VISÃO COMPUTACIONAL APLICADA NO RECONHECIMENTO DE OBJETOS CIRCULARES E VERMELHOS, COM RETORNO DE POSIÇÃO E RAIO

AMARO THEODORO DAMASCENO NETO, FERNANDO HENRIQUE CHAVES MACEDO, RAPHAEL D. C. SILVA

Instituto de Estudos Superiores da Amazônia
Av. Governador José Malcher 1128, 66055-260 Nazaré Belém, PA, BRASIL
E-mails: amarotheodoro@gmail.com, fhc128@hotmail.com

Abstract - *This article presents techniques from computer vision and high level programming, applied to location of an object with circular shape and reddish. There will be a demonstration of the steps needed since the pre-processing to final recognition of position and radius of the object.The techniques been applied: INRANG, Smooth, Hough Circles. The result with the use of filters and detection technique was positive, because it was possible to locate exactly where (position, relative to the X and Y on the screen) and distance (radius) the object was.*

Keywords - *Computer vision, programming, OpenCV. INRANG, Smooth, Hough Circles*

Resumo - Este artigo apresenta técnicas de visão computacional e programação de alto nível, aplicadas na localização de um objeto de formato circular e vermelho. Demonstraremos as etapas necessárias desde o pré-processamento até o reconhecimento final de posição e raio do objeto. As técnicas aplicadas foram : InRange, Smooth, Hough Circles. O resultado com o uso desses filtros e com a técnica de detecção foram positivos, pois conseguiu-se localizar com exatidão a posição (em relação ao plano X e Y na tela) e distancia (raio) da localização do objeto.

Palavras-chave— Visão computacional, Programação, Opencv. InRange, Smooth, Hough Circles

1 Introdução

Este projeto abrangerá as etapas necessárias para reconhecimento de objetos circulares e vermelhos a partir de um sinal gerado de uma câmera digital em tempo real. Essas técnicas fazem parte da tecnologia aplicada de visão computacional que aliada s a programação podem criar aplicativos capazes de auxiliar na navegação visual com possibilidades de desenvolvimento aplicado a diversas áreas como: auxiliadores de inspeções para obter-se controle de qualidade de processos, cálculo de posição, orientação de detalhes para um braço robótico e outras.

Um exemplo de aplicação que vem sendo bastante desenvolvida, é a navegação visual, que consiste no estabelecimento de um rumo ou rota com base em pontos de referência, (Martins, 2010), como pode-se constatar na definição de rota de um veículo ilustrada na Figura 1.

Figura 1. Exemplo de Navegação Visual, Referência: Martins, 2010.

Isso se deve, principalmente, à utilização de algoritmos de processamento de imagens, aliados a captura de informações advinda de câmeras para estimar posição, orientação, etc.

Motivado pelo grade crescimento do interesse em relação a essas aplicabilidade, este artigo demostrar as técnicas de visão computacional necessárias para que se identifique objetos circulares e vermelhos em uma tela digital, retornando sua posição e raio.

2 Estrutura do Aplicativo

O aplicativo desenvolvido nesse trabalho utilizou a linguagem de programação *Python* para estruturar as etapas de pré-processamento e processamento da imagem capturadas.

O Pré-processamento, constitui-se da troca de cor das imagens do formato RBG para HSV facilitando por dois filtros digitais: o primeiro denominado, *In Range,* realça a cor vermelha e o segundo, denominado *Smooth***,** suaviza e destaca o contorno dos objetos realçados.

Após as etapas de filtragem, pôde-se aplicar a localização dos objetos circulares e vermelhos no enquadramento do vídeo. Essa localização é determinada pela técnica *Hough Circles,* que informa o deslocamento horizontal e vertical, dos objetos, além da profundidade.

A forma de trabalho com a captura de cada frame da imagem pelo programa desenvolvido pode ser observado de forma simplificada no fluxograma da Figura 2.

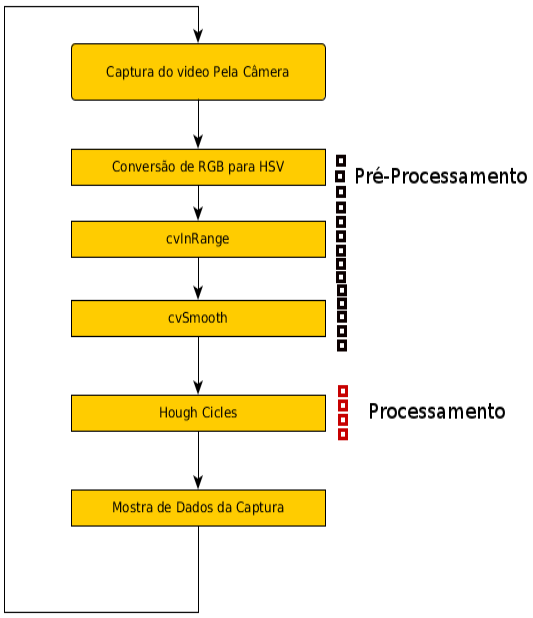

Figura 2. Fluxograma das etapas de processamento

3 Materiais utilizados

Os componentes utilizados para a construção desse trabalho constituem de:

Notebook

Câmera Digital com capacidade de gerar gravações na resolução *Full HD*, a 60 *frames* por segundo.

