Protótipo de um Algoritmo para Robô Seguidor de Piso Tátil Utilizando Visão Computacional

Renato Mello Konflanz[1], **Marcos Antonio Moretto**[1]

[1]Universidade Comunitária da Região de Chapecó (Unochapecó)
Chapecó – SC – Brasil

renato.konflanz@unochapeco.edu.br, marquinhos@unochapeco.edu.br

Abstract. *This work aims to propose an algorithm that allows a robot to autonomously locate and navigate within a tactile floor environment using computational vision, in order to offer a form of autonomous navigation for robots without the need to modify the environment in that it will act. During the development of the work, the algorithm is proposed, developed and tested within the Unochapecó campus. In the end, it is concluded that the proposed algorithm has potential to be used in transport and assistance systems for blind people in places that already have tactile floors, presenting a low cost alternative for autonomous robot localization and navigation.*

Resumo. *Este trabalho visa propor um algoritmo que permita um robô se localizar e navegar de forma autônoma dentro de um ambiente com pisos táteis se utilizando de visão computacional, de forma a oferecer uma forma de navegação autônoma para robôs sem que seja necessário a modificação do ambiente em que ele atuará. Durante o desenvolvimento do trabalho, o algoritmo é proposto, desenvolvido e testado dentro do câmpus da Unochapecó. Ao fim, é concluído que o algoritmo proposto tem potencial de ser utilizado em sistemas de transporte e de assistência para cegos em locais que já possuem pisos táteis, apresentando uma alternativa com baixo custo para localização e navegação autônoma de robôs.*

1. Introdução

A área da robótica cresceu exponencialmente nas últimas décadas. A grande inovação que o campo trás para a sociedade gera debates interdisciplinares que variam desde em quais áreas essas máquinas podem ser utilizadas para trazer valor para a humanidade até quais são seus impactos negativos e positivos na economia e para as pessoas. Atualmente, robôs são utilizados em vários campos e desempenham diferentes tarefas. As mais comuns estão na indústria, onde as máquinas auxiliam na manufatura principalmente de carros e eletrônicos. Ainda, podem ser usados na agricultura, mineração, transporte, saúde, educação, exploração do espaço e do mar, segurança, utilizações domésticas, entre outros. Em decorrência do grande impacto que a área tem na economia mundial, em 2014 o mercado da robótica valia aproximadamente 29 bilhões de dólares [Keisner et al. 2016].

Visão computacional e processamento de imagens são tecnologias que também têm potencial de trazer valor de muitas formas. [Gonzalez and Woods 2009] dizem que "hoje em dia, não existe praticamente mais nenhuma área de empreendimento técnico que não seja impactada de uma forma ou de outra pelo processamento digital de imagens".

[Das et al. 2017] e [Gonzalez and Woods 2009] citam em suas obras algumas áreas que utilizam essas tecnologias: medicina, ciências biológicas, astronomia, geografia, arqueologia, reconhecimento de caracteres, reconhecimento de gestos, busca de imagens, reconhecimento facial etc.

O problema de localização de um robô é antigo. Por décadas cientistas tentaram criar e aperfeiçoar métodos para que robôs autônomos pudessem se localizar em ambientes abertos e fechados, com o objetivo de aumentar suas capacidades de mobilidade para realizar uma série de atividades que antes não podiam. [Wang 1988, traduzido pelo autor] escreve que "determinar a localização de um robô é um importante problema em navegação de veículos autônomos em um ambiente desestruturado". [Guibas et al. 1997, traduzido pelo autor] exemplificam o problema: "Um robô está em uma posição desconhecida em um ambiente o qual ele tem um mapa. Ele olha seus arredores, e baseado em suas observações ele infere o seu local (ou um grupo de locais) no mapa onde ele poderia estar localizado".

Um método comum para que as máquinas saibam onde estão indo é o de seguir linhas, em que o robô detecta uma linha no chão e a segue, esperando chegar no local desejado. Um dos inconvenientes desta técnica é ter que marcar o chão com as linhas que apontam para onde o robô precisa se mover. [Siegwart and Nourbakhsh 2004] deixam claro que a principal desvantagem de técnicas de navegação baseadas em marcações físicas é a significante mudança do ambiente que deve ser realizada. Este problema pode ser resolvido caso o robô possa se utilizar de marcações que já existem no local em que precisa se localizar, como um piso tátil, que ajuda a guiar os cegos em ambientes abertos e fechados.

Possibilitar que robôs seguidores de linha possam se localizar dentro de um ambiente sem o trabalho de ter que ativamente colocar as linhas no chão do local agrega valor à tecnologia, já que ela pode operar sem grandes esforços além da obtenção das máquinas e o mapeamento do local em que se localizará. Essa tecnologia pode ser empregada para usos de transporte e auxílio para pessoas com deficiência visual em locais com calçadas acessíveis, como universidades, edifícios e cidades. Com pequenas modificações, pode ser utilizado também em robôs industriais em funções como a de reposição de estoque em prateleiras e transporte de materiais dentro de fábricas.

O objetivo principal deste trabalho é propor um algoritmo que permita um robô se localizar e navegar de forma autônoma dentro de um ambiente com pisos táteis se utilizando da linguagem de programação Python com a biblioteca OpenCV.

Para atingir os objetivos propostos nesse trabalho foi desenvolvido um algoritmo utilizando a ferramenta OpenCV e técnicas exploradas de visão computacional. Com o algoritmo desenvolvido, ele foi testado dentro do campus da Unochapecó. Seu objetivo era se movimentar dentro da universidade de um ponto a outro (definidos por uma pessoa) autonomamente usando o piso tátil como guia. O desempenho do seguidor de linha foi avaliado e descrito no trabalho com relação a sua frequência de erro em se manter na linha, frequência de erro em decisões nas intersecções e velocidade máxima de movimentação.

