PAULO ROBERTO GODOI DE OLIVEIRA

**AUTO-LOCALIZAÇÃO E CONSTRUÇÃO DE MAPAS DE AMBIENTE
PARA ROBÔS MÓVEIS BASEADOS EM VISÃO OMNIDIRECIONAL
ESTÉREO**

São Paulo
2008

PAULO ROBERTO GODOI DE OLIVEIRA

**AUTO-LOCALIZAÇÃO E CONSTRUÇÃO DE MAPAS DE AMBIENTE
PARA ROBÔS MÓVEIS BASEADOS EM VISÃO OMNIDIRECIONAL
ESTÉREO**

Dissertação submetida à Escola
Politécnica da Universidade de São
Paulo para obtenção do título de Mestre
em Engenharia

São Paulo
2008

PAULO ROBERTO GODOI DE OLIVEIRA

**AUTO-LOCALIZAÇÃO E CONSTRUÇÃO DE MAPAS DE AMBIENTE
PARA ROBÔS MÓVEIS BASEADOS EM VISÃO OMNIDIRECIONAL
ESTÉREO**

Dissertação submetida à Escola
Politécnica da Universidade de São
Paulo para obtenção do título de Mestre
em Engenharia

Área de concentração:
Engenharia de Controle e Automação
Mecânica

Orientador :
Prof. Dr. Eduardo Lobo Lustosa Cabral

São Paulo
2008

FICHA CATALOGRÁFICA

Oliveira, Paulo Roberto Godoi de

Auto-localização e construção de mapas de ambiente para robôs móveis baseados em visão omnidirecional estéreo / P.R.G. de Oliveira -- São Paulo, 2008.

164 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1. Visão computacional 2. Visão omnidirecional estéreo 3. Auto-localização e mapeamento simultâneos I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II. t.

AGRADECIMENTOS

Agradeço especialmente ao meu pai, por todo amor, amizade, apoio, confiança, respeito e sabedoria durante esta longa jornada, não somente pessoal mas também profissional. Agradeço ao meu orientador, Professor Doutor Eduardo Lobo Lustosa Cabral, pela oportunidade de ser seu orientado, pelos três anos e meio de trabalho juntos neste projeto, inúmeras horas de leitura e correções de textos, apoio nos momentos difíceis, confiança no meu trabalho e nas minhas decisões. Agradeço ao apoio da Fundação de Amparo à Pesquisa do Estado de São Paulo, FAPESP, pela confiança e apoio neste projeto.

Foram muitos os amigos que fizeram parte da minha vida durante a graduação e pós-graduação e estes dedicaram inúmeras horas de conversa e trocas de idéias, opiniões, dúvidas e sugestões sobre o presente projeto, sendo que, cada um deles a sua maneira contribuiu de forma importante para a existência deste documento, portanto agradeço aos amigos: Carlos Corteline, Cesar Kiyono, Daniel Marques, Danillo Nakano, Erick Takarabe, Fabio Kobiraki, Fausto Chinen, Geraldo Vidal, Giovane Amianti, Gustavo Katsuragi, Leandro Malavazi, Marco Gomes, Marcos Tsuchie, Renato Otake, Rodrigo Alves, Rodrigo Ferrarezi, Rodrigo Polastro e Thiago Pappalardo e as amigas: Catia Petri, Daniela Agonila, Erika Kawamura, Flavia Ribeiro, Gizela Fonseca e Paola Neves.

Um agradecimento especial a minha família, Priscilla e Angela, por conseguirem agüentar um politécnico em casa...

RESUMO

Este projeto consiste no desenvolvimento de um sistema para auto-localização e construção de mapas de ambiente para robôs móveis em um ambiente estruturado, ou seja, que pode ser descrito através de primitivas geométricas. O mapa é construído a partir da reconstrução de imagens adquiridas por um sistema de visão omnidirecional estéreo baseado em um espelho duplo de perfil hiperbólico. A partir de uma única imagem obtida, utilizando-se algoritmos de visão estéreo, realiza-se a reconstrução tridimensional do ambiente em torno do robô e, assim, obtêm-se as distâncias de objetos presentes no ambiente ao sistema de visão. A partir da correspondência da reconstrução de várias imagens tomadas em diferentes posições cria-se o mapa do ambiente.

Além do mapa global do ambiente o sistema também realiza o cálculo da localização do robô no ambiente utilizando informações obtidas na correspondência da reconstrução da sequência de imagens e a odometria do robô.

O sistema de construção de mapas de ambiente e auto-localização do robô é testado em um ambiente virtual e um ambiente real. Os resultados obtidos tanto na construção do mapa global do ambiente, como na localização do robô, mostram que o sistema é capaz de obter informação com a acuracidade necessária para permitir a sua utilização para navegação de robôs móveis. O tempo computacional necessário para reconstruir as imagens, calcular a posição do robô e criar o mapa global do ambiente possibilita que o sistema desenvolvido seja usado em uma aplicação que necessite da geração do mapa global em um intervalo de tempo na ordem de poucos segundos.

Ressalta-se que este projeto teve como ponto de partida um projeto de iniciação científica financiado pela FAPESP. Esse trabalho de iniciação científica foi publicado na forma de um trabalho de conclusão de curso (Oliveira, 2005).

ABSTRACT

This project aims the development of a system for self localization and environment map building for mobile robots in a structured environment. The map is built from images acquired by an omnidirectional stereo system with a hyperbolic double lobed mirror. From a single acquired image, using stereo vision algorithms, the environment around the robot is tridimensionally reconstruct and the distances of objects in the environment from the system are calculated. From the matching of several reconstructed environments obtained from images taken in different positions the global environment map is created.

Besides the global map the system also calculates the localization of the mobile robot using information obtained from the matching of the sequence of image reconstructions and the robot odometry.

The map building and robot localization system is tested in virtual and real environments. The computational time required to make the calculation is of the order of few seconds. The results obtained both for the map building and for the robot localization show that the system is capable of generating information with enough accuracy to allow it to be used for mobile robot navigation.

This project had as start point a scientific initiation project supported by FAPESP. The scientific initiation project was published as a graduation work (Oliveira, 2005).

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

1	INTRODUÇÃO	17
1.1	ESTADO DA ARTE	19
1.1.1	<i>Sistema de visão estéreo omnidirecional.....</i>	<i>19</i>
1.1.2	<i>Sistemas para localização de robôs móveis</i>	<i>23</i>
1.1.3	<i>Construção de mapas de ambiente</i>	<i>27</i>
1.1.4	<i>Problema de auto-localização e mapeamento simultâneos.....</i>	<i>29</i>
1.2	OBJETIVOS	31
1.3	JUSTIFICATIVAS	32
1.4	ESTRUTURA DA DISSERTAÇÃO	33
1.5	SUMÁRIO	35
2	ESPELHO DUPLO DE PERFIL HIPERBÓLICO	36
2.1	MODELO DE FORMAÇÃO DA IMAGEM	38
2.2	CÁLCULO DA DISTÂNCIA.....	40
2.3	CÁLCULO DO PERFIL DO ESPELHO	41
2.4	CARACTERÍSTICAS DO ESPELHO UTILIZADO	42
2.5	SUMÁRIO	43
3	AMBIENTES DE TESTE	44
3.1	AMBIENTE SIMULADO	44
3.2	AMBIENTE REAL	48
4	PROCESSO DE RECONSTRUÇÃO TRIDIMENSIONAL	54
4.1	CORRESPONDÊNCIA	55
4.1.1	<i>Programação dinâmica</i>	<i>56</i>
4.2	DETECÇÃO DAS BORDAS HORIZONTAIS	56
4.3	DETECÇÃO DE BORDAS VERTICAIS	60
4.4	SUMÁRIO	65
5	ALGORITMOS DE AUTO-LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS	66

5.1	TRIMMED ITERATIVE CLOSEST POINT (TRICP).....	66
5.1.1	<i>Cálculo do alinhamento entre a amostra e o modelo</i>	70
5.2	ALGORITMO DE AUTO-LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS SEM ODOMETRIA	74
5.3	ALGORITMO DE AUTO-LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS COM ODOMETRIA	79
5.4	SUMÁRIO	85
6	RESULTADOS DOS ALGORITMOS NO MATLAB	86
6.1	RESULTADOS DO ALGORITMO DE AUTO-LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS SEM ODOMETRIA EM AMBIENTE SIMULADO	88
6.2		92
6.3	RESULTADOS DO ALGORITMO DE AUTO-LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS SEM ODOMETRIA EM AMBIENTE REAL	92
6.4	RESULTADOS DO ALGORITMO DE AUTO-LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS COM ODOMETRIA EM AMBIENTE SIMULADO	96
6.5		101
6.6	RESULTADOS DO ALGORITMO DE AUTO-LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS COM ODOMETRIA EM AMBIENTE REAL.....	101
6.6.1	<i>Resultados para uma volta completa</i>	102
6.6.2	<i>Resultados para duas voltas completas</i>	106
6.7	SUMÁRIO	111
7	IMPLEMENTAÇÃO DO SOFTWARE EMBARCADO	113
7.1	IMPLEMENTAÇÃO NA CÂMERA DVT	113
7.2	IMPLEMENTAÇÃO EM HARDWARE DEDICADO.....	113
7.3	SISTEMA OPERACIONAL	114
7.4	PROGRAMAÇÃO DO ALGORITMO DE AUTO-LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS	115
7.5	SUMÁRIO	117
8	RESULTADOS DO ALGORITMO EM C++	119
8.1	RESULTADOS DO ALGORITMO DE AUTO-LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS COM ODOMETRIA EM AMBIENTE SIMULADO	119

8.2	RESULTADOS DO ALGORITMO DE AUTO-LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS COM ODOMETRIA EM AMBIENTE REAL.....	123
8.2.1	<i>Resultados para uma volta completa.....</i>	124
8.2.2	<i>Resultados para duas voltas completas.....</i>	128
8.3	ANÁLISE DO TEMPO DE PROCESSAMENTO DO ALGORITMO DE AUTO-LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS (SLAM) COM ODOMETRIA EM C++ NO AMBIENTE REAL	134
8.4	SUMÁRIO	137
9	CONCLUSÕES	139
9.1	CONSIDERAÇÕES FINAIS.....	140
9.2	TRABALHOS FUTUROS.....	140
10	BIBLIOGRAFIA.....	142
	ANEXO A – MANUAL DE UTILIZAÇÃO DO SOFTWARE.....	147
A1	ARQUIVOS DE ENTRADA DE DADOS	147
A1.1	<i>Arquivo de configuração</i>	148
A1.2	<i>Arquivo de odometria.....</i>	159
A1.3	<i>Arquivos das distâncias e alturas.....</i>	160
A2	ARQUIVOS DE SAÍDA DE DADOS	160
A2.1	<i>Arquivo do tempo de execução.....</i>	161
A2.2	<i>Arquivo dos mapas globais</i>	161
A2.3	<i>Arquivo de odometria.....</i>	162
A2.4	<i>Arquivo do mapa estático do TrICP.....</i>	162
A2.5	<i>Arquivo do mapa móvel antes do TrICP.....</i>	162
A2.6	<i>Arquivo do mapa móvel depois do TrICP.....</i>	162
A2.7	<i>Arquivo de vetores das iterações.....</i>	163
A2.8	<i>Arquivo de auto-localização.....</i>	163
A2.9	<i>Arquivo de erros.....</i>	163
A2.10	<i>Arquivo de configurações.....</i>	164
	ANEXO B – CONTEÚDO DO CD	165

LISTA DE FIGURAS

Figura 1.1 – Sistema de visão estéreo omnidirecional baseada em um par de espelhos coaxiais radialmente simétricos.	22
Figura 2.1 – Secção transversal do sistema de visão estéreo omnidirecional mostrando os raios de luz incidentes e refletidos nos espelhos emitidos por um ponto P no espaço.	36
Figura 2.2 – Ângulos de elevação máximo e mínimo do sistema de visão omnidirecional estéreo com espelho duplo de seções com perfis hiperbólicos.	37
Figura 2.3 – Mudança no perfil externo do espelho para algumas variações do ângulo de rotação do eixo da hipérbole da seção externa do espelho para o mesmo campo de visão.	38
Figura 2.4 – Imagem do sistema de visão estéreo omnidirecional.	43
Figura 3.1 – Vista superior do ambiente simulado.	44
Figura 3.2 – Trajetória do robô pelo ambiente simulado com as posições de aquisição das imagens mostradas em vermelho.	46
Figura 3.3 – Exemplo de uma imagem adquirida pelo sistema de visão estéreo omnidirecional no ambiente simulado.	48
Figura 3.4 – Vista superior da sala do ambiente real (dimensões em cm).	49
Figura 3.5 – Fotografia do ambiente real.	50
Figura 3.6 – Trajetória simulada no ambiente real. Pontos em vermelho indicam as posições de aquisição das imagens.	50
Figura 3.7 – Visualização da imagem adquirida pelo sistema de visão omnidirecional no 5° ponto da trajetória simulada.	52
Figura 3.8 – Visualização da imagem adquirida pelo sistema de visão omnidirecional no 20° ponto da trajetória simulada.	52
Figura 3.9 – Visualização da imagem adquirida pelo sistema de visão omnidirecional no 34° ponto da trajetória simulada.	53
Figura 3.10 – Visualização da imagem adquirida pelo sistema de visão omnidirecional no 47° ponto da trajetória simulada.	53
Figura 4.1 – Geometria epipolar. (a) As linhas epipolares são linhas radiais na imagem omnidirecional. (b) Quando a imagem é transformada para coordenadas retangulares as linhas epipolares se tornam linhas verticais.	55

Figura 4.2 – Imagens resultantes após a eliminação das bordas verticais para se obter somente as bordas horizontais.....	58
Figura 4.3 – Reconstrução das imagens utilizando o processo original.	59
Figura 4.4 – Reconstrução das imagens utilizando o processo original e posterior seleção dos pontos mais próximos do sistema para cada incremento angular de 1°.....	59
Figura 4.5 – Reconstrução das imagens utilizando o algoritmo de detecção de somente das bordas horizontais.	60
Figura 4.6 – Imagem panorâmica obtida da imagem omnidirecional adquirida no primeiro ponto da trajetória no ambiente simulado.....	61
Figura 4.7 – Resultado do algoritmo de Canny (1986) para a imagem da figura 4.8.....	62
Figura 4.8 – Histograma da soma de pixels das retas verticais por incremento angular de 1° (somente a maior reta em cada incremento angular de 1° é considerada) presentes nas imagens da figura 4.7.....	62
Figura 4.9 – Bordas verticais importantes detectadas nas imagens da figura 4.7.....	63
Figura 4.10 – Quinas verticais detectadas (estrelas vermelhas) nas imagens da figura 4.7.	64
Figura 5.1 – Pseudocódigo do algoritmo de construção de mapa global do ambiente.....	77
Figura 5.2 – Diagrama lógico do algoritmo de construção de mapa global do ambiente.....	78
Figura 5.3 – Pseudocódigo do algoritmo de construção de mapa global do ambiente com odometria.	83
Figura 5.4 – Diagrama lógico do algoritmo de construção de mapa de ambiente com odometria.	84
Figura 6.1 – Valor das arestas em centímetros e a numeração adotada para as mesmas no ambiente simulado.	87
Figura 6.2 – Valor das arestas em centímetros e a numeração adotada para as mesmas no ambiente real.	87
Figura 6.3 – Mapa do ambiente simulado reconstruído a partir das 37 imagens seqüenciadas na ordem direta obtido com o algoritmo sem odometria. Os números em vermelho indicam o erro de reconstrução nos comprimentos das arestas em mm.	88
Figura 6.4 – Evolução do erro quadrático relativo percentual obtido com o algoritmo sem odometria no ambiente simulado.	89
Figura 6.5 – Trajetórias real e calculada pelo algoritmo sem odometria no ambiente simulado.	90
Figura 6.6 – Desvio percentual do erro de localização para o algoritmo sem odometria no ambiente simulado.	91

Figura 6.7 – Erro absoluto na localização do robô para o algoritmo sem odometria no ambiente simulado.	91
Figura 6.8 – Número de iterações na construção de mapas para o algoritmo sem odometria no ambiente simulado.	92
Figura 6.9 – Mapa do ambiente reconstruído a partir das 60 imagens com dois ciclos seqüenciados na ordem direta obtido com o algoritmo sem odometria.	93
Figura 6.10 – Evolução do erro quadrático relativo percentual obtido com o algoritmo sem odometria no ambiente real.	94
Figura 6.11 – Trajetórias real e calculada pelo algoritmo sem odometria no ambiente real.	95
Figura 6.12 – Número de iterações necessárias para incorporar cada imagem na construção do mapa do ambiente real usando o algoritmo sem odometria.	96
Figura 6.13 – Mapa do ambiente reconstruído a partir das 37 imagens com o algoritmo com odometria. Os números em vermelho indicam o erro de reconstrução nos comprimentos das arestas em mm.	97
Figura 6.14 – Evolução do erro quadrático relativo percentual obtido com o algoritmo com odometria no ambiente simulado.	98
Figura 6.15 – Trajetórias simulada, calculada pelo algoritmo com odometria e calculada pela odometria no ambiente simulado.	99
Figura 6.16 – Desvio percentual do erro de localização para o algoritmo com odometria no ambiente simulado.	100
Figura 6.17 – Erro absoluto na localização do robô para o algoritmo com odometria no ambiente simulado.	100
Figura 6.18 – Número de iterações necessárias para incorporar as informações das imagens na construção do mapa usando o algoritmo com odometria no ambiente simulado.	101
Figura 6.19 – Mapa do ambiente reconstruído a partir das 60 imagens obtido com o algoritmo com odometria. Os números em vermelho indicam o erro de reconstrução nos comprimentos das arestas em mm.	102
Figura 6.20 – Evolução do erro quadrático relativo percentual obtido com o algoritmo com odometria no ambiente real.	103
Figura 6.21 – Trajetórias simulada, calculada pelo algoritmo com odometria e calculada pela odometria no ambiente real.	104
Figura 6.22 – Desvio percentual do erro de localização para o algoritmo com odometria no ambiente real.	105

Figura 6.23 – Erro absoluto na localização do robô para o algoritmo com odometria no ambiente real.....	105
Figura 6.24 – Número de iterações na construção do mapa para o algoritmo com odometria no ambiente real.....	106
Figura 6.25 – Mapa do ambiente reconstruído com o algoritmo com odometria com duas voltas do robô no ambiente com as imagens seqüenciadas na ordem direta.	107
Figura 6.26 – Evolução do erro quadrático relativo percentual obtido com o algoritmo com odometria no ambiente real com duas voltas do robô.	108
Figura 6.27 – Trajetórias simulada, calculada pelo algoritmo com odometria e calculada pela odometria no ambiente real com duas voltas do robô.	109
Figura 6.28 – Desvio percentual do erro de localização para o algoritmo com odometria no ambiente real com duas voltas do robô.....	110
Figura 6.29 – Erro absoluto na localização do robô para o algoritmo com odometria no ambiente real com duas voltas do robô.....	110
Figura 6.30 – Número de iterações na construção de mapas para o algoritmo com odometria no ambiente real com duas voltas do robô.....	111
Figura 8.1 – Mapa do ambiente reconstruído a partir das 37 imagens seqüenciadas na ordem direta obtido com o algoritmo com odometria em C++. Os números em vermelho correspondem aos erros de reconstrução das respectivas arestas.	119
Figura 8.2 – Evolução do erro quadrático relativo percentual obtido com o algoritmo com odometria em C++ no ambiente simulado.	120
Figura 8.3 – Trajetórias simulada, calculada pelo algoritmo com odometria em C++ e calculada pela odometria no ambiente simulado.	121
Figura 8.4 – Desvio percentual do erro de localização para o algoritmo em C++ com odometria no ambiente simulado.	122
Figura 8.5 – Erro absoluto na localização do robô para o algoritmo com odometria em C++ no ambiente simulado.	122
Figura 8.6 – Número de iterações necessárias para incorporar as informações de cada imagem na construção do mapa utilizando o algoritmo com odometria em C++ no ambiente simulado.....	123
Figura 8.7 – Mapa do ambiente reconstruído a partir das 60 imagens seqüenciadas na ordem direta obtido com o algoritmo com odometria em C++. Valores em vermelho consistem nos erros de reconstrução das arestas.	124

Figura 8.8 – Evolução do erro quadrático relativo percentual obtido com o algoritmo com odometria em C++ no ambiente real.....	125
Figura 8.9 – Trajetórias simulada, calculada pelo algoritmo com odometria em C++ e calculada pela odometria no ambiente real.	126
Figura 8.10 – Desvio percentual do erro de localização para o algoritmo com odometria em C++ no ambiente real.....	127
Figura 8.11 – Erro absoluto na localização do robô para o algoritmo com odometria em C++ no ambiente real.....	127
Figura 8.12 – Número de iterações na construção de mapas para o algoritmo com odometria em C++ no ambiente real.....	128
Figura 8.13 – Mapa do ambiente reconstruído a partir das 60 imagens com duas voltas no ambiente seqüenciadas na ordem direta obtido utilizando o algoritmo com odometria em C++.	129
Figura 8.14 – Evolução do erro quadrático relativo percentual obtido com o algoritmo com odometria em C++ no ambiente real com duas voltas no ambiente.	130
Figura 8.15 – Trajetórias simulada, calculada pelo algoritmo com odometria em C++ e calculada pela odometria no ambiente real com duas voltas no ambiente.	131
Figura 8.16 – Desvio percentual do erro de localização para o algoritmo com odometria em C++ no ambiente real com dois ciclos.....	132
Figura 8.17 – Erro absoluto na localização do robô para o algoritmo com odometria em C++ no ambiente real com dois ciclos.	133
Figura 8.18 – Número de iterações na construção de mapas para o algoritmo com odometria no ambiente real com dois ciclos.....	133
Figura 8.20 – Análise do tempo de processamento da imagem omnidirecional no algoritmo com odometria em C++ no ambiente real.....	136
Figura 8.21 – Tempo de execução para cada etapa do algoritmo de auto-localização e mapeamento com odometria em C++. Valores em segundos.....	137

LISTA DE TABELAS

Tabela 1.1 – Outros parâmetros do espelho.....	42
Tabela 3.1 – Coordenadas das posições de aquisição das 10 imagens da trajetória do robô adquiridas com espaçamento de 1m.	46
Tabela 3.2 – Coordenadas das posições de aquisição das 37 imagens da trajetória do robô adquiridas com espaçamento de 0,25m.	47
Tabela 3.3 – Coordenadas das posições de aquisição das imagens da trajetória do robô no ambiente real.	51

1 INTRODUÇÃO

Para a navegação de robôs móveis é necessário extrair informação do ambiente de maneira a fornecer subsídios para o robô se locomover no ambiente de forma segura e atingir seus objetivos. A aquisição de informação é feita através de sensores. A escolha do sensor mais adequado depende do tipo de tarefa de navegação que se pretende realizar e da autonomia desejada.

Inúmeros sensores podem ser utilizados, dentre esses, citam-se: ultra-som, infravermelho, sistema de posicionamento global, lasers e visão computacional. Sensores de ultra-som e lasers são sensíveis a superfícies refletoras, tornando-se úteis para medirem distâncias. Sensor de infravermelho possui boa resolução angular, tornando-se útil para detectar a presença ou não de objetos. Os sistemas de posicionamento global (GPS) podem ser utilizados para determinar a localização do robô. Sensores baseados em câmeras são capazes de capturar imagens de forma contínua e fornecer a posição de objetos.

Em certas aplicações de visão computacional é interessante ter um campo visual mais abrangente do que o obtido com uma câmera convencional. Imagens panorâmicas contêm informação de todo o ambiente em uma única imagem. Um sistema de visão panorâmica ou omnidirecional fornece uma imagem de 360° do ambiente. Devido a essa característica, visão panorâmica tem se tornado uma ferramenta importante, especialmente na área de robôs móveis.

A visão omnidirecional (do latim “*omnis*”, em português “oni”, significa tudo ou todo) tem sido muito utilizada em inúmeras aplicações, dentre elas citam-se: vigilância remota, reconstrução tridimensional de ambientes, rastreamento visual e navegação de robôs móveis.

Robôs móveis são programados para realizarem tarefas de maneira autônoma. A autonomia de um robô é fortemente condicionada pela sua capacidade de perceber o ambiente de navegação. Um robô deve ser capaz de interagir coerentemente com o seu mundo, sendo capaz de recuperar descrições úteis, usando informação adquirida pelos sensores e utilizando eficientemente essas descrições com o objetivo de realizar uma tarefa específica. A tarefa mais básica que um robô móvel deve realizar autonomamente é a navegação sem colisão com obstáculos.

Outra tarefa que proporciona uma maior autonomia a um robô móvel é a construção de mapas de ambiente. Um mapa é uma representação da estrutura do ambiente que pode ser utilizado para diversas tarefas, inclusive navegação. Nas tarefas de exploração de ambientes o robô navega adquirindo informações para a construção de um mapa. O robô pode ainda, baseando-se em um mapa existente, navegar resolvendo os problemas de localização, planejamento de trajetória e verificar se a posição final foi atingida.

A utilização dos mapas de ambiente remete diretamente ao tipo de estrutura de controle do robô móvel. Segundo Mataric (2002) o controle do robô é o processo de se obter informações a respeito do ambiente, através dos sensores do robô, processá-los como necessário de modo a tomar decisões a respeito de como agir, e então executar estas ações no ambiente. As quatro classes de metodologia de controle de robô são compostas pelo controle reativo, deliberativo, híbrido e baseado em comportamento.

O controle reativo é baseado no forte acoplamento entre as saídas dos sensores e as entradas nos atuadores, permitindo que o robô responda muito rapidamente a qualquer mudança no ambiente. Suas desvantagens estão relacionadas com o fato de que para obter resposta tão rápida é necessário que não haja processamento na utilização de modelos internos, memória e aprendizagem.

O controle deliberativo utiliza toda a informação sensorial disponível e todo o conhecimento armazenado internamente para raciocinar a respeito de quais ações devem ser tomadas em seqüência. O planejamento requer que exista uma representação interna do ambiente, como o mapa do ambiente global, que permita ao robô prever os resultados de possíveis ações em vários estados. A desvantagem desta metodologia está relacionada com o alto tempo necessário para a tomada das decisões.

O sistema de controle híbrido contém ambos componentes reativos e deliberativos que devem interagir de forma a produzir saídas coerentes. Esta arquitetura combina os melhores aspectos do controle reativo e deliberativo, e também deve utilizar um modelo interno de representação do ambiente.

O controle baseado em comportamento é composto por componentes definidos por padrões observáveis de atividade emergente da interação do robô com o ambiente. Os sistemas de controle baseados em comportamento e os reativos compartilham algumas propriedades similares, entretanto, sistemas baseados em comportamentos são

fundamentalmente mais poderosos devido a sua capacidade de armazenar representações, enquanto o controle reativo não pode fazê-lo.

1.1 Estado da arte

A seguir os principais temas abordados neste trabalho são sucintamente descritos, iniciando com um apanhado da literatura na área dos sistemas de visão estéreo omnidirecional, seguido de uma descrição dos sistemas de localização mais comumente utilizados para obtenção da localização de robôs móveis, os principais métodos utilizados na construção de mapas de ambiente e uma visão geral do problema de auto-localização e mapeamento simultâneos.

1.1.1 Sistema de visão estéreo omnidirecional

Existem inúmeras formas de aquisição de imagens omnidirecionais. Câmeras giratórias, composição de um conjunto de câmeras dispostas em diferentes posições do ambiente; lentes especiais como “olho de peixe” e a conciliação de meios refletores e refratores (espelhos e câmeras) são alguns exemplos.

No caso de câmeras giratórias, a câmera gira com velocidade de rotação constante em torno do seu eixo vertical. A resolução horizontal da imagem panorâmica não depende da resolução angular da câmera e sim da resolução angular da rotação. A desvantagem da utilização de câmeras giratórias está relacionada ao longo tempo de aquisição das imagens, tornando-as inaplicáveis para problemas de tempo real e para ambientes dinâmicos, Yagi (1999).

A composição de câmeras no ambiente pode ser feita de tal maneira a obter uma imagem panorâmica. No entanto, é difícil obter um sistema compacto em razão da existência de inúmeras câmeras.

Lentes especiais, tais como “olho de peixe”, fornecem a visão de um hemisfério, no entanto a imagem obtida tem boa resolução no centro e baixa resolução na periferia. Assim, as projeções dos objetos presentes no mundo se tornam distorcidas pela projeção na parte periférica da imagem.

Outra maneira de se obter visão omnidirecional é com a utilização de meios catadióptricos. Conciliando uma câmera e um espelho convexo, sendo este alinhado com o eixo óptico da câmera, obtém-se um sensor potencialmente útil, por exemplo, para aplicação em tarefas de navegação de robôs móveis. Dentre os vários tipos de sistemas de visão omnidirecional este é o mais atrativo devido à sua simplicidade e compacticidade. Inúmeros pesquisadores têm investigado e utilizado esse tipo de sistema nos últimos anos para variadas aplicações. Entre esses inúmeros autores podem-se citar alguns, como por exemplo, Baker e Nayar (1998), Chahl e Srinivasan (1997), Conroy e Moore (1999), Gaspar et al (2002), Geyer e Daniilidis (2000), Hicks e Bajcsy (1999a), Ollis et al. (1999), Svoboda et al. (1998) e Yamazawa et al (1999).

Outra forma de visão omnidirecional pode utilizar espelhos convexos e côncavos para obter uma imagem omnidirecional estéreo frontal. Um exemplo desse tipo de sistema é apresentada por Stürzl e Srinivasan (2005), que utilizam três espelhos, sendo dois deles convexos e um côncavo para obter não somente visão omnidirecional, como também visão estéreo para uma faixa reduzida da imagem obtida.