Objetos circulares de cores distintas (duas bolas de plastico nas cores vermelha e amarela)

Equipamentos fotográficos para auxiliar na qualidade das amostras captadas (tripé e refletores)

4 Programação

Python é uma linguagem de programação de alto nível, orientada a objetos, de tipagem dinâmica e forte. Foi lançada por Guido Van Rossum em 1991. Atualmente possui um modelo de desenvolvimento comunitário e aberto.

Projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade ou expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca e por módulos e *frameworks* desenvolvidos por terceiros. (Kamal, 2008).

Com base nessas características da linguagem e da necessidade de um código dinâmico, buscamos seus recursos capazes de trabalhar com a biblioteca. *Opencv (python-opencv).* (OpenCV 2.1 Python Reference, 2011)

O código completo do projeto comentado encontra-se no anexo do artigo.

5 OpenCV

A biblioteca *OpenCV* foi desenvolvida pela Intel e possui mais de 500 funções. Foi idealizada com o objetivo de tornar a visão computacional acessível a usuários e programadores em áreas relacionadas à interação humano computador em tempo real e à robótica. A biblioteca está disponível com o código fonte. (Bradski, 2008).

A biblioteca está dividida em cinco grupos de funções:

- Processamento de imagens;
- Análise estrutural;
- Análise de movimento e rastreamento de objetos;
- Reconhecimento de padrões
- Calibração de câmera e reconstrução 3D.

As principais funções usadas no desenvolvimento do software são apresentadas a seguir, juntamente com os conceitos de processamento de imagens e visão computacional que devem ser empregados em seu uso.

6 Processamento de Imagem

O processamento de imagens é definido como qualquer forma de processamento de dados no qual a entrada e saída são imagens tais como fotografias ou quadros de vídeo. (Binford,1991)

Ao contrário do tratamento de imagens, que se preocupa somente na manipulação de figuras para sua representação final, o processamento está voltado para reconhecimento de padrões. A maioria das técnicas envolve o tratamento de imagem como sinal bidirecional, no qual são aplicados padrões de processamento de sinal.

Uma suposta imagem digital é formada por elementos básicos chamados de pixel, cada pixel possui duas propriedades básicas: a primeira refere-se a sua localização espacial (coordenadas x e y) no plano definido pela imagem e a segunda é a intensidade. Formalmente, uma imagem digital é uma função bidimensional (x,y), na qual x e y referem-se às coordenadas espaciais do pixel. (Binford, 1991)

Baseado nesses conceitos será apresentada a metodologia utilizando as técnicas de tratamento de imagem para reconhecimento de objetos mostrando sua posição e raio em tempo real, baseando-se simplesmente nas coordenada x e y no plano da imagem gerada pela câmera.

6.1 Padronização do Sinal

Antes de utilizar o sinal nos filtros de tratamento de imagem, deve-se converter o padrão de cor do sinal para o formato adequado à biblioteca do *OpenC*V para facilitar o processamento e destacando os objetos de interesse. O vídeo é gerado e armazenado em disco no padrão RGB, sistema de cores aditivas formado por Vermelho (*Red*), Verde (*Green*) e Azul (*Blue*), posteriormente ele é convertidopara o formato das componentes Matriz (*hue*), saturação (*saturation*) e valor (*value*), através do comando: *cvCvtColor(frame, self.hsv_frame, CV_BGR2HSV)* (Brys, 2011) .
Segundo Alvy Ray Smith, esse sistema de cores define o espaço de cor (Brys, 2011), utilizando seus três parâmetros, conforme descrito abaixo:

- Matiz (tonalidade): Verifica o tipo de cor, abrangendo todas as cores do espectro, desde o vermelho até o violeta, mais o magenta. Atinge valores de 0 a 360, mas para algumas aplicações, esse valor é normalizado de 0 a 100%.
- Saturação: Também chamado de "pureza". Quanto menor esse valor, mais com tom de cinza aparecerá a imagem. Quanto maior o valor, mais "pura" é a imagem. Atinge valores de 0 a 100%.
- Valor (brilho): Define o brilho da cor. Atinge valores de 0 a 100%.