Este trabalho está organizado em 4 capítulos. O primeiro introduz o assunto tratado no trabalho e explica o porquê de ser válido para a academia. O segundo capítulo apresenta a fundamentação teórica do trabalho, descrevendo a pesquisa bibliográfica re-

alizada e conceitos necessários para o entendimento completo do trabalho. A terceira parte foca na metodologia do trabalho, apresentando o que foi feito para atingir o objetivo proposto e explicando as decisões tomadas. No capítulo quatro está apresentado os resultados do algoritmo desenvolvido e no quinto as conclusões da pesquisa.

2. Fundamentação teórica

2.1. Conceitos e técnicas de visão computacional e processamento de imagens

Visão computacional é um campo amplo e diverso. Ao longo do tempo foram desenvolvidos uma série de conceitos sobre imagens digitais e técnicas úteis que visam resolver diferentes problemas. Essa seção apresentará alguns desses conceitos, métodos e sua utilidades.

2.1.1. Representação de uma imagem digital

Uma imagem digital pode ser encarada como uma matriz com duas dimensões com valores de intensidade em cada posição, representando uma cor entre branco e preto. Cada um desses elementos dentro da matriz são chamados de *pixels* [Gonzalez and Woods 2009]. Os valores de intensidade variam com a quantidade de *bits* que são usados para guardar o valor de cada *pixel*, de forma que seguem a regra $0 <= intensidade < \textit{(quantidade de bits)}^2$. Na maioria das vezes, são utilizados 8 *bits* para representar o nível de cinza de uma imagem, fazendo com que seu valor máximo seja 255, que representa o branco. A Figura 1 mostra em (a) uma imagem digital com quatro *pixels* distribuídos em duas colunas e duas linhas, enquanto em (b) é mostrado a matriz numérica que representa essa imagem com níveis de intensidade de 0 até 255.

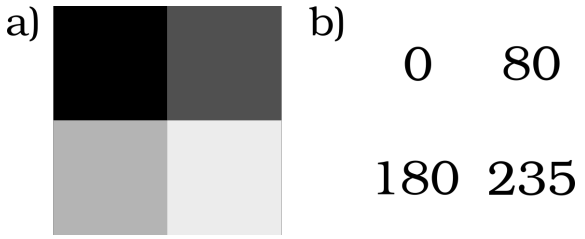

Figura 1. Representação de uma imagem digital em preto e branco

2.1.2. Representação de uma imagem digital colorida

Na subseção anterior foi apresentado como uma imagem em preto e branco é representada digitalmente. Imagens coloridas precisam de mais dados para serem interpretadas por um computador, já que não podem se basear em apenas um valor de intensidade que varia de um extremo ao outro, como imagens em preto e branco que variam do branco até o preto. Essa subseção apresenta algumas formas de representação digital de uma imagem com cores.

2.1.3. Modelo RGB

Um método comum para representação de imagens coloridas é o RGB (sigla do inglês *red, green* e *blue*) que se utiliza de três valores para cada *pixel* da imagem. Cada valor representa, respectivamente, seu nível de vermelho, verde e azul. Dessa forma, uma imagem digital em RGB pode ser representada como uma matriz bidimensional em que em cada posição (x, y) guarda um vetor de uma dimensão com três valores no formato $[R, G, B]$ (cada posição nesse vetor pode também ser chamada de canal) [Gonzalez and Woods 2009]. Dessa forma, uma imagem no padrão RGB contém três vezes mais informações que imagens em preto e branco. Normalmente usa-se 8 *bits* para representar cada canal, fazendo com que cada *pixel* utilize 24 *bits* de memória. Ao juntar os três canais, pode-se formar outras cores no espectro visível. A escolha dessas três cores para a representação de outras não é inesperada, já que se tratam das três cores luz primárias.

O sistema RGB pode ser representado pelo cubo na Figura 2, que mostra um plano tridimensional em que cada dimensão representa uma cor do esquema. A figura mostra como as cores secundárias se formam nos vértices em que duas cores primárias se encontram. Também é perceptível que o branco é a junção total das três cores, enquanto o preto ocorre quando nenhuma delas é adicionada. Dentro do cubo se encontram as misturas de todas as outras cores.

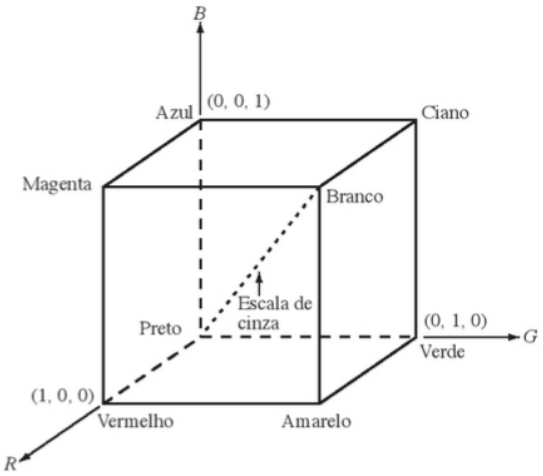

Figura 2. Cubo RGB

[Saravanan et al. 2016] explicam que esse sistema de representação de cores é amplamente usado em quase todos os sistemas computacionais, em televisões e filmes. Apesar disso, declaram também que o sistema não é uma boa escolha para se utilizar em processamento de imagens, pois não há separação entre crominância (que se refere às cores) e luminância (que se refere à intensidade da luz na cor), que é útil nessas aplicações.

2.1.4. Modelo HSI e derivados

O modelo RGB funciona bem para implementação em *hardware* e se adapta ao fato de que os olhos humanos são bastante perceptivos às três cores luz primárias. Porém, não funciona bem quando é preciso descrever uma cor, já que humanos não conseguem se referir a cores descrevendo qual a sua porcentagem de vermelho, verde e azul [Gonzalez and Woods 2009]. Quando humanos descrevem uma cor, se referem a três elementos: matiz, saturação e brilho. A matiz é um atributo associado com o comprimento de onda do espectro eletromagnético visível, representando a cor dominante para seu observador. Dessa forma, o que diferencia cores como azul, vermelho e o verde são suas diferentes matizes. Saturação é o atributo que representa o quão pura a cor é em relação à quantidade de luz branca que se mistura com a cor original da matiz, ou seja, quanto mais próxima de branco uma cor é, menor sua saturação [Cheng et al. 2001]. Brilho é uma definição subjetiva, que incorpora a noção de intensidade de uma cor (como em imagens em preto e branco) que representa quanta luz chega ao olho do espectador. O conjunto da matiz e saturação de uma cor é chamado de cromicidade. O modelo de cores HSI (sigla do inglês *hue*, *saturation* e *intensity*) utiliza estes conceitos para representar cores. [Saravanan et al. 2016] explicam que a separação dos elementos de cromicidade e o de brilho no modelo HSI faz com que seja mais propício para o campo de processamento de imagens, principalmente na área de segmentação de imagens baseada em cores.