Além da consideração da estrutura utilizada para obter um sistema omnidirecional, outra propriedade que qualifica esse tipo de sistema é a existência ou não de um centro único de projeção (Swaminathan et al., (2006)). O centro único de projeção é uma característica que dependendo da aplicação pode ser importante em um sistema de visão omnidirecional. O centro único de projeção garante a unicidade entre um ponto do espaço tridimensional e a sua projeção na imagem omnidirecional.

O formato do espelho determina a formação da imagem de um sistema de visão catadióptrico. Muitas formas diferentes de espelhos têm sido utilizadas e testadas. Os primeiros espelhos a serem utilizados foram os espelhos de formatos simples, tais como, planos, cônicos, hiperbólicos, parabólicos, elípticos e esféricos, devido à sua simplicidade geométrica e de fabricação.

Diversos autores apresentam estudos detalhados da utilização de espelhos em formato de hipérboles, parábolas, elipses e esferas em sistemas de visão omnidirecional, entre esses pode-se citar Svoboda et al. (1998) e Yamazawa et al. (1999). Os espelhos em formato de cone têm boa resolução na parte periférica, enquanto os espelhos esféricos têm boa resolução na parte central. O espelho hiperbólico tem a vantagem de boa resolução em ambas as partes central e periférica, sendo que a abrangência do seu campo visual é definida no projeto do perfil do espelho. O espelho parabólico tem resolução intermediária entre a dos espelhos

esférico e hiperbólico, porém contém o maior campo visual obtido dentre esses tipos de espelhos convexos.

Uma das grandes vantagens de um sistema catadióptrico é que o espelho pode ser projetado com um formato especial de modo a preservar certas propriedades do espaço tridimensional na imagem. A escolha das propriedades que devem ser preservadas na formação da imagem está naturalmente relacionada com a aplicação desejada. Na literatura são apresentadas inúmeras possibilidades para a propriedade a ser preservada, podendo-se citar o trabalho de Gaspar et al. (2002).

Outra ferramenta muito importante em visão computacional é a visão estéreo. Com visão estéreo é possível determinar a distância de objetos dentro do campo visual do sistema de visão. Com a medida de distância dos objetos ao sistema pode-se pensar em realizar diversas tarefas, como por exemplo: planejamento de trajetória para navegação de robôs móveis; localização de objetos no espaço para manipulação por robôs; reconstrução tridimensional de ambientes para interação virtual; metrologia dimensional de peças; inspeção visual de equipamentos; etc.

A visão estéreo baseia-se no princípio de comparação das imagens de uma mesma cena obtidas, normalmente, por duas câmeras (ou uma única com movimento) posicionadas em locais diferentes, mas próximos entre si. O problema fundamental em visão estéreo é determinar onde pixels, pontos ou outras características correspondentes estão presentes nas duas imagens; esse problema é chamado de correspondência. A procura por pontos correspondentes de forma eficiente, em geral, é realizada ao longo da chamada curva epipolar. A procura na linha epipolar restringe o problema de busca bidimensional para um problema de busca unidimensional. A forma da curva epipolar depende da geometria do sistema de visão.

Uma vez obtidos os dois pontos correspondentes nas duas imagens, pode-se calcular a distância do ponto físico no espaço ao sistema de visão através da técnica de triangulação, ou seja, dados os pontos correspondentes nas duas imagens e a localização relativa dos dois sistemas de visão, a partir de considerações geométricas calcula-se a posição do ponto no espaço.

Outra forma de obter-se visão estéreo é mostrada nos trabalhos de Orghidan et al. (2005 e 2006) que utilizam um conjunto câmera-espelho para obter a imagem omnidirecional e um conjunto laser-espelho para projetar um padrão conhecido e calibrado no ambiente que

se deseja obter a visão estéreo. O sistema de visão deve reconhecer o padrão impresso pelo laser para que seja possível obter os dados das distâncias dos objetos ao sistema. Uma desvantagem deste sistema é que ele depende da potência do laser utilizado para imprimir o padrão no ambiente.

Devido ao ângulo de visão de 360° sistemas de visão estéreo omnidirecional são ideais para aplicações de navegação de robôs e reconstrução de ambientes. Nessas aplicações de visão estéreo múltiplas câmeras ou câmeras móveis são freqüentemente usadas para compensar o pequeno campo de visão de sistemas de visão estéreo convencionais. Dois espelhos substituem diversas câmeras na visão estéreo omnidirecional.

Nene e Nayar (1998) apresentam diversas configurações de sistemas estéreos omnidirecionais que utilizam espelhos planos, parabólicos, elípticos e hiperbólicos, todos assegurando centro único de projeção.

Visão estéreo omnidirecional baseada em um par de espelhos axialmente coaxiais (ou espelho duplo com duas seções coaxiais) com campos de visão que se intersectam, como mostra a figura 1.1, foi sugerido inicialmente por Southwell et al. (1996). Nesse sistema os centros dos dois espelhos são colineares com o eixo da câmera e os espelhos têm um formato radialmente simétrico em torno do seu eixo.

Figura 1.1 – Sistema de visão estéreo omnidirecional baseada em um par de espelhos coaxiais radialmente simétricos.

Como visto, existem inúmeras formas de se implementar um sistema de visão estéreo omnidirecional. Ollis et al. (1999) e Mouaddib et al. (2005) apresentam um estudo sobre configurações de sistemas que proporcionam visão estéreo omnidirecional. As diferentes configurações utilizam um ou mais espelhos e também uma ou mais câmeras de forma a obter um campo abrangente. Uma análise do campo visual, da resolução da imagem e da acuracidade na medida da distância obtidas em cada configuração é realizada nesses estudos.

Cabral et al. (2004) desenvolveram um sistema de visão omnidirecional estéreo baseado em um par de espelhos coaxiais com perfil hiperbólico capaz de realizar reconstrução tridimensional. Esse sistema é utilizado nesse trabalho para obter as imagens do ambiente.

1.1.2 Sistemas para localização de robôs móveis

Conhecer a posição do robô é um problema fundamental em aplicações de robôs móveis. Em busca de uma solução, pesquisadores e engenheiros têm desenvolvido uma variedade de sistemas, sensores e técnicas para o cálculo da posição de robôs móveis. Talvez o resultado mais importante ao se examinar a literatura de posicionamento de robôs móveis é que hoje, ainda não existe uma solução realmente elegante para o problema. Segundo Borenstein et al. (1996) e Borenstein et al. (1997) as muitas soluções parciais podem ser aproximadamente categorizadas em dois grupos, sendo estes: medidas de posição relativa e posição absoluta. Os métodos de medidas de posição relativa, também conhecido como *Dead-reckoning*, utilizam odometria e navegação inercial, enquanto os métodos de medidas de posição absoluta, também conhecido como sistemas baseados na referência, utilizam bússula, marcos ativos, sistema de posicionamento global, navegação por marcos de ambiente e correspondência com modelo. A seguir descrevem-se brevemente os métodos citados.

Odometria é largamente utilizada como método de posicionamento para robôs móveis. Ela apresenta boa precisão, é barata e permite altas taxas de amostragem. Entretanto, a idéia fundamental da odometria é a integração da informação do movimento incremental ao longo do tempo, o que leva inevitavelmente ao ilimitado acúmulo de erros. Especificamente, erros de orientação causarão grande erro de posição lateral, que aumenta proporcionalmente com a distância de viagem do robô. Apesar destas limitações, a maior parte dos pesquisadores concorda que a odometria é uma parte importante do sistema de navegação do robô e que as tarefas de navegação serão simplificadas se a precisão da odometria puder ser melhorada.

Inúmeros pesquisadores propõem métodos para unir os dados da odometria com medidas de posição absoluta para obter estimativas da posição mais confiáveis, como por exemplo, Byrne et al. (1992) e Chenavier e Crowley (1992).

Navegação inercial utiliza giroscópios e acelerômetros para medir as taxas de rotação e aceleração respectivamente. O sistema de navegação inercial tem a vantagem de ser auto-contido, ou seja, ele não precisa de referências externas. Entretanto, os dados do sensor inercial acumulam erros com o tempo devido a necessidade de integração dos dados. Qualquer erro constante, mesmo pequeno, cresce sem limites com a integração. Sensores inerciais não são adequados para posicionamento preciso por um longo período de tempo.

A bússola, ou compasso magnético, é um sensor que fornece uma medida de direção absoluta, sendo esta medida extremamente importante para resolver as necessidades de navegação de robôs móveis. A orientação do robô é o parâmetro mais significativo para navegação, pois tem grande influência nos erros acumulados de medidas de posição relativa. Uma desvantagem de qualquer bússola é que o campo magnético da Terra é normalmente distorcido perto de linhas de transmissão de eletricidade ou estruturas metálicas. Esta desvantagem faz com que o uso desse tipo de sensores seja de difícil aplicação em ambientes internos.

Os sistemas de navegação por marcos ativos são os mais comuns e os mais utilizados para navegação de navios e aviões, bem como nos sistemas de robôs móveis comerciais. Marcos ativos podem ser detectados no ambiente com confiabilidade e fornecem informação de posicionamento preciso com um mínimo de processamento. A montagem precisa do marco é necessária para o posicionamento preciso. Dois tipos diferentes de marcos ativos podem ser citados: trilaterização e triangulação. Na trilaterização a determinação da posição do robô é baseada na medida das distâncias aos marcos ativos conhecidos. No sistema de navegação por trilaterização existem normalmente três ou mais transmissores montados em localizações conhecidas do ambiente e um receptor a bordo do robô. Na triangulação existem três ou mais transmissores ativos montados em localizações conhecidas. Um sensor de rotação no robô registra os ângulos nos quais ele “enxerga” os marcos transmissores com relação aos eixos longitudinais do robô. Por meio dessas medidas as coordenadas x e y e a orientação do robô podem ser calculadas.

O sistema de posicionamento global (GPS) é uma tecnologia para navegação em ambientes externos. O GPS foi desenvolvido pelo Departamento de Defesa Americano, sendo composto por um conjunto de satélites que transmitem um sinal de rádio-frequência

codificado. Usando-se métodos de trilaterização avançados, receptores no solo podem medir as suas posições baseado no tempo de viagem dos sinais de rádio-freqüência dos satélites, permitindo o cálculo da latitude, longitude e altitude do receptor. O governo Norte Americano intencionalmente adiciona pequenos erros nos tempos e posições dos satélites para prevenir que uma nação hostil utilize o GPS para auxiliar no lançamento de mísseis de precisão. Esse erro é chamado de disponibilidade seletiva e pode ser eliminado através do uso de um GPS diferencial. Atualmente, O Departamento de Defesa dos Estados Unidos fornece dois serviços, o *Standard Positioning Service*, gratuito e com o erro de disponibilidade seletiva e o *Precise Positioning Service* codificado e sem o erro de disponibilidade seletiva. Os problemas associados ao uso do GPS para navegação de robôs móveis são o bloqueio periódico do sinal devido a folhagem e terreno montanhoso, interferências e acuracidade insuficiente para sistemas de navegação primários.

Marcos do ambiente são características distintas que o robô pode reconhecer por meio dos seus sensores. Marcos do ambiente podem ser formas geométricas e podem incluir informações adicionais, na forma de códigos de barra, por exemplo. Em geral, marcos do ambiente têm uma posição fixa e conhecida, relativa àquela com a qual o robô pode se localizar. Os marcos são cuidadosamente escolhidos para serem facilmente identificados. Antes de o robô utilizar os marcos do ambiente para navegação, as características dos marcos devem ser conhecidas e armazenadas na memória do robô. A principal tarefa na localização é então reconhecer os marcos do ambiente de maneira confiável e calcular a posição do robô. Os marcos do ambiente podem ser divididos em dois grupos: marcos naturais e marcos artificiais.

Os marcos naturais do ambiente funcionam melhor em ambientes altamente estruturados, ou seja, feitos pelo homem. Dessa forma definem-se os marcos naturais do ambiente como sendo objetos ou características que já estão no ambiente e tem uma função que não é somente a navegação do robô. Os marcos artificiais do ambiente são aqueles objetos ou marcas especialmente designados que são colocados no ambiente com o único propósito de permitir a navegação do robô.

O principal problema de se usar marcos naturais do ambiente na navegação é detectar e combinar as características dos marcos a partir das informação dos sensores. A escolha natural do sensor para esta tarefa é a visão computacional. A maior parte dos marcos naturais do ambiente baseados em visão são longas bordas verticais, como portas, junções de paredes e luzes no teto.

A detecção é muito mais fácil com marcos artificiais, que são projetados para apresentar um contraste ótimo no ambiente. Além disso, o formato e tamanho exatos do marco artificial do ambiente são conhecidos previamente. O tamanho e a forma podem propiciar uma riqueza de informação geométrica quando transformado sobre projeção perspectiva. Muitos sistemas de posicionamento com marcos artificiais do ambiente são baseados em visão computacional.

Podem-se resumir as características da navegação baseada em marcos do ambiente como se segue. Marcos naturais do ambiente oferecem flexibilidade e não requerem modificação no ambiente. Marcos artificiais do ambiente são baratos e podem ter informação adicional codificada como parâmetros ou formas. A máxima distancia efetiva entre o robô e um marco natural do ambiente é substancialmente menor que no sistema de marcos ativos. A exatidão no posicionamento depende da distância e do ângulo entre o robô e o marco do ambiente. É necessário mais processamento quando comparado com o sistema de marcos ativos, sendo que em muitos casos os computadores de bordo do robô não são capazes de processar os marcos naturais do ambiente com rapidez necessária para aplicações em tempo real. As condições ambientais, como iluminação, podem representar um problema, sendo que com visibilidade marginal os marcos do ambiente podem não ser detectados ou outros objetos no ambiente com características similares podem ser confundidos com um marco legítimo do ambiente. Os marcos do ambiente devem estar disponíveis no ambiente de trabalho ao redor do robô. Navegação por marcos do ambiente requer uma posição aproximada na localização inicial do robô para que o mesmo saiba onde procurar os marcos do ambiente. Um banco de dados dos marcos do ambiente e as suas localizações no ambiente devem ser mantidos na memória do robô.

Posicionamento baseado em mapa é a técnica na qual o robô utiliza os seus sensores para criar um mapa do seu ambiente local. Este mapa é então comparado com um mapa global previamente guardado na memória. Se a correspondência for encontrada, então o robô pode calcular a sua posição atual e a sua orientação no ambiente. O mapa pré-armazenado pode ser um modelo do ambiente em CAD, ou pode ser construído através de dados prévios do sensor. A vantagem do posicionamento baseado em mapa é que ele utiliza a ocorrência natural da estrutura típica dos ambientes internos para obter sua auto-localização. Posicionamento baseado em mapa também permite que o robô aprenda um novo ambiente e melhore a precisão do posicionamento através da exploração do ambiente. Desvantagens do posicionamento baseado em mapa são as exigências para a exatidão dos sensores e a

necessidade de que existam características estacionárias e facilmente distinguíveis que possam ser usadas para comparação. A construção dos mapas de ambiente e o problema de correspondência são apresentados com maiores detalhes no próximo item.

1.1.3 Construção de mapas de ambiente

Robôs móveis navegam em ambientes e realizam tarefas com certo grau de autonomia. Uma tarefa elementar de navegação consiste no robô partir de uma posição inicial e atingir uma posição final. Outras tarefas que o robô pode realizar para facilitar a navegação consistem no planejamento da trajetória, auto-localização, rastreamento visual de marcos do ambiente, detecção e desvios de obstáculos e construção de mapas de ambiente.

Para a construção de mapas de ambiente, pode-se classificar o ambiente segundo a sua estrutura, podendo ser estruturado ou não estruturado. Desouza e Kak (2002) propõem que um ambiente estruturado contenha alguma regularidade na informação, que possa ser extraída para fornecer subsídios para a navegação, como retas e planos. Um ambiente não estruturado não contém nenhuma regularidade na informação, não existem restrições geométricas que possam ser parametrizadas ou utilizadas para navegação.

Outra classificação no tipo de ambiente é se ele é interno ou externo, podendo ser estruturado ou não. Giachetti et al. (1998), como exemplo de navegação em ambientes externos estruturados, estudaram a navegação de carros autônomos que trafegam em estradas. Como exemplos de navegação em ambientes externos não-estruturados têm-se os trabalhos de navegação em planetas, como Marte, por Goldberg et al. (2002). Ambientes internos como salas, corredores e laboratórios, podem ser considerados como estruturados, pois em sua maioria contêm retas e planos bem definidos, assim sendo a navegação pode ser auxiliada por um mapa que descreve a estrutura ou a posição dos objetos no ambiente.

Tipicamente, existem duas abordagens para a representação de mapas de ambiente. São os mapas geométricos ou topológicos. Mapas geométricos contêm informação da geometria do ambiente, da posição dos objetos e distâncias entre esses. Como exemplos, citam-se as grades de ocupação, conexão de polígonos e gráficos de Voronoi. Os mapas topológicos não possuem qualquer informação sobre a geometria do ambiente, eles são representados por elos conectados a nós. Nos nós o robô pode tomar certas ações, como por exemplo, virar à direita ou à esquerda, e reconhecer marcos para se localizar no ambiente. O

reconhecimento dos marcos visuais dá ao robô uma localização qualitativa no ambiente, obtendo sua posição em termos do quanto está mais próximo ou mais distante do alvo. Na navegação com mapas geométricos a localização é quantitativa, sabendo o robô a sua posição exata no ambiente.

Segundo Manassis (2003) a construção de mapas para navegação consiste de três etapas:

- 1- Alinhamento: processo que visa obter a transformação de duas medições distintas em um único sistema de coordenadas através da informação que relaciona as duas medições;
- 2- Correspondência: processo realizado após o alinhamento dos pontos, que consiste em corresponder os pontos das duas medições;
- 3- Integração: processo que incorpora os pontos correspondidos em um único sistema de coordenadas para a construção do modelo final.

O processo de alinhamento consiste em determinar qual é a transformação que levou um conjunto de pontos P para um conjunto de pontos Q . O conhecimento dessa transformação alinha os dois conjuntos de dados permitindo a descrição dos pontos em um único sistema de coordenadas. Com as informações descritas no mesmo sistema de coordenadas utiliza-se algum critério para corresponder o conjunto de pontos P ao conjunto de Q .

A maior parte dos métodos de correspondência se baseia no algoritmo *Iterative Closest Point* (ICP), originalmente desenvolvido por Besl e McKay (1992), que utiliza o critério da mínima distância entre os pontos. O ICP calcula iterativamente a transformação rígida e a correspondência entre os pontos, convergindo para a solução da correspondência que minimiza a somatória das distâncias ao quadrado entre os pontos de dois conjuntos de dados.

Uma versão robusta do ICP, chamada de *Trimmed ICP* (TrICP), é descrita nos trabalhos de Chetverikov et al. (2002a) e Chetverikov e Stepanov (2002b). Nesses trabalhos o TrICP utiliza a informação dos mínimos quadrados para cada iteração do ICP. O alinhamento é realizado considerando um fator de sobreposição entre o conjunto de dados. O ICP original é um caso particular do TrICP para quando a taxa de sobreposição dos conjuntos de dados é de 100%.

Um método diferente de correspondência é proposto por Yagi et al. (1995) e Deccó (2004) nos quais a métrica utilizada é o mínimo ângulo entre o robô e os pontos escolhidos do ambiente. O método é proposto para a correspondência de mapas adquiridos com um sistema de visão omnidirecional composto por uma câmera e um espelho hiperbólico. Como o ângulo azimutal do ponto plano da imagem descreve a orientação do ponto no ambiente, pode-se utilizá-lo como medida para encontrar a correspondência entre um par de mapas. Nesse método, para cada novo mapa adquirido, os valores dos ângulos azimutais dos pontos dos mapas são dispostos em ordem crescente e são correspondidos pela mínima diferença angular entre eles.

Outro método de correspondência é proposto por Nakamura e Ishiguro (2002), que projetam um espelho que mapeia pontos do chão linearmente com o aumento dos pixels no plano da imagem e realizam a construção de um mapa plano do ambiente para a navegação. Primeiramente são extraídas retas de um par de imagens. Para cada par é calculado o histograma angular das retas nas direções x e y . Pela máxima correlação cruzada entre os histogramas de um par de imagens encontra-se a transformação rígida que alinha o par.

Resumidamente, para a construção de mapas de ambiente, pode-se classificar o ambiente como Estruturado ou Desestruturado e Externo ou Externo, representá-lo de uma forma Geométrica e/ou Topológica, e dentre as metodologias utilizadas para a construção de mapas utiliza-se normalmente o ICP, porém Proximidade Angular e Histograma Angular também são utilizados.

1.1.4 Problema de auto-localização e mapeamento simultâneos

Segundo Dissanayake et al. (2001) o problema de auto-localização e construção de mapas de ambiente simultâneos (SLAM) consiste em um robô autônomo iniciar a navegação em uma localização desconhecida, em um ambiente desconhecido e então construir um mapa desse ambiente de maneira incremental, enquanto utiliza o mapa simultaneamente para calcular a sua localização. Smith et al. (1990), Leonard e Whyte (1992) e Dissanayake et al. (2001) provam que é possível uma solução para o problema SLAM, por meio de análises fundamentadas na teoria da estimação.

A grande vantagem do SLAM é que elimina a necessidade de estruturas artificiais ou um conhecimento topológico a priori do ambiente. Um grande número de abordagens tem

sido proposto para o problema de SLAM. Normalmente essas técnicas adotam um dos três principais métodos discutidos a seguir.

O primeiro e mais popular deles é baseado na teoria da estimação usando filtro de Kalman. A popularidade desse método é devido a dois fatores principais. Primeiro, ele fornece uma solução recursiva para o problema da navegação e uma maneira de calcular estimativas consistentes para a incerteza na localização do robô e nas posições dos marcos do ambiente, com base em modelos estatísticos para o movimento dos robôs e observações relativas dos marcos do ambiente. Segundo, uma grande quantidade de experiência tem sido desenvolvida na indústria aeroespacial, marinha e outras aplicações de navegação, da qual a comunidade que pesquisa veículos autônomos pode se aproveitar.

O segundo método consiste em evitar a necessidade de estimativas absolutas da posição e de medições precisas das incertezas para utilizar conhecimento mais qualitativo da localização relativa dos marcos do ambiente e do robô para a construção do mapa global e planejamento da trajetória. Esse método geral foi desenvolvido por diferentes grupos e de diferentes formas como se pode ver nos trabalhos de Brooks (1986), Kuipers e Byun (1991) e Levitt e Lawton (1990). A abordagem qualitativa para a navegação e para a solução do problema SLAM possui muitas vantagens potenciais sobre a metodologia da teoria da estimação em termos de limitar a necessidade por modelos precisos e os requisitos computacionais resultantes desta necessidade.

O terceiro método é bastante amplo e afasta-se do rigor matemático do filtro de Kalman, ou do formalismo estatístico, retendo uma abordagem essencialmente numérica ou computacional para a navegação e para o problema de SLAM. Essa abordagem inclui o uso de correspondência com marcos artificiais do ambiente. Os trabalhos de Zang (1994) e de Cozman e Krotkov (1997) utilizam registro de mapa global, enquanto Hanebeck e Schmidt (1996) usam marcos naturais do ambiente e outras medidas para descrever incertezas.

Filliat e Meyer (2003a, 2003b) fazem uma revisão das estratégias de localização, criação de mapas e planejamento da trajetória focando nos problemas de integração da percepção, categorização e controle motor que devem ser resolvidos para que o robô alcance o seu objetivo mais básico, que é o de partir de um ponto inicial e chegar a um ponto final, evitando passar por lugares indesejados, ou bater em obstáculos, ou ainda escapar de um lugar perigoso. Para tanto, fazem uso de uma analogia entre os animais e robôs se inspirando em seus processos cognitivos para desenvolver uma técnica de navegação baseada em mapas do ambiente.

Atualmente grande parte dos sistemas de mapeamento está apta a lidar com ruídos na odometria e nos dados dos sensores, entretanto, eles assumem que o ambiente é estático durante o mapeamento. Se uma pessoa anda dentro do alcance dos sensores do robô durante o mapeamento, o mapa resultante conterá evidências a respeito de um objeto na localização correspondente. Além disso, se o robô retornar para esta localização e varrer a área uma segunda vez, a estimativa da posição será menos acurada, uma vez que as novas medições não contêm nenhum vestígio correspondente àquela pessoa. A exatidão reduzida do mapa resultante pode ter uma influência negativa no desempenho do robô.

Para métodos de construção de mapas de ambiente com robôs móveis em um ambiente dinâmico pode-se citar os trabalhos de Hähnel et al. (2003) e de Wang e Thorpe (2002). Hähnel et al. (2003) utilizam um algoritmo iterativo de Minimização do Valor Esperado, onde no passo da expectativa computa-se uma estimativa probabilística a respeito de quais medições possam corresponder a objetos estáticos e no passo da maximização utilizam-se estas estimativas para determinar a posição do robô. Wang e Thorpe (2002) estudam o problema do SLAM com a detecção e acompanhamento de objetos móveis DTMO (*Detection and Tracking of Moving Objects*), com a finalidade de resolver ambos os problemas simultaneamente, para aplicações em ambientes internos e externos.

1.2 Objetivos

Este trabalho têm por objetivo desenvolver um algoritmo de auto-localização e construção de mapas de ambiente simultâneos para um robô móvel em um ambiente estruturado, ou seja, um ambiente que pode ser descrito através de primitivas geométricas. O mapa do ambiente é construído utilizando-se imagens adquiridas por um sistema de visão omnidirecional estéreo com espelho duplo de perfil hiperbólico.

Com este objetivo os seguintes itens são desenvolvidos:

- Desenvolvimento do algoritmo de auto-localização e construção de mapas de ambiente simultâneos em MATLAB;
- Testes do sistema em ambiente simulado;
- Implementação do algoritmo desenvolvido em MATLAB em C++ de forma a tornar possível a sua utilização embarcada;

- Testes do sistema em ambiente real;
- Resultados e conclusões

1.3 Justificativas

Os mapas de ambiente são construídos para serem utilizados em tarefas de navegação, fornecendo ao robô subsídios para o reconhecimento do ambiente de navegação, planejamento de trajetória, auto-localização e a realização de demais tarefas.

A consideração do ambiente com sendo estruturado cobre os principais campos de aplicação dos robôs móveis, uma vez que, praticamente todos os ambientes construídos pelo homem são estruturados, contendo alguma regularidade na informação que pode ser extraída para fornecer subsídios para a navegação, tais como retas e planos. Ambientes internos como salas e corredores podem ser considerados como estruturados.

A utilização de um espelho omnidirecional possibilita que todo o ambiente ao redor do robô seja registrado em uma única tomada de imagem, ou contrário do que acontece quando informação estéreo é obtida com a utilização de duas câmeras não omnidirecionais, ainda elimina-se a dificuldade do alinhamento e calibração deste sistema.

Como visto um sistema de visão estéreo omnidirecional baseado em um espelho duplo com ambas as seções com perfil hiperbólico apresenta algumas características que o tornam mais eficiente em relação a outros sistemas de visão omnidirecional: possui um centro único de projeção, garante o alinhamento perfeito das duas imagens; garante naturalmente que as linhas epipolares são linhas radiais na imagem omnidirecional e linhas verticais nas imagens perspectivas obtidas através da projeção da imagem omnidirecional em um cilindro. Essas características fazem esse sistema ser ideal para recuperação tridimensional e, assim, capaz de fornecer informação para a construção de um mapa detalhado do ambiente em torno do sistema.

Nota-se ainda que a possibilidade de obtenção de informação estéreo com uma única aquisição de imagem, sem a necessidade de translação do robô para obter uma segunda imagem e conseqüente adição de erro na informação estéreo, torna o a informação das distâncias obtidas pelo sistema utilizado muito mais confiável que nos sistemas estéreos convencionais que possuem apenas um espelho omnidirecional.

Ainda a utilização de visão computacional aliada à odometria se mostra um sistema bastante robusto para auto-localização em ambientes internos, sem a possibilidade de utilização de GPS, sem nenhuma alteração do mesmo, como marcos artificiais de ambiente, e sem conhecimento prévio do ambiente.

A utilização de um modelo com pontos pertencentes não somente das quinas mas das arestas inteiras do ambiente torna o sistema muito mais útil do que apenas o registro dos ângulos azimutais das quinas.

1.4 Estrutura da dissertação

Neste trabalho o Capítulo 1 é dedicado a uma introdução geral ao trabalho, passando pela revisão do estado da arte dos principais temas abordados nesse trabalho, que são: visão estéreo omnidirecional; os principais métodos utilizados para localização de robôs móveis; os principais métodos de construção de mapas de ambiente; e as principais técnicas para se resolver o problema de auto-localização e mapeamento simultâneos. Ainda nesse Capítulo encontram-se os objetivos e justificativas do trabalho.

No Capítulo 2 o sistema de visão omnidirecional estéreo utilizado nesse trabalho é explorado com maiores detalhes. O modelo de formação da imagem é descrito na Seção 2.1, as equações utilizadas no cálculo das distâncias dos objetos ao sistema de visão são apresentadas na Seção 2.2, o cálculo do perfil do espelho é mostrado na Seção 2.3 e o perfil do espelho utilizado nesse trabalho é apresentado.

Os ambientes utilizados para os testes do algoritmo de auto-localização e mapeamento simultâneos são mostrados em detalhes no Capítulo 3.. Na Seção 3.1 é descrito o ambiente virtual e na Seção 3.2 o ambiente real.

No Capítulo 4 o processo de reconstrução tridimensional é apresentado. A Seção 4.1 apresenta o processo de correspondência de pontos entre as imagens das seções interna e externa do espelho, incluindo uma breve descrição do algoritmo de programação dinâmica. As modificações nos resultados das bordas detectadas pelo do algoritmo de Canny e utilizadas na correspondência estéreo é discutida na Seção 4.2 e uma tentativa de obtenção da transformação de coordenadas utilizando-se apenas a detecção de bordas verticais é descrita na Seção 4.3.

