ÉRICO MEIRA VERGNIANO
THIAGO SOUSA SILVA

SONORIZAÇÃO DE IMAGEM OBTIDA POR VISÃO ESTÉREO EM DISPOSITIVO EMBARCADO PARA AJUDA DE DEFICIENTES VISUAIS

São Paulo
2017

ÉRICO MEIRA VERGNIANO
THIAGO SOUSA SILVA

SONORIZAÇÃO DE IMAGEM OBTIDA POR VISÃO ESTÉREO EM DISPOSITIVO EMBARCADO PARA AJUDA DE DEFICIENTES VISUAIS

Texto apresentado à Escola Politécnica da Universidade de São Paulo como requisito para a conclusão do curso de graduação em Engenharia Mecatrônica, junto ao Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos (PMR).

Orientador:

Prof. Dr. Eduardo Lobo Lustosa Cabral

São Paulo
2017

Catalogação-na-publicação

Vergniano, Erico Meira ; Silva, Thiago
Sonorização de imagem obtida por visão estéreo em dispositivo embarcado para ajuda de deficientes visuais / E. M. Vergniano, T. S. Silva -- São Paulo, 2017.
68 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.VISÃO COMPUTACIONAL 2.SONORIZAÇÃO 3.DEFICIENTES I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos

AGRADECIMENTOS

Aos nossos pais, que nos apoiaram e nos incentivaram quando mais precisávamos.

Ao nosso orientador, que dedicou muito do seu tempo para nos orientar.

E a todos que de alguma forma nos ajudaram nessa longa jornada.

RESUMO

Pesquisas no campo da visão computacional e da sonorização de imagens vêm trazendo novas maneiras de interação entre deficientes visuais e o ambiente em que eles vivem. Através de dispositivos capazes de traduzir informações espaciais para informações sonoras, o deficiente visual é capaz de ter uma maior noção do espaço em que ele se encontra e, portanto, melhorar sua qualidade de vida. Este trabalho tem como proposta construir um dispositivo embarcado de auxílio à deficientes visuais baseado no uso da visão computacional e da sonorização de imagens, seguindo da análise de resultados do sistema em funcionamento e da validação da possibilidade de desenvolvimento de dispositivo comercial. O dispositivo proposto é composto por um sistema de visão estéreo, um sistema de áudio e um microprocessador. Os componentes utilizados são escolhidos considerando os requisitos propostos de sistemas embarcados, tais como custo, leveza, compactabilidade e exigência computacional. O algoritmo embarcado no dispositivo se divide em duas principais partes: o algoritmo de visão estéreo e o algoritmo de sonorização. O primeiro é composto de três principais algoritmos: calibração, retificação e correspondência. A saída desta primeira etapa é um mapa de disparidade, que contêm as informações necessárias para gerar um modelo tridimensional do ambiente. O mapa de disparidade é a entrada para o algoritmo de sonorização. A partir dele, o método transforma a localização tridimensional dos objetos do mundo real em informações sonoras (duração, frequência e amplitude da onda sonora). O algoritmo embarcado de reconstrução de imagens tridimensionais, de sonorização de imagens e coordenação de processos é implementado na linguagem C++ com auxílio das bibliotecas OPENCV e OPENAL. Para a calibração das câmeras e eventuais análises de resultados são utilizados algoritmos em MATLAB®.

Palavras-chaves: deficientes visuais, visão computacional, sonorização de imagens, visão estéreo, mapa de disparidade.

ABSTRACT

Research in the field of computer vision and image sonorization has brought new ways of interaction between visually impaired people and the environment in which they live. Through devices capable of translating spatial information into sound information, the visually impaired is able to gain a greater sense of the space in which he or she is and thus improve his or her quality of life. This work proposes to build an embedded device to aid visually impaired people based on the use of computer vision and image sonorization, followed by the analysis of the results of the system in operation and the analysis of the possibility of developing a commercial device. The proposed device consists of a stereo vision system, an audio system and a microprocessor. The components used are chosen considering the proposed embedded system requirements, such as cost, lightness, compactness and computational requirements. The algorithm embedded in the device is divided into two main parts: the stereo vision algorithm and the sonorization algorithm. The first one is composed of three main algorithms: calibration, rectification and correspondence algorithms. The output of this first step is the disparity map, which contains the information needed to generate a three-dimensional model of the environment. The disparity map is the input to the sonarization algorithm. From there, the method transforms the three-dimensional location of the real-world into sound information (duration, frequency, and amplitude of the sound wave). The embedded algorithm for the reconstruction of three-dimensional images, image sonorization and process coordination was implemented in the C++ language with the help of the OPENCV and OPENAL libraries. For the calibration of the cameras and eventual analysis of results, we used MATLAB® algorithms.

Keywords: visually impaired, computer vision, image sonorization, stereo vision, disparity map.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

1 INTRODUÇÃO

A *World Health Organization*(WHO, 2014) estima que existem 39 milhões pessoas no mundo que são cegas e aponta que outros 246 milhões sofrem de perda de visão moderada ou severa.

Considerando as limitações inerentes à deficiência visual, as pessoas cegas enfrentam um grande problema que é de se locomover em seu ambiente sem colidir em obstáculos inesperados. Daí a necessidade de um dispositivo de orientação externa para o deficiente visual. Duas tecnologias de ajuda para cegos na navegação, a bengala para cegos e o cão guia, têm sido utilizados pelos cegos por muitos anos (MOLTON et al, 1998; CARDIN et al., 2007).A bengala é ainda o dispositivo mais simples, barato e confiável utilizado na detectação de obstáculos no solo, vãos e escadas. No entanto tem alcance limitado uma vez que obstáculos não situados no solo são mais dificilmente detectados (LEE et al., 2014; CARDIN et al., 2007). O cão guia por sua vez é capaz de detectar e analisar situações complexas tais como: cruzamentos, escadas, perigos em potencial, reconhecer caminhos e mais. Porém o cão guia tem um alto custo, principalmente em razão do custo de treinamento e apresenta tempo de vida útil relativamente menor que a do usuário, requerendo sua substituição por outro periodicamente (CARDIN et al., 2007).

Nos últimos anos diversos estudos e iniciativas foram adotadas para atender aos deficientes visuais. Inclusive tecnologias voltadas para a possibilidade de restaurar a visão através da ligação do sistema nervoso é intensivamente investigada por ambas as áreas de medicina e engenharia. No campo da inclusão digital já existe dispositivos auxiliares, como a leitura eletrônica de Braille. No entanto, dispositivos auxiliares na navegação de cegos ainda não estão sendo empregados pela comunidade cega. Apesar de vários grupos de pesquisa estarem trabalhando em soluções para assistência à navegação dos deficientes visuais, os dispositivos auxiliares desenvolvidos para a navegação de cegos ainda não alcançaram resultados totalmente satisfatórios, devido ao alto custo, dificuldade de uso, portabilidade ou/e outros fatores.

Nesse sentido, este trabalho dedica-se a estudar, desenvolver e embarcar um dispositivo auxiliar de navegação para deficientes visuais utilizando a tecnologia de visão estéreo e sonorização de imagem.

2 OBJETIVOS

O presente trabalho consiste em desenvolver e embarcar um dispositivo composto por: um sistema de reconstrução de imagens tridimensionais utilizando visão estéreo e um sistema de sonorização de imagens tridimensionais. O intuito é ajudar pessoas cegas no reconhecimento de objetos presentes ao seu redor, com ajuda de um dispositivo que sonoriza imagens tridimensionais do ambiente.

O dispositivo de auxílio proposto é composto por um sistema de visão estéreo, um sistema de áudio e um microprocessador. O sistema de visão estéreo é responsável pela captação de pares de imagens à frente do usuário questão utilizadas na reconstrução tridimensional. O sistema de áudio transmite ao deficiente visual as informações das imagens reconstruídas em formato de áudio. O microprocessador é responsável pela execução dos algoritmos de reconstrução das imagens tridimensionais, conversão destas em formato de áudio e pela coordenação dos demais processos envolvidos durante a captura e a sonorização das imagens.

Projetos anteriores foram desenvolvidos utilizando a tecnologia de sonorização de imagens tridimensionais obtidas por estereografia. No entanto, deseja-se utilizar essa tecnologia em um produto comercial que possa ser usado facilmente por deficientes visuais no cotidiano. Porém, antes de se obter um produto comercial é necessário embarcar essa tecnologia para verificar sua viabilidade. Os desafios envolvidos neste processo estão descritos nos tópicos a seguir:

- Embarcar um sistema de visão estéreo incluindo o algoritmo de reconstrução de imagens tridimensionais.
- Embarcar um algoritmo de sonorização de imagens, ou seja, converter as informações da imagem obtida da reconstrução tridimensional em formato de áudio.
- Desenvolver um dispositivo compacto, leve e de baixo custo com intuito de ser utilizado por deficientes visuais no cotidiano sem dificultar sua locomoção.

Com a realização desse projeto, a próxima etapa seria a criação de um produto comercial, portanto, o custo e características portáteis como leveza e compactabilidade são importantes neste trabalho.

3 ESTADO DA ARTE

Nas últimas décadas diversos trabalhos foram feitos com objetivo de criar sistemas que possam ajudar pessoas com deficiência visual. Em muitos destes trabalhos foram pesquisados métodos, que estão diretamente relacionados ao tema deste trabalho e que ajudam na navegação independente do deficiente visual ou na identificação de objetos ao seu redor. A seguir é descrito os métodos sugeridos nestes trabalhos em relação à obtenção de imagens tridimensionais por visão estéreo, sonorização de imagens e outros métodos desenvolvidos para ajudar na identificação de objetos para deficientes visuais.

3.1 Visão Estéreo

No trabalho de MOLTON et al. (1998) é descrito o desenvolvimento de um sistema que ajuda deficientes visuais no desvio de obstáculos. Esse trabalho é uma parte importante da função de mobilidade de um projeto maior - *Autonomous System for Mobility, Orientation, Navigation and Communication* (ASMONC). Este projeto como um todo tem como objetivo permitir a navegação completa de deficientes visuais. Outros sensores, tais como um sonar, são susceptíveis de serem incluídos para melhorar a robustez. Porém MOLTON et al. (1998) afirmam que o sonar falha no reconhecimento de pequenos obstáculos e por isto existe a necessidade do desenvolvimento de um sistema de visão estéreo para reconhecimento de pequenos obstáculos, o que é apresentado no trabalho deles. O projeto foi implementado com um processador TI C40, com duas câmeras montadas em hastes rígidas sobre cada ombro do deficiente visual e alinhadas paralelamente. As câmeras foram calibradas usando o método proposto por TSAI (1987). Os métodos utilizados pelo sistema de visão foram o algoritmo *Ground Plane Obstacle Detection* (GPOD) e o *Dynamic Ground Plane Recalibration* (DGPR). O resultado dos testes no sistema desenvolvido mostrou que o uso de visão estéreo com o GPOD e o DGPR pode ser aplicado no projeto ASMONC. No entanto ainda é necessário aumentar a velocidade de processamento dos algoritmos.

VALSARAJ et al. (2016) propõem um sistema de visão estéreo que fornece mapas de profundidade em tempo real a partir de câmaras monocromáticas. Todo o processo é realizado num único FPGA. Para VALSARAJ et al. (2016), FPGAs, como já experimentado em estudos anteriores, são bastante adequados para o problema por serem muito mais rápidos para

aplicações de visão computacional. Quanto ao algoritmo de correspondência utilizado no projeto, eles escolheram a abordagem de correspondência local, pois métodos globais necessariamente requerem uma memória relativamente grande e mais tempo de processamento. Um aspecto interessante no trabalho de VALSARAJ et al. (2016) é que o processo de retificação foi ignorado pois era esperado que a qualidade das câmeras suprisse a necessidade desta etapa e assim pudessem salvar recurso computacional do FPGA. O projeto desenvolvido utilizou uma placa de desenvolvimento Atlys da Digilent para configurar, desenvolver e executar o sistema e para as câmeras foi usado um pacote chamado VmodCAM da Digilent. Como resultado o processamento é realizado em tempo real e com tempo de latência muito baixa. Exibição do mapa de profundidade ocorre com uma taxa de 20 quadros por segundo. Eventuais ruídos da câmera, bem como o tamanho limitado da janela no algoritmo de disparidade resultaram em valores não confiáveis em determinadas partes da imagem.

Já no trabalho de BANZ et al. (2010) também foi implementado um sistema de visão estéreo em FPGA sendo este um Xilinx Virtex-5. No entanto o método utilizado para resolver o problema de correspondência foi o *Semi-Global Matching* (SGM). Para BANZ et al. (2010) a escolha deste método é por ele superar os métodos locais em termos de qualidade dos mapas de disparidade e robustez. O processamento baseado em SGM, mesmo sendo mais complexo em relação aos métodos locais pode ser implementado em tempo real, já que a arquitetura de hardware escolhida cumpria as exigências da aplicação. Os resultados obtidos deste projeto foram mapas de disparidade de imagens VGA (640x480 *pixels*), em tempo real com 30 quadros por segundo, com 128 níveis de disparidade e uma frequência de *clock* de 39 MHz.

A abordagem implementada por HERNANDEZ-JUAREZ et al. (2016) foi o desenvolvimento de um sistema de visão estéreo que também utiliza o método de correspondência SGM. Porém o sistema foi incorporado em um dispositivo GPU Tegra X1 e obteve como resultado 42 quadros por segundo com imagens de 640x480 *pixels* com 128 níveis de disparidade.

Diversos métodos de correspondência para visão estéreo foram propostos até então. Uma das propostas foi descrita por MUKHERJEE et al. (2014), que apresenta uma abordagem para o problema de disparidade em visão estéreo onde a estimativa é feita baseada na segmentação dos valores de luminosidade dos *pixels* da imagem. Quando comparado o algoritmo desenvolvido com as abordagens de estimativa de profundidade tradicionalmente

utilizadas e ainda com outros algoritmos recentemente desenvolvidos, verificou-se que o algoritmo desenvolvido teve um melhor desempenho do que os tradicionais SAD, NCC e do que algoritmos recentes. No entanto quando comparado com algoritmos que tratam o problema de oclusão ou que usam refinamento iterativo, concluiu-se que esses algoritmos tiveram uma melhor precisão, mas com maior custo computacional. Outro problema enfrentado por MUKHERJEE et al.(2014) é que o algoritmo desenvolvido tem menor desempenho na presença de sombras e em regiões pouco iluminadas.

3.2 Sonorização de Imagens

MEIJER (1992) propôs o clássico método conhecido como "*Piano Transform*" que foi implemento em um sistema de assistência para deficientes visuais chamado vOICe (ZHANG et al., 2014). O vOICe foi um trabalho pioneiro na área de sonorização de imagens. Este projeto consistia em sonorizar imagens em tons de cinza com 16 níveis luminosidade e com resolução de 176x64 pixel. Cada uma das 64 colunas de uma imagem era escutada por cerca de 15 ms, fazendo uma varredura completa da imagem em cerca de 1 segundo. Os sons produzidos variam em função da posição e do brilho dos *pixels* em cada coluna da imagem que está sendo sonorizada. Cada *pixel* da coluna corresponde a uma frequência diferente. Quanto mais alta a posição do *pixel*, maior a frequência. O brilho dos *pixels* na imagem é codificado em termos de amplitude da onda senoidal que é emitida para aquele *pixel*. Silêncio corresponde à cor preta e um som com volume máximo significa branco (HENRIQUES et al., 2010; MEIJER, 1992).

Uma abordagem modificada do vOICe é apresentada por ZHANG et al. (2014). Nesta abordagem cada coluna da imagem equivale a uma Transformada de Fourier Discreta (DFT) de um sinal de áudio, assim a Transformada Inversa Rápida de Fourier (IFFT) pode ser utilizada para implementar a inversa da DFT e calcular o sinal de áudio convertido. O resultado apresentado mostra que este método é mais eficaz e tem um desempenho melhor em tempo real do que o original apresentado por MEIJER (1992).

No trabalho de HENRIQUES et al. (2010) é desenvolvido um método para sonorização de imagens que converte a informação de cores de uma imagem para o formato de som. Eles utilizam os atributos de cores HSV (*hue, saturation* e *brightness)* de uma imagem e convertem nos atributos de som que são frequência, timbre e volume. A forma que foi implementada a transmissão do som ao usuário é similar ao procedimento utilizado no vOICe,

só que a varredura é feita por linhas da imagem ao invés de colunas como é feito no vOICe. Para facilitar o reconhecimento de alterações de movimento e cor, o protótipo tem outra característica útil, que consiste em jogar apenas as diferenças entre os quadros. Fazendo isso, o protótipo armazena um instante de uma cena e a subtrai de todos os quadros subsequentes. Como consequência, ele só gera som para os segmentos que mudam, não disponibilizando toda a informação da imagem em tempo real. Por fim os resultados apresentados demonstram em que o som sintetizado permite a obtenção da informação da cor da imagem, localização de objetos e geometria de objetos de uma só cor.