O modelo de cores HSI pode ser representado por um cilindro, como na Figura 3, em que a matiz é representada por um ângulo de 0° a 360°, a saturação por a distância medida a partir do centro (de forma que cores mais próximas do centro são mais claras e portanto, menos saturadas) e a intensidade é um valor no eixo da altura.

Similarmente ao modelo RGB, uma imagem no esquema de cores HSI pode ser representada computacionalmente por uma matriz bidimensional em que em cada posição (x, y) guarda um vetor de uma dimensão com três valores no formato $[H, S, I]$.

De forma parecida ao modelo HSI, outros modelos foram criados, como o HSB, HSL e HSV. Apesar deles funcionarem de forma levemente diferente, eles mantêm a premissa de separar os elementos de matiz e saturação do elemento que representa brilho, sendo assim, todos podem ser úteis dentro da área de processamento de imagens e visão computacional.

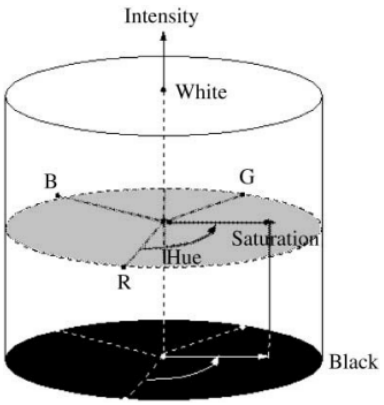

Figura 3. Representação gráfica do modelo HSI

2.1.5. Segmentação de imagem por cores

Em uma imagem colorida pode ser útil utilizar suas cores para segmentá-la em duas ou mais partes. Para tal, pode-se usar a técnica de segmentação de imagens por cores, como mostrado por [Shadeed et al. 2003]. Nessa técnica, criam-se limites para cada dimensão do modelo de cor, de forma que se o modelo HSV for usado, há um limite máximo e um limite mínimo para H, S e V que serão utilizado para criar uma máscara que segmentará a imagem. Um algoritmo itera sobre os *pixels* da imagem e cria uma máscara seguindo a Equação 1, em que $g(x, y)$ é a máscara criada e $H_{(x,y)}$, $S_{(x,y)}$ e $V_{(x,y)}$ representam, respectivamente, o valor de H, S e V do *pixel* localizado em (x, y) na imagem original.

$$g(x,y) = \begin{cases} 1 & \text{se } H_{min} <= H_{(x,y)} <= H_{max} \text{ e} \\ & S_{min} <= S_{(x,y)} <= S_{max} \text{ e} \\ & V_{min} <= V_{(x,y)} <= V_{max} \\ 0 & \text{caso contrário} \end{cases} \quad (1)$$

Equação 1. Função de segmentação com cores por limites

Na Figura 4 é mostrado em (a) uma imagem colorida representada em HSV de um chão cinza com um piso tátil em vermelho, no qual os três canais variam de 0 até 255. Em (b) é mostrado o resultado de uma máscara obtida utilizando $H_{min} = 0$, $H_{max} = 255$, $S_{min} = 51$, $S_{max} = 255$, $V_{min} = 0$ e $V_{max} = 255$. É importante observar que nesse caso apenas no valor de saturação (S) existe um limite, já que nos outros canais o mínimo e o máximo correspondem ao valor mínimo e máximo que eles podem ter. A máscara funciona para a segmentação do piso tátil porque o valor de saturação do vermelho na imagem original é mais elevado do que o valor de saturação do chão. A Figura 4(c) mostra a multiplicação da imagem original com a máscara obtida.

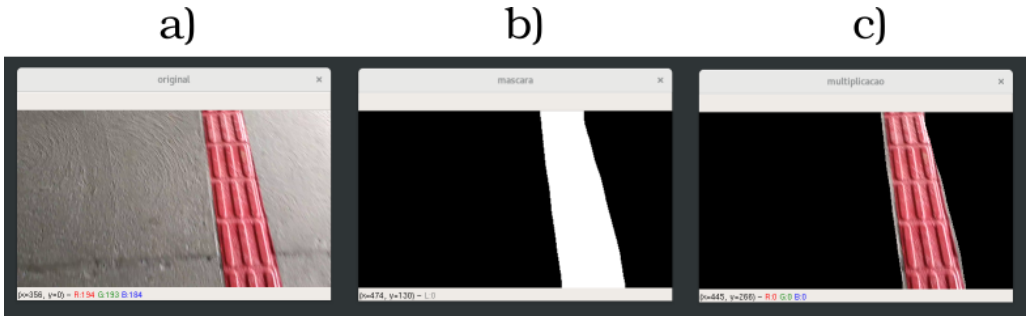

Figura 4. Resultado do processo de segmentação por cores

2.1.6. Erosão e dilatação de imagens

Erosão e dilatação de imagens são operações morfológicas (que se baseiam em teoria de conjuntos) que são aplicadas a imagens binárias com objetivo de reduzir ruído, isolar ou juntar diferentes elementos em uma imagem e encontrar pontos de intensidades desnivelados [OpenCV 2014]. Em uma imagem binária, a operação de dilatação aumenta os elementos em branco da imagem, enquanto a erosão os diminui.

