# Tracking de uma bola em um jogo de futebol

Bruno Buss e Lucas Pierezan

6 de julho de 2011

**Resumo**

Este relatório diz respeito ao trabalho de realizar o *tracking* de uma bola em jogo de futebol. Apresentamos aqui as ideias e métodos utilizados bem como dificuldades encontradas.

## 1 Introdução

Nesse trabalho foi implementado um algoritmo para realizar o *tracking* de uma bola em um jogo de futebol. Este problema tem diversas aplicações como a detecção automática de eventos do jogo (como gol e impedimento) e geração de dados estatísticos de interesse de emissoras de tv e times de futebol. Na literatura podemos encotrar alguns trabalhos sobre esse assunto [][][][] indicando seu aspécto desafiador devido a fênomenos como oclusão e distorção da imagem da bola em alta velocidade. Baseamos nosso trabalho principalmente em ideias presentes em []. Outras abordagens como [] e [] utilizam como sub-procedimento a detecção de jogadores o que aumentaria consideravelmente a complexidade do trabalho.

Como em [] utilizamos a Transformada Circular de Hough (CHT) como ferramenta principal. A CHT é um método clássico utilizado para achar padrões circulares em imagens []. Existem muitas adaptações da transformada[] e buscamos incorporar um conjunto de adaptações priorizando os aspéctos particulares do nosso problema. Utilizamos a CHT como procedimento para gerar um valor de quão circular é uma região, para posterior filtragem. Na seção 2 explicamos com mais detalhe como isso é feito.

Além da CHT, utilizamos também uma análise de similaridade de histograma para gerar um valor de quão similar é o histograma de uma região comparado com o histograma da bola, que supomos saber. Com os valores de circularidade e similaridade de histograma geramos um *score* final que é então utilizado para localizar a bola. Na seção 3 explicamos como essa análise de histograma é realizada e como é gerado o *score* final.

Na seção 3 explicamos como utilizamos o conceito de Região de Interesse (ROI) para restringir a busca da bola usando informações obtidas em iterações passadas do algoritmo. O comportamento dessa ROI é descrito em detalhes bem como os problemas gerador por essa abordagem.

Vale ressaltar que o algoritmo foi implementado em C++ e utilizamos a biblioteca OpenCV para efetuar alguns sub-procedimentos de processamento de imagens que julgamos secundários.

## 2 Calculando valor de circularidade

Como já foi dito, para gerar o valor de circularidade, utilizamos a Transformada Circular de Hough (CHT). A CHT trabalha no espaço das arestas, ou seja, buscando padrões circulares no espaço das arestas. Aqui já podemos ver a hipotese da bola estar relativamente bem contrastada na imagem, aparecendo então no espaço das arestas.

Para achar as arestas da imagem $I$, em RGB, primeiramente convertemos essa para a imagem $G_I$ em escalas de cinza através da função "cvtColor" do openCV. Aplicamos também um borramento gaussiano em $G_I$ para reduzir a influência de ruidos usando a função "GaussianBlur". Finalmente utilizamos a função "Canny" para localizar as arestas. Nossa implementação da CHT se encontra na função "houghC1", no arquivo "hough.cpp".

A CHT busca círcunferências no espaço das arestas por um esquema de votação em um espaço acumulador. O espaço acumulador é o espaço dos parâmetros da circunferência que estamos buscando. Se estamos buscando circunferências que tenham raio variando no intervalo $[minR, maxR]$ e centro $(x, y) \mid minX \leq x \leq maxX \; e \; minY \leq y \leq maxY$ o espaço acumulador é da forma $[minR, maxR] \times [minX, maxX] \times [minY, maxY]$. Cada ponto no espaço acumulador representa uma possível circunferência e portanto é da forma $(cx, cy, R)$. Cada ponto de aresta $P = (x, y)$ e raio $R$ vota, no espaço acumulador, nos pontos de centro que poderiam gerar o ponto $P$ como parte de uma circunferência. De fato, na implementação clássica da CHT, para cada ponto de aresta $P$ e raio $R$ é realizada uma votação no conjunto de pontos $S_{P,R} = \{(cx, cy, R) \mid (cx - x)^2 + (cy - y)^2 = R^2\}$ no espaço acumulador. O método consiste então em processar todos os pontos de arestas, para os valores $R$ de raio que estão sendo considerados, e acumular votos no espaço dos parâmetros. Os pontos, no espaço dos parâmetros, com maior número de votos são apontados como potenciais centros de circunferêmcias. É importante, ao processar o número de votos recebido pelo ponto $(cx, cy, R)$, normalizarmos dividindo pelo máximo de votos que este ponto poderia ter rerebido. Com isso obtemos um valor de circularidade em $[0, 1]$ que é invariante ao tamanho do raio sendo buscado.

