UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Sistema autônomo de processamento de imagem para *tracking* de jogadores em partidas de tênis

Autor: Igor Hueb de Castro

Orientador: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos

2015

Igor Hueb de Castro

Sistema autônomo de processamento de imagem para tracking de jogadores em partidas de tênis

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica - Ênfase em Eletrônica

ORIENTADOR: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos
2015

C
354s

Castro, Igor Hueb de
Sistema autônomo de processamento de imagem para tracking de jogadores em partidas de tênis / Igor Hueb de Castro; orientador Evandro Luis Linhari Rodrigues. São Carlos, 2015.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2015.

1. Processamento de Imagens. 2. Visão Computacional. 3. esporte. 4. Engenharia Esportiva. I. Título.

FOLHA DE APROVAÇÃO

Nome: Igor Hueb de Castro

Título: "Sistema autônomo de visão computacional para aquisição de informações em partida de tênis"

Trabalho de Conclusão de Curso defendido e aprovado em 19 / 06 / 2015,

com NOTA 6,2 (seis, dois), **pela Comissão Julgadora:**

Prof. Associado Evandro Luís Linhari Rodrigues - (Orientador - SEL/EESC/USP)

Mestre André Luís Martins - (Doutorando - SEL/EESC/USP)

Prof. Dr. Marcelo Andrade da Costa Vieira - (SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

Dedicatória

Dedico esse trabalho para meus avós, que sempre acreditaram nos sonhos dos netos.

Igor Hueb de Castro.

Agradecimentos

Ao meu orientador Prof. Dr. Evandro Luís Linhari Rodrigues por acreditar nessa ideia e na minha capacidade.

À minha família, por tudo que sempre fizeram.

À república 29, por ser minha casa longe de casa.

À Atlética CAASO e ao handebol CAASO, por terem me tornado a pessoa que sou hoje.

Ao Bruno Gregório, meu amigo, que me ajudou com todas as dúvidas na área.

À Vitória Andreo, por sempre me ouvir e ajudar com tudo.

Igor Hueb de Castro.

"The ultimate measure of a man is not where he stands in moments of comfort and convenience, but where he stands at times of challenge and controversy."

Martin Luther King Jr.

Resumo

Conforme o valor dos investimentos em esportes aumenta, também aumenta sua visibilidade e o interesse por novas tecnologias que permitem com que os atletas possuam certa vantagem sobre seus competidores. A visão computacional vem sendo usada ativamente nessa atividade, já que proporciona uma forma não invasiva de aquisição de informação. Esse trabalho propõe um método para identificação de posição dos jogadores por meio do processamento de vídeos de partidas de tênis, considerando as restrições de sistemas embarcados, o que limita as decisões em tamanho de código, espaço de memória necessário e tempo de processamento. O método foi construído com algoritmos de processamento de imagem utilizando rotinas do OpenCV em C++, para que os resultados fossem melhor observados e utilizados foi proposta a inclusão de uma transformação de perspectiva para apresentação do posicionamento dos jogadores. Os algoritmos construídos apresentaram baixa demanda de recursos computacionais o que estimula sua implementação futura em sistemas embarcados. Os resultados de detecção de posição alcançados atingiram os objetivos propostos apresentando erros inferiores a 50 cm. Essa opção oferece a oportunidade de implementação futura de análises de informações a respeito dos atletas na partida.

Palavras-Chave: Visão computacional, esporte, processamento de imagens, engenharia esportiva.

Abstract

As the amount of money applied to sports increases, their visibility also increases together with the need for new technologies that can give athletes an edge in competition. Computer Vision has become a good alternative for development in the area, as it do not depend on sensors used on clothing. The focus of this work is to propose a method of player tracking through the analysis of video from tennis matches, considering the restrictions in embbeded systems, that limitate the size of code, memory and processing time. The method was constructed with algoritms of image processing using OpenCV rotines in C++, in order to improve the display and use of the information, a perspective transform was used. The developed algoritm presented low demand of computing resources that helps the future implementation in embbeded systems . The results in player detection reached the objectives with an error smaller than 0,5m. This option offers the future opportunity of implementation in a system of infering information about the players.

Keywords: Computer vision, sports, image processing, sports engineering tennis.

Lista de Figuras

Lista de Tabelas

Siglas

SVM	*Support Vector Machine* - Máquina de suporte com vetores
HOG	*Histograms of Oriented Gradients* - Histograma de gradientes orientados
MBR	*Minimum Bounding Rectangle* - Retângulo mínimo de contenção
Pixel	*Picture element* - Elemento de imagem

Sumário

Capítulo 1

Introdução

Cada vez mais a quantidade de dinheiro que vem sendo aplicada em esportes é maior, seja considerando premiações, salários de jogadores, valores dos patrocínios dos times, e até a quantidade de dinheiro envolvido com as apostas. Assim, surge uma necessidade por parte dos times, atletas e treinadores de estarem sempre à frente de seus competidores, seja em relação a equipamentos, como tênis, uniformes, e também tecnologias, como sensores cardíacos e GPS.

A área da engenharia que busca desenvolver tecnologias para os esportes se chama Engenharia Esportiva, e ela vem sendo desenvolvida ao longo dos últimos anos nas suas mais diversas frentes, seja relacionado a transmissão de eventos esportivos, a auxilio de arbitragem ou até no desenvolvimento de produtos, tanto para o usuário casual como para o atleta de alto rendimento.

Em contrapartida, a área de visão computacional esta relacionada com a capacidade de máquinas em interpretar, automaticamente ou auxiliadas por um usuário, imagens dos mais diversos tipos. Essa área de estudo vem sendo aplicada aos mais diversos problemas, criando soluções para diferentes ambientes e condições.

Uma das principais características da visão computacional é sua capacidade de obter informações sem a utilização de sensores ligados ao corpo, o que permite simultaneamente, a captação de dados sem interferir diretamente no objeto medido além de que a distância que essa medida pode ser realizada é relativamente grande, quando comparada aos outros tipos de sensores utilizados, como de infravermelho ou mesmo RFIDs, que possuem um raio de atuação de poucos metros.

O fato da visão computacional possibilitar a análise por imagens, sem a necessidade de sensores ligados ao atleta é de vital importância para a área do esporte, já que os mesmos po-

dem gerar lesões em atividades de contato, como o Rugby, ou podem ser de difícil utilização em ambientes como uma piscina, no caso da natação e pólo aquático.

Outro fator importante a ser considerado é que a maioria das federações proibe a utilização de sensores ligados diretamente ao corpo do atleta, já que os mesmos poderiam transmitir informações mais específicas, como a taxa de batimento cardíaco do atleta, e mesmo algumas outras condições de fadiga muscular, que proporcionariam uma vantagem desleal para o mesmo. Como a visão computacional usa apenas imagens, que qualquer pessoa treinada poderia ver, a mesma não é considerada uma vantagem desleal no esporte.

Com base nos argumentos anteriores, fica claro que o papel da visão computacional na aquisição de informações para o esporte é que a mesma passe a ser uma forma de retirar a subjetividade da análise dos profissionais, permitindo que dados explícitos sejam comparados, como a distância percorrida por um atleta, ou até a máxima velocidade frontal e lateral que o mesmo consegue alcançar, permitindo assim que decisões mais consistentes possam ser tomadas.

O projeto consiste no desenvolvimento de um sistema de aquisição e interpretação de imagens de partidas esportivas, focando na posição dos jogadores e nas informações que podem ser inferidas a partir da mesma, como velocidade máxima, tempo de resposta, distância percorrida. O projeto foi desenvolvido levando em consideração que a realidade brasileira, em que os atletas jogam em diferentes ambientes e situações, estando sempre viajando e com necessidade das informações em tempo real, dessa forma o foco na criação foi para uma plataforma embarcada que utiliza apenas uma câmera.

O esporte a ser analisado nesse trabalho é o tênis, com a possibilidade de desenvolvimento futuro para outros esportes, como o futebol. Inicialmente foi considerado o tênis por uma variedade de motivos, entre eles é possível citar, a existência de apenas 2 jogadores, que permanecem durante a maior parte do jogo em regiões bem definidas e opostas, evitando assim que eles se sobreponham. Outro fator interessante é que o mesmo é dividido em pontos de curta duração, e após cada um deles, as posições iniciais dos atletas voltam a ser semelhantes. Além da facilidade de trabalhar com o esporte, as transmissões de tênis envolvem, em geral, apenas uma câmera posicionada em um ângulo oblíquo acima dos jogadores, fazendo com que seja relativamente fácil obter uma quantidade boa de casos de teste.

Outro objetivo claro do projeto é a identificação, dentre as diferentes técnicas abordadas, qual a que produz melhores resultados em uma situação de jogo, já que elas são usadas em situações do cotidiano, normalmente os movimentos das pessoas são bem comportados e

seguem padrões bem distintos. Há uma necessidade de entender inicialmente qual dessas técnicas produz melhor resultados, antes de implementá-la em um sistema mais complexos.

1.1 Objetivos

- Implementação de sistema de detecção de jogadores em partida de tênis.
- Implementação de sistema de pré processamento de imagens.
- Tornar o sistema autônomo, capaz de se calibrar automaticamente, sem operador.
- Analisar diferentes técnicas para realizar a detecção de jogadores
- Analisar a viabilidade de tornar o sistema embarcado, facilitando sua venda como um produto no mercado.

1.2 Relevância

A diferenciação desse trabalho para os desenvolvidos anteriormente e mesmo para os produtos apresentados no mercado é uma adequação maior para uma realidade esportiva brasileira, ou seja, permitir a criação futura de um sistema sistema embarcado que realize a detecção de jogadores, já que não serão todas equipes que o possuirão. O desenvolvimento focado em um futuro sistema embarcado trás uma inovação relevante para o mercado brasileiro.

1.3 Organização da Monografia

No capítulo 2 serão levantados todos os fundamentos teóricos utilizados na durante o desenvolvimento do projeto, além do estado da arte atual da tecnologia, incluindo todos os sistemas comerciais que são aplicados na área. No capítulo 3 serão apresentados os materiais e métodos utilizados nas diferentes etapas de desenvolvimento. No capítulo 4 serão apresentados os resultados para cada etapa de desenvolvimento. No capítulo 5 serão apresentadas todas as conclusões alcançadas durante o desenvolvimento do projeto.

Capítulo 2

Embasamento Teórico

2.1 Estado da Arte

De forma geral, apesar de ser possível conseguir informações sobre preço, hardware e condições de utilização dos sistemas que estão sendo vendidos no mercado hoje em dia, é muito difícil encontrar qualquer informação das técnicas utilizadas em seu desenvolvimento, sendo as mesmas protegidas por patentes e segredos empresariais.

2.1.1 Trabalhos Acadêmicos

Existem no mercado, hoje, algumas opções de empresas que fornecem serviços relacionados a visão computacional, tanto para a aquisição de informações durante as partidas, como forma de auxiliar na arbitragem ou mesmo na transmissão de TV dos esportes, mas da mesma forma, existe uma visível falta de trabalhos científicos que abordam esse assunto, de forma que conhecimento na área fica muito restrito a pessoas que trabalham ou trabalharam na área.

A principal pesquisa na área, feita por universidades, foi realizada na *École Polytechnique Federale de Lausane*, na Suíça, seu *Computer Vision Laboratory* vem contribuindo nos últimos anos com vários trabalhos, nos mais diversos níveis, relacionado ao posicionamento dos jogadores e bolas, além de informações que podem ser extrapoladas dessa análise. Boa parte da pesquisa com relação a identificação de jogadores esta sendo usada como bibliografia nesse trabalho.[1, 2, 3]

Outro trabalho muito relevante na área, e que trouxe muitas soluções para o projeto foi a dissertação de mestrado *"Automatic Feature Extraction From Tennis Videos for Content Based Retrieval"* desenvolvida por Thejaswi Hanumantha Raya, que trás soluções para muitos dos problemas abordados nesse trabalho, apesar de ser focado em extração e classificação

automática de clipes para a transmissões esportivas.[4]

No mercado esportivo, existem diversas empresas criando sistemas que utilizam visão computacional, mas cada um é específico para uma modalidade, portanto é necessário separar por esportes, antes de entender suas características. Como ponto em comum, todos sistemas apresentam uma quantidade relativamente grande de câmeras, de 8 a 12, de alta resolução, para poder fornecer dados o mais precisos possíveis, além de trabalharem de forma assistida, com um operador que corrige qualquer identificação errada do programa.

2.1.2 Hawkeye Innovations

Na área do tênis, a empresa que se destaca é a *Hawkeye Innovations* do grupo Sony, que possui uma tecnologia de auxílio de arbitragem muito utilizada em torneios internacionais, e amplamente reconhecida por quem costuma assistir transmissões do esporte. Na figura 2.1. é possível observar a saída do seu sistema, que proporciona um modelo em três dimensões da bola durante todo o ponto, permitindo que seja determinado se a bola tocou o solo dentro ou fora da quadra. A *Hawkeye* também possui um software de auxílio no treinamento esportivo do tênis, que identifica os movimentos dos membros do atleta, e os compara com um padrão, de forma a identificar pontos que podem ser melhorados.