Operações morfológicas são filtros espaciais não lineares. As operações consistem em convoluções de um *kernel* B sobre uma imagem A, em que B geralmente tem formato retangular ou circular e contém um ponto de âncora que normalmente é posicionado no meio do *kernel*, seu ponto $(0, 0)$. Na dilatação, caso qualquer *pixel* branco do *kernel* se sobreponha ao conjunto de *pixels* brancos da imagem, a imagem resultada da técnica terá um *pixel* branco no ponto de âncora de B no momento da intersecção. Na erosão, o *pixel* branco aparecerá na imagem final apenas quando todo o conjunto de *pixels* brancos do *kernel* se sobrepuser ao conjunto de *pixels* brancos da imagem.

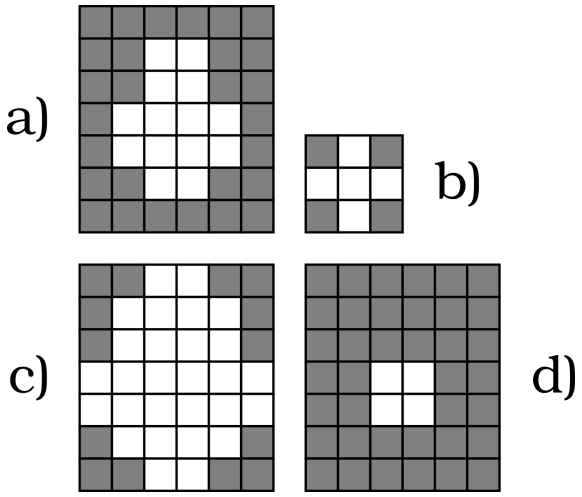

Figura 5. Exemplo de erosão e dilatação

A Figura 5 mostra em (a) uma imagem binária e em (b) um filtro B com âncora em seu centro. Em (c), pode-se ver o resultado da dilatação de (a) com o filtro (b) e em (d) a erosão de (a) com o *kernel* (b). É possível notar como na dilatação a imagem original é expandida e na erosão o oposto ocorre.

Essas duas operações originam outras duas técnicas importantes no processamento de imagens, a *opening* e a *closing*. A primeira é constituída de uma operação de erosão seguida de dilatação. Seu objetivo é remover ruídos da imagem, que são apagados completamente durante a erosão. A dilatação então é aplicada para que expanda-se novamente a imagem, mas sem o ruído inicial. A Figura 6(a) mostra à esquerda uma imagem e à direita ela após uma operação de *opening*. De forma parecida, *closing* é uma operação de dilatação seguida de erosão, objetivando reduzir ruídos internos na parte branca da imagem. A Figura 6(b) mostra uma operação de *closing*.

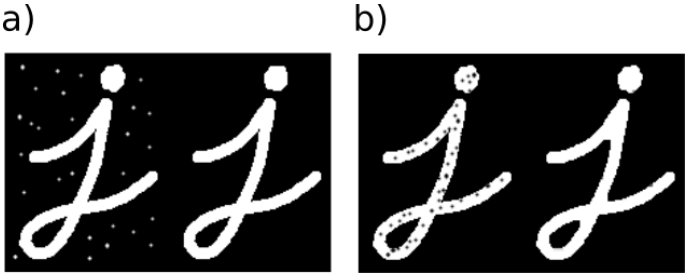

Figura 6. Exemplo de *opening* e *closing*

2.2. Robôs seguidores de linhas

Robôs seguidores de linhas são robôs que se movem autonomamente que seguem uma linha que está desenhada no chão [Pakdaman et al. 2010]. Suas operações básicas são: capturar a posição da linha no chão com alguma tecnologia e controlar sua movimentação e velocidade de acordo com essa informação, de forma que o robô se mova apenas por cima da linha. Eles podem também ter outros tipos de sensores que os ajudam a realizar tarefas mais complexas, como sensores de proximidade utilizados por [Punetha et al. 2013] em sua pesquisa para evitar que o robô colida com objetos enquanto segue seu caminho.

[Pakdaman et al. 2010] explicam que esse tipo de robô pode ser utilizado para propósitos militares, serviços de entrega, sistemas de transporte e aplicações de assistência para cegos. Outra utilidade é apresentada por [Punetha et al. 2013], que mostram em seu trabalho o uso de um robô seguidor de linha que leva remédios para pacientes em um hospital e concluem que tal aplicação da tecnologia diminuiria significativamente o custo com enfermeiros dessas instituições.

2.2.1. Seguidores de linha com infravermelho

O tipo mais comum de seguidor de linha é o que usa sensores de raios infravermelhos para detectar a linha que está desenhada no chão. Esses dados são então enviados para algum dispositivo que possa interpretá-los, como um microcontrolador ou um computador. Durante a fase de interpretação, o algoritmo deve usar os dados dos sensores para estimar se está em cima da linha ou se deve ajustar sua posição para que se mova usando o traço branco ou preto no chão como guia.

Seguidores de linhas baseados em sensores infravermelhos levam a vantagem de serem relativamente simples e rápidos de serem implementados. São ideais para ambientes que foram criados ou altamente adaptados para esse propósito, como indústrias que os utilizam para transporte interno. Entretanto, têm desvantagens também, já que não podem atuar em ambientes que não foram projetados com linhas e chão das cores necessárias, que obrigatoriamente devem usar as cores preto e branco. Em geral, para utilizar esse tipo de robô deve-se modificar significativamente o ambiente de trabalho, que pode gerar muito esforço e torná-lo visualmente não atraente [Siegwart and Nourbakhsh 2004].

Ainda, o tamanho das linhas não podem variar durante o percurso, já que os sensores do robô são posicionados para conseguirem detectar de forma correta apenas uma espessura de traço. A distância entre os sensores e o chão também deve ser a menor possível, preferencialmente menor que 1 centímetro. O robô também deve se preocupar em proteger a parte do chão que será analisada, já que a luz do ambiente pode atrapalhar a leitura dos sensores [Pakdaman et al. 2010].

