UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

ESTUDO DE *TRACKING* PARA INSERÇÃO DINÂMICA DE LINHA DE IMPEDIMENTO EM VÍDEOS DE JOGOS DE FUTEBOL

RODRIGO BUSATO SARTOR

BLUMENAU
2011

2011/2-25

RODRIGO BUSATO SARTOR

ESTUDO DE *TRACKING* PARA INSERÇÃO DINÂMICA DE LINHA DE IMPEDIMENTO EM VÍDEOS DE JOGOS DE FUTEBOL

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Paulo César Rodacki Gomes, Dr. – Orientador

BLUMENAU
2011

2011/2-25

SISTEMA DE *TRACKING* PARA INSERÇÃO DINÂMICA DE LINHA DE IMPEDIMENTO EM VÍDEOS DE JOGOS DE FUTEBOL

Por

RODRIGO BUSATO SARTOR

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Paulo César Rodacki Gomes, Dr – Orientador, FURB
Membro:	Prof. Dalton Solano dos Reis, M. Sc. – FURB
Membro:	Prof. Antônio Carlos Tavares – FURB

Blumenau, 14 de dezembro de 2011

Dedico este trabalho a toda minha família, amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

Inicialmente agradeço aos meus pais Clademir e Iara pelo apoio que recebi meus avós Gualdino, Nilva, Alcides e Erna, a meu irmão Rafael pelas diversas correções deste trabalho, e a minha irmã Estefani. Agradeço a toda minha família que mesmo estando longe sempre me deu amor, carinho e me apoiou incondicionalmente não somente durante a conclusão deste trabalho, mais em todo o decorrer da graduação. Sem o apoio deles certamente eu não teria conseguido terminar mais uma etapa em minha vida.

Aos amigos que sempre quando precisei, estavam prontos para me ajudar e apoiar, pelos momentos de distração que foram muito necessários para a conclusão deste trabalho.

Ao meu orientador Paulo César Rodacki Gomes, pelo apoio e confiança depositada neste trabalho.

Viva como se você fosse morrer amanhã. Aprenda como se você fosse viver para sempre.

Mahatma Gandhi

RESUMO

Este trabalho apresenta os passos iniciais para realizar a calibração automática de câmeras, utilizada no intuito de realizar o rastreamento de um jogador de futebol, indicando a posição da linha de impedimento durante uma jogada. O método proposto utiliza apenas como entrada de dados de vídeo sem nenhuma informação adicional de marcação de pontos de interesse. A validade da proposta é apresentada através da implementação de um protótipo em linguagem Java.

Palavras-chave: Computação gráfica. Visão computacional. Filtros de imagem.

ABSTRACT

This paper presents the initial steps to perform the automatic calibration of cameras, used in order to catch the tracking of a soccer player, indicating the position of the offside line when he is moving. The proposed method uses as data entry video files only. The validity of the proposal is presented by implementing a prototype in Java.

Keywords: Computer graphics. Computer vision. Image filters

LISTA DE ILUSTRAÇÕES

Figura 1- Imagem com uso da linha de impedimento..... 13
Quadro 1- Função da imagem 17
Figura 2- Formato de uma imagem..... 17
Quadro 2 – Luminancia..... 18
Figura 3 – Imagem antes da utilização dos filtros 18
Figura 4- Imagem após calculo da luminância..... 19
Quadro 3- Negativo..... 19
Figura 5- Imagem após aplicação da negativa 20
Figura 6- Imagem após filtragem laplaciana..... 20
Figura 7- Imagem após filtragem gaussiana..... 21
Figura 8- Resultado da filtragem laplaciana na figura 6. 22
Figura 9- Resultado da filtragem *LoG* aplicado na negativa..... 23
Figura 10- Imagem após calculo da luminância..... 24
Figura 11 - Tela com a imagem estática carregada 25
Figura 12 – Cena no ambiente tridimencional 26
Figura 13 – Extração dos segmentos de reta..... 27
Figura 14 – Diagrama de casos de uso..... 29
Figura 15 – Diagrama de classes 30
Figura 16 – Diagrama das classes descendentes de CodecVídeo..... 32
Figura 17 – Diagrama de sequência. 33
Figura 18 – Diagrama de sequência dos codecs..... 34
Quadro 4 – Gerando uma cópia do frame 36
Quadro 5 – Método filterExecute()..... 37
Quadro 6 – Implementação da equação laplaciana da gaussiana. 38
Quadro 7 – implementação do filtro *threshold*..... 38
Quadro 8 – implementação da delimitação da área do blob..... 39
Quadro 9 – populando a matriz de confiaça entre os blobs 39
Figura 19 – Tela inicial 40
Figura 20 – Execução do *tracking* 41
Figura 21 – Realce das linhas..... 42

SUMÁRIO

1 INTRODUÇÃO 10
1.1 OBJETIVOS DO TRABALHO 11
1.2 ESTRUTURA DO TRABALHO 11
2 FUNDAMENTAÇÃO TEÓRICA 13
2.1 POSIÇÃO DE IMPEDIMENTO 13
2.2 DETECÇÃO DE OBJETOS EM MOVIMENTO 14
2.2.1 Tracking 14
2.2.2 Objetos em movimento 15
2.2.3 Optical flow 15
2.3 FILTROS DE IMAGEM 16
2.3.1 IMAGENS DE VÍDEO 16
2.4 REALCE DAS LINHAS 20
2.5 TRABALHOS CORRELATOS 24
2.5.1 Calibração de câmeras para cálculo de impedimentos 24
2.5.2 Ambiente virtual tridimensional para cálculo de impedimento 25
2.5.3 Acompanhamento de cenas com calibração automática de câmeras 26
3 DESENVOLVIMENTO DO PROTÓTIPO 28
3.1 REQUISITOS PRINCIPAIS DO PROTÓTIPO 28
3.2 ESPECIFICAÇÃO 28
3.2.1 CASOS DE USO 28
3.2.2 DIAGRAMA DE CLASSES 29
3.2.2.1 Codecs 31
3.2.3 DIAGRAMA DE SEQUÊNCIA 33
3.3 IMPLEMENTAÇÃO 34
3.3.1 Técnicas e bibliotecas utilizadas 35
3.3.2 Operacionalidade da implementação 40
3.4 RESULTADOS E DISCUSSÃO 42
4 CONCLUSÕES 44
4.1 EXTENSÕES 44
REFERÊNCIAS BIBLIOGRÁFICAS 45

1 INTRODUÇÃO

É de conhecimento que todo ser humano está sujeito a cometer erros, mas existem algumas circunstâncias onde falhas humanas são muito frequentes, como no esporte, onde um erro, seja por falta de visão ou atraso de uma fração de segundo, pode definir o resultado de um jogo ou mesmo de um campeonato.

Um dos principais exemplos de esporte nestas condições é o futebol, em que um juiz e dois auxiliares analisam as jogadas sem poder contar com recursos auxiliares como vídeos das mesmas, e qualquer erro pode ser decisivo. Um dos lances das regras de futebol que mais gera discussões entre jogadores e torcedores é o impedimento que, segundo a Confederação Brasileira de Futebol (2010), é caracterizado quando um jogador encontra-se mais próximo da linha de meta adversária do que a bola e o penúltimo adversário.

Para se resolver várias dúvidas relacionadas a erros de arbitragem, ou mesmo na inserção de artifícios publicitários durante as transmissões, passou-se a utilizar a computação gráfica.

Este trabalho investiga o problema no intuito de mostrar a linha de impedimento durante jogadas de ataque em vídeos de jogos de futebol, acompanhando o último jogador da defesa e verificando se durante o andamento de determinado lance ele está mantendo os atacantes em posição de impedimento. Para que a linha de impedimento possa seguir o jogador serão utilizadas técnicas de visão computacional para fazer este acompanhamento (*tracking)* dos jogadores de interesse.

Neste tipo de problema, para o cálculo da distância dos jogadores, o sistema de coordenadas da câmera deve ser alinhado com as coordenadas do universo. Para isso transformações geométricas devem ser aplicadas para igualar os dois sistemas. Tais transformações devem ser obtidas através de um algoritmo de calibração de câmeras. Esta calibração deve ser dinâmica, realizada a cada quadro de vídeo permitindo a mudança de posição da câmera. A partir desta calibração seria possível calcular a distância entre os jogadores e a linha de fundo a partir das imagens de vídeo, e esta distância define então se sua posição é regular em relação à regra do impedimento.

No protótipo implementado neste trabalho, o usuário deve abrir um vídeo contendo o lance em que se deseja calcular o possível impedimento. A ideia é que o sistema identifique os jogadores, o usuário informe quais deles são os atacantes e quais são os defensores. Em seguida o sistema deve calcular as coordenadas do campo, identificar o último jogador da

defesa, inserir uma linha de marcação junto a ele e realizar seu acompanhamento identificando um possível impedimento. O presente trabalho aborda a etapa inicial deste processo que seria o rastreamento dinâmico dos jogadores nos vídeos. Após esta etapa, futuros trabalhos devem investigar e implementar a identificação dos jogadores, a detecção de linhas de campo para cálculo de homografia, o cálculo de distâncias nas coordenadas do mundo a partir das coordenadas de vídeo e a inserção dinâmica da linha de impedimento no vídeo.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver os passos iniciais para a realização de um protótipo de software que irá inserir uma linha de impedimento durante um vídeo de uma partida de futebol.