3.3 Outros métodos de identificação de objetos

SHOVAL et al. (2003) apresentaram dois sistemas que ajudam na navegação de pessoas com deficiência visual, são eles o NavBelt e o Guidecane. O NavBelt é utilizado como um cinto equipado com uma série de sensores de distância ultrassônicos. As informações de distâncias obtidas pelo cinto são transmitidas por meio de fones de ouvido. Uma limitação do NavBelt é ser extremamente difícil para o usuário compreender os sinais de distâncias rapidamente. Já o dispositivo chamado GuideCane supera este problema de forma eficaz. O GuideCane usa a mesma tecnologia do NavBelt mas os sensores estão alocados em um dispositivo com rodas acoplado à uma haste. O dispositivo segue à frente do usuário, o qual empurra o dispositivo através da haste. Quando o GuideCane detecta um obstáculo, um motor é acionado para redirecionar os sentidos das rodas. O usuário sente imediatamente o desvio das rodas através da haste evitando o obstáculo. Apesar de ser mais fácil de usar em relação ao NavBelt, o GuideCane apresenta dificuldades para o usuário na presença de superfícies irregulares e degraus.

Já no trabalho de BAHADIR et al. (2012) é desenvolvido um sistema de detecção de obstáculo vestível, totalmente integrado à estrutura têxtil. Esse sistema permite a detecção de obstáculos para pessoas com deficiência visual. O protótipo desenvolvido também funciona com sensores ultrassônicos. Além disto possuí vibradores, fontes de alimentação e um microcontrolador. O princípio de funcionamento do sistema baseia-se em duas funções principais: A detecção de obstáculos por meio dos sensores de ultrassom e as distâncias transmitidas ao usuário por meio de vibradores. O sistema desenvolvido obteve bons resultados quanto à detecção de objetos e na transmissão da informação. No entanto o sistema exige um

grande número de atuadores e detectores e somente é capaz de identificar a posição do obstáculo na área de detecção dos sensores.

O *tactile vision substitution systems* (TVSS) é uma abordagem desenvolvida que visa ajudar deficientes visuais. O TVSS é um sistema que envia informação visual através de estimuladores em contato com a pele. Imagens captadas por uma câmera são traduzidas em forma de energia por estimuladores (elétricos ou vibratórios) grudados na pele. Estes estimuladores foram testados em várias partes do corpo (abdômen, costas, coxa e ponta do dedo). Após testes com TVSS em deficientes visuais, eles obtiveram informações consideráveis sobre a plasticidade do cérebro e adaptação do deficiente sobre a resposta dos estimuladores, bem como sobre outros assuntos relacionados. No entanto, o projeto teve limitações práticas, devido ao fato dos vibro-atuadores serem volumosos e exigirem considerável energia e no caso de sistemas eletro-táteis exigirem tensões relativamente elevadas especialmente em áreas como a ponta dos dedos, por causa das camadas de proteção da pele sobre os receptores sensoriais. Uma nova abordagem é proposta para resolver estes problemas, que consiste em fazer a estimulação eletro-táctil ser aplicada na língua dos deficientes visuais, isto porque a língua é muito sensível, é protegida pela boca, os receptores sensoriais encontram-se perto da superfície e a presença de saliva assegura bom contato elétrico. Eles demonstraram que a percepção com a estimulação elétrica da língua é melhor do que com estimulação no dedo e a língua requer apenas cerca de três por cento (5-15 V) da tensão, e muito menos corrente (0,4-2,0mA) do que a ponta do dedo. No entanto essa abordagem exige um treinamento de muitas horas do deficiente visual (BACH-Y-RITA et al., 1998). Atualmente um produto comercial implementa essa tecnologia denominado por BrainPort® V100 (BRAINPORT T., 2016a). Este produto inclui uma câmera de vídeo montada em um par de óculos de sol, um controlador de mão, uma matriz de estimuladores táteis de língua (*tongue array*), duas baterias de lítio reutilizáveis com carregador e uma maleta (BRAINPORT T., 2016b).

Para MOLTON et al. (1998) a bengala de deficientes visuais e o cão-guia têm sido utilizados por cegos por muitos anos. Dispositivos de locomoção eletrônicos, utilizando sonar para detectar obstáculos, são de difíceis aceitação do mercado, pois a informação útil obtida a partir deles não é significativamente melhor do que a obtida pela bengala de deficientes visuais. Além disto, esses dispositivos também são considerados dispendiosos e difíceis de se utilizar. Em relação aos dispositivos eletrônicos que utilizam pulsos elétricos ou vibratórios aplicado à

pele, quando são utilizados no ambiente exterior, experimentos mostraram que há excesso de informações para serem transmitidas devido aos detalhes do fundo encobrirem a informação primária necessária para a mobilidade. Além disso, a resposta da pessoa a um estímulo táctil logo se torna cansativo. Desta forma, MOLTON et al. (1998) justificam sua preferência pelo uso de visão estéreo aplicado à dispositivos de ajuda ao deficiente visual.

4 PRÉ-REQUISITOS

Neste tópico é descrito os pré-requisitos de projeto, tais como os de compactabilidade, tempo máximo de processamento, custo, leveza.

4.1 Custos

Os principais componentes utilizados neste projeto é um processador, uma câmera de visão estéreo, uma bateria, um fone de ouvido e a estrutura do óculos. Levando em conta tais componentes, é definido o custo máximo de cada um deles e, a partir desta estimativa, é determinado o custo máximo total para o projeto.

Considerando que o custos dos componentes não podem ser significativamente altos, porém ainda precisam atender as demais necessidades do projeto. É escolhido uma estrutura do óculos que possa ser fabricada com o uso de uma impressora 3D, isso pois, a prototipação utilizando uma impressora 3D permite uma rápida fabricação com custo relativamente baixo de uma unidade quando comparamos com processos de fabricação convencionais. A escolha do processador é um Raspberry Pi Model B e o custo é um parametro levado em consideração para tal decisão, pois o Raspberry Pi Model B além de atender os demais requisitos como leveza, compactibilidade e capacidade de processamento suficiente para atender as necessidades do projeto, ele também tem um custo relativamente baixo em comparação a placas de processamentos dedicadas, as quais seriam ideais para construção de um produto comercial. Já a decisão dos outros equipamentos para a construção do protótipo não é influenciada pelo custo, mas sim por outro requisitos de projeto.

Além disso, espera-se uma tolerância de até 10% a mais do que o esperado para o custo total. Na Tabela 1 a seguir é mostrado uma estimativa de custos para a realização do projeto.

Tabela 1- Estimativas de custo dos componentes do projeto

Componente	Valor máximo (R$)
Processador	200,00
Bateria	100,00
Fone de ouvido	20,00
Câmera de visão estéreo	400,00
Estrutura do óculos	100,00

Sendo assim, conclui-se que o valor máximo aproximado para os custos é de 900 reais.

4.2 Leveza e compactabilidade

Para estabelecer um pré-requisito em termos de leveza, deve ser considerado o fato de que a bateria e o processador serão colocados no braço do usuário e que o peso da estrutura do óculos, da câmera e do equipamento sonoro é determinado de acordo com o peso dos óculos.

A compactabilidade do dispositivo é um parâmetro importante, porém difícil de mensurar. Por isso, a compactabilidade será determinada de acordo com o bom senso, ou seja com a escolha de equipamento que possuam o menor volume possível. É claro que a leveza possui influência em tal parâmetro, no entanto, outros aspectos devem ser considerados, tais como se o dispositivo não é grande a ponto de causar transtornos ao usuário e se o dispositivo fica bem fixado e sem folgas quando colocado para garantir o conforto do usuário quando utilizar o equipamento.

4.3 Tempo de processamento

O tempo de processamento é importante porque ele está diretamente ligado com o entendimento do sinal sonoro pelo deficiente visual. Se o tempo de processamento for muito demorado, o usuário receberá a informação da imagem muito tempo depois que ela foi capturada. Assim, o tempo máximo de processamento de uma imagem antes de ser sonorizada esperado neste projeto é de 1,5 segundos.

5 DESCRIÇÃO GERAL DOS ALGORITMOS

Para a realização deste projeto é necessário o desenvolvimento de quatro algoritmos: calibração, retificação, correspondência e sonorização. O processo de calibração ocorre somente uma vez e é necessário para obter os parâmetros do sistema de câmera estéreo que são utilizados no algoritmo de retificação. Após a obtenção dos parâmetros do sistema de câmera estéreo o próximo algoritmo usado é o de retificação, o qual é utilizado para diminuir a complexidade computacional do algoritmo de correspondência. Já o algoritmo de correspondência é utilizado para obter a imagem tridimensional. Por fim o algoritmo de sonorização é utilizado para sonorizar a imagem tridimensional para o deficiente visual. Nos próximos tópicos é descrito com mais detalhes esses algoritmos.

6 CALIBRAÇÃO E RETIFICAÇÃO

A calibração é o processo de determinação de parâmetros que relacionam as informações das imagens obtidas pela câmera (expressa em *pixels*) com um sistema de coordenadas global externo (BROWN M. et al., 2001). Tais parâmetros podem ser usados para corrigir distorções das lentes da câmera e mensurar tamanho de objetos.

Neste projeto a calibração das câmeras foi feita através de um *software* de calibração estéreo de alta qualidade desenvolvido para MATLAB disponível em BOUGUET (2015). O modelo de calibração utilizado por este *software* é o modelo de câmera *pinhole*, neste modelo, pontos do mundo real 3D são projetados em um plano, chamado plano da imagem, que formará a imagem capturada pela câmera. A Figura 1ilustra o modelo de câmera *pinhole*, onde o ponto *P=(X, Y, Z)* do mundo real é mapeado para o plano de imagem onde possui coordenadas (*u*,*v*).

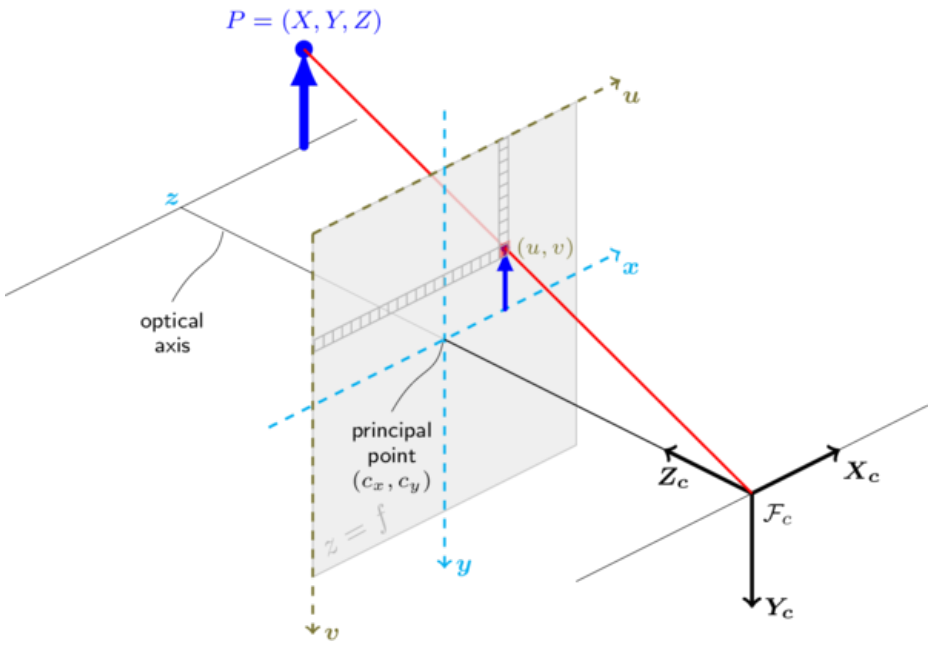

Figura 1– Modelo de camera pinhole (OPENCV DOCUMENTATION, 2017)

Para que tal transformação seja feita, torna-se necessária a obtenção das seguintes matrizes de transformação:

$$s\,m = A[R|T]M \tag{1}$$

Expandindo a Equação 1, obtém-se:

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}f_x & 0 & c_x\\0 & f_y & c_y\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_1\\r_{21} & r_{22} & r_{23} & t_2\\r_{31} & r_{32} & r_{33} & t_3\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix} \tag{2}$$

Na Equação 2, (*X*,*Y*,*Z*) são as coordenadas do ponto 3D no mundo real; (*u*,*v*) são as coordenadas em pixels do ponto projetado; A é a matriz de parâmetros intrínsecos; f_x, f_y são as distâncias focais representadas em unidades de pixels; (c_x, c_y) é a coordenada do ponto principal e $[R/T]$ é a matriz de parâmetros extrínsecos e descreve a movimentação da câmera ao redor de uma cena estática ou vice-versa.

Lentes possuem distorções, tais como distorções radiais e tangenciais. Essas distorções podem ser corrigidas através do seguinte modelo que é equivalente à transformação anterior:

$$\begin{bmatrix}x\\y\\z\end{bmatrix} = R\begin{bmatrix}X\\Y\\Z\end{bmatrix} + t \tag{3}$$

A imagem de projeção normalizada é dada pelas Equações 4 e 5:

$$x' = x/z \tag{4}$$

$$y' = y/z \tag{5}$$

As Equações 6 e 7 corrigem as distorções das lentes:

$$x'' = x'\frac{1 + k_1r^2 + k_2r^4 + k_3r^6}{1 + k_4r^2 + k_5r^4 + k_6r^6} + 2p_1x'y' + p_2(r^2 + 2x'^2) \tag{6}$$

$$y'' = y'\frac{1 + k_1r^2 + k_2r^4 + k_3r^6}{1 + k_4r^2 + k_5r^4 + k_6r^6} + 2p_2x'y' + p_1(r^2 + 2y'^2) \tag{7}$$

Onde:

$$r^2 = x'^2 + y'^2 \tag{8}$$

$$u = f_x x'' + c_x \tag{9}$$

$$v = f_y y'' + c_y \tag{10}$$

onde os coeficientes k_1, k_2, k_3, k_4, k_5 *e* k_6 são os coeficientes de distorções radiais e p_1 e p_2 são os de distorções tangenciais(OPENCV DOCUMENTATION, 2017).

As funções usadas pelo toolbox do MATLAB usam imagens com padrões de calibração e o modelo anterior para estimar os parâmetros extrínsecos e intrínsecos de cada uma das duas câmeras. As Figuras 2 e 3 ilustram uma compilação das 20 imagens capturadas por cada uma das câmeras. O algoritmo identifica as quinas do padrão de tabuleiro das imagens e dado que as dimensões de cada quadrado é dado na entrada do programa, ele calcula as coordenadas (*u*,*v*) de cada quina. Assim, com um número suficiente de imagens, o algoritmo é capaz de identificar todos os parâmetros da câmera através da resolução de um sistema de equações.

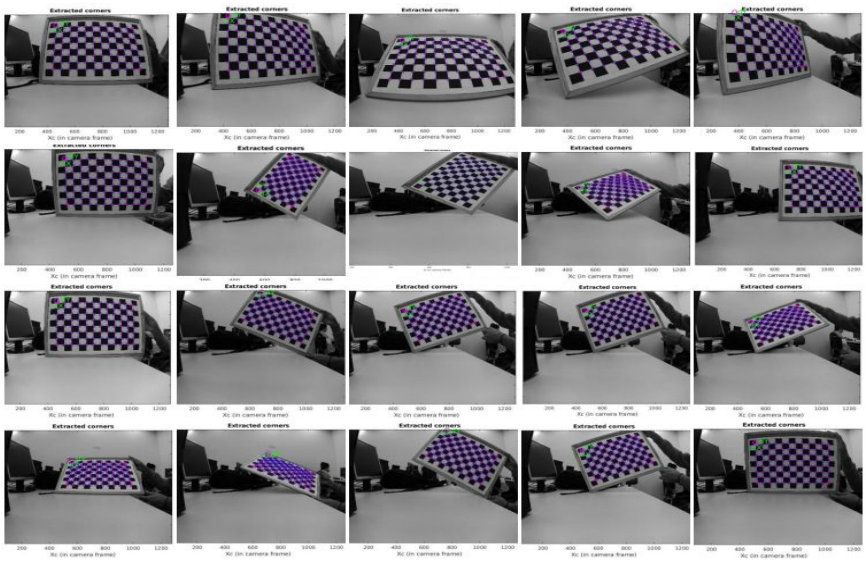

Figura 2– Compilação de 20 padrões capturados pela câmera esquerda

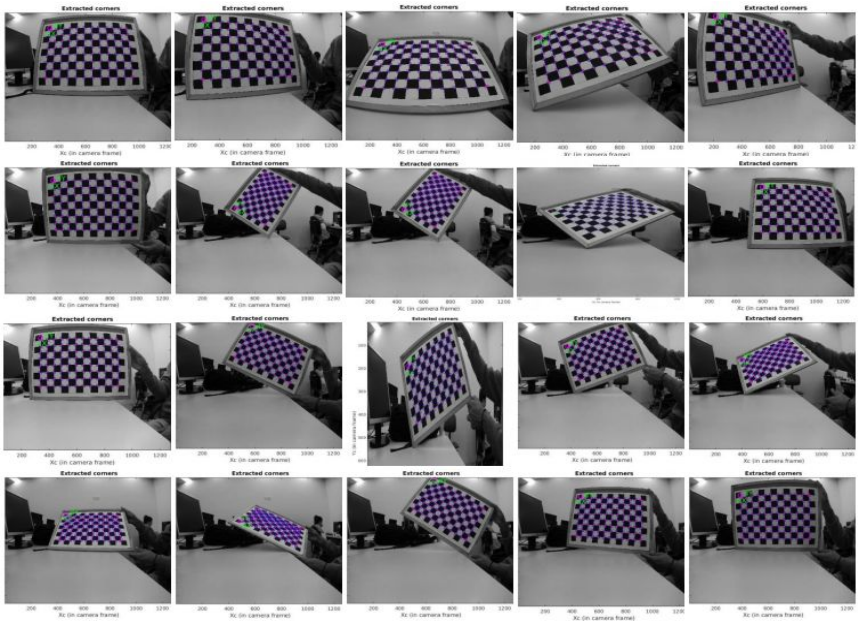

Figura 3– Compilação de 20 padrões capturados pela câmera direita

Os resultados dos parâmetros intrínsecos determinado pelo algoritmo de calibração para câmera esquerda são:

Distância focal : $f = [\ 827.45601 \quad 827.84860\ ] +/- [\ 2.53500 \quad 2.6613\ ]$

Ponto principal: $c = [\ 639.50000 \quad 479.5000\ ] +/- [\ 0.00000 \quad 0.0000\ ]$

Inclinação da câmera: $alfa = [\ 0.00000\ ] +/- [\ 0.0000\ ]$

Distorção: $k = [-0.35243 \quad 0.11536 \quad -0.00383 \quad 0.0050 \quad 0.0000\ ]$

$+/- \quad [0.00337 \quad 0.00328 \quad 0.00048 \quad 0.00041 \quad 0.0000\ ]$

Os resultados dos parâmetros intrínsecos determinado pelo algoritmo de calibração para câmera direita são:

Distância focal : $f = [\ 862.19326 \quad 863.87070\ ] +/- [\ 4.05841 \quad 4.41531\ ]$

Ponto principal: $c = [\ 639.50000 \quad 479.5000\ ] +/- [\ 0.00000 \quad 0.0000\ ]$

Inclinação da câmera: $alfa = [\ 0.00000\ ] +/- [\ 0.0000\ ]$

Distorção: $k = [-0.36286 \quad 0.13135 \quad -0.00245 \quad 0.0396 \quad 0.0000]$

$+/- \quad [0.00665 \quad 0.00845 \quad 0.00092 \quad 0.00051 \quad 0.0000]$

Uma vez que a calibração de cada uma das câmeras está feita, o algoritmo realiza a calibração estéreo. A calibração estéreo determina a matriz de transformação que relaciona as coordenadas de um ponto na câmera esquerda para a câmera direita ou vice-versa. A

Figura4mostra o posicionamento dos sistemas de câmera estéreo e dos padrões de calibração no espaço. O resultado obtido de rotação e translação que relacionam a câmera direita à câmera esquerda são:

Rotação: om = [-0.02098 -0.03313 -0.00396]+/- [0.00044 0.00032 0.00026]

Translação: T = [-60.10443 0.10154 0.78708]+/-[0.14521 0.18009 0.60058]

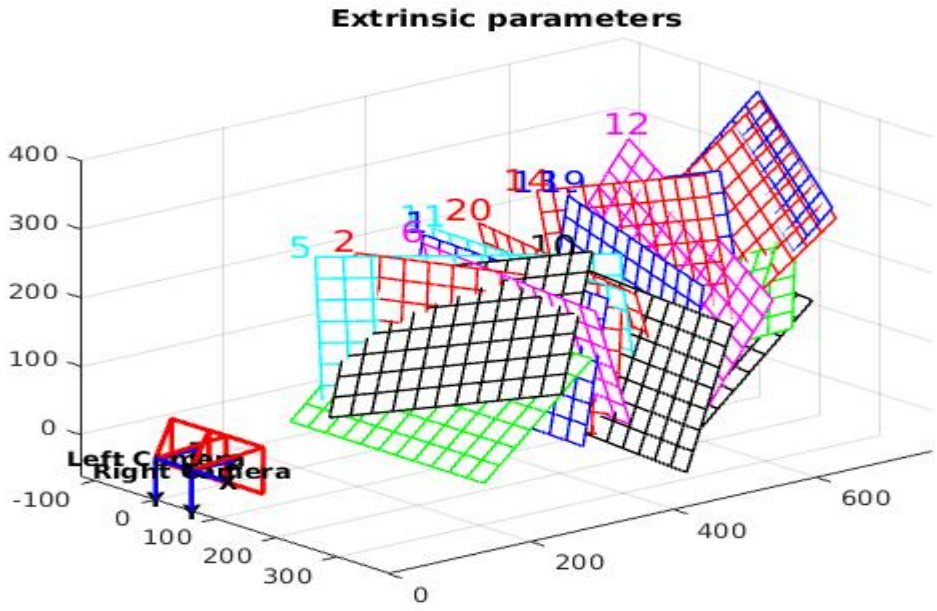

Figura 4– Resulta da calibração estéreo. Posicionamento dos padrões de calibração em relação a câmera estéreo.

Um dos resultados do processo de calibração de um sistema de câmeras estéreo é uma matriz de transformação linear (matriz fundamental) que retifica o par de imagens obtidas pelo sistema de câmera estéreo. A obtenção da matriz fundamental pelo algoritmo de calibração é necessária para que possa ser feita retificação das imagens. A retificação é necessária para que a complexidade do algoritmo de correspondência seja reduzida. A retificação faz com que os *pixels* correspondentes dos pares das duas imagens estejam alinhados horizontalmente. Dessa forma, a região de busca de pontos correspondentes entre as imagens é reduzida, assim como a complexidade computacional e a probabilidade de falsas correspondências (BROWN M. et al., 2001).

Para demonstrar os efeitos da retificação sobre um par de imagens não retificadas são expostas as figuras a seguir. A Figura 5 ilustra um par de imagens antes da retificação, enquanto que a 6 mostra as respectivas imagens retificadas.

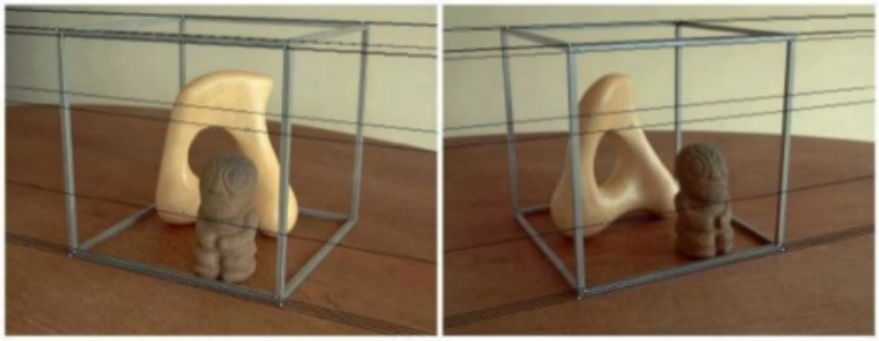

Figura 5– Imagem não retificada(GOSTA, M. et al., 2010).

Figura 6– Imagem após retificação (GOSTA, M. et al., 2010).

Considerando (u_l,v_l) e (u_r,v_r) comosendo a coordenada de um determinado ponto no sistema de coordenadas das câmeras esquerda e direita, respectivamente, temos que após a retificação, tais coordenadas se relacionam conforme mostrado nas Equações 11 e 12:

$$u_l = u_r + d \tag{11}$$

$$v_l = v_r \tag{12}$$

onde *d* é chamado de disparidade.

A biblioteca OPENCV disponibiliza métodos que auxiliam na retificação de imagens. Estes métodos são utilizados neste projeto para obtenção da retificação das imagens antes do

processo de correspondência.O resultado obtido a partir do algoritmo de retificação disponibilizado pela biblioteca do OPENCV em conjunto com os parâmetros de calibração estéreo do sistema de câmeras do projeto obtidos pelo *toolbox* do MATLAB pode ser visualizado nas Figuras 7 e 8.

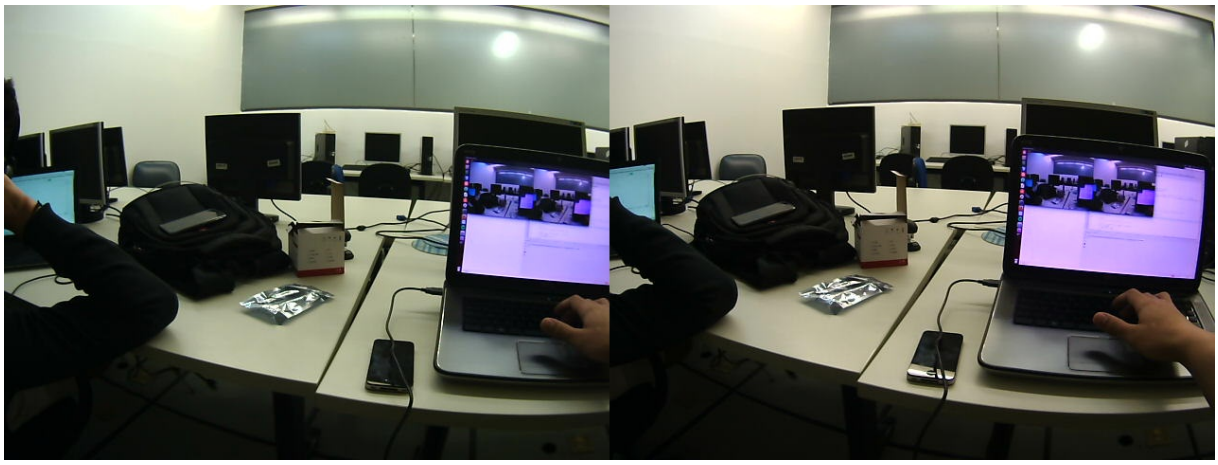

Figura 7– Imagens não retificadas obtidas pelo sistema de estéreo.

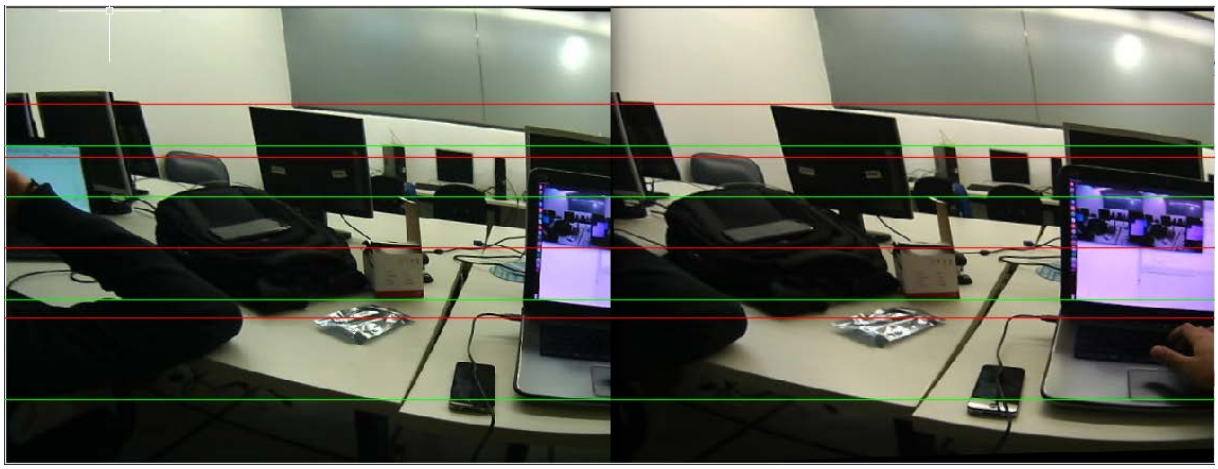

Figura 8– Imagens retificadas obtidas pelo sistema estéreo após utilização do toolbox do MATLAB.

Os algoritmos de calibração e retificação desenvolvidos utilizando a biblioteca do OPENCV e o *toolbox* de calibração de câmeras do MATLAB® foram validados e foram bem-sucedidos. O resultado da retificação apresentava um pequeno deslocamento entre as linhas das

imagens estéreis (as linhas da imagem à esquerda estavam cerca de seis *pixels* acima das linhas correspondes da imagem direita), mas o deslocamento foi corrigido manualmente adicionando um pequeno deslocamento no sentido oposto. Os resultados da retificação das imagens foram avaliados como bons e satisfatórios para propósito deste projeto.

7 CORRESPONDÊNCIA

O problema de correspondência consiste em determinar pontos de correspondência entre o par de imagens obtido pelo sistema de câmera estéreo. Não existe uma solução geral para o problema de correspondência devido a correspondências ambíguas causadas, por exemplo, por oclusão ou falta de textura (BROWN M. et al., 2001).

Uma vez achado os pontos de correspondência no par de imagens, o deslocamento entre dois pontos correspondentes fica determinado, tal deslocamento é denominado disparidade. O conjunto de todas as disparidades entre as duas imagens é chamado de mapa de disparidade (BROWN M. et al., 2001).

A informação contida no mapa de disparidade é a que representa a profundidade dos objetos na imagem, desta forma o mapa de disparidade é a imagem tridimensional que será sonorizada para o deficiente visual. Foi desenvolvido um algoritmo de cálculo da disparidade baseado no algoritmo de MUKHERJEE et al. (2014). Após o desenvolvimento, esse algoritmo foi validado com imagens disponíveis em bancos de dados da área obtidas em SCHARSTEIN (2014). As imagens desse banco de dados são consideradas bem comportadas e os resultados utilizando essas imagens foram satisfatórios. A validação foi feita com essas imagens, pois a câmera estéreo estava indisponível até o momento. Porém quando foi feito testes com as câmeras compradas os resultados não se mostraram satisfatórios, isto porque o algoritmo utilizado usa as cores das imagens como parâmetro e existia uma grande variação de cores entre as duas imagens obtidas pelo sistema de câmera estéreo comprada.

A partir disto foi pesquisado outro algoritmo de cálculo da disparidade que fosse adequado ao sistema de câmera comprado. A biblioteca OPENCV disponibiliza um método de correspondência utilizando o algoritmo de cálculo da disparidade denominado de S*emi-Global Block Matching* que foi desenvolvido baseado no trabalho de HIRSHMULLER (2008). Esse método foi utilizado nesse projeto para obtenção dos mapas de disparidades. Também, foi utilizado após o cálculo da disparidade um filtro de suavização conhecido como *MedianBlurring,* disponibilizado pela biblioteca do OPENCV para refinar o resultado da disparidade calculada.

Foi feito testes com o algoritmo de cálculo da disparidade que utiliza a abordagem *Semi-Global Matching* da biblioteca do OPENCV e o resultado se mostrou relativamente bom

e, portanto, foi escolhido esse algoritmo como o algoritmo padrão para o cálculo da disparidade desse projeto, no entanto o programa permite a escolha entre algoritmos de calculo da disparidade, permitindo assim a troca do algoritmo padrão de calculo da disparidade pelo algoritmo baseado na abordagem de MUKHERJEE et al. (2014) .

6.1. Algoritmo desenvolvido baseado no trabalho de MUKHERJEE et al. (2014).

O algoritmo de correspondência baseado no trabalho MUKHERJEE et al. (2014) que foi desenvolvido durante o projeto utiliza-se de pouco esforço computacional quando comparado com a maioria dos algoritmos de correspondência disponíveis. Isso porque antes de fazer o cálculo de disparidade, o método agrupa pixels com luminosidades próximas em clusters através do algoritmo de *clustering* chamado *K-Means*.Uma vez que a imagem está agrupada em *clusters* de mesma luminosidade, o método identifica as fronteiras entre os *clusters*, de forma a definir o contorno de *clusters* adjacentes. Assim, somente a disparidade nessas fronteiras é de fato calculada, enquanto que dentro dessas fronteiras, ao invés do cálculo, uma estimativa é feita para a disparidade.

Dessa forma, o primeiro passo para o cálculo do método consiste em converter a imagem capturada pela câmera estéreo no formato RGB para o formato LAB. O espaço de cores em LAB é mais adequado para a percepção humana devido ao fato de os olhos humanos serem mais sensíveis às mudanças de luminosidade do que de cores. Assim, o algoritmo extrai a componente L das imagens, que corresponde a luminosidade.

O próximo passo consiste em usar o algoritmo de *K-Means clustering* sobre os valores de L da imagem esquerda. Tal algoritmo agrupa cada pixel dentro de um dos *K* grupos, onde cada pixel pertence ao grupo que possui a média de valores de L mais próxima da luminosidade do pixel. O resultado de uma imagem antes e depois do agrupamento baseado no algoritmo *K-Means* pode ser visto nas Figuras 9 e 10.

Figura 9 - Imagem esquerda utilizada para validação do algoritmo desenvolvido utilizando abordagem de K-Means.

Figura 10–Resultado da imagem esquerda após agrupamento utilizando algoritmo de K - Means.