O Capítulo 5 dedica-se a explicar com detalhes o algoritmo de auto-localização e mapeamento simultâneos desenvolvido nesse trabalho. Na Seção 5.1 é apresentado o método *Trimmed Iterative Closest Point*, mostrando as deduções das equações utilizadas para o cálculo da rotação, translação e fator de escala na transformação de coordenadas calculada entre dois conjuntos de pontos. Em seguida, na Seção 5.2 o algoritmo sem odometria inicialmente desenvolvido é explicado e na Seção 5.3 o algoritmo final desenvolvido no projeto é explicado em detalhes.

O Capítulo 6 apresenta os resultados para os algoritmos desenvolvidos e implementados no MATLAB, inicialmente para o algoritmo sem odometria em ambiente simulado (Seção 6.1) e em ambiente real (Seção 6.2) e para o algoritmo com odometria em ambiente simulado (Seção 6.3) e em ambiente real (Seção 6.4).

O Capítulo 7 descreve as principais características e implementações realizadas para tornar o algoritmo desenvolvido próprio para aplicações embarcadas, passando pelas explicações do hardware e software da câmera utilizada (Seção 7.1), pela escolha do hardware para executar a aplicação (Seção 7.2), pela escolha do sistema operacional utilizado para executar a aplicação (Seção 7.3) e finalmente pela estrutura geral da implementação do software em C++ (Seção 7.4).

O Capítulo 8 dedica a mostrar os resultados do algoritmo desenvolvido e implementado em C++ em ambiente simulado (Seção 8.1) e ambiente real (Seção 8.2). Em seguida uma análise do tempo de processamento do algoritmo é mostrada na Seção 8.3.

O capítulo 9 apresenta as conclusões e considerações finais do trabalho, assim como uma visão geral dos trabalhos futuros que podem ser realizados a partir dos desenvolvimentos e resultados obtidos nesse trabalho.

O Capítulo 10 apresenta a lista de referências bibliográficas utilizadas no presente trabalho.

Em seguida são apresentados os anexos. O Anexo A explica em detalhes como utilizar o software desenvolvido em C++ e o Anexo B apresenta sucintamente o conteúdo do CD disponibilizado com o volume da dissertação.

1.5 Sumário

Nesse capítulo foram mostrados os principais sistemas de visão estéreo omnidirecional, sendo que o sistema utilizado nesse trabalho é um sistema catadióptrico formado por um espelho duplo de perfil hiperbólico e uma câmera. Os principais métodos de localização de robôs móveis são sucintamente descritos, indicando como a visão computacional e a geração de mapas de ambiente se relacionam na tarefa de localização. O problema de construção de mapas de ambiente é abordado, mostrando-se os principais métodos de construção de mapas. O problema de SLAM, que consiste na solução simultânea da localização de robôs móveis e da construção de mapas de ambiente, é também apresentado.

2 ESPELHO DUPLO DE PERFIL HIPERBÓLICO

Nesse Capítulo o sistema de visão estéreo omnidirecional utilizado nesse trabalho é descrito em detalhes.

A figura 2.1 apresenta um desenho esquemático do sistema de visão omnidirecional estéreo mostrando dois raios de luz incidentes e refletidos nos dois perfis do espelho. Devido à simetria tangencial do sistema apenas uma seção transversal radial do espelho precisa ser considerada.

Figura 2.1 – Seção transversal do sistema de visão estéreo omnidirecional mostrando os raios de luz incidentes e refletidos nos espelhos emitidos por um ponto P no espaço.

Uma câmera perspectiva juntamente com um espelho hiperbólico forma um sistema de aquisição de imagem de amplo ângulo de visão com dois centros de projeção, localizados nos pontos focais da hipérbole. Na figura 2.1, os pontos F_0 e F_I são os dois focos do perfil interno do espelho hiperbólico e os pontos F_0 e F_E são os focos do perfil externo do espelho hiperbólico. Todos os raios incidentes no perfil interno do espelho passam pelo foco F_I e são refletidos passando pelo outro foco F_0 . Analogamente, todos os raios de luz incidentes no

perfil externo do espelho hiperbólico passam pelo foco F_E e são refletidos passando pelo outro foco F_0 . O foco F_0 do perfil interno e externo dos espelhos hiperbólicos deve coincidir com o centro focal da câmera.

Para se obter um bom sistema de visão estéreo é desejável que as duas seções do espelho tenham o mesmo campo de visão. O campo de visão é definido pelos ângulos de elevação mínimo e máximo (ϕ_{min} e ϕ_{max}), ver figura 2.2. Obter um mesmo campo de visão para ambas as seções do espelho com perfil hiperbólico e com o eixo óptico da seção externa alinhada com o eixo da câmera resulta em uma seção externa com um diâmetro muito maior do que a seção interna, fazendo com que a resolução da imagem interna seja muito inferior à resolução da seção externa.

Figura 2.2 – Ângulos de elevação máximo e mínimo do sistema de visão omnidirecional estéreo com espelho duplo de seções com perfis hiperbólicos.

O erro na medição de distâncias é proporcional à resolução de ambas as imagens interna e externa, portanto, é importante que as duas imagens possuam a mesma resolução radial, ou seja, as imagens devem possuir o mesmo número de pixels na direção radial.

O espelho duplo de perfil hiperbólico usado nesse trabalho resolve o problema da diferença de resolução nas imagens das seções interna e externa do espelho. O eixo óptico do perfil hiperbólico interno é alinhado com o centro óptico da câmera, enquanto o eixo óptico do perfil hiperbólico externo não é alinhado com o centro óptico da câmera. O perfil da seção

externa é uma hipérbole rotacionada em torno do ponto F_0 de um ângulo λ . Essa solução garante o mesmo campo de visão com a mesma resolução nas imagens geradas em ambas as seções do espelho.

A figura 2.3 apresenta as mudanças no perfil externo do espelho para alguns ângulos de rotação. É fácil observar a diminuição da seção externa com o aumento do ângulo de rotação. Observa-se que todos os perfis mostrados possuem a mesma distância entre os focos da seção interna (ponto F_I) e externa do espelho (ponto F_E) e o mesmo campo de visão.

Figura 2.3 – Mudança no perfil externo do espelho para algumas variações do ângulo de rotação do eixo da hipérbole da seção externa do espelho para o mesmo campo de visão.

2.1 Modelo de formação da imagem

Um espelho hiperbólico é definido no sistema de coordenadas do espelho centrado no ponto focal F_0 como,

$$z = c + \frac{a\sqrt{b^2 + r^2}}{b}, \quad (2-1)$$

onde c , b e a são parâmetros da hipérbole, com a restrição de que $c^2 = a^2 + b^2$. Esses parâmetros são diferentes para os perfis interno e externo. A equação (2-1) é válida para o

perfil interno da hipérbole descrita no sistema de coordenadas da câmera F_0-rz e para o perfil externo da hipérbole descrito no sistema de coordenadas do perfil externo $F_0-r_Ez_E$.

Dado um ponto P no espaço, como mostrado na figura 2.1, dois raios de luz emitidos por este ponto atingem a superfície do espelho, um na seção interna e o outro na seção externa. O raio de luz incidente com ângulo de elevação ϕ_I atinge o espelho interno no ponto M_I e é projetado no plano da imagem com raio ρ_I . O raio de luz incidente com um ângulo de elevação ϕ_E atinge o perfil externo do espelho no ponto M_E e é projetado no plano da imagem com raio ρ_E .

A equação para qualquer raio de luz incidente na superfície interna do espelho é dada por:

$$z = -r \cotan \phi_I + z_{FI}, \quad (2-2)$$

onde z_{FI} é a coordenada z do ponto F_I . Similarmente, para qualquer raio de luz incidente no perfil externo do espelho tem-se:

$$z = -(r - r_{FE}) \cotan \phi_E + z_{FE}, \quad (2-3)$$

onde (r_{FE}, z_{FE}) são as coordenadas do ponto focal F_E .

Para qualquer raio de luz refletido de ambos os perfis interno ou externo do espelho vale a seguinte equação:

$$\tan \theta = \frac{\rho}{f} = \frac{r_M}{z_M}, \quad (2-4)$$

onde f é a distância focal da lente, θ é o ângulo vertical da câmera, (r_M, z_M) são as coordenadas do ponto na superfície do espelho onde incide o raio de luz e ρ é o raio na imagem.

As coordenadas de qualquer ponto na superfície do perfil interno do espelho devem obedecer a equação (2-1) do perfil da seção interna. Igualmente, as coordenadas de qualquer ponto na superfície do perfil externo devem obedecer a equação (2-1) descrita no sistema de coordenadas $(F_0 - r_E z_E)$.

A equação (2-5) mostra a transformação de coordenadas do sistema de coordenadas da câmera para o sistema de coordenadas do perfil externo do espelho.

$$\begin{bmatrix} r \\ z \end{bmatrix} = \begin{bmatrix} \cos \lambda & -\sin \lambda \\ \sin \lambda & \cos \lambda \end{bmatrix} \begin{bmatrix} r^{(E)} \\ z^{(E)} \end{bmatrix}, \quad (2-5)$$

onde $(r^{(E)}, z^{(E)})$ são as coordenadas de um ponto descrito no sistema de coordenadas do perfil externo.

2.2 Cálculo da distância

Dado um ponto P no espaço, como mostrado na figura 2-1, a informação disponível para calcular as suas coordenadas no espaço (r_P, z_P) são os raios das suas projeções nas imagens interna e externa $(\rho_I$ e $\rho_E)$. Note que os raios de um ponto nas imagens são obtidos através de uma busca por pontos correspondentes. As coordenadas do ponto P são obtidas pela solução simultânea das equações (2-2) e (2-3) para os raios incidentes em P nos pontos refletidos nas seções interna e externa do espelho $(M_I$ e $M_E)$.

$$\begin{bmatrix} r_P \\ z_P \end{bmatrix} = \frac{1}{(\tan \phi_E - \tan \phi_I)} \begin{bmatrix} \tan \phi_I [z_{FI} - z_{FE}] - r_{FE} \\ z_{FE} \tan \phi_E - z_I \tan \phi_I + r_{FE} \end{bmatrix}. \quad (2-6)$$

As únicas variáveis desconhecidas na equação (2-6) são os ângulos de elevação $(\phi_I$ e $\phi_E)$, portanto, para obter a posição do ponto no espaço é necessário conhecer o valor da tangente dos ângulos de elevação. Por meio de considerações ópticas e geométricas a tangente de ϕ_I pode ser derivada, resultando em:

$$\tan \phi_I = \frac{2k_I \rho_I (f + \sqrt{f^2 + \rho_I^2})}{k_I^2 (f + \sqrt{f^2 + \rho_I^2})^2 - \rho_I^2}, \quad (2-7)$$

onde $k_I = (c_I - a_I)/(c_I + a_I)$. Igualmente para o perfil externo do espelho, a mesma análise resulta em:

$$\tan \phi_E^{(E)} = \frac{2k_E \rho_E^{(E)} (f^{(E)} + \sqrt{f^{(E)2} + \rho_E^{(E)2}})}{k_E^2 (f^{(E)} + \sqrt{f^{(E)2} + \rho_E^{(E)2}})^2 - \rho_E^{(E)2}}, \quad (2-8)$$

onde $k_E = (c_E - a_E)/(c_E + a_E)$ e $f^{(E)}$ e $\rho_E^{(E)}$ são respectivamente a distância focal e o raio da imagem dados no perfil externo descrito no sistema de coordenadas $(F_0 - r_E, z_E)$. $f^{(E)}$ e $\rho_E^{(E)}$

são calculados invertendo-se a equação (2-5), resultando em,

$$\begin{bmatrix} \rho_E^{(E)} \\ f^{(E)} \end{bmatrix} = \begin{bmatrix} \cos \lambda & \sin \lambda \\ -\sin \lambda & \cos \lambda \end{bmatrix} \begin{bmatrix} \rho_E \\ f \end{bmatrix}. \quad (2-9)$$

O ângulo de Elevação ϕ_E no sistema de coordenadas $(F_0 - rz)$ é obtido de $\phi_E^{(E)}$ por:

$$\phi_E = \phi_E^{(E)} - \lambda. \quad (2-10)$$

Em resumo, dados os raios dos pontos correspondentes das imagens interna e externa (ρ_I e ρ_E), usando as equações (2-7), (2-8), (2-9) e (2-10) as tangentes dos ângulos de elevação dos raios de luz incidentes são calculados e por meio da equação (2-6) a posição do ponto no espaço é obtido. Note que nesse processo não é necessário obter os ângulos de elevação, mas apenas as suas tangentes, o que torna os cálculos muito rápidos e fáceis de serem implementados em um sistema digital. A derivação das equações presentes nesta seção é mostrada com detalhes no trabalho de Souza Jr. (2004).

2.3 Cálculo do perfil do espelho

Lembrando que os perfis das duas seções do espelho devem possuir o mesmo campo de visão, que é descrito pelos ângulos de elevação mínimo e máximo da cena. Para atingir a melhor resolução, a projeção do espelho na imagem deve ocupar o maior espaço possível, no caso toda a imagem, e para atingir igual resolução em ambos os perfis do espelho as duas imagens devem possuir o mesmo raio, ou seja,

$$(\rho_{E,\max} - \rho_{E,\min}) = (\rho_{I,\max} - \rho_{I,\min}), \quad (2-11)$$

onde $\rho_{I,\max}$, $\rho_{I,\min}$, $\rho_{E,\max}$ e $\rho_{I,\min}$ são respectivamente o maior e o menor raio da imagem interna e externa dos perfis do espelho.

Os parâmetros necessários para o projeto do espelho são os ângulos de elevação mínimo e máximo desejados, a distância focal da lente (f), o raio máximo da imagem ($\rho_{\max}$) e a distância axial entre os focos das hipérboles interna e externa (Δz_{IE}). Com exceção de $\rho_{\max}$, estes parâmetros são escolhidos baseando-se nos requisitos da aplicação. Os ângulos de elevação mínimo e máximo e a distância axial entre os pontos focais das hipérboles são escolhidos para obter a precisão desejada para uma máxima distância de um ponto ao sistema

de visão ver (Souza Jr., 2004).

Restrições geométricas fornecem três relações entre as excentricidades das hipérboles interna e externa, c_I e c_E , o ângulo λ e as coordenadas dos pontos focais F_I e F_E .

$$2c_I = z_{FI}; \quad 2c_E = \sqrt{r_{FE}^2 + z_{FE}^2}; \quad \tan \lambda = r_{FE} / z_{FE}. \quad (2-12)$$

As equações (2-11) e (2-12) em conjunto com as equações para os raios de luz incidentes e refletidos e os ângulos de elevação mínimo e máximo para os perfis interno e externo do espelho, equações (2-2) e (2-3), formam um sistema de equações não lineares. A solução deste sistema resulta na forma do perfil das seções interna e externa do espelho.

2.4 Características do espelho utilizado

A câmera utilizada possui um CCD de resolução de 1280 por 1024 pixels com um raio máximo de imagem igual a 3,5945mm. A lente possui distância focal igual a 25mm.

Todos os testes apresentados nesse relatório utilizaram o espelho duplo de perfil hiperbólico apresentado em Cabral et al. (2004 e 2007), que possui as seguintes características: 1) campo de visão que engloba ângulos de elevação da cena entre 22,6° e 85°; 2) distância entre os centros focais das seções interna e externa do espelho igual a 150mm; 3) raio externo do espelho igual a 60mm; e 4) ângulo de rotação do eixo do espelho externo igual a 3,5067°. A tabela 1.1 apresenta outros parâmetros desse espelho e a figura 2.4 apresenta uma foto do sistema constituído pelo espelho e pela câmera.

Tabela 1.1 – Outros parâmetros do espelho.

Parâmetro	Valor	Parâmetro	Valor
r_{FE}	25,756mm	z_{FE}	420,300mm
r_{FI}	0,000mm	z_{FI}	270,300mm
c_E	210,544mm	a_E	193,610mm
c_I	135,150mm	a_I	123,794mm
k_E	0,042	k_I	0,044

Na tabela 1.1, r_{FE} e z_{FE} representam respectivamente o raio e a altura do foco da hipérbole externa, r_{FI} e z_{FI} representam respectivamente o raio e a altura do foco da hipérbole interna, c_E e a_E representam os parâmetros da hipérbole externa, c_I e a_I representam os parâmetros da hipérbole interna e k_E e k_I representam as constantes geométricas das hipérboles externa e interna, para maiores detalhes sobre esse espelho ver Cabral et al. (2004 e 2007).

Figura 2.4 – Imagem do sistema de visão estéreo omnidirecional.

2.5 Sumário

Neste Capítulo foi apresentado o sistema de visão omnidirecional estéreo utilizado nesse trabalho. O sistema é composto de uma câmera CCD e um espelho duplo com as duas seções de perfis hiperbólicos. O eixo da hipérbole do perfil da seção externa do espelho é inclinado em relação ao eixo da hipérbole interna para obter uma mesma resolução nas imagens geradas pelos dois espelhos. Apresenta-se o modelo de formação da imagem do espelho e a derivação das equações que realizam cálculo de distâncias de objetos ao sistema de visão. Finalmente, as equações utilizadas para projetar o espelho duplo de seções hiperbólicas e as características do espelho utilizado são apresentadas.

3 AMBIENTES DE TESTE

No Capítulo anterior foi apresentado com detalhes o sistema de visão desenvolvido por Cabral et al. (2004). Nesse Capítulo são apresentados os dois ambientes que são utilizados para realizar os testes dos algoritmos desenvolvidos. O primeiro ambiente apresentado é um ambiente simulado digitalmente e fotografado pelo sistema de visão, que também é simulado nesse ambiente. O segundo ambiente é real e foi logicamente fotografado com o sistema de visão real que foi transladado para várias posições no ambiente.

3.1 Ambiente simulado

Um ambiente virtual para testes dos algoritmos desenvolvidos foi criado no programa PovRay©. Uma vista superior desse ambiente simulado pode ser vista na figura 3.1.

Figura 3.1 – Vista superior do ambiente simulado.

Basicamente o ambiente é composto por uma sala retangular de 4m de largura por 8m de comprimento. A sala tem 3m de altura, sendo que a altura zero corresponde a altura da imagem gerada na câmera. Nessa sala estão presentes diversos objetos:

- Dois armários idênticos de 2m de comprimento por 0,5m de profundidade e 1,5m de altura;
- Uma mesa com base cilíndrica de 1m de comprimento por 0,8m de largura e 0,05m de espessura;
- Uma janela e uma porta;
- Um tablado de 2,5m por 1,8m de largura e 0,2m de altura;
- Um quadro de 2,5m de comprimento por 1m de altura com 0,05m de profundidade;
- Uma esfera com 0,25m de raio;
- Quatro caixas idênticas, de 0,4m por 0,4m com 0,25 de altura, afastadas simetricamente 1m do centro da sala.

De modo a testar o algoritmo de SLAM desenvolvido uma trajetória foi criada na sala, com pontos de aquisição de imagens pelo sistema de visão omnidirecional. A figura 3.2 apresenta essa trajetória com pontos que marcam as posições de aquisição das imagens. A distância entre os pontos de aquisição de imagens é de no máximo 1m em cada direção. A tabela 3.1 mostra as coordenadas dos pontos no sistema de coordenadas do ambiente onde são adquiridas as imagens.

Imagens do ambiente foram também são adquiridas com espaçamento entre os pontos de aquisição de imagens em um mesmo eixo de no máximo 0,25m. As coordenadas das posições de aquisição das imagens são mostradas na tabela 3.2, notando-se que nesse caso são adquiridas 37 imagens.

Figura 3.2 – Trajetória do robô pelo ambiente simulado com as posições de aquisição das imagens mostradas em vermelho.

Tabela 3.1 – Coordenadas das posições de aquisição das 10 imagens da trajetória do robô adquiridas com espaçamento de 1m.

Ponto	Coord. X	Coord. Z	Ponto	Coord. X	Coord. Z
1	3000	1500	6	0	0
2	3000	500	7	-1000	500
3	3000	-500	8	-2000	500
4	2000	-500	9	-3000	500
5	1000	-500	10	-3000	1500

Tabela 3.2 – Coordenadas das posições de aquisição das 37 imagens da trajetória do robô adquiridas com espaçamento de 0,25m.

Ponto	Coord. X	Coord. Z	Ponto	Coord. X	Coord. Z	Ponto	Coord. X	Coord. Z
1	3000	1500	14	1750	-500	27	-1500	500
2	3000	1250	15	1500	-500	28	-1750	500
3	3000	1000	16	1250	-500	29	-2000	500
4	3000	750	17	1000	-500	30	-2250	500
5	3000	500	18	750	-375	31	-2500	500
6	3000	250	19	500	-250	32	-2750	500
7	3000	0	20	250	-125	33	-3000	500
8	3000	-250	21	0	0	34	-3000	750
9	3000	-500	22	-250	125	35	-3000	1000
10	2750	-500	23	-500	250	36	-3000	1250
11	2500	-500	24	-750	375	37	-3000	1500
12	2250	-500	25	-1000	500			
13	2000	-500	26	-1250	500			

Para efeito de exemplo de visualização, a imagem correspondente ao ponto seis da trajetória simulada com dez pontos de aquisição de imagens é mostrada na figura 3.3.

Figura 3.3 – Exemplo de uma imagem adquirida pelo sistema de visão estéreo omnidirecional no ambiente simulado.

3.2 Ambiente real

Após os algoritmos terem sido desenvolvidos e testados no ambiente simulado, eles são testados em um ambiente real. A sala MT-15A da Escola Politécnica da USP foi utilizada como o ambiente de teste para os algoritmos desenvolvidos nesse projeto. A figura 3.4 mostra uma vista superior dessa sala, que possui uma geometria em formato de “L” com 6,9m de comprimento.

Figura 3.4 – Vista superior da sala do ambiente real (dimensões em cm).

Como é possível observar na fotografia mostrada na figura 3.5 algumas mesas são utilizadas para aumentar a complexidade do ambiente. Nessa fotografia observam-se no chão as marcações em branco utilizadas para posicionar o sistema de visão estéreo omnidirecional.

Da mesma forma que no ambiente simulado, uma trajetória foi criada na sala para simular a movimentação do robô. A trajetória é composta por 59 pontos de aquisição de imagens espaçadas em até 0,3m. Esta trajetória pode ser vista na figura 3.6.

Figura 3.5 – Fotografia do ambiente real.

Figura 3.6 – Trajetória simulada no ambiente real. Pontos em vermelho indicam as posições de aquisição das imagens.

A tabela 3.3 apresenta as coordenadas das posições de aquisição das imagens (em centímetros) em relação à primeira imagem adquirida. O eixo x está na horizontal orientado para esquerda e o eixo y está na vertical orientado para cima.

Tabela 3.3 – Coordenadas das posições de aquisição das imagens da trajetória do robô no ambiente real.

Ponto	Coord. X	Coord. Y	Ponto	Coord. X	Coord. Y	Ponto	Coord. X	Coord. Y
1	0	0	21	480	120	41	175	390
2	30	0	22	480	150	42	145	390
3	60	0	23	480	180	43	115	390
4	90	0	24	480	210	44	85	390
5	120	0	25	450	210	45	55	390
6	150	0	26	420	210	46	25	390
7	180	0	27	390	210	47	0	390
8	210	0	28	360	210	48	0	360
9	240	0	29	330	210	49	0	330
10	270	0	30	300	210	50	0	300
11	300	0	31	270	210	51	0	270
12	330	0	32	250	210	52	0	240
13	360	0	33	235	210	53	0	210
14	390	0	34	235	240	54	0	180
15	420	0	35	235	270	55	0	150
16	450	0	36	235	300	56	0	120
17	480	0	37	235	330	57	0	90
18	480	30	38	235	360	58	0	60
19	480	60	39	235	390	59	0	30
20	480	90	40	205	390	1	0	0

Para efeito de exemplo de visualização as imagens correspondentes aos pontos cinco, vinte, vinte, trinta e quatro e quarenta e sete da trajetória simulada com cinquenta e nove pontos de aquisição de imagem são mostradas nas figuras 3.7, 3.8, 3.9 e 3.10.

Figura 3.7 – Visualização da imagem adquirida pelo sistema de visão omnidirecional no 5° ponto da trajetória simulada.

Figura 3.8 – Visualização da imagem adquirida pelo sistema de visão omnidirecional no 20° ponto da trajetória simulada.

Figura 3.9 – Visualização da imagem adquirida pelo sistema de visão omnidirecional no 34° ponto da trajetória simulada.

Figura 3.10 – Visualização da imagem adquirida pelo sistema de visão omnidirecional no 47° ponto da trajetória simulada.

4 PROCESSO DE RECONSTRUÇÃO TRIDIMENSIONAL

O processo de reconstrução tridimensional foi originalmente desenvolvido por Cabral et al. (2004). Esse processo não possuía inicialmente preocupação com relação ao tempo de processamento e o programa foi implementado no ambiente do software Matlab®.

O primeiro passo do processo de reconstrução é transformar a imagem, originalmente polar, para coordenadas retangulares. A imagem retangular, ou panorâmica, obtida nessa transformação possui 360 pixels de comprimento na horizontal, que representam os 360° em torno do sistema de visão. Observa-se que essa mudança de coordenadas pode ser realizada de diferentes formas e com diferentes graus de complexidade, sempre com o objetivo de manter o máximo possível as características da imagem original. A imagem omnidirecional tem radialmente 1024 pixels, na transformação da imagem omnidirecional para panorâmica, os 62 pixels centrais são descartados da imagem omnidirecional, resultando em 225 pixels para a altura de cada imagem panorâmica dos dois perfis do espelho.

Após a transformação da imagem polar para cartesiana, o algoritmo de detecção de bordas proposto por Canny (1986) é utilizado para obter os pixels das bordas de objetos no ambiente.

Um algoritmo de programação dinâmica proposto por Fielding e Kam (2000) é utilizado para realizar a correspondência dos pixels das imagens interna e externa do espelho. No processo de correspondência utilizam-se como referência os pixels das bordas presentes na imagem. A procura dos pontos correspondentes das bordas é realizada na imagem retangular original suavizada por um filtro de resposta impulsiva finita tipo Blackman simétrico, normalizado e com dimensão de sete por sete pixels. A vantagem de se utilizar o filtro de Blackman é que apresenta uma resposta com atraso de fase linear, ou seja, não causa distorção da fase nas frequências da imagem e mantém a amplitude da intensidade da imagem dentro dos limites da escala de cinza.

A programação dinâmica garante unicidade e monotonicidade na correspondência. Observa-se que na medida em que no processo de correspondência procuram-se somente pixels correspondentes nas regiões das bordas detectadas, esse processo se torna extremamente rápido. Uma vez correspondidos os pontos das imagens interna e externa é realizada a triangulação dos pontos para se obter a posição dos mesmos no espaço. Na triangulação utilizam-se as equações desenvolvidas por Cabral et al. (2004).

Nesse trabalho esse processo de reconstrução tridimensional das imagens foi modificado como é discutido em itens posteriores desse Capítulo.

4.1 Correspondência

Um problema fundamental em visão estéreo é estabelecer a correspondência entre os elementos das duas imagens. A restrição da linha epipolar reduz o problema de busca de pontos correspondentes em duas imagens a uma busca unidimensional ao longo da linha epipolar. A restrição epipolar para sistema catadióptricos com espelhos hiperbólicos foi estudado por Svoboda et al. (1998).

O arranjo de espelho duplo com seções coaxiais garante o alinhamento vertical das duas hipérboles e uma imagem mostra as duas vistas da cena. O espelho de perfil duplo garante linhas epipolares correspondentes em ambas as imagens omnidirecional e panorâmica, como mostra a Figura 4.1. As linhas epipolares são linhas radiais na imagem omnidirecional e linhas verticais nas imagens panorâmicas.

O fato das linhas epipolares serem correspondentes permite utilizar um método de busca eficiente para realizar a correspondência estéreo e, assim, obter a reconstrução tridimensional das imagens de forma extremamente rápida.

Figura 4.1 – Geometria epipolar. (a) As linhas epipolares são linhas radiais na imagem omnidirecional. (b) Quando a imagem é transformada para coordenadas retangulares as linhas epipolares se tornam linhas verticais.

A imagem panorâmica é utilizada para o processo de correspondência. Pontos correspondentes são procurados nas linhas epipolares correspondentes que são as colunas das imagens interna e externa. As bordas horizontais são utilizadas como características para a correspondência. Para quantificar a similaridade dos pixels correspondentes, os valores de intensidade dos pixels nas posições das quinas horizontais nas imagens panorâmicas internas e externas são comparadas utilizando-se uma função de similaridade. A função de similaridade utilizada é a Soma das Diferenças Absoluta (SDA), utilizando uma janela de 7 pixels de altura por 3 pixels de comprimento. A utilização da SAD é preferida em detrimento de outras funções de similaridade mais complexas para cálculos em aplicações de tempo-real, devido ao seu baixo custo computacional (Fielding e Kam, 2000).

O método de correspondência utilizado é a programação dinâmica, que é implementado utilizando um método proposto por Fielding e Kam (2000). Restrições de máxima e mínima disparidade são adicionadas à programação dinâmica para minimizar o custo computacional. Embora a programação dinâmica apresente um alto custo computacional, a alta qualidade dos seus resultados justifica a sua utilização.

4.1.1 Programação dinâmica

A programação dinâmica é um método de organizar um problema de otimização de forma a explorar a estrutura recursiva dos cálculos necessários. Muitos trabalhos têm usado essa abordagem para correspondência estéreo. A principal vantagem e motivo para se usar a programação dinâmica é a restrição de ordenação monotônica que estabelece que em uma linha epipolar, se um determinado ponto na imagem A de coordenada x_{Ai} corresponde a um determinado ponto na imagem B de coordenada x_{Bj} , então x_{Ai+1} só pode corresponder com x_{Bj+k} se $k \geq 1$. Sem a restrição de ordenação monotônica é necessário que seja feita uma busca exaustiva.