A base matemática para essa padronização pode ser vista nas fórmulas de 1 a 6.

$$H = (60\,x\frac{G-B}{MAX-MIN})+0 \quad (1),$$
$$if\ MAX = R\,e\,G \geq B$$

$$H = \left\{60\,x\frac{G-B}{MAX-MIN})+360\right\} \quad (2),$$
$$if\ MAX = R\,e\,G < B$$

$$H = \left\{60\,x\frac{G-B}{MAX-MIN})+120\right\} \quad (3),$$
$$if\ MAX = G$$

$$H = \left\{60\,x\frac{G-B}{MAX-MIN})+240\right\} \quad (4),$$
$$if\ MAX = B$$

$$S = \frac{MAX-MIN}{MAX} \quad (5),$$

$$V = MAX \quad (6)$$

Seja uma cor definida por (R, G, B), onde R, G e B estão entre 0 e 1, que são, respectivamente, o maior e o menor valor possível para cada. A transformação para os parâmetros (H, S, V) dessa cor, pode ser determinada pelas fórmulas de um a

seis. Seja MAX e MIN os valores máximo e mínimo, respectivamente, dos valores (R, G, B).
A tonalidade (H) varia de 0 a 360 indicando o ângulo no circulo onde ela esta definida; a saturação S e o brilho V variam de 0 a 1. representando o menor e o maior valor possível.

Figura 3. Foto com cores padrão RGB, antes da conversão.

A Figura 3, é um *frame* do vídeo original, e a partir dele foi gerada a Figura 5 com os padrões em HSV.

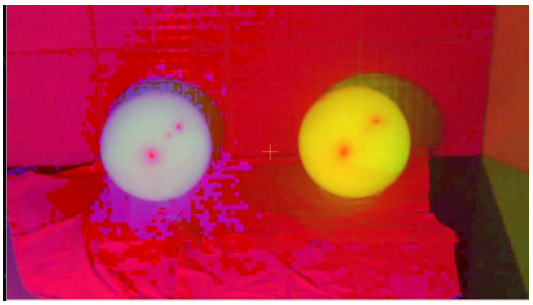

Figura 4. Foto com cores padrão HSV, após a conversão.

Observando as imagens percebemos que os valores de azul, verde e vermelho, figura 3, são substituídos pelos valores de tonalidade, saturação e brilho, figura 4.
Após a conversão a entrada do sinal está pronta para o tratamento de imagem que veremos nos próximos tópicos.

6.2 Técnica cvInRange

Esta função funciona como um filtro passa faixa, no caso somente a cor vermelho passará pelo filtro.
Para isso é utilizado o comando: " **void cvInRange(const CvArr* src, const CvArr* lower, const CvArr* upper, CvArr* dst) ; ".** (Bradski, Gary, Kaehler, Adrian 2008)
Cada componente do comando está descrito abaixo:
Src - Matriz de cores primeira fonte.
Lower - Matriz de fronteira, inclusive menores.
Upper - Matriz limite exclusivo superior.
Dst - Matriz de destino.

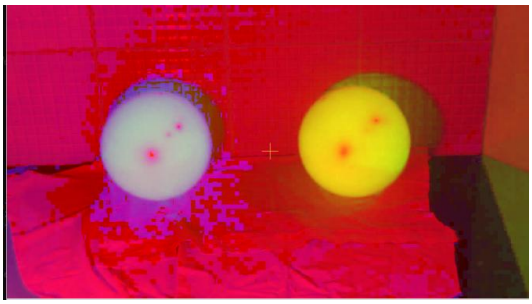

Figura 5. Imagem de Entrada, ates de usar o filtro InRange.

Figura 6. Figura original, após se usar o filtro InRange

Como o resultado do filtro de cor, o programa filtra somente a coloração vermelha na imagem, ou seja, identifica com a cor preta as partes da imagem tratada que predominam a cor vermelha na imagem original, Figura 6.

6.3 Técnica cvSmooth

Na biblioteca *OpenCV*, existe uma função relativa a filtragem de suavização chamada cvSmooth, e como a operação de suavização nada mais é que um processo de diminuição de ruído, em programação é efetuado através de uma única função (Bradski, Gary, Kaehler, Adrian 2008). Nessa biblioteca é possível utilizar as técnicas listadas na Figura 7.

