UNIVERSIDADE FEDERAL DE SANTA CATARINA

SISTEMA DE NAVEGAÇÃO ROBÓTICA POR VISÃO COMPUTACIONAL

Sergio Genilson Pfleger

Florianópolis

2013/1

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Sistema de navegação robótica por visão computacional

Sergio Genilson Pfleger

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para a obtenção do grau de bacharel em Ciências da Computação

Florianópolis – SC

2013/1

Sergio Genilson Pfleger

Sistema de navegação robótica por visão computacional

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Orientadora: .....
Patricia Della Mea Plentz

Banca examinadora

.....
Edson Roberto de Pieri

.....
Elder Santos

DEDICATÓRIA

Dedico este trabalho a minha Família

AGRADECIMENTOS

Agradeço, primeiramente, Patricia Della Mea Plentz, minha orientadora.

Agradeço também aos meus amigos que me auxiliaram neste trabalho, direta ou indiretamente e pacientemente me ouviram falar do meu trabalho, especialmente a Ronan Rasia.

Ao pessoal que me emprestou material para a execução dos experimentos: Alexandre P. de Pinho, João R. Rover, Ramon D. Miranda.

Agradeço também a minha noiva, Maryah E. M. Haertel, que me aturou durante a execução deste trabalho e ao longo do curso.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE REDUÇÕES

OpenCV …… **Open** Source **C**omputer **V**ision

USB ..... **U**niversal **S**erial **B**us

LED …… **L**ight-**E**mitting **D**iode

SLAM ……..…… **S**imultaneous **L**ocalization **A**nd **M**apping

RESUMO

Robôs com navegação autônoma são utilizados em diversas situações, principalmente onde é difícil o controle ou a ação humana, como explorações espaciais ou operações de resgate, ou até mesmo tarefas domésticas como limpeza, onde a ação humana é repetitiva. Desenvolver métodos de navegação precisos e confiáveis em ambientes desestruturados e sujeitos a alterações são desafios enfrentados constantemente. Este trabalho propõe um sistema de navegação robótica baseado em visão computacional utilizando câmeras. O processamento de imagens será realizado através de softwares rodando em um computador embarcado num robô de propósitos gerais, juntamente com as câmeras. Em detrimento aos sinais interpretados pelo computador, este deverá enviar sinais a um módulo Arduino responsável pelo controle das rodas do protótipo robótico.

Palavras chave: Visão computacional, navegação robótica, Arduino.

1. INTRODUÇÃO

Robôs com navegação autônoma são utilizados em diversas situações, principalmente onde é difícil o controle ou a ação humana, como explorações espaciais ou operações de resgate, ou onde a ação humana é repetitiva e desgastante.

Ambientes de mundo real sofrem alterações constantes, diferentemente de ambientes controlados de laboratório, de modo que é necessário fazer uma leitura constante do espaço onde o robô está inserido e perceber caminhos possíveis para trajetória. O uso de sensores não visuais para tais tarefas permite somente encontrar um ponto de um obstáculo a cada leitura do meio, fazendo a navegação do robô se assimilar de um ser humano andando pelo escuro. Para se construir um mapa completo do ambiente, seriam necessárias diversas leituras do ambiente onde o robô se encontra imerso.

Fazendo o uso de câmeras é possível fazer uma leitura mais completa do ambiente, identificando um número muito maior de obstáculos, a cada leitura, de modo que seja possível encontrar caminhos mais rapidamente e localizando objetos a distancias consideráveis. É possível a construção de um mapa com uma única captura de imagens.

Neste trabalho será abordado um sistema de navegação baseado em visão por câmeras, onde as imagens adquiridas são interpretadas por um computador que será responsável por encontrar uma trajetória e enviar comandos para um módulo de controle dos motores, para que o robô possa seguir uma trajetória.

Será também abordado um protótipo robótico simples capaz de se locomover ao longo da trajetória desejada.

1.1 OBJETIVOS GERAIS

Este trabalho tem por objetivo desenvolver um sistema robótico com capacidade de navegação por ambientes desestruturados baseado em visão computacional.

1.2 OBJETIVOS ESPECIFICOS

Este trabalho tem por objetivos específicos:

- Fazer o reconhecimento espacial de um ambiente utilizando câmeras;
- Identificar objetos utilizando uma cor específica;
- Construir um protótipo robótico capaz de responder aos comandos recebidos de um computador;
- Comunicar o computador com o protótipo robótico;
- Aproximar-se do objeto identificado, evitando obstáculos;

2. VISÃO COMPUTACIONAL

Visão computacional é o nome empregado a sistemas mistos de câmeras e computadores, onde as imagens adquiridas pelas câmeras são interpretadas de forma a extrair informações do ambiente. A visão computacional é empregada em diversas áreas, como medicina, biologia, física, em segurança pública e privada e na indústria, e de diversas formas distintas.

Conforme Dudek e Jenkin [Dudek and Jenkin, 2010], visão computacional é surpreendentemente poderosa como meio de sensoriamento e tão difícil quanto de usar em contexto robótico.

Neste trabalho ela será empregada na análise do ambiente em torno do robô de modo a identificar obstáculos. Comumente algoritmos de navegação utilizam marcadores para se localizar num ambiente. Ao robô encontrar um marcador é fácil estimar sua posição. Como objetiva-se navegação por ambientes naturais, onde marcadores não existem, algoritmos que identificam marcadores específicos não são muito úteis. Desta forma, a visão computacional, que pode ser implementada de diversas formas, usando diversos algoritmos distintos, se restringe e os modelos mais comuns são os baseados em visão estereoscópica e a visão por fluxo ótico.

O modelo baseado em visão estereoscópica utiliza câmeras aos pares. Um par de câmeras, direcionadas para uma mesma cena, que capture imagens simultâneas pode ser comparado com a visão humana, que é constituído por um par de olhos apontando para uma mesma cena. O conceito básico para este modelo é procurar em uma das imagens um pixel (ou conjunto de pixels) correspondente a um pixel (ou conjunto de pixels) da outra imagem. Com as câmeras deslocadas uma em relação à outra, ao adquirir as imagens, objetos

mais próximos às câmeras estarão mais deslocados, um em relação ao outro, na representação dos objetos nas imagens. Objetos mais distantes às câmeras, por outro lado, assumem praticamente a mesma posição nas suas representações. Com base nisto, se pode calcular a distância entre os objetos e o robô, criando o mapa do ambiente onde ele se encontra.

O modelo baseado em visão por fluxo ótico é similar a visão estereoscópica, porém utiliza uma única câmera. Também utiliza pares de imagens, porém adquiridas após deslocamento temporal e espacial da câmera. Neste caso, uma imagem é adquirida e faz-se um pequeno deslocamento da câmera e se adquire a outra imagem. A partir do par de imagens faz-se a correspondência de pixels entre elas. Pixels referentes a objetos mais próximos à câmera sofrem uma variação maior na posição de suas representações na imagem. Pode-se então calcular a distância relativa entre objetos e a câmera.

Para este trabalho o modelo adotado será o de visão estereoscópica, por este conseguir mapear mesmo com o robô estando parado. Alguns conceitos de visão por fluxo ótico serão utilizados para detecção de deslocamento.

2.1 OpenCV

O OpenCV (***Open*** *Source* ***C****omputer* ***V****ision*) é uma biblioteca de visão computacional para aplicações em tempo real desenvolvida pela Intel [OpenCV Reference]. É escrito em linguagem C++ [OpenCV] e fornece diversas funções que facilitam o trabalho com visão computacional. Esta biblioteca é largamente utilizada em trabalhos correlatos e será também utilizada neste trabalho.

2.2 CAPTURA DE IMAGENS

Uma imagem fotográfica é a representação de uma cena do mundo real. Esta representação é produzida através de uma câmera, que recebe luz dos objetos que compõem a cena. Esta luz é focada sobre um sensor ótico contido na câmera através de lentes. O sensor ótico é composto por vários pontos (pixels), alinhados em forma de grade. Os sinais luminosos que incidem no sensor são convertidos em dados, conforme as intensidades de luz que incidem sobre cada pixel.

Fazendo uso da biblioteca OpenCV, a tarefa de capturar imagens se torna bastante simples. A função *CvCapture* cvCaptureFromCAM(int index)* inicializa uma estrutura *CvCapture* para a leitura de um stream de vídeo [OpenCV Online Documentation]. O parâmetro *int index* é o número da câmera. A função *IplImage* cvQueryFrame(CvCapture* capture)*, por sua vez, captura um frame do vídeo e o retorna como imagem *IplImage.*

2.3 CALIBRAÇÃO DE CÂMERAS

Entre diversos problemas relacionados com aquisição de imagens, podemos evidenciar as imperfeições do conjunto ótico que faz a aquisição das imagens, como distorção nas lentes e deformações no sensor ótico. As imperfeições fazem com que as imagens adquiridas pelas câmeras necessitem de correções, caso contrário estas podem interferir nos resultados.

Para a calibração é usado um padrão e realizada uma seqüência de capturas de imagens deste padrão em diversas posições e distâncias da câmera. Estas imagens são processadas pelo algoritmo de calibração, identificando as deformações. Entre os padrões mais conhecidos para esta finalidade está o padrão de tabuleiro de xadrez (*Chessboard*), que tem uma

geometria simples e seus cantos podem ser facilmente identificados utilizando visão computacional.

O tabuleiro de xadrez necessita ser plano. Deformações no tabuleiro levariam a interpretações erradas do algoritmo de calibração. Desta forma o padrão de calibração deve ser construído sobre uma superfície plana. Neste trabalho foi utilizado um tabuleiro de 9 x 7 impresso sobre papel comum e colado sobre uma superfície de vidro. Cada célula do tabuleiro é quadrada e possui 27,5 mm de lado.

A captura de diversas imagens é importante para explorar os defeitos das câmeras em todos os pontos da imagem, para minimizar erros de medida e minimizar erros ao se procurar pelos cantos das células do tabuleiro.

Figura 1 À Esquerda (a): Padrão de tabuleiro de xadrez com os cantos identificados pelo algoritmo. À Direita (b): a mesma imagem, porém corrigida utilizando os parâmetros de calibração aferidos com auxílio dos pontos identificados a esquerda.

O OpenCV fornece funções que identificam os cantos do tabuleiro de xadrez, conforme indica a ***Figura 1a***. Identificados os pontos dos cantos das células do tabuleiro faz-se a calibração. Esta retorna as matrizes de dados referentes às características do sistema. Com estas matrizes pode-se fazer o mapeamento da imagem original para a imagem corrigida, exibida na ***Figura***

1b. Observa-se o surgimento de pequenas curvaturas nas extremidades da figura, causadas, principalmente, pela distorção das lentes.

2.4 VISÃO ESTEREOSCÓPICA

Segundo Otuyama [Otuyama, 1998], Visão Estéreo (*Stereo Vision*) é o ramo da visão computacional que analisa o problema da reconstrução da informação tridimensional de objetos a partir de um par de imagens capturadas simultaneamente, mas com um pequeno deslocamento lateral.

É dito visão estereoscópica quando ocorre a extração de informações relativas à posição, velocidade e/ou tamanho de objetos, em um espaço tridimensional, a partir da análise imagens de duas ou mais câmeras.

Existem vários algoritmos destinados a visão computacional estereoscópica. Em sua maioria, estes percorrem as imagens das diferentes câmeras em busca de informação semelhante.

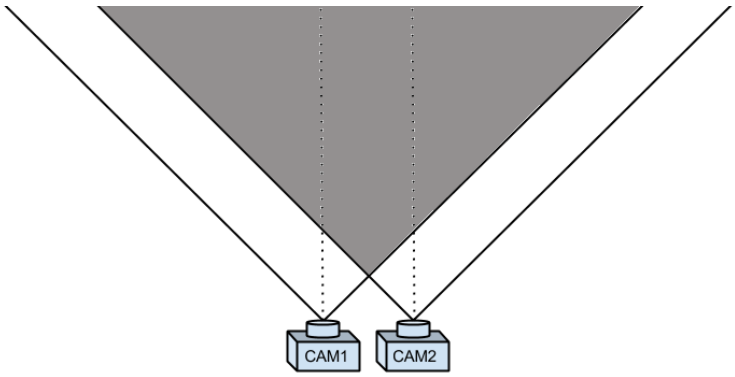

Figura 2 Par de câmeras posicionadas paralelamente, direcionadas para uma mesma cena. A área em cinza representa a região vista pelas duas câmeras.

Em geral, algoritmos que apresentam melhores resultados na reconstrução tridimensional são também os mais demorados. Nesta aplicação, além do resultado da reconstrução, é importante também que a velocidade de execução do algoritmo seja razoável, pois os resultados são requeridos em tempo real para a navegação. Sob estas considerações, foi escolhido o

algoritmo de *Block Matching* do OpenCV, que é capaz de fornecer bons resultados de reconstrução e tempo, considerando a resolução das câmeras adotadas.

Para a visão estereoscópica são necessárias duas câmeras direcionadas para a mesma cena capturando imagens simultâneas, conforme ilustra a ***Figura 2*****.** Sabe-se que a captura de imagens simultâneas por câmeras distintas é de difícil implementação, porém, podem ser consideradas simultâneas imagens onde o tempo entre uma captura e outra é pequeno.

Para que os resultados da visão estereoscópica sejam satisfatórios, faz-se necessária a calibração estéreo.

2.5 CALIBRAÇÃO ESTÉREO

Além das informações sobre cada câmera em especifico, fazem-se necessárias também informações sobre o par de câmeras, tais como alinhamento, rotação e resolução para que se possa fazer referência entre os pixels de uma imagem com os pixels equivalentes na outra imagem. A este processo refere-se como calibração estéreo (*Stereo Calibration*).

Figura 3 Par de imagens adquiridas pelas duas câmeras simultaneamente do tabuleiro de Xadrez para efetuar a calibração estéreo.

O processo é similar ao descrito anteriormente: adquiri-se uma seqüência de imagens, porém, nas duas câmeras, simultaneamente; encontram-se os pontos das células do tabuleiro; faz-se a calibração e são geradas as matrizes de calibração; e faz-se o mapeamento das imagens. As ***Figura 3***, ***Figura 4*** e ***Figura 5*** ilustram este processo.