Uma vez que cada *pixel* da imagem esquerda está dentro de um dos *K* grupos, o próximo passo é determinar as fronteiras de cada um dos agrupamentos. As fronteiras são determinadas de tal forma que se um pixel está na fronteira, então ele é atribuído o valor um, caso contrário, zero. Assim, se um determinado pixel pertence ao grupo *i* (*i* de 1 a *K*), ele é dito estar na fronteira se qualquer um dos pixels vizinhos a ele pertencer a outro grupo diferente de *i*. O procedimento descrito pode ser visto na Equação 13 a seguir, no primeiro caso não há a reconhecimento de *pixel* pertencente às fronteiras no segundo já ocorre o reconhecimento de um *pixel* pertencente à fronteira:

$$\begin{array}{lcl} 47\ 47\ 47 & & 0\ 0\ 0 \\ 47\ 47\ 47 & \Rightarrow & 0\ 0\ 0 \\ 47\ 47\ 47 & & 0\ 0\ 0 \\ & & \\ 13\ 28\ 28 & & 0\ 0\ 0 \\ 13\ 28\ 28 & \Rightarrow & 0\ 1\ 0 \\ 28\ 28\ 28 & & 0\ 0\ 0 \end{array} \quad (13)$$

Com as fronteiras definidas, o algoritmo aplica dois filtros para refinar a fronteira, um de preenchimento, que altera o valor de um pixel de ‘zero’ para ‘um’ caso ele esteja cercado por ‘uns’, como mostra a Equação 14 abaixo e outro de remoção, que altera o valor de um pixel de ‘um’ para ‘zero’ caso ele não esteja conectado por quatro vizinhos com valores ‘um’, conforme Equação 15 a seguir:

$$\begin{array}{lcl} 1\ 1\ 1 & & 1\ 1\ 1 \\ 1\ 0\ 1 & \Rightarrow & 1\ 1\ 1 \\ 1\ 1\ 1 & & 1\ 1\ 1 \end{array} \quad (14)$$

$$\begin{array}{lcl} 1\ 0\ 0 & & 1\ 0\ 0 \\ 0\ 1\ 0 & \Rightarrow & 0\ 0\ 0 \\ 0\ 1\ 0 & & 0\ 1\ 0 \\ & & \\ 0\ 1\ 0 & & 0\ 1\ 0 \\ 1\ 1\ 1 & \Rightarrow & 1\ 1\ 1 \\ 0\ 1\ 0 & & 0\ 1\ 0 \end{array} \quad (15)$$

As Figuras11 e 12 mostram, respectivamente, as fronteiras dos agrupamentos antes e depois da aplicação dos filtros.

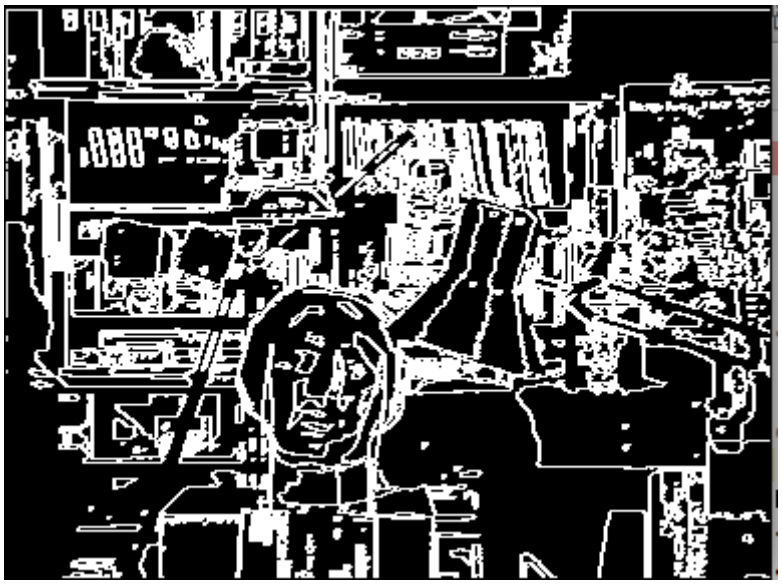

Figura 11– Resultado do reconhecimento das fronteiras dos agrupamentos.

Figura 12– Fronteiras resultantes dos agrupamentos obtidas após aplicação dos filtros de preenchimento e de remoção.

O próximo passo consiste no cálculo da disparidade entre os pixels da imagem direita e esquerda através do método de *Block Matching*. No entanto, ao invés de calcular a disparidade para todos os pixels, somente a disparidade dos pixels que correspondem às fronteiras são calculados. Dessa forma, para cada pixel da imagem esquerda que corresponde a um pixel de

fronteira calculado, é procurado em uma determinada região da mesma linha da imagem direita o pixel correspondente.

Dentro dessa região, é considerado como *pixel* da imagem direita correspondente ao da imagem esquerda aquele que obtiver o menor valor para a soma das diferenças absolutas da luminosidade, conforme mostrado pela Equação 16 (BROWN M. et al., 2001).A Figura 13 mostra a região horizontal onde o algoritmo faz a busca pelo *pixel* correspondente.

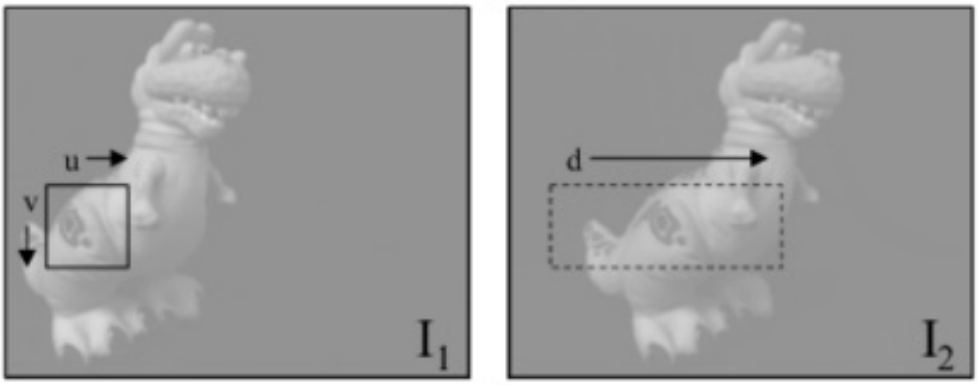

Figura 13– Região de busca horizontal do método Block Matching (BROWN M. et al., 2001).

$$\sum_{u,v} |l_1(u,v) - l_2(u+d,v)| \tag{16}$$

Uma vez que as disparidades das fronteiras estão determinadas, é feita uma etapa de preenchimento dos valores de disparidade dos *pixels* intermediários (entre as fronteiras). Tal preenchimento é a etapa final do processo para determinar o mapa de disparidade e é feito da seguinte forma: o algoritmo percorre cada linha da esquerda para direita e preenche a disparidade entre dois *pixels* de fronteiras por meio de uma interpolação linear entre os valores de disparidade de cada uma das fronteiras. A Figura 14 mostra o resultado final do procedimento de determinação do mapa de disparidade da imagem da Figura 9:

Figura 14– Mapa de disparidade da imagem de validação utilizando o algoritmo utilizando abordagem de K-Means.

6.2. Algoritmo Semi Global Matching

O algoritmo utilizado da bibliotecaOPENCV é o *Semi Global Matching* baseado no trabalho de HIRSCHMULLER, H. (2008).Ocálculo da correspondência por métodos Globais *(Global methods)* são cálculos de complexidade computacional significativamente maiores do que métodos locais (*local methods*). Existe mais de uma abordagem para métodos globais, porém a abordagem utilizada pelo algoritmo do OPENCV é por programação dinâmica (*dynamic programming*). Programação dinâmica é um método matemático que reduz a complexidade computacional do problema de disparidade decompondo o problema de disparidade original em subproblemas pequenos e simples. Os algoritmos de correspondência baseados em programação dinâmica não tratam a suavidade localmente como nos métodos locais. Em vez disso, a suavidade é tratada globalmente, o que na programação dinâmica significa que a continuidade é considerada dentro das linhas de busca horizontais. Os algoritmos de programação dinâmica são caracterizados como computacionalmente eficientes e capazes de identificar explicitamente oclusões na imagem esquerda e direita. Sua principal fraqueza é a consequência do tratamento unidimensional da suavidade. A Figura 15 ilustra o mapa de disparidade obtido com o método *Semi Global Matching* do OPENCV da imagem da Figura 9.

Figura 15– Mapa de disparidade obtido pelo método Semi-Global Matching da API do OPENCV.

6.3. Comparação entre os algoritmos

O algoritmo desenvolvido baseado na abordagem utilizando *K-Means* apresenta um resultado mais satisfatório para a imagem da Figura 9 do que o resultado utilizando o método de SGM da API do OPENCV, como pode ser visto comparando as imagens das Figuras 14 e 15.

No entanto, quando o resultado foi obtido a partir das imagens capturadas pela câmera utilizada neste projeto, o método semi-global apresentou uma melhor correspondência. Isso se deve, como mencionado anteriormente, ao fato do algoritmo baseado em *K-Means* utilizar as cores das imagens como parâmetro (mais especificamente, a luminosidade) enquanto que a câmera estéreo comprada possui uma diferença significativa nas representações das cores entre as câmeras esquerda e direita. A Figura 16 mostra a imagem capturada de um objeto real pela câmera estéreo desse projeto, enquanto que as Figuras 17 e 18 mostram, respectivamente, o mapa de disparidade produzido pelo algoritmo baseado em *K-Means* para este projeto e mapa de disparidade resultante do método semi-global do OPENCV.

Figura 16– Imagem capturada de um objeto pela câmera estéreo utilizada neste projeto.

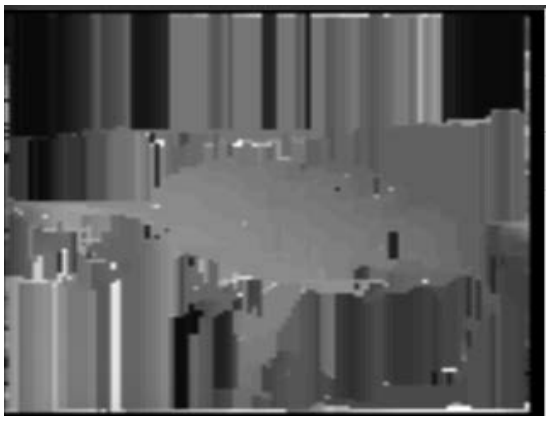

Figura 17 – Mapa de disparidade obtido pelo algoritmo desenvolvido utilizando abordagem de K-Means.

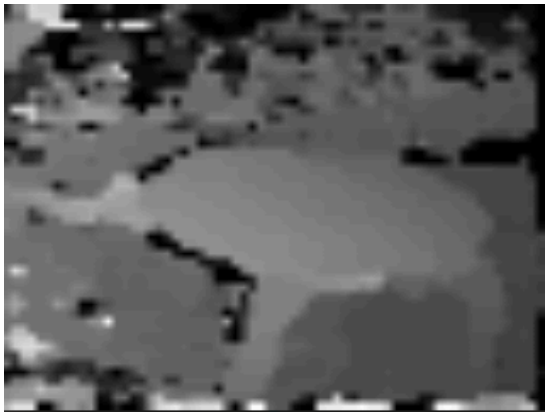

Figura 18 – Mapa de disparidade obtido pelo método Semi-Global Matching da API do OPENCV.

Outro fator importante foi o tempo de execução dos dois algoritmos para o cálculo da disparidade quando executados no *Raspberry Pi model 3*. O método semi-global apresentou um tempo de execução médio de 0.8 segundos no microprocessador, enquanto que o algoritmo baseado em *K-Means* resulta em tempo de cerca de 1.4 segundos.

Os dois algoritmos foram implementados no projeto, podendo os dois serem escolhidos para execução nas configurações de execução do programa. No entanto, o método semi-global foi o definido como o padrão para ser implementado devido ao menor tempo de execução e por não ter influência da variação das cores das câmeras esquerda e direita da câmera estéreo.

8 SONORIZAÇÃO

Nesse projeto foi desenvolvido um algoritmo de sonorização com o auxílio da biblioteca OPENAL baseado no algoritmo de MEIJER (1992). A primeira parte do algoritmo usado nesse projeto consiste em comprimir o mapa de disparidade obtido pelo algoritmo anterior de 382x288 para 64x48 *pixels*. O algoritmo permite que as imagens possam ser sonorizadas em diferentes tempos de sonorização. Por exemplo, definindo o tempo de sonorização como 2 segundos,cada uma das 64 colunas da imagem comprimida é então sonorizada por 0,03125 segundos sequencialmente.

O protocolo de sonorização utilizado neste projeto foi desenvolvido para funcionar da seguinte forma: primeiramente é definido um sistema de coordenadas no mapa de disparidade como pode ser visto Figura 19. Desta forma a coordenada *x* de cada *pixel* é representada pelo instante de tempo em que é sonorizado, de tal forma que o instante zero segundo corresponde aos *pixels* da coluna mais à esquerda da imagem e o instante final, dois segundos,corresponde aos *pixels* da coluna mais à direita da imagem. A coordenada *y* do *pixel* é representada pela frequência do som, de tal forma que quanto mais elevado o *pixel* no eixo *Y*, mais grave o som. Já a distância *z* do objeto da cena à câmera, que é representada pela intensidade luminosa do *pixel*, define a amplitude da onda (volume do som).

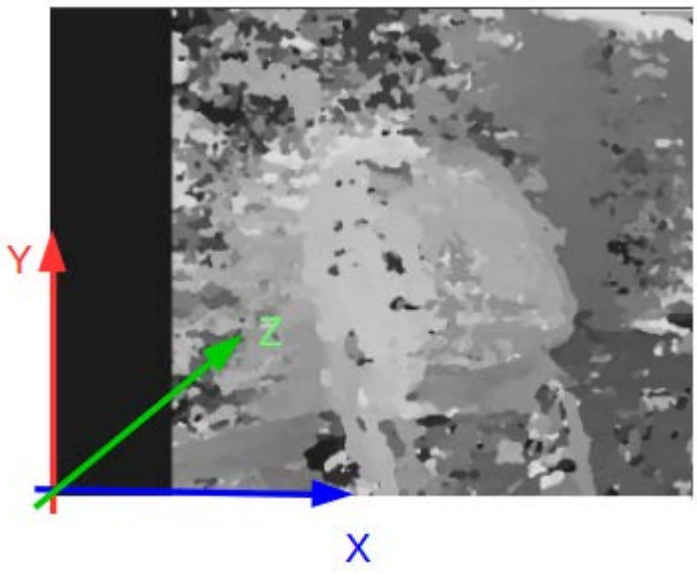

Figura 19–Posição do sistema de coordenadas usado na sonorização em relação ao mapa de disparidade.

A imagem sonorizada correspondente ao mapa de disparidade obtido pelo algoritmo de correspondência estéreo. A disparidade num mapa de disparidade está diretamente relacionada à profundidade do objeto no espaço, como já foi descrito anteriormente. Uma disparidade é representada por um tom de cinza *d* no mapa de disparidade podendo variar de 0 à d_{max} (máxima disparidade contida num mapa de disparidade, um número entre 0 e 255). Estes valores de disparidades, portanto, serão os valores atribuídos à p_{ij} no algoritmo de sonorização.

O algoritmo desenvolvido de transformação da imagem em som baseado no algoritmo de MEIJER (1992) considera uma imagem em tons de cinza com *M* linhas e *N* colunas (em *pixels*). O valor de cada pixel na matriz da imagem p_{ij} corresponde a um tom de cinza entre os G possíveis tons de cinza, i. e.

$$\begin{gathered} p_{\mathrm{ij}} \epsilon \{d_1, d_2, \ldots, d_G\} \\ i = 1,2,\ldots,M \\ j = 1,2,\ldots,N \end{gathered} \tag{17}$$

A conversão em som inicia, selecionando uma coluna por vez das *N* colunas, começando pela coluna mais esquerda com $j = 1$ e terminando pela coluna mais à direita da imagem $j = N$. Cada coluna é então sonorizada por τ segundos consecutivamente fazendo com que a imagem inteira seja sonorizada em *T* segundos, conforme mostrado na Equação 18:

$$\begin{gathered} T = \sum_{j=1}^{N} \tau = N\tau \\ i = 1,2,\ldots,M \\ j = 1,2,\ldots,N \end{gathered} \tag{18}$$

Cada pixel da coluna selecionada é usado para excitar uma onda senoidal de som num intervalo de frequência audível. A frequência de cada pixel depende de sua posição em *i* na coluna, quanto mais baixo um determinado pixel se localiza na coluna, maior será a frequência de sua onda sonora, e quanto mais alto o pixel se localiza mais baixa será a frequência de sua

onda. No algoritmo desenvolvido a frequência de cada pixel f_{ij} na coluna é dada por uma distribuição exponencial:

$$f_{\mathrm{ij}} = f_{\mathrm{min}} \left(\frac{f_{max}}{f_{min}}\right)^{\frac{i-1}{M-1}} \quad i = 1,2,\ldots,M \quad j = 1,2,\ldots,N \tag{19}$$

Onde f_{min}e f_{max}, são as frequências máximas e mínimas respectivamente que devem ser definidas. A onda senoidal que representa um certo pixel na posição $\{i, j\}$é dada pela Equação 20:

$$s_{ij} = a_{ij} sin(2\pi f_{ij} t + \phi_i) \quad i = 1,2,\ldots,M \quad j = 1,2,\ldots,N \tag{20}$$

No algoritmo de sonorização desenvolvido, a intensidade luminosa de um certo *pixel* da imagem de disparidade representa a distância de objetos no espaço e tal intensidade, no algoritmo de sonorização, é representada pelo volume ou amplitude da onda sonora do pixel. A amplitude do som no algoritmo desenvolvido é representado por uma variável do tipo‘*short*’ em C++ que corresponde à um número inteiro entre [-32.767;32.767]. Para isso é preciso converter a intensidade luminosa do pixel em tom de cinza p_{ij} em amplitude da onda a_{ij}. O método de conversão de intensidade luminosa para amplitude da onda utiliza uma relação linear dada pela seguinte equação:

$$a_{ij} = \frac{32767}{d_{max}} p_{ij} \quad i = 1,2,\ldots,M \quad j = 1,2,\ldots,N \tag{21}$$

A sonorização é feita coluna por coluna varrendo a imagem da esquerda para direita, portanto, para o término da construção da onda sonora de uma coluna é preciso somar as ondas correspondentes de cada *pixel* da respectiva coluna antes desta ser sonorizada. Assim a onda

sonora senoidal que representa uma certa coluna na posição *j* é representada pela seguinte relação:

$$s_j = \sum_{i=1}^{M} s_{ij} = \sum_{i=1}^{M} a_{ij}.sin(2\pi f_{ij}.t + \phi_{ij}) \qquad (22)$$
$$i = 1,2,\ldots,M$$
$$j = 1,2,\ldots,N$$

As fases ϕ_i são constantes arbitrárias definidas experimentalmente de tal forma que evitasse a sincronizarão das ondas entre os *pixels* de uma certa coluna.