Tipo de suavização	Descrição
CV_BLUR_NO_SCALE	Somatório em uma vizinhança de pixels param1×param2
CV_BLUR	Somatório em uma vizinhança de pixels param1×param2 com subseqüente escala por 1/(param1×param2)
CV_GAUSSIAN	Convolução da imagem com um kernel Gaussiano param1×param2
CV_MEDIAN	Mediana de uma vizinhança de pixels param1×param1 (ou seja, a vizinhança é quadrada)
CV_BILATERAL	Aplicação de um filtro 3x3 bilateral com cor sigma=param1 e espaço sigma=param2, como descrito em [2]

Figura 7. Sub comandos para suavização

Porém, será utilizado somente a suavização do tipo Gaussian por possuir a melhor resposta de destaque do contorno para os vídeos testados do projeto, essa função tem o objetivo de fazer a suavização e identificação das bordas da imagem facilitando o processamento da imagem e é executada pelo seguinte comando:

void cvSmooth(const CvArr* src, CvArr* dst, int smoothtype=CV_GAUSSIAN, int param1=3, int param2=0, double param3=0, double param4=0); (Bradski, Gary, Kaehler, Adrian 2008)

- SRC é a imagem de origem
- DST é a imagem de destino
- SMOOTHYPE é o tipo da suavização que pode ser aplicada

Quando o *smoothtype Gaussiano* é usado, param3 indica o sigma da *Gaussiana* (desvio padrão). E se ele for zero, ele é calculado a partir do tamanho do *kernel*, usando a Fórmula (7):

$$sigma = (\frac{n}{21}) x\, 0.3\, x\, 0.8 \qquad (7),$$

onde sigma é param1 ou param2, dependendo da orientação do *kernel* (horizontal ou vertical).

Com resultado a Figura 8:

Figura 8. Imagem original e Imagem após filtro de suavização.

6.4 Técnica Hough Circles

Um método de detecção de círculos é a aplicação da transformada de *Hough Circles*. Através do uso da equação da circunferência procura-se em toda a imagem contornos cujos pontos que o definem pertencem à zona interna definida pela equação, consegue obter-se uma função de acumulação de pontos que mostra quais os centros e raios dos círculos.

Sendo esses pontos um conjunto de coordenadas (x; y) pretende-se encontrar valores possíveis para os parâmetros (xc; yc), que correspondam aos pontos centrais da circunferência. Então constrói-se um espaço de Hough, que é uma matriz, com a mesma dimensão da imagem digital, em que as colunas e linhas representam, respectivamente, os possíveis valores de xc e yc. Cada célula dessa matriz ganha, em estado inicial, o valor 0, e para cada ponto (x; y) da imagem, incrementa-se no espaço de Hough, todas as células (xc; yc) representando centros da circunferência, de raio r, que passam por (x; y). Finalmente, as células contendo os valores mais

altos indicarão os centros "mais prováveis" de circunferências.

Sendo assim, para calcular todos os valores de (xc;yc) para um determinado ponto (x;y), geralmente não se utiliza a equações (1), pois sua parametrização em função de xc e yc não produz, diretamente, implementações eficientes.

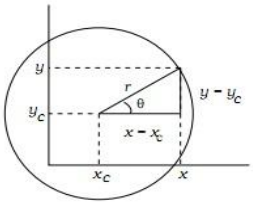

Figura 9. Circunferência de raio r e centro xc; yc,
Fonte: Boer, Sachse, Dössel 2011

Sendo assim, para calcular todos os valores de (xc;yc) para um determinado ponto (x;y), geralmente não se utiliza a equações (1), pois sua parametrização em função de xc e yc não produz, diretamente, implementações eficientes.

$$r^2 = (x - xc)^2 + (y - yc)^2 \quad (8),$$

A formula mais usada baseia-se na representação em coordenadas polares. Percebe-se que, através da Figura 9, e utilizando conceitos elementares de trigonometria, que as duas equações seguintes são validas:

$$xc = x - r * \cos(\theta) \quad (9),$$

$$yc = y - r * \sin(\theta) \quad (10),$$

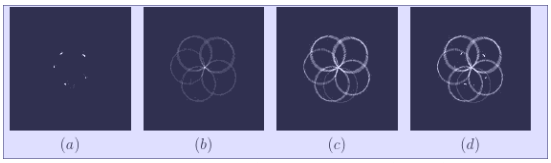

Figura 10. Passos realizados por Hough Circles

(a) Resposta inicial da imagem de entrada.
(b)Seu respectivo espaço de Hough.
(c)Espaço de Hough com ajuste de contraste para facilitar a Visualização.
(d) Imagem composta pela adição da imagem original com o espaço de Hough.

cvHoughCircles(CvArr* image, void* circle_storage,int method, double dp, double min_dist, double param1, double param2, int min_radius, int max_radius). (Bradski, Gary, Kaehler, Adrian 2008)