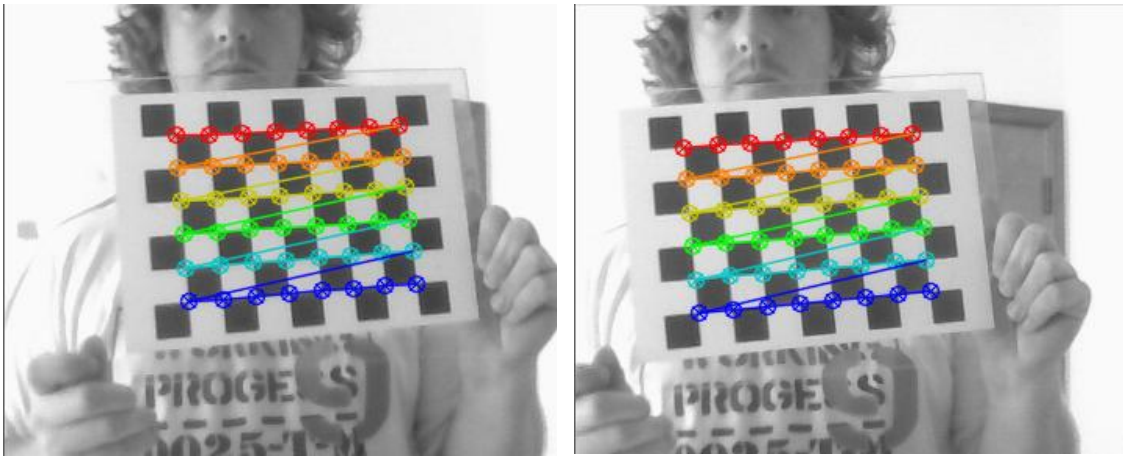

Figura 4 O mesmo par de imagens com os cantos identificados.

Na ***Figura 5*** nota-se as imagens das duas câmeras com pequenas correções em relação as imagens originais. Uma das correções perceptíveis é a correção do desnível das câmeras. Para isto, a imagem mais a direita está deslocada para cima por alguns pixels. Nota-se também que há correspondência entre os pixels da imagem a direita com os pixels da imagem a esquerda ao longo das linhas traçadas sobre as imagens. Esta correspondência faz-se necessária para que seja possível a reconstrução tridimensional do ambiente. O algoritmo de calibração estéreo utilizado neste trabalho é o proposto no livro Learning OpenCV [Learning OpenCV].

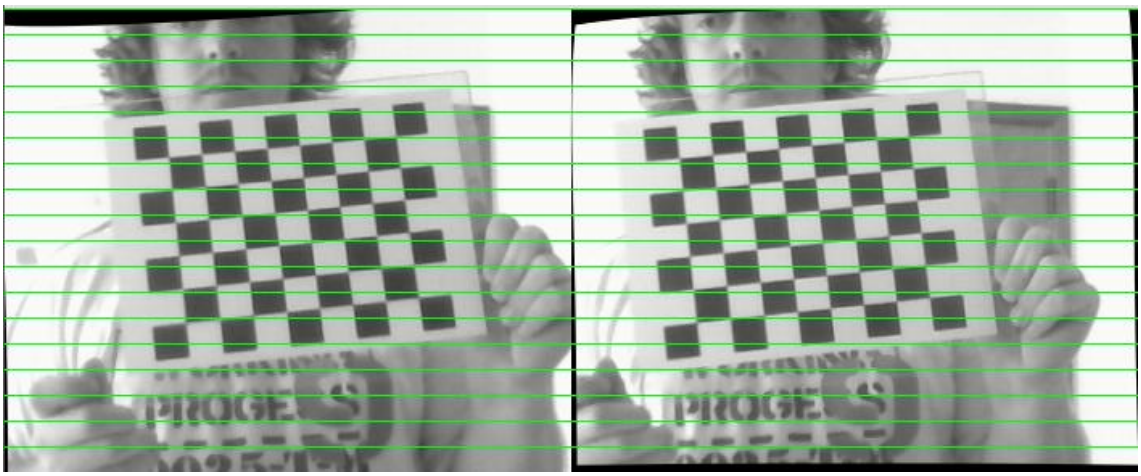

Figura 5 Imagens das duas câmeras após calibração estéreo das câmeras.

A calibração das câmeras faz-se necessária toda vez que as câmeras sofrerem alguma mudança de posição. Como o suporte das câmeras é frágil em qualquer pequeno movimento de uma das câmeras faz-se necessária outra calibração.

Após estas etapas de calibração pode-se dar início a etapa de visão estereoscópica, que neste trabalho utiliza o algoritmo de *block matching*, descrito a seguir

2.6 BLOCK MATCHING E MAPAS DE DISPARIDADE

O algoritmo de *Stereo Block Matching* (neste trabalho referido como *block matching*) utiliza pequenos blocos de pixels de uma das duas imagens e procura o bloco correspondente na outra imagem. Para que haja correspondências entre blocos das duas imagens é importante que a cena, da qual as imagens fazem a abstração, contenham vários detalhes. Como resultado do algoritmo se obtém o mapa de disparidade (*disparity map*).

A disparidade refere-se a quanto que um bloco está deslocado em relação ao seu correspondente na outra imagem. Esta informação está diretamente relacionada com a distância do objeto em relação ao par de

câmeras: quanto mais próximo das câmeras maior a disparidade. Quanto mais distante das câmeras, menor a disparidade.

Observa-se na ***Figura 6*** dois objetos (círculo e triângulo) posicionados entre as linhas pontilhadas que representam o centro da Câmera 1 e da Câmera 2. Quanto mais próximo das câmeras o objeto estiver, mais próximo das extremidades da imagem capturada estará a representação do objeto.

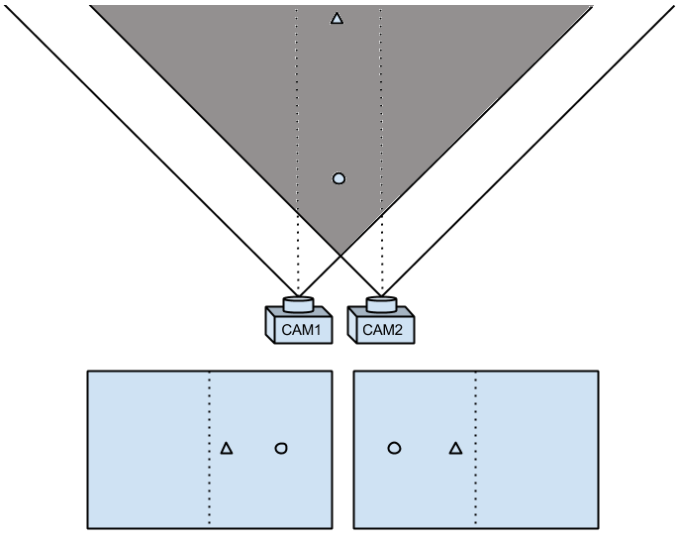

Figura 6 Dois objetos (círculo e triângulo) dentro da área de captura das duas câmeras. As linhas pontilhadas representam o centro da imagem.

A ***Figura 7*** representa a sobreposição das duas imagens obtidas na ***Figura 6***. As distâncias d_1 e d_2 representam a disparidade. O objeto mais próximo (o círculo) das câmeras apresenta uma disparidade (d_1) maior. O objeto mais distante (o triângulo) apresenta uma disparidade (d_2) menor. Desta forma, observamos que a disparidade está correlacionada com a distância do objeto as câmeras.

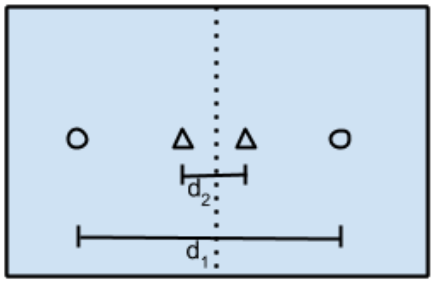

Figura 7 Sobreposição das imagens obtidas pelas duas câmeras dos objetos (círculo e triângulo). As distâncias ilustram as disparidades.

Com base na ***Figura 6*** e na ***Figura 7*** podemos deduzir algumas limitações para este sistema. A menor distância das câmeras onde será possível detectar a disparidade é no ponto mais próximo coberto pelas duas câmeras. A menor disparidade é zero, quando os pixels correspondentes assumem a mesma posição nas duas imagens.

A disparidade entre duas imagens é representada por um mapa de disparidade (*disparity map*). Mapas de disparidade, comumente, são em tons de cinza, onde tons de cinza mais claros referem-se à disparidade maior, logo, a objetos mais próximos as câmeras, e tons mais escuros referem-se a menor disparidade, logo, a objetos mais distantes da câmera.

Figura 8 As figuras mais à esquerda representam as imagens capturadas pelas câmeras, corrigidas, a imagem a direita é o mapa de disparidade obtido a partir das imagens à esquerda.

Observa-se um contorno em preto em torno de alguns objetos. Isto se deve aos trechos de imagem que estão presentes na imagem da esquerda,

porém estão oclusos pelo objeto na imagem da direita devido às diferentes posições das duas câmeras.

A ***Figura 9*** ilustra a distância dos objetos em relação às câmeras, num gráfico onde as linhas representam os contornos dos objetos vistos pela câmera, numa visão de cima para baixo (*top view*). Os traços mais à esquerda representam a mão da ***Figura 8***. Na extremidade direita os traços retos representam o fundo preto da imagem, onde o *block matching* não retornou resultado. Esta forma de visualização pode ser utilizada como um mapa dos objetos perante o robô e pode ser diretamente utilizada para navegar entre os objetos.

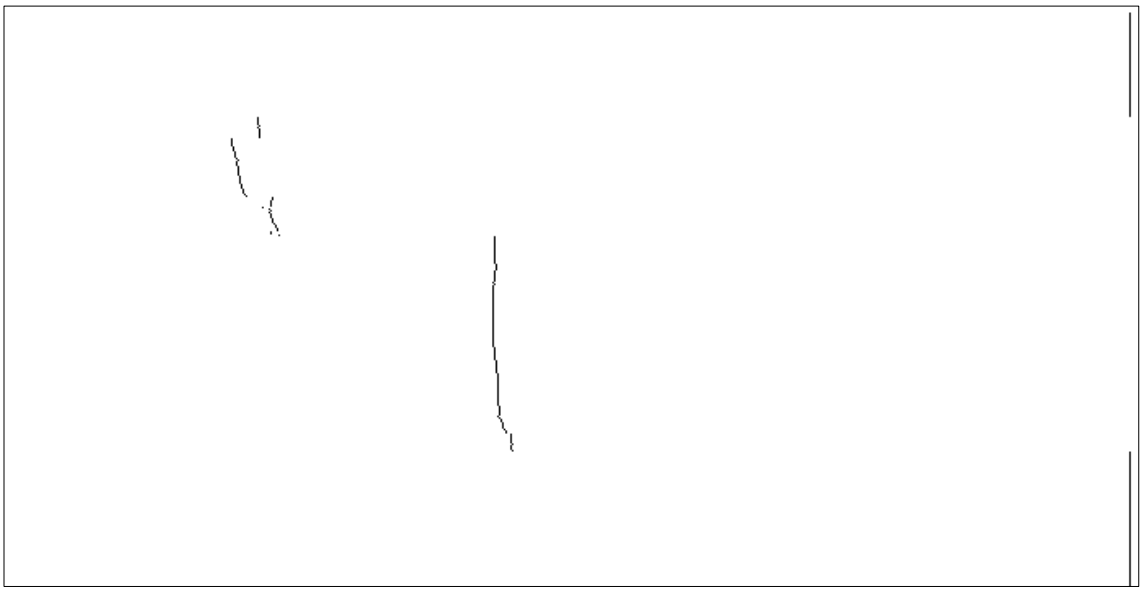

Figura 9 Mapa de disparidade visto do topo. Traços mais a direita representam contornos de objetos mais distantes. Traços mais a esquerda representa contorno de objetos mais próximos.

2.7 PARÂMETROS DO *BLOCK MATCHING*

Para fazer o *block matching* o OpenCV oferece a função void cvFindStereoCorrespondenceBM(const CvArr* left, const CvArr* right, CvArr*

disparity, CvStereoBMState* state). Os parâmetros *left* e *right* são as imagens de entrada na função, respectivamente a imagem da esquerda e da direita. O parâmetro *disparity* é o mapa de disparidade, como saída da função.

O parâmetro *state*, por sua vez, possui algumas peculiaridades: é um objeto da classe *CvStereoBMState* e nele estão as configurações referentes ao funcionamento do *block matching*. Para as principais variáveis do *CvStereoBMState* construiu-se uma pequena interface, conforme a ***Figura 10***.

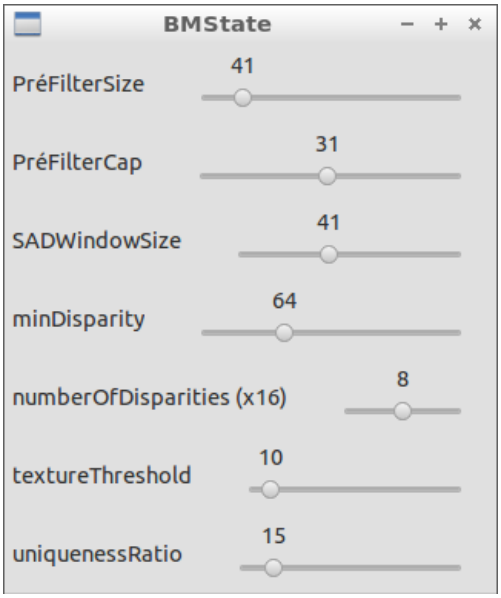

Figura 10 Janela de configuração do BMState.

O algoritmo de *block matching* implementado no *OpenCV* é uma algoritmo muito rápido, que utiliza apenas uma passada e utiliza a soma das diferenças absolutas entre os pixels da imagem da esquerda e da direita.

A mudança destas variáveis influencia diretamente sobre os resultados obtidos no *block matching*, nos seus mais variados aspectos, desde uma variação na profundidade mínima lida até a inutilização completa dos resultados. A ***Figura 11*** ilustra o resultado de uma má escolha para estas variáveis.

Figura 11 Resultado de um block matching usando variáveis mal escolhidas.

Ainda para ilustrar as influências da escolha das variáveis do *BMState* efetuou-se testes mudando a variável *numberOfDisparities* e verificou-se que a distância mínima varia, como mostra a ***Tabela 1*** a seguir. Esta variável necessita ser múltiplo de 16, desta forma, os valores para *numberOfDisparities* da tabela são multiplicados por 16.

Além da distância mínima, esta variável influencia sobre a qualidade do resultado e sobre a largura da imagem resultante. Para este trabalho foi escolhido o valor 8 (x16).

Tabela 1 Relação entre a variável numberOfDisparities e a distância mínima, medida entre um obstáculo e as câmeras.

numberOfDisparities (x16)	Distância mínima aproximada
4	160 cm
5	93 cm
6	72 cm
7	59 cm
8	47 cm
9	38 cm
10	32 cm
11	27 cm

2.8 RELAÇÃO ENTRE DISPARIDADE E PROFUNDIDADE

O mapa de disparidade nos informa a profundidade relativa dos objetos na imagem, de forma similar a visão humana [Mendes & Wolf]. A partir do

mapa de disparidade podemos construir um mapa de profundidade (*depth map*), que nos informa a profundidade real.

Conforme Mattocia [Mattocia, 2012], podemos adquirir um mapa de profundidade aplicando a ***Equação 1*** sobre o mapa de disparidade, onde *Z* é a profundidade, *B* é a distancia entre as câmeras, *f* é a distancia focal e *d* a disparidade, com duas câmeras paralelas.

$$Z = \frac{B.f}{d} \qquad \textit{Equação 1}$$

Para verificar a ***Equação 1*** foram realizadas alguns testes, posicionando-se um objeto em diversas profundidades, fazendo as medidas das distâncias entre as câmeras com uma régua e correlacionando com as medidas de disparidade obtidas, conforme a ***Tabela 2***.