Na figura abaixo, Figura 20, pode ser visto o princípio do algoritmo de MEIJER (1992) para uma imagem de 8 colunas e 8 linhas (*N*=8 e *M*=8):

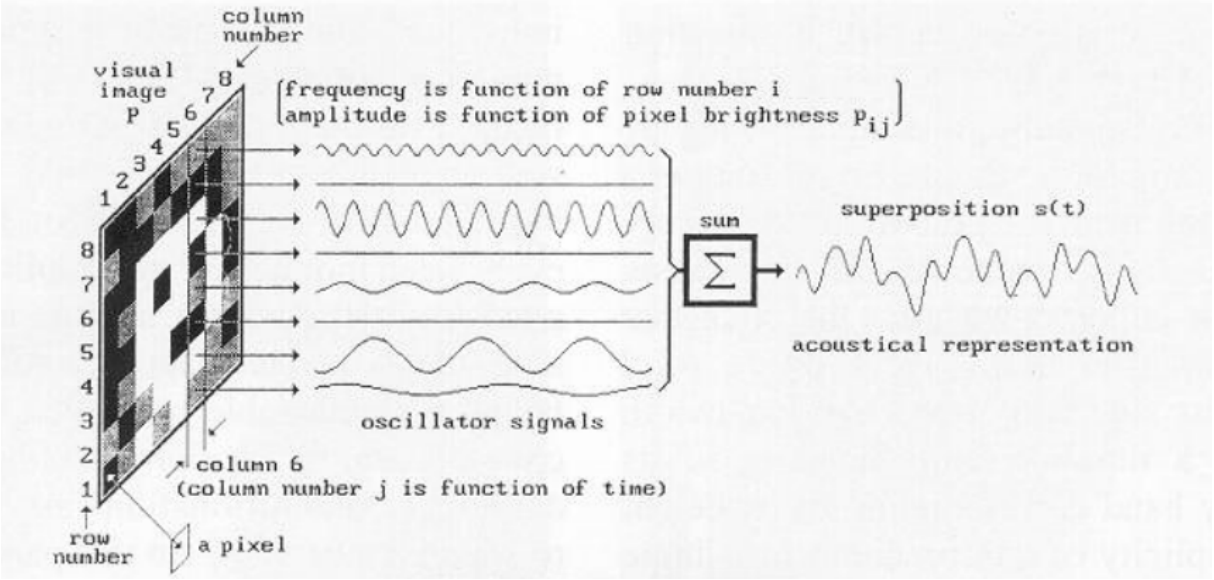

Figura 20 – Princípio do algoritmo de sonorização proposto por MEIJER (1992).

Para o algoritmo de sonorização desenvolvido neste trabalho o mapa de disparidade obtido pelo algoritmo de cálculo da disparidade é comprimido antes de entrar no processo de sonorização, pois com o aumento do número de linhas e colunas, também ocorre o aumento da complexidade computacional e a dificuldade de o usuário assimilar a quantidade de informação transmitida no som. A imagem de disparidade é comprimida para 48 linhas e 64 colunas (*N*=

64 e *M*=48). Além disto, as frequências mínimas e máximas definidas para este projeto são f_{min}= 100Hz e f_{max} = 4000Hz, estas frequências foram definidas a partir de testes experimentais realizados com objetivo de melhorar a qualidade do som, tanto quanto no quesito de reconhecimento nas diferenças de frequências quanto no conforto do som produzido ao deficiente visual. O tempo de sonorização de uma imagem completa pode ser definido pelo programa, uma imagem pode ser confortavelmente assimilada com uma taxa de sonorização de uma imagem por segundo, fazendo com que cada coluna seja sonorizada durante 1/64 segundos. Na Figura 21 é mostrado o modelo de sonorização implementado neste trabalho.

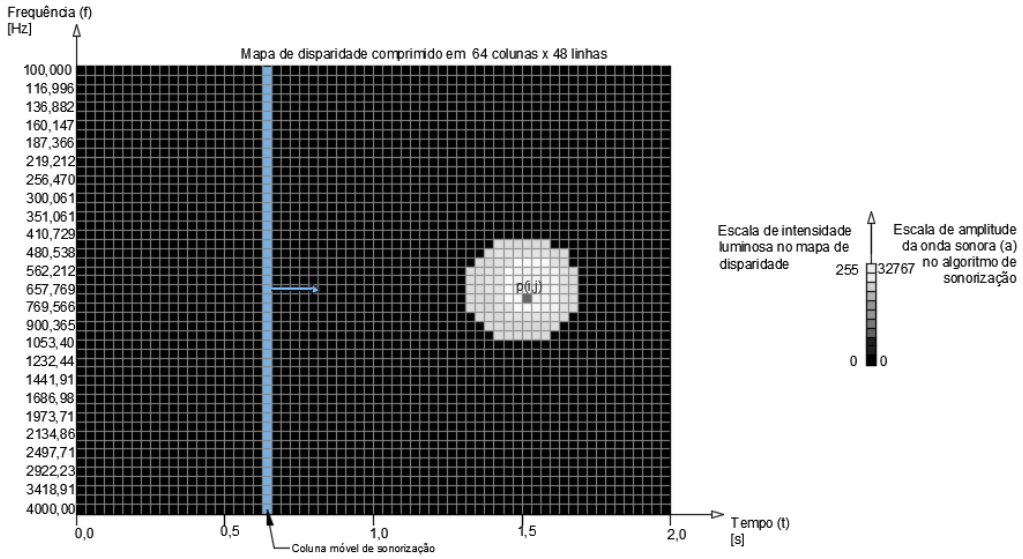

Figura 21- Protocolo do algoritmo de sonorização implementado no atual projeto.

O algoritmo de sonorização foi testado e validado com mapas de disparidade de uma esfera simulada computacionalmente, como pode ser visto nas Figuras 22, 25 e 28. As ondas sonoras produzidas pelo algoritmo de sonorização das Figuras 22, 25 e 28 podem ser vistas nas Figuras 23, 26 e 29 respectivamente e os espectrogramas destas ondas podem ser vistos nas Figuras24, 27 e 30.

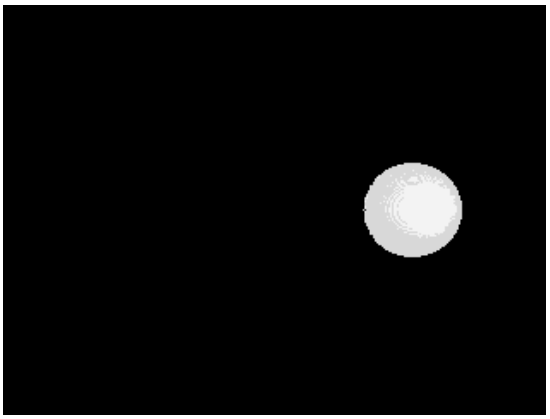

Figura 22- Mapa de disparidade de uma esfera simulada centrada na posição (300, 0, 200) mm.

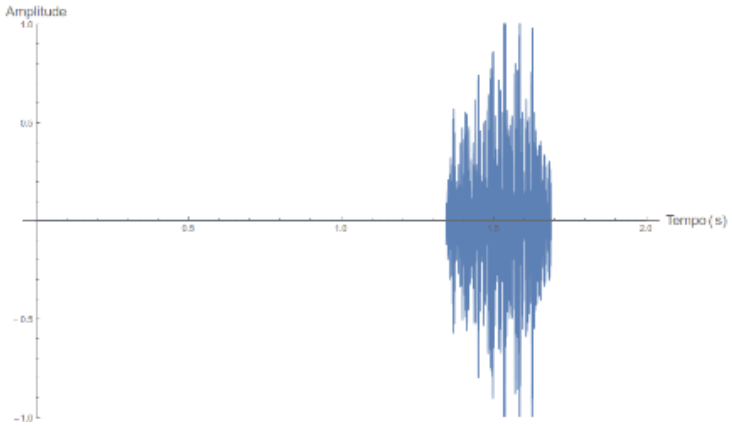

Figura 23 - Onda sonora produzida pelo algoritmo de sonorização para o mapa de disparidade da Figura 22.

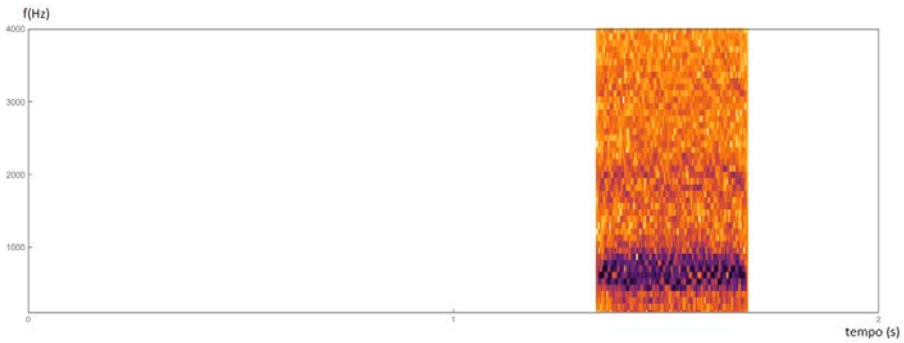

Figura 24 - Espectrograma da onda sonora da Figura 23.

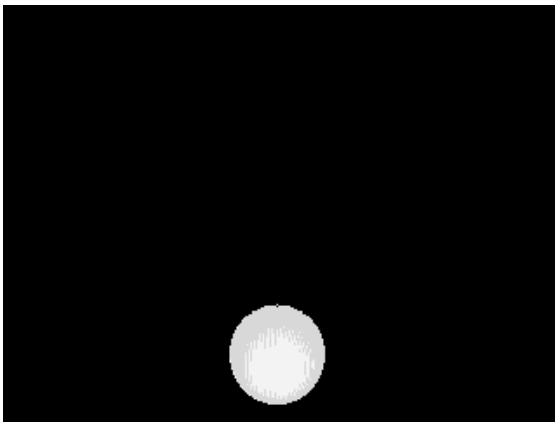

Figura 25 - Mapa de disparidade de uma esfera simulada centrada na posição (0, 300, 200)mm.

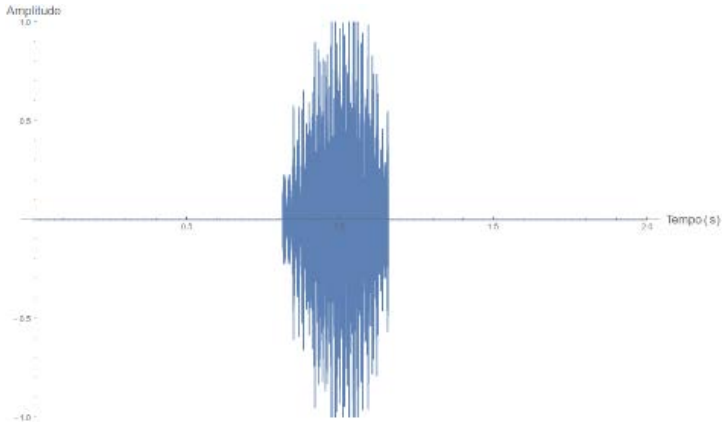

Figura 26 - Onda sonora produzida pelo algoritmo de sonorização para o mapa de disparidade da Figura 25.

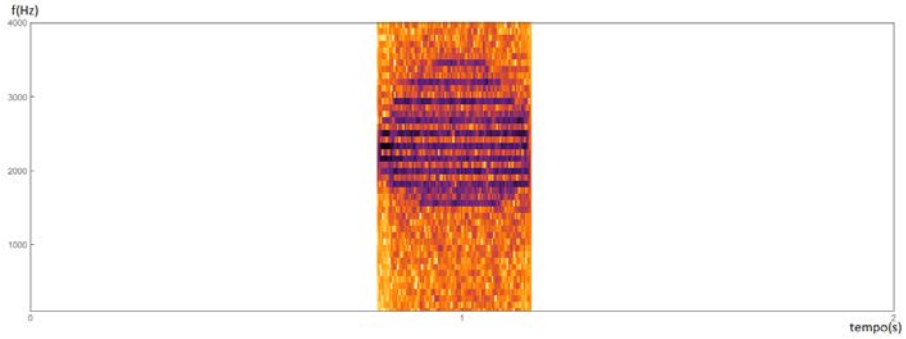

Figura 27 - Espectrograma da onda sonora da Figura 26.

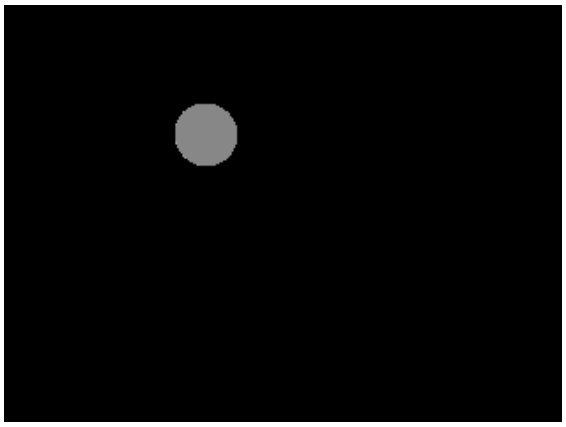

Figura 28 - Mapa de disparidade de uma esfera simulada centrada na posição (-250, -250, -250)mm.

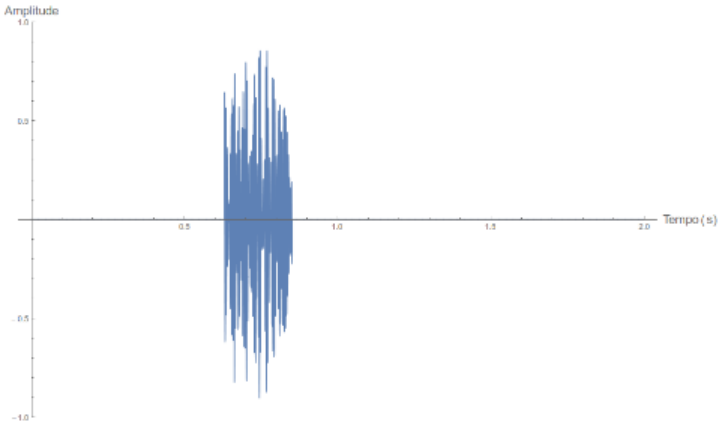

Figura 29 - Onda sonora produzida pelo algoritmo de sonorização para o mapa de disparidade da Figura 28.

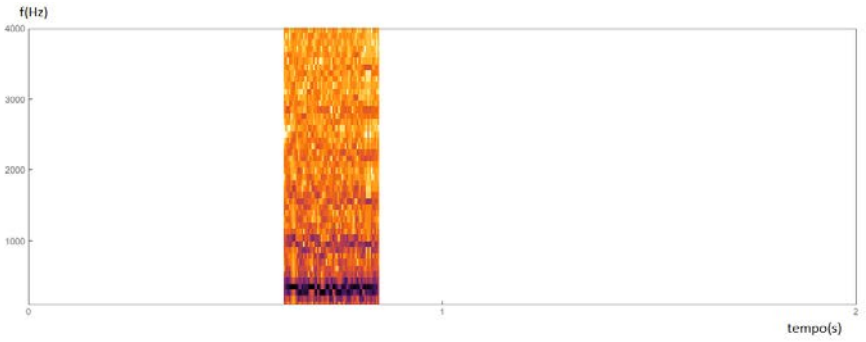

Figura 30 - Espectrograma da onda sonora da Figura 29.

A geração de som a partir das ondas sonoras é realizada utilizando a biblioteca OPENAL, que consiste de uma coletânea de funções para tratamento de sons. e este foi baseado no algoritmo de MEIJER (1992). Após o seu desenvolvimento, esse algoritmo foi testado e validado com uma imagem de disparidade de uma esfera simulada e se obteve resultado satisfatório.

9 PROTÓTIPO

Para a realização deste projeto são necessárias a utilização e a montagem de seis componentes principais: microprocessador, fonte de alimentação, suporte do microprocessador e da fonte, sistema de câmera estéreo, sistema de sonorização, suporte do sistema de câmera estéreo e de sonorização.

O microprocessador é necessário para adquirir as imagens, fazer o cálculo da disparidade das imagens do sistema de câmera estéreo, processar o som a ser sonorizado para o deficiente visual e a coordenar os algoritmos do sistema como um todo. O microprocessador escolhido para atender as necessidades deste projeto foi o *Raspberry Pi* 3® – *Modelo B.*

A fonte de alimentação fornece energia para o funcionamento do microprocessador, essa fonte precisa ser portátil, portanto o modelo escolhido para o projeto foi um uma bateria suplementar externa de telefone celular. Devido ao peso do microprocessador e da fonte de alimentação, ambos ficam afixadas no braço do deficiente visual suportados por uma braçadeira.