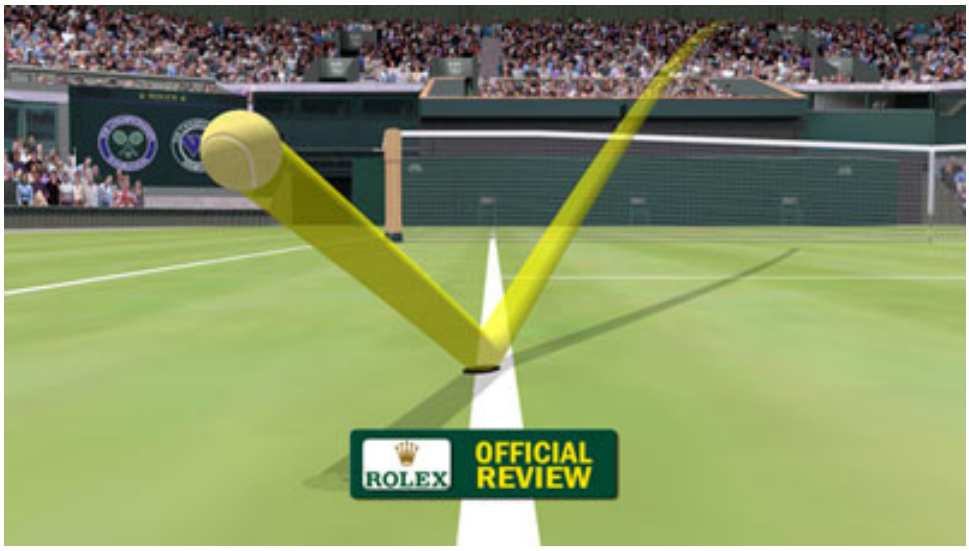

Figura 2.1: Saída de vídeo do sistema da *Hawkeye Innovations*.

2.1.3 Prozone

Quando é considerado o futebol, existem algumas alternativas que possuem diferentes soluções para a aquisição de dados. Ambas são baseadas na utilização de câmeras fixas, em

quantidades diversas. O sistema desenvolvido pela Prozone, empresa líder de mercado, é baseado na instalação de 8 câmeras fixas no estádio. A figura 2.2 apresenta a resposta apresentada pelo sistema da Prozone durante uma partida de futebol, esse sistema e reconhecido como o mais completo e com menores erros. Além de todo o software de detecção, muitas informações são interpoladas a partir da posição dos jogadores e bola, como velocidade máxima, esquema tático, pressão defensiva e muitos outros.

A instalação desse sistema tem um custo de aproximadamente 1 milhão de reais, devido ao alto grau de calibração das câmeras, e sua operação tem um custo mensal de 40 mil reais, se tornando assim, uma alternativa muito cara de produto, que não consegue suprir todo o mercado brasileiro.

Figura 2.2: Interface do sistema Prozone, realizando análise de posicionamento dos jogadores.

2.1.4 SportsVU

Outra empresa da área é a SportsVU, desenvolvedora do sistema STATS, fundada pelo americano David James, precursor da análise estatística no esporte, seu sistema é baseado na utilização de 3 câmeras, sendo cada uma voltada para uma região diferente do campo, como mostrado na imagem 2.3. O STATS apresenta menos dados que o sistema da Prozone,

porém o seu foco são as informações importantes para tornar uma transmissão esportiva mais relevante para os telespectadores, sendo menos utilizado por times profissionais, e mais por canais esportivos.

Figura 2.3: No detalhe, sistema de 3 câmeras usado no STATS.

2.2 Representação de Imagens

Na área da visão computacional, imagens são representadas por matrizes, de forma que o valor de cada elemento da matriz, a partir de agora chamado de *pixel*, representa sua cor. Essa conversão da imagem em *pixels* é feita logo após sua captura pelo sensores da câmera, e entender esse processo não é do interesse desse trabalho.

A matriz utilizada na representação possui duas ou três dimensões, sendo as duas primeiras coordenadas para a representação do *pixel* na imagem, e a terceira responsável por criar a percepção de cor.

Existem vários sistemas de representação de cor em visão computacional[6], quando é representada uma imagem em tons de cinza, a partir de agora chamada de *grayscale*, a matriz tem apenas duas dimensões, o valor de cada elemento representa sua cor, sendo o menor a

preta, e o maior a branca, com valores intermediários representando uma escala de cinza. Em outras representações de cores mais complexas, essa lógica geral se mantém.

Existem outras formas conhecidas de representação das cores, uma delas é o RGB, no qual, as matrizes tem três dimensões com a terceira delas possuindo 3 elementos, ou seja, a imagem é formada pela soma de três matrizes diferentes. Cada canal representa a quantidade de cor vermelha(*Red*), verde(*Green*) e azul(*Blue*) que somada, produzirá a cor desejada. O valor dentro de cada elemento da matriz, em cada canal, funciona da mesma forma que no *grayscale*.

Cada um desses sistemas de cores gera um espaço tridimensional, em que o valor(cor) do pixel é representado como uma posição, essa representação é importante para um entendimento claro da representação e pode ser utilizada para realizar comparações de forma mais eficiente.

Um outro sistema de representação de cores é o HSV, em que cada canal representa uma característica diferente da cor, a matiz(*Hue*) representa qual a tonalidade de cor esta sendo considerada, a saturação(*Saturation*) representa a pureza do tom e o valor(*Value*) define o brilho da cor. O espaço de cores HSV é muito utilizado na identificação de cores, para diferentes condições de iluminação.

2.3 Processamento de Imagens

Um dos focos da visão computacional é na identificação de objetos e regiões de interesse em imagens e vídeos, isso se dá a partir da separação de certas características únicas a elas nessas imagens, por exemplo, se procuramos uma bola de tênis na imagem, existem duas características únicas a ela em qualquer quadro considerado, a forma da bola, e sua cor característica. É necessário, por tanto, utilizar filtros, de forma que seja possível primeiro só visualizar os objetos amarelos e posteriormente procurar dentre esses objetos, apenas aqueles com a forma redonda.

Uma das formas de se segmentar o fundo dos objetos sendo considerados é a utilização de uma binarização adequada, a segmentação de uma imagem é uma transformação nos valores de cada *pixel* da mesma, de forma que eles assumam apenas os valores máximo e mínimo permitidos. A decisão de qual o valor do *pixel* após a transformação é feita a partir de um limiar(*threshold*), sendo que valores de pixel acima do limiar passam ao valor máximo e os a baixo passam para o mínimo.

Para a utilização da binarização, o ideal é que a imagem seja passada antes, por algum outro sistema que a transforme em níveis de cinza, essa transformação pode ocorrer de diferentes maneiras.

No projeto, a região de interesse são os pontos que estão em movimento em *frames* consecutivos da imagem(jogadores e bola), portanto foram utilizadas técnicas que separem regiões desse tipo. Uma delas é o filtro de diferenças, que consiste na subtração, em cada canal, de *frames* consecutivos para que identificar regiões em que ocorrerão movimentação.

Outro parâmetro que foi considerado durante o desenvolvimento do projeto foi a cor da roupa dos jogadores, que para a maioria dos esportes, é bem distinta e diferente do solo da quadra ou campo.

Outra necessidade desse trabalho é de seguir os jogadores em diferentes quadros, portanto é necessário não só identificá-los em todos os quadros, mas também conseguir distinguir a trajetória que esta sendo desenvolvida por eles. Para tanto, é importante conseguir segmentar a quadra em diferentes regiões, a partir de suas linhas. As linhas são separadas a partir de uma binarização, e o algoritmo de Hough[5] foi utilizado para detectar suas formas.

A transformada de Hough pode ser utilizado para detectar forma geométricas bem definidas, como circunferências e retas. Uma reta no espaço polar é definida pelo valor em módulo e ângulo que um vetor perpendicular a ela, passando pela origem assume, considerando um ponto no espaço retangular, infinitas retas passam por ele, e elas são representadas no espaço polar por uma senóide, como pode ser observado na parte direita da 4.3. Realizando esse processo para todos os *pixels* brancos de uma imagem binarizada, é possível identificar linhas a partir do cruzamento das senóides. Um dos parâmetros escolhidos é a quantidade de intersecções de curvas que definem uma reta.

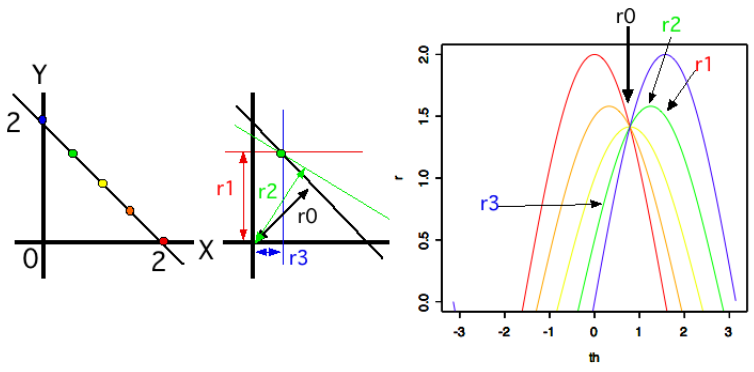

Figura 2.4: Transformada de Hough.

2.4 Transformações Lineares

Em visão computacional, muitas vezes é necessário realizar uma mudança de perspectiva, para que a imagem seja possível realizar medidas e comparações. Para isso é realizada uma transformação de perspectiva, a partir de uma matriz de transformação linear produzida por 4 pontos em 2 imagens distintas.

Além da transformação de perspectiva, é possível realizar uma transformação afim [6] com relação a rotação, translação e escala, podendo ser representadas pelas figuras 2.5 e 2.6, essas transformações são muito usadas para realizar correções de uma movimentação da câmera.