Tabela 2 Comparação entre a distância de um objeto e a disparidade

Distância (cm)	Disparidade
50	1820
60	1622
70	1464
80	1350
90	1260
100	1190
110	1128
120	1080
130	1040
140	1012
150	971
160	947
170	922
180	903
190	885
200	872

Para ilustrar a correspondência entre a distância e a disparidade os dados da ***Tabela 2*** foi confeccionado o gráfico da ***Figura 12***.

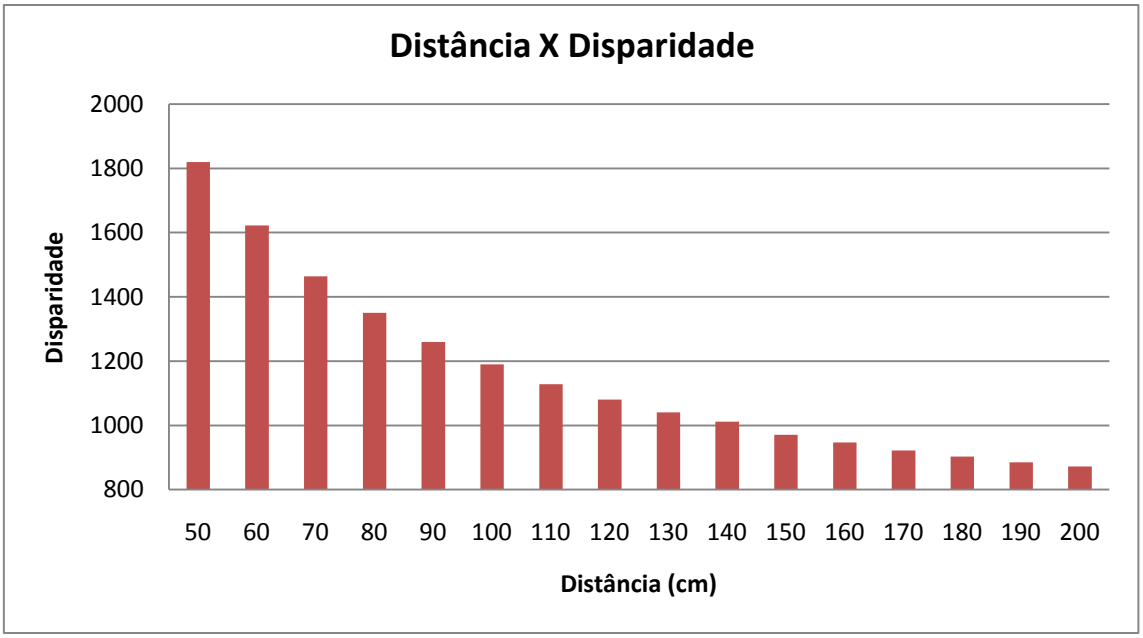

Figura 12 Disparidade em função da distância entre as câmeras e um objeto.

A distância entre as câmeras *B* pode ser medida com uma régua simples (7,35cm). A distância focal *f* pode ser obtida a partir das matrizes de calibração oriundas do algoritmo de calibração, neste caso 558,65px. Com um cálculo simples, utilizando a ***Equação 1***, podemos determinar a profundidade de um ponto. Utilizando as disparidades da ***Tabela 2***, por exemplo, 1820, a profundidade calculada seria 22,56 cm. Comparando com o valor medido da profundidade para esta disparidade observamos que os valores não são condizentes.

Supondo que o valor da distância focal *f* constante na matriz de calibração não seja o valor real da distância focal e tendo em mãos a ***Tabela 2*** com alguns valores reais de disparidade e profundidade, podemos propor a ***Equação 2***, baseando-se na ***Equação 1***, onde *Z* é a profundidade, *a* e *b* são constantes que queremos calcular e *d* a disparidade. Neste caso, *b* representa a multiplicação entre a distância focal *f* e a distância entre as câmeras *B* da ***Equação 1*** e *a* é uma constante de ajuste.

$$Z = a + \frac{b}{d} \qquad \textit{Equação 2}$$

Fazendo substituições simples com os valores da ***Tabela 2*** pode-se calcular os valores para *a* e *b*. Usando os pares Z_1 = 60, d_1 = 1622 e Z_2 = 170, d_2 = 922 chegamos a ***Equação 3***.

$$Z = -84{,}88 + \frac{235004{,}63}{d} \qquad \textit{Equação 3}$$

A ***Figura 13*** mostra uma comparação entre os valores calculados para a profundidade *Z,* utilizando a ***Equação 3*** sobre a disparidade ***Tabela 2***, e os valores da distância medida. Observa-se que a distância calculada segue uma curva, enquanto que os valores medidos seguem uma reta. Desta forma, pode-se afirmar que a profundidade *Z* não é inversamente proporcional a disparidade d, conforme sugerem as ***Equações 1***, ***2*** e ***3***. Quando Mattocia [Mattocia, 2012] enuncia a ***Equação 1*** assume que as câmeras estão posicionadas paralelamente, porém, o alinhamento das câmeras deste trabalho é feito de forma manual. Possivelmente, a imprecisão do alinhamento das câmeras faz com que a ***Equação 1*** não seja aplicável a este caso.

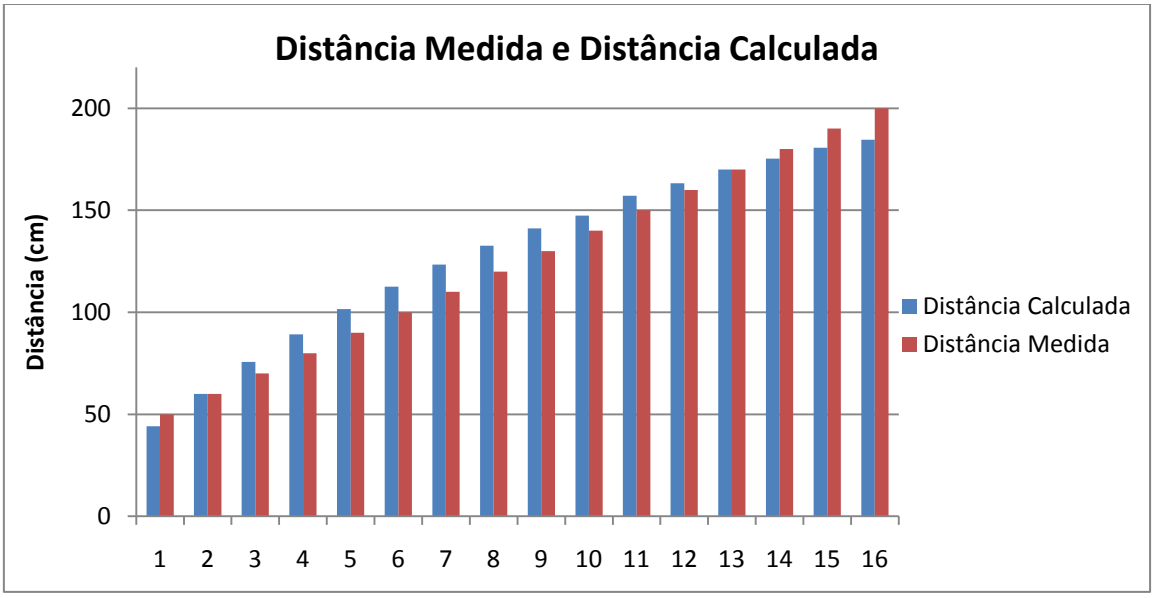

Figura 13 Comparação entre a distância medida e distância calculada

2.9 DIAGRAMAS DE FLUXO DOS ALGORITMOS

Os tópicos a seguir apresentam a estrutura dos algoritmos utilizados neste trabalho:

Aquisição Estéreo:

O algoritmo de aquisição estéreo adquire pares de imagens de duas câmeras e as armazena em uma pasta para serem processadas pelo algoritmo de calibração de câmeras. A ***Figura 14*** exibe o diagrama de fluxo do algoritmo.

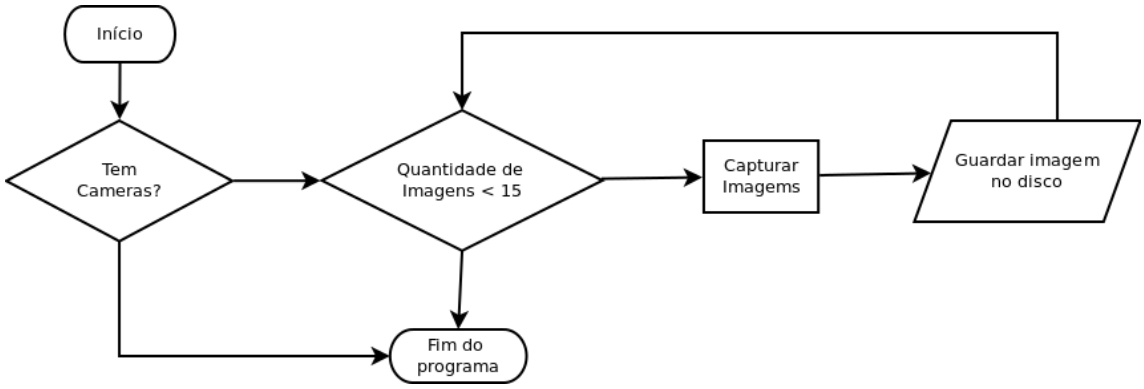

Figura 14 Diagrama do algoritmo de aquisição estéreo.

Calibração Estéreo de Câmeras:

Este algoritmo pega as imagens do algoritmo de aquisição estéreo, procura pelos cantos do tabuleiro de xadrez (padrão de calibração) e gera as matrizes de calibração. Este algoritmo é o proposto no livro Learning OpenCV [Learning OpenCV] e seu diagrama de fluxo não é apresentado aqui.

Algoritmo Principal

Este algoritmo faz o sensoriamento do ambiente através das câmeras e envia os comandos ao protótipo. A ***Figura 15*** mostra o diagrama de fluxo do algoritmo.

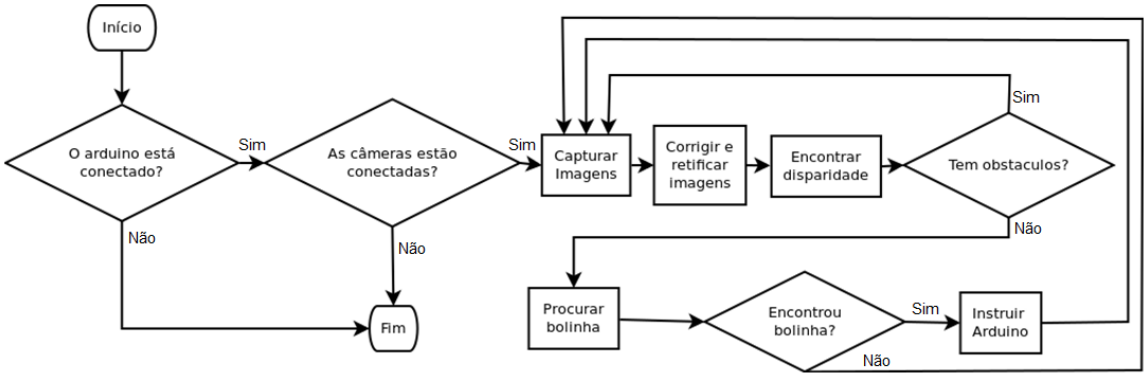

Figura 15 Diagrama do algoritmo principal

3. PROTÓTIPO ROBÓTICO

A teoria e os algoritmos sobre visão computacional foram aplicados em um protótipo robótico simples, constituído de um computador (notebook), duas câmeras, um par de motores, um par de rodas conectadas aos motores, uma roda do tipo louca, um módulo Arduino, bateria e uma base metálica. A ***Figura 16*** ilustra o protótipo desenvolvido e as seções a seguir detalham os componentes deste protótipo.

Figura 16 Protótipo desenvolvido

3.1 COMPUTADOR

O computador, neste trabalho, é responsável pelo processamento das imagens captadas pelas câmeras, executar o processamento sobre elas e fornecer ao Arduino as informações referentes ao movimento que o robô deve executar.

Considerando que o processamento de imagens é uma atividade que exige uma grande capacidade de processamento (de ordem mínima O(largura x altura)), é importante que o sistema operacional e outros processos ocupem pouco o processador. A linguagem escolhida para a aplicação deve ser igualmente leve e robusta, além de ser compatível com o OpenCV.

O sistema operacional Lubuntu foi escolhido para este trabalho. O Lubuntu é baseado em Linux e derivado do Ubuntu, porém com uma interface gráfica muito mais leve. A linguagem de programação escolhida para o desenvolvimento da aplicação foi o C++, que é a linguagem na qual o OpenCV é escrito. A ***Tabela 3*** mostra os softwares utilizados e suas respectivas versões.

Tabela 3 Softwares utilizados e suas respectivas versões.

Software	Versão
Lubuntu	Lubuntu 12.04.2 LTS
OpenCV	2.3.1
C++	gcc (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3

O computador utilizado neste trabalho é um notebook de configurações modestas: Processador AMD Turion II X2 M500 com 4 GB de Memória.

3.2 CÂMERAS

As câmeras utilizadas neste trabalho são webcams comuns, de resolução máxima 640 x 480 e conectadas ao computador através de cabos USB. As demais características da câmera estão listada no ***Anexo A***.

3.3 ARDUINO

O Arduino é uma plataforma de prototipagem eletrônica de código aberto baseado em hardware e software flexível e de fácil uso [Arduino]. Esta

plataforma possui diversos módulos que podem ser acoplados, como sensores, módulos de Wireless e Ethernet.

Esta plataforma se encontra disponível no mercado em diversos formatos, tamanhos e capacidades. Estão disponíveis também diversos componentes prontos para uso juntamente com o Arduino. Para este trabalho foi utilizado um Arduino UNO.

Conforme o fabricante, o Arduino fornece 40 mA de corrente a 5V em cada pino de entrada/saída, quando estiverem ativados. Desta forma, para suprir a tensão especificada para os motores (12V) é necessário um pequeno circuito, conforme o desenho esquemático da ***Figura 17***.

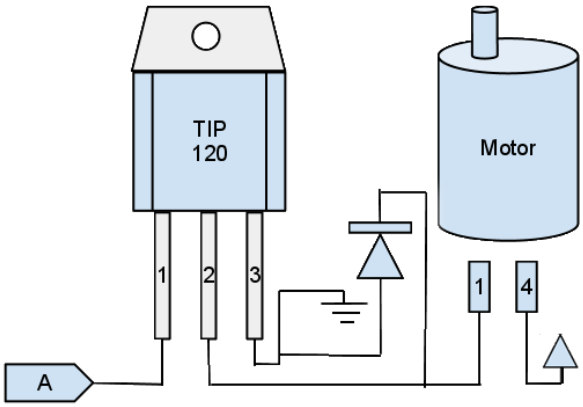

Figura 17 Desenho esquemático da ligação do Arduino ao motor.

