DESENVOLVIMENTO E IMPLEMENTAÇÃO DO SISTEMA DE VISÃO DE UM ROBÔ HUMANOIDE

Thomas Vergutz, tvergutz@gmail.com[1]
Rogério Sales Gonçalves, rsgoncalves@mecanica.ufu.br[1]

[1]Laboratório de Automação e Robótica, Faculdade de Engenharia Mecânica, Universidade Federal de Uberlândia, Campus Santa Mônica – Bloco 1M – Av. João Naves de Ávila, 2121, Uberlândia – MG – CEP 38408-902

Resumo: *A utilização de sistemas inteligentes móveis e interativos cresce a cada dia e neste trabalho é mostrada como a interface homem-máquina é substancialmente melhorada com o uso de um sistema de visão robótica. No presente trabalho é apresentada uma solução para a interação de robôs humanoides com o meio ambiente a partir de câmeras e processamento digital de imagens. O mesmo tem por objetivo usar a biblioteca OpenCV para o processamento de imagens, e por finalidade embarcar esta visão computacional em um sistema robótico móvel, que é capaz de jogar futebol de forma autônoma.*

Palavras-chave: *Visão computacional, robótica, robôs humanoides, OpenCV*

1. INTRODUÇÃO

Nas últimas décadas, observam-se inovações crescentes nas áreas de robótica, computação e interface homem-máquina, pela necessidade cada vez maior sentida pelas pessoas de inserir a tecnologia em suas vidas; o mundo encontra-se em um momento que é difícil separar nosso cotidiano da tecnologia. Este fato é explorado pelas empresas do ramo, sabendo que a geração atual clama por novas tecnologias e novas maneiras de incorporá-las a suas vidas.

A visão computacional, que pode ser definida como a transformação de dados de uma imagem ou vídeo em uma decisão ou em outra representação (Bradsky e Kaehler, 2008), não é mais vista como um problema isolado de análise de imagens, e sim como um método de interação entre máquina e meio ambiente.

Esta interação nos remete a inteligência artificial, que é a área de pesquisa que se dedica a buscar dispositivos ou métodos de mimetizar ou até mesmo superar a inteligência humana. A inteligência artificial (IA) também pode ser definida como o ramo da ciência da computação que se ocupa da automação do comportamento inteligente (Luger, 2004).

Com esta nova ferramenta, a visão computacional, abre-se um leque de oportunidades para melhorar a inteligência artificial de autômatos que devem se posicionar no meio ambiente. E, além disso, criar uma interface homem-máquina, capaz de interagir melhor com o ser humano, uma vez que absorva o que o cerca de maneira similar.

Para utilização da IA e visão computacional, necessita-se de um agente que possa usufruir dessas técnicas. Para este trabalho foi utilizado um robô móvel, que é um agente autônomo capaz de extrair informações do ambiente e utilizar esse conhecimento do mundo para deslocar-se (Pio *et al.*, 2006).

Desta forma este trabalho apresenta o desenvolvimento do sistema de visão de um robô humanoide para ser aplicado em partidas simuladas de futebol da RoboCup (http://www.robocup.org/robocup-soccer/humanoid/).

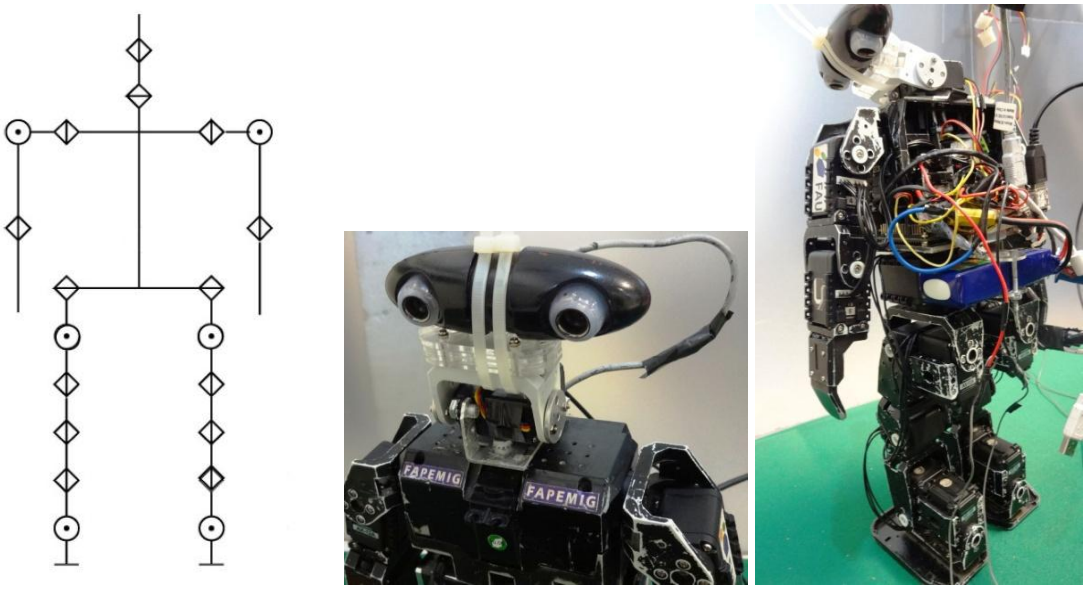
Para atingir este objetivo, primeiramente é apresentado o robô humanoide utilizado como plataforma para o desenvolvimento do sistema de visão. Após é apresentado o sistema de visão abordando a biblioteca OpenCV e as principais funções utilizadas para manipular uma imagem, isolar um intervalo de cores e determinar o centroide do objeto de cor definida. Finalmente é apresentado o fluxograma de funcionamento do robô humanoide e os testes experimentais realizados.