4.2 Detecção das bordas horizontais

Durante o desenvolvimento do projeto verificou ser necessário reduzir o número de pontos resultantes da reconstrução tridimensional. O objetivo é reduzir o número de pontos no

mapa global, que cresce excessivamente ao serem adicionados novas amostras de mapas locais e também reduzir o tempo de processamento do algoritmo de construção de mapas e auto-localização do robô, que se torna muito alto com o aumento de pontos. Outra vantagem da redução do número de pontos é a obtenção de um mapa de ambiente mais limpo, com as bordas dos objetos presentes no ambiente mais definidas, e com uma redução significativa de ruídos da reconstrução.

Para realizar a redução do número de pontos optou-se, inicialmente, pela seleção de apenas um ponto por incremento angular de 1° no mapa local.. Essa seleção restringe o número de pontos no mapa local em no máximo 360 pontos (um ponto para cada incremento angular de 1°).

Como mencionado, a detecção de bordas nas imagens é realizada pelo método de Canny (1986). Outra forma de reduzir o número de pontos na reconstrução tridimensional é utilizar somente as bordas horizontais mais próximas das imagens. Assim, após a detecção das bordas nas imagens interna e externa, eliminou-se as bordas verticais de forma que somente as bordas horizontais mais próximas do sistema de visão fossem utilizadas no processo de correspondência. Observa-se que todos os pixels de uma borda vertical estão na mesma distância ao sistema de visão, isso permite desconsiderá-las. Dessa forma eliminam-se inúmeros pixels da imagem selecionando-se apenas os pixels de maior interesse para serem candidatos à correspondência. Essa modificação resulta em uma reconstrução com pouco ruído e com um tempo de processamento muito menor do que o necessário se fossem utilizadas todas as bordas encontradas nas imagens, pois se reduz consideravelmente o número de pixels a serem processados pelo algoritmo de programação dinâmica.

Para efeito de visualização, a figura 4.2 apresenta a detecção de somente as bordas horizontais em comparação com a detecção de todas as bordas. Na imagem superior e inferior da esquerda, têm-se os resultados da detecção de bordas pelo método de Canny (1986) nas imagens externa e interna do espelho. Nessas imagens foram eliminados os pontos de bordas com pequenos números de pixels, esses pontos são considerados ruídos. Nas imagens da direita têm-se apenas as borda horizontais.

Figura 4.2 – Imagens resultantes após a eliminação das bordas verticais para se obter somente as bordas horizontais.

Considerando a primeira imagem adquirida na trajetória realizada no ambiente simulado, tem-se nas Figuras 4.3, 4.4 e 4.5 respectivamente a reconstrução do ambiente em 2D realizada com o processo original, com o processo original e posterior seleção dos pontos mais próximos do sistema de visão para cada incremento angular de 1° e com o algoritmo de detecção das bordas horizontais.

Figura 4.3 – Reconstrução das imagens utilizando o processo original.

Figura 4.4 – Reconstrução das imagens utilizando o processo original e posterior seleção dos pontos mais próximos do sistema para cada incremento angular de 1° .

Figura 4.5 – Reconstrução das imagens utilizando o algoritmo de detecção de somente das bordas horizontais.

Nota-se que ambos os processos para redução de pontos no mapa removem aproximadamente 60% dos pontos obtidos na reconstrução original e preservam de modo satisfatório a geometria definida pelas bordas dos objetos e do ambiente. Também se pode observar que a diferença no número de pontos obtidos nos dois processos de redução de pontos é muito pequena.

4.3 Detecção de bordas verticais

Outra modificação implementada e testada para o processo de reconstrução é a detecção e reconstrução apenas das bordas verticais presentes nas imagens do espelho. Assim, as retas verticais que geralmente consistem em quinas verticais de paredes, de objetos e de portas são utilizadas como marcos naturais do ambiente. A idéia é que ao se utilizar as quinas verticais de uma seqüência de imagens é possível calcular a translação e a rotação do robô utilizando o algoritmo de construção de mapas e auto-localização com um número extremamente reduzido de pontos. Após a obtenção da posição e orientação do robô, que

também define a transformação de corpo rígido entre as imagens, a reconstrução de todos os pontos das imagens é realizada para a geração do mapa de ambiente. Isso em princípio permite um processo extremamente rápido de construção de mapa.

Para uma melhor visualização, a figura 4.6 mostra a imagem panorâmica obtida pela transformação da imagem omnidirecional adquirida no primeiro ponto da trajetória do ambiente simulado. Na imagem da figura 4.6 nota-se a presença de linhas verticais fortes representando as quinas verticais das paredes e dos objetos no ambiente. É importante notar que essas linhas encontram-se na mesma linha epipolar da imagem, ou seja, no mesmo ângulo azimutal tanto na imagem externa (imagem superior) quanto na imagem interna (imagem inferior). Além disso, as retas verticais nas imagens representam somente a posição de um único ponto no plano do chão do ambiente (plano xy). Lembra-se que cada pixel na direção horizontal das imagens retangulares representa 1° da imagem omnidirecional.

Figura 4.6 – Imagem panorâmica obtida da imagem omnidirecional adquirida no primeiro ponto da trajetória no ambiente simulado.

A figura 4.7 apresenta o resultado da detecção de bordas com o método de Canny (1986) para as imagens interna (esquerda) e externa (direita) da figura 4.6. Pode-se observar que as linhas principais das quinas foram detectadas com sucesso.

Figura 4.7 – Resultado do algoritmo de Canny (1986) para a imagem da figura 4.8.

A partir da imagem binária com as bordas, obtida pelo método de Canny (1986), para cada incremento angular de 1° detecta-se a maior reta vertical e monta-se um histograma com o número total de pixels pertencente a esta reta. Este histograma pode ser visto na figura 4.8.

Figura 4.8 – Histograma da soma de pixels das retas verticais por incremento angular de 1° (somente a maior reta em cada incremento angular de 1° é considerada) presentes nas imagens da figura 4.7.

Adotou-se um limite mínimo para o número de pixels das retas verticais para que uma reta possa ser considerada uma reta importante e, assim, ser utilizada no processo de reconstrução. Esse limite está representado na figura 4.8 pela linha vermelha horizontal e é de 10 pixels. Note que o histograma superior foi gerado pela imagem externa e o histograma inferior gerado pela correspondente imagem interna.

Na figura 4.9 podem-se visualizar as retas detectadas pelo método proposto. Nessa figura os pixels pretos (mais ressaltados) representam as retas detectadas e os pixels cinzas (menos ressaltados) representam as bordas não detectadas como bordas verticais importantes ou nem mesmo como bordas verticais.

Figura 4.9 – Bordas verticais importantes detectadas nas imagens da figura 4.7.

Na Figura 4.10, que apresenta o ambiente reconstruído em 2D, pode-se ver mais claramente as quinas detectadas pelo método proposto. As estrelas vermelhas representam as retas verticais, ou quinas, detectadas. Das cinco quinas, ou retas verticais detectadas, apenas um ponto foi detectado falsamente, correspondendo a uma sombra. No restante, as quinas foram detectadas de forma correta.

Apesar dessa idéia se mostrar muito promissora, a detecção de quinas falsas causa um erro grande no cálculo da translação e rotação do sistema de visão pelo algoritmo de construção de mapas e auto-localização, tornado o método inviável para a geração de mapas globais consistentes do ambiente. Nota-se que o número de quinas necessárias para que o erro no cálculo da translação e rotação do robô causado por uma quina falsa seja desprezível é muito maior que o número de retas verticais importantes presentes na imagem, inviabilizando, assim, a utilização das retas verticais como marcos do ambiente que podem ser utilizados para resolver o problema de SLAM nesse projeto.

Figura 4.10 – Quinas verticais detectadas (estrelas vermelhas) nas imagens da figura 4.7.

Nota-se que Yagi et al. (1995), assim como alguns outros autores, utilizam a idéia de detectar as bordas verticais do ambiente para obter um mapa simplificado do mesmo baseado nos vértices de objetos obtidos nesta detecção. Nesses trabalhos as bordas são detectadas diretamente na imagem omnidirecional, sendo representadas por linhas radiais na imagem. Os vértices, ou quinas, são descritos pelos seus respectivos ângulos azimutais e a informação estéreo vem da solução de um sistema de equações diferenciais para cada ponto azimutal que possua correspondência em imagens consecutivas. Neste sistema não existem mapas locais e o mapa global é formado pela união linear das quinas verticais encontradas durante a navegação. Outra informação relevante diz respeito que sem outro sensor adicional é impossível para o robô saber a linha no mapa global é um espaço preenchido ou não, como uma parede ou uma porta aberta.

Nesse trabalho a idéia é de que toda informação contida nas imagens estéreo do ambiente possam ser utilizadas na formação de um mapa global rico em detalhes e ainda assim obter uma estimativa confiável para a rotação e translação do robô durante a navegação. Infelizmente um ambiente comum possui poucas quinas verticais naturais relevantes. Além disso, como pode ser visto nos testes, a presença de sombras pode gerar uma quina falsa muito facilmente e sem um sensor adicional para verificar se a quina está no local indicado ou

não a robustez do algoritmo de auto-localização e mapeamento simultâneo fica fortemente comprometida.

4.4 Sumário

Este capítulo apresentou o processamento da imagem omnidirecional para obter um mapa local do ambiente onde se localiza o robô. O processamento da imagem omnidirecional consiste dos seguintes passos: 1) transformação da imagem omnidirecional, em coordenadas polares, para uma imagem panorâmica, em coordenadas retangulares; 2) obtenção das características presentes nas imagens interna e externa usadas para realizar a correspondência estéreo; 3) utilizando a função Soma das Diferenças Absoluta (SAD) calcula-se a similaridade entre os pixels das imagens interna e externa onde estão localizadas as características obtidas no passo (2) obtendo-se possíveis candidatos à correspondência; 4) utilizando um algoritmo de programação dinâmica, que realiza a busca de pontos correspondentes nas linhas epipolares, obtém-se os pontos correspondentes das imagens; 5) dados os pontos correspondentes das imagens, utilizando-se um algoritmo de triangulação calcula-se a posição desses pontos no espaço, criando-se um mapa local do ambiente.

Como características utilizadas no processo de correspondência são testadas três possibilidades: todas as quinas presentes nas imagens, somente as quinas horizontais mais próximas do sistema de visão e somente as quinas verticais. Para detectar as quinas nas imagens utiliza-se o algoritmo de Canny (1986). Das três possibilidades de características analisadas, apenas as bordas horizontais serão utilizadas na construção do mapa local do ambiente. As bordas verticais não se apresentaram robustas o suficiente para um cálculo confiável da localização do robô e a utilização de todas as bordas presentes no ambiente gera muitos pontos nos mapas dificultando o seu processamento.

5 ALGORITMOS DE AUTO-LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS

No algoritmo de auto-localização e mapeamento simultâneos desenvolvido nesse trabalho a principal ferramenta utilizada para o cálculo do alinhamento e da correspondência dos mapas locais gerados é o *Trimmed Iterative Closest Point* (TrICP) desenvolvido por Chetverikov et al. (2002). Esse algoritmo é descrito nesse capítulo incluindo detalhes de como é implementado nesse trabalho em especial. Dois algoritmos diferentes baseados no TrICP são apresentados: um algoritmo que não utiliza a odometria como informação adicional para a geração do mapa de ambiente e um algoritmo que utiliza a informação da odometria.

5.1 Trimmed Iterative Closest Point (TrICP)

O algoritmo TrICP encontra a transformação de corpo rígido que minimiza a soma das distâncias ao quadrado entre os pontos de uma amostra e os pontos de um modelo, sendo que ao menos uma parte dos pontos da amostra encontra correspondência com pontos do modelo.

Para facilitar o entendimento, nesse trabalho, o conjunto de dados que é processado pelo TrICP é o resultado da reconstrução tridimensional da imagem adquirida pelo sistema de visão estéreo omnidirecional. Nota-se também que apesar do algoritmo de reconstrução tridimensional ser capaz, como o próprio nome diz, de calcular a coordenada de um dado ponto no espaço tridimensional (x , y , z), apenas a projeção desse ponto no plano do chão (plano xy) é utilizado no algoritmo TrICP. Esta simplificação é justificada, principalmente, pelo fato de que a projeção de um ponto no plano xy garante um mapa de ambiente conciso com o valor das distâncias dos objetos no ambiente ao eixo ótico do sistema de visão. Outro fator importante é que com esta simplificação o algoritmo TrICP também é simplificado e conseqüentemente diminui-se o esforço computacional, que passa de uma análise em 3D para 2D.

O algoritmo é composto por um processo iterativo de seis passos explicados a seguir. Nesse algoritmo define-se como modelo um mapa do ambiente, que é mantido estático e inalterado durante as iterações, e como amostra os mapas das vizinhanças do ambiente (ou mapa local) onde se encontra o sistema de visão (robô).

- 1) Para cada ponto da amostra calcula-se a menor distância ao quadrado, d_i^2 , em relação a todos os pontos do modelo contidos dentro de um círculo com um determinado raio;
- 2) Ordenam-se as distâncias ao quadrado (d_i^2) em ordem crescente e selecionam-se os pontos correspondentes. O número de pontos correspondidos na amostra, N_{po} , é dado por: $N_{po} = N_p \lambda$, sendo λ o grau de encobrimento da amostra e N_p o número de pontos da mesma;
- 3) Calcula-se a somatória das distâncias ao quadrado dos pontos correspondentes dos dois conjuntos de dados, S'_{LTS} (*Least Trimmed Square*);
- 4) Verifica-se a condição de parada do processo iterativo. Se a condição de parada for satisfeita, então parar; se não se faz $S_{LTS} = S'_{LTS}$ e executa o passo 5;
- 5) Calcula-se para os pares selecionados das possíveis correspondências o movimento (ou transformação) ótimo, ou seja, a rotação e a translação ($\mathbf{R}$, $\mathbf{t}$) da amostra em relação ao modelo que minimiza S_{LTS} ;
- 6) Aplica a transformação na amostra de acordo com ($\mathbf{R}$, $\mathbf{t}$) e volta para o passo 1.

A seguir seguem os detalhes da implementação de cada passo do algoritmo. Nota-se que foram implementadas algumas modificações com relação ao algoritmo inicialmente proposto por Chetverikov et al. (2002). Essas modificações têm por objetivo aumentar a robustez do algoritmo para o problema específico tratado nesse trabalho.

No passo 1, dado um ponto no espaço da amostra e um raio (que varia de acordo com o progresso do resultado) usa-se uma região circular, definida em torno do ponto considerado, de modo que apenas os pontos do modelo presentes dentro dessa região sejam considerados durante a busca. O raio da região circular de busca é atualizado ao longo do processo iterativo, conforme os dois conjuntos de dados se aproximam.

Para a atualização do raio, o algoritmo mantém o registro do valor da maior distância na amostra considerada, d_{max} , para os pontos correspondentes na amostra e no modelo. Um algoritmo de ordenação rápida (*quicksort*) é usado para ordenar as distâncias.

Nota-se que apesar do algoritmo minimizar a soma das distâncias ao quadrado, que devem ter o seu valor reduzido monotonicamente, o valor de algumas distâncias individuais,

incluído a maior distância, pode ocasionalmente aumentar. Observa-se que o valor de d_{max} diminui fortemente nas primeiras iterações.

O valor inicial de d_{max} é o parâmetro que reflete a maior distância possível entre pontos correspondentes e depende das dimensões e orientação dos dois conjuntos de dados. A escolha cuidadosa desse valor aumenta a chance de sucesso de algoritmo TrICP e também diminui consideravelmente o tempo de processamento das amostras. O valor inicial utilizado nesse trabalho é de 500 mm.

As condições de parada do processo iterativo verificadas no passo 4 são: número máximo de iterações atingido ou o valor do erro quadrático médio, definido pela equação (5-1), atingir um valor esperado.

$$e(\lambda) = \frac{S'}{N_{po}}, \quad (5-1)$$

onde S' é o *Least Trimmed Square*, definido pela equação,

$$S' = \sum_{i=1}^{N_{po}} d_i^2 \quad (5-2)$$

O número de iterações máximo utilizado como critério de parada do algoritmo é definido pelo tamanho do maior conjunto de pontos do modelo ou da amostra.

Para a integração desse algoritmo com o de construção de mapa de ambiente uma mudança importante foi implementada com relação ao algoritmo inicialmente proposto por Chetverikov et al. (2002). A mudança realizada é que nesse trabalho o algoritmo não necessariamente considera a convergência monotônica do erro quadrático médio, sendo guardado um registro do mínimo global durante as iterações até que algum dos critérios de parada seja atingido. Quando isso ocorre dois erros quadráticos médios são comparados, o erro da iteração que atingiu o mínimo global e o erro da iteração que atingiu o critério de parada. A amostra considerada como saída do algoritmo é a do mínimo global se o erro quadrático médio da iteração que atingiu o critério de parada for 10% maior do que o erro mínimo global, caso contrário a saída será a amostra do critério de parada.

Chetverikov et al. (2005) sugeriram um algoritmo para obter o grau de encobrimento, definido pela variável λ . O algoritmo utilizado nesse processo é denominado pelos autores de “seção de ouro” (*Golden Section*). O algoritmo da seção de ouro implementa uma variação do algoritmo de bissecção com intervalos ótimos para busca do mínimo. Uma implementação desse algoritmo é apresentada por Press et al. (2002). O problema desse algoritmo é que cada iteração da seção de ouro requer um processamento completo do TrICP, inviabilizando, assim, a sua utilização em tempo real. Nesse algoritmo, Chetverikov et al. (2005) sugerem a minimização da seguinte função,

$$\psi(\lambda) = \frac{e(\lambda)}{\lambda^{1+\xi}}, \quad (5-3)$$

onde $e(\lambda)$ é o erro quadrático médio determinado em função do grau de encobrimento λ e ξ é um parâmetro constante de ajuste, sendo maior do que zero. A vantagem em se minimizar $\psi(\lambda)$ ao invés do próprio erro, $e(\lambda)$, segundo Chetverikov et al. (2005), é que $\psi(\lambda)$ possui um único mínimo, enquanto que $e(\lambda)$ pode não possuir nenhum mínimo ou pode ter um mínimo global difícil de ser diferenciado dos mínimos locais.

O valor do parâmetro de ajuste ξ sugerido por Chetverikov et al. (2005) é de 2,0. Note que este parâmetro define a potência a qual o grau de encobrimento será elevado, e uma vez que o grau de encobrimento deverá assumir um valor entre 0,5 e 1,0 quando maior a potência, mais próximo de zero o resultado da equação (5-3) e quanto menor a função $\psi(\lambda)$ se aproxima da própria função do erro quadrático médio $e(\lambda)$. Nesse trabalho, após alguns testes com diferentes valores de ξ , observou-se que valores até 0,5 influenciam muito pouco no resultado do mínimo global encontrado e valores superiores a 3,5 alteram drasticamente o resultado. Optou-se, portanto, em manter o valor de 2,0. Após vários testes observou-se que o grau de recobrimento, λ , que minimiza o erro quadrático médio e ao mesmo tempo maximiza a utilização dos pontos da amostra gira em torno de 0,7, sendo adotado, portanto este valor como uma constante padrão para o algoritmo.

Oliveira (2005) mostra que é possível medir a qualidade da transformação de coordenadas encontrada analisando-se um parâmetro de saída do algoritmo TrICP chamado Erro Quadrático Relativo Percentual (S_{rel}). Esse parâmetro mede a diferença percentual entre o Erro Quadrático na primeira e na última iteração do TrICP.

A transformação entre dois sistemas de coordenadas cartesianas pode ser interpretada como sendo o resultado de um movimento de corpo rígido e pode, portanto, ser decomposto em uma rotação e uma translação. Na prática, as medições não são exatas, ou seja, apresentam incertezas, portanto, uma maior precisão em determinar os parâmetros da transformação é obtida usando-se mais do que três pontos, que é o mínimo número necessário de pontos.

5.1.1 Cálculo do alinhamento entre a amostra e o modelo

Dentre os algoritmos estudados para realizar o alinhamento entre os sistemas de coordenadas da amostra e do modelo, escolheu-se a implementação de um caso especial para pontos co-planares utilizando-se o quatêrion unitário descrito por Horn (1987).

Consideram-se dois conjuntos de dados m e p . Os pontos destes conjuntos são representados pelos vetores de posição $\mathbf{r}_{m,i}$ e $\mathbf{r}_{p,j}$ respectivamente. O algoritmo de alinhamento transforma o conjunto de dados p (denominado por amostra) para o sistema de coordenadas do conjunto de dados m (denominado por modelo ou mapa).

No processo de alinhamento procura-se por uma transformação de coordenadas do sistema de coordenadas da amostra para o sistema de coordenadas do mapa da seguinte forma:

$$\mathbf{r}_{m,j} = s\mathbf{R}(\mathbf{r}_{p,j}) + \mathbf{r}_0, \text{ para } j = 1 \dots N_{po}. \quad (5-4)$$

onde N_{po} é o número de pontos da amostra correspondidos, s é um fator de escala, $\mathbf{r}_0$ é a translação entre os dois conjuntos e $\mathbf{R}$ é a matriz de rotação, que representa a rotação do conjunto de pontos da amostra em relação ao conjunto de pontos do mapa. O número de pontos correspondidos, N_{po} é dado por: $N_{po} = N_p \lambda$, onde N_p é o número de pontos da amostra e λ é o grau de encobrimento da amostra com relação ao mapa.

Usando-se o fato de que a rotação é uma operação linear e mantêm os comprimentos dos vetores, tem-se que:

$$\|\mathbf{R}(\mathbf{r}_{p,j})\|^2 = \|\mathbf{r}_{p,j}\|^2, \text{ para } j = 1 \dots N_{po}. \quad (5-5)$$

onde $\|\mathbf{r}_{p,j}\|^2 = \mathbf{r}_{p,j} \cdot \mathbf{r}_{p,j}$ é o comprimento ao quadrado do vetor $\mathbf{r}_{p,j}$.

A partir da Equação (5-4), o erro entre cada ponto correspondido do modelo e da amostra é dado por:

$$\mathbf{e}_j = \mathbf{r}_{m,j} - s\mathbf{R}(\mathbf{r}_{p,j}) - \mathbf{r}_0, \text{ para } j = 1 \dots N_{po}. \quad (5-6)$$

A não ser que os conjuntos de dados sejam perfeitos não é possível encontrar uma escala, uma rotação e uma translação que respeite a equação (5-4) para todos os pontos. Assim, ao invés disso, minimiza-se o a soma dos erros ao quadrado, ou seja:

$$\sum_{j=1}^{N_{po}} \|\mathbf{e}_j\|^2, \text{ para } j = 1 \dots N_{po}. \quad (5-7)$$

Para facilitar o cálculo da transformação entre os conjuntos de pontos da amostra e do modelo, definem-se as coordenadas de todos os pontos com relação ao centro dos pontos correspondidos na fase de correspondência do TrICP. Assim, tem-se $\mathbf{r}'_{m,j}$ como sendo as posições dos pontos do modelo relativas ao centro dos pontos correspondidos do modelo e $\mathbf{r}'_{p,i}$ como sendo as posições dos pontos da amostra relativas ao centro dos pontos correspondidos da amostra, ou seja:

$$\mathbf{r}'_{m,i} = \mathbf{r}_{m,i} - \bar{\mathbf{r}}_m, \text{ para } j = 1, \dots, N_{po}; \quad (5-8)$$

$$\mathbf{r}'_{p,i} = \mathbf{r}_{p,i} - \bar{\mathbf{r}}_p, \text{ para } j = 1, \dots, N_{po};$$

onde $\bar{\mathbf{r}}_m$ e $\bar{\mathbf{r}}_p$ são as posições médias dos pontos correspondidos respectivamente do modelo e da amostra, dados por:

$$\bar{\mathbf{r}}_p = \frac{1}{N_{po}} \sum_{j=1}^{N_{po}} \mathbf{r}_{p,j}; \quad (5-9)$$

$$\bar{\mathbf{r}}_m = \frac{1}{N_{po}} \sum_{j=1}^{N_{po}} \mathbf{r}_{m,j}.$$

A rotação existente entre os dois conjuntos é calculada de forma a minimizar a soma das distâncias ao quadrado entre os pontos correspondentes, ou seja, essencialmente deseja-se minimizar a seguinte equação:

$$\sum_{j=1}^{N_{po}} \|\mathbf{r}'_{m,j} - \mathbf{r}'_{p,j}\|^2. \quad (5-10)$$

Considerando que a amostra esteja rotacionada de um ângulo θ com relação ao mapa, então para minimizar a equação (5-10) deve-se maximizar o produto escalar dos vetores de posição:

$$\sum_{i=1}^{Npo} \left| \mathbf{r}'_{m,i} \right\| \left| \mathbf{r}'_{p,i} \right| \cos \theta, \quad (5-11)$$

ou,

$$C \cos \theta + S \sin \theta, \quad (5-12)$$

onde:

$$C = \sum_{j=1}^{Npo} (\mathbf{r}'_{m,j} \cdot \mathbf{r}'_{p,j}); \quad (5-13)$$

$$S = \sum_{j=1}^{Npo} (\mathbf{r}'_{m,j} \wedge \mathbf{r}'_{p,j}).$$

A solução da equação (5-12) é facilmente obtida, sendo dada por:

$$\begin{aligned} \sin \theta &= \pm \frac{S}{\sqrt{S^2 + C^2}}; \\ \cos \theta &= \pm \frac{C}{\sqrt{S^2 + C^2}}. \end{aligned} \quad (5-14)$$

O cálculo da translação entre a amostra e o modelo é realizado transformando-se a equação (5-7) utilizando-se as coordenadas dos pontos do modelo e da amostra em relação às posições médias desses conjuntos de pontos. Assim, notando-se que,

$$\sum_{i=1}^{Npo} \mathbf{r}'_{m,i} = 0; \quad (5-15)$$

$$\sum_{i=1}^{Npo} \mathbf{r}'_{p,i} = 0;$$

a partir da equação (5-6) pode-se reescrever o erro $\mathbf{e}_j$ da seguinte forma:

$$\mathbf{e}_j = \mathbf{r}'_{m,j} - s \mathbf{R}(\mathbf{r}'_{p,j}) - \mathbf{r}'_0, \text{ para } j = 1, \dots, Npo \quad (5-16)$$

onde:

$$\mathbf{r}'_0 = \mathbf{r}_0 - \bar{\mathbf{r}}_m + s\mathbf{R}(\bar{\mathbf{r}}_p), \quad (5-17)$$

onde $\mathbf{R}(\bar{\mathbf{r}}_p)$ é a média dos pontos correspondidos da amostra rotacionados, ou seja, a translação é apenas a diferença entre o centróide do modelo e o centróide da amostra rotacionada multiplicada pelo fator de escala.

Substituindo a equação (5-16) na equação (5-7), a somatória dos erros ao quadrado se torna:

$$\sum_{i=1}^{N_{po}} \left\| \mathbf{r}'_{m,i} - s\mathbf{R}(\mathbf{r}'_{p,i}) - \mathbf{r}'_0 \right\|^2 \quad (5-18)$$

ou,

$$\sum_{j=1}^{N_{po}} \left\| \mathbf{r}'_{m,j} - s\mathbf{R}(\mathbf{r}'_{p,j}) \right\|^2 - 2\mathbf{r}'_0 \cdot \sum_{j=1}^{N_{po}} \left\| \mathbf{r}'_{m,j} - s\mathbf{R}(\mathbf{r}'_{p,j}) \right\| + N_{po} \left\| \mathbf{r}'_0 \right\|^2. \quad (5-19)$$

Dado que as distâncias são medidas com relação aos seus próprios centros, a somatória do termo do meio da equação (5-19) é zero. O primeiro termo independe de $\mathbf{r}'_0$ e o terceiro termo é sempre maior ou igual a zero. O erro total é obviamente minimizado com $\mathbf{r}'_0 = 0$. Retornando se para a Equação (5-17) e forçando $\mathbf{r}'_0 = 0$ obtêm se:

$$\mathbf{r}_o = \bar{\mathbf{r}}_m - s\mathbf{R}(\bar{\mathbf{r}}_p). \quad (5-20)$$

O erro entre cada ponto correspondido do modelo e da amostra, dado pela equação (5-6), pode ser rescrito da seguinte forma:

$$\mathbf{e}_j = \mathbf{r}'_{m,j} - s\mathbf{R}(\mathbf{r}'_{p,j}), \text{ para } j = 1, \dots, N_{po}, \quad (5-21)$$

pois $\mathbf{r}'_0 = 0$. Então o erro total a ser minimizado é:

$$\sum_{j=1}^{N_{po}} \left\| \mathbf{r}'_{m,j} - s\mathbf{R}(\mathbf{r}'_{p,j}) \right\|^2. \quad (5-22)$$

Expandindo o erro total dado pela equação (5-22), obtém-se a equação (5-23).

$$\sum_{j=1}^{N_{po}} \left\| \mathbf{r}'_{m,j} \right\|^2 - 2s \sum_{j=1}^{N_{po}} \mathbf{r}'_{m,j} \cdot \mathbf{R}(\mathbf{r}'_{p,j}) + s^2 \sum_{j=1}^{N_{po}} \left\| \mathbf{r}'_{p,j} \right\|^2, \quad (5-23)$$

que pode ser escrita na forma:

$$S_m - 2sD + s^2 S_p, \quad (5-24)$$

onde,

$$S_m = \sum_{j=1}^{Npo} \|\mathbf{r}_{m,j}\|^2, \quad S_p = \sum_{j=1}^{Npo} \|\mathbf{r}_{p,j}\|^2, \quad D = \sum_{j=1}^{Npo} \mathbf{r}_{m,j}' \mathbf{R}(\mathbf{r}_{p,j}), \quad (5-25)$$

onde S_m e S_p são as somas dos comprimentos dos pontos referente ao centróide ao quadrado e D é o produto escalar das coordenadas referentes ao conjunto de pontos do mapa com as coordenadas rotacionadas da amostra. A equação (5-24) pode ser reescrita da seguinte forma:

$$\left(s\sqrt{S_p} - D/\sqrt{S_p} \right)^2 + \frac{(S_m S_p - D^2)}{S_p}. \quad (5-26)$$

Minimizando a equação (5-26) com respeito a s , obtém-se $s = D/\sqrt{S_p}$. Substituindo D e S_p pelas equações definidas em (5-25), s pode ser reescrito na forma:

$$s = \frac{\sum_{j=1}^{Npo} \mathbf{r}_{m,j}' \mathbf{R}(\mathbf{r}_{p,j})}{\sum_{j=1}^{Npo} \|\mathbf{r}_{p,j}\|^2}, \quad (5-27)$$

onde $\mathbf{R}(\mathbf{r}_{p,j})$ indica os pontos da amostra relativos ao centro dos pontos correspondidos da amostra rotacionados.