O sistema de visão estéreo comprado para o protótipo é um sistema de câmeras estéreo com interface USB modelo KYT-U100-960R1 da *Kayeton Tecnology Co*. A decisão desta compra foi feita visando resultados mais precisos no algoritmo de cálculo da disparidade. A comunicação entre câmera e o microprocessador é feita por um cabo USB.

O sistema de sonorização das imagens de disparidade é composto por um fone de ouvido comum que se comunica com o microprocessador pela saída *Audio Output Jack.*

Tanto o fone de ouvido quanto o sistema de câmera estéreo são suportados por uns óculos projetados para esse protótipo para que o deficiente visual possa usar sem dificultar sua locomoção. Um desenho dos óculos projetados podem ser visto na Figura 31.

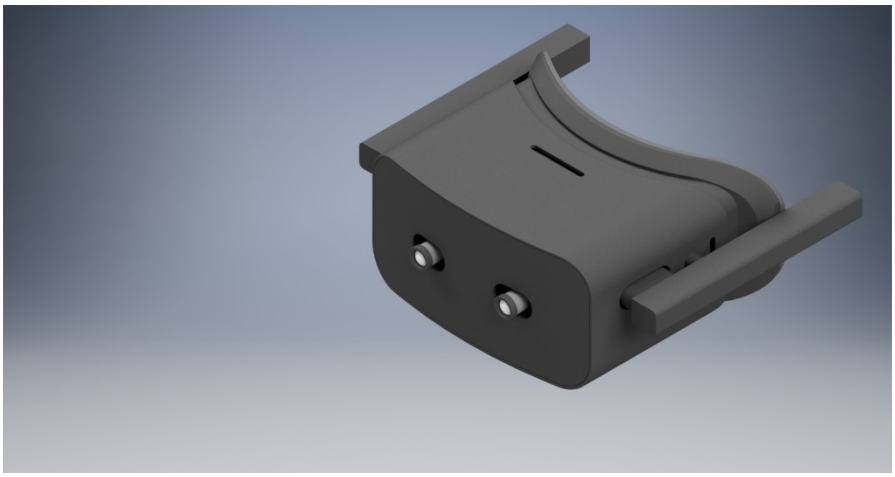

Figura 31 - Óculos projetados para o suporte do sistema estéreo e de sonorização.

Um esquema da montagem do protótipo do dispositivo com todos os componentes sendo usado por um usuário pode ser visualizado na Figura 32.

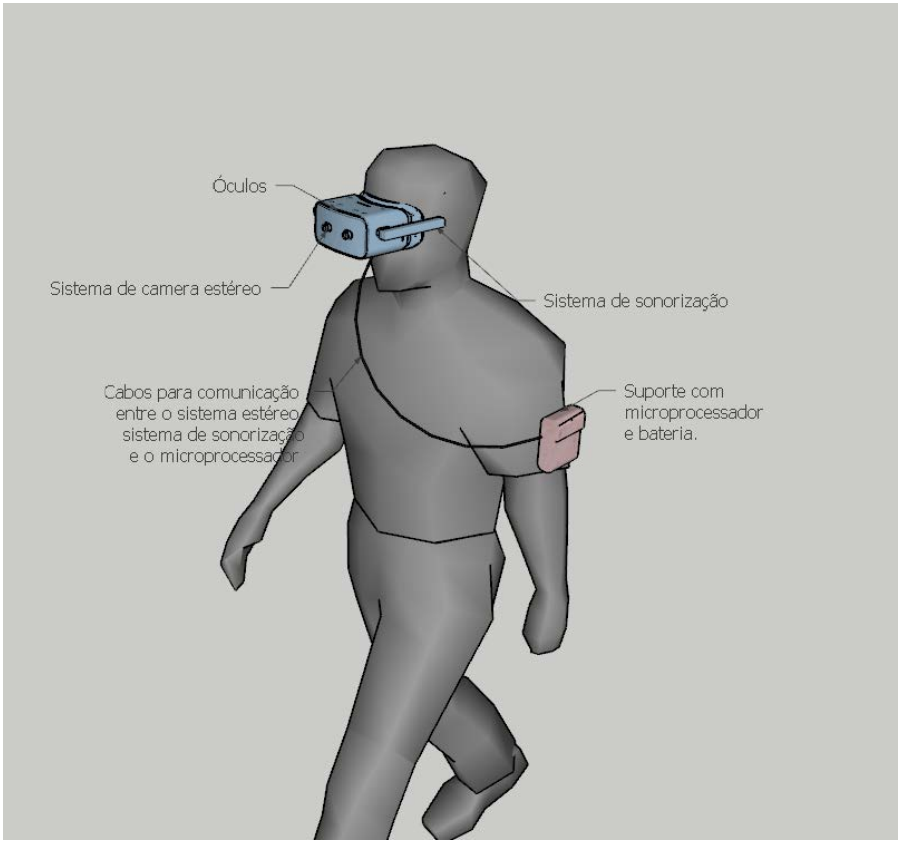

Figura 32 – Esquema do protótipo sendo usado pelo deficiente visual.

Para a construção do protótipo são necessários diversos equipamentos conforme descrito anteriormente. Na Figura 33 pode ser vistos os equipamentos utilizados nesse projeto:

Figura 33 - Equipamento utilizado para construção do protótipo

10 RESULTADOS DO PROTÓTIPO

A proposta desse projeto é de embarcar os sistemas de sonorização e de reconstrução de imagens tridimensionais em um microprocessador portátil e de baixo custo. No presente projeto, esses sistemas estão embarcados em um microprocessador *Raspberry Pi 3® – Model B*, e a relação de processamento dos algoritmos do presente trabalho obtida pelo *Raspberry Pi* e um processador *Intel I7* podem ser vistos na Tabela 1. O tempo médio de processamento dos algoritmos foi obtido através de 100 medidas de intervalo de tempo.

Tabela 2 - Comparação do tempo médio de um ciclo de processamento dos algoritmos de sonorização e de reconstrução tridimensional.

	Tempo médio de um processamento em segundos		
Processador	**Retificação + *Semi Global Matching***	**Retificação + *Local Matching* por *K-Means***	**Sonorização**
Computador com intel I7	0.22	0.56	0.01
Microprocessador *Raspberry PI*	0.86	1.48	0.21

Na Figura 35pode ser visto os resultados de um mapa de disparidade calculado a partir de um par de imagens capturados pelo sistema de visão estéreo Figura 34. Já onda sonora produzida pelo algoritmo de sonorização, com tempo de duração de um segundo, para o mapa de disparidade da Figura 35 pode ser visto na Figura 36 e o espectrograma resultante da respectiva onda sonora, pode ser visto na Figura 37:

Figura 34 - Par de imagens capturados pelo sistema estéreo.

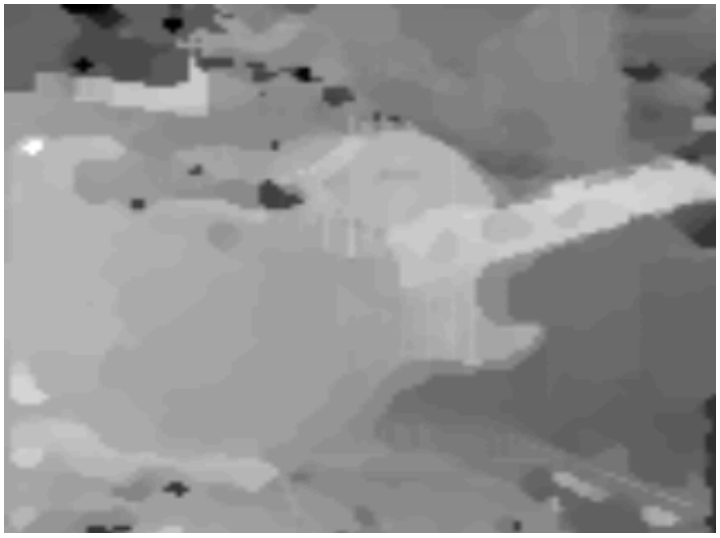

Figura 35 - Mapa de disparidade das imagens da Figura 33, obtido pelo algoritmo de correspondência escolhido para o projeto - SGM.

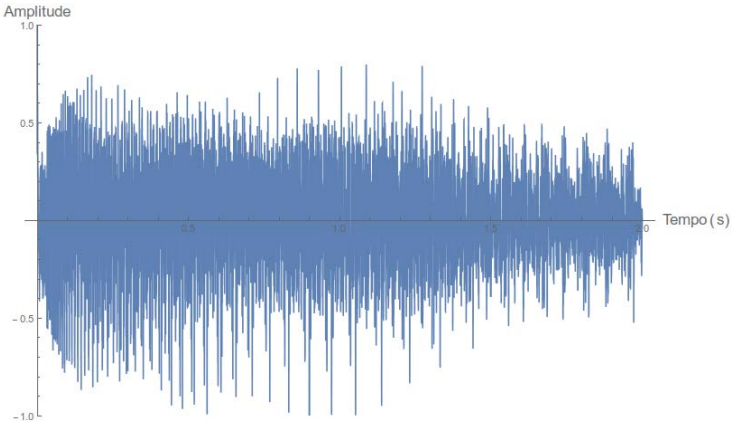

Figura 36 - Onda sonora produzida pelo algoritmo de sonorização, para o mapa de disparidade da Figura 35.

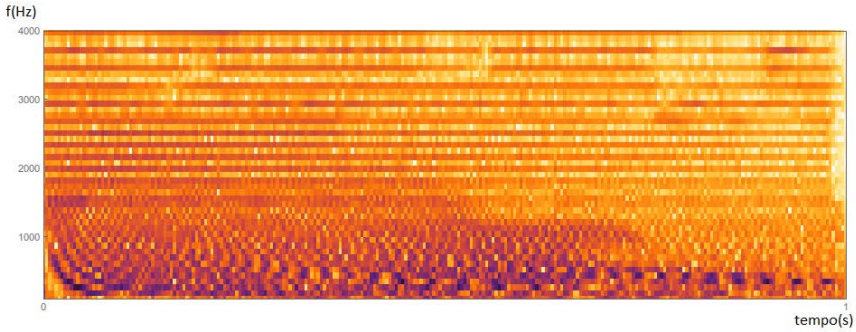

Figura 37 - Espectrograma da onda sonora da Figura 36.

Os requisitos deste trabalho exigem que o protótipo seja de baixo custo, compactos e leves para garantir as características portáteis para o deficiente visual. Na tabela abaixo segue os resultados do protótipo quanto à leveza compactabilidade e custo dos equipamentos do protótipo:

Tabela 3 - Valores de peso, volume e preço dos equipamentos do projeto

Componentes	Peso [g]	Volume útil [cm3]	Preço [R$]
Sistema de áudio	11	-	20
Raspberry Pi 3 Model B	23	392	189
Fonte de alimentação	180		80
Suporte da fonte	22		20
Cabo USB	34	-	320
Sistema de câmeras estéreo	26	~3300	
Óculos	330		135
Total	626	3692	764

11 CONCLUSÃO

Tendo em vista a disponibilidade recursos, pode-se afirmar que os resultados apresentados pelo protótipo foram satisfatórios. Os mapas de disparidade produzidos pelo algoritmo de correspondência são satisfatórios e o algoritmo de sonorização traduziu de forma adequada as informações contidas nos mapas de disparidade.

Considerando os requisitos do projeto tal como à compactabilidade, leveza do dispositivo e custo, pode-se afirmar que foram atendidos. O protótipo apresenta tamanho, peso e custo satisfatórios, além de apresentar uma portabilidade adequada. Em termos de processamento, os algoritmos são executados em um período considerado curto e que não afetam a usabilidade do dispositivo.

O dispositivo, no entanto, possui limitações e melhorias a serem feitas. Embora o mapa de disparidade obtido seja satisfatório, ele apresenta algumas irregularidades que podem ser eliminadas através de um algoritmo melhor. Porém, para que um melhor algoritmo seja implementado sem que o tempo de execução aumente e, assim, prejudique o entendimento do som produzido, seria necessário um microprocessador com maior capacidade de processamento. Sendo assim, um circuito eletrônico dedicado poderia ser utilizado para o processamento dos algoritmos, o que melhoraria o desempenho e portabilidade do dispositivo, já que por ser dedicado, não haveria componentes desnecessários, como acontece com a utilização do *Raspberry Pi*.

Outro problema identificado é na percepção da variação do volume do som sonorizado (informação correspondente à profundidade dos objetos). A variação do volume só é perceptível para curtas distancias, por volta de zero à 60 cm de profundidade, enquanto que a informação importante ao deficiente visual fica entre 40cm à 150 cm de profundidade. Para a próxima etapa deve ser feito um estudo para melhor o algoritmo de disparidade a fim de que o algoritmo de sonorização consiga transmitir de forma adequada as informações de médias e grandes profundidades.

Apesar dos requisitos do protótipo serem atendidos, os óculos são considerados grandes e saída de som é localizada significativamente distante dos ouvidos do deficiente visual trazendo, assim, desconforto ao usuário dos óculos. Etapas posteriores deve ser considerado

melhorias no equipamento de tal forma a se tornar mais compacto e com saída de som mais próxima ao ouvido.

Por fim no presente trabalho o dispositivo foi validado. O protótipo funcionou e mostrou-se ser proveitoso. Sendo assim foi verificado a viabilidade da criação de um novo produto comercial para o mercado para atender a demanda de deficientes visuais.

REFERÊNCIAS BIBLIOGRÁFICAS

BACH-Y-RITA, P. et al (1998). *Form perception with a 49-point electrotactile stimulus array on the tongue: A technical note.* Journal of Rehabilitation Research and Development, Vol. 35, n.4, p.427-430, Outubro 1998. Disponível em: <http://www.rehab.research.va.gov/jour/98/35/4/BACH.pdf>

BAHADIR, S. K.; KONCAR, V.; KALAOGLU, F. (2012). *Wearable obstacle detection system fully integrated to textile structures for visually impaired people. Sensors and Actuators* A., p.297-311, 10 mar. 2012. Disponível em: <http://www.sciencedirect.com/science/journal/09244247>

BANZ, C.; BARIK, A.; HESSELBARTH, S.; et al. (2010). *Real-Time Stereo Vision System using Semi-Global Matching Disparity Estimation: Architecture and FPGA-Implementation*. International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), IEEE, p. 93-101, 2010. Disponível em: <http://ieeexplore.ieee.org/document/5642077/>

BROWN, M.; BURSCHKA, D.; HAGER, G. (2003). *Advance in computational stereo. Transactions on Pattern Analysis and Machine Intelligence*, IEEE, Vol. 25, n.8, p. 993-1008, Agosto 2003. Disponível em: <http://ieeexplore.ieee.org/document/1217603/>, Acesso em 26 de jun. 2017.

CARDIN, S., THALMANN, D., VEXO, F., (2007). *A wearable system for mobility improvement of visually impaired people*. The Visual Computer, Vol. 23, n. 2, p. 109–118, fev. 2007. Disponível em: <http://link.springer.com/article/10.1007/s00371-006-0032-4>

HENRIQUES, J. T.; CAVACO, S.; CORREIA, N. (2014). *See-Through-Sound: Transforming images into sonic representations to help the blind*, Journal of Information Technology Research (JITR), jan. 2014, Disponível em:<http://www.igi-global.com/article/see-through-sound/111252>

HERNANDEZ-JUAREZ, D.; CHACÓN, A.; ESPINOSA, A.; et al. (2016). *Embedded real-time stereo estimation via Semi-Global Matching on the GPU*. The International Conference on Computational Science, ICCS, Vol. 80, p. 143-153, jun. 2016. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050916306561>

HIRSCHMULLER, H. (2008). *Stereo Processing by Semiglobal Matching and Mutual Information*, PAMI(30), n. 2, p. 328-341, Fevereiro 2008.

LEE, C. L.; CHEN, C. Y.; SUNG, C. P.; LU, S. Y. (2014). *Assessment of a simple obstacle detection device for the visually impaired.* Applied Ergonomics, Vol. 45, n. 4, p. 817-824, jul. 2014. Disponivel em: <http://www.sciencedirect.com/science/article/pii/S0003687013002226>

MEIJER, P. B. L (1992). *An Experimental System for Auditory Image Representations.* Transactions on biomedical engineering, IEEE, Vol. 39, n.2, p.112-121, fev. 1992. Disponível em: <http://ieeexplore.ieee.org/document/121642>

MOLTON, N.; SE, S.; BRADY, J. M.; LEE D.; PROBERT D. (1998). *A stereo vision-based aid for the visually impaired.* Image and Vision Computing, Vol. 16, n.4, p. 251-263, mar. 1998. Disponível em : <http://www.sciencedirect.com/science/article/pii/S0262885697000875>

GOSTA, M.; GRGIC, M. (2010). *Accomplishments and Challenges of Computer Stereo Vision.*52nd International Symposium, ELMAR, p.57-64, set. 2010. Disponível em: <http://ieeexplore.ieee.org/document/5606082/?reload=true>

MUKHERJEE, S.; GUDDETI, R. M. R. (2014). *A Hybrid Algorithm for Disparity Calculation From Sparse Disparity Estimates Based on Stereo Vision*. International Conference on Signal Processing and Communications (SPCOM), IEEE, jul. 2014. Disponível em: <http://ieeexplore.ieee.org/document/6983949/>

SHOVAL, S.; ULRICH, I.; BORENSTEIN, J. (2003). *NavBelt and the Guide-Cane, obstacle-avoidance systems for the blind and visually impaired*. Robotics and Automation Magazine, Special Issue on Robotics in Bio-Engineering, IEEE, Vol. 10, n.1, p. 9-20, Março 2003. Disponível em : <http://ieeexplore.ieee.org/document/1191706/>

TSAI, R. Y. (1987). A Versatile Camera Calibration Techniaue for High-Accuracy 3D Machine Vision Metrology Using Off-the-shelf TV Cameras and Lenses, Journal of robotics and automation, IEEE, Vol. 3, n.4, p. 323-344, 1987. Disponível em: <https://www.vision.caltech.edu/bouguetj/calib_doc/papers/Tsai.pdf>

VALSARAJ, A.; BARIK, A.; et al. (2016). *Stereo Vision System implemented on FPGA*. Procedia Technology,Vol.24, p. 1105-1112, jan. 2016. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2212017316303346>

ZHANG, X. et al. (2014). *An Efficient Method of Image-Sound Conversion Based on IFFT for Vision Aid for the Blind*. Lecture Notes on Software Engineering, Vol. 2, n.1, p.54-57, fev. 2014. Disponível em: <http://www.lnse.org/papers/94-VC3007.pdf>

BOUGUET, J. (2015). *Camera Calibration Toolbox for Matlab*. Disponível em: <http://www.vision.caltech.edu/bouguetj/calib_doc/>, Acesso em 26 de jun. 2017.