2. DESCRIÇÃO DO ROBÔ HUMANOIDE EDROM

O robô humanoide da Equipe de Desenvolvimento em Robótica Móvel (EDROM) foi construído a partir do kit robótico comercial Bioloid Comprehensive Kit aproveitando-se do seu exoesqueleto e dos servomotores. Esta sendo utilizada uma placa controladora Roboard 110 única para os servomotores, sensores e processamento de imagem. É utilizada também uma câmera comercial com duas lentes, estéreo, para o reconhecimento do meio ambiente.

A Figura 1(a) representa a disposição das articulações dos robôs desenvolvidos sendo: seis graus de liberdade para cada perna, três graus de liberdade para cada braço e dois para cabeça. A Fig. 1(b) representa detalhes da cabeça do robô. Na Figura 2 tem-se o robô desenvolvido.

O robô utiliza servomotores da Dynamixel modelo AX-12A, exceto no pescoço que são utilizados dois servomotores Hitec HS-485HB. A câmera utilizada é da marca Minoru 3D Webcam. Na Tabela 1 são apresentados os dados do robô humanoide desenvolvido pela EDROM.



(a) **(b)**

Figura 1. (a) Esquema das articulações do robô humanoide proposto; (b) Detalhes do robô.

Figura 2. Robô Humanoide EDROM.

Tabela 1. Dados do robô humanoide EDROM

Altura	44 cm
Massa	2.25 Kg
Graus de liberdade	20
Tipos de servomotores	Dynamixel AX-12A e HITEC HS-485HB
Sensor	Phidget Spatial 3/3/3
Camera	Minoru 3D 320x240 640x480
CPU	1.5 GHz Vortex86
Bateria	14.8, Li-Po 2200mH

3. SISTEMA DE VISÃO DO ROBÔ HUMANOIDE

O sistema operacional utilizado é o Lubuntu 10.04 com a biblioteca OpenCV.

O OpenCV é uma biblioteca de visão computacional, disponível em http://SourceForge.net/projects/opencvlibrary. Essa biblioteca é escrita em C e C++, e roda sobre as plataformas Linux (Dalal e Patel, 2013), Windows e MacOS. Foi desenvolvido para grande eficiência computacional, com forte foco em aplicações em tempo real (Brahmbhatt, 2013). O OpenCV foi escrito em C podendo tirar vantagem de processadores com vários núcleos (Bradsky e Kaehler, 2008).

Neste tópico será apresentado o fluxograma utilizado para o sistema de visão do robô desde a captura da imagem do cenário seu tratamento e tomada de decisões para movimentação do robô humanoide.

3.1. Procedimento de captura de imagem

Primeiramente, para compreender como o OpenCV manipula a imagem tirada de uma câmera ou arquivo, precisa-se entender a estrutura do tipo "IplImage". Esta estrutura guardará, não apenas o endereço de memória que está a imagem que está sendo analisada, mas também guardará a largura, altura, o tipo de pixel, a origem do sistema de coordenadas, e várias outras informações da imagem referida.

Portanto, esse tipo, o IplImage, é geralmente um apontador, pois ele aponta um endereço de memória onde encontra-se esta imagem. Então é necessário inicializar uma variável do tipo IplImage, e então carregar uma imagem para ela.

Um exemplo simples de captura de imagem, resultando na visualização de um vídeo de aproximadamente 3 segundos é representado na Fig. 3.

```
1   #include "cv.h"
2   #include "highgui.h"
3
4   int main()
5   {
6      IplImage  *img = 0;
7      CvCapture  *capture = 0;
8      capture = cvCaptureFromCAM(1);
9
10     for(int i = 0;i<90;i++)
11     {
12        img = cvQueryFrame( capture );
13        cvShowImage("Nome da Janela",img);
14     }
15     cvReleaseImage( &img );
16     cvDestroyWindow("Nome  da Janela");
17  }
```

Figura 3. Exemplo de funcionamento do OpenCV.

No código da Fig. 3, as linhas 1 e 2 são para declarar as bibliotecas necessárias para o OpenCV, lembrando que elas devem ser instaladas e configuradas. Estas bibliotecas inserem dentro do seu programa todas as soluções já implementadas pelo OpenCV. Já dentro do main, linha 4, primeiramente, na linha 6, é declarado um ponteiro de "IplImage", para servir de carregador da imagem. Logo depois disto, linha 7, é declarado um ponteiro do tipo "CvCapture", que funciona como uma interface entre a câmera e o programa, então na variável capture será colocado a localização da câmera de onde deve ser pega a imagem. Esta localização é dada por "cvCaptureFromCAM()", na linha 8, e o numero 1 significa que é a câmera de ID 1, e é a primeira câmera localizada pelo sistema operacional na sua inicialização.