O transistor TIP120 utilizado é do tipo NPN e possibilita a condução de até 5A a 60V. Outro componente externo é um diodo comum. Este evita que o motor devolva uma corrente de retorno ao circuito, causado pela inércia do motor. O terminal A, conectado ao pino 1 do TIP120, é onde se conecta uma das saídas do Arduino. O terminal 4 do motor é conectado a uma bateria. O circuito se limita a ativar o motor somente em um sentido.

A ***Figura 18*** mostra o circuito construído. Mais a esquerda estão três terminais, dois em azul, que são as saídas para as rodas, e um em cinza o conector da bateria. Próximo aos conectores azuis, os transistores, um para cada roda. Sobre o resto da placa se encontram dois soquetes brancos, disponíveis para um Arduino Uno, e um soquete preto, disponível para um Arduino Nano.

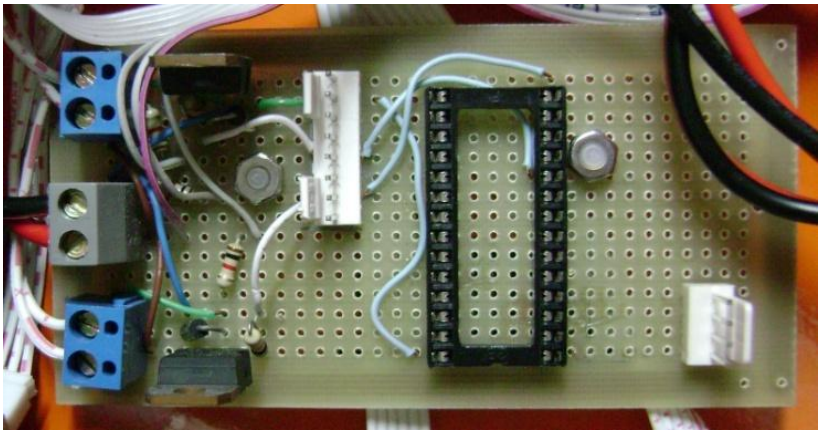

Figura 18 Placa produzida

O Arduino é conectado ao computador através de um cabo USB, pelo qual recebe os comandos. Este cabo também fornece a alimentação para o Arduino.

3.4 ALGORITMO DE CONTROLE DAS RODAS

O movimento das rodas do robô tratado neste estudo é controlado pelo Arduino, segundo os comandos recebidos do computador. O movimento das rodas do robô influencia diretamente sobre o movimento do robô. Desta forma, alguns requisitos são desejados:

- Arrancadas suaves;
- Curvas suaves;
- Paradas suaves, exceto em emergências;

- Tolerância a erros do computador;
- Velocidade de acordo com as capacidades de processamento de imagens;
- Responsivo;

Atentando para estes requisitos, foi proposto um algoritmo onde a quantidade de comandos possíveis, a partir do computador, seja reduzida. Desta maneira, o computador envia ao Arduíno os comandos:

- Direita (d) quando o computador desejar curvar para a direita;
- Esquerda (e) quando o computador desejar curvar a esquerda;
- Seguir (s) quando o computador optar por seguir reto;
- Parar (p) quando houver a necessidade de uma parada abrupta.

É sabido que o Arduino possui capacidades limitadas. Para minimizar a quantidade de dados na comunicação entre o computador e o Arduino, será sempre enviado somente um caractere (d, e, s ou p) para cada comando.

A partir destes comandos recebidos pelo computador, o Arduino altera a velocidade de cada roda de acordo com cada comando, de forma incremental. Comandos repetidos, como “seguir” aumentam a velocidade do robô linearmente. Para isto, foi definida uma variável vDireita e vEsquerda para as rodas da direita e da esquerda, respectivamente.

Estas variáveis podem assumir valores inteiros entre 0 e 10, sendo 0 a velocidade mínima, quando a roda estiver parada e 10 a velocidade máxima quando a roda estiver em velocidade máxima. O comportamento do *sofware* em relação aos comandos recebidos se torna o seguinte:

- Comando Seguir (s): Este comando incrementa a velocidade das duas rodas até o máximo de 10.

- Comando Esquerda (e): Decrementa a velocidade da roda da esquerda e incrementa a velocidade da roda direita
- Comando Direita (d): Incrementa a velocidade da roda da esquerda e decrementa a velocidade da roda esquerda
- Comando Parar (p): Seta a velocidade das duas rodas como zero.
- Quando não recebe comandos: Decrementa a velocidade de ambas as rodas enquanto a velocidade for maior que zero.

Os comandos são recebidos pelo Arduino a cada intervalo *t* de tempo. É ideal que *t* seja de valor próximo ao tempo entre frames obtidos das câmeras.

Após o processo de leitura do *buffer* de entrada e configurar as velocidades, uma segunda parte do algoritmo se responsabiliza por ligar e desligar os terminais do Arduino correspondentes as rodas de acordo com a velocidade. O tempo *t'* que cada roda deve ficar ligada por intervalo de tempo *t* é representado pela ***Equação 4***:

$$t' = \frac{velocidadeDaRoda}{velocidadeMaxima} * t \qquad \textit{Equação 4}$$

Desta forma, quando *t* for suficientemente pequeno, o movimento do robô parecerá homogêneo. A ***Figura 19*** ilustra a aceleração em função do tempo, saindo do repouso até a velocidade máxima, quando se recebe uma seqüência de comandos “s” maior que dez.

Com este algoritmo é esperado se obter arrancadas, curvas e paradas menos abruptas, reduzindo o risco de derrapagens. Caso ocorra algum problema no computador ou na comunicação entre o computador e o Arduino, o robô para gradualmente, já que a velocidade é decrementada até zero quando não se recebe dados. Da mesma forma, se o computador reduzir a taxa de comandos (ficar lento), por algum motivo, a velocidade do robô também

será reduzida, de acordo com a taxa de comandos. Além disto, se o computador, após a interpretação de um frame, enviar um comando errado, ocorrerá apenas um pequeno desvio da rota desejada originalmente.

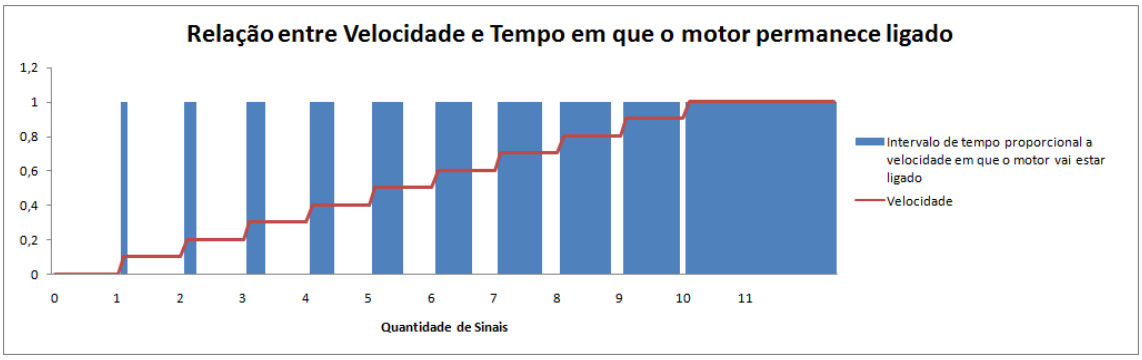

Figura 19 Relação entre velocidade e tempo em que o motor permanece ligado.

3.5 RODAS, MOTORES, BATERIA E CHASSI

As rodas, motores, bateria e chassi são de um robô Stinger [Stinger Robot]

3.6 PROTÓTIPO COMPLETO

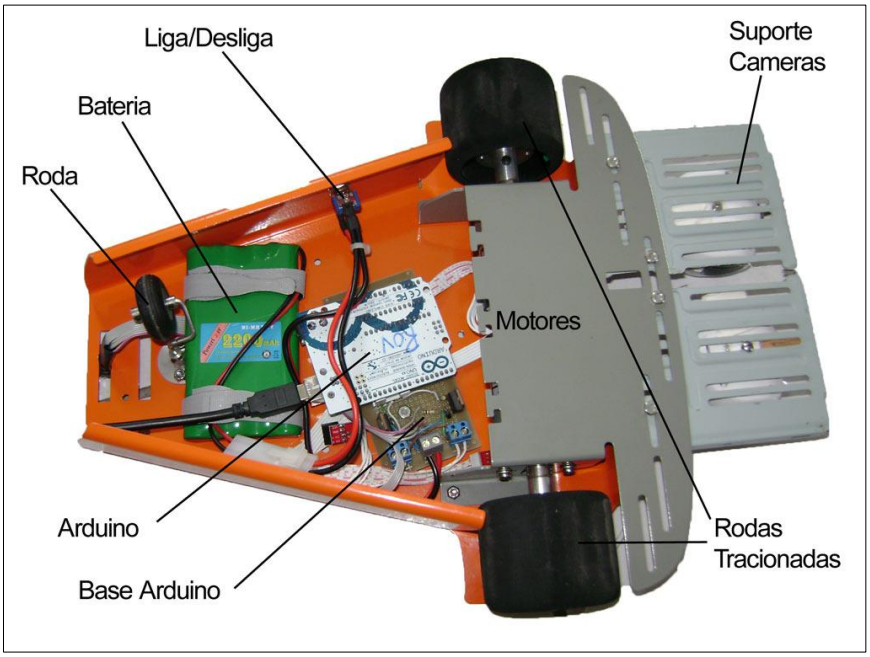

Figura 20 Vista inferior do protótipo e seus componentes

A ***Figura 20*** mostra o protótipo completo com seus componentes, visto por baixo. Estão destacadas as rodas, a bateria, o circuito de base construído para o Arduino, o Arduino, o chassi, um interruptor para ligar e desligar as baterias e o suporte das câmeras. Os motores estão oclusos por uma base cinza.

A ***Figura 21*** mostra a base do protótipo visto de cima, com seus componentes. Estão visíveis o Chassi, as Câmeras e LED's indicadores (o central indica se a bateria está ligada ou não e os laterais se as rodas foram ativadas).

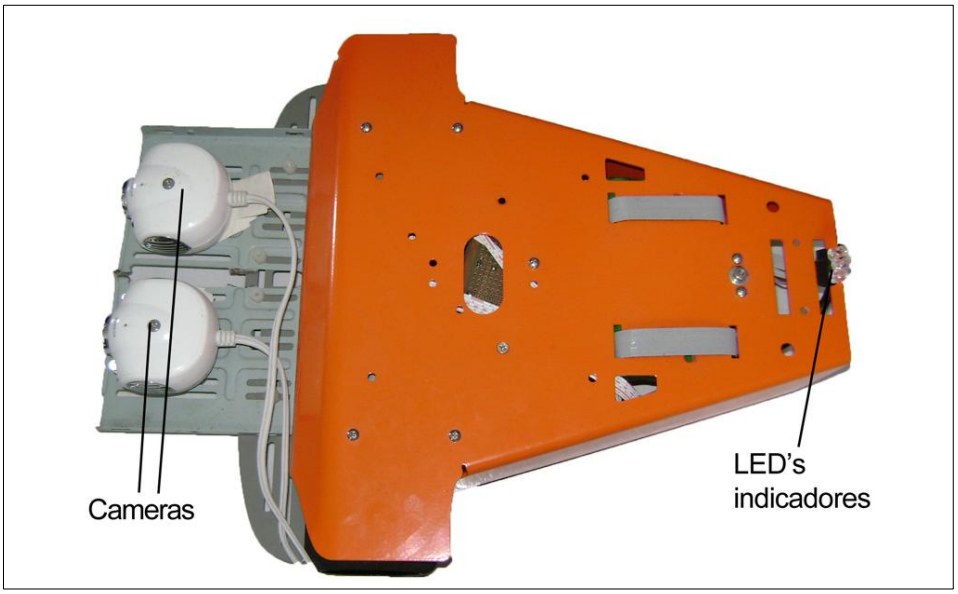

Figura 21 Vista superior da base do protótipo

4. TESTES REALIZADOS

Dois tipos de teste foram realizados: um para verificar os potenciais da visão computacional empregada neste trabalho e outro para verificar o resultado do trabalho como um todo, integrando a visão e o protótipo robótico.

4.1 TESTES DA VISÃO COMPUTACIONAL

Um dos atributos a se verificar da visão computacional é o alcance do conjunto e a capacidade de obter disparidade de objetos, mesmo em distância elevada. Para isto, foram realizados alguns testes em local onde se pode observar objetos a distâncias superiores a 10 metros.

A ***Figura 22*** apresenta o mapa de disparidade de imagens capturadas junto a alguns carros. Observa-se que o solo, que apresenta textura uniforme, não é identificado. Ao fundo os carros aparecem no mapa de disparidade. O carro mais claro, ao centro da imagem, está a uma distância de 11,15 m.

Figura 22 Mapa de disparidade de objetos distantes

A fim de verificar o potencial de retornar disparidade diferente com objetos quase tão distante quanto, a ***Figura 23*** apresenta o mesmo cenário, porém com o acréscimo de uma pessoa entre os carros e a câmera (9,20m e 6,75m respectivamente). Como o olho humano não é bom em discernir tons de cinza muito próximos, a ***Figura 24***, que é a visão de topo (*top view*) dos mapas de

disparidade das ***Figura 23*** sobrepostos, nos mostra que, nos dois casos, identificou algo mais próximo das câmeras do que os carros ao fundo.

Figura 23 Mapa de disparidade de objetos distantes. No topo, pessoa a 9,20 m. Em baixo, pessoa a 6,75m.

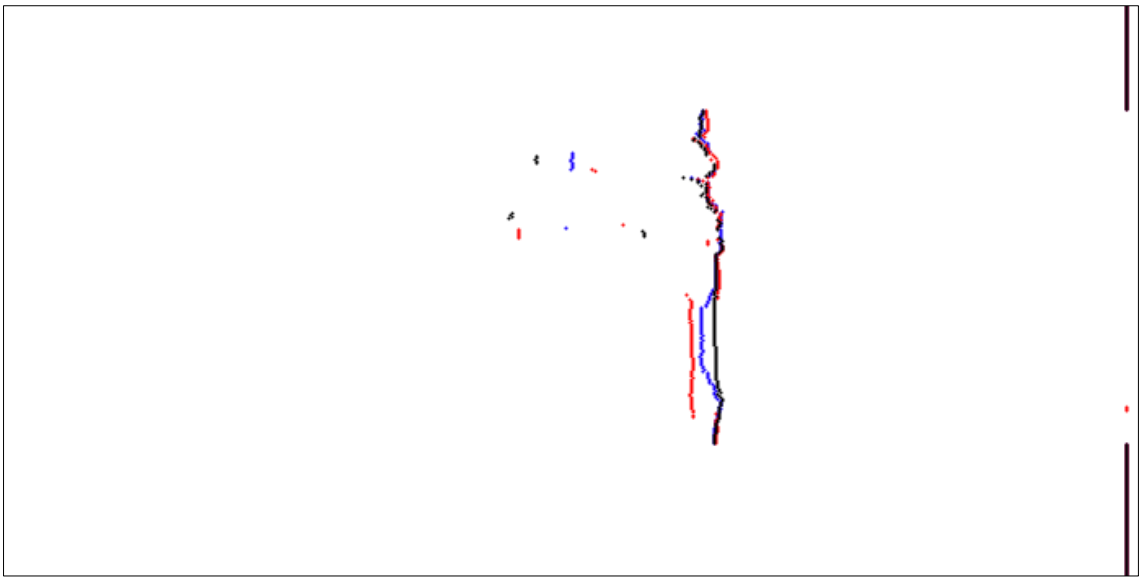

Figura 24 Top view da disparidade com objetos distantes. Linha preta é a disparidade sem a pessoa. Linha azul é a disparidade com a pessoa a 9,20m. Linha vermelha é a disparidade da pessoa a 6,75 m.

