcadores fiduciais medimos com a estação as coordenadas dos quatro vértices($\mathbf{p_1}$ $\mathbf{p_2}$ $\mathbf{p_3}$ $\mathbf{p_4}$) em $\mathbb{R}^3$, do quadrado preto que envolve o marcador (ver figura 3.7). Assim, pela média desses pontos temos o ponto

médio dos vértices que é o centro($\mathbf{p_c}$) do sistema de coordenada local do marcador. A normal pode ser calculada pelo produto vetorial. Assim temos

$$\mathbf{p_c} = \frac{1}{4}(\mathbf{p_1} + \mathbf{p_2} + \mathbf{p_3} + \mathbf{p_4}) \tag{3-4}$$

$$\vec{n} = (\mathbf{p_3} - \mathbf{p_4}) \times (\mathbf{p_1} - \mathbf{p_4}) \tag{3-5}$$

Para os cálculos necessários para transformar os pontos calculados relativamente ao marcador para o sistema de coordenada global vamos escrever essa transformação na forma de uma matriz 4x4 de transformação de corpo rígido, vamos chamar de Q. Vamos chamar de $\mathbf{p^M} = (x^M\ y^M\ z^M\ 1)$ o ponto no sistema do marcador e $\mathbf{p^G} = (x^G\ y^G\ z^G\ 1)$ no sistema de coordenada global. A matriz Q satisfaz a relação

$$\mathbf{p^M} = Q_{MG}\mathbf{p^G}, \tag{3-6}$$

onde Q_{MG}é a transformação de corpo rígido que leva do sistema de coordenada global para o do marcador. Como sabemos que uma transformação de corpo rígido é inversível, temos que

$$\mathbf{p^G} = Q_{MG}^{-1}\mathbf{p^M} = Q_{GM}\mathbf{p^M}, \tag{3-7}$$

onde Q_{GM} é a transformação de corpo rígido que leva do sistema de coordenada do marcador para o global. No apêndice C é mostrado como montar essa matriz a partir de três vetores. Esse apêndice mostra como calcular a matriz Q que transforma de um sistema de coordenada qualquer U, desde que respeite a regra da mão direita, para o sistema de coordenada da base canônica. O método apresentado no apêndice C tem como dados de entrada dois vetores que representam o eixo Z e o eixo Y do sistema de coordenada S na base canônica, além da posição do centro de coordenadas de U no sistema de coordenadas canônica.

Sendo assim para calcularmos Q_{MG} basta termos $\mathbf{p_c}$, $\vec{n}$ e o vetor para o eixo Y é $ei\vec{x}o_y = (\mathbf{p_1} - \mathbf{p_4})$.

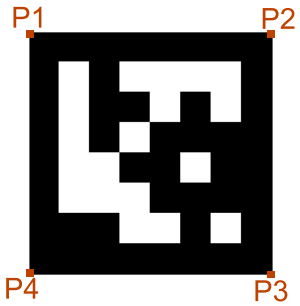


Figura 3.7: Configuração dos pontos medidos no marcador fiducial

A seleção dos pontos a serem acompanhados pelo SLAM é fácil nesse tipo de marcador, pois os vértices do retângulo são perfeitamente localizados. Outra questão facilitadora é que nenhum cálculo é necessário para encontrar os pontos em coordenadas globais, pois eles são exatamente os pontos de medição $\mathbf{p_1}$ $\mathbf{p_2}$ $\mathbf{p_3}$ $\mathbf{p_4}$. Além disso, o fator métrico é apenas a medição do tamanho do lado do retângulo que pode ser tanto calculado por $||\mathbf{p_1} - \mathbf{p_2}||$ quanto fornecido pelo usuário, já que o marcador fiducial foi impresso por ele.

3.3.3 Mapeamento dos marcadores naturais

A medição dos marcadores naturais é um pouco mais complicada, pois os pontos medidos pela estação total precisam ser marcados manualmente na imagem para fazer a correspondência, já que a textura natural não tem artefatos predefinidos como os fiduciais. Para fazer a medição da posição em $\mathbb{R}^3$ do marcador é necessário escolher 3 pontos na textura,$\mathbf{b_1} = (u, v, 1)^T$,$\mathbf{b_2}$ e $\mathbf{b_3}$. Depois mede-se suas posições correspondentes no espaço 3D $\mathbf{b'_1} = (x, y, z, 1)$,$\mathbf{b'_2}$,$\mathbf{b'_3}$, calculadas com a estação total, de preferência o mais separados possíveis e não colineares para que seja possível determinar um plano. O ideal é que sejam na forma de L, igual ao $\mathbf{p_1}$,$\mathbf{p_3}$,$\mathbf{p_4}$ na figura 3.7, mas muitas vezes isso não é possível como mostrado na figura 3.8. O sistema de coordenadas calculado será sempre considerando que $\mathbf{b_1}$ está mais próximo ao canto inferior esquerdo, $\mathbf{b_2}$ está mais próximo ao canto inferior direito e $\mathbf{b_3}$ está mais próximo ao canto superior esquerdo.

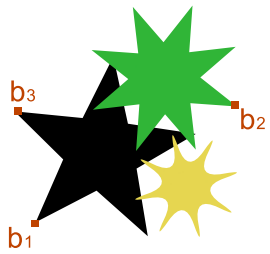


Figura 3.8: Configuração dos pontos medidos no marcador natural

O centro do sistema de coordenadas do marcador natural vai ser o ponto médio dos 3 pontos medidos igual ao dos marcadores fiduciais, mas isso não garante mais que será o centro da textura já que ele vai depender de $\mathbf{b_1}$,$\mathbf{b_2}$ e $\mathbf{b_3}$. As fórmulas para cálculo do centro e da normal no sistema do modelo e da imagem são

$$\mathbf{b_c} = \frac{1}{3}(\mathbf{b_1} + \mathbf{b_2} + \mathbf{b_4}) \tag{3-8}$$

$$\mathbf{b'_c} = \frac{1}{3}(\mathbf{b'_1} + \mathbf{b'_2} + \mathbf{b'_4}) \tag{3-9}$$

$$\vec{n} = (\mathbf{b_2} - \mathbf{b_1}) \times (\mathbf{b_3} - \mathbf{b_1}) \tag{3-10}$$

$$\vec{n'} = (\mathbf{b'_2} - \mathbf{b'_1}) \times (\mathbf{b'_3} - \mathbf{b'_1}) \tag{3-11}$$

Sendo assim para calcularmos Q_{MG} basta termos $\mathbf{b'_c}$, $\vec{n'}$ e o vetor para o eixo Y é $ei\vec{x}o_y = (\mathbf{b'_3} - \mathbf{b'_1})$. Os dados $\mathbf{b_c}$, $\vec{n}$ serão usados a seguir.