Em nossa implementação da CHT, o espaço acumulador é representado pela estrutura *acumulator* e pode ser encontrado no arquivo "hough.cpp". Essa estrutura é basicamente uma matriz tridemensional $m$ com as informações do $minR$ e $maxR$. Essa matriz $m$ contém inteiros, que indicam o quantidade de votos recebida por um ponto.

Para realizar a votação de um ponto de aresta $P$ considerando um raio

$R$ foi necessário encontrar meios de discretizar a círcunferência de raio $R$. Essa discretização foi feita utilizando-se a representação paramétrica da circunferência $x = rcos(\theta)$ $e$ $y = rsin(\theta)$ variando $theta$ de pequenos saltos de tamanho "step" calculados em função de $R$. Em nossa implementação utilizamos a informação do vetor gradiente para restringir a região de votação do ponto de aresta $P$. Ocorre que, se $P$ for parte de um arco de circunferência, o vetor gradiente de $P$ em $G_i$, $(P_x, P_y)$ possui a direção do centro da circunferência (apontando para a região mais clara). Portanto, usando a segunda hipotese de que a bola é mais clara que o fundo, ao considerar o conjunto $S_{P,R}$ dos pontos que são votados ao processar o ponto $P$ com raio $R$ votamos apénas em um arco dessa circunferência centrado na direção de $(P_x, P_y)$. Esse processamento de votação de um ponto de aresta se encontra na função "incCirc", no arquivo "discretizacao.cpp".

Após o processamento de votação, através da função "incCirc" descrita acima, usamos os valores de votação obtidos para gerar um valor de circularidade de uma circunferência $(cx, cy, R)$. Para isso, criamos a estrutura "acmPoint", que representra uma circunferência $(cx, cy, R)$ junto com seus *scores* (de circularidade, histograma e final). Para calcular o valor de circularidade do ponto $(cx, cy, R)$, com base na votação previamente, utilizamos a função "calcVNorm". Nessa função, consideramos não só a votação no próprio ponto, mas também de seus 8 vizinhos realizando uma média ponderada de acordo com uma máscara de pesos fixa. Como já mencionado, normalizamos o valor obtido pelo perímetro do círculo discretizado para obter um *score* em $[0, 1]$ independende do tamanho do raio. Os perímetros das circunferências discretizadas são pré-calculados na função "fillPreCalcDiscP()".

Por final, na função "houghC1", durante o processo de cálculo do diversos valores de circularidade, utilizamos uma *heap* para guardar os "thNCirc" maiores valores encontrados, que são então retornados como produto final de nossa função.

## 3 Calculando valor de similaridade de histograma

No princípio utilizando somente a CHT e o cálculo do valor de circularidade, como descrito anteriormente, para testes. Realizamos testes utilizando a abordagem de retornar como localização da bola o maior valor de circularidade encontrado na imagem. Já nessa etapa nosso algoritmo de *tracking* obteve um bom corpotamento nos casos onde a bola estava bem formada no espaço das arestas. Porém, mesmo nos casos em que a bola está bem contrastada na imagem a ocorrência de outros padrões circulares, como a cabeça do jogador, produziam muitos erros. Para amenizar esse problema optamos por criar outro *score*, independente de circularidade, que indicasse semelhança com alguma propriedade da bola. Considerando o caráter homogêneo da coloração de uma bola de futebol optamos por uma análise de semelhança

de histograma, cujas funções se encontram no arquivo "histograma.cpp".

