

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

ANDERSON LUIS COELHO ZAPELLO

**DISPOSITIVO DE DETECÇÃO DE OBSTÁCULOS COM RESPOSTA VIBROTÁTIL
PARA PESSOAS COM DEFICIÊNCIA VISUAL**

FLORIANÓPOLIS, SC

2017

ANDERSON LUIS COELHO ZAPELLO

**DISPOSITIVO DE DETECÇÃO DE OBSTÁCULOS COM RESPOSTA VIBROTÁTIL
PARA PESSOAS COM DEFICIÊNCIA VISUAL**

Proposta de Trabalho de Conclusão de Curso
submetido ao Programa de graduação da Universidade
Federal de Santa Catarina para a obtenção do Grau de
Bacharel em Ciências da Computação.

FLORIANÓPOLIS, SC

2017

ANDERSON LUIS COELHO ZAPELLO

**DISPOSITIVO DE DETECÇÃO DE OBSTÁCULOS COM RESPOSTA VIBROTÁTIL
PARA PESSOAS COM DEFICIÊNCIA VISUAL**

BANCA EXAMINADORA

Prof.^a. Dr.^a. Patrícia Della Mea Plentz - UFSC
Orientadora

Ms. Sergio Genilson Pflieger - UFSC
Co-orientador

Prof. Dr. Mario Antonio Ribeiro Dantas - UFSC
Membro Examinador

Prof. Dr. Márcio Bastos Castro - UFSC
Membro Examinador

FLORIANÓPOLIS, SC
2017

AGRADECIMENTOS

Agradeço a Deus por tudo, pela minha família, pelos meus amigos, pelos meus estudos e por mais uma etapa concluída na minha vida, após muito sacrifício. Obrigado Deus por ter me dado forças para não desistir mesmo nos momentos de incerteza, jamais terei como Lhe agradecer plenamente.

Agradeço a Professora Prof.^a. Dr.^a. Patrícia Della Mea Plentz, por ter aceitado me orientar, pela paciência e por alguns puxões quando eu me ausentava.

Agradeço a minha família, sem ela eu jamais teria chegado até aqui.

Nada do que eu diga será capaz de se aproximar da gratidão que tenho pela Grazieli Biduski, minha companheira amada e fiel, que esteve sempre ao meu lado em todos os momentos e principalmente nos momentos mais difíceis, que acreditou em mim mais do que eu mesmo. Sem você meu amor eu jamais teria conseguido. Obrigado por ser tão paciente, compreensiva (mesmo que nem sempre fosse) e por não ter me deixado desistir nunca.

Agradeço também a Meg, minha cachorrinha, que me alegrou nos momentos complicados, que fez com que eu tirasse o foco dos problemas. Além de me fazer ver que a vida pode ser mais simples e fácil e que nós humanos costumamos complicar demais as coisas.

RESUMO

Uma das maiores dificuldades que pessoas com problemas de baixa visão ou cegueira possuem é relacionada a mobilidade. Este projeto tem como objetivo desenvolver um sistema que auxilie pessoas com deficiência visual a ter sua mobilidade melhorada. Para isso foi desenvolvido um dispositivo que utiliza visão computacional para detectar obstáculos e informar ao usuário através de informações táteis, geradas por micromotores de vibração, a distância dos obstáculos, quanto maior a vibração mais próximos eles se encontram.

O sistema é composto de um dispositivo vestível que contém duas câmeras para obtermos uma visão estereó do ambiente. As imagens capturadas são enviadas para um notebook que o usuário poderá carregar em uma mochila. O notebook contém o software que faz o processamento das imagens e gera o mapa de profundidade que é utilizado para fazer o mapeamento entre os obstáculos identificados e suas distâncias. As informações são enviadas para o grid de motores que usuário estará usando, com isso ele é informado através de vibrações se o obstáculo se encontra a sua direita, esquerda ou centro além de uma noção da altura que o obstáculo se encontra.

Palavras chave: deficiência visual, tecnologia assistiva, visão computacional, motores vibratórios, mapa de profundidade.

LISTA DE FIGURAS

Figura 1 – Representação de uma câmera pinhole	21
Figura 2 – Exemplo de distorção radial em um grid retangular.	22
Figura 3 – Exemplo de retificação estéreo	24
Figura 4 – Um exemplo de mapa de disparidade	26
Figura 5 – Triangulação em sistema estéreo	28
Figura 6 – Relação inversamente proporcional da profundidade de disparidade	28
Figura 7 – Sistema proposto por Lee e Medioni	29
Figura 8 – Bloco funcional do projeto do RASHID.	31
Figura 9 – Exemplo de funcionamento e os componentes utilizados no projeto do Poggi	31
Figura 10 – Esquema funcional e informacional do projeto	33
Figura 11 – Primeiro modelo da câmera utilizado no protótipo	37
Figura 12 – Câmera interna de notebook utilizada.....	37
Figura 13 – Conexões entre webcam e cabo USB.....	38
Figura 14 – Fluxograma do software do módulo de captura e processamento de imagens	39
Figura 15 – Exemplo de pares de imagens usadas na calibração	40
Figura 16 – Exemplo de pares de imagens com detecção do tabuleiro	41
Figura 17 – Exemplo de pares de imagens não retificadas e retificadas	42
Figura 18 – Etapas realizadas pelo módulo.	44
Figura 19 – Exemplo de um mapa de disparidade e segmentação do mapa.	46
Figura 20 – Exemplos de mapa de disparidade para calcular os histogramas.....	48
Figura 21 – Representação da codificação dos valores enviado para o Arduíno	49
Figura 22 – Conexão entre módulo informativo vibrátil e de identificação de obstáculos.	51
Figura 23 – Arduíno Uno e detalhe dos componentes utilizados	53
Figura 24 – Fluxo de dados e interação entre módulos.	54
Figura 25 – Representação do grid de motores vibratórios e regiões do mapa de disparidade	55
Figura 26 – Motor vibratório em escala real.	56
Figura 27 – Transistor NPN BC337	57
Figura 28 – Circuito projetado e esquemático do módulo informativo vibrátil	58
Figura 29 – Dispositivo criado para módulo de captura de imagens.....	59
Figura 30 – Dispositivo controlador dos motores vibratórios	59
Figura 31 – Visão mais detalhada do controlador	60
Figura 32 – Visão interna e externa do dispositivo informativo vibrátil.....	61

Figura 33 – Usuário com dispositivo informativo vibrátil	61
Figura 34 – Sistema completo em uso.....	62
Figura 35 – Exemplos das amostras utilizadas no teste 1.....	64
Figura 36 – Gráfico disparidade x distância para resolução 352x288.....	65
Figura 37 – Gráfico disparidade x distância para resolução 640x480.....	66
Figura 38 – Gráfico da distância real x distância estimada	67
Figura 39 – Gráfico dos erros das estimativas.....	67
Figura 40 – Mapa de disparidade de objeto a 25 cm	69
Figura 41 – Mapa de disparidade de objeto a 40 cm	69
Figura 42 – Mapa de disparidade de objeto a 55 cm	70
Figura 43 – Mapa de disparidade para 2 objetos a distâncias diferentes.....	72
Figura 44 – Tempo de execução da captura das duas imagens	74
Figura 45 – Tempo de execução do algoritmo de geração do mapa de disparidade	75
Figura 46 – Tempo de execução do cálculo, codificação e envio das intensidades	75
Figura 47 – Tempo total de execução do módulo de detecção de obstáculos	76
Figura 48 – Tempo total de execução do processamento do fluxo inteiro do sistema	76

LISTA DE QUADROS

Quadro 1 – Artefatos e versões utilizadas.	36
Quadro 2 – Especificações da câmera	38
Quadro 3 – Exemplo de cálculo de histogramas para mapa de disparidades	48
Quadro 4 – Mapeamento de intensidades.....	48
Quadro 5 – Especificações técnicas do Arduino UNO.....	52
Quadro 6 – Especificações técnicas do motor vibratório	56
Quadro 7 – Medições de Disparidade x Distância	65
Quadro 8 – Distância real x distâncias estimadas.....	66
Quadro 9 – Distância estimada e erro.....	68
Quadro 10 – Mapa de disparidade x longa distância.....	70

SUMÁRIO

1	INTRODUÇÃO	11
1.1	OBJETIVOS	11
1.1.1	<i>Objetivo Geral</i>	12
1.1.2	<i>Objetivos Específicos</i>	12
1.2	MÉTODO DA PESQUISA	12
1.3	ESTRUTURA DO TRABALHO	13
2	FUNDAMENTAÇÃO TEÓRICA	15
2.1	TECNOLOGIA ASSISTIVA	15
2.1.1	<i>Produtos de Tecnologia Assistiva</i>	15
2.1.2	<i>Tecnologia Assistida para pessoas com deficiência visual</i>	16
2.1.3	<i>Mobilidade e Orientação de pessoas com deficiência visual</i>	18
2.2	VISÃO COMPUTACIONAL	18
2.3	MODELO DE CÂMERAS	20
2.4	DISTORÇÕES DAS LENTES	21
2.5	VISÃO ESTÉREO	22
2.5.1	<i>Calibração estéreo</i>	23
2.5.2	<i>Retificação estéreo</i>	24
2.6	DISPARIDADE E CORRESPONDÊNCIA ESTÉREO	25
2.7	TRABALHOS RELACIONADOS	29
3	PROTÓTIPO	33
3.1	RECURSOS UTILIZADOS PARA O DESENVOLVIMENTO DO PROJETO	34
3.1.1	<i>Arduíno</i>	34
3.1.2	<i>Ambiente de desenvolvimento e bibliotecas</i>	35
3.1.3	<i>OpenCV</i>	36
3.2	MÓDULO DE CAPTURA E PROCESSAMENTO DE IMAGENS	36
3.2.1	<i>Câmeras</i>	37
3.2.2	<i>Computador</i>	38
3.2.3	<i>Software</i>	38
3.2.3.1	<i>Calibração e retificação</i>	39
3.2.3.2	<i>Inicialização das câmeras</i>	43
3.2.3.3	<i>Captura e processamento de imagens</i>	43
3.3	MÓDULO DE IDENTIFICAÇÃO DE OBSTÁCULOS	44
3.3.1	<i>Mapa de disparidade</i>	44
3.3.2	<i>Mapeamento da disparidade para Intensidade</i>	46
3.3.3	<i>Codificação das intensidades</i>	49
3.3.4	<i>Envio das informações para o Arduíno</i>	50
3.4	MÓDULO INFORMATIVO VIBRÁTIL	51

3.4.1	<i>Controlador</i>	51
3.4.1.1	Hardware.....	51
3.4.1.2	Software.....	53
3.4.2	<i>Atuadores</i>	55
3.5	PROTÓTIPO COMPLETO	58
4	EXPERIMENTOS E RESULTADOS	63
4.1	TESTES DE VISAO COMPUTACIONAL.....	63
4.1.1	<i>Parâmetros utilizados</i>	63
4.1.2	<i>Testes distância x disparidade</i>	64
4.1.2.1	Resultados.....	65
4.1.3	<i>Testes de distância mínima e distância máxima</i>	68
4.1.3.1	Resultados.....	68
4.2	TESTES DE DESEMPENHO.....	73
4.2.1	<i>Resultados</i>	74
4.3	TESTES DE INTEGRAÇÃO.....	77
4.3.1	<i>Resultados</i>	78
4.4	DIFICULDADES E PROBLEMAS	89
5	CONSIDERAÇÕES FINAIS	91
5.1	TRABALHOS FUTUROS.....	92
	REFERÊNCIAS	93
	APÊNDICE A – ARQUIVO XML COM OS PARÂMETROS DE CALIBRAÇÃO DAS CÂMERAS	96
	APÊNDICE B – CÓDIGO FONTE DOS MÓDULOS DESENVOLVIDOS	99
	APÊNDICE C – CÓDIGO FONTE DO MÓDULO INFORMATIVO VIBRÁTIL	114

1 INTRODUÇÃO

Segundo a Organização Mundial da Saúde (OMS) (2014) existem aproximadamente 285 milhões de pessoas no mundo com deficiência visual: 39 milhões com cegueira total e 246 milhões consideradas com baixa visão. Se olharmos para o Brasil, segundo o Instituto Brasileiro de Geografia e Estatística (IBGE) (2010), através do censo de 2010, existem mais de 500 mil pessoas totalmente cegas e em torno de 6 milhões que enxergam com grande dificuldade.

Conforme Bueno (1992), um dos principais problemas que as pessoas com deficiência visual enfrentam é o de orientação e mobilidade. Mobilidade está relacionado a capacidade de mover-se com facilidade e segurança. Já a orientação diz respeito a percepção que o indivíduo tem do ambiente e da sua posição nele. Desse modo, a mobilidade é uma capacidade inata, mas a orientação é uma habilidade que deve ser aprendida como um relacionamento entre o indivíduo e o ambiente. Isso significa que os deficientes visuais devem adquirir o sentido de orientação através de outros meios, sejam eles auditivos ou táteis. Já a mobilidade é alcançada através de sistemas de treinamento que podem envolver mecanismos mecânicos, eletrônicos, como por exemplo bengalas, radares etc. Para essas ferramentas que auxiliam as pessoas com algum tipo de problema damos o nome de tecnologia assistiva.

O objetivo da tecnologia assistiva é superar a lacuna entre o que uma pessoa com deficiência quer fazer e o que a infraestrutura social existente permite que elas façam. Ela consiste de equipamentos, dispositivos e sistemas que podem ser usados para superar as barreiras sociais, de infraestrutura entre outras barreiras sentidas pelas pessoas com deficiência e que impedem a sua participação plena e igualitária em todos os aspectos da sociedade (HERSH; JOHNSON, 2008).

Visto que a mobilidade é algo de muita importância para o ser humano e todas as dificuldades encontradas por deficientes visuais em executarem essa tarefa de forma autônoma, este trabalho tem como tema desenvolver um dispositivo que utilize visão computacional para auxiliar na mobilidade de pessoas com deficiência visual, esta tarefa é realizada através da identificação de obstáculos que estejam localizados a frente do usuário e sua proximidade será informada através de vibrações com diferentes intensidades em seu corpo.

1.1 OBJETIVOS

Os objetivos desse trabalho são especificados a seguir.

1.1.1 Objetivo Geral

Este trabalho tem por objetivo desenvolver um protótipo de baixo custo de um dispositivo que usa visão computacional para o auxílio na mobilidade de pessoas com deficiência visual. Ele deve ser capaz de identificar obstáculos que estejam a frente do usuário e informar, através de um grid de motores vibratórios, a proximidade que o usuário se encontra desses obstáculos. Esses motores deverão vibrar com intensidades diferentes conforme o usuário se aproxima ou se afasta do obstáculo. Quanto maior a intensidade mais próximo do obstáculo. O dispositivo será composto de duas câmeras para termos visão estereoscópica, um notebook que conterà o software para a análise e processamento das imagens para a identificação e obtenção das distancias dos obstáculos e também conterà um grid de motores vibratórios.

1.1.2 Objetivos Específicos

- Implementar um algoritmo que transforme o mapa de profundidade em informação tátil.
- Desenvolver os protótipos dos dispositivos, tanto o de visão computacional, quanto o de informação da proximidade do obstáculo.
- Avaliar as distâncias máximas e mínimas aceitas pelo sistema.
- Avaliar e validar o protótipo desenvolvido junto a um deficiente visual.

1.2 MÉTODO DA PESQUISA

Este projeto consiste de uma pesquisa aplicada em Ciências da Computação, pois serão utilizados conhecimentos em visão computacional para o desenvolvimento de um sistema, através de um protótipo, que auxilie na mobilidade de pessoas com problemas de baixa visão ou cegueira. As técnicas e ferramentas relevantes serão pesquisadas em artigos científicos, livros-textos e internet.

O sistema consistirá de duas câmeras USB do mesmo modelo acopladas em um óculos e que estarão conectadas a um notebook em uma mochila do usuário. A utilização de duas câmeras é devido a necessidade de visão estéreo, pois isso permite a extração da profundidade que é essencial para podermos identificar a distância que se encontram os obstáculos. O

notebook será responsável por receber as imagens e realizar a análise e processamento das imagens através de um software desenvolvido usando a biblioteca de visão computacional OpenCV, a principal tarefa desse software será extrair o mapa de profundidade das imagens. A partir do mapa gerado, o algoritmo desenvolvido nesse trabalho fará o mapeamento dos obstáculos detectados para informações táteis. Estas informações serão enviadas ao grid de motores vibratórios responsáveis por informar que há um obstáculo na frente do usuário.

Inicialmente será implementado o algoritmo que converte o mapa de profundidade em informações táteis. Como ainda não teremos desenvolvido o software que gera o mapa a partir das imagens capturadas pelas duas câmeras usaremos o dispositivo Kinect da Microsoft que fornece o mapa pronto através de sua API. A captura dessas imagens será realizada no laboratório da UFSC, pois o Kinect é de propriedade da Universidade. O algoritmo será validado usando um protótipo que utilizará leds no lugar dos motores e quanto maior a intensidade do led mais próximo do obstáculo o usuário estará.

Após a implementação do algoritmo será desenvolvido o dispositivo vestível que conterá as câmeras. Junto a essa etapa será desenvolvido o software que fará a geração do mapa de profundidade. Esse software será implementado utilizando a biblioteca de visão computacional OpenCV.

O grid de motores será o responsável por informar ao usuário através da intensidade da vibração dos motores a distância de um obstáculo, quanto maior a vibração mais próximo esse obstáculo estará. Essa matriz conterá 9 motores dispostos em 3 linhas e 3 colunas, com isso será possível informar ao usuário que um obstáculo está a esquerda, direita ou centro. O dispositivo será controlado por um Arduíno e software de controle do grid também será desenvolvido nesse trabalho.

A última etapa do desenvolvimento serão os testes de integração de todos os artefatos desenvolvidos, tanto os softwares quanto os hardwares. Por questões de segurança os primeiros testes serão realizados pelo próprio autor e terão como cenário a UFSC. Após essa primeira validação será utilizado algum usuário portador de deficiência visual para termos um feedback real, esse teste será supervisionado pelo autor.

1.3 ESTRUTURA DO TRABALHO

Este trabalho está organizado em 5 Capítulos conforme a seguir. O Capítulo 1 contém a uma breve introdução, que nos apresenta a motivação do projeto, os objetivos e os métodos que serão utilizados no trabalho. O Capítulo 2 contém os principais tópicos utilizados como

fundamentação teórica no trabalho. O Capítulo 3 apresenta o detalhamento do protótipo que foi desenvolvido. O Capítulo 4 são mostrados os testes realizados e os resultados obtidos. O Capítulo 5 apresenta as considerações finais do trabalho, assim como as possibilidades de trabalhos futuros

2 FUNDAMENTAÇÃO TEÓRICA

2.1 TECNOLOGIA ASSISTIVA

A expressão tecnologia assistiva é relativamente nova e seu conceito ainda está em processo de construção e sistematização. Apesar do conceito ser remetido a diferentes concepções ao longo da história da humanidade e possuir características particulares dependendo do país utilizado como referência é possível identificar que o principal objetivo é a qualidade de vida. A partir do momento em que o desenvolvimento tecnológico traz soluções através do desenvolvimento de ferramentas ou práticas que possam favorecer, compensar, potencializar ou auxiliar pessoas que tem suas funções ou habilidades comprometidas por alguma deficiência, independentemente do tipo, ou até mesmo causadas pelo processo natural de envelhecimento estamos falando do conceito de tecnologia assistiva (RODRIGUES e ALVES, 2014; GALVAO FILHO, 2009; BERSCH, 2008,2009).

Mesmo que o conceito de tecnologia assistida possa ainda estar sendo construído e existir denominações diferentes em diversos países, tais como “ajudas técnicas” e “tecnologia de apoio” o uso de tecnologias que auxiliam o ser humano estão presentes desde os primórdios da humanidade, pois qualquer pedaço de pau improvisado como uma bengala pode ser considerado tecnologia assistiva (RODRIGUES e ALVES, 2014; GALVAO FILHO, 2009). Podemos ver isso através do que diz Manzini:

Os recursos de tecnologia assistiva estão muito próximos do nosso dia-a-dia. Ora eles nos causam impacto devido à tecnologia que apresentam, ora passam quase despercebidos. Para exemplificar, podemos chamar de tecnologia assistiva uma bengala, utilizada por nossos avós para proporcionar conforto e segurança no momento de caminhar, bem como um aparelho de amplificação utilizado por uma pessoa com surdez moderada ou mesmo veículo adaptado para uma pessoa com deficiência (apud GALVAO FILHO, 2009, p. 128).

Segundo Rodrigues (2014), para promover a padronização da terminologia adotada no Brasil o Comitê de ajudas técnicas adotou o seguinte conceito de tecnologia assistiva:

[...]é uma área do conhecimento, de característica interdisciplinar, que engloba produtos, recursos, metodologias, estratégias, práticas e serviços que objetivam promover a funcionalidade, relacionada à atividade e participação, de pessoas com deficiência, incapacidades ou mobilidade reduzida, visando sua autonomia, independência, qualidade de vida e inclusão social (BRASIL, 2009).

2.1.1 Produtos de Tecnologia Assistiva

Os produtos de tecnologia assistiva se encaixam em um conjunto muito grande de recursos e possibilidades, pois qualquer produto, seja uma ferramenta, equipamentos,

dispositivos, instrumentos, sistemas, softwares produzidos especialmente ou disponibilizados no mercado que auxiliem as atividades de uma pessoa idosa ou com deficiência de modo a prevenir, compensar, controlar, reduzir ou neutralizar as suas limitações é considerado um produto de tecnologia assistiva. Isto significa uma aplicação específica da tecnologia para melhorar a qualidade de vida das pessoas (ISO, 2002; GALVAO FILHO, 2009).

Como a variedade de produtos é grande podemos agrupá-los a partir da complexidade da tecnologia aplicada, dividindo em dois principais grupos: baixa tecnologia ou *low-tech* e alta tecnologia ou *high-tech*. Essa divisão não significa que um seja mais eficiente ou possui mais funcionalidades que o outro é, mas somente a sofisticação dos seus componentes. Com isso são considerados produtos de tecnologia assistidas desde uma colher adaptada, uma bengala, até sistemas automatizados de controle de ambiente, sendo que os dois primeiros são classificados como *low-tech* e o último como *high-tech* (GALVAO FILHO, 2009).

Para Sanz (2013), não há um produto assistivo que seja aplicável para qualquer tipo de problema e por definição ele é a tecnologia aplicada, desde o início, para resolver problemas específicos. Com isso o projeto é limitado pela aplicação final e que deve ser levado em conta se o produto quer ser aplicável. Então, temos que nos concentrar em um problema específico e, portanto, coletivo, para ser capaz de projetar e implementar uma solução prática. Desse modo, podemos ver a importância da escolha da população-alvo e do problema. Esta é uma escolha arbitrária, uma vez que existem muitos e subjetivamente avaliados problemas na vida das pessoas com deficiência, mas que determinaram o design, implementações, testes e a praticidade do novo dispositivo.

2.1.2 Tecnologia Assistida para pessoas com deficiência visual

De acordo com a Organização Mundial da Saúde (2014), existem aproximadamente 285 milhões de pessoas no mundo com deficiência visual: 39 milhões com cegueira total, 246 milhões consideradas com baixa visão e aproximadamente 90% das pessoas com deficiências visuais vivem em ambientes de baixa renda. Em 2010, a deficiência visual foi considerada pela OMS como um problema grave e global de saúde e constatou que 80% dos casos são evitáveis destaca o Brasil como um bom exemplo na prestação de serviços oculares através do sistema público de saúde. Mas mesmo assim, se olharmos com detalhes para o Brasil, segundo o Instituto Brasileiro de Geografia e Estatística (2010), através do censo de 2010, existem mais de 500 mil pessoas totalmente cegas e em torno de 6 milhões que enxergam com grande dificuldade.

De acordo com a OMS, existem quatro níveis funcionais de visão: visão normal, deficiência visual moderada, deficiência visual grave e cegueira. O agrupamento das deficiências moderada e grave são chamados de baixa visão e juntos com a cegueira representam a deficiência visual. Para a avaliação da capacidade visual são usados dois componentes: acuidade (aquilo que se enxerga a determinada distância) e campo visual (a amplitude da área alcançada pela visão). A cegueira não significa a total incapacidade de ver, pois ela reúne pessoas com diversos graus de visão residual, com isso é considerada cega quem tem o prejuízo da visão nas tarefas diárias de um modo incapacitante. É considerada uma pessoa com baixa visão aquela que tem o comprometimento visual mesmo após tratamento de correção e tem uma acuidade visual menor que 6/18 para a percepção de luz ou um campo visual menor que 10 graus a partir do ponto de fixação, mas mesmo assim é capaz de utilizar a visão para planejar ou executar uma tarefa (OMS, 2008, 2014; TALEB et al, 2012).

Quando um indivíduo tem um uma deficiência sensorial, tecnologias assistivas podem fornecer ajuda na entrada de informações e é possível melhorar a performance nas atividades diárias através da adoção de algumas tecnologias que são classificadas da seguinte maneira: cuidados pessoais, alarmes e alertas, preparação e consumo de alimentos, controle ambiental, dinheiro e finanças.

Podemos ver alguns exemplos para cada uma dessas classes, no que se refere aos cuidados pessoais, as pessoas com deficiência visual precisam de ajuda para identificar diferentes elementos de suas roupas e a rotulagem pode ser agrupada nos seguintes tipos de sistemas: sistemas de etiquetagem tátil (etiquetas táteis utilizando Braille, por exemplo), sistemas com *tags* de rádio frequência. Além disso, a fim de ajudar as pessoas com deficiência visual em monitorar sua saúde geral, pode-se encontrar uma série de dispositivos com saída de áudio ou tátil para medir a temperatura, pressão arterial, peso corporal, nível de glicose no sangue. Pode-se encontrar diferentes tipos de dispositivos de tempo e muitos deles têm saída de voz, mas também há temporizadores com braile ou outra exibição tátil (HERSH, 2008).

A preparação e o consumo de alimentos podem ser um verdadeiro desafio para deficientes visuais. A solução para este problema pode ser uma simples mudança na concepção de fogões comuns, talheres e ferramentas de cozinha, a fim de torná-los seguros para usar e também a utilização de dispositivos que informem através de uma saída de voz, como por exemplo uma balança que informa através de áudio o valor medido, painéis que informam a temperatura do alimento (HERSH, 2008).

A vida diária dos deficientes visuais dentro de suas casas pode ser bastante facilitada através de diversos dispositivos tecnológicos desenvolvidos para eles, mas além disso eles

também sofrem um grande problema que também necessita de recursos extras para melhorar a sua qualidade de vida, que é a questão de mobilidade no mundo exterior, isto é, em ambientes desconhecidos.

2.1.3 Mobilidade e Orientação de pessoas com deficiência visual

Para as pessoas com baixa visão ou cegueira, a mobilidade e a orientação são grandes desafios e para isso utilizam muitos métodos para se orientarem no ambiente e se moverem com segurança que vão desde os seus próprios sentidos, como audição, olfato como dispositivos criados para auxiliá-los nessas tarefas. A mobilidade está relacionada a capacidade de mover-se com facilidade e segurança, já a orientação diz respeito a percepção que o indivíduo tem do ambiente e da sua posição nele. Desse modo, a mobilidade é uma capacidade inata, mas a orientação é uma habilidade que deve ser aprendida como um relacionamento entre o indivíduo e o ambiente. Isso significa que os deficientes visuais devem adquirir o sentido de orientação através de outros meios, sejam eles auditivos ou táteis. Já a mobilidade é alcançada através de sistemas de treinamento que podem envolver mecanismos mecânicos, eletrônicos, como por exemplo bengalas, radares etc (BUENO, 1992).

Segundo Hersh (2008), os ambientes urbanos, em sua maioria, não foram projetados usando princípios modernos de acessibilidade e com isso muitas pessoas com deficiência visual tem grandes dificuldades em circular por esses ambientes e muitos não se sentem capazes de sair de suas casas sem um guia, tornando-se dependentes de outros para realizar tarefas fora de suas residências. Com isso, o desenvolvimento de dispositivos que possam auxiliar o passeio independente dessas pessoas é uma área de aplicação importante para a tecnologia assistiva.

A navegação de pessoas com deficiência visual levanta questões sobre a orientação, detecção e desvio de objetos ou obstáculos. O avanço da tecnologia permitiu a implementação de vários equipamentos que auxiliam a mobilidade e a maioria ainda não substitui o uso dos tradicionais cães guia e a bengala longa, mas são úteis para a tomada de decisões em situações incomuns (HERSH, 2008).

2.2 VISÃO COMPUTACIONAL

Segundo Dawson-Howe (2014), visão computacional é a análise automática de imagens e vídeos por computadores a fim de obter algum entendimento do mundo. Ela é inspirada no sistema de visão humana e no seu surgimento pensou-se ser um problema relativamente simples

de resolver, isso ocorreu devido ao nosso sistema visual nos fazer acreditar que a tarefa é intuitiva. Mas na verdade o sistema visual humano é muito complexo e o quanto o cérebro está envolvido nessa tarefa varia de 25% a 50%, segundo algumas estimativas.

Diniz (2012, p.11), define visão computacional como “[...] o conjunto de técnicas e métodos pelos quais é possível interpretar, extrair informações e tomar decisões sobre objetos físicos baseadas nas imagens destes objetos. Ou seja, é a ciência e a tecnologia que permite que máquinas vejam”.

Visão computacional é uma tarefa bastante complexa, pois tentamos a partir de uma ou mais imagens descrever o mundo que nós vemos e reconstruir suas propriedades, tais como forma, iluminação, cores e para isso temos que normalmente recorrer a modelos matemáticos, modelos probabilísticos, inteligência artificial entre outras técnicas para extrairmos informações relevantes. É surpreendente que os seres humanos e animais fazem isso tão facilmente, enquanto os algoritmos são tão propensos a erros (SZELISKI, 2010).

De acordo Serrano et al., (2004), o ser humano capta a luz através dos olhos, e que esta informação flui através do nervo óptico até o cérebro onde é processada. Há indícios que o primeiro passo desta transformação é encontrar elementos mais simples que decompõem a imagem (como segmentos e arcos). Em seguida, o cérebro interpreta a cena e finalmente age de acordo com o que foi analisado. A visão computacional, em uma tentativa de reproduzir este comportamento, tradicionalmente definiu quatro fases principais. Estas fases não são necessariamente sequenciais e dependendo do problema podem ser realizadas mais de uma vez:

- **Aquisição:** consiste em capturar a imagem através de algum tipo de sensor.
- **Pré-processamento:** consiste do processamento de imagem digital, a fim de facilitar as etapas posteriores. Neste estágio, se necessário, são utilizados filtros e transformações geométricas, para eliminar partes indesejáveis ou ressaltar partes que interessam.
- **Segmentação:** consiste em isolar elementos de uma cena de interesse.
- **Classificação:** consiste em distinguir os objetos segmentados a partir da análise de suas características.

Segundo Jahne (2000), dado a complexidade da visão computacional é útil dividi-la e tratá-la em componentes ou módulos de função, sendo que os principais são listados a seguir:

- **Fonte de radiação:** Se nenhuma radiação é emitida da cena ou do objeto de interesse, nada pode ser observado e posteriormente processado. Assim, a iluminação apropriada é necessária para objetos que não são eles próprios radiantes.

- **Câmera/Sensor:** A câmera através da lente e dos sensores, absorve a radiação/luminosidade proveniente dos objetos e converte em sinais adequados para o posterior processamento. Esta câmera pode ser uma simples câmera ótica ou sistemas mais complexos como por exemplo os tomógrafos.
- **Unidade de processamento:** Ele processa os dados de entrada, geralmente de maior dimensão, extraindo recursos adequados que podem ser usados para medir propriedades de objetos. Outro componente importante é um sistema de memória para coletar e armazenar conhecimento sobre a cena, incluindo mecanismos para excluir coisas sem importância.
- **Atores:** Os atores reagem ao resultado da observação visual. Eles se tornam parte integrante do sistema de visão quando o sistema de visão está respondendo ativamente à observação, por exemplo, rastreando um objeto de interesse ou usando uma navegação orientada por visão.

2.3 MODELO DE CÂMERAS

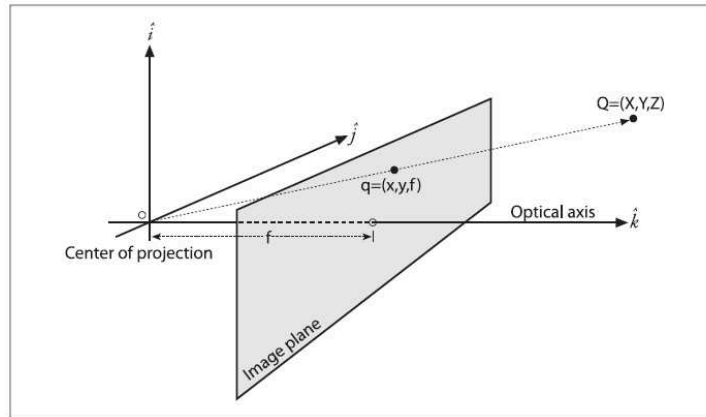
Para Zhang (2014), um dispositivo que captura a luz que provem de uma cena e a grava em forma de uma imagem, pode ser chamada de câmera. As câmeras possuem um modelo que descrevem matematicamente como as coordenadas 3D dos pontos de uma cena se relacionam com os pontos 2D da projeção dessa cena no plano de imagem. O modelo largamente utilizado em visão computacional é o modelo de câmera ideal conhecido como modelo *pinhole* ou modelo de perspectiva.

No modelo de perspectiva a luz proveniente da cena passa através do centro da câmera e é projetado no plano de projeção ou de imagem. Desse modo a imagem projetada no plano está sempre em foco e a relação entre o tamanho da imagem gerada e o objeto que a gerou é dado somente pelo parâmetro da câmera conhecido como distância focal (STURM, 2014; BRADSKI, 2016).