O rastreador de marcadores naturais detecta os pontos-chave do modelo($\mathbf{m'} = (x\ y\ 1)^T$) em relação a uma das extremidades da imagem, no nosso caso temos o canto inferior esquerdo crescendo para o cima, e na escala em pixels. Para que o rastreador calcule a posição da câmera na mesma escala e considerando o mesmo centro é necessário corrigir a posição dos pontos da seguinte forma

$$\breve{\mathbf{m}} = s(\mathbf{m'} - \mathbf{b_c}), \tag{3-12}$$

onde s é o fator de escala. Sabendo que a relação entre Q_{MG} é uma transformação de corpo rígido, então ela preserva os tamanhos relativos e as distâncias, logo $s = ||\mathbf{b_2} - \mathbf{b_1}||/||\mathbf{b'_2} - \mathbf{b'_1}||$.

Os pontos característicos que serão acompanhados pelo SLAM precisam ser facilmente identificados pelo seu detector, mas diferentemente dos marcadores fiduciais, os marcadores naturais não contam com o retângulo preto ao redor, então é necessário realizar um processamento de imagem para encontrá-los. O processamento é, resumidamente, executar o detector na textura e escolher dentre os pontos detectados os mais afastados no formato mostrado na figura 3.8.

3.4 Algoritmo de rastreamento usando SLAM

O algoritmo proposto nesse trabalho, teoricamente, não está atrelado a nenhum algoritmo de SLAM específico. Um algoritmo de SLAM sempre depende de uma posição a partir de onde constrói um mapa para manter sua posição conhecida. Esse é a única assertiva que assumimos para propor o conceito de Local SLAM. No entanto, é preciso existir uma forma de mapear as características do marcador com o mapa do algoritmo de SLAM. Para fim de demonstração de conceito, vamos usar o algoritmo de Davison[14]. Ele foi escolhido por algumas razões: o próprio autor do algoritmo disponibiliza uma implementação para o algoritmo; a biblioteca é bem documentada; a implementação funciona em tempo real; tem se mostrado funcional para fins de demonstração de conceitos em diversos artigos científicos. O funcionamento

desse algoritmo já foi explicado com maiores detalhes na seção de trabalhos relacionados. A seguir será dado apenas um breve resumo a fim de facilitar o entendimento.

Algoritmo 1 algoritmo de atualização do Algoritmo de Davison

1: **loop**
2: captura novo quadro da câmera
3: preditar posição da câmera no quadro usando filtro de Kalman Estendido(EKF).
4: dentro do conjunto de pontos conhecidos do mapa, selecionar os que estão visíveis.
5: procurar os retalhos de cada ponto na região onde o ponto deveria estar baseado na posição preditada pelo EKF
6: Aplicar o corretor do EKF para corrigir a posição da câmera usando os pontos localizados.
7: Processo de adicionar novos pontos ao mapa.
8: **end loop**

Nesse algoritmo, não existe nenhum tipo de inicialização. Ele considera que a câmera está sempre à mesma distância de um retângulo preto, como mostrado na figura 3.9 e o mapa contem apenas os quatro vértices do retângulo. Então para ser possível fazer a inicialização com outros tipos de marcadores e de diferentes posições da câmera, o algoritmo foi modificado. A modificação foi a inclusão de um algoritmo substitutivo ao algoritmo original 1 para quando for requisitado uma nova inicialização. Quando não for necessária a inicialização, o algoritmo 1 é executado. O algoritmo de inicialização 2 tem como parâmetros de entrada os dados que são fixos no algoritmo tradicional e a imagem sobre o qual os parâmetros foram calculados. Eles são a posição da câmera em coordenadas globais, a posição dos retalhos do marcador em coordenadas globais e a posição deles na imagem da câmera atual.

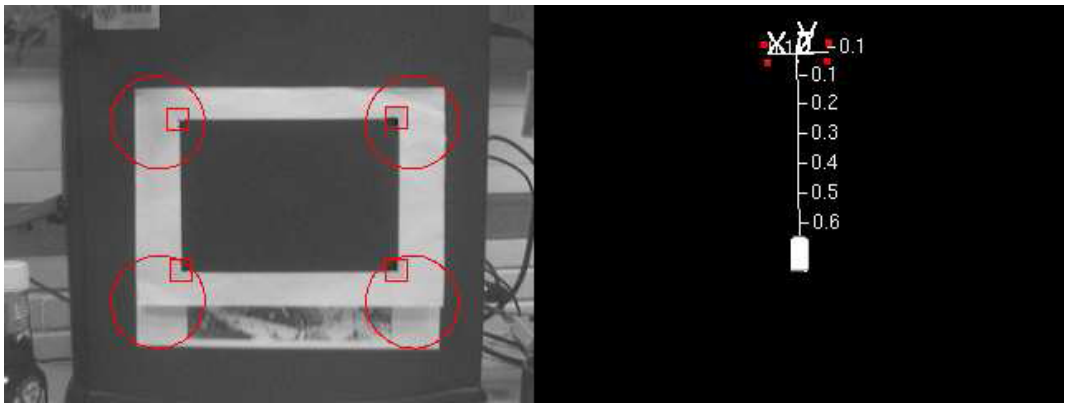


Figura 3.9: Configuração padrão de inicialização do algoritmo de Davison

Algoritmo 2 Processo de inicialização para o Algoritmo de Davison

1: adiciona os retalhos conhecidos ao mapa.
2: posiciona a câmera usando a posição dada como entrada.
3: preditar posição da câmera no quadro usando filtro de Kalman Estendido(EKF).
4: Associa os pontos 3D conhecidos do marcador com suas posições no plano da imagem.
5: Aplicar o corretor do EKF para corrigir a posição da câmera usando os pontos localizados.
6: Sai da inicialização e volta para o passo 1 do processo normal.

Na próxima seção será explicado quando usar o algoritmo 1 ou o algoritmo 2, ou seja, quando é necessário fazer a inicialização.

3.5 Heurísticas de reinicialização

Definir quando reinicializar o algoritmo de SLAM não é uma tarefa óbvia, pois queremos reinicializar quando dispomos de uma posição de maior qualidade dos rastreadores de marcadores, ao mesmo tempo em que não queremos piorar o rastreamento que muitas vezes pode ter uma posição da câmera mais precisa. O rastreador de marcadores fiduciais é bem confiável, mas o rastreador de marcadores naturais pode dar resultados errados do posicionamento da câmera, caso o algoritmo de RANSAC não consiga remover os pontos correspondentes errados. Além do mais, os dois tipos de marcadores se vistos de muito longe também não fornecem uma posição boa para a câmera por conta da pouca área na imagem que eles ocupam. Sendo assim, resolvemos propor duas Heurísticas de reinicialização para serem testadas.