2.2.2. Seguidores de linha com visão computacional

Robôs seguidores de linhas que utilizam visão computacional para navegarem seguem o mesmo modelo dos que usam de sensores infravermelhos, porém, a linha é detectada a partir de imagens captadas por uma câmera localizada no robô. A câmera é apontada para o chão e captura imagens periodicamente em espaços curtos de tempo, que são enviadas

para um microcontrolador ou um computador para processamento. Nesta fase é utilizado uma série de técnicas de processamento de imagens e visão computacional para detectar a linha no chão e decidir para qual lado o robô deve movimentar-se, como mostram [Elhady et al. 2014]. Geralmente essa série de técnicas (também chamado de *pipeline*) envolve a definição de uma região de interesse, conversão do modelo de cores da imagem, aplicação de algum filtro para facilitar a segmentação da imagem, utilização de uma técnica para segmentação da imagem que separa a linha do chão e uso de processos morfológicos para reduzir imperfeições na segmentação.

[Elhady et al. 2014] analisam as desvantagens dos seguidores de linha com sensores infravermelhos e declaram que elas podem ser corrigidas através do uso de técnicas de visão computacional como detector de linha. Com o apoio de uma câmera, o robô pode atuar em ambientes que não foram necessariamente criados para um seguidor de linha operar, já que ele pode ser adaptado para seguir diferentes tipos de linhas de quaisquer cores, como as linhas do piso tátil, que esse trabalho busca realizar. Dessa forma, não é necessário fazer grandes mudanças no ambiente em que o robô irá deslocar-se. As linhas podem também variar de tamanho e cor durante o percurso desde que o algoritmo que tratará a imagem seja desenvolvido para detectar todas as variações. Ainda, não há preocupações em proteger a linha da luz ambiente, já que ela é beneficial para a obtenção de imagens mais claras.

As desvantagens desse tipo de tecnologia estão em sua maior complexidade e custo em relação às alternativas. Desenvolver um seguidor de linha desse tipo exige conhecimentos em processamento de imagens e visão computacional, que envolve uma curva de aprendizado maior que o estudo de seguidores com sensores infravermelhos. O preço final é elevado porque o custo de uma câmera é maior que o custo de sensores infravermelhos e o processador deve ser mais potente do o que pode ser usado em sua alternativa. Ainda, é desafiador lidar com possíveis mudanças na iluminação do ambiente, como a transição entre dias e noites em locais abertos, já que com pouca luz torna-se difícil a diferenciação da linha do chão. Esse problema pode ser solucionado com a utilização de uma fonte própria de luz, se essa for possível.

3. Metodologia

3.1. *Pipeline* de visão computacional

A *pipeline* utilizada no algoritmo desenvolvido nesse trabalho é:

1. Leitura da imagem da câmera em RGB;
2. Diminuição da resolução da imagem para a metade da original (resultando e uma imagem com 640 *pixels* de largura por 480 *pixels* de altura);
3. Conversão da imagem para o modelo de cores HSL;
4. Aplicação de um filtro de média para a suavização da imagem;
5. Aplicação de uma segmentação por cores na imagem;
6. Aplicação de uma erosão seguida de dilatação na imagem.

O algoritmo pode também em teoria ser utilizado em resoluções de câmera diferentes, mas imagens menores ou maiores não foram testadas neste trabalho. Se usado com resoluções diferentes, é recomendado redimensionar a imagem da câmera para que fique com resolução próxima da utilizado neste projeto.

Em decorrência da existência de pisos táteis de duas cores no ambiente que o protótipo foi testado – vermelho e azul – é necessário realizar duas segmentações diferentes, gerando duas máscaras. De forma dinâmica é detectado se o robô está em um ambiente com pisos vermelhos ou azuis, aplicando as duas segmentações e verificando em uma área de interesse perto do centro qual das duas máscaras têm mais *pixels* brancos. A Figura 7 mostra os dois tipos de pisos do ambiente e abaixo a aplicação das segmentações necessárias.

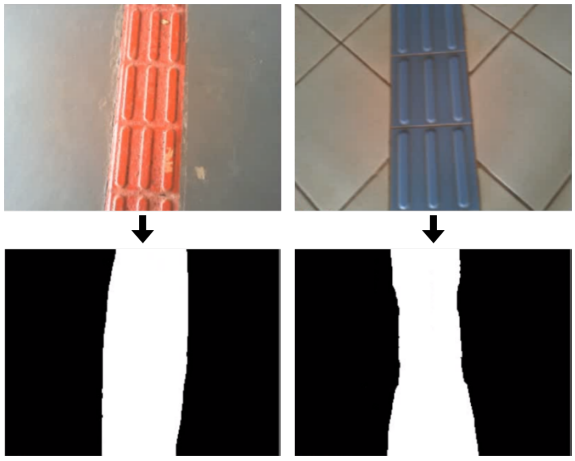

Figura 7. Ambientes diferentes que o robô passará

A Figura 8 mostra mais exemplos das máscaras binárias geradas quando imagens captadas pela câmera do robô são submetidas pela *pipeline* de visão computacional desenvolvida.

Figura 8. Exemplos de máscaras geradas pela *pipeline*

3.2. Mapeamento do ambiente

Para que um robô consiga se localizar no ambiente ele deve ter um mapa para poder inferir qual sua posição atual e como chegar até outros lugares. O mapa do ambiente em que o robô foi testado – uma parte do corredor do campus da Unochapecó – foi estruturado como um grafo, que está representado visualmente na Figura 9. Em nível de estrutura de código em Python, o grafo está organizado em um dicionário com três chaves que guarda em sua chave final uma direção, que pode ser "direita", "esquerda", "frente" ou "tras". As três chaves representam, respectivamente, em qual vértice o robô está, de qual vértice o robô veio e para qual vértice o robô está indo. Dessa forma, $map[1][2][3]$ = "*frente*" quer dizer que caso o robô esteja no vértice 1, tenha vindo do vértice 2 e pretenda ir para o vértice 3, ele deve seguir em frente.