Depois dessas declarações, foi utilizado um "for", linha 10, pois como quer-se exibir um vídeo, se este vídeo tivesse apenas um frame, ele seria muito pequeno. Como a taxa da câmera em questão é de 30 frames por segundo (fps), tem-se aproximadamente três segundos de vídeo.

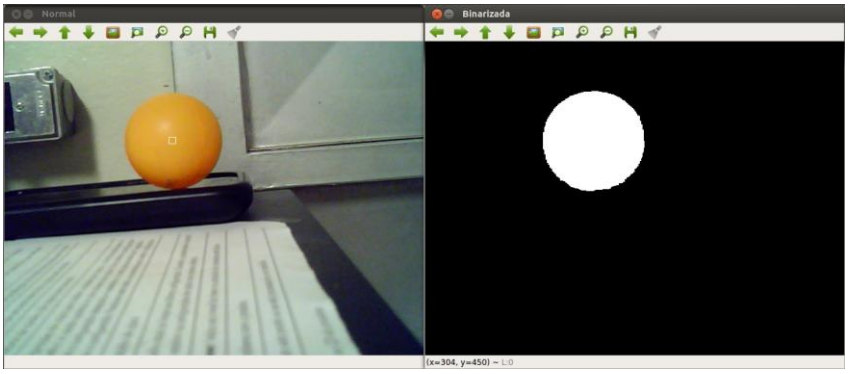
Na linha 12 "img = cvQueryFrame(capture);" é capturada a imagem que está na câmera e colocada dentro da variável 'img". Então, finalmente, na linha 13, cria-se com "cvShowImage", uma janela com o nome "Nome da Janela", e mostra-se nela a imagem que está em "img". E isto se repete por 90 vezes, pelo simples fato de que o programa deseja mostrar aproximadamente 3 segundos, e sabendo que a câmera adquire 30 imagens por segundo, seria o numero lógico. Depois dessas repetições, libera-se a memória onde estão guardadas as informações da imagem, linha 15, e fecha-se a janela, linha 16.

Apesar de ser um exemplo simples pode-se verificar o potencial do OpenCV, pois entre você capturar e mostrar, podem ser aplicados filtros, ou reconhecimento facial. Além disto, em vez de ser mostrada alguma imagem, o sistema pode tomar alguma decisão, de acordo com o que ele "vê". Esta tomada de decisões será explicada no item 4.

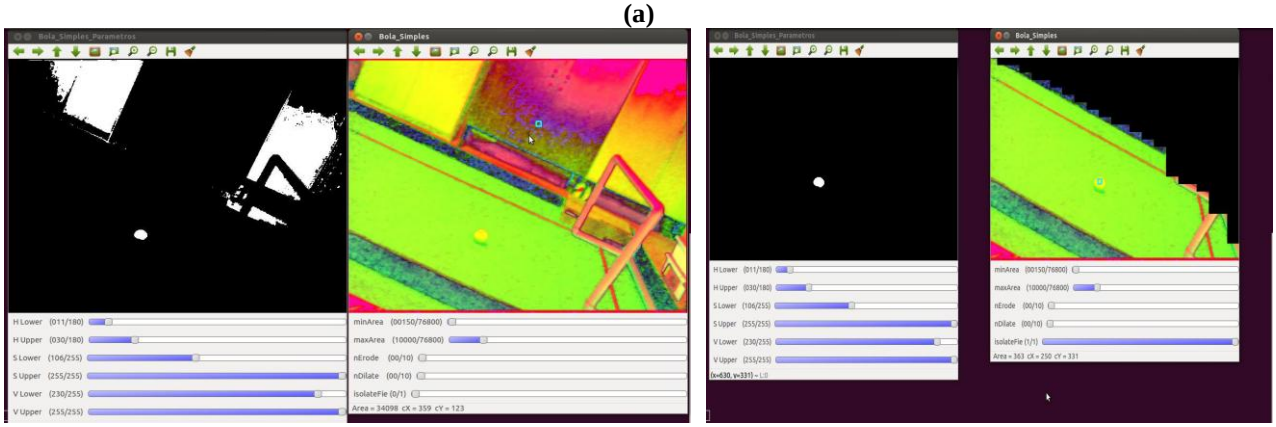
3.2. Isolando um intervalo de cores

Como nas partidas de futebol o cenário é estruturado por cores, neste tópico será descrito a identificação dos objetos do cenário.

O objetivo desta seção será isolar um intervalo de cores, como na Fig. 4(a), para então ser calculado o centroide desta figura, e deixar o sistema tomar uma decisão. A Figura 4 apresenta um teste real do algoritmo aplicado no campo de jogo estruturado, Fig. 4(b), com a identificação das cores e a Fig. 4(c) a bola isolada.