5.2 Algoritmo de auto-localização e mapeamento simultâneos sem odometria

No trabalho anterior de Oliveira (2005) o mapa do ambiente era obtido satisfatoriamente utilizando-se o algoritmo TrICP para uma seqüência de imagens adquiridas durante uma navegação em um ambiente simulado. Além das imagens, nenhum dado adicional era utilizado. Os únicos conhecimentos prévios fornecidos ao algoritmo de geração de mapas de ambiente eram o S_{rel} limite e o ângulo limite para rotação da amostra. O valor de S_{rel} utilizado era baseado em conhecimento empírico para medição da qualidade da reconstrução do TrICP. O limite para a rotação do robô utilizado era de cinco graus, pois se assumia que o robô a ser utilizado possuía uma arquitetura de movimentação holomônica, ou

seja, o robô seria capaz de realizar mudanças na direção de navegação sem a necessidade de realizar curvas e sem mudar a sua orientação.

Inicialmente uma imagem é adquirida pelo sistema de visão omnidirecional, então essa imagem é processada e sua reconstrução tridimensional é realizada. Essa primeira reconstrução da imagem é então considerada o mapa inicial para o algoritmo de construção do mapa do ambiente. Para tornar mais rápido o processo de correspondência o mapa é filtrado de modo a serem considerados apenas os pontos mais próximos ao centro da imagem por incremento angular de 1° , dessa forma o mapa filtrado tem no máximo 360 pontos. A partir dessa primeira etapa o algoritmo incorpora novas imagens adquiridas até que não haja mais nenhuma imagem para ser integrada ao mapa global.

O algoritmo de construção do mapa do ambiente tem a possibilidade de utilizar as três últimas reconstruções do ambiente, que são nomeadas segundo a ordem de aquisição, como: última, penúltima e antepenúltima reconstrução da imagem adquirida. O primeiro passo do algoritmo é descartar a penúltima reconstrução e atualizar a ordem das reconstruções das imagens adquiridas anteriormente, de forma que a penúltima reconstrução se torna a antepenúltima e a última se torna a penúltima. Nota-se que são armazenadas a amostra original e a versão com número de pontos reduzidos das três últimas imagens adquiridas pelo sistema de visão estéreo omnidirecional. O objetivo de guardar essas imagens é ter mais possibilidades de tentar corresponder as imagens caso ocorra alguma falha.

Para cada nova imagem adquirida a sua reconstrução é considerada a última amostra do ambiente em relação ao mapa global. Após a aquisição de uma imagem, ela é processada, realizando-se a reconstrução tridimensional e a redução do número de pontos. Considera-se que a amostra é a reconstrução tridimensional da última imagem adquirida e o mapa é a fusão das imagens já adquiridas e reconstruídas.

Para cada novo mapa local do ambiente reconstruído até dez tentativas de correspondência são realizadas entre a imagem reconstruída e o mapa. As dez tentativas de correspondência são as seguintes: mapa filtrado – última amostra filtrada, última amostra filtrada – mapa filtrado, mapa – última amostra, última amostra – mapa, penúltima amostra filtrada – última amostra filtrada, última amostra filtrada – penúltima amostra filtrada, penúltima amostra – última amostra, última amostra – penúltima amostra, antepenúltima amostra – última amostra, última amostra – antepenúltima amostra. Nota-se que no caso de uma tentativa de correspondência falhar sempre se tenta realizar a correspondência do par modelo-amostra na outra direção, ou seja, primeiro tenta-se corresponder a amostra com o

mapa transformando a amostra para o sistema de coordenadas do mapa, se esse tentativa falhar, então, tenta-se fazer a correspondência transformando o mapa para o sistema de coordenadas da amostra. Isso é realizado porque, apesar dos mínimos locais serem os mesmos para o mesmo par, os caminhos para a minimização não são simétricos.

Um pseudocódigo deste algoritmo é mostrado no quadro da figura 5.1 e o diagrama lógico do mesmo é mostrado na figura 5.2.

Para determinar se o processo de correspondência foi bem sucedido duas condições são verificadas, a mudança relativa percentual do erro quadrático inicial e final da amostra e o ângulo de rotação final. Para que o valor do erro quadrático relativo percentual seja aceito para finalizar o processo, ele deve ser superior a 50% e o ângulo total da rotação não pode ser superior a 5°. Se o processo de correspondência falhar em todas as tentativas a imagem não é integrada ao mapa global.

Se algum dos processos de correspondência for bem sucedido é realizada uma fusão da amostra com o mapa global. A fusão é realizada com o objetivo de diminuir o número de pontos no mapa final, assim, para cada ponto da amostra transformada pelo TrICP são procurados no mapa os pontos que se enquadram dentro do círculo de um determinado raio escolhido pelo usuário, no caso o valor padrão vale 50 mm, e um único ponto médio substitui os pontos encontrados.

Após esse processo de fusão ocorre a integração final entre o mapa e a amostra gerando um único conjunto de pontos, sendo agora esse novo mapa o mapa global do ambiente. Uma filtragem é realizada com o objetivo de eliminar pontos espúrios e erros da reconstrução tridimensional, removendo-se os pontos isolados em um raio de 50 mm.

No decorrer do processo, os pontos que são correspondidos menos do que três vezes são eliminados do mapa global de modo a eliminar pontos desnecessários para as futuras iterações.

```

VetorPar[1] = {mapaFiltrado,ultimaFiltrada};
VetorPar[2] = {ultimaFiltrada,mapaFiltrado};
VetorPar[3] = {mapa,ultima};
VetorPar[4] = {ultima,mapa};
VetorPar[5] = {penultimaFiltrada,ultimaFiltrada};
VetorPar[6] = {ultimaFiltrada,penultimaFiltrada};
VetorPar[7] = {penultima,ultima};
VetorPar[8] = {ultima,penultima};
VetorPar[9] = {antepenultima,ultima};
VetorPar[10] = {ultima,antepenultima};

ultima = PegaNovaImagemReconstruida( );
ultimaFiltrada = FiltraMapaLocal(ultima);
mapa = ultima;
mapaFiltrado = ultimaFiltrada;

cont_corresp = 0;
Enquanto (TemImagem( ) == verdadeiro)

    antepenultima = penultima;
    antepenultimaFiltrada = penultimaFiltrada
    penultima = ultima;
    penultimaFiltrada = ultimaFiltrada;
    ultima = PegaNovaImagemReconstruida( );
    ultimafiltrada = filtraMapaLocal(ultima);

    mapaFiltrado = FiltraMapaGlobal(mapa);

    contador = 0;
    UltimoResultado("Falso");
    Enquanto (exit != verdadeiro && contador < 10)
        contador = contador + 1;
        transformacao = TrICP(VetorPar[contador]);
        Se (AceitaTransformacao(transformacao) == verdadeiro)
            UltimoResultado("OK");
            exit = verdadeiro;

    Se (UltimoResultado() == "OK")
        cont_corresp++;
        FusaoPontosProximos(mapa,mapalocal);
        mapa = Integra(mapa,mapalocal);
        mapa = FiltraRuidoMapa(mapa);
        Se (cont_corresp % 5 == 0)
            mapa = FiltraPontosMapa(mapa);

//Final

```

Figura 5.1 – Pseudocódigo do algoritmo de construção de mapa global do ambiente.

Os fatores que influenciam na capacidade desse algoritmo obter uma boa correspondência são: a distância entre imagens consecutivas, o número de pontos novos que potencialmente podem ser acrescentados ao mapa global e a ocorrência de mudança de direção na trajetória do robô.

5.3 Algoritmo de auto-localização e mapeamento simultâneos com odometria

O ponto de partida para o algoritmo desenvolvido neste trabalho foi o algoritmo apresentado na Seção anterior, onde o mapa do ambiente é obtido satisfatoriamente utilizando-se apenas as informações provenientes do TrICP para uma sequência de imagens adquiridas durante a navegação em um ambiente simulado.

O principal desafio para alcançar o objetivo proposto, ou seja, a construção de mapas de ambientes em tempo viável para uma aplicação embarcada está no fato de que os fatores que tornam a primeiro algoritmo bem sucedido são também os fatores que o tornam lento, ou seja, uma grande quantidade de reconstruções de imagens armazenadas e as múltiplas tentativas de correspondência utilizando o algoritmo TrICP. Portanto, é desejado um algoritmo mais otimizado computacionalmente. No foco dessa melhoria estão o algoritmo TrICP e a sequência de tentativas de correspondências utilizadas.

Uma das formas de diminuir o tempo de processamento é utilizar na reconstrução tridimensional da imagem somente os pontos das bordas horizontais. Isso diminui os pontos presentes nos mapas locais diminuindo o seu tempo da construção e, logicamente, torna as tentativas de correspondência muito mais rápidas. Outra forma é diminuir o número de tentativas de correspondências.

Muitas implementações foram testadas até a convergência para o algoritmo final apresentado a seguir. As experiências com os algoritmos testados revelaram muitas informações importantes para o desenvolvimento do algoritmo final.

- Primeiramente, ficou claro que era necessário que o algoritmo tivesse formas diferentes de tratar a etapa de correspondência para cada nova imagem adquirida baseado no conhecimento dos resultados das correspondências anteriores.

- Outra informação importante diz respeito à utilização do mapa global em tentativas de correspondências, estando ele filtrado ou não, observando-se que a sua utilização acaba gerando um erro maior do que o erro presente nas imagens reconstruídas e também aumenta o tempo de processamento.
- Outra característica notada é que a ocorrência de uma falha na correspondência de uma imagem reconstruída gera uma situação de instabilidade. Nesse caso o mapa global pode ficar muito “defasado” das novas amostras adquiridas e o retorno para uma situação de estabilidade, onde ocorre a correspondência dos mapas locais como mapa global, se torna cada vez mais difícil. Falhas consecutivas de correspondência das imagens reconstruídas pode causar uma situação onde o retorno se torna impossível, ou seja, não existe mais uma forma de conectar um determinado mapa local com o mapa global.
- Ficou claro que para que os objetivos do projeto fossem alcançados seria necessária a utilização de mais uma fonte de informação sobre a distância percorrida pelo robô entre as aquisições das imagens. Essa nova informação utilizada é a odometria.

O algoritmo de SLAM desenvolvido utiliza a informação de odometria do robô. A odometria é empregada como ponto de partida para o cálculo da transformação da amostra em relação ao modelo no processamento feito pelo TrICP. Observa-se que o emprego da odometria melhora consideravelmente a qualidade da correspondência, pois evita que o processo fique preso em mínimos locais que podem ser encontrados durante as iterações do cálculo da translação e rotação. Também é importante citar que com a utilização da informação da odometria o número de iterações do TrICP necessários para a convergência do processo iterativo diminui consideravelmente.

Na medida em que um robô real não é utilizado adotou-se um modelo para o erro de odometria. Assim, considera-se um erro de odometria incremental e aleatório de até 10% da distância percorrida para a translação do robô. Para o erro na medida de rotação do robô consideram-se duas possibilidades: um erro incremental e aleatório de até 10% da rotação relativa ou um erro aleatório com distribuição uniforme dentro de um intervalo de $[-1,5^\circ$ a $1,5^\circ]$, sendo utilizado o maior dos erros obtidos entre estes dois casos.

Na prática, o erro da odometria pode ser aumentado sem que o processo deixe de funcionar. Para isso é suficiente que a variável de controle de qualidade da transformação do

TrICP, o Erro Quadrático Relativo Percentual, seja igualmente flexibilizado. Por exemplo, se o algoritmo trabalha com um modelo de odometria que possua erro incremental da ordem de 20% o valor do Erro Quadrático Relativo Percentual também deverá ser flexibilizado no sentido de contemplar a mudança entre os erros quadráticos inicial e final no TrICP mais o erro natural da transformação.

O algoritmo permite uma relaxação do Erro Quadrático Relativo Percentual durante as iterações na taxa de 10% para cada iteração adicional que o algoritmo não for capaz de obter uma transformação bem sucedida para a correspondência entre as imagens reconstruídas. Após conseguir novamente uma nova correspondência o valor do Erro Quadrático Relativo Percentual retorna ao valor especificado como padrão para o algoritmo.

O rastreamento dos resultados anteriores permite que o algoritmo possua dois comportamentos com relação à nova amostra adquirida. O primeiro comportamento, considerado o padrão, é sempre utilizado independente do resultado anterior das tentativas de correspondência. O segundo comportamento, considerado como opcional, é utilizado em conjunto com o comportamento padrão quando o resultado anterior das tentativas de correspondência não obteve sucesso.

A utilização do comportamento opcional tem a função de corrigir a somatória dos resultados da odometria das tentativas de correspondência que falharam, uma vez que as tentativas de correspondência no comportamento normal são feitas apenas com base na reconstrução da última imagem apenas. Para exemplificar, suponha que a tentativa de correspondência entre as reconstruções das imagens A e B , utilizando odometria ab , tenha falhado em uma determinada iteração do algoritmo, nenhuma informação nova foi adicionada ao mapa global nesta iteração. Na próxima iteração a imagem C será igualmente reconstruída e a correspondência será tentada entre as reconstruções das imagens B e C com odometria bc . Caso esta reconstrução seja bem sucedida os pontos da reconstrução da imagem C deveram ser adicionados ao mapa global, mas a transformação calculada, $tnorm$, corrige apenas a odometria relativa entre B e C , sendo que a odometria entre A e B não é corrigida mas é utilizada na integração da amostra de C no mapa global, resultando em uma transformação total de $ab + bc + tnorm$. Se o comportamento opcional for utilizado, a reconstrução de C tentará ser correspondida diretamente com o mapa global, na tentativa de calcular mais uma nova transformação, top , que corrija o erro residual deixado pela odometria utilizada anteriormente, resultando em transformação total de $ab + bc + tnorm + top$.

No comportamento padrão a correspondência é realizada entre o par de amostras da última e penúltima imagens adquiridas. Duas tentativas de correspondência são realizadas nesse comportamento: entre a penúltima amostra e a última amostra com odometria, e entre a última amostra com odometria e a penúltima amostra.

No comportamento opcional a correspondência é realizada apenas entre a reconstrução da última imagem adquirida, transformada com a translação e rotação calculada no comportamento padrão, somada à odometria, e o mapa global completo sem restringir o número máximo de pontos utilizados em 360. Assim, a tentativa é entre o mapa global e a última amostra transformada.

Para determinar se o processo de correspondência foi bem sucedido apenas a mudança relativa percentual do erro quadrático inicial e final da amostra é verificada. Para ter a transformação de corpo rígido calculada aceita a variação relativa do erro deve ser inferior a trinta por cento mais a relaxação atual no comportamento padrão e inferior a 50% no comportamento opcional. Se o processo de correspondência falhar nas duas tentativas do comportamento padrão a imagem não é integrada ao mapa global e o comportamento opcional não é habilitado pois o mesmo serve apenas para tentar corrigir a odometria quando os dados da reconstrução da imagem devem ser adicionados ao mapa global.

Se algum dos dois processos de correspondência for bem sucedido uma fusão dos pontos próximos entre si é realizada com o objetivo de diminuir o número de pontos no mapa final. A fusão é realizada de forma que para cada ponto da amostra transformada pelo TrICP são procurados no mapa os pontos que se estão dentro do círculo de um determinado raio escolhido pelo usuário, no caso o valor padrão é 50 mm, e um único ponto médio substitui os todos os pontos nesse círculo.

Após esse processo de fusão ocorre a integração final entre o mapa e a amostra gerando um único conjunto de pontos, que é o novo mapa global do ambiente. Como no algoritmo anterior, uma filtragem é realizada com o objetivo de eliminar pontos espúrios e erros da reconstrução tridimensional, removendo-se os pontos isolados em um raio de 50mm. Também, no decorrer do processo, a cada cinco correspondências bem sucedidas, os pontos do mapa global que são correspondidos menos do que três vezes são eliminados de modo a eliminar pontos desnecessários para as futuras iterações.

Um pseudocódigo desse algoritmo é apresentado no quadro da figura 5.3 e um diagrama lógico é apresentado na figura 5.4.

```

CarregaOdometria( );
RandomizaErroOdometria( );
ultima = PegaNovaImagemReconstruida( );
mapa = ultima;
cont_corresp = 0;
Enquanto (TemImagem( ) == verdadeiro)
    penultima = ultima;
    ultima = PegaNovaImagemReconstruida( );
    ultimaOdometria = AplicaOdometria(ultima);
    Se (UltimoResultado() == "OK")
        Srel = SrelPadrao;
    Senao
        Srel += 10;

    transformacao = TrICP(penultima,ultimaOdometria);
    Se (AceitaTransformacao(transformacao) == falso)
        transformacao = TrICP(ultimaOdometria,penultima);
    Se (AceitaTransformacao(transformacao) == falso)
        continue;

    Se (UltimoResultado() != "OK" AND OpcionalLigado() == verdadeiro)
        transformacaotemp = TrICP(mapa,ultimaOdometria)
        Se (AceitaTransformacao(transformacao) == verdadeiro)
            transformacao += transformacaotemp;

    cont_corresp++;
    mapalocal = ultimoOdometria + transformacao;
    FusaoPontosProximos(mapa,mapalocal);
    mapa = Integra(mapa,mapalocal);
    mapa = FiltraRuidoMapa(mapa);
    Se (cont_corresp % 5 == 0)
        mapa = FiltraPontosMapa(mapa);

//Final

```

Figura 5.3 – Pseudocódigo do algoritmo de construção de mapa global do ambiente com odometria.

Figura 5.4 – Diagrama lógico do algoritmo de construção de mapa de ambiente com odometria.

5.4 Sumário

Neste Capítulo foram mostradas as deduções das fórmulas utilizadas nos cálculos realizados pelo algoritmo TrICP para a obtenção da translação e rotação entre dois conjuntos de pontos rigidamente ligados. O TrICP é a parte principal dos algoritmos de auto-localização e construção de mapas de ambiente (SLAM) desenvolvidos nesse trabalho. O TrICP é capaz de gerar informações robustas o suficiente para o algoritmo de SLAM sem odometria para geração de um mapa global de um ambiente.

Por meio da análise dos resultados das correspondência anteriores, da redução do número de tentativas de correspondência e da utilização da odometria o algoritmo com odometria pode ser utilizado em uma aplicação embarcada. Observa-se o desenvolvimento de um algoritmo embarcado para SLAM é um dos objetivos desse trabalho.

6 RESULTADOS DOS ALGORITMOS NO MATLAB

Os testes dos algoritmos de auto-localização e mapeamento simultâneos são inicialmente realizados no ambiente simulado. Para esses testes os algoritmos foram implementados no ambiente do programa Matlab®.

Os testes no ambiente simulado são realizados para o conjunto de 37 imagens com a trajetória iniciando na imagem um e terminando na imagem 37. Para facilitar a nomenclatura, a sequência de imagens nessa ordem é denominada trajetória direta.

Os testes no ambiente real são realizados para o conjunto de 60 imagens com a trajetória iniciando na imagem um e terminando na imagem um, completanto uma ou duas voltas completas ao redor do circuito. Para facilitar a nomenclatura, a sequência de imagens nessa ordem é denominada trajetória direta.

Os resultados mostrados nessa seção são referentes aos algoritmos de auto-localização e mapeamento simultâneos sem e com odometria.

Os resultados obtidos com os algoritmos são: 1) o mapa global com a localização do robô calculada pelo algoritmo; 2) o gráfico da evolução do erro quadrático relativo percentual, S_{rel} , em função das aquisições das imagens, ou seja, das iterações do algoritmo; 3) a trajetória real, a trajetória calculada e a trajetória fornecida pela odometria; 4) a evolução do erro percentual da auto-localização ao longo da trajetória realizada; 5) o desvio absoluto em milímetros entre a trajetória calculada pela auto-localização e a trajetória real; e 6) o número de iterações necessárias para a convergência do algoritmo para cada iteração.

Dado o mapa global, uma análise dos erros de reconstrução é feita com base nos comprimentos e posições das arestas dos ambientes. Para simplificar as descrições foi adotada uma numeração para as arestas mais importantes e que tem o seu erro de reconstrução informado. Assim, por exemplo, o valor -100 significa que a aresta possui um comprimento de 100 mm menor do que o comprimento real. As figuras 6.1 e 6.2 mostram os valores dos comprimentos das arestas em centímetros e a numeração utilizada para as arestas dos ambientes simulado e real respectivamente.

Figura 6.1 – Valor das arestas em centímetros e a numeração adotada para as mesmas no ambiente simulado.

Figura 6.2 – Valor das arestas em centímetros e a numeração adotada para as mesmas no ambiente real.

6.1 Resultados do algoritmo de auto-localização e mapeamento simultâneos sem odometria em ambiente simulado

Nesta seção o algoritmo de auto-localização e mapeamento simultâneos sem odometria, implementado no programa MatLab, é testado no ambiente simulado. A Figura 6.3 apresenta o resultado da sequência de imagens espaçadas de vinte e cinco centímetros, na ordem direta, ou seja, do ponto 1 ao 37. Nesta figura os pontos representam as bordas dos objetos no ambiente e as estrelas representam os pontos de aquisição de imagens calculadas pelo algoritmo durante a trajetória.

Figura 6.3 – Mapa do ambiente simulado reconstruído a partir das 37 imagens seqüenciadas na ordem direta obtido com o algoritmo sem odometria. Os números em vermelho indicam o erro de reconstrução nos comprimentos das arestas em mm.

Quanto aos resultados de reconstrução do ambiente, observa-se que o erro de reconstrução na arresta (1) é de cerca de -10cm ; na arresta (2) de cerca de -50cm ; na arresta (3) de -10cm ; na arresta (4) de -20cm ; na arresta (5) de 50cm ; na arresta (6) é imperceptível; na arresta (7) de -70cm ; na arresta (8) de -30cm ; na arresta (9) de -10cm ; e na arresta (10) de

–50cm. O erro total no comprimento da sala reconstruída, medido no eixo x , é de aproximadamente –80cm em 8m, representando um erro percentual de 10%. O erro do comprimento da sala medido no eixo y é de aproximadamente –20cm em quatro metros, representando um erro percentual de 5%. Além desses erros, nota-se uma deformação do perímetro da sala..

A coordenada (0, 0) no mapa da figura 6.3 representa a última posição do robô calculada pelo algoritmo de auto-localização. Nesse caso, essa posição está coerente com a posição final do robô no ambiente simulado. No geral nota-se que o algoritmo foi capaz de reconstruir o ambiente de modo correto e com as paredes do ambiente razoavelmente bem definidas.

A figura 6.4 apresenta a evolução do erro quadrático relativo percentual ao longo da incorporação das informações de novas imagens pelo algoritmo de auto-localização e mapeamento simultâneos sem odometria. É possível notar que apenas a imagem 16 não foi integrada ao mapa global, visto que o seu valor do Erro Quadrático Relativo Percentual foi abaixo de 50 %,

Figura 6.4 – Evolução do erro quadrático relativo percentual obtido com o algoritmo sem odometria no ambiente simulado.

A figura 6.5 apresenta em detalhes a trajetória real e a calculada pelo algoritmo de auto-localização. Pode-se observar claramente que o algoritmo, apesar de seguir corretamente a trajetória, apresentando um erro que aumenta conforme as novas imagens são adquiridas e processadas.

Figura 6.5 – Trajetórias real e calculada pelo algoritmo sem odometria no ambiente simulado.

As figuras 6.6 e 6.7 mostram respectivamente o erro percentual de localização do robô em relação à trajetória navegada e o erro absoluto em relação à posição simulada. Na figura 6.6 nota-se que o erro total de localização do robô converge para um valor em torno de 10%. Nota-se que o erro relativo no eixo x inicia-se na imagem 10 porque é nessa imagem que ocorre a primeira translação nesse eixo. Além do pico inicial, para o erro de auto-localização na direção y notam-se apenas pequenos erros. Com relação à figura 6.7 pode-se notar que o cresce até a incorporação das informações da imagem 30, quando então passa a apresentar um padrão oscilatório no intervalo de 800mm a 900mm. O erro total de localização atinge um máximo de 900mm quando as informações da imagem 33 são incorporadas. Finalmente, observa-se que a maior contribuição para o erro total de localização do robô é na direção x .

Figura 6.6 – Desvio percentual do erro de localização para o algoritmo sem odometria no ambiente simulado.

Figura 6.7 – Erro absoluto na localização do robô para o algoritmo sem odometria no ambiente simulado.

A figura 6.8 apresenta o número de iterações necessárias para a convergência do processo de correspondência do TrICP. Observa-se que é necessária uma média de 13 iterações para a incorporação das informações de cada imagem ao mapa global. Esse gráfico reflete a dificuldade do TrICP lidar com cada nova imagem adquirida. Ressalta-se que o número máximo de 30 iterações não foi alcançado em nenhuma das imagens.

Figura 6.8 – Número de iterações na construção de mapas para o algoritmo sem odometria no ambiente simulado.

6.2 Resultados do algoritmo de auto-localização e mapeamento simultâneos sem odometria em ambiente real

Nesta seção o algoritmo de auto-localização e mapeamento simultâneos sem odometria implementado no programa MatLab é testado no ambiente real. Os testes são realizados para o conjunto de 59 imagens no ambiente real apresentado no Capítulo 3. Utiliza-se a trajetória direta, ou seja, a trajetória que inicia na imagem de número um e segue até a imagem de número cinquenta e nove. Após essa sequência de movimentos uma nova volta completa é dada na sala.

A figura 6.9 apresenta o resultado da sequência de imagens reais processadas pelo algoritmo de auto-localização e mapeamento simultâneo sem odometria implementado em MatLab. Nessa figura os pontos representam as bordas de objetos no ambiente e as estrelas representam as posições do robô calculadas durante a trajetória.

Figura 6.9 – Mapa do ambiente reconstruído a partir das 60 imagens com dois ciclos sequenciados na ordem direta obtido com o algoritmo sem odometria.

Quanto aos resultados de reconstrução do ambiente nota-se claramente que o resultado da reconstrução não foi bem sucedido, entretanto se observa que apesar da enorme quantidade de ruído é possível ver o mapa do ambiente reconstruído em escala menor que o ambiente original. O mesmo vale para a auto-localização do robô, nota-se claramente que a trajetória reconstruída está correta do ponto de vista da direção e sentido de movimentação, mas os valores encontrados estão em uma escala inferior à do movimento real.

A apresentação desse resultado mal sucedido tem o objetivo de mostrar a importância da utilização da odometria como estimativa inicial da transformação. Nesse caso, o erro é causado pelo fato do ambiente ser constituído quase que somente por linhas paralelas fazendo com que para o sistema de visão ser praticamente impossível distinguir um local do outro no ambiente; ou seja, para o sistema de visão o ambiente é visto quase que da mesma forma em

qualquer posição da sala. Assim, sem a informação de odometria o algoritmo se perde e aceita transformações entre uma imagem e outra com uma translação menor do que a real, gerando erros grandes de localização do robô e por consequência erros grandes na reconstrução do ambiente.

A figura 6.10 apresenta a evolução do erro quadrático relativo percentual ao longo da incorporação das informações de novas imagens pelo algoritmo de auto-localização e mapeamento simultâneos sem odometria. Nota-se que 23 imagens não foram integradas ao mapa global de um total de 60 imagens.

Figura 6.10 – Evolução do erro quadrático relativo percentual obtido com o algoritmo sem odometria no ambiente real.

A figura 6.11 apresenta em detalhes a trajetória simulada e a calculada. Note que a trajetória calculada pelo algoritmo é uma versão em menor escala da trajetória real. Nesse caso não é relevante uma análise do erro percentual e absoluto da trajetória calculada em relação à trajetória original, uma vez que o resultado obtido claramente não foi satisfatório.

Figura 6-11 – Trajetórias real e calculada pelo algoritmo sem odometria no ambiente real.

A figura 6.12 apresenta o número de iterações necessárias para a convergência do processo de correspondência para a integração de cada uma das imagens adquiridas na trajetória. Observa-se uma média de 18 iterações para a integração de cada mapa local ao mapa global. Esse gráfico também reflete a dificuldade do algoritmo incorporar as informações de cada nova imagem adquirida. Ressalta-se que o número máximo de 30 iterações foi alcançado para as imagens 21, 30 e 31 e o limite de 40 iterações foi alcançado para as imagens 12 e 14

Figura 6.12 – Número de iterações necessárias para incorporar cada imagem na construção do mapa do ambiente real usando o algoritmo sem odometria.