A partir do grafo criado é possível também utilizar o algoritmo BFS para descobrir o caminho mínimo entre quaisquer dois vértices, já que os dois primeiros níveis do dicionário representam as arestas entre os vértices. Com o conjunto de vértices de caminho mínimo, pode-se obter as direções para chegar até o destino final. Se o objetivo do robô for partir do vértice 1 e chegar ao vértice 5, o caminho mínimo será então *[1, 3, 4, 5]* e o conjunto de direções *["frente", "frente", "direita"]* (assumindo que o robô está virado em direção ao vértice 3 em sua posição inicial).

Duas funções foram desenvolvidas para serem utilizadas na lógica do algoritmo final: uma para encontrar o caminho mínimo entre quaisquer dois vértices do grafo usando o algoritmo BFS e outra para extrair o conjunto de direções a partir de um caminho através da análise da estrutura montada com o *array* tridimensional.

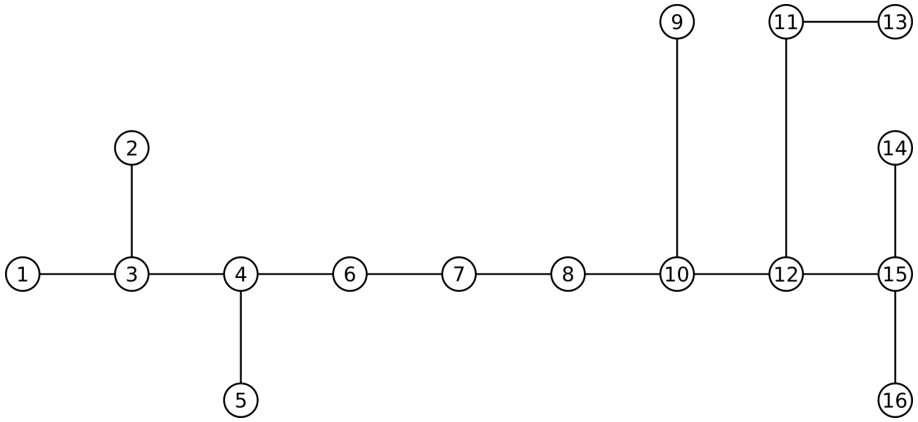

Figura 9. Mapa do ambiente

3.3. Estabilizando o robô no centro da linha

Na maior parte do tempo, o principal trabalho que o robô precisará desempenhar é manter-se no centro da linha para garantir que está no caminho correto. Para seguir a linha o algoritmo contabiliza o número de *pixels* brancos em uma determinada linha da imagem (foi utilizado a linha posicionada em 1/4 da altura da imagem) e faz uma média normalizada de suas posições, resultando em um número entre 0 e 1, em que 0.5 significa que o piso está exatamente no centro da imagem. Na Figura 10 o posicionamento dessa média é representado por um círculo vermelho em cima da linha horizontal.

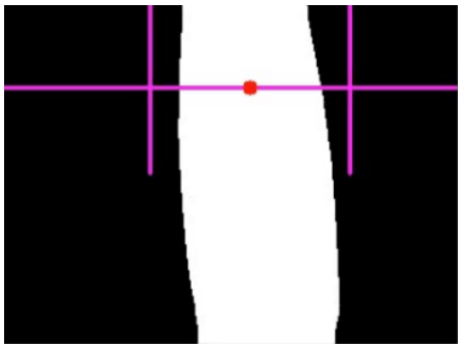

Figura 10. Máscara gerada com indicador da média dos *pixels* brancos

Obtendo a média normalizada é possível enviar para o módulo de deslocamento que tipo de movimento deve ser realizado, sendo que valores de média entre 0 e 0.5 geram curvas para a esquerda e entre 0.5 e 1 à direita. A acentuação da curva depende do valor da média, sendo que valores próximos a 0 geram curvas fechadas à esquerda e em valores próximos a 1 geram curvas fechadas à direita. A Figura 11 mostra uma máscara com média próxima a 0, em que é possível perceber que a câmera não está alinhada com o piso tátil e deve fazer uma curva para a esquerda para endireitar o robô.

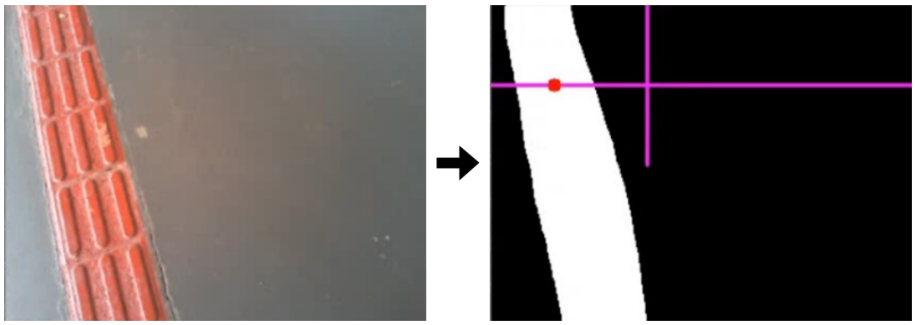

Figura 11. Exemplo de imagem com média à esquerda

3.4. Detectando intersecções

Para detectar intersecções no chão, o algoritmo se utiliza de linhas verticais posicionadas à direita e esquerda do ponto de média que detectam se há a presença de um piso tátil em sua coluna. A Figura 12 demonstra à esquerda uma foto sem uma intersecção e à direita com, em que as linhas rosas detectam a presença do piso sobre elas. Caso haja a detecção de uma linha sobre essas colunas é deduzido que há uma intersecção em frente do robô, significando que foi chegado em um novo vértice do mapa.

Figura 12. Exemplo de imagem sem e com intersecção

A Figura 13 mostra em sua primeira linha máscaras de casos onde não há intersecções em frente da câmera. Na linha de baixo é mostrado casos em que há intersecções em frente e elas são detectadas pelas colunas à esquerda e à direita da média dos *pixels* brancos.