Os objetivos específicos são:

a) identificar na imagem do campo as linhas da lateral, do final do campo, e as linhas que delimitam a grande e pequena área;
b) realizar o acompanhamento automático do último defensor, utilizando técnicas de *tracking* com detecção de movimento, após o usuário ter indicado para o sistema quais dos jogadores são os defensores.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado em 4 capítulos. O capítulo 2 apresenta uma fundamentação teórica sobre a qual este trabalho é embasado. Onde incialmente será apresentada uma visão geral sobre o que é o impedimento, e o que é dito na regra do futebol. Em seguida são apresentados os temas inerentes à realização deste trabalho, dando uma visão de como se trabalha com imagens de um vídeo, de como realizar um realce, e segmentação das linhas do campo, e por último, são mostrados alguns trabalhos correlatos.

No capítulo 3 são abordados temas referentes ao desenvolvimento do protótipo, tais

como a definição dos principais requisitos, sua especificação através de diagramas da *Unifield Modeling Language* (UML) e detalhes referentes a seu desenvolvimento. Também são comentados temas referentes à operacionalidade do protótipo e discutidas as conclusões obtidas com a realização deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

As seções deste capitulo foram divididas entre os principais passos desenvolvidos neste trabalho. Inicialmente a seção 2.1 mostra uma explicação sobre o impedimento e o que diz na regra do futebol sobre ele. A partir da seção seguinte são descritas as técnicas utilizadas para a obtenção do resultado desejado. A seção 2.2 fala sobre os filtros utilizados nas imagens e transformações. Na seção 2.3 é demostrado como foi realizado o realce das linhas, para posteriormente identificar as retas. Por fim, na seção 2.4 são comentados trabalhos correlatos.

2.1 POSIÇÃO DE IMPEDIMENTO

A posição de impedimento é uma das regras do futebol que mais geram polêmica, pois o juiz auxiliar, também conhecido como “bandeirinha”, deve analisar uma jogada normalmente em grande velocidade e tomar uma decisão rápida, tendo como única ferramenta sua visão. Neste contexto passou a se utilizar a computação gráfica durante as transmissões de televisão para poder analisar precisamente se o “bandeirinha” e o juiz tomaram a decisão correta.

Segundo as regras que regem o futebol, o impedimento está descrito na regra 11, onde determina que “Um jogador estará em posição de impedimento quando se encontrar mais próximo da linha de meta adversária do que a bola e o penúltimo adversário” (CONFEDERAÇÃO BRASILEIRA DE FUTEBOL, 2010, p. 27).

Na figura 1 verificam-se duas linhas indicando a posição dos jogadores, uma em cima do penúltimo jogador de defesa em vermelho e a outra em amarelo no ultimo atacante. Como o jogador defensivo está mais próximo da linha de meta, a jogada é legal.

Fonte: Starosky (2003, p. 16).

Figura 1- Imagem com uso da linha de impedimento

2.2 DETECÇÃO DE OBJETOS EM MOVIMENTO

A detecção de movimento vem tendo um grande uso dentro da área visão computacional com o objetivo de identificar em uma sequencia de vídeo os objetos que estão estáticos do que estão em movimento. Segundo Santos (2008, p. 19), as maiores partes dos problemas relacionados à visão computacional estão associadas à detecção de objetos, devido a diversos fatores como mudanças de luminosidade, sombras, imagens tremidas e oclusão. Com isso se torna difícil conseguir um método genérico para resolver este problema. Usualmente vem se utilizando para a resolução deste problema uma média de resultados obtidos nas imagens para formar um ambiente de aprendizagem estatístico.

2.2.1 Tracking

Segundo Tyagi et al. (2007, p. 2), *tracking* é um caso especial de registro de imagens, onde procura-se um objeto dado em uma imagem alvo. O espaço de busca é limitado, pois é suposto que o objeto tem uma trajetória de movimento contínua, assim é esperado que ele se encontrasse próximo da sua antiga localização.

Para realizar a segmentação de objetos (regiões de interesse separada do fundo) da cena, tipicamente são utilizados sistemas que dividem a cena em duas regiões, que são o primeiro e segundo plano, onde apenas o primeiro plano contém eventos de interesse, e onde o segundo plano é o fundo sendo ele relativamente imutável ao longo do tempo. Para alcançar esta separação, são utilizados técnicas tais como de detecção de movimento através do algoritmo *Optical flow* (DENMAN; CHANDRAN; SRIDHARAN, 2007).

Muitos dos sistemas de *tracking* têm como primeiro passo a detecção de movimento, uma vez que este movimento é detectado, podem-se utilizar diversos métodos para manter o *tracking*. Um dos métodos é a utilização do algoritmo de segmentação adaptativa *Non-Homogeneous Detector* (NHD), para a definição dos elementos em movimento. Após a detecção utiliza-se o algoritmo *Optical flow*, para manter o *tracking* do objeto. A Integração destes algoritmos permite que possa ser aplicado o Optical flow somente aos pixels em movimento, reduzindo o consumo de recursos e a presença de erros no *tracking* (ROSA, 2010, p. 18).

2.2.2 Objetos em movimento

Um das maneiras mais fáceis para a detecção dos movimentos é a comparação pixel por pixel. Uma forma de se fazer esta comparação é gerando outra imagem onde os pixels serão gerados conforme a diferença entre as imagens comparadas. Com isto, se o pixel apresentar uma diferença maior do que um valor estipulado, o valor do pixel será um, caso contrário zero. A cena resultante deste processo é um novo frame onde houve esta diferença foi marcada com o valor um e as demais regiões que estão estáticas com zero (GONZALEZ; WOODS, 2008, p. 778).

O resultado deste processo é a geração de uma imagem binária, formada apenas pelas cores pretas e brancas. César Junior e Costa (2001, p 203) descrevem uma imagem binaria como um tipo simples e útil de imagens, pois representa uma cena apenas com dois valores, zero e um.

Dentro da literatura existem diversos método que realiza a detecção de um movimento através da remoção do fundo da imagem, um destes é o método *Background Subtraction*. Este método consiste em subtrair a imagem atual de outra usada como referência, contendo apenas o fundo da cena e construída a partir de uma sequência de imagens. A fidelidade da cor é importante durante a remoção do fundo, pois permite ao programa a correta classificação dos objetos, independentes da projeção de sombras sobre eles. O que permite ignorar as sombras do objeto, deixando de classifica-la como sendo pertencente ao objeto em movimento (SANTOS, 2008, p. 23).

Após estas operações, torna-se possível identificar os objetos que se moveram durante a cena. Tendo-se o contorno do objeto é possível definir uma forma, porém não é identificado que tipo de objeto é, e também é impossível verificar se ele pertence a alguma categoria ou não (ROSA, 2010, P. 16).

2.2.3 Optical flow

A detecção de movimento é usada para localizar os objetos em translação, resultando em uma única imagem binária, onde são mostradas as regiões de movimento. O algoritmo *Optical flow* fornece meios para determinar e representar o movimento dentro de um

sequencia de imagens.

O algoritmo proposto por Denman, Chandran e Sridharan (2007), inicia considerando todos os pixels como estacionários. Quando é detectado algum movimento em um pixel, toda região a sua volta é examinada para determinar o *Optical flow* para aquele pixel. O tamanho da área examinada é determinado pela aceleração máxima permitida para um pixel, tanto no eixo “x” quanto para o eixo “y”, onde estes valores são estipulados de acordo com os requisitos da cena. Esta área é analisada partindo de dentro para fora, partindo do pixel central e continuando pelos pixels mais extremos até encontrar o pixel correspondente. Caso não ocorrer nenhuma correspondência, a área ao lado é analisada (a uma distancia de um pixel), assim por diante até encontrar um pixel correspondente.

Uma vez determinado o movimento de um pixel, então sua nova posição pode ser prevista. Assumindo um modelo de velocidade constante, a localização do pixel no próximo frame é dada pela soma da posição atual mais a diferença entre a posição atua e a posição anterior. Tanto para o eixo “x” quanto para o eixo “y”.

Se o pixel já estava cadastrado como em movimento, então a posição esperada é a usada como posição inicial para a busca.

2.3 FILTROS DE IMAGEM

Após a etapa de reconhecimento do movimento se torna necessário identificar as linhas do campo, para que em seguida se possa calcular a distancia dos jogadores até a linha de fundo, para se identificar o impedimento. Para isso se torna necessária à realização de diversos filtros. Esta seção apresenta uma explicação sobre como se é trabalhado com os frames de um arquivo de vídeo, após é discutido sobre os filtros que consistem em detectar pontos passives de estarem sobre uma linha presentes na imagem e como suavizar problema na imagem como os ruídos.

2.3.1 IMAGENS DE VÍDEO

Um vídeo é uma sequência S de imagens I_t, onde I_t indica a imagem no tempo t do

vídeo. É comum referenciar uma imagem de vídeo como quadro (ou *frame*), onde uma propriedade importante do vídeo é referente à sua taxa de amostragem, que é medida em quadros por segundo (FPS), sendo que para uma visualização de boa qualidade esta taxa deve estar próxima a 30 quadros por segundo.