3.5.1 Heurística 1 - Quando possível

Nessa heurística, toda vez que os rastreadores encontrarem um marcador o algoritmo de SLAM passa pela inicialização. Essa heurística é baseada na suposição que os rastreadores de marcador são sempre melhores que a posição dada pelo SLAM. O algoritmo de SLAM seria apenas um interpolador de posições entre marcadores. No entanto, a reinicialização não pode impedir que o SLAM mapei um número mínimo de pontos antes do marcador sair da imagem. Por conta disso, estabelecemos duas regras. A primeira regra é o número mínimo de pixel que o marcador precisa ocupar na imagem para conseguirmos uma reinicialização de qualidade. O segundo é que o marcador precisa estar a mais de um certo numero de pixel da borda da imagem, pois

nessa posição ele está prestes a sair da imagem então precisamos criar um mapa ao redor antes que ele saia da imagem e a calibração seja impossível.

3.5.2 Heurística 2 - Apenas com marcador no centro

Nessa heurística, quando os rastreadores encontrarem um marcador no centro da imagem, de frente para câmera e com uma área na imagem mínima, o algoritmo de SLAM passa pela inicialização. Essa heurística é baseada na ideia que quando a câmera está de frente, centralizada e o marcador ocupando uma boa área da imagem os rastreadores funcionam com mais precisão.

3.6 Processamento geral

Considerando que hoje em dia a maioria dos processadores tem mais de um núcleo, o algoritmo foi desenhado para funcionar utilizando dois processos em paralelo. A primeira parte do algoritmo tem a tarefa de executar o processamento do Visual SLAM. Como esse processo tem coerência temporal, é vital que não tenha gargalos. O segundo processo ficou responsável por detectar e rastrear os marcadores. Esse processo foi chamado de relocalizador, já que ele tenta calcular constantemente os dados para reposicionar o algoritmo de SLAM. Esse processo não tem a obrigação de funcionar em tempo real já que sempre que o rastreador de marcadores naturais for executado o tempo real será perdido. Cada um dos processos vai ser mais detalhado abaixo.

3.6.1 Relocalizador

A programação paralela pode trazer uma série de problemas de concorrência. Então decidimos por manter esse processo com um mínimo de acoplamento com o processo principal. Ele tem um espaço de memória protegido por semáforos para entrada de dados e outro, também protegido, para saída de dados. No espaço de entrada fica armazenada a próxima imagem a ser processada, a última posição da câmera calculada com sucesso e a marca temporal (timestamp) na qual a imagem foi capturada. No espaço de saída é armazenado o resultado do último marcador encontrado com sucesso. O resultado é composto pela posição da câmera em coordenadas globais (coordenada da origem e orientação da câmera), pelos retalhos juntamente com suas posições tanto em coordenadas globais quanto na imagem e pela marca de tempo da imagem de onde esses dados foram calculados.

O processamento principal do Relocalizador é basicamente: caso haja uma imagem nova disponível no espaço de entrada, procurar um marcador fiducial, caso não encontre, procura um marcador natural. Se alguma das buscas resultar em sucesso, corrige a posição da câmera para coordenadas globais e a copia para o espaço de saída juntamente com os dados adicionais. O marcador fiducial é procurado primeiro, pois ele funciona em tempo real. Sendo assim, caso ele tenha sucesso, é possível manter o relocalizador funcionando em tempo real. Isso não seria possível se o rastreador de marcadores naturais fosse feito primeiro, já que ele não consegue atingir tempo real.

3.6.2 Processo de Atualização do Visual SLAM

A tarefa desse processo é executar a atualização do SLAM e decidir quando reinicializá-lo. Esse processo apenas testa se a heurística de reinicialização é verdadeira ao mesmo tempo que existe a detecção de um marcador no Relocalizador feita há pouco tempo atrás. Se essas duas condições são satisfeitas o algoritmo de inicialização é executado, caso contrário a atualização original é feita no algoritmo de SLAM.

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

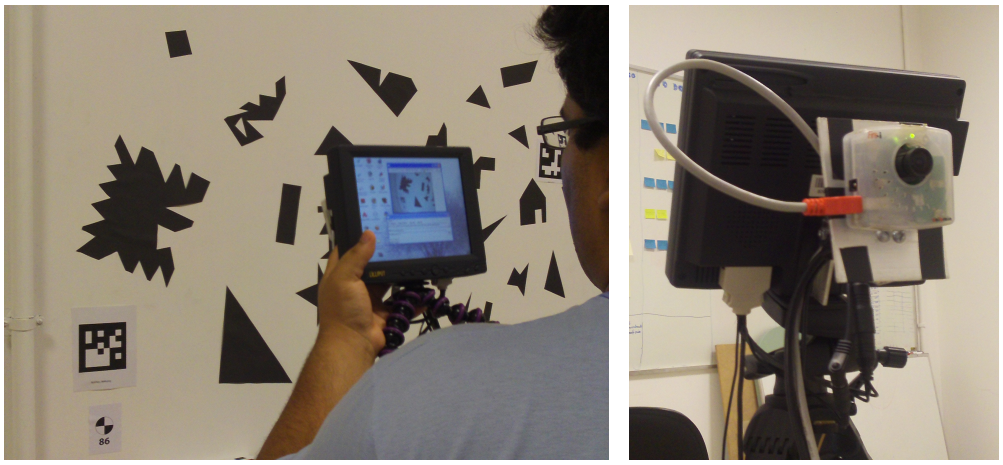


Figura 4.1: Equipamento utilizado.

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.

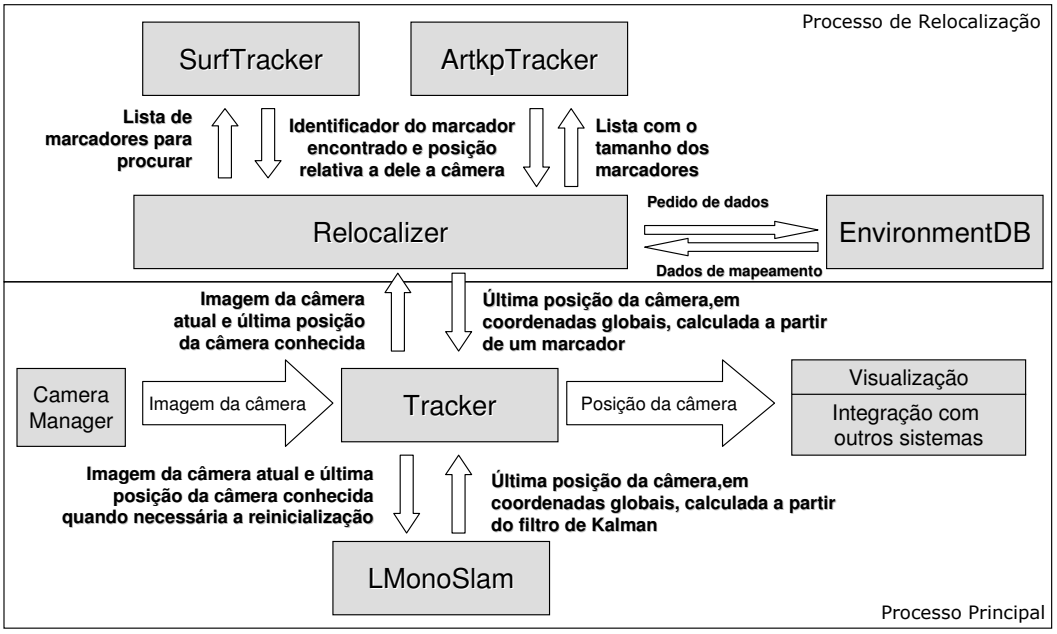


Figura 4.2: Arquitetura resumida da implementação do algoritmo proposto.