A análise de objetos próximos também é interessante. Tendo em vista que a disparidade aumenta a medida que a distância entre as câmeras e o

objeto se torna menor, quanto mais próximo estiver o objeto, mais informação quanto a posição do objeto consegue-se adquirir. Para ilustrar isto, a ***Figura 25*** nos mostra um livro a uma distância de 53,5 cm no seu ponto mais próximo e 61,6 cm no seu ponto mais distante. A ***Figura 26*** representa a visão de topo (*top view*) da ***Figura 25***. Observa-se nitidamente que o objeto (livro, neste caso) está de perfil para as câmeras.

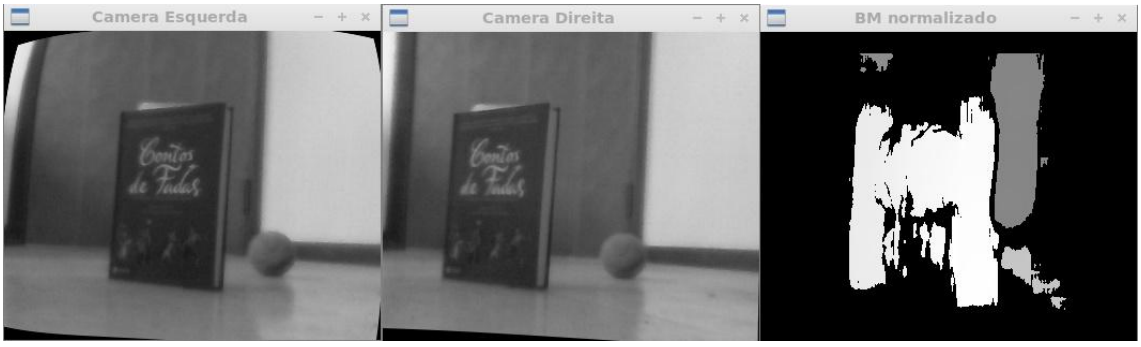

Figura 25 Mapa de disparidade de um objeto de perfil

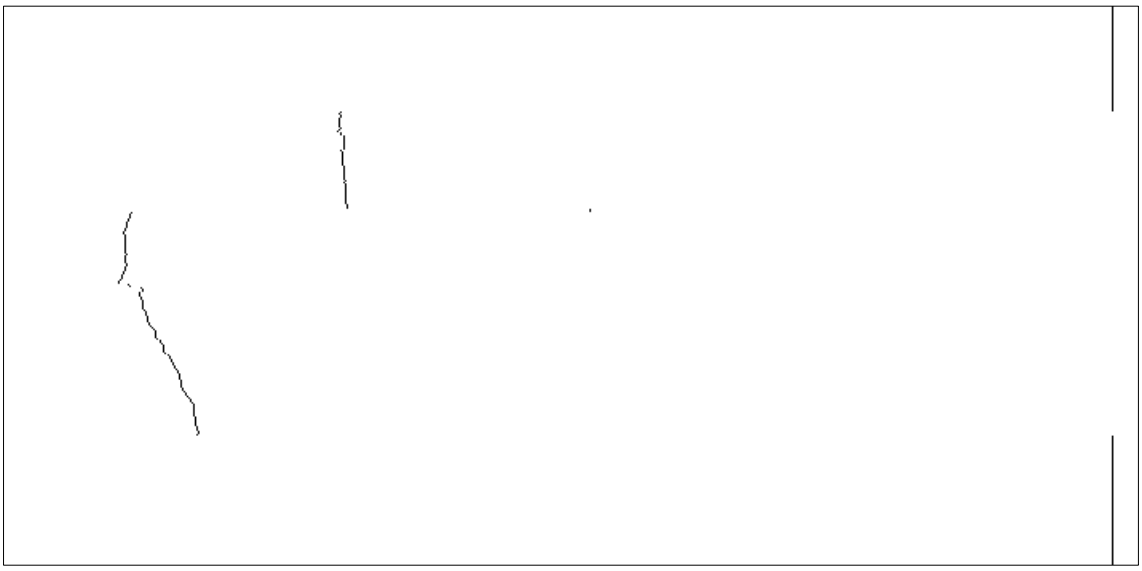

Figura 26 Top view do mapa de disparidade de um objeto de perfil. Mais a esquerda as linhas representando o contorno do objeto. Mais ao centro o contorno do segundo objeto. À direita linha representando os pontos onde não se encontrou disparidade.

4.2 TESTES DO PROTÓTIPO COMPLETO

Para analisar a desenvoltura do protótipo como um todo, considerando a integração do sistema de visão computacional com o conjunto robótico, foi

desenvolvido um teste onde, através das câmeras, o robô deve localizar uma bolinha na cor verde e se aproximar dela, até uma distância de 50 cm. A distância de 50 cm foi definida por ser um valor próximo do limite mínimo de distância mínima onde é possível fazer a leitura da disparidade, com alguma folga.

Foi desenvolvida uma função, junto com o algoritmo principal, que separa a imagem em camadas, conforme as cores vermelho, verde e azul (RGB) e subtrai as camadas entre si, afim de encontrar o objeto que contenha o componente predominante verde.

As câmeras utilizadas apresentam qualidade reduzida para a cor azul, fazendo com que a subtração da cor azul atrapalhe nos resultados. Assim sendo, o resultado da subtração somente da cor vermelha da camada de cor verde é melhor do que quando se subtrai também a cor azul.

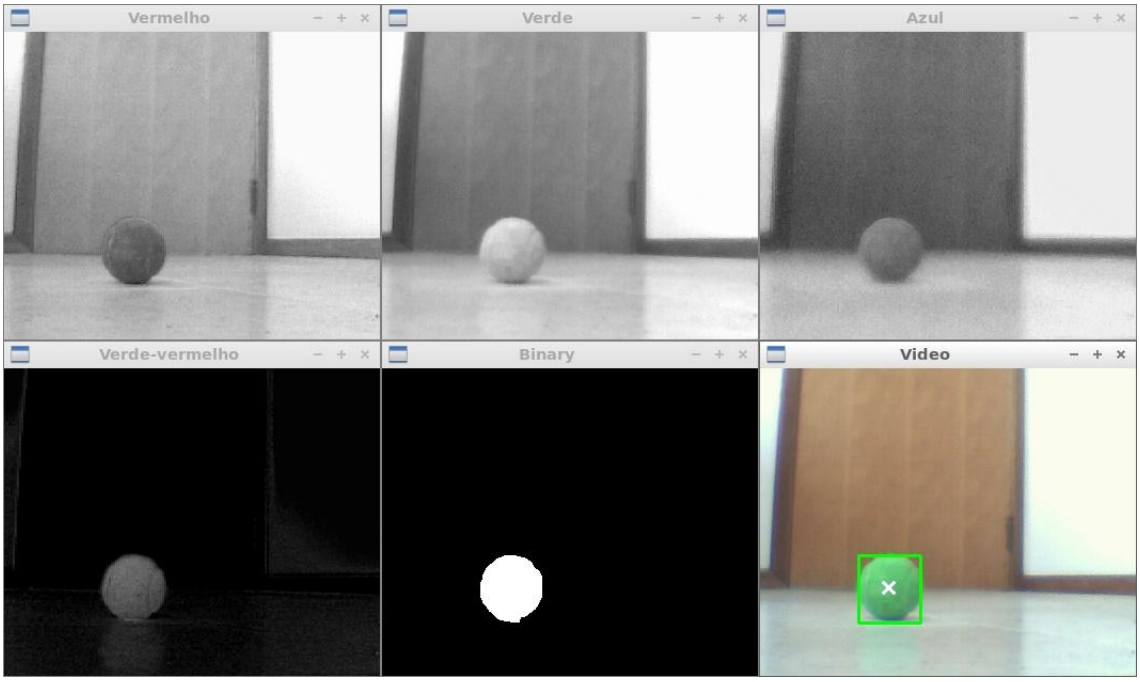

Figura 27 Decomposição da imagem nas cores vermelho verde e azul (topo), subtração da cor vermelha da cor verde (em baixo a esquerda), thresholding e binarização (em baixo no centro) e localização do centro da bolinha (em baixo a direita)

Após as subtrações das imagens é aplicado um *threshold* sobre a imagem resultante, de modo que só se mantenham pixels com intensidade de cor num intervalo entre 50 e 255, onde 255 é a intensidade máxima. Após a aplicação do *threshold* a imagem é binarizada, obtendo-se os valores 1 nos pixels que continham o componente de cor e 0 nos pixels que não o continham ou continham em intensidade inferior a 50. Com a imagem binarizada é possível calcular a posição do centro da bolinha. Este dado nos informa para onde o robô deve se virar e seguir.

A ***Figura 27*** ilustra o processo de decomposição em cores, subtração, *thresholding*, binarização e cálculo do centro da bolinha.

Tendo a posição da bolinha na imagem basta enviar os comandos ao Arduino para o robô se aproximar da bolinha e se virar para o centro da bolinha.

Resultados:

Quando o Arduino lê os comandos em intervalos de tempos aproximadamente igual ao intervalo de tempo que o computador leva para processar um par de imagens o robô se perde. Observando as imagens capturadas ao longo de cada passo, se observa um *delay* entre as imagens e o passo. Ou seja, a câmera passa para o computador imagens adquiridas em um pequeno intervalo de tempo anterior. Desta forma, o computador processa uma imagem atrasada e os comandos enviados não são válidos. Isto ocorre por que a câmera possui um *buffer* interno onde armazena os frames adquiridos e envia ao computador o primeiro frame que ainda estiver no *buffer*. Ou seja, se o *buffer* for grande o bastante para armazenar cinco imagens, o computador

receberá a imagem de cinco frames atrás, gerando comandos para o Arduino não condizentes com o comando que deveria ser gerado para aquele instante.

Este problema pode ser contornado reduzindo o intervalo de tempos entre as leituras dos comandos, e por conseqüência, o tamanho dos passos do motor. Desta forma os passos do robô são pequenos o bastante para que o problema causado pelo *delay* da câmera seja reduzido. Esta solução tem como efeito colateral uma velocidade muito pequena do robô, já que a taxa de comandos que o computador envia para o Arduino não aumenta. Apesar de esta solução reduzir a velocidade de deslocamento do protótipo, é uma via que retorna resultados positivos: O protótipo efetivamente se aproxima da bolinha e para a 50 cm dela.

Um vídeo com a demonstração do protótipo se deslocando em direção a bolinha se encontra disponível em: http://www.youtube.com/watch?v=gC2h5__X6J4

4.3 DIFICULDADES E PROBLEMAS

As principais dificuldades encontradas para visão estéreo ocorrem quando há pouca luminosidade ou luminosidade excessiva e quando o ambiente do qual se adquire as imagens é pobre em detalhes (como, por exemplo, uma parede com uma única cor e iluminação). Em todos estes casos, os blocos de pixels de uma imagem podem ser comparados com diversos outros blocos da outra imagem, impossibilitando o *block matching*.

Câmeras de baixa qualidade também podem ser problemas. Neste trabalho foram utilizadas duas webcams comuns. Ao se adquirir imagens com resolução 640x480 o ruído na imagem se tornou elevado, e os resultados do *block matching* não foram razoáveis. Uma redução na resolução para 480x320

resultou em resultados relativamente bons. Certamente, a utilização de câmeras com resolução maior e qualidade melhor trariam melhores resultados. Em contrapartida, resolução maior exigiria maior capacidade de processamento.

5. TRABALHOS CORRELATOS

Herath [Herath et al., 2006] em seu artigo apresenta uma combinação de métodos para resolver os problemas referentes ao SLAM em um ambiente interno restrito utilizando visão estéreo. As suas contribuições são a inclusão de algoritmos que encontram características ou detalhes e os rastreiam, um filtro de ruído estéreo e um robusto algoritmo de validação de características do ambiente.

As características são importantes na reconstrução estéreo, pois estas fazem com que os blocos de pixels sejam únicos em uma imagem de modo que seja possível a reconstrução. Conseguir rastrear estas características numa seqüência de imagens significa conseguir se localizar, no mapa que está sendo construído, a atual posição do robô.

Kim [Kim et al., 2008] aborda em seu trabalho um método de navegação que lida com ambientes dinâmicos, onde há objetos que se movem ou onde o próprio ambiente sofre transformações. É proposto um método de navegação baseado em movimentos em vez de métodos baseados em aparências. Desta forma, é feito um reconhecimento topológico pelas características do ambiente.

Outro assunto abordado em seu trabalho é a solução de problemas onde o robô sofre um deslocamento desconhecido ou sofre uma oclusão visual. Para isto é necessário que o robô faça um reconhecimento de sua localização global. Considerando há variações de iluminação e de pontos de vista, buscas por cenas similares falham na localização global. Como solução é utilizada uma formulação Bayesiana probabilística para a localização.

6. CONCLUSÕES

O objetivo de construir um protótipo robótico com sua navegação baseada em dados obtidos por sensores óticos foi atingido, ainda que com de forma limitada.

O uso de visão computacional, apesar de delicado, é viável e oferece grandes qualidades para a robótica. Com o uso de câmeras de boa qualidade e computadores com capacidade suficiente para o processamento das imagens destas câmeras pode-se obter uma boa reconstrução do ambiente em torno do robô.

Ainda que com câmeras de qualidade reduzida é possível fazer boa identificação de obstáculos, mesmo que estes estejam a distâncias consideravelmente grandes.

A construção de protótipos robóticos de baixo custo, com visão computacional, mostrou-se possível, onde o Arduino tornou-se um grande aliado. O protótipo apresenta velocidade baixa, da forma como foi implementada. Algumas melhorias podem ser feitas.