A análise de histograma consiste em comparar o histograma de uma região circular, representada por um "acmPoint" $C$, e gerar para $C$ um valor de similaridade com o histograma de cores da bola sendo procurada. Para isso, supomos conhecer o histograma de cores da bola. Esta análise de histograma é realizado nas imagens em formato RGB, já que a transformação para tons de cinza acarretaria em muita perda de informação. Vale observar também que os histogramas são normalizados de forma que geramos uma informação que é idependente do tamanho do raio sendo buscado.

Na construção do histograma, realizado na função "calcHistogramScore", dividimos a faixa de cor de cada canal $[0, 255]$ em sub-faixas identificando cores na mesma sub-faixa. Portanto nosso histograma $H$ tem domínio no espaço de cor $I \times I \times I$ onde $I$ é o número de sub-faixas que estamos considenrado. Essa abordagem é interessante uma vez que pequenas modificações nas intensidades dos pixels não alteram o formato do histograma e a complexidade de tempo para processa-lo diminui consideravelmente.

Para calcular o valor de similaridade primeiramente calculamos um valor de incompatibilidade, invertendo este ao final. Usamos o método Chi-Square da forma como realizado pelo openCV, como indicado em []. Neste método calculamos $d(H_1, H_2) = \sum \frac{(H_1(x)-H_2(x))^2}{(H_1(x)+H_2(x))}$, para $x$ variando no domínio, como sendo o valor de incompatibilidade. Nosso valor de similaridade então é computado $scoreHistograma(H_1, H_2) = 1 - \frac{d(H_1,H_2)}{2}$, já que a função $d(H_1, H_2)$ é atinge um valor máximo 2.

Ao final, tendo calculado para um "acmPoint" seu valor de circularidade ("scoreCircular") e seu valor de histograma ("scoreHistogram") calculamos então o *score* final ("scoreFinal"), que leva em consideração as duas pontuações anteriores. Para calcular o *score* final optamos por utilizar uma média ponderada dando mais peso ao valor de similaridade de histograma. Isso porque, no decorrer do algoritmo, só vamos efetuar a análise de histograma em pontos já filtrados pelo *score* de circularidade. Outro fator importante é que, para círculos pequenos, as informações geradas pelo histograma são muito ruidosas, já que um pixel tem grande influência no formato do histograma. Com isso em mente, para círculos pequenos, utilizamos outro conjunto de pesos na média ponderada para calcular o *score* final, dando menos prioridade ao histograma. Essa política para calcular o *score* final se encontra na função "calculaScore" dentro da estrutura "acmPoint".

# 4 ROI e Comportamento Geral

Utilizando as ferramentas descritas na seção anterior já somos capazes de localizar a bola em condições adequadas de contorno, utilizando o cálculo dos valores de circularidade e histograma. Porém, em situações de ocluções parciais, esses valores já não são sufiscientes para garantir um bom desem-

penho da localização já que a bola não está bem formada no espaço das arestas. Nessa situação é comum gerarmos erros indicando localizações de circunferências longe das encontradas anteriormente e com raios distintos. Com isso em mente, utilizamos o conceito de região de interesse (ROI) para levar em consideração informações obtidas nas iterações passadas e ao final, realizar de fato o *tracking* da bola. O comportamento da ROI se encontra principalmente no arquivo "roi.cpp" e parte no arquivo "main.cpp".