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.

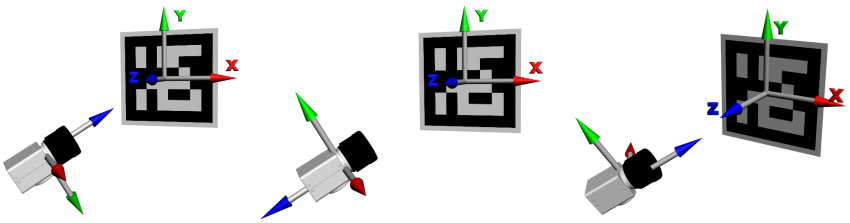


Figura 4.3: Sistema de coordenada do ArToolKitPlus, no meio, do SurfTracking e do OpenGL e, ao final, do SLAM de Davison

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

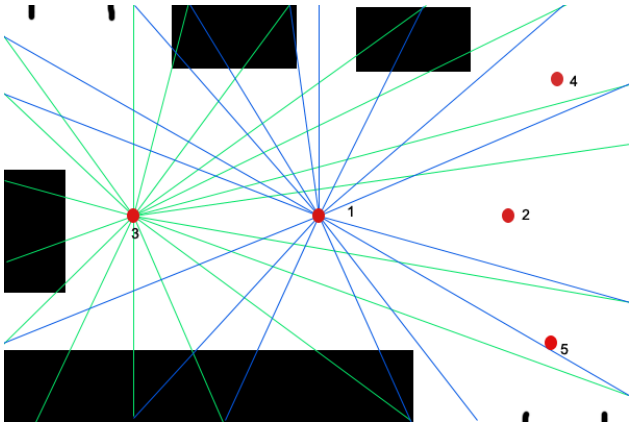
A estação total é na verdade um instrumento de Engenharia Cartográfica(Surveying). Ela foi projetada originalmente para medir prédios, grandes propriedades de terra ou mesmo cidades. Então decidimos testar a precisão real desse equipamento para fazer mapeamento 3D de uma sala, por conta de ser uma aplicação 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

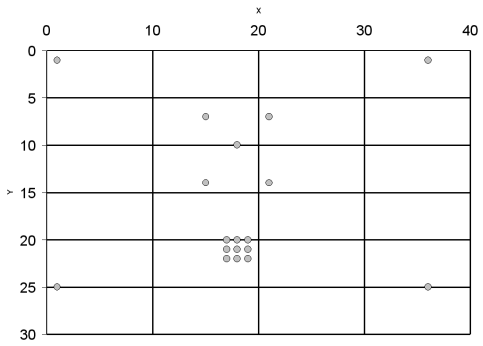
Tabela 4.1: Tabela de resultados de medida para sala

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.

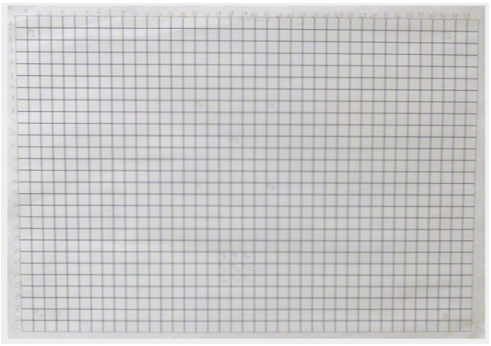
Na tabela 4.1 podemos observar que o erro para cada uma das medidas foi inferior a 5 mm e que ele diminui com a inclusão de mais medidas.

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.

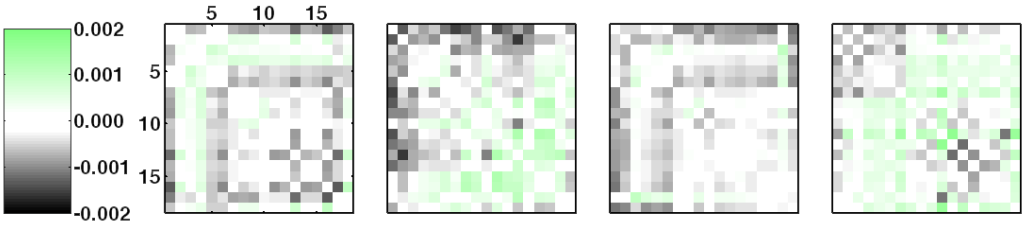


Figura 4.6: Da esquerda para direita as matrizes de distância das medidas à 20m, 12m, 8m e 9m. A unidade do gráfico está em metros.A matriz tem tamanho 18x18 já que foram 18 pontos medidos de cada distância