6.1 TRABALHOS FUTUROS

Tendo em vista os resultados alcançado com este trabalho, os seguintes trabalhos podem ser propostos:

- Testes com câmeras melhores. Qualidade melhor nas imagens resulta em uma riqueza maior de detalhes. Resolução maior nas imagens, apesar de requerer um poder de processamento maior fornece mais informação, como, por exemplo, uma maior resolução na leitura da profundidade. Câmeras com um *delay* menor entre o comando de captura de imagem e a imagem

retornada podem tornar o protótipo mais veloz na sua navegação. O *delay* foi o maior gargalo na velocidade do protótipo proposto;

- Utilização de técnicas SLAM (*Simultaneous Localization and Mapping*) para construção de mapas e localização nos mapas criados. O trabalho correlato de Herath [Herath et al., 2006] emprega técnicas de SLAM em visão estéreo, assim como o trabalho de Kim [Kim et al., 2008] aborda problemas típicos de localização;
- Estudo de trajetórias utilizando mapas construídos com a visão. Neste trabalho foram apenas contempladas trajetórias simples, onde o protótipo possuía visão direta ao objetivo, sem a transposição de obstáculos. O estudo de trajetórias mais complexas oferece ao protótipo uma navegação mais rica;
- Utilização de câmeras estéreo de fábrica. Pares de câmera posicionadas manualmente estão suscetíveis a erros grandes no alinhamento dos planos ópticos em relação a câmeras posicionadas paralelamente por métodos industriais ;
- Utilização de filtros por software para remoção de ruídos. Muitas vezes, ao fazer as aquisições, se observa ruídos que não são importantes na contemplação dos objetivos deste trabalho. Desta forma, a remoção destes é interessante, pois podem atrapalhar nas leituras do ambiente;
- Otimização da relação entre o Arduino e o computador. A comunicação entre o Arduino e o computador tratada neste trabalho é interessante considerando a sua robustez em relação a

falhas do computador. Porém, da forma proposta, requer uma boa sincronização entre as duas partes: à medida que o tempo entre comandos provindos do computador se torna maior que o tempo entre as leituras feitas pelo Arduino, a velocidade do protótipo se degrada.

REFERÊNCIAS BIBLIOGRÁFICAS

[Choset et al., 2005] Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G. A., Burgard, W., Kavraki, L. E. and Thrun, S. (2005). **Principles of Robot Motion: Theory, Algorithms, and Implementations.** MIT Press, Cambridge, MA.

[Dudek and Jenkin, 2010] Dudek, G. and Jenkin, M., (2010). **Computational Pinciples of Mobile Robotics**. Cambridge University Press, Cambridge, MA.

[Learning OpenCV, 2008] Bradski, G., Kaehler, A. (2008). **Learning OpenCV: Computer Vision with the OpenCV Library**. O'Reilly Media, USA.

[Herath et al., 2006] Herath, D. C., Kodagoda, S. and Dissanayake, G. (2006). **Simultaneous Localisation and Mapping: A Stereo Vision Based Approach.** International Conference on Inteligent Robots and Systems. Beijing, China.

[Kim et al., 2008] Kim, J. and Kweon, I. S. (2008). **Vision-based Autonomous Navigation based on Motion Estimation.** International Conference on Control, Automation and Systems. Seoul, Korea.

[OpenCV Reference] Intel. **Open Source Computer Vision Library: Reference Manual.** Intel Corporation, USA, 2000-2001.

[OpenCV] Disponível em http://opencv.org/. Acessado em 22 de maio de 2013.

[Stinger Robot] Disponível em http://www.roboticsconnection.com/p-73-stinger-robot-kit.aspx. Acessado em 15 de novembro de 2011.

[OpenCV Online Documentation] Disponível em http://opencv.willowgarage.com/documentation/. Acessado em 19 de maio de 2013.

[Arduino] Disponível em http://arduino.cc. Acessado em 15 de novembro de 2011.

[Otuyama, 1998] Otuyama, J. M., (1998). Visão Estéreo. Disponível em http://www.inf.ufsc.br/~visao/1998/otuyama/index.html. Acessado em 22 de junho de 2012.

[TIP120] Fairchild Semiconductor Corporation, 2007. TIP120/TIP121/TIP122 NPN Epitaxial Darlington Transistor Datasheet.

[Mendes & Wolf] Mendes, C. C. T and Wolf, D. F. . **Desvio de Obstáculos utilizando o Método Estéreo Semi-global.** Universidade de São Paulo (USP), São Carlos, SP, Brasil

[Mattoccia] Mattoccia, S. (2012). **Stereo Vision: Algorithms and Applications.** University of Bologna, Italy. Disponível em http://www.vision.deis.unibo.it/smatt/Seminars/StereoVision.pdf . Acessado em 23 de maio de 2013.

[Linux command - lsusb] Disponível em http://linuxcommand.org/man_pages/lsusb8.html. Acessado em 15 de maio de 2013.

[Kalman Filter – OpenCV Answers] Disponível em http://answers.opencv.org/question/4464/kalman-filter/. Acessado em 15 de maio de 2013.

[Arduino Serial Lib – GitHub] Disponível em https://github.com/todbot/arduino-serial/blob/master/arduino-serial-lib.c. Acessado em 16 de maio de 2013.

ANEXO A - DADOS DAS CÂMERAS

Os dados a seguir, referentes as câmeras, foram obtidos a partir dos comandos:

$ lsusb

e

$ lsusb -s 001:005 -v

digitados no terminal do Lubuntu. O primeiro comando lista os dispositivos USB que estão conectados a máquina [Linux command - lsusb]. O segundo comando mostra detalhes (listados a baixo) referentes ao dispositivo 005 (neste caso, a câmera identificada na lista do comando anterior) conectados ao barramento 001.

Bus 001 Device 005: ID 093a:2800 Pixart Imaging, Inc.

Couldn't open device, some information will be missing
Device Descriptor:
bLength 18
bDescriptorType 1
bcdUSB 2.00
bDeviceClass 239 Miscellaneous Device
bDeviceSubClass 2 ?
bDeviceProtocol 1 Interface Association
bMaxPacketSize0 64
idVendor 0x093a Pixart Imaging, Inc.
idProduct 0x2800
bcdDevice 1.00
iManufacturer 1
iProduct 2
iSerial 0
bNumConfigurations 1
Configuration Descriptor:
bLength 9
bDescriptorType 2
wTotalLength 511
bNumInterfaces 2
bConfigurationValue 1
iConfiguration 3
bmAttributes 0x80
(Bus Powered)
MaxPower 500mA
Interface Association:

```
    bLength                 8
    bDescriptorType        11
    bFirstInterface         0
    bInterfaceCount         2
    bFunctionClass         14 Video
    bFunctionSubClass       3 Video Interface Collection
    bFunctionProtocol       0
    iFunction               2
  Interface Descriptor:
    bLength                 9
    bDescriptorType         4
    bInterfaceNumber        0
    bAlternateSetting       0
    bNumEndpoints           1
    bInterfaceClass        14 Video
    bInterfaceSubClass      1 Video Control
    bInterfaceProtocol      0
    iInterface              2
    VideoControl Interface Descriptor:
      bLength              13
      bDescriptorType      36
      bDescriptorSubtype    1 (HEADER)
      bcdUVC               1.00
      wTotalLength         77
      dwClockFrequency     30.000000MHz
      bInCollection         1
      baInterfaceNr( 0)     1
    VideoControl Interface Descriptor:
      bLength               9
      bDescriptorType      36
      bDescriptorSubtype    3 (OUTPUT_TERMINAL)
      bTerminalID           2
      wTerminalType      0x0101 USB Streaming
      bAssocTerminal        0
      bSourceID             4
      iTerminal             0
    VideoControl Interface Descriptor:
      bLength              26
      bDescriptorType      36
      bDescriptorSubtype    6 (EXTENSION_UNIT)
      bUnitID               4
      guidExtensionCode     {f07735d1-898d-0047-812e-7dd5e2fdb898}
      bNumControl           8
      bNrPins               1
      baSourceID( 0)        3
      bControlSize          1
      bmControls( 0)     0xff
      iExtension            0
    VideoControl Interface Descriptor:
      bLength              18
      bDescriptorType      36
      bDescriptorSubtype    2 (INPUT_TERMINAL)
      bTerminalID           1
      wTerminalType      0x0201 Camera Sensor
      bAssocTerminal        0
```

```
    iTerminal             0
    wObjectiveFocalLengthMin      0
    wObjectiveFocalLengthMax      0
    wOcularFocalLength            0
    bControlSize                  3
    bmControls           0x00000200
      Zoom (Absolute)
  VideoControl Interface Descriptor:
    bLength                      11
    bDescriptorType              36
    bDescriptorSubtype            5 (PROCESSING_UNIT)
  Warning: Descriptor too short
    bUnitID                       3
    bSourceID                     1
    wMaxMultiplier                0
    bControlSize                  2
    bmControls           0x0000057f
      Brightness
      Contrast
      Hue
      Saturation
      Sharpness
      Gamma
      White Balance Temperature
      Backlight Compensation
      Power Line Frequency
    iProcessing                   0
    bmVideoStandards         0x20
      PAL - 525/60
  Endpoint Descriptor:
    bLength                       7
    bDescriptorType               5
    bEndpointAddress         0x83  EP 3 IN
    bmAttributes                  3
      Transfer Type               Interrupt
      Synch Type                  None
      Usage Type                  Data
    wMaxPacketSize           0x0010  1x 16 bytes
    bInterval                     6
Interface Descriptor:
  bLength                         9
  bDescriptorType                 4
  bInterfaceNumber                1
  bAlternateSetting               0
  bNumEndpoints                   0
  bInterfaceClass                14 Video
  bInterfaceSubClass              2 Video Streaming
  bInterfaceProtocol              0
  iInterface                      0
  VideoStreaming Interface Descriptor:
    bLength                      14
    bDescriptorType              36
    bDescriptorSubtype            1 (INPUT_HEADER)
    bNumFormats                   1
    wTotalLength                323
```

bEndPointAddress 129
bmInfo 0
bTerminalLink 2
bStillCaptureMethod 2
bTriggerSupport 1
bTriggerUsage 0
bControlSize 1
bmaControls(0) 27
VideoStreaming Interface Descriptor:
bLength 27
bDescriptorType 36
bDescriptorSubtype 4 (FORMAT_UNCOMPRESSED)
bFormatIndex 1
bNumFrameDescriptors 5
guidFormat {59555932-0000-1000-8000-00aa00389b71}
bBitsPerPixel 16
bDefaultFrameIndex 1
bAspectRatioX 0
bAspectRatioY 0
bmInterlaceFlags 0x00
Interlaced stream or variable: No
Fields per frame: 2 fields
Field 1 first: No
Field pattern: Field 1 only
bCopyProtect 0
VideoStreaming Interface Descriptor:
bLength 50
bDescriptorType 36
bDescriptorSubtype 5 (FRAME_UNCOMPRESSED)
bFrameIndex 1
bmCapabilities 0x00
Still image unsupported
wWidth 640
wHeight 480
dwMinBitRate 614400
dwMaxBitRate 18432000
dwMaxVideoFrameBufferSize 614400
dwDefaultFrameInterval 333333
bFrameIntervalType 6
dwFrameInterval(0) 333333
dwFrameInterval(1) 500000
dwFrameInterval(2) 666666
dwFrameInterval(3) 1000000
dwFrameInterval(4) 2000000
dwFrameInterval(5) 10000000
VideoStreaming Interface Descriptor:
bLength 50
bDescriptorType 36
bDescriptorSubtype 5 (FRAME_UNCOMPRESSED)
bFrameIndex 2
bmCapabilities 0x00
Still image unsupported
wWidth 320
wHeight 240
dwMinBitRate 153600

dwMaxBitRate 4608000
dwMaxVideoFrameBufferSize 153600
dwDefaultFrameInterval 333333
bFrameIntervalType 6
dwFrameInterval(0) 333333
dwFrameInterval(1) 500000
dwFrameInterval(2) 666666
dwFrameInterval(3) 1000000
dwFrameInterval(4) 1016960
dwFrameInterval(5) 10000000
VideoStreaming Interface Descriptor:
bLength 50
bDescriptorType 36
bDescriptorSubtype 5 (FRAME_UNCOMPRESSED)
bFrameIndex 3
bmCapabilities 0x00
Still image unsupported
wWidth 160
wHeight 120
dwMinBitRate 38400
dwMaxBitRate 1152000
dwMaxVideoFrameBufferSize 38400
dwDefaultFrameInterval 333333
bFrameIntervalType 6
dwFrameInterval(0) 333333
dwFrameInterval(1) 500000
dwFrameInterval(2) 666666
dwFrameInterval(3) 1000000
dwFrameInterval(4) 1016960
dwFrameInterval(5) 10000000
VideoStreaming Interface Descriptor:
bLength 50
bDescriptorType 36
bDescriptorSubtype 5 (FRAME_UNCOMPRESSED)
bFrameIndex 4
bmCapabilities 0x00
Still image unsupported
wWidth 176
wHeight 144
dwMinBitRate 50688
dwMaxBitRate 1520640
dwMaxVideoFrameBufferSize 50688
dwDefaultFrameInterval 333333
bFrameIntervalType 6
dwFrameInterval(0) 333333
dwFrameInterval(1) 500000
dwFrameInterval(2) 666666
dwFrameInterval(3) 1000000
dwFrameInterval(4) 1016960
dwFrameInterval(5) 10000000
VideoStreaming Interface Descriptor:
bLength 50
bDescriptorType 36
bDescriptorSubtype 5 (FRAME_UNCOMPRESSED)
bFrameIndex 5

```
    bmCapabilities                  0x00
      Still image unsupported
    wWidth                        352
    wHeight                       288
    dwMinBitRate                  202752
    dwMaxBitRate                  6082560
    dwMaxVideoFrameBufferSize     202752
    dwDefaultFrameInterval        333333
    bFrameIntervalType              6
    dwFrameInterval( 0)           333333
    dwFrameInterval( 1)           500000
    dwFrameInterval( 2)           666666
    dwFrameInterval( 3)           1000000
    dwFrameInterval( 4)           1016960
    dwFrameInterval( 5)           10000000
  VideoStreaming Interface Descriptor:
    bLength                       26
    bDescriptorType                 36
    bDescriptorSubtype              3 (STILL_IMAGE_FRAME)
    bEndpointAddress                0
    bNumImageSizePatterns           5
    wWidth( 0)                    640
    wHeight( 0)                   480
    wWidth( 1)                    320
    wHeight( 1)                   240
    wWidth( 2)                    160
    wHeight( 2)                   120
    wWidth( 3)                    176
    wHeight( 3)                   144
    wWidth( 4)                    352
    wHeight( 4)                   288
    bNumCompressionPatterns         5
  VideoStreaming Interface Descriptor:
    bLength                       6
    bDescriptorType                 36
    bDescriptorSubtype              13 (COLORFORMAT)
    bColorPrimaries                 1 (BT.709,sRGB)
    bTransferCharacteristics        1 (BT.709)
    bMatrixCoefficients             4 (SMPTE 170M (BT.601))
Interface Descriptor:
  bLength             9
  bDescriptorType     4
  bInterfaceNumber    1
  bAlternateSetting   1
  bNumEndpoints       1
  bInterfaceClass    14 Video
  bInterfaceSubClass  2 Video Streaming
  bInterfaceProtocol  0
  iInterface          0
  Endpoint Descriptor:
    bLength             7
    bDescriptorType     5
    bEndpointAddress    0x81  EP 1 IN
    bmAttributes        5
      Transfer Type          Isochronous
```

```
      Synch Type              Asynchronous
      Usage Type              Data
    wMaxPacketSize     0x0a60  2x 608 bytes
    bInterval               1
Interface Descriptor:
  bLength                 9
  bDescriptorType         4
  bInterfaceNumber        1
  bAlternateSetting       2
  bNumEndpoints           1
  bInterfaceClass        14 Video
  bInterfaceSubClass      2 Video Streaming
  bInterfaceProtocol      0
  iInterface              0
  Endpoint Descriptor:
    bLength               7
    bDescriptorType       5
    bEndpointAddress    0x81  EP 1 IN
    bmAttributes          5
      Transfer Type           Isochronous
      Synch Type              Asynchronous
      Usage Type              Data
    wMaxPacketSize    0x0b20  2x 800 bytes
    bInterval             1
Interface Descriptor:
  bLength                 9
  bDescriptorType         4
  bInterfaceNumber        1
  bAlternateSetting       3
  bNumEndpoints           1
  bInterfaceClass        14 Video
  bInterfaceSubClass      2 Video Streaming
  bInterfaceProtocol      0
  iInterface              0
  Endpoint Descriptor:
    bLength               7
    bDescriptorType       5
    bEndpointAddress    0x81  EP 1 IN
    bmAttributes          5
      Transfer Type           Isochronous
      Synch Type              Asynchronous
      Usage Type              Data
    wMaxPacketSize    0x1300  3x 768 bytes
    bInterval             1
Interface Descriptor:
  bLength                 9
  bDescriptorType         4
  bInterfaceNumber        1
  bAlternateSetting       4
  bNumEndpoints           1
  bInterfaceClass        14 Video
  bInterfaceSubClass      2 Video Streaming
  bInterfaceProtocol      0
  iInterface              0
  Endpoint Descriptor:
```

