Trabalho de Conclusão de Curso | 2015

ÁDRIA ALINE CASTRO DA SILVA

LOCALIZAÇÃO E MAPEAMENTO SILMULTÂNEOS BASEADO NA EXTRAÇÃO DE CARACTERÍSTICAS

PALMAS / TO

2015

ÁDRIA ALINE CASTRO DA SILVA

LOCALIZAÇÃO E MAPEAMENTO SILMULTÂNEOS BASEADO NA EXTRAÇÃO DE CARACTERÍSTICAS

Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas da Informação da Fundação Universidade do Tocantins, como requisito à obtenção do título de Bacharel em Sistemas da Informação.

Orientador: Igor Yepes

PALMAS / TO

2015

COMISSÃO EXAMINADORA

_____

Prof. Nome (orientador)

_____

Prof. Nome (titular)

_____

Prof. Nome (titular)

_____

Prof. Nome (suplente)

Palmas, ____ de _____ de 20__.

“Há um tempo em que é preciso abandonar as roupas usadas, que já tem a forma do nosso corpo, e esquecer os nossos caminhos, que nos levam sempre aos mesmos lugares. É o tempo da travessia: e, se não ousarmos fazê-la, teremos ficado, para sempre, à margem de nós mesmos”.

Fernando Pessoa

Dedicatória

A Deus, que em sua infinita sabedoria guia meus caminhos me proporcionando saúde, serenidade e disposição para enfrentar todas as etapas desta árdua caminhada.

A minha mãe, meu pai que me proporcionaram amor e apoio incondicional durante todo o período da Universidade.

Aos meus amigos da Universidade, em especial a meus amigos Diógenes, Agnélio, Mariana, Marinete, Célia, Kayque, Atila, Mauricio e Francisco, que sempre estiveram dispostos a ajudar e auxiliar no que era preciso.

A todos os professores do curso, que foram tão importantes na minha vida acadêmica. Em especial, agradeço ao meu orientador, que teve paciência e que me ajudou bastante á concluir este trabalho.

LOCALIZAÇÃO E MAPEAMENTO SILMULTÂNEOS BASEADO NA EXTRAÇÃO DE CARACTERÍSTICAS

RESUMO

Uma das características que um robô pode possuir para ser considerado autônomo é a capacidade de criar uma representação do ambiente em que está inserido e simultaneamente se localizar dentro dele. Isso porque, para a realização de um determinado objetivo o robô deve ser capaz de conhecer o ambiente, e saber sua localização dentro dele, de forma análoga a realizada por humanos, podendo assim estabelecer a melhor rota a ser seguida para desempenhar as tarefas necessárias ou para simplesmente realizar um mapeamento o mais fiel possível do entorno. Este trabalho descreve um processo baseado em técnicas de visão computacional para determinação da pose (posição e orientação) e mapeamento de ambiente por um robô móvel a partir de imagens capturadas por uma câmera simples, sem uso de sensores adicionais. O trabalho aborda técnicas probabilísticas para as estimativas de pose e mapeamento, onde ambas as etapas são estimadas simultaneamente.

Palavras-chave: SLAM, SURF, SIFT, Robótica móvel.

Lista de Figuras

Lista de Símbolos

SLAM	*Simultaneous Localization and Mapping*
VSLAM	*Visual Simultaneous Localization and Mapping*
SIFT	*Scale Invariant Features*
SURF	*Speeded Up Robus Features*
RANSAC	*Random Sample Consensus*
DoG	*Difference of Gaussian*
Haar	*Haar wavelet*
KF	*Kalman Filter*

Sumário

1. INTRODUÇÃO

De acordo com Thrun, Fox e Burgard (2005), Robótica é a ciência de perceber e manipular o mundo físico através de dispositivos mecânicos controlados por computador. Com o passar dos anos, foi observado no campo da robótica à importância e o potencial da robótica móvel autônoma, que vem buscando auxiliar e substituir atividades realizadas por humanos.

A utilização de robôs móveis autônomos para realizar ou auxiliar atividades humanas em nossa sociedade vem demonstrando o quanto é promissora essa área. Alguns exemplos são seu uso em aplicações domésticas, como aspiradores de pó e cortadores de grama robóticos, atividades urbanas como cadeiras robotizadas ou ate mesmo aplicação na segurança civil como resgate e exploração em ambientes hostis.

Uma das características para autonomia completa do robô é a capacidade de determinar a localização de objetos no ambiente (mapeamento) e estimar a pose (posição e orientação) com relação a esses objetos (localização). Assim surge o problema: Para se obter a posição e orientação do robô dentro de um ambiente é necessário um mapa, e para obter o mapa e necessário informações de posição e orientação do robô em relação ao ambiente. O problema do SLAM, busca resolver ambas as etapas (mapeamento e localização) simultaneamente.

Uma das técnicas utilizadas no SLAM é a utilização de câmera como sensor para obtenção de informação do ambiente. Quando câmera é utilizada como sensor, é classificado pela literatura como SLAM Visual, que consiste na utilização sequencial de imagens para estimar o mapeamento e localização da câmera. Um dos processos para realização do SLAM consiste na extração de referencias espaciais do ambiente, designado pela literatura como marcos ou "*landmarks"*, que são utilizados como referencia pelo robô de maneira a auxiliar sua navegação.

O presente trabalho tem como objetivo conceituar os principais processos para obtenção de marcos visuais baseados na extração de características e apresentar um algoritmo para correspondência de pontos característicos em imagens com a técnica SURF.

1. REVISÃO DA LITERATURA

Este capítulo aborda os principais conceitos que envolvem o processo de Localização e Mapeamento simultâneos com visão computacional. A visão computacional pode ser definida como um conjunto de métodos e técnicas aplicados a sistemas computacionais para que estes possam ser capazes de extrair e interpretar informações de imagens.

1.1. LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS (SLAM)

Localização e mapeamento são uma das funções básicas que um sistema robótico deve possuir para ser considerado autônomo. Sistemas robóticos autônomos ou simplesmente robôs moveis são capazes de cumprir objetivos em ambientes desconhecidos ou parcialmente conhecidos, com pouca ou nenhuma interferência humana.

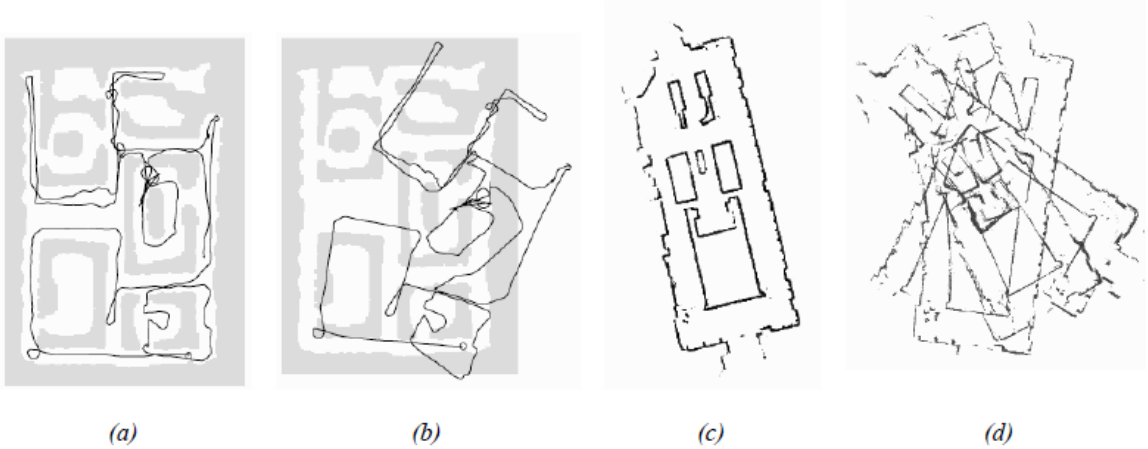
Leonard e Durrante-Why (1991) resumem o problema de navegação autônoma em três perguntas:

- Onde estou?
- Aonde vou?
- Como eu vou chegar lá?

A primeira questão diz respeito à sua localização no ambiente, a segunda e a terceira dizem respeito ao objetivo a ser cumprido pelo robô, onde o mesmo planeja um caminho que resulta na concretização desse objetivo. Assim fica claro que a resposta a essas perguntas especificam características essenciais para robôs moveis, que são a capacidade de estimar sua pose (posição e orientação) e de criar um mapa do ambiente no qual está inserido. De forma sucinta, o robô deve ser programado com "inteligência"

para perceber o seu ambiente e tomar decisões que aumentam suas chances de alcançar seus objetivos.

Os robôs utilizam sensores para obter informações do ambiente, fornecendo dados referentes à localização desses objetos no ambiente e as características que possuem. Inicialmente o robô não tem informação alguma de seus estados de localização e mapeamento precisando assim estimá-los conforme se movimenta pelo ambiente, porem, após longos períodos de navegação, erros são gerados pela imprecisão de sensores, que aumentam gradativamente conforme o robô se locomove pelo ambiente, resultando assim incertezas nas estimativas que impedem que sejam obtidos resultados confiáveis. Erros nas estimativas resultam em uma representação errada do ambiente, e consequentemente em erro na localização. A influencia de incertezas nas etapas de localização e mapeamento podem ser representado pela Figura abaixo:



(a) Localização sem erros (b) Localização com erro (c) Mapeamento sem erros (d) Mapeamento com erros

Figura 1 - Influência da incerteza na localização e mapeamento
Fonte: THRUN, FOX e BURGARD, 2005.

Esses erros são imprevisíveis e, portando, difícil de serem modelados computacionalmente. Individualmente esses erros não podem ser previstos, mas quando considerados como um conjunto de fenômenos isolados percebeu-se que eles podem ser tratados estaticamente, apresentando padrões que podem ser modelados (GUIZILINI, 2008, p.5).

Thrun, Burgard e Fox (1999-2000, p.14), destacam cinco elementos que estão ligados à imprecisão ou incertezas das informações obtidas pelo sistema robótico, são elas:

- **Ambientes** – Mundos físico são inerentemente imprevisíveis. Enquanto o grau de incerteza em ambientes bem estruturados como linhas de montagem é baixo, ambientes como rodovias e casas particulares são altamente dinâmicos e imprevisíveis.

- **Sensores –** Os sensores são inerentemente limitados no que eles podem perceber. Limitações surgem a partir de dois fatores principais. Primeiro, o alcance e resolução de um sensor estão sujeitos a leis físicas. Por exemplo, câmeras não podem ver através das paredes, e até mesmo dentro da faixa perceptual a resolução espacial das imagens da câmera é limitada. Segundo, sensores estão sujeitos a ruídos, o que perturba as medições de formas imprevisíveis e limita, portanto, a informação que pode ser extraída a partir de medições do sensor.

- **Robots –** Funcionamento de robôs envolve acionamento de motores que são, pelo menos em certa medida, imprevisíveis, devido a variáveis como ruído no sinal de controle e desgaste de engrenagens e outros componentes. Alguns atuadores, como pesados robôs industriais, são bastante precisos. Outros, como os robôs móveis de baixo custo, podem ser extremamente imprecisos.

- **Modelos –** Modelos são inerentemente imprecisos. Modelos são abstrações do mundo real. Como tal, eles modelam apenas parcialmente os processos físicos subjacentes ao robô e seu ambiente. Erros de modelos são uma fonte de incerteza que têm sido largamente ignorada na área de robótica.

- **Computação –** Os robôs são sistemas de tempo real, o que limita a quantidade de computação que pode ser levada a cabo. Muitos algoritmos *state-of-the-art* são aproximados, conseguindo respostas no tempo desejado mediante sacrifício da precisão.

Todos esses elementos cooperam com incertezas nas estimativas. O sistema a ser desenvolvido deve ser analisado e projetado para atuar de acordo com o objetivo do robô levando em consideração as questões relatadas acima.

Com o aumento do uso de sistemas robóticos em atividades humanas, ou seja, o uso de robôs em ambientes desestruturados, a capacidade de lidar com a incerteza é

fundamental para a construção de sistemas confiáveis. A partir daí surge à ideia de robótica probabilística. A robótica probabilística representa a incerteza explicitamente, usando cálculos da teoria da probabilidade, onde os estados de localização e mapeamento do robô deixam de apresentar valores únicos e se tornam distribuições de probabilidades que indicam todos os estados possíveis dadas a precisão das observações realizadas até aquele momento (GUIZILINI, 2008, p.5).

Abordagens probabilísticas são tipicamente mais robustas em face das limitações de sensores como ruídos, dinâmica do ambiente e assim por diante. Elas costumam se adaptar melhor em ambientes complexos e não estruturados, onde a capacidade de lidar com a incerteza é de uma importância ainda maior (THRUN, BURGARD & FOX, 1999-2000, p.14).

Assim, o SLAM **(***Simultaneous Localization and Mapping***)** busca solucionar o problema de incertezas nas estimativas estimando localização e mapeamento simultaneamente, uma vez que para o mapeamento é necessária à localização do robô para que possam ser distribuídas as informações obtidas pelos sensores em um sistema de coordenadas e, por sua vez, para a localização é necessário o mapa do ambiente para que através das informações obtidas pelos sensores possam estimar a posição do robô no ambiente (BOAS, 2011, p. 20).

Sendo assim, pode-se verificar que ambas as etapas são correlacionadas, uma vez que uma depende da outra para ser estimada. Santana (2011, p.23) afirma que “a correlação existe porque um erro na localização terá efeito na percepção universal da localização de todas as características mapeadas”.

Ao invés de tentar gerar modelos computacionais para resolver o problema de erro de mapeamento e o problema de localização de forma independente, o SLAM calcula as estimativas levando em consideração as incertezas em ambas as etapas, pois é preciso lidar com as incertezas nas estimativas de forma global, ao invés de trata-las individualmente.

O problema da Localização e Mapeamento Simultâneos (SLAM) consiste em estimar os estados de localização e mapeamento simultaneamente, trabalhando com

abordagens probabilísticas com o intuito de tratar os erros sistematicamente antes que se acumulem. A Figura 2 ilustra o problema de SLAM:

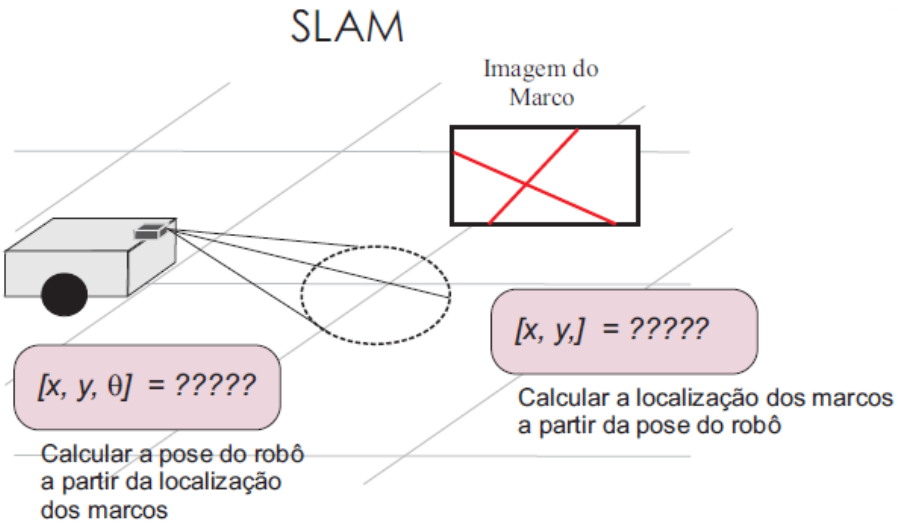


Figura 2 - Problema de SLAM
Fonte: SANTANA, 2011, p. 23

Como ilustrado na Figura 2, o SLAM consiste na utilização das mesmas informações sensoriais para calculo das estimativas. Uma das técnicas para SLAM e a utilização de *landmarks* ou marcos, que são estruturas no ambiente utilizado pelo robô para se localizar conforme se locomove. Essa abordagem será vista nas próximas seções.