O modelo *pinhole* pode ser considerado muito preciso para a maioria das aplicações, mesmo sendo bastante simples, o grande problema dele é negligenciar muitos aspectos que câmeras reais possuem, principalmente os relacionados a distorções, tanto radiais como geométricas. Para resolver esse problema ele pode ser estendido, mas independentemente dessas limitações para muitas câmeras regulares o modelo pode ser considerado uma boa aproximação (STURM, 2014; SOLEM, 2012).

O modelo matemático de uma câmera pinhole pode ser visto na Figura 1. A distância f entre o centro de projeção e o plano de imagem é chamado distância focal. A linha que passa através do centro ótico é ortogonal ao plano da imagem e é chamado eixo ótico. O ponto de intersecção entre o eixo ótico e o plano de imagem é chamado ponto principal. A distância entre o objeto e o plano de imagem é dado pela coordenada Z (STURM, 2014).

Figura 1 – Representação de uma câmera pinhole



Fonte: BRADSKI, 2008, p. 372

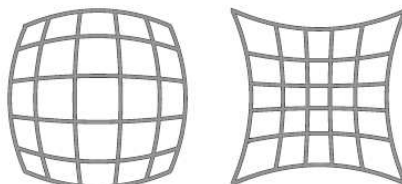
2.4 DISTORÇÕES DAS LENTES

Para o modelo de perspectiva ou *pinhole* as linhas retas no mundo resultam em linhas retas na imagem, conforme o modelo de projeção linear. Na teoria existem lentes que não introduzem nenhuma distorção na imagem gerada, mas na prática elas não são perfeitas. O principal motivo da introdução dessas distorções é causado pelo processo de fabricação das lentes, primeiro pelo custo por ser mais caro e difícil fabricar uma lente parabólica matematicamente ideal que uma lente esférica, segundo pelo alinhamento da lente na câmera que pode gerar distorções. As duas principais distorções encontradas são as tangenciais e radiais, onde as primeiras são causadas no processo de fabricação e as segundas pela forma da lente (BRADSKI, 2008; SZELISKI, 2010).

Segundo Dawson-Howe (2014), distorção radial é a distorção que é radialmente simétrica em que o nível de distorção está relacionada com a distância a partir do eixo óptico da câmera. Essa distorção é uma mudança na ampliação, caso ela diminua a medida que a distância do centro óptico aumenta, o efeito é chamado de distorção em barril, mas se ela aumentar então o efeito é chamado de distorção de almofada.

Na Figura 2, podemos ver os dois casos de distorção radial (barril e almofada). Já a distorção tangencial ocorre quando o plano de imagem não é perfeitamente paralelo com as lentes o que resulta em uma ampliação irregular, mas neste caso, a ampliação irá variar de um lado do plano de imagem para o outro (DAWSON-HOWE, 2014).

Figura 2 – Exemplo de distorção radial em um grid retangular.



Legenda: A imagem da esquerda mostra o efeito em forma de barril e a imagem da esquerda o efeito em forma de almofada.

Fonte: JAHNE, 2000, p 73

2.5 VISÃO ESTÉREO

O mundo é tridimensional e como uma imagem nada mais é do que uma projeção de uma cena sobre um plano perpendicular, as imagens são bidimensionais. Durante o processo de projeção de 3D para 2D uma das dimensões é perdida, em cada pixel da imagem somente um ponto da cena real é projetado, a informação de profundidade é matematicamente apagada. Uma importante tarefa da visão estéreo computacional é recuperar a terceira dimensão de uma ou múltiplas imagens (SHAH;1997, SANZ;2013).

As imagens formadas nos olhos são bidimensionais. Mesmo assim, somos capazes de perceber a profundidade das cenas. Esse mecanismo é importante, pois permite avaliar corretamente distâncias. Podemos, por exemplo, pegar objetos e desviar de obstáculos facilmente. Mesmo possuindo dois olhos, percebemos o mundo como parte de uma única imagem. No cérebro, as imagens provenientes dos dois olhos são fundidas em uma única imagem. Através da mudança de posição dos objetos percebidos por cada olho, o cérebro é capaz de recuperar a informação de distância dos objetos percebidos e enxergar em três dimensões. Esse processo recebe o nome de estereopsia (DINIZ, 2012, p.16).

A visão estéreo utiliza duas ou mais câmeras com campos de visão sobrepostas para estimar a estrutura 3D da cena a partir de projeções 2D. Dentre as inúmeras configurações de sistemas de visão estéreo o mais comum usa exatamente duas câmeras e é chamado de visão binocular. A visão binocular funciona, o sistema visual humano é a prova disso. Com isso o princípio básico da visão estéreo binocular é extremamente similar e inspirado na maneira em que os olhos humanos fazem. É observado uma cena de dois pontos de vista diferentes e

utilizando o princípio de triangulação e a informação de profundidade é recuperada a partir das disparidades (KLETTE,2014; LU et al,2011; HOGUE,2014).

2.5.1 Calibração estéreo

Um sistema estéreo perfeitamente alinhado e que as câmeras possuam exatamente as mesmas características é muito difícil de ser construído na prática, com isso é necessário um processo chamado calibração que determina a geometria interna e externa do sistema de câmeras, isto é, os parâmetros intrínsecos e extrínsecos. Através desses parâmetros é possível mapear entre os sistemas de coordenadas da imagem, da câmera e do mundo (STIVANELLO et al, 2010). Calibração estéreo nada mais é que o processo de calcular os relacionamentos geométricos no espaço entre as duas câmeras (BRADSKI, G; KAEHLER, A. 2008).

Os parâmetros intrínsecos ou internos são específicos de cada câmera, tais como: tamanho focal, dimensões do sensor, aspectos de ratio do sensor (altura por largura), parâmetros de distorção radial, coordenadas do ponto principal, fator de escala. Os parâmetros extrínsecos são aqueles aplicados para identificar a localização e direção das câmeras no sistema de coordenadas do mundo. Estimativas precisas destas geometrias são necessárias a fim de relacionar a informação da imagem, expresso em pixels, para um sistema de coordenadas do mundo externo (BROWN et al, 2003; KETTLE, 2014).

Os métodos de calibração de câmera são responsáveis por estimar tais parâmetros. Os parâmetros intrínsecos estão intimamente ligados ao hardware utilizado, existindo diversos métodos para suas obtenções, não estando necessariamente ligados a visão estéreo (BEZERRA, 2007, p. 25).

Segundo Zhang (2014), o uso de padrões de calibração ou conjunto de marcadores é uma das mais confiáveis maneiras de estimar os parâmetros intrínsecos de uma câmera ou de um conjunto de câmeras. Um dos métodos mais utilizados foi proposto por Zhang e requer apenas que o conjunto de câmeras observem um padrão planar, isto é, um padrão 2D em diferentes orientações e posições, podendo serem movidas livremente tanto o padrão quanto o conjunto de câmeras. Recomenda-se o seguinte procedimento de calibração:

1. Imprimir um padrão 2D e fixar ele em uma superfície plana.
2. Capturar algumas imagens do plano do modelo sob diferentes orientações através do movimento do plano ou do Câmera.
3. Detectar os pontos característicos nas imagens.
4. Estimar os cinco parâmetros intrínsecos e todos os parâmetros extrínsecos usando a solução de forma fechada.

5. Estimar os coeficientes da distorção radial.
6. Refinar todos os parâmetros, incluindo parâmetros de distorção da lente, através da estimativa de máxima verossimilhança.

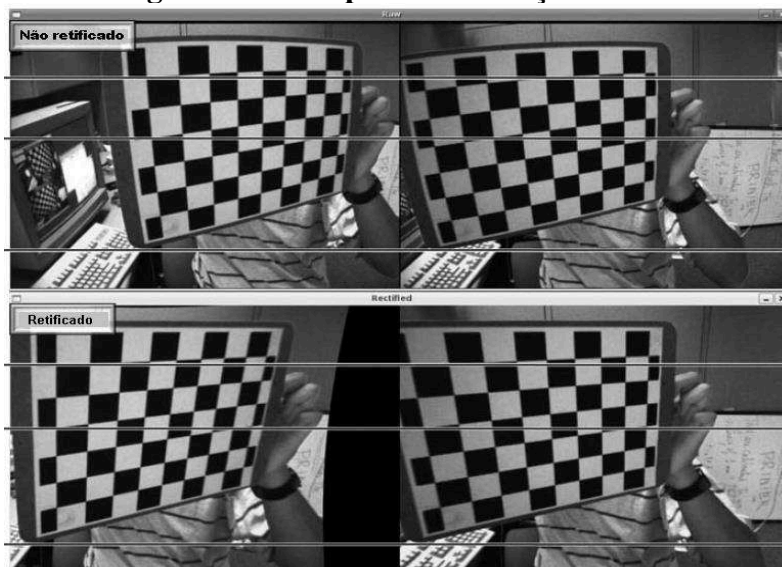
2.5.2 Retificação estéreo

De acordo com Kettle (2014), a complexidade da visão estéreo ocorre principalmente devido a importante tarefa de identificar os pontos correspondentes em um par de imagens, e que foram geradas por duas câmeras distintas. Para reduzir essa complexidade, é conveniente modificar as imagens de tal maneira que pareça que elas tenham sido capturadas por um par de câmeras idênticas ou o mais próximo disso. Esse processo é conhecido como retificação geométrica e é de extrema importância para os sistemas de visão estéreo.

Retificação estéreo é o processo de corrigir as imagens individuais para que elas pareçam ter sido tiradas por duas câmeras com plano de imagens alinhados, pois um alinhamento perfeito é muito difícil de ser realizado manualmente. O procedimento de retificação realiza todas as transformações e rotações necessárias nas imagens para adequá-las (BRADSKI, G; KAEHLER, A. 2008).

A Figura 3, mostra um exemplo de retificação estéreo e remoção de distorção radial onde as imagens superiores são os pares de imagens originais a esquerda e direita não retificadas e as inferiores são o par já retificado, é possível verificar que a distorção em forma de barril no topo do padrão xadrez nas imagens não retificadas foram corrigidas nas imagens retificadas.

Figura 3 – Exemplo de retificação estéreo



Fonte: BRADSKI, 2008, p 439. Adaptado Autor.

2.6 DISPARIDADE E CORRESPONDÊNCIA ESTÉREO

De acordo com Stivanello (2008), ao fazermos uso de um sistema de visão estéreo é possível recuperarmos a informação de profundidade dos pontos da imagem e que foram perdidas na projeção dos pontos no espaço 3D para a imagem em 2D, através da distância relativa desses mesmos pontos em diferentes imagens capturadas sob diversos pontos de vista. A essa distância relativa damos o nome de disparidade.

Segundo Szeliski (2012), correspondência estéreo é o processo de a partir de duas ou mais imagens estimar um modelo 3D de uma cena por encontrar *pixels* correspondentes nas imagens e converter suas posições 2D em profundidades 3D. Para Stivanello (2008), para que o processo de correspondência de pontos seja realizado duas questões devem ser levadas em consideração, a primeira é qual os elementos serão correspondidos e a segunda é qual a medida de similaridade será usada. A partir de quais pontos serão correspondidos os algoritmos podem ser classificados em métodos esparsos ou métodos densos.

Os métodos densos se diferem dos métodos esparsos pela quantidade de pontos que geram o mapa de disparidade. Para os densos todos ou quase todos os pontos da imagem possuem um valor de disparidade, já para os esparsos os valores de disparidade são gerados apenas para determinadas regiões selecionadas, tais como cantos ou bordas por exemplo. Apesar de os métodos esparsos serem mais rápidos eles são limitados devido a falta de informações relativas a profundidade fazendo com que os pontos faltantes tenham que ser interpolados, desse modo os métodos mais utilizados atualmente são os densos (CYGANNEK, 2011).

Os métodos podem ser classificados também como locais, globais e semi-globais. Os locais assumem que os pontos da imagem são envoltos por uma janela de pontos vizinhos onde a disparidade é a mesma e a correspondência dos pontos é realizada utilizando a correspondência destas janelas utilizando funções de custo baseadas em similaridades. Os métodos globais procuram propagar a informação de disparidade de um ponto para os seus vizinhos e empregam a minimização de uma função de energia sobre uma grande região da imagem. Já os métodos semi-globais realizam várias otimizações parciais em apenas uma dimensão afim de aproximar uma otimização global diferentemente dos métodos globais que realizam uma otimização global sobre toda a imagem (MENDES, 2012).

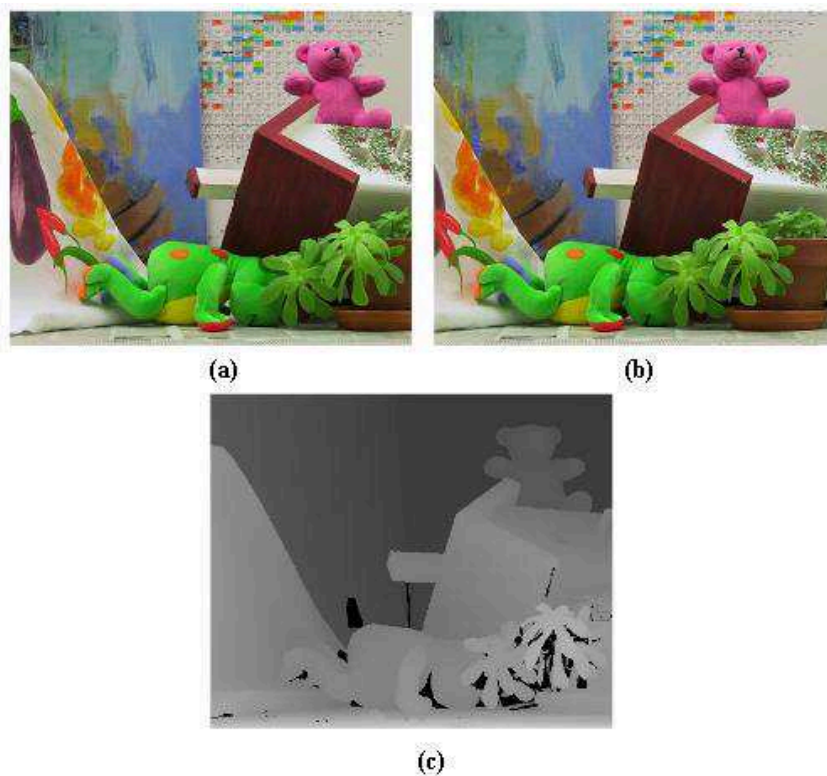
Esses algoritmos tomam como referência uma das imagens e para cada um dos pontos dela é realizada uma busca pelos pontos correspondentes na outra imagem. Os pontos que tiverem correspondentes têm sua distância relativa calculada e esse valor será armazenado como

um valor de intensidade em uma outra imagem e é chamado de mapa de disparidade (STIVANELLO, 2012).

O conjunto de todas as disparidades calculadas entre o par de imagens gerará o que se chama de mapa de disparidade. Só é possível determinar a disparidade se o ponto estiver presente em ambas as imagens (BEZERRA, 2007). Quando um ponto aparece em apenas uma imagem, dizemos que ocorreu uma oclusão. Um mapa de disparidade normalmente é representado como uma imagem em escala de cinza, onde a intensidade de cada pixel é proporcional a disparidade daquele *pixel* no par de imagens. Nessa representação os objetos que se encontram mais próximos da câmera aparecem mais claros, enquanto objetos mais distantes da câmera aparecem mais escuros. Um dos principais usos de mapas de disparidade é para sistemas que identificam e evitam obstáculos (SIEGWART, 2011).

A Figura 4, exibe um exemplo de mapa de disparidade gerado a partir de duas imagens obtidas da mesma cena por um sistema estéreo.

Figura 4 – Um exemplo de mapa de disparidade



Legenda: (a-b) par de imagens capturadas das câmeras esquerda e da câmera direita respectivamente.
(c) Mapa de disparidade

Fonte: SZELISKI (2010) Adaptado pelo Autor.

O processo de medir ou estimar distâncias de sensores de dados, que podem ser sistema de câmeras estéreo ou outros, tipicamente em um *array* 2D de dados é chamado estimacão de

profundidade e é uma das tarefas fundamentais em visão computacional. A partir de uma imagem 2D de uma cena 3D, o objetivo da estimativa de profundidade é recuperar, para cada pixel da imagem, a distância do centro da câmera ao ponto de cena 3D mais próximo ao longo da direção de visualização do pixel. A matriz resultante 2D dos valores de distância é chamada de mapa de profundidade, que está alinhado com o sistema de coordenadas da câmera. O desafio da estimativa de profundidade é recuperar, com suficiente precisão, o mapa de profundidade da imagem dada (KOCH, 2014).

Para Koch (2014), devido o mapa de profundidade ser representado em uma imagem 2D e cada pixel representa um valor de profundidade, esse modelo também é chamado de 2.5D. O mapa de profundidade não pode ser considerado uma representação 3D completa, pois contém apenas a profundidade para os elementos mais próximos que são visíveis da cena 3D e não considera regiões ocultas do ponto de vista da câmera.

Assumindo que nós temos um sistema de visão estéreo perfeitamente alinhado, sem distorções como na Figura 5: duas câmeras cujos planos de imagens são exatamente coplanares um com o outro, com eixos ópticos paralelos a uma distância conhecida e com os tamanhos focais iguais, isto é $f_l = f_r$. Assumimos também que os pontos \mathbf{c}_l esquerdo e \mathbf{c}_r direito estejam calibrados e tenham as mesmas coordenadas de pixels em suas respectivas imagens (BRADSKI, 2008). Utilizando um processo chamado de triangulação as disparidades são convertidas em diferentes escalas de profundidade que representam a estrutura da cena em 3D, através da formula:

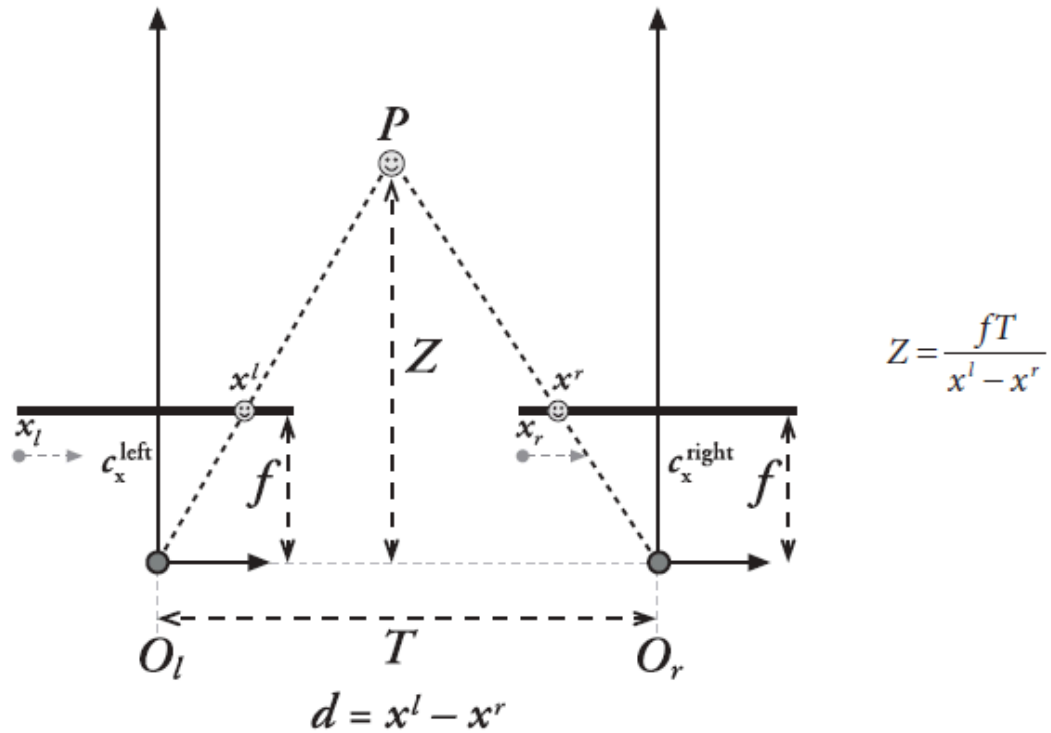
$$Z = \frac{fT}{x_l - x_r}$$

onde a profundidade Z é dada distância focal f multiplicado pela distância entre os centros óticos das câmeras que na equação é dado por T e divididos pela disparidade que é dado pela diferença entre os pontos x_l e x_r .

Analisando a fórmula utilizada para calcular a profundidade podemos verificar que ela é inversamente proporcional à disparidade e existe um relacionamento não linear entre estes dois termos. Quando o valor da disparidade é próxima de 0 pequenas diferenças na disparidade causam grandes diferenças na profundidade, mas quando a disparidade tem um grande valor, pequenas diferenças de disparidades não causam grandes diferenças de profundidade. A consequência disso é que sistemas de visão computacional estéreo tem uma alta resolução de

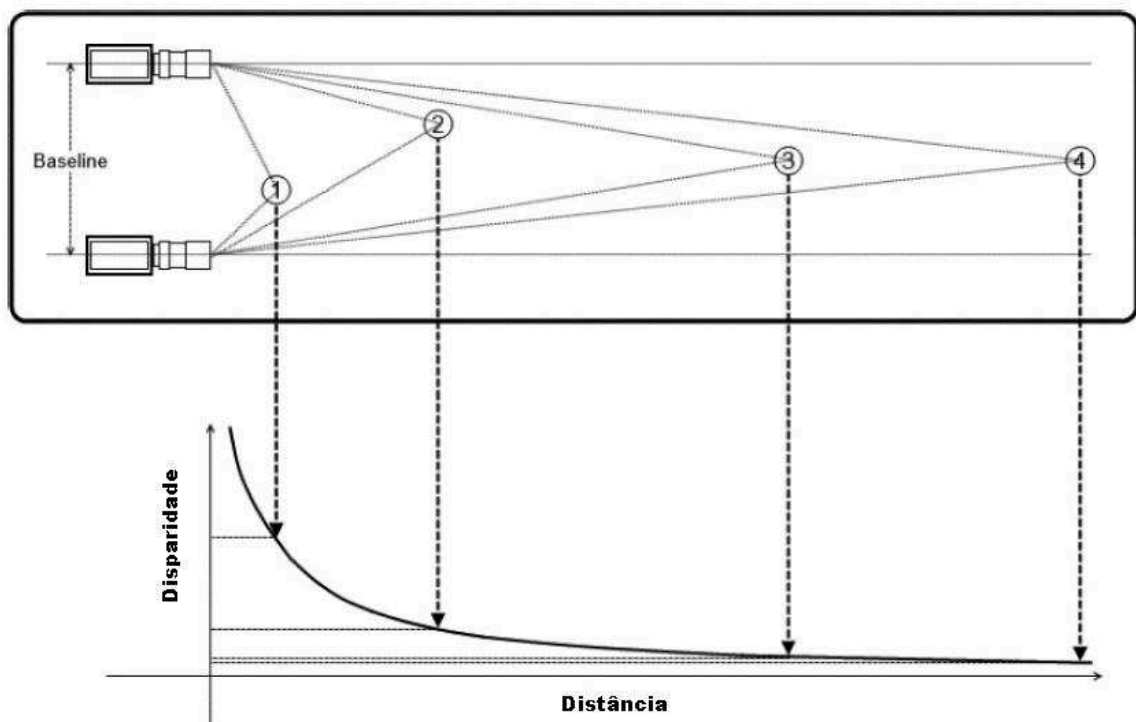
profundidade apenas para objetos próximos das câmeras (BRADSKI, 2016). A Figura 6 ilustra o que foi dito anteriormente.

Figura 5 – Triangulação em sistema estéreo



Fonte: BRADSKI (2008) Adaptado Autor.

Figura 6 – Relação inversamente proporcional da profundidade de disparidade



Fonte: BRADSKI (2016) Adaptado Autor.

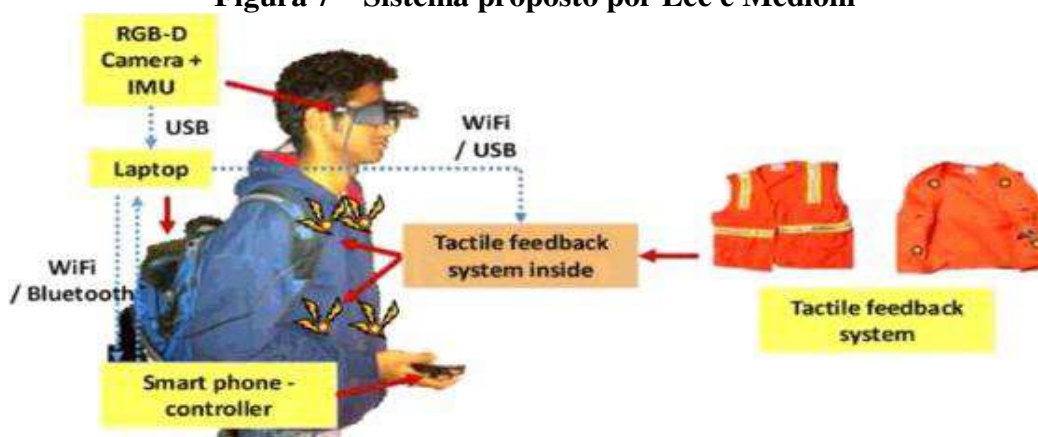
2.7 TRABALHOS RELACIONADOS

Esta sessão apresentará alguns trabalhos relacionados com o assunto do projeto desenvolvido. A busca desses trabalhos foi realizada em duas das principais bases de dados científicas e tecnológicas: *Science Direct*, *IEEE Xplore Digital Library*. Os termos utilizados na busca foram: “*Assistive technology*”, “*visual impairment*”, “*stereo vision*”, “*wearable*” e as buscas foram realizadas a partir do ano 2012 até a data corrente (2017). As 3 bases retornaram juntas 45 trabalhos e a seleção posterior foi realizada manualmente pelo autor, buscando os mais relevantes para o trabalho proposto a partir de uma leitura rápida com destaque para o título, abstract e palavras chaves, com isso foram retirados alguns trabalhos que estavam fora do escopo abordado. Também foram utilizados no refinamento os termos secundários como “OpenCV”, “Arduíno”.

Esta busca foi bastante importante pois a leitura e análise de alguns trabalhos já realizados por outros pesquisadores serve como base para o desenvolvimento de trabalhos futuros e colabora no entendimento deste projeto. Segue um breve resumo dos trabalhos encontrados.

O trabalho de Lee (LEE; MEDIONI, 2016), apresenta um sistema de navegação interior baseado em câmeras RGB-D para pessoas com deficiência visual. O sistema é composto de um smartphone utilizado como interface com o usuário, uma câmera RGB-D utilizada na altura da cabeça, software de navegação e um colete para as respostas sensoriais, além de um notebook que contém o software. A câmera se comunica com o notebook através de Bluetooth. O colete possui 4 motores vibratórios para guiar o usuário ao longo do caminho computado. Podemos ver o sistema na Figura 7 abaixo:

Figura 7 – Sistema proposto por Lee e Medioni



Fonte: Lee e Medioni, 2016. p 6

O projeto de Lee se assemelha ao trabalho proposto em sua estrutura física, pois ambos utilizam o mesmo conceito e componentes: utilizam câmeras para a visão computacional, também utilizam dispositivos vestíveis com respostas táteis para ajudar na navegação do usuário. Mas as semelhanças terminam aí, pois o trabalho de Lee utiliza uma câmera RGB-D que gera automaticamente o mapa de profundidade (letra D no nome da câmera) que facilita muito o trabalho, mas possui um custo que extrapola a ideia de ferramenta de baixo custo, além disso ele tem o objetivo de navegação e utiliza diversas técnicas para isso, o trabalho proposto tem apenas o intuito de ser um identificador de obstáculos.

Costa (2016), em seu trabalho propõem um algoritmo de detecção de colisão de objetos que será utilizado no módulo de visão computacional que será integrado em um protótipo desenvolvido pela mesma equipe. O algoritmo foi desenvolvido para fornecer uma navegação segura ao longo de uma rota pré-definida na presença de obstáculos. Ele utiliza um sistema de imagens estéreo pré calibrado e a partir dele obtém mapas de disparidade e calcula informações de profundidade buscando identificar objetos e informar ao usar da detecção.

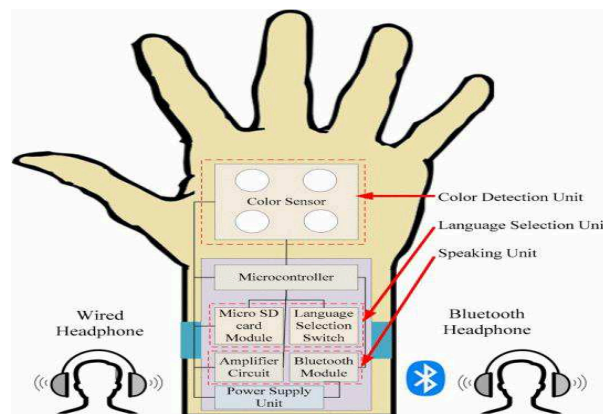
Apesar do trabalho de Costa ser o desenvolvimento de um algoritmo o artigo mostra a utilização de conceitos que também foram utilizados no projeto proposto, que é a utilização de mapas de disparidade e mapas de profundidade para a detecção de obstáculos. Costa propõe um novo algoritmo para isso, o nosso trabalho utiliza algoritmos da OpenCV para a geração dos mapas e propõe um método simples de detectar obstáculos e suas distâncias a partir desses mapas.

O trabalho de Mascetti (2016), apresenta duas propostas de utilização de sonificação para guiar pessoas com deficiência visual, ele utiliza como base um sistema de identificação de faixa de pedestres e a ideia é desenvolver um módulo que informa ao usuário a existência de uma faixa. O projeto de Mascetti não é diretamente similar ao trabalho proposto, mas ele possui algo também explorado nesse projeto que é a ideia de utilizar outros canais sensoriais para avisar ao usuário de determinada situação, ambos utilizam um sistema de visão computacional para identificar obstáculos ou faixa de pedestres e utilização um módulo sensorial para informar ao usuário, no nosso caso utilizamos o tato e ele utiliza a audição. Como nós apenas detectamos o obstáculo, mas não o identificamos (não dizemos o que é) achamos que o melhor sentido para isso é o tato.

O trabalho proposto por RASHID (2016) é um dispositivo capaz de detectar as cores e transformar os nomes das cores em som para que as pessoas com deficiência visual possam reconhecê-las. Ele será implementado e desenvolvido usando um Arduino Nano como microcontrolador e outros dispositivos eletrônicos como sensores de cor, módulo Bluetooth. A

imagem abaixo esboça o projeto. Este trabalho não utiliza visão computacional, mas de modo bastante interessante ele resolve um problema comum aos deficientes visuais que é identificação de cores, do mesmo modo que o trabalho proposto ele desenvolve um dispositivo vestível e utiliza o microcontrolador Arduíno. A Figura 8 exhibe o bloco funcional proposto por Rashid.

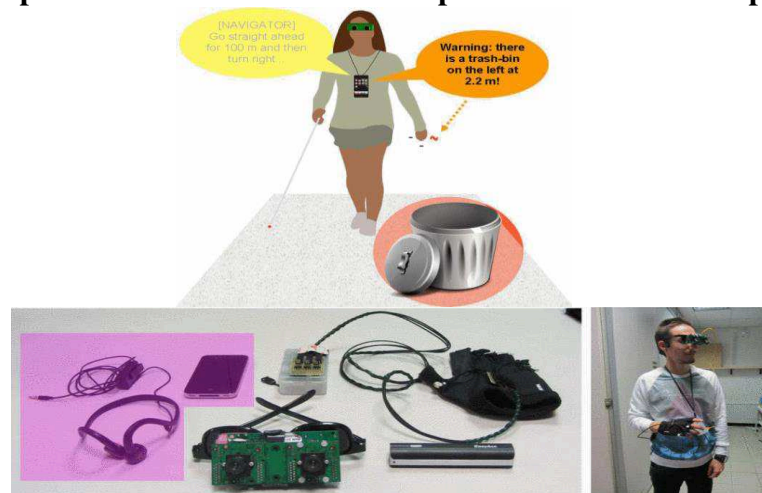
Figura 8 – Bloco funcional do projeto do RASHID.



Fonte: Rashid, 2016. p 2

Poggi e Mattoccia (2016), propõem um sistema vestível de ajuda a mobilidade a pessoas com deficiência visual e ele é baseado em visão computacional e técnicas de aprendizado de máquina. É um sistema bastante completo que oferece além da detecção de obstáculos também a identificação deles. Ele usa uma câmera RGB-D e informa o usuário através de som ou informações táteis. A Figura 9 exibem um exemplo de funcionamento e os componentes utilizados.