bLength 7
bDescriptorType 5
bEndpointAddress 0x81 EP 1 IN
bmAttributes 5
Transfer Type Isochronous
Synch Type Asynchronous
Usage Type Data
wMaxPacketSize 0x13fc 3x 1020 bytes
bInterval 1

ANEXO B – CÓDIGOS DO ARDUINO

Este código é executado no Arduino e controla as rodas do robô.

```
char byteRecebido = -1;
//iniciando velocidades das rodas
int vDireita = 0;
int vEsquerda = 0;
int contDireita = 0;   //contadores auxiliares
int contEsquerda = 0;
int rodaEsquerda = 11; //pino do arduino que aciona a roda Esquerda
int rodaDireita = 9; //pino do arduino que aciona a roda Direita
int tempoEntreLeitura = 70; //deve ser proximo ao fps das cameras
int quantTiques = 10; //quantidade de passos em que uma leitura dividida
int tempoDelay = tempoEntreLeitura/quantTiques;

void setup() {
      Serial.begin(9600);     // opens serial port, sets data rate to 9600 bps
      pinMode(rodaEsquerda, OUTPUT);
      pinMode(rodaDireita, OUTPUT);
}

void loop() {
      lerEntrada();
      andar();
}

void lerEntrada(){
       // send data only when you receive data:
      if (Serial.available() > 0) {
            // lendo o byte:
            byteRecebido =  Serial.read();
            //limpar o resto dos bytes do buffer. Evita que comandos atrasados sejam executados
            while(Serial.available()>0){
              Serial.read();
            }
            // se o byte indica para ir para a esquerda:
```

```
                if(byteRecebido == 'e'){
                  if(vDireita<quantTiques)
                    vDireita++;   //e a velocidade da roda DIREITA aumenta
                  else if(vEsquerda>0)
                    vEsquerda--;  //velocidade da roda ESQUERDA diminui

                }
                // se o byte indica para ir para a direita:
                else if(byteRecebido == 'd'){
                 if(vEsquerda<quantTiques)
                  vEsquerda++;
                else if(vDireita>0)
                  vDireita--;
                }
                //se o byte indicar para parar (emergncia):
                else if(byteRecebido == 'p'){
                  vDireita = 0;
                  vEsquerda = 0;
                }
                //se o byte indicar para seguir -> as duas rodas aumentam a velocidade
                else if(byteRecebido == 's'){
                  if(vDireita<quantTiques)
                    vDireita++;
                  if(vEsquerda<quantTiques)
                    vEsquerda++;
                }
            }
            // se o computador no falar nada -> desacelera gradualmente.
            else{
              if(vDireita>0)
                vDireita--;
              if(vEsquerda>0)
                vEsquerda--;
            }
        }
```

```
void andar(){
  contDireita = vDireita;
  contEsquerda = vEsquerda;
  for(int i = 0; i<quantTiques; i++){
    if(contDireita>0){        //liga a roda direita por uma quantidade de vezes equivalente a velocidade
      digitalWrite(rodaDireita, HIGH);
      contDireita--;
    }
    else
      digitalWrite(rodaDireita, LOW);
    if(contEsquerda>0){      //liga a roda esquerda por uma quantidade de vezes equivalente a velocidade
      digitalWrite(rodaEsquerda, HIGH);
      contEsquerda--;
    }
    else
      digitalWrite(rodaEsquerda, LOW);
    delay(tempoDelay);
  }
}
```

ANEXO C – CÓDIGO DE CAPTURA DE IMAGENS ESTÉREO

Este código é responsável por fazer a captura de 15 pares de imagens para servir de entrada para a calibração de câmeras.

```
#include <stdio.h>
#include <time.h>
#include "opencv2/highgui/highgui.hpp"

int main( int argc, char **argv){
    CvCapture *capture = 0;
    CvCapture *capture2 = 0;
    CvCapture *capture0 = 0;
    IplImage  *frame = 0;
 IplImage  *frame2 = 0;
    int       key = 0;
  bool fim = false;
  int umaCamera = 0;
  int outraCamera = 0;

//verificando cameras existentes e escolhendo cameras diferentes que a 0
for(int i = -1; i<10; i++){
  capture0 = cvCreateCameraCapture(i);
  if(capture0){
            fprintf(stderr, "Camera %d conectada!\n", i);
            if(i>0){
                    if(umaCamera == 0)
                            umaCamera = i;
                    else
                            outraCamera = i;
            }
  }
  cvReleaseCapture( &capture0 );
}
```

```
 /* initialize camera */
 capture = cvCaptureFromCAM(umaCamera);
 capture2 = cvCaptureFromCAM(outraCamera);

cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, 480);
cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, 360);
cvSetCaptureProperty(capture2, CV_CAP_PROP_FRAME_WIDTH, 480);
cvSetCaptureProperty(capture2, CV_CAP_PROP_FRAME_HEIGHT, 360);

 /* always check */
 if ( !capture ) {
    fprintf( stderr, "Impossivel iniciar camera 1!\n" );
    return 1;
 }
 if ( !capture2 ) {
    fprintf( stderr, "Impossivel iniciar camera 2!\n" );
    return 1;
 }

 /* create a window for the video */
 cvNamedWindow( "Camera Direita", CV_WINDOW_AUTOSIZE );
 cvNamedWindow( "Camera Esquerda", CV_WINDOW_AUTOSIZE );

int contador = 1;
clock_t tempo = clock();
char arquivoEsquerda[100];
char arquivoDireita[100];

 while( key != 'q' && !fim) {

   /* get a frame */
    frame = cvQueryFrame( capture );
frame2 = cvQueryFrame( capture2 );

    /* always check */
    if( !frame and !frame2) break;
```

```
  if((clock()-tempo)>2 * CLOCKS_PER_SEC && contador <= 15){
          //salvando imagens
          sprintf(arquivoEsquerda, "imagensStereo/%de.png", contador);
          sprintf(arquivoDireita, "imagensStereo/%dd.png", contador);
          cvSaveImage(arquivoEsquerda,frame);
          cvSaveImage(arquivoDireita,frame2);
          fprintf(stderr, "Captura %d realizada\n", contador );
          contador++;
          tempo = clock();

  }
  if(contador>=15){
          fprintf(stderr, "Fim da captura. Encerrando o programa!\n");
          fim = true;
  }
      /* display current frame */
      cvShowImage( "Camera Esquerda", frame );
  cvShowImage( "Camera Direita", frame2 );

    key = cvWaitKey( 1 );

   }

   /* free memory */
   cvDestroyWindow( "Camera Esquerda" );
   cvDestroyWindow( "Camera Direita" );
   cvReleaseCapture( &capture );
   cvReleaseCapture( &capture2 );
   return 0;
}
```