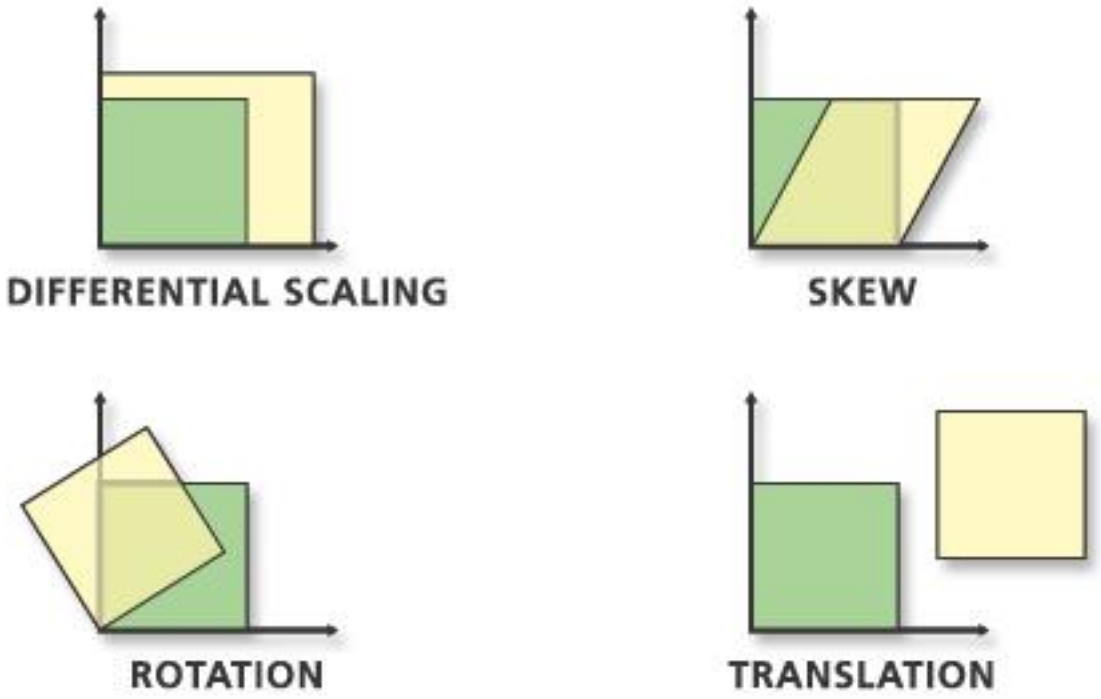

Figura 2.5: Resultados de diferentes tipos de transformação afim. Ex

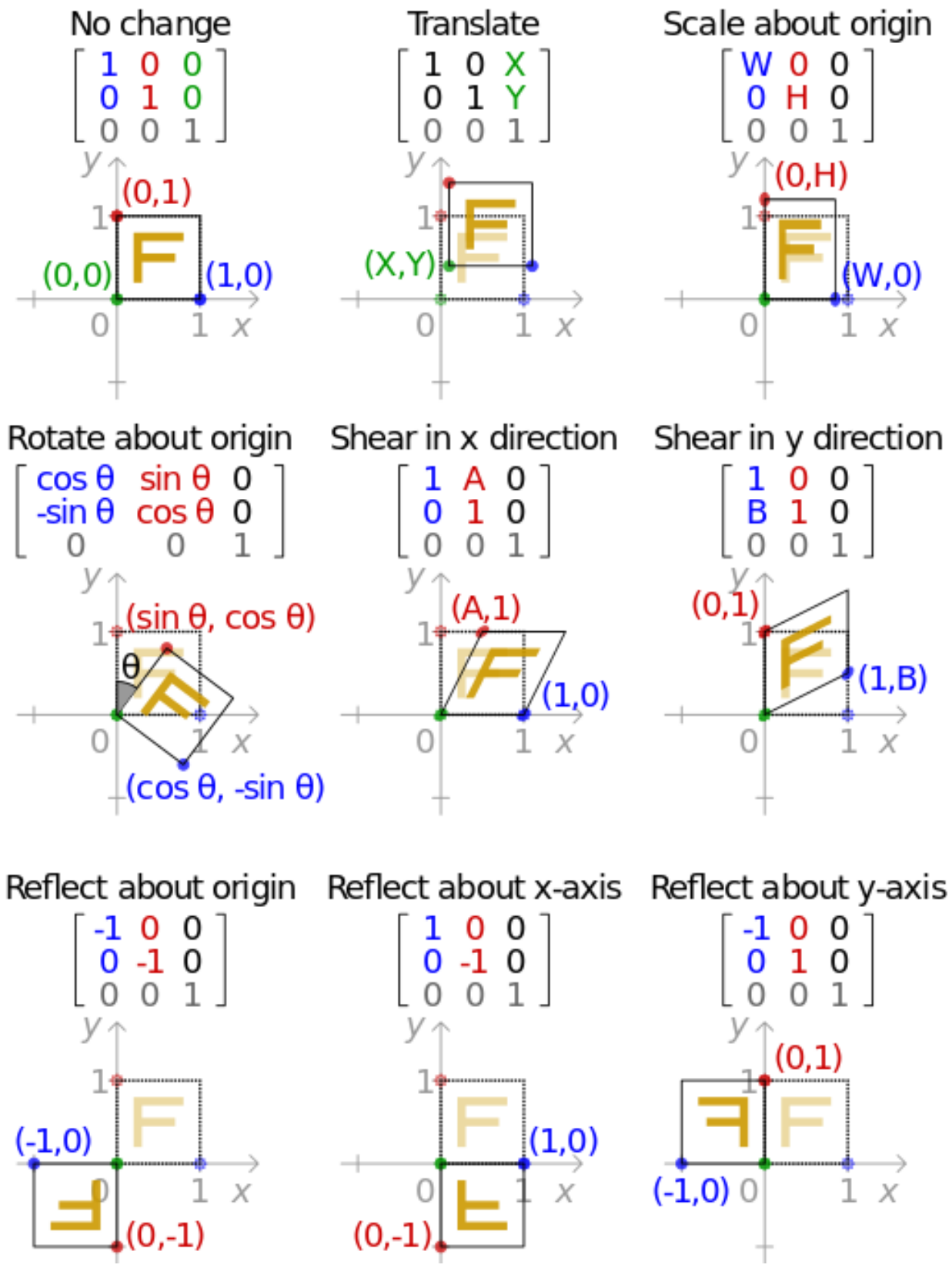

Figura 2.6: Exemplos de transformações afins e suas matrizes de transformação.

2.5 Características do Esporte

Outra necessidade desse trabalho é de seguir os jogadores em diferentes quadros, portanto é necessário não só identificá-los em todos os quadros, mas também conseguir distinguir a trajetória que esta sendo desenvolvida por eles.

Algumas pesquisas recentes indicam que não é necessária a identificação correta dos jogadores em todos os frames [3], mas sim um algoritmo capaz de entender que as pessoas não podem simplesmente desaparecer de um momento para o outro, permitindo assim identificar falsos negativos e coibir muitos dos erros cometidos na identificação e traçar trajetórias mais condizentes com a realidade.

Essas pesquisas entendem que a quantidade de *frames* para identificação é muito grande e consequentemente a probabilidade de um erro ocorrer nesse processo muito maior, portanto não pode ser permitido ao programa considerar todos os resultados obtidos nessa etapa como corretos sem antes ser feita uma estimativa de velocidade e trajetória, para ver se esta condizente com o esperado.

Algumas características dos jogos a serem considerados são importantes também, apesar da aparente movimentação aleatória dos jogadores durante uma partida, algumas considerações importantes podem ser feitas[7], de forma a simplificar e muito a análise. Em uma partida de futebol, dependendo do esquema tático, cada jogador costuma ocupar uma parte específica do campo na defesa e se movimentar para frente de uma determinada forma quando seu time esta com a posse da bola. Em uma partida de tênis, cada jogador se desloca sempre de forma a alcançar a bola antes que ela toque o piso a segunda vez consecutiva no seu lado da quadra.

Levando em consideração essas informações junto com as característica do desenvolvimento do sistema de visão computacional, é possível conseguir uma forma mais simples de obtenção da posição dos jogadores.

Capítulo 3

Materiais e Métodos

3.1 Especificações do Sistema

O trabalho foi testado e desenvolvido em um computador pessoal, um SONY Vaio PCG 41213, com 2GB de memoria RAM, e processador i3 de 2,10GHz, instalado com o sistema operacional Ubuntu 14.10 para 64bits .

O desenvolvimento foi feito em C++ com a biblioteca OpenCV, a escolha foi feita pela familiaridade do desenvolvedor com a linguagem e a grande quantidade de informações relacionadas a essa biblioteca na literatura, além da grande quantidade de funções utilizadas que já estão implementadas nela.

Uma rotina de detecção de movimento, desenvolvida em MATLAB, foi utilizada como parâmetro de comparação com relação a qualidade do trabalho desenvolvido. Toda a rotina de MATLAB utilizada pode ser encontrada no anexo 1.

3.2 Compartimentalização do Desenvolvimento

O desenvolvimento do projeto foi compartimentalizado, de forma que uma parte do projeto pôde ser desenvolvida sem que a parte anterior houvesse sido terminada ou sequer começada. Isso permitiu que as partes consideradas mais difíceis pudessem ser desenvolvidas desde o início, e que mais atenção fosse dada a elas.

Na figura 3.1, é possível observar as diferentes etapas de desenvolvimento adotadas.

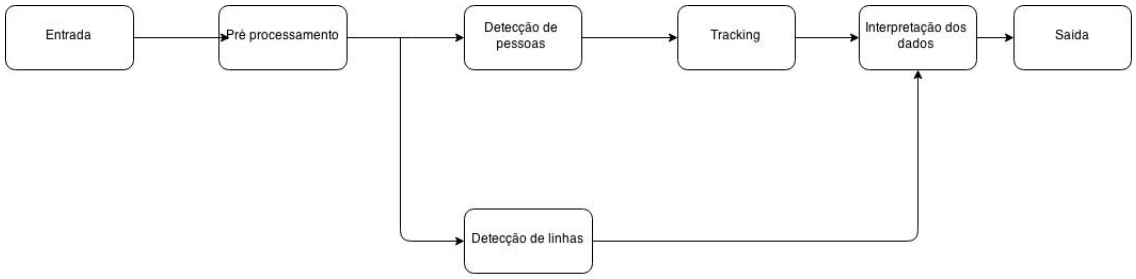

Figura 3.1: Diagrama de blocos que rege o desenvolvimento do sistema.

3.3 Pré Processamento

Foi considerado, durante o desenvolvimento, que o sistema poderia ser usado a partir de imagens filmadas especificamente para a utilização no programa ou também, a partir de vídeos de terceiros, permitindo uma maior flexibilidade ao usuário. A grande diversidade de vídeos de entrada faz com que seja necessária uma etapa de pré processamento, de forma que todas entradas do programa estejam semelhantes. No pré processamento são usadas as linhas da quadra para a calibração da imagem, essa informação ainda é utilizada na etapa de interpretação dos dados, portanto é uma saída paralela da primeira etapa. As etapas de pré processamento são representadas na imagem 3.2.

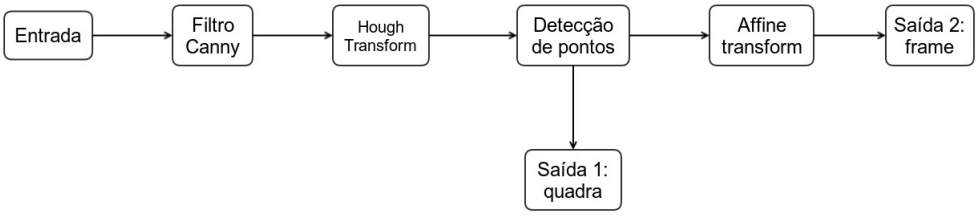

Figura 3.2: Diagrama de blocos do pré processamento do vídeo.

3.4 Detecção de Linhas

Para realizar a identificação das linhas da quadra, foi utilizado a transformada de Hough para linhas, a partir de uma imagem que passou anteriormente por um filtro de Canny, que identifica bordas, não só das linhas da quadra, mas também dos jogadores e boa parte da torcida. A partir das informações do filtro é possível realizar a transformada e identificar

linhas retas na imagem.

As linhas detectadas são analisadas e suas intersecções encontradas, a partir de suas posições relativas é possível encontrar os cantos da quadra para cada *frame*. O canto superior esquerdo é o ponto em que temos a menor soma de x e y, o canto inferior direito é o ponto com maior soma das suas coordenadas, e os outros dois pontos são intermediários, com maior x e menor y ou o contrário.

A partir da posição dos cantos em *frames* consecutivos é possível detectar se houve movimento da câmera, e corrigi-lo a partir da matriz de transformação, que detecta translação, rotação e escala. Aplicando a matriz de transformação obtida em todos os *pixels* do segundo *frame*, é obtido uma imagem em que as linhas da quadra estão alinhadas com a primeira, é necessário passar a imagem resultante por um filtro da média para retirar pequenas descontinuidades nas linhas e imperfeições.

3.5 Detecção de Jogadores - Técnicas Consideradas

A identificação dos jogadores foi a etapa para qual foi dada maior atenção durante o desenvolvimento do sistema. Nela, é necessário identificar a posição inicial deles, para posteriormente entender qual a movimentação que eles tem em quadros seguidos.

Como o desenvolvimento foi feito em C++, focado em um sistema embarcado, foi possível perceber que o desenvolvimento do filtro de Kalman em um ambiente OpenCV exige muita memória, portanto essa alternativa foi descartada.

Algumas opções ao filtro de Kalman foram testadas durante o desenvolvimento, entre elas é possível citar, a técnica de Lucas Kanade, a técnica de histogramas com gradientes orientados, em inglês HOG, que utiliza uma máquina de vetores de suporte, em inglês SVM, , uma técnica de detecção de descritores e um filtro de diferença baseado na distancia entre pixels no espaço RGB.

A técnica de Lucas Kanade, como mostrado na figura 3.3, procura imagens iguais em quadros distintos, para traçar um caminho com elas; essa técnica é muito boa para detectar movimentos de corpos rígidos sem rotação. Infelizmente esse algoritmo não é eficiente para a detecção de pessoas, principalmente enquanto elas praticam atividades físicas, já que as mesmas podem ter posições relativas de braço, corpo e pernas muito distintas.

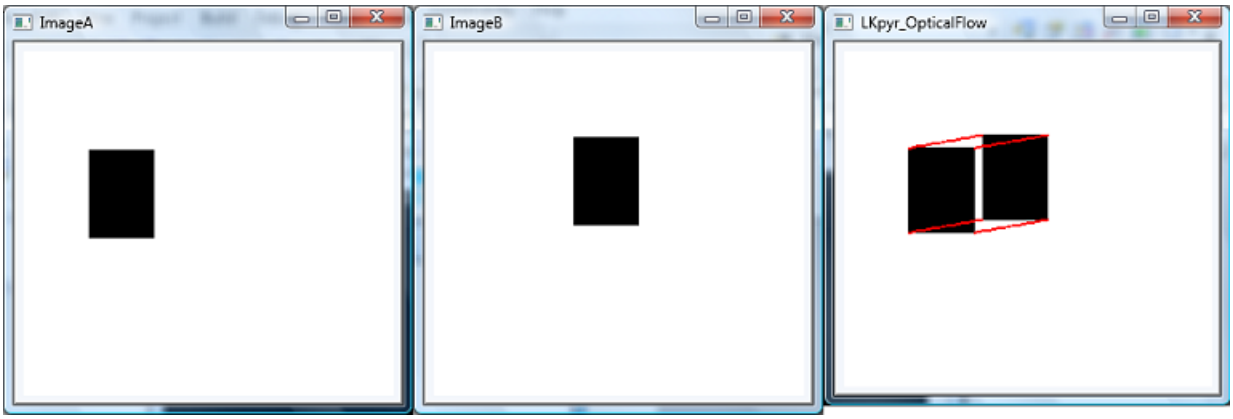

Figura 3.3: Exemplo de utilização da técnica de Lucas Kanade.