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

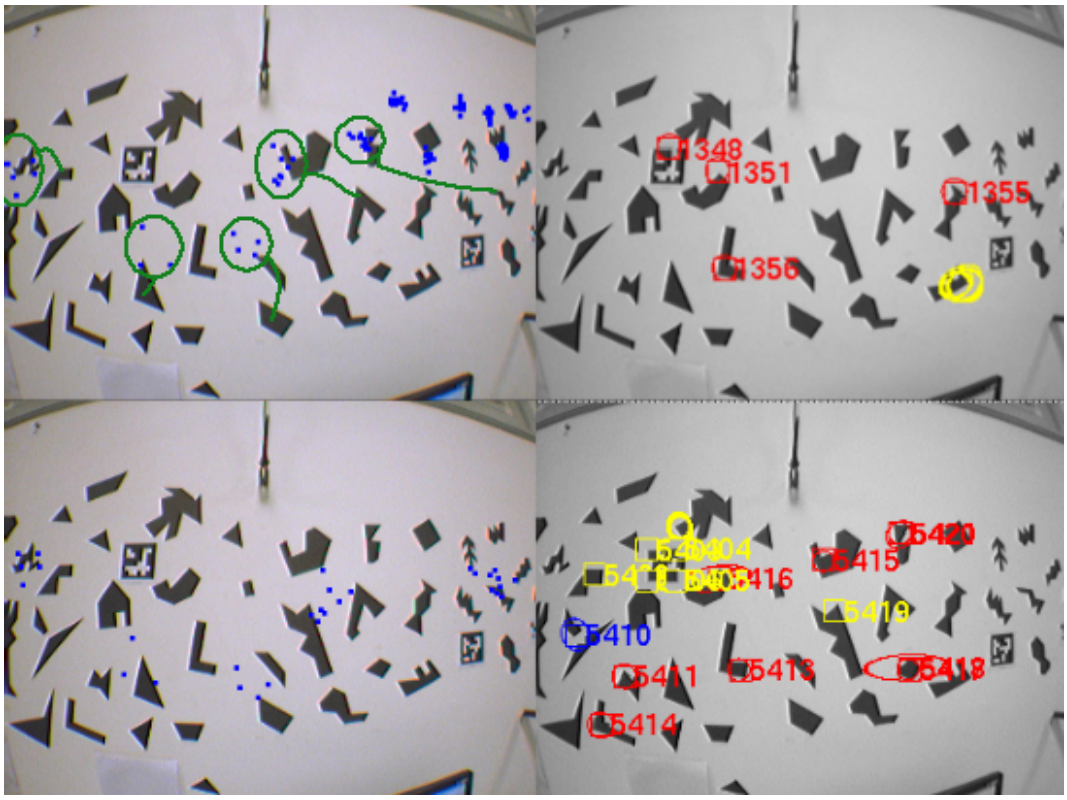


Figura 4.9: Diferença no mapeamento gerado pelas diferentes políticas de reinicialização. Em cima à esquerda, a reprojeção das quinas dos objetos conhecidos na parede pelo algoritmo LSLAM-QP e à direita o mapa de características acompanhas pelo SLAM correspondente. Abaixo, o mesmo para o algoritmo LSLAM-MC.

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-

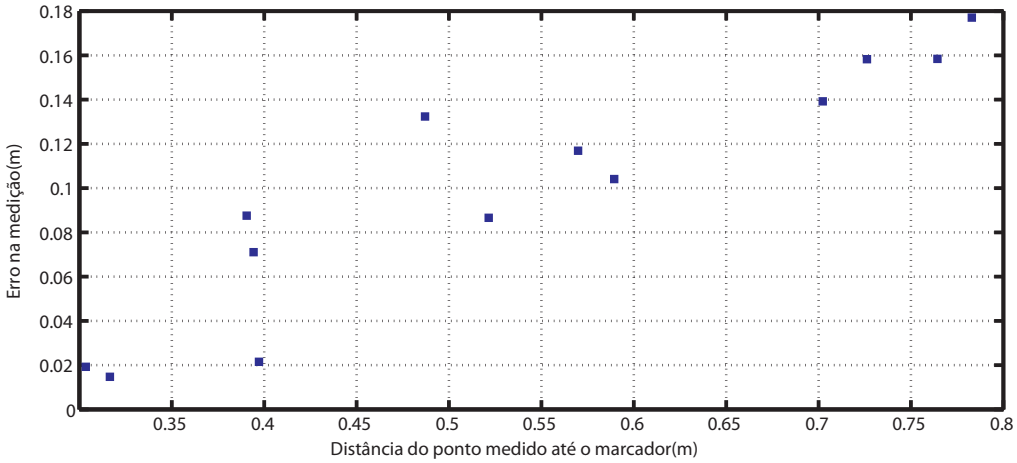
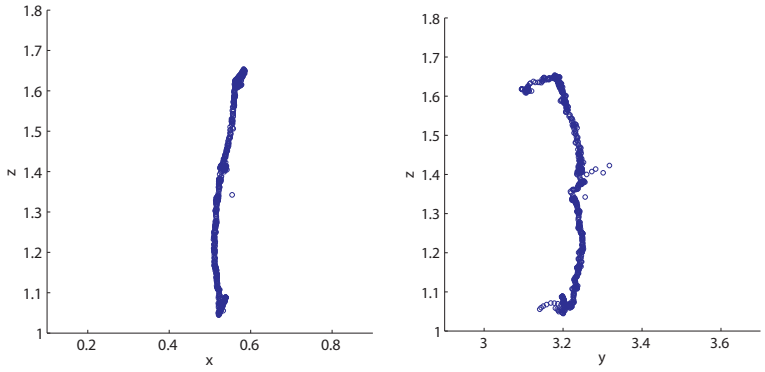


Figura 4.10: Gráfico do erro em cada retalho mapeado pelo SLAM pela distância ao marcador inicial.

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.

5
Conclusão e Trabalhos Futuros

Nessa dissertação apresentamos um algoritmo para estimação de posição de câmera baseado em técnicas de visão computacional. Para isso, propromos uma nova abordagem no uso das técnicas de SLAM. Essa que tradicionalmente é usada para navegação por ambientes desconhecidos e foi alterada para navegação por ambientes conhecidos. A capacidade de gerar o mapa do ambiente a medida que a câmera se locomove foi utilizada para a movimentação entre depois pontos conhecidos. Esses pontos conhecidos são identificados com marcadores naturais ou fiduciais. Esse algoritmo foi chamado de Local SLAM, pois usamos a técnica de SLAM apenas localmente.

Pelos testes feitos, o algoritmo mostrou ser capaz de rastrear a posição da câmera por ambientes conhecidos. No entanto, o algoritmo de SLAM usado, de Davison, se mostrou capaz apenas de rastrear a posição da câmera com qualidade por uma distância muito pequena a apartir do marcador no qual foi inicializado. Sendo assim, sua adoção se limita ao uso com marcadores naturais, pois esses rastreadores não funcionam em tempo real e pelos testes feitos o algoritmo Local SLAM funciona razoavelmente bem com marcadores naturais, embora ele ainda não tem condições de suporte a uma aplicação comercial.

Como o fundamento principal do algoritmo proposto é a constante reinicialização em paralelo com um algoritmo de SLAM, para trabalhos futuros sugerimos em algoritmos alternativos nessas duas linhas. Para reinicialização seria de grande utilidade um algoritmo de reinicialização baseado em objetos não planares, como por exemplo, rastreadores de estruturas geométricas presentes no ambiente como linhas, quadrados e círculos. Algoritmo de detecção de movimento em imagens borradas(motion blur) também ajudariam a não perder o rastreamento resultante de movimentos bruscos do usuário. Na linha de algoritmos de SLAM, o algoritmo que usamos é baseado no filtro de Kalman Estendido. Como extensão deste trabalho, seria interessante testar outros filtros que sejam capazes de fazer estimação de posição, apesar de até hoje na literatura não tenham um filtro tão geral e rápido para substituir o filtro de Kalman. Uma possibilidade seria tentar adicionar sensores como

acelerômetros e giroscópios para ajudar a melhorar o rastreamento e permitir aumentar a distância entre os marcadores no ambiente.

Referências Bibliográficas

[1] ANDTOM DRUMMOND, E. E. **Monocular slam as a graph of coalesced observations**. In: IEEE 11TH INTERNATIONAL CONFERENCE ON COMPUTER VISION, p. 1–8, October 2007.