Figura 9 – Exemplo de funcionamento e os componentes utilizados no projeto do Poggi



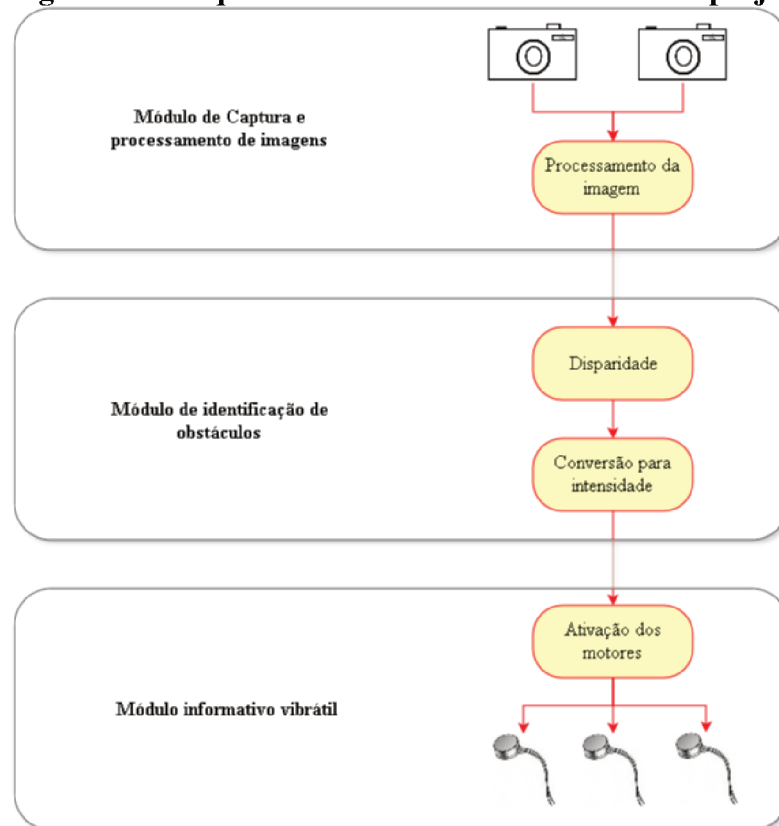
Fonte: Poggi e Mattoccia, 2016. p 1 e p 2

Assim como o projeto proposto ele utiliza informações táteis para passar para o usuário informações relativas a obstáculos, mas ele também utiliza o som, pois diferente desse trabalho ele também se propõe a identificar os objetos, conforme podemos ver na Figura 9. Ele é mais um dos projetos que utiliza câmeras RGB-D, mas como esse trabalho tem como um dos objetivos ser de baixo custo esse tipo de câmera não é viável, mas realmente seria de extrema importância, pois contornaria diversos problemas encontrados na geração de mapa de disparidade.

3 PROTÓTIPO

O principal objetivo desse trabalho é construir um protótipo de uma ferramenta de tecnologia assistiva que utilize visão computacional para auxiliar a mobilidade de pessoas com deficiência visual. O protótipo é constituído de *hardware* e *software* e os módulos que foram desenvolvidos são exibidos na Figura 10.

Figura 10 – Esquema funcional e informacional do projeto



Fonte: Autor

O sistema desenvolvido trabalha da seguinte maneira:

- Duas imagens são capturadas através do modo de visão estéreo e para isso é utilizado um par de câmeras.
- As imagens originais sofrem alguns processamentos como redução na resolução, são convertidas do formato RGB para tons de cinza. Esses processamentos são para simplificar e otimizar os processamentos de visão computacional posteriores.
- É extraído o mapa de disparidade do par de imagens capturadas e a saída é uma imagem em tons de cinza que possuem informações relativas a distâncias e profundidades.

- Essas informações são extraídas e convertidas em intensidades de vibrações para serem repassadas aos motores.
- O módulo final envia a informação vibrátil ao usuário.

3.1 RECURSOS UTILIZADOS PARA O DESENVOLVIMENTO DO PROJETO

A seguir são apresentados alguns recursos importantes que foram utilizados no projeto, os que não estiverem listados nessa Seção estão contidos em seções próprias dentro dos respectivos módulos.

3.1.1 Arduíno

Arduíno é uma plataforma de computação física *open source* baseada em uma placa microcontroladora e um ambiente de desenvolvimento que implementa a linguagem *Processing*. Arduíno pode ser usado para desenvolver os mais diversos tipos de projetos que podem rodar usando somente a placa ou mesmo ligado a outros dispositivos. É programável via porta USB. Ele é composto de duas partes principais: a placa, que é o hardware onde os programas criados são executados e a IDE onde são criados os sketches (programas criados para o Arduíno) e que dizem o que a placa deve fazer (MASSIMO, 2011).

A plataforma nasceu no Ivrea Interaction Design Institute como uma ferramenta para prototipagem rápida e fácil, voltada para estudantes sem formação em eletrônica e programação. Ao longo dos anos o Arduíno tem sido utilizado em milhares de diferentes projetos e aplicações ao redor do mundo, que vão desde a construção de instrumentos de baixo custo para provar princípios físicos e químicos até sistemas robóticos. Existem diversas plataformas semelhantes ao Arduíno, mas ele oferece algumas vantagens sobre essas outras opções:

- **Custo:** as placas do Arduíno são relativamente mais baratas se comparadas a outras plataformas de microcontroladores. Além de poderem ser construídas manualmente, pois o projeto é aberto.
- **Cross-plataform:** A IDE do Arduíno roda em diversos sistemas operacionais, tais como sistemas Linux, Windows e Mac OS. A maioria dos sistemas de microcontroladores são limitados ao sistema operacional Windows.

- Ambiente de programação simples e claro – A IDE do Arduino é fácil de usar para quem é iniciante, mas ao mesmo tempo é flexível o suficiente para que usuários avançados possam aproveitar os benefícios da plataforma.
- *Software* de fonte aberta e extensível - O software do Arduino é publicado como ferramentas de código aberto e são disponíveis para a extensão por programadores experientes. A linguagem pode ser expandida através de bibliotecas C++ e quem quiser entender os detalhes técnicos podem acessar diretamente a linguagem de programação AVR-C em que é baseada. Da mesma forma, você pode adicionar código AVR-C diretamente em seus programas Arduino.
- *Open source e hardware* extensível - Os planos das placas Arduino são publicados sob uma licença *Creative Commons*, para que designers de circuitos experientes possam fazer sua própria versão do módulo, estendê-la e melhorá-la. Mesmo os usuários relativamente inexperientes podem construir a versão do módulo, a fim de entender como ele funciona e economizar dinheiro (ARDUÍNO, [201-?]).

Existem diversos modelos de plataformas do Arduino que atendem aos mais diversos propósitos, tais como: internet das coisas, impressão 3D, vestíveis, de uso geral. O que diferencia a maioria das placas são as especificações tais como: microprocessador, velocidade da CPU, número de entradas digitais e analógicas, quantidade de memória entre outras (ARDUÍNO, [201-?]).

Nesse trabalho, o modelo do Arduino que será utilizado é o UNO e ele será integrante do módulo informativo vibrátil e será responsável por receber os sinais enviados pelo notebook, através de uma interface USB, fazer o processamento desses sinais e ativar os motores correspondentes no grid de motores com as respectivas intensidades.

3.1.2 Ambiente de desenvolvimento e bibliotecas

No desenvolvimento desse trabalho optou-se por usar *softwares open source*. Foram utilizadas duas IDEs para o desenvolvimento dos módulos, o ambiente utilizado para desenvolver o módulo de captura e processamento de imagens e o módulo de identificação de obstáculos foi o Eclipse CDT e a linguagem utilizada foi o C/C++. Já para o módulo informativo vibrátil, que fará o controle dos motores e será executado no Arduino foi utilizada a própria IDE da plataforma e a linguagem utilizada foi o C/C++. A biblioteca de visão computacional utilizada foi a OpenCV e será explicada na Seção 3.1.3.

O Quadro 1 lista os principais artefatos (IDEs, compiladores, bibliotecas) utilizados e suas respectivas versões.

Quadro 1 – Artefatos e versões utilizadas.

Artefato	Versão
IDE: Eclipse	Neon.2 Release (4.6.2)
IDE: Arduíno	1.8.1
Biblioteca: OpenCV	3.2-dev
Compilador: gcc/g++	5.4.0

Fonte: Autor

O sistema operacional utilizado tanto para desenvolver os módulos quanto para executá-los no computador utilizado no protótipo é o Ubuntu 16.04.01 LTS 64 bits, com a versão do kernel 4.4.0.

3.1.3 OpenCV

OpenCV (*Open Source Computer Vision*) é uma biblioteca de visão computacional, escrita em C e C++ e que pode ser executada em diversos sistemas operacionais, tais como Linux, Windows, Android. Ela é liberada sobre licença BSD e, assim, é gratuito para uso acadêmico e comercial. Foi projetada com foco em aplicações em tempo real, com eficiência computacional e para aproveitar os benefícios de processadores multicore (BRADSKI; KAEHLER, 2008).

Segundo Bradski (2008), um dos principais objetivos da biblioteca OpenCV é fornecer uma infraestrutura de visão computacional de uso simples e que ajude a construir rapidamente aplicações de visão bastante sofisticadas. Ela oferece centenas de funções e algoritmos que abrangem as mais diversas áreas da visão computacional, incluindo calibração de câmera, visão estéreo, robótica, entre outras.

3.2 MÓDULO DE CAPTURA E PROCESSAMENTO DE IMAGENS

Esse módulo é composto de *hardware* e *software*. O *hardware* é constituído de duas câmeras fixadas em um óculos e conectadas a um notebook através de cabos USBs. O *software* é executado no notebook e ele é responsável por capturar as imagens e realizar os processamentos necessários, tais como redução da resolução e conversão da escala de cores das imagens.

A responsabilidade do módulo é capturar as imagens provenientes das câmeras fazer um processamento prévio sobre essas imagens que tem como objetivo principal reduzir e otimizar o processamento do módulo de identificação de obstáculos que usa as imagens como entrada para os seus algoritmos. A seguir são descritos de maneira mais aprofundada os componentes do módulo.

3.2.1 Câmeras

Devido a necessidade de as câmeras serem acopladas a um óculos, tentou-se utilizar inicialmente duas câmeras do tipo endoscópico que seriam conectadas ao computador via USB. A Figura 11, ilustra o modelo. Mas durante testes iniciais verificou-se que elas possuíam grande qualidade para objetos bem próximos, mas para objetos a uma distância maior que 1 metro a qualidade caía consideravelmente.

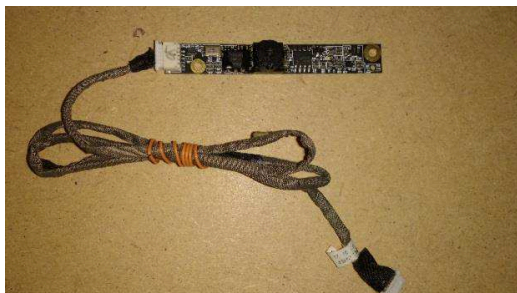
Com esse problema buscou-se alternativas dentre as opções encontrou-se que webcams integradas de notebook podem ser adaptadas para serem utilizadas via USB. Foram utilizadas duas câmeras do mesmo modelo, não é obrigatório que sejam do mesmo modelo, mas facilitam bastante a questão de calibração e retificação, e são câmeras utilizadas no notebook HP Pavilion DV2000. Não foram encontradas maiores especificações sobre elas além das que estão no Quadro 2. A Figura 12 mostra o modelo utilizado e a Figura 13 mostra as conexões realizadas entre o cabo da webcam e um cabo USB para poder ser utilizada na projeto.

Figura 11 – Primeiro modelo da câmera utilizado no protótipo

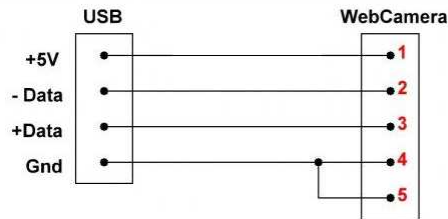


Fonte: DEALEXTREME

Figura 12 – Câmera interna de notebook utilizada



Fonte: Autor

Figura 13 – Conexões entre webcam e cabo USB

Fonte: Autor

Quadro 2 – Especificações da câmera

Característica:	Valor:
Resolução	640x480 (VGA) 352x288
View angle	66°
Distância focal	4 cm
Sistemas operacionais suportados	Windows/Linux
Conexão	USB 2.0/1.1

Fonte: Autor

3.2.2 Computador

O computador é responsável por todo o processamento das imagens adquiridas pelas câmeras, essas imagens são enviadas ao computador via USB. O computador utilizado foi um notebook com as seguintes especificações:

- Notebook HP Pavilion dv6.
- Processador: Intel Core I7 CPU Q720 1.6GHZ.
- Memória: 4 GB RAM.
- Placa de Vídeo: GeForce GT 230M (Memória: 1 GB RAM).

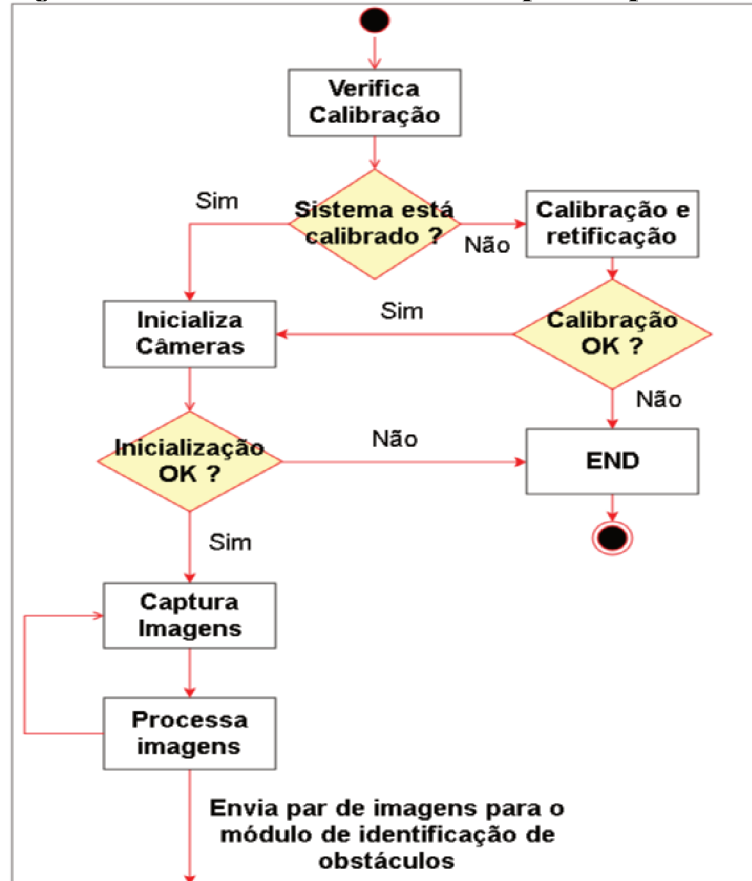
O sistema operacional utilizado é o Ubuntu 16.04.01 LTS 64 bits, com a versão do kernel 4.4.0.

3.2.3 Software

O módulo possui um fluxo bastante simples como exibido na Figura 14. Na inicialização do sistema é verificado se o sistema está calibrado, essa verificação é feita através da existência de um arquivo XML que contem as matrizes geradas pelo processo de calibração e retificação e que são salvas no fim do processo. Caso a calibração não tenha sido realizada ainda é inicializada a etapa de calibração e retificação e será explicada mais adiante. Se as câmeras já tiverem sido calibradas o sistema inicializa as câmeras e alguns dos seus atributos, como por

exemplo a quantidade de frames por segundo, FPS. Após a inicialização o módulo entra em loop capturando as imagens fazendo os processamentos necessários e repassando para o módulo de identificação de obstáculos. Nas seções a seguir essas etapas são descritas com mais detalhes.

Figura 14 – Fluxograma do software do módulo de captura e processamento de imagens



Fonte: Autor

3.2.3.1 Calibração e retificação

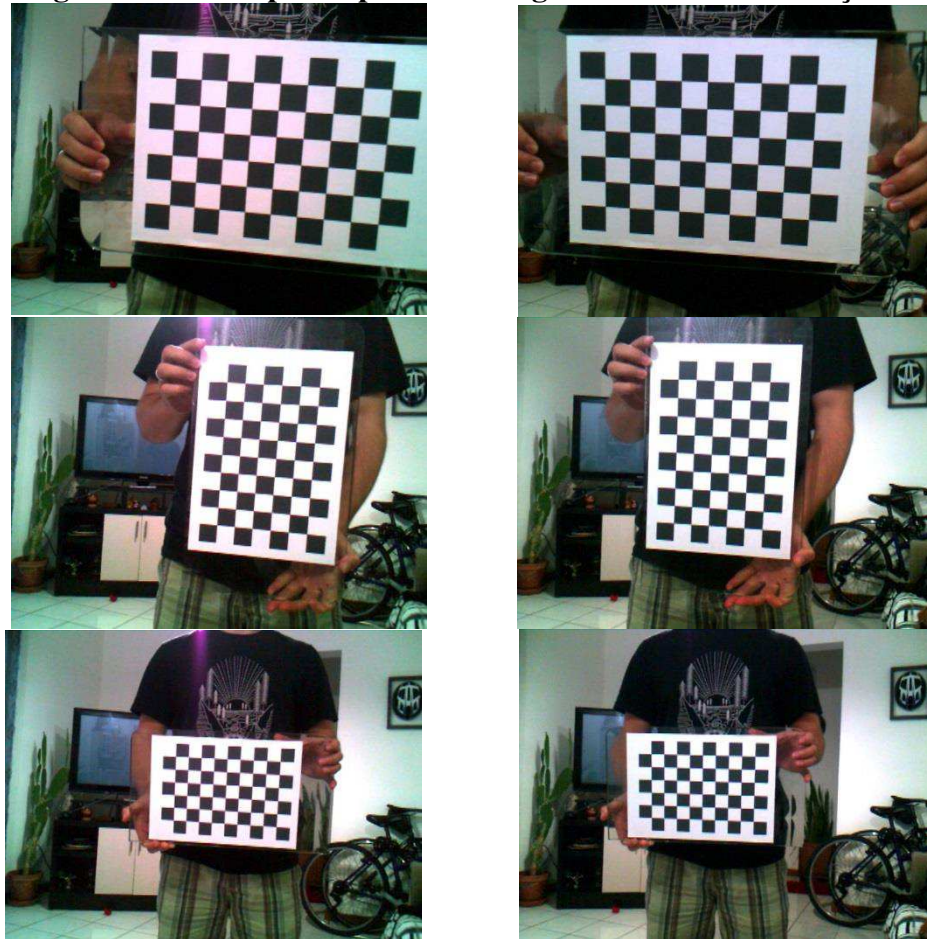
O processo de calibração e retificação é extremamente importante em sistemas de visão estéreo, pois é através dele que obtemos os valores dos parâmetros do par de câmeras do sistema, sejam eles os parâmetros intrínsecos ou extrínsecos e também removemos algumas distorções que as imagens possam conter. Esse processo poderia ser realizado apenas uma vez, caso pudéssemos garantir que as câmeras ficassem totalmente fixas e não fossem movidas. Se garantirmos essas condições, o processo de calibração e retificação deve ser realizado pelo menos uma vez na inicialização do sistema.

Ao inicializarmos o sistema, verificamos se ele já se encontra calibrado através da verificação da existência de um arquivo XML específico (`my_calib_params.xml`). Se esse arquivo existir então o sistema já foi calibrado previamente e o processo não é realizado

novamente, pois os parâmetros já estão salvos nesse arquivo XML, explicaremos o salvamento dessas informações mais adiante. Caso o arquivo não exista, então o processo de calibração é iniciado e será detalhado a seguir.

Para que o processo de calibração gere todos os parâmetros necessários, vamos utilizar um padrão planar bem definido e comumente chamado de tabuleiro de xadrez. Precisamos fornecer ao processo algumas imagens desse padrão e a tarefa é encontrar alguns pontos específicos nele (cantos quadrados no tabuleiro de xadrez), essas imagens possuem diferentes orientações e localizações do tabuleiro. Não existe um número fixo de imagens que devem ser utilizadas, segundo Bradski (2008), são necessários no mínimo 10 imagens para que o resultado da calibração seja satisfatório e que quanto maior o número menor o erro gerado no processo. Para a calibração das câmeras do projeto utilizamos um conjunto de 24 pares de imagens, a Figura 15 mostra alguns exemplos dessas imagens.

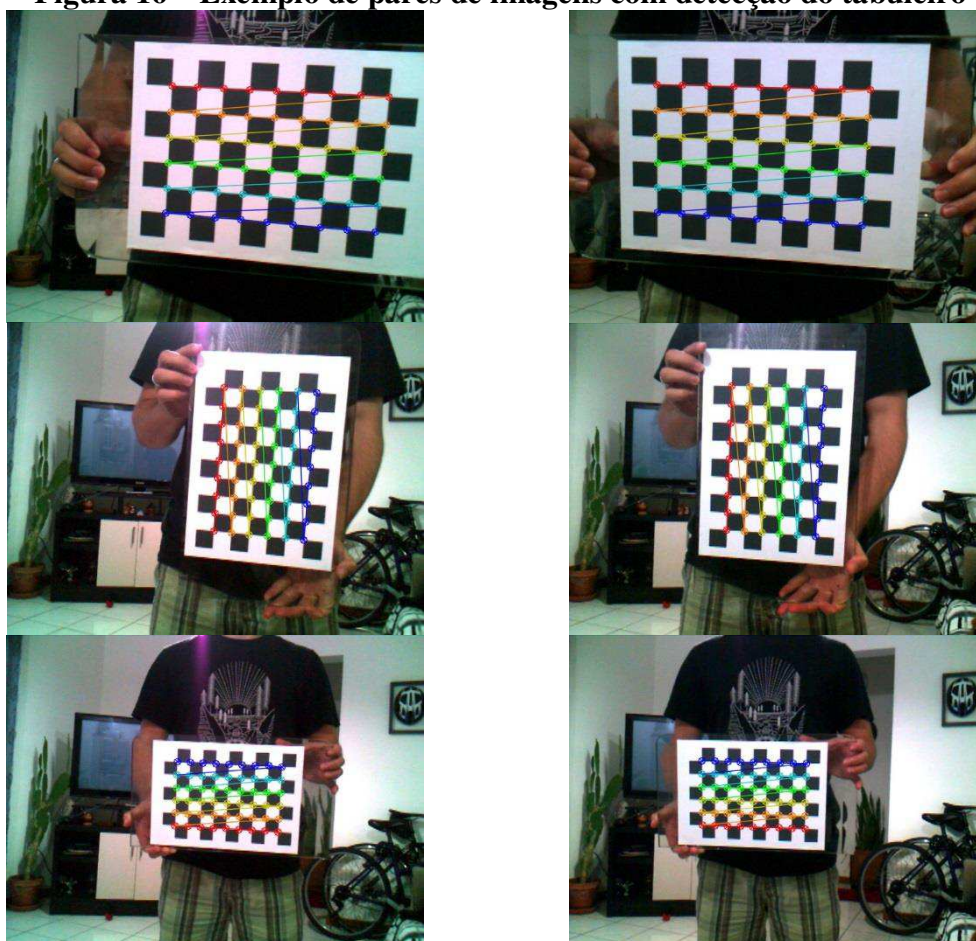
Figura 15 – Exemplo de pares de imagens usadas na calibração



Fonte: Autor

Para encontrarmos o padrão do tabuleiro de xadrez nas imagens nós utilizamos a função da OpenCV *findChessboardCorners()*, e é necessário informar a quantidade de quadrados horizontais e verticais que o tabuleiro possui, no projeto foi utilizado um tabuleiro 9x6 cada um dos quadrados possui o tamanho real de 5.7 cm. A função retorna true se o padrão é encontrado e também uma lista com as coordenadas dos pontos de cada um dos cantos. Esses pontos são armazenados na ordem da esquerda para a direita e de cima para baixo. A Figura 16, exhibe exemplos da detecção do padrão em pares de imagens.

Figura 16 – Exemplo de pares de imagens com detecção do tabuleiro



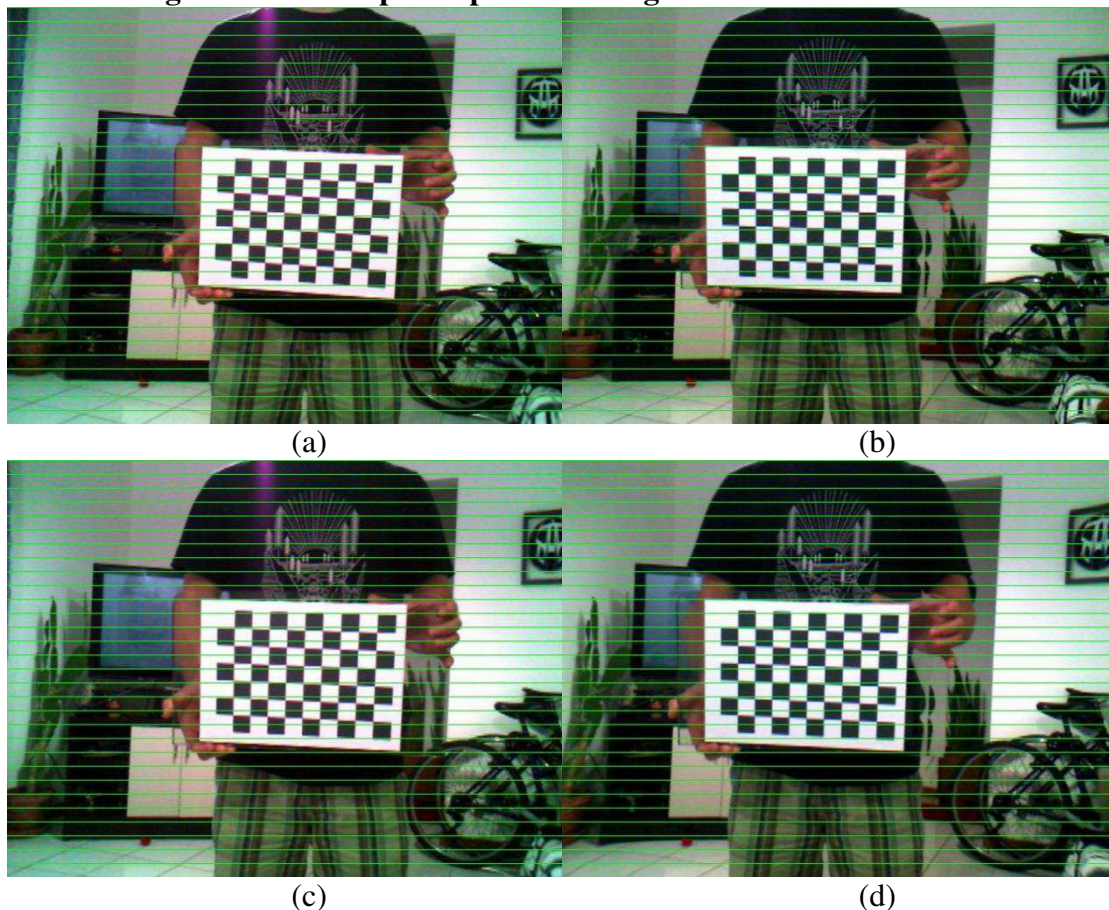
Fonte: Autor

Após encontrarmos os pontos necessários na imagem estamos prontos para o processo de calibração. Para isso nós utilizamos a função *stereoCalibrate()*, da OpenCV. Passamos para a função a lista de pontos encontradas anteriormente e a função nos retorna as matrizes dos parâmetros intrínsecos e extrínsecos de ambas as câmeras, assim como coeficientes de distorções e vetores de translação e rotação. O valor que a função retorna representa o erro

médio de reprojeção, este número nos dá uma boa estimativa da precisão dos parâmetros encontrados e ele deveria ser o mais próximo de zero possível.

Após calibrarmos precisamos fazer o processo de retificação, que é realizada através da função *stereoRectify()*. A função utiliza as matrizes computadas por *stereoCalibrate()* como parâmetros de entrada e como saída ela fornece duas matrizes de rotação e também duas matrizes de projeção nas novas coordenadas. A função computa as matrizes de rotação para cada uma das câmeras, conseqüentemente ela faz todas as linhas epipolares paralelas e assim simplifica o problema de correspondência estéreo. A Figura 17, exibe exemplos da retificação das imagens exibidas anteriormente nas Figuras 15 e 16.

Figura 17 – Exemplo de pares de imagens não retificadas e retificadas



Legenda: (a-b) par de imagens não retificadas esquerda e da câmera direita respectivamente.
(c-d) par de imagens retificadas esquerda e da câmera direita respectivamente.

Fonte: Autor

Após o processo de retificação é necessário criarmos os mapas de retificação, para cada uma das câmeras, que serão usados para mapear as imagens capturadas pelas câmeras em tempo real e que não são calibradas e retificadas para imagens calibradas e retificadas. A criação do mapa é feita pela função *initUndistortRectifyMap()*, esta tarefa é realizada uma única vez

durante a calibração, pois posteriormente esse mapa é salvo e disponível para o módulo sem precisarmos calibrar novamente. A função *remap()* é responsável por fazer o mapeamento entre a imagem capturada e o mapa de retificação gerado pela função *initUndistortRectifyMap()*, esta função é chamada para todo o frame capturado pelas câmeras e fica no looping de captura de imagens do sistema.

Após todo o processo de calibração todas as matrizes geradas para as duas câmeras são salvas em um arquivo XML, que é carregado na inicialização do sistema. **O APÊNDICE B** exibe um exemplo do arquivo salvo e explica resumidamente as matrizes salvas.

3.2.3.2 Inicialização das câmeras

A biblioteca OpenCV simplifica muito o uso *streaming* de vídeos em tempo real a partir de câmeras, para utilizarmos uma câmera só precisamos criar um objeto que a represente através da função *VideoCapture()*, onde é necessário passar o identificador do dispositivo de captura de vídeo aberto, isto é o índice da câmera. Após a criação da câmera é verificado se a captura de vídeo já foi inicializada, através da função *isOpened()* que retorna true caso tenha iniciado com sucesso e false caso não tenha conseguido, se uma das duas câmeras não puder ser inicializada o sistema não inicia. Na inicialização também alteramos a quantidade de FPS do sistema, o *default* é 30, passamos para 15 FPS por questões de desempenho.

3.2.3.3 Captura e processamento de imagens

Após termos inicializado as câmeras e termos recuperados os valores das matrizes necessárias para fazermos o mapeamento das imagens capturadas em tempo real descalibradas e não retificadas para calibradas e retificadas, precisamos efetivamente capturar os frames das câmeras e fazemos isso através da função *read()*, que captura, decodifica e retorna o próximo frame de vídeo disponível. São capturados os frames das duas câmeras simultaneamente, ou o mais perto disso, e logo após são realizados alguns processamentos simples com o intuito de melhorar a imagem corrigindo distorções, eliminando ruídos, para o processamento que será realizado posteriormente pelo módulo de identificação de obstáculos.

O primeiro processamento é transformar a imagem de colorida para escala de cinza, através da função *cvtColor()*, logo após realizamos uma equalização de histograma para termos um ajuste no contraste. Por último através da função *remap()* são aplicadas as matrizes de

transformação obtidas no processo de calibração e retificação para que as imagens sejam corrigidas das eventuais distorções.

No final as imagens (esquerda e direita) estão prontas para serem processadas pelo módulo de identificação de obstáculos e a partir delas ser gerado o mapa de disparidade e seu processamento para identificarmos a existência de algum obstáculo e sua distância do usuário.

3.3 MÓDULO DE IDENTIFICAÇÃO DE OBSTÁCULOS

Tendo em visto o maior desempenho possível neste módulo o módulo anterior nos entrega o par de imagens pré-processadas, com mínimo de distorções e ruídos possíveis e já retificadas. Esse módulo é composto apenas de software e a responsabilidade do módulo é identificar obstáculos a partir das imagens repassadas pelo módulo anterior, calcular a distância desses obstáculos, mapear essas distâncias para intensidades de vibração dos motores e repassar essa informação para o módulo informativo vibrátil. A seguir são descritos de maneira mais aprofundada os componentes do módulo. Todas essas tarefas são realizadas em etapas como exibidas na Figura 18 e são descritas nas próximas seções.



Fonte: Autor

3.3.1 Mapa de disparidade

Para a geração do mapa de disparidade optou-se pelo método semi-global, pois segundo Mendes (2012) os seus resultados possuem um bom balanço entre desempenho e custo computacional se comparados com os métodos locais e globais. Outro fator para a escolha foi a disponibilização de uma implementação de um método semi-global pela biblioteca OpenCV. A biblioteca usa uma versão modificada do algoritmo original de Hirschmuller. Diferentemente

do original que considera 8 direções esta implementação considera somente 5 direções. A implementação da OpenCV faz correspondência de blocos não de pixels individuais, por isso o nome, *semi-global blocking match* (SGBM). É possível termos o comportamento *default* do algoritmo de Hirschmuller na OpenCV basta ajustar os parâmetros necessários, mas isso torna o algoritmo mais lento.

A entrada do algoritmo que gera o mapa de disparidade é um par de imagens retificadas e a saída é uma imagem em tons de cinza contendo os valores das disparidades. A função contém diversos parâmetros que influenciam diretamente na qualidade do mapa gerada, mas também no seu desempenho. Alguns dos principais parâmetros são:

- **minDisparity**: Mínimo valor possível da disparidade. Normalmente é zero, mas pode até mesmo ser um valor negativo.
- **numDisparities**: Valor máximo menos o valor mínimo da disparidade. Este valor é sempre maior que zero. Na atual implementação o valor precisa ser múltiplo de 16.
- **blockSize**: Tamanho do bloco de pixels usado para a busca de similaridades. Pequenos blocos geralmente contêm mais detalhes, mas acabam contendo muito ruído, já blocos maiores são mais suaves, mas menos detalhados. Este parâmetro deve ser um número ímpar para que o bloco contenha um centro.
- **P1**: Penalidade aplicada para pequenas variações de disparidade.
- **P2**: Penalidade aplicada para grandes variações de disparidade.
- **disp12MaxDiff**: O valor de disparidade é aceito somente se a distância da primeira correspondência e a distância da segunda correspondência tiverem a diferença máxima de **disp12MaxDiff**.
- **mode**: define como o algoritmo será executado, se setado para **MODE_HH** ele executará de modo similar ao original.
- **uniquenessRatio**: percentual que a disparidade calculada pela função custo deveria ser melhor que o segundo melhor valor.