BRAINPORT TECHNOLOGIES WICAB, INC. *History* (2016a). Disponível em: <http://www.wicab.com/history>, Acesso em: 29 de out. 2016.

BRAINPORT TECHNOLOGIES WICAB, INC *BrainPort V100 Technology* (2016b). Disponível em: <http://www.wicab.com/brainport-v100-technology/>, Acesso em: 29 de out. 2016.

KAYETON, KAYETON. Stereo USB2.0 Camera KYT-U100-960R1 (2017). Disponível em:<http://www.kayetoncctv.com/product/960p-aptina-ar0130-dual-lens-usb-camera module/>, Acesso em 24 de set. 2017.

OPENCV DOCUMENTATION, *Camera Calibration and 3D Reconstruction* (2017). Disponível em: <https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html>

RASPBERRYPI, RASPBERRY PI ORGANIZATION. *Products – Raspberry-Pi-3-Model-b* (2017). Disponível em: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>, Acesso em 24 de set. 2017.

SCHARSTEIN, D.; HIRSCHMÜLLER, H.; KITAJIMA, Y.; KRATHWOHL, G.; NESIC, N.; WANG, X.; WESTLING, P. (2014). *High-resolution stereo datasets with subpixel-accurate ground truth. In German Conference on Pattern Recognition (GCPR 2014)*, Münster, Germany, September 2014. Disponível em <http://vision.middlebury.edu/stereo/>, Acesso em 26 de jun. 2017.

WHO, WORLD HEALTH ORGANIZATION. *Visual impairment and blindness* (2014). Disponível em: <http://www.who.int/mediacentre/factsheets/fs282/en/>, Acesso em 24 de nov. 2016.

APÊNDICE A – CÓDIGO DOS ALGORITMOS EM C++

MAIN FILE:

```
/*

 * File:   main.cpp
 * Author: erico ; thiago
* Created on 10 de Janeiro de 2017, 17:25
*/

#include <cstdlib>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <opencv2/core/core.hpp>
#include "matching/StereoDisp.hpp"
#include "rectify/config/CalibParameters.hpp"
#include "matching/config/MatchingConfig.hpp"
#include "rectify/Rectify.hpp"
#include "sonar/Sonar.hpp"
#include <time.h>
#include <cmath>
#include <opencv2/ximgproc.hpp>
#include "config/InitialConfig.hpp"

usingnamespace std;

void executeSVS(){

// declaration of Maps for rectification
    cv::Mat mapL[2];
    cv::Mat mapR[2];
// declaration of calibration parameters
    CalibParameters cp;
// declaration of variable of end loop condition
bool exitloop =false;
int stop=0;
int sph =3;

//declaration of videos capture
    cv::VideoCapture cap;
    cap.set(CV_CAP_PROP_FRAME_WIDTH,1280);
    cap.set(CV_CAP_PROP_FRAME_HEIGHT,480);
    cap.set(CV_CAP_PROP_BUFFERSIZE,1);
    cap.set(CV_CAP_PROP_FPS,10);

//try to use camera
if(!cap.open(1))
return;

//read data from calibration file.
    cp.readCalibDataFile("calib_file/Calib_Result_stereo.yaml");

//calculate rectification maps.
    sv::stereoRectifyMaps(cp.camera_matrix_l_old,
            cp.distortion_coefficients_l,
            cp.camera_matrix_r_old,
            cp.distortion_coefficients_r,
            cp.R,
            cp.T,
```

```
            cv::Size(640,480),
            mapL[0],
            mapL[1],
            mapR[0],
            mapR[1]
);

//loop for sv:
while(!exitloop){
//declaration of Mats
        cv::Mat img;
        cv::Mat im_left_rect;
        cv::Mat im_right_rect;
        cv::Mat left_disp;
        cv::Mat right_disp;
        cv::Mat filtered_disp;
        cv::Mat mix_filtered_left_disp;
        cv::Mat kmeans_disp;
        cv::Mat disp_comp;

//take a picture with a cam,it needs to be five per time.
for(int i =0; i <5; i++)
            cap.read(img);
if(img.empty())break;

//use file of sphere images as disparity map
if(SPH == ON){
            string s ="/TCC/tcc/es/"+ std::to_string(sph)+".png";
            left_disp = cv::imread(s, CV_LOAD_IMAGE_GRAYSCALE);
            left_disp=left_disp/16;
            left_disp.convertTo(left_disp, CV_8U);
            cv::imshow("sphere",left_disp*16);
            cv::waitKey(10);
            sph++;
if(sph ==4){
                sph =1;
}
}

else{
//this block make the rectification with calibration parameters
if(RECT_BY_PARAM == ON){
int offset =0;
float angle =0;
if(RECT_MANUAL == ON){
offset = OFFSET_RECT ;
                    angle= ANGLE_RECT;
}
                cv::Rect leftROI(0, offset, img.cols /2,  img.rows - offset);
                cv::Rect rightROI(leftROI.width,0, img.cols - leftROI.width , img.rows - offset);
                im_right_rect = sv::StereoRectifyImg(img(rightROI).clone(), mapR);
                im_left_rect = sv::StereoRectifyImg(img(leftROI).clone(), mapL);

                cv::Point2f center(0,0);
                cv:: Mat r = cv::getRotationMatrix2D(center, angle,1.0);
                cv::warpAffine(im_right_rect, im_right_rect, r, im_right_rect.size());
}
//this block make the rectification with manual adjustments .
if(RECT_MANUAL == ON && RECT_BY_PARAM == OFF){
int offset = OFFSET_RECT;//offset of pixels for rectification;
                cv::Rect leftROI(0, offset, img.cols /2, img.rows - offset);
                cv::Rect rightROI(leftROI.width,0, img.cols - leftROI.width, img.rows - offset);
                im_left_rect = img(leftROI).clone();
                im_right_rect = img(rightROI).clone();
```

```
                cv::Point2f center(0,0);
                cv:: Mat r = cv::getRotationMatrix2D(center, ANGLE_RECT,1.0);
                cv::warpAffine(im_right_rect, im_right_rect, r, im_right_rect.size());
}
//resize rectified images for disparity calculation
            cv::resize(im_right_rect, im_right_rect, cv::Size(IMAGE_SIZE_WIDTH, IMAGE_SIZE_HEIGHT));
            cv::resize(im_left_rect, im_left_rect, cv::Size(IMAGE_SIZE_WIDTH, IMAGE_SIZE_HEIGHT));


// this block use semi global matching method for disparity calculation
if(SEMI_GLOBAL_MATCHING == ON){
                cv::Ptr<cv::StereoSGBM> left_matcher = cv::StereoSGBM::create(MIN_DISP, MAX_DISP, WIN_SIZE,24*
WIN_SIZE*WIN_SIZE,24* WIN_SIZE*WIN_SIZE, WIN_SIZE, PREFILTER_CAP, UNIQUENESS_RATIO, SPECKLE_WINDOW_SIZE,
SPECKLE_RANGE,false);
                left_matcher-> compute(im_left_rect, im_right_rect, left_disp);
// this block use a wls_filter
if(WLS_FILTER == ON){
                    cv::Ptr<cv::StereoSGBM> right_matcher = cv::StereoSGBM::create(MIN_DISP, MAX_DISP, WIN_SIZE,24*
WIN_SIZE*WIN_SIZE,24* WIN_SIZE*WIN_SIZE, WIN_SIZE, PREFILTER_CAP, UNIQUENESS_RATIO, SPECKLE_WINDOW_SIZE,
SPECKLE_RANGE,false);
                    right_matcher->compute(im_right_rect, im_left_rect, right_disp);

                    cv::Ptr<cv::ximgproc::DisparityWLSFilter> wls_filter;
                    wls_filter = cv::ximgproc::createDisparityWLSFilter(left_matcher);


                    wls_filter->setLambda(LAMBDA);
                    wls_filter->setSigmaColor(SIGMA_COLOR);
                    wls_filter->filter(left_disp, im_left_rect, filtered_disp, right_disp);

                    filtered_disp = filtered_disp /16;
                    filtered_disp.convertTo(filtered_disp, CV_8U);
                    cv::Rect roif(MAX_DISP,0, filtered_disp.cols - MAX_DISP, filtered_disp.rows-10);
                    filtered_disp = filtered_disp(roif).clone();

                    right_disp = right_disp /16;
                    right_disp.convertTo(right_disp, CV_8U);
}
                left_disp = left_disp /16;
                left_disp.convertTo(left_disp, CV_8U);
                cv::Rect roi(MAX_DISP,0, left_disp.cols - MAX_DISP, left_disp.rows-10);
                left_disp = left_disp(roi).clone();
               cv::medianBlur(left_disp,left_disp,MB_I);
}
// this block use kmeans method for disparity calculation
if(KMEANS_MATCHING == ON){
                kmeans_disp = sv::stereoDisp(im_left_rect, im_right_rect, K, MAX_DISP, WIN_SIZE, DISP_SCALE);
                cv::Rect roikm(MAX_DISP,0, kmeans_disp.cols - MAX_DISP, kmeans_disp.rows-10);
                kmeans_disp = kmeans_disp(roikm).clone();
                cv::medianBlur(kmeans_disp,kmeans_disp,MB_I);
}
// this block use kmeans method for disparity calculation
if(MIX_FILTER_DISP == ON){
                mix_filtered_left_disp =(0.8* filtered_disp +.8* left_disp);
}
//this block makes the compression of disparity before image show
if(COMPRESS_DISP_IMSHOW == ON){
                cv::resize(filtered_disp, filtered_disp, cv::Size(COMPRESS_IMAGE_SIZE_WIDTH, COMPRESS_IMAGE_SIZE_HEIGHT));
                cv::resize(left_disp, left_disp, cv::Size(COMPRESS_IMAGE_SIZE_WIDTH, COMPRESS_IMAGE_SIZE_HEIGHT));
}
//this block shows images
if(SHOW_IM_RECT == ON){
                cv::imshow("im_right_rect", im_right_rect);
```

```
                cv::waitKey(10);
                cv::imshow("im_left_rect", im_left_rect);
                cv::waitKey(10);
}
if(SHOW_IM_SGBM == ON){
                cv::resize(left_disp, left_disp, cv::Size(IMAGE_SIZE_WIDTH*2, IMAGE_SIZE_HEIGHT*2));
                cv::imshow("SGBM left disparity", left_disp * DISP_SHOW_SCALE);
                cv::waitKey(10);
}
if(SHOW_IM_KMEANS_MATCHING == ON){
                cv::resize(kmeans_disp, kmeans_disp, cv::Size(IMAGE_SIZE_WIDTH*2, IMAGE_SIZE_HEIGHT*2));
                cv::imshow("kmeans left disparity", kmeans_disp * DISP_SHOW_SCALE);
                cv::waitKey(10);
}
if(SHOW_IM_FILTERED == ON){
                cv::resize(filtered_disp, filtered_disp, cv::Size(IMAGE_SIZE_WIDTH*2, IMAGE_SIZE_HEIGHT*2));
                cv::imshow("filtered_disparity", filtered_disp * DISP_SHOW_SCALE);
                cv::waitKey(10);
}
if(SHOW_IM_MIX_F_D == ON){
                cv::resize(mix_filtered_left_disp, mix_filtered_left_disp, cv::Size(IMAGE_SIZE_WIDTH*2,
IMAGE_SIZE_HEIGHT*2));
                cv::imshow("mix filtered-left disparity", mix_filtered_left_disp * DISP_SHOW_SCALE);
                cv::waitKey(10);
}
}
//this compress image before sonar
        cv::resize(mix_filtered_left_disp, disp_comp, cv::Size( COMPRESS_IMAGE_SIZE_WIDTH, COMPRESS_IMAGE_SIZE_HEIGHT));
double max_disp_comp;
double min_disp_comp;
        cv::minMaxLoc(disp_comp,&min_disp_comp,&max_disp_comp);
if(SONAR == ON){
//sonar
            sv::sonar(disp_comp, SOUND_TIME, max_disp_comp);
}
if(stop ==1)
            exitloop =true;
}
    sv::exit_al();

}
int main(int argc,char** argv){
clock_t Ticks[2];
    Ticks[0]= clock();
    executeSVS();
    Ticks[1]= clock();
double Tempo =(Ticks[1]- Ticks[0])*1000.0/ CLOCKS_PER_SEC;
printf("Tempo gasto: %g ms.", Tempo);
return0;

}
```