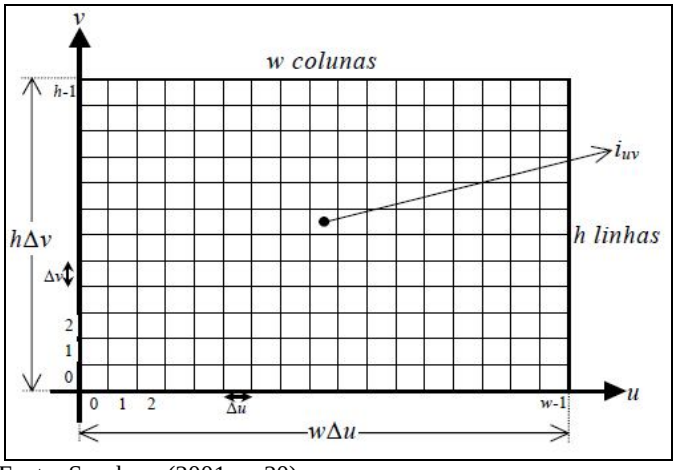
Segundo Szenberg (2001) uma imagem *I* pode ser descrita pela função representada pelo quadro 1.

$$I : D \subset \Re^2 \rightarrow \Re n$$

Quadro 1- Função da imagem

Onde *D* é o domínio da imagem e *n* é um numero inteiro maior que zero que equivale a quantidade de escalares utilizada para representar as informações de cor de um ponto qualquer. Cada uma destes escalares são quantizadas em um certo números de bits, frequentemente um byte para cada escalar, que então é representado por um número de 0 a 255.

A imagem do vídeo é representada pela figura 2 onde as resoluções horizontais e verticais são denotadas por *w* e *h*, respectivamente, e as dimensões por $w\Delta u$ *e* $h\Delta v$, onde Δu *e* Δv são tidos como igual a 1. A localização de cada ponto (*pixel*), é determinada através da junção dos pares ordenados (*u*,*v*), e sua informação de cor é definida como $i_{uv.}$



Fonte: Szenberg (2001, p. 20).

Figura 2- Formato de uma imagem

Alguns dos filtros e transformações utilizados durante o desenvolvimento deste trabalho têm como entrada ou saída imagens monocromáticas, onde, estas são caracterizadas por terem valor de *n* igual a um na função descrita pelo quadro 1, e podem ser representadas

por tons de cinza (SZENBERG, 2001, p. 20). Neste trabalho o problema de transformar imagens coloridas em monocromáticas é tratado através do cálculo da luminância, conforme quadro 2.

```
L = 0.299*R + 0.587*G + 0.114*B
Onde R, G e B definem a cor de um ponto.
```

Quadro 2 – Luminância

A figura 3 demonstra uma imagem sem ser realizada nenhuma filtragem, já figura 4 ilustra a transformação em tons de cinza realizada através do cálculo da luminância da figura 3.

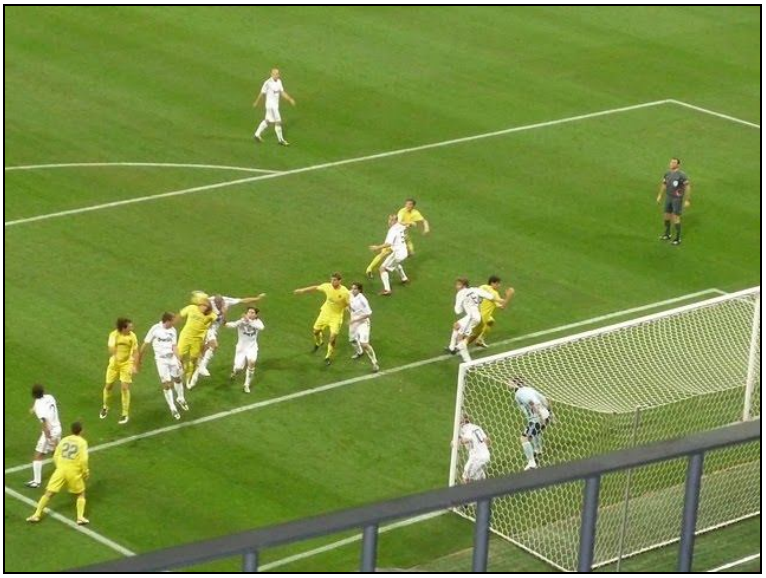


Figura 3 – Imagem antes da utilização dos filtros

Figura 4 – Imagem após calculo da luminância

Segundo Szenberg (2001) para a segmentação das linhas como as geralmente encontradas em campos de futebol, é interessante considerar o negativo de uma imagem. No caso de imagens em tons de cinza, quantizados em um byte, para imagens em escala de cinza seu negativo é representado no quadro 3.

N = 255-***I***

Quadro 3 – Negativo

Onde o valor de ***I*** pertencente à função representada pelo quadro 3, é a intensidade de cor de um ponto da imagem. Para imagens coloridas, esta fórmula deve ser aplicada para cada canal de cor.

A figura 5 ilustra a transformação negativa aplicada a figura 4.

Figura 5 – Imagem após aplicação da negativa

2.4 REALCE DAS LINHAS

O realce de linhas é realizado para detectar pontos candidatos a estarem sobre segmentos de retas, presentes na imagem, para isso foi aplicado uma filtragem laplaciana, definida através do operado matemático ∇, essa filtragem é classificada com passa-altas (frequências). Esse tipo de filtragem funciona mantendo os trechos de alta frequência e eliminando as baixas, utilizado para detectar as bordas da imagem (PINHO, 2011). A imagem 5 mostra como fica a imagem após passar por um filtro laplaciano.

Fonte: Szenberg (2001, p. 24).

Figura 6 – Imagem após filtragem laplaciana

Foi observado que a imagem utilizada possui vários pontos com frequências altas, mais que não estão sobre nenhum segmento de reta, provavelmente devido a interferências, ocasionadas pela transmissão da televisão, ou mesmo pela textura do gramado, por este motivo é aplicado um filtro passa-baixa eliminando as altas frequências. Para usar o filtro passa-baixa foi utilizado um filtro gaussiano (SZENBERG, 2001, p. 20).

A figura 6 é o resultado da aplicação de um filtro gaussiano.

Fonte: Szenberg (2001, p. 24).

Figura 7 – Imagem após filtragem gaussiana

Aplicando o filtro laplaciano, a figura 7, obteremos a figura 8, onde é perceptível a melhora da nitidez nas linhas.

Fonte: Szenberg (2001, p. 25).

Figura 8 – Resultado da filtragem laplaciana na figura 6

A composição dos dois filtros é conhecida como filtragem laplaciana da gaussiana, também chamado na literatura de *LoG*.

As marcações das linhas do campo, ou seja, os segmentos de retas que se deseja identificar estão representados por duas linhas paralelas, isso ocorre, pois o filtro laplaciano encontra as fronteira das regiões, que é onde estão localizadas as altas frequências. Para evitar esta duplicação da linha pode-se realizar mais uma composição, a transformação negativa antes do filtro *LoG*, onde o resultado pode ser visto na figura 9. Pode-se notar que a única diferença dela para a figura 8, é exclusão das linhas paralelas, substituindo por uma única linha central.

Fonte: Szenberg (2001, p. 26).

Figura 9 – Resultado da filtragem *LoG* aplicado na negativa

Como um próximo passo, é realizada a segmentação da imagem, com o interesse de obter a partir de uma imagem original uma imagem resultante com a indicação dos possíveis pontos em que passe uma reta. Isto é, são excluídos os pontos através da atribuição do valor zero, os pontos que não estão sobre nenhum segmento de reta. Para os pontos não excluídos é atribuído um valor referente ao grau de certeza da passagem de um segmento de reta. Para realizar a segmentação é utilizado o filtro de limiarização (*threshold)*.

O principio da limiarização consiste em dividir uma imagem em duas partes (fundo e o objeto). Produzindo ao seu final uma imagem binária. A forma mais simples de limiarização consiste na bipartição do histograma, convertendo os pixels cujo tom de cinza é maior ou igual a um certo valor limiar em branco e os demais em pretos (MARQUES FILHO; VIEIRA NETO, 1999, p. 71).

A figura 9 representa o resultado obtido neste trabalho após a aplicação da transformada negativa, ao filtro *LoG,* e após realizada a segmentação da imagem através do *threshold* sobre a figura 3.

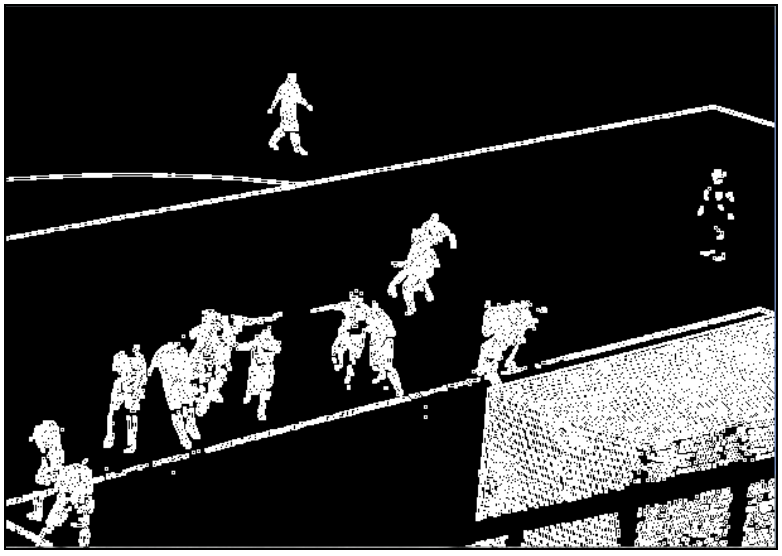


Figura 10 – Imagem após calculo da luminância

2.5 TRABALHOS CORRELATOS

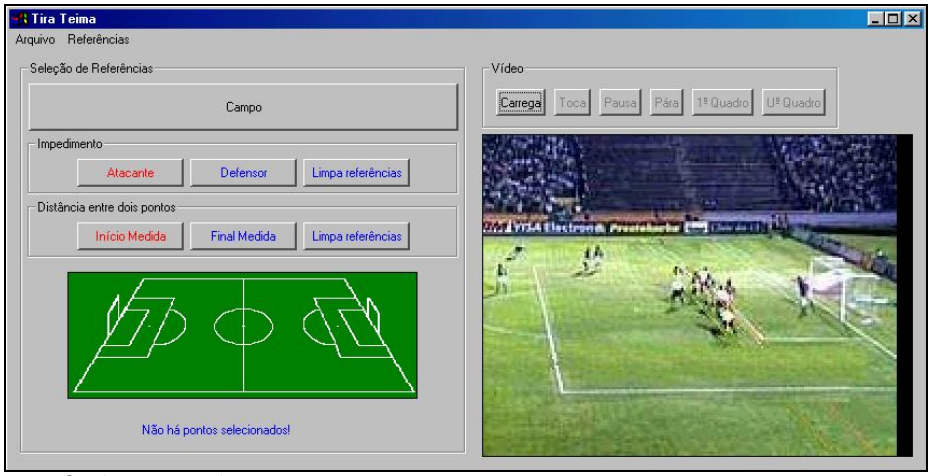
Existem diversos trabalhos que são realizados em cima da marcação de impedimento, tanto na área profissional como os exibidos nas transmissões de televisão, como no âmbito acadêmico. Dentre os trabalhos acadêmicos foram destacados três: o de Starosky (2003) o de Cristofolini (2004), e o de Szenberg (2001).