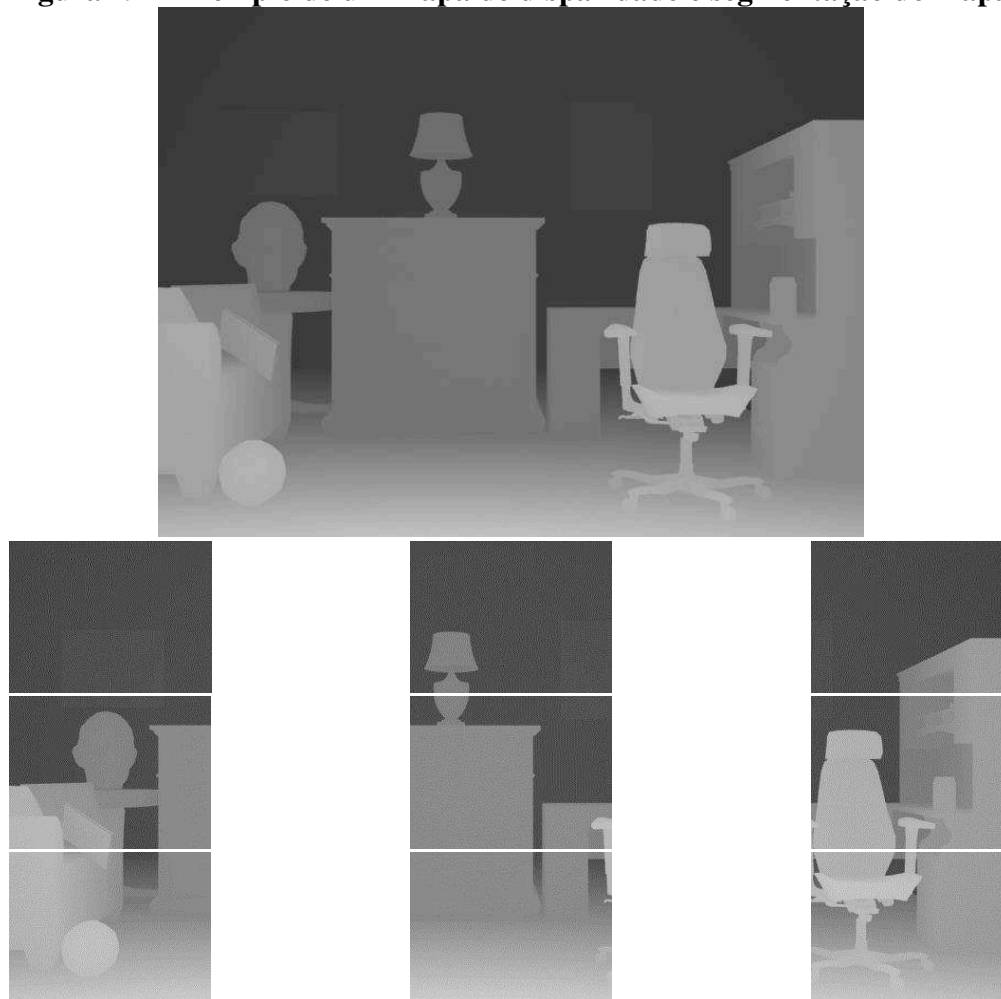
Estes parâmetros são passados na construção do objeto *sgbm* a partir da função da OpenCV *StereoSGBM::create()* e isto é realizado na inicialização do módulo através da função *init_detect_module()*. Para gerarmos o mapa de disparidade usamos a função da OpenCV *compute()* passando o par de imagens e o retorno é o mapa de disparidade do mesmo tamanho das imagens de entrada.

3.3.2 Mapeamento da disparidade para Intensidade

A partir do mapa de disparidade nós fazemos a conversão da disparidade em intensidade de vibração dos motores, mas para chegar a isso precisamos de vários passos: segmentar o mapa de disparidade, isto é, dividir a imagem em regiões de interesse para podermos identificar se naquela determinada região temos algum objeto.

A segmentação do mapa de disparidade nada mais é do que dividirmos a imagem em regiões de interesse, no nosso caso dividimos em 9 regiões, conforme o exemplo da Figura 19. A função implementada que realiza essa tarefa é a *segment_disp_map()* que recebe o mapa de disparidade e o divide em 9 regiões retangulares de igual tamanho e geram 9 novas imagens, onde cada uma delas é 1/3 da altura original e 1/3 da largura original, como estamos usando imagens de resolução 640 por 480, cada uma das imagens é 213 por 160 pixels.

Figura 19 – Exemplo de um mapa de disparidade e segmentação do mapa.



Legenda: A imagem superior exibe um mapa de disparidade e as imagens abaixo mostra o mapa de disparidade segmentado em 9 regiões de interesse.

Fonte: Zucheuil (2014) adaptado Autor

Após a segmentação do mapa de disparidade é utilizado o cálculo de histogramas sobre cada uma das 9 regiões segmentadas para identificarmos a intensidade predominante na imagem. Pois dado uma imagem que é totalmente preta, podemos inferir que não há nenhum obstáculo próximo, já em uma imagem totalmente branca representa um obstáculo bem próximo do usuário. Criamos então dois histogramas, o primeiro é menos refinado e representa apenas 3 escalas proximidades: curta, média ou longa. Já o segundo é um refinamento, temos entre cada uma das escalas temos mais 3, totalizando 9 escalas.

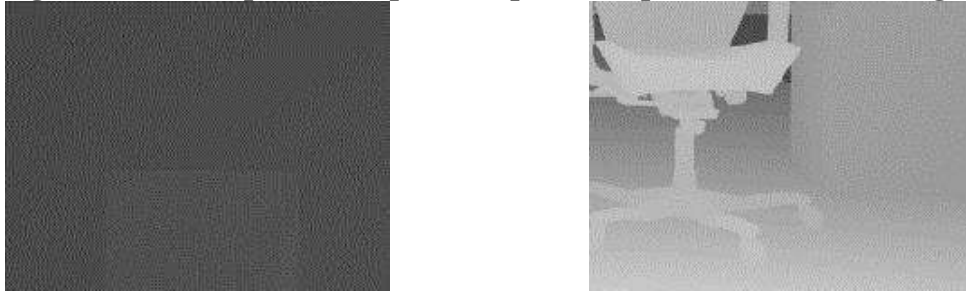
Para o cálculo dos histogramas são definidas algumas variáveis importantes, a primeira é o *range* que representa o intervalo dos valores que podem ser assumidos na imagem, neste caso como as imagens são mapas de disparidades em escala de cinza os valores do range são [0,255], onde 0 é a cor totalmente preta e 255 a cor totalmente branca. Outra variável importante é o *bin* que representa o número de subdivisões que o *range* vai ser dividido.

Nós utilizamos dois *bins* um com o valor de 3 e o outro com o valor 9, no primeiro valor o *range* será agrupado em 3 subdivisões e no segundo valor em 9 subdivisões, isto é, para o valor 3 o range será subdividido da seguinte maneira: [0-85], [86-170] e [171-255] e para o valor 9 o range será subdividido da seguinte maneira: [0-28], [29-56], [57-84], [85-112], [113-140], [141-168], [169-198], [199-227] e [228-255]. Com isso a função *calcHist()* é chamada duas vezes, uma para cada bin, e conta quantos pixels da imagem pertence a cada um dos intervalos das subdivisões e gera os histogramas.

Para tornar mais claro podemos pegar como exemplo algumas imagens exibidas na Figura 20 e o Quadro 3 exibe os valores de histogramas calculados para elas. Na primeira imagem podemos ver no histograma 1 (bin = 3) que 100% dos pixels se encontram na região mais escura, com isso sabemos que não temos um objeto próximo, nem em uma distância média, o histograma 2 (bin = 9) nos dá informações mais detalhadas pois divide a região mais escura em 3 partes, então podemos ver que 99% dos pixels são classificados no intervalo 3 da região mais escura.

Na imagem 2 temos um cenário diferente, pois o histograma 1 (bin=3) nos mostra que temos aproximadamente 78% dos pixels na região intermediária e 21% na região mais clara, com isso podemos verificar que existe algum objeto próximo. Se olharmos o histograma 2 dessa imagem, podemos ver que os 21% dos pixels se encontram no intervalo 6 da região mais clara, nos informando que está próximo, mas não tão próximo quanto se estivesse no intervalo 9.

Figura 20 – Exemplos de mapa de disparidade para calcular os histogramas



Fonte: Autor.

Quadro 3 – Exemplo de cálculo de histogramas para mapa de disparidades

IMAGEM 1			
HISTOGRAMA 1 BIN = 3	HISTOGRAMA 2 BIN = 9		
[0] = 34080 (100%)	[0] = 0 (0%)	[3] = 0 (0%)	[6] = 0 (0%)
[1] = 0 (0%)	[1] = 54 (0.15%)	[4] = 0 (0%)	[7] = 0 (0%)
[2] = 0 (0%)	[2] = 34026 (99.84%)	[5] = 0 (0%)	[8] = 0 (0%)
IMAGEM 2			
HISTOGRAMA 1 BIN = 3	HISTOGRAMA 2 BIN = 9		
[0] = 404 (1.18%)	[0] = 0 (0%)	[3] = 1555 (4.56%)	[6] = 7167 (21.02%)
[1] = 26509 (77.78%)	[1] = 0 (0%)	[4] = 14189 (41.63%)	[7] = 0 (0%)
[2] = 7167 (21.02%)	[2] = 404 (1.18%)	[5] = 10765 (31.58%)	[8] = 0 (0%)

Legenda: Quadro utilizado para demonstrar os valores dos 2 histogramas calculados.

Fonte: Autor.

Após o cálculo dos histogramas é realizado o mapeamento do histograma para intensidade, isso é realizado através da função implementada *calc_intensity()*, que recebe os histogramas como entrada e retorna a intensidade relativa ao histograma. O mapeamento é realizado de forma simples, primeiro é verificado o histograma 1 (bin=3) é verificado qual a região com maior predominância, seguindo a seguinte regra: se a região 2 (mais clara) tiver um índice maior que 5% ela é a região predominante, se não tiver é verificado se a região 1 (intermediária) tem um índice maior que 5% se tiver é a predominante, caso contrário a região 0 (mais escura é a predominante). Após o cálculo da região predominante, usamos o histograma 2 (bin=9) para verificarmos qual o valor da intensidade para aquela região, verificamos então qual o intervalo possui o maior valor e o mapeamento é feito conforme a Quadro 4.

Quadro 4 – Mapeamento de intensidades

Intervalo	Intensidade
0-1	0
2-4	20
5-7	40
8-10	100
11-13	120
14-17	140

Intervalo	Intensidade
18-49	200
50-81	220
82-255	240

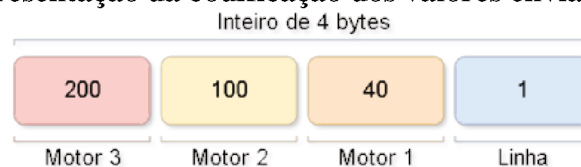
Fonte: Autor.

3.3.3 Codificação das intensidades

As intensidades mapeadas precisam ser enviadas para os motores de vibração que pertencem ao módulo informativo vibrátil. O Arduino é uma ótima ferramenta para agilizar o desenvolvimento de protótipos, mas uma das suas principais desvantagens é o seu baixo poder computacional, quando comparado aos computadores que estamos acostumados, além disso o software que é executado nele tem como característica ser executado dentro de um looping infinito. Com isso dentro do ciclo do looping ele precisa ler a intensidade, decodificar e ativar o motor.

Para conseguirmos ativar os 9 motores, teríamos que mandar a informação da intensidade codificada em um byte e seriam necessários pelo menos 9 bytes e 9 ciclos para que todos os motores fossem ativados e de forma serial. Para tentarmos otimizar esse processamento optou-se por codificar as intensidades agrupadas de 3 em 3 em um inteiro de 4 bytes, como sobraria um byte foi adicionado a informação da linha do grid. Com isso aumentou-se o número de bytes enviados de 9 para 12, mas em apenas 3 passadas no looping já é possível ativar os 9 motores. A Figura 21 exibe um exemplo dessa codificação e como os bytes são preenchidos.

Figura 21 – Representação da codificação dos valores enviado para o Arduino



Legenda: Nesse exemplo a linha do grid de 3 motores que será ativado é a linha 1 (de 3) e o motor 1 será ativado com o valor de 40, o motor 2 com o valor 100 e o motor 3 com o valor 200. Onde 0 é a menor intensidade e 255 a intensidade máxima.

Fonte: Autor.

A função utilizada para codificar é a *encode_intensities()*, que simplesmente coloca os 4 valores (linha do grid, intensidade do motor 1, intensidade do motor 2 e intensidade do motor 3) em um inteiro de 4 bytes. Isso é realizado fazendo o deslocamento de bits, o primeiro valor não é deslocado, o segundo valor é deslocado 8 bits, o terceiro valor é deslocado 16 bits e o por

fim o quarto valor é deslocado 24 bits, com isso no final temos os 4 bytes representado em 1 inteiro de 4 bytes.

3.3.4 Envio das informações para o Arduino

A maneira mais simples de se comunicar com o Arduino é utilizando a interface USB Serial. Usando a comunicação serial é possível trocar dados entre a placa e qualquer outro dispositivo com capacidade de comunicação serial, no nosso caso utilizaremos um notebook. Comunicação serial significa que um byte é transmitido por unidade de tempo, pode parecer lento, mas é uma maneira simples e comum de transmitir dados.

A comunicação entre o computador que executa o módulo de identificação de obstáculos e o módulo informativo vibrátil que é executado no Arduino foi implementada de forma bastante simples e o código fonte está disponível no **APÊNDICE C**. Fisicamente a comunicação é estabelecida via cabo USB e a própria placa faz a abstração necessária, pois o Arduino UNO utiliza o Virtual COM Port (VCP) que torna a comunicação extremamente fácil. O VCP driver permite que o computador faça a USB comportar-se como uma porta serial COM. Com isso basta conectar o cabo USB na placa e no computador e o canal de comunicação serial está pronto e estabelecido. Desse modo é possível escrever um simples código para porta serial e todos os dados serão enviados sobre o cabo USB.

O código criado facilita a conexão entre o PC e o Arduino e o envio de dados. A função *init_comm_arduino()* é responsável por verificar se o Arduino está conectado na porta padrão `/dev/ttyACM0`, se não estiver o programa é encerrado, pois sem ele não temos como informar ao usuário as distâncias. Se a placa estiver conectada a função faz a abertura da porta serial e configura algumas opções usadas na comunicação como o baud rate (setado para 57600 em ambos os lados) e os bits de paridade.

Após o estabelecimento da conexão o notebook está pronto para enviar os dados relativos a intensidade de vibração dos motores e o Arduino está pronto para receber. Como já foi explicado anteriormente, os dados são enviados codificados em valores do tipo inteiro e para enviarmos esses dados sobre a conexão serial foi implementada a função *send_int_to_arduino()*, que recebe o dado como parâmetro e o envia ao Arduino. Foi criado também a função *recv_int_to_arduino()* para recebermos dados enviados pela placa, mas foi usado somente com o propósito de debug. Como todas as intensidades são disponibilizadas ao mesmo tempo, para não precisarmos chamar a função *recv_int_to_arduino()* ela foi encapsulada em uma função que faz as chamadas em looping *send_intensities()*.

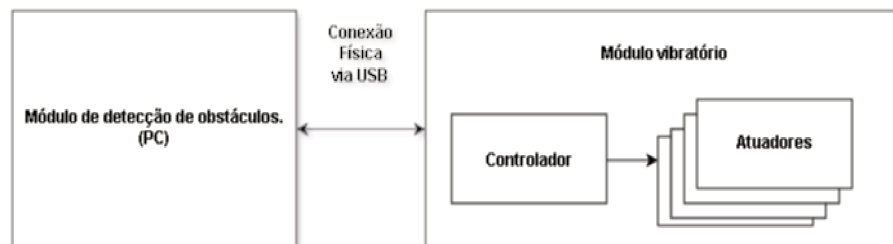
3.4 MÓDULO INFORMATIVO VIBRÁTIL

Esse módulo é de fundamental importância para o projeto, pois é de sua responsabilidade informar ao usuário a proximidade de um obstáculo, ele não é responsável por identificar e calcular as distâncias, pois isso é de responsabilidade do módulo de identificação de obstáculos.

O sistema possui dois componentes principais: controlador e atuadores. O controlador é responsável por receber as informações de distâncias de obstáculos e acionar os motores vibratórios de acordo com a intensidade relativa a distância informada. Os atuadores, como o próprio nome diz, são responsáveis por atuarem informando, no caso desse projeto vibrando, de acordo com a intensidade passada pelo controlador. Os atuadores são entidades totalmente passivas e não realizam nenhum processamento, pois trata-se de um componente puramente de hardware.

O módulo interage diretamente com o módulo de identificação de obstáculos de modo bidirecional, através de uma comunicação serial estabelecida por um cabo USB. A Figura 22 esquematiza essa conexão:

Figura 22 – Conexão entre módulo informativo vibrátil e de identificação de obstáculos.



Fonte: Autor

3.4.1 Controlador

3.4.1.1 Hardware

Para a implementação deste módulo do trabalho foi utilizado o Arduino UNO como o controlador do sistema, esse modelo foi escolhido devido as diversas vantagens já citadas, mas principalmente devido ao autor já possuir um exemplar da placa e ter alguma experiência em desenvolvimentos nessa plataforma, além dele atender aos requisitos do projeto. O Quadro 5 com as especificações técnicas disponibilizadas pelo fabricante:

Quadro 5 – Especificações técnicas do Arduino UNO

Características:	Valores:
Microcontrolador	ATmega328P
Tensão de Operação	5V
Tensão de entrada (recomendado)	7-12V
Tensão de entrada (limite)	6-20V
Pinos de entrada/saída digitais	14 (6 fornecem PWM)
Pinos PWM de entrada/saída digitais	6
Pinos de entrada analógico	6
Corrente DC por pino de entrada/saída	20 mA
Corrente DC para o pino 3.3 V	50 mA
Memória Flash	32 KB (ATmega328P) dos quais 0.5 KB usado pelo bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Velocidade do Clock	16 MHz
LED_BUILTIN	13
Comprimento	68.6 mm
Largura	53.4 mm
Peso	25 g

Fonte: ARDUÍNO

Para que o Arduino possa controlar os motores vibratórios é necessário utilizar os pinos de saída digitais. Os pinos digitais podem assumir somente dois níveis lógicos bem definidos, nível alto e baixo. Comumente, o nível lógico alto é a tensão de alimentação do Arduino (5V ou 3.3V) e nível lógico baixo é 0V (pino conectado ao GND). Os pinos do Arduino podem atuar como entrada ou saída, nesse projeto utilizaremos elas apenas como saída, com isso os pinos podem fornecer 0, nível baixo ou 5 V, nível alto, fazendo com que eles forneçam corrente. Segundo o fabricante a corrente máxima de cada pino de saída é em torno de 20mA. Essa corrente é mais do que suficiente para ligar um LED de alto-brilho e alguns sensores, porém não é suficiente para ligar a maioria dos motores. Caso uma corrente maior que o limite passe por um pino, este poderá ser danificado.

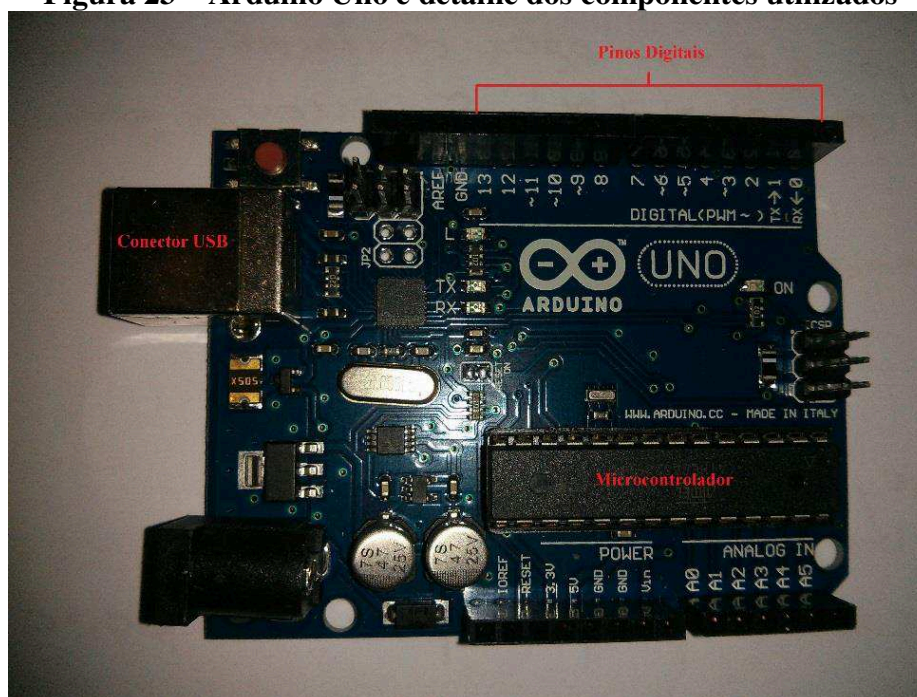
Como vimos acima, o Arduino fornece uma corrente muito pequena para conseguirmos ativar um motor vibratório, a velocidade máxima de vibração exigida pelos motores utilizados no projeto é algo em torno de 180mA, muito acima do que o Arduino é capaz de oferecer, dado essa limitação foi necessário utilizarmos outra fonte de energia que fornecesse a corrente necessária para alimentarmos os motores. Essa fonte externa será explicada com mais detalhes na Seção que descreve os atuadores (3.5.2).

Além do problema de a corrente não ser o suficiente as portas digitais por padrão não cumprem os requisitos necessários ao módulo informativo vibrátil, pois elas fornecem apenas 2 níveis lógicos, alto e baixo, e com isso somos apenas capazes de ligar e desligar os motores,

ao setarmos o nível alto ligamos o motor na velocidade máxima e ao setarmos o valor baixo desligamos o motor. Para contornarmos essa limitação precisamos utilizar a técnica Modulação por Largura de Pulso (Pulse Width Modulation – PWM). Explicaremos essa técnica melhor na Seção que descreve a implementação do software do controlador (3.5.1.2), mas com elas conseguimos simular uma saída analógica em uma porta digital, fazendo com que consigamos variar a intensidade da vibração dos motores.

A Figura 23, mostra a placa Arduíno UNO utilizada no projeto e os principais componentes utilizados: pinos de saídas digitais, conector USB e microcontrolador.

Figura 23 – Arduíno Uno e detalhe dos componentes utilizados

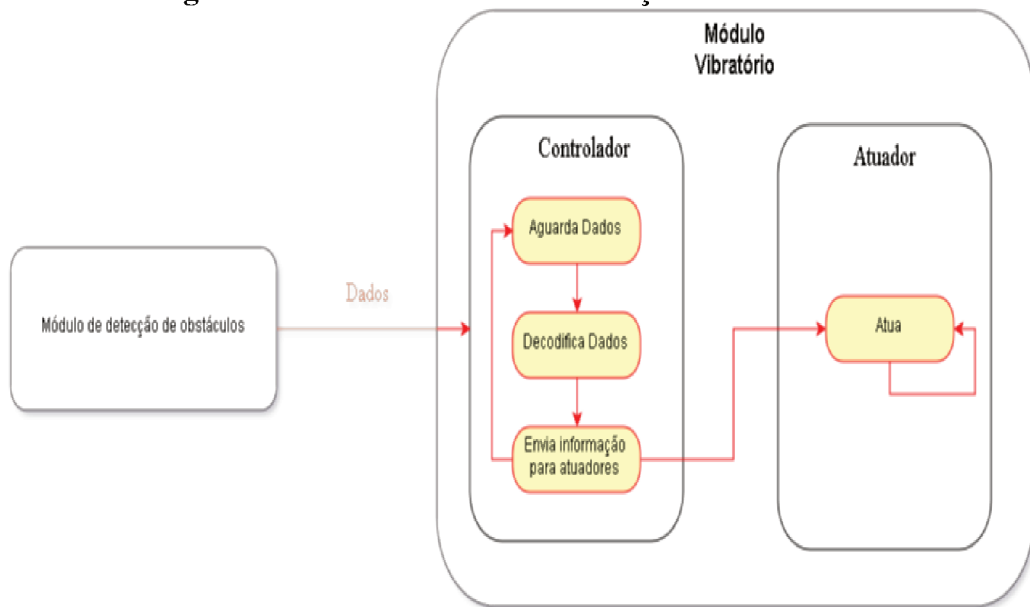


Fonte: Autor

3.4.1.2 Software

O software implementado no controlador é bastante simples, pois a tarefa principal é receber os dados com as informações da intensidade da vibração dos motores, enviado pelo módulo de identificação de obstáculos, decodificar esses dados e ativar os motores a partir dos valores decodificados. A Figura 24 mostra o fluxo dos dados e as principais atividades realizadas. Para simplificar o processamento do controlador, optou-se por ele não receber o valor da intensidade diretamente e não ter que calculá-la ou mapeá-la.

Figura 24 – Fluxo de dados e interação entre módulos.



Fonte: Autor

O software desenvolvido para controlar o sistema vibratório foi implementado na linguagem C/C++ e foram utilizadas as bibliotecas e a IDE padrão do Arduino. O Arduino espera que pelo menos duas funções existam no programa implementado: uma chamada *setup()* e outra chamada *loop()*. *Setup()* é a função onde se coloca todo o código necessário para configurar o microcontrolador e é executado apenas uma vez ao iniciar a placa, já a função *loop()* contém o core do programa que executa repetidamente até desligarmos a placa. O APÊNDICE D lista o código criado.

Dado a obrigatoriedade de implementarmos as funções descritas acima, utilizamos a função *setup()* para inicializarmos os pinos que serão utilizados para ativar os motores. Para utilizarmos os pinos do Arduino precisamos setar o tipo do pino utilizado, no nosso caso o pino será utilizado como saída (OUTPUT), com isso inicializamos os 9 pinos utilizados através da função: *pinMode(id_pino, OUTPUT)*, também inicializamos a conexão serial, através da função: *Serial.begin()*. A partir desse momento já somos capazes de nos conectarmos com o módulo de identificação de obstáculos e ativarmos os motores vibratórios.

A função *loop()*, como já foi dito, é a função que o Arduino repete infinitamente, nela nós adicionamos a função que lê os dados enviados pelo módulo de identificação de obstáculos, se tiverem dados disponíveis, chama uma função de decodificação, pois como foi explicado na Seção 3.3.3. Codificação das intensidades, para otimizarmos o envio dos dados é transmitido um valor inteiro de 32 bits que contem a informação de intensidade de vibração de 3 motores (1 byte por motor) e a informação que é possível recuperar os ids dos motores. Devido a essa

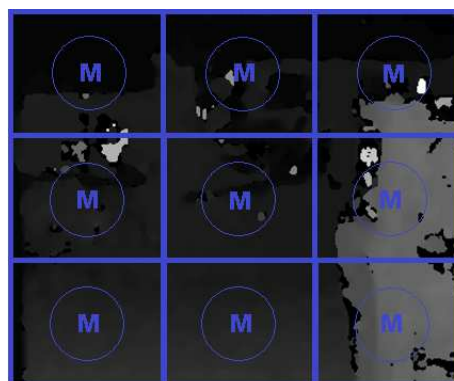
codificação que é necessário a função de decodificação, após os valores serem decodificados os motores são ativados.

Como dito anteriormente, as portas digitais por padrão enviam apenas dois sinais lógicos: alto e baixo, para contornarmos essa limitação foi necessário utilizar uma técnica chamada PWM, que é utilizada para obter resultados analógicos por meios digitais. Ela consiste na geração de uma onda quadrada, um sinal que troca o valor entre os sinais alto e baixo. Este padrão de alto e baixo pode simular tensões entre 5V e 0V mudando a porção de tempo que o sinal fica alto contra o tempo que o sinal fica baixo. A duração do tempo alto é chamada largura de pulso e sua alteração provoca mudança no valor médio da onda, indo desde 0V (0%) a 5V (100%) no caso do Arduino. Desse modo conseguimos ativar os motores de forma a termos intensidades que se relacionam com as distâncias. Para utilizarmos essa técnica sobre as portas digitais precisamos utilizar a função *analogWrite()* que permite um range entre 0 e 255, onde 0 é o motor desligado e 255 é o motor com a sua velocidade máxima (desde que a corrente do circuito permita), no lugar da função *digitalWrite()* que só permite Alto (HIGH) e baixo (LOW).

3.4.2 Atuadores

Os atuadores têm papel importantíssimo no projeto, pois é através dele que o usuário terá contato com as informações relativas a distâncias mapeadas para informações táteis, ou vibratórios. Para que o usuário pudesse ter um número significativo de informações optou-se por utilizarmos um grid de motores vibratórios com 9 motores, dispostos 3 a 3 conforme a Figura 25 e cada uma das posições representa uma das regiões segmentadas do mapa de disparidade. Com isso é possível informar através da vibração a existência e proximidade de um obstáculo em uma das 9 regiões do grid.

Figura 25 – Representação do grid de motores vibratórios e regiões do mapa de disparidade



Legenda: Para cada um dos motores do grid está associada uma região da imagem.

Fonte: Autor

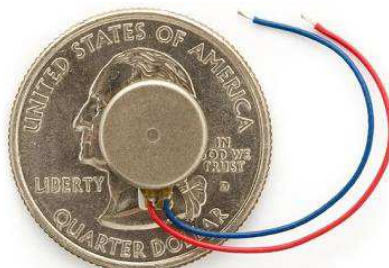
Os motores vibratórios escolhidos foram do tipo flat devido ao seu pequeno tamanho, seu peso e por não possuírem eixo são altamente recomendados para indicadores não audíveis. São usados em um grande número de aplicações para indicar ao usuário quando um status mudou. É possível alterar a velocidade de vibração alterando a intensidade de corrente que circula pelo motor, quanto maior a corrente maior a velocidade de vibração e quanto menor a corrente menor a velocidade de vibração. O Quadro 6 informa algumas das especificações técnicas segundo o fabricante e a Figura 26 mostra uma imagem em tamanho real do modelo escolhido.

Quadro 6 – Especificações técnicas do motor vibratório

Características:	Valores:
Material	Ferro
Tensão nominal	DC 3V
Tensão de trabalho	DC 2.5~4.0V
Temperatura de trabalho	-20C°~+60C°
Velocidade de Rotação	Mín. 9000RPM
Corrente Nominal	Max. 90mA
Corrente de partida	Máx, 120mA
Tensão de partida	DC 2.3V
Resistência de isolamento	10Mohm
Dimensões	4,0 cm x 1,0 cm x 0,3 cm
Peso	5g

Fonte: DEALEXTREME

Figura 26 – Motor vibratório em escala real.



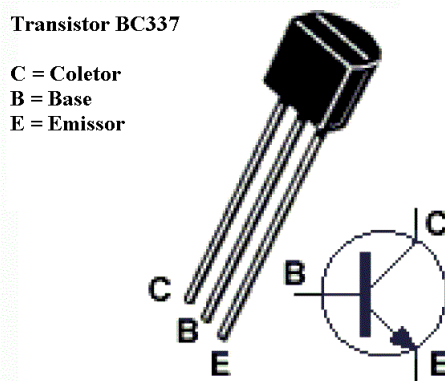
Fonte: SPARKFUN <https://www.sparkfun.com/products/8449>

Como já foi dito anteriormente, o Arduíno não é capaz de fornecer através de seus pinos a corrente necessária para ativar os motores de vibração do modo como o projeto necessita. Além disso se ligarmos o motor diretamente as portas do Arduíno há uma grande chance de danificarmos a placa, pois como a corrente de um circuito é exigido pela carga e não limitada pela fonte, ao ligarmos diretamente ao pino é possível que a corrente exigida seja maior do que o limite fornecido pela porta podendo danificá-la.

Para resolvermos esse problema utilizamos um transistor NPN BC337 usado como um switch que usa a baixa corrente emitida pelo pino digital da saída do Arduíno para controlar

uma corrente muito maior do motor. Como mostrado na Figura 27, o transistor tem 3 terminais: a base, o coletor e o emissor. A base será ligada ao pino de saída digital do Arduino, o Coletor é ligado a um terminal do motor e o Emissor ao ground (terra). Como usaremos o transistor como um interruptor ao aplicarmos tensão na base a corrente flui entre o emissor e o coletor.