1.1.1. MARCOS NO SLAM

Marcos ou *landmarks* são características que podem ser naturalmente re-observadas e distintas do resto do ambiente. Marcos devem ser facilmente observáveis, ou seja, um mesmo marco deve ser reconhecido de diversos ângulos, devem ser distinguíveis, estáticos e devem ter em abundancia no ambiente (SANTANA, 2011). O sistema robótico deve através de seus sensores, detectar um marco e ser capaz de reconhecê-lo em um momento posterior.

Quando há erro na detecção de marcos, o número de referencias utilizado pelo robô para corrigir suas estimativas diminui, e as falhas em reconhecer um marco conduzem a erros nas estimativas de mapeamento e localização.

Quando marcos são utilizados no processo de SLAM, as características deles são inseridas em um vetor de estado à medida que são identificados, caso não haja correspondência com outros marcos já adquiridos anteriormente. Caso seja uma reobservação de um marco já detectado em um momento anterior, apenas uma atualização das estimativas é feita (SANTANA, 2011). Ao processo de identificação de um marco, dar-se o nome de correspondência ou *matching*.

De acordo com Bigheti (2011, p. 11), a detecção de *landmark* é dividida basicamente em três fases:

- Capturar os dados do ambiente através do sensor;
- Analisar os dados capturados a fim de identificar as informações que caracterizam um landmark;
- Armazenar as características que correspondem aos landmarks na memória do sistema de controle do robô para representar o mapa do ambiente. Esses landmarks também serão usados para estimar a localização do robô.

De acordo com a contribuição de Bigheti, um sensor é utilizado para coletar (detectar) informações do ambiente (características). Depois de detectadas, essas características precisam passar por um processo de determinação de marcos, pois se deve levar em consideração que nem todas as características são relevantes. Outra questão é a definição do numero de marcos a inicializar no sistema, uma vez que, sem um processo de determinação, todas as características seriam utilizadas como marcos, aumentando assim o consumo de processamento do sistema.

De acordo com o deslocamento do robô, características são obtidas, e essas devem ser correspondidas. Caso haja uma correspondência, as estimativas de mapeamento e localização são atualizadas. Grande parte da literatura utiliza uma cena como marco para ambientes internos, e um grupo de marcos similar determina um local do ambiente. Outra abordagem é utilizar objetos individuais da cena como marcos. No entanto, há certas desvantagens quanto ao uso de objetos individuais como marcos: a primeira é de que os objetos nem sempre estão disponíveis em todas as imagens, a segunda é a difícil correspondência devido à similaridade entre os objetos. Já a cena, preserva a organização dos maiores elementos, como, por exemplo, mesas, camas, janelas, etc. O trabalho de Guizilini (2008) apresenta como critério para determinação de um conjunto de marcos a distancia entre características na imagem e a diferença de contraste entre elas. O uso de

reconhecimento de cenas internas pode ser visto com mais detalhes no trabalho de Torralba e Quattoni (2001).

1.1.1.1. MARCO ARTIFICIAL

Marcos artificiais são estruturas adicionadas previamente ao ambiente, onde o sistema robótico já tem conhecimento prévio do marco a ser utilizado. Sendo assim, o nome marco artificial dá-se ao fato de que as estruturas utilizadas para auxiliar na navegação não fazem parte do ambiente natural. A desvantagem no uso de marcos artificiais é a necessidade de adequar o ambiente para que o robô possa navegar.

Esse tipo de *landmark* é detectado facilmente pelos sensores do robô, pois suas características foram artificialmente criadas com o intuito de serem explícitas no ambiente de navegação do robô (BIGHETI, 2011, p. 09). Essa abordagem permitiu o surgimento de robôs autônomos em ambientes controlados e previsíveis, como em linhas de montagem de fábricas, que podem ser modificadas sem grandes dificuldades e apresentarão sempre os mesmos tipos de movimentos.

1.1.1.2. MARCO NATURAL

Marco natural diz respeito ao uso das estruturas que fazem parte do ambiente natural, ou seja, o ambiente não precisa ser modificado; a navegação do sistema robótico acontece em um ambiente natural, ampliando a gama de situações reais onde pode ser utilizado. Os *landmarks* naturais podem basear-se na estrutura do ambiente como bordas e cantos de portas e janelas. Mas, também podem basear-se na variação da intensidade de luminosidade de certa região de uma imagem (BIGHETI, 2011, p. 10). A Figura 3 ilustra a detecção de marcos natural em uma imagem.

Figura 3-Detecção de marcos naturais
Fonte: Autoria própria.

1.1.2. SENSOR NO SLAM

No problema de SLAM, um robô móvel usa seus sensores para explorar o ambiente, ganha conhecimento sobre ele, interpreta o cenário, constrói um mapa adequado e, em seguida, calcula a sua posição relativa usando o mapa que está sendo criado (SANTANA & MEDEIROS, 2011, p. 01). Nos trabalhos iniciais de SLAM, era comum o uso de sensores de distancia, como *laser scanners* e sonares (ver Figura 4). O *laser scanner* é eficiente e não necessitam de recursos computacionais avançados, no entanto é caro e não funciona bem em superfícies translúcidas (vidro, acrílico etc.). Os sonares são mais baratos em relação aos *lasers*, porém suas medições tendem a ser mais ruidosas. Esses sensores fornecem somente informações relativas à posição dos objetos no ambiente em relação ao robô, geralmente em um único plano, dificultando a caracterização única dos objetos.

Sistemas de visão fornecem uma caracterização mais rica das estruturas presentes no ambiente. Seu uso tem se tornado propício devido ao alto poder de processamento dos computadores e a riqueza de informações obtidas do ambiente em uma única imagem.

De acordo com Chen, citado por Araújo et al (2011), usar câmeras nos proporciona uma fonte de informação extremamente rica sobre o ambiente. Em

decorrência disso, há uma forte tendência no uso de câmeras de vídeo como sensores, pelos seguintes benefícios que estas proporcionam:

- Sistemas de visão são na maioria das vezes, baratos.
- Câmeras são leves e de dimensões aceitáveis.
- São facilmente integradas ao hardware do robô.
- Consomem pouca energia.

A Figura abaixo ilustra os principais sensores utilizados no SLAM.

Figura 4 - Sensores utilizados em SLAM
Fonte: SANTANA, 2011, p. 33

Atualmente, diversos trabalhos são apresentados na literatura utilizando como sensor principal uma câmera de vídeo. O trabalho apresentado por Buscarriollo (2008) apresenta a viabilidade da aplicação de métodos baseados em visão e laser para sistemas de posicionamento dinâmico para projetos VSNT (Veículo Submersível não Tripulado). Santana e Medeiros (2011) propõe uma abordagem para o SLAM usando linhas no chão como marcos e Filtro de Kalman para tratar as incertezas em um sistema de visão monocular.

1.2. SLAM VISUAL

O SLAM Visual ou VSLAM (*Visual Simultaneous Localization and Mapping*) consiste em calcular as estimativas de localização e mapeamento usando como fonte sensorial uma câmera de vídeo. Em SLAM Visual, é utilizada sequência de imagens e os parâmetros intrínsecos e extrínsecos da câmera para obter as estimativas de mapeamento e localização.

Para serem utilizadas como marcos, as imagens precisam ser processadas de forma a se extrair informações discriminantes de cada imagem. As duas principais abordagens do SLAM Visual são:

- **Baseado em primitivas geométricas**: baseia-se na utilização de primitivas geométricas como pontos, retas, contornos entre outros.
- **Registro direto da imagem**: as abstrações dos dados provém dos níveis de intensidade dos *pixels* da imagem.

A técnica baseada em primitivas geométricas também é conhecida na literatura de técnica baseada na extração e correspondência de característica. De uma forma sucinta, essa técnica extrai características definidas por primitivas geométricas das imagens e usa a correspondência com outras imagens como entrada no sistema, tratando as incertezas com filtros estatísticos. A segunda abordagem utiliza a imagem como um todo ou apenas parte dela, calculando a correspondência entre esta através da comparação entre as intensidades de *pixels* e tratando as incertezas com métodos de otimização. Ela também é conhecida como técnica baseada em textura. A única restrição é que esse ambiente possua regiões planares e com texturas homogêneas, uma vez que o uso de registro direto de imagem compara as intensidades dos *pixels* das regiões que devem ser alinhadas. A Figura 5 ilustra a detecção de características baseado em textura.

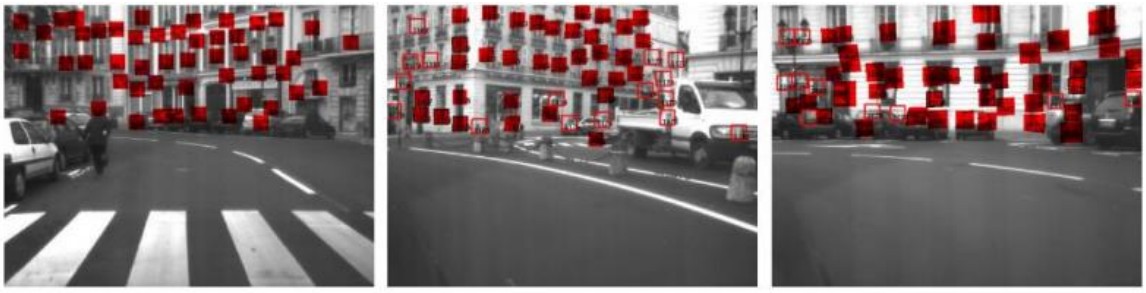


Figura 5 - Detecção de característica baseado em textura
Fonte: BRITTO, KOERICH, 2009

A escolha da abordagem empregada depende da finalidade do sistema proposto. A técnica baseada em extração de características é mais adequada quando as imagens

possuem objetos distintos e de fácil observação. Já os métodos baseados na intensidade dos *pixels* são recomendados quando as imagens não são tão ricas em detalhes (ZITOVÁ & FLUSSER, 2003). A Figura 6 ilustra um ambiente de difícil detecção e extração de características. A Figura 7 ilustrara um ambiente interno de fácil detecção e extração de características.

Figura 6 - Ambiente de difícil detecção e extração de característica
Fonte: http://pt.wikipedia.org/wiki/Parque_do_Carmo

Figura 7 - Ambiente de fácil detecção e extração de características
Fonte: Autoria própria.

De acordo com Santana e Medeiros (2011, p. 01), os principais desafios em SLAM Visual são:

a) como detectar características em imagens;

b) como reconhecer que uma característica detectada é ou não a mesma que uma detectada previamente;

c) como decidir se uma nova característica detectada será ou não adotada como uma nova marca;

d) como calcular a posição 3D de marcas a partir de imagens 2D; e

e) como estimar a incerteza associada com os valores calculados.

Graças ao avanço da computação visual, algoritmos de visão computacional podem ser utilizados para desenvolver estratégias especificas para superar todos esses problemas.

Nos próximos tópicos serão apresentados procedimentos para solucionar os problemas citados a cima abordando as técnicas de SLAM Visual baseado na extração e correspondência de característica. Os módulos que trabalham o processamento de imagem são comumente designados pela literatura como Módulo Visual e o módulo que trabalha com as estimativas de SLAM são designados Módulos SLAM. O processo para estimativas no processo de SLAM Visual baseado na extração de características pode ser representado de um modo geral pela Figura abaixo:

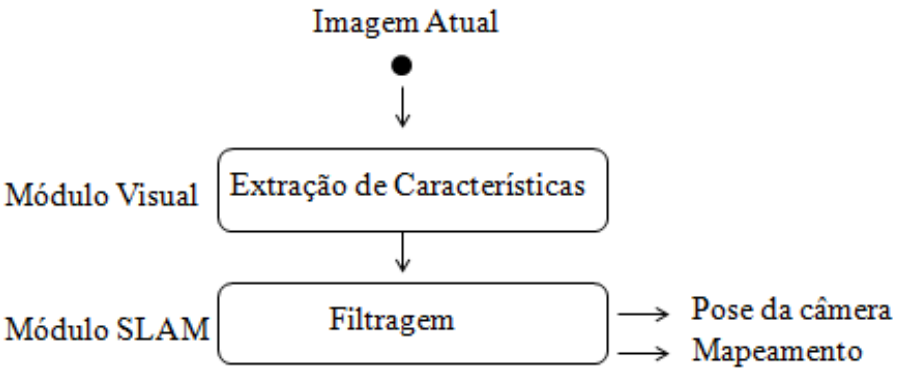


Figura 8 - Processo SLAM Visual com extração de características
Fonte: Autoria própria.

A imagem obtida pelo sistema é processada no Módulo Visual para extrair delas características a serem utilizadas pelo sistema como marcos. À medida que novas características são extraídas, estas são correlacionadas com outras características para verificar se são as mesmas em diferentes imagens. Através dos pares de pontos

correlacionados em imagens distintas, torna-se possível determinar tanto a translação e rotação da câmera quanto a sua localização no ambiente. Baseado nisso, é possível dividir o processo de obtenção dos parâmetros (estimativas) em três etapas principais: extração de características, correspondência de dados e estimação paramétrica.

Como dito na secção 1.1, erros inerentes aos sensores e ao próprio sistema robótico precisam ser tratados. O Módulo de SLAM recebe como entrada os dados resultantes do Modulo Visual, e trabalha com um estimador recursivo para determinar os estados de posição e localização da câmera.

1.2.1. SISTEMAS DE VISÃO

Para percepção do ambiente, a literatura classifica os sistemas de visão em duas categorias, são elas: Visão Monocular e Visão Estéreo. A seguir é realizada uma breve explanação de cada uma delas.