2.5.1 Calibração de câmeras para cálculo de impedimentos

Segundo Starosky (2003, p. 7), este trabalho é a apresentação de um método para a calibragem de câmeras que serão utilizadas no cálculo do impedimento e da distância entre dois pontos dentro do campo de futebol. Para isso o autor usa uma imagem estática de lances de jogos de futebol. Neste trabalho foi proposto um método que utiliza uma imagem *raster* e um modelo de campo, onde são informados apenas quatro pontos de referência que são definidos na imagem, gerando um sistema de coordenadas tridimensional. Após isto as posições dos jogadores, ou os dois pontos desejados, são indicados pelo usuário e, a partir daí, são calculadas as posições dos pontos no sistema de coordenadas em 3D e determinada a

distancia entre dois pontos do campo.

A figura 11 demonstra como é apresentada a tela do protótipo após a imagem desejada ser carregada.



Starosky (2003, p. 43).

Figura 11 - Tela com a imagem estática carregada

2.5.2 Ambiente virtual tridimensional para cálculo de impedimento

Este trabalho é uma extensão do desenvolvido por Starosky (2003), onde a diferença está no cálculo da calibragem de câmeras. O autor Cristofolini (2004) passou a utilizar "n" pontos de referência em vez dos originais quatro. Além disso, o autor passou a gerar uma reconstrução da cena em um ambiente virtual tridimensional. Este ambiente oferece recursos para visualizar a cena de qualquer posição e ângulo, movimentar-se dentro do campo, medir distâncias e calcular o impedimento. Para gerar o ambiente tridimensional o autor usou a *irrlitch engine*, que é uma *game engine*[1] para desenvolvimento de jogos 3D, no desenvolvimento dos objetos 3D que compõem a cena, tais como o campo, trave, jogadores e a bola foi utilizado o software 3D Studio.

A figura 11 apresenta a cena tridimensional gerada pelo protótipo.

[1] G*ame engine* ou motor de jogos representa toda a tecnologia do núcleo de um jogo, entre suas funcionalidades estão à implementação da detecção de colisão, movimentação de câmeras, entre outros.

Cristofolini (2004, p. 43).

Figura 12 – Cena no ambiente tridimensional

2.5.3 Acompanhamento de cenas com calibração automática de câmeras

Szenbert (2001) apresenta um algoritmo que sem utilizar nenhuma informação adicional recupera em tempo real a posição e os parâmetros da câmera em uma sequência de imagens contendo algumas visualizações conhecidas. Para conseguir isso ele explora a existência de segmentos de retas na imagem cujas posições são conhecidas no plano tridimensional.

Inicialmente são extraidos os segmentos de retas longas da primeira imagem, com posse disto é realizado um reajuste onde se obtêm os pontos de interesse. Estes pontos passam por um procedimento que encontra a câmera responsável pela visualização do modelo. A partir da segunda imagem da sequência apenas uma parte do algoritmo torna-se necessária. Com isso é possível realizar este processamento em tempo real.

Na figura 13 é mostrada uma sobreposição na imagem dos segmentos de reta extraídos e reconstruídos.

Szenbert (2001, p. 100).

Figura 13 – Extração dos segmentos de reta

3 DESENVOLVIMENTO DO PROTÓTIPO

Durante este capítulo será discutido sobre o processo de desenvolvimento do protótipo proposto neste trabalho. Serão abordadas as seguintes etapas: requisitos principais do protótipo, especificação, implementação e os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROTÓTIPO

O sistema proposto deverá:

a) permitir inserir um vídeo de uma partida de futebol (Requisito Funcional - RF);
b) disponibilizar o vídeo para visualização com o rastreamento de jogadores (RF);
c) disponibilizar o vídeo para a visualização do realce de linhas (RF);
d) utilizar a linguagem de programação Java, no ambiente de desenvolvimento Eclipse (Requisito Não-Funcional - RNF);
e) utilizar o *framework Java Media Framework* (JMF) (RNF) ;
f) disponibilizar o vídeo gerado sem atrasos ou falhas de reprodução (RNF).

3.2 ESPECIFICAÇÃO

Para especificar o sistema, será utilizado UML, descrito por Furlan (1998). Os diagramas que serão apresentados são os de casos de uso, de classes e de sequencia.

Para a geração destes diagramas foi utilizado à ferramenta *Enterprise Architect* versão 7.5

3.2.1 CASOS DE USO

A figura 13 demonstra o diagrama de casos de uso do sistema desenvolvido. O usuário deve informar um vídeo contendo um jogo de futebol, o protótipo deve exibir este vídeo

indicando a linha de impedimento.

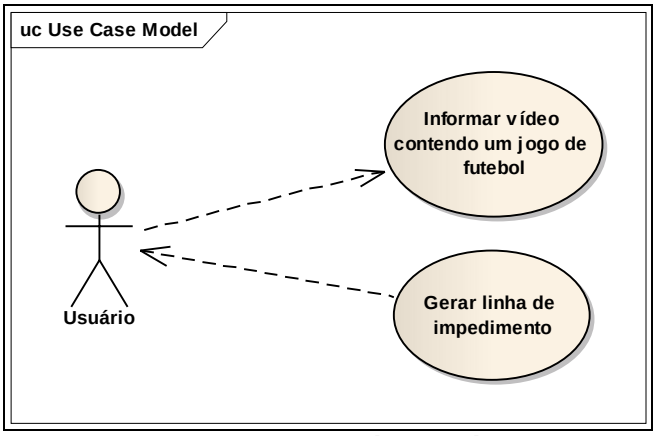


Figura 14 – Diagrama de casos de uso

Inicialmente o usuário deverá informar um arquivo de vídeo contendo um jogo de futebol, em seguida o protótipo deverá exibir este mesmo arquivo vídeo indicando a linha de impedimento.

3.2.2 DIAGRAMA DE CLASSES

A figura 14 demonstra o diagrama de classes desenvolvido no protótipo proposto.

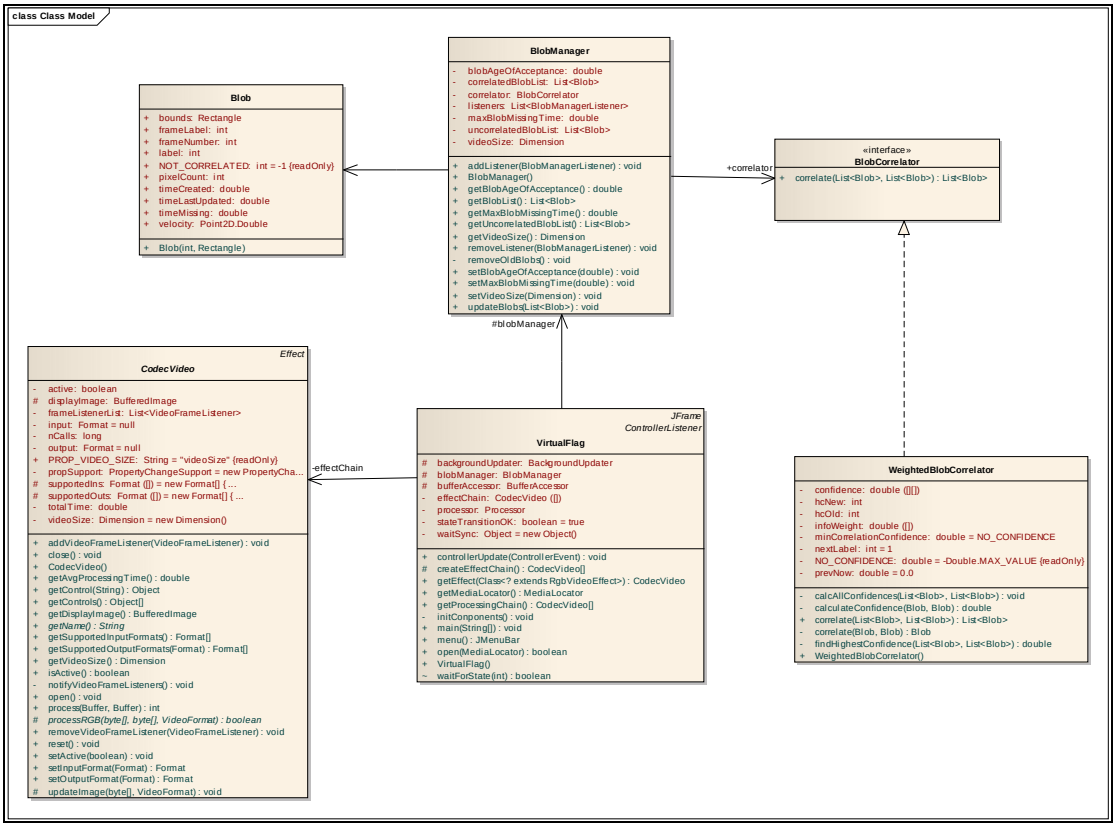


Figura 15 – Diagrama de classes

A classe `VirtualFlag` representa o programa principal, contém a interface com o usuário. Esta classe é responsável pelos métodos de leitura do vídeo escolhido pelo usuário. Os métodos contidos nesta classe são responsáveis em manter a compatibilidade com as interfaces do JMF, que é a API utilizada como base para o processamento de vídeos neste trabalho. Dentro do método `open` são realizadas as operações referentes a configuração do arquivo de mídia, para poder ser utilizado pelo JMF. Esta classe também contem as informações para montar o layout de tela e exibir a mesma. O método `createEffectChain` é responsável por popular um array de `CodecVideo`, com todos codecs necessários. Este array será setado para dentro do JMF ao chamar o método `setCodecChain` passando este array por parâmetro, este método é pertencente a classe `TrackControl` nativa do JMF, responsável por controlar e manipular os dados das faixas da mídia individualmente.