Figura 27 – Transistor NPN BC337



Fonte: Autor

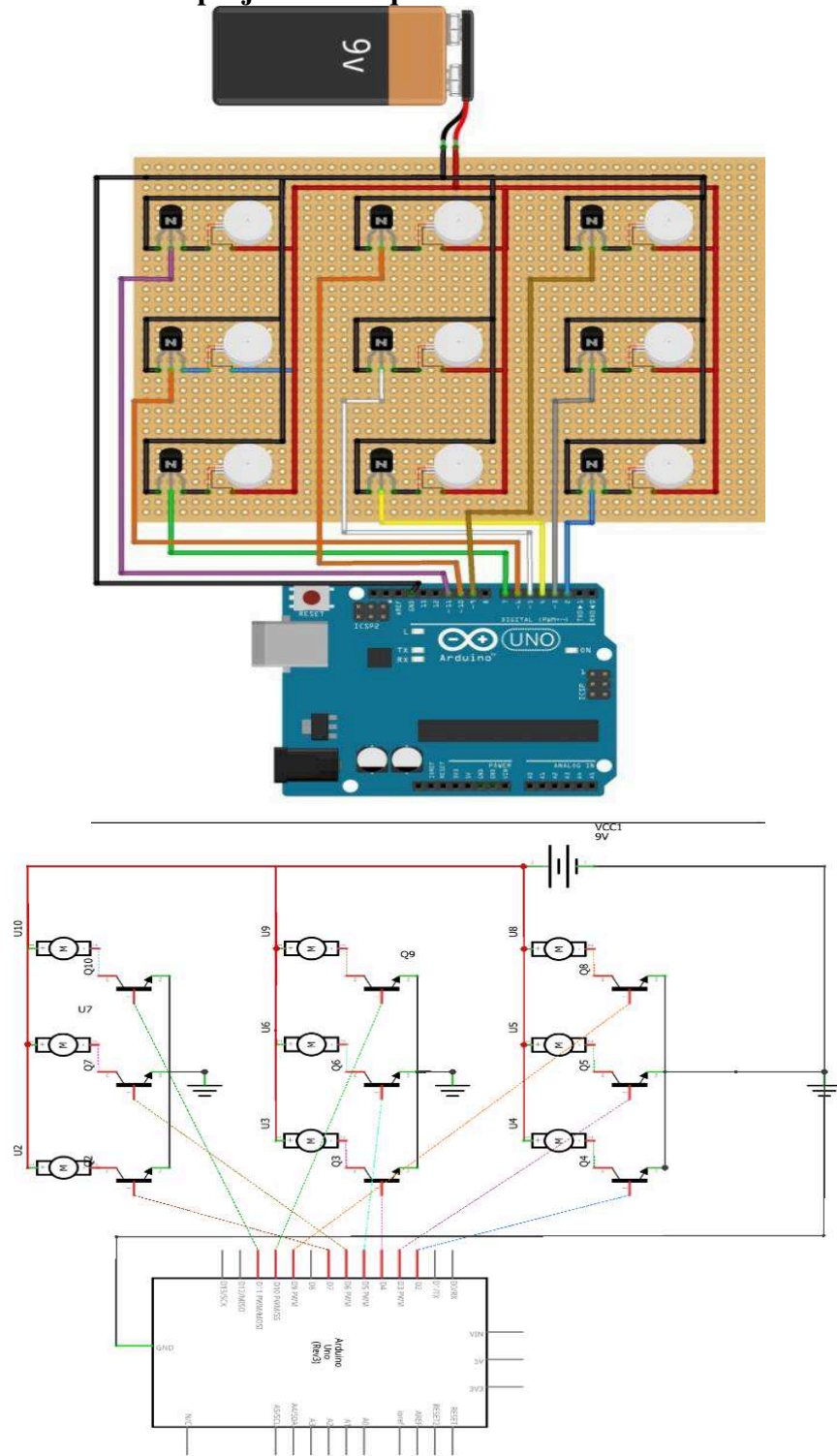
Outro problema contornado foi a limitação da corrente máxima fornecida pelo pino VCC de 5V, se alimentado via USB essa corrente máxima é em torno de 450mA, o que não é suficiente para alimentar simultaneamente os 9 motores do projeto. O contorno foi simples, bastou utilizarmos uma fonte externa que forneça a corrente necessária, optou-se por utilizar uma bateria de 9V, devido ao preço e a facilidade de adaptar ao sistema, pois só é necessário um clip de bateria.

A Figura 28 mostra o circuito projetado para o módulo informativo vibrátil, nela é possível ver todos os componentes utilizados:

- 9 motores vibratórios.
- 9 transistores NPN BC337.
- 1 Bateria de 9 V.
- 1 Arduino UNO.
- 9 pinos de saída digital.

Podemos visualizar também que para cada motor é utilizado um pino de saída digital do Arduino, com isso podemos controlar de modo independente cada um dos motores, vibrando com a intensidade necessária para informar o usuário a existência de um obstáculo. A Figura 28, exibe também o modo esquemático do circuito projetado.

Figura 28 – Circuito projetado e esquemático do módulo informativo vibrátil



Fonte: Autor

3.5 PROTÓTIPO COMPLETO

Nessa Seção são mostrados os dois dispositivos criados que junto ao software formam o protótipo proposto nesse trabalho. Todos os dispositivos foram criados pelo autor de modo artesanal e com materiais de baixo custo.

O primeiro dispositivo desenvolvido é o utilizado pelo módulo de captura de imagens e é composto de um par de câmeras acoplados a um óculos e elas são responsáveis pela captura das imagens e estão ligadas ao notebook, responsável pelo processamento das imagens, por cabos USBs. A distância entre as lentes das câmeras (*baseline*) é de 75 mm e elas foram fixadas de modo que não se movessem e necessitassem recalibração. A Figura 29 exhibe o dispositivo construído.

Figura 29 – Dispositivo criado para módulo de captura de imagens



Fonte: Autor

O segundo dispositivo é composto de duas partes, mas os dois juntos fazem parte do módulo informativo vibrátil, que é responsável por informar ao usuário a proximidade de um obstáculo. Na Figura 30, temos a primeira parte que foi chamada de controlador pois possui a placa do Arduino e o circuito responsável por ativar os motores vibratórios. Esse dispositivo é ligado ao notebook por um cabo USB e ligado ao dispositivo de atuação por dois conectores molex de 6 vias. A Figura 31, mostra o dispositivo em detalhes, podendo ser visto o circuito desenvolvido, o Arduino e a bateria usada para alimentar os motores vibratórios.

Figura 30 – Dispositivo controlador dos motores vibratórios



Fonte: Autor

Figura 31 – Visão mais detalhada do controlador

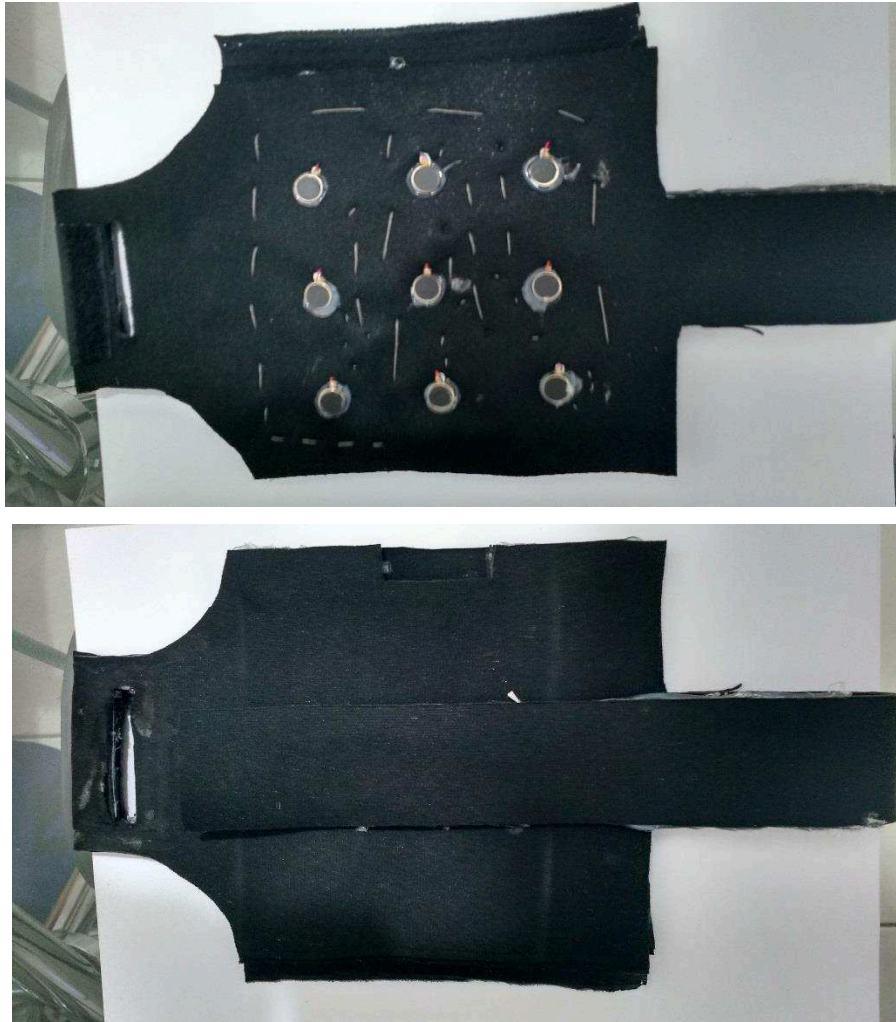


Fonte: Autor

Na Figura 32, temos a segunda parte do módulo informativo vibrátil que é composta pelos atuadores, que nada mais são do que os motores vibratórios. Os 9 motores vibratórios foram dispostos em 3 colunas separados por 4 cm cada um e presos a um tecido de neoprene que será colocado no braço do usuário conforme a Figura 33. Esse dispositivo é ligado ao dispositivo controlador através de um cabo com 10 pares de fios, um usado na alimentação do circuito e os outros nove são um para cada motor.

É possível encontrar na literatura diversas regiões do corpo onde dispositivos similares são utilizados, mas como não era o escopo desse trabalho encontrar a região que possui a maior sensibilidade para tal uso, a escolha da região do braço para utilizar o dispositivo informativo foi por questões puramente de praticidade em vestir, pois o dispositivo foi criado pensando na facilidade do usuário em colocá-lo e tirá-lo, além disso, a região utilizada demonstrou ter sensibilidade suficiente para detectar diferentes intensidades.

Figura 32 – Visão interna e externa do dispositivo informativo vibrátil.



Fonte: Autor

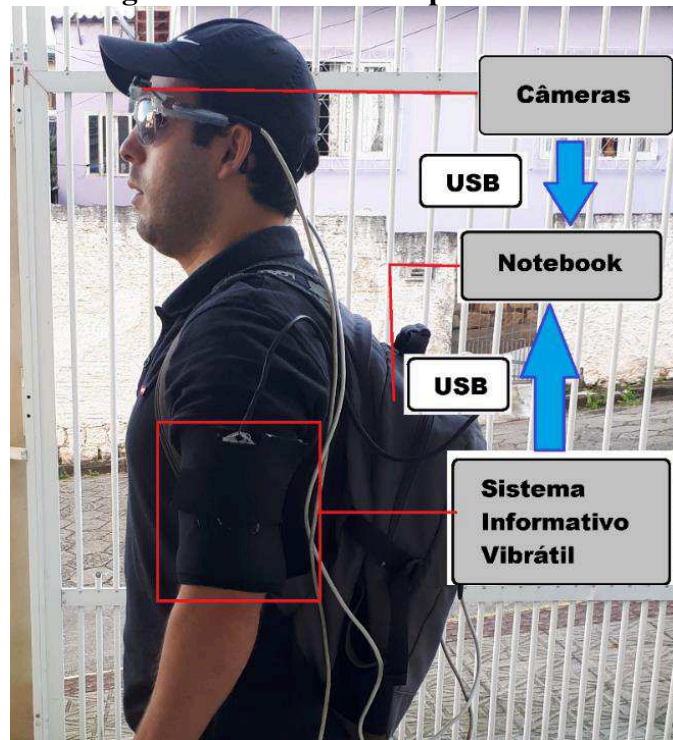
Figura 33 – Usuário com dispositivo informativo vibrátil



Fonte: Autor

Nas próximas imagens podemos ver o sistema completo em uso.

Figura 34 – Sistema completo em uso



Fonte: Autor

4 EXPERIMENTOS E RESULTADOS

Nessa Seção iremos apresentar os experimentos realizados para demonstrar o funcionamento do protótipo desenvolvido. Os testes foram divididos em três grupos, o primeiro para validar o sistema de visão computacional estéreo do protótipo, o segundo para verificar a performance do sistema desenvolvido e o terceiro para validar a integração com de todos os módulos.

4.1 TESTES DE VISAO COMPUTACIONAL

Esta Seção apresenta dois testes realizados para a validar se o sistema de visão computacional desenvolvido apresenta resultados aceitáveis para ser utilizado como uma ferramenta auxiliar para a locomoção de pessoas com deficiência visual. O primeiro teste realizado é para verificar se o sistema é capaz de detectar objetos em diferentes distâncias e se consegue estimar essa distância. O segundo teste foi realizado para identificar a distância mínima e máxima que o sistema pode atuar.

4.1.1 Parâmetros utilizados

Foram realizados diversos testes manuais que antecederam este teste para encontrarmos os melhores parâmetros para serem utilizados no algoritmo de geração do mapa de disparidade. Como o foco do projeto não era encontrar o melhor conjunto de parâmetros e como eles podem variar dependendo de condições externas como por exemplo a iluminação do ambiente. Foram utilizados parâmetros que retornaram um mapa de disparidade que o autor julgou aceitável. Esta configuração foi utilizada em todos os testes.

Abaixo segue a lista dos parâmetros utilizados e seus valores:

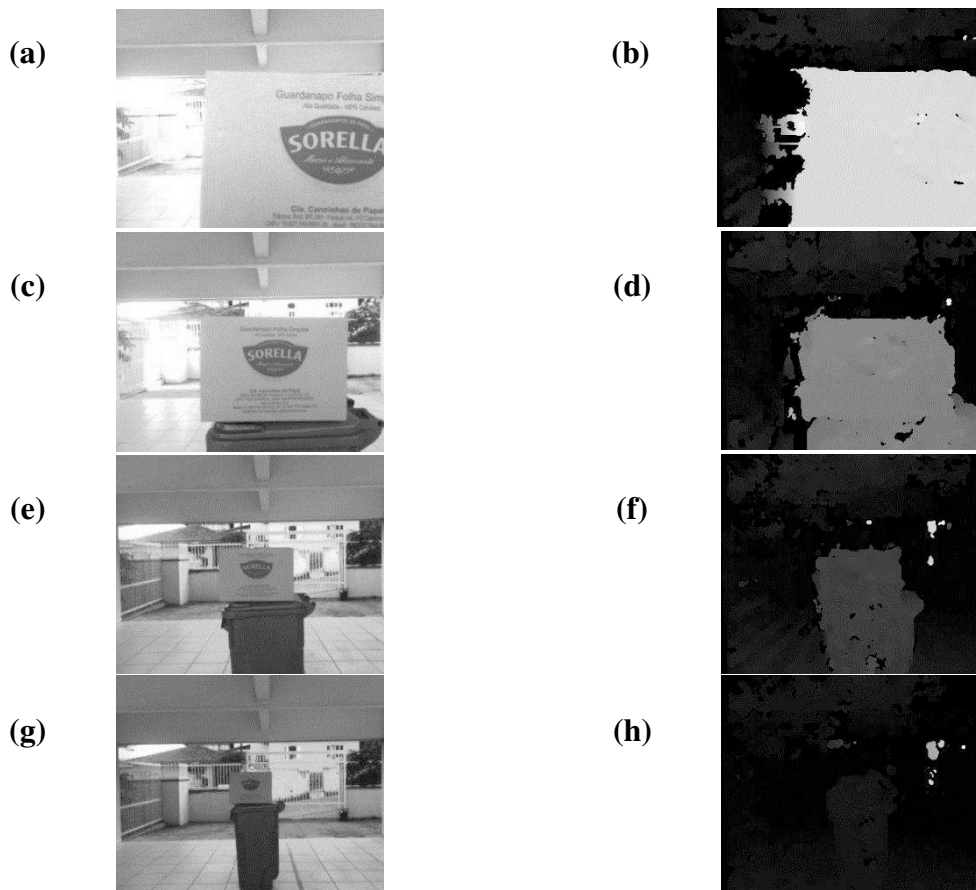
- $\text{minDisparity} = 0$
- $\text{numDisparities} = 96$
- $\text{SADWindowSize} = 9$
- $P1 = 240$
- $P2 = P1 * 15$
- $\text{disp12MaxDiff} = 25$
- $\text{preFilterCap} = 55$

- uniquenessRatio = 12
- speckleWindowSize = 82
- speckleRange = 95
- fullDP = false

4.1.2 Testes distância x disparidade

O teste consistiu em colocar em frente as câmeras o mesmo objeto, mas em distâncias conhecidas, que variaram dentro do seguinte range: 40 cm até 6 metros, a variação pode ser vista melhor Quadro 7. Esse objeto era capturado pelo sistema, o seu mapa de disparidade era gerado e a partir dele a sua distância era estimada. O teste foi realizado duas vezes uma para a resolução 352x288 e outra para a resolução 640x480, o intuito é identificar qual das resoluções possuem um resultado melhor. A Figura 35 exibe algumas imagens dos objetos e seus mapas de disparidades.

Figura 35 – Exemplos das amostras utilizadas no teste 1.



Legenda: Imagem do objeto localizado em diversas distâncias e seu mapa de disparidade
(a-b) 55 cm da câmera (c-d) 100 cm da câmera (e-f) 200 cm da câmera (g-h) 350cm da câmera

Fonte: Autor.

4.1.2.1 Resultados

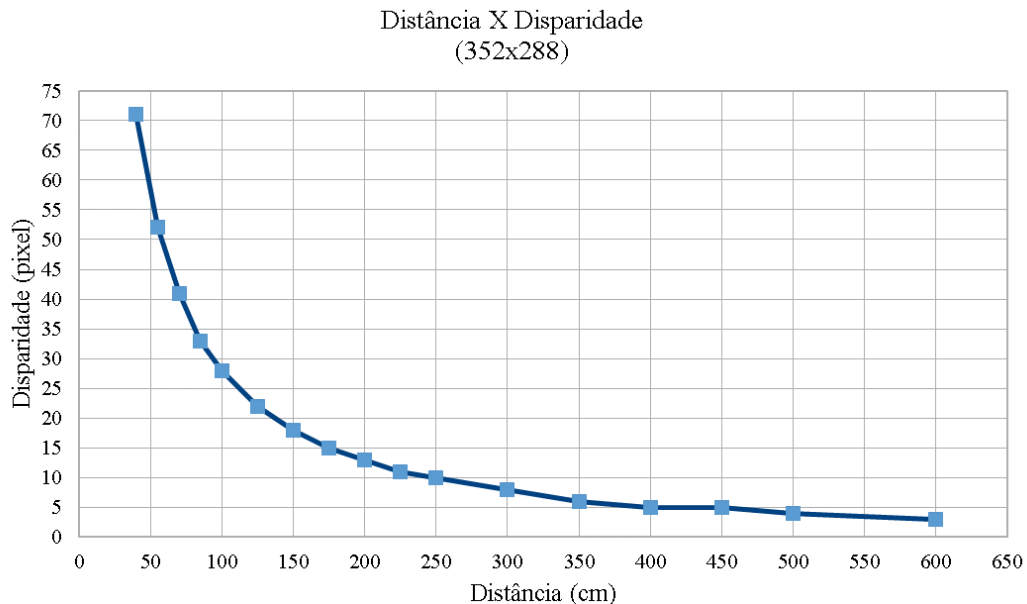
No Quadro 7 e nas Figuras 36 e 37, podemos visualizar que o sistema se comporta conforme a literatura descreve e o que podemos ver na Figura 6, que a disparidade é inversamente proporcional a distância, pois para distâncias pequenas como por exemplo 55 cm temos uma disparidade alta (52 e 85, para as resoluções 352x288 e 640x480 respectivamente) se compararmos com distâncias maiores como 200 cm (13 e 21).

Quadro 7 – Medições de Disparidade x Distância

Distancia real (cm)	Disparidade (352x288)	Disparidade (640x480)
40	71	
55	52	85
70	41	67
85	33	56
100	28	45
125	22	36
150	18	29
175	15	25
200	13	21
225	11	18
250	10	16
300	8	13
350	6	11
400	5	9
450	5	8
500	4	7
600	3	6

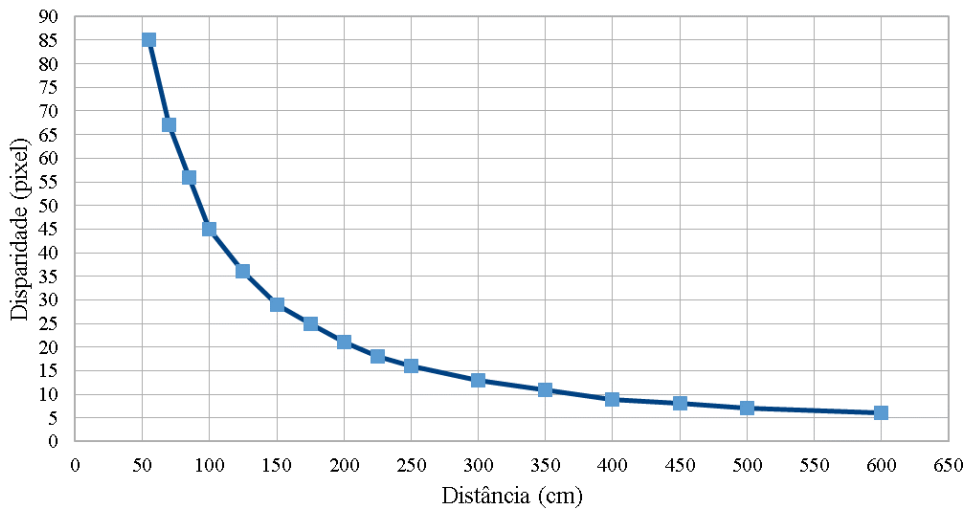
Fonte: Autor.

Figura 36 – Gráfico disparidade x distância para resolução 352x288



Fonte: Autor.

Figura 37 – Gráfico disparidade x distância para resolução 640x480
Distância X Disparidade
(640x480)



Fonte: Autor.

Nesse teste podemos perceber também que o sistema não possui uma precisão muito grande, pois conforme a distância vai aumentando a distância estimada vai ficando mais distante do valor correto. Isso ocorre para ambas as resoluções, sendo que a resolução menor (352x288) tem resultados piores em distâncias maiores, mas para distâncias menores ela tem um erro menor se comparada a resolução maior. Podemos verificar isso através do Quadro 8 e das Figuras 38 e 39. O erro entre o valor estimado e o valor real pode ter várias origens, mas possivelmente é causado pela calibração das câmeras, pois os valores utilizados para fazer o cálculo da distância são retirados diretamente das matrizes geradas na calibração.

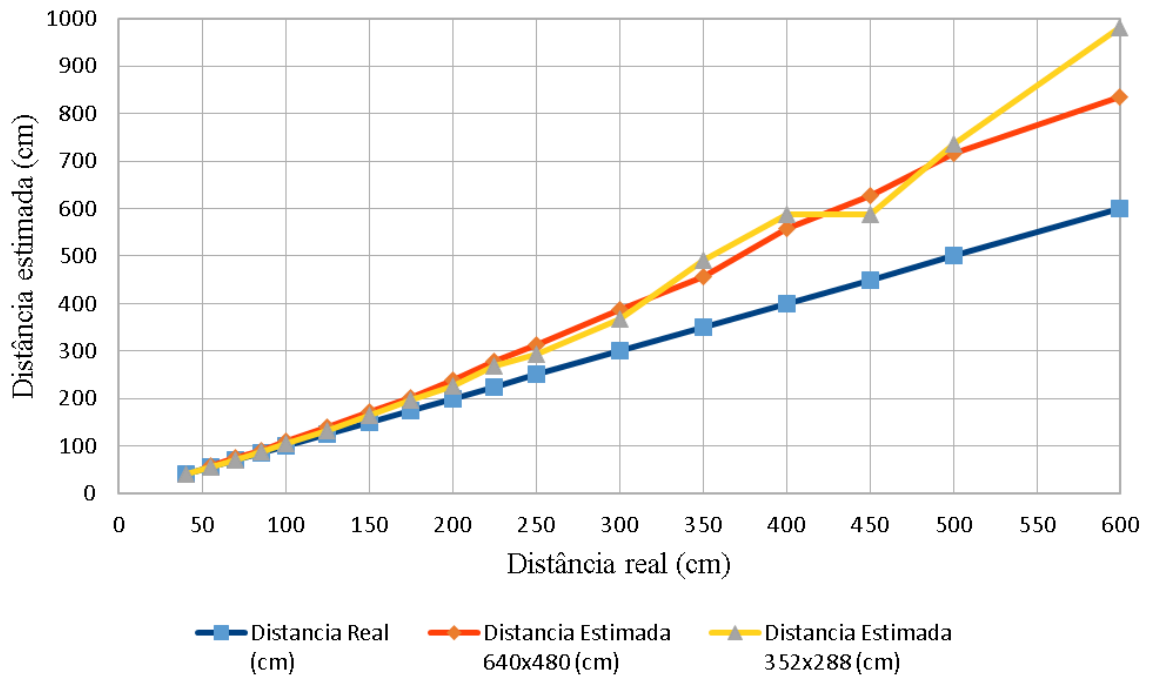
Quadro 8 – Distância real x distâncias estimadas

Distancia Real (cm)	Distancia Estimada 640x480 (cm)	Distancia Estimada 352x288 (cm)
40		41.4
55	59	56.6
70	74.9	71.7
85	89.6	89.1
100	111.5	105.1
125	139.4	133.7
150	173	163.5
175	200.7	196.2
200	239	226.4
225	278.8	267.5
250	313.7	294.3
300	386.1	367.9
350	456.3	490.5
400	557.7	588.6
450	627.4	588.6
500	717	735.8
600	836	981.1

Fonte: Autor.

Figura 38 – Gráfico da distância real x distância estimada

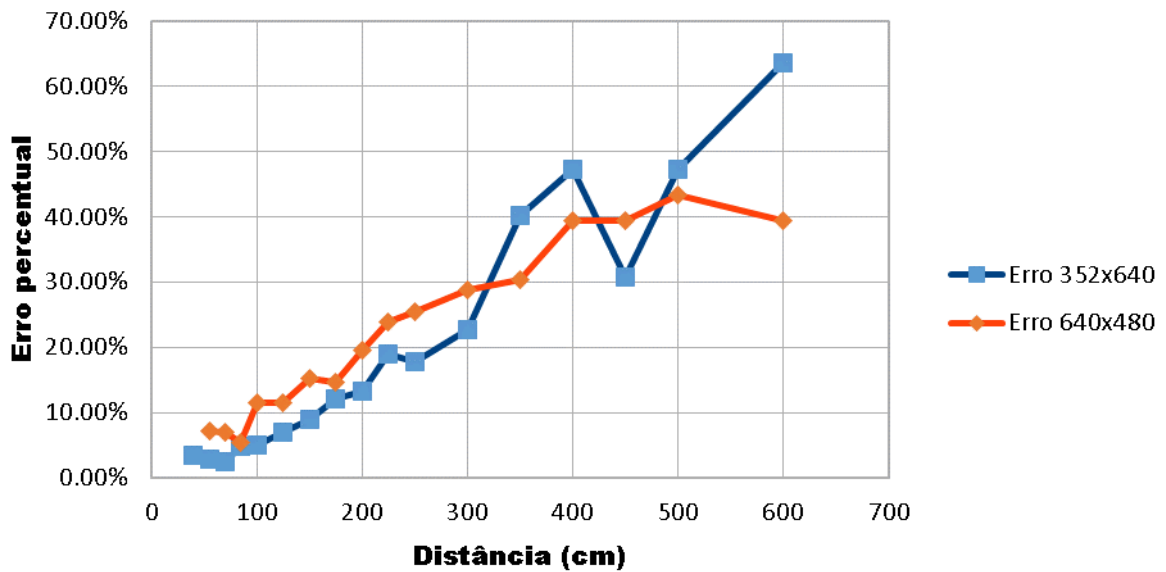
Distância Real X Distância estimada 352 X Distância estimada 640



Fonte: Autor.

Figura 39 – Gráfico dos erros das estimativas

Erro 352 X Erro 640



Fonte: Autor.

Este teste nos mostrou que apesar de não ser muito preciso a distância estimada pelo sistema utilizando o mapa de disparidade é possível a partir dele ter uma certa noção do quanto estamos perto ou longe de um objeto de um modo relativo sem ser necessário uma precisão.

4.1.3 Testes de distância mínima e distância máxima

O teste utilizou como base o mesmo conjunto de amostras utilizados no teste anterior e as mesmas resoluções. As imagens são de um objeto disposto na frente do sistema em diferentes distâncias e foi verificado a margem de erro que consta no Quadro 9

Quadro 9 – Distância estimada e erro.

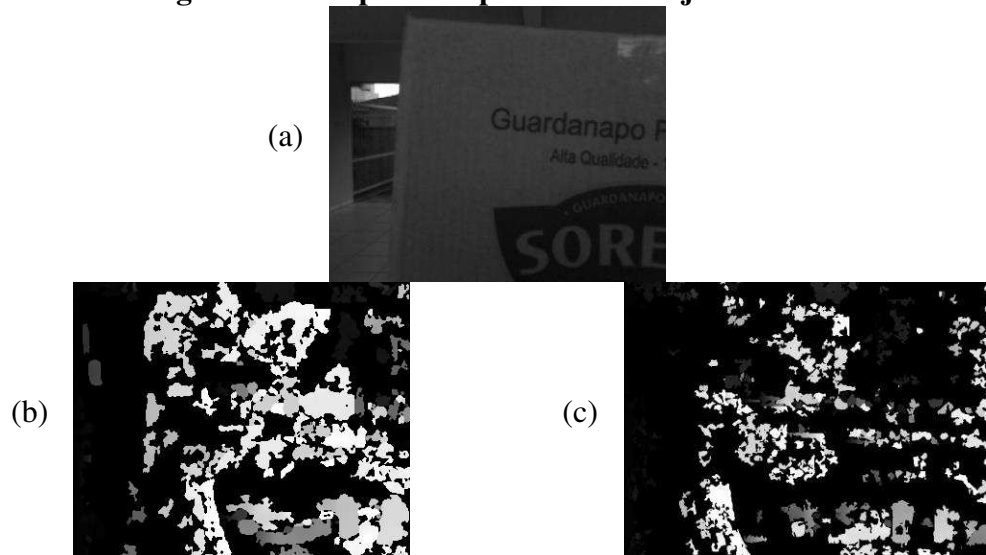
Distancia Real (cm)	Distancia Estimada 352x288 (cm)	Erro Estimativa 352x288	Distancia Estimada 640x480 (cm)	Erro Estimativa 640x480
40	41.4	3.50%	-	-
55	56.6	2.91%	59	7.27%
70	71.7	2.43%	74.9	7.00%
85	89.1	4.82%	89.6	5.41%
100	105.1	5.10%	111.5	11.50%
125	133.7	6.96%	139.4	11.52%
150	163.5	9.00%	173	15.33%
175	196.2	12.11%	200.7	14.69%
200	226.4	13.20%	239	19.50%
225	267.5	18.89%	278.8	23.91%
250	294.3	17.72%	313.7	25.48%
300	367.9	22.63%	386.1	28.70%
350	490.5	40.14%	456.3	30.37%
400	588.6	47.15%	557.7	39.43%
450	588.6	30.80%	627.4	39.42%
500	735.8	47.16%	717	43.40%
600	981.1	63.52%	836	39.33%

Fonte: Autor.

4.1.3.1 Resultados

Para distâncias menores que 40 cm o mapa de disparidade começou a apresentar muitos ruídos, podemos verificar isso através do mapa gerado para um objeto a 25 cm da câmera, conforme a Figura 40. Isso ocorreu tanto para as imagens na resolução 640x480, quanto nas imagens na resolução 352x288.

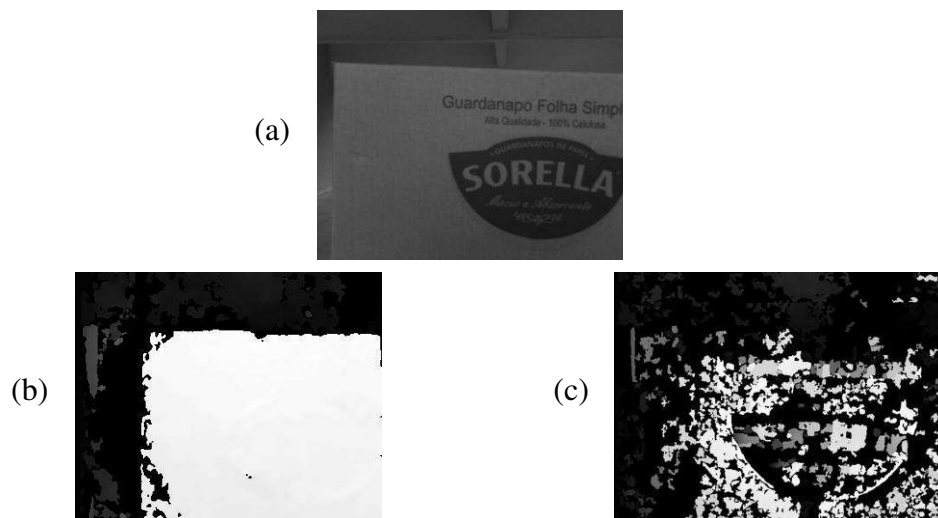
Figura 40 – Mapa de disparidade de objeto a 25 cm



Legenda: (a) Imagem do objeto localizado a 25 cm das câmeras do sistema
 (b) mapa de disparidade para resolução 352x288
 (c) mapa de disparidade para resolução 640x480
 Fonte: Autor.

Para objetos a uma distância de 40 cm foram obtidos dois comportamentos distintos, para a resolução 352x288 o mapa de disparidade foi gerado de modo satisfatório e para a resolução 640x480 ele apresentou muito ruído incapacitando a obtenção de uma medida satisfatória da distância, podemos visualizar isso na Figura 41.