(a)



(b) **(c)**

Figura 4. (a) Cor isolada pelo OpenCV; (b) Campo estruturado identificado e tratado por cores; (c) bola isolada com o seu centroide.

Para realizar esta tarefa, é necessária uma função que funcione como filtro, e esta é representada na Fig. 5.

```
1  IplImage* Filtro(IplImage* img, unsigned int HSV[6])
2  {
3          IplImage* imgHSV = cvCreateImage(cvGetSize(img), 8, 3);
4          cvCvtColor(img, imgHSV, CV_BGR2HSV);
5          IplImage* imgBinaria = cvCreateImage(cvGetSize(img), 8, 1);
6
7          cvInRangeS(imgHSV, cvScalar(HSV[0], HSV[1], HSV[2]), cvScalar(HSV[3],
8  HSV[4],HSV[5]), imgBinaria);
9
10         cvErode( imgBinaria, imgBinaria, NULL, 1);
11         cvDilate( imgBinaria, imgBinaria, NULL, 3);
12         cvReleaseImage(&imgHSV);
13         return imgBinaria;
14 }
```

Figura 5. Exemplo de filtro usado no OpenCV.

Antes de detalhar o código da Fig. 5 será apresentado o formato RGB.

O formato RGB (*Red, Green and Blue*) é uma maneira de apresentar as cores, definindo-as em valores de vermelho, verde e azul. Porem, para o ser humano, pensar assim é complicado, pois nosso olho não consegue separar as cores em

intensidades de vermelho, verde e azul. Por isso o formato HSV (*Hue, Saturation and Value*) é muito utilizado, e define a matiz, saturação e brilho (Bradsky e Kaehler, 2008).

Este formato é mais fácil, pois a variável matiz, representa as cores, e depois de escolhida a cor, a saturação é a intensidade da cor, e o brilho é a quantidade de branco na cor, conforme Fig. 6.

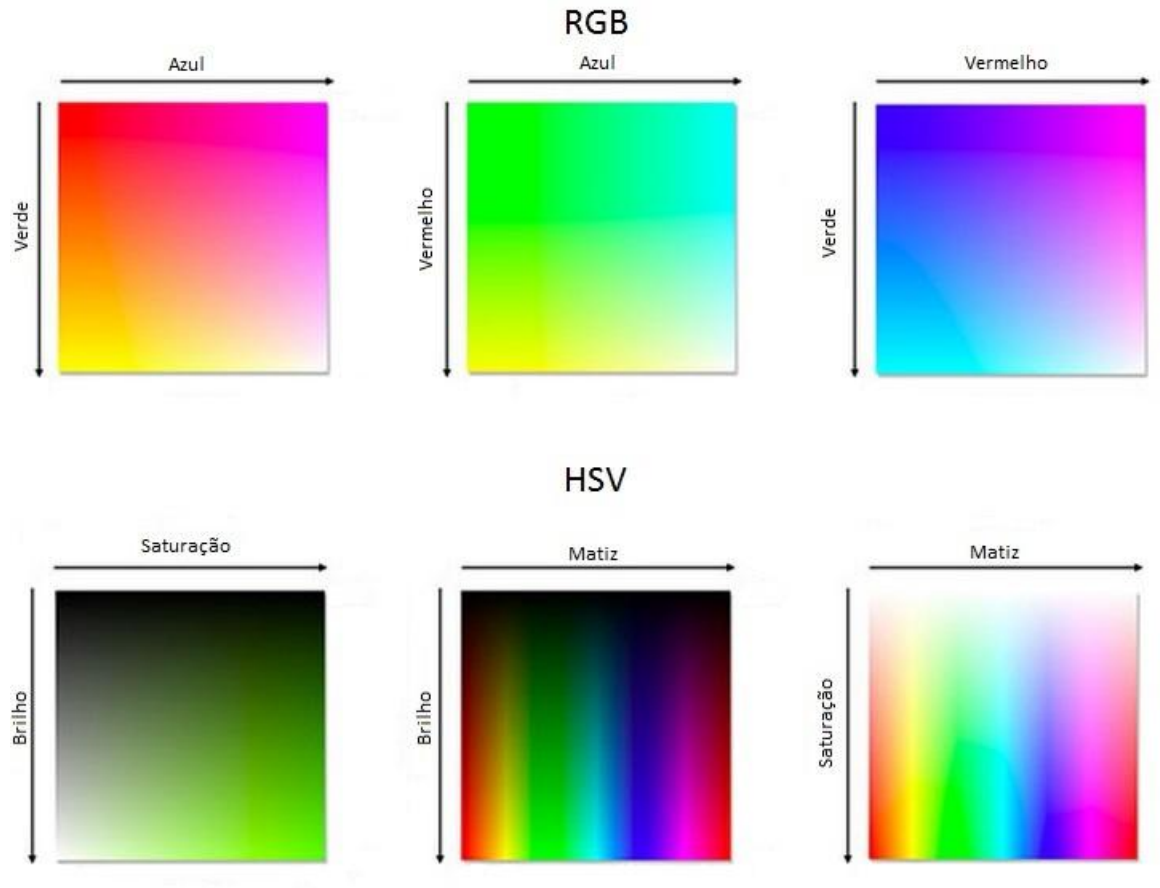


Figura 6. Comparação entre HSV e RGB. Fonte: http://mkweb.bcgsc.ca/color_summarizer/?faq#whatare