6.3 Resultados do algoritmo de auto-localização e mapeamento simultâneos com odometria em ambiente simulado

Nesta seção o algoritmo de auto-localização e mapeamento simultâneos com odometria implementado no programa MatLab é testado no ambiente simulado. A figura 6.13 apresenta o resultado da sequência de imagens espaçadas de vinte e cinco centímetros, na ordem direta, ou seja, do ponto 1 ao 37, processadas pelo algoritmo. Nessa figura os pontos representam as bordas de objetos no ambiente e as estrelas representam os pontos de aquisição de imagens durante a trajetória.

Figura 6.13 – Mapa do ambiente reconstruído a partir das 37 imagens com o algoritmo com odometria. Os números em vermelho indicam o erro de reconstrução nos comprimentos das arestas em mm.

Quanto aos resultados de reconstrução do ambiente, observa-se que o erro de reconstrução na arresta (1) é imperceptível; na arresta (2) de -20 cm; na arresta (3) de 10 cm; na arresta (4) de -10 cm; na arresta (5) de 50 cm; na arresta (6) de 10 cm; na arresta (7) de -70 cm; na arresta (8) de -10 cm; na arresta (9) é imperceptível; e na arresta (10) é imperceptível. O erro total no comprimento da sala reconstruída, medido no eixo x , é visualmente imperceptível e medido no eixo y é de aproximadamente -20 cm para um comprimento de 4 m, representando um erro percentual de 5% .

A coordenada $(0, 0)$ no mapa representa a última posição do robô calculada pelo algoritmo de auto-localização. Essa posição está coerente com a posição final do robô no ambiente simulado. No geral nota-se que o algoritmo é capaz de reconstruir o ambiente de modo correto e com as paredes do ambiente bem definidas. Somente a quina esquerda da parte central inferior da imagem apresenta uma baixa qualidade de reconstrução, causada pela grande distância dessa quina à trajetória do robô.

A figura 6.14 apresenta a evolução do erro quadrático relativo percentual ao longo da incorporação das informações de novas imagens pelo algoritmo de auto-localização e

mapeamento simultâneos. É possível notar que apenas as imagens 8 e 10 não foram integradas ao mapa global.

Figura 6.14 – Evolução do erro quadrático relativo percentual obtido com o algoritmo com odometria no ambiente simulado.

A figura 6.15 apresenta em detalhes a trajetória simulada, a calculada e a obtida utilizando-se apenas a odometria sem nenhuma correção. Pode-se observar claramente que o algoritmo não tem dificuldade de seguir a trajetória correta, apresentando somente um pequeno erro. Além disso, observa-se que a trajetória calculada é mais precisa do que a trajetória obtida através da odometria na maior parte da trajetória.

Figura 6.15 – Trajetórias simulada, calculada pelo algoritmo com odometria e calculada pela odometria no ambiente simulado.

As figuras 6.16 e 6.17 apresentam respectivamente o erro percentual em relação à trajetória navegada e o erro absoluto em relação à posição simulada. Na figura 6.16 nota-se que o erro total de localização do robô converge para um valor em torno de 2% da sua posição. Nota-se que o erro relativo no eixo x inicia-se na imagem 10 porque é nessa imagem que ocorre a primeira translação nesse eixo. Além do pico inicial para o erro de auto-localização na direção y , notam-se apenas pequenos erros. Com relação à figura 6.17 pode-se notar o mesmo comportamento da figura anterior. O erro total de localização atinge um máximo de 140mm para a imagem 33. Finalmente, nota-se que o erro oscila entre 40mm e 140mm, e que a maior contribuição para o erro total é o erro na direção y .

Figura 6.16 – Desvio percentual do erro de localização para o algoritmo com odometria no ambiente simulado.

Figura 6.17 – Erro absoluto na localização do robô para o algoritmo com odometria no ambiente simulado.

A figura 6.18 apresenta o número de iterações necessárias para a convergência do processo de correspondência no TrICP. Observa-se uma média de 5 iterações para a incorporação das informações de cada imagem ao mapa global. Este gráfico reflete a dificuldade do algoritmo lidar com cada nova imagem adquirida. Ressalta-se que o número máximo de 30 iterações não foi alcançado em nenhuma das imagens.

Figura 6.18 – Número de iterações necessárias para incorporar as informações das imagens na construção do mapa usando o algoritmo com odometria no ambiente simulado.

6.4 Resultados do algoritmo de auto-localização e mapeamento simultâneos com odometria em ambiente real

Nesta seção o algoritmo de auto-localização e mapeamento simultâneos com odometria implementado no programa MatLab é testado no ambiente real. Os testes são realizados para o conjunto de 59 imagens do ambiente real. Utiliza-se a trajetória direta, ou seja, a trajetória que se inicia na imagem 1 e segue crescentemente até retornar à imagem 1 novamente. Dois testes são realizados, um dando uma volta e outro dando duas voltas completas na sala.

6.4.1 Resultados para uma volta completa

A figura 6.19 apresenta o resultado da sequência de imagens reais processadas pelo algoritmo de auto-localização e mapeamento simultâneo com odometria implementado em MatLab. Nessa figura os pontos representam as bordas de objetos no ambiente e as estrelas representam as posições do robô calculadas durante a trajetória.

Figura 6.19 – Mapa do ambiente reconstruído a partir das 60 imagens obtido com o algoritmo com odometria. Os números em vermelho indicam o erro de reconstrução nos comprimentos das arestas em mm.

Quanto aos resultados de reconstrução do ambiente, observa-se que o erro de reconstrução na arresta (1) de -10 cm; na arresta (2) de -10 cm; na arresta (3) de -10 cm; na arresta (4) de mais -10 cm; na arresta (5) de -5 cm; na arresta (6) é imperceptível; na arresta (7) de mais -5 cm; na arresta (8) é imperceptível. O erro total no comprimento da sala reconstruída, medido no eixo x , é visualmente imperceptível e o erro medido no eixo y é de -10 cm. Nota-se que o algoritmo foi capaz de reconstruir o ambiente de modo correto e com

as paredes do ambiente razoavelmente bem definidas, porém nota-se que a maior parte das quinas não estão conectadas.

A coordenada (0, 0) no mapa representa a última posição do robô calculada pelo algoritmo de auto-localização. Essa posição está coerente com a posição final do robô no ambiente. Observa-se que o erro final, ou seja, o erro na posição calculada do robô na imagem 60 é de 140 mm. Observa-se que esse erro representa o erro de posicionamento total do robô, pois o mesmo percorre uma volta completa na trajetória.

A figura 6.20 apresenta a evolução do erro quadrático relativo percentual ao longo da incorporação das informações de novas imagens pelo algoritmo de auto-localização e mapeamento simultâneos. Nota-se que 17 imagens não foram integradas ao mapa global de um total de 60 imagens. Lembrando que nesse algoritmo, após uma tentativa mal sucedida para incluir a reconstrução da imagem no mapa global, o Erro Quadrático Relativo Percentual é flexibilizado, portanto observa-se que as imagens 16, 19, 21, 23, 29, 41 e 46, apesar de apresentarem erros relativos acima do valor limite de 30%, tiveram a sua reconstrução adicionada ao mapa global.

Figura 6.20 – Evolução do erro quadrático relativo percentual obtido com o algoritmo com odometria no ambiente real.

A figura 6.21 apresenta em detalhes a trajetória simulada, a calculada e a obtida utilizando-se apenas a odometria. Pode-se observar claramente que o algoritmo não tem dificuldade de seguir a trajetória correta, entretanto, alguns pontos apresentam um erro considerável. Observa-se que a trajetória calculada é mais precisa do que a trajetória obtida por meio da odometria na maior parte da trajetória.

Figura 6.21 – Trajetórias simulada, calculada pelo algoritmo com odometria e calculada pela odometria no ambiente real.

As figuras 6.22 e 6.23 mostram respectivamente o erro percentual em relação à trajetória navegada e o erro absoluto em relação à posição real. Na figura 6.22 observa-se que o erro total de localização converge para um valor em torno de 0,8%. Observa-se que o erro relativo no eixo x inicia-se na imagem 18, pois é nessa imagem que ocorre a primeira translação nesse eixo. Além do pico inicial para o erro na direção x , notam-se apenas pequenos erros no cálculo da localização do robô. Com relação à figura 6.23 pode-se notar que o erro total de posição do robô atinge um máximo de 210 mm na imagem 23. Observa-se que o erro possui um comportamento oscilatório no intervalo entre 20mm e 210mm e que os erros nos dois eixos contribuem de forma semelhantemente para o erro total.

Figura 6.22 – Desvio percentual do erro de localização para o algoritmo com odometria no ambiente real.

Figura 6.23 – Erro absoluto na localização do robô para o algoritmo com odometria no ambiente real.

A figura 6.24 apresenta o número de iterações necessárias para a convergência da correspondência no TriCP na integração da informação de cada uma das imagens adquiridas na trajetória. Observa-se que são necessárias uma média de oito iterações para a integração de cada mapa local ao mapa global. Esse gráfico também reflete a dificuldade do algoritmo incorporar as informações de cada nova imagem adquirida. Ressalta-se que o número máximo de 30 iterações não foi alcançado para nenhuma das imagens adquiridas.

Figura 6.24 – Número de iterações na construção do mapa para o algoritmo com odometria no ambiente real.

6.4.2 Resultados para duas voltas completas

A figura 6.25 apresenta o resultado da sequência de imagens reais com duas voltas no ambiente real processadas pelo algoritmo de auto-localização e mapeamento simultâneo com odometria implementado em MatLab. Nessa figura os pontos representam as bordas de objetos no ambiente e as estrelas representam as posições do robô calculadas durante a trajetória. As estrelas de cor escuras são referentes às localizações do robô na primeira volta e as estrelas de cor clara são referentes às localizações na segunda volta.

Figura 6.25 – Mapa do ambiente reconstruído com o algoritmo com odometria com duas voltas do robô no ambiente com as imagens seqüenciadas na ordem direta.

Para esse caso não será feita uma análise detalhada dos erros da reconstrução. Nota-se apenas que o algoritmo foi bem sucedido em construir o mapa do ambiente nas duas voltas e que o mapa global pode ser visto claramente como sendo composto por dois mapas muito semelhantes. Esses dois mapas consistem respectivamente na reconstrução do ambiente na primeira e na segunda volta do robô. O mapa construído na segunda volta está transladado para baixo e para direita em relação ao mapa construído na primeira volta. Esse fato ocorre em razão dos erros de localização do robô. Mesmo com esses problemas pode-se concluir que o algoritmo foi capaz de reconstruir o ambiente de modo correto e com as paredes do ambiente razoavelmente bem definidas.

A coordenada (0, 0) no mapa representa a última posição do robô calculada pelo algoritmo de auto-localização. Essa posição está coerente com a posição final do robô no ambiente. Observa-se que o erro final de localização do robô, que é o erro na imagem 119, é de 320 mm. Esse erro representa o erro de posicionamento total do robô após o mesmo percorrer duas voltas completas no ambiente.

A figura 6.26 apresenta a evolução do erro quadrático relativo percentual ao longo da incorporação das informações de novas imagens pelo algoritmo de auto-localização e

mapeamento simultâneos com odometria. Nota-se que 40 imagens não foram integradas ao mapa global de um total de 119 imagens. Dessas 119 imagens não incorporadas 17 foram na volta e 23 na segunda volta. Lembrando que nesse algoritmo, após uma tentativa mal sucedida para incluir a reconstrução da imagem no mapa global, o Erro Quadrático Relativo Percentual é flexibilizado, as imagens 16, 20, 41, 58, 64(5), 68(9), 78(19), 81(22) e 100(41) apesar de apresentarem erro relativo maior do que o limite de 30% tiveram a sua reconstrução adicionada ao mapa global. Nota-se que o número entre parentes refere-se à numeração original das imagens.

Figura 6.26 – Evolução do erro quadrático relativo percentual obtido com o algoritmo com odometria no ambiente real com duas voltas do robô.

A figura 6.27 apresenta em detalhes a trajetória simulada, a calculada e a obtida utilizando-se apenas a odometria. Pode-se observar claramente que o algoritmo não tem dificuldade de seguir a trajetória correta, apesar de alguns pontos apresentam um erro considerável. Observa-se que a trajetória calculada é mais precisa do que a trajetória obtida por meio da odometria na maior parte da trajetória. Nota-se também que de forma geral a localização do robô calculada na primeira volta é mais exata na maior parte do percurso do que a calculada na segunda volta. Isso ocorre em razão da acumulação de erros de localização do robô.

Figura 6.27 – Trajetórias simulada, calculada pelo algoritmo com odometria e calculada pela odometria no ambiente real com duas voltas do robô.

As figuras 6.28 e 6.29 mostram respectivamente o erro percentual em relação à trajetória navegada e o erro absoluto em relação à posição real. Na figura 6.28 nota-se que o erro total converge para um valor em torno de 1% da posição real do robô. Observa-se que o erro relativo no eixo x inicia-se na imagem 18, pois é nessa imagem que ocorre a primeira translação nesse eixo. Além do pico inicial para o erro na direção x , notam-se apenas pequenos erros para na auto-localização do robô. Com relação à figura 6.29 pode-se notar que o erro de localização do robô total atinge um máximo de 325 mm para a imagem 81. Observa-se também que o erro possui um comportamento oscilatório na faixa de valores entre 50mm e 230mm na primeira volta e entre 150mm e 325mm na segunda volta. É possível notar um padrão de aumento e diminuição do erro durante cada volta. Na primeira volta inicialmente o erro cresce até a imagem 17, decresce até a imagem 41 e cresce novamente até a imagem 60. Na segunda volta o erro cresce até a imagem 81(22), decresce até a imagem 102(43) e depois cresce novamente até o final do percurso. O número entre parentes refere-se à numeração original das imagens. Finalmente, observa-se que o erro na direção x apresenta a maior contribuição no erro total.

Figura 6.28 – Desvio percentual do erro de localização para o algoritmo com odometria no ambiente real com duas voltas do robô.

Figura 6.29 – Erro absoluto na localização do robô para o algoritmo com odometria no ambiente real com duas voltas do robô.

A figura 6.30 apresenta o número de iterações necessárias para a convergência da correspondência na integração da informação de cada uma das imagens adquiridas ao longo da trajetória. Observa-se uma média de oito iterações para a integração de cada mapa local ao mapa global, semelhante ao caso onde o robô dá apenas uma volta no ambiente. Esse gráfico também reflete a dificuldade do algoritmo incorporar as informações de cada nova imagem adquirida. Ressalta-se que o número máximo de 30 iterações não foi alcançado para nenhuma das imagens adquiridas. Nota-se um padrão semelhante no número de iterações (picos) utilizadas por volta das imagens 20, 30 e 40 na primeira volta e 80, 90 e 100 na segunda volta.

Figura 6.30 – Número de iterações na construção de mapas para o algoritmo com odometria no ambiente real com duas voltas do robô.

6.5 Sumário

Nesse capítulo foram mostrados todos os resultados relevantes dos dois algoritmos apresentados no Capítulo 5.

Os resultados do algoritmo sem odometria para o ambiente simulado foram muito satisfatórios, visto que nenhuma informação adicional às imagens é fornecida para os

cálculos. O erro máximo das reconstruções das distâncias foi de 10% e o erro de posicionamento final do robô foi de 850mm.

No ambiente real o algoritmo sem odometria não apresentou resultados muito bons tanto a auto-localização do robô, quanto a construção do mapa global. Esse teste evidência a necessidade de se ter alguma informação adicional para o algoritmo em determinadas situações, principalmente nos casos onde o ambiente possui muitos mapas locais com pouca informação e muita redundância.

No caso do algoritmo com odometria testado no ambiente simulado, o mapa global do ambiente é de excelente qualidade, apresentando bordas do ambiente muito bem definidas e erros muito pequenos. Um fator muito importante apresentado por esse algoritmo é o tempo de processamento, que em relação ao algoritmo sem odometria, diminuiu em aproximadamente oito vezes. O tempo de processamento do algoritmo sem odometria possibilita a sua implementação embarcada em um robô móvel.

Para o algoritmo com odometria no ambiente real e com uma volta do robô, os resultados tanto da auto-localização, quanto da construção do mapa global são de excelente qualidade. A auto-localização calculada é notadamente muito próxima da posição real do robô na trajetória e a qualidade do mapa global final é muito boa, apresentando as bordas muito bem definidas e baixíssimos erros de reconstrução.

Os resultados do algoritmo com odometria no ambiente real para duas voltas completas do robô são muito parecidos com os resultados obtidos com somente uma volta do robô. No mapa global construído nota-se um aspecto de sobreposição de dois mapas, um gerado na primeira volta e outro na segunda volta. Esse erro é gerado pelo erro acumulado na localização do robô. Contudo, observa-se que esses dois mapas estão muito próximos entre si.

7 IMPLEMENTAÇÃO DO SOFTWARE EMBARCADO

Os resultados dos testes apresentados na seção anterior permitem concluir que o algoritmo de auto-localização e mapeamento simultâneos com odometria está pronto para ser implementado em um hardware eletrônico dedicado. A seguir são descritas as etapas realizadas no sentido de tornar o sistema desenvolvido embarcado.

7.1 Implementação na Câmera DVT

Inicialmente a idéia era utilizar o processador Hitachi SH4 da câmera Legend 544, com 64 Mb de memória RAM e 16 Mb de memória flash, para implementar o algoritmo de reconstrução tridimensional e um outro hardware dedicado para implementar o algoritmo de auto-localização e mapeamento simultâneos.

O processador Hitachi tem 32bits e possui arquitetura RISC. Esse processador foi especialmente desenvolvido com instruções específicas para processamento de imagens e apresenta alto desempenho. Este processador foi utilizado, por exemplo, em consoles de vídeo game como o Sega Dreamcast.

A câmera Legend 544 possui um ambiente de controle e desenvolvimento chamado “Intellect”. Nesse ambiente é possível ajustar todos os parâmetros da câmera, adquirir imagens e implementar programas. Contudo, o software “Intellect” da câmera não permite programar funções essenciais para a implementação a reconstrução tridimensional das imagens. Assim, optou-se por utilizar a câmera Legend apenas para a função de aquisição da imagem.

Para o controle de aquisição de imagens pela câmera Legend foi implementado um servidor TCP/IP na câmera. Esse servidor é capaz de receber comandos para a aquisição da imagem de um computador externo, comandar a aquisição de imagens pela câmera e enviar as imagens via internet para o computador externo, sem o uso do programa “Intellect”.

7.2 Implementação em hardware dedicado

Tendo em vista que o algoritmo de auto-localização e mapeamento simultâneo requer grande capacidade de processamento, procurou-se por processadores com capacidade de processamento equivalente à dos computadores pessoais tradicionais. Pensando-se em computadores embarcados, um hardware de arquitetura PC104 seria a escolha mais lógica, entretanto, esse tipo de hardware com o poder computacional desejado possui um custo muito elevado. Assim, optou-se por uma escolha de mais baixo custo, mas não muito usual.

A tecnologia de “motherboard” para computadores pessoais se desenvolveu muito nos últimos anos. Hoje em dia é possível encontrar placas mãe com todos os periféricos embarcados e com o tamanho padrão micro ATX (244 mm por 220 mm) compatível com a presente aplicação. A escolha de uma placa de computador pessoal permite a utilização de processadores potentes com grande quantidade de memória RAM.

A memória tipo “flash” em conjunto com a tecnologia USB permite o armazenamento de alta quantidade de informação sem a utilização de discos rígidos convencionais e conseqüentemente com grande economia de energia. Isso é interessante tendo em vista um sistema embarcado.

Um circuito de alimentação deve ser especialmente desenvolvido para a placa mãe, utilizando baterias, sendo então possível utilizar um computador com poder computacional compatível com os computadores pessoais convencionais e ainda ter uma grande economia de energia. Esse hardware, além da placa mãe com o microprocessador e memória RAM, pode utilizar apenas um dispositivo USB com sua memória carregada com o sistema operacional apropriado.

A placa mãe selecionada foi o modelo PcChips M871G microATX, equipada com o microprocessador AMD Sempron de 2,8 GHz, com tecnologia híbrida de 64bits. A placa possui 1GB de memória RAM, modelo DDR400 da marca Kingston. Para a alimentação do sistema são utilizadas baterias com tensão de 12V e diferentes capacidades para atender as demandas de energia das diversas partes do sistema.

7.3 Sistema operacional

Na seleção do sistema operacional para o sistema embarcado pode-se escolher entre um sistema operacional convencional ou um voltado para tempo real atendendo a

normalização POSIX. Optou-se por selecionar um sistema operacional convencional em razão da necessidade de tempo para o aprendizado do software e facilidade de operação. Outra escolha a ser feita é se o sistema operacional será baseado em uma plataforma Microsoft Windows (Windows XP, Windows CE, Windows NT) ou UNIX (Real Time LINUX, QNX, IRIX). O principal fator a ser considerado é a facilidade de desenvolvimento e custo. Foi escolhida a plataforma Windows, que dispõem de uma documentação muito extensa e que está disponibilizada gratuitamente para fins acadêmicos. Tendo em vista que uma memória com interface USB será utilizada para armazenar o sistema operacional, chegou-se à conclusão de que o sistema teria que ser desenvolvido. Com essas definições, as escolhas possíveis seriam o JAVA SE, Windows CE ou o Windows XP Embedded. O JAVA SE e o Windows CE são plataformas presentes em vários produtos da vida cotidiana, como computadores de mão e celulares, entretanto não oferecem suporte para processadores mais complexos, pois estes fogem do foco de sistemas embarcados comerciais. Sendo assim, a única opção possível é a utilização do Windows XP Embedded.

O Windows XP Embedded é composto pelo núcleo do Windows XP e é totalmente modularizado, ou seja, ele gera um sistema operacional que se adequa ao espaço disponível e pode ser instalado apenas com os componentes que o usuário necessita em seu sistema operacional. Por ser essencialmente um Windows XP, ele executa qualquer programa desenvolvido para esse sistema operacional, facilitando a instalação de “drivers” e componentes para qualquer equipamento.

Foi desenvolvido um sistema operacional que ocupa apenas 100MB e é capaz de realizar as tarefas necessárias para a presente aplicação, incluindo a interface de rede TCP/IP.

7.4 Programação do algoritmo de auto-localização e mapeamento simultâneos

Uma vez escolhidos o hardware e o sistema operacional resta escolher a linguagem de programação e a plataforma de desenvolvimento onde será implementada a programação do algoritmo de auto-localização e mapeamento simultâneos. As opções são JAVA, C ou C++. Optou-se por utilizar C++ por ser uma linguagem com capacidade de acessar diretamente o hardware, apresentar uma vasta documentação, ser orientada a objetos e ser de conhecimento prévio do programador.

Como plataforma de desenvolvimento optou-se inicialmente pelo DEV C++ da BloodShed, que é um software gratuito sob a licença GNU, e posteriormente pelo Microsoft Visual Studio 2005. A necessidade de mudança da plataforma de desenvolvimento ocorreu em virtude da necessidade de se ter uma ferramenta mais complexas de análise e otimização do software, detecção de vazamento de memória e maior quantidade de documentação.

A programação do algoritmo de auto-localização e construção de mapa gerou um código relativamente extenso, possuindo mais de dez mil linhas. Esse programa é composto por sete bibliotecas especialmente desenvolvidas para a presente aplicação. Por uma questão de espaço o código fonte não está anexado nesse trabalho. As principais características do algoritmo implementado são apresentadas a seguir.

Os nomes das classes utilizadas no programa são: “Imagem”, “Analise”, “FFT”, “CConfigclass”, “Tricp”, “MapaOdometria” e “LOG”.

A classe “FFT” implementa os algoritmos de Transformada Rápida de Fourier em duas dimensões e sua transformada inversa. Essas transformadas são utilizadas opcionalmente para o cálculo de convoluções.

A classe “Imagem” é responsável por todo o processamento da imagem omnidirecional adquirida pelo sistema de visão. Esta classe utiliza a classe “FFT”, quando necessário, para realizar as convoluções do algoritmo Canny. Sua sequência de execução é a seguinte:

1. Ler imagem;
2. Transformar a imagem omnidirecional em retangular;
3. Detectar as bordas na imagem com o algoritmo de Canny (1986);
4. Selecionar as bordas mais próximas do sistema de aquisição, ou seja, as bordas mais próximas da parte de baixo das imagens interna e externa;
5. Encontrar os pontos das bordas que sejam máximos locais.

A mudança de coordenadas que transforma a imagem omnidirecional, em coordenadas polares, para a imagem retangular, em coordenadas cartesianas, é realizada por meio de uma matriz Jacobiano (Souza Jr., 2004). O cálculo dessa matriz é realizado apenas uma vez durante toda a execução do programa e ela é utilizada no processamento de todas as imagens adquiridas.

A classe “Analise” é responsável pela correspondência de pontos das imagens interna e externa do espelho e pela reconstrução bidimensional. Essa classe utiliza a classe “Imagem” para realizar o processamento das imagens. A sequência de execução dessa classe é a seguinte:

1. Correspondência;
2. Reconstrução bidimensional.

A correspondência é realizada utilizando o método de programação dinâmica desenvolvido por Fielding e Kam (2000). Na correspondência utilizam-se as posições de máximo local das bordas nas imagens, porém verifica-se a similaridade entre os possíveis pontos correspondentes com as imagens retangulares originais, conforme descrito por Sousa Jr. (2004).

A classe “Tricp” é responsável pela implementação do algoritmo TrICP utilizado na correspondência dos mapas locais com o mapa global. Os mapas locais consistem em mapas de partes do ambiente, que são os resultados da reconstrução bidimensional das imagens interna e externa do espelho.

A classe “Mapaodometria” utiliza objetos das classes “Tricp” e “Analise” para implementar o algoritmo de auto-localização e construção simultânea de mapa, apresentados nos itens 4.3 e 4.4.

A classe “CConfigclass” é utilizada para encapsular todas as variáveis de configuração do algoritmo de auto-localização e mapeamento simultâneos.

A classe “LOG” é utilizada para manter um registro das execuções das principais atividades do programa e os seus tempos de processamento.

O programa principal simplesmente controla o fluxo das imagens que devem ser integradas ao mapa global, chamando a classe “MapaOdometria”, que por sua vez chama todas as outras classes apresentadas.

7.5 Sumário

O algoritmo de auto-localização e mapeamento simultâneos com odometria foi codificado em C++, possibilitando a sua utilização como software ser embarcado em um robô

móvel autônomo. As tentativas de pré-processar a imagem diretamente na câmera Legend não foram bem sucedidas devido às limitações do software da mesma.

O algoritmo será implementado em um computador pessoal quase sem nenhum periférico, consistindo apenas da placa mãe, de processador e de memória RAM. No lugar do disco rígido será utilizado um cartão de memória USB com capacidade suficiente para carregar o sistema operacional escolhido. O sistema operacional que melhor se adapta à aplicação é o Windows XP Embedded, que consiste essencialmente no núcleo do Windows XP com somente as bibliotecas adicionais necessárias para fazer com que o programa funcione corretamente.

O desenvolvimento do software foi realizado em dois ambientes, inicialmente no DEV C++ da BloodShed e posteriormente no Visual Studio 2005. Em razão das ferramentas existentes no Visual Studio 2005 foi possível realizar uma otimização e uma análise precisa do extenso código, assim como, a remoção de vazamento de memória do código originalmente concebido no DEV C++.

8 RESULTADOS DO ALGORITMO EM C++

Esse capítulo apresenta os resultados do sistema implementado em C++ no hardware escolhido em condições reais de operação. Ressalta-se que para verificar a eficácia do sistema não é necessário realizar a navegação de um robô para a aquisição das imagens, basta adquirir as imagens em pontos distintos no ambiente.

8.1 Resultados do algoritmo de auto-localização e mapeamento simultâneos com odometria em ambiente simulado

A figura 8.1 apresenta o resultado da sequência de imagens espaçadas de vinte cinco centímetros, na ordem direta, ou seja, do ponto 1 ao 37, processadas pelo algoritmo de auto-localização e mapeamento simultâneos com odometria implementado em C++. Nessa figura os pontos representam as bordas de objetos no ambiente e as estrelas representam os pontos de aquisição de imagens calculadas pelo algoritmo durante a trajetória.

Figura 8.1 – Mapa do ambiente reconstruído a partir das 37 imagens sequenciadas na ordem direta obtido com o algoritmo com odometria em C++. Os números em vermelho correspondem aos erros de reconstrução das respectivas arestas.

Quanto aos resultados de reconstrução do ambiente, observam-se os seguintes erros de reconstrução: na arresta (1) de -10 cm; na arresta (2) de -30 cm; na arresta (3) de 20 cm; na arresta (4) é imperceptível; na arresta (5) de 50 cm; na arresta (6) de 10 cm; na arresta (7) de -70 cm; na arresta (8) de -10 cm; na arresta (9) é imperceptível; e na arresta (10) de -10 cm. O erro total no comprimento da sala reconstruída, medido no eixo x , é de 10 cm em 8 m, representando um erro percentual de $1,25\%$, e medido no eixo y , é de aproximadamente -20 cm em 4 m, representando um erro percentual de 5% . O erro maior na direção y ocorre em razão de um pequeno desalinhamento entre as arresta esquerda e direita.

A figura 8.2 apresenta a evolução do erro quadrático relativo percentual ao longo da incorporação das informações de novas imagens pelo algoritmo de auto-localização e mapeamento simultâneos com odometria em C++. É possível notar que apenas a imagem 9 não foi integrada ao mapa global.

Figura 8.2 – Evolução do erro quadrático relativo percentual obtido com o algoritmo com odometria em C++ no ambiente simulado.

A figura 8.3 apresenta em detalhes a trajetória simulada, a calculada e a obtida utilizando-se apenas a odometria sem nenhuma correção. Pode-se observar claramente que o algoritmo não tem dificuldade de seguir a trajetória correta, apresentando somente um

pequeno erro. Além disso, observa-se que a trajetória calculada é mais precisa do que a trajetória obtida através da odometria na maior parte da trajetória.