STEREO DISP FILE:

```
/*
 * File:   StereoDisp.cpp
* Author: erico ; thiago
 *
 * Created on 10 de Janeiro de 2017, 17:29
*/

#include "StereoDisp.hpp"

namespace sv {
```

```
    cv::Mat stereoDisp(cv::Mat he_l, cv::Mat he_r,int K,int max_disp,int win_size,int disp_scale){
// Convert the left image from rgb to lab.
        cv::Mat lab_he_l;
        cvtColor(he_l, lab_he_l, cv::COLOR_BGR2Lab);

// Split the channels of left image.
        std::vector<cv::Mat> lab_channels_l;
        cv::split(lab_he_l, lab_channels_l);

// Get the intensity L of LAB from left image.
        cv::Mat Int_l_g(lab_channels_l[0]);
// Convert the right image from rgb to lab.
        cv::Mat lab_he_r;
        cvtColor(he_r, lab_he_r, cv::COLOR_BGR2Lab);


// Split the channels of right image.
        std::vector<cv::Mat> lab_channels_r;
        cv::split(lab_he_r, lab_channels_r);

// Get the intensity L of LAB from right image.
        cv::Mat Int_r_g(lab_channels_r[0]);

// Initialize remaining parameters
double win_lim =(int) win_size /2;

// Compress images;
        cv::Mat Int_l;
//  cv::resize(Int_l_g,Int_l ,cv::Size(),compress_factor,compress_factor);
        Int_l=Int_l_g;

        cv::Mat Int_r;
//   cv::resize(Int_r_g,Int_r,cv::Size(),compress_factor,compress_factor);
        Int_r=Int_r_g;


// Classify the Intensities in 'L' Space Using K-Means Clustering
        cv::Mat leftImage = fastCMeans(Int_l, K);


// Find the dimensions of the left image
int nrLeft = leftImage.rows;
int ncLeft = leftImage.cols;

// Initialize the disparity map for the left image
        cv::Mat disp_map = cv::Mat::zeros(nrLeft, ncLeft, CV_32F);

// Determine segment boundaries of the left image
        cv::Mat segb_left = cv::Mat::zeros(nrLeft -1, ncLeft -1, CV_32F);
for(int i =0; i < nrLeft -2; i++){
for(int j =1; j < ncLeft -2; j++)
if(leftImage.at<int8_t>(i, j)!= leftImage.at<int8_t>(i, j +1)||
                        leftImage.at<int8_t>(i, j)!= leftImage.at<int8_t>(i +1, j +1)||
                        leftImage.at<int8_t>(i, j)!= leftImage.at<int8_t>(i +1, j)||
                        leftImage.at<int8_t>(i, j)!= leftImage.at<int8_t>(i +1, j -1))
                    segb_left.at<float>(i, j)=255;
}
        bwmorph(segb_left,1,100);
        bwmorph(segb_left,2,100);
//Calculate disparities of left image pixels lying on segment boundaries
for(int i = win_lim +1-1; i < nrLeft - win_lim -1; i++)
for(int j = win_lim +1+ max_disp -1; j < ncLeft - win_lim -1-1; j++)
if(segb_left.at<float>(i, j)>0){
                    disp_map.at<float>(i, j)= pixelDisp(Int_r, i, win_lim, j, Int_l, max_disp, win_size)* disp_scale;
```

```
}
// Replicate segment boundary disparities row-wise
for(int i = win_lim +1-1; i <= nrLeft -1- win_lim -1; i++){
int last_idx = win_lim +1+ max_disp -1;
//Determine disparity of first pixel in row
int j = last_idx;
            disp_map.at<float>(i, j)= pixelDisp(Int_r, i, win_lim, j, Int_l, max_disp, win_size)* disp_scale;
float last_disp = disp_map.at<float>(i, last_idx);
            last_idx++;
for(j = win_lim +2+ max_disp -1; j <= ncLeft -1- win_lim -1-1; j++){
if(j == ncLeft -1- win_lim -1-1){
                    disp_map.at<float>(i, j)= pixelDisp(Int_r, i, win_lim, j, Int_l, max_disp, win_size)* disp_scale;
continue;
}
// Determine disparity of other pixels
if(segb_left.at<float>(i, j)>0){
double rep_size = j - last_idx;
if(last_idx > win_lim +2+ max_disp -1)
if(disp_map.at<float>(i, j)== last_disp){
for(int index = last_idx; index <= j -1; index++)
                                disp_map.at<float>(i, index)= last_disp;
}
                    last_disp = disp_map.at<float>(i, j);
                    last_idx = j +1;
}
}
}
// Determine disparities of all remaining pixels of first and last row
for(int j = win_lim +2+ max_disp -1; j <= ncLeft -1- win_lim -2-1; j++){
if(disp_map.at<float>(win_lim +1-1, j)==0)
                disp_map.at<float>(win_lim +1-1, j)= pixelDisp(Int_r, win_lim +1-1, win_lim, j, Int_l, max_disp, win_size)*
disp_scale;
if(disp_map.at<float>(nrLeft -1- win_lim -1, j)==0)
                disp_map.at<float>(nrLeft -1- win_lim -1, j)= pixelDisp(Int_r, nrLeft -1- win_lim -1, win_lim, j, Int_l,
max_disp, win_size)* disp_scale;
}

// Disparity map reconstruction based on already determined disparities
// Determine boundaries separating regions in disparity map which have been
// already populated from regions where disparities are yet to be determined
        cv::Mat bin_perim = disp_map.clone();
        bwmorph(bin_perim,2,1000000);

int n_rows = bin_perim.rows;
int n_cols = bin_perim.cols;

cv::Mat per_top = cv::Mat::zeros(n_rows, n_cols, CV_32F);
        cv::Mat per_bottom = cv::Mat::zeros(n_rows, n_cols, CV_32F);

int copy_idx =0;

// For every pixel in the disparity map whose disparity has not yet been
// determined, find its nearest pixel whose disparity has been already
// determined, and which lies ABOVE the said pixel in the same column

for(int j =1-1; j <= n_cols -1; j++)
for(int i =1-1; i <= n_rows -1; i++){
if(bin_perim.at<float>(i, j)==0.0)
                    per_top.at<float>(i, j)= copy_idx;
else
                    copy_idx = i;
}

// For every pixel in the disparity map whose disparity has not yet been
```

```
// determined, find its nearest pixel whose disparity has been already
// determined, and which lies BELOW the said pixel in the same column
for(int j =1-1; j <= n_cols -1; j++)
for(int i = n_rows -1; i >=1-1; i--){
if(bin_perim.at<float>(i, j)==0)
                    per_bottom.at<float>(i, j)= copy_idx;
else
                    copy_idx = i;
}

        disp_map = disp_map / disp_scale;

for(int i = win_lim +1-1; i <= nrLeft -1- win_lim -1; i++)
for(int j = win_lim +2+ max_disp -1; j <= ncLeft -1- win_lim -1-1; j++)
if(disp_map.at<float>(i, j)==0){
float a = disp_map.at<float>(per_top.at<float>(i, j), j);
float b = disp_map.at<float>(per_bottom.at<float>(i, j), j);
disp_map.at<float>(i, j)= b < a ? b : a;
}

        disp_map = disp_map*disp_scale;
        disp_map.convertTo(disp_map, CV_8U);
//return matrix;
return disp_map;
}
void bwmorph(cv::Mat &im,int action,int n){
int stop_flag =1;
if(n == INF) n =500;
int rows = im.rows;
int cols = im.cols;
int _im[rows][cols];

// set all to zeros
for(int i =0; i < rows; i++)
for(int j =0; j < cols; j++)
                _im[i][j]=0;
//FILL
if(action ==1){
for(int iterator =0; iterator < n; iterator++){
if(stop_flag >0)
                    stop_flag =0;
else
break;
for(int i =1; i < im.rows -1; i++)
for(int j =1; j < im.cols -1; j++)
if(im.at<float>(i, j)==0&&
(float)255== im.at<float>(i +1, j -1)&&
(float)255== im.at<float>(i +1, j)&&
(float)255== im.at<float>(i +1, j +1)&&
(float)255== im.at<float>(i, j -1)&&
(float)255== im.at<float>(i, j +1)&&
(float)255== im.at<float>(i -1, j -1)&&
(float)255== im.at<float>(i -1, j)&&
(float)255== im.at<float>(i -1, j +1)){
                            _im[i][j]=1;
                            stop_flag++;
}
for(int i =0; i < rows; i++)
for(int j =0; j < cols; j++)
if(_im[i][j]==1)
                            im.at<float>(i, j)=(float)255;
}
}
// REMOVE
```

```
elseif(action ==2){
for(int iterator =0; iterator < n; iterator++){
if(stop_flag >0)
                    stop_flag =0;
else
break;
for(int i =1; i < im.rows -1; i++)
for(int j =1; j < im.cols -1; j++)
if(im.at<float>(i, j)!=(float)0&&
                                im.at<float>(i +1, j)!=(float)0&&
                                im.at<float>(i -1, j)!=(float)0&&
im.at<float>(i, j +1)!=(float)0&&
                                im.at<float>(i, j -1)!=(float)0){
                            _im[i][j]=1;
stop_flag++;
}
for(int i =0; i < rows; i++)
for(int j =0; j < cols; j++)
if(_im[i][j]==1)
                            im.at<float>(i, j)=0;
}
}
}
// Function to calculate disparity of a given pixel using SAD cost function
int pixelDisp(cv::Mat Int_r,int i,int win_lim,int j, cv::Mat Int_l,int max_disp,int win_size){
        cv::Mat B = Int_l(cv::Range((i - win_lim),(i + win_lim)+1), cv::Range((j - win_lim),(j + win_lim)+1));
        cv::Mat candidates = Int_r(cv::Range((i - win_lim),(i + win_lim)+1), cv::Range((j - max_disp - win_lim),(j -1+
win_lim)+1));
int min_S = win_size * win_size *255;
int disp =0;
for(int k =1; k < max_disp; k++){
            cv::Mat C = candidates(cv::Range(0, win_size), cv::Range(k -1, k + win_size -1));
            cv::Mat BC;
            cv::absdiff(B, C, BC);
int S = cv::sum(BC)[0];
if(S < min_S){
                min_S = S;
                disp = max_disp +1- k;
}
}
return disp;
}
    cv::Mat fastCMeans(cv::Mat im,int c){
//Reshape the matrix
    cv::Mat samples(im.rows*im.cols,1, CV_32F);
for(int y=0; y<im.rows; y++){
constint8_t* im_ptr =(constint8_t*)(im.data+y*im.step);
for(int x=0; x<im.cols; x++)
            samples.at<float>(y+x*im.rows)=*(im_ptr+x);
}
    cv::Mat labels;
int attempts =5;
    cv::Mat centers;
    cv::kmeans(samples, c, labels, cv::TermCriteria(/*CV_TERMCRIT_ITER|*/CV_TERMCRIT_EPS,10000,0.001), attempts,
cv::KMEANS_PP_CENTERS, centers);
    cv::Mat new_image (im.size(), im.type());
for(int y=0; y<im.rows; y++)
for(int x=0; x<im.cols; x++)
{
int cluster_idx = labels.at<int8_t>(y+x*im.rows,0);
            new_image.at<int8_t>(y,x)=centers.at<float>(cluster_idx);
}
return new_image;
}
```

SONAR FILE:

```
/*
 * File:   Sonar.cpp
 * Author: erico ; thiago
 *
 * Created on April 19, 2017, 8:22 PM
 */

#include <sys/types.h>
#include "Sonar.hpp"
#include "config/SonarConfig.hpp"

namespace sv {
constchar* al_err_str(ALenum err){
switch(err){
                CASE_RETURN(AL_NO_ERROR);
                CASE_RETURN(AL_INVALID_NAME);
CASE_RETURN(AL_INVALID_ENUM);
                CASE_RETURN(AL_INVALID_VALUE);
                CASE_RETURN(AL_INVALID_OPERATION);
                CASE_RETURN(AL_OUT_OF_MEMORY);
}
return"unknown";
}
#undef CASE_RETURN

#define __al_check_error(file,line) \
do { \
        ALenum err = alGetError(); \
        for(; err!=AL_NO_ERROR; err=alGetError()) { \
std::cerr << "AL Error " << al_err_str(err) << " at " << file << ":" << line << std::endl; \
        } \
    }while(0)

#define al_check_error() \
    __al_check_error(__FILE__, __LINE__)

void init_al(){
        ALCdevice *dev =NULL;
        ALCcontext *ctx =NULL;

constchar*defname = alcGetString(NULL, ALC_DEFAULT_DEVICE_SPECIFIER);
        std::cout <<"Default device: "<< defname << std::endl;

        dev = alcOpenDevice(defname);
        ctx = alcCreateContext(dev,NULL);
        alcMakeContextCurrent(ctx);
}
void exit_al(){
        ALCdevice *dev =NULL;
        ALCcontext *ctx =NULL;
        ctx = alcGetCurrentContext();
        dev = alcGetContextsDevice(ctx);

        alcMakeContextCurrent(NULL);
        alcDestroyContext(ctx);
        alcCloseDevice(dev);

}
bool isPlaying(ALuint src){
```

```
        ALint value;
bool res =false;
        alGetSourcei(src, AL_SOURCE_STATE,&value);
if(value == AL_PLAYING)
            res =true;
return res;
}
float*freqs;
float*fase;
int*amplscale;
int rows;
int cols;
bool initialized=false;

void initVar(cv::Mat img,int max_disp){
        rows= img.rows;
        cols= img.cols;
        freqs=newfloat[rows];
        fase =newfloat[rows];
        amplscale =newint[max_disp+1];
float fator =1.069694631;


//Done ones
        freqs[0]= FREQ_MIN;
fase[0]= FASE;

for(int i =0; i < rows; i++){
freqs[i]=freqs[i]= FREQ_MIN*std::pow((FREQ_MAX/FREQ_MIN),(((float)i)/(rows -1)));
            fase[i]=(i)*float(M_PI)/(rows);//fase= Pi/rows
}
if(AMP_LINEAR==ON)
{
for(int i =0; i < max_disp+1; i++){
                amplscale[i]=255/max_disp*i;
}
}
else
{
for(int i =0; i < max_disp+1; i++){
                amplscale[i]= std::round((255/(std::pow(10,(AMP_FACTOR*LOG_255 /(max_disp+1-
1))*(max_disp+1))))*std::pow(10,(AMP_FACTOR*LOG_255 /(max_disp+1-1))* i));
}
}
        amplscale[0]=0;
        initialized=true;
}

int sonar(cv::Mat img,float seconds,int max_disp){
for(int wj =0;wj < img.rows;wj++)
{
for(int wi =0;wi < img.cols;wi++)
{
if(img.at<int8_t>(wj,wi)<0)
                    img.at<int8_t>(wj,wi)=0;
img.at<int8_t>(wj,wi)=(int)(AMP_FACTOR_2*img.at<int8_t>(wj,wi))*(1.0-std::pow(E,(float)((float)(-
1.0)*(img.at<int8_t>(wj,wi))/((float)MAX_DISP))));
if(img.at<int8_t>(wj,wi)>max_disp)
                     img.at<int8_t>(wj,wi)=max_disp;
if(img.at<int8_t>(wj,wi)<CORTE)
                    img.at<int8_t>(wj,wi)=0;
}
}
         cv::Mat imgshow = img;
```

```
         cv::resize(imgshow, imgshow, cv::Size(320,240));
         cv::imshow("nova", imgshow*(255/max_disp));
         cv::waitKey(10);


/* initialize OpenAL */
        init_al();

/* Create buffer to store samples */
        ALuint buf;
        alGenBuffers(1,&buf);
        al_check_error();

/* Fill buffer with Sine-Wave */

unsigned sample_rate =44096;
size_t buf_size = seconds* sample_rate;
if(initialized==false)
{
            initVar(img,max_disp);
}

int ampls[rows];
short*samples;
        samples =newshort[buf_size];
short tsample =0;
int cCol =0;
for(int i =0; i < buf_size;++i){
if(cCol %((int)(buf_size / cols))==0){
for(int k =0; k < rows; k++){
int pos = img.at<int8_t>(k, cCol /((int)(buf_size / cols)))-1;
if(pos<0)
                        pos=0;
if(pos>=max_disp)
pos=max_disp;
                    ampls[k]= amplscale[pos];
if(ampls[k]>255||ampls[k]<0)
ampls[k]=0;
}
}
for(int j =0; j < rows; j++){
long  tnextL =(long)(32600.0/(255.0*rows)*(float)(ampls[j]* std::sin((2.0*float(M_PI)* freqs[j])*(float)i/((float)
sample_rate)+ fase[j])));
short tnext =(short)tnextL;
                tsample += tnext;
}
if(tsample >32600||tsample <-32600)
{
                std::cout<<tsample<<std::endl;
                exit_al();
return1;
}
            samples[i]= tsample;
            tsample =0;
}
/* Download buffer to OpenAL */
        alBufferData(buf, AL_FORMAT_MONO16, samples, buf_size *2, sample_rate);
        al_check_error();

/* Set-up sound source and play buffer */
        ALuint src =0;
        alGenSources(1,&src);
        alSourcei(src, AL_BUFFER, buf);
```

```
/*to change the volume*/
float newVolume =1.0f;
        alSourcef(src, AL_GAIN, newVolume);
        alSourcePlay(src);

/* While sound is playing, sleep */
        al_check_error();

while( isPlaying(src)){};
/* Dealloc OpenAL */
        exit_al();
//   al_check_error();

return0;
}
}
```

RECTFY FILE:

```
/*

 * File:   Rectify.cpp
 * Author: erico ; thiago
 *
* Created on March 9, 2017, 8:48 AM
 */

#include "Rectify.hpp"
#include <iostream>

namespace sv
{
//calculate rectify maps for rectification using calib-parameters
void stereoRectifyMaps(cv::Mat cameraMatrix1,
            cv::Mat distCoeffs1,
            cv::Mat cameraMatrix2,
            cv::Mat distCoeffs2,
            cv::Mat R,
            cv::Mat T,
            cv::Size imageSize,
            cv::Mat &mapL1,
            cv::Mat &mapL2,
            cv::Mat &mapR1,
cv::Mat &mapR2){
        cv::Mat Q ,R1 ,R2 ,P1, P2;

        cv::stereoRectify(cameraMatrix1,distCoeffs1, cameraMatrix2, distCoeffs2, imageSize,  R, T,  R1,  R2, P1,  P2, Q,
cv::CALIB_ZERO_DISPARITY,-1, imageSize,0,0);

        cv::Mat cameraMatrix[]={cameraMatrix1,cameraMatrix2};
cv::Mat distCoeffs[]={distCoeffs1,distCoeffs2};
        cv::Mat Rs[]={R1,R2};
        cv::Mat Ps[]={P1,P2};


        sv::initUndistortRectifyMap(cameraMatrix,distCoeffs,Rs,Ps,imageSize,mapL1,mapL2,mapR1,mapR2);
}
//Precompute maps for cv::remap()
void initUndistortRectifyMap(cv::Mat cameraMatrix[],cv::Mat distCoeffs[], cv::Mat R[], cv::Mat newCameraMatrix[], cv::Size
size,  cv::Mat &mapL1, cv::Mat &mapL2, cv::Mat &mapR1, cv::Mat &mapR2){

        cv::initUndistortRectifyMap(cameraMatrix[0],distCoeffs[0],R[0],newCameraMatrix[0],size, CV_16SC2, mapL1, mapL2);
        cv::initUndistortRectifyMap(cameraMatrix[1],distCoeffs[1],R[1],newCameraMatrix[1],size, CV_16SC2, mapR1, mapR2);
```

```
}
//rectify image
    cv::Mat StereoRectifyImg(cv::Mat img, cv::Mat map[]){
        cv::Mat rimg;
        remap(img, rimg, map[0], map[1], CV_INTER_LINEAR);
return rimg;
}
}
```

CALIBPARAMETERS FILE:

```
/*

 * File:   CalibParameters.cpp
 * Author: erico ;thiago
 *
 * Created on March 9, 2017, 10:39 AM
 */

#include "CalibParameters.hpp"

CalibParameters::CalibParameters(){}
CalibParameters::~CalibParameters(){}

//read from file calib parameter for rectification
void CalibParameters::readCalibDataFile(std::string pathfile){
        cv::FileStorage fs2(pathfile, cv::FileStorage::READ);
fs2["camera_matrix_l"]>> camera_matrix_l;
fs2["camera_matrix_l_old"]>> camera_matrix_l_old;
        fs2["distortion_coefficients_l"]>> distortion_coefficients_l;
fs2["camera_matrix_r"]>> camera_matrix_r;
        fs2["camera_matrix_r_old"]>> camera_matrix_r_old;
fs2["distortion_coefficients_r"]>> distortion_coefficients_r;
        fs2["fc_left"]>> fc_left;
fs2["fc_right"]>> fc_right;
fs2["cc_left"]>> cc_left;
        fs2["cc_right"]>> cc_right;
        fs2["kc_left"]>> kc_left;
        fs2["kc_right"]>> kc_right;
fs2["T"]>> T;
        fs2["R"]>> R;
        fs2["om"]>>om;
        fs2.release();

}
```