UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

ANALISADOR DE IMAGENS DE ALVOS DE COMPETIÇÕES PARA A PLATAFORMA ANDROID

ANDRÉ LUÍS HENSEL

BLUMENAU
2012

2012/2-02

ANDRÉ LUÍS HENSEL

ANALISADOR DE IMAGENS DE ALVOS DE COMPETIÇÕES PARA A PLATAFORMA ANDROID

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. Mauro Marcelo Mattos, Dr. - Orientador

BLUMENAU
2012

2012/2-02

ANALISADOR DE IMAGENS DE ALVOS DE COMPETIÇÕES PARA A PLATAFORMA ANDROID

Por

ANDRÉ LUÍS HENSEL

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

Presidente:	Prof. Mauro Marcelo Mattos, Dr. – Orientador, FURB
Membro:	Prof. Dalton Solano dos Reis, M. Sc. – FURB
Membro:	Prof. Aurélio Faustino Hoppe, FURB

Blumenau, 10 de Dezembro de 2012

Dedico este trabalho a todos aqueles que me ajudaram diretamente ou indiretamente na realização deste.

AGRADECIMENTOS

À minha família, que sempre esteve presente.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, Mauro Marcelo Mattos, por ter acreditado na conclusão deste trabalho.

Penso noventa e nove vezes e nada descubro; deixo de pensar, mergulho em profundo silêncio – e eis que a verdade se me revela.

Albert Einstein

RESUMO

Este trabalho apresenta uma aplicação para dispositivos da plataforma Android, no qual se detectou os alvos de competições de tiro e os disparos sofridos pelos mesmos através da análise em tempo real das imagens transmitidas por uma câmera IP. Foram efetuados também os cálculos e desenhos das distâncias dos disparos em relação ao centro do alvo e das suas respectivas pontuações. Para a detecção dos alvos e dos disparos, utilizou-se a biblioteca gráfica OpenCV e a técnica de processamento de imagem transformada de Hough, que é comumente utilizada na localização das formas circulares em imagens. Para o desenho das informações dos alvos e dos disparos foram utilizadas as bibliotecas gráficas OpenCV e OpenGL ES. Como resultado obteve-se a detecção dos alvos e dos disparos de forma mais precisa e rápida em relação ao processo manual, auxiliando o competidor.

Palavras-chave: Visão computacional. Processamento de imagens. Android. OpenCV.

ABSTRACT

This work presents an application for the Android platform devices, which detected the target of shooting competitions and shooting suffered by them through real-time analysis of the images transmitted by an IP camera. The calculations and drawings of the shooting distances from the center of the target and its respective scores are also made. For detection of the targets and shots was used the graphics library OpenCV and the image processing technique Hough transform, which is commonly used in locating the circular shapes in images. For the target information drawing and the shooting were used the graphics libraries OpenCV and OpenGL ES. The result was the targets and shots detection in a more accurately and rapidly mode compared to manual process, helping the competitor.

Key-words: Computer vision. Image processing. Android. OpenCV.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ADT - *Android Development Tools*

API – *Application Programming Interface*

EA – *Enterprise Architect*

FCCTE – Federação Catarinense de Caça e Tiro Esportivo

FPS – Frames Por Segundo

GB – *Giga Byte*

GPS - *Global Positioning System*

HTTP – *Hyper Text Transfer Protocol*

IP – *Internet Protocol*

MB – *Mega Byte*

OHA – *Open Handset Alliance*

RF – Requisito Funcional

RBG – *Red Green Blue*

RNF – Requisito Não Funcional

SVC – Sistema de Visão Computacional

UML – *Unified Modeling Language*

VBO – *Vertex Buffer Object*

SUMÁRIO

1 INTRODUÇÃO

Os dispositivos móveis têm evoluído de forma notável nos últimos tempos tanto na sua capacidade de processamento, armazenamento e também na quantidade de funcionalidades disponíveis. Segundo Schemberger, Freitas e Vani (2009), esta evolução foi motivada pelo rápido crescimento no número de consumidores. Diante deste cenário, surge a plataforma Android, para dispositivos móveis, tendo como base um sistema operacional Linux e um conjunto de bibliotecas para desenvolvimento na linguagem Java (GOOGLE, 2012b).

Estes dispositivos móveis podem ser empregados nos esportes, que cada vez mais utilizam-se da tecnologia para buscar evitar erros humanos na análise de lances ou situações que podem definir resultados de um jogo ou mesmo de um campeonato. Dentre os esportes pode-se destacar o futebol que poderia utilizar-se do processamento de imagens para auxiliar no cálculo e visualização da linha do impedimento, a fim de auxiliar os árbitros e evitar erros humanos. Outro esporte que pode ser auxiliado pelo uso do processamento de imagens são as competições de tiro esportivo que acontecem em âmbito nacional.

Em Santa Catarina as competições de tiro esportivo são organizadas pela Federação Catarinense de Caça e Tiro Esportivo (FCCTE), que possui sua sede em Blumenau. Uma das principais modalidades de tiro esportivo é a modalidade de carabina apoiada. Nesta modalidade o competidor utiliza-se de uma carabina com o calibre de munição 22, para acertar um alvo a 50 metros de distância, tendo sua pontuação resultada com base na distância do disparo efetuado com o centro do alvo. Em algumas das competições, para facilitar a análise e o cálculo das pontuações, apenas os valores inteiros são considerados, podendo o disparo variar entre zero (0) e dez (10). Porém, principalmente nas competições estaduais é necessário uma análise e um cálculo mais preciso das pontuações, sendo então considerado o valor decimal. Neste caso o disparo pode variar entre zero (0) e 10,9.

Diante do exposto desenvolveu-se uma aplicação para dispositivos móveis que possibilite auxiliar os competidores na análise dos seus disparos em alvos e que ao mesmo tempo busque auxiliar o organizador de uma competição na apuração com precisão dos resultados obtidos pelos competidores.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho é desenvolver uma aplicação para dispositivos móveis da plataforma Android, fazendo uso de processamento de imagens para auxiliar a análise de alvos de competições de tiro.

Os objetivos específicos do trabalho são:

a) disponibilizar um módulo de identificação de alvos e de seus disparos;
b) disponibilizar um módulo de destaque de alvos e de seus disparos.

1.2 ESTRUTURA DO TRABALHO

A estrutura deste trabalho está apresentada em quatro capítulos. O segundo capítulo contém a fundamentação teórica necessária para o entendimento deste trabalho.

O terceiro capítulo apresenta como foi desenvolvida a aplicação na plataforma Android, os casos de uso da aplicação, os diagramas de classe e toda especificação que define a aplicação. Ainda no terceiro capítulo são apresentadas as partes principais da implementação e também os resultados e discussões que aconteceram durante toda a etapa de desenvolvimento do trabalho.

Por fim, o quarto capítulo refere-se às conclusões do presente trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

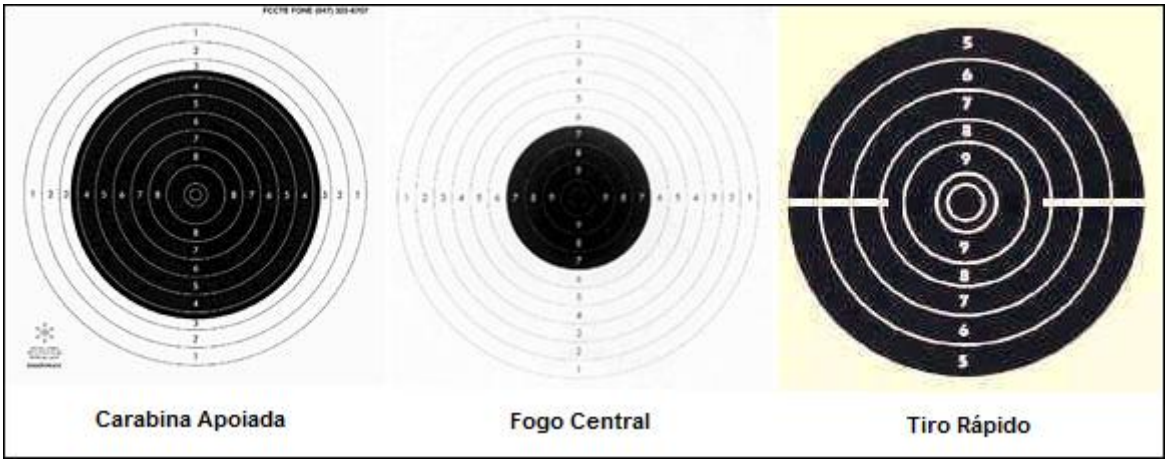
Neste capítulo são apresentados os assuntos e técnicas utilizadas para o desenvolvimento e entendimento deste trabalho. Na seção 2.1 é feito uma descrição da competição de tiro esportivo. Na seção 2.2 é feito uma introdução a visão computacional, do operador Canny e da transformada de Hough. Na seção 2.3 é feito uma descrição da biblioteca OpenCV. Na seção 2.4 é feito uma descrição da biblioteca OpenGL ES. Por fim, na seção 2.5 são descritos os trabalhos correlatos.

2.1 COMPETIÇÃO DE TIRO ESPORTIVO

Segundo a Confederação Brasileira de Tiro Esportivo (CBTE, 2012) o tiro foi lançado como esporte no século 16, com competições entre clubes na Europa ocorrendo principalmente no primeiro dia do ano, em feriados religiosos e em ocasiões especiais. Mais tarde foi amplamente conhecido como tiro esportivo através da sua inclusão nos jogos olímpicos.

No estado de Santa Catarina as competições de tiro esportivo são organizadas pela Federação Catarinense de Caça e Tiro Esportivo (FCCTE), que possui sua sede e fórum na cidade de Blumenau. A FCCTE é o órgão estadual responsável por promover campeonatos estaduais nas modalidades de carabina apoiada, tiro ao prato, armas curtas e carabina livre (FCCTE, 2012c).

As modalidades utilizam alvos de diferentes cores e tamanhos, conforme podemos ver alguns exemplos na Figura 1 (FCCTE, 2012a). O calibre de munição também varia entre as modalidades, sendo o mais comum o calibre 22 e depois o calibre 32 (FCCTE, 2012b).



Fonte: FCCTE (2012a).

Figura 1 - Tipos de alvo

2.1.1 Modalidade de carabina apoiada

A modalidade de carabina apoiada é bastante praticada pelos clubes de caça e tiro de Santa Catarina. Nesta modalidade o competidor utiliza uma carabina com o calibre de munição 22 para acertar um alvo que fica localizado a 50 metros de distância num estande de tiro coberto e com iluminação (FCCTE, 2012d).

Durante uma etapa da competição desta modalidade o competidor deve utilizar 22 alvos, sendo eles alvos oficiais de 10 zonas (Figura 2) fornecidos pela FCCTE. Os 2 (dois) primeiros são considerados alvos de ensaio, onde o competidor pode efetuar quantos disparos achar necessário afim de regulagem da arma. Nestes alvos a pontuação não é considerada. Os demais 20 (vinte) alvos são considerados alvos válidos, onde é permitido apenas 1 (um) disparo por alvo, sendo eles considerados para o cálculo da pontuação final e total alcançada pelo competidor durante a etapa. No caso da existência de mais de 1 (um) disparo num alvo válido, apenas o disparo de menor valor é considerado para o cálculo da pontuação. Ao todo o competidor tem direito há 35 minutos para realizar todos os seus disparos nos alvos e finalizar sua prova (FCCTE, 2012d).

Fonte: FCCTE (2012a).
Figura 2 - Alvo da modalidade de carabina apoiada

Ao final da prova do competidor os alvos são recolhidos e encaminhados para a apuração dos resultados. Para efetuar a apuração são utilizadas máquinas próprias para tal fim fornecidas pela FCCTE. Esta máquina faz uma leitura dos alvos identificando os disparos e gravando a pontuação sobre o alvo. Os valores inteiros das pontuações são considerados para a classificação e premiação do competidor e os valores decimais são considerados para critérios de desempate. Os resultados finais são impressos e entregues ao competidor (FCCTE, 2012d).

2.2 VISÃO COMPUTACIONAL

Visão computacional pode ser definida como um conjunto de métodos e técnicas através dos quais sistemas computacionais podem ser capazes de interpretar imagens (WANGENHEIM, 1998).

Segundo Trindade (2009) o desenvolvimento de sistemas de visão computacional requer uma entrada de dados geralmente obtida através de sensores, câmeras ou vídeos. Estas imagens são processadas e transformadas em uma informação esperada, como por exemplo, receber uma imagem colorida, binarizar a imagem, exibir a imagem preta e branca e níveis de cinza. O processo de transformação da imagem é realizado por métodos contidos em bibliotecas de processamento gráfico, como por exemplo, a biblioteca OpenCV.

Ainda segundo Trindade (2009) a organização de um Sistema de Visão Computacional (SVC) não possui um modelo fixo, mas algumas etapas são muito comuns na maioria dos

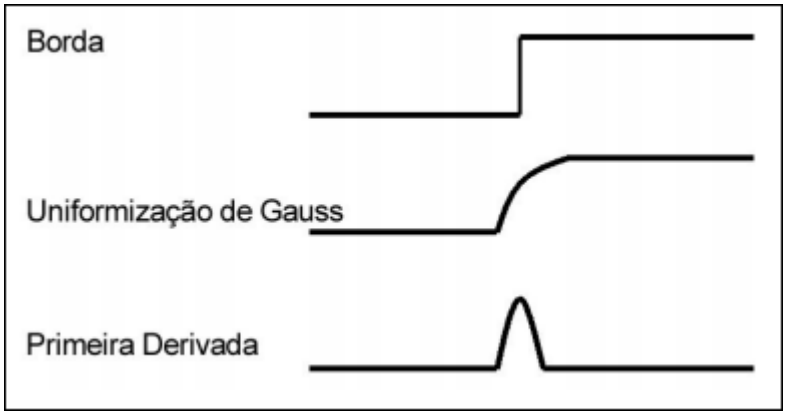
projetos que utilizam SVC.