A classe abstrata `CodecVideo` é a classe responsável por implementar a interface `Effect` do JMF, e por processar cada frame do vídeo separadamente através do método `process()`, onde este método é especifico do JMF sendo chamado a cada frame do vídeo.

Esta classe é abstrata, todos os filtros necessários são executados por sub-classes desta,

permitindo que estas apenas desenvolvam suas rotinas específicas, mantendo o métodos genéricos realizado na classe mãe. O diagrama destas classes pode ser visto pela figura 16.

A classe `Blob` é responsável por armazenar as informações referentes à área com movimento detectado, armazenando informações como velocidade do movimento, tempo total em que este movimento não é mais encontrado, os valores que limitam a área do movimento, armazenado por um objeto `Rectangle`, a quantidade de pixel nesta área, entre outras informações.

O gerenciamento destas áreas são realizados através da classe `BlobManager`, onde possui uma lista para armazenar os blobs que tem correlação e os que não tem mais correlação nenhuma. O método `updateBlobs` é responsável por atualizar a lista das áreas em movimento, relacionando os blobs antigos com os recém detectados, e removendo os que não são mais considerados em movimento. A remoção dos blobs antigos se da através do método `removeOldBlobs`. O método `getBlobList`, retorna uma lista com todos os blobs correlacionados.

A cada atualização dos blobs executados pelo método `updateBlobs` deve-se verificar se os novos blobs, tem relação com os antigos, essa correlação é realizada pelo método correlate pertencente a classe `WeightedBlobCorrelator`. Dentro do método `calculateConfidence`, comparado a posição somada a velocidade do blob antigo, tanto para sua posição em “x” quanto em “y”, o resultado é reduzido da posição do novo blob, com esta informação é possível verificar se o antigo tem relação com o novo ou não.

3.2.2.1 Codecs

A interface `Effect` nativa do JMF, é usada para implementar unidades de processamento de mídias que recebem como parâmetros os buffers com os dados de entrada e saída e devem executar os processamentos necessários, como efeitos e filtros e ao final devolve um buffer com a nova mídia.

A figura 16 exibe o diagrama de classes partindo da classe `CodecVideo`, onde está é responsável por implementar a interface `Effect`. A classe `CodecVideo` possui um método abstrato `processRGB`, para que as classes filhas possam implementar seus filtros dentro da implementação deste método, onde ele é chamado de dentro do método `process` da classe `CodecVideo`, este método é responsável por trabalhar *frame* a *frame*. Com isso o método

`processRGB`, acaba se tornando o principal método de cada subclasse da `CodecVideo`, pois nele é aplicado os filtros e técnicas da responsabilidade de cada subclasse.

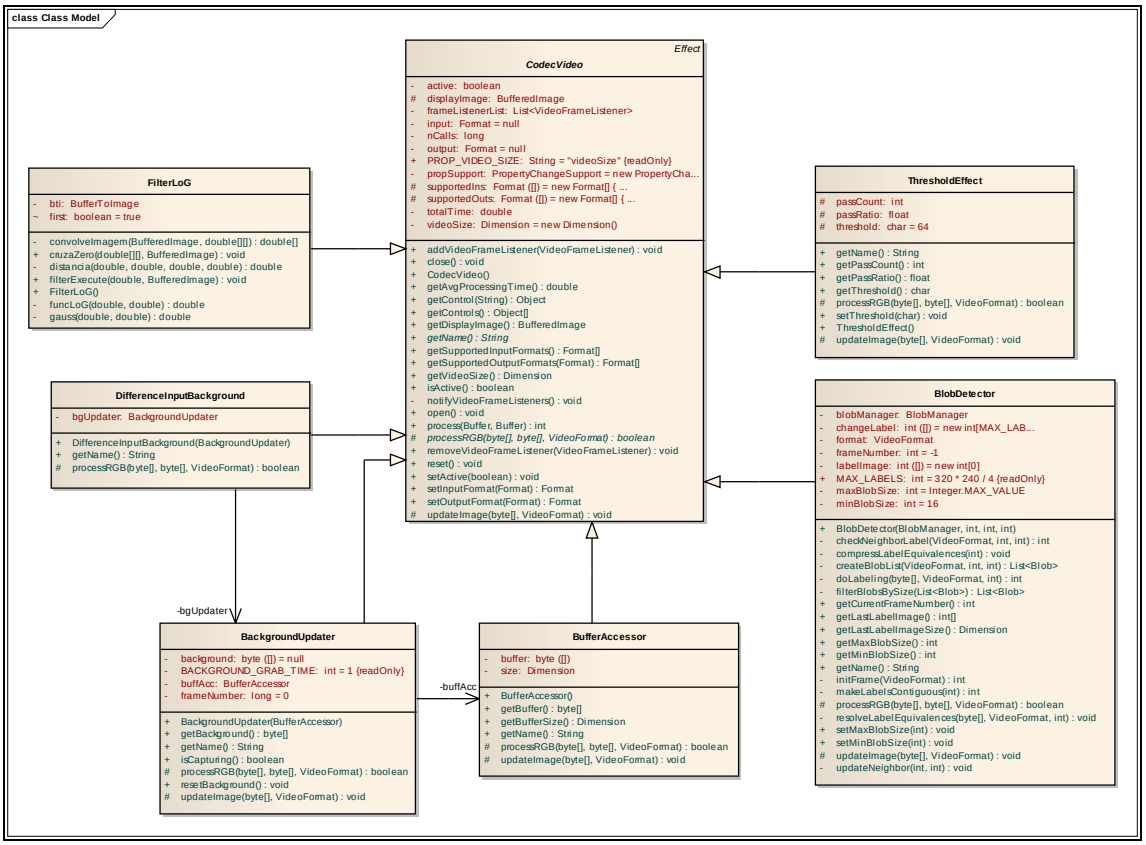


Figura 16 – Diagrama das classes descendentes de CodecVídeo

Na classe `FilterLog` é realizada as implementações da filtragem laplaciana da gaussiana através do método `filterExecute()`. A implementação deste filtro está descrito na seção 2.4.

Como os buffers que entram e saem dos codecs não podem ser armazenados em um cache, então é usada a classe `BufferAccessor` para realizar a copia quadro a quadro do buffer de entrada e dispoibliza-lo. Este novo buffer é utilizado pela classe `BackgroundUpdater` que é responsável por atualizar o fundo da imagem.

O fundo atualizado agora é utilizado pela classe `DifferenceInputBackground`, onde irá verificar se existe alguma diferença entre o frame de entrada e o frame contendo o fundo (background) da cena.

Para ser realizado o método do `threshold`, é utilizada a classe `ThresholdEffect`.

A classe `BlobDetector` é utilizada para reconhecer o movimento, e criar a lista com os

novos blobs através do método `creatBlobList`, após a criação da lista, é passado um filtro para manter apenas os blobs que obedeça o tamanho mínimo e máximo estipulado para o blob.

3.2.3 DIAGRAMA DE SEQUÊNCIA

A seguir são descritos os diagramas de sequência do protótipo, eles foram divididos em dois diagramas menores para facilitar a sua interpretação.

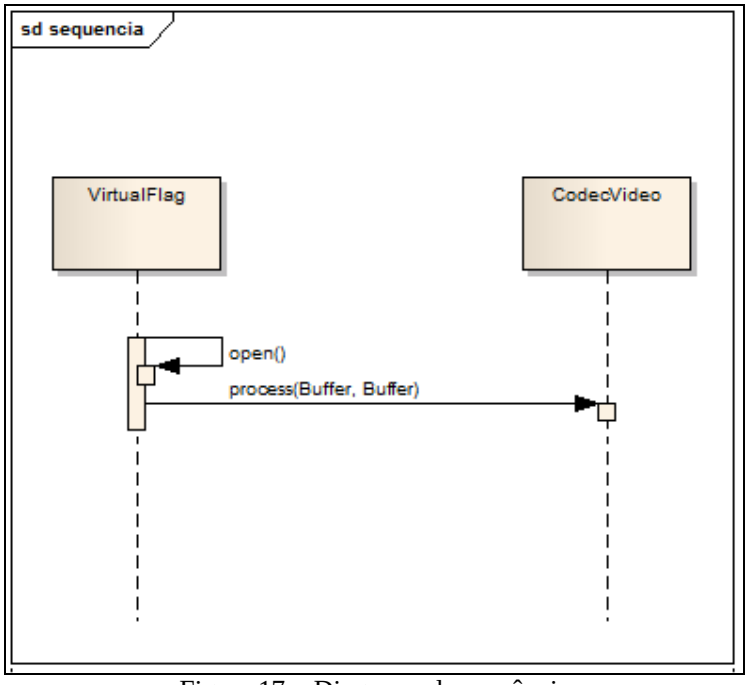


Figura 17 – Diagrama de sequência

A figura 17 demonstra a primeira parte do diagrama de sequência da aplicação, a classe `VirtualFlag` utiliza a classe `CodecVideo` para processar os frames separadamente do vídeo, e gerar o filtros necessários nos *frames*. Para demonstrar a sequência de funcionamento dos processos do codec, é utilizado o diagrama da figura 18.