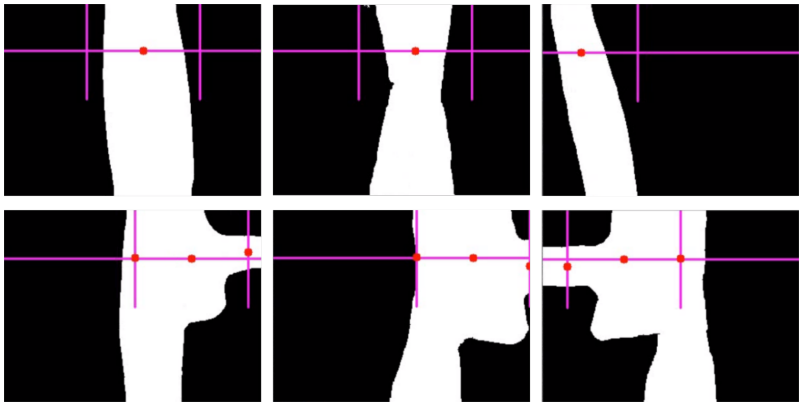

Figura 13. Exemplos de máscaras sem e com intersecções

É importante notar que a função é escrita para uma distância entre o chão e a câmera específica e resultará em erros caso a câmera seja posicionada com um afastamento muito menor do que o ideal.

3.5. Lógica de operação

A primeira ação do algoritmo é pedir ao usuário o número do vértice no qual o robô está atualmente. Esse vértice deve estar em uma das pontas do grafo, ou seja, deve ter apenas um vizinho e estar de frente para ele. Após isso, o algoritmo pede ao usuário o número do vértice de destino que o robô deve chegar. Tendo os dois vértices, são usadas as funções que obtém o caminho mínimo entre eles e o conjunto de direções que eles geram (por exemplo, *[“frente”, “frente”, “direita”, “esquerda”]*) para obter as direções que devem ser tomadas pelo robô para chegar até seu vértice final. Após a obtenção do conjunto de direções o algoritmo começa a capturar as imagens da câmera e aplicar a elas a *pipeline* de visão computacional apresentada nesse capítulo. A média é então obtida e em seguida a função que detecta se há uma intersecção em frente é chamada para a verificação se o robô está em um dos vértices do grafo ou não.

Caso a função detecte que há um vértice em frente, é marcado através de um contador que nessa imagem da câmera foi detectado uma intersecção, e caso 5 imagens consecutivas sejam marcadas como intersecções (para evitar possíveis erros falso-positivos), o algoritmo infere que a intersecção existe em frente do robô e ele deve tomar uma decisão de movimentação. A decisão é definida pelo valor no conjunto de direções, que determina se ele deve seguir em frente, fazer uma curva de 90 graus para a esquerda ou direita. Em todos os ciclos em que a imagem não apresenta sinais de uma intersecção o algoritmo apenas usa o valor da média obtida para manter o robô seguindo a linha em frente. Ao chegar no vértice final o algoritmo pede ao usuário o número de um novo vértice para ele dirigir-se, reiniciando o ciclo do caminho.

4. Resultados

O algoritmo foi desenvolvido e testado com o transporte de um *notebook* com uma câmera de forma manual, em que a pessoa carregando-os segue os comandos do algoritmo, que a ajudam a estabilizar a câmera sempre no centro do piso tátil (comandando curvas leves à esquerda ou direita) e avisa quando uma intersecção está visível, dando direções

seguintes (trás, frente, direita ou esquerda) ou avisando que o destino final foi alcançado. A Figura 14 mostra a interface do algoritmo para testes manuais, apresentando: (1) a imagem original da câmera, (2) a imagem processada pela *pipeline* com os indicadores de média de *pixels* e limites laterais e (3) uma caixa de mensagens que dá ordens para a pessoa que está carregando o *notebook*. O algoritmo utilizado para os testes é o mesmo apresentado na seção anterior. Deve ser notado que para o uso em um robô ele deve ser adaptado para interagir com o *hardware* para transformar os comandos em movimentos nos atuadores da máquina.

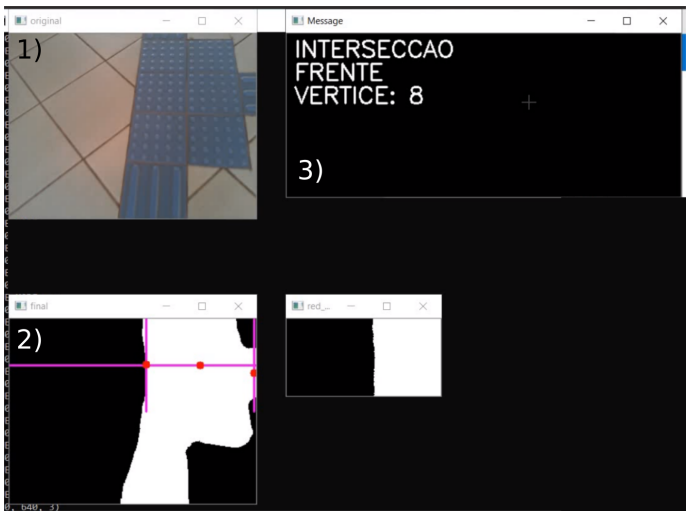

Figura 14. Interface para testes

Quanto ao desempenho dos testes, em condições de iluminação onde o sol não está presente acima do piso tátil o algoritmo tem bom desempenho, detectando de forma correta as linhas do piso e intersecções, conduzindo o transportador (e em tese, o robô) até o local de destino final com velocidade de caminhada média de cerca de 1m/s. Nessas condições, em cem por cento das vezes o algoritmo conseguiu chegar no ponto desejado, mas esse resultado deve-se parcialmente à estabilidade da câmera que um humano consegue garantir durante seu deslocamento. Em condições em que o sol está presente em parte das linhas e não está em outras o desempenho é inferior, necessitando que a velocidade de caminhada seja menor para dar tempo à câmera se adaptar durante o caminho às diferentes condições de luz. Isso ocorre devido à diferença de cores entre as partes com mais e menos iluminação, que dificulta a segmentação por cores de maneira correta. A Figura 15 mostra na primeira linha uma imagem sem a incidência de luz sobre o piso tátil e seu processamento à direita. Na segunda linha é apresentado uma imagem com a incidência do sol sobre a linha, deixando claro que o processamento da imagem é distorcido por ele. Em algumas ocorrências a linha segmentada pode aumentar a ponto de ser detectada intersecções que não existem, mas na maior parte do tempo esse problema é mitigado pelo mecanismo de segurança que apenas detecta uma intersecção quando 5 imagens consecutivas têm essa característica.