A função que faz essa transformação de uma imagem em RBG para HSV é a "cvCvtColor("imgEmRGB", "saidaEmHSV", CV_BGR2HSV);", linha 4, e ela facilitará na definição dos limites da cor desejada. Contudo, antes de ser usada, deve-se criar primeiramente a variável onde ela será armazenada. E como foi visto anteriormente, item 4.2, o tipo IplImage precisa de algumas definições na sua criação.

Por esse motivo usa-se uma função chamada "cvCreateImage()", linhas 3 e 5, neste exemplo ela é usada duas vezes e com pequenas diferenças, na linha 3 "cvCreateImage(cvGetSize(img), 8, 3);" ela é usada para criar a variável que guardará a imagem depois de ser convertida para HSV, então, na sequência, ela precisa ser do mesmo tamanho da imagem original (em RGB), as cores tem 8 bits de profundidade, e existem 3 camadas de cor, que são a matiz, a saturação, e o brilho. Já no segundo momento, linha 5, que ela é utilizada, "cvCreateImage(cvGetSize(img), 8, 1);" a única mudança, é que tem-se apenas uma camada de cor, pois como esta será uma variável que guardará apenas a imagem binarizada, não é necessário mais de uma camada, diminuindo assim, o custo computacional e o espaço de memória utilizado pelo programa.

Uma função fundamental para o funcionamento deste filtro é "cvInRangeS()", linha 7, esta função faz a binarização da imagem, ela recebe valores máximos e mínimos de cada uma das camadas de cor, deixando todos os pixels dentro destes limites brancos, e todos os outros pretos.

Já as funções "cvErode()" e "cvDilate()" , linhas 10 e 11, servem para reduzir o ruído, pois algumas vezes pixels são pego dentro dos limites, mas encontram-se longe do objeto, a função "cvErode()" funciona tirando as bordas dos objetos, porém se o objeto for pequeno, no caso poucos pixels, acaba por eliminar o objeto, que neste caso é um ruído. E por fim, "cvDilate()" faz o efeito contrário, crescendo o objeto, como ele é aplicado depois do "cvErode()" ele apenas cresce o objeto procurado.

Estas duas funções podem ser usadas iterativamente, para melhores resultados, porém elas têm um custo computacional grande, e como neste projeto é utilizado um controlador limitado RB-110, foram usados apenas uma iteração do "cvErode()" e apenas três iterações do "cvDilate()".

Por fim, deve-se liberar o espaço das variáveis que não serão usadas, para isso usa-se a função "cvReleaseImage();", linha 12, e então retorna-se o valor da imagem binarizada para descobrir o centroide de um objeto de cor definida.

3.3. Descobrindo o centroide de um objeto de cor definida

Agora que foi isolado o objeto de análise, no caso a bola de cor laranja, é necessária dar uma tarefa para o robô, ele precisa descobrir onde este objeto está, e dependendo da sua localização tomar uma decisão.

Para isto na Fig. 7, é mostrada a função que tem por objetivo procurar o centroide, da figura de uma cor definida, na imagem captada da câmera, Fig. 4(c).

```
1  void ProcuraBola(int cor, CvCapture *capture) {
2          IplImage *img = 0,*saida = 0;
3          char c;
4          unsigned int HSV[6],width = 0,height = 0;
5          CvMoments *moments = (CvMoments*)malloc(sizeof(CvMoments));
6          //Hmin Smin Vmin Hmax Smax Vmax
7          if(cor == 1) //Cor da bola
8          { HSV[0] = 10;HSV[1] = 150;HSV[2] = 220;HSV[3] = 30;HSV[4] = 255;HSV[5] = 255; }
9
10         else if(cor == 2) //Gol Amarelo
11         { HSV[0] = 30;HSV[1] = 90;HSV[2] = 50;HSV[3] = 65;HSV[4] = 255;HSV[5] = 255; }
12         else if(cor == 3) //Gol Azul
13         { HSV[0] = 151;HSV[1] = 170;HSV[2] = 80;HSV[3] = 166;HSV[4] = 255;HSV[5] = 240;}
14         Else { printf("Cor nao definida"); }
15         img          = cvQueryFrame( capture );
16         saída        = Filtro(img,HSV);
17         cvMoments(saida,moments,1);
18         moment10 = cvGetSpatialMoment(moments,1,0);
19         moment01 = cvGetSpatialMoment(moments,0,1);
20         area         = cvGetCentralMoment(moments,0,0);
21         if((moment10 == 0)&&(moment01 == 0)) {
22                 conta++;
23                 if(conta >= 25)
24                 {
25                         estado = 0;
26                 }
27         }
28         else {
29                 centroidx = moment10/area;
30                 centroidy = moment01/area;
31                 conta = 0;
32         }
33         img = 0;
34         cvReleaseImage(&img);
35 }
```

Figura 7. Exemplo de programa para descobrir o centroide do objeto de cor definida.