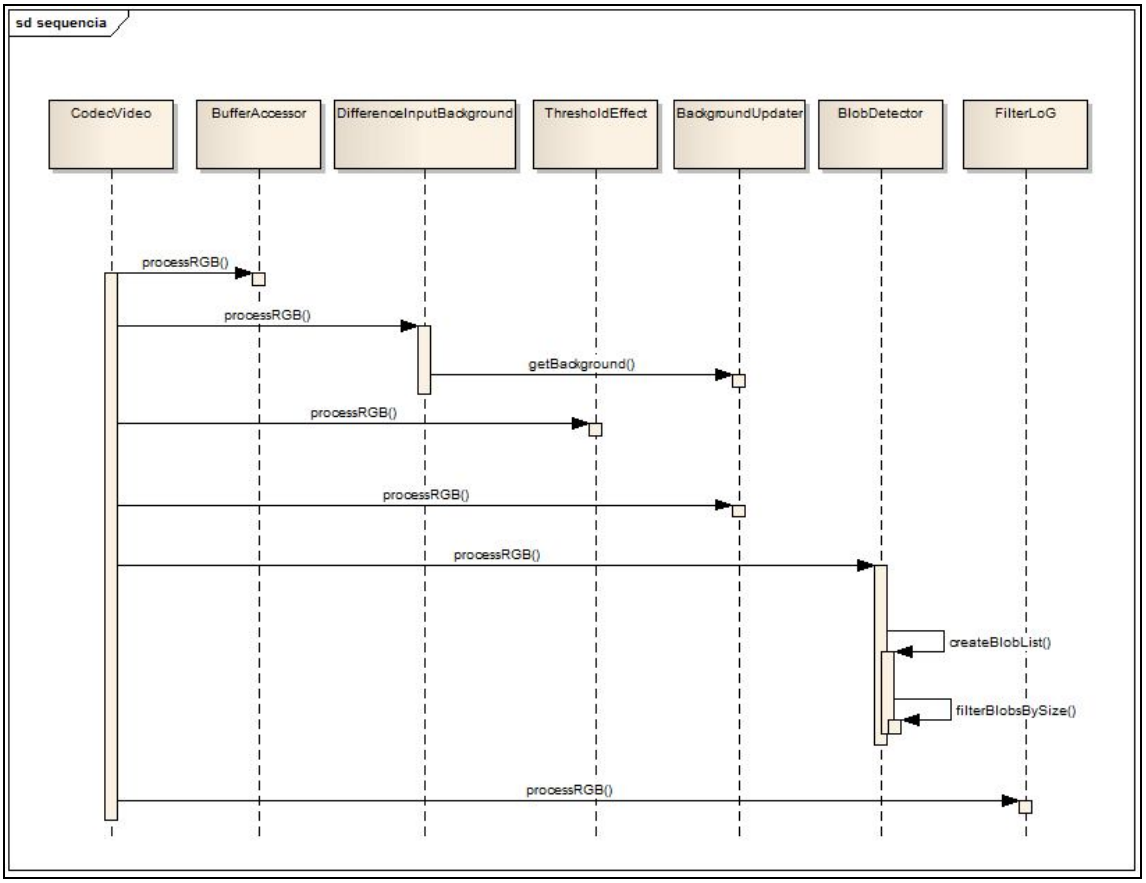


Figura 18 – Diagrama de sequência dos codecs

As classes `BufferAccessor`, `DifferenceInputBackground`, `ThresholdEffect`, `BackgroundUpdater`, `BlobDetector` e `FilterLog` são herdadas da classe `CodecVideo`. Estas classes são adicionadas a um array de codecs, onde este array é setado pelo método `setCodecChain` para dentro do `TrackControl`, assim se tornam pertencentes a lista de codecs, onde a API do JMF se encarrega de executa-las uma vez a cada *frame*, como todas implementam o método `proccesRGB`, este é tem sua execução solicitada através do método `process` da classe `CodecVideo`.

3.3 IMPLEMENTAÇÃO

Nesta seção serão apresentadas as considerações sobre a implementação do protótipo, as técnicas e ferramentas utilizadas, bem como a operacionalidade da implementação.

3.3.1 Técnicas e bibliotecas utilizadas

Para a desenvolvimento deste trabalho foi utilizada a linguagem Java na versão 1.6 com a API Java Media Framework (JMF). A utilização do JMF se da, pois ela é distribuída gratuitamente pela Oracle (2011), é uma API específica para o tratamento de vídeo e áudio, tendo suporte para diversos formatos tanto de vídeo como de áudio.

O ambiente utilizado para o desenvolvimento deste protótipo foi o Eclipse Plataform com a versão Helios (ECLIPSE, 2011).

O método `process(Buffer in, Buffer out)` da classe `CodecVideo` é um método nativo da API do JMF e um dos principais métodos deste protótipo, pois ele é responsável por realizar todo o processamento *frame* a *frame*. É neste método que é aplicado ou realizada a chamada para serem aplicados a todos os filtros necessários gerando um *frame* final que será exibido ao usuário.

A preparação da mídia para ser utilizada através dos métodos da API JMF, é realizada através do método `open()`, implantado na classe `VirtualFlag`. Dentro deste método adicionamos a lista de codecs que irão trabalhar em cima do arquivo de vídeo. Estes codecs serão responsáveis de trabalhar frame a frame do vídeo, podendo realizar as alterações necessárias nestes frames.

O método responsável por trabalhar com o *frame* é o `process()`, sendo que este é um método descendente da classe nativa do JMF Effect. Este método recebe como parâmetro dois buffers com os *frames*, um sendo o *frame* de entrada e o outro o de saída. Para se trabalhar com estes *frames* inicialmente é realizada a transformação do buffer inicial e final em um array de bytes que serão manipulados.

A rotina desenvolvida para realizar a conversão do buffer para o array de bytes é realizada pelo método `process()` da classe `CodecVideo`, e estre trecho do método é apresentado no quadro 4.

```
byte[] bin;

if (in.getData() instanceof byte[]) {
      bin = (byte[]) in.getData();
} else if (in.getData() instanceof int[]) {
      int[] iin = (int[]) in.getData();
      bin = new byte[iin.length * 3];
      int bi, ii;
      for (bi = 0, ii = 0; bi < bin.length; bi += 3, ii++){
            int v = iin[ii];
            bin[bi + 2] = (byte) (v & 0xff);
            bin[bi + 1] = (byte) ((v >> 8) & 0xff);
            bin[bi] = (byte) ((v >> 16) & 0xff);
      }
}else {
      return PlugIn.BUFFER_PROCESSED_FAILED;
}
```

Quadro 4 – Gerando uma cópia do *frame*

Para ser realizada esta conversão inicialmente é instanciado um novo array que conterá os bytes de entrada. Na sequencia é realizado um teste para verificar se o retorno do método `getBytes()`pertencente ao Buffer, é um array de bytes ou de `int`, caso, seja bytes, apenas adiciona um array sobre o outro. Caso contrario, é verificado se é um array do tipo primitivo `int`, então dever ser convertido, cada valor inteiro em um byte. Como um valor `int` ocupa três bytes, o tamanho do novo array tem que ser 3 vezes maior, para poder suportar, para poder realizar a divisão um valor `int` nestas três partes, é utilizado o deslocamento de bits, onde para o primeiro byte, é deslocado 16 casas, para o segundo 8 e para terceiro, não é descolado nenhum.

Outra parte importe da implementação do método `process()` é a chamada que ele realiza para o método `processRGB()`, para a classe `CodecVideo`. Este é um método abstrato, ou seja, não possui implementação, passando esta responsabilidade para as classes descendentes de `CodecVideo`, que implementam este método para realizar seus filtros. Isso permite que cada filtragem possa ser separada em classes diferentes, e trabalharem sobre o mesmo *frame*.

A classe `FilterLoG`, que estende a `CodecVideo`, é responsável por realizar o realce das linhas do campo, para um posterior reconhecimento das linhas.