1.2.1.1. SISTEMA DE VISÃO MONOCULAR

Sistemas de Visão Monocular utilizam como sensor uma única câmera, não fornecendo de imediato às informações de profundidade; devido a isso, esses sistemas possuem o problema de não estimar com alta precisão as coordenadas 3D dos marcos a partir de uma imagem. Um exemplo a ser citado utilizando sistema de visão monocular é o trabalho apresentado por Davison (2007), este nós apresenta um algoritmo em tempo real capaz de recuperar trajetórias 3D de uma única câmera. Santana (2011) apresentou uma técnica de SLAM para ambientes planos utilizando como marcos as linhas presentes no chão em um sistema de visão monocular.

1.2.1.2. SISTEMAS DE VISAO ESTÉREO

Os sistemas estéreos utilizarem duas ou três câmeras para capturar as imagens do ambiente, assim é possível capturar imagens de vários ângulos de um mesmo marco, e diferente da visão monocular, através do uso de técnicas de triangulação é possível obter de imediato as informações de profundidade dos marcos (coordenadas 3D). No entanto, esses tipos de sistemas apresentam um maior custo computacional.

1.2.2. CARACTERÍSTICAS

Para serem utilizadas com marcos, as imagens precisam passar por um processo designado extração de características. O processo se constitui pela extração de um conjunto de características (*features*) de cada imagem. Essas características são projeções na imagem de marcos visuais presentes na cena visualizada e, geralmente, se apresentam como linhas, regiões ou pontos que se sobressaem em relação a sua vizinhança (SILVA, 2011, pg.13).

Essas características são também denominadas pela literatura como pontos de interesse, sendo selecionados por um método definido como detector de pontos.

De acordo com Jain et al (1995 apud BRITTO KOERICH, 2009,p. 16), as características utilizadas em sistemas de visão são classificadas em :

- **Características Globais**: descritores obtidos com base em todos os pontos de uma região, sua localização, intensidade e relações espaciais. Exemplos: área, perímetro, descritores de Fourier e momentos (por exemplo: Hu e Zernike).
- **Características Locais**: normalmente calculadas a partir do contorno de um objeto ou a partir de pequenas regiões na imagem. Curvatura de contorno ou superfície, segmentos de contorno, código de cadeia, detecção de cantos e concavidades são exemplos de características locais.
- **Características Relacionais**: posições relativas de entidades diferentes como regiões, contornos fechados, transições, concavidades, ou qualquer outra característica local. Distâncias entre características e medidas de orientação relativa.

A Figura 9 ilustra a detecção de características locais em uma imagem.

Figura 9 - Detecção de características locais (linhas)
Fonte: Autoria própria.

As características, de acordo com Tuytelaars e Mikolajczyk (2006, apud GUIZILINI, 2008, p. 25), devem possuir as seguintes propriedades:

- **Repetibilidade:** Alterações nas condições de observabilidade de um objeto, como mudança de luminosidade ou de ponto de vista, não devem alterar as características detectadas. Pode ser obtida de duas maneiras:
  - **Invariância:** As alterações sofridas pela imagem são modeladas de forma a permitir uma compensação antes das características serem detectadas.
  - **Robustez:** Diminuição da rigorosidade na detecção das características, o que pode comprometer a precisão.
- **Quantidade:** Deve existir em grandes quantidades.
- **Precisão:** Os padrões que a definem devem ser precisamente determinados.
- **Eficiência:** Sua detecção deve ser rápida.

A escolha do método de extração de característica é um dos fatores mais importantes para a construção de um sistema de visão ou reconhecimento de padrões, uma vez que o desempenho do sistema esta ligado diretamente ao poder de discriminação do conjunto de características escolhido**.** Isso porque, a interpretação ou entendimento de

uma cena demanda o reconhecimento de seus objetos. E para o reconhecimento de objetos ou padrões contidos em uma cena é preciso a definição de um método de extração de características (BRITTO & KOERICH, 2009).

1.2.3. DESCRITOR DE CARACTERISTICAS

Depois que uma característica é localizada na imagem, esta deve ser modelada de alguma forma, para que ela possa ser armazenada e utilizada posteriormente na correspondência com outra característica. Essa modelagem é feita por uma técnica denominada descritor. O principal objetivo do descritor e detectar as propriedades principais de uma característica de maneira que possa ser correspondido e reconhecido em correspondências posterior com características obtidas de outras imagens. Esse vetor de característica é normalmente formado por descritores locais ou globais. Descritores locais provaram ser bem sucedidos em aplicações que requerem associação e reconhecimento de imagens. De acordo com (BUENO, 2011, pg. 06):

> "Descritores locais de imagem, computados em regiões ao redor de pontos de interesse, são usados de maneira eficaz em diversas aplicações de visão computacional e processamento de imagens, tais como: reconhecimento de objetos, reconhecimento de texturas, recuperação de imagens por conteúdo, mineração de dados de vídeo, construção de panoramas, reconstrução 3D, calibração de câmera, recuperação de imagens semi-replicas, entre outros".

Segundo Neto, Silva e Blasechi (2014) os descritores são utilizados para comparar a mesma região em diferentes imagens permitindo assim a correspondência de imagens. Isso diz respeito à característica de repetibilidade do descritor, que nada mais é do que a capacidade que o descritor tem de identificar confiantemente os mesmos pontos de interesse em condições de exibição diferente, permitindo identificar características já observadas anteriormente, possibilitando assim o reconhecimento de uma cena ou mesmo a um objeto especifica dentro da cena.

Em SLAM isso é importante para o que chamamos de associação de dados, que consiste em fazer a correspondência de um dado já adquirido com um já inserido no

sistema. No SLAM isso que dizer que, se um marco já observado anteriormente é reobservado, este deve ser identificado, evitando assim erros nas estimativas.

A maneira de como uma característica é descrita é importante, e a escolha do descritor vai depender do tipo de aplicação a ser desenvolvida. De acordo com Tuytelaars e Mikolajczyk (2006, apud GUIZILINI, 2008, p. 26), um descritor deve possuir as seguintes características:

- **Repetibilidade**: dadas duas características com propriedades similares, os seus descritores resultantes também devem ser similares entre si.
- **Distinção**: o descritor deve conseguir armazenar informações suficientes para ser capaz de distinguir entre duas características diferentes.
- **Compactação**: informações redundantes ou altamente correlacionadas não devem ser armazenadas, visando com isso poupara memória e tempo de processamento durante a correspondência.
- **Eficiência**: as informações contidas em um descritor devem ser armazenadas de maneira eficiente, permitindo um fácil acesso e utilização.

Além das características descriminantes das imagens, o descritor também possui informações de coordenadas 2D da imagem, uma orientação e uma escala. Geralmente os descritores são determinados utilizando uma região em torno da primitiva. A seleção de características (pontos de interesse) pode ser ilustrada na seguinte Figura:

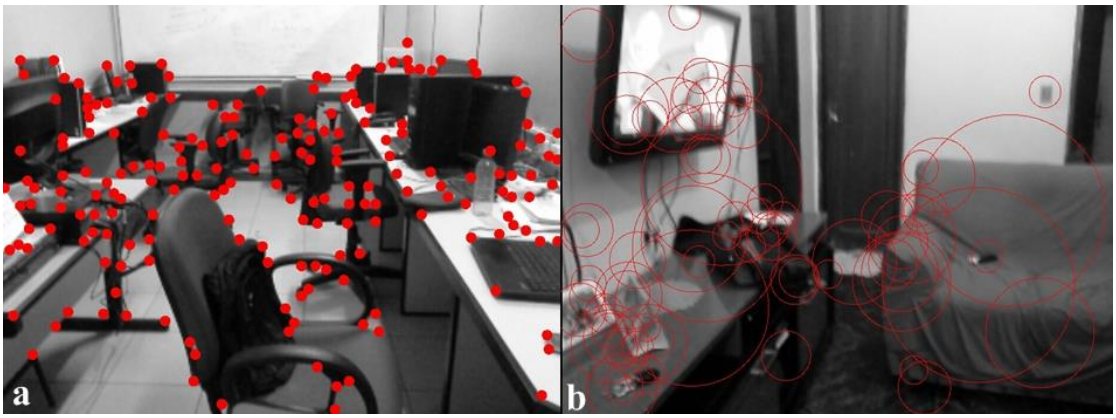


Figura 10 - a) Detecção de Corner b) Detecção de Scale Space
Fonte: Autoria própria.

O descritor é utilizado no sistema de SLAM como marco, sendo assim, este deve ser invariante a rotação e escala, possibilitando que o descritor possa ser reconhecido de qualquer ângulo. Bay, Tuytelaars e Gool (2006) apontam a repetibilidade como a propriedade mais valiosa de um descritor. Vários são os métodos na literatura para detecção e extração de características em imagens, entre eles, os mais citados como referencia para descritores locais são: SIFT (*Scale Incariant Features*) e o SURF (*Speeded Up Robust Features).*

1.2.4. CORRESPONDENCIA DE IMAGENS

A correspondência de dados, também denominado como o problema de re-observação de marcos, consiste em fazer correspondência entre descritores, buscando pontos candidatos a serem seus equivalentes com outro descritor. “A comparação de pontos é baseada na similitude dos descritores correspondentes” (GONZÁLES, 2010, p.59).

Geralmente, a correspondência leva em consideração que pixels correspondentes têm intensidade semelhante, no entanto, vários são os pixels com intensidade idêntica em uma imagem, o que leva a descritores considerar os pixels vizinhos ao ponto selecionado na imagem, definindo uma janela. Então a correspondência entre pontos é determinada utilizando medida de semelhança, que é aplicado em janelas ao redor dos pontos chave selecionado. Muitas das aplicações de Visão Computacional exigem a identificação de elementos repetitivos entre duas imagens (GONZÁLES, 2010, p.59).

A Figura abaixo mostra o resultado da extração e associação de pontos características selecionados em duas imagens distintas.

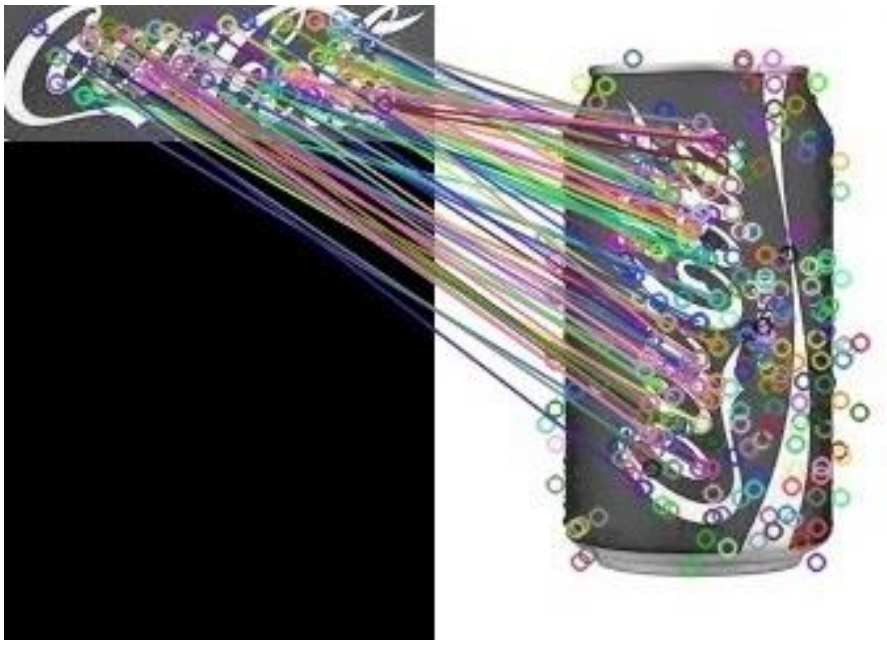


Figura 11 - Correspondência entre duas imagens através da técnica SURF
Fonte: Autoria própria.

Os descritores são vetores que podem ser comparados utilizando-se a distância entre vetores, por exemplo, à distância Euclidiana e a distância de Mahalanobis. Os candidatos à melhor correspondência são geralmente pontos próximos, de maneira que o melhor candidato é o ponto que apresenta a menor distancia (GONZÁLES, 2010).

A busca por correspondências entre imagens pode ser dividida em três partes (PEREIRA, SANTOS AND COZMAN, 2012, p.3):

1. Encontrar pontos chaves na imagem. Definindo pontos de interesse como aqueles que sejam encontrados em diferentes condições de imagem.
2. A vizinhança dos pontos de interesse é representada por um vetor de características ou descritores, os quais devem ser distintivos e robustos ao ruído, às deformações geométricas da imagem e aos erros.
3. Comparar os vetores descritores das imagens. Essa comparação é geralmente baseada na distancia entre vetores.

A primeira e a segunda etapa diz respeito ao que já foi mencionado na seção anterior, que é localizar as características em imagens (discriminar suas principais características) e representa-las de maneira a permitir sua correspondência posterior. A toda nova detecção de marcos, este é comparado com os marcos já inseridos no sistema,

para verificar se é um novo marco ou se esta reobservando um marco já detectado anteriormente.

Durante o processo, pontos instáveis (*outliers*) são detectados levando a falsas correspondências, o que pode ser um fator crítico ao sistema de visão computacional, por isso, o sistema deve tratar as falsas correspondências. Uma das soluções mais simples para eliminação de *outliers* é apresentada no trabalho de Lowe (2004), onde se é utilizado um método de comparação entre a menor distancia com a segunda melhor distancia, selecionando somente correspondentes próximos de um limiar.

No trabalho apresentado por Lowe, foi rejeitada correspondência com distância superior a 0.8, o que elimina 90% das falsas correspondências e descartando menos do que 5 % das combinações corretas. A Figura abaixo ilustra falsas correspondências entre imagens:

Figura 12 - Erro na associação de dados utilizando a técnica SIFT
Fonte: Autoria própria.

Geralmente, as falsas correspondências entre pontos chaves são tratadas por técnica de restrição geométrica, como RANSAC (do inglês, *RAndom SAmple Consensus*), uma técnica robusta proposta por Fischler e Bolles (1981). Ainda de acordo com Fischler

e Bolles, o algoritmo de RANSAC é capaz de interpretar informações que contem uma porcentagem significativa de erros. De forma sucinta, o RANSAC é aplicado em sistemas que necessitam estimar um modelo (como rotação e translação de câmera) utilizando as observações ruidosas provenientes de imagens.

Vale ressaltar que técnicas de eliminação de falsas correspondências é aplicado após todas as correspondência serem efetivadas. Após a finalização da etapa de correspondência, um modelo específico de matriz essencial precisa ser modelado. Essa matriz essencial permite extrair a pose relativa entre as imagens, e a partir dessa matriz, métodos são utilizados para a extração de parâmetros (intrínsecos e extrínsecos) da câmera. Neste trabalho não será abortado esses métodos, mais sobre este assunto pode ser visto no trabalho apresentado por Nistér (2004), ao qual nos apresenta um algoritmo denominado de Algoritmo dos Cinco Pontos para calculo da matriz essencial.