Para entender melhor o programa devem ser citadas algumas variáveis globais que são definidas no programa central, pois existe a necessidade de várias funções utilizarem-se desses valores (apesar de apenas esta função atualizá-los).

As variáveis globais utilizadas nesta função são:

- double area, guarda a área do objeto procurado;
- int moment10, é centroidx*area, porém, se 0, mostra que não foi encontrado nenhum objeto;
- int moment01, é centroidy*area, porém, se 0, mostra que não foi encontrado nenhum objeto;
- int centroidx, guarda o valor em x até o centroide;
- int centroidy, guarda o valor em y até o centroide.
- int conta, serve de controle para saber a quantos frames não existe objeto encontrado.
- int estado, controla quando o robô deve começar a procurar o objeto, caso 0, deve começar do início, caso 1, objeto no campo de visão, deve-se continuar com o programa.

No início da função, linhas de 2 a 16, são declaradas algumas variáveis necessárias, e para começar, a função deve recebe a definição da cor a ser procurada e o ponteiro que indica onde está a imagem capturada da câmera. Dependendo do valor da variável "cor" o programa iguala-se a um dos "if", e define assim, os limites da cor que deverá ser procurada.

Para procurar o objeto de cor definida pela variável "HSV[]", o programa utiliza-se da função vista na Fig. 5, como visto em "saída = Filtro(img,HSV);", linha 18.

Para fazer a pesquisa do centroide da figura é utilizada a função "cvMoments();", linha 19, nesta função entra-se com os valores de, respectivamente, imagem a ser analisada, variável onde serão guardados os momentos e o número 1, que diz a função que tem-se uma imagem binarizada para ser tratada.

Esta função gera momentos estatísticos do objeto procurado, do momento de grau zero obtém-se a área do objeto, e dos dois momentos de primeira ordem, consegue-se obter as coordenadas X e Y do objeto se os dividirmos pela área. Porém, os valores de centroides só serão calculados caso as variáveis "moment10" e "moment01" sejam diferentes de zero, linha 23, pois caso elas sejam iguais a zero, significa que nenhum objeto foi encontrado pela função "Filtro()".

Como está sendo captado um vídeo a aproximadamente 30 fps, e a câmera não fica estática, fica em cima de um robô móvel, algumas vezes pode-se ter um falso negativo da procura de objeto, por isso foi adicionada a variável "conta", com ela, espera-se 25 frames (menos de um segundo) antes de definir o objeto como fora de visão, linhas 25 a 28. Isto gerou expressiva melhora no programa em geral.

Ao fim da função libera-se espaço na memória, ao utilizar "cvReleaseImage()" na variável "img", linha 36.

4. PROGRAMAÇÃO GERAL DO CONTROLE DO ROBÔ HUMANOIDE

Cada programa do robô humanoide é rodado em uma *thread*, ou um processo, porém, algumas vezes necessita-se de um programa que ao invés de ser rodado sequencialmente, precisa ser rodado concorrentemente, para isto são utilizadas as *threads*, que utilizam-se da capacidade do sistema operacional de rodar programas concorrentes, e rodam partes diferentes de um mesmo programa em processos separados, como esquematizado na Fig. 8.

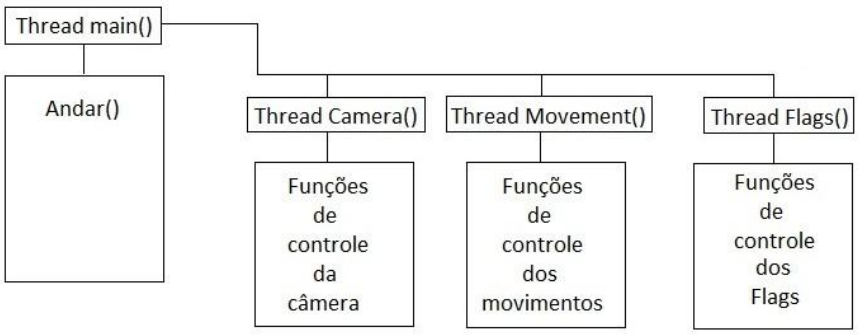


Figura 8. Exemplo do funcionamento das *threads* deste projeto.

As várias *threads* de um programa podem ser executadas concorrentemente (se não forem dependentes). O uso de *threads* traz, para os programadores, a facilidade de escrever aplicações concorrentes, que rodam em máquinas monoprocessadas e multiprocessadas, tendo a vantagem do processador adicional quando este existe (Penha *et al*, 2002).

A *thread* "main()" que roda a função "Andar()" é responsável pela tomada de decisão do robô, ela junta as informações de todas as outras *threads*, e principalmente as informações vindas do processamento de imagem, e as transforma em ações de movimento, ela só não é responsável pelo controle da cabeça, função que é feita pela *thread* "Camera()", essa *thread* também é responsável pela constante aquisição e processamento das imagens, e controla na sua totalidade as funções discutidas neste trabalho.