[2] AZUMA, R. T. **A survey of augmented reality**. Presence, 6:355–385, 1997.

[3] BAUER, J.; SÜNDERHAUF, N. ; PROTZEL, P. **Comparing several implementations of two recently published feature detectors**.

[4] BAY, H.; ESS, A.; TUYTELAARS, T. ; GOOL, L. V. **Speeded-up robust features (surf)**. Computer Vision and Image Understanding, 110(3):346 – 359, 2008. Similarity Matching in Computer Vision and Multimedia.

[5] BAY, H.; FASEL, B. ; GOOL, L. V. **Interactive museum guide: Fast and robust recognition of museum objects**. In: PROCEEDINGS OF THE FIRST INTERNATIONAL WORKSHOP ON MOBILE VISION, May 2006.

[6] BAY, H.; TUYTELAARS, T. ; GOOL, L. V. **Surf: Speeded up robust features**. In: IN ECCV, p. 404–417, 2006.

[7] BOUGUET, J. Y. **Camera calibration toolbox for matlab**, 2008.

[8] BRADSKI, G. **The OpenCV Library**. Dr. Dobb's Journal of Software Tools, 2000.

[9] BRUNS, E.; BROMBACH, B.; ZEIDLER, T. ; BIMBER, O. **Enabling mobile phones to support large-scale museum guidance**. IEEE MultiMedia, 14(2):16–25, 2007.

[10] CASTLE, R.; KLEIN, G. ; MURRAY, D. W. **Video-rate localization in multiple maps for wearable augmented reality**. In: ISWC '08: PROCEEDINGS OF THE 2008 12TH IEEE INTERNATIONAL SYMPOSIUM ON WEARABLE COMPUTERS, p. 15–22, Washington, DC, USA, 2008. IEEE Computer Society.

[11] CASTLE, R. O.; GAWLEY, D. J.; KLEIN, G. ; MURRAY, D. W. **Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras**. In: PROC. INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, ROME, ITALY, APRIL 10-14, 2007, p. 4102–4107, 2007.

[12] CASTLE, R. O.; GAWLEY, D. J.; KLEIN, G. ; MURRAY, D. W. **Video-rate recognition and localization for wearable cameras**. In: PROC 18TH BRITISH MACHINE VISION CONFERENCE, WARWICK, SEPT 11-13, 2007, p. 1100–1109, 2007.

[13] CLEMENTE, L. A.; DAVISON, A. J.; REID, I. D.; NEIRA, J. ; TARDÓS, J. D. **Mapping large loops with a single hand-held camera**. In: Burgard, W.; Brock, O. ; Stachniss, C., editors, PROCEEDINGS OF ROBOTICS: SCIENCE AND SYSTEMS, Atlanta, GA, USA, June 2007. The MIT Press.

[14] DAVISON, A. J. **Real–time simultaneous localisation and mapping with a single camera**. In: ICCV '03: PROCEEDINGS OF THE NINTH IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION, volume 2, p. 1403–1410, Washington, DC, USA, 2003. IEEE Computer Society.

[15] EADE, E.; DRUMMOND, T. **Unified loop closing and recovery for real time monocular slam**. In: 19TH BRITISH CONFERENCE ON MACHINE VISION, BMVC' 08, p. 1–4, september 2008.

[16] EVANS, C. **Notes on the opensurf library**. Technical Report CSTR-09-001, University of Bristol, January 2009.

[17] FIALA, M. **Artag, a fiducial marker system using digital techniques**. In: CVPR '05: PROCEEDINGS OF THE 2005 IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR'05) - VOLUME 2, volume 2, p. 590–596, Washington, DC, USA, 2005. IEEE Computer Society.

[18] FÖCKLER, P.; ZEIDLER, T.; BROMBACH, B.; BRUNS, E. ; BIMBER, O. **Phoneguide: museum guidance supported by on-device object recognition on mobile phones**. In: MUM '05: PROCEEDINGS OF THE 4TH INTERNATIONAL CONFERENCE ON MOBILE AND UBIQUITOUS MULTIMEDIA, p. 3–10, New York, NY, USA, 2005. ACM.

[19] G., S.; D., W.; G., R.; E., T.; M., W.; D., S. ; B., H. W. **Global pose estimation using multi-sensor fusion for outdoor augmented reality**. In: ISMAR '09: PROCEEDINGS OF THE 2009 8TH IEEE INTERNATIONAL SYMPOSIUM ON MIXED AND AUGMENTED REALITY, p. 153–162, Washington, DC, USA, October 2009. IEEE Computer Society.

[20] HARTLEY, R. I.; ZISSERMAN, A. **Multiple View Geometry in Computer Vision**. Cambridge University Press, New York, NY, USA, second edition, 2004.

[21] JUAN, L.; GWON, O. **A comparison of sift, pca-sift and surf**. volume 3, p. 143–152, 2009.

[22] KATO, H.; BILLINGHURST, M. **Marker tracking and hmd calibration for a video-based augmented reality conferencing system**. In: IWAR '99: PROCEEDINGS OF THE 2ND IEEE AND ACM INTERNATIONAL WORKSHOP ON AUGMENTED REALITY, p. 85, Washington, DC, USA, 1999. IEEE Computer Society.

[23] KLEIN, G. Tese de Doutorado, University of Cambridge, Cambridge,UK, 2006.

[24] KLEIN, G.; MURRAY, D. **Parallel tracking and mapping for small ar workspaces**. In: MIXED AND AUGMENTED REALITY, 2007. ISMAR 2007. 6TH IEEE AND ACM INTERNATIONAL SYMPOSIUM ON, p. 225–234, June 2008.

[25] LIEBERKNECHT, S.; BENHIMANE, S.; MEIER, P. ; NAVAB, N. **A dataset and evaluation methodology for template-based tracking algorithms**. In: ISMAR '09: PROCEEDINGS OF THE 2009 8TH IEEE INTERNATIONAL SYMPOSIUM ON MIXED AND AUGMENTED REALITY, p. 145–151, Washington, DC, USA, 2009. IEEE Computer Society.

[26] LOWE, D. G. **Distinctive image features from scale-invariant keypoints**, volume 60, p. 91–110. Kluwer Academic Publishers, Hingham, MA, USA, 2004.

[27] MIYASHITA, T.; MEIER, P. G.; TACHIKAWA, T.; ORLIC, S.; EBLE, T.; SCHOLZ, V.; GAPEL, A.; GERL, O.; ARNAUDOV, S. ; LIEBERKNECHT, S. **An augmented reality museum guide**. In: ISMAR, p. 103–106, 2008.

[28] MOURIKIS, A. I.; ROUMELIOTIS, S. I. **Predicting the accuracy of Cooperative Simultaneous Localization and Mapping (C-SLAM)**. International Journal of Robotics Research, 25(12):1273–1286, December 2006.