A troca dinâmica entre as máscaras que segmentam o chão vermelho e as que segmentam o chão azul funciona bem e de forma rápida quando o piso é alterado, mostrando que o algoritmo consegue se adaptar a mais de um tipo de terreno.

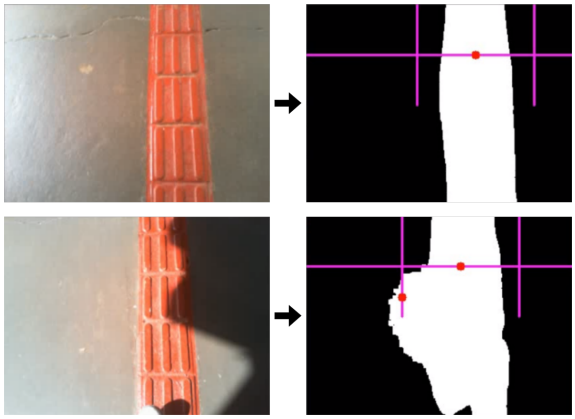

Figura 15. Processamentos sem e com iluminação variante

5. Conclusões

Visão computacional é uma área que cresce amplamente na academia e indústrias em decorrência de sua grande aplicabilidade em diferentes áreas. Esta pesquisa demonstra que a tecnologia pode ser aplicada ao problema de localização de robôs autônomos, empregando-a à clássica técnica de seguidores de linha, mas a aperfeiçoando com o novo conjunto de ferramentas. O algoritmo desenvolvido tem potencial de ser usado em sistemas de transporte e de assistência para cegos em locais que já possuem pisos táteis, apresentando melhora em relação à outras tecnologias que poderiam ser usadas para os mesmos fins. Ainda, com pequenas mudanças, o protótipo pode também ser utilizado para detecção de outros tipos de linhas que se diferenciam do chão por sua cor.

A pesquisa resultou em um algoritmo com as seguintes funções:

1. Detecção de linhas;
2. Detecção de intersecções em linhas;
3. Uma forma de mapear ambientes;
4. Utilização da detecção de linhas e do mapa para criar um direcionador que aponta o caminho entre dois locais do mapa.

Conclui-se então que é possível utilizar técnicas de visão computacional e o método de seguir linhas para resolver o problema de localização de robôs.

O objetivo geral desta pesquisa foi atingido, pois o algoritmo foi proposto, desenvolvido e testado. Apesar do algoritmo não ter sido testado em um *hardware*, os testes manuais mostram a eficácia do algoritmo desenvolvido, que demonstra que é possível integrá-lo a um robô.

5.1. Trabalhos futuros

Para trabalhos futuros, pode-se confeccionar um robô com o *hardware* proposto neste trabalho e aplicar o algoritmo a ele, observando seu desempenho e o modificando para contornar possíveis problemas, como uma câmera instável. É observado também que a adaptação do algoritmo a diferentes padrões de iluminação pode ser melhorada. Para tal, deve-se buscar uma nova forma de segmentação do chão e linha ou uma forma de transformar a segmentação por cores dinâmica enquanto o robô está andando, adaptando-a de acordo com as condições de luz. Ainda, vale a pena explorar outras possibilidades

de *hardware* que possam utilizar a mesma *pipeline* de visão computacional apresentada neste trabalho com custo econômico ou computacional menor.

Referências

Cheng, H., Jiang, X., Sun, Y., and Wang, J. (2001). Color image segmentation: advances and prospects. *Pattern Recognition*, 34(12):2259–2281.

Das, A., Degeling, M., Wang, X., Wang, J., Sadeh, N., and Satyanarayanan, M. (2017). Assisting users in a world full of cameras: A privacy-aware infrastructure for computer vision applications. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1387–1396.

Elhady, W. E., Elnemr, H. A., and Selim, G. (2014). Implementation and evaluation of image processing techniques on a vision Navigation line follower robot. *Journal of Computer Science*, 10(6):1036–1044.

Gonzalez, R. C. and Woods, R. E. (2009). *Processamento Digital de Imagens*. Pearson, 3 edition.

Guibas, L. J., Motwani, R., and Raghavan, P. (1997). The robot localization problem. *SIAM J. Comput.*, 26(4):1120–1138.

Keisner, A., Raffo, J., and Wunsch-Vincent, S. (2016). Robotics: Breakthrough technologies, innovation, intellectual property. *Foresight and STI Governance*, 10(20):7–27.

OpenCV (2014). Eroding and dilating.

Pakdaman, M., Sanaatiyan, M. M., and Rezaei, M. (2010). A line follower robot from design to implementation: Technical issues and problems. pages 5 – 9.

Punetha, D., Kumar, N., and Mehta, V. (2013). Development and applications of line following robot based health care management system. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 2:2446–2450.

Saravanan, G., Yamuna, G., and Nandhini, S. (2016). Real time implementation of RGB to HSV/HSI/HSL and its reverse color space models. In *2016 International Conference on Communication and Signal Processing (ICCSP)*. IEEE.

Shadeed, W. G., Abu-Al-Nadi, D. I., and Mismar, M. J. (2003). Road traffic sign detection in color images. In *10th IEEE International Conference on Electronics, Circuits and Systems, 2003. ICECS 2003. Proceedings of the 2003*, volume 2, pages 890–893 Vol.2.

Siegwart, R. and Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots (Intelligent Robotics and Autonomous Agents series)*. The MIT Press.

Wang, C.-M. J. (1988). Location estimation and uncertainty analysis for mobile robots. pages 1231 – 1235.