ANEXO D – CÓDIGO PRINCIPAL

Este código captura as imagens, as interpreta e se comunica com o Arduino. Trechos deste código são oriundos de [Kalman Filter – OpenCV Answers] e [Arduino Serial Lib – GitHub], além de outros sites consultados.

```
#include <stdio.h> // Standard input/output definitions
#include "../../OpenCV-2.3.1/include/opencv/cv.h"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/video/tracking.hpp>
#include "arduino-serial-lib.h"
#include <unistd.h>   // UNIX standard function definitions
#include <fcntl.h>    // File control definitions
#include <errno.h>    // Error number definitions
#include <termios.h>  // POSIX terminal control definitions
#include <string.h>   // String function definitions
#include <sys/ioctl.h>

using namespace std;
using namespace cv;

//inicializando parametros para o BMState
CvStereoBMState* BMState;
 int preFilterSize=41;           //31/31
int preFilterCap=31;             //63/31
int SADWindowSize=41;            //41/15;
int minDisparity=64;             //-64/-192
int numberOfDisparities=8;       //8*16/192
int textureThreshold=10;         //10/10
int uniquenessRatio=15;          //15/15
```

```
void on_trackBar( int )
{
  //preFilterSize precisa ser entre 5..255 e impar
  if(preFilterSize<5)
        BMState->preFilterSize = 5;
  else if (preFilterSize%2 == 1)
        BMState->preFilterSize = preFilterSize;
  //precisa ser maior que 1
  if(preFilterCap>=1)
        BMState->preFilterCap = preFilterCap;
  //SADWindowSize precisa ser entre 5..255 e impar
  if(SADWindowSize<5)
        BMState->SADWindowSize = 5;
  else if (SADWindowSize%2 == 1)
        BMState->SADWindowSize = SADWindowSize;
  BMState->minDisparity = -minDisparity;
  //numberOfDisparities precisa ser maior que 1
  if(numberOfDisparities>=1)
        BMState->numberOfDisparities = numberOfDisparities*16;
  BMState->textureThreshold = textureThreshold;
  BMState->uniquenessRatio = uniquenessRatio;
}

//parametros relativos ao filtro de kalman
Mat thresh_frame;
vector<Mat> channels;
CvCapture *capture;
vector<Vec4i> hierarchy;
vector<vector<Point> > contours;

//------------Kalman------------
#define drawCross( img, center, color, d )\
line(img, Point(center.x - d, center.y - d), Point(center.x + d, center.y + d), color,
2, CV_AA, 0);\
line(img, Point(center.x + d, center.y - d), Point(center.x - d, center.y + d), color,
2, CV_AA, 0 );
```

```
Point kalman(Mat frame, CvSize size, KalmanFilter KF, Mat_<float> measurement){
    split(frame, channels);
      //utilizando os canais vermelho e verde. O canal azul da camera é ruim
    subtract(channels[1], channels[2], channels[1]);
    imshow("1-2", channels[1]);
    threshold(channels[1], thresh_frame, 50, 255, CV_THRESH_BINARY);
    medianBlur(thresh_frame, thresh_frame, 5);
    findContours(thresh_frame, contours, hierarchy, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
    Mat drawing = Mat::zeros(thresh_frame.size(), CV_8UC1);
    for(size_t i = 0; i < contours.size(); i++)
    {
        if(contourArea(contours[i]) > 500)
            drawContours(drawing, contours, i, Scalar::all(255), CV_FILLED, 8, vector<Vec4i>(), 0, Point());
    }
    thresh_frame = drawing;

    findContours(thresh_frame, contours, hierarchy, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
    drawing = Mat::zeros(thresh_frame.size(), CV_8UC1);
    for(size_t i = 0; i < contours.size(); i++)
    {
        if(contourArea(contours[i]) > 500)
            drawContours(drawing, contours, i, Scalar::all(255), CV_FILLED, 8, vector<Vec4i>(), 0, Point());
    }
    thresh_frame = drawing;

   // Get the moments
    vector<Moments> mu(contours.size() );
    for( size_t i = 0; i < contours.size(); i++ )
        { mu[i] = moments( contours[i], false ); }

   //  Get the mass centers:
    vector<Point2f> mc( contours.size() );
```

```
for( size_t i = 0; i < contours.size(); i++ )
        { mc[i] = Point2f( mu[i].m10/mu[i].m00 , mu[i].m01/mu[i].m00 ); }
Mat prediction = KF.predict();
Point predictPt(prediction.at<float>(0),prediction.at<float>(1));
for(size_t i = 0; i < mc.size(); i++)
{
        drawCross(frame, mc[i], Scalar(255, 0, 0), 5);
        measurement(0) = mc[i].x;
        measurement(1) = mc[i].y;
}
Point measPt(measurement(0),measurement(1));
Mat estimated = KF.correct(measurement);
Point statePt(estimated.at<float>(0),estimated.at<float>(1));
drawCross(frame, statePt, Scalar(255, 255, 255), 5);
printf("( %d , %d )\n",statePt.x, statePt.y );
vector<vector<Point> > contours_poly( contours.size() );
vector<Rect> boundRect( contours.size() );
for( size_t i = 0; i < contours.size(); i++ )
{
        approxPolyDP( Mat(contours[i]), contours_poly[i], 3, true );
        boundRect[i] = boundingRect( Mat(contours_poly[i]) );
}
for( size_t i = 0; i < contours.size(); i++ )
{
        rectangle( frame, boundRect[i].tl(), boundRect[i].br(), Scalar(0, 255, 0),
2, 8, 0 );
}
//mostrar o centro da imagem
line(frame, Point(size.width/2,0), Point(size.width/2, size.height), Scalar(255,
255, 255), 1, CV_AA, 0 );
imshow("Video", frame);
imshow("Binary", thresh_frame);

if(contours.size()<1){
        return Point(0,0);
}
return statePt;
```

```
}

//------------COMUNICACAO COM O ARDUINO
// uncomment this to debug reads
//#define SERIALPORTDEBUG
// takes the string name of the serial port (e.g. "/dev/tty.usbserial","COM1")
// and a baud rate (bps) and connects to that port at that speed and 8N1.
// opens the port in fully raw mode so you can send binary data.
// returns valid fd, or -1 on error
int serialport_init(const char* serialport, int baud)
{
  struct termios toptions;
  int fd;

   //fd = open(serialport, O_RDWR | O_NOCTTY | O_NDELAY);
  fd = open(serialport, O_RDWR | O_NONBLOCK );

  if (fd == -1)  {
            perror("serialport_init: Unable to open port ");
            return -1;
  }

   //int iflags = TIOCM_DTR;
   //ioctl(fd, TIOCMBIS, &iflags);     // turn on DTR
   //ioctl(fd, TIOCMBIC, &iflags);    // turn off DTR

  if (tcgetattr(fd, &toptions) < 0) {
            perror("serialport_init: Couldn't get term attributes");
            return -1;
  }
   speed_t brate = baud; // let you override switch below if needed
   switch(baud) {
            case 4800:   brate=B4800;   break;
            case 9600:   brate=B9600;   break;
#ifdef B14400
            case 14400:  brate=B14400;  break;
#endif
```

```
            case 19200:  brate=B19200;  break;
#ifdef B28800
            case 28800:  brate=B28800;  break;
#endif
            case 38400:  brate=B38400;  break;
            case 57600:  brate=B57600;  break;
            case 115200: brate=B115200; break;
    }
    cfsetispeed(&toptions, brate);
    cfsetospeed(&toptions, brate);

    // 8N1
    toptions.c_cflag &= ~PARENB;
    toptions.c_cflag &= ~CSTOPB;
    toptions.c_cflag &= ~CSIZE;
    toptions.c_cflag |= CS8;
    // no flow control
    toptions.c_cflag &= ~CRTSCTS;

    //toptions.c_cflag &= ~HUPCL; // disable hang-up-on-close to avoid reset

    toptions.c_cflag |= CREAD | CLOCAL;  // turn on READ & ignore ctrl lines
    toptions.c_iflag &= ~(IXON | IXOFF | IXANY); // turn off s/w flow ctrl

    toptions.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); // make raw
    toptions.c_oflag &= ~OPOST; // make raw

    // see: http://unixwiz.net/techtips/termios-vmin-vtime.html
    toptions.c_cc[VMIN]  = 0;
    toptions.c_cc[VTIME] = 0;
    //toptions.c_cc[VTIME] = 20;

    tcsetattr(fd, TCSANOW, &toptions);
    if( tcsetattr(fd, TCSAFLUSH, &toptions) < 0) {
            perror("init_serialport: Couldn't set term attributes");
            return -1;
    }
```

```
    return fd;
}

//
int serialport_close( int fd )
{
  return close( fd );
}

//
int serialport_writebyte( int fd, uint8_t b)
{
  int n = write(fd,&b,1);
  if( n!=1)
            return -1;
  return 0;
}

//
int serialport_write(int fd, const char* str)
{
  int len = strlen(str);
  int n = write(fd, str, len);
  if( n!=len ) {
            perror("serialport_write: couldn't write whole string\n");
            return -1;
  }
  return 0;
}

//
int serialport_read_until(int fd, char* buf, char until, int buf_max, int timeout)
{
    char b[1];  // read expects an array, so we give it a 1-byte array
    int i=0;
    do {
```

```
        int n = read(fd, b, 1);  // read a char at a time
        if( n==-1) return -1;    // couldn't read
        if( n==0 ) {
            usleep( 1 * 1000 );  // wait 1 msec try again
            timeout--;
            continue;
        }
#ifdef SERIALPORTDEBUG
        printf("serialport_read_until: i=%d, n=%d b='%c'\n",i,n,b[0]); // debug
#endif
        buf[i] = b[0];
        i++;
    } while( b[0] != until && i < buf_max && timeout>0 );

    buf[i] = 0;  // null terminate the string
    return 0;
}

//
int serialport_flush(int fd)
{
    sleep(2); //required to make flush work, for some reason
    return tcflush(fd, TCIOFLUSH);
}

void erro(char* msg)
{
  fprintf(stderr, "%s\n",msg);
  return;
}


int main( int argc, char **argv){
  CvCapture *capture = 0;
  CvCapture *capture2 = 0;
  CvCapture *capture0 = 0;
  IplImage  *frame = 0;
```

```
IplImage  *frame2 = 0;
IplImage* vistaSuperior=cvCreateImage(cvSize(480,1000),IPL_DEPTH_8U,1);

int key = 0;
int umaCamera = 0;
int outraCamera = 0;

double maiorValor = 0;
double distanciaEntreCameras = 73.5; //milimetros
double z = 0;  //profundidade
double zMin = 10000;
double disparidadeMaxima = 0;

int auxiliar = 0 ;
CvScalar s;

CvSize size;
IplImage* disparity_left;
IplImage* real_disparity;
IplImage* img_esquerda;
IplImage* img_direita;

IplImage* img_esquerda_;
IplImage* img_direita_;

Point position; //posicao da bolinha

KalmanFilter KF(4, 2, 0);
Mat_<float> state(4, 1);
Mat_<float> processNoise(4, 1, CV_32F);
Mat_<float> measurement(2,1);
measurement.setTo(Scalar(0));

KF.statePre.at<float>(0) = 0;
KF.statePre.at<float>(1) = 0;
KF.statePre.at<float>(2) = 0;
KF.statePre.at<float>(3) = 0;
```

```
        KF.transitionMatrix = *(Mat_<float>(4, 4) << 1,0,1,0,   0,1,0,1,  0,0,1,0,  0,0,0,1);
// Including velocity
        KF.processNoiseCov  =  *(cv::Mat_<float>(4,4)  <<  0.2,0,0.2,0,     0,0.2,0,0.2,
0,0,0.3,0,  0,0,0,0.3);

        setIdentity(KF.measurementMatrix);
        setIdentity(KF.processNoiseCov, Scalar::all(1e-4));
        setIdentity(KF.measurementNoiseCov, Scalar::all(1e-1));
        setIdentity(KF.errorCovPost, Scalar::all(.1));

      //--- VARIAVEIS DE COMUNICACAO COM O ARDUINO
        const int buf_max = 256;
        int fd = -1;
        char serialport[] = "/dev/ttyACM0";
        int baudrate = 9600;  // default
        char quiet=0;
        char eolchar = '\n';
        int timeout = 5000;
        char buf[buf_max];
        int rc,n;

        fd = serialport_init(serialport, baudrate);
        if( fd==-1 ) erro("couldn't open port");
        if(!quiet) printf("opened port %s\n", serialport);
        serialport_flush(fd);



      //verificando cameras existentes e escolhendo cameras diferentes que a 0
        for(int i = -1; i<10; i++){
                capture0 = cvCreateCameraCapture(i);
                if(capture0){
                        fprintf(stderr, "Camera %d conectada!\n", i);
                        if(i>0){
                                if(umaCamera == 0)
                                        umaCamera = i;
```

```
                        else
                                outraCamera = i;
                    }
                }
                cvReleaseCapture( &capture0 );
        }



         /* initialize camera */
        capture = cvCaptureFromCAM(umaCamera);
        capture2 = cvCaptureFromCAM(outraCamera);

        cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, 480);
        cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, 360);
        cvSetCaptureProperty(capture2, CV_CAP_PROP_FRAME_WIDTH, 480);
        cvSetCaptureProperty(capture2, CV_CAP_PROP_FRAME_HEIGHT, 360);

        printf("brilho -> %f\n", cvGetCaptureProperty(capture,
CV_CAP_PROP_BRIGHTNESS));
        printf("saturacao -> %f\n", cvGetCaptureProperty(capture,
CV_CAP_PROP_SATURATION));
        printf("contraste -> %f\n", cvGetCaptureProperty(capture,
CV_CAP_PROP_CONTRAST));
        printf("hue -> %f\n", cvGetCaptureProperty(capture, CV_CAP_PROP_HUE));

         /* always check */
        if ( !capture ) {
                fprintf( stderr, "Impossivel iniciar camera 1!\n" );
                return 1;
        }
        if ( !capture2 ) {
                fprintf( stderr, "Impossivel iniciar camera 2!\n" );
                return 1;
        }
```

```
    frame = cvQueryFrame( capture );
    frame2 = cvQueryFrame( capture2 );

  // image_left and image_right are the input 8-bit single-channel images
  // from the left and the right cameras, respectively
    size = cvGetSize(frame);
    disparity_left =  cvCreateImage( size, IPL_DEPTH_16S, 1 );
    real_disparity =  cvCreateImage( size, IPL_DEPTH_8U, 1 );
    IplImage* depth = cvCreateImage( size, IPL_DEPTH_32F, 3 );

    BMState = cvCreateStereoBMState();
    assert(BMState != 0);
    BMState->preFilterSize = preFilterSize;
    BMState->preFilterCap = preFilterCap;
    BMState->SADWindowSize = SADWindowSize;
    BMState->minDisparity = -minDisparity;
    BMState->numberOfDisparities = numberOfDisparities*16;
    BMState->textureThreshold = textureThreshold;
    BMState->uniquenessRatio = uniquenessRatio;

  //criando janela com barras para configurar o BMState
    cvNamedWindow("BMState", 1);
    cvCreateTrackbar( "PréFilterSize", "BMState", &preFilterSize, 255, on_trackBar
);
    cvCreateTrackbar( "PréFilterCap", "BMState", &preFilterCap, 63, on_trackBar );
    cvCreateTrackbar( "SADWindowSize", "BMState", &SADWindowSize, 100,
on_trackBar );
    cvCreateTrackbar( "minDisparity", "BMState", &minDisparity, 200, on_trackBar
);
    cvCreateTrackbar( "numberOfDisparities (x16)", "BMState",
&numberOfDisparities, 16, on_trackBar );
    cvCreateTrackbar( "textureThreshold", "BMState", &textureThreshold, 100,
on_trackBar );
    cvCreateTrackbar( "uniquenessRatio", "BMState", &uniquenessRatio, 100,
on_trackBar );
```

```
#if 1
//    BMState->minDisparity *= size.width/640;
//    BMState->numberOfDisparities *= size.width/640;
#endif

  img_esquerda = cvCreateImage(size, IPL_DEPTH_8U, 1);
  img_direita = cvCreateImage(size, IPL_DEPTH_8U, 1);
  img_esquerda_ = cvCreateImage(size, IPL_DEPTH_8U, 1);
  img_direita_ = cvCreateImage(size, IPL_DEPTH_8U, 1);

  CvMat* disparity_left_visual = cvCreateMat( size.height, size.width, CV_8U );

//carregando dados de calibraçao
  CvMat *mx1 = (CvMat *)cvLoad("mx1.xml",NULL,NULL,NULL);
  CvMat *my1 = (CvMat *)cvLoad("my1.xml",NULL,NULL,NULL);
  CvMat *mx2 = (CvMat *)cvLoad("mx2.xml",NULL,NULL,NULL);
  CvMat *my2 = (CvMat *)cvLoad("my2.xml",NULL,NULL,NULL);
  CvMat *q = (CvMat *)cvLoad("Q.xml", NULL, NULL, NULL);
//carregando a "camera matrix" -> a distancia focal é igual para as duas cameras
  CvMat *cameraMatrix = (CvMat *)cvLoad("M1.xml",NULL,NULL,NULL);
  double distanciaFocal = CV_MAT_ELEM( *cameraMatrix, double, 0, 0 );
  printf("Distancia Focal: %f\n", distanciaFocal);

   /* create a window for the video */

  while( key != 'q' ) {

  /* get a frame */
          frame = cvQueryFrame( capture );
          frame2 = cvQueryFrame( capture2 );

      /* always check */
          if( !frame and !frame2) break;

          cvCvtColor(frame2, img_direita, CV_RGB2GRAY);
          cvCvtColor(frame, img_esquerda, CV_RGB2GRAY);
//fazendo remapeamento
```

```
            cvRemap(img_esquerda, img_esquerda_, mx1, my1);
            cvRemap(img_direita, img_direita_, mx2, my2);

        /* display current frame */
            cvShowImage( "Camera Esquerda", img_esquerda_ );
            cvShowImage( "Camera Direita", img_direita_ );

            cvFindStereoCorrespondenceBM( img_esquerda_, img_direita_,
disparity_left, BMState);
            cvNormalize( disparity_left, disparity_left_visual, 0, 256, CV_MINMAX );
            cvShowImage( "BM normalizado", disparity_left_visual );

            cvSet(vistaSuperior,cvScalar(0));
    //top View

            zMin= 10000;
            disparidadeMaxima = 0;
    //top view
            for(int i=0; i<size.width; i++){
            maiorValor = -3020,000000;  //-3088
            for(int j = 0; j<size.height; j++){
                    s=cvGet2D(disparity_left, j, i);  //pegando o valor do pixel
                    if(maiorValor < s.val[0])
                            maiorValor = s.val[0];
            }
            s.val[0] = 111;

            z = -1571.471 + 92386.45/sqrt(maiorValor+1040); //aproximacao
    //      printf("Em %d a disparidade eh %f e z = %f\n",i,maiorValor, z);
            if(z<zMin){
                    zMin = z;
                    disparidadeMaxima = maiorValor+1040;
            }
            auxiliar = -(int) maiorValor/3+350;
            if(auxiliar<0) {
                    auxiliar=0;
                    printf("Em %d a intensidade=%f\n",i,maiorValor);
```

```
            }
            cvSet2D(vistaSuperior,auxiliar,i,s);
        }
        cvShowImage("Vista Superior", vistaSuperior);
        printf("Distancia Minima medida eh %f e disparidadeMaxima eh %f\n ",zMin, disparidadeMaxima);

        if(zMin<500){ //mais perto que 500 milimetros
            printf("Obstáculo a 50 cm\n");
        }
        else{
            position = kalman(frame, size, KF, measurement);
            printf("Bolinha em ( %d , %d )\n",position.x, position.y );
            if(position.x>0 && position.x < size.width/2-20){ //se estiver fora do centro horizontal da imagem com folga de 20 px
                printf("Virar para a direita\n");
                rc = serialport_writebyte(fd, 'd');//diz pro arduino para virar
                if(rc==-1) erro("Erro ao falar com o arduino");
            }
            else if (position.x > size.width/2+20){
                printf("Virar para a esquerda\n");
                rc = serialport_writebyte(fd, 'e');//diz pro arduino para virar
                if(rc==-1) erro("Erro ao falar com o arduino");
            }
            else if(position.x==0){
                printf("Nao achei a bolinha\n");
                rc = serialport_writebyte(fd, 'p');//diz pro arduino para virar
                if(rc==-1) erro("Erro ao falar com o arduino");
            }
            else{
                printf("Tá no meio\n");
                rc = serialport_writebyte(fd, 's');//diz pro arduino serguir
                if(rc==-1) erro("Erro ao falar com o arduino");
            }
        }
        /* exit if user press 'q' */
        key = cvWaitKey( 1 );
```

```
}

cvReleaseStereoBMState( &BMState );
    /* free memory */
cvDestroyWindow( "Camera 1" );
cvDestroyWindow( "Camera 2" );
cvReleaseCapture( &capture );
cvReleaseCapture( &capture2 );
return 0;
}
```