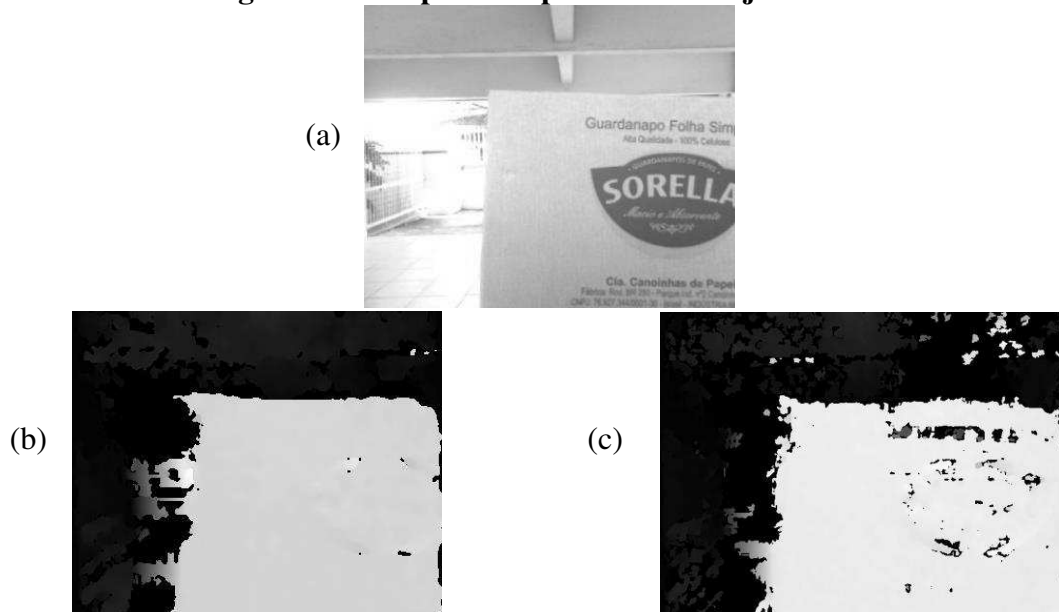
Figura 41 – Mapa de disparidade de objeto a 40 cm



Legenda: (a) Imagem do objeto localizado a 40 cm das câmeras do sistema
 (b) mapa de disparidade para resolução 352x288
 (c) mapa de disparidade para resolução 640x480
 Fonte: Autor.

Para distâncias de 55 cm ambas as resoluções apresentaram resultados satisfatórios, conforme a Figura 42.

Figura 42 – Mapa de disparidade de objeto a 55 cm



Legenda: (a) Imagem do objeto localizado a 55 cm das câmeras do sistema
 (b) mapa de disparidade para resolução 352x288
 (c) mapa de disparidade para resolução 640x480
















Fonte: Autor.

Distância máxima:

A distância máxima é mais difícil de ser calculada, pois como vimos no Quadro 9 o valor calculado vai se deteriorando quando a distância aumenta. Podemos ver que o objeto ainda é perceptível em grandes distâncias, mas o valor calculado da sua distância possui erros muito grandes.

Quadro 10 – Mapa de disparidade x longa distância

Distância (cm)	Objeto	Resolução (352x288)	Resolução (640x480)
250			

300			
350			
400			
500			
600			

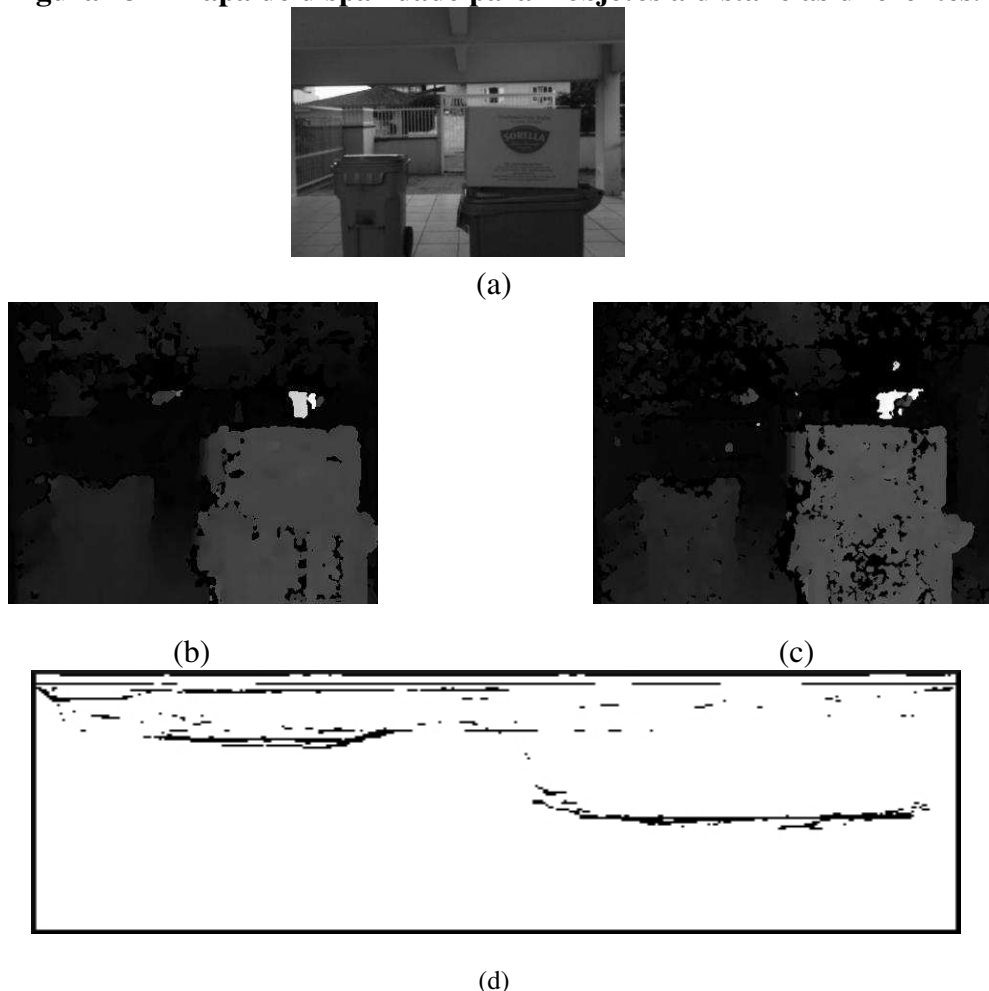
Fonte: Autor.

Para distâncias maiores que 600 cm o erro e a quantidade de ruídos se mostrou excessivo, com isso e para manter uma margem de segurança a distância máxima que o sistema atua é 6m.

Diferenciar objetos a distância diferentes:

Para demonstrar que através do mapa de disparidade é possível identificar dois objetos em distâncias diferentes foi realizado o seguinte experimento, foi colocado um objeto a uma distância de 150 cm e outro a uma distância de 250 cm, conforme a Figura 43 e foi gerado o mapa de disparidade para as duas resoluções (352 e 640). Através da imagem u-disparity, que é uma representação de uma visão de cima dos objetos podemos verificar que através do mapa de disparidade podemos identificar objetos em distâncias diferentes.

Figura 43 – Mapa de disparidade para 2 objetos a distâncias diferentes.



Legenda: (a) Dois objetos localizados a distâncias diferentes 150 cm e 250 cm.
 (b) mapa de disparidade para resolução 352x288
 (c) mapa de disparidade para resolução 640x480
 (d) vista superior do mapa de disparidade (u-disparity).

Fonte: Autor.

Como a disparidade não possui um comportamento linear com relação a distância, após os testes verificou-se a necessidade de subdividir os ranges de disparidades que antes eram

uniformes, isto é, todos os intervalos de distância (perto, médio e longe) possuíam a mesma quantidade de disparidade: $255/3$. A nova divisão foi realizada do seguinte modo:

- Distâncias entre 0 e 150 cm, são classificados como perto e o range da disparidade varia de 255 até 17.
- Distâncias entre 150 cm e 300 cm são classificados como média distância e o range da disparidade varia de 16 até 8.
- Distâncias maiores que 300 cm são classificados como longa distância e o range da disparidade varia entre 7 e 0.

4.2 TESTES DE DESEMPENHO

O teste de desempenho foi realizado para verificar se o sistema seria viável em tempo real e também para escolher qual das resoluções teria um desempenho melhor, visto que em termos de erros das distâncias estimadas elas possuem um comportamento semelhante para distância pequenas, mas para distâncias maiores a resolução 640x480 obteve melhores resultados.

Para realizar o teste foi utilizado os mesmos cenários dos testes anteriores, onde um objeto era colocado em distâncias conhecidas as distâncias variaram entre 55 cm e 600 cm. Foram pegos os 100 primeiros frames de cada cena e calculado a média do tempo de execução para os seguintes algoritmos:

- **Captura das imagens:** onde é realizada a captura das imagens da câmera da esquerda, direita e realizado uma conversão de imagens para preto e branco e também o remapeamento para corrigir distorções.
- **Geração do mapa de disparidade:** nesse código é gerado o mapa de disparidade é usado como entrada as imagens pré-processadas e gerado um mapa de disparidade.
- **Calculo, codificação e envio das intensidade:** foram agrupadas essas 3 funções para tornar mais claro o processo, a partir do mapa gerado a imagem é segmentada, é calculado os histogramas para identificar possíveis objetos e suas intensidades. Logo após é codificado as intensidades e são enviados para o sistema vibratório.
- **Detecção de objetos:** para termos uma ideia do tempo que leva toda a etapa após a captura das imagens foram somados o tempo da geração do mapa de disparidade, cálculo, codificação e envio de intensidades.

- **Tempo total de execução:** é o tempo que leva para ser analisado/processado um par de frames capturado- pelas câmeras.

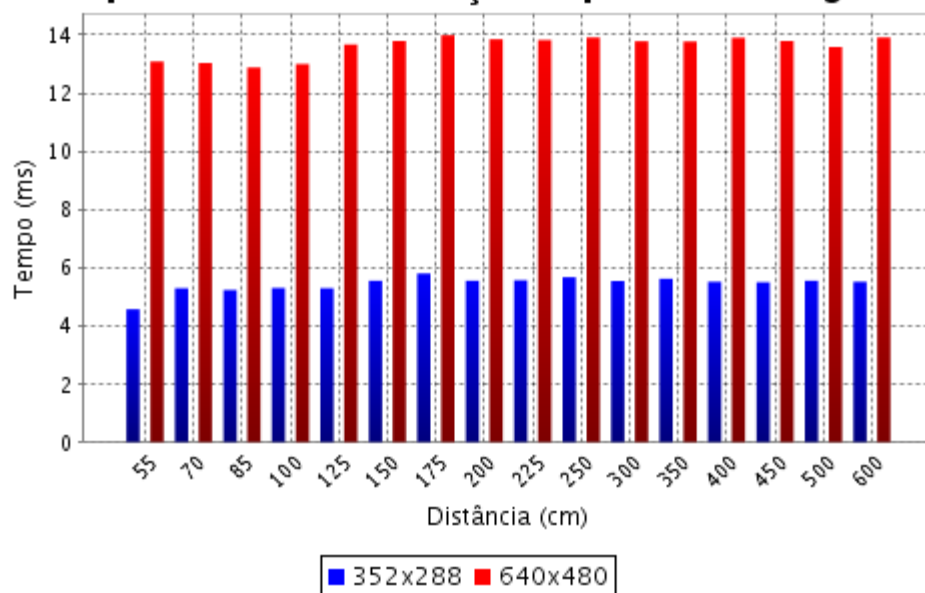
Foi utilizado o seguinte trecho de código C++11 para calcular o tempo de execução:

```
auto start = high_resolution_clock::now();
//Código que será verificado o tempo de execução
auto end = high_resolution_clock::now();
duration<double, milli> fp_ms = end - start;
```

4.2.1 Resultados

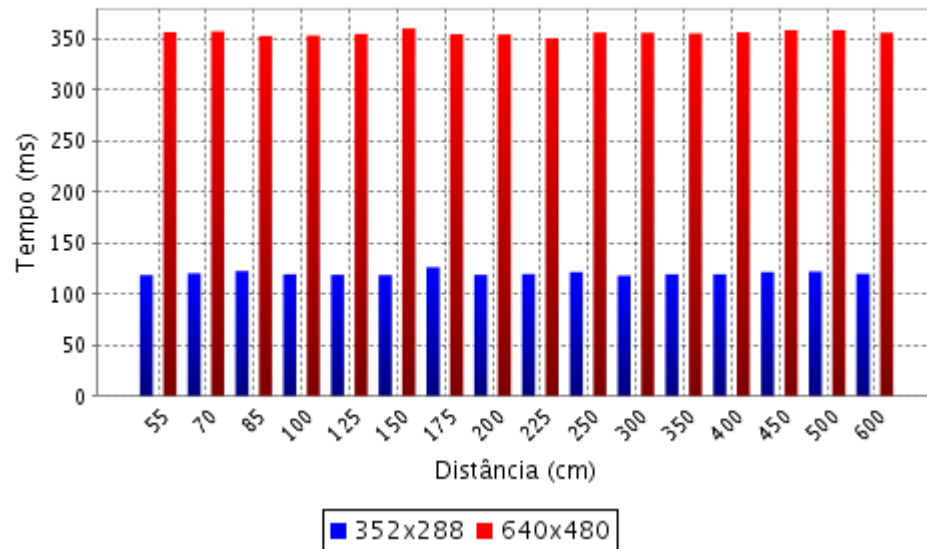
Como podemos ver nos gráficos abaixo o tempo de execução da resolução 640x480 é 3x maior que o tempo de execução da resolução 352x288, isso já era previsto, pois a quantidade de pixels da primeira é 3x maior e o tempo de execução do algoritmo que gera a disparidade é proporcional a quantidade de pixels na imagem. Podemos ver também nos gráficos que a distância não afeta o tempo de execução, que na média se manteve constante independente da distância dos obstáculos.

Figura 44 – Tempo de execução da captura das duas imagens
Tempo médio de execução captura de imagens



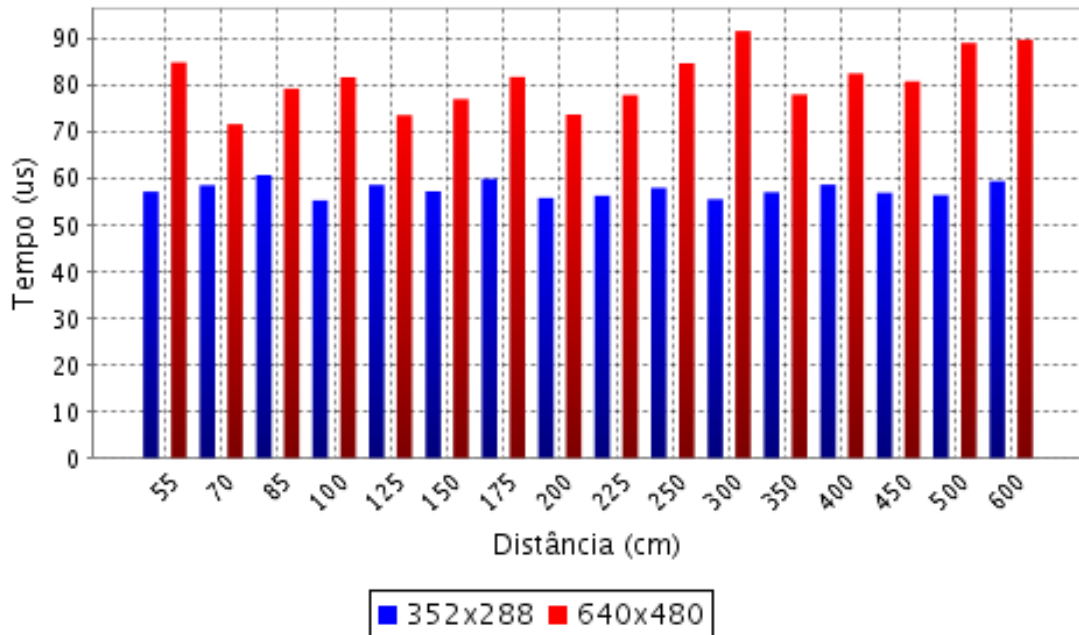
Fonte: Autor.

Figura 45 – Tempo de execução do algoritmo de geração do mapa de disparidade
Tempo médio de execução geração do mapa de disparidade



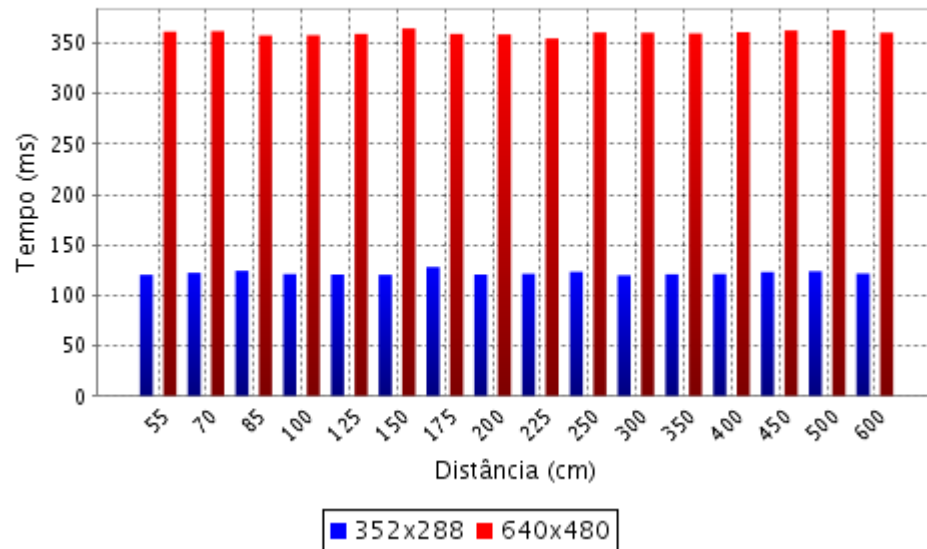
Fonte: Autor.

Figura 46 – Tempo de execução do cálculo, codificação e envio das intensidades
Tempo médio de execução cálculo, codificação e envio de intensidades



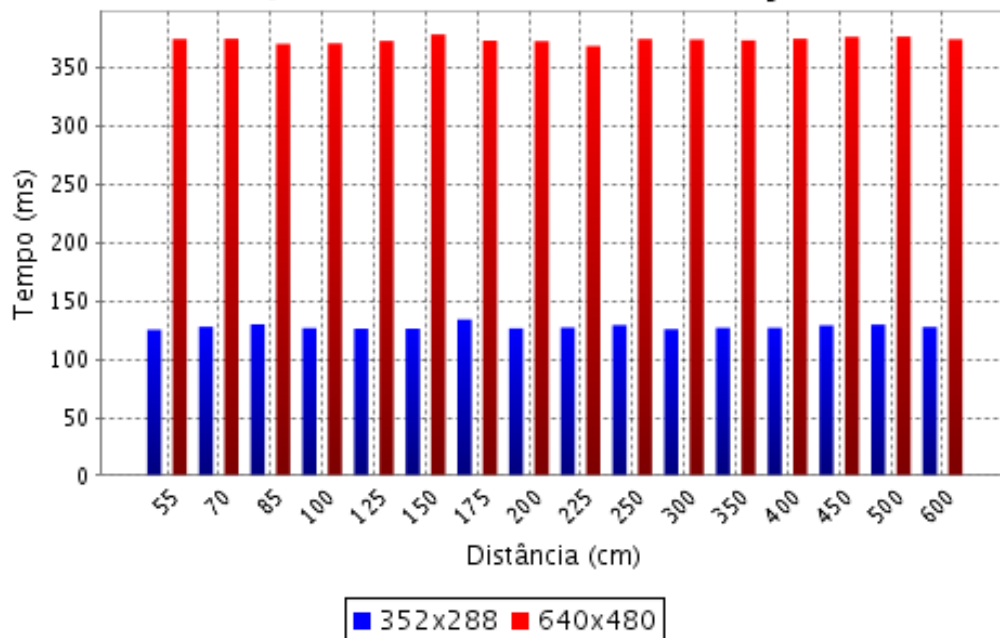
Fonte: Autor.

Figura 47 – Tempo total de execução do módulo de detecção de obstáculos
Tempo médio de execução de detecção de obstáculos



Fonte: Autor.

Figura 48 – Tempo total de execução do processamento do fluxo inteiro do sistema
Tempo médio total de execução



Fonte: Autor.

A partir dos resultados foi possível verificar que o maior tempo de processamento gasto é na etapa de geração do mapa de disparidade, em média 95% do tempo gasto desde a captura das imagens até o envio das intensidades para o módulo informativo é gasto na geração do mapa

de disparidade, independente da resolução escolhida. Isso abre a oportunidade de uma possível otimização através da paralelização dessa etapa.

Conseguimos verificar também que o tempo de execução de 1 par de frames (imagem esquerda e direita) leva em média 120 ms para a resolução 352x288 e 3x mais para a resolução 640x480. Com isso através da resolução 352x288 conseguimos uma taxa de frames maior e um tempo de resposta menor o que é de extrema importância para um sistema que deve ser o mais próximo do tempo real possível. A taxa de frames atingida pela resolução 352x288 foi de aproximadamente 8 frames por segundo, já na resolução 640x480 o valor foi inferior a 3 frames por segundo. Neste trabalho optou-se então por utilizar a menor resolução de 352x288 no sistema.

A taxa de frames por segundo pode não ser tão relevante assim se levarmos em conta que a atualização dos motores vibratórios, pois se atualizarmos 8 vezes em menos de um segundo possivelmente o usuário não consiga diferenciar as mudanças, mas ela se torna relevante na questão de não inserirmos um delay significativo entre a captura e o envio da informação, tentando minimizar ao máximo esse atraso.

4.3 TESTES DE INTEGRAÇÃO

Os testes de integração foram realizados no intuito de testar todas as partes desenvolvidas, para isso o sistema foi posicionado em frente a alguns cenários que pudessem exercitar diversas situações e todo o fluxo do sistema.

Em todos os testes a seguir foram realizadas as mesmas etapas:

- **1. Captura de imagens:** onde as imagens são capturadas (já na resolução 352x288) pelo par de câmeras acopladas ao óculos, após a captura elas são pré-processadas (mudança de colorida para escala de cinza, remoção de distorções e retificação), isso é realizado no módulo de captura e processamento de imagens. Na tabela abaixo ela é representada apenas pela imagem da câmera esquerda.
- **2. Mapa de disparidade:** nessa etapa o par de imagens retificados na etapa anterior é usado para gerar o mapa de disparidade. O módulo de identificação de obstáculos.
- **3. Segmentação do mapa de disparidade, cálculo dos histogramas e cálculo de intensidades:** na tabela abaixo as 3 tarefas foram agrupadas apenas para fins ilustrativos, mas elas são realizadas serialmente conforme a Figura 18. Dado o mapa de disparidade, ele é segmentado em 9 regiões e para cada região é calculado os histogramas para identificar a maior incidência dos pixels e com isso identificar






possíveis objetos/obstáculos. Após o cálculo os histogramas são convertidos em intensidades. Essas etapas são realizadas no módulo de identificação de obstáculos.







- **4. Codificação:** nessa etapa os valores de cada um dos histogramas são convertidos para um valor inteiro que é enviado para o Arduíno que é o controlador do sistema vibratório. Na tabela é exibido o valor inteiro da codificação e o valor hexadecimal. Essa etapa é realizada pelo módulo de identificação de obstáculos.
- **5. Vibração:** dado a dificuldade em mostrar o sistema vibrando, optou-se por medir a corrente atuando em cada um dos motores vibratórios, para isso colocou-se um multímetro em série. Essa etapa é realizada pelo módulo informativo vibrátil.

4.3.1 Resultados

Os resultados de cada teste são comentados após as suas respectivas tabelas.












Teste 1 – Objeto distante entre 1 m e 1,5 m e localizado na maior parte a esquerda.

1. Captura das imagens					2. Mapa de disparidade										
															
3. Segmentação do Mapa de disparidade, cálculo dos histogramas e intensidade para cada segmento															
1)					2)					3)					
	Far	Medium	Near			Far	Medium	Near			Far	Medium	Near		
H 1	8661	300	2271		H 1	10557	0	675		H 1	10988	33	211		
H 2	8282	32	2271		H 2	7405	0	675		H 2	6178	2	146		
	346	242	0				2773	0	0				4763	27	65
	33	26	0				379	0	0				47	4	0
Intensity	200				Intensity	200				Intensity	0				

4)		5)		6)																																																																			
	<table border="1"> <thead> <tr> <th></th> <th>Far</th> <th>Medium</th> <th>Near</th> </tr> </thead> <tbody> <tr> <td>H 1</td> <td>2509</td> <td>313</td> <td>8410</td> </tr> <tr> <td rowspan="3">H2</td> <td>2154</td> <td>135</td> <td>8410</td> </tr> <tr> <td>188</td> <td>84</td> <td>0</td> </tr> <tr> <td>167</td> <td>94</td> <td>0</td> </tr> <tr> <td>Intensity</td> <td colspan="3">200</td> </tr> </tbody> </table>		Far	Medium	Near	H 1	2509	313	8410	H2	2154	135	8410	188	84	0	167	94	0	Intensity	200				<table border="1"> <thead> <tr> <th></th> <th>Far</th> <th>Medium</th> <th>Near</th> </tr> </thead> <tbody> <tr> <td>H 1</td> <td>6748</td> <td>733</td> <td>3751</td> </tr> <tr> <td rowspan="3">H2</td> <td>2148</td> <td>0</td> <td>3673</td> </tr> <tr> <td>4386</td> <td>709</td> <td>42</td> </tr> <tr> <td>214</td> <td>24</td> <td>36</td> </tr> <tr> <td>Intensity</td> <td colspan="3">200</td> </tr> </tbody> </table>		Far	Medium	Near	H 1	6748	733	3751	H2	2148	0	3673	4386	709	42	214	24	36	Intensity	200				<table border="1"> <thead> <tr> <th></th> <th>Far</th> <th>Medium</th> <th>Near</th> </tr> </thead> <tbody> <tr> <td>H 1</td> <td>11171</td> <td>52</td> <td>9</td> </tr> <tr> <td rowspan="3">H 2</td> <td>1357</td> <td>22</td> <td>9</td> </tr> <tr> <td>9732</td> <td>30</td> <td>0</td> </tr> <tr> <td>82</td> <td>0</td> <td>0</td> </tr> <tr> <td>Intensity</td> <td colspan="3">20</td> </tr> </tbody> </table>		Far	Medium	Near	H 1	11171	52	9	H 2	1357	22	9	9732	30	0	82	0	0	Intensity	20		
	Far	Medium	Near																																																																				
H 1	2509	313	8410																																																																				
H2	2154	135	8410																																																																				
	188	84	0																																																																				
	167	94	0																																																																				
Intensity	200																																																																						
	Far	Medium	Near																																																																				
H 1	6748	733	3751																																																																				
H2	2148	0	3673																																																																				
	4386	709	42																																																																				
	214	24	36																																																																				
Intensity	200																																																																						
	Far	Medium	Near																																																																				
H 1	11171	52	9																																																																				
H 2	1357	22	9																																																																				
	9732	30	0																																																																				
	82	0	0																																																																				
Intensity	20																																																																						
7)		8)		9)																																																																			
	<table border="1"> <thead> <tr> <th></th> <th>Far</th> <th>Medium</th> <th>Near</th> </tr> </thead> <tbody> <tr> <td>H 1</td> <td>4727</td> <td>69</td> <td>6436</td> </tr> <tr> <td rowspan="3">H2</td> <td>3786</td> <td>0</td> <td>6436</td> </tr> <tr> <td>701</td> <td>0</td> <td>0</td> </tr> <tr> <td>240</td> <td>69</td> <td>0</td> </tr> <tr> <td>Intensity</td> <td colspan="3">200</td> </tr> </tbody> </table>		Far	Medium	Near	H 1	4727	69	6436	H2	3786	0	6436	701	0	0	240	69	0	Intensity	200				<table border="1"> <thead> <tr> <th></th> <th>Far</th> <th>Medium</th> <th>Near</th> </tr> </thead> <tbody> <tr> <td>H 1</td> <td>4370</td> <td>1758</td> <td>5104</td> </tr> <tr> <td rowspan="3">H 2</td> <td>68</td> <td>1758</td> <td>5104</td> </tr> <tr> <td>1119</td> <td>0</td> <td>0</td> </tr> <tr> <td>3183</td> <td>0</td> <td>0</td> </tr> <tr> <td>Intensity</td> <td colspan="3">200</td> </tr> </tbody> </table>		Far	Medium	Near	H 1	4370	1758	5104	H 2	68	1758	5104	1119	0	0	3183	0	0	Intensity	200				<table border="1"> <thead> <tr> <th></th> <th>Far</th> <th>Medium</th> <th>Near</th> </tr> </thead> <tbody> <tr> <td>H 1</td> <td>8497</td> <td>2735</td> <td>0</td> </tr> <tr> <td rowspan="3">H 2</td> <td>70</td> <td>2735</td> <td>0</td> </tr> <tr> <td>2238</td> <td>0</td> <td>0</td> </tr> <tr> <td>6189</td> <td>0</td> <td>0</td> </tr> <tr> <td>Intensity</td> <td colspan="3">100</td> </tr> </tbody> </table>		Far	Medium	Near	H 1	8497	2735	0	H 2	70	2735	0	2238	0	0	6189	0	0	Intensity	100		
	Far	Medium	Near																																																																				
H 1	4727	69	6436																																																																				
H2	3786	0	6436																																																																				
	701	0	0																																																																				
	240	69	0																																																																				
Intensity	200																																																																						
	Far	Medium	Near																																																																				
H 1	4370	1758	5104																																																																				
H 2	68	1758	5104																																																																				
	1119	0	0																																																																				
	3183	0	0																																																																				
Intensity	200																																																																						
	Far	Medium	Near																																																																				
H 1	8497	2735	0																																																																				
H 2	70	2735	0																																																																				
	2238	0	0																																																																				
	6189	0	0																																																																				
Intensity	100																																																																						
4. Codificação																																																																							
13158401 (0xC8C801)		348702723 (0x14C8C803)		1690880007 (0x64C8C807)																																																																			
5. Vibração																																																																							
150 mA (Motor 1)		150 mA (Motor 2)		0 (Motor 3)																																																																			
150 mA (Motor 4)		150 mA (Motor 5)		15 mA (Motor 6)																																																																			
150 mA (Motor 7)		150 mA (Motor 8)		75 mA (Motor 9)																																																																			






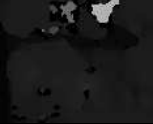


No teste 1 podemos ver através do mapa de disparidade que o objeto foi identificado e se encontra localizado mais a esquerda e também ao centro. Através do cálculo de histogramas isso foi validado e podemos ver que para os segmentos 1, 2, 4, 5, 7 e 8 foi calculado o valor 200 de intensidade informando que o objeto se encontra próximo (valores 200, 220 ou 240 representam objetos próximos). Um detalhe importante é o segmento 9 que foi calculado o valor 100 representando que havia algo a uma distância média, mas na foto podemos ver que não havia nada, o chão foi identificado como um obstáculo, isso fica mais evidente nos testes posteriores com objetos mais distantes.

Teste 2 – Objeto distante entre 1 m e 1,5 m e localizado na maior parte no centro.

Captura das imagens				Mapa de disparidade			
							
Segmentação do Mapa de disparidade, cálculo dos histogramas e intensidade para cada segmento							
1)				2)			
	Far	Medium	Near		Far	Medium	Near
H 1	10932	6	294	H 1	8524	0	2708
H 2	8067	0	227	H 2	6329	0	2646
	1316	6	67		1634	0	62
	1549	0	0		561	0	0
Intensity	0			Intensity	200		
3)				4)			
	Far	Medium	Near		Far	Medium	Near
H 1	9200	254	1778	H 1	10715	11	506
H 2	5634	42	1614	H 2	2131	0	502
	3529	200	154		2830	1	4
	37	12	10		5754	10	0
Intensity	200			Intensity	40		
5)				6)			
	Far	Medium	Near		Far	Medium	Near
H 1	1178	189	9865	H 1	5659	246	5327
H 2	1080	0	9714	H 2	652	0	5327
	5	12	151		4953	160	0
	93	177	0		54	86	0
Intensity	200			Intensity	200		
7)				8)			
	Far	Medium	Near		Far	Medium	Near
H 1	8307	2750	175	H 1	565	251	10416
H 2	949	2750	175	H 2	565	209	10416
	847	0	0		0	25	0
	6511	0	0		0	17	0
Intensity	100			Intensity	200		
9)				Codificação			
	Far	Medium	Near	3368550401 (0xC8C80001)		3368560643 (0xC8C82803)	
H 1	3539	1413	6280	3368576007 (0xC8C86407)			
H 2	131	1413	6280				
	1042	0	0				
	2366	0	0				
Intensity	200						
Vibração							
0 mA (Motor 1)		150 mA (Motor 2)		150 mA (Motor 3)			
30 mA (Motor 4)		150 mA (Motor 5)		150 mA (Motor 6)			
75 mA (Motor 7)		150 mA (Motor 8)		150 mA (Motor 9)			

No teste 2 podemos ver através do mapa de disparidade que o objeto foi identificado e se encontra localizado mais ao centro e a direita. Através do cálculo de histogramas isso foi validado e podemos ver que para os segmentos 2, 3, 5, 6, 8 e 9 foi calculado o valor 200 de intensidade informando que o objeto se encontra próximo (valores 200, 220 ou 240 representam objetos próximos). Como no teste 1 o chão foi identificado como um obstáculo no segmento 7 que foi calculado o valor 100.



Teste 3 – Objeto distante entre 1 m e 1,5 m e localizado na maior parte a direita.