O funcionamento da ROI é baseado na hipotese de movimento contínuo da bola. Portanto, se em uma iteração localizamos a bola em uma posição $C$, com raio $R$, esperamos que no frame seguinte a bola se encontre próxima de $C$ e com raio próximo de $R$. Na nossa implementação, ao encontrar a bola, representada pelo "acmPoint" $(cx, cy, R)$ geramos uma jenala retangular $Ret$ centrada em $(cx, cy)$ e com lados distando $scaleRad \times R$ de $(cx, cy)$. O fator de escala "scaleRad" utilizado nos testes foi 3. Essa janela $Ret$ será nossa ROI na próxima iteração. Ou seja, na próxima iteração vamos buscar a bola somente na sub-imagem gerada pela janela $Ret$. Além disso, na iteração seguinte, vamos buscar circunferências com raio no intervalo $[R - deltaR, R + deltaR]$. Nos testes consideramos uma variação de raio de 1 unidade de pixel.

Dado uma ROI, para gerar o único "acmPoint" que indicaremos como a localização da bola utilizamos a função "findBall". Nesta obtemos o conjunto $B$ dos "acmPoints" com os 10 maiores valores de circularidade. Os "acmPoints" em $B$ são submetidos a análise de histograma, se o histograma da bola foi fornecido, e então gera-se o *score* final destes. Retornamos como localização da bola o "acmPoint" com maior *score* final.

A abordagem de ROI descrita acima aumentou consideravelmente o desempenho do algoritmo já que trabalhamos só com parte da imagem. Além disso, a taxa de acerto do algoritmo também aumentou. Porém, essa estratégia introduz a problemática de que, ao errar a primeira vez, o *tracking* nas iterações futuras é comprometido. De fato, nos testes, percebemos situações em que, ao primeiro erro do algoritmo, a ROI converge para uma região que não é a bola e muitas vezes sequer circular. Portanto, como última etapa do nosso trabalho, foi necessário pensar mecanismos para detectar que o algoritmo está errando e então reiniciar a busca da bola na imagem inteira. Adotamos então a solução de que ao receber por $"countRestart" = 5$ iterações consecutivas um "acmPoint" (que supostamente é a bola) com *score* final abaixo de um limite fixo "thRestart" então reiniciamos a busca da bola na imagem inteira. Com isso, concluimos o funcionamento geral do algoritmo.

# 5 Considerações Finais

O algoritmo apresentou um desempenho relativamente bom nos testes que foram realizados. A ocorrência de erros se dá, em sua grande maioria, quando a hipotese de contraste da bola não é respeitada ou em movimentos

muito rápidos, onde seu formato circular e a continuidade do movimento se perdem.

Vale observar que muitos parâmetros são utilizados no funcionamento do algoritmo como: limites do Canny, pesos de médias ponderadas, "thNCirc", "thRestart", "deltaR", "scaleRad", variação angular em torno do vetor gratiente e etc. Muitos desses parâmetros estão relacionados. Por exemplo, limites baixos para Canny fazem com que valores de circularidades sejam maiores e portanto exige um aumento no "thRestart". Além disso, a alteração destes parâmetros gera modificações consideraveis no comportamento do algoritmo. Sugerimos como trabalho futuro um estudo mais detalhado da relação entre esses parâmetros e melhor ajuste dos mesmos. Indicamos também a possibilidade de utlizar uma estratégia de valoração dinâmica desses.

Indicamos também como trabalho futuro a utilização de técnicas de *template matching* para gerar mais um valor de similaridade com a bola e estudo de outras estratégias para reiniciar a busca.

## Referências


[1] D'Orazio, T. e Ancona, N. e Cicirelli, G. e Nitti, M.
A Ball Detection Algorithm for Real Soccer Image Sequences
Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 1 - Volume 1

[2] Choi, K. e Seo, Y.
Tracking Soccer Ball in TV Broadcast Video
Image Analysis and Processing – ICIAP 2005

[3] Tong, Xiao-Feng e Lu, Han-Qing e Liu, Qing-Shan
An Effective and Fast Soccer Ball Detection and Tracking Method
17th International Conference on Pattern Recognition (ICPR'04) - Volume 4

[4] Yuen, H. K. e Princen, J. e Illingworth, J. e Kittler, J.
Comparative study of Hough transform methods for circle finding
Image and Vision Computing, Volume 8 Issue 1, February 1990

[5] OpenCV C++ Reference
`http://opencv.willowgarage.com/documentation/cpp/index.html`