A técnica de HOG com SVM, como mostrada na figura 3.4, é um método em que inicialmente é necessário treinar o SVM, ou seja, é necessário fornecer uma série de imagens de pessoas em situação de jogo, para que as diferentes configurações de movimento possam ser detectadas. Com a técnica é fornecida uma SVM default, que é amplamente utilizada para a detecção de pessoas caminhando, na impossibilidade de treinar uma nova máquina de vetores, foi utilizada a fornecida, como forma de teste.

Figura 3.4: Imagens de pessoas usadas para a calibração do *default* do SVM.

Para que a detecção dos jogadores fosse automatizada, foi utilizada uma técnica semelhante a um filtro de diferenças, onde é calculada, para cada *pixel* da imagem, a distância, no espaço RGB, entre o valor dele e o valor do pixel correspondente no *frame* anterior. O diagrama da figura 3.5 indica as etapas de desenvolvimento.

Figura 3.5: Diagrama de blocos da detecção dos jogadores.

A comparação das cores no espaço RGB foi escolhido pelo fato que o fundo é predominantemente de uma única cor, numa quadra ou campo, portanto tem resultados melhores em comparação com um filtro de diferenças tradicional.

A partir da imagem de diferenças em *grayscale*, foi realizada uma binarização, para diminuir o ruído, com o valor do *threshold* escolhido empiricamente.

Na imagem binarizada resultante, foi possível detectar regiões que pertencem ao mesmo jogador, mas que não estão conectadas, gerando assim erro de detecção múltipla. Para que esse erro fosse minimizado, foi realizada uma dilatação na imagem com uma máscara de tamanho k:

$$k = 2.(\frac{width}{360}) + 1 \tag{3.1}$$

Para *frame* resultante, foi feita uma análise baseada em uma região de interesse. A imagem foi varrida por essa região, de forma que, se uma quantidade mínima de *bits* internos a região fossem brancos, era considerado que ocorreu movimento naquela área.

O tamanho N da região de interesse foi definido como:

$$N = \frac{width}{45} \tag{3.2}$$

A quantidade mínima de pontos brancos dentro da região P, para que seja considerado o movimento é:

$$P = \frac{N^2}{2} \tag{3.3}$$

O ajuste dos valores nas equações anteriores foram definidos baseados em [4] e de forma a proporcionar respostas equivalentes para diferentes resoluções de imagem.

Todas as regiões em que foi reconhecido o movimento, foram pintadas de branco em uma imagem preta de mesmo tamanho que o *frame* estudado, isso retira o ruído proporcionado por regiões em que não foram detectados *bits* brancos o suficiente para a detecção de movimento.

Com a nova imagem, foi encontrado o contorno ao redor das regiões brancas, e com esse contorno, foi definido o MBR, *minimum bounding rectangle*, o menor retângulo que encapsula toda a região.

Para cada MBR encontrado, foi definida sua posição como a média entre as coordenadas x dos seus cantos, e a maior coordenada y, considerando a origem no canto superior esquerdo da tela. Na imagem 4.2, a posição dos objetos de interesse estão marcadas como pequenas circunferências na borda de cada MBR.

3.6 Análise da Posição dos Jogadores

Com a posição de cada jogador bem definida, foi necessário entender claramente em qual ponto da quadra esse jogador estava, portanto foi realizada uma transformada de perspectiva, considerando os pontos relativos aos objetos e a posição dos cantos da quadra, encontrados anteriormente.

Foi traçada um diagrama, apresentado na figura 3.6, seguindo as proporções corretas de uma quadra de tênis oficial, de forma a representar a nova posição dos jogadores e bola.

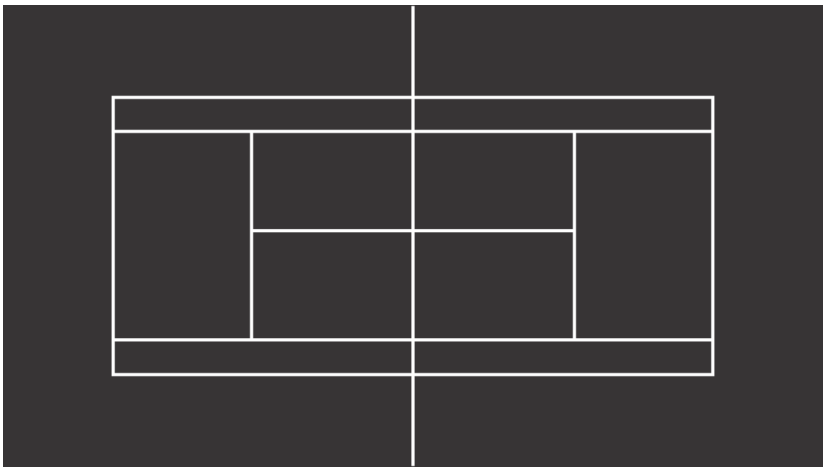

Figura 3.6: Quadra traçada para mudança de perspectiva na representação.

A mudança de perspectiva foi necessária para estimar o valor de deslocamento e velocidade dos objetos, mas foi também uma alternativa considerada para diminuir o trabalho computacional em um sistema embarcado, já enviar informações para a tela é uma das atividades que exige uma maior quantidade de processamento e memória de um sistema.

A partir da posição dos jogadores, da bola e das linhas da quadra, é possível identificar jogadores em *frames* consecutivos[8, 9] além de identificar sua velocidade, considerando o intervalo de tempo real entre cada quadro.

A identificação de jogadores em sequências de vídeos é feita a partir de sua posição relativa, sempre que um objeto novo é detectado a partir do movimento, uma comparação é feita para todos os quadros subsequentes, se existe um objeto se movendo próximo a posição anterior do mesmo, a posição é atualizada, se não ocorreu movimentação nas proximidades, é considerado que o objeto permaneceu parado.

Essa técnica de identificação só pode ser usada porque existem poucos momentos de superposição nas partidas de tênis, e portanto os erros são minimizados.

A partir da posição de cada objeto identificado, foi necessário identificar os jogadores e a bola,além desses, muitos outros objetos foram detectados. Toda movimentação de árbitro, torcida ou pessoal técnico é detectada e apresentada no vídeo. Objetos que se movimentam por uma quantidade de quadros muito pequena, e permanecem parados por boa parte de tempo são considerados como alheios a partida, objetos que se movimentam na maior parte dos quadros, mas que permanecem sempre do mesmo lado da quadra, são jogadores, enquanto objetos que se movimentam livremente pelo *frame* são considerados bolas.

Ao final de cada ponto da partida, há um intervalo de alguns segundos em que a posição dos jogadores não é importante, portanto foram analisados clipes de duração de apenas um ponto.

A saída do programa é uma planilha com a posição dos objetos detectados em todos os *frames*.

Capítulo 4

Resultados e Análise

4.1 Pré Processamento

A saída em vídeo da etapa de pré processamento de imagem do projeto pode ser vista na figura 4.1. Foi possível perceber que a quantidade de *pixels* da imagem, é o principal fator que limita o desempenho dessa etapa.

Figura 4.1: Saída do pré processamento de vídeo, após a aplicação do filtro de média.

Para vídeos de menor qualidade, transformadas afins fazem com que as linhas da quadra se tornem descontínuas, dando a impressão que a imagem muda muito de um *frame* para o

outro, apesar de a quadra permanecer centralizada.

Portanto, quanto menor a qualidade da imagem, maior deve ser a máscara do filtro de média aplicado para que o vídeo permaneça fluido, o que, por sua vez, prejudica a detecção dos objetos na etapa seguinte, já que as imagens na saída ficam cada vez menos claras. Foi necessário encontrar um equilíbrio, em que a impressão de fluidez se mantém, e o vídeo ainda pode ser usado para detecção de jogadores e bola; a menor qualidade de vídeo que foi considerada como boa o suficiente para ser usada foi o de 720p, para qualquer valor menor que esse, a rotina de pré processamento prejudica o resultado final do sistema.

Um filtro de média com máscara 5x5 foi o que apresentou, visualmente, melhores resultados em diversas qualidades de vídeo.

A técnica utilizada para a retirada de movimentação do video funciona melhor para movimentações relativamente grandes; para pequenos tremores que surgem de erros de captação em uma imagem outras técnicas teriam que ser utilizadas.

Outro fator importante da técnica aplicada é que ela produz resultados melhores para as regiões mais próximas ao centro, quanto mais longe do centro do frame o pixel esta, maior será o ruído introduzido.

4.2 Detecção dos Jogadores

Considerando as diferentes técnicas testadas para identificar os jogadores, foram observados alguns resultados.

Com relação a técnica de Lucas Kanade, não foi possível obter nenhum resultado de identificação, visto o problema discutido anteriormente de que os jogadores não apresentam formas rígidas a serem observadas, o que dificulta a execução do mesmo.

A técnica HOG com o SVM *default* também não se mostrou capaz de detectar nenhum jogador, isso se deve ao fato de que a calibração feita anteriormente, considerava uma movimentação bem comportada das pessoas, no caso do tênis, os jogadores estão correndo e em padrões muito distintos dos usados na calibração da máquina, isso dificulta qualquer aplicação do SVM *default* para esses casos.

Considerando agora a técnica para identificação automática dos jogadores a partir da distancia no espaço RGB entre pixels correspondentes, ela demanda um esforço computacional muito maior comparada a outras alternativas em que apenas regiões menores são consideradas, já que em todo quadro, é necessário realizar uma varredura da imagem para encontrar

diferenças nas cores e posições. Em compensação, os resultados são muito melhores.

Foram realizados testes com dois tipos diferentes de vídeos, no primeiro, ele com a câmera parada enquanto no segundo momento foi utilizado um video de uma transmissão de TV em que a câmera segue a bola.

Para o vídeo que precisou ser passado pela rotina de pré processamento, os resultados são piores que o do vídeo filmado parado, mas ainda são muito melhores que a rotina que estava sendo aplicada anteriormente. Em comparação com o exemplo de MATLAB, os resultados são muito próximos, apresentando ainda a versatilidade de ser aplicada a sistema com a câmera sofrendo certo nível de movimentação.

Os resultados melhores dessa alternativa se dão devido principalmente a dois fatores, o primeiro deles é que nessa técnica, não é considerada apenas uma região próxima a anterior, mas detectado movimento em todo o quadro, e a segunda é que a distância no espaço RGB leva em consideração as duas características que foram consideradas anteriormente de forma separadas, a movimentação entre frames consecutivos e o valor RGB do pixel em frames consecutivos.

Outro fator importante é que é realizada uma dilatação na imagem para que todas as regiões de movimentação de um único objeto sejam detectadas com uma única área interconectada, porém em alguns momentos isso não foi suficiente. Principalmente no início de cada ponto, quando cada jogador movimenta partes específicas do corpo para sacar, podem ser detectados mais de uma região, quando elas deviam ter sido detectas como uma só. Esse erro teve de ser ajustado na etapa de identificação de qual jogador corresponde cada região encontrada.

É importante ressaltar que as técnicas anteriores fazem a identificação dos jogadores, enquanto essa técnica automática apenas detecta regiões onde ocorrem movimentação, e essas devem ser posteriormente analisadas, pela posição e comportamento, para identificar os jogadores, a bola e outros objetos, como árbitros.

Diferentemente das técnicas analisadas anteriormente, até a bola é detectada nessas situações. A única ressalva é que quando a bola está próxima do jogador, ambos objetos passam a ser detectados como um único, isso pode inclusive alterar as dimensões do MBR e comprometer em parte a posição detectada do jogador.

A detecção da bola ocorre antes da transformada de perspectiva, porém ao realizar a transformada, foi realizada uma suposição, que todos objetos se encontram no mesmo plano do chão, o que não é verdade para o caso da bola, já que em geral ela esta se movendo em

todo o espaço tridimensional. Dessa forma, a trajetória e velocidade da bola encontrados não podem ser considerados como corretos.

Uma alternativa encontrada na literatura[4] para esse problema é a utilização do áudio junto ao vídeo para a encontrar momentos em que a posição da bola pode ser considerada como correta. Em geral, a bola anda em trajetórias retas, com o ponto inicial e final bem definidos, por posições em que ela esta junto aos jogadores.

O áudio da bola tocando o chão e batendo na rede da raquete é bem definido e pode ser usado para detectar diferentes frames em que é possível analisar corretamente a posição da bola.

Como limites e dificuldades dessa técnica, foi possível perceber que a qualidade da imagem utilizada influencia muito no resultado, quanto melhor a qualidade do vídeo e maior sua estabilidade, melhor o seu desempenho. Vídeos que foram tratados por técnicas utilizadas em transmissões esportivas trouxeram uma resposta ainda superior, indicando que as técnicas de pré processamento não foram suficientes para igualar as condições de um vídeo filmado com a câmera parada ou mesmo com rotinas de pré processamento comerciais.