Figura 8.3 – Trajetórias simulada, calculada pelo algoritmo com odometria em C++ e calculada pela odometria no ambiente simulado.

As figuras 8.4 e 8.5 mostram respectivamente o erro percentual em relação à trajetória navegada e o erro absoluto em relação à posição simulada. Na figura 8.4 nota-se que o erro total de localização do robô converge para um valor em torno de 0,5%. Nota-se que o erro relativo no eixo x inicia-se na imagem 10 porque é nessa imagem que ocorre a primeira translação nesse eixo. Além do pico inicial para o erro de auto-localização na direção y , notam-se apenas pequenos erros. Com relação à figura 8.5 pode-se notar o mesmo comportamento da figura 8.4. O erro total de localização atinge um máximo de 83 mm para a imagem 25. Finalmente, nota-se que o erro oscila no intervalo de 45 mm a 85 mm e que os dois eixos contribuem igualmente para o erro total.

Figura 8.4 – Desvio percentual do erro de localização para o algoritmo em C++ com odometria no ambiente simulado.

Figura 8.5 – Erro absoluto na localização do robô para o algoritmo com odometria em C++ no ambiente simulado.

A figura 8.6 apresenta o número de iterações necessárias para a convergência da correspondência no TrICP. Observa-se que uma média de 4 iterações são necessárias para a incorporação das informações de cada imagem ao mapa global. Este gráfico reflete a dificuldade do TrICP lidar com cada nova imagem adquirida. Ressalta-se que o número máximo de 30 iterações não foi alcançado em nenhuma das imagens.

Figura 8.6 – Número de iterações necessárias para incorporar as informações de cada imagem na construção do mapa utilizando o algoritmo com odometria em C++ no ambiente simulado.

8.2 Resultados do algoritmo de auto-localização e mapeamento simultâneos com odometria em ambiente real

O algoritmo de auto-localização e mapeamento simultâneos implementado em C++ é também testado no ambiente real. Os testes são realizados para o conjunto de 59 imagens apresentadas no Capítulo 6 na trajetória direta, ou seja, a trajetória que inicia na imagem 1 e segue crescentemente até retornar à imagem 1 novamente, dando uma e duas voltas completas na sala.

8.2.1 Resultados para uma volta completa

A figura 8.7 apresenta o resultado da sequência de imagens reais processadas pelo algoritmo de auto-localização e mapeamento simultâneo com odometria implementado em C++. Nessa figura os pontos representam as bordas de objetos no ambiente e as estrelas representam as posições do robô calculadas durante a trajetória.

Pelo figura 8.7 observam-se pequenos erros de reconstrução, sendo que o erro de reconstrução na arresta (1) é de -10 cm; na arresta (2) é de -20 cm; na arresta (3) é de -10 cm; na arresta (4) é de 10 cm; na arresta (5) é de -15 cm; na arresta (6) é imperceptível; na arresta (7) é de 5 cm; e na arresta (8) é imperceptível. O erro total no comprimento da sala reconstruída, medido no eixo x , é -10 cm na borda superior e o erro medido no eixo y é também de -10 cm.

Figura 8.7 – Mapa do ambiente reconstruído a partir das 60 imagens sequenciadas na ordem direta obtido com o algoritmo com odometria em C++. Valores em vermelho consistem nos erros de reconstrução das arestas.

A coordenada $(0, 0)$ no mapa representa a última posição do robô calculada pelo algoritmo de auto-localização. Essa posição está coerente com a posição final do robô no ambiente. Observa-se que o erro final de posição do robô, ou seja, o erro resultante na

imagem 60 é de 280 mm. Esse erro representa o erro de posicionamento total do robô após percorrer uma volta completa na sala. Nota-se que o algoritmo é capaz de reconstruir o ambiente de modo correto e com as paredes do ambiente razoavelmente bem definidas, porém nota-se que a maior parte das quinas não estão conectadas e algumas paredes apresentam um formato ondulado.

A figura 8.8 apresenta a evolução do erro quadrático relativo percentual ao longo da incorporação das informações de novas imagens pelo algoritmo de auto-localização e mapeamento simultâneos. Nota-se que do total de 60 imagens, 19 imagens não foram integradas ao mapa global. Lembra-se que nesse algoritmo, após uma tentativa mal sucedida para incluir a reconstrução da imagem no mapa global, o Erro Quadrático Relativo Percentual é flexibilizado. Assim, as imagens 19, 26, 30 e 44 apesar de apresentarem erros acima da linha limite de 30% também tiveram a sua reconstrução adicionada ao mapa global.

A figura 8.9 apresenta em detalhes a trajetória simulada, a calculada e a obtida utilizando-se apenas a odometria. Pode-se observar claramente que o algoritmo não tem dificuldade de seguir a trajetória correta, entretanto, alguns pontos apresentam um erro considerável. Observa-se que a trajetória calculada é mais precisa do que a trajetória obtida por meio da odometria na maior parte da trajetória.

Figura 8.8 – Evolução do erro quadrático relativo percentual obtido com o algoritmo com odometria em C++ no ambiente real.

Figura 8.9 – Trajetórias simulada, calculada pelo algoritmo com odometria em C++ e calculada pela odometria no ambiente real.

As figuras 8.10 e 8.11 mostram respectivamente o erro percentual de localização do robô em relação à trajetória navegada e o erro absoluto em relação à posição real. Na figura 8.10 nota-se que o erro total converge para um valor em torno de 1% e apresenta no final um pequeno aumento terminando em 1,5%. Observa-se que o erro de localização do robô no eixo x inicia-se na imagem 18, pois é nessa imagem que ocorre a primeira translação nesse eixo. Além do pico inicial para o erro na direção x , notam-se apenas pequenos erros. Com relação à figura 8.11 pode-se notar que o erro total de localização do robô atinge um máximo de 280 mm para a imagem 60. Observa-se que o erro possui um comportamento oscilatório no intervalo de 75mm a 175mm até a imagem 54, quando então ocorre um aumento. Nota-se que o erro na direção x representa praticamente o erro total.

A figura 8.12 apresenta o número de iterações necessárias para a convergência da correspondência no TrICP na integração da informação de cada uma das imagens adquiridas ao longo da trajetória. Observa-se uma média de 7 iterações para a integração de cada mapa local ao mapa global. Esse gráfico também reflete a dificuldade do TrICP incorporar as informações de cada nova imagem adquirida. Ressalta-se que o número máximo de 30 iterações não foi alcançado para nenhuma das imagens adquiridas.

Figura 8.10 – Desvio percentual do erro de localização para o algoritmo com odometria em C++ no ambiente real.

Figura 8.11 – Erro absoluto na localização do robô para o algoritmo com odometria em C++ no ambiente real

Figura 8.12 – Número de iterações na construção de mapas para o algoritmo com odometria em C++ no ambiente real.

8.2.2 Resultados para duas voltas completas

A figura 8.13 apresenta o resultado da sequência de imagens reais processadas com duas voltas no ambiente pelo algoritmo de auto-localização e mapeamento simultâneos com odometria implementado em C++. Nessa figura os pontos representam as bordas de objetos no ambiente e as estrelas representam as posições do robô calculadas durante a trajetória. As estrelas de cor escuras são referentes às localizações do robô na primeira volta e as estrelas de cor clara são referentes às localizações do robô na segunda volta.

Figura 8.13 – Mapa do ambiente reconstruído a partir das 60 imagens com duas voltas no ambiente seqüenciadas na ordem direta obtido utilizando o algoritmo com odometria em C++.

Para esse caso não será feita uma análise detalhada dos erros da reconstrução. Nota-se apenas que o algoritmo foi bem sucedido em fazer o mapeamento do ambiente nas duas voltas e que o mapa global é composto claramente por dois mapas sobrepostos, sendo que o mapa referente à segunda volta está ligeiramente transladado para direita em relação ao mapa construído na primeira volta.

A coordenada (0, 0) no mapa representa a última posição do robô calculada pelo algoritmo de auto-localização. Essa posição está coerente com a posição final do robô no ambiente. Observa-se que o erro final na localização do robô, obtido na imagem 118, é de 330 mm. Esse erro representa o erro de posicionamento total do robô após percorrer duas voltas completas no ambiente. Nota-se que o algoritmo é capaz de reconstruir o ambiente de modo correto e com as paredes do ambiente razoavelmente bem definidas.

A figura 8.14 apresenta a evolução do erro quadrático relativo percentual ao longo da incorporação das informações de novas imagens pelo algoritmo de auto-localização e mapeamento simultâneos com odometria. Nota-se que de um total de 118 imagens, 31 imagens não foram integradas ao mapa global, sendo 15 na primeira volta e 16 na segunda

volta. Lembrando que nesse algoritmo, após uma tentativa mal sucedida para incluir a reconstrução da imagem no mapa global, o Erro Quadrático Relativo Percentual é flexibilizado. Assim, observa-se que as imagens 16, 45, 47, 104(45) apesar de estarem situadas acima da linha limite de 30% também tiveram a sua reconstrução incorporada ao mapa global. O número entre parentes refere-se à contagem original das imagens.

Figura 8.14 – Evolução do erro quadrático relativo percentual obtido com o algoritmo com odometria em C++ no ambiente real com duas voltas no ambiente.

A figura 8.15 apresenta em detalhes a trajetória simulada, a calculada e a obtida utilizando-se apenas a odometria. Pode-se observar claramente que o algoritmo não tem dificuldade de seguir a trajetória correta, entretanto, alguns pontos apresentam um erro considerável. Observa-se que a trajetória calculada é mais precisa do que a trajetória obtida através da odometria na maior parte da trajetória. Nota-se também que de forma geral a localização do robô calculada na primeira volta está mais precisa na maior parte do percurso do que a calculada na segunda volta. Isso ocorre porque em razão dos erros acumulados de localização do robô.

Figura 8.15 – Trajetórias simulada, calculada pelo algoritmo com odometria em C++ e calculada pela odometria no ambiente real com duas voltas no ambiente.

As figuras 8.16 e 8.17 apresentam respectivamente o erro percentual de localização do robô em relação à trajetória navegada e o erro absoluto em relação à posição real. Na figura 8.16 nota-se que o erro total converge para um valor em torno de 1%. Observa-se que o erro relativo no eixo x inicia-se na imagem 18, pois é nessa imagem que ocorre a primeira translação nesse eixo. Além do pico inicial para o erro na direção x , notam-se apenas pequenos erros para a localização do robô. Com relação à figura 8.17 pode-se notar que o erro total de localização do robô atinge um máximo de 430 mm para a imagem 80. Observa-se que o erro possui um comportamento oscilatório no intervalo de 100 mm a 250 mm na primeira volta e de 200mm a 450mm na segunda volta. Nesse caso, os erros nas direções x e y contribuem de forma equivalente para o erro total. É possível notar um padrão no aumento e diminuição do erro durante uma volta; na primeira volta o erro inicialmente cresce até a imagem 19, decresce até a imagem 44 e finalmente cresce até a imagem 59; na segunda volta o erro cresce até a imagem 80(21), decresce até a imagem 102(43) e depois cresce até o final do percurso. O número entre parenteses refere-se à contagem original das imagens.

A figura 8.18 apresenta o número de iterações necessárias para a convergência da correspondência no TrICP na incorporação da informação de cada uma das imagens

adquiridas ao longo da trajetória. Observa-se uma média de 8 iterações para a integração de cada mapa local ao mapa global, semelhante ao caso de apenas uma volta. Esse gráfico reflete a dificuldade do TrICP incorporar as informações de cada nova imagem adquirida. Ressalta-se que o número máximo de 30 iterações não foi alcançado para nenhuma das imagens adquiridas.

Figura 8.16 – Desvio percentual do erro de localização para o algoritmo com odometria em C++ no ambiente real com dois ciclos.

Figura 8.17 – Erro absoluto na localização do robô para o algoritmo com odometria em C++ no ambiente real com dois ciclos.

Figura 8.18 – Número de iterações na construção de mapas para o algoritmo com odometria no ambiente real com dois ciclos.

8.3 Análise do tempo de processamento do algoritmo de auto-localização e mapeamento simultâneos (SLAM) com odometria em C++ no ambiente real

Como mencionado nos objetivos do trabalho deseja-se desenvolver um algoritmo de SLAM que possa ser utilizado em uma aplicação embarcada. Nessa Seção são apresentados resultados do tempo de processamento médio de algumas funções do algoritmo de auto-localização e mapeamento simultâneos com odometria em C++ no ambiente real.

A figura 8.19 mostra o tempo de execução de cada uma das 60 iterações na construção do mapa global do ambiente real. O tempo da iteração é dividido em três etapas principais de processamento, sendo essas: o processamento completo da imagem omnidirecional, descrito no Capítulo 2, incluindo a reconstrução bidimensional da imagem; o processo de correspondência do Trimmed ICP; e a atualização do mapa global em cada iteração.

Figura 8.19 – Análise do tempo de processamento por iteração do algoritmo com odometria em C++ no ambiente real.

Nota-se que a etapa mais exigente em termos computacionais é a de processamento da imagem que em certos casos ocupa 95% do tempo da iteração. Observa-se que o TriCP é extremamente rápido ocupando poucos milissegundos em cada iteração para fazer a correspondência e alinhamento dos mapas. A etapa de integração é mostrada pelo tempo de geração do mapa global resultante de cada iteração. O tempo médio de processamento de uma imagem é de 2,5 segundos.

A figura 8.20 apresenta o tempo computacional de cada uma das etapas de processamento das imagens. O processamento das imagens é a etapa que mais contribui no tempo total de processamento do algoritmo, como evidenciado pela figura 8.19. As etapas mostradas nesta análise são:

- 1) Leitura do arquivo da imagem, transformação da imagem de coordenadas polar para cartesianas e gravação intermediária da imagem resultante desse processamento;
- 2) Detecção de bordas nas imagens interna e externa em coordenadas cartesianas pelo algoritmo de Canny;
- 3) Pós-processamento das imagens com as bordas detectadas e cálculo da Soma das Diferenças Absoluta (SDA);
- 4) Correspondência dos pontos de bordas das imagens interna e externa utilizando o algoritmo de programação dinâmica;
- 5) Construção do mapa local por meio da triangulação dos pontos correspondidos nas imagens interna e externa.

Nota-se que as duas primeiras etapas gastam aproximadamente dois terços do tempo total de processamento completo das imagens e que o tempo gasto para construção do mapa local do ambiente é da ordem de poucos milissegundos.

Figura 8.20 – Análise do tempo de processamento da imagem omnidirecional no algoritmo com odometria em C++ no ambiente real.

A figura 8.21 em formato de pizza mostra o tempo de processamento das etapas separadas de acordo com as figura 8.19 e 8.20 e a contribuição percentual de cada etapa no tempo total de processamento das 60 imagens de uma volta completa no ambiente real. Pelos resultados é possível observar que as duas primeiras etapas do processamento da imagem gastam em média 60% do tempo total de execução do processamento para auto-localização e mapeamento simultâneos no ambiente real. O tempo de execução total médio é de cerca de 155 segundos, resultando em uma média de 2,6 segundos por imagem.

Figura 8.21 – Tempo de execução para cada etapa do algoritmo de auto-localização e mapeamento com odometria em C++. Valores em segundos.

8.4 Sumário

Nesse capítulo foram mostrados todos os resultados relevantes do algoritmo de auto-localização e mapeamento simultâneos com odometria implementado em C++. Os resultados são mostrados tanto para o ambiente simulado quanto para o ambiente real com uma e duas voltas completas.

No caso do ambiente simulado o mapa global final resultante é de excelente qualidade, com as bordas do ambiente muito bem definidas e com erros pequenos, semelhante aos resultados obtidos com o algoritmo implementado no MatLab. Um fator muito importante a ser observado é a diferença no tempo de processamento, que para o algoritmo implementado em C++ diminuiu em aproximadamente duas vezes em relação ao algoritmo implementado no MatLab.

Os resultados mais importantes são os obtidos no ambiente real, pois mostram uma aplicação prática real do sistema desenvolvido. No caso de uma volta no ambiente real os resultados tanto da auto-localização quanto do mapa global final são de excelente qualidade. A localização do robô calculada pelo algoritmo foi notadamente muito próxima da posição real ao longo da trajetória e a qualidade do mapa global final é muito boa, com as bordas muito bem definidas com baixíssimos erros de reconstrução. Os erros encontrados são muito próximos dos erros do mesmo algoritmo implementado em MatLab.

Para o caso de duas voltas completas no ambiente real os resultados são análogos aos obtidos com uma volta completa. Nota-se um comportamento de sobreposição de dois mapas globais causado pelo fato do algoritmo de construção de mapa do ambiente trabalhar de forma incremental na incorporação das informações dos mapas locais sequenciais. Os resultados desse algoritmo em C++ são muito semelhantes aos encontrados na sua implementação em MatLab.

Os tempos de processamento do algoritmo de auto-localização e mapeamento simultâneos com odometria em C++ foram determinados por meio de uma série de execuções do algoritmo. A maior parte do tempo gasto pela execução do algoritmo é gasto no processamento da imagem. Das tarefas envolvidas no processamento das imagens, as que ocupam a maior parte do tempo são: a leitura do arquivo da imagem, a mudança de coordenadas da imagem de polar para cartesiana e a detecção de bordas pelo o algoritmo de Canny. Juntas essas etapas consistem em cerca de 60% do tempo total de execução.

9 CONCLUSÕES

Com relação ao algoritmo de correspondência e alinhamento dos mapas locais, o TrICP se mostrou robusto para pequenas translações e rotações. O principal ponto fraco do algoritmo é a possibilidade de ficar preso em um ponto de mínimo local durante o processo iterativo de obter a transformação de coordenadas que transforma o conjunto de pontos da amostra para o sistema de coordenadas do conjunto de pontos do modelo. Outro problema apresentado pelo TrICP é a dificuldade de se obter uma estimativa do erro de posição do robô durante o cálculo da auto-localização que é realizado no processo de correspondência entre mapa global e local. Entretanto pode-se perceber através dos resultados que o erro de localização se mantém dentro de um intervalo aceitável.

Considera-se que as alterações implantadas no algoritmo de auto-localização e mapeamento simultâneos em relação ao algoritmo desenvolvido inicialmente por Oliveira (2005) foram essenciais para os bons resultados apresentados. Essas alterações consistiram basicamente em: 1) usar somente as bordas dos objetos mais próximos do sistema de visão; e 2) utilizar a informação da odometria para iniciar o processo iterativo de cálculo da translação e rotação na correspondência entre os mapas locais.

O tempo de processamento do algoritmo com odometria implementado em C++ é cerca de nove vezes menor do que o tempo de processamento do algoritmo implementado no MatLab. O pequeno tempo de processamento permite que o algoritmo desenvolvido nesse trabalho seja executado em tempo real com uma taxa de aquisição de imagens suficientemente rápida para ser embarcado em um robô móvel.

Os testes do algoritmo desenvolvido para auto-localização e mapeamento simultâneos com odometria realizados no MatLab e posteriormente com o algoritmo implementado em C++, tanto para imagens simuladas como para imagens reais, apresentaram resultados de alta qualidade. Esses resultados permitem concluir que o algoritmo é satisfatório para ser utilizado embarcado em um robô móvel para auxiliar na navegação do mesmo em ambientes desconhecidos.

9.1 Considerações finais

O algoritmo final proposto para resolver o problema de SLAM foi bem sucedido tanto nos testes com ambiente simulado quanto nos testes em ambiente real. A capacidade do algoritmo está no fato de utilizar a odometria como estimativa inicial para a transformação de coordenadas entre os mapas locais obtidos das imagens em sequência. Observa-se que a incorporação da odometria faz com que o algoritmo proposto seja extremamente robusto.

A flexibilização no valor do Erro Quadrático Relativo Percentual para aceitar transformações de coordenadas também desempenha um papel importante no sucesso do algoritmo desenvolvido. Essa flexibilização é na verdade um voto de confiança no algoritmo TrICP, permitindo aceitar algumas transformações que a princípio não seriam aceitas. Isso também contribui para que o mapa global não apresente descontinuidades em áreas muito complexas e não permite que o erro da odometria, que é utilizada como estimativa inicial no algoritmo TrICP, se acumule muito.

Finalmente, pode-se concluir que os resultados obtidos com o algoritmo proposto são de qualidade excelente, apresentando erros de reconstrução muito pequenos em relação ao tamanho do ambiente e ao comprimento total das trajetórias testadas.

9.2 Trabalhos futuros

Dado o sucesso do algoritmo proposto um provável trabalho futuro seria a sua implantação em um robô móvel autônomo real. O algoritmo de SLAM desenvolvido pode trabalhar em conjunto com outras tarefas do robô, realizando, por exemplo, planejamento de trajetória e detecção de objetos fixos.

Para aprimorar o algoritmo de SLAM desenvolvido e melhorar a sua capacidade para ser utilizado em um robô móvel, algumas melhorias ainda podem ser feitas:

- Implantar o algoritmo em um sistema operacional em tempo real. Para isso é necessário a reprogramação dos métodos e classes desenvolvidas para *threads*.
- Aplicar ao software desenvolvido otimizações de métodos e classes, aumento das opções do arquivo de configuração e etc.

- Implantação de reconhecimento de revisita no mesmo ambiente, evitando que o comportamento mostrado nos teste com mais de uma volta ocorra. Nesse caso, deve-se garantir que a adição de informação de uma revisita no ambiente gere pontos com maior confiabilidade e não apenas novos pontos e pontos duplos.
- Obter uma estimativa para o erro da reconstrução a partir dos dados de saída do algoritmo TrICP. Com base nessa estimativa é possível utilizar filtros preditivos para melhorar ainda mais a estimativa inicial proposta pela odometria e conseqüentemente melhorar ainda mais a qualidade do mapa global final apresentado.
- Filtrar os pontos que consistem de ruídos também pode ser aplicado no algoritmo. Entretanto ressalta-se que isso pode aumentar consideravelmente o tempo de processamento, pois o número de pontos no mapa global final pode facilmente atingir cinco mil.

10 BIBLIOGRAFIA

- Baker e Nayar (1998).** S. Baker e S. K. Nayar, “A theory of catadioptric image formation”, In *Proc. of the 6th Int. Conf. on Computer Vision - ICCV’98*, pp. 35-42, 1998.
- Besl e McKay (1992).** Besl, P. J. e McKay, N. D., “A Method for Registration of 3-D Shapes”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14(2), pp.239-256, 1992.
- Byrne et al. (1992).** Byrne, R.H.; Klarer, P.R.; Pletta, J.B., “Techniques for Autonomous Navigation.” *Sandia Report SAND92-0457*, Albuquerque, Março 1992.
- Borenstein et al. (1996).** Borenstein, J.; Everett, H. R.; Feng, L. “Sensors and Methods for Mobile Robot Positioning”, *para o United States Department of Energy*, 1996.
- Borenstein et al. (1997).** Borenstein, J.; Everett, H. R.; Feng, L.; Wehe, D. “Mobile Robot Positioning – Sensors and Techniques”, *Journal of Robotic Systems*, vol. 14, nº4, pp. 231-249, 1997.
- Brooks (1986).** Brooks, R. A., “A Robust Layared Control System for mobile Robot”, *IEEE Journal of Robotics and Automation*, vol. 2, nº1, pp. 14-23, 1986.
- Cabral et al. (2004).** Cabral, E. L. L.; Junior, J. C. S.; Hunold, M. C., “Omnidirectional Stereo Vision with a Hyperbolic Double Lobed Mirror”, *ICPR 2004*.
- Cabral et al. (2007)** Cabral, E. L. L.; Oliveira, P. R. G.; Souza Jr., J. C., “An Omnidirectional Stereo Vision System”, *in procceding of COBEM’07*, Brasil, Brasília, 2007.
- Canny (1986).** Canny, J. “A Computational Approach to Edge Detection”, *IEEE Transactions on Pattern Analysis and Machine Intelligence archive*, Volume 8 , Issue 6, Novembro, 1986.
- Chahl e Srinivasan (1997).** J. S. Chahl e M. V. Srinivasan, “Reflective surfaces for panoramic imaging”, *Applied Optics*, 30(31):8275-8282, 1997.
- Chenavier e Crowley (1992).** Chenavier, F. e Crowley, J., “Position Estimation for a Mobile Robot Using Vision and Odometry.” *Proceedings of IEEE International Conference on Robotics and Automation*, Nice, França, 12-14, pp. 2588-2593, Maio 1992.
- Chetverikov et al. (2002a).** Chetverikov, D.; Svirko, D.; Stepanov, D.; Krsek, P., “The Trimmed Iterative Closest Point Algorithm”, *Proceeding of ICPR’02*, Québec, 2002.

- Chetverikov e Stepanov (2002b).** Chetverikov, D.; Stepanov, D., “Robust Euclidean Alignment of 3D Point Sets”, *First Hungarian Conference on Computer Graphics and Geometry*, Budapest, pp.70-75, 2002.
- Chetverikov et al. (2005).** Chetverikov, D.; Stepanov, D.; Krsek, P. “Robust Euclidian Alignment of 3D Point Sets: The Trimmed Iterative Closest Point Algorithm”, *Image and Vision Computing*, 2005.
- Conroy e Moore (1999).** T. L. Conroy e J. B. Moore, “Resolution invariant surfaces for panoramic vision systems”, *In Proc. of the 7th Int. Conf. on Computer Vision - ICCV’99*, pp. 392-397, 1999.
- Cozman e Krotkov (1997).** Cozman, F.; Krotkov, E., “Automatic mountain detection and pose estimation for teleoperation of lunar rovers,” *in Proc. IEEE Int. Conf. Robot. Automat.*, 1997.
- Deccó (2004).** Deccó, C. C. G., “Construção de Mapas de Ambiente para Navegação de Robos Móveis com Visão Omnidirecional Estéreo”, tese de doutorado, 2004.
- Desouza e Kak (2002).** Desouza, G. N.; Kak, A. C., “Vision for Móbile Robot Navigation: A Sursey”, *IEEE Transactions of Pattern Analysis and Machine Intelligence*, vol. 24, pp. 237-267, Fevereiro 2002.
- Dissanayake et al. (2001).** Dissanayake, M. W. M. G.; Newman, P.; Clark, S.; Durrant-Whyte, H. F., “A Solution to the Simultaneous Localization and Map Building (SLAM) Problem”, *IEEE Transactions on Robotics and Automation*, vol. 17, nº3, Junho 2001.
- Fielding e Kam (2000).** Fielding, G.; Kam, M., “Weighted matchings for dense stereo correspondence”, *Pattern recognition 33*, pp. 1511-1524, Junho de 1999.
- Filliat e Meyer (2003a).** Filliat, D.; Meyer, J. A., “Map-based navigation in mobile robots. I. A Review of localization strategies”, *Journal of Cognitive Systems Research*, 2003.
- Filliat e Meyer (2003b).** Filliat, D.; Meyer, J. A., “Map-based navigation in mobile robots. II. A review of map-learning and path-planning strategies”, *Journal of Cognitive Systems Research*, 2003.
- Gaspar et al. (2002).** Gaspar, J.; Decco, C. C. G.; Okamoto, J.; Santos-Victor, J., “Constant Resolution Omnidirectional Cameras”, *OMNIVIS’02 Workshop on Omni-directional Vision*, Copenhagen, Dinamarca, Junho 2002

- Geyer e Daniilidis (2000).** C. Geyer e K. Daniilidis, “A unifying theory for central panoramic systems and practical applications”, In *Proc. European Conf. on Computer Vision - ECCV'00*, pp. 445-461, Dublin, Irlanda, 2000.
- Giachetti et al. (1998).** Giachetti, A.; Campani, M.; Torre, V., “The Use of Optical Flow for Road Navigation”, *IEEE Transactions on Robotics and Automation*, vol. 14, n°1, pp. 34-48, 1998.
- Goldberg et al. (2002).** Goldberg, S. B.; Maimone, M. W.; Matthies, L., “Stereo Vision and Rover Navigation Software for Planetary Exploration”, *IEEE Aerospace Conference Proceedings*, vol. 5, pp. 2025-2036, Março 2002.
- Hähnel et al. (2003).** Hähnel, D.; Schulz, D.; Burgard, W., “Mapping with mobile robots in populated environments”, In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.
- Hanebeck e Schmidt (1996).** Hanebeck, U. D.; Schmidt, G., “Set theoretical localization of fast mobile robots using an angle measurement technique,” in *Proc. IEEE Int.Conf. Robot. Automat.*, pp. 1387–1394, Minneapolis, MN, Abril 1996.
- Hicks e Bajcsy (1999a).** R. A. Hicks e R. Bajcsy., “Catadioptric sensors that approximate wide-angle perspective projections”, In *Proc. of the IEEE 2nd Workshop on Omnidirectional Vision - OMNIVIS'00*, pg: 97-103, June 2000.
- Horn (1987).** Horn, B. K. P., “Closed-form solution of absolute orientation using unit Quaternions”, *Journal of the Optical Society of America A* 4, pp.629–642, 1987.
- Kuipers e Byun (1991).** Kuipers, B. J.; Byun, Y. T., “A robot exploration and mapping strategy based on semantic hierachy of spacial representation,” *J. Robot. Autonom.Syst.*, vol. 8, pp. 47–63, 1991.
- Leonard e Whyte (1992).** Leonard, J. e Durrant-Whyte, H., “Directed Sonar Sensing for Mobile Robot Navigation”, Boston, MA: Kluwer Academic, 1992.
- Levitt e Lawton (1990).** Levitt, T.S.; Lawton, D.T., “Qualitative navigation for mobile robots” *Artif. Intell. J.*, vol. 44, n°3, pp. 305–360, 1990.
- Manassis (2003).** Manassis, A., “Overview of Related work on Scene Modelling”.
http://www.dai.ed.ac.uk/Cvonline/LOCAL_COPIES/MANESSIS/liter.