- **IMAGE** - A entrada de 8 bits de canal único em grayscale.
- **circle_storage** – Armazena os círculos detectados. Pode ser um armazenamento de memória (neste caso, uma seqüência de círculos é criado no armazenamento e retornado pela função) ou de uma única linha / matriz única coluna (CvMat *) do tipo CV_32FC3, para que parâmetros os círculos são por escrito. O cabeçalho da matriz é modificado pela função que a sua colunas ou linhas irá conter um número de linhas detectadas.Se circle_storage é uma matriz e se número real de linhas exceder o tamanho da matriz, o maior número possível de círculos é retornado. Cada círculo é codificado como três números de ponto flutuante: Centro de coordenadas (x, y) e o raio.(Bradski, 2008)
- **method -** Atualmente, o único método implementado é o CV_HOUGH_GRADIENT.
- **Dp -** Resolução do acumulador usado para detectar os centros dos círculos. Por exemplo, se for 1, o acumulador vai ter a mesma resolução que a imagem de entrada, se for 2 - acumulador terá largura de duas vezes menor e altura, etc.
- **min_dist** - Distância mínima entre os centros dos círculos detectados. Se o parâmetro é muito pequeno, os círculos vizinho múltiplos podem ser falsamente detectada, além de um único e verdadeiro. Se ele for muito grande, alguns círculos podem ser perdidas.
- **Param1 –** O primeiro método específico de parâmetros. Em caso de CV_HOUGH_GRADIENT é o limite máximo mais elevado das duas passadas para detector de bordas Canny (a mais baixa será duas vezes menor).
- **Param2 -** O método específico do segundo parâmetro. Em caso de CV_HOUGH_GRADIENT é acumulador limiar na fase de detecção de centro. Se for um valor pequeno por menor que seja, os círculos mais falso podem ser detectado. Círculos, correspondentes aos valores maiores acumulador, será devolvido primeiro
- **min_radius -** Raio mínimo do círculos para pesquisa.
- **max_radius -** Raio máximo do círculos para pesquisa.

Com esses implementos os resultados podem ser vistos na Figura 11.

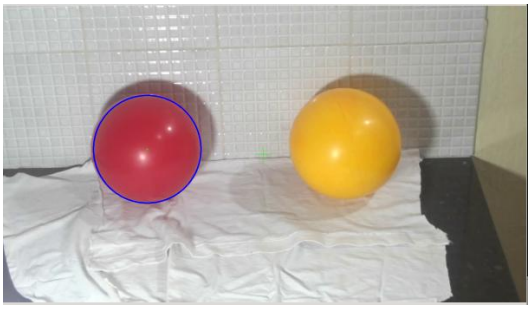

Figura 11. Programa em funcionamento, localizando somente a esfera de cor vermelha, e ignorando completamente a outra esfera de cor amarela.

6.5 Limitações

Os processos de visão computacional, muitas vezes, necessitam de uma etapa de pré-processamento envolvendo o processamento de imagens. As imagens de onde queremos extrair alguma informação em alguns casos precisam ser convertidas para um determinado formato ou tamanho e precisam ainda ser filtradas para remover ruídos provenientes do processo de aquisição da imagem.

Os ruídos podem aparecer de diversas fontes, como por exemplo, o tipo de sensor utilizado, a iluminação do ambiente, as condições climáticas no momento da aquisição da imagem, a posição relativa entre o objeto de interesse e a câmera. Note que ruído não é apenas interferência no sinal de captura da imagem, mas também interferências que possam atrapalhar a interpretação ou o reconhecimento de objetos na imagem.

Os filtros são as ferramentas básicas para remover ruídos de imagens, neste caso, o ruído é aquele que aparece no processo de aquisição da imagem. A Figura abaixo apresenta um exemplo de uma imagem com ruído (à esquerda) e da imagem filtrada (à direita).

Figura 12. Imagem com ruído (à esquerda) e da imagem filtrada (à direita).
Fonte: Maregani, Stringhin 2009.

A maior desvantagem da aplicação da *Hough Circles* é a necessidade de percorrer toda a imagem e efetuar uma acumulação de "possíveis" centros de círculos. Isto para uma gama de valores de raio que deverá ser definida de modo mais estreita possível para que haja pouco consumo de tempo de processamento. (Oliveira, 2007)

7 Resultados

Todos os dados de localização da circunferência vermelha foram realizados com vídeos de com 2 minutos e 30 segundos de duração e foram acrescentados outras esferas de cores diferentes.