Dentre as etapas comuns de um SVC podem-se citar a etapa de aquisição, que consiste na captura de uma ou mais imagens através de um sensor geralmente contido em uma câmera digital, a etapa de pré-processamento onde um ou mais métodos de processamento de imagem são aplicados na imagem capturada, como por exemplo, um método de detecção de bordas, e a etapa de extração de características que consiste na extração de informações da imagem, como por exemplo, formas, cantos, bordas, etc.

2.2.1 Operador Canny

Conforme Bueno (2000), as bordas de uma imagem é o resultado de mudanças em alguma propriedade física ou especial de superfícies iluminadas. A maioria dos algoritmos de detecção de bordas empregam operadores diferenciais de primeira ou segunda ordem. Estes operadores ressaltam o contorno das bordas mas também ampliam o ruído da cena. Para suavizar a imagem, o operador Canny utiliza uma máscara Gaussiana e isto também pode atenuar as bordas fracas, onde o contraste é pequeno.

Segundo Conci (2010), o operador Canny é um filtro de convolução que usa a primeira derivada. Ele suaviza o ruído e localiza as bordas combinando um operador diferencial com o filtro Gaussiano (Figura 3).



Fonte: Conci (2010, p. 1).

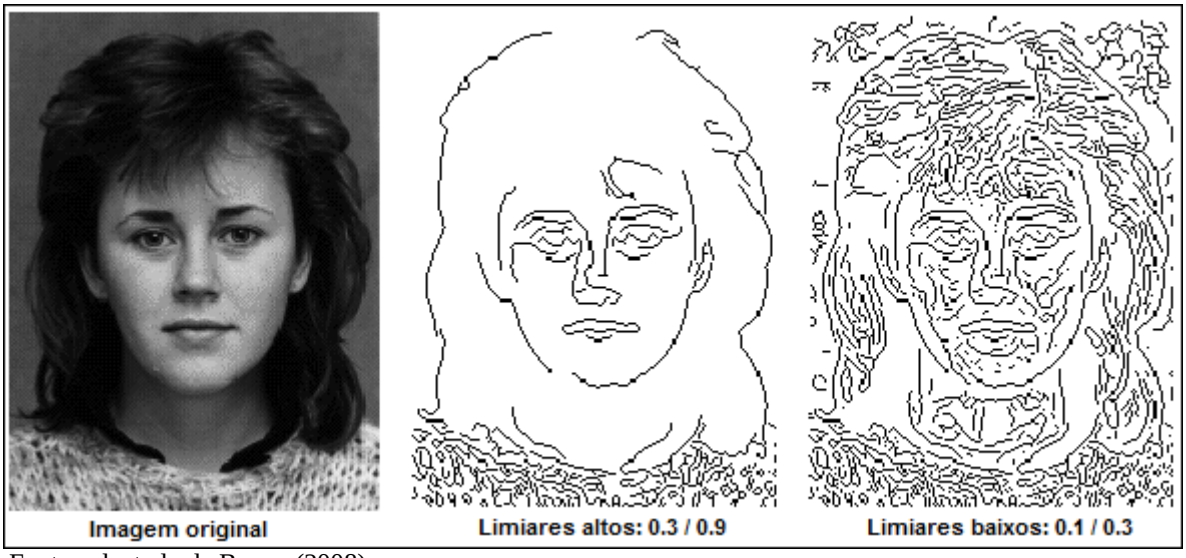
Figura 3 - Processo de detecção de bordas por Canny

Ainda segundo Conci (2010), o alvo do algoritmo de Canny é obter um método de detecção ótimo de borda. Isto significa que o algoritmo deve marcar tantas bordas quanto possível, as bordas marcadas devem estar tão perto quanto possível da borda real, cada borda na imagem deve ser marcada uma vez e o ruído não deve criar bordas falsas.

O algoritmo de Canny pode ser dividido em quatro etapas. Na primeira etapa é

realizada a redução de ruído através da convolução da imagem por uma máscara Gaussiana. Na segunda etapa ocorre o cálculo dos gradientes da intensidade da imagem, gerando uma máscara para cada sentido possível de uma borda (horizontal, vertical e duas diagonais). Na terceira etapa os resultados da primeira etapa com cada uma das máscaras da segunda etapa são armazenados. Para cada *pixel*, marca-se então o maior resultado do gradiente nesse *pixel* e o sentido da máscara que produziu essa borda. Na quarta e última etapa são utilizados dois limiares, um superior e outro inferior, para definir o valor da intensidade do gradiente para ser considerado uma borda. Por fim tem-se uma imagem binária onde cada *pixel* é marcado como um *pixel* de borda ou de não borda (CONCI, 2010).

Segundo Bueno (2000), considerando um segmento de borda, para todo valor superior ao limite de limiarização ele é automaticamente aceito e para todo valor abaixo do limite inferior de limiarização ele é imediatamente rejeitado. Pontos situados entres os dois limites serão aceitos se estiverem relacionados aos pixels que apresentam respostas fortes. A Figura 4 mostra a utilização do operador Canny com valores de limiares altos e baixos.



Fonte: adaptado de Bueno (2008).

Figura 4 - Aplicação do operador Canny e diferentes limiares

2.2.2 Transformada de Hough

Segundo Jamundá (2000), a transformada de Hough é um método padrão para detecção de formas que são facilmente parametrizadas (linhas, círculos, elipses) em imagens computacionais. De modo geral, a transformada é aplicada após a imagem sofrer um pré-processamento, comumente uma detecção de bordas.

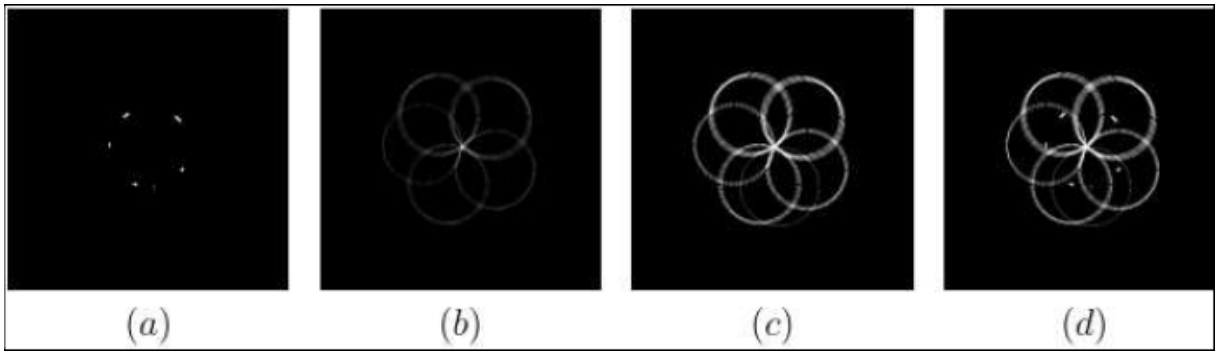
Ainda segundo Jamundá (2000), uma imagem computacional é formada por um conjunto de pontos que possuem determinadas características, os quais são chamados de *pixels*. A transformada de Hough consiste em mapear um *pixel* da imagem em uma curva de espaço de parâmetros, organizado na forma de um acumulador multi dimensional.

Segundo Pistori, Pistori e Costa (2005, p. 2-5), a transformada de Hough pode ser calculada tomando a informação do gradiente a partir de uma imagem em preto e branco, e acumulando cada ponto não nulo a partir da imagem gradiente em todo ponto que está a um determinado raio de distância dele.

Segundo Poffo (2010, p. 21-22), dessa forma todos os pontos que se encontram ao longo do contorno de um círculo de um determinado raio contribuem para a transformação no centro do círculo, e assim, os picos na imagem transformada correspondem aos centros de características circulares de um determinado tamanho na imagem original.

Ainda segundo Poffo (2010, p. 22), após a detecção de um pico e um possível círculo encontrado em um determinado ponto, pontos próximos (em uma distância dentro da metade do raio original) são excluídos como possíveis centros para evitar a detecção da mesma característica circular repetidamente.

A Figura 3 mostra a transformação de Hough, onde a Figura 5(a) é a imagem original contendo um círculo altamente corrompido, a Figura 5(b) seu respectivo espaço de Hough, a Figura 5(c) é o mesmo espaço de Hough com ajuste de contraste para facilitar a visualização e a Figura 5(d) é a imagem composta pela adição da imagem original com o espaço de Hough.



Fonte: Pistori, Pistori e Costa (2005, p. 3).

Figura 5 - Transformação circular de Hough

2.3 OPEN SOURCE COMPUTER VISION (OPENCV)

O OpenCV é uma biblioteca de funções de programação para computação gráfica em

tempo real. Possui interfaces em C, C++, Python e Java, rodando tanto em sistemas Windows, Linux, Android e Mac (OPENCV, 2012a).

O OpenCV possui uma estrutura modular, o que quer dizer que o pacote inclui várias bibliotecas estáticas e compartilhadas. Dentre os módulos pode-se citar o `core`, que possui as definições básicas das estruturas de dados e funções básicas utilizadas pelos demais módulos, o módulo `imgproc`, que é utilizado para processamento de imagem (filtros lineares e não-lineares, transformações geométricas de imagem, histogramas); o módulo `video`, utilizado para análise de vídeos (estimativas de movimento, subtração de fundo e algoritmos de rastreamento de objetos); o módulo `calib3d`, utilizado para calibragem de câmeras e elementos de reconstrução 3D, dentre outros (OPENCV, 2012b).

A classe `Mat`, definida dentro do módulo core, representa um *array* multidimensional utilizado para armazenar diversos tipos de valores de vetores ou matrizes, imagens coloridas ou em escala de cinza, nuvens de pontos, histogramas (OPENCV, 2012c).

A função `cvtColor` pode ser utilizado para converter uma imagem, armazenada dentro de um objeto `Mat`, de um espaço de cores a outro, como por exemplo do espaço RBG para uma escala de cinza (OPENCV, 2012f).

Após a conversão da imagem para uma escala de cinza é possível aplicar um filtro na imagem, para auxiliar na detecção das bordas. Para tal finalidade é possível utilizar a função `GaussianBlur` (OPENCV, 2012e). Esta função borra a imagem através de um *kernel* de Gauss especificado.

Com a imagem convertida para uma escala de cinza e filtrada, é possível passar para o próximo passo e detectar as bordas na imagem. A função `Canny` pode então ser aplicada na imagem para detectar as bordas. Esta função encontra as bordas na imagem através do algoritmo de Canny (OPENCV, 2012d).

Por fim para a detecção de círculos a biblioteca OpenCV possui a implementação da transformada de Hough através da função `HoughCircles`. Esta função localiza círculos numa imagem em escala de cinza, sendo possível auxiliar na detecção informando o raio mínimo e máximo dos círculos (OPENCV, 2012d).

Na implementação da OpenCV para a plataforma Android não é necessário a pré-detecção das bordas numa imagem para a utilização da transformada de Hough, pois a função `HoughCircles` faz a utilização do algoritmo de Canny internamente.

2.4 OPENGL ES

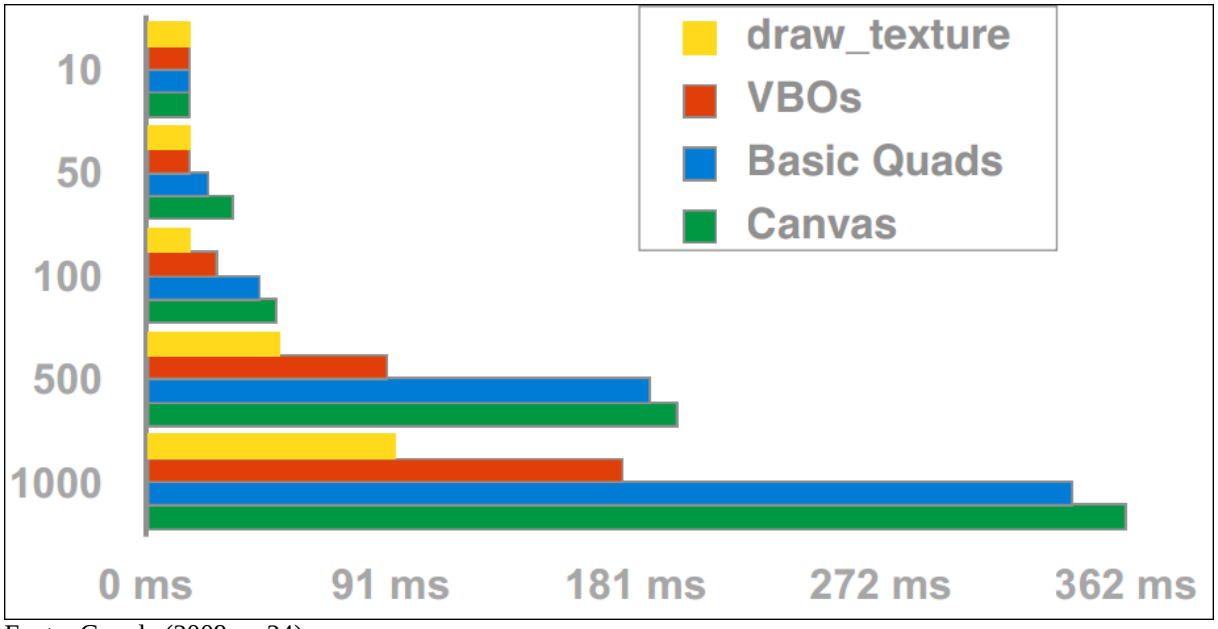
Segundo Cohen e Manssour (2006, p. 15), “pode-se definir OpenGL como uma especificação aberta e multiplataforma de uma biblioteca de rotinas gráficas e de modelagem utilizada para o desenvolvimento de aplicações de Computação Gráfica.”

A OpenGL ES consiste numa subseção da API para *desktops* OpenGL, criando uma flexível e poderosa interface de baixo nível entre o software e o acelerador gráfico (KHRONOS GROUP, 2012).