A *thread* "Movement()" controla para que os movimentos do robô não se sobreponham, portanto, quando a *thread* "main()" diz qual deverá ser o movimento, a *thread* "Movement()" assegura-se que o movimento anterior será finalizado, e que o próximo movimento será inicializado e mantido.

E por fim, a *thread* "Flags()" pega as informações de todos os sensores e os transforma em variáveis de interpretação facilitada.

Para melhor entendimento, a Fig. 9 representa um exemplo de como o programa se comporta caso o robô caia de costas. Primeiramente a *thread* "Flags()" muda a variável "Flag[5]" para 0, esta variável é responsável por dizer se o robô está de pé, se caiu para frente ou se caiu de costas, possibilitando a *thread* "main()" mudar o movimento para o movimento de levantar, depois disto a *thread* "Movement()" finalizar o último movimento, inicia o movimento de levantar, e finaliza-o, e então a *thread* "Flags()" retorna o valor da variável "Flag[5]" para 1.

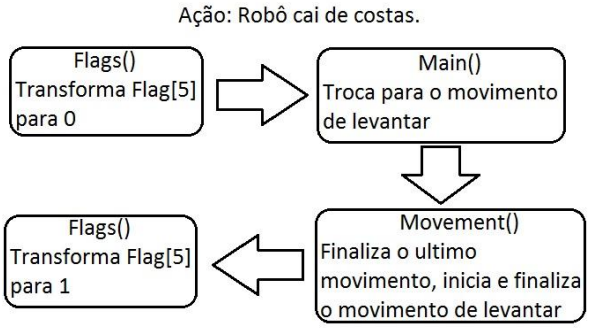


Figura 9. Fluxograma de comportamento do programa caso o robô caia de costas.

Na Figura 10 é representada toda a sequencia necessária para o robô humanoide localizar e chutar a bola. A Figura 10(a) representa o robô de costas para a bola colocada em uma posição aleatória. As Figuras 10(b) e 10(c) representam o robô girando para encontrar a bola. A Figura 10(d) representa o instante em que o robô localizou a bola e as Figs. 10(e-f) o robô andando ao encontro desta. Na Figura 10(g) é representado o robô parado instante antes de realizar o chute e as Figs. 10(h-i) representam a movimentação para o chute.

No link http://ufuedrom.wordpress.com pode-se ter acesso aos vídeos dos robôs humanoides.

5. CONCLUSÕES

Para o desenvolvimento deste trabalho foram pesquisados vários artigos sobre o tema de visão computacional, incluindo pesquisas em andamento, bem como este tema aplicado a biblioteca OpenCV, que foi escolhida para ser a base da visão computacional aplicada ao robô.

Para complementar a pesquisa, foram estudados especificamente, dentro da biblioteca OpenCV, filtros e códigos utilizados para a definição de cores e contornos, bem como seus métodos de funcionamento e suas equações e aplicabilidades no contexto da competição.

Para sustentação da câmera foi desenvolvido uma junta com dois motores, a fim de mover a câmera em X e Y, de fato que, é uma forma mais rápida do que mover o robô inteiro e similar ao comportamento humano (movimentos do pescoço).

Foi desenvolvida uma comunicação entre a câmera e o controlador, sendo utilizado um controle concorrente de seus movimentos e da detecção de objetos pela câmera. Para isto foram utilizadas *threads* na programação em C++.

Para a navegação foram desenvolvidos vários tipos de locomoção, como andar em linha reta, girar, andar de lado e levantar e a orientação desta locomoção foi feita pela câmera. Para isto foram utilizados *threads* para processamento concorrente, tendo sido tomado o devido cuidado para que diferentes *threads* não acessem o mesmo endereço de memória simultaneamente.

6. AGRADECIMENTOS

Os autores agradecem a UFU/FEMEC, FAPEMIG, CAPES e ao CNPq pelo auxílio parcial para execução deste projeto. Especiais agradecimentos a EDROM (Equipe de Desenvolvimento em Robótica Móvel)

7. REFERÊNCIAS

Bradsky, G., Kaehler, A., 2008, “Learning OpenCV. Computer vision with the opencv library”.
Brahmbhatt, S., 2013, “Pratical OpenCV”. Ed. Apress, 244 p.
Dalal, J., Patel, S., 2013, “Instant OpenCV Starter”, Ed. Packt Publishing, 56 p.
Luger, G. F., 2004, “Inteligência artificial: estruturas e estratégias para a solução de problemas complexos”.
Pio, J. L. de S., Castro, T. H. C., Júnior, A. N. de C., 2006, “A robótica móvel como instrumento de apoio à aprendizagem de computação”. In: Anais do Simpósio Brasileiro de Informática na Educação, p. 497-506.
Penha, D. O., Corrêa, J. B. T.; Martins, C. A. P. S., 2002, “Análise Comparativa do Uso de Multi-Thread e OpenMp Aplicados a Operações de Convolução de Imagem”. Pontifícia Universidade Católica de Minas Gerais, BH.