- Mataric (2002).** Mataric, M. J., “Situated Robotics”, *Encyclopedia of Cognitive Science*, Nature Publishers Group, Macmillian Reference Ltd., 2002.
- Mouaddib et al. (2005).** Mouaddib, E. M.; Sagawa, R.; Echigo, T.; Yagi, Y. “Stereovision with a Single Camera and Multiple Mirrors”, in *Proceedings of the IEEE 2005*, Espanha, 2005.
- Nakamura e Ishiguro (2002).** Nakamura, T.; Ishiguro, H. “Automatic 2D Map Construction using a Special Catadioptric Sensor”, *Proceeding od the IEEE Int. Conference on Intelligent Robots and Systems*, Outubro 2002.
- Nene e Nayar (1998).** S. A. Nene and S. K. Nayar, “Stereo with mirrors”, In *Proc. of the 6th Int. Conf. on Computer Vision - ICCV’98*, Bombay, Índia, Janeiro 1998.
- Oliveira (2005).** Oliveira, P. R. G. “Construção de Mapas de Ambiente Baseados em Visão Omnidirecional Estéreo com Espelho Duplo de Perfil Hiperbólico”, *Trabalho de formatura para a obtenção do título de Engenheiro pela Escola Politécnica da USP*, São Paulo, 2005.
- Ollis et al. (1999).** M. Ollis, H. Herman e S. Singh, “Analysis and design of panoramic stereo vision using equi-angular cameras”, *The Robotics Institute, Carnegie Mellon University*, CMU-RI-TR-99-04, Janeiro 1999.
- Orghidan et al. (2005).** Orghidan, R.; Mouaddib, E. M.; Salvi, J. "Omnidirectional Depth Computation from a Single Image", in *Proceesings of the IEEE 2005*, Espanha, 2005.
- Orghidan et al. (2006).** Orghidan, R.; Salvi, J.; Mouaddib, E. M. “Modelling and accuracy estimation of a new omnidirectional depth computation sensor”, *Pattern Recognition Letters* 27, pp. 843-853, 2006.
- Press et al. (2002).** Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P. “Numerical Recipes in C”, *Cambridge University Press*, 2^o edição, 2002.
- Rosenfeld e Thurston (1971).** Rosenfeld, A.; Thurston, M. “Edge and Curve Detection for Visual Scene Analisys”, in *IEEE trans. on computers*, C-20(5): 562-569, 1971.
- Smith et al. (1990).** Smith, R.; Self, M.; Cheeseman, P., “Estimating uncertain spatial relationships in robotics”, in *Autonomous Robot Vehicles*, pp. 167–193, Nova York: Springer Verlag, 1990.

- Souza Jr. (2004).** Souza Jr, J. C., “Visão Estéreo Omnidirecional utilizando espelho duplo de perfil hiperbólico”, *tese de doutorado*, 2004.
- Southwell et al. (1996).** Southwell, M. F. D.; Basu, A.; and Reyda, J., “Panoramic stereo”, In *Proc. of the Int. Conf. on Pattern Recognition - ICPR*, pp. 378-382, Vienna, August 1996.
- Stürzl and Srinivasan (2005).** Stürzl, W.; Srinivasan, M. V. “Omnidirectional Vision with Frontal Stereo”, *DAGM 2005*, pp. 49-57, Springer Verlag, Berlin, 2005
- Svoboda et al. (1998).** Svoboda, T.; Pajdla, T.; Hlavác, V., “Epipolar Geometry for Panoramic Cameras”, *Computer Vision ECCV`98*, Freiburg, Germany, July 1998.
- Swaminathan et al. (2006).** Swaminathan, R.; Grossberg, M. D.; Nayar, S. K. “Non-Single Viewpoint Catadioptric Cameras: Geometry and Analysis”, *International Journal of Computer Vision* 66(3), pp. 211-229, Springer Science, 2006.
- Wang e Thorpe (2002).** Wang, C. C.; Thorpe, C., “Simultaneous localization and mapping with detection and tracking of moving objects”, In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2002.
- Yagi et al. (1995).** Yagi, Y.; Nishizawwa, Y.; Yachida, M., “Map-Based Navigation for Mobile Robot with Omnidirectional Image Sensor COPIS”, *IEEE Transactions on Robotics and Automation*, vol.11, n.5, Oct. 1995.
- Yagi (1999).** Yagi, Y., “Omnidirectional sensing and its applications”, *IEICE Trans. Inf. & Syst.*, Vol. E82-D, n.3, March 1999.
- Yamazawa et al. (1999).** Yamazawa, K.; Yagi, Y.; Yachida, M., “Omnidirectional imaging with hyperboloidal projection”, In *Proc. of the IEEE/RSJ Int. Conf. Intell. Robots and Systems*, July 1999.
- Zang (1994).** Zhang, Z., “Iterative point matching for registration of free-form curves and surfaces” *Int. J. Computer Vision*, vol. 13, no. 2, pp. 119–152, 1994.

ANEXO A – MANUAL DE UTILIZAÇÃO DO SOFTWARE

O programa “visao.exe” foi programado com o intuito de ser um software portátil para qualquer sistema operacional que possua as bibliotecas padrão de C++. No CD que acompanha este volume não somente o programa executável pode ser encontrado, mas também o seu código fonte completo.

Nos sistemas operacionais da família Windows o programa é executado em um modo de linha de comando. Como o usuário poderá ver mais adiante são muitas as variáveis de controle que podem ser alteradas no programa, portanto, ao invés de entrar todos os parâmetros através de argumentos na execução, um arquivo de configuração é utilizado.

O arquivo de configuração fica no mesmo diretório do executável e deve conter não somente as configurações das variáveis de configuração dos algoritmos mas também os caminhos que serão utilizados, como por exemplo o diretório de entrada e saída de dados e nomes dos arquivos auxiliares.

Além do arquivo de configuração o programa também utiliza um arquivo com a odometria real da trajetória, um arquivo com a matriz de reconstrução das distâncias e um arquivo com a matriz de reconstrução das alturas. Estes arquivos auxiliares devem estar no diretório de entrada de dados apontado pelo arquivo de configuração.

No processo da geração do mapa global um arquivo de log com os tempos das principais operações do programa é gerado em tempo real para o usuário no mesmo diretório do executável.

Em adição a estes arquivos acima citados os arquivos das imagens que formão a trajetória simulada do robô devem estar no diretório de entrada de dados apontado no arquivo de configuração.

A seguir detalha-se cada um dos arquivos de entrada e saída de dados do programa.

A1 Arquivos de entrada de dados

Os arquivos de entrada de dados são os arquivos de configuração (visão.config), de odometria (qualquer_nome.txt), das distâncias (Rp.txt), das alturas (Zp.txt) e obviamente os arquivos das imagens na sequência da trajetória simulada. Cada um destes arquivos é explicado a seguir.

10.1.1 A1.1 Arquivo de configuração

O arquivo de configuração deve se chamar “visão.config” e deve obrigatoriamente estar presente no mesmo diretório do executável “visão.exe”.

Neste arquivo são escritas as configurações que alteram as variáveis de controle do programa. Estas variáveis serão descritas mais detalhadamente a seguir.

Note que para inserir uma linha de comentário no arquivo de configuração basta escrever o caractere sustenido “#” na frente na linha que se deseja comentar. Também é importante manter em mente que quando se for informar um parâmetro na forma de um número real o separador decimal considerado é o ponto “.”. Na descrição dos parâmetros o exemplo de aplicação informa o valor padrão que é utilizado caso o mesmo não seja informado no arquivo de configuração.

A1.1.1 Settings.Anguloajuste

Este parâmetro formatado como um número real é utilizado para rotacionar a imagem omnidirecional em um determinado número de graus. Isso é útil na transformação de coordenadas da imagem de polar para retangular, fazendo com que a haste de suporte do espelho duplo fique na extremidade esquerda da imagem cartesiana, que propositalmente é descartada no processamento da imagem.

Exemplo:

Settings.Anguloajuste = 0.0

A1.1.2 Settings.Numero_imagem_inicial

Este parâmetro formatado como um número inteiro é utilizado para informar ao programa qual o número da imagem onde se inicia a trajetória simulada do robô. Note que este número pode ser informado com um valor superior ao do parâmetro “Settings.Numero_imagem_final”, informando neste caso que a seqüência de imagens deve ser processada na ordem inversa.

Exemplo:

Settings.Numero_imagem_inicial = 1

A1.1.3 Settings.Numero_imagem_final

Este parâmetro formatado como um número inteiro é utilizado para informar ao programa qual o número da imagem onde termina a trajetória simulada do robô. Note que este número pode ser informado com um valor inferior ao do parâmetro “Settings.Numero_imagem_inicial”, informando neste caso que a sequência de imagens deve ser processada na ordem inversa.

Exemplo:

Settings.Numero_imagem_final = 2

A1.1.4 Settings.Diretorio_entrada

Este parâmetro formatado como uma sequência de caracteres é utilizado para informar ao programa qual o diretório relativo de entrada de dados. O valor padrão para este parâmetro é o próprio diretório do executável, portanto no exemplo será mostrado outro valor.

Exemplo:

Settings.Diretorio_entrada = Entrada/

A1.1.5 Settings.Diretorio_saida

Este parâmetro formatado como uma sequência de caracteres é utilizado para informar ao programa qual o diretório relativo de saída de dados. O valor padrão para este parâmetro é o próprio diretório do executável, portanto no exemplo será mostrado outro valor.

Exemplo:

Settings.Diretorio_entrada = Resultados/

A1.1.6 Settings.Arquivo_odometria

Este parâmetro formatado como uma sequência de caracteres é utilizado para informar ao programa qual o nome do arquivo de odometria que deve ser lido.

Exemplo:

Settings.Arquivo_odometria = odometria.txt

A1.1.7 Settings.Arquivo_imagem

Este parâmetro formatado como uma seqüência de caracteres é utilizado para informar ao programa qual o prefixo do nome do arquivo de imagem que deve ser lido pelo programa. Note que seguido deste prefixo é encontrada a numeração da imagem na seqüência.

Exemplo:

Settings.Arquivo_imagem = imagem

A1.1.8 Settings.Remove_zero

Este parâmetro formatado como um booleano é utilizado para informar ao programa se ele deve remover os zeros do prefixo do nome da imagem na troca de casas decimais na ordem crescente ou adicionar zeros ao prefixo do nome da imagem na troca de casas na ordem decrescente.

Por exemplo caso o prefixo informado em “Settings.Arquivo_imagem” seja “imagem00”, na transição da imagem “imagem009.bmp” para a imagem “imagem010.bmp” ocorre a remoção de um zero ao prefixo da imagem que a partir deste ponto será “imagem0”.

Exemplo:

Settings.Remove_zero = false

A1.1.9 Settings.Filtro

Este parâmetro formatado como um booleano é utilizado para informar ao programa se ele deve realizar a filtragem de apenas um ponto por incremento angular de um grau na reconstrução bidimensional da imagem.

Exemplo:

Settings.Filtro = true

A1.1.10 Settings.Corte_inferior

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor de corte inferior no valor em milímetros das distâncias encontradas na reconstrução estéreo, ou seja, qualquer distância encontrada com valor inferior a este parâmetro é desprezada na reconstrução.

O valor padrão para este parâmetro é sem corte, mas para exemplificar mostra-se um outro valor sugerido.

Exemplo:

Settings.Corte_inferior = 100.0

A1.1.11 Settings.Corte_superior

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor de corte superior no valor em milímetros das distâncias encontradas na reconstrução estéreo, ou seja, qualquer distância encontrada com valor superior a este parâmetro é desprezada na reconstrução.

Este parâmetro é útil para garantir que pontos distantes que possuem considerável erro de reconstrução sejam desprezados pelo algoritmo. Note que além da incerteza com relação a posição do ponto aumentar com a distância ainda deve-se considerar o efeito da quantização da imagem na reconstrução.

O valor padrão para este parâmetro é sem corte, mas para exemplificar mostra-se um outro valor sugerido.

Exemplo:

Settings.Corte_superior = 2000.0

A1.1.12 Settings.UsaOpcional

Este parametro formatado como um booleano é utilizado para informar ao programa se ele deve realizar a utilização comportamento opcional do algoritmo de auto-localização e mapeamento simultâneos com odometria.

Exemplo:

Settings.UsaOpcional = true

A1.1.13 Settings.UsaFFT

Este parâmetro formatado como um booleano é utilizado para informar ao programa se ele deve realizar as convoluções no domínio do espaço ou da frequência, sendo “true” para utilizar a FFT e a IFFT e “false” para utilizar a convolução espacial.

Exemplo:

Settings.UsaFFT = false

A1.1.14 Settings.LogIncremental

Este parâmetro formatado como um booleano é utilizado para informar ao programa se ele deve incrementar o arquivo de log dos tempos de execução ou apagar o atual e iniciar um novo arquivo.

Exemplo:

Settings.LogIncremental = true

A1.1.15 Settings.Pausalteracao

Este parâmetro formatado como um booleano é utilizado para informar ao programa se ele deve realizar uma pausa entre cada iteração do algoritmo de construção de mapa do ambiente. Isso é útil para consultar os arquivos intermediários gerados na iteração, que são sobrescritos a cada iteração.

Exemplo:

Settings.PausaAlteracao = false

A1.1.16 Settings.Sigma

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor do sigma da equação da gaussiana utilizada no algoritmo Canny.

Exemplo:

Settings.Sigma = 1.0

A1.1.17 Settings.TamanhoFiltro

Este parâmetro formatado como um número inteiro é utilizado para informar ao programa qual o valor de w na fórmula “ $2w + 1$ ” que calcula o tamanho da janela para o filtro gaussiano convolucionado com a imagem cartesiana. Além da entrada com número inteiro também pode ser utilizado o valor “auto”, que permite ao algoritmo calcular o tamanho desta janela automaticamente.

Exemplo:

Settings.TamanhoFiltro = auto

A1.1.18 Settings.LimiteSuperior

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor do limite superior da para o algoritmo Canny aceitar uma quina fraca, e informa o valor de limite inferior para que a quina seja considerada forte. Além da entrada com número inteiro também pode ser utilizado o valor “auto”, que permite ao algoritmo calcular este valor limite automaticamente.

Exemplo:

Settings.LimiteSuperior = auto

A1.1.19 Settings.LimiteInferior

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor de limite inferior para que a quina seja aceita, sendo classificada como quina fraca. Além da entrada com número inteiro também pode ser utilizado o valor “auto”, que permite ao algoritmo calcular este valor limite automaticamente.

Exemplo:

Settings.LimiteInferior = auto

A1.1.20 Settings.Srelim

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor de limite superior para a o Erro Quadrático Relativo Percentual utilizado para aceitar uma transformação realizada com o algoritmo TrICP.

Exemplo:

Settings.Srelim = 30.0

A1.1.21 Settings.RelaxaSrel

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor da relaxação do Srelim utilizada no algoritmo de auto-localização e mapeamento simultâneos com odometria a cada iteração onde não ajam correspondências bem sucedidas.

Exemplo

Settings.RelaxaSrel = 10.0

A1.1.22 Settings.AnguloLimite

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor de limite superior para aceitar a transformação de coordenadas calculada baseada na rotação encontrada.

Este parâmetro pode ser útil em testes em que se sabe antecipadamente que nenhuma rotação será realizada.

Além da entrada com o número real este parâmetro também aceita uma entrada com valor “off”, que desliga esta checagem.

Exemplo:

Settings.AnguloLimite = off

A1.1.23 Settings.UsaBlackman

Este parâmetro formatado como um booleano é utilizado para informar ao programa se ele deve utilizar a filtragem com o filtro Blackman antes de processar os pontos obtidos em Canny no função de similaridade SDA.

Exemplo:

Settings.UsaBlackman = false

A1.1.24 Settings.IntensidadeRelativa

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor do limite inferior entre a relação da diferença entre o máximo e o mínimo da intensidade da janela da imagem pela média da intensidade da janela da imagem para aceitar um ponto da imagem como possível candidato a mínimo local antes de ser processado pelo algoritmo de programação dinâmica na correspondência entre os pontos da imagem interna e externa.

Exemplo:

Settings.IntensidadeRelativa = 0.15

A1.1.25 Settings.PP_MeiaJanelaVertical

Este parâmetro formatado como um número inteiro é utilizado para informar ao programa qual o valor de w na fórmula " $2w + 1$ " que calcula o tamanho da janela que é utilizada para a pré-seleção de candidatos para mínimos locais antes de serem processados pelo algoritmo de programação dinâmica para a obtenção da correspondência entre os pontos da imagem interna e externa.

Exemplo:

Settings.PP_MeiaJanelaVertical = 3

A1.1.26 Settings.DisparidadeMinima

Este parâmetro formatado como um número inteiro é utilizado para informar ao programa qual o valor da mínima disparidade aceita para considerar um ponto na imagem interna e externa possíveis candidatos a mínimo local na correspondência dos pontos realizada na programação dinâmica.

Este parâmetro deve ser ajustado conforme a construção do espelho utilizado na obtenção da imagem.

Exemplo:

Settings.DisparidadeMinima = 10

A1.1.27 Settings.DisparidadeMaxima

Este parâmetro formatado como um número inteiro é utilizado para informar ao programa qual o valor da máxima disparidade aceita para considerar um ponto na imagem interna e externa possíveis candidatos a mínimo local na correspondência dos pontos realizada na programação dinâmica.

Este parâmetro deve ser ajustado conforme a construção do espelho utilizado na obtenção da imagem.

Exemplo:

Settings.DisparidadeMaxima = 90

A1.1.28 Settings.PD_MeiaJanelaVertical

Este parâmetro formatado como um número inteiro é utilizado para informar ao programa qual o valor de w na fórmula " $2w + 1$ " que calcula o tamanho da janela vertical que é utilizada para o cálculo do custo entre dois possíveis pontos correspondentes na matriz de custo da programação dinâmica.

Exemplo:

Settings.PD_MeiaJanelaVertical = 3

A1.1.29 Settings.PD_MeiaJanelaHorizontal

Este parâmetro formatado como um número inteiro é utilizado para informar ao programa qual o valor de w na fórmula “ $2w + 1$ ” que calcula o tamanho da janela horizontal que é utilizada para o cálculo do custo entre dois possíveis pontos correspondentes na matriz de custo da programação dinâmica.

Exemplo:

Settings.PD_MeiaJanelaHorizontal = 1

A1.1.30 Settings.CustoOclusao

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor do custo da oclusão entre dois possíveis pontos correspondentes na matriz de custo da programação dinâmica.

Exemplo:

Settings.CustoOclusao = 2.5

A1.1.31 Settings.Loop

Este parâmetro formatado como um número inteiro é utilizado para informar ao programa qual o número de vezes que as imagens na sequência devem ser percorridas pelo algoritmo de construção de mapas simulando um determinado número de voltas em uma trajetória fechada.

Note que neste caso é necessário alterar o arquivo de odometria manualmente, adicionando a distância percorrida entre o ultimo ponto da trajetória e o primeiro ponto da trajetória e depois repedindo-se os valores do arquivo convencional.

Exemplo:

Settings.Loop = 0

A1.1.32 Settings.Fusao

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor da distância máxima entre os pontos que devem ser aglutinados na integração entre o mapa global e último mapa local obtido.

Exemplo:

Settings.Fusao = 50.0

A1.1.33 Settings.RaioFiltro

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor da distância máxima para que o ponto não seja considerado ruído caso não sejam encontrados outros pontos na região delimitada. Este parâmetro é utilizado em conjunto com o próximo parâmetro descrito.

Exemplo:

Settings.RaioFiltro = 125.0

A1.1.34 Settings.PontosFiltro

Este parâmetro formatado como um número inteiro é utilizado para informar ao programa qual o valor do número de pontos que devem ser encontrados na região definida pelo parâmetro acima descrito para que mesmo não seja considerado ruído.

Exemplo:

Settings.PontosFiltro = 1

A1.1.35 Settings.Iterações

Este parâmetro formatado como um número inteiro é utilizado para informar ao programa qual o valor do intervalo do número de iterações bem sucedidas em que o mapa global deve ser filtrado de acordo com o número de repetições do mesmo ponto.

Exemplos:

Settings.Iteracoes = 5

A1.1.36 Settings.PontosRepetidos

Este parâmetro formatado como um número inteiro é utilizado para informar ao programa qual o valor do número mínimo de repetições do mesmo ponto para que ele não seja removido na filtragem descrita no parâmetro acima.

Exemplos:

Settings.PontosRepetidos = 3

A1.1.37 Settings.Porcentagem

Este parâmetro formatado como um número real é utilizado para informar ao programa qual o valor da porcentagem que deve ser utilizada no cálculo do Erro Quadrático Percentual e não deve ser superior a 100 nem inferior a 0.

Exemplo:

Settings.Porcentagem = 20.0

10.1.2 A1.2 Arquivo de odometria

O arquivo de odometria pode possuir qualquer nome, devendo este ser informado no arquivo de configuração, tendo o seu nome padrão “odometria.txt” e deve obrigatoriamente estar presente no diretório de entrada de dados indicado no arquivo de configuração.

A primeira linha deste arquivo deve informar ao programa o número de imagens consideradas neste arquivo de odometria. Por exemplo se forem 59 imagens, com 59 dados de odometria entre eles a primeira linha deste arquivo deverá conter o número 59.

A partir da segunda linha, deve-se informar no referencial da imagem reconstruída sem nenhuma rotação a distância percorrida entre a imagem anterior e a imagem em questão e o ângulo em graus com o sentido desta translação. Note que a segunda linha deverá conter o valor 0.0 e 0.0, pois não há imagens anteriores, sendo este o ponto inicial da auto-localização do robô no início da trajetória.

Como exemplo suponha que a translação entre a imagem 1 e 2 da trajetória seja de 200 milímetros com sentido descendente no eixo y e que a trajetória possui apenas estas duas imagens, o arquivo de odometria fica na forma ilustrada abaixo.

```
2
0.0 0.0
200.0 -90.0
```

O usuário pode fazer uso do parâmetro que faz pausa entre as iterações para obter as referências corretamente.

10.1.3 A1.3 Arquivos das distâncias e alturas

O arquivo de distância deve possuir o nome “rP.txt” e o arquivo das alturas deve chamar-se “zP.txt” e devem obrigatoriamente estar presente no diretório de entrada de dados indicado no arquivo de configuração.

Estes dois arquivos armazenam um matriz de dimensão 225 por 225 que respectivamente armazenam os valores da distância e altura para um par de pontos encontrados a uma terminada altura em pixels da imagem interna e externa, lembrando que a imagem cartesiana possui dimensão 225 por 360 pixels.

O valor da linha considerada é relativa a imagem interna e a coluna a imagem externa., assim por exemplo se na mesma epipolar o ponto correspondente encontrado pela programação dinâmica na imagem interna e externa possuem respectivamente as alturas em pixel igual a 45 e 60 o valor da distância do ponto ao sistema de visão reconstruído no espaço será dado pelo valor da matriz na coordenada 45x60.

A2 Arquivos de saída de dados

Os arquivos de saída de dados são gerados durante a execução do programa “visão.exe”, sendo estes, o arquivo de log (timelog.txt) dos tempos das principais operações, gerado em tempo real no mesmo diretório do executável, os arquivos numerados com os mapas globais (mapaX.txt) intermediários, onde o X corresponde ao número do mapa global, o arquivo de odometria (odometriaxy.txt) com coordenadas x e y respectivamente utilizadas na execução da instância do programa devidamente rotacionado e com o erro aleatório

inserido, o arquivo do mapa local estático (Mpre.txt) utilizado na última tentativa aceita do TrICP, o arquivo do mapa local móvel utilizado na última tentativa aceita do TrICP antes da aplicação da transformação de coordenadas (Ppre.txt), o arquivo do mapa local móvel utilizado na última tentativa aceita do TrICP depois da aplicação da transformação de coordenadas (Ppos.txt), o arquivo que armazena o status dos principais vetores de resultados da operação (vetores.txt), o arquivo de auto-localização relativa (auto-localização.txt), o arquivo de com os valores finais dos erros quadráticos do TrICP a cada iteração (erros.txt) e o arquivo com as configurações utilizadas (configurações.txt) no arquivo de entrada de configurações (visao.config) de uma forma mais alto nível. Cada um destes arquivos é explicado com maiores detalhes a seguir.

10.1.4 A2.1 Arquivo do tempo de execução

O arquivo de log do tempo de execução mostra os tempos de execução de cada convolução realizada no programa, na ordem, as quatro convoluções da imagem interna, seguida do tempo total no Canny, com o percentual do tempo que o mesmo passou realizando convoluções, as quatro convoluções da imagem externa seguida do tempo total no Canny, com o percentual do tempo que o mesmo passou realizando convoluções mais as duas convoluções com o filtro de blackman das imagens interna e externa antes de passarem pelo algoritmo de programação dinâmica, seguido do tempo total da iteração.

Inicialmente o arquivo mostra o tempo total de inicialização das variáveis e arquivos de entrada e ao final da operação é mostrado o tempo total da operação completa, o tempo total utilizado nas convoluções e o percentual do tempo total gasto nesta operação.

Este arquivo é criado no mesmo diretório do executável “visão.exe”.

10.1.5 A2.2 Arquivo dos mapas globais

Os arquivos dos mapas globais são criados no diretório de saída apontado no arquivo de configuração, sendo o número de arquivos igual ao número de imagens da trajetória. A contagem do mapa, entretanto é diferente, sendo o mapa0.txt o mapa inicial referente ao mapa local da primeira imagem da trajetória e o último mapa global igual a $n-1$, onde n é o número total de imagens utilizadas na trajetória.

O arquivo é composto pela coordenada x seguida da coordenada y e por final o número total de repetições do mesmo ponto encontrado. Este número é utilizado para realizar a filtragem dos pontos a cada número de iterações definido pelo usuário no arquivo de configuração no parâmetro “Settings.PontosRepetidos”.

10.1.6 A2.3 Arquivo de odometria

O arquivo de odometria é criado no diretório de saída apontado no arquivo de configuração, e se chama “odometriaxy.txt”.

O arquivo é composto pela coordenada x seguida da coordenada y da odometria considerada no algoritmo de construção de mapas, já levando em consideração o erro aleatório gerado na execução do programa.

10.1.7 A2.4 Arquivo do mapa estático do TrICP

O arquivo do mapa estático da última tentativa aceita no TrICP chama-se “Mpre.txt” e é criado no diretório de saída apontado no arquivo de configuração

O arquivo é composto pela coordenada x seguida da coordenada y do mapa em questão.

10.1.8 A2.5 Arquivo do mapa móvel antes do TrICP

O arquivo do mapa móvel antes da última tentativa aceita no TrICP chama-se “Ppre.txt” e é criado no diretório de saída apontado no arquivo de configuração

O arquivo é composto pela coordenada x seguida da coordenada y do mapa local em questão.

10.1.9 A2.6 Arquivo do mapa móvel depois do TrICP

O arquivo do mapa móvel depois da última tentativa aceita no TrICP chama-se “Ppos.txt” e é criado no diretório de saída apontado no arquivo de configuração

O arquivo é composto pela coordenada x seguida da coordenada y do mapa local em questão.

10.1.10 **A2.7 Arquivo de vetores das iterações**

O arquivo que armazena o estado dos principais vetores utilizados nas iterações do algoritmo de criação de mapas e auto-localização chama-se “vetores.txt” e é criado no diretório de saída apontado no arquivo de configuração.

Na primeira linha este arquivo indica qual o valor de *Srelim* utilizado na execução do programa “visão.exe”, nas linhas posteriores é informado o número da imagem, seguido da bandeira de saída resultante das tentativas do algoritmo TrICP, que pode variar de algoritmo para algoritmo indicando em qual tentativa do TrICP a transformação obtida foi aceita, seguida do *Srel* obtido na última tentativa efetuada do TrICP, seguido do *Srel* médio obtido na última tentativa efetuada do TrICP, seguido do número total de iterações do TrICP para convergência dos mapas.

Para exemplificar o valor da bandeira retornada neste arquivo utiliza-se o algoritmo descrito no capítulo 4.3, que realiza duas tentativas de TrICP por iteração. O valor retornado para este algoritmo pode ser 0, indicando que o TrICP foi aceito em sua primeira tentativa, 1, indicando que o TrICP foi aceito em sua segunda tentativa e 2 indicando que nenhum resultado do TrICP foi aceito nesta iteração do algoritmo.

10.1.11 **A2.8 Arquivo de auto-localização**

O arquivo que armazena os valores das autolocalizações obtidas pelo algoritmo chama-se “autolocalizacao.txt” e é criado no diretório de saída apontado no arquivo de configuração.

Este arquivo contém os valores das autolocalizações obtidas nos eixos x e y e da direção do robô para cada iteração do algoritmo de geração de mapa. Se em uma determinada iteração não for possível calcular a auto-localização do robô o valor encontrado para x e y e a rotação serão zero.

10.1.12 **A2.9 Arquivo de erros**

O arquivo que armazena os valores do Erro Quadrático, Erro Quadrático Médio e Erro Quadrático Percentual por iteração do algoritmo chama-se “erros.txt” e é criado no diretório de saída apontado no arquivo de configuração.

10.1.13 A2.10 Arquivo de configurações

Este arquivo mostra as configurações utilizadas na execução do programa da mesma forma que os valores são exibidos na tela no início do programa, sendo útil para guardas as configurações utilizadas em um determinado arquivo de configuração.

ANEXO B – CONTEÚDO DO CD

O CD que acompanha este volume possui três diretórios. O diretório “código” apresenta o código do programa implementado em C++. Em anexo também segue o projeto completo feito no Visual Studio 2005. No diretório “imagens” as imagens do ambiente simulado e real utilizadas neste trabalho são disponibilizadas. No diretório PovRay o código que possibilita a geração do ambiente simulado utilizado no trabalho é armazenado.

Na raiz do CD também se encontra este trabalho na versão PDF.