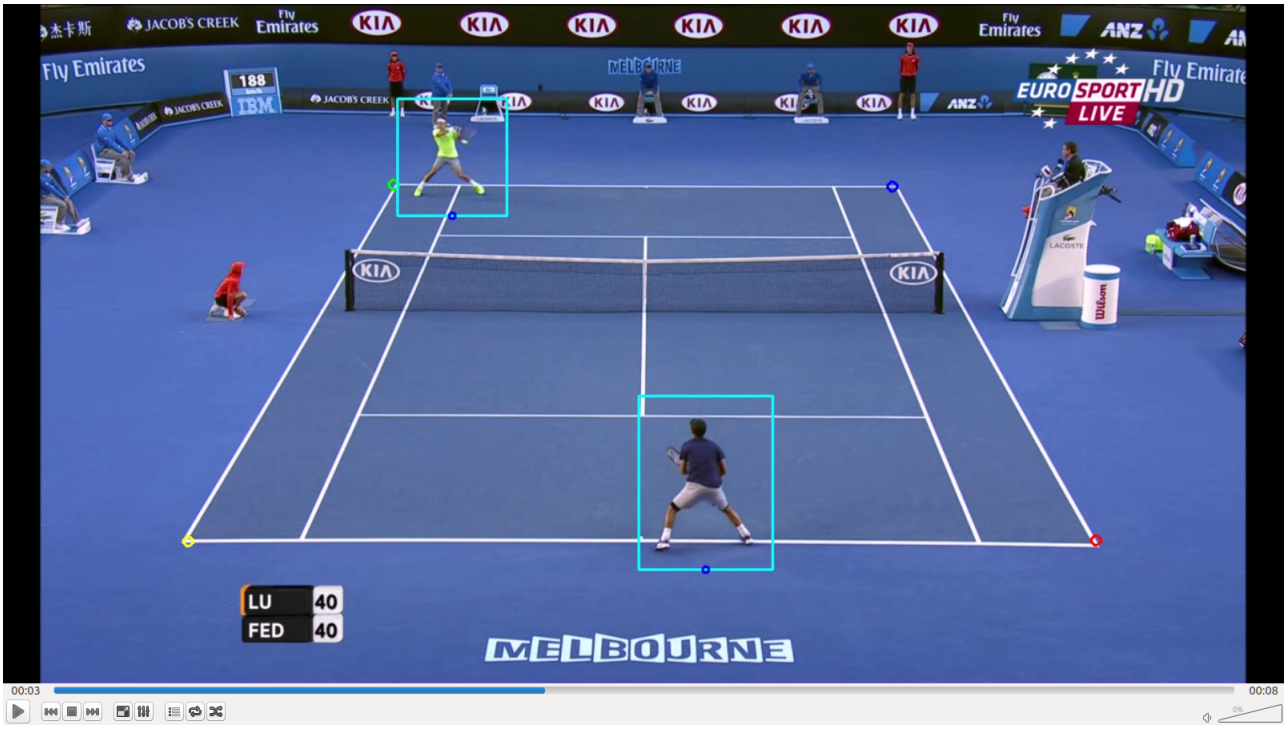

Figura 4.2: Saída da detecção do movimento.

4.3 Detecção de Linhas

A identificação de qual jogador corresponde cada região de interesse encontrada, e os dados para a realização da transformada de perspectiva dependem da identificação das linhas da quadra e de suas intersecções. A utilização da transformada de Hough traz como resultado uma série de linhas detectadas na imagem. Essas linhas apresentam não só àquelas que limitam a quadra, mas também outras que aparecem graças a torcida ou a outros fatores, como propagandas.

O ajuste dos valores da transformada tiveram que ser feitos para especificamente para cada partida, a partir de tentativa e erro, eles levaram em consideração que as regiões de torcida não fossem detectadas, e de forma que cada linha da quadra fosse detectada apenas duas vezes, isso é, na sua borda exterior e na borda interior.

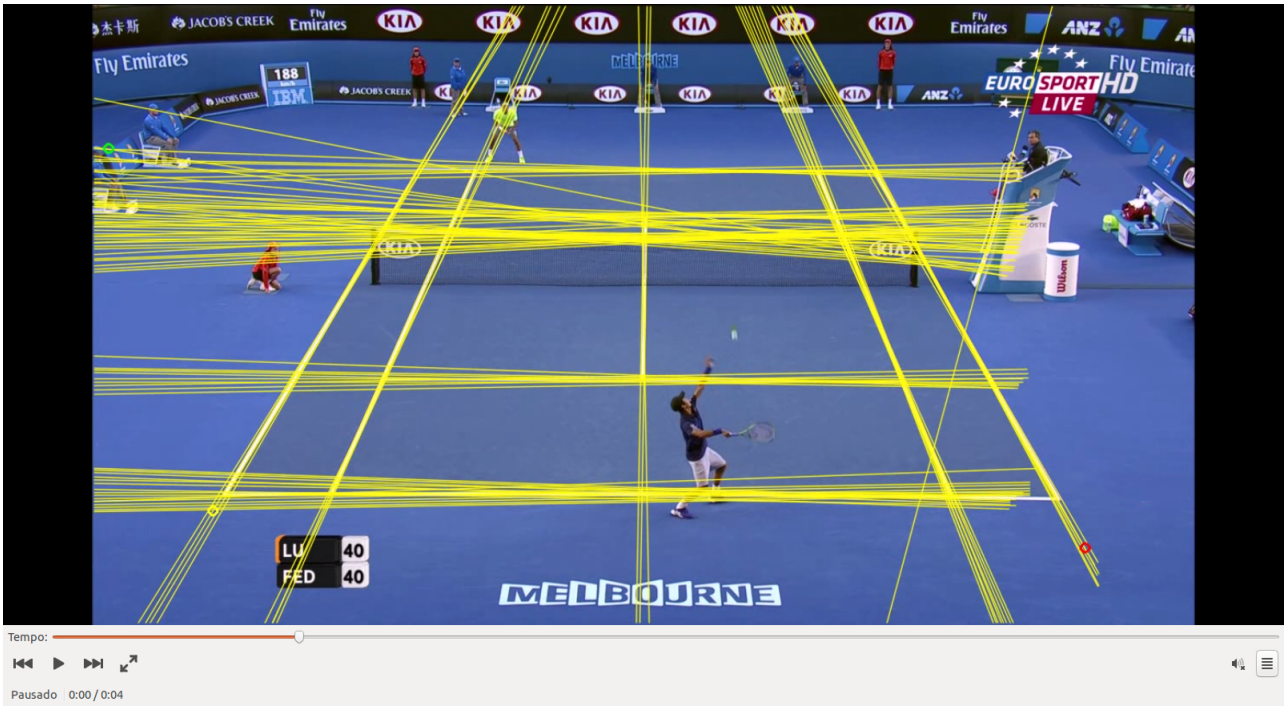

Figura 4.3: Detecção de linhas desajustada.

As imagens 4.3 e 4.4 apresentam uma situação em que a detecção de bordas esta desregulada e uma em que os parâmetros foram ajustados corretamente. O ajuste foi feito a partir do parâmetro de escolha da transformada de Hough, que permite escolher qual a quantidade mínima de *pixels* pertencente àquele segmento de reta, para que ele fosse mostrado.

O melhor ajuste encontrado ainda não detecta apenas as linhas de interesse, mas também outras, como as linhas da rede. A parte superior da rede sempre é detectada como linha,

já que é da mesma cor que as linha da quadra, mas nesse caso são detectadas várias linhas nessa posição, já que a rede apresenta uma curvatura. Essas linhas também foram levadas em consideração na análise dos pontos de interesse.

Figura 4.4: Detecção de linhas bem ajustada.

Os pontos a serem considerados foram aqueles onde ocorrem a intersecção entre diferentes retas, dentro do tamanho da imagem, e dessa forma eles foram analisados para encontrar os extremos da quadra. Os pontos de extremo são encontrados considerando sua posição relativas aos outros, isso gera alguns limites com relação a posição em que o vídeo pode ser gravado.

A câmera deve estar posicionada de forma que as linhas de fundo da quadra estejam o mais horizontais possível. Caso as linhas de fundo de quadra estejam em ângulo, elas devem ser tais que as linhas laterais da quadra não façam um ângulo maior que noventa graus com uma linha horizontal.

4.4 Identificiação dos Jogadores

A saída do programa, após a transformada de perspectiva, sem a identificação dos jogadores esta apresentada na figura 4.5, ela foi desenhada levando em consideração as dimensões reais de uma quadra de tênis, que tem 23,77m de comprimento e 10,97m de largura, repre-

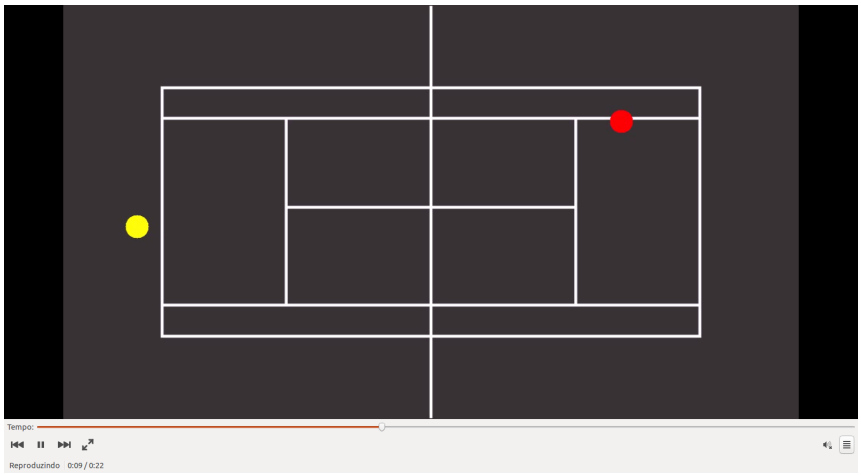

Figura 4.6: Resultado do projeto, com a identificação dos jogadores.

sentados por 437 *pixels* de comprimento e 79 *pixels* de largura, dessa forma, cada elemento representa uma área real de 5,44cm x 13,88cm.

A identificação dos diferentes jogadores e bolas foi feita de duas formas diferentes, a primeira foi baseada no tamanho do MBR detectado, de forma que os dois maiores MBR detectados são jogadores.

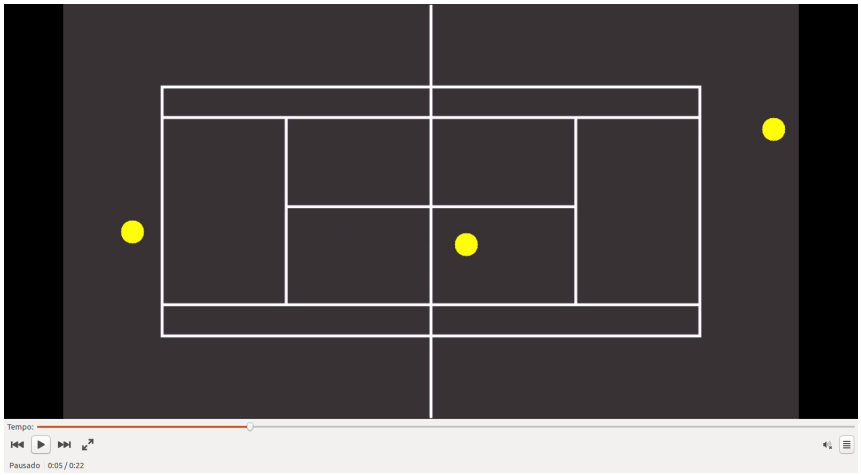

Figura 4.5: Resultado do projeto, sem a identificação dos jogadores.

A segunda alternativa é semelhante a detecção anterior, em que é considerada uma região próxima a posição anterior, para encontrar o deslocamento.

Como para a condição necessária da câmera, o tamanho dos jogadores e bola são bem específicos, a melhor alternativa foi a de definição pelo tamanho. O resultado dessa alternativa é apresentado na figura 4.6.

Outra saída do programa é uma tabela ou banco de dados com todas as posições assumidas

pelos jogadores no período considerado,em geral, um ponto. A tabela a seguir apresenta um exemplo de resultado para alguns *frames*.

Tabela 4.1: Resultado do programa, tabela de posicionamento dos jogadores em cada *frame*.

Frame	X do Jogador 1	Y do Jogador 1	X do Jogador 2	Y do Jogador 2
9	161	394	1158	227
10	168	395	1158	226
11	169	396	1158	224
12	175	397	1162	222
13	180	398	1162	220
14	184	401	1167	219
15	194	402	1172	218
16	119	377	1236	211
17	133	377	1278	206
18	215	404	1241	210
19	222	405	1201	215
20	233	407	1201	215
21	134	377	1201	215
22	238	407	134	377
23	242	405	134	377
24	209	395	134	377
25	206	395	134	377

Apesar do *tracking* dos jogadores, quando observado no vídeo sem a transformação de perspectiva, ser muito bom, ao realizar a transformada é possível perceber uma clara diminuição na precisão da posição dos jogadores, isso se dá pela mudança frequente nas bordas do MBR, o que gera uma mudança na posição detectada de cada objeto, mesmo que esse não se movimente de fato.