Nos vídeos, a câmera com o tripe foi movida (simulando o movimento das esferas nos eixos x e y) e ajustado o zoom (simulando a situação o mudando de raio na imagem).

E os dados do processamento de imagem foram salvos em formato txt e gerado gráficos no Matlab para análise, a baixo temos a imagem dos dados sendo gerados pelo programa, Figura 13.

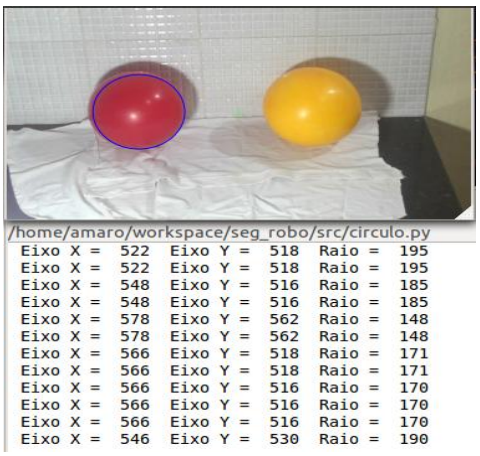

Figura 13: Resultado da Captura sendo gerados na imagem.

O programa obteve ótimas respostas muito rápidas nos vídeos, não houveram problemas na identificação da esfera correta durante todos os teste de captura, as maiores dificuldades foram encontradas na identificação do raio da circunferência onde ocorreram ruídos decorrentes da variação da iluminação. Os dados do raio variavam bastante, pois a variação da luz provoca diferença na tonalidade da cor, gerando erros da identificação correta do raio e modificando o centro da esfera interferindo na sua posição no eixo x e y.

Em seguida colocaremos amostras dos dados gerados em gráficos dos testes realizados, sendo o a coordenada x o tempo em segundos e a coordenada y a variação de posição ou do raio.

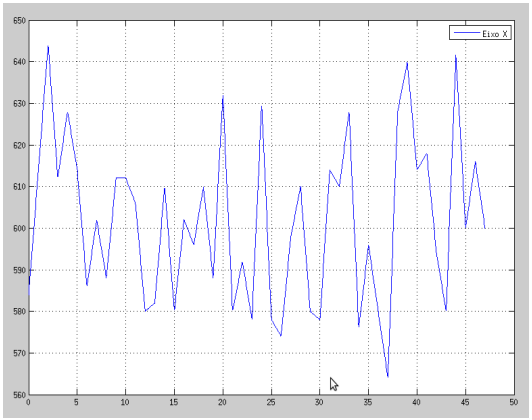

Figura 14: variação no Eixo X no tempo em segundos

- No tempo de 0 a 5 segundo temos um deslocamento para direita e retorno para posição inicial.
- No tempo de 33 a 40 segundo temos um deslocamento para esquerda e retorno para posição inicial.

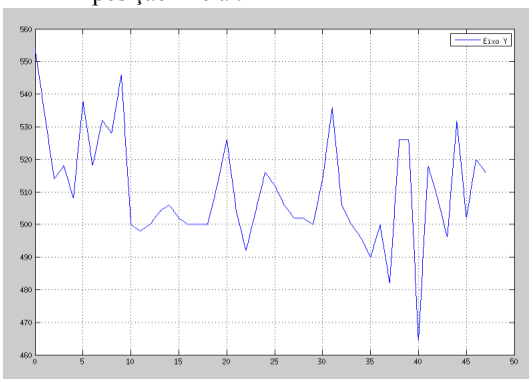

Figura 15. variação no Eixo Y no tempo em segundos

- No tempo de 0 a 5 segundo temos um deslocamento para baixo e retorno para posição inicial.
- No tempo de 30 a 36 segundo temos um deslocamento para cima e para baixo.

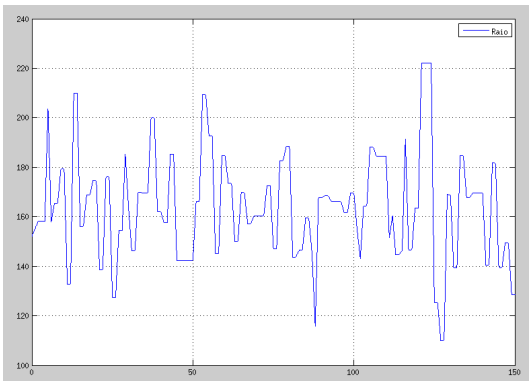

Figura 16: variação do Raio no tempo em segundos

- Gráficos com movimentos de ida para frente e para trás.
- Muito ruido sendo gerado pela iluminação.
- Possui ruídos gerados pelo raio. Figura 16.