[29] REIF, R.; GÜNTHNER, W. A.; SCHWERDTFEGER, B. ; KLINKER, G. **Pick-by-vision comes on age: evaluation of an augmented reality supported picking system in a real storage environment**. In: AFRIGRAPH '09: PROCEEDINGS OF THE 6TH INTERNATIONAL CONFERENCE ON COMPUTER GRAPHICS, VIRTUAL REALITY, VISUALISATION AND INTERACTION IN AFRICA, p. 23–31, New York, NY, USA, 2009. ACM.

[30] SCHALL, G.; MENDEZ, E. ; SCHMALSTIEG, D. **Virtual redlining for civil engineering in real environments**. In: ISMAR '08: PROCEEDINGS OF THE 7TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON MIXED AND AUGMENTED REALITY, p. 95–98, Washington, DC, USA, 2008. IEEE Computer Society.

[31] SCHMALSTIEG, D.; FUHRMANN, A.; HESINA, G.; SZALAVÁRI, Z.; ENCARNAÇÃO, L. M.; GERVAUTZ, M. ; PURGATHOFER, W. **The studierstube augmented reality project**, february 2002.

[32] SCHWERDTFEGER, B.; ; KLINKER, G. **Supporting order picking with augmented reality**. In: ISMAR '08: PROCEEDINGS OF THE 7TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON MIXED AND AUGMENTED REALITY, p. 91–94, Washington, DC, USA, September 2008. IEEE Computer Society.

[33] SHI, J.; TOMASI, C. **Good features to track**. In: IN: PROCEEDINGS OF IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 593–600, 1994.

[34] TOMASI, C.; KANADE, T. **Detection and tracking of point features**. Tech. rep., Carnegie Mellon University, April 1991.

[35] TUMLER, J.; DOIL, F.; MECKE, R.; PAUL, G.; SCHENK, M.; PFISTER, E. A.; HUCKAUF, A.; BOCKELMANN, I. ; ROGGENTIN, A. **Mobile augmented reality in industrial applications: Approaches for solution of user-related issues**. In: ISMAR '08: PROCEEDINGS OF THE 7TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON MIXED AND AUGMENTED REALITY, p. 87–90, Washington, DC, USA, 2008. IEEE Computer Society.

[36] VIOLA, P.; JONES, M. **Rapid object detection using a boosted cascade of simple features**. Computer Vision and Pattern Recognition, IEEE Computer Society Conference on, 1:511, 2001.

[37] WAGNER, D.; DIETER, S. **Artoolkitplus for pose tracking on mobile devices**. In: COMPUTER VISION WINTER WORKSHOP 2007, p. 139–146, February 2007.

[38] WAGNER, D.; SCHMALSTIEG, D. **First steps towards handheld augmented reality**. In: ISWC '03: PROCEEDINGS OF THE 7TH IEEE INTERNATIONAL SYMPOSIUM ON WEARABLE COMPUTERS, p. 127, Washington, DC, USA, 2003. IEEE Computer Society.

[39] WILLIAMS, B.; CUMMINS, M.; NEIRA, J.; NEWMAN, P.; REID, I. ; TARDÓS, J. **A comparison of loop closing techniques in monocular slam**. Robot. Auton. Syst., 57(12):1188–1197, 2009.

[40] ZHANG, Z. **A flexible new technique for camera calibration**. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11):1330–1334, 2000.

A
Videos

Como o resultado dessa dissertação são videos, ela será acompanhada de um DVD com os videos citados. Como interpretar os videos está explicado na figura A.1

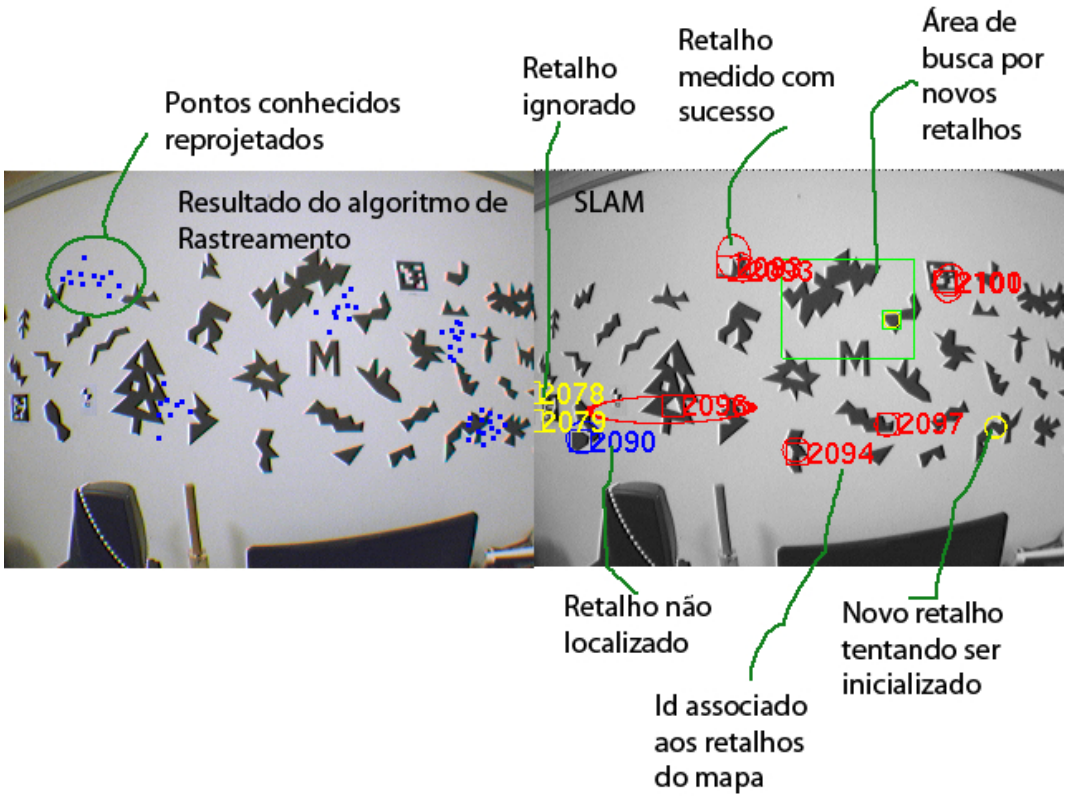


Figura A.1: Imagem que explica como interpretar os vídeos

Os vídeos usados nessa dissertação são:

- *ex1_parede_grande.avi* - resultado referenciado na seção 4.4
- *ex2_parede_pequena.avi* - resultado referenciado na seção 4.5
- *ex3_init_naturais.avi* - resultado referenciado na seção 4.6

B
Estimação dos parâmetros extrínsecos a partir da Homografia e dos parâmetros intrínsecos

A homografia calculada pelo algoritmo Direct Linear Transformation (DLT) [20] é uma transformação invertível que relaciona o plano da imagem com a sua projeção no espaço 3D. Assim, se conhecermos a transformação invertível que relaciona a posição da câmera com o plano da imagem, então será possível relacionar a posição da câmera com o espaço 3D. Essa é a ideia de como calcular a posição da câmera a partir da homografia. A transformação que relaciona o plano da imagem com a posição da câmera é a matriz de parâmetros intrínsecos, que vamos considerar como dado de entrada junto com a homografia. Sendo assim os parâmetros extrínsecos podem ser calculados da forma abaixo:

Dados de entrada:

-matriz dos parâmetros intrínsecos

$$A = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix},$$

onde (u_0, v_0) é o centro da imagem, α e β o fator de escala nos eixos u e v da imagem e por fim γ é o parâmetro que descreve a assimetria entre os dois eixos.