Esse erro é aumentado com a transformada de perspectiva, até que na última etapa do desenvolvimento, ele fica bem evidente, alterando em partes o resultado final. Torna-se necessário mudar a definição da posição do jogador, baseado no seu MBR.

A partir da tabela de posições gerada, é possível retirar todas as informações com relação aos jogadores.

Com relação a aplicação do programa em um sistema embarcado, o desenvolvido foi di-

vidido em três partes diferentes, de forma que o pré processamento e a identificação dos jogadores e bola seriam feitas em diferentes dispositivos. A única etapa realizada no dispositivo embarcado seria o *tracking* dos objetos em movimento e das linhas da quadra. Tendo em vista o teste da rotina desenvolvida em uma aplicação embarcada, o programa foi testado em uma Raspberry Pi 2, e mesmo limitando a quantidade de informação a ser processada a um mínimo, um clipe de oito segundos de duração e duzentos e vinte *frames* levou mais de 30 minutos para ser processado, o que indica que a capacidade de processamento da placa precisa ser muito maior.

Considerando os resultados apresentados, é possível perceber que em muitos momentos esta ocorrendo a detecção e o *tracking* dos jogadores, mas considerando os objetivos gerais do trabalho, e considerando ainda todas as etapas de desenvolvimento apresentadas anteriormente, fica claro que ainda faltam muitas fases de desenvolvimento a serem cumpridas para que um produto completo possa ser apresentado.

Capítulo 5

Conclusão

A partir dos resultados obtidos, é possível inferir algumas conclusões, entre elas, a mais importante é que é possível criar um sistema de detecção de jogadores em partidas de tênis com relativamente baixo esforço computacional e erro de detecção.

Dentre as várias etapas de desenvolvimento realizadas, a de pré processamento conseguiu centralizar uma imagem em movimento como o projetado, mas a aplicação dessa imagem centralizada no sistema desenvolvido não obteve resultados visualmente superiores às da imagem com pouca movimentação, permitindo perceber que técnicas mais complexas poderiam ter sido utilizadas, mesmo que isso limitasse a velocidade da solução.

A qualidade da imagem foi identificada como um fator relevante em diferentes etapas no desenvolvimento do projeto. Uma melhor qualidade de imagem, ou seja, aquela que possui mais *pixels*, gera melhores resultados no programa, porém esse mesmo fator é responsável pelo aumento da necessidade de processamento, de forma que deve existir um equilíbrio entre os dois fatores para otimizar o desempenho. Conforme o número de *pixels* que o jogador ocupa é reduzido, diminui também o quanto uma alteração nos elementos influencia no valor total da média calculada no processo. Como solução para esse problema pode se propor a utilização de uma imagem de resolução média(720p).

Considerando a etapa de detecção dos jogadores, foi possível criar um sistema em que toda movimentação é detectada. Isso permitiu que os jogadores fossem detectados durante a maior parte do ponto, porém foi necessário reiniciar a detecção no início de cada ponto.

A maior fonte de erro nessa etapa ocorre quando o jogador e a bola são detectados em um mesmo MRB, de forma que a posição detectada do jogador muda rapidamente.

Na etapa de diferenciação dos objetos, devido principalmente a posição relativa dos jogadores na quadra e dela no video, foi possível utilizar um algoritmo simples para a separação

de jogadores, a partir do seu tamanho. Nessa etapa, desde que ambos os jogadores tenham sido detectados e não ocorra oclusão, não foram detectadas fontes significativas de erros. A diferenciação dos jogadores ser tão simples do ponto de vista computacional é um dos fatores que permite a utilização da distância no espaço RGB para a detecção da posição dos jogadores.

A utilização de apenas uma câmera gera algumas dificuldades com relação a profundidade, principalmente na detecção de objetos que não ficam todo o tempo no chão, no caso de partidas de tênis, a bola. Essa dificuldade pode ser contornada com a utilização de outras ferramentas, que não foram abordadas nesse trabalho.

Com relação ao desenvolvimento focado em sistemas embarcados, todo o software utilizado pode ser utilizado em uma aplicação embarcada, e o atendimento dos requisitos de velocidade de processamento poderá ser atendido em função da definição das especificações do sistema embarcado.

Ficou possível observar que a criação de um sistema de detecção de jogadores com baixo custo de implementação, que pode oferecer informações relevantes e ajudar no planejamento e análise de partidas, tanto de tênis como de outros esportes. Isso Possibilita a criação de alternativas para o mercado que no momento apresentam soluções de alto custo, e de difícil implementação.

5.1 Sequência do trabalho

Mesmo com todo o trabalho e as diferentes técnicas aplicadas e testadas para o desenvolvimento do sistema, ainda restam muitos pontos a serem abordados e trabalhados.

Com as limitações em se embarcar o sistema, se torna necessário fazer um levantamento de outras opções de hardware embarcado para melhorar o desempenho do sistema.

Como alternativa ao sistema embarcado, pode-se optar pelo desenvolvimento de um sistema não embarcado, que pode ser usado por técnicos e jogadores para uma análise posterior. Outra alternativa seria tornar o sistema capaz de ser operado em paralelo, de forma a aumentar o rendimento do mesmo.

A alternativa escolhida para continuar o desenvolvimento, sugerida acima, também influencia qual deve ser a atenção dada para as melhorias aplicadas ao sistema.

Caso o foco seja para imagens que já chegam ao programa com uma qualidade e pré processamento com boa qualidade, o foco do desenvolvimento deve ser em melhorar a iden-

tificação dos jogadores de forma automática, para que no final, o desenho do deslocamento de cada jogador seja mais suave.

Outro ponto importante de desenvolvimento semelhante ao anterior, é a identificação correta do posicionamento da bola durante todo o jogo. Isso é importante não só para encontrar outras informações nas partidas de tênis, mas também para que seja aplicada em outros esportes.

Caso o foco do desenvolvimento seja na utilização de imagens de menor qualidade ou com maior movimentação, deve ser dada uma maior atenção para a etapa de pré processamento, de forma a implementar rotinas semelhantes as usadas na TV para produzir imagens o mais estáveis possível.

Quando todas etapas precedentes estiverem sido cumpridas, só então seria possível obter uma posição confiável em um plano bidimensional para a partir daí se obter alguma informação dos jogadores e da partida.

A partir de uma melhora na detecção da posição dos jogadores e bola, é possível melhorar as informações obtidas. Se o posicionamento dos jogadores e da bola forem muito precisos, é possível até realizar uma análise de tempo de reação dos jogadores em saques dos adversários.

O ideal seria que ao menos a posição, a velocidade máxima, velocidade média e o tempo de reação de cada jogador fossem medidos.

Referências Bibliográficas

[1] H. Ben Shitrit, M. Raca, F. Fleuret, and P. Fua, "Tracking multiple players using a single camera," tech. rep., Springer Verlag, 2013.

[2] H. Ben Shitrit, J. Berclaz, F. Fleuret, and P. Fua, "Multi-commodity network flow for tracking multiple people," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, no. 8, pp. 1614–1627, 2014.

[3] X. Wang, V. Ablavsky, H. B. Shitrit, and P. Fua, "Take your eyes off the ball: Improving ball-tracking by focusing on team play," *Computer Vision and Image Understanding*, vol. 119, pp. 102–115, 2014.

[4] T. H. Raya, *Automatic feature extraction from tennis videos for content based retrieval*. THE UNIVERSITY OF ALABAMA IN HUNTSVILLE, 2011.

[5] J. Pokornỳ, "Tracking players in low-resolution videos of soccer games," 2012.

[6] L. Shapiro and G. C. Stockman, "Computer vision. 2001," *ed: Prentice Hall*, 2001.

[7] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.

[8] K. Okuma, D. G. Lowe, and J. J. Little, "Self-learning for player localization in sports video," *arXiv preprint arXiv:1307.7198*, 2013.

[9] J. I. Agbinya and D. Rees, "Multi-object tracking in video," *Real-Time Imaging*, vol. 5, no. 5, pp. 295–304, 1999.

Apêndice A

Apêndice 1

Código do programa desenvolvido em C++ com OpenCV, para o pré processamento.

```
using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{
        Mat frame1, gray1, result1, frame2, gray2, result2, contorno, zoom;
        Mat original;
        Mat peq1, peq2, peq3, peq4;
        int k, i, j, pos1x=20, pos1y=20, pos2x, pos2y, produto_calc=0, produto_maior=0;
        int cont1, cont2, conta_quadro=0;
        float relacao, A, B;
        int filtro;
        Point A1, A2, B1, B2, C1, C2;
        float alfa1, alfa2, beta1, beta2, r1, r2;
        int x, y;
        int xa1, xa2, xb1, xb2, xc1, xc2, ya1, ya2, yb1, yb2, yc1, yc2;
        double Sx, Sy, Tx, Ty, teta;
        int last_tx=0, last_ty=0;

        //A: superior esquerdo
        //B: inferior direito
        //C: inferior esquerdo

        VideoCapture cap("tcc3.mp4");

        //testa se abriu corretamente o arquivo
        if (!cap.isOpened())
        {
                std::cout << "!!! Failed to open file: " << argv[1] << std::endl;
                return -1;
        }

        int width  = cap.get(CV_CAP_PROP_FRAME_WIDTH);
        int height = cap.get(CV_CAP_PROP_FRAME_HEIGHT);

        //variáveis para serem utilizadas na correção
        A=width/2;
        B=height/2;
        filtro = 2*(width/360)+1;

        //cria saida de video
        VideoWriter writer;
        writer.open( "saida1.wmv", CV_FOURCC('X','V','I','D'), cap.get(CV_CAP_PROP_FPS), cv::Size(width, height), true );
```

```
//cria as janelas
//namedWindow("gray");
//namedWindow("antes");
//namedWindow("depois");

//faz a calibração inicial no primeiro frame, para comparação posterior
cap.read(frame1);
peq1 = Mat(frame1, Rect(120, 80, (frame1.cols-160), (frame1.rows-160)));
cvtColor(peq1, gray1, CV_RGB2GRAY);
//especifico para o YCrCb
/*
vector<Mat> chanels;
split(gray1, chanels);
result1 = chanels[0];
*/
threshold(gray1, result1, 150, 255, THRESH_BINARY);
Canny(result1, result2, 30, 160);
//imshow("depois", result2);
//cvWaitKey(0);

vector<Vec2f> lines, lines_ok;
HoughLines(result2, lines, 1, CV_PI/180, 80, 0, 0 ); //valores experimentais
cout << "lines = " << lines.size()<< endl;

//printa as linhas
/*
for(i = 0; i < lines.size(); i++ )
{
        float rho = lines[i][0], theta = lines[i][1];
        Point pt1, pt2;
        double a = cos(theta), b = sin(theta);
        double x0 = a*rho, y0 = b*rho;
        //cout << "polar: rho = " << rho << " theta = " << theta << endl;
        //cout << " retangular: a = " << a << " b = " << b << endl;
        //cout << endl;
        pt1.x = cvRound(x0 + 1000*(-b))+80;
        pt1.y = cvRound(y0 + 1000*(a))+80;
        pt2.x = cvRound(x0 - 1000*(-b))+80;
        pt2.y = cvRound(y0 - 1000*(a))+80;
        line( frame1, pt1, pt2, Scalar(0,255,255), 1, CV_AA);
}

imshow("depois", frame1);
cvWaitKey(10);
*/
//calcular os pontos de intersecção das retas
A1.x=1000;
A1.y=1000;
B1.x=10;
B1.y=10;
C1.x=1000;
C1.y=10;


for (i=0; i<lines.size(); i++)
{
        for (j=i+1; j<lines.size(); j++)
        {
                alfa1=(-1)/tan(lines[i][1]);
                alfa2=(-1)/tan(lines[j][1]);
                r1=lines[i][0];
                r2=lines[j][0];
                beta1=r1/sin(lines[i][1]);
                beta2=r2/sin(lines[j][1]);
```

```
                x=(beta2-beta1)/(alfa1-alfa2);
                y=alfa1*x+beta1;

                Point teste;
                teste.x=x+80;
                teste.y=y+80;

                if((x+80>=0)&&(x+80<width)&&(y+80>=0)&&(y+80<height))
                {
                        //cout << "x = "<< x << " y = " << y << endl;
                        circle(frame1, teste, 3, Scalar(255, 0, 0), 2);

                        if((teste.x+teste.y)<=(A1.x+A1.y))
                        {
                                A1.x=teste.x;
                                A1.y=teste.y;
                        }
                        if((teste.x+teste.y)>=(B1.x+B1.y))
                        {
                                B1.x=teste.x;
                                B1.y=teste.y;
                        }
                        if((teste.x<=C1.x)&&(teste.y>=C1.y))
                        {
                                C1.x=teste.x;
                                C1.y=teste.y;
                        }
                }

        }
}

for(;;)
{
        //le o segundo frame do video, para corrigir
        cap.read(frame1);
        if (!cap.read(frame1))
        {
                cout << "fim do video " << endl;
                break;
        }

        //fransforma a imagem em binária, para diminuir o ruído no filtro canny
        peq1 = Mat(frame1, Rect(80, 120, (frame1.cols-160), (frame1.rows-200)));
        //imshow("depois", peq1);
        cvtColor(peq1, gray1, CV_RGB2GRAY);
        //Usa o canal Y como uma imagem grayscale
        /*
        vector<Mat> chanels;
        split(gray1, chanels);
        result1 = chanels[0];
        */
        threshold(gray1, result1, 180, 255, THRESH_BINARY);

        //realiza o Canny
        Canny(result1, result2, 30, 160);

        //Usa o hough lines

        vector<Vec2f> lines, lines_ok;
        HoughLines(result2, lines, 1, CV_PI/180, 80, 0, 0 ); //valores experimentais

        cout << "lines = " << lines.size()<< endl;
```

```
//printa as linhas

for(i = 0; i < lines.size(); i++ )
{
        float rho = lines[i][0], theta = lines[i][1];
        Point pt1, pt2;
        double a = cos(theta), b = sin(theta);
        double x0 = a*rho, y0 = b*rho;
        //cout << "polar: rho = " << rho << " theta = " << theta << endl;
        //cout << " retangular: a = " << a << " b = " << b << endl;
        //cout << endl;
        pt1.x = cvRound(x0 + 1000*(-b))+80;
        pt1.y = cvRound(y0 + 1000*(a))+120;
        pt2.x = cvRound(x0 - 1000*(-b))+80;
        pt2.y = cvRound(y0 - 1000*(a))+120;
        line( frame1, pt1, pt2, Scalar(0,255,255), 1, CV_AA);
}


//calcular os pontos de intersecção das retas
A2.x=1000;
A2.y=1000;
B2.x=10;
B2.y=10;
C2.x=1000;
C2.y=10;


for (i=0; i<lines.size(); i++)
{
        for (j=i+1; j<lines.size(); j++)
        {
                alfa1=(-1)/tan(lines[i][1]);
                alfa2=(-1)/tan(lines[j][1]);
                r1=lines[i][0];
                r2=lines[j][0];
                beta1=r1/sin(lines[i][1]);
                beta2=r2/sin(lines[j][1]);

                x=(beta2-beta1)/(alfa1-alfa2);
                y=alfa1*x+beta1;

                Point teste;
                teste.x=x+80;
                teste.y=y+120;

                if((x+80>=0)&&(x+80<width)&&(y+80>=0)&&(y+80<height))
                {
                        //cout << "x = "<< x << " y = " << y << endl;
                        //printa o ponto
                        //circle(frame1, teste, 3, Scalar(255, 0, 0), 2);

                        if((teste.x+teste.y)<=(A2.x+A2.y))
                        {
                                A2.x=teste.x;
                                A2.y=teste.y;
                                //cout << "A: " << A2.x << " " << A2.y << endl;
                        }
                        if((teste.x+teste.y)>=(B2.x+B2.y))
                        {
                                B2.x=teste.x;
                                B2.y=teste.y;
                                //cout << "B: " << B2.x << " " << B2.y << endl;
                        }
                        if((teste.x<=C2.x)&&(teste.y>=C2.y))
```

```
                                        {
                                                C2.x=teste.x;
                                                C2.y=teste.y;
                                                //cout << "C: " << C2.x << " " << C2.y << endl;
                                        }
                                }

                        }
                }

                //printa oos pontos de referencia

                circle(frame1, A2, 5, Scalar(0, 255, 0), 2);
                circle(frame1, B2, 5, Scalar(0, 0, 255), 2);
                circle(frame1, C2, 5, Scalar(0, 255, 255), 2);

                frame2= frame1.clone();


                writer << frame1;
                cvWaitKey(10);
        }
}
```