Para realização o realce das linhas, o primeiro passo é transformar a imagem para monocromática e realizada a sua transformação negativa, o próximo passo é realizar a detecção das bordas do campo. A realização desta detecção foi implementada através do filtro Marr e Hildreth, que tem seu desenvolvimento descrito por Miranda (2008), com algumas adaptações para este trabalho. O objetivo de usar este filtro foi pelo fato dele implementar o filtro laplaciano da gaussiana de forma eficiente. Este filtro é executado pelo método

`filterExecute()` da classe `FilterLoG`, e pode ser visualizado pelo quadro 5.

```
public void filterExecute(double s, BufferedImage src) {
      int width;
      int k;
      int n;
      int nLin = src.getHeight();
      int nCol = src.getWidth();;
      double[][]  amostraLoG; // amostra bidimensional do Laplaciano do
                                 Gaussiano
      double[][] arrayLaplace;
      WritableRaster srcWR = src.getRaster();

      // Cria uma Gaussiana e um filtro com a derivada da Gaussiana
      width = (int)(3.35*s + 0.33);
      n = 2*width + 1;

      amostraLoG = new double[n][n];
      for (int i = 0; i < n; i++){
            for (int j = 0; j < n; j++){
                  amostraLoG[j][i] = funcLoG(distancia((double)i,
                    (double)j, (double)width, (double)width), s);
            }
      }
      // Convolução da imagem fonte com a Gaussiana nas direções X e Y
      arrayLaplace = convolveImagem(src, amostraLoG);

      // Localiza os cruzamentos zeros
      cruzaZero(arrayLaplace, src);

      // Limpa as bordas
      for (int i = 0; i < nLin; i++) {
            for (int j = 0; j <= width; j++)
                  srcWR.setSample(j, i, 0, 0);
            for (int j = nCol - width - 1; j < nCol; j++)
                  srcWR.setSample(j, i, 0, 0);
      }
      for (int j = 0; j < nCol; j++) {
        for (int i = 0; i <= width; i++)
                  srcWR.setSample(j, i, 0, 0);
            for (int i = nLin - width - 1; i < nLin; i++)
                  srcWR.setSample(j, i, 0, 0);
      }
}
```

Quadro 5 – Método `filterExecute()`

O método `funcLoG` chamado para montar o array com uma amostra do LoG, este método é o responsável por implementar a equação laplaciana da gaussiana, o seu desenvolvimento pode ser visto no quadro 6.

```
private double funcLoG(double r, double sigma) {
      double x1;
      x1 = gauss(r, sigma);
      return (r*r-2*sigma*sigma)/(sigma*sigma*sigma*sigma) * x1;
}

private double gauss(double r, double sigma) {
      return Math.exp(((-r*r)/(2*sigma*sigma)));
}
```

Quadro 6 – Implementação da equação laplaciana da gaussiana.

Após a criação do *array* com a amostra do LoG, este é utilizado sobre o Buffer da imagem original, para a obtenção de uma nova imagem, já com o filtro pronto, a junção do array do Log, com o *Buffer* da imagem é realizado pelos métodos `convolveImagem` e `cruzaZero`.

O ultimo passo desenvolvido, já com o *Buffer* da imagem após o filtro LoG, é aplicar o filtro *threshold* para remover as imperfeições finais e tornar a imagem binaria, pois os pixels de interesse ficam em branco e os demais em preto. Para a execução deste, foi criado outro codec, pois como o *threshold* é utilizado em mais métodos deste protótipo, não poderia ficar amarrado a um único codec. Para isso foi criada a classe `ThresholdEffect`. Esta classe também é descendente da `CodecVideo`, por isso a implementação do *threshold* é realizada através do método `processRGB()`, que é demostrada pelo quadro 7.

```
for(int i = 0; i < bin.length; i+=3, p++) {
      if ((char) bin[i] > threshold || (char) bin[i+1] >
         threshold || (char) bin[i+2] > threshold) {
            bout[p] = (byte) 255;
            passCount++;
      } else {
            bout[p] = 0;
      }
}
```

Quadro 7 – Implementação do filtro *threshold*.

O codec `BlobDetector` é responsável por detectar as áreas em movimento, e delimitar estas áreas, que serão os blobs. A delimitação destas áreas é uma das partes realizadas pelo método `createBlobList()` chamado de dentro do `processRGB()`. O algoritmo utilizado para delimitar as áreas dos blobs está descrito no quadro 8.

```
i = format.getSize().width;
for(y = 1; y < format.getSize().height; y++) {
      i++;
      for(x = 1; x < format.getSize().width - 1; x++, i++) {
            if (labelImage[i] != 0) {
                  int index = labelImage[i] - 1;
                  Blob blob = blobs.get(index);
                  ++blob.pixelCount;
                  if (x < blob.bounds.x)
                        blob.bounds.x = x;
                  if (y < blob.bounds.y)
                        blob.bounds.y = y;
                  if (x > blob.bounds.x + blob.bounds.width)
                        blob.bounds.width = x - blob.bounds.x;
                  if (y > blob.bounds.y + blob.bounds.height)
                        blob.bounds.height = y - blob.bounds.y;
            }
      }
      i++;
}
```

Quadro 8 – Implementação da delimitação da área do blob.

Após montar a lista com todos novos blobs, encontrados, deve-se verificar uma correlação entre os novos blobs e os antigos, para detectar o que continua em movimento e o que não se moveu mais. Para se fazer esta correlação deve-se verificar se inicialmente é montada uma matriz de confiança desta relação entre os blobs novos e velhos, como pode ser vista no quadro 9.

```
private double calculateConfidence(Blob newBlob, Blob oldBlob) {
      double predictedX = oldBlob.bounds.getCenterX() +
         oldBlob.velocity.x;
      double predictedY = oldBlob.bounds.getCenterY() +
        oldBlob.velocity.y;
      double dx = newBlob.bounds.getCenterX() - predictedX;
      double dy = newBlob.bounds.getCenterY() - predictedY;
      double deltaDist = Math.sqrt(dx*dx + dy*dy);   // leave
        distance squared?

      double dw = Math.abs(newBlob.bounds.getWidth() –
        oldBlob.bounds.getWidth());
      double dh = Math.abs(newBlob.bounds.getHeight() –
        oldBlob.bounds.getHeight());

      double c =
            infoWeight[Info.POSITION.ordinal()] * deltaDist +
            infoWeight[Info.SIZE.ordinal()] * (dw + dh);
      return -c;
}
```

Quadro 9 – Populando a matriz de confiaça entre os blobs

3.3.2 Operacionalidade da implementação

Nesta seção é apresentada a operacionalidade do protótipo, onde para serem demostrados foram desenvolvidos dois *players*, cada qual, com uma parte do desenvolvimento. Optou-se por desenvolver dois protótipos distintos para serem demostradas partes distintas do desenvolvimento, como este protótipo ainda está em uma fase de estudo, e cada uma destas partes não são dependentes uma da outra, fica mais visível os resultados atingidos. A primeira parte foi o desenvolvimento do acompanhamento dos movimentos através do uso do *tracking*. A segunda parte é o desenvolvimento são os passos iniciais para o reconhecimento das linhas do campo, onde foi desenvolvido até o realce das linhas.

Ao executar ambos os protótipos são exibidos uma tela inicial onde se deve selecionar um arquivo de vídeo, a tela inicial do protótipo pode ser vista na figura 19.

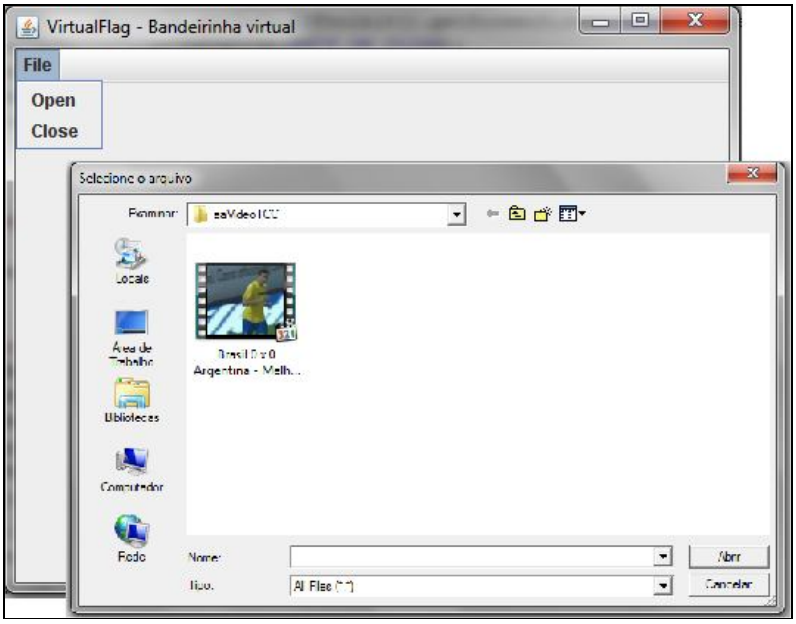


Figura 19 – Tela inicial

Após a seleção do arquivo de vídeo desejado, são realizados todos os filtros necessários de acordo com o protótipo executado.

A figura 20 é uma sequência de imagens com o resultado do acompanhamento de movimentos realizado, onde as regiões em que foi detectado algum movimento são demarcadas para que se possa mostrar o acompanhamento do movimento dinamicamente

durante a cena. Estas áreas são descritas no desenvolvimento como *blobs*. Esta nomenclatura foi amplamente encontrada na literatura, para descrever este tipo de região. Como pode ser observado, existem objetos fixos que estão sendo marcados como se estivessem em movimento, isto ocorre quando existe uma deslocação da câmera, gerando uma diferença entre o *frame* anterior e o atual na região do objeto fixo, com isto o algoritmo interpreta que ocorreu algum movimento na região. Para tentar amenizar estes fatos, o autor implementou um filtro que elimina os *blobs* que não apresentarem movimento por um determinado tempo, assim, durante uma cena, em que a câmera não se move estas regiões são descartadas.

Figura 20 – Execução do *tracking*

Para que se possa ser reconhecido quando é encontrado um movimento novo, o *blob* foi demarcado em azul, e quando o movimento já esta sendo acompanhado, ele é demarcado em amarelo.