-Homografia

$$H = \begin{bmatrix} | & | & | \\ h_1 & h_2 & h_3 \\ | & | & | \end{bmatrix},$$

com $\mathbf{m} = H\mathbf{m}'$ onde $\mathbf{m} = (u, v, 1)$ é um ponto do plano na imagem e $\mathbf{m}' = (X, Y, 1)$ um ponto no modelo. h_1, h_2 e h_3 são os vetores coluna da homografia. Zhang considera que a coordenada Z do modelo é sempre zero, por isso a homografia é sempre 3x3.

Dados de Saída:

-Rotação R

$$R = \begin{bmatrix} | & | & | \\ r_1 & r_2 & r_3 \\ | & | & | \end{bmatrix}$$

-Translação t

$$t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

Cálculo:

O cálculo pode ser feito pelas seguintes equações:

$$\lambda_{r_1} = \frac{1}{\|A^{-1}h_1\|} \tag{B-1}$$

$$\lambda_{r_2} = \frac{1}{\|A^{-1}h_2\|} \tag{B-2}$$

$$\lambda_{r_3} = \frac{\lambda_{r_1} + \lambda_{r_2}}{2} \tag{B-3}$$

$$r_1 = \lambda_{r_1} A^{-1} h_1 \tag{B-4}$$

$$r_2 = \lambda_{r_2} A^{-1} h_2 \tag{B-5}$$

$$r_3 = r_1 \times r_2 \tag{B-6}$$

$$t = \lambda_{r_3} A^{-1} h_3 \tag{B-7}$$

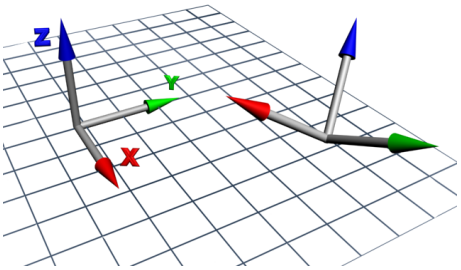
C
Cálculo da Transformação de Corpo Rígido

Esse apêndice descreve como calcular a matriz de transformação de um sistema de coordenada em $\mathbb{R}^3$ A para um outro sistema em $\mathbb{R}^3$ B. A figura C.1(a) exemplifica os dois sistemas, o sistema A sobre a grade e o sistema de base B em uma outra posição. Cada sistema de coordenadas pode ser entendido como um espaço vetorial e a transformação entre eles como uma mudança de base vetorial. Sendo assim, tendo dois vetores na base A que sejam colineares com dois eixos da base B e o ponto na base A que representa a origem da base B podemos calcular a transformação. Esse ponto vamos chamar de $\mathbf{p^A_{B0}}$. Como essa transformação é uma mudança de sistema de coordenada ela é uma transformação de corpo rígido, ou seja, é composta por uma rotação em $\mathbb{R}^3$ R e uma translação t. A matriz de transformação que leva da base A para a base B tem dimensão 4x4 e é descrita por:

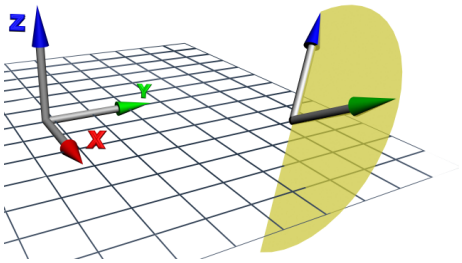
$$Q^{BA} = T(t)\dot{R} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \tag{C-1}$$

$$\mathbf{p^B} = Q^{BA}\mathbf{p^A}, \tag{C-2}$$

onde $T(w)$ é a matriz 4x4 que descreve a translação w e $\dot{R}$ é a matriz de rotação aumentada para o espaço homogêneo.



C.1(a): O sistema de coordenada A sobre a grade e B flutuando



C.1(b): Em amarelo o plano YZ da base B

A tranformação que leva o centro do sistema de coordenada de A para B deveria ser simples dado que temos a posição de um em relação ao outro,

mas a translação t não é possivel de ser calculada diretamente, visto que, se t não tiver norma nula, então a rotação R acontecerá fora da origem de B. Por consequência, a transformação que movimenta o centro do sistema de coordenada é uma composição da rotação R e t. Como não conhecemos R, ainda, fica impossível de calcular t. No entanto, se reformularmos o cálculo de Q^{BA} de forma que uma outra translação t' aconteça antes da da rotação,como descrito na equação C-3, o cálculo se simplifica, posto que a rotação acontecerá com os centros das bases A e B sobrepostos. Nesses termos, a translação que leva o centro de A para o centro de B é $t' = -\mathbf{p^A_{B0}}$.

$$Q^{BA} = \dot{R}\, T(t') = \begin{bmatrix} R & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} I & t' \\ 0^T & 1 \end{bmatrix} \tag{C-3}$$

Para calcular a rotação R, basta calcular os três vetores unitários que formam a sua base. Dois deles são dados de entrada, e como os eixos são ortogonais entre si, o terceiro pode ser calculado pelo produto vetorial dos outros dois. Vamos assumir que os vetores dados sejam um colinear a Z, que vamos chamar de v_z, e um outro vetor sobre o plano YZ na base B com a coordenada Y positiva não nula,que vamos chamar de v_{yz}. Esse subespaço vetorial está representado na figura C.1(b) em amarelo. Vamos assumir que esses sistemas de coordenadas seguem a regra da mão direita. Sendo assim o cálculo de R é dado por:

$$\vec{v_z} = \frac{v_z}{norma(v_z)} \tag{C-4}$$

$$\vec{v_{yz}} = \frac{v_{yz}}{norma(v_{yz})} \tag{C-5}$$

$$\vec{v_x} = \vec{v_{yz}} \times \vec{v_z} \tag{C-6}$$

$$\vec{v_y} = \vec{v_z} \times \vec{v_x} \tag{C-7}$$

$$R = \begin{bmatrix} - & \vec{v_x} & - \\ - & \vec{v_y} & - \\ - & \vec{v_z} & - \end{bmatrix} \tag{C-8}$$