Captura das imagens					Mapa de disparidade									
														
Segmentação do Mapa de disparidade, cálculo dos histogramas e intensidade para cada segmento														
1)					2)					3)				
	Far	Medium	Near			Far	Medium	Near			Far	Medium	Near	
H 1	11186	0	46		H 1	11132	0	100		H 1	9411	204	1617	
H 2	7707	0	38		H 2	7037	0	100		H 2	5729	13	1546	
	8972	0	8			2772	0	0			3603	1	71	
	2582	0	0			1323	0	0			79	190	0	
Intensity	0				Intensity	0				Intensity	200			
4)					5)					6)				
	Far	Medium	Near			Far	Medium	Near			Far	Medium	Near	
H 1	10823	77	332		H 1	11155	35	42		H 1	3719	866	6647	
H 2	1766	17	332		H 2	2046	0	42		H 2	1545	29	6647	
	3728	60	0			5363	35	0			2112	132	0	
	5329	0	0			3746	0	0			62	705	0	
Intensity	40				Intensity	20				Intensity	200			

7)				8)				9)			
	Far	Medium	Near		Far	Medium	Near		Far	Medium	Near
H 1	8349	2883	0	H 1	8032	3200	0	H 1	2567	2620	6045
H 2	713	2883	0	H 2	0	3200	0	H 2	1002	801	6045
	1380	0	0		1557	0	0		380	79	0
	6256	0	0		6475	0	0		1185	1740	0
Intensity	100			Intensity	100			Intensity	200		
Codificação											
3355443201 (0xC8000001)				3356764163 (0xC8142803)				3362022407 (0xC8646407)			
Vibração											
0 mA (Motor 1)				0 mA (Motor 2)				150 mA (Motor 3)			
30 mA (Motor 4)				15 mA (Motor 5)				150 mA (Motor 4)			
75 mA (Motor 7)				75 mA (Motor 7)				150 mA (Motor 7)			

No teste 3 podemos ver através do mapa de disparidade que o objeto foi identificado e se encontra localizado totalmente a direita. Através do cálculo de histogramas isso foi validado e podemos ver que para os segmentos 3, 6 e 9 foi calculado o valor 200 de intensidade informando que o objeto se encontra próximo (valores 200, 220 ou 240 representam objetos próximos). Como nos testes anteriores o chão foi identificado como um obstáculo no segmento 7 e 8 que foi calculado o valor 100.













Teste 4 – Objeto distante entre 2m e 3m e localizado na maior parte a esquerda.

Captura das imagens				Mapa de disparidade							
											
Segmentação do Mapa de disparidade, cálculo dos histogramas e intensidade para cada segmento											
1)				2)				3)			
	Far	Medium	Near		Far	Medium	Near		Far	Medium	Near
H 1	11117	112	3	H 1	11084	0	148	H 1	11013	172	47
H 2	7942	0	2	H 2	7008	0	148	H 2	5826	25	0
	1133	112	1		3018	0	0		5065	147	0
	2042	0	0		1058	0	0		122	0	47
Intensity	0			Intensity	0			Intensity	0		

4)				5)				6)			
	Far	Medium	Near		Far	Medium	Near		Far	Medium	Near
H 1	2187	9045	0	H 1	11039	193	0	H 1	11203	29	0
H 2	697	3593	0	H 2	2477	193	0	H 2	1265	0	0
	551	5452	0		6170	0	0		9826	29	0
	939	0	0		2392	0	0		112	0	0
Intensity	120			Intensity	20			Intensity	20		
7)				8)				9)			
	Far	Medium	Near		Far	Medium	Near		Far	Medium	Near
H 1	2319	8913	0	H 1	7668	3564	0	H 1	8580	2652	0
H 2	984	5216	0	H 2	0	3564	0	H 2	71	2652	0
	212	3697	0		1979	0	0		2096	0	0
	1123	0	0		5689	0	0		6413	0	0
Intensity	100			Intensity	100			Intensity	100		
Codificação											
1 (0x1)				336885763 (0x14147803)				1684300807 (0x64646407)			
Vibração											
0 mA (Motor 1)				0 mA (Motor 2)				0 mA (Motor 3)			
90 mA (Motor 4)				15 mA (Motor 5)				15 mA (Motor 6)			
75 mA (Motor 7)				75 mA (Motor 8)				75 mA (Motor 9)			







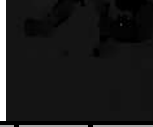

No teste 4 podemos ver através do mapa de disparidade que o objeto foi identificado e se encontra localizado mais a esquerda. Através do cálculo de histogramas isso foi validado e podemos ver que para os segmentos 3 e 7 foram calculados os valores 120 e 100 de intensidade informando que o objeto se encontra a média distância (valores 100, 120 ou 140 representam objetos a média distância). Novamente o chão foi identificado como um obstáculo conforme os valores calculados para os segmentos 8 e 9. Um detalhe positivo é que nos segmentos 1, 2 e 3 não foi identificado nenhum objeto, apesar de alguns ruídos no mapa de disparidade, mas que não se mostraram significativos.

Teste 5 – Objeto distante entre 2m e 3m e localizado na maior parte no centro.

Captura das imagens				Mapa de disparidade							
											
Segmentação do Mapa de disparidade, cálculo dos histogramas e intensidade para cada segmento											
1)				2)							
	Far	Medium	Near		Far	Medium	Near				
H 1	11232	0	0	H 1	10812	157	263				
H 2	7140	0	0	H 2	6160	22	263				
	611	0	0		2175	135	0				
	3481	0	0		2477	0	0				
Intensity	0			Intensity	0						
3)				4)							
	Far	Medium	Near		Far	Medium	Near				
H 1	11054	168	10	H 1	10966	74	192				
H 2	5344	0	0	H 2	1106	10	192				
	5573	168	10		4168	64	0				
	137	0	0		5692	0	0				
Intensity	20			Intensity	40						
5)				6)							
	Far	Medium	Near		Far	Medium	Near				
H 1	4277	6939	16	H 1	9615	1617	0				
H 2	1069	3776	16	H 2	1222	1171	0				
	306	3163	0		8307	440	0				
	2902	0	0		86	6	0				
Intensity	100			Intensity	100						
7)				8)							
	Far	Medium	Near		Far	Medium	Near				
H 1	7817	3415	0	H 1	3047	8185	0				
H 2	843	3415	0	H 2	112	5658	0				
	431	0	0		11	2527	0				
	6543	0	0		2924	0	0				
Intensity	100			Intensity	100						
9)				9)							
	Far	Medium	Near		Far	Medium	Near				
H 1	6255	4977	0	H 1	6255	4977	0				
H 2	139	4803	0	H 2	139	4803	0				
	1288	174	0		1288	174	0				
	4828	0	0		4828	0	0				
Intensity	100			Intensity	100						
Codificação											
335544321 (0x14000001)				1684285443 (0x64642803)				1684300807 (0x64646407)			
Vibração											
0 mA (Motor 1)				0 mA (Motor 2)				15 mA (Motor 3)			
30 mA (Motor 4)				75 mA (Motor 5)				75 mA (Motor 6)			
75 mA (Motor 7)				75 mA (Motor 8)				75 mA (Motor 9)			

No teste 5 podemos ver através do mapa de disparidade que o objeto foi identificado e se encontra localizado mais ao centro, mas também um pouco a direita. Através do cálculo de histogramas isso foi validado e podemos ver que para os segmentos 5, 6, 7 e 8 foram calculados os valores 100 de intensidade informando que o objeto se encontra a média distância (valores 100, 120 ou 140 representam objetos a média distância). Como nos outros testes, novamente o chão foi identificado como um obstáculo conforme os valores calculados para o segmento 6.

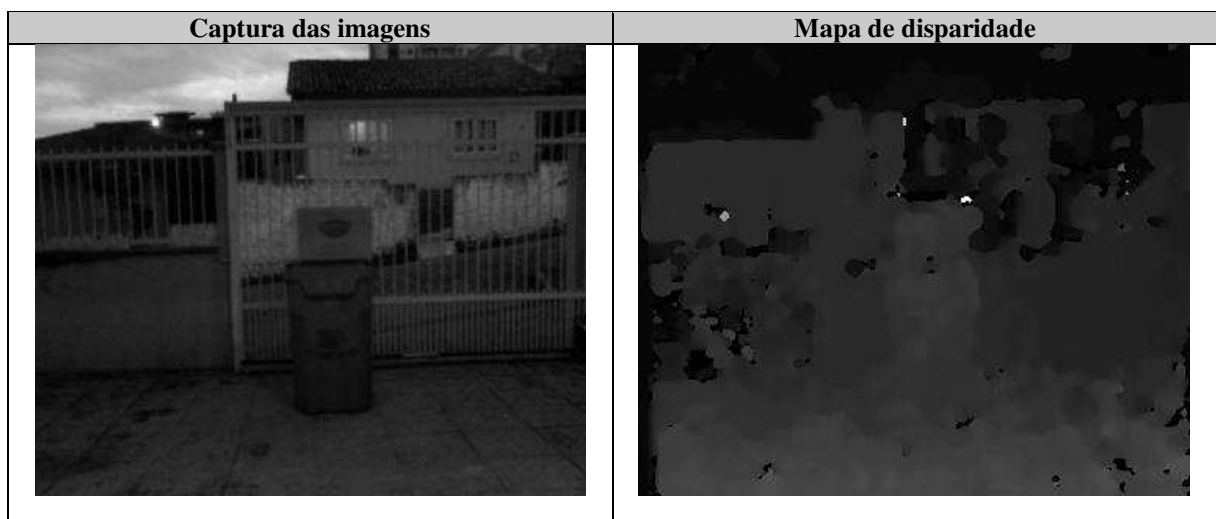
Teste 6 – Objeto distante entre 2m e 3m e localizado na maior parte a direita.

Captura das imagens				Mapa de disparidade			
							
Segmentação do Mapa de disparidade, cálculo dos histogramas e intensidade para cada segmento							
1)				2)			
	Far	Medium	Near		Far	Medium	Near
H 1	11232	0	0	H 1	10958	5	269
H 2	7196	0	0	H 2	5826	0	269
	414	0	0		1968	5	0
	3622	0	0		3164	0	0
Intensity	0			Intensity	0		
3)				4)			
	Far	Medium	Near		Far	Medium	Near
H 1	10754	428	20	H 1	11156	0	76
H 2	5233	89	0	H 2	1059	0	76
	5206	369	20		4039	0	0
	315	0	0		6058	0	0
Intensity	0			Intensity	40		
5)				6)			
	Far	Medium	Near		Far	Medium	Near
H 1	11230	0	2	H 1	3980	7252	0
H 2	1474	0	2	H 2	940	5452	0
	1524	0	0		2640	1800	0
	8232	0	0		400	0	0
Intensity	40			Intensity	100		

7)				8)				9)			
	Far	Medium	Near		Far	Medium	Near		Far	Medium	Near
H 1	8093	3139	0	H 1	7275	3957	0	H 1	3475	7757	0
H 2	860	3139	0	H 2	7	3957	0	H 2	426	7145	0
	685	0	0		111	0	0		446	574	0
	6548	0	0		7157	0	0		2603	38	0
Intensity	100			Intensity	100			Intensity	100		
Codificação											
1 (0x1)				1680353283 (0x64282803)				1684300807 (0x64646407)			
Vibração											
0 mA (Motor 1)				0 mA (Motor 2)				0 mA (Motor 3)			
30 mA (Motor 4)				30 mA (Motor 5)				75 mA (Motor 6)			
75 mA (Motor 7)				75 mA (Motor 8)				75 mA (Motor 9)			

No teste 6 podemos ver através do mapa de disparidade que o objeto foi identificado e se encontra localizado totalmente a direita. Através do cálculo de histogramas isso foi validado e podemos ver que para os segmentos 6 e 9 foram calculados os valores 100 de intensidade informando que o objeto se encontra a média distância (valores 100, 120 ou 140 representam objetos a média distância). Como nos outros testes, novamente o chão foi identificado como um obstáculo conforme os valores calculados para o segmento 7 e 8. Um detalhe positivo é que nos segmentos 1, 2 e 3 não foi identificado nenhum objeto, apesar de alguns ruídos no mapa de disparidade, mas que não se mostraram significativos.

Teste 7 – Objeto distante entre 3,5m e 6m e localizado na maior parte no centro.




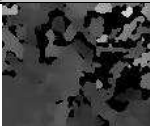
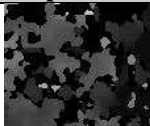








Segmentação do Mapa de disparidade, cálculo dos histogramas e intensidade para cada segmento												
1)				2)				3)				
	Far	Medium	Near		Far	Medium	Near		Far	Medium	Near	
H 1	11232	0	0	H 1	11220	0	12	H 1	11209	18	5	
H 2	7103	0	0	H 2	6712	0	12	H 2	6806	0	5	
	305	0	0		1810	0	0		4205	18	0	
	3794	0	0		2698	0	0		198	0	0	
Intensity	0			Intensity	0			Intensity	0			
4)				5)				6)				
	Far	Medium	Near		Far	Medium	Near		Far	Medium	Near	
H 1	11186	19	27	H 1	11211	0	21	H 1	11232	0	0	
H 2	1693	19	27	H 2	535	0	21	H 2	728	0	0	
	4140	0	0		809	0	0		9896	0	0	
	5353	0	0		9867	0	0		608	0	0	
Intensity	40			Intensity	40			Intensity	20			
7)				8)				9)				
	Far	Medium	Near		Far	Medium	Near		Far	Medium	Near	
H 1	7434	3798	0	H 1	6898	4334	0	H 1	6921	4311	0	
H 2	1244	3778	0	H 2	36	4334	0	H 2	316	4297	0	
	1023	20	0		25	0	0		1604	14	0	
	5167	0	0		6837	0	0		5001	0	0	
Intensity	100			Intensity	100			Intensity	100			
Codificação												
1 (0x1)				338176003 (0x14282803)				1684300807 (0x64646407)				
Vibração												
0 mA (Motor 1)				0 mA (Motor 2)				0 mA (Motor 3)				
30 mA (Motor 4)				40 mA (Motor 5)				40 mA (Motor 6)				
75 mA (Motor 7)				75 mA (Motor 8)				75 mA (Motor 9)				

No teste 7 podemos ver através do mapa de disparidade que o objeto foi identificado e se encontra localizado totalmente ao centro, mas também podemos ver que o chão foi identificado como um obstáculo em toda a parte inferior da imagem. Através do cálculo de histogramas isso foi validado e podemos ver que para o segmento 5 foi calculado o valor 40 de intensidade informando que o objeto se encontra a longa distância (valores 0, 20 ou 40 representam objetos a longa distância). Como nos outros testes, novamente o chão foi identificado como um obstáculo conforme os valores calculados para o segmento 7, 8 e 9, mas

dessa vez como o objeto se encontra distante das câmeras o chão foi identificado como um obstáculo em toda a região inferior da imagem. Um detalhe positivo é que nos segmentos 1,2 e 3 não foi identificado nenhum objeto, apesar de alguns ruídos no mapa de disparidade, mas que não se mostraram significativos.

Teste 8 – Objeto distante entre 2m e 3m e localizado na esquerda e na direita.

Captura das imagens				Mapa de disparidade							
											
Segmentação do Mapa de disparidade, cálculo dos histogramas e intensidade para cada segmento											
											
	Far	Medium	Near		Far	Medium	Near		Far	Medium	Near
H 1	5094	6138	0	H 1	10464	768	0	H 1	10700	532	0
H2	1467	5957	0	H 2	2901	501	0	H 2	5797	454	0
	2258	181	0		4471	223	0		2095	50	0
	1369	0	0		3092	44	0		2808	28	0
Intensity	100			Intensity	100			Intensity	0		
											
	Far	Medium	Near		Far	Medium	Near		Far	Medium	Near
H 1	4507	6725	0	H 1	11232	0	0	H 1	9349	1808	75
H2	342	6663	0	H 2	80	0	0	H 2	1424	204	75
	2072	62	0		6662	0	0		5273	874	0
	2093	0	0		4490	0	0		2652	730	0
Intensity	100			Intensity	20			Intensity	120		
											
	Far	Medium	Near		Far	Medium	Near		Far	Medium	Near
H 1	2407	8825	0	H 1	6714	4518	0	H 1	3865	6748	619
H2	549	8451	0	H 2	308	4346	0	H 2	336	2838	619
	119	374	0		157	172	0		10	1683	0
	1739	0	0		6249	0	0		3519	2227	0
Intensity	100			Intensity	100			Intensity	200		

Codificação		
6579201 (0x646401)	2014602243 (0x78146403)	3362022407 (0xC8646407)
Vibração		
75 mA (Motor 1)	75 mA (Motor 2)	0 mA (Motor 3)
75 mA (Motor 4)	15 mA (Motor 5)	90 mA (Motor 6)
75 mA (Motor 7)	75 mA (Motor 7)	150 mA (Motor 7)

No teste 8 podemos ver através do mapa de disparidade que existem dois objetos um localizado a esquerda e outro localizado a direita e que o obstáculo da direita se encontra mais próximo que o da esquerda. Através do cálculo de histogramas isso foi validado e podemos ver que para o objeto a esquerda os segmentos 1, 3 e 7 tiveram calculados os valores 100 de intensidade informando que o objeto se encontra a média distância (valores 100, 120 ou 140 representam objetos a longa distância). Já para o objeto a direita ele foi identificado nos segmentos 6 e 9, no segmento 6 ele se encontra a média distância (valor 120 de intensidade) o que é realmente verdade, pois aquela parte da traseira do carro está praticamente alinhada ao pilar, o segmento 9 se encontra próximo (valor de intensidade 200). Como nos outros testes, novamente o chão foi identificado como um obstáculo conforme o valor calculado para o segmento 8. Nessa imagem foi identificado muito ruído no segmento 2, acredito que isso tenha sido causado pela baixa iluminação (o teste foi realizado a noite) e devido a proximidade do muro.

Conclusão geral:

Através dos testes foi possível verificar que o sistema atua conforme o especificado, ele é capaz de detectar obstáculos em diferentes distâncias e transmitir para o dispositivo vibrátil essas informações. Também foi identificado que o chão deve ser tratado de uma forma diferente, pois do modo que o sistema foi projetado ele é identificado como um obstáculo quando na verdade não é.

4.4 DIFICULDADES E PROBLEMAS

Durante o desenvolvimento do projeto foram encontrados alguns problemas, tais como a dificuldade de sincronizar as câmeras o que causa um *delay* entre a captura das imagens e isso gera problemas na criação de mapas de disparidades em cenas que algum objeto se move muito rápido. Como não há sincronismo entre as câmeras é possível que um objeto esteja em distâncias diferentes em cada uma das imagens fazendo com que o mapa de disparidade pode

informar que o objeto está mais próximo ou distante do que ele realmente está. Como o projeto tem como premissa utilizar uma solução de baixo custo, isso dificultou bastante, pois algumas câmeras comerciais para visão computacional já possuem esse mecanismo em hardware. Outro problema com as câmeras foi na construção do dispositivo estéreo, como foram utilizadas duas câmeras e elas precisam ficar o mais coplanares é muito difícil fazer isso manualmente, acredito que isso também possa ter afetado na qualidade da calibração e conseqüentemente nos resultados encontrados.

Como as câmeras utilizadas são de baixo custo a qualidade delas deixou a desejar em questões relacionadas com iluminação, controle de estabilidade e foco.

Outro grande problema é o ajuste fino dos parâmetros utilizados pelo algoritmo de geração do mapa de disparidade. Eles são bastante sensíveis e para cada condição de ambiente se mostram diferentes. Ao encontrarmos o valor que gerava resultados satisfatórios eles não foram mais alterados.

5 CONSIDERAÇÕES FINAIS

O objetivo principal de construir um protótipo de baixo custo de uma ferramenta de auxílio para pessoas com deficiência visual foi atingido, ainda que com algumas limitações, o protótipo foi construído com sucesso e foi possível verificar que a visão computacional pode auxiliar pessoas com limitações visuais. O sistema conseguiu identificar objetos localizados dentro de uma distância razoável e vibrar de modo que fosse possível identificar a proximidade.

Como a distância máxima e mínima são dependentes de alguns parâmetros, tais como a distância entre as câmeras e a disparidade máxima utilizada no algoritmo que gera o mapa de disparidade, para esse trabalho encontrou-se os seguintes valores: distância mínima 55 cm para a resolução 640x480 e 40 cm para a resolução 352x288, já a distância máxima foi considerada 500 cm para ambas as resoluções.

A qualidade das câmeras influenciaram significativamente no resultado do protótipo, mas os maiores problemas foram relacionados ao não sincronismo entre a captura das imagens de ambas as câmeras, como há um delay que algumas vezes se mostrou significativo o par de imagens nem sempre é casado causando muito ruído no mapa de disparidade e consequentemente na identificação de obstáculos. A construção manual do sistema de câmeras estéreo é outro problema, pois é extremamente difícil conseguir que elas sejam coplanares e isso causa problema na calibração e retificação das imagens. A utilização de câmeras melhores poderiam ajudar mas estavam fora do escopo desse trabalho.

Uma das limitações encontradas foram relativas a detecção do solo como um obstáculo, isso pode ser contornado com uma mudança na inclinação do ângulo das câmeras, seja mudando fisicamente o acoplamento das câmeras ou o usuário inclinando um pouco mais a cabeça. Também pode ser adicionado um algoritmo que detecte o solo e remova ou minimize sua detecção.

Foi possível validar que informações visuais relativas a proximidade de obstáculos podem ser passadas através de um sistema vibratório. A utilização do Arduíno acelerou bastante o processo de desenvolvimento desse módulo. Assim como a biblioteca OpenCV foi de extrema importância para todo o desenvolvimento relacionado a visão computacional. Com isso verificou-se que mesmo sistemas de baixo custo, com soluções abertas tanto de software como de hardware, podem contribuir muito no aprendizado e na vida das pessoas como produtos reais.

Por questões de limitação de tempo, não foi possível realizar testes do dispositivo sendo utilizado por um deficiente visual. Por isso alguns *feedbacks* ficaram faltando, tais como, a

usabilidade do dispositivo, se a região do braço é realmente uma boa opção para utilizar o dispositivo informativo.

5.1 TRABALHOS FUTUROS

Com os resultados obtidos através desse trabalho verificou-se que é possível desenvolver ferramentas assistivas utilizando visão computacional, mas dado o escopo e as limitações do trabalho não foi possível explorar alternativas mais robustas, tais como:

- Utilizar câmeras de visão estéreo de fábrica e com sincronismo via hardware o que facilitaria os problemas encontrados na calibração e no delay entre a captura das imagens. Também poderia ser utilizado câmeras que geram automaticamente o mapa de disparidade, conhecidas como RGB-D. Isso colocaria o foco em desenvolver técnicas para detectar o chão ou melhorias na detecção de obstáculos, por exemplo.
- Como foi verificado nos testes 95% do tempo de execução, desde a captura das imagens até o envio ao dispositivo informativo, são gastos na geração do mapa de disparidade para otimizar isso poderiam ser utilizadas técnicas de paralelização dos algoritmos de geração do mapa de disparidade ou até mesmo de detecção de obstáculos. A própria OpenCV disponibiliza uma versão paralela do algoritmo SGBM via CUDA.

REFERÊNCIAS

- ARDUÍNO. **Compare board specs**. Disponível em: <<http://www.arduino.cc/en/Guide/Introduction>>. Acesso em: 30 de outubro de 2016.
- ARDUÍNO. **Products**. Disponível em: <<http://www.arduino.cc/en/Main/Products>>. Acesso em: 30 de outubro de 2016.
- ARDUÍNO. **What is arduino?**. Disponível em: <<http://www.arduino.cc/en/Guide/Introduction>>. Acesso em: 30 de outubro de 2016.
- BERSCH, R. C. R. Design de um serviço de tecnologia assistiva em escolas públicas. 2009.
- BERSCH, R. C. R. Introdução à tecnologia assistiva. **Porto Alegre: CEDI**, 2008.
- BEZERRA, J. P. A. **Um sistema de visão para navegação robusta de uma plataforma robótica semi-autônoma**. 2007.
- BRADSKI, G.; KAEHLER, A. **Learning OpenCV: Computer vision with the OpenCV library**. O'Reilly Media, Inc., 2008.
- BRADSKI, G.; KAEHLER, A. **Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library**. O'Reilly Media, Inc., 2016.
- BRASIL. Subsecretaria Nacional de Promoção dos Direitos da Pessoa com Deficiência. Comitê de Ajudas Técnicas. **Tecnologia assistiva**. Brasília, DF: CORDE, 2009c. Disponível em: <<http://www.pessoacomdeficiencia.gov.br/app/sites/default/files/publicacoes/livro-tecnologia-assistiva.pdf>>. Acesso em: 15 de maio de 2017.
- BROWN, M. Z.; BURSCHKA, D.; HAGER, G. D. **Advances in Computational Stereo**. IEEE Trans. Pattern Anal. Mach. Intell. 2003. 25(8), 993–1008.
- BUENO, G. A. **Orientação e mobilidade na habilitação de deficientes visuais**. Revista da Faculdade de Educação, [S.l.], v. 18, n. 2, p. 205-215, dec. 1992. ISSN 1806-9274. Disponível em: <<http://www.revistas.usp.br/rfe/article/view/33494>>. Acesso em: 12 de maio 2016.
- COSTA, P. et al. **Obstacle detection and avoidance module for the blind**. In: World Automation Congress (WAC), 2016. IEEE, 2016. p. 1-6.
- CYGANEK, B.; SIEBERT, J. P. **An introduction to 3D computer vision techniques and algorithms**. John Wiley & Sons, 2011.
- DAWSON-HOWE, K. **A Practical Introduction to Computer Vision with OpenCV**. Somerset, GB: Wiley, 2014.
- DINIZ, W. F. S. **Acionamento de dispositivos robóticos através de interface natural em realidade aumentada**. Biblioteca Digital da UNICAMP. Campinas. 2012. Disponível em: <<http://www.bibliotecadigital.unicamp.br/document/?code=000858895>>. Acesso em: 05 Julho de 2016.

GALVÃO FILHO, T. A. **Tecnologia Assistiva para uma escola inclusiva: apropriação, demandas e perspectivas.** 2009. 346f. Tese (Doutorado em educação) – Faculdade de Educação, Universidade Federal da Bahia, 2009.

HARTLEY, R.; ZISSERMAN, A. **Multiple View Geometry in Computer Vision.** (2nd ed.). New York, NY, USA: Cambridge University Press.

HERSH, A. M.; JOHNSON A. M. **Assistive Technology for Visually Impaired and Blind People.** Springer, 2008.

HOGUE, A; JENKIN, M. R.M. **Active Stereo Vision** In K. Ikeuchi (Ed.), *Computer Vision: A Reference Guide* (pp. 8–12). Boston, MA: Springer US. 2014

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA – IBGE. Disponível em: <<http://www.sidra.ibge.gov.br/bda/popul/default.asp?t=3&z>>. Acesso em: 12 de maio de 2016.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *International Standard ISO 9999: 2007 (E) Assistive products for persons with disability—Classification and terminology* 4th ed. 2007.

JAHNE, B. (Ed.). **Computer vision and applications: a guide for students and practitioners.** Academic Press, 2000.

KLETTE, R. **Concise computer vision.** Springer, London, 2014.

KOCH, R. **Depth Estimation.** In K. Ikeuchi (Ed.), *Computer Vision: A Reference Guide* (pp. 183–186). Boston, MA: Springer US. 2014

LEE, Y. H.; MEDIONI, G. **RGB-D camera based wearable navigation system for the visually impaired.** *Computer Vision and Image Understanding*, v. 149, p. 3-20, 2016.

LU, K, et al. **Binocular stereo vision based on OpenCV.** *Proc. The IET International Conference on Smart and Sustainable City (ICSSC-2011)*, 2011.

MASCETTI, S. et al. **Sonification of guidance data during road crossing for people with visual impairments or blindness.** *International Journal of Human-Computer Studies*, v. 85, p. 16-26, 2016.

MASSIMO, B. **Getting started with Arduino.** Make: Books, 2011.

MENDES, C. C. T. **Navegação de robôs móveis utilizando visão estéreo.** 2012. Tese de Doutorado. Universidade de São Paulo.

ORGANIZAÇÃO MUNDIAL DA SAÚDE – OMS. **Visual impairment and blindness.** 2014. Disponível em: <<http://www.who.int/mediacentre/factsheets/fs282/en/>>. Acesso em: 12 de maio de 2016.

ORGANIZAÇÃO MUNDIAL DA SAÚDE – OMS. **Change the definition of blindness.**

2008. Disponível em: <<http://www.who.int/blindness/ChangetheDefinitionofBlindness.pdf>>. Acesso em: 18 de maio de 2017.

POGGI, M.; MATTOCCIA, S. **A wearable mobility aid for the visually impaired based on embedded 3D vision and deep learning**. In: Computers and Communication (ISCC), 2016 IEEE Symposium on. IEEE, 2016. p. 208-213.

RASHID, H. et al. **Bilingual wearable assistive technology for visually impaired persons**. In: Medical Engineering, Health Informatics and Technology (MediTec), 2016 International Conference on. IEEE, 2016. p. 1-6.

RODRIGUES, P. R.; ALVES, L. R. G. **Tecnologia Assistiva – Uma revisão do tema**. HOLOS, [S.l.], v. 6, p. 170-180, jan. 2014. ISSN 1807-1600. Disponível em: <<http://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/view/1595> >. Acesso em: 16 maio 2017.

SANZ, P.R. **ATAD: Assistive Technology for an Autonomous Displacement**. 2013.

SERRANO, J. F. V, et al. **Visión por computador**. Madrid, ES: Dykinson, 2004.

SHAH, M. **Fundamentals of computer vision**. University of Central Florida, Florida-USA. 1997.

SIEGWART, R.; NOURBAKHSI, I. R.; SCARAMUZZA, D. **Introduction to autonomous mobile robots**. MIT press, 2011.

SOLEM, J. E. **Programming Computer Vision with Python: Tools and algorithms for analyzing images**. O'Reilly Media, Inc., 2012.

STIVANELLO, M. E. et al. **Dense correspondence with regional support for stereo vision systems**. In: 2010 23rd SIBGRAPI Conference on Graphics, Patterns and Images. 2010.

STURM, P. **Pinhole camera model**. In K. Ikeuchi (Ed.), Computer Vision: A Reference Guide (pp. 610–613). Boston, MA: Springer US. 2014

SZELISKI, R. **Computer Vision: Algorithms and Applications**. New York, NY, USA: Springer-Verlag New York, Inc, 2010.

TALEB, A. et al. **As condições de saúde ocular no Brasil–2012**. Conselho Brasileiro de Oftalmologia. São Paulo, 2012.

ZHANG, Z. **Perspective Camera**. In K. Ikeuchi (Ed.), Computer Vision: A Reference Guide (pp. 590–592). Boston, MA: Springer US. 2014

ZHANG, Z. **Geometric Calibration**. In K. Ikeuchi (Ed.), Computer Vision: A Reference Guide (pp. 333–338). Boston, MA: Springer US. 2014

ZUCHEUL L. 2014. **“Room: left disparity map”**. Disponível em: <http://videoprocessing.ucsd.edu/~zuchoul/Room_gnd.png>. Acesso em: 30 Junho 2017.

APÊNDICE A – ARQUIVO XML COM OS PARÂMETROS DE CALIBRAÇÃO DAS CÂMERAS

Abaixo segue um exemplo do arquivo que o sistema salva após a calibração e retificação das câmeras. Segue também uma breve explicação de cada uma das matrizes salvas.

C1	Matriz que contém os dados intrínsecos da câmera 1.
C2	Matriz que contém os dados intrínsecos da câmera 2.
D1	Coefficientes de distorções da câmera 1
D2	Coefficientes de distorções da câmera 2
E	Matriz essencial.
F	Matriz fundamental.
R	Matriz de rotação entre o Sistema de coordenadas da primeira e segunda câmeras.
T	Vetor de translação entre os sistemas de coordenadas das câmeras.
R1	Matriz 3x3 da transformação de retificação (matriz de rotação) para a primeira câmera.
R2	Matriz 3x3 da transformação de retificação(matriz de rotação) para a segunda câmera.
P1	Matriz 3x4 de projeção no novo (retificado) Sistema de coordenadas para a primeira câmera.
P2	Matriz 3x4 de projeção no novo (retificado) Sistema de coordenadas para a segunda câmera.
Q	Matriz 4x4 de mapeamento da disparidade para profundidade.

```
<?xml version="1.0"?>
<opencv_storage>
<C1 type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    7.8574941466727068e+02 0. 3.1461823178515385e+02 0.
    7.8248345012225252e+02 2.4103608906614471e+02 0. 0. 1.</data></C1>
<C2 type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    7.8574941466727068e+02 0. 3.3012528350065190e+02 0.
    7.8248345012225252e+02 2.5442563975509080e+02 0. 0. 1.</data></C2>
<D1 type_id="opencv-matrix">
  <rows>1</rows>
  <cols>5</cols>
  <dt>d</dt>
  <data>
    2.1946475291755796e-01 -1.5341730588041840e+00 0. 0.
```



```

    5.2120826004556298e-01</data></D1>
<D2 type_id="opencv-matrix">
  <rows>1</rows>
  <cols>5</cols>
  <dt>d</dt>
  <data>
    1.5545994569565214e-01 -1.7601864028515941e+00 0. 0.
    6.1910881192215239e+00</data></D2>
<R type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    9.9974621895479809e-01 -1.3521458484998588e-02
    -1.8018541728548172e-02 1.3559511029371212e-02
    9.9990608267454073e-01 1.9913540867477083e-03 1.7989923463687454e-02
    -2.2351713341276619e-03 9.9983566982923633e-01</data></R>
<T type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -1.5351725231384382e+01 -8.9491676011720422e-01
    8.7880165825778611e-01</data></T>
<E type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    -2.8015604798655811e-02 -8.7671883126778893e-01
    -8.9651970356680721e-01 1.1547549970025928e+00
    -4.6196416305275295e-02 1.5333367755405156e+01
    6.8652775961162016e-01 -1.5362384018228919e+01
    -4.6695815763894172e-02</data></E>
<F type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    2.6563060579021832e-07 8.3473255328503436e-06 4.5835836540882007e-03
    -1.0994534994449149e-05 4.4167641335219755e-07
    -1.1135932121927536e-01 -2.4051011161577082e-03
    1.1206098178497836e-01 1.</data></F>
<R1 type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    9.9618751705380326e-01 4.4746137067747896e-02
    -7.4888010279970316e-02 -4.4632824227847132e-02
    9.9899837590899454e-01 3.1868374036289025e-03 7.4955599308021828e-02
    1.6777575922387279e-04 9.9718685810818319e-01</data></R1>
<R2 type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    9.9667904324798007e-01 5.8100621709718178e-02
    -5.7054382016067828e-02 -5.8186834492232872e-02

```

```

    9.9830569950268944e-01 1.5044008559635200e-04 5.6966455410747477e-02
    3.1698734028472639e-03 9.9837106070866521e-01</data></R2>
<P1 type_id="opencv-matrix">
  <rows>3</rows>
  <cols>4</cols>
  <dt>d</dt>
  <data>
    9.1996498265223249e+02 0. 3.8191055870056152e+02 0. 0.
    9.1996498265223249e+02 2.4553649711608887e+02 0. 0. 0. 1. 0.</data></P1>
<P2 type_id="opencv-matrix">
  <rows>3</rows>
  <cols>4</cols>
  <dt>d</dt>
  <data>
    9.1996498265223249e+02 0. 3.8191055870056152e+02
    -1.4170107951851927e+04 0. 9.1996498265223249e+02
    2.4553649711608887e+02 0. 0. 0. 1. 0.</data></P2>
<Q type_id="opencv-matrix">
  <rows>4</rows>
  <cols>4</cols>
  <dt>d</dt>
  <data>
    1. 0. 0. -3.8191055870056152e+02 0. 1. 0. -2.4553649711608887e+02 0.
    0. 0. 9.1996498265223249e+02 0. 0. 6.4922933952101608e-02 0.</data></Q>
</opencv_storage>

```

APÊNDICE B – CÓDIGO FONTE DOS MÓDULOS DESENVOLVIDOS

CÓDIGO FONTE DA COMUNICAÇÃO DO MÓDULO DE IDENTIFICAÇÃO DE OBSTÁCULOS COM O MÓDULO INFORMATIVO VIBRÁTIL

Código fonte da interface de comunicação entre o módulo de identificação de obstáculos com o módulo informativo vibrátil. Esse código foi implementado na linguagem C/C++ e pertence ao módulo de identificação de obstáculos.