8. RESPONSABILIDADE AUTORAL

“Os autores são os únicos responsáveis pelo conteúdo deste trabalho”.

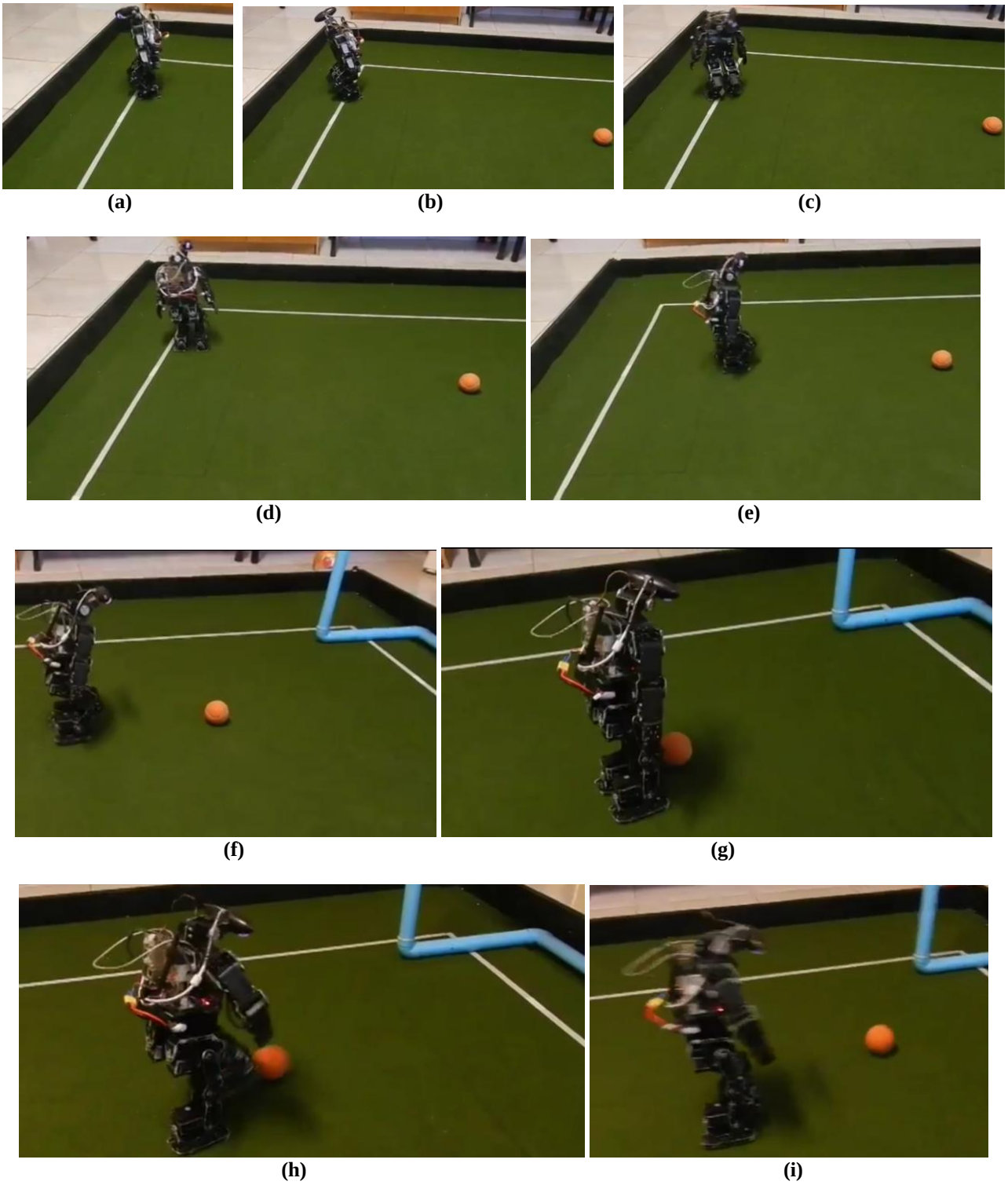


Figura 10. Teste prático com o robô humanoide para localizar e chutar a bola.

DEVELOPMENT AND IMPLEMENTATION OF A HUMANOID ROBOT'S VISION SYSTEM

Thomas Vergutz, tvergutz@gmail.com[1]
Rogério Sales Gonçalves, rsgoncalves@mecanica.ufu.br[1]

[1]Laboratório de Automação e Robótica, Faculdade de Engenharia Mecânica, Universidade Federal de Uberlândia, Campus Santa Mônica – Bloco 1M – Av. João Naves de Ávila, 2121, Uberlândia – MG – CEP 38408-902

Abstract*. The use of mobile intelligent and interactive systems grows daily, and in this work we show how the human-machine interface is substantially improved with the use of a robotic vision system, in this paper we show a solution found to the interaction of humanoid robot with the environment, using cameras and digital image processing. The goal is to use the OpenCV library for image processing, and as purpose to embed this computer vision on a mobile robotic system that will be able to play soccer without human control.*

Keywords*: Computer vision, robotics, humanoid robots, OpenCV.*