1.2.4.1. SCALE-INVARIANT FEATURE TRANSFORM (SIFT)

O SIFT foi desenvolvido por Lowe (2004). É definido por Lowe como sendo uma técnica para processamento de imagens que permite a detecção e extração de descritores locais razoavelmente invariantes a mudanças de iluminação, ruído de imagem, rotação, escala e pequenas mudanças de perspectiva.

A técnica é composta por duas partes: o detector e o descritor. O detector diz respeito à parte do algoritmo responsável pela detecção de características na imagem (pontos de interesse). O descritor diz respeito à parte de descrição das características.

De acordo com Lowe (2014, pg.20) o SIFT é formado por quatro processos principais, onde as duas primeiras etapas dizem respeito ao detector, e as duas últimas ao descritor. São elas:

- **Detecção de extremo no espaço escalar:** A primeira etapa procura por todas as escalas e localizações de uma imagem. É implementado eficientemente utilizando-se uma função de diferença de filtros gaussianos para identificar pontos de interesse invariáveis à escala e rotação;

- **Localização de pontos chave:** Para cada extremo detectado, um modelo detalhado é ajustado para determinar a localização e escala. Pontos chaves são selecionados baseando-se em medidas de estabilidade.

- **Atribuição da orientação:** Uma ou mais orientações são atribuídos a cada ponto chave através dos gradientes locais da imagem. Todas as operações futuras são realizadas com relação a dados da imagem transformados em relação à orientação, escala e localização de cada ponto chave, proporcionando assim invariância a estas transformações.

- **Descritor dos pontos chave:** Os gradientes locais da imagem são medidos em uma região em torno de cada ponto chave. Estas medidas são transformadas em uma representação que permite níveis significativos de distorções e alterações na iluminação.

Segundo Junior (2008), Lowe (2004) e Gonzáles (2010):

- A etapa inicial procura por pontos invariantes a transformações na escala da imagem. Esta característica possibilita a detecção posterior desses mesmos pontos com a câmera próxima ou distante do objeto ou cena. Esta etapa procura por pontos tem todas as escalas utilizando uma função denominada de espaço de escala, que é modelada através da função gaussiana.

- A segunda etapa consiste em selecionar pontos para serem pontos chave, rejeitando os que possuem baixo contraste (seriam sensíveis a ruídos) e próximos à extremidade. A filtragem de pontos sensíveis a ruído é feita utilizando a expansão de Taylor da função Diferença de Gaussiano (DoG).

- A terceira etapa consiste em definir a orientação de cada ponto chave através dos gradientes locais da imagem.

Belo (2006, p. 63) diz que ao se atribuir uma orientação para cada ponto chave, pode-se representar os descritores em relação a esta orientação, conseguindo-se assim invariância quanto à rotação.

Após a seleção dos pontos chave, invariantes a rotação e escala, é criado um descritor para cada um desses pontos baseando-se nos histogramas da região em relação do ponto. Para obtenção do histograma é preciso rotacionar os pontos vizinhos e as

orientações dos gradientes, tendo assim um descritor invariante a rotação. O descritor é representado então por um vetor, onde o valor de cada posição do vetor se refere a uma das direções dos histogramas (JUNIOR, 2008).

As etapas do SIFT podem ser vista com mais detalhes no trabalho de Lowe (2004), que nos apresenta com detalhes as técnicas em cada etapa e o porquê do uso dessas técnicas.

Na Figura seguinte podemos observar o gradiente da imagem e o descritor de pontos chave.

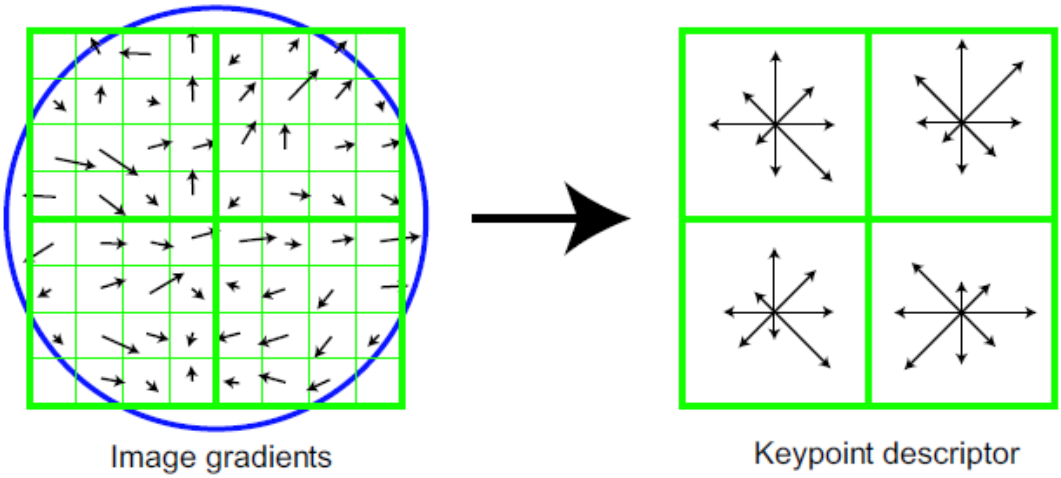


Figura 13 - Gradiente da imagem e descritor de pontos chave
Fonte: Lowe, 2004, p.15

1.2.4.2. SPEEDED-UP ROBUST FEATURES (SURF)

O SURF, proposto por Bay, Tuytelaars e Gool (2006), apresenta um descritor invariante a escala e a rotação em imagem. Assim como o SIFT, é formado por um detector e um descritor.

O SURF utiliza como técnica para a localização dos pontos chave a matriz hessiana, de acordo com Bay, Tutelaars e Gool (2006), o detector foi baseado na matriz hessiana por causa de seu bom desempenho em tempo de computação e precisão. Para a orientação é utilizada a transformada Haar (*Haar wavelet*). O descritor é formado a partir

da distribuição de intensidade em torno dos pontos de interesse, semelhante à informação de gradiente extraída pelo SIFT.

O SURF foi inspirado pelo algoritmo SIFT, no entanto, seus criadores garantem um processamento muito rápido na detecção e comparação de pontos de interesse.

De acordo com Santos:

> A técnica SURF surgiu como um desafio de criar um detector e descritor de características, que, em comparação ao estado da arte, fosse rápido de calcular sem sacrificar desempenho. Para isso, foi simplificado o processo de detecção mantendo a precisão e reduzindo o tamanho do vetor de descrição de forma a mantê-lo suficientemente destino (SANTOS, 2010, p. 09**).**

Os autores também destacam a dimensão do vetor descritor, que segundo eles, reduz o tempo para calculo de características e correspondência, e simultaneamente aumenta a robustez (BAY, TUYTELAARS, & GOOL, 2006).

O SURF se destaca pela velocidade na detecção, viabilizando assim seu uso em aplicações de tempo real. Segundo Gil (2009, apud SANTOS, 2010, p. 23), "O detector e descritor SURF é a melhor técnica a ser utilizada no SLAM visual, porque tem custo computacional reduzido, facilitando a extração de característica em tempo real". Voltolini e Kim destacam que:

> Para acelerar os cálculos, SURF utiliza imagem integral para calcular respostas a wavelet de Haar em sentidos horizontal e vertical. Estas respostas são amostradas numa grade regular e as suas somatórias (com e sem sinal) servem como descritores. Este processo leva a resultados mais rápidos, porém menos acurados (VOLTOLINI & KIM, p.1).

O vetor de característica do SURF é composto por 64 elementos, contra 128 do SIFT. A dimensionalidade do vetor influencia no tempo de execução para as estimativas, sendo assim, para uma maior agilidade computacional é preciso de vetores com menor dimensão (Bay et al. 2008).

Santos (2010) resume a obtenção do descritor SURF em 4 etapas :

- **Detecção dos pontos de interesse:** É utilizada uma matriz Hessina básica de aproximação ao qual são detectadas regiões com aparência de borrões onde o determinante é o máximo.

- **Representação dos pontos de interesse no espaço escalar:** O espaço escalar é dividido em oitavas. Cada uma dessas oitavas representa uma serie de mapas filtrados que foram obtidos através da convolação da imagem inicial com um filtro com crescimento de tamanho.

- **Localização dos pontos de interesse:** A localização na imagem e dada através de escalas, onde o máximo do determinante da matriz Hessiana é interpolado nos espaços da imagem e da escala. Isto é importante porque a diferença em escala entre as primeiras camadas de cada oitava é relativamente grande.

- **Descrição e correlação de pontos de interesse:** o vetor de descrição representa a distribuição de intensidade dentro da vizinhança dos pontos de interesse. É identificada uma orientação reproduzida baseada em informações de uma região circular em torno dos pontos selecionados. Logo em seguida, é construída uma região quadrada alinhada a orientação selecionada e daí é extraído o descritor SURF.

1.2.4.3. SURF X SIFT

O SIFT tem uma melhor acurácia em relação ao SURF, no entanto, pesquisas demonstram que o SURF tem desempenho semelhante ao SIFT, enquanto ao mesmo tempo é muito mais rápido. Por outro lado, quando a velocidade não é um fator critico no sistema, o SIFT supera o SURF.

A tabela a seguir mostra a diferença da representação do espaço escalar no SIFT e no SURF (SANTOS, 2010, p.13).

Técnicas	Vantagens	Desvantagens
SIFT	Invariante à translação, rotação e efeitos de escala; Pouco sensível a variações na luminosidade;	Elevada complexidade computacional na extração de características
SURF	Mais robusto que o SIFT a variações de escala; Extração de características mais rápida que no SIFT	Normalmente seleciona poucos pontos de interesse

Quadro 01 - Diferenças entre o SIFT e o SURF
Fonte: SANTOS, 2010, p. 13

1.3. MÓDULO SLAM

É no módulo SLAM que a posição atual da câmera e obtida ao mesmo tempo em que são estimadas as coordenadas dos marcos visualizadas na imagem. A seguir, falaremos de umas das técnicas mais abordadas na literatura, O Filtro de Kalman.

1.3.1. TECNICAS DE FILTRAGEM

As coordenadas e orientação dos pontos-chave obtidos no processo do Módulo Visual são utilizadas como entrada para inicializar o Módulo SLAM. Como já visto as informações obtidas pelos sensores robóticos tendem a ser caracterizado por incertezas e ruídos, sendo assim, de uma forma sucinta, as técnicas de filtragem tem como objetivo utilizar medições contaminadas por ruídos e outras incertezas e gerar resultados que tendem a se aproxima dos valores reais das medidas e valores associados. O trabalho abordara de forma simplória o Filtro de Kalman.

1.3.1.1. FILTRO DE KALMAN

O Filtro de Kalman (*Kalman Filter*) foi criado por Rudolf Kalman na década de sessenta, na área de engenharia elétrica voltada a teoria do controle de sistemas. O Filtro de Kalman tem como objetivo estimação de estados de um sistema dinâmico linear e discreto no tempo. De acordo com Silva et al. (2011, p.06):

> O Filtro de Kalman é um algoritmo recursivo baseado na teoria de análise de sinais, o qual minimiza o erro médio quadrático de forma recursiva, ou seja, considera toda informação até o momento t, para a estimação ótima do vetor de estado (vetor contendo as variáveis explicadas do modelo).

O Filtro de Kalman é um método estocástico, onde sistemas apresentam processos não determinísticos com origens em eventos aleatórios. Supõe-se que medições de um sistema estejam sujeito a ruídos, a incertezas nas informações obtidas do sistema real, assim, o filtro de Kalman utiliza técnicas probabilísticas para ajustar os parâmetros da estimativa a cada nova medição, obtendo a estimativa do erro a cada atualização, resolvendo problemas relacionados a incertezas nas estimativas obtidas pelo sistema, tendo como resultados valores que se aproximam do sistema real.

O filtro de Kalman produz estimativas dos valores reais de grandezas medidas e valores associados predizendo um valor, estimando a incerteza do valor predito e calculando uma média ponderada entre o valor predito e o valor medido (CRUZ, 2013, p.13)". O KF tende a apresentar uma melhor estimativa, pois a media ponderar apresenta uma melhor estimativa de incerteza.

Com o objetivo de representar matematicamente as incertezas presentes nos sistemas reais, é utilizado um ruído Gaussiano Branco. Matematicamente, através do Teorema do Limite Central, é possível provar que, como nos sistemas reais, o ruído é causado por um somatório de pequenas fontes de perturbação, podendo ser representado por uma Função densidade de probabilidade (ABREU, 2008). Sendo assim, o Filtro de Kalman trata de estimativas contaminadas por adição de ruído branco com distribuição de probabilidade Gaussiana.

1.3.1.2. O ALGORITMO DE FILTRO DE KALMAN

Algoritmo de KF é formado por equações matemáticas, que pode ser agrupada em dois grupos distintos, são elas: equações de atualização de tempo e equação de atualização de medição ou de correção. De acordo com Ramos (2012, p.25):

> As equações de predição são responsáveis pelo avanço das variáveis de estado e das covariâncias no tempo para se obter as estimativas a priori. As equações de atualização trazem informações das medições e as incorporam nas estimativas a priori para obter um ganho, que é utilizado para fazer a estimação posterior.

De uma forma sucinta, as equações da etapa de predição utilizam a estimativa do estado no tempo anterior para obter uma estimativa no tempo atual, sem incluir as informações do estado atual. Na fase de atualização, a estimativa obtida na etapa de predição é combinada com a observação atual para melhorar a estimativa do estado. Suas equações matemáticas compõem um processo recursivo ótimo capaz de reduzir a soma dos quadrados das diferenças entre os valores reais e os estimados (ABREU, 2008, p.10). A Figura 14 ilustra as fases do algoritmo e a transação das fases com informações de equações utilizadas em cada uma das etapas.

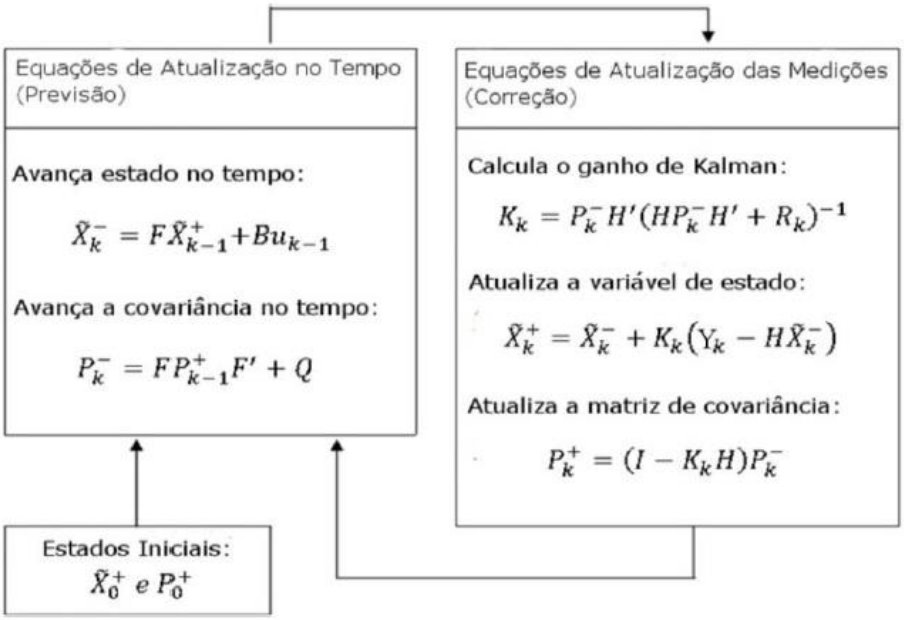


Figura 14 - Fases do Filtro de Kalman
Fonte: RAMOS, 2011, p.26

Como dito inicialmente, o Filtro de Kalman e aplicado a modelos lineares, e em geral, sistemas dinâmicos apresentam modelos de sistema não lineares, por isso se tem a necessidade de estender o Filtro de Kalman para linearização (em torno da estimativa do

estado) do sistema. O Filtro de Kalman Estendido se caracteriza por ser uma extensão do Filtro de Kalman para aplicação em sistemas não lineares, onde a não linearidade do sistema é através de expansões em series de Taylor. Alem do FKE (Filtro de Kalman Estendido), a literatura também apresenta o Filtro de Kalman Unscented para sistemas não lineares.