Código do processamento da posição dos objetos:

```
#include <opencv/cvaux.h>
#include <opencv/highgui.h>
#include <opencv/cxcore.h>
#include <opencv2/imgproc/imgproc_c.h>
#include <stdio.h>
#include <stdlib.h>
#include <opencv2/video/background_segm.hpp>
#include <iostream>
#include <fstream>

using namespace cv;
using namespace std;

//variaveis globais
int jogador1[2][2], jogador2[2][2], quadra1[4][2], quadra2[4][2], quadra3[2][2], quadra4[2][2], contaclick=0;

int main(int argc, char* argv[])
{
//define variáveis
Mat frame2, frame1, frame3, result2, result1, result3, result4, peq1, peq2, gray1, bw1, linha1;
int i, j, conta_anterior=0, conta_frame=0;
vector<int> posi_ant;
float dist_x, dist_y, dist_z;

//variaveis para detectar as linhas
Point canto1_2, canto2_2, canto3_2, canto4_2;
float alfa1, alfa2, beta1, beta2, r1, r2;
int x, y;

//abre o arquivo para salvar a informação da bola e dos jogadores
ofstream arq;
//ofstream arq2;
arq.open("posi");
//arq2.open("diagrama");

//le arquivo da memória e salva em cap
VideoCapture cap("tcc3.mp4");

int width  = cap.get(CV_CAP_PROP_FRAME_WIDTH);
int height = cap.get(CV_CAP_PROP_FRAME_HEIGHT);
```

```
//carrega a imagem da quadra
Mat quadra = imread("quadra.png");

//cria saida de video
VideoWriter writer, writer2;
writer.open( "saida1.wmv", CV_FOURCC('X','V','I','D'), cap.get(CV_CAP_PROP_FPS), cv::Size(width, height), true );
//writer2.open( "saida2.wmv", CV_FOURCC('X','V','I','D'), cap.get(CV_CAP_PROP_FPS), cv::Size(width, height), true );

//testa se abriu corretamente o arquivo
if (!cap.isOpened())
{
std::cout << "!!! Failed to open file: " << argv[1] << std::endl;
return -1;
}

//prepara para a exibição do video

//namedWindow("frame");
//namedWindow("difference");

//define os cantos da quadra
Point canto1_1, canto2_1, canto3_1, canto4_1;
canto1_1.y=141;
canto1_1.x=169;
canto2_1.y=578;
canto2_1.x=169;
canto3_1.y=141;
canto3_1.x=1110;
canto4_1.y=578;
canto4_1.x=1110;

//desenha circuilos ao redor deles
//circle(quadra, canto1_1, 3, Scalar(255, 0, 0), 2);
//circle(quadra, canto2_1, 3, Scalar(255, 0, 0), 2);
//circle(quadra, canto3_1, 3, Scalar(255, 0, 0), 2);
//circle(quadra, canto4_1, 3, Scalar(255, 0, 0), 2);

//namedWindow("quadra");
//imshow("quadra", quadra);
//cvWaitKey(0);
//////////////////////////////////////////////////////////////////Pontos de interesse//////////////////////////////////////
//(141, 169) (578, 169) (141, 1110) (578, 1110)

destroyWindow("quadra");

cap.read(frame1);

//cvWaitKey(10);

//limita manualemnte a imagem para que só detecte as linhas da quadra

for(;;)
{
conta_frame++;

if (!cap.read(frame2))
{
cout << "fim do video"<< endl;
break;
}

//imshow("frame", peq1);

result1 = Mat::zeros(frame1.rows, frame1.cols,  CV_8UC1);
```

```
for(i=0; i<frame1.cols; i++)
for(j=0; j<frame1.rows; j++)
{
{
Point ponto;
ponto.x=i;
ponto.y=j;
Vec3b value1 = frame1.at<Vec3b>(ponto);
Vec3b value2 = frame2.at<Vec3b>(ponto);
dist_x= value2.val[0]-value1.val[0];
dist_y= value2.val[1]-value1.val[1];
dist_z= value2.val[2]-value1.val[2];
result1.at<uint8_t>(ponto)=sqrt(dist_x*dist_x + dist_y*dist_y + dist_z*dist_z);
}
}

//converte o resultado para B&W
blur(result1, result2, Size(2,2), Point(-1,-1), BORDER_DEFAULT);
threshold(result1, result2, 70, 255, THRESH_BINARY);

//realiza uma abertura da imagem para unir pontos separados
int k;
k=2*(width/360)+1;
dilate(result2, result3, k);

//DESENHAR O MINIMUM BOUND RECTANGLE
//começa por identificar os macroblocos
int N, dim;
int a, b;
int conta_pixel=0, conta_bloco=0, conta_mega=0; //contador de quantos pixels estão brancos no macrobloco e quantos macroblocos na imagem
vector<int> vetor_posi;
vector<int> vetor_mega;
vector<int> posi_atual;
vector<int> tam;
vector<Point> pe;

//N=frame1.cols/45;
N=25;
dim=N*N/2;

//varre toda a imagem

for(i=0; i<frame1.cols-N; i++)
{
for(j=0; j<frame1.rows-N; j++)
{

peq1 = Mat(result3, Rect(i, j, N, N));
Scalar pixel = sum(peq1);
conta_pixel=pixel[0]/255;

if(conta_pixel>40)
{
//cria um vetor com a posição inicial de todos macroblocos encontrados na imagem
vetor_posi.push_back(i);
vetor_posi.push_back(j);
conta_bloco++;
}
}
}

cout << "quantidade de blocos: " << conta_bloco << endl;
//se a quantidade dde macroblocos for 0, descosidera esse quadro
if(conta_bloco==0) continue;
```

```
//printa todos macroblocos em um fundo preto
Mat result4 = Mat::zeros(result1.rows, result1.cols, CV_8UC1);

for(i=0; i<conta_bloco; i++)
{
Point canto1, canto2;
canto1.x=vetor_posi[2*i];
canto1.y=vetor_posi[2*i+1];
canto2.x=vetor_posi[2*i]+N;
canto2.y=vetor_posi[2*i+1]+N;
rectangle(result4, canto1, canto2, 255);
}

frame1=frame2.clone();
frame3=quadra.clone();

// usar o print dos marcoblocos para desenhar um MBR a partir dos contornos gerados
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;

findContours(result4, contours, hierarchy, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);

vector<vector<Point> > contours_poly( contours.size() );
vector<Rect> boundRect( contours.size() );


////////////////////////////////ENCONTRA AS LINHAS DA QUADRA\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
//diminui a imagem para trabalhar só com a parte central
peq1 = Mat(frame2, Rect(80, 120, (frame1.cols-160), (frame1.rows-200)));
cvtColor(peq1, gray1, CV_RGB2GRAY);
threshold(gray1, bw1, 180, 255, THRESH_BINARY);

//realiza o Canny
Canny(bw1, linha1, 30, 160);

//Usa o hough lines

//cout << "teste1" << endl;
vector<Vec2f> lines, lines_ok;
vector<Point> pontos;
HoughLines(linha1, lines, 1, CV_PI/36, 100, 0, 0 ); //valores experimentais


//calcular os pontos de intersecção das retas
canto3_2.x=1000;
canto3_2.y=1000;
canto2_2.x=10;
canto2_2.y=10;
canto1_2.x=1000;
canto1_2.y=10;
canto4_2.y=1000;
canto4_2.x=10;

for (i=0; i<lines.size(); i++)
{
for (j=i+1; j<lines.size(); j++)
{
alfa1=(-1)/tan(lines[i][1]);
alfa2=(-1)/tan(lines[j][1]);
r1=lines[i][0];
r2=lines[j][0];
beta1=r1/sin(lines[i][1]);
beta2=r2/sin(lines[j][1]);
```

```
x=(beta2-beta1)/(alfa1-alfa2);
y=alfa1*x+beta1;

Point teste;
teste.x=x+80;
teste.y=y+120;

if((x+80>=0)&&(x+80<width)&&(y+80>=0)&&(y+80<height))
{
//cout << "x = "<< x << " y = " << y << endl;
//printa o ponto
//circle(frame1, teste, 3, Scalar(255, 0, 0), 2);

pontos.push_back(teste);

if((teste.x+teste.y)<=(canto3_2.x+canto3_2.y))
{
canto3_2.x=teste.x;
canto3_2.y=teste.y;
//cout << "A: " << A2.x << " " << A2.y << endl;
}
if((teste.x+teste.y)>=(canto2_2.x+canto2_2.y))
{
canto2_2.x=teste.x;
canto2_2.y=teste.y;
//cout << "B: " << B2.x << " " << B2.y << endl;
}
if((teste.x<=canto1_2.x)&&(canto1_2.y>=canto1_2.y))
{
canto1_2.x=teste.x;
canto1_2.y=teste.y;
//cout << "C: " << C2.x << " " << C2.y << endl;
}
if((teste.y<=canto4_2.y))
{
canto4_2.x=teste.x;
canto4_2.y=teste.y;
//cout << "C: " << C2.x << " " << C2.y << endl;
}

}

}
}

for (i=0; i<pontos.size(); i++)
{
Point aux;
aux=pontos[i];
if((abs(aux.y-canto4_2.y)<=20)&&(aux.x>=canto4_2.x))
{
canto4_2.x=aux.x;
canto4_2.y=aux.y;
}
}

//printa oos pontos de referencia

circle(frame2, canto3_2, 5, Scalar(0, 255, 0), 2);
circle(frame2, canto2_2, 5, Scalar(0, 0, 255), 2);
circle(frame2, canto1_2, 5, Scalar(0, 255, 255), 2);
circle(frame2, canto4_2, 5, Scalar(255, 0, 0), 2);
//////////////////////////////////////////FIM DAS LINHAS//////////////////////////////////////////

//////////////////////////DEFINE A TRANFORMADDA//////////////////////////////////////////////////
```

```
Point2f canto1[4], canto2[4];
canto1[0]=canto1_1;
canto1[1]=canto2_1;
canto1[2]=canto3_1;
canto1[3]=canto4_1;
canto2[0]=canto1_2;
canto2[1]=canto2_2;
canto2[2]=canto3_2;
canto2[3]=canto4_2;

Mat transform;

transform = getPerspectiveTransform(canto2, canto1);

//////////////////////////FIM DA TRANSFORMAÇÃO////////////////////////////////////////////////

//printa os MBR
for(i=0; i<contours.size(); i++)
{
Point aux1, aux2, aux3, aux4;
//Mat in;
//Mat out;
approxPolyDP( Mat(contours[i]), contours_poly[i], 3, true );
boundRect[i] = boundingRect( Mat(contours_poly[i]) );
//rectangle( frame2, boundRect[i].tl(), boundRect[i].br(), Scalar(255, 255, 0), 2, 8, 0 );
aux1=boundRect[i].tl();
aux2=boundRect[i].br();
aux3.y=aux2.y;
aux3.x=(aux1.x+aux2.x)/2;

Mat in = (Mat_<double>(3,1) << aux3.x, aux3.y, 1);
Mat out;

out=transform*in;

//cout << in << endl << transform << endl << out << endl;

aux4.x=out.at<double>(0,0)/out.at<double>(2,0);
aux4.y=out.at<double>(1,0)/out.at<double>(2,0);


//circle( frame2, aux1, 3, Scalar(255, 0, 0), 2);
//circle( frame3, aux2, 20, Scalar(0, 255, 255), -1);
int num;
num=abs((aux1.x-aux2.x)*(aux1.y-aux2.y));
arq << conta_frame << " " << aux4.x << " " << aux4.y << " " << num << endl;
//arq2 <<  "frame: " << conta_frame << ", " << aux2 << endl;


}

posi_ant=posi_atual;

//imshow("frame", frame2);
namedWindow("difference");
imshow("difference", result4);

//cvWaitKey(10);

writer<<frame2;
//writer2<<frame3;

vetor_posi.clear();
}
```

```
//arq2.close();
arq.close();

cap.release();
return 0;
}
```