- comm_detect_module.cpp

```
#include "comm_detect_module.h"
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdint.h>
#include <fcntl.h>
#include <termios.h>
#include <errno.h>
#include <sys/ioctl.h>

char * arduino_dev = NULL;
int arduino_fd;

bool get_arduino_device(void)
{
    DIR *d;
    struct dirent *dir;
    d = opendir("/dev");
    if(d){
        while ((dir = readdir(d)) != NULL){
            if(strstr(dir->d_name, "ttyS0") != NULL){
                int size = sizeof(char) * (strlen(dir->d_name) + strlen("/dev/")) + 1;
                arduino_dev = (char*)malloc(size);
                memset(arduino_dev, 0x00, size);
                snprintf(arduino_dev, size, "/dev/%s", dir->d_name);
                return true;
            }
        }
        closedir(d);
    }
    return false;
}

bool open_serial_port_and_configure(void)
{
    struct termios toptions;

    /* open serial port */
    arduino_fd = open(arduino_dev, O_RDWR | O_NOCTTY);

    /* wait for the Arduino to reboot */
    usleep(3500000);

    /* get current serial port settings */
    tcgetattr(arduino_fd, &toptions);
    /* set 57600 baud both ways */
```

```

cfsetispeed(&toptions, B57600);
cfsetospeed(&toptions, B57600);
/* 8 bits, no parity, no stop bits */
toptions.c_cflag &= ~PARENB;
toptions.c_cflag &= ~CSTOPB;
toptions.c_cflag &= ~CSIZE;
toptions.c_cflag |= CS8;
/* Canonical mode */
toptions.c_lflag |= ICANON;
/* commit the serial port settings */
tcsetattr(arduino_fd, TCSANOW, &toptions);

return true;
}

void recv_int_to_arduino(void)
{
char buf[64];
int n, i;

memset(buf, 0x00, sizeof(buf));
/* Receive string from Arduino */
n = read(arduino_fd, buf, 64);
/* insert terminating zero in the string */
buf[n] = 0;

printf("%i bytes read, buffer contains: %s\n", n, buf);
}

void send_int_to_arduino(int msg)
{
char send[11];

sprintf(send, "%d", msg);
write(arduino_fd, send, sizeof(send));
}

bool init_comm_arduino(void)
{
if (!get_arduino_device()){
printf("Nao encontrou arduino device... \n");
return false;
}

open_serial_port_and_configure();

return true;
}

```

- comm_detect_module.h

```

#ifndef INCLUDE_COMM_DETECT_MODULE_H_
#define INCLUDE_COMM_DETECT_MODULE_H_

bool get_arduino_device(void);
bool open_serial_port_and_configure(void);
void recv_int_to_arduino(void);
void send_int_to_arduino(int msg);
bool init_comm_arduino(void);

#endif /* INCLUDE_COMM_DETECT_MODULE_H_ */

```

Código fonte do módulo de captura de imagens. Esse código foi implementado na linguagem C/C++.

capture_module.cpp

```

#include "capture_module.h"
#include "common.h"
#include <chrono>

using namespace chrono;

extern command_line_params_t cmd_line_params;

static string calib_params_file_def = "./calib_640x480_params.xml";
static string calib_params_file = "./calib_352x288_params.xml";

Size board_size(9,6);
float square_size = 48.0;
string images_calib_filename = "./images_calib_list.xml";
vector<string> images_calib_list;
Size image_size;

bool save_calib_params_in_file(calib_params_t * params)
{
    FileStorage fs1(calib_params_file, FileStorage::WRITE);
    fs1 << "CM1" << params->CM1;
    fs1 << "CM2" << params->CM2;
    fs1 << "D1" << params->D1;
    fs1 << "D2" << params->D2;
    fs1 << "R" << params->R;
    fs1 << "T" << params->T;
    fs1 << "E" << params->E;
    fs1 << "F" << params->F;
    fs1 << "R1" << params->R1;
    fs1 << "R2" << params->R2;
    fs1 << "P1" << params->P1;
    fs1 << "P2" << params->P2;
    fs1 << "Q" << params->Q;
    fs1 << "map1x" << params->map1x;
    fs1 << "map1y" << params->map1y;
    fs1 << "map2x" << params->map2x;
    fs1 << "map2y" << params->map2y;

    return true;
}

bool read_string_image_list( const string& filename, vector<string>& l )
{
    l.resize(0);
    FileStorage fs(filename, FileStorage::READ);
    if( !fs.isOpened() ){
        return false;
    }
    FileNode n = fs.getFirstTopLevelNode();
    if( n.type() != FileNode::SEQ ){
        return false;
    }
    FileNodeIterator it = n.begin(), it_end = n.end();
    for( ; it != it_end; ++it ){
        l.push_back((string)*it);
    }
}

```

```

    return true;
}

bool get_calibration_params(calib_params_t *cams_params)
{
    string filename = cams_params->res_def?calib_params_file_def:calib_params_file;
    FileStorage fs(filename, FileStorage::READ);
    try
    {
        if( !fs.isOpened() ){
            cout << "File doesn't exist !" << endl;
            return false;
        }
    }
    catch(cv::Exception& e)
    {
        const char * err_msg = e.what();
        cout << "Error: " << err_msg << endl;
        return false;
    }

    fs["CM1"] >> cams_params->CM1;
    fs["CM2"] >> cams_params->CM2;
    fs["D1"] >> cams_params->D1;
    fs["D2"] >> cams_params->D2;
    fs["R"] >> cams_params->R;
    fs["T"] >> cams_params->T;
    fs["E"] >> cams_params->E;
    fs["F"] >> cams_params->F;
    fs["R1"] >> cams_params->R1;
    fs["R2"] >> cams_params->R2;
    fs["P1"] >> cams_params->P1;
    fs["P2"] >> cams_params->P2;
    fs["Q"] >> cams_params->Q;
    fs["map1x"] >> cams_params->map1x;
    fs["map1y"] >> cams_params->map1y;
    fs["map2x"] >> cams_params->map2x;
    fs["map2y"] >> cams_params->map2y;

    cout << "File " << filename << " opened !" << endl;

    return true;
}

bool stereo_calibrate(calib_params_t * cams_params)
{
    vector<vector<Point2f> > image_points_l;
    vector<vector<Point2f> > image_points_r;
    vector<vector<Point3f> > object_points;
    vector<string> good_image_list;
    vector<Point2f> corners_l;
    vector<Point2f> corners_r;

    int num_images = (int)images_calib_list.size()/2;

    object_points.resize(num_images);
    for( int i = 0; i < num_images; i++ )
    {
        for( int j = 0; j < board_size.height; j++ )
            for( int k = 0; k < board_size.width; k++ )
                object_points[i].push_back(Point3f(float(j*square_size),
                                                    float(k*square_size), 0));
    }

    Mat img_l, img_r, img_gray_l, img_gray_r;
    bool found_l = false;

```

```

bool found_r = false;
int i, j;
cout << "Num Images : " << num_images << endl;
for( i = j = 0; i < num_images; i++ )
{
    const string& filename_l = images_calib_list[i*2];
    img_l = imread(filename_l, IMREAD_COLOR);
    const string& filename_r = images_calib_list[i*2+1];
    img_r = imread(filename_r, IMREAD_COLOR);

    if(img_l.empty() || img_r.empty()){
        break;
    }

    cvtColor(img_l, img_gray_l, CV_BGR2GRAY);
    cvtColor(img_r, img_gray_r, CV_BGR2GRAY);

    image_size = img_gray_l.size();

    found_l = findChessboardCorners( img_gray_l, board_size, corners_l,
        CALIB_CB_ADAPTIVE_THRESH |
        CALIB_CB_FAST_CHECK |
        CALIB_CB_NORMALIZE_IMAGE |
        CALIB_CB_FILTER_QUADS);
    found_r = findChessboardCorners( img_gray_r, board_size, corners_r,
        CALIB_CB_ADAPTIVE_THRESH |
        CALIB_CB_FAST_CHECK |
        CALIB_CB_NORMALIZE_IMAGE |
        CALIB_CB_FILTER_QUADS);

    if (found_l && found_r)
    {
        cornerSubPix( img_gray_l, corners_l, Size(11,11),
            Size(-1,-1),
            TermCriteria(
                TermCriteria::EPS+TermCriteria::COUNT,
                4200000000, 0.000000000000001)
            );
        cornerSubPix( img_gray_r, corners_r, Size(11,11),
            Size(-1,-1),
            TermCriteria(
                TermCriteria::EPS+TermCriteria::COUNT,
                4200000000, 0.000000000000001 )
            );
        drawChessboardCorners( img_l, board_size, Mat(corners_l), found_l );
        drawChessboardCorners( img_r, board_size, Mat(corners_r), found_r );
        image_points_l.push_back(corners_l);
        image_points_r.push_back(corners_r);
    }

    imshow("image gray 1", img_l);
    imshow("image gray 2", img_r);
    int k = waitKey(10);
}

destroyAllWindows();

cams_params->CM1 = Mat(3, 3, CV_64FC1);
cams_params->CM2 = Mat(3, 3, CV_64FC1);
cams_params->D1 = Mat::zeros(8, 1, CV_64F);
cams_params->D2 = Mat::zeros(8, 1, CV_64F);

cout << "Starting Calibration..." << endl;

int flags = CALIB_FIX_K4|CALIB_FIX_K5|CALIB_FIX_K6;
double rms_l = calibrateCamera(object_points,

```

```

        image_points_l,
        image_size,
        cams_params->CM1,
        cams_params->D1,
        cams_params->R1,
        cams_params->T1,
        flags,
        TermCriteria( TermCriteria::EPS+TermCriteria::COUNT,
                     4200000000, 0.000000000000001 ));
    cout << "done with RMS_L error=" << rms_l << endl;

    double rms_r = calibrateCamera(object_points,
        image_points_r,
        image_size,
        cams_params->CM2,
        cams_params->D2,
        cams_params->R2,
        cams_params->T2,
        flags,
        TermCriteria( TermCriteria::EPS+TermCriteria::COUNT,
                     4200000000, 0.000000000000001 ));
    cout << "done with RMS_R error=" << rms_r << endl;

    int flags_calib = CV_CALIB_SAME_FOCAL_LENGTH +
        CV_CALIB_FIX_PRINCIPAL_POINT +
        CV_CALIB_USE_INTRINSIC_GUESS +
        CV_CALIB_FIX_K3 + CV_CALIB_FIX_K4 + CV_CALIB_FIX_K5;

    double rms = stereoCalibrate(object_points,
        image_points_l,
        image_points_r,
        cams_params->CM1,
        cams_params->D1,
        cams_params->CM2,
        cams_params->D2,
        image_size,
        cams_params->R,
        cams_params->T,
        cams_params->E,
        cams_params->F,
        flags_calib,
        TermCriteria( TermCriteria::COUNT+TermCriteria::EPS,
                     4000000000,0.00000000001 ));

    cout << "done with RMS error=" << rms << endl;

    cout << "Done Calibration! " << endl;
    cout << "Starting Rectification ..." << endl;

    Rect valid_roi[2];
    stereoRectify(cams_params->CM1,
        cams_params->D1,
        cams_params->CM2,
        cams_params->D2,
        image_size,
        cams_params->R,
        cams_params->T,
        cams_params->R1,
        cams_params->R2,
        cams_params->P1,
        cams_params->P2,
        cams_params->Q,
        CALIB_ZERO_DISPARITY,
        0,
        image_size,

```



```

        &valid_roi[0],
        &valid_roi[1]);

    cout << "Done Rectification !" << endl;
    cout << "Applying Undistort ..." << endl;

    initUndistortRectifyMap(cams_params->CM1,
        cams_params->D1,
        cams_params->R1,
        cams_params->P1,
        image_size,
        CV_32FC1,
        cams_params->map1x,
        cams_params->map1y);
    initUndistortRectifyMap(cams_params->CM2,
        cams_params->D2,
        cams_params->R2,
        cams_params->P2,
        image_size,
        CV_32FC1,
        cams_params->map2x,
        cams_params->map2y);

    cout << "Undistort complete !" << endl;

    return true;
}

bool is_calibrated(calib_params_t * cams_params)
{
    if(!get_calibration_params(cams_params)){
        cout << "Cameras aren't calibrated !" << endl;
        bool ok = read_string_image_list(images_calib_filename, images_calib_list);
        if(!ok || images_calib_list.empty()){
            cout << "Can not open " << images_calib_filename <<
                " or the string list is empty" << endl;
            return false;
        }
        stereo_calibrate(cams_params);
        save_calib_params_in_file(cams_params);
    }
    return true;
}

void print_cam_properties(VideoCapture &left_cam, VideoCapture &right_cam)
{
    cout << "Left - FPS: " << left_cam.get(CAP_PROP_FPS)
        << " - HEIGHT: " << left_cam.get(CAP_PROP_FRAME_HEIGHT )
        << " - WIDTH: " << left_cam.get(CAP_PROP_FRAME_WIDTH) << endl;

    cout << "Right - FPS: " << right_cam.get(CAP_PROP_FPS)
        << " - WIDTH: " << right_cam.get(CAP_PROP_FRAME_WIDTH)
        << " - HEIGHT: " << right_cam.get(CAP_PROP_FRAME_HEIGHT ) << endl;
}

bool init_cams(VideoCapture &left_cam,
    VideoCapture &right_cam,
    command_line_params_t * cmd_line_params)
{
    int width, height;

    left_cam = VideoCapture(0 + CAP_V4L2);
    right_cam = VideoCapture(1 + CAP_V4L2);

    if (!left_cam.isOpened() || !right_cam.isOpened())
    {

```

```

        cout << "Cannot open the video cam" << endl;
        return false;
    }

    print_cam_properties(left_cam, right_cam);

    if(cmd_line_params->resolution_default) {
        left_cam.set(CV_CAP_PROP_FRAME_WIDTH, width_max);
        left_cam.set(CV_CAP_PROP_FRAME_HEIGHT,height_max);
        right_cam.set(CV_CAP_PROP_FRAME_WIDTH,width_max);
        right_cam.set(CV_CAP_PROP_FRAME_HEIGHT,height_max);
        width = width_max;
        height = height_max;
    }else{
        left_cam.set(CV_CAP_PROP_FRAME_WIDTH,width_min);
        left_cam.set(CV_CAP_PROP_FRAME_HEIGHT,height_min);
        right_cam.set(CV_CAP_PROP_FRAME_WIDTH,width_min);
        right_cam.set(CV_CAP_PROP_FRAME_HEIGHT,height_min);
        width = width_min;
        height = height_min;
    }
    cmd_line_params->image_size = Size(width, height);

    cout << "Resolution - w: " << cmd_line_params->image_size.width
        << " h: " << cmd_line_params->image_size.height << endl;
    print_cam_properties(left_cam, right_cam);

    return true;
}

bool capture_images(VideoCapture &left_cam,
                    VideoCapture &right_cam,
                    Mat &img_U1, Mat &img_U2, calib_params_t cams_params)
{
    Mat img_l, img_r, img_gray_l, img_gray_r;

    left_cam.grab();
    right_cam.grab();

    left_cam.retrieve(img_l,0);
    right_cam.retrieve(img_r,0);

    cvtColor(img_l, img_gray_l, CV_BGR2GRAY);
    cvtColor(img_r, img_gray_r, CV_BGR2GRAY);

    remap(img_gray_l, img_U1, cams_params.map1x,
          cams_params.map1y, INTER_LINEAR, BORDER_TRANSPARENT);
    remap(img_gray_r, img_U2, cams_params.map2x,
          cams_params.map2y, INTER_LINEAR, BORDER_TRANSPARENT);

    return true;
}

```

- capture_module.h

```

#ifndef CAPTURE_MODULE_H_
#define CAPTURE_MODULE_H_

#include "common.h"
#include <iostream>

using namespace cv;
using namespace std;

const int width_max = 640;

```

```

const int width_min = 352;
const int height_max = 480;
const int height_min = 288;

typedef struct {
    Mat CM1 ;
    Mat CM2 ;
    Mat D1 ;
    Mat D2 ;
    Mat R ;
    Mat T ;
    Mat E ;
    Mat F ;
    Mat R1 ;
    Mat R2 ;
    Mat T1 ;
    Mat T2 ;
    Mat P1 ;
    Mat P2 ;
    Mat Q ;
    Mat map1x;
    Mat map1y;
    Mat map2x;
    Mat map2y;
    bool res_def;
}calib_params_t;

bool get_calibration_params(calib_params_t *cams_params);
bool read_string_image_list( const string& filename, vector<string>& l );
bool stereo_calibrate(calib_params_t * cams_params);
bool save_calib_params_in_file(calib_params_t * params);
bool is_calibrated(calib_params_t *cams_params);
bool init_cams(VideoCapture &cam_left, VideoCapture &cam_right, command_line_params_t *
cmd_line_params);
bool capture_images(VideoCapture &left_cam, VideoCapture &right_cam, Mat &img_U1, Mat
&img_U2, calib_params_t cams_params);

#endif /* CAPTURE_MODULE_H_ */

```

CÓDIGO FONTE DO MÓDULO DE DETECÇÃO DE OBSTÁCULOS

Código fonte do módulo de detecção de obstáculos. Esse código foi implementado na linguagem C/C++.

```

- detect_module.cpp

#include "detect_module.h"
#include "comm_detect_module.h"
#include "common.h"
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include <chrono>

using namespace cv;
using namespace std;
using namespace chrono;

const int num_rows = 3;
const int num_cols = 3;
Size roi_size;
vector<vector<int>> intesity_map({{0,20,40},{100,120,140},{200,220,240}});

```

```

sgbm_params_t sgbm_params;
Ptr<StereoSGBM> sgbm;
enum proximity { NEAR, MEDIUM, FAR };

extern command_line_params_t cmd_line_params;

bool init_detect_module(void)
{
    int ret = init_comm_arduino();
    if(ret == false){
        return false;
    }

    sgbm_params.block_size = 9;
    sgbm_params.num_disp = 96;
    sgbm_params.pre_filter_cap = 55;
    sgbm_params.min_disp = 0;
    sgbm_params.uniqueness_ratio = 12;
    sgbm_params.disp_max_diff = 25;
    sgbm_params.speckle_range = 95;
    sgbm_params.speckle_size = 82;
    sgbm_params.P1 = 240;

    sgbm = StereoSGBM::create(sgbm_params.min_disp, //int minDisparity
        sgbm_params.num_disp, //int numDisparities
        sgbm_params.block_size, //int SADWindowSize
        sgbm_params.P1, //int P1 = 0
        sgbm_params.P1* 15, //int P2 = 0
        sgbm_params.disp_max_diff, //int disp12MaxDiff = 0
        sgbm_params.pre_filter_cap, //int preFilterCap = 0
        sgbm_params.uniqueness_ratio, //int uniquenessRatio = 0
        sgbm_params.speckle_size, //int speckleWindowSize = 0
        sgbm_params.speckle_range, //int speckleRange = 0
        false); //bool fullDP = false

    return true;
}

bool segment_disp_map(Mat disp_map, vector<Mat>& list_roi)
{
    Mat roi;
    int disp_map_rows = disp_map.size().height;
    int disp_map_cols = disp_map.size().width;

    for(int i = 0; i < num_cols; i++){
        for(int j = 0; j < num_rows; j++){
            roi = disp_map( Rect(j*(disp_map_cols/num_cols) ,
                i*(disp_map_rows/num_rows),
                (disp_map_cols/num_cols) ,
                (disp_map_rows/num_rows) ));

            list_roi.push_back(roi);
        }
    }
    return true;
}

bool calc_histograms(Mat disp_map, vector<MatND>& histograms)
{
    vector<Mat> list_roi;

    segment_disp_map(disp_map, list_roi);

    int hist_size_1[] = {3}; // bin size histogram_1
    int hist_size_2[] = {9}; // bin size histogram_2
    float range_h1[] = { 0, 8, 17, 256 };

```

```

const float *ranges_h1[] = { range_h1 };
float range_h2[] = { 0, 2, 5, 8, 11, 14, 18, 50, 82, 256 };
const float *ranges_h2[] = { range_h2 };

// Calculate histogram
MatND hist_1;
MatND hist_2;

unsigned int count = 0;

while(count < list_roi.size()){
    Mat roi = list_roi.at(count);
    roi_size = roi.size();
    calcHist( &roi, 1, 0, Mat(), hist_1, 1, hist_size_1, ranges_h1, false, false );
    calcHist( &roi, 1, 0, Mat(), hist_2, 1, hist_size_2, ranges_h2, false, false );

    histograms.push_back(hist_1.clone());
    histograms.push_back(hist_2.clone());

    count++;
}

return true;
}

bool get_max_id(Mat hist, float *max, int *id)
{
    float tmp_max = hist.at<float>(0);
    int tmp_id = 0;
    if(tmp_max < hist.at<float>(1)){
        tmp_max = hist.at<float>(1);
        tmp_id = 1;
    }
    if(tmp_max < hist.at<float>(2)){
        tmp_max = hist.at<float>(2);
        tmp_id = 2;
    }
    *max = tmp_max;
    *id = tmp_id;

    return true;
}

int get_intensity(int x, int y)
{
    return intesity_map.at(x).at(y);
}

bool calc_intensity(vector<MatND>& histograms, vector<int>& intensities)
{
    int count = 0;
    float max = 0;
    int id = 0;

    while(count < histograms.size()){
        MatND hist_1 = histograms.at(count).clone();
        MatND hist_2 = histograms.at(count+1).clone();

        if(((hist_1.at<float>(NEAR)/(roi_size.height*roi_size.width)) * 100) >= 5.0){
            MatND tmp = hist_2.rowRange(6,8);
            get_max_id(tmp, &max, &id);
            intensities.push_back(get_intensity(NEAR,id));
        }else if(((hist_1.at<float>(MEDIUM)/(roi_size.height*roi_size.width)) * 100) >=
5.0){
            MatND tmp = hist_2.rowRange(3,5);
            get_max_id(tmp, &max, &id);

```

```

        intensities.push_back(get_intensity(MEDIUM,id));
    }else{
        MatND tmp = hist_2.rowRange(0,2);
        get_max_id(tmp, &max, &id);
        intensities.push_back(get_intensity(FAR,id));
    }
    count = count + 2;
}

return true;
}

void encode_intensities(vector<int>& intensities, vector<uint>& encode_intensities)
{
    uint val = 0;

    for(int i = 0; i < 3; i++){
        val = (i*i)+i+1;
        for(int j = 0; j < 3; j++){
            uint tmp = intensities.at((i*3) + j) << ((j*8) + 8);
            val = val + tmp;
        }

        encode_intensities.push_back(val);
        val = 0;
    }
}

bool send_intensities(vector<uint>intensities)
{
    for(int i = 0; i < intensities.size(); i++){
        if(!send_int_to_arduino(intensities.at(i))){
            return false;
        }
    }
    return true;
}

void show_images(Mat &img_U1, Mat &img_U2, Mat &disp8_1)
{
    if(cmd_line_params.show_view){
        namedWindow("left", 1);
        imshow("left", img_U1);
        cvMoveWindow("left",0,0);
        namedWindow("right", 1);
        imshow("right", img_U2);
        cvMoveWindow("right",645,0);
        namedWindow("disp",1);
        imshow("disp", disp8_1);
        cvMoveWindow("disp",320,320);

        waitKey(5);
    }
}

void add_padding(Mat img_U1, Mat &img_U1_b)
{
    copyMakeBorder(img_U1,img_U1_b,
0,0,sgbm_params.num_disp,0,BORDER_CONSTANT,Scalar(0));
}

Mat remove_padding(Mat img_U1_b)
{

```

```

    Rect Rec(sgbm_params.num_disp, 0, img_U1_b.size().width-sgbm_params.num_disp,
img_U1_b.size().height);
    Mat img_U1 = img_U1_b(Rect).clone();

    return img_U1;
}

bool detect_obstacle(Mat &img_U1, Mat &img_U2)
{
    Mat disp_1, disp_1_b, disp8_1;
    Mat img_U1_bord, img_U2_bord;

    add_padding(img_U1, img_U1_bord);
    add_padding(img_U2, img_U2_bord);

    sgbm->compute(img_U1_bord, img_U2_bord, disp_1_b);

    Mat disp_map_16 = disp_1_b.clone();
    disp_map_16.convertTo(disp_map_16, CV_8U);
    disp_map_16 = remove_padding(disp_map_16);
    normalize(disp_1_b, disp8_1, 0, 255, CV_MINMAX, CV_8U);

    disp8_1 = remove_padding(disp8_1);
    show_images(img_U1, img_U2, disp8_1);

    vector<MatND>histograms;
    vector<int> intensity;
    vector<uint>enc_intensities;

    calc_histograms(disp_map_16/*disp8_1*/, histograms);

    calc_intensity(histograms, intensity);

    encode_intensities(intensity, enc_intensities);

    send_intensities(enc_intensities);

    return true;
}

```

detect_module.h

```

#ifndef INCLUDE_DETECT_MODULE_H_
#define INCLUDE_DETECT_MODULE_H_

#include "opencv2/core/core.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>

using namespace cv;
using namespace std;

typedef struct {
    int min_disp;
    int num_disp;
    int block_size;
    int P1;
    int P2;
    int disp_max_diff;
    int pre_filter_cap;
    int uniqueness_ratio;
    int speckle_size;
    int speckle_range;
    bool full_DP;
}

```

```

} sgbm_params_t;

bool init_detect_module(void);
bool detect_obstacle(Mat &img_U1, Mat &img_U2);
#endif /* INCLUDE_DETECT_MODULE_H_ */

```

- main.cpp

```

#include "./include/capture_module.h"
#include "./include/detect_module.h"
#include "./include/comm_detect_module.h"
#include "common.h"
#include <stdio.h>
#include <iostream>
#include <chrono>

using namespace cv;
using namespace std;
using namespace chrono;

calib_params_t cams_params;
VideoCapture left_cam;
VideoCapture right_cam;
command_line_params_t cmd_line_params;

int get_params(int argc, char ** argv)
{
    cv::CommandLineParser parser(argc, argv,
        "{w|9|}{h|6|}{n|10|}{d|1000|}{t|1|}{V||}{S||}{R||}{C||}"
        "{@input_data|0|}");

    cmd_line_params.board_size.width = parser.get<int>( "w" );
    cmd_line_params.board_size.height = parser.get<int>( "h" );
    cmd_line_params.show_view = parser.has("V");
    cmd_line_params.save_images = parser.has("S");
    cmd_line_params.calib = parser.has("C");
    cmd_line_params.square_size = parser.get<float>("t");
    cmd_line_params.num_images_capture = parser.get<int>("n");
    cmd_line_params.delay_capture = parser.get<int>("d");
    cmd_line_params.resolution_default = parser.has("R");

    if (!parser.check()){
        parser.printErrors();
        return -1;
    }
    if(cmd_line_params.resolution_default){
        cams_params.res_def = true;
    }else{
        cams_params.res_def = false;
    }

    return 0;
}

int main( int argc, char** argv )
{
    get_params(argc, argv);

    if(!is_calibrated(&cams_params)){
        exit(-1);
    }
}

```



```
bool ok = init_cams(left_cam, right_cam, &cmd_line_params);
if(!ok){
    cout << "Init Cams Problem " << ok << "!!!!" << endl;
    exit(-1);
}
ok = init_detect_module();
if(!ok){
    cout << "Detect module Problem " << ok << "!!!!" << endl;
    exit(-1);
}
Mat img_U1, img_U2;

while(true){
    capture_images(left_cam, right_cam, img_U1, img_U2, cams_params);

    detect_obstacle(img_U1, img_U2);
}
}
```

APÊNDICE C – CÓDIGO FONTE DO MÓDULO INFORMATIVO VIBRÁTIL

Código fonte do módulo informativo vibrátil, o código foi desenvolvido na linguagem C/C++ e foram utilizadas as bibliotecas do Arduíno. Este código é executado pelo Arduíno.

```

#define ON true
#define OFF false

// variables for pattern timing
unsigned long current_ms = millis();
unsigned long previous_ms = 0;
unsigned long interval_in_ms = 100;

// variables for software PWM
unsigned long current_micros = micros();
unsigned long previous_micros = 0;
unsigned long interval_in_micros = 50;

const byte pwm_max = 200;

unsigned long int code_rcv;
const unsigned long int mask_byte_1 = 0x000000FF;
const unsigned long int mask_byte_2 = 0x0000FF00;
const unsigned long int mask_byte_3 = 0x00FF0000;
const unsigned long int mask_byte_4 = 0xFF000000;

typedef struct pwm_pins_motors {
    int pin;
    byte pwm_value;
    bool pwm_state;
    int pwm_tick_count;
} pwm_pin_motor_t;

const int motors_count = 3 * 3;
const byte motors_pins[motors_count] = {2,4,7,3,5,6,9,10,11};
const int motors_countNoPWM = 3;
const byte motors_pinsNoPWM[motors_count] = {2,4,7};
pwm_pin_motor_t myPWMPins[motors_count];

void initialize_motors(void)
{
    for (int index=0; index < motors_count; index++) {
        myPWMPins[index].pin = motors_pins[index];
        myPWMPins[index].pwm_value = 0;
        myPWMPins[index].pwm_state = ON;
        myPWMPins[index].pwm_tick_count = 0;

        pinMode(motors_pins[index], OUTPUT);
    }
}

void set_pwm_value(byte value, int id_motor)
{
    myPWMPins[id_motor].pwm_value = value;

```

```

}

void parse_int_rcv(unsigned long int value)
{
    byte byte_1, byte_2, byte_3, byte_4;

    byte_1 = value & mask_byte_1;
    byte_2 = (value & mask_byte_2) >> 8;
    byte_3 = (value & mask_byte_3) >> 16;
    byte_4 = (value & mask_byte_4) >> 24;

    if(byte_1 == 1){
        set_pwm_value(byte_2, 0);
        set_pwm_value(byte_3, 1);
        set_pwm_value(byte_4, 2);
    } else if (byte_1 == 3){
        set_pwm_value(byte_2, 3);
        set_pwm_value(byte_3, 4);
        set_pwm_value(byte_4, 5);
    } else if (byte_1 == 7) {
        set_pwm_value(byte_2, 6);
        set_pwm_value(byte_3, 7);
        set_pwm_value(byte_4, 8);
    }
}

void handlePWM()
{
    current_micros = micros();
    // check to see if we need to increment our PWM counters yet
    if (current_micros - previous_micros >= interval_in_micros) {
        // Increment each pin's counter
        for (int index=0; index < motors_countNoPWM; index++) {
            // each pin has its own tickCounter
            myPWMPins[index].pwm_tick_count++;

            // determine if we're counting on or off time
            if (myPWMPins[index].pwm_state == ON) {
                // see if we hit the desired on percentage
                // not as precise as 255 or 1024, but easier to do math
                if (myPWMPins[index].pwm_tick_count >= myPWMPins[index].pwm_value) {
                    myPWMPins[index].pwm_state = OFF;
                }
            } else {
                if (myPWMPins[index].pwm_tick_count >= pwm_max) {
                    myPWMPins[index].pwm_state = ON;
                    myPWMPins[index].pwm_tick_count = 0;
                }
            }
            digitalWrite(myPWMPins[index].pin, myPWMPins[index].pwm_state);
        }
        for (int index=0; index < motors_count; index++) {
            analogWrite(myPWMPins[index].pin, myPWMPins[index].pwm_value);
        }
        // reset the micros() tick counter.
        digitalWrite(13, !digitalRead(13));
        previous_micros = current_micros;
    }
}
}

```

```
void setup() {
  Serial.begin(57600);
  pinMode(13, OUTPUT);
  initialize_motors();
}

void loop() {
  handlePWM();

  current_ms = millis();
  if (current_ms - previous_ms >= interval_in_ms) {
    previous_ms = current_ms;
    while (Serial.available() > 0) {
      code_rcv = Serial.parseInt();
      parse_int_rcv(code_rcv);
    }
  }
}
```