Todos os gráfico acima possuem ruídos da variação do raio provocando sinais de subida e decida, sendo o gráfico do Raio o mais afetado.

8 Conclusão

A partir dos resultados alcançados, pode-se destacar a importância do desenvolvimento de projetos utilizando visão computacional, pois com essa tecnologia podemos criar sistemas que simulam a capacidade de enxergar, como nesse artigo focamos na detecção de objetos, pode-se concluir que essa tecnologia pode expandir em vários segmentos, dentre os quais pode-se citar como exemplo a navegação de carros autônomos sem o auxilio humano, como citado na referencias.
Para dar continuidade ao projeto, planeja-se um aumento de robustez no algoritmo de processamento de imagem, bem como um desenvolvimento mais profundo do sistema de navegação virtual.

Agradecimentos

Agradecimentos especiais aos nossos Orientador, Raphael Diego Comesanha e Silva que ajudou-nos a cumprir os objetivos desse artigo e aos professores Max Ricardo Pantoja Trindade e Roger Ribeiro da Silva pelo *feedback* no desenvolvimento deste artigo.

Referências Bibliográficas

Binford, T. O., Jain, R. C. (1991). Computer Vision Graphics and Image Processing, vol.53 p.112-117 .

Boer, I. H. de, Sachse, F. B., and Dössel, S. Mang O. *Methods for Determination of Electrode Positions in Tomographic Images*. Disponível em: <http://ijbem.k.hosei.ac.jp/volume2/number2/deboer/paper_ijbem.htm. Acesso em: 20 out. 2011.

Bradski, Gary, Kaehler, Adrian (2008). Learning OpenCV – Computer Vision with the OpenCV library, O'Reilly.

Brys, Leandro Monteiro. (2011). *Formação das cores*. Disponível em: http://shdo.com.br/softwares/baixar/mrgb. Acesso em: 30 out. 2011.

Brys, Leandro Monteiro. *Transformação RGB para HSV*. Disponível em: http://www6.ufrgs.br/engcart/PDASR/formulario1.html. Acesso em: 30 out. 2011.

Kamal, Ibrahim (2008). *Line Tracking Sensors and Algorithm* . Disponível em: http://ikalogic.com/tut_line_sens_algo.php. Acesso em: 10 agosto 2011.

Ling Chen Chuan Xuan. *Open CV*. Disponível em: http://www.cmlab.csie.ntu.edu.tw/~jsyeh/wiki/doku.php?id=%E9%99%B3%E8%BB%92%E7%BF%8E:opencv. Acesso em: 21 out. 2011.

Martins, Carlos (2010). Augmented Driving no Ipohone. Disponível em: http://abertoatedemadrugada.com/2010/04/augmented-driving-no-iphone.html. Acesso em: 20 dez. 2011.

Maregani, Maurício, stringhin, Denise (2009).Introdução à visão Computacional Usando OpenCv .Disponivel em: seer.ufrgs.br/rita/article/download/rita_v16_n1_p125/7289.acesso em 21 out. 2011

Oliveira, André Joaquim Barbosa de. (2007). Desenvolvimento de software para aplicação no controle e monitorização de plataforma móvel de recolha de bolas de golfe. Disponível em: http://intranet.dei.uminho.pt/gdmi/galeria/temas/pdf/38026.pdf . Acesso em: 21 out. 2011.

OPENCV 2.1 Python Reference. *Open CV V2. Documentation*. Disponível em: http://opencv.willowgarage.com/documentation/python/index.html Acesso em: 15 outubro 2011.