Código da função utilizada para fazer a transformada de perspectiva:

```
#include <stdlib.h>
#include <opencv2/video/background_segm.hpp>
#include <iostream>
#include <fstream>

using namespace cv;
using namespace std;

int main()
{

//abre o arquivo para leitura
ifstream arq;
arq.open("posi", ios::in);

int cont=0, conta_frame=0;
int a, x, y, anterior=0,inicial[4];
int i, num_jog=0, area;
Point ponto, jog1, jog2;
int maior1, maior2;
//definir jogadores como uma matriz Nx4, a primeira coluna é o frame, a segunda é o id do jogador, a terceira e a quarta são as coordenadas
//vector<int> quadro;
//vector<int> id;
//vector<Point> posi;
//vector<Vec1i> total;

Mat quadra = imread("quadra.png");
Mat frame = quadra.clone();

int height= quadra.rows;
int width= quadra.cols;

VideoWriter writer;
writer.open( "saida1.wmv", CV_FOURCC('X','V','I','D'), 10, cv::Size(width, height), true );
/*
a=0;
quadro.push_back(a);
id.push_back(a);

ponto.x=0;
ponto.y=0;
posi.push_back(ponto);
*/
while(arq >> a >> x >> y >> area)
{
int cont=0;

ponto.x=x;
ponto.y=y;

//quadro.push_back(a);
//posi.push_back(ponto);

//definir se o jogador detectado já tem id ou não;
/*
if(a==1)
```

```
{
cont++;
id.push_back(cont);
}
else
{

}

*/
cout << a << " (" << x << ", " << y << ")" << " " << area << endl;
if(a==anterior)
{
cont++;
if(cont==1)
{
if(area>=maior1)
{
maior2=maior1;
maior1=area;
jog2=jog1;
jog1=ponto;
}
else
{
maior2=area;
jog2=ponto;
}
}
else
{
if(area>=maior1)
{
maior2=maior1;
maior1=area;
jog2=jog1;
jog1=ponto;
}
if((area<maior1)&&(area>=maior2))
{
maior2=area;
jog2=ponto;
}
}
//circle( frame, ponto, 20, Scalar(0, 255, 255), -1);
}
else
{
cont=0;
conta_frame++;
frame = quadra.clone();
//circle( frame, ponto, 20, Scalar(0, 255, 255), -1);
maior1=area;
jog1=ponto;
if(conta_frame>0)
{
circle( frame, jog1, 20, Scalar(0, 255, 255), -1);
if(conta_frame>0) circle( frame, jog2, 20, Scalar(0, 0, 255), -1);
writer<< frame;
}
}
anterior=a;
}

arq.close();
```

}

Anexo I

Anexo 1

Funcao MATLAB de deteccao de objetos a partir de filtro de Kalman:

```
function multiObjectTracking()

% Create system objects used for reading video, detecting moving objects,and displaying the results.
obj = setupSystemObjects();

tracks = initializeTracks(); % Create an empty array of tracks.

nextId = 1; % ID of the next track

% Detect moving objects, and track them across video frames.
while ~isDone(obj.reader)
frame = readFrame();
[centroids, bboxes, mask] = detectObjects(frame);
predictNewLocationsOfTracks();
[assignments, unassignedTracks, unassignedDetections] = ...
detectionToTrackAssignment();

updateAssignedTracks();
updateUnassignedTracks();
deleteLostTracks();
createNewTracks();

displayTrackingResults();
end

function obj = setupSystemObjects()
% Initialize Video I/O
% Create objects for reading a video from a file, drawing the tracked
% objects in each frame, and playing the video.

% Create a video file reader.
obj.reader = vision.VideoFileReader('video1.wmv');

% Create two video players, one to display the video,
% and one to display the foreground mask.
obj.videoPlayer = vision.VideoPlayer('Position', [20, 400, 700, 400]);
obj.maskPlayer = vision.VideoPlayer('Position', [740, 400, 700, 400]);

% Create system objects for foreground detection and blob analysis

% The foreground detector is used to segment moving objects from
% the background. It outputs a binary mask, where the pixel value
% of 1 corresponds to the foreground and the value of 0 corresponds
% to the background.
```

```
obj.detector = vision.ForegroundDetector('NumGaussians', 3, ...
'NumTrainingFrames', 40, 'MinimumBackgroundRatio', 0.7);

% Connected groups of foreground pixels are likely to correspond to moving
% objects.  The blob analysis system object is used to find such groups
% (called 'blobs' or 'connected components'), and compute their
% characteristics, such as area, centroid, and the bounding box.

obj.blobAnalyser = vision.BlobAnalysis('BoundingBoxOutputPort', true, ...
'AreaOutputPort', true, 'CentroidOutputPort', true, ...
'MinimumBlobArea', 400);
end

function tracks = initializeTracks()
% create an empty array of tracks
tracks = struct(...
'id', {}, ...
'bbox', {}, ...
'kalmanFilter', {}, ...
'age', {}, ...
'totalVisibleCount', {}, ...
'consecutiveInvisibleCount', {});
end

function frame = readFrame()
frame = obj.reader.step();
end

function [centroids, bboxes, mask] = detectObjects(frame)

% Detect foreground.
mask = obj.detector.step(frame);

% Apply morphological operations to remove noise and fill in holes.
mask = imopen(mask, strel('rectangle', [3,3]));
mask = imclose(mask, strel('rectangle', [15, 15]));
mask = imfill(mask, 'holes');

% Perform blob analysis to find connected components.
[~, centroids, bboxes] = obj.blobAnalyser.step(mask);
end

function predictNewLocationsOfTracks()
for i = 1:length(tracks)
bbox = tracks(i).bbox;

% Predict the current location of the track.
predictedCentroid = predict(tracks(i).kalmanFilter);

% Shift the bounding box so that its center is at
% the predicted location.
predictedCentroid = int32(predictedCentroid) - bbox(3:4) / 2;
tracks(i).bbox = [predictedCentroid, bbox(3:4)];
end
end

function [assignments, unassignedTracks, unassignedDetections] = ...
detectionToTrackAssignment()

nTracks = length(tracks);
nDetections = size(centroids, 1);

% Compute the cost of assigning each detection to each track.
cost = zeros(nTracks, nDetections);
```

```
for i = 1:nTracks
cost(i, :) = distance(tracks(i).kalmanFilter, centroids);
end

% Solve the assignment problem.
costOfNonAssignment = 20;
[assignments, unassignedTracks, unassignedDetections] = ...
assignDetectionsToTracks(cost, costOfNonAssignment);
end

function updateAssignedTracks()
numAssignedTracks = size(assignments, 1);
for i = 1:numAssignedTracks
trackIdx = assignments(i, 1);
detectionIdx = assignments(i, 2);
centroid = centroids(detectionIdx, :);
bbox = bboxes(detectionIdx, :);

% Correct the estimate of the object's location
% using the new detection.
correct(tracks(trackIdx).kalmanFilter, centroid);

% Replace predicted bounding box with detected
% bounding box.
tracks(trackIdx).bbox = bbox;

% Update track's age.
tracks(trackIdx).age = tracks(trackIdx).age + 1;

% Update visibility.
tracks(trackIdx).totalVisibleCount = ...
tracks(trackIdx).totalVisibleCount + 1;
tracks(trackIdx).consecutiveInvisibleCount = 0;
end
end

function updateUnassignedTracks()
for i = 1:length(unassignedTracks)
ind = unassignedTracks(i);
tracks(ind).age = tracks(ind).age + 1;
tracks(ind).consecutiveInvisibleCount = ...
tracks(ind).consecutiveInvisibleCount + 1;
end
end

function deleteLostTracks()
if isempty(tracks)
return;
end

invisibleForTooLong = 20;
ageThreshold = 8;

% Compute the fraction of the track's age for which it was visible.
ages = [tracks(:).age];
totalVisibleCounts = [tracks(:).totalVisibleCount];
visibility = totalVisibleCounts ./ ages;

% Find the indices of 'lost' tracks.
lostInds = (ages < ageThreshold AND visibility < 0.6) | ...
[tracks(:).consecutiveInvisibleCount] >= invisibleForTooLong;

% Delete lost tracks.
tracks = tracks(~lostInds);
end
```

```
function createNewTracks()
centroids = centroids(unassignedDetections, :);
bboxes = bboxes(unassignedDetections, :);

for i = 1:size(centroids, 1)

centroid = centroids(i,:);
bbox = bboxes(i, :);

% Create a Kalman filter object.
kalmanFilter = configureKalmanFilter('ConstantVelocity', ...
centroid, [200, 50], [100, 25], 100);

% Create a new track.
newTrack = struct(...
'id', nextId, ...
'bbox', bbox, ...
'kalmanFilter', kalmanFilter, ...
'age', 1, ...
'totalVisibleCount', 1, ...
'consecutiveInvisibleCount', 0);

% Add it to the array of tracks.
tracks(end + 1) = newTrack;

% Increment the next id.
nextId = nextId + 1;
end
end

function displayTrackingResults()
% Convert the frame and the mask to uint8 RGB.
frame = im2uint8(frame);
mask = uint8(repmat(mask, [1, 1, 3])) .* 255;

minVisibleCount = 8;
if ~isempty(tracks)

% Noisy detections tend to result in short-lived tracks.
% Only display tracks that have been visible for more than
% a minimum number of frames.
reliableTrackInds = ...
[tracks(:).totalVisibleCount] > minVisibleCount;
reliableTracks = tracks(reliableTrackInds);

% Display the objects. If an object has not been detected
% in this frame, display its predicted bounding box.
if ~isempty(reliableTracks)
% Get bounding boxes.
bboxes = cat(1, reliableTracks.bbox);

% Get ids.
ids = int32([reliableTracks(:).id]);

% Create labels for objects indicating the ones for
% which we display the predicted rather than the actual
% location.
labels = cellstr(int2str(ids'));
predictedTrackInds = ...
[reliableTracks(:).consecutiveInvisibleCount] > 0;
isPredicted = cell(size(labels));
isPredicted(predictedTrackInds) = {' predicted'};
labels = strcat(labels, isPredicted);
```

```
% Draw the objects on the frame.
frame = insertObjectAnnotation(frame, 'rectangle', ...
bboxes, labels);

% Draw the objects on the mask.
mask = insertObjectAnnotation(mask, 'rectangle', ...
bboxes, labels);
end
end

% Display the mask and the frame.
obj.maskPlayer.step(mask);
obj.videoPlayer.step(frame);
end

end
```