O segundo protótipo desenvolvido, demostra a parte inicial do desenvolvimento necessário para efetuar o reconhecimento das linhas do campo. Este processo foi realizado até o realce das linhas que pode ser observado na figura 21.

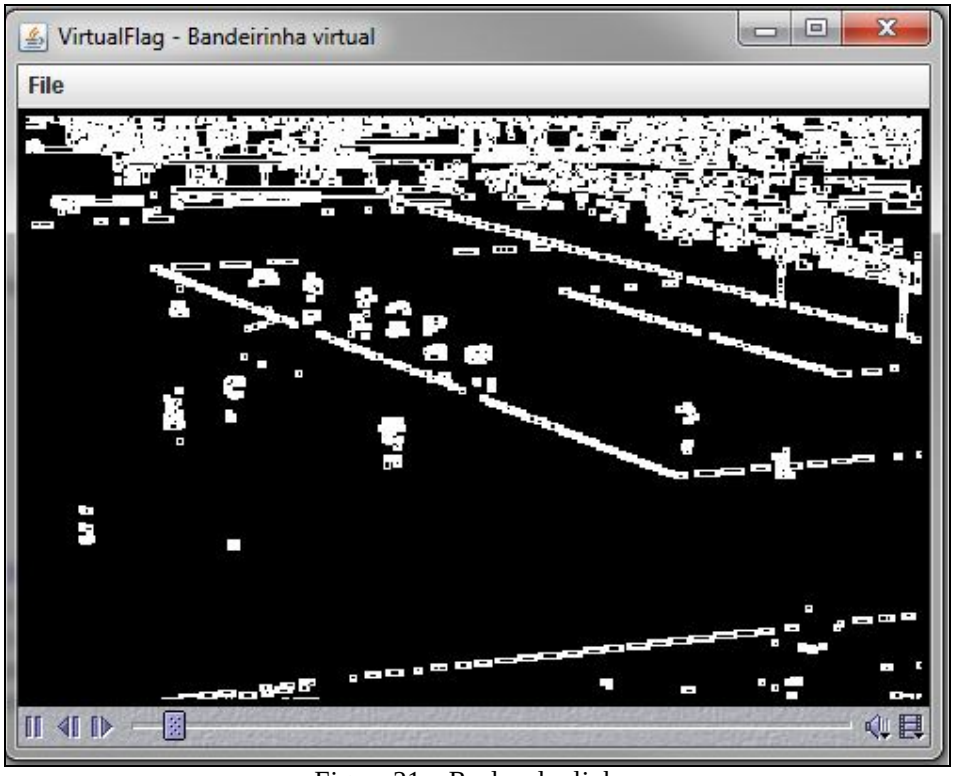


Figura 21 – Realce das linhas

Como pode ser visto na imagem a execução deste vídeo é toda em preto e branco, isto se da pela detecção dos pontos de borda da imagem, onde é removido todo o resto da imagem.

3.4 RESULTADOS E DISCUSSÃO

Foi realizado o rastreamento de objetos em movimento com uma visualização dinâmica, onde foram demarcadas as regiões onde teve algum movimento detectado, porem, se faz necessário a realização de um refinamento no algoritmo, pois estão sendo detectados movimentos de objetos que não são de interesse, como as placas publicitarias, as traves do gol, e em alguns casos a torcida.

Notou-se durante os testes que para o vídeo ter uma melhor qualidade quando é realizado o acompanhamento das cenas é aconselhado utilizar um vídeo com a um *codec* no formato MJPEG, isto ocorre pelo fato do algoritmo realizar a transformação dos dados de um frame para um *array* de bytes de uma forma mais otimizada quando o vídeo está neste formato. Foi possível utilizar outros formatos de *codecs*, porem a as cores e a qualidade da

imagem obtida durante a reprodução do vídeo não tiveram bons resultados, o que prejudicou o reconhecimento dos movimentos, pois acabou identificando falhas geradas no vídeo.

O próximo passo a ser desenvolvido para se obter a cálculo da linha de impedimento é agregar o cálculo de homografia e reconhecimento de linhas realizado nos trabalhos do Starosky (2003) e do Cristofolini (2004), para então se chegar ao desenho dinâmico de linhas de impedimento nos vídeos de jogos de futebol.

Foram desenvolvidos alguns testes de performance a fim de verificar se é viável o desenvolvimento de tal sistema. Verificou-se que no protótipo referente à identificação do movimento rodou a uma velocidade média 28 FPS, e o protótipo que realiza o realce de linhas, teve um desempenho de 23 FPS, o que revela que é necessário algum refinamento para se atingir o valor desejado que é 30 FPS, porém os valores foram satisfatórios.

O presente trabalho abordou a etapa inicial deste problema, sendo que sua continuidade abre novas perspectivas de pesquisa e desenvolvimento nesta área, no âmbito do curso de Bacharelado em Ciência da Computação da FURB.

4 CONCLUSÕES

Este trabalho realizou um estudo para o desenvolvimento de uma aplicação para gerar o cálculo de impedimento de forma dinâmica durante vídeos de uma partida de futebol, onde foram implementados na forma de dois protótipos as etapas de filtragem de imagem e realce de linhas, e detecção do movimento em cena, porém a pesquisa para a implementação das demais etapas continua em desenvolvimento.

Existe a necessidade de ser realizado algum refinamento na referente ao desempenho dos protótipos para atingirem a média de 30 FPS por segundo, porem foram atingidos valores próximos, chegando a ter uma média de 28 FPS no protótipo de detecção de movimentos.

Outra questão de suma importância é a acurácia do rastreamento e a precisão dos cálculos de distâncias feitos após a calibração de câmeras.

Em relação aos trabalhos correlatos, o presente trabalho abordou a questão de rastreamento para que futuramente a solução deste problema possa ser agregada aos trabalhos de cálculo de homografia e calibração de câmera já realizados (STAROSKY, 2003) e (CRISTOFOLINI, 2004).

4.1 EXTENSÕES

Como sugestão natural para extensões seria concluir a inserção da linha de impedimento. Outra sugestão viável seria aplicar estes conceitos para fazer um rastreamento do jogador pelo campo, onde poderia ser verificada a distancia percorrida, velocidade alcançada, e com isso retirado varias estatísticas no decorrer da partida.

Este tipo de rastreamento poderia ser utilizado também em outros esportes como basquete ou futebol de salão, por exemplo, que por terem uma qualidade de vídeo bastante regular devido a possuírem iluminação padrão, facilitaria a interpretação dos segmentos de reta.

REFERÊNCIAS BIBLIOGRÁFICAS

CÉSAR JUNIOR, Roberto M.; COSTA, Luciano F. **Shape analysis and classification: theory and practice**. Boca Raton: CRC Press, 2001.

CONFEDERAÇÃO BRASILEIRA DE FUTEBOL. **Regras oficiais de futebol**. Rio de Janeiro: Sprint, 2010. Disponível em: <http://www.cbf.com.br/media/58890/livro_de_regras_2010_2011.pdf>. Acesso em: 29 mar. 2011.

CRISTOFOLINI, Diogo. **Protótipo de um ambiente virtual tridimensional para utilização no cálculo de impedimento de jogadores de futebol**. 2004. 51 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

DENAMN, Simon; CHANDRAN, Vinod; SRIDHARAN, Sridha. **An adaptive optical flow technique for person tracking systems.** Brisbane, 2007. Disponível em: <http://eprints.qut.edu.au/14756/1/14756.pdf>. Acesso em: 29 nov. 2011.

ECLIPSE. **Eclipse**. 2011. Disponível em: <http://www.eclipse.org/>. Acesso em: 05 jul. 2011.

FURLAN, Davi. **Modelagem de objetos através da UML –** the unified modeling language. São Paulo: Markron Books, 1998. 329 p.

GONZALEZ, Rafael C.; WOODS, Richard E. **Processamento de imagens digitais**. Tradução de Roberto Marcondes Cesar Junior, Luciano da Fontoura Costa. São Paulo: Edgard Blücher, 2008.

MARQUES FILHO, Ogê.; VIEIRA NETO, Hugo. **Processamento Digital de Imagens.** Rio de Janeiro: Brasport, 1999. 307 p.

MIRANDA, José I. **Implementação Java do filtro de Marr e Hildreth para detecção de bordas**. 2008. Disponível em< http://www.infoteca.cnptia.embrapa.br/handle/doc/31739>. Acesso em: 16/10/2011.

ORACLE. **Oracle**. 2011. Disponível em: <http://www.oracle.com>. Aceso em: 02 nov. 2011.

PINHO, Márcio S. **Manipulação de Imagens**. Porto Alegre: 2011. Disponível em: < http://www.inf.pucrs.br/~pinho/CG/Aulas/Img/IMG.htm>. Acesso em: 20 set. 2011.

ROSA, André L. B. **Sistema de *tracking* de objetos a partir de várias câmeras.** 2010. 48 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SANTOS, Daniel. **Sistema óptico para identificação de veículos em estradas.** 2008. 65 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

STAROSKY, Maiko. **Calibração de câmeras para utilização no cálculo de impedimentos de jogadores de futebol a partir de imagens**. 2003. 55 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SZENBERG, Flávio. **Acompanhamento de cenas com calibração automática de câmeras.** 2001. 144 f. Tese (Doutorado em Ciências em Informática) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro. Disponível em: <http://www.tecgraf.puc-rio.br/~szenberg/doutorado>. Acesso em: 03 abr. 2011

TYAGI, Ambrish et al. **Fusion of multiple camera views for kernel-based 3D tracking**. [S.l.], 2007. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.5917&rep=rep1&type=pdf>. Acesso em: 03 abr. 2010.