Anexo - Programa desenvolvido para processamento de imagem

```
# -*- coding: iso-8859-1 -*-
Created on 07/03/2011
@author: amaro e fernando
import sys
from opencv.cv import *
from opencv.highgui import *
from com_serial import pyserial
class capt_circulo(object):
        size = cvSize(1920, 1080) # resolução do vídeo
        hsv_frame = cvCreateImage(size, 8, 3)
        thresholded = cvCreateImage(size, IPL_DEPTH_8U, 1)
        thresholded2 = cvCreateImage(size, IPL_DEPTH_8U, 1)
        #escala da cor vermelha para o filtro
        hsv_min = cvScalar(0, 50, 170, 0)
        hsv_max = cvScalar(10, 180, 256, 0)
        hsv_min2 = cvScalar(170, 50, 170, 0)
        hsv_max2 = cvScalar(256, 180, 256, 0)
        storage = cvCreateMemStorage(0) # Storage (é uma pilha), uma estrutura de baixo nível utilizada para
armazenar
 dnamica da estruturas de dados em sequencias do reconhecimento
        #Cria arquivos para salvar os dados de captura do vídeo
        amostraDados_X = open('tY.txt', 'w')
        amostraDados_Raio = open('tRaio.txt', 'w')
        def start(self):
                cvNamedWindow ('Camera', 0) #cria janela para imagem capturada e processada
                cvNamedWindow ('Apos o Filtro de Cor', 0) #cria janela com a imagem dos filtros
        aplicados
                capture = cvCreateFileCapture("1.avi")
 #Cria captura de uma arquivo de vídeo
                #sai do programa se não encontrar entrada de dados
                if not capture:
                        print "Could not open webcam"
                sys.exit(1)
                while 1:
                        frame = cvQueryFrame(capture) #captura os frames do vídeo
                        #sai do programa se acabar os dados de entrada
                        if not frame:
```

```
                self.amostraDados_X.close()
                self.amostraDados_Raio.close()
                        print 'Fechado'
                        break
                #marca alvo no meio da tela para indicar o cento
                cvLine(frame, cvPoint(frame.width/2,(frame.height/2)-30),
cvPoint(frame.width/2,(frame.height/2)+30),
                CV_RGB(0,255,0), 1,CV_AA, 0 );
                cvLine(frame, cvPoint((frame.width/2)-30,frame.height/2),
cvPoint((frame.width/2)+30,frame.height/2),
                CV_RGB(0,255,0), 1,CV_AA, 0 );
                cvCvtColor(frame, self.hsv_frame, CV_BGR2HSV) # Converter uma imagem de um
espaço de cores HSV:
                #passa duas vezes pelo filtro de cor
        cvInRangeS(self.hsv_frame, self.hsv_min, self.hsv_max, self.thresholded)
        cvInRangeS(self.hsv_frame, self.hsv_min2, self.hsv_max2, self.thresholded2)
                cvOr(self.thresholded, self.thresholded2, self.thresholded) #calcula junÃ§Ã£o entre
duas matrizes
        cvSmooth(self.thresholded, self.thresholded, CV_GAUSSIAN,9, 9) #filtro de suavização e detecção de
bordas
                circulo = cvHoughCircles(self.thresholded, self.storage, CV_HOUGH_GRADIENT, 2,
self.thresholded.height/4, 100, 90, 70, 260) #identificação de círculos
                #variÃ¡veis de captura
                self.maxRadius = 0
                self.x = 0
                self.y = 0
                found = False
                #desenhando contorno no circulo encontrado na imagem
                for i in range(circulo.total):
                        circle = circulo[i]
                        if circle[2] > 70:
                        found = True
                        self.maxRadius = circle[2]
                        self.x = circle[0]#circle[0]
                        self. y = circle[1]#circle[1]
                        cvCircle( frame, cvPoint(cvRound(self.x),cvRound(self.y)),3,
CV_RGB(0,255,0), -1, 8, 0 )
                        cvCircle( frame,
cvPoint(cvRound(self.x),cvRound(self.y)),cvRound(self.maxRadius), CV_RGB(0,0,255),
3, 8, 0 )
                        self.maxRadius2 = self.maxRadius
                        self.x2 = self.x
                        self.y2 = self.y
                #escreve os dados capturados dos círculos
                if found == True:
                        print " ",cvRound(self.x), " ",cvRound(self.y), " ", cvRound(self.maxRadius)
                        dadoString_X = str(self.y)
                        dadoString_Raio = str(self.maxRadius)
        self.amostraDados_X.write(dadoString_X)
        self.amostraDados_X.write("\n")
        self.amostraDados_Raio.write(dadoString_Raio)
```

```
            self.amostraDados_Raio.write("\n")
                        if found == False:
                                    print "P ",cvRound(self.x2), " ",cvRound(self.y2), " ",
                        cvRound(self.maxRadius2)
                                    dadoString_X = str(self.y2)
                                    dadoString_Raio = str(self.maxRadius2)
            self.amostraDados_X.write(dadoString_X)                 self.amostraDados_X.write("\n")
            self.amostraDados_Raio.write(dadoString_Raio)
            self.amostraDados_Raio.write("\n")
                        #exibe os frame das imagens capturadas
                        cvShowImage ('Camera', frame)
                        cvShowImage( "Apos o Filtro de Cor", self.thresholded )
                        k= cvWaitKey(1)
                        #encerra a captura se apertar ECS no teclado
                        if k == '\x1b': # ESC
            self.amostraDados_X.close()
            self.amostraDados_Raio.close()
                                    print 'Fechado'
                                    break
cvDestroyAllWindows() # fecha todos as janeles do programa
#self.portSerial.Stop_Serial()
#inicia a classe da captura
if __name__ == '__main__':
            captura = capt_circulo()
            captura.start()
```