A versão 1.0 do OpenGL ES foi projetada de acordo com a versão 1.3 do OpenGL, já a versão 1.1 do OpenGL ES é baseada na versão 1.5 do OpenGL e a versão 2.0 do OpenGL ES é baseada na versão 2.0 do OpenGL (KHRONOS GROUP, 2012).

Dentre as principais diferenças que pode-se destacar é a ausência do suporte aos comandos `glBegin` e `glEnd`, ausências das primitivas `QUADS`, `QUAD_STRIP` e `POLYGON` e o uso de *arrays* de vértices para desenhar os objetos geométricos (KHRONOS GROUP, 2004).

Testes de desempenho feitos pela Google (2009, p. 23-25) dentro de um dispositivo G1, que é um dispositivo desenvolvido pela Google em parceria com a HTC, comparam o uso de Canvas e OpenGL na renderização de grande quantidades de imagens 2D (Figura 6).



Fonte: Google (2009, p. 24).

Figura 6 - OpenGL ES versus Canvas

Nos testes foram comparadas quatro funcionalidades diferentes no desenho de objetos 2D, as três primeiras fazem uso da OpenGL ES e a última faz o desenho através da classe `Canvas`. O comparativo é feito através do número de objetos desenhados pelo tempo.

A primeira funcionalidade utiliza a `GL_OES_draw_texture` para desenhar as imagens e a segunda funcionalidade utiliza vetores de vértices para desenhar as texturas na forma de *buffers* de vértices, conhecidos como *Vertex Buffer Object* (VBO). Ambas funcionalidades não são suportadas por todos os dispositivos que utilizam a plataforma Android (GOOGLE, 2009, p. 23-25).

A terceira funcionalidade desenha as texturas através de um plano de vértices básicos usando a memória principal do dispositivo e com projeção ortográfica. Já que essa funcionalidade requer apenas os recursos básicos do OpenGL ES 1.0, é suportado por todos os dispositivos (GOOGLE, 2009, p. 23-25).

A última funcionalidade desenha as texturas através de objetos do tipo `Bitmap` em um objeto `Canvas`. Esta funcionalidade se utiliza da unidade central de processamento do dispositivo e também é a mais simples de ser implementada (GOOGLE, 2009, p. 23-25).

Com isso pode-se concluir que a utilização da classe `Canvas`, embora mais simples de ser utilizada, tem um desempenho muito baixo, não sendo aconselhado para desenhar muitos objetos na tela. Já a utilização do OpenGL ES é aconselhável para a utilização em aplicações que necessitam de altas taxas de atualizações de quadros e que usam rotações e escala (GOOGLE, 2009, p. 23-25).

2.5 TRABALHOS CORRELATOS

É possível encontrar-se vários trabalhos que descrevem o uso de processamento de imagens para auxílio ao usuário. Dentre os trabalhos acadêmicos foram destacados três: o de Poffo (2010), o de Starosky (2003) e o de Vasselai (2010).

2.5.1 Visual Autonomy – Protótipo para reconhecimento de placas de trânsito

O trabalho de Poffo (2010) descreve um protótipo de uma ferramenta, que tem por objetivo detectar e reconhecer placas de trânsito.

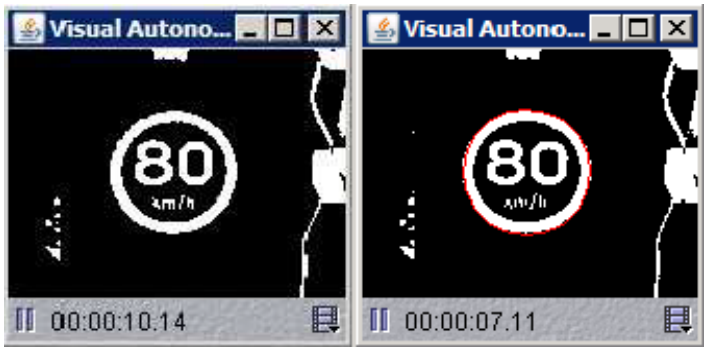
No trabalho foram utilizadas técnicas de processamento de imagens, como a transformada de Hough, que foi utilizada para localizar o formato circular de um tipo de placa

de trânsito.

A transformada de Hough, segundo Pedrini e Schwartz (2008, p. 174-182), é uma técnica de segmentação para detecção de um conjunto de pontos em uma imagem digital que pertençam a uma forma parametrizada, assim como linhas, círculos, elipses e outras.

Na etapa de reconhecimento, Poffo (2010) submeteu a imagem a uma rede neural *multilayer perceptron* treinada usando o algoritmo *backpropagation* e para diminuir a ocorrência de falso-positivos na detecção de placas, foi utilizado o operador Sobel de detecção de borda, que conforme Pedrini e Schwartz (2008, p. 179), calcula a informação de gradiente.

A Figura 7 mostra a detecção de uma placa através da utilização da transformada de Hough.



Fonte: Poffo (2010, p. 42).

Figura 7 - Placa detectada

2.5.2 Calibração de câmeras para utilização no cálculo de impedimentos de jogadores de futebol a partir de imagens

O trabalho de Starosky (2003, p. 7) apresenta um método para a calibragem de câmeras utilizadas no cálculo do impedimento e da distância entre dois pontos dentro do campo de futebol para sistemas do tipo “tira-teima”. Para alcançar este objetivo, o autor usa uma imagem estática de lance de jogos de futebol.

No trabalho foi proposto um método que utiliza uma imagem *raster* de um modelo de campo, onde são informados apenas quatro pontos de referência que são definidos na imagem, gerando um sistema de coordenadas tridimensional. Após isso 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 distância entre os dois pontos

do campo.

A Figura 8 mostra a distância calculada entre dois pontos em uma imagem estática de um lance de jogo de futebol.

Fonte: Starosky (2003, p. 49).

Figura 8 - Tela apresentando a distância calculada

2.5.3 Um estudo sobre realidade aumentada para a plataforma Android

O trabalho de Vasselai (2010) apresenta uma aplicação com realidade aumentada utilizando coordenadas geográficas para registro dos objetos virtuais, chamados pontos de interesse, desenvolvida para dispositivos móveis da plataforma Android. A impressão de aumento de realidade é garantida através do uso da câmera de vídeo, na qual os objetos virtuais são desenhados acima das imagens da câmera. A interação da realidade com a virtualidade ocorre através da movimentação do dispositivo, acionando a bússola e o acelerômetro.

A biblioteca OpenGL ES foi utilizada no trabalho para desenhar os pontos de interesse na tela, sendo estes pontos representados por setas e painéis que direcionam para a localização de cada ponto. Já a interação com a aplicação foi feita através do uso da bússola e do acelerômetro do dispositivo móvel. Além disso, a localização do dispositivo determina a distância em que se encontram os pontos de interesse.

Para apresentar os pontos de interesse, Vasselai (2010) sobrepôs três camadas de tela, sendo a primeira camada do OpenGL ES, a segunda camada da câmera e por fim a camada

das ferramentas.

A Figura 9 e a Figura 10 mostram respectivamente o desenho de setas com a distância para os pontos de interesse e painéis com o nome dos pontos de interesse.

Fonte: Vasselai (2010, p. 85).

Figura 9 - Desenho das setas

Fonte: Vasselai (2010, p. 85).

Figura 10 - Desenho dos painéis

3 DESENVOLVIMENTO DO SISTEMA

Neste capítulo são detalhadas as etapas do desenvolvimento, abordando os principais requisitos, a especificação, a implementação e por fim são listados resultados e discussão.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O sistema deverá:

a) mostrar ao usuário a pontuação total dos disparos efetuados estatísticas de tempo entre os disparos, horário de início da prova e horário restante para o fim da prova (Requisito Funcional - RF);
b) mostrar ao usuário, projetando sobre a imagem do alvo, a pontuação e distância para o centro do alvo (RF);
c) permitir ao usuário visualizar os resultados obtidos pelo competidor (RF);
d) implementar o sistema utilizando a linguagem de programação Java (Requisito Não-Funcional - RNF);
e) utilizar o ambiente de desenvolvimento Eclipse para desenvolvimento do sistema cliente (RNF);
f) utilizar a biblioteca OpenCV para a análise das imagens (RNF);
g) utilizar o banco de dados SQLite para armazenar as análises das imagens dos alvos com disparos (RNF);
h) ser executado na plataforma Android (RNF);
i) levar menos do que 1 segundo para analisar uma imagem de um alvo (RNF).

3.2 ESPECIFICAÇÃO

A especificação do sistema apresentado utiliza alguns dos diagramas da *Unified Modeling Language* (UML) em conjunto com a ferramenta Enterprise Architect (EA) 7.5 para a elaboração dos diagramas de casos de uso, de classes e de sequência. Alguns diagramas

estão na sua forma resumida para facilitar sua visualização.

3.2.1 Casos de Uso

Nesta seção são descritos os casos de uso de todos os recursos do sistema. Na Figura 11 é apresentado o diagrama de casos de uso da aplicação.

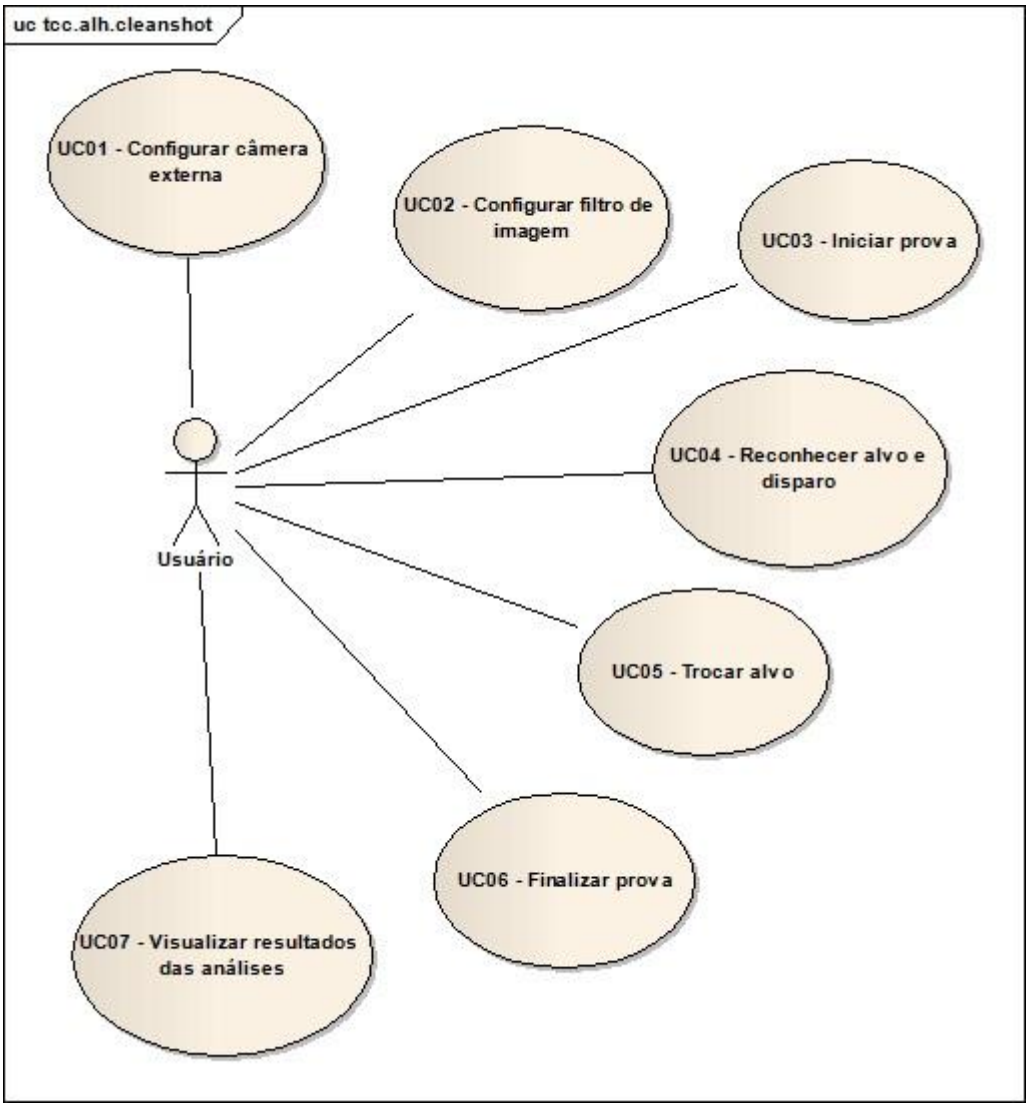


Figura 11 - Diagrama de casos de uso

3.2.1.1 Configurar câmera externa

Este caso de uso descreve o processo de configuração e inicialização da captura de

imagens através de uma câmera externa. Detalhes deste caso de uso estão descritos no Quadro 1.

UC01 – Configurar câmera externa	
Descrição	Informar as configurações de acesso a uma câmera externa.
Pré-Condição	Uma câmera externa deve estar ligada e com as configurações de acesso remoto informadas.
Cenário Principal	1. O `Usuário` entra na aplicação de análise de imagens; 2. O `Usuário` pressiona o botão `Configurar Câmera`; 3. A aplicação mostra a tela de configurações da câmera; 4. O `Usuário` pressiona o botão `Acesso Remoto`; 5. A aplicação mostra a tela de informações de câmera; 6. O `Usuário` informa o endereço IP da câmera, o nome do usuário e a senha de acesso; 7. O `Usuário` pressiona o botão `OK` para confirmar as informações; 8. A aplicação tenta validar as informações.
Exceção	Se no passo 6 do cenário principal o `Usuário` informar algum dado inválido a aplicação mostrará uma mensagem na tela informando que não foi possível acessar a câmera.
Pós-Condição	As imagens capturas pela câmera são apresentadas ao `Usuário`.

Quadro 1 – Caso de uso UC01

3.2.1.2 Configurar filtro de imagem

Este caso de uso descreve o processo de configuração do filtro Canny e do raio do alvo. Detalhes deste caso de uso estão descritos no Quadro 2.

UC02 – Configurar filtro de imagem	
Descrição	Configurar os parâmetros do filtro Canny e o tamanho do raio máximo do alvo.
Pré-Condição	Uma câmera externa deve estar corretamente configurada e um alvo deve estar posicionado em frente à câmera.
Cenário Principal	1. O `Usuário` pressiona o botão `Menu` do equipamento; 2. O `Usuário` pressiona o botão `Configurar Filtro`; 3. O `Usuário` pressiona o botão `Ligar` do filtro Canny; 4. A aplicação apresenta a imagem aplicando o filtro Canny; 5. O `Usuário` aumenta ou diminui os valores do filtro Canny através dos botões `Mais` e `Menos`; 6. A aplicação aplica as alterações e mostra a imagem ao usuário; 7. O `Usuário` pressiona o botão `Desligar` do filtro Canny; 8. O `Usuário` aumenta ou diminui os valores do raio do alvo; 9. A aplicação aplica as alterações, analisa a imagem e mostra as informações do alvo caso o mesmo for detectado.
Exceção	Se no passo 9 não for apresentado as informações do alvo, é necessário o `Usuário` continuar a modificar as configurações até a detecção do alvo pela aplicação.
Pós-Condição	O `Usuário` poderá começar a análise das imagens capturadas pela câmera.

Quadro 2 – Caso de uso UC02

3.2.1.3 Iniciar prova

Este caso de uso descreve o processo de início da análise dos alvos. Detalhes deste caso de uso estão descritos no Quadro 3.

UC03 – Iniciar prova	
Descrição	Iniciar a análise de alvos.
Pré-Condição	O `Usuário` deve configurar corretamente o filtro no caso de uso `UC02`.
Cenário Principal	1. O `Usuário` pressiona o botão `Menu` do equipamento; 2. O `Usuário` pressiona o botão `Visualizar`; 3. O `Usuário` pressiona o botão `Iniciar`; 4. A aplicação inicia uma nova análise de alvos.
Pós-Condição	Iniciar o caso de uso `UC04`.

Quadro 3 – Caso de uso UC03

3.2.1.4 Reconhecer alvo e disparo

Este caso de uso descreve o processo de detecção de um alvo e de seus disparos. Detalhes deste caso de uso estão descritos no Quadro 4.

UC04 – Reconhecer alvo e disparo	
Descrição	A aplicação analisa as imagens procurando detectar um alvo e seus disparos, desenhando os objetos virtuais sobre a imagem do alvo.
Pré-Condição	Nenhuma pré-condição foi detectada.
Cenário Principal	1. A aplicação analisa as imagens capturas pela câmera externa buscando detectar um alvo; 2. A aplicação detecta um alvo válido e mostra as informações do alvo ao `Usuário`; 3. O `Usuário` pressiona o botão `Finalizar Troca`; 4. A aplicação inicia a análise das imagens buscando os disparos; 5. A aplicação detecta um disparo e mostra as informações do disparo ao `Usuário`.
Exceção	Se no passo 2 a aplicação não detectar corretamente um alvo ou o `Usuário` pular diretamente ao passo 3 antes da detecção de um alvo, os disparos não poderão ser analisados pela aplicação.
Pós-Condição	O `Usuário` pode iniciar o caso de uso `UC05`.

Quadro 4 – Caso de uso UC04

3.2.1.5 Trocar alvo

Este caso de uso descreve o processo de troca de um alvo. Detalhes deste caso de uso estão descritos no Quadro 5.

UC05 – Trocar alvo	
Descrição	Efetua a troca de alvo iniciando a análise de um novo alvo pela aplicação.
Pré-Condição	Nenhuma pré-condição foi detectada.
Cenário Principal	1. O `Usuário` pressiona o botão Trocar Alvo; 2. A aplicação inicia uma nova análise de alvos.
Pós-Condição	A aplicação retorna ao caso de uso UC04.

Quadro 5 – Caso de uso UC05

3.2.1.6 Finalizar prova

Este caso de uso descreve o processo de finalização da análise de alvos. Detalhes deste caso de uso estão descritos no Quadro 6.

UC06 – Finalizar prova	
Descrição	Finaliza a análise de alvos e disparos gravando as informações dos resultados obtidos durante a análise no banco de dados.
Pré-Condição	Pelo menos um alvo e um disparo devem ter sido analisados pela aplicação.
Cenário Principal	1. O `Usuário` pressiona o botão `Finalizar`; 2. A aplicação grava no banco de dados todas as análises dos disparos.
Exceção	Se no passo 2 a aplicação não conseguir acesso ao cartão de memória, a gravação das análises não será efetuada.
Pós-Condição	A aplicação monta a lista de disparos gravados e mostra na tela.

Quadro 6 – Caso de uso UC06

3.2.1.7 Visualizar resultados das análises

Este caso de uso descreve o processo de visualização das análises de alvos e disparos gravados pela aplicação. Detalhes deste caso de uso estão descritos no Quadro 7.

UC07 – Visualizar resultados das análises	
Descrição	Permite a visualização das informações das análises gravadas efetuadas pela aplicação.
Pré-Condição	Pelo menos uma análise deve existir no banco de dados.
Cenário Principal	1. O `Usuário` pressiona o botão `Consulta`; 2. A aplicação mostra uma tela com a lista de análises gravadas; 3. O `Usuário` escolhe uma das análises; 4. A aplicação monta a lista com os disparos gravados e mostra na tela; 5. O `Usuário` seleciona um dos disparos na lista; 6. A aplicação mostra as informações do disparo selecionado pelo usuário na tela.
Exceção	Se no passo 3 o `Usuário` fechar a tela com a lista das análises sem selecionar uma das análises, a aplicação não monta a lista dos disparos.
Pós-Condição	As informações do disparo serão mostradas ao `Usuário` na tela.

Quadro 7 – Caso de uso UC07

3.2.2 Diagrama de classes

Nesta seção são descritas as principais classes e estruturas desenvolvidas que formam toda a aplicação. Para facilitar o entendimento de como as classes estão reunidas, na Figura 12 estão sendo demonstrados os pacotes, suas dependências bem como as classes que os compõem.

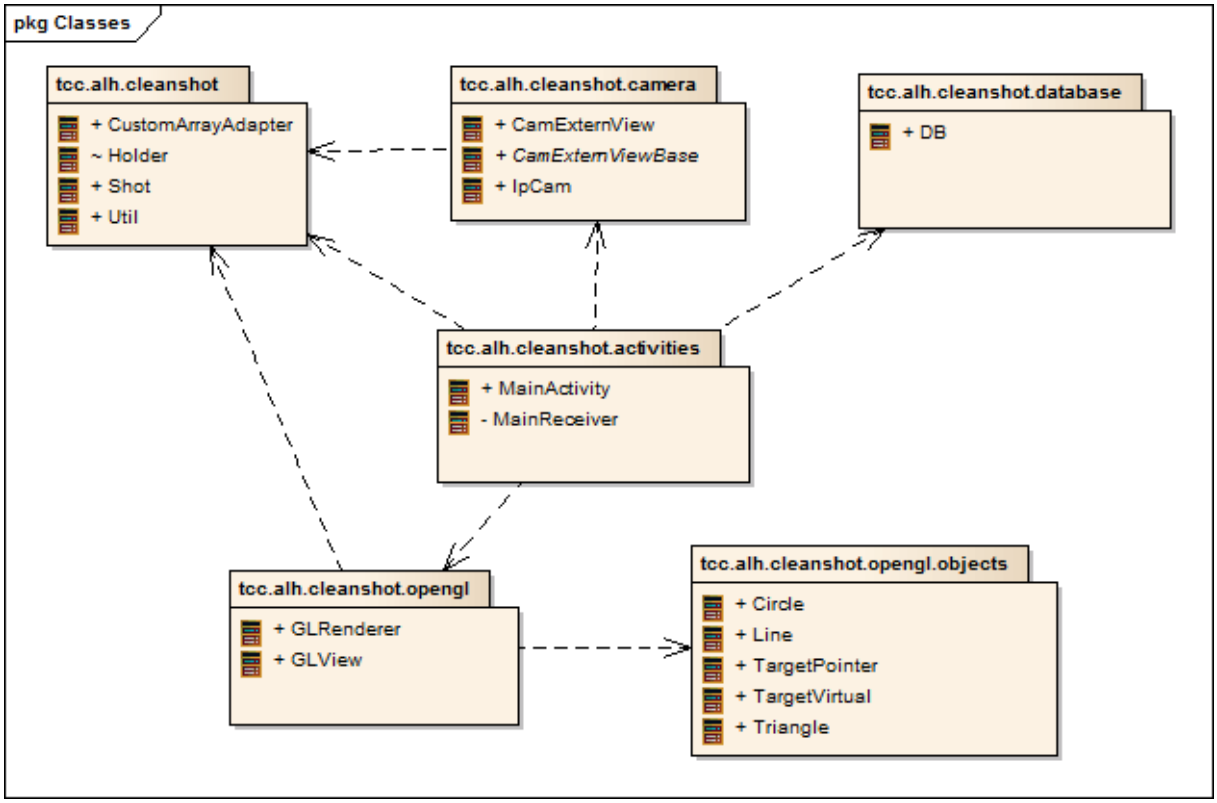


Figura 12 - Diagrama de pacotes

3.2.2.1 Pacote `tcc.alh.cleanshot`

O primeiro pacote abriga as classes `Holder`, `CustomArrayAdapter`, `Util` e `Shot` conforme apresentado na Figura 13.

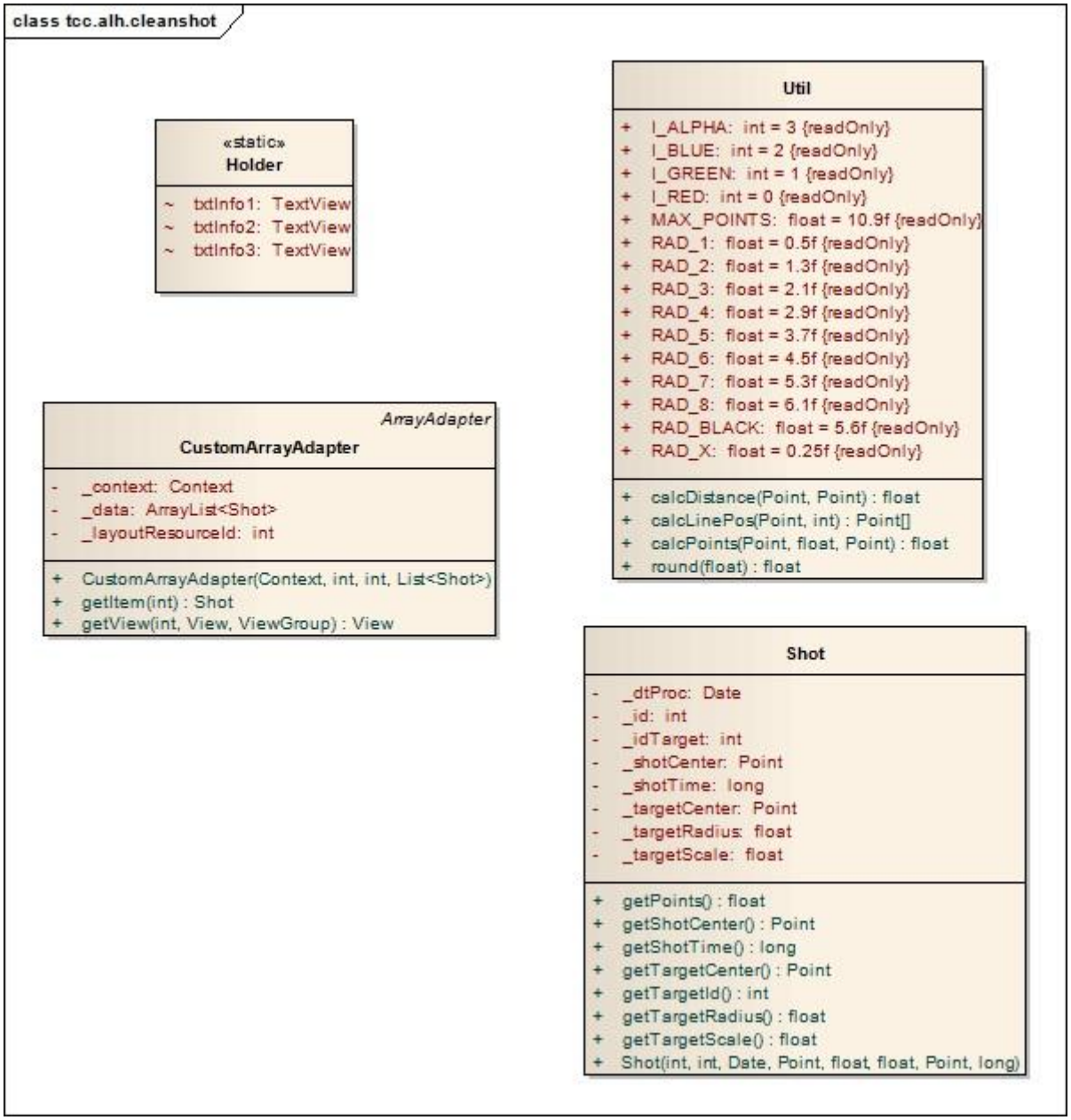


Figura 13 - Pacote `tcc.alh.cleanshot`

A classe `Holder` e a classe `CustomArrayAdapter` trabalham em conjunto e são utilizados pela classe `MainActivity` para mostrar na tela a lista de disparos efetuados pelo usuário.

A classe `Util` contém uma série de constantes utilizadas ao longo da aplicação e as funções que calculam a distância entre o centro do alvo e o centro do disparo e a pontuação de um disparo.

Por fim a classe `Shot` possui as informações que representam um disparo analisado pela aplicação e os métodos para retornar essas informações.

3.2.2.2 Pacote `tcc.alh.cleanshot.activities`

O segundo pacote abriga a classe `MainActivity` (Figura 14).

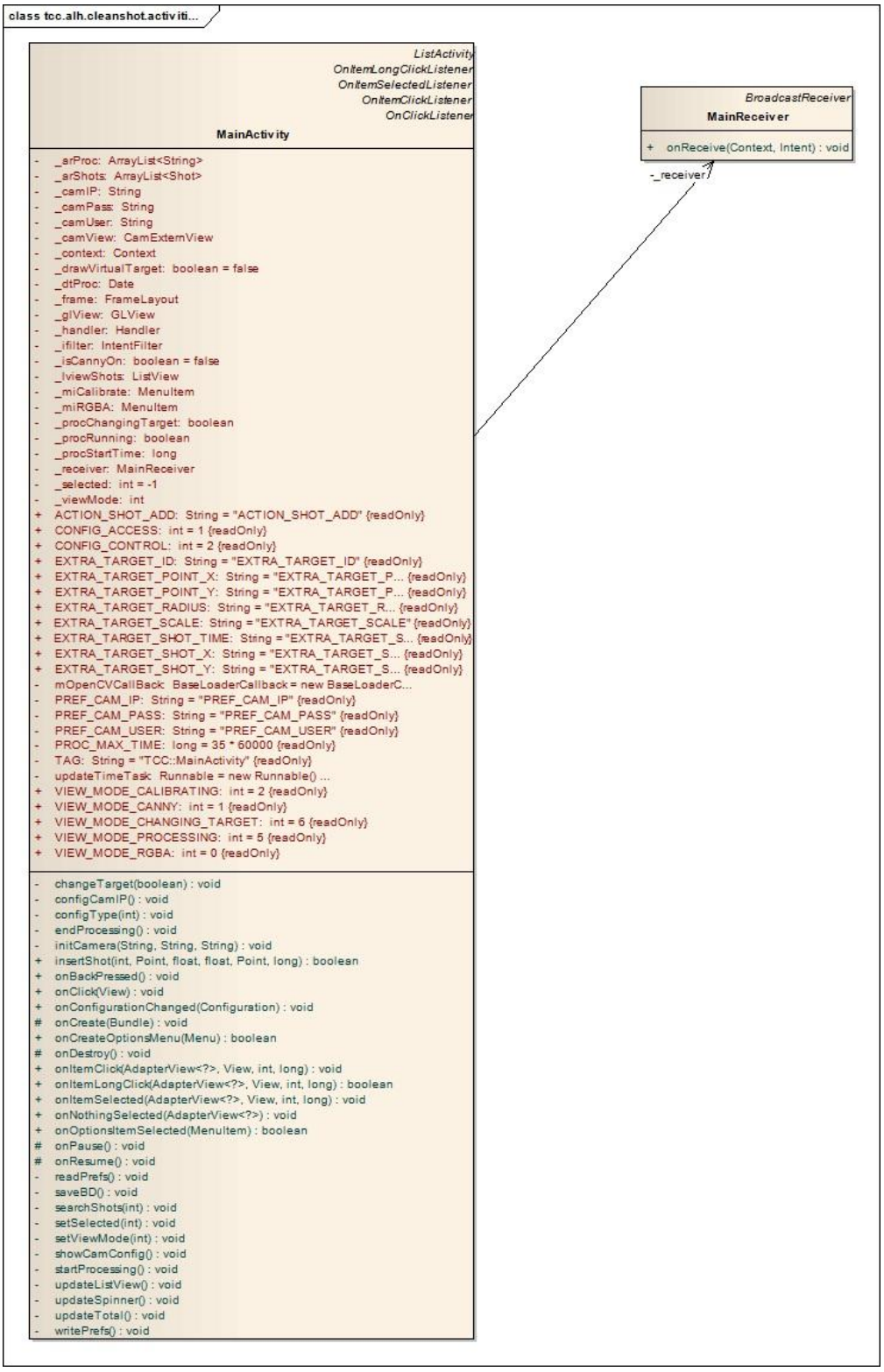


Figura 14 - Pacote `tcc.alh.cleanshot.activities`

A classe `MainActivity` possui a responsabilidade de manter o ciclo de vida da

aplicação. Esta classe é uma extensão da classe `Activity` do Android. Através do *layout* principal da tela a classe controla a interação entre as funcionalidades da aplicação e o usuário. O *layout* da tela possui um `FrameLayout` central que possui duas camadas, sendo a primeira camada responsável pelo desenho das imagens capturadas pela câmera externa e a segunda camada responsável por desenhar os objetos virtuais através da utilização da biblioteca gráfica OpenGL ES.

Através da execução do método `OpenCVLoader.initAsync` e do método de *callback* `mOpenCVCallBack` a classe verifica se a biblioteca do OpenCV, que é utilizada na análise das imagens, se encontra instalada no equipamento.

Esta classe também permite a configuração de uma câmera externa, além da inicialização das classes `GLView` e `CameraExternView` responsáveis pelo desenho dos objetos virtuais e análise das imagens capturadas pela câmera externa.

3.2.2.3 Pacote `tcc.alh.cleanshot.camera`

O terceiro pacote abriga as classes `CamExternView`, `CamExternViewBase` e `IpCam` conforme a Figura 15.

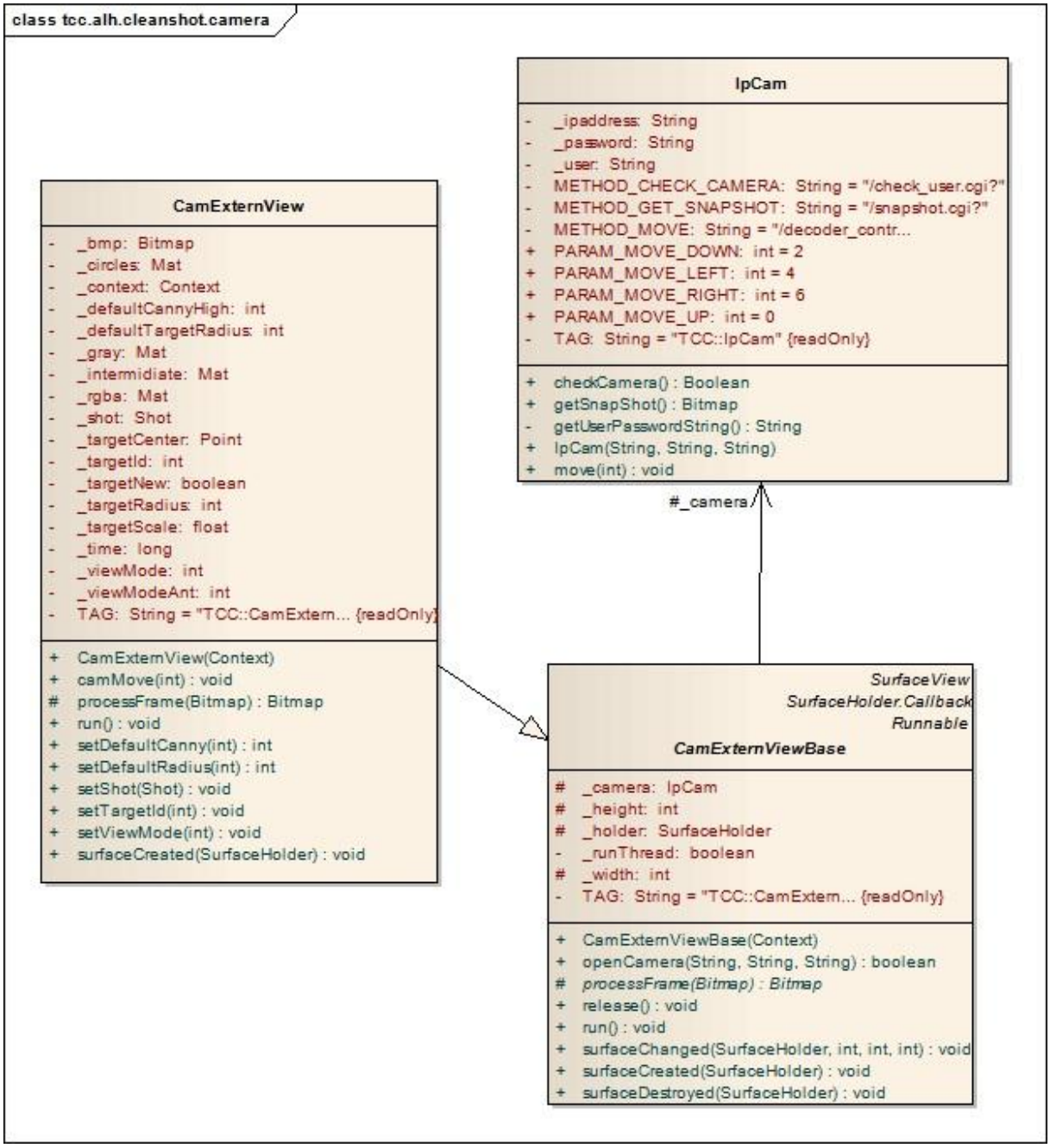


Figura 15 - Pacote `tcc.alh.cleanshot.camera`

A classe `IpCam` possui a implementação as funções de acesso a câmera IP utilizada no desenvolvimento desta aplicação. Dentre elas destacam-se a função `checkCamera` que é utlizada para verificar se as informações passadas pelo usuário são válidas e a função `getSnapShot` que solicita uma imagem a câmera externa.

A classe `CamExternViewBase` é uma extenção da classe `SurfaceView` do Android. Ela utiliza a classe `IpCam` para solicitar e receber as imagens da câmera, solicitar a análise das imagens e disponibilizá-las no objeto `SurfaceHolder`.

A última classe do pacote é a classe `CamExternView` que é uma extensão da classe `CamExternViewBase` implementando o método abstrato `processFrame`. Este método é

responsável pela análise dos alvos e dos disparos além de desenhar sobre as imagens algumas informações referentes às análises efetuadas.

3.2.2.4 Pacote `tcc.alh.cleanshot.database`

O quarto pacote é composto pela classe DB conforme Figura 16.

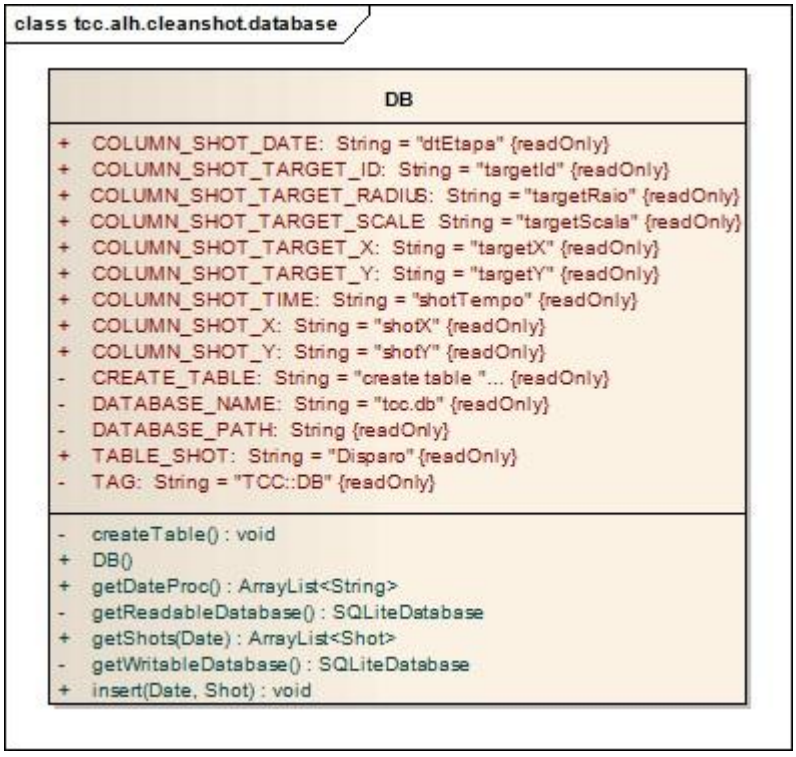


Figura 16 - Pacote `tcc.alh.cleanshot.database`

A classe `DB` contém as constantes para criação da base de dados e da tabela que armazenará as análises dos disparos efetuados pela aplicação. A classe também implementa os métodos necessários para inserir novas análises e consultar as análises gravadas no banco de dados.

3.2.2.5 Pacote `tcc.alh.cleanshot.opengl`

O quinto pacote é composto pelas classes `GLView` e `GLRenderer` (Figura 17). Estes são responsáveis por desenhar a camada da OpenGL ES da tela.

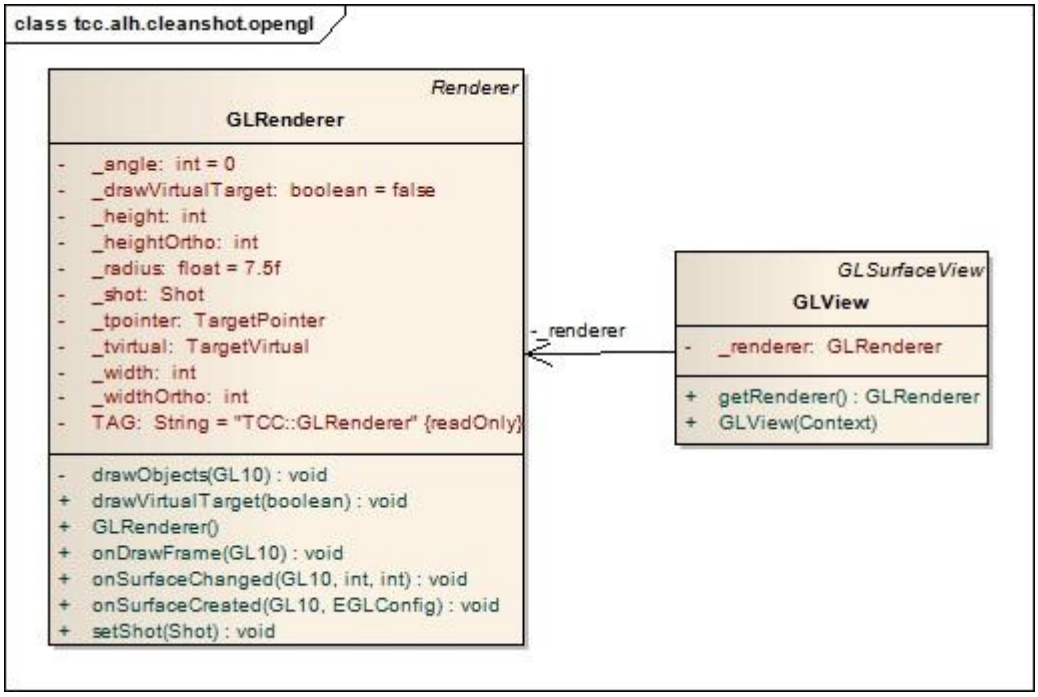


Figura 17 - Pacote `tcc.alh.cleanshot.opengl`

A classe `GLView` é uma extensão da classe `GLSurfaceView` do Android e trata todo o ciclo de vida da OpenGL ES na aplicação. Também é responsável por inicializar a classe `GLRenderer`.

A classe `GLRenderer` é uma implementação da interface `GLSurfaceRenderer` do Android e tem como objetivo renderizar os objetos gráficos. O método `onDrawFrame` é executado constantemente, redesenhando todos os objetos gráficos na `view` da OpenGL ES.

3.2.2.6 Pacote `tcc.alh.cleanshot.opengl.objects`

O sexto e último pacote é composto pelas classes `TargetPointer`, `TargetVirtual`, `Triangle`, `Circle` e `Line` (Figura 18), as quais são utilizadas pela classe `GLRenderer` para desenhar os objetos virtuais na tela.

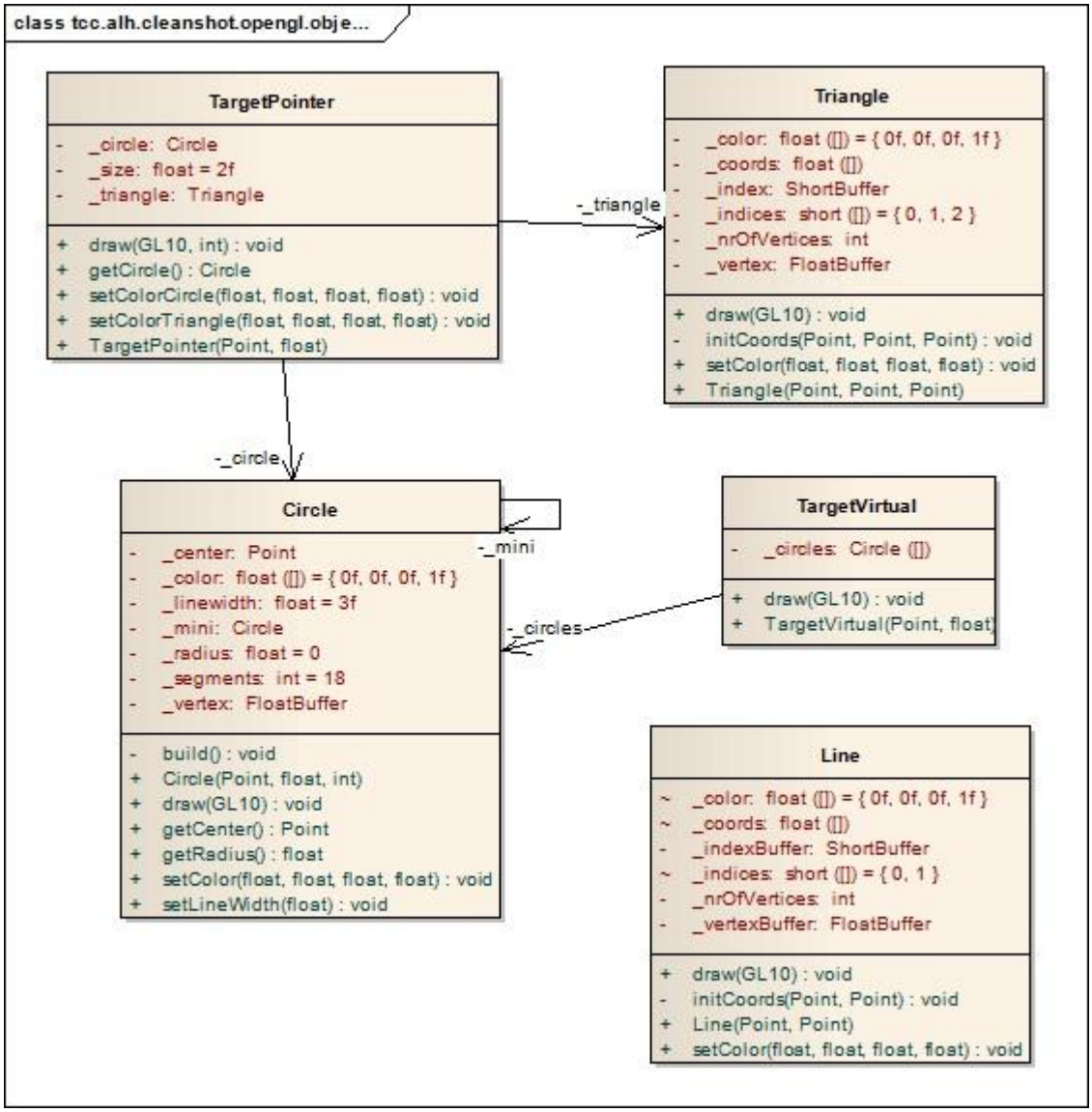


Figura 18 - Pacote `tcc.alh.cleanshot.opengl.objects`

A classe `Line` implementa um objeto gráfico que corresponde a uma linha. Um objeto `Line` será desenhado para interligar o centro do alvo e o centro do disparo. Outro objeto `Line` será desenhado para servir como divisória entre as informações de distância e pontuação.

A classe `Circle` é a implementação de um objeto gráfico que corresponde a um círculo. Este objeto será utilizado no desenho da localização do disparo, do centro do disparo, do centro do alvo e do alvo virtual.

Já a classe `Triangle` implementa um objeto gráfico que corresponde a um triângulo. Este objeto será utilizado pela classe `TargetPointer` para desenhar três triângulos que ficarão ao redor do círculo que representa o disparo e terão suas posições alteradas com base no atributo `_angle` da classe `GLRenderer` para simular um movimento de rotação.

A classe `TargetPointer` é responsável por desenhar o objeto que representa um disparo. Para desenhar tal objeto esta classe utiliza o objeto `Circle` e três objetos `Triangle`

que serão posicionados ao redor do objeto `Circle`.

Por fim, a classe `TargetVirtual` é responsável por desenhar um objeto que representa um alvo virtual. Para desenhar o objeto esta classe utilizará vários objetos do tipo `Circle`, apenas variando o raio, de acordo com as constantes especificadas na classe `Util`.

Todas as classes possuem o método `onDraw` que desenhará os objetos com base no vetor de vértices do mesmo.

3.2.3 Diagrama de seqüência

O diagrama de seqüência da Figura 19 apresenta a interação do `Usuário` com a aplicação na análise dos alvos, dos disparos e no desenho dos objetos virtuais conforme casos de uso `UC03`, `UC04` e `UC05`.

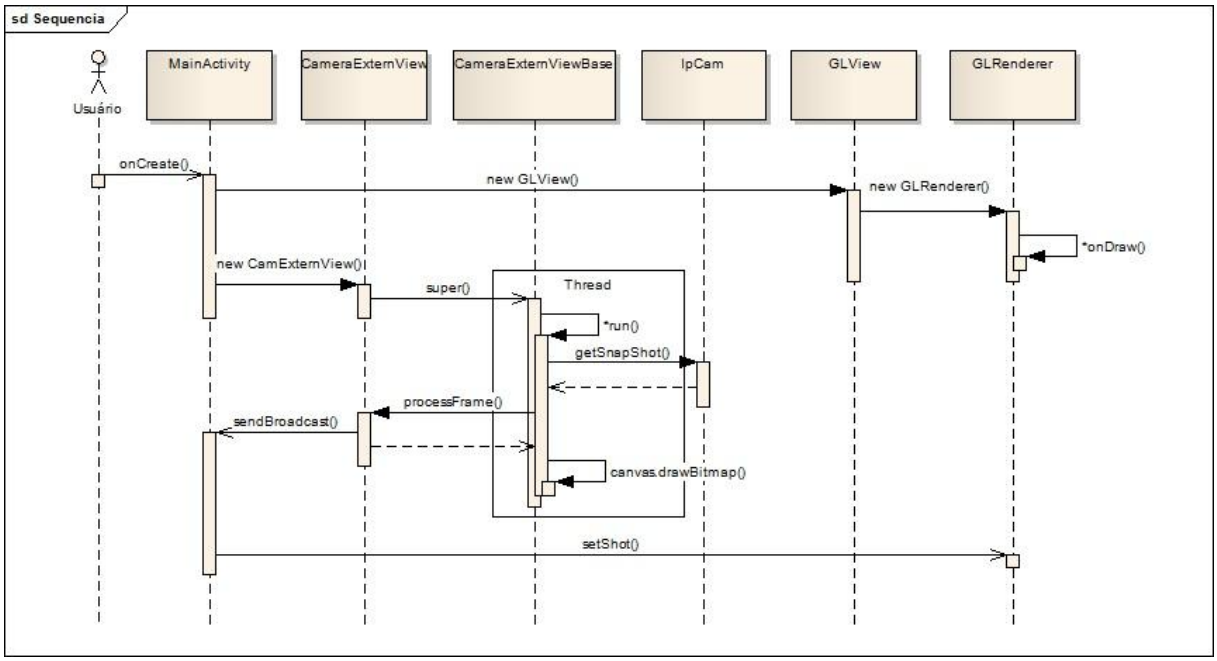


Figura 19 - Diagrama de seqüência

Na inicialização da aplicação pelo `Usuário`, a classe `MainActivity`, através do método `onCreate`, inicializa a classe `GLView`. Por sua vez a classe `GLView` inicializa a classe `GLRenderer` que começará o desenho dos objetos gráficos através do método `onDraw`.

A inicialização da classe `CamExternView` também é feita dentro do método `onCreate`. Uma *thread* é então iniciada e uma imagem é requisitada para a câmera através do método `getSnapShot`. A imagem é então analisada dentro do método `processFrame` e desenhada na tela através do método `canvas.drawBitmap`.

Durante a análise, dentro do método `processFrame`, as informações dos disparos detectados são enviadas através do método `sendBroadcast` para a classe `MainActivity`. Essas informações são repassadas para a classe `GLRenderer`, através do método `setShot`, onde são então desenhadas na camada da OpenGL através de objetos gráficos.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas na implementação, assim como principais classes e rotinas implementadas durante o desenvolvimento da aplicação.

3.3.1 Técnicas e ferramentas utilizadas

A aplicação foi desenvolvida em Java para Android versão 2.2, também conhecido como Froyo. O ambiente de desenvolvimento utilizado foi o Eclipse na versão 4.2, também conhecido como Juno, em conjunto com os *plugins* do *Android Development Tools* (ADT), que fornecem vários recursos para criação, depuração e execução de aplicações Android. Na fase de execução e depuração da aplicação foi utilizado um dispositivo *tablet* XOOM da fabricante Motorola, que possui o sistema operacional Android 3.2.

O *tablet* XOOM possui um processador com dois cores de 1 GHz, com 1 *Giga Byte* (GB) de memória e resolução de 1280x800 *pixels* (TECHMUNDO, 2011).

Para a captura das imagens dos alvos foi utilizada a câmera IP da fabricante Foscam modelo IP Wireless FI8918W. A câmera possui suporte a duas resoluções, sendo a primeira de 640x480 *pixels*, a uma taxa de 15 Frames por Segundo (FPS) e a segunda de 320x240 *pixels*, a uma taxa de 30 FPS (FOSCAM, 2010).

3.3.2 Os cálculos de distância e pontuação

Os dois principais cálculos da aplicação são respectivamente o cálculo da distância entre o centro do alvo e o centro do disparo implementado através do método `calcDistance`, e o cálculo da pontuação do disparo implementado através do método `calcPoints`.

Para calcular a distância utilizou-se a equação euclidiana entre dois pontos (Quadro 8).

```
public static float calcDistance(Point targetCenter, Point shotCenter) {
        // Calcular a distância ente dois pontos
        return  FloatMath.sqrt((float)  (Math.pow((shotCenter.x  -  targetCenter.x),  2)  +
Math.pow((shotCenter.y - targetCenter.y), 2)));
}
```

Quadro 8 – Método `calcDistance`

Através do resultado do cálculo da distância foi possível calcular a pontuação de um

disparo. Primeiro é necessário converter o valor da distância para centímetros, multiplicando pela escala do alvo. Por fim subtrair a distância do valor máximo possível para uma pontuação. A implementação deste cálculo pode ser observado no Quadro 9.

```
public static float calcPoints(Point targetCenter, float targetRadius, Point shotCenter) {
        // Calcular a escala do alvo
        float scale = (float) (Util.RAD_BLACK / targetRadius);
        // Calcular a distância
        float dist = FloatMath.sqrt((float) (Math.pow((shotCenter.x - targetCenter.x), 2) +
Math.pow((shotCenter.y - targetCenter.y), 2)));

        // Calcular a pontuação
        return round(MAX_POINTS - (dist * scale));
}
```

Quadro 9 – Método `calcPoints`

3.3.3 A câmera externa

Na classe `IpCam` foram implementados os métodos de comunicação com a câmera externa utilizada no desenvolvimento deste trabalho. Todo acesso as funções da câmera foram efetuados através da interface `HttpClient`, da interface `HttpResponse` e do método `HttpPost`. A interface `HttpClient` encapsula os objetos necessários para executar requisições HTTP. A interface `HttpResponse` recebe a resposta HTTP resultante da execução de um método `HttpPost` pela interface `HttpClient`.

No método `checkCamera` (Quadro 10) é necessário a validação do nome de usuário e senha informados na tela de informações da câmera. Um retorno com o nome de usuário significa que a validação ocorreu com sucesso.

```
private static String METHOD_CHECK_CAMERA   = "/check_user.cgi?";

public Boolean checkCamera() {
// {...}
        // Iniciar parametrização dos parâmetros
        HttpParams httpParameters = new BasicHttpParams();
        // Timeout da conexão
        HttpConnectionParams.setConnectionTimeout(httpParameters, 1000);
        // Timeout do socket
        HttpConnectionParams.setSoTimeout(httpParameters, 2000);

        HttpClient client = new DefaultHttpClient(httpParameters);
        HttpPost    post    =    new    HttpPost(_ipaddress    +    METHOD_CHECK_CAMERA    +
getUserPasswordString());

        try {
                HttpResponse response = client.execute(post);
                BufferedReader          rd          =          new          BufferedReader(new
InputStreamReader(response.getEntity().getContent()));
// {...}
                rd.close();
        } catch (IOException e) {
                e.printStackTrace();
        }
// {...}
}
```

Quadro 10 – Método `checkCamera`

Para buscar uma imagem da câmera utilizou-se o método `getSnapShot` (Quadro 11). O retorno da requisição HTTP é transferido para um `InputStream` e depois convertido para um `Bitmap` através da utilização da função `decodeStream` da classe `BitmapFactory` presente na API do Android.

```
private static String METHOD_GET_SNAPSHOT   = "/snapshot.cgi?";

public Bitmap getSnapShot() {
// {...}
        // Iniciar parametrização dos parâmetros
        HttpParams httpParameters = new BasicHttpParams();
        // Timeout da conexão
        HttpConnectionParams.setConnectionTimeout(httpParameters, 1000);
        // Timeout do socket
        HttpConnectionParams.setSoTimeout(httpParameters, 2000);

        HttpClient client = new DefaultHttpClient(httpParameters);
        HttpPost    post    =    new    HttpPost(_ipaddress    +    METHOD_GET_SNAPSHOT    +
getUserPasswordString());

        try {
                HttpResponse response = client.execute(post);
                InputStream ips = response.getEntity().getContent();
                bmp = BitmapFactory.decodeStream((InputStream) ips);
// {...}
        } catch (IOException e) {
                e.printStackTrace();
        }

        return bmp;
}
```

Quadro 11 – Método `getSnapShot`

3.3.4 As telas da aplicação

A aplicação possui três telas, uma que permite o acesso às funcionalidades da aplicação, outra que permite o acesso às configurações de acesso e controle da câmera externa e a última permite a informação dos dados de acesso à câmera externa.

3.3.4.1 Tela principal

A tela principal da aplicação é implementada pela classe `MainActivity`, que estende a classe `Activity` da API do Android e controla o ciclo de vida da aplicação. O Quadro 12 mostra de forma resumida a implementação do método `onCreate` da classe, onde são efetuadas as inicializações da aplicação, incluindo a verificação da biblioteca do OpenCV através do método `OpenCVLoader.initAsync`.

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Remover título da tela
        // Utilizar tela cheia - Fullscreen

        // Especificar o layout da view
        setContentView(R.layout.act_main);

        // Alocar o handler responsável por atualizar o tempo durante o processamento
        // Inicializar o frame que vai conter as camadas da câmera e do OpenGL
        _frame = (FrameLayout) findViewById(R.id.act_main_frame);

        // Inicializar a listview que vai mostrar os disparos processados
        // Carregar a biblioteca do opencv
        Log.i(TAG, "Carregando a biblioteca do OpenCV.");
        if (!OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_2, this, mOpenCVCallBack))
                Log.e(TAG, "Não foi possivel se conectar ao OpenCV Manager!");

        // Inicializar os arrays
        // Especificar os listeners dos controles da view
        // Criar e registrar o receiver que irá receber as informações dos alvos e disparos
        // Ler as preferências do usuário
        // Atualizar a lista dos processamentos
        // Especificar o tipo de visão inicial.
        setViewMode(VIEW_MODE_RGBA);
}
```

Quadro 12 – Método `onCreate`

A interface gráfica da tela principal possui um `FrameLayout` que recebe duas camadas sobrepostas. Estas camadas são criadas e sobrepostas através do método `addView` após a verificação da biblioteca do OpenCV, pelo método `mOpenCVCallBack` (Quadro 13).

```
private BaseLoaderCallback    mOpenCVCallBack = new BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
                switch (status) {
                        case LoaderCallbackInterface.SUCCESS: {
                                // Alocar a view do OpenglES
                                _glView = new GLView(_context);
                                // Alocar a view da câmera
                                _camView = new CamExternView(_context);
                                // Adicionar as camadas ao frame
                                _frame.addView(_glView);
                                _frame.addView(_camView);
                        }
                        break;
                }
        }
};
```

Quadro 13 – Método `mOpenCVCallBack`

A primeira camada dentro do `FrameLayout` é a camada que apresentará as imagens capturadas pela câmera externa. A classe `CamExternViewBase` ao ser instanciada executa o método `onSurfaceCreated` que inicializa a execução da *Thread* (Quadro 14) que fará as requisições para a câmera externa, solicitará o processamento da imagem e disponibilizará a imagem para o objeto `SurfaceHolder`.

```
public void run() {
// {...}
        while (_runThread) {
                Bitmap bmp = null;
                synchronized (this) {
                        if (null != _camera) bmp = _camera.getSnapShot();
                        if (bmp == null) {
                                // Criar um bitmap vazio, todo preto
// {...}
                        }
                        // Processar a imagem
                        bmp = processFrame(bmp);
                }

                if (bmp != null) {
                        Canvas canvas = (null == _holder ? null : _holder.lockCanvas());
                        if (canvas != null) {
                                canvas.drawColor(Color.BLACK);

                                float scaleWidth = (float) _width / bmp.getWidth();
                                float scaleHeight = (float) _height / bmp.getHeight();
                                Matrix matrix = new Matrix();
                                matrix.postScale(scaleWidth, scaleHeight);

                                // Desenhar a imagem
                                canvas.drawBitmap(bmp, matrix, null);
                                _holder.unlockCanvasAndPost(canvas);
                        }
                        bmp.recycle();
                }
// {...}
}
```

Quadro 14 – Thread da classe `CamExternViewBase`

Por sua vez a classe `CamExternView` é responsável pelo processamento das imagens adquiridas pela classe pai `CamExternViewBase`, através do método `processFrame`, que pode ser visto resumidamente no Quadro 15. O trecho do código simula uma detecção de um alvo através do modo de visão definido pela constante `VIEW_MODE_CHANGING_TARGET`.

```
protected Bitmap processFrame(Bitmap bmp) {
// {...}
        // Atualizar o modo de visão
        // Joga a imagem dentro de um array n-dimensional
        Utils.bitmapToMat(bmp, _rgba);

        // Transformar a imagem para cinza
        Imgproc.cvtColor(_rgba, _gray, Imgproc.COLOR_BGRA2GRAY);
        // Aplicar o filtro de blur
        Imgproc.GaussianBlur(_gray, _gray, new Size(3, 3), 1f, 1f);
        // Acumulador de pontos da transformada
        int iAccumulator = 60;
        // Raio configurado
        int rad = _defaultTargetRadius;
        // Intervalo máximo do raio
        int interval = 2;
        // Detectar os círculos na imagem
        Imgproc.HoughCircles(_gray, _circles, Imgproc.CV_HOUGH_GRADIENT, 1, _gray.rows(),
_defaultCannyHigh, iAccumulator, rad, rad + interval);
        // Encontrou algum círculo
        if (_circles.cols() > 0) {
                // Desenhar informações do alvo
        }
        // Criar um Bitmap da imagem processada
        _bmp = Bitmap.createBitmap(_rgba.cols(), _rgba.rows(), Bitmap.Config.ARGB_8888);
        Utils.matToBitmap(_rgba, _bmp);
// {...}
        return _bmp;
}
```

Quadro 15 – Método `processFrame`

O tipo de análise efetuada pelo método `processFrame` é especificado pelo modo de

visão em que se encontra a aplicação. Este modo de visão é alterado pelo usuário através da utilização dos recursos da aplicação.

A segunda camada dentro do `FrameLayout` é a camada da OpenGL que desenha alguns dos objetos virtuais. É implementada pela classe `GLView` e os objetos gráficos são desenhados pela classe `GLRenderer`.

A classe `GLView` é uma extensão da classe `GLSurfaceView` da OpenGL e tem por finalidade configurar a `view` e inicializar a classe `GLRenderer`. No Quadro 16 pode-se ver o construtor da classe e a inicialização da classe `GLRenderer`.

```
public GLView(Context context) {
        super(context);

        // Fundo Translúcido / Transparente
        this.setEGLConfigChooser(8, 8, 8, 8, 16, 0);
        this.getHolder().setFormat(PixelFormat.TRANSLUCENT);
        this.setDebugFlags(DEBUG_CHECK_GL_ERROR);

        // Renderizador dos objetos gráficos
        _renderer = new GLRenderer();
}
```

Quadro 16 – Construtor `GLView`

A classe responsável por transformar e desenhar os objetos gráficos na `view` é a `GLRenderer`, que implementa os métodos da interface `Renderer` do OpenGL. O método `onSurfaceCreated` (Quadro 17) inicializa as configurações do OpenGL, modificando a cor de fundo para transparente, habilitando o teste de profundidade, entre outros.

```
public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f); // Transparência
        gl.glClearDepthf(1.0f); // Limpar o buffer de profundidade

        gl.glDepthFunc(GL10.GL_LEQUAL); // Tipo de teste de profundidade
        gl.glDisable(GL10.GL_DITHER); // Desabilitar dithering

        gl.glEnable(GL10.GL_DEPTH_TEST); // Habilitar teste de profundidade
        gl.glShadeModel(GL10.GL_SMOOTH); // Habilitar sombreamento suave
}
```

Quadro 17 – Método `onSurfaceCreated`

Já o método `onSurfaceChanged` é executado sempre que houver uma mudança no tamanho da `view`. O método também configura a `viewport`, as matrizes de projeção e modelo e o tamanho do `ortho`, como pode ser visto no Quadro 18 abaixo.

```
public void onSurfaceChanged(GL10 gl, int width, int height) {
// {...}
        // Inicializar a viewport
        gl.glViewport(0, 0, width, height);

        // Selecionar a matrix de projeção
        gl.glMatrixMode(GL10.GL_PROJECTION);
        // Limpar a matrix de projeção
        gl.glLoadIdentity();
// {...}
        // Especificar a dimensão do ortho
        gl.glOrthof(0, 320, 240, 0, -1f, 1f);

        // Selecionar a matrix de modelo
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        // Limpar a matrix de modelo
        gl.glLoadIdentity();
}
```

Quadro 18 – Método `onSurfaceChanged`

O método `onDrawFrame` é responsável por desenhar todos os objetos gráficos, limpar a tela e o buffer de profundidade, selecionar e inicializar a matriz de modelo e executar o método `onDrawObjects`, conforme Quadro 19 abaixo.

```
public void onDrawFrame(GL10 gl) {
        // Limpar tela e buffer de profundidade
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        // Selecionar a matrix de modelo
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        // Limpar a matriz
        gl.glLoadIdentity();

        // Desenhar os objetos gráficos
        drawObjects(gl);
// {...}
}
```

Quadro 19 – Método `onDrawFrame`

Por fim o método `onDrawObjects` (Quadro 20) desenha algumas das informações do disparo na tela da aplicação.

```
private void drawObjects(GL10 gl) {
        Shot shot = _shot;
// {...}
        // Inicializar e desenhar o alvo virtual
        if (null == _tvirtual && _drawVirtualTarget) _tvirtual = new
TargetVirtual(shot.getTargetCenter(), shot.getTargetScale());
        if (null != _tvirtual && _drawVirtualTarget) _tvirtual.draw(gl);

        // Buscar os pontos do alvo e do disparo
        Point p1 = shot.getTargetCenter();
        Point p2 = shot.getShotCenter();

        // Calcular a posição da linha divisória
        Point[] p = Util.calcLinePos(p1, _widthOrtho);
        Point p3 = p[0];
        Point p4 = p[1];

        // Desenhar a linha do centro do alvo para o disparo
        Line line1 = new Line(p1, p2);
        line1.setColor(0f, 0f, 1f, 1f);
        line1.draw(gl);

        // Desenhar a linha divisória
        Line line2 = new Line(p3, p4);
        line2.setColor(0f, .5f, .25f, 1f);
        line2.draw(gl);

        // Desenhar o centro do alvo
        Circle center = new Circle(shot.getTargetCenter(), 1f, 8);
        center.setColor(1f, 1f, 1f, 1f);
        center.draw(gl);

        // Desenhar o apontador do disparo
        if (null != _tpointer) _tpointer.draw(gl, _angle);
// {...}
}
```

Quadro 20 – Método `onDrawObjects`

As classes `Triangle`, `Circle` e `Line` desenham de maneira semelhante as formas geométricas, preparando os vetores e os *buffers* nos seus construtores e desenhando-os através do método `draw`. Já as classes `TargetPointer` e `TargetVirtual` chamam o método `draw` dos seus objetos filhos, que são formados pelas classes `Triangle` e `Circle`, para realizar os desenhos.

3.3.4.2 Tela de configurações da câmera

Esta tela pode ser acessada através do botão `Configurar Câmera` da tela principal, que chama o método `showCamConfig` da classe `MainActivity`. Este método irá iniciar um novo `Dialog` com seu próprio *layout* e configurar os *listeners* dos botões deste `Dialog`, como pode ser observado no Quadro 21.

```
private void showCamConfig() {
        final Dialog dialog = new Dialog(this);

        // Especificar o layout utilizado pela tela
        dialog.setContentView(R.layout.act_config);
        // Informar o título da tela
        dialog.setTitle(getString(R.string.act_config_info));

        Button access = (Button) dialog.findViewById(R.id.act_config_btnCamAccess);
        Button control = (Button) dialog.findViewById(R.id.act_config_btnCamControl);

        // Listener do botão 1
        access.setOnClickListener(new View.OnClickListener() {
                public void onClick(View v) {
// {...}
                }
        });

        // Listener do botão 2
        control.setOnClickListener(new View.OnClickListener() {
                public void onClick(View v) {
// {...}
                }
        });

        // Mostrar o dialog na tela
        dialog.show();
}
```

Quadro 21 – Método `showCamConfig`

O botão `Acesso Remoto` permite o acesso a tela de informações da câmera e o botão `Controle Remoto` habilita na tela principal as opções de controle da câmera.

3.3.4.3 Tela de informações da câmera

A tela pode ser acessada através do botão `Acesso Remoto`, disponível na tela de configurações da câmera. Este botão chama a função `configCamIP` (Quadro 22) da classe `MainActivity` que inicializa um `AlertDialog` customizado com seu próprio *layout*. Nesta tela é possível informar os dados de acesso a uma câmera externa. Os dados informados são validados no método `initCamera` através da tentativa de acesso a câmera externa e depois gravados nas preferências do usuário pelo método `writePrefs` através da classe `SharedPreferences` disponível na API do Android.

```
private void configCamIP() {
        AlertDialog.Builder customDialog = new AlertDialog.Builder(this);
        // Informar o título da tela
        customDialog.setTitle(getString(R.string.ipcam_title));

        // Inflar o layout da view da tela
        LayoutInflater              layoutInflater              =              (LayoutInflater)
getApplicationContext().getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        final View view = layoutInflater.inflate(R.layout.act_cam_ip, null);
// {...}
        // Listener do botão 1
        customDialog.setPositiveButton(getString(R.string.ipcam_ok),                          new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface arg0, int arg1) {
// {...}
                        // Validar as informações
                        initCamera(ip, user, pass);
                }
        });
// {...}
        // Informar a view da tela
        customDialog.setView(view);
        // Mostrar o dialog na tela
        customDialog.show();
}
```

Quadro 22 – Método `configCamIP`

3.3.5 O banco de dados

O acesso ao banco de dados SQLite e suas funcionalidades pode ser feito através da classe `DB`, que ao ser instanciada verifica se o arquivo do banco de dados existe no dispositivo móvel. Caso o arquivo não exista, a classe cria o mesmo e também executa o método `createTable` para criar a tabela utilizada pela aplicação para armazenar as informações das análises dos alvos e dos disparos, como apresentado no Quadro 23.

```
private final String          DATABASE_NAME          = "tcc.db";
private final String          DATABASE_PATH;

public DB() {
        boolean needCreate = false;

        // Montar o path (DATABASE_PATH) para o SDCard
        // Criar os diretórios caso não existam

        // Verificar se o arquivo do banco de dados existe
        File dbfile = new File(DATABASE_PATH + File.separator + DATABASE_NAME);
        if (!dbfile.exists()) {
                try {
                        // Criar o arquivo do banco de dados
                        dbfile.createNewFile();
                        needCreate = true;
                } catch (IOException e) {
                        Log.i(TAG, "Erro ao criar o BD: " + e.getMessage());
                }
        }

        // Criar a tabela do banco de dados
        if (needCreate) createTable();
}
```

Quadro 23 – Verificação e criação do arquivo do banco de dados

O método `createTable` (Quadro 24) busca uma instância do banco de dados, através

do método `getWritableDatabase` e dispara o método `execSQL` da classe `SQLiteDatabase` da API do Android para execução do comando SQL para criação da tabela, definido pela constante `CREATE_TABLE`.

```
private void createTable() {
        // Buscar a instância do banco de dados
        SQLiteDatabase sqldb = getWritableDatabase();
// {...}
        // Executar o comando SQL
        sqldb.execSQL(CREATE_TABLE);
        sqldb.close();
// {...}
}
```

Quadro 24 – Método `createTable`

O Quadro 25 apresenta a implementação do método `getWritableDatabase`, que busca uma instância do objeto de banco de dados `SQLiteDatabase` com permissão de leitura e gravação.

```
private SQLiteDatabase getWritableDatabase() {
        return  SQLiteDatabase.openDatabase(DATABASE_PATH  +  File.separator  +  DATABASE_NAME,
null, SQLiteDatabase.OPEN_READWRITE);
}
```

Quadro 25 – Método `getWritableDatabase`

Através do método `insert` (Quadro 26), as informações de um disparo são inseridas no banco de dados.

```
public void insert(Date dtProc, Shot shot) {
// {...}
        ContentValues insertValues = new ContentValues();
        // Inserir os dados do disparo (objeto Shot) no objeto insertValues

        // Inserir no banco de dados
        sqldb.insert(TABLE_SHOT, null, insertValues);
        sqldb.close();
// {...}
}
```

Quadro 26 – Método `insert`

O método `getShots` (Quadro 27) retorna um `ArrayList` de objetos `Shot`, com base na data da análise, que representam as informações dos disparos efetuados em uma determinada análise.

```
public ArrayList<Shot> getShots(Date dtProc) {
        ArrayList<Shot> lstShot = new ArrayList<Shot>();

// {...}
        Cursor c = sqldb.query(TABLE_SHOT, null, COLUMN_SHOT_DATE + "=?", new String[] { new
SimpleDateFormat("yyyyMMdd HHmmss").format(dtProc) }, null, null, COLUMN_SHOT_DATE);

// {...}
        while (!c.isAfterLast()) {
                // Inserir os dados do Cursor num objeto Shot
                // Inserir o objeto Shot na lista
                lstShot.add(shot);
                c.moveToNext();
        }
// {...}
        return lstShot;
}
```

Quadro 27 – Método `getShots`

Por fim, o método `getDateProc` (Quadro 28) retorna um `ArrayList` de objetos

`String`, que representam as datas de todas as análises efetuadas e gravadas pela aplicação.

```
public ArrayList<String> getDateProc() {
        ArrayList<String> lstDate = new ArrayList<String>();

// {...}
        Cursor c = sqldb.query(TABLE_SHOT, new String[] { COLUMN_SHOT_DATE }, null, null,
COLUMN_SHOT_DATE, null, COLUMN_SHOT_DATE + " DESC");

// {...}
        while (!c.isAfterLast()) {
                // Inserir a String representando a data na lista
                lstDate.add(data);
                c.moveToNext();
        }
// {...}
        return lstDate;
}
```

Quadro 28 – Método `getDateProc`

3.3.6 Operacionalidade da implementação

A operacionalidade da aplicação é apresentada na forma de funcionalidades e casos de uso, sendo representados através de imagens. Serão apresentadas imagens da aplicação funcionando no dispositivo Motorola XOOM, utilizado durante o desenvolvimento do trabalho.

No momento que a aplicação é iniciada uma tela principal é apresentada, conforme pode ser conferido na Figura 20.

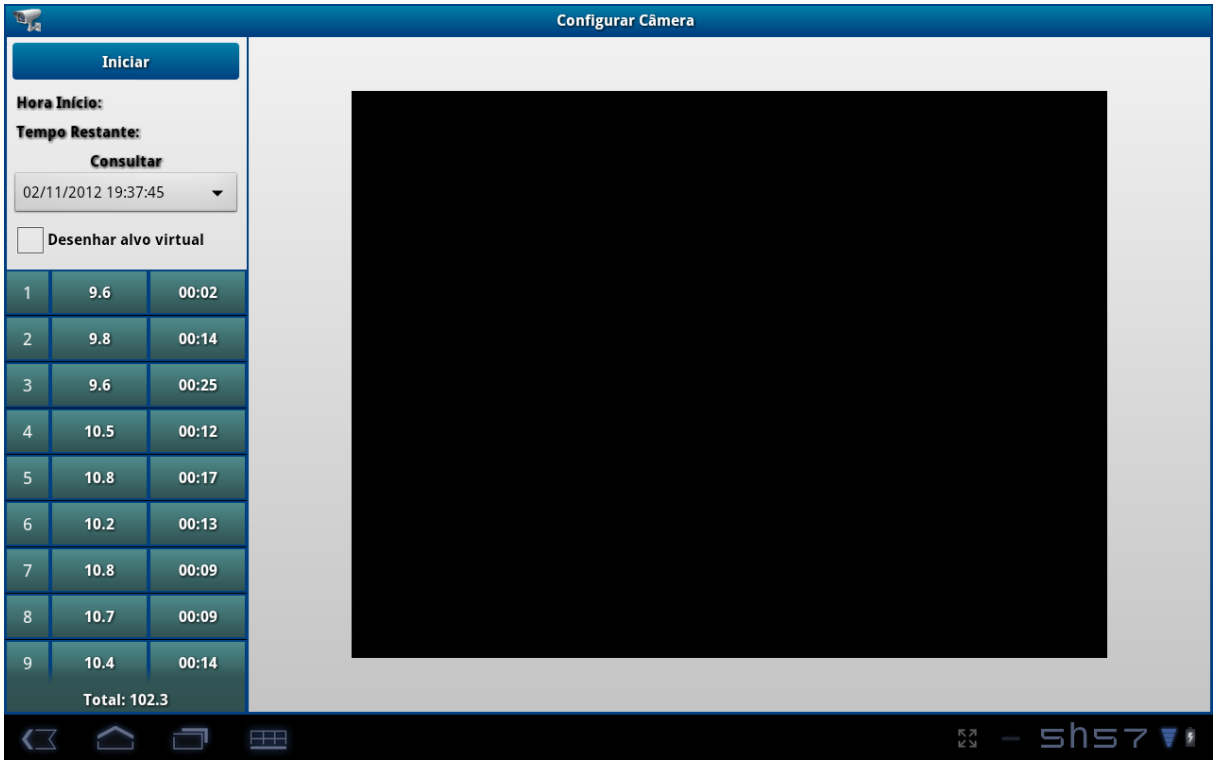


Figura 20 - Tela principal

3.3.6.1 Configuração da câmera

Inicialmente é necessário configurar uma câmera externa, para ser possível acessar as demais funcionalidades do sistema. Através do botão `Configurar Câmera` é possível acessar a tela de configurações da câmera como pode ser visto na Figura 21.

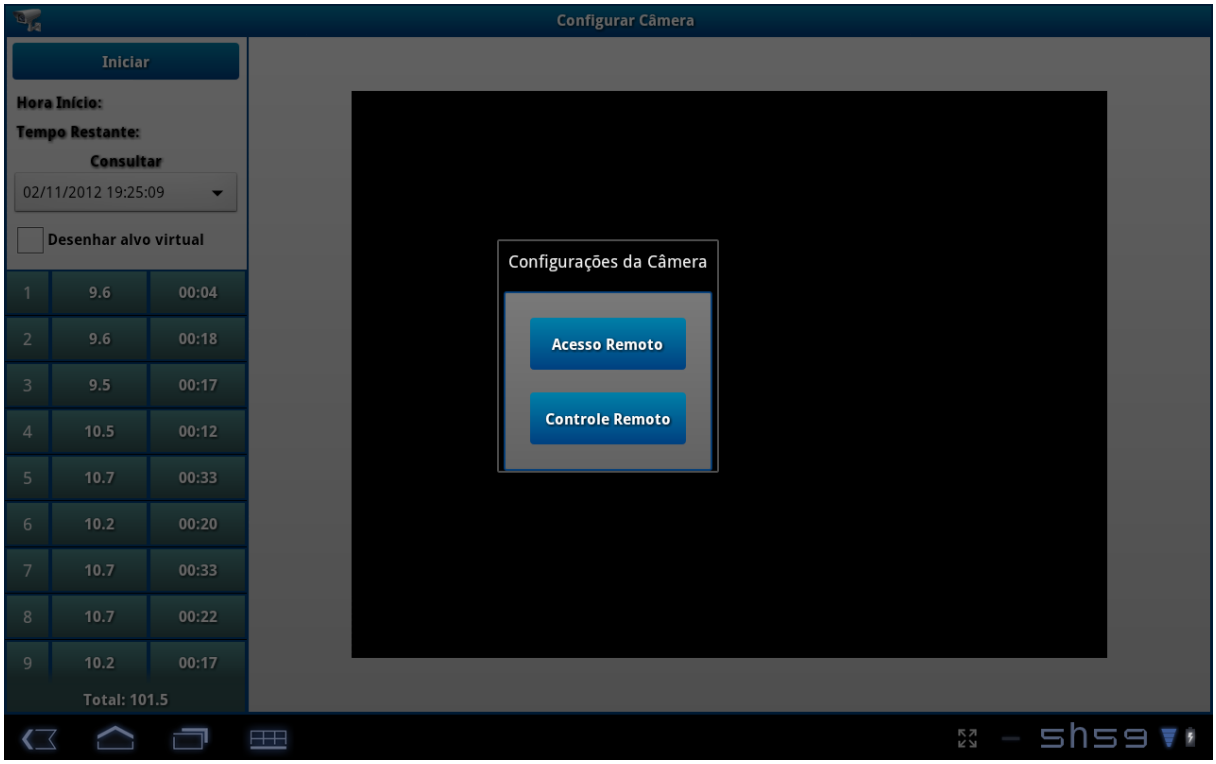


Figura 21 - Configurações da câmera

O botão `Acesso Remoto` abre outra tela que permite ao usuário informar os dados de acesso a câmera IP, tal como o endereço IP, o nome de usuário e a senha de acesso (Figura 22). O botão `Controle Remoto` habilita na tela principal os botões de controle do *pan* da câmera, como pode ser visto em destaque na Figura 23. Estes botões permitem controlar a câmera remotamente, sendo possível movimentar a visão da câmera horizontalmente e verticalmente.