2 METODOLOGIA

Esta seção tem por objetivo apresentar as ferramentas e métodos adotados para a solução do problema proposto.

A seção 2.1 resgata a definição do problema para uma melhor compreensão da importância do algoritmo proposto. A seção 2.2 apresenta as etapas necessárias para a implementação do algoritmo proposto e a seção 2.3 descreve os recursos utilizados.

2.1 VISÃO GERAL

A presente pesquisa teve inicio com o Projeto VANT (Veiculo Aéreo não Tripulado) cujo objetivo primordial é o desenvolvimento de um VANT para monitoramento, inspeção e reconhecimento em áreas de difícil acesso. A importância deste trabalho está em agregar conhecimento inicial do processo de SLAM, para posteriormente a técnica ser estudada com foco no seu desenvolvimento para o protótipo da aeronave.

Este trabalho tem como objetivo apresentar as principais etapas do processo de SLAM baseado em visão, de uma forma simples e objetiva, visando o fácil entendimento do problema e de suas principais etapas para pesquisadores que estejam iniciando na área.

Como já citado no trabalho, o termo SLAM se refere à Localização e Mapeamento Simultâneo. Isso diz respeito à capacidade que robôs devem possuir para serem verdadeiramente autônomos, pois para cumprir um determinado objetivo é necessário o conhecimento do ambiente em que está inserido, além de saber se localizar dentro dele. Para isso, o sistema deve possuir módulo que analisa certos dados de tal maneira que a partir de sua interpretação se tenha como resultados as estimativas de mapeamento e localização para assim poder criar uma representação visual (mapa do ambiente e localização no mapa) dessas estimativas. Uma das abordagens utilizadas para estimar os estados de localização e mapeamento são os marcos.

- Marcos são características que podem ser facilmente re-observadas e distintas do resto do ambiente. São usados pelo robô como referencia para descobrir onde esta e se localizar.

Em SLAM, os marcos podem ser representados por uma cena ou por objetos que compõem a cena, assim um conjunto de cenas ou objetos define um lugar no ambiente. Quando imagens são utilizadas para as estimativas de SLAM, estas devem passar por um processo de extração de características, essas características extraídas são utilizadas pelo sistema como marcos, estes marcos em sistemas de visão são denominadas de descritores, que nada mais são um conjunto de características extraídos de uma imagem.

A importância em corresponder às características esta ligada diretamente a capacidade de reobservação de marcos (vide secção 1.1.1), essa reobservação de marcos caracteriza uma instancia do problema de SLAM.

- A reobsevação de um marco pela câmera em perspectivas diferentes é evidenciada pela correspondência repetida dos referentes pontos detectados em outras imagens.

O objetivo deste trabalho pode ser assim resumido:

- Extrair características de imagem e determinar seus respectivos descritores;
- Corresponder imagens de maneira que possa ser reconhecido Cena com cena e Objetos em uma cena.

2.2 FLUXO DO ALGORITMO PROPOSTO

O fluxo para o desenvolvimento do algoritmo do presente trabalho é ilustrado na Figura 15:

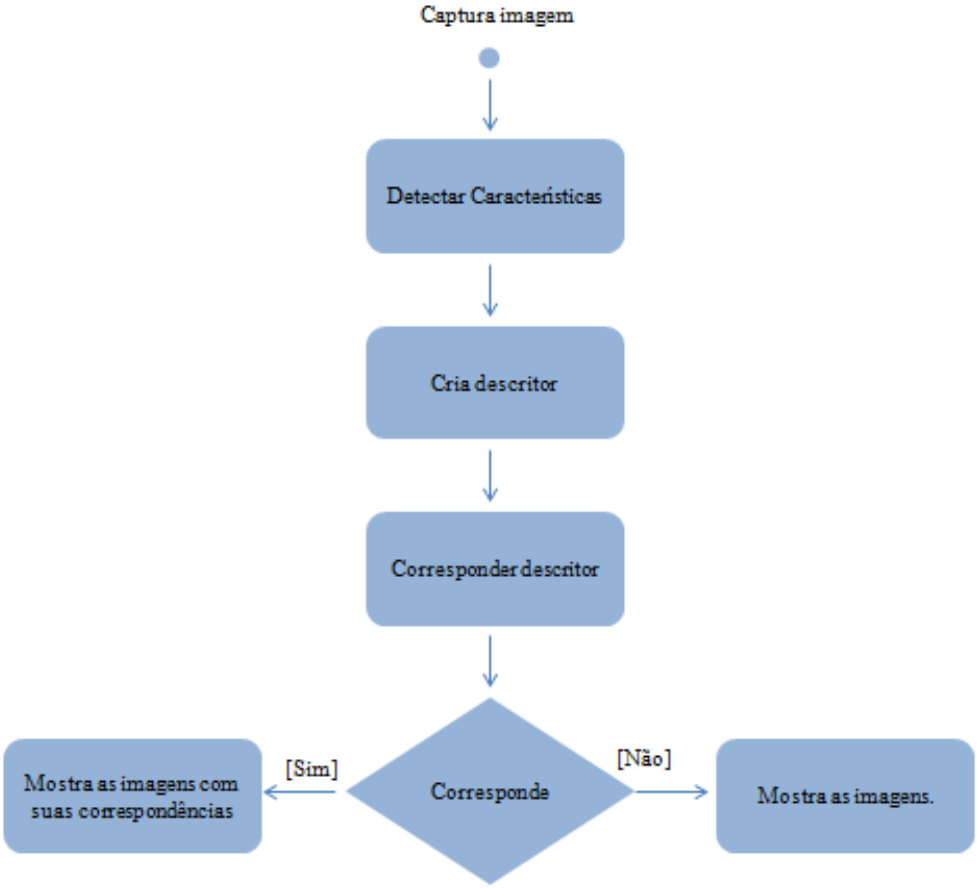


Figura 15 - Fluxo do algoritmo proposto
Fonte: Autoria própria

A primeira etapa consiste na obtenção da imagem para correspondência, neste caso as imagens foram salvas na pasta do projeto criado e o caminho da imagem foi colocado direto no código fonte. A segunda etapa consiste em selecionar os pontos características nas imagens e crias seus respectivos descritores utilizando a técnica SURF. A terceira etapa consiste em corresponder às imagens, de modo que se identifiquem pontos correspondentes, da seguinte maneira:

- A primeira imagem, designada como A e correspondida com a segunda imagem designada como B para correspondência e identificar se a imagem A tem pontos correspondidos com a imagem B.
- Se houver correspondência entre as imagens, o sistema abre uma janela contendo as duas imagens com seus respectivos pontos correspondentes, caso contrario, as duas imagens são visualizadas sem correspondência.

2.3 RECURSOS UTILIZADOS

Para implementação do algoritmo com a técnica SURF foi utilizada a biblioteca OpenCv na linguagem de programação C++ na IDE Visual Studio 2013.O computador utilizado foi um notebook com Sistema Operacional Windows equipado com 4GB de RAM, placa de vídeo dedicada e HD de 500GB e processado Intel Core i5 com 2.50GHz. As imagens utilizadas no experimento foram em parte obtidas do Google imagens e outras a partir da câmera de um Tablet com resolução de 5 megapixels.

Para apresentação da técnica de extração de características, foi escolhida a técnica SURF, por apresentar, de acordo com a literatura, resultados satisfatórios e maior velocidade de processamento, uma vez que este é um fator critico para sistemas em tempo real.

2.3.1. OpenCV

A OpenCV é uma biblioteca de visão computacional de e software de aprendizado de máquina licenciado pela BSD (*Berkeley Software Distribution*). Seu objetivo é proporcionar uma infraestrutura comum para aplicações de visão por computador.

A biblioteca conta com mais de 2500 algoritmos, os quais podem ser usados para detectar e reconhecer rostos, identificar objetos, extrair modelos 3D de objetos, encontrar imagens similares a partir de um banco de dados, reconhecimento de padrões entre outros. Ela suporta as linguagens de C++, C, Pthon, Java e MATLAAB, sendo compatível com os sistemas operacionais Windows, Linux, Android e Mac-OS. Uma vez instalada a biblioteca, deve-se adicionar ao caminho padrão do sistema (PATH) as pastas onde estão os arquivos que serão utilizados.

2.3.2. RECURSOS UTIIZADOS DA BIBLIOTECA OPENCV

As três principais classes disponíveis utilizadas na implementação pode ser observado no quadro abaixo:

Recurso	Função
SurfFeatureDetector	Classe para detectores de características de imagem 2D.
SurfExtractorDescriptor	Classe para calcular descritores para pontos-chave de imagem.
FlannBasedMatcher	Classe para executar a correspondência entre descritores.

Quadro 02 – Principais classes utilizadas
Fonte: Autoria própria.

SurfFeatureDetector dispõe o método detect para realizar a detecção características em uma imagem. O quadro 03 ilustra a instância da classe SurfFeatureDetector e os parâmetros utilizado pelo método detect.

Criando o Objeto	Parâmetro do método
SurfFeatureDetector surf	surf.detect(mat, vector<KeyPoint> keypoints)

Quadro 03 – Método detect
Fonte: Autoria própria.

O método armazena os pontos detectados da imagem em um vetor de keypoints. Keypoints é uma classe de dados para detectores de pontos.

O primeiro parâmetro e um tipo Mat onde é armazenada a imagem a ser processada.

Como se sabe, as imagens são formadas por pixels, e no âmbito computacional as imagens podem ser reduzidas para matrizes numéricas, essas matrizes são formadas por números que representam os valores de intensidade dos pontos de pixels da imagem. OpenCv é uma biblioteca de visão computacional que tem como principal objetivo processar e manipular essa informação. Para isso a OpenCv usa o tipo Mat para armazenar os valores de pixel de uma imagem. Para escrever uma matriz para um arquivo de

imagem, usa-se o imread. O imread tem como parâmetro o nome do arquivo a ser carregado e bandeiras, que especificam o tipo de cor de uma imagem a ser carregada. Os tipo de cor são:

- **CV_LOAD_IMAGE_ANYDEPTH** - Se for definido, o retorno imagem / 32-bit de 16 bits quando a entrada tem a profundidade correspondente, caso contrário, convertê-lo em 8-bit.
- **CV_LOAD_IMAGE_COLOR** - Se for definido, sempre converter uma imagem para a uma cor.
- **CV_LOAD_IMAGE_GRAYSCALE** - Se for definido, sempre converter a imagem para a escala de cinza.

Após detectar as características nas imagens, é necessário criar os descritores dos pontos característicos detectados. O SurfExtractorDescriptor utiliza o método compute para criar o descritor da imagem. O descritor computado é armazenado em uma variável do tipo Mat.

O método compute armazena somente características relevantes dos pontos característicos detectados na imagem, de acordo com critérios estabelecidos pela técnica empregada (vide seção 1.2.4.2). O quadro 04 ilustra o objeto criado da classe e os parâmetros utilizados pelo método compute.

Criando o Objeto	Parâmetro do método
SurfExtractorDescriptor extrator	extractor.compute(Mat, vector<KeyPoint> , Mat);

Quadro 04 – Método compute
Fonte: Autoria própria.

O primeiro parâmetro se refere à variável do tipo Mat onde a imagem foi armazenada; o segundo faz referência ao vetor de pontos chaves criado pelo método detect da classe SurfFeatureDetector e o terceiro parâmetro, também do tipo Mat, é outra variável definida para armazenar o descritor computado pelo método.

Após definidos os descritores da imagem, podemos utilizar o FlannBasedMatcher para fazer a correspondência entre dois descritores. A Figura seguinte ilustra a criação de um vetor de correspondência pelo FlannBasedMatcher.

```
FlannBasedMatcher metCorresondencia;
std::vector< DMatch > vetCorrespondentes;
metCorresondencia.match(descritor1, descritor2, vetCorrespondentes);
```

Figura 16 - Criação de um vetor de pontos correspondentes entre dois descritores
Fonte: Autoria própria.

A classe FlannBasedMatcher dispõe o método match para realizar as devidas correspondências armazenado-as em um vetor do tipo Dmatch. Para isso, elas recebem os descritores criados pelo método compute da classe SurfExtractorDescriptor. DMatch é uma classe para descritores de pontos chaves, que contem, entre outros, um atributo com o valor da distancia entre os pontos características.

3 IMPLEMENTAÇÃO

O código possui três classes principais, são elas:

- **Detector:** classe responsável por detectar pontos salientes em uma imagem. Formalmente, dada uma imagem de entrada, a classe é responsável por detectar e retornar um conjunto de pontos salientes;
- **ExtratorCaracteristica:** classe responsável por criar os descritores a partir dos pontos característicos obtidos na classe Detector;
- **Correspondência:** csta classe tem como objetivo corresponder os descritores definidos na classe ExtratorCaracteristica. De forma sucinta, esta classe implementa um método que determina se cada ponto saliente detectados em uma imagem corresponde ao mesmo ponto saliente em outra imagem.

A Figura seguinte ilustra a estrutura do projeto:

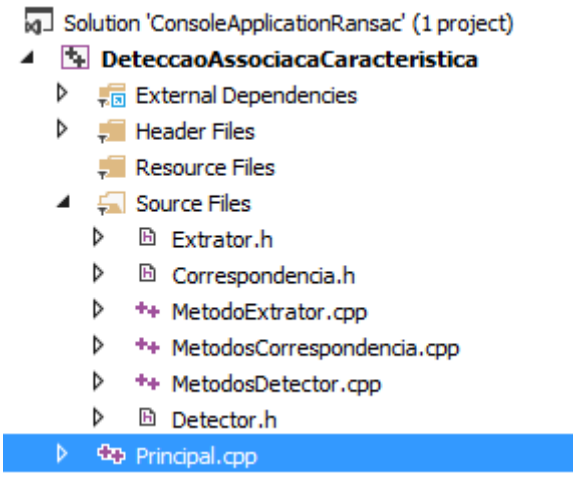


Figura 17 - Estrutura do projeto
Fonte: Autoria própria.

A primeira etapa consistem em detectar pontos chaves nas imagens, essa etapa é implementada pelo método detectarCaracteristica da classe Detector. A figura 18 ilustra a implementação do método:

```
void Detector::detectarCaracteristicas(Mat img, SurfFeatureDetector surfDetector)
{
    surfDetector.detect(img, pontoschavesExtraidos);
    imagem = img;
}
```

Figura 18 - Método para detectar características em imagem
Fonte: Autoria própria

O método detectarCaracteristicas tem como parâmetro um tipo mat e um objeto da classe SurfFeaturaDetector. O método detect do objeto passado em parâmetro computa os pontos chaves e armazena em um vetor de keypoints. Esse vetor é um atributo da classe Detector, como ilustra a Figura a seguir :

```
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/nonfree/nonfree.hpp"
#include "opencv2/highgui/highgui.hpp"

using namespace cv;

class Detector
{

    Mat imagem;
    vector<KeyPoint> pontoschavesExtraidos;
    Mat pontosNaImag;

public :

    void detectarCaracteristicas(Mat img, SurfFeatureDetector surfDetector);
    void desenhaPontosChaves();
    vector<KeyPoint> getpontoschavesExtraidos();
};
```

Figura 19 - Classe Detector
Fonte: Autoria própria

Além do método detectarCaracteristicas, temos mais dois métodos, são eles :

- desenhaPontosChaves : desenha os pontos detectados na imagem;
- getpontoschavesExtraidos: retorna o vetor de pontos chaves.

A implementação de todos os métodos da classe pode ser observada na Figura abaixo:

```
#include "Detector.h"

void Detector::detectarCaracteristicas(Mat img, SurfFeatureDetector surfDetector)
{
    surfDetector.detect(img, pontoschavesExtraidos);
    imagem = img;
}
void Detector::desenhaPontosChaves()
{

    drawKeypoints(imagem, pontoschavesExtraidos, pontosNaImag, Scalar::all(-1), DrawMatchesFlags::DEFAULT);
    imshow("Pontos Selecionados", pontosNaImag);


}

vector<KeyPoint> Detector:: getpontoschavesExtraidos()
{
    return pontoschavesExtraidos;
}
```

Figura 20 - Implementação dos métodos da classe Detector
Autor: Autoria própria.

Como o objetivo deste tópico está em demonstrar a implementação da correspondência entre imagens, os outros métodos não serão detalhados, no entanto, será mostrado o resultado da chamada esses métodos na seção 6.

A etapa seguinte consistem em montar o descritor de características das imagens, a classe responsável por esta etapa e a Extrator. A Figura 21 ilustra a classe, e a Figura 22 ilustra a implementação dos métodos da classe.

```
class ExtratorCaracteristicas
{

    Mat descritorCaracteristica;


public:

    void criarDescritor(Mat imagem, vector<KeyPoint>  pontoschavesSelecionados, SurfDescriptorExtractor extractor);
    Mat getDescritor();

};
```

Figura 21 - Classe ExtratorCaracteristica
Fonte : Autoria própria.

```
#include "Extrator.h"

void ExtratorCaracteristicas::criarDescritor(Mat imagem, vector<KeyPoint>  pontoschavesSelecionados
, SurfDescriptorExtractor extractor)
{

    extractor.compute(imagem, pontoschavesSelecionados, descritorCaracteristica);


}
Mat ExtratorCaracteristicas::getDescritor()
{
    return descritorCaracteristica;
}
```

Figura 22 - Implementação dos métodos da classe ExtratorCaracteristica
Autor: Autoria própria.

O método criarDescritor usa o método compute da classe SurfDescriptorExtrator para computar o descritor dos pontos característicos da imagem e o armazena em uma variável da classe do tipo Mat.Após essa etapa,podemos fazer a correspondência entre descritor. A Figura 23 ilustra a classe responsável pelos métodos de correspondência.

```
class Correspondencia
{

    Mat imagem_correspondencia;
    Mat descritor_a_correspondencia;
    vector< DMatch > vetorPontosCorrespondentes;
    vector< DMatch > vetboasCorrespondencias;
    double max_dist = 0;
    double min_dist = 100;
    vector<Point2f> vtPefimA; vector<Point2f> vtPefimB;

public:

    void criarVetorCorrespondencia(Mat descritor, Mat descritor2, FlannBasedMatcher flmatcher);
    void desenhaCorrespondencia(Mat img, vector<KeyPoint>  pontoschavesA,
    Mat img2, vector<KeyPoint>  pontoschavesB);

    vector< DMatch > getvetPontosCorrespondentes();

    void calculoDistancia();

    void boasCorrespondencias();

    void desenhaBoasCorrespondencias(Mat img, vector<KeyPoint>  pontoschavesA,
    Mat img2, vector<KeyPoint>  pontoschavesB);

    void marcaObjetoCena(Mat img, vector<KeyPoint>  pontoschavesA,Mat img2, vector<KeyPoint>  pontoschavesB);

    void iniHomograph(vector<KeyPoint> pontoschaves_detectadosImg, vector<KeyPoint>
    pontoschaves_detectadosImg2, vector<Point2f> imA, vector<Point2f> imB);
};
```

Figura 23 - Classe correspondência
Fonte: Autoria própria.

O primeiro método consistem em criar o vetor de correspondência, que recebe como parâmetros descritores de cada imagem e um objeto da classe FlannBasedMatcher. A figura a seguir ilustra a implementação do método :

```
#include "Correspondencia.h"
void Correspondencia::criarVetorCorrespondencia(Mat descritor, Mat descritor2, FlannBasedMatcher flmatcher)
{

    flmatcher.match(descritor, descritor2, vetorPontosCorrespondentes);
    descritor_a_correspondencia = descritor;
}
```

Figura 24 - Método responsável por criar o vetor de correspondência entre os descritores.
Fonte: Autoria própria.

O método match calcular os pontos correspondentes e adiciona no atributo da classe vetorPontosCorrespondentes.

Como se sabe, pode ocorrer de se ter falsas correspondências entres os pontos correspondentes detectados pelo FlannBasedMatcher, por isso, o método que os elimina deve ser empregado. O método a seguir trata as falsas correspondências com o método de comparação de menor distancia entre vetores, dado um limiar (vide seção 1.2.4). A Figura a seguir ilustra o método que adiciona ao vetor da classe vetboasCorrespondencias somente os pontos correspondentes no vetor vetorPontosCorrespondente que satisfaça a seguinte condição :

- A distância entre os pontos correspondidos no vetor de pontos correspondentes computado pelo FlannBasedMatcher seja menor ou igual a 0.2.

```
void Correspondencia::boasCorrespondencias()
{

    for (int i = 0; i < descritor_a_correspondencia.rows; i++)
    {
        if (vetorPontosCorrespondentes[i].distance <= 0.2 )
        {
            vetboasCorrespondencias.push_back(vetorPontosCorrespondentes[i]);
        }

    }

}
```

Figura 25 - Vetor de boas correspondências
Fonte: Autoria própria.

Após montar o vetor de boas correspondências, podemos usar o método desenhaBoasCorrespondencias para mostrar as imagens com suas respectivas correspondências. O método responsável por mostrar as imagens é ilustrado na Figura abaixo.

```
void Correspondencia::desenhaBoasCorrespondencias(Mat img, vector<KeyPoint>
]pontoschaves_A, Mat img2, vector<KeyPoint>  pontoschaves_B)
{
      boasCorrespondencias();
      drawMatches(img, pontoschaves_A, img2, pontoschaves_B,vetboasCorrespondencias,
      imagem_correspondencia, Scalar::all(-1), Scalar::all(-1),
      vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
      imshow("Correspondencia", imagem_correspondencia);
}
```

Figura 26 - Método desenhar as boas correspondências entre imagem
Fonte: Autoria própria.

O método tem como parâmetro, as imagens em correspondência e seus respectivos pontos chaves detectados pelo método detectarCaracteristicas da classe Detector (Figura 27), e o vetor resultante do método boasCorrespondencias. Este método usa a função drawMatches para desenhar as correspondências do vetor vetboasCorrepondencias nas imagens passada como parâmetro. O desenho e armazenado na variável imagem_correspondencia e é utilizada na função imshow para mostrar a imagem com suas respectivas correspondências. O método drawMatches permite ainda escolher se quer ou não desenhar pontos característicos na imagem pelo parâmetro DrawMatchesFlags.

A classe principal e responsável pela chamada aos métodos para a realização de cada etapa. Os atributos da classe principal e as instancias das classes podem ser observados na Figura abaixo:

```
#include "Detector.h"
#include "Extrator.h"
#include "Correspondencia.h"
int main()
{
    /** Matrizes para armazenar os pixels das imagens*/
    Mat imagemA;
    Mat imagemB;
    Mat descritorComputadoA, descritorComputadoB;
    /* Define a min. hessiana*/
    int hessiana = 1500;
    FlannBasedMatcher flBuscaCorrespondencia;
    /* Variaveis definidos para armazenar os pontos chaves obtidos */
    vector<KeyPoint>  pontoschavesDetectadosA, pontoschavesDetectadosB;
    /* Instancia da classe SurfFeatureDetector */
    SurfFeatureDetector surf(hessiana);
    /* Instancia da classe SurfDescriptorExtractor */
    SurfDescriptorExtractor descritorExtrator;
    /**
    * Instancia da classe Detector
    */
    Detector encontrarPontosImgA;
    Detector encontrarPontosImgB;
    /* Instancia da classe Extrator */
    ExtratorCaracteristicas extrairCaracteristicasImgA;
    ExtratorCaracteristicas extrairCaracteristicasImgB;
    /* Instancia da classe Correspondencia */
    Correspondencia corresponderImagens;
```

Figura 28 – Classe Principal
Fonte: Autoria própria

A chamada aos métodos descritos anteriormente serão apresentados em três etapas, a fim de facilitar a compreensão. Estas etapas podem ser ilustradas na Figura abaixo:

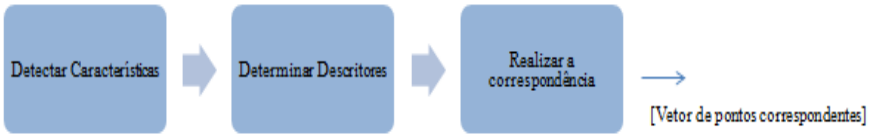


Figura 29 - Etapas para chamada aos métodos
Fonte: Autoria própria.

Inicialmente, definimos uma matriz para os arquivos de imagem, de acordo com a Figura abaixo:

```
imagemA = imread("cocalata.jpg", CV_LOAD_IMAGE_COLOR);
imagemB = imread("sala4.jpg", CV_LOAD_IMAGE_COLOR);
```

Figura 30 - Definição da matriz para as imagens
Fonte: Autoria própria.

O objeto criado da classe detector faz chama ao método detectar características passando a matriz mat das imagens e o objeto surf da classe SurfFeatureDetector.O surf é o responsável por implementar o método de detecção de pontos chaves nas imagens e tem como parâmetro um int que defini o mínimo da matriz hessiana,cujo valor definido foi 1500.A chamada ao método responsável pela detecção de pontos característicos nas imagens é ilustrado abaixo :

```
/* Detecta os pontos caracteristicos das imagens */
encontrarPontosImgA.detectarCaracteristicas(imagemA, surf);
encontrarPontosImgB.detectarCaracteristicas(imagemB, surf);
```

Figura 31 - Chamada ao método detectarCaracteristica
Fonte : Autoria própria.

Após isso, método getpontoschavesExtraidos é utilizados para para retornar os pontos chaves de cada imagem computados pelo método detectarCaracteristica e armazena-los em variáveis da classe principal. A chamada a esse método pode ser ilustrado na Figura seguinte:

```
/* Adiciona os pontos extraidos da imagem*/
 pontoschavesDetectadosA = encontrarPontosImgA.getpontoschavesExtraidos();
 pontoschavesDetectadosB = encontrarPontosImgB.getpontoschavesExtraidos();
```

Figura 32 - Retornando pontos chaves detectados
Fonte: Autoria própria.

A etapa agora consistem em definir o descritor das imagens, para isso, utilizamos o método criarDescritor da classe extrairCaracteristicas.A figura a seguir ilustra a chamada ao método :

```
/* O metodo computa os respectivos descritores de cada imagem */
 extrairCaracteristicasImgA.criarDescritor(imagemA, pontoschavesDetectadosA, descritorExtrator);
 extrairCaracteristicasImgB.criarDescritor(imagemB, pontoschavesDetectadosB, descritorExtrator);
```

Figura 33 - Criando descritores
Fonte: Autoria própria.

O método recebe como parâmetro a matriz da imagem, os pontos chaves detectados pelo descritor Surf, e um objeto da classe SurfDescriptorExtractor responsável por determinar o descritor da imagem.Depois utilizamos o método getDescritor para retornar o descritor computado e o armazenar em uma variável da classe principal para ser utilizado no método que faz a correspondência entre os descritores criados.A Figura abaixo ilustra a chamada ao método :

```
/* Armazena os descritores de cada imagem em uma variavel */
descritorComputadoA = extrairCaracteristicasImgA.getDescritor();
descritorComputadoB = extrairCaracteristicasImgB.getDescritor();
```

Figura 34 - Retornando descritor computado
Fonte: Autoria própria.

A ultima etapa consistem em determinar os pontos correspondentes entre os descritores das imagens criados , depois, mostrar as imagens com suas respectivas correspondências, caso exista. A Figura abaixo ilustra a chamada ao método que criar o vetor de correspondências dos descritores e a chamada ao método que mostra as imagens.

```
/* O metodo computa os respectivos descritores de cada imagem */
 extrairCaracteristicasImgA.criarDescritor(imagemA, pontoschavesDetectadosA
 , descritorExtrator);
 extrairCaracteristicasImgB.criarDescritor(imagemB, pontoschavesDetectadosB,
 descritorExtrator);
```

Figura 35 - Chamado ao método que cria o vetor de correspondência e o método que mostra as imagens
Fonte: Autoria própria

O método criarVetorCorrespendencia calcula o vetor de correspondência entre os descritores das imagens e o armazena em um vetor. Esse vetor criado é utilizado no SLAM para atualizar as estimativas de pode da câmera e localização do marco no ambiente.

Alguns dos métodos observados na Figura 23 não foram mostrados acima, eles podem ser observados na Figura abaixo:

```
void criarVetorCorrespondencia(Mat descritor, Mat descritor2, FlannBasedMatcher flmatcher);
void desenhaCorrespondencia(Mat img, vector<KeyPoint>  pontoschavesA,
Mat img2, vector<KeyPoint>  pontoschavesB);

vector< DMatch > getvetPontosCorrespondentes();

void calculoDistancia();

void boasCorrespondencias();

void desenhaBoasCorrespondencias(Mat img, vector<KeyPoint>  pontoschavesA,
Mat img2, vector<KeyPoint>  pontoschavesB);

void marcaObjetoCena(Mat img, vector<KeyPoint>  pontoschavesA,Mat img2, vector<KeyPoint>  pontoschavesB);

void iniHomograph(vector<KeyPoint> pontoschaves_detectadosImg, vector<KeyPoint>
pontoschaves_detectadosImg2, vector<Point2f> imA, vector<Point2f> imB);
```

Figura 36 - Métodos da classe correspondência
Fonte: Autoria própria.

O primeiro tem a mesma função apresentada pelo método desenhaboasCorrespondencias, no entanto, ele apresenta o desenho das correspondências sem utilizar o filtro de distancia entre vetores, apresentando todos os pontos correspondidos obtidos pelo método criarVetorCorrespondencia (vide Figura 24).

O segundo é um método utilizado para mostrar à mínima e a máxima distancia entre os pontos característicos dos descritores.

Os dois últimos são utilizados para marcar um objeto na cena. As Figuras abaixo ilustram a implementação dos métodos:

```
void Correspondencia::desenhaCorrespondencia(Mat img, vector<KeyPoint>  pontoschavesA,
    Mat img2, vector<KeyPoint>  pontoschavesB)
{

        drawMatches(img, pontoschavesA, img2, pontoschavesB, vetorPontosCorrespondentes,
        imagem_correspondencia, Scalar::all(-1), Scalar::all(-1),
        vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
        imshow("Correspondencia", imagem_correspondencia);
}
```

Figura 37 - Implementação do método desenhaCorrespondencia
Fonte:Autoria própria.

```
void Correspondencia::calculoDistancia()
{

    /*
    *calcula a maxima e minima distancia entre os pontos chaves
    *
    */
    for (int i = 0; i < descritor_a_correspondencia.rows; i++)
    {
        double distancia = vetorPontosCorrespondentes[i].distance;
        if (distancia < min_dist) min_dist = distancia;
        if (distancia > max_dist) max_dist = distancia;
    }
    printf("Maxima distancia : %f \n", max_dist);
    printf("Minima distancia : %f \n", min_dist);
}
```

Figura 38 - Mínima e máxima distancia dos pontos característicos
Fonte: Autoria própria.

```
void Correspondencia::marcaObjetoCena(Mat img, vector<KeyPoint>  pontoschavesA, Mat img2, vector<KeyPoint>  pontoschavesB)
{
    boasCorrespondencias();
    drawMatches(img, pontoschavesA, img2, pontoschavesB, vetboasCorrespondencias,
        imagem_correspondencia, Scalar::all(-1), Scalar::all(-1),
        vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);

    Mat H = findHomography(vtPefimA, vtPefimB, CV_RANSAC);

    vector<Point2f> cantosObjetos(4);
    cantosObjetos[0] = cvPoint(0, 0); cantosObjetos[1] = cvPoint(img.cols, 0);
    cantosObjetos[2] = cvPoint(img.cols, img.rows); cantosObjetos[3] = cvPoint(0, img.rows);
    std::vector<Point2f> cantosCenas(4);
    perspectiveTransform(cantosObjetos, cantosCenas, H);
    line(imagem_correspondencia, cantosCenas[0] + Point2f(img.cols, 0), cantosCenas[1] + Point2f(img.cols, 0), Scalar(0, 255, 0), 4);
    line(imagem_correspondencia, cantosCenas[1] + Point2f(img.cols, 0), cantosCenas[2] + Point2f(img.cols, 0), Scalar(0, 255, 0), 4);
    line(imagem_correspondencia, cantosCenas[2] + Point2f(img.cols, 0), cantosCenas[3] + Point2f(img.cols, 0), Scalar(0, 255, 0), 4);
    line(imagem_correspondencia, cantosCenas[3] + Point2f(img.cols, 0), cantosCenas[0] + Point2f(img.cols, 0), Scalar(0, 255, 0), 4);
    imshow("Detecção do Objeto na cena", imagem_correspondencia);

}
void Correspondencia::iniHomograph(vector<KeyPoint> pontoschaves_detectadosImg, vector<KeyPoint>
pontoschaves_detectadosImg2, vector<Point2f> vetimgH,vector<Point2f> vetimgHp)
{
    boasCorrespondencias();
    for (int i = 0; i < vetboasCorrespondencias.size(); i++)
    {
        vetimgH.push_back(pontoschaves_detectadosImg[vetboasCorrespondencias[i].queryIdx].pt);
        vetimgHp.push_back(pontoschaves_detectadosImg2[vetboasCorrespondencias[i].trainIdx].pt);
    }
    vtPefimA = vetimgH;
    vtPefimB = vetimgHp;
}
```

Figura 39 - Implementação dos métodos para marcar um objeto em uma cena
Fonte: Autoria própria.

A Figura abaixo ilustra a chamada aos métodos apresentados da Figura 39.

```
vector<Point2f> vetimA; vector<Point2f> vetimB;
corresponderImagens.iniHomograph(pontoschavesDetectadosA, pontoschavesDetectadosB, vetimA, vetimB);
corresponderImagens.marcaObjetoCena(imagemA, pontoschavesDetectadosA, imagemB, pontoschavesDetectadosB);
```

Figura 40 - Chamada aos métodos iniHomograph e marcaObjetoCena
Fonte:Autoria própria.

3 RESULTADOS

Para detectar objetos em uma cena ou verificar se uma cena atual é ou não a mesma em comparação com alguma já adquirida, são necessários algoritmos capazes de extrair características invariantes no aspecto de rotação e escala em relação ao ponto de observação, para, posteriormente, fazer a comparação entre duas imagens. A técnica SURF apresenta essas características. A seguir, pode ser observada uma correspondência entre um objeto em uma cena:

Figura 41 - Correspondência entre imagens
Fonte: Autoria própria.

A imagem anterior demonstra a correspondência para o reconhecimento de um objeto em uma cena. Nenhuma técnica de eliminação de ruído foi empregada, apenas mostra os pontos correspondentes detectados pelo FlannBasedMatcher sem o filtro de menor distância, no presente caso, foi utilizado o método desenhaCorrespondencia da classe Correspondência. Assim, pode ser observado que, neste caso, não houve associação errada do objeto com a sua correspondência na cena, utilizando diretamente os resultados da correspondência dada pelo FlannBasedMatcher.

Figura 42 - Correspondência entre duas imagens (objeto e cena)
Fonte: Autoria própria.

Diferente da figura anterior, a Figura 42 ilustra a correspondência errada de um objeto na cena, no caso, seis pontos foram correspondidos erroneamente, no presente caso, o método desenhaCorrespondencia também foi utilizado para correlacionar as imagens. As falsas correspondências podem ser melhor observadas na figura abaixo:

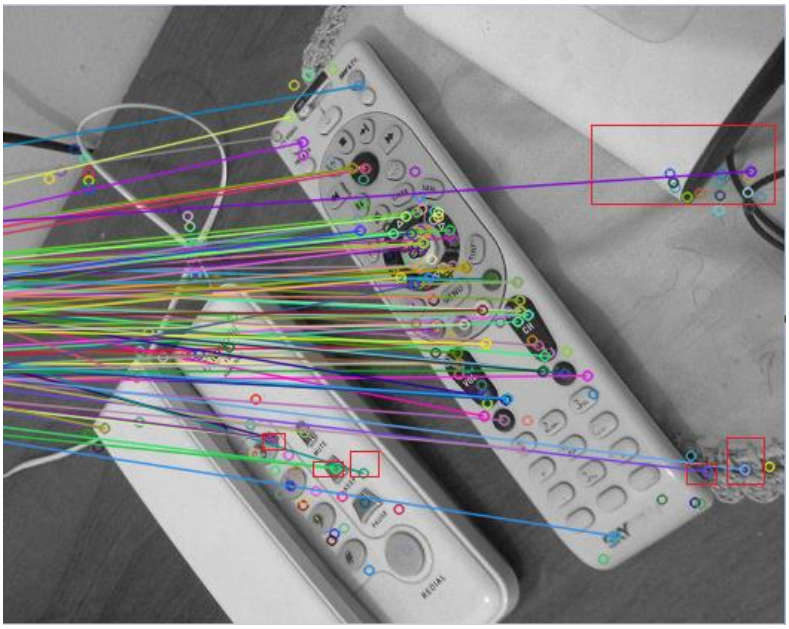
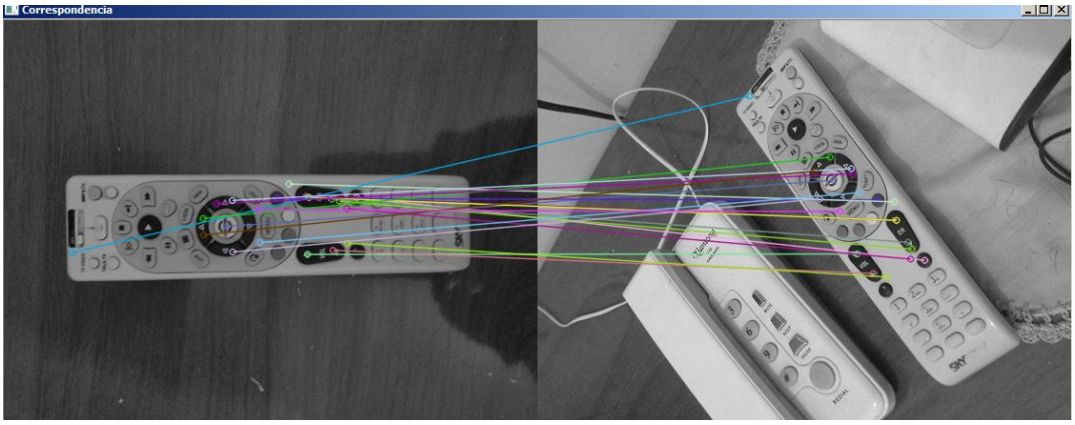


Figura 43 - Falsas correspondências
Fonte: Autoria própria.

A seguir, as mesmas imagens são correspondidas utilizando a técnica de filtro por boas correspondências dado a menor distancia por um limiar. O método utilizado é o desenhaboasCorrespondencia da classe Correspondencia.



A figura seguinte mostra a correspondência de pontos chaves entre duas cenas sem tratamento de falsas correspondências.

Figura 44 - Correspondência entre cenas sem tratamento de falsas correspondências
Fonte: Autoria própria.

Complementando, a Figura seguinte exibe a mesma correspondência anterior, mas desta vez com aplicação da técnica para eliminação de falsas correspondências com o método desenhaBoasCorrespondencias da classe Correspondência.

Figura 45 - Correspondência entre cenas com tratamento de falsas correspondências
Fonte: Autoria própria.

A seguir, o resultado de outras imagens correspondidas com o tratamento de falsas correspondências é dada pelas Figuras 46 e 47 abaixo:

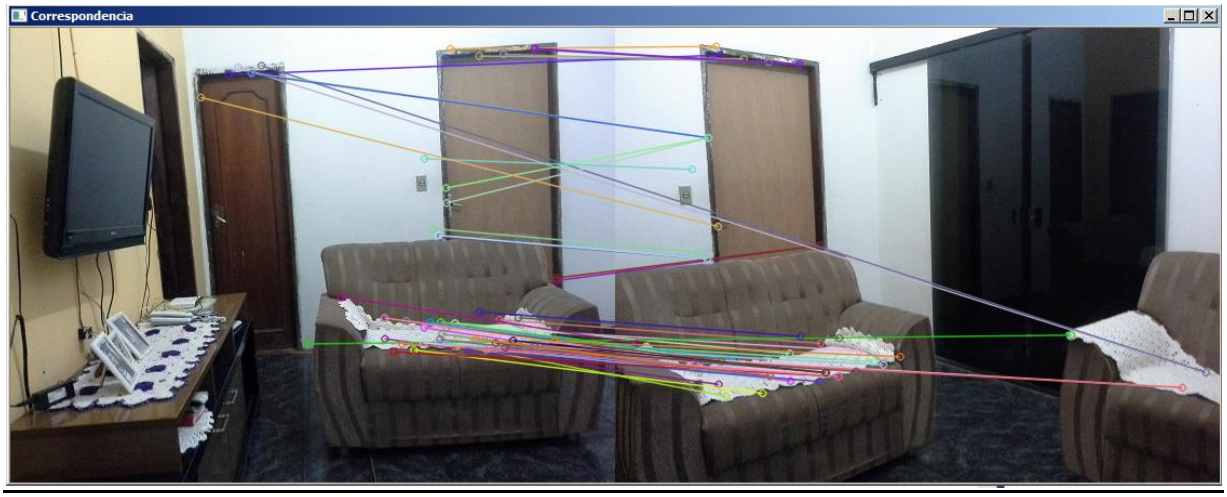


Figura 46 - Reconhecimento de cena
Fonte: Autoria própria.

Figura 47 - Reconhecimento de objeto em uma cena
Fonte: Autoria própria.

A figura abaixo ilustra o objeto que é correspondido na cena marcado, utilizando o método marcaObjetoCena da classe Correspondencia.

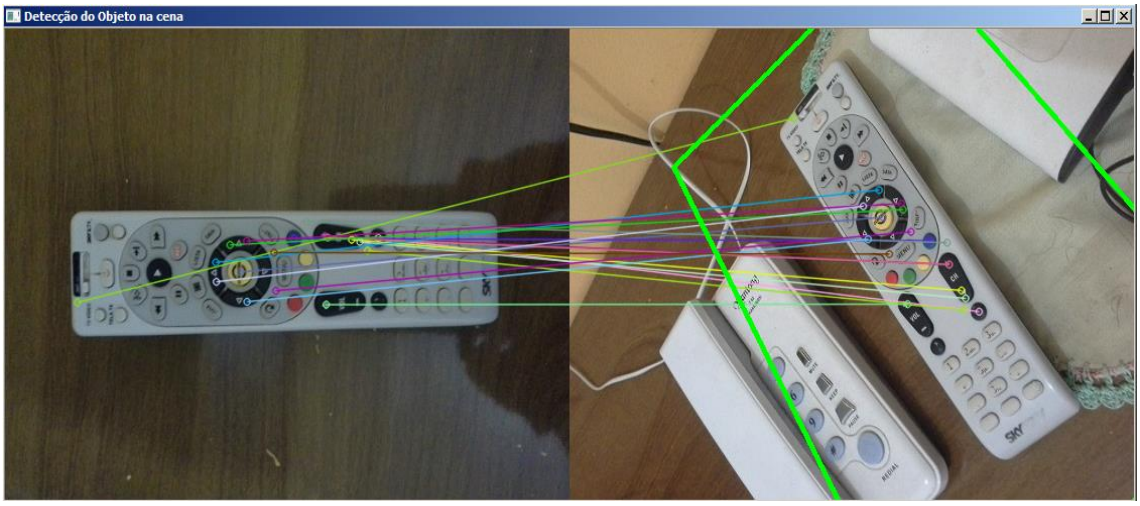
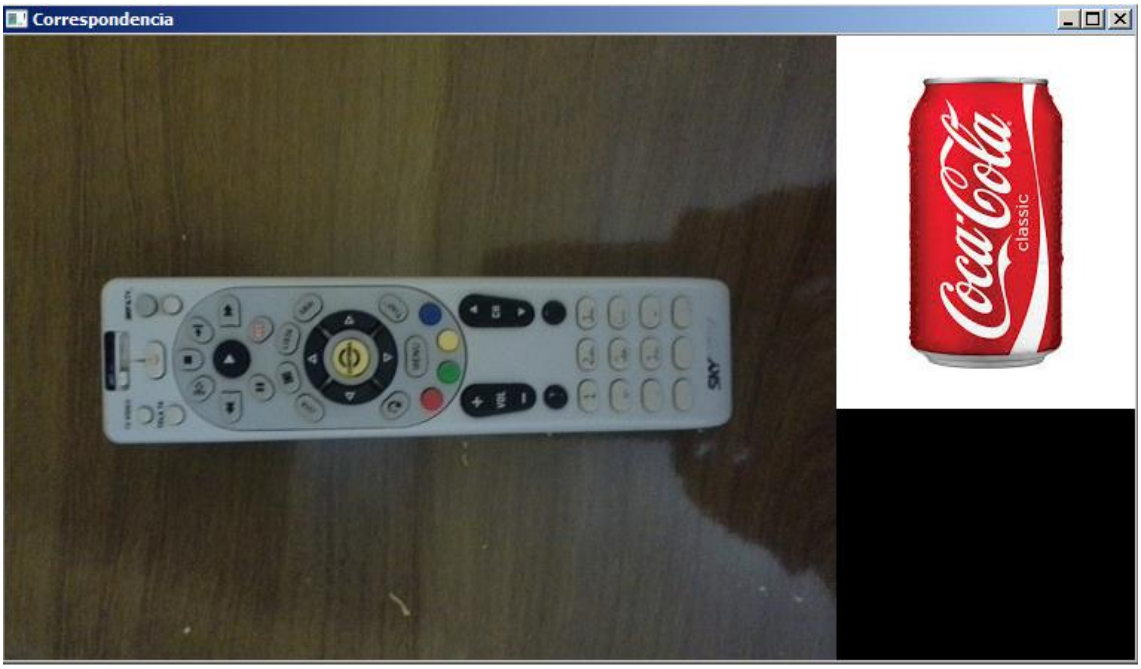


Figura 48 - Objeto identificado na cena destacado
Fonte: Autoria própria.



.

Figura 49 - Sem correspondências detectadas
Fonte: Autoria própria

4 CONSIDERAÇÕES FINAIS

O algoritmo para correspondência entre imagens apresentado e muito elementar. Deixa muito espaço para melhorias, por exemplo, o critério para correspondência leva em consideração apenas os descritores estabelecidos, em um cenário real, onde elementos podem apresentar formar e texturas semelhantes, e consequentemente, características semelhantes. Apenas o uso de descritores não serviria.

Uma técnica que poderia ser adotada neste caso é o Reconhecimento de Padrões. Reconhecer implica em conhecer de novo em um processo onde existe algum conhecimento prévio e algum tipo de armazenamento do conhecimento sobre o objeto a ser reconhecido. O termo padrão se refere a qualquer elemento que possa ser definido, mesmo que sujeito a variações.

Reconhecimento de objetos ou cenas é uma das principais funções da área de visão computacional e esta relacionada diretamente com o reconhecimento de padrões. Por exemplo: Um objeto pode ser definido pro diversos padrões, como: textura, forma, dimensão, etc. e o reconhecimento individual de cada um destes padrões pode facilitar o reconhecimento como um todo. Assim se um objeto definido como x, tiver correlação com outro objeto, e que atenda as condições de padrões estabelecidas pelo algoritmo, este será adotado no sistema com uma correspondência positiva, caso contrario, será tratado como objetos diferentes, evitando assim, correlação entre objetos similares em um cenário real. Entende-se por cenário real, o ambiente pelo qual o robô deve mapear e estimar sua devida localização.

Outro ponto a ser trabalhado é que, para cada imagem, um objeto é instanciado, o que, não seria adequado, uma vez que, sistemas de SLAM trabalham com muitas, até mesmo milhares de imagens, e instanciar um objeto para cada uma não seria viável em nível de velocidade de processamento. O ideal é trabalhar com vetores de imagens, vetor de pontos chaves e vetor de descritores na classe a ser implementada para cada uma das etapas.

Neste trabalho, foi apresentado um método para marcar um objeto em uma cena com a técnica RANSAC disponível pela OpenCv. Como o objetivo era mostrar a implementação de um vetor de correspondência,este método não foi apresentado com

detalhes, no entanto, é importante ressaltar a utilização dessa técnica no SLAM. Como pode ser observado na Figura abaixo, o RANSAC tem como retorno uma matriz de homografia, observe :

```
Mat H = findHomography(vtPefimA, vtPefimB, CV_RANSAC);
```

Figura 50 - Matriz homográfica
Fonte: Autoria própria.

Esta matriz de homografia e importante no SLAM, pois como descrito na seção 1.2.4, as correspondências são utilizadas para definição de um modelo de matriz definida como matriz essencial. A matriz de homografia retornada pelo método é definida na literatura como matriz fundamental, e a partir da matriz fundamental e possível definir a matriz essencial para determinar o movimento da câmera. Esta etapa já consiste em uma das varias etapas que são necessárias para estimativas de Localização e Mapeamento Simultâneos.

O método RANSAC requer que os pontos de entrada já estejam computados, no caso, o vetor de pontos chaves das imagens e o vetor de pontos correspondentes. Além da definição da matriz fundamental, o RANSAC elimina quaisquer valores aberrantes que ainda podem estar contidos no vetor de pontos correspondidos.

5 CONCLUSÃO

O presente trabalho teve como objetivo apresentar os processos básicos para extração de marcos visuais em imagens para serem utilizados como marcos para SLAM. Algumas etapas preliminares são necessárias, como Correções do sistema óptico, calibração da câmera, tratamento de imagens etc., porém, buscou-se apresentar de forma simples as principais fases necessárias para implementação de sistemas de SLAM baseado em visão – a literatura, em grande parte, tende a apresentar conteúdo exaustivo e de alta complexidade técnica dos processos necessários.

O processo apresentado na revisão da literatura para obtenção de marcos do ambiente pode ser ilustrado de uma forma geral na imagem abaixo:

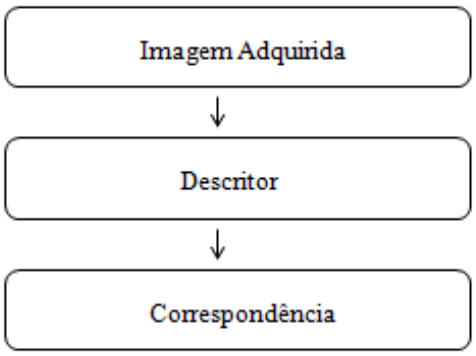


Figura 51 - Principais processos para obtenção de marcos do ambiente
Fonte: Autoria própria.

De acordo com a figura anterior, a imagem adquirida é processada pelo descritor para a determinação de suas características principais e construção de seus respectivos descritores. Após isso, é verificada a correspondência das imagens com imagens anteriores a fim de procurar pontos em comuns entre elas.

O método SURF utilizado se mostrou eficiente em seu propósito para as duas ultimas etapas do processo. O experimento mostrou que as características obtidas pelo SURF são invariantes a fenômenos de rotação, translação e escala, e que a escolha certa das configurações (distancia entre vetores, mínima hessiana) vai depender do aplicativo proposto, no experimento, a configuração do mínimo da matriz como 1500 se apresentou favorável, principalmente para reconhecimento de objetos em uma cena.

Espera-se que este texto sirva como base a trabalhos futuros do projeto VANT, bem como a outros pesquisadores que estejam iniciando trabalhos na área de SLAM Visual. Com base no estudo aqui apresentado já foi possível direcionar as atividades do Grupo de Pesquisa ÍCARO (Inteligência Computacional, Automação e Robótica) da Fundação Universidade do Tocantins, o qual está desenvolvendo o projeto VANT em parceria com o Corpo de Bombeiros Militar do Tocantins, com os protótipos de drones já construídos e totalmente funcionais.

Na próxima etapa do projeto, o grupo iniciará o desenvolvimento do sistema SLAM com base no levantamento aqui realizado, sendo então possível validar as técnicas propostas.

4. REFERENCIAS BIBLIOGRAFICAS

ABREU, L. B. **Estudo da Utilização de Filtros de Kalman para Auto Localização.** 2008. 96 f. Dissertação (Mestrado em Engenharia de Electrotécnica e de Computadores) – Universidade do Porto, Porto, 2008.

A. M. Santana and A. A. D. Medeiros, 2011, "**A Line-Based Approach to Slam Using Monocular Vision**".IEEE T Latin America Transactions, Vol.9, No.3, pp. 231-239.

ARAÚJO, V. M. U. **Técnicas Visuais de Localização e Mapeamento Simultâneos sem Extração de Primitivas Geométricas da Imagem**. 2011. 54 f. Dissertação (Mestrado em Engenharia de Computação) - Universidade Federal do Rio Grande do Norte, Rio Grande do Norte, 2011.

BELOI, F. A. W. **Desenvolvimento de Algoritmos de Exploração e Mapeamento Visual para Robôs Móveis de Baixo Custo.** 2006. 100 f. Dissertação (Mestrado em Engenharia Elétrica) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2006.

BIGHETI, J. A. **Navegação de Robôs em Ambientes Internos Usando Slam**. 2011. 110 f. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Estadual Paulista, São Paulo, 2011.

BOAS, E. R. V. **Mapeamento e Localização Simultânea de Ambientes Dinâmicos Aplicados na Navegação de Veículo Autônomo Inteligente**. 2011. 80 f. Dissertação (Mestrado em Ciências Engenharia Elétrica) - Universidade Federal de Itajubá, Itajubá, 2011.

BUENO, L. M. **Análise de descritores locais de imagens no contexto de detecção de semi-réplicas**. 2011. 72 f. Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Campinas, Campinas, 2011.

BUSCARIOLLO, P. H. **Sistema de Posicionamento dinâmico baseado em visão computacional e laser**. 2008. 145 f. Tese (Doutorado em Engenharia) - Escola Politécnica da Universidade de São Paulo, 2008.

BRITTO, KOERICH. **Extração de Características**. Paraná: Pontifícia Universidade Católica do Paraná (PUCPR), 2009. 49: color; Acompanha texto. Disponível em: http://www.ppgia.pucpr.br/~alekoe/AM/2009/2-ExtracaoCaracteristicas-ApreMaq-2009a.pdf. Acessado em 12 de fevereiro de 2015.

CHEN, ZHENHE, JAGATH SAMARABANDU e RANGA RODRIGO (2007), '**Recent advances in simultaneous localization and map-building using computer vision**', Advanced Robotics 21.

CRUZ, S. M. **Implementação de um filtro de Kalman estendido em arquitetura reconfiguráveis aplicado ao problema de localização em robótica móvel.** 2013. 81 f. Dissertação (Mestrado em Sistemas Mecatrônicas) – Universidade de Brasília, Brasília, 2013.

Gil, Arturo, Oscar Martinez Mozoa, Monica Ballesta & Oscar Reinoso (2009), '**A comparative evaluation of interest point detectors and local desccriptors for visual SLAM**',*Machine Vision and Applications*.

GONZÁLES, G. L. G. **Aplicação da Técnica SIFT para determinação de Campos de Deformações de Materiais usando Visão Computacional.** 2010. 104 f. Dissertação (Mestrado em Engenharia Mecânica) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2010.

GUIZILINI, V. C. **Localização e Mapeamento Simultâneos com auxílio visual omnidirecional**. 2008. 100 f. Dissertação (Mestrado em Engenharia) - Universidade de São Paulo, São Paulo, 2008.

H. Bay, T. Tuytelaars, and L. V. Gool, "**SURF: Speeded up robust features**," in Proc. of the 9th European Conference on Computer Vision (ECCV'06), ser. Lecture Notes in Computer Science, A. Leonardis, H. Bischof, and A. Pinz, Eds, vol. 3951. Graz, Austria: SpringerVerlag, May 2006, pp. 404–417.

Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. **Speeded-up robust features (surf)**. Computer Vision and Image Understanding, 110(3): 346 - 359, 2008, Similarity Matching in Computer Vision and Multimedia.

JUNIOR, P. L. J. D. **Robótica Subaquática: Uma Abordagem Baseada em Visão Computacional Aplicada a SLAM.** 2008. 119 f. Monografia (Engenharia da Computação) – Fundação Universidade Federal do Rio Grande, Rio Grande, 2008.

Leonard J. and Durrant-Whyte, H.F., 1991, "**Mobile Robot Localization by Tracking Geometric Beacons**".IEEE Transaction on Robotics and Automation, Vol.7, No.3, pp. 376-382.

Lowe, David G. (2004). **Distinctive image features from scale-invariant key points**. *International Journal of Computer Vision* 60(2): 91-110.

RAMOS, D. C. **Aplicação de técnicas de fusão sensorial para mapeamento e localização simultâneos para robôs terrestres.** 2012. 96 f. Dissertação (Mestrado em Engenharia de Automação e Sistemas) – Universidade Federal de Santa Catarina, Florianópolis, 2012.

SANTANA, A. M. **Localização e Mapeamento Simultâneos de Ambientes Planos Usando Visão Monocular e Representação Híbrida do Ambiente**. 2011. 121 f. Tese (Doutorado em Engenharia Elétrica e de Computação) – Universidade Federal do Rio Grande do Norte, Natal, 2011.

SANTOS, L. G. **Localização de Robôs Móveis Autônomos Utilizando Fusão Sensorial de Odometria e Visão Monocular.** 2010. 51 f. Dissertação (Mestrado em Ciências) - Universidade Federal do Rio Grande do Norte, Rio Grande do Norte, 2010.

SILVA, B. M. F. **Odometria Visual Baseada em Técnicas de *Structure from Motion*.** 2011. 79 f. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal do Rio Grande do Norte, Natal, 2011.

THRUN, S.; BURGARD, W.; FOX, D. **Probabilistic Robotics**. Cambridge, MA, MIT Press, 2005.

Tuytelaars,T.;Mikolajczk,K.**A survey on Local Invariant Features**. *K.U. Leuven & University of Surrey*.Maio 2006.

VOLTOLINI, M. C.; KIM, H. Y. **Simples: Um descritor de características locais rápido e simples**. 2013.

Zitová, B. and J. Flusser, 2003**, Image registration methods: a survey**, *Image and Vision Computing*, 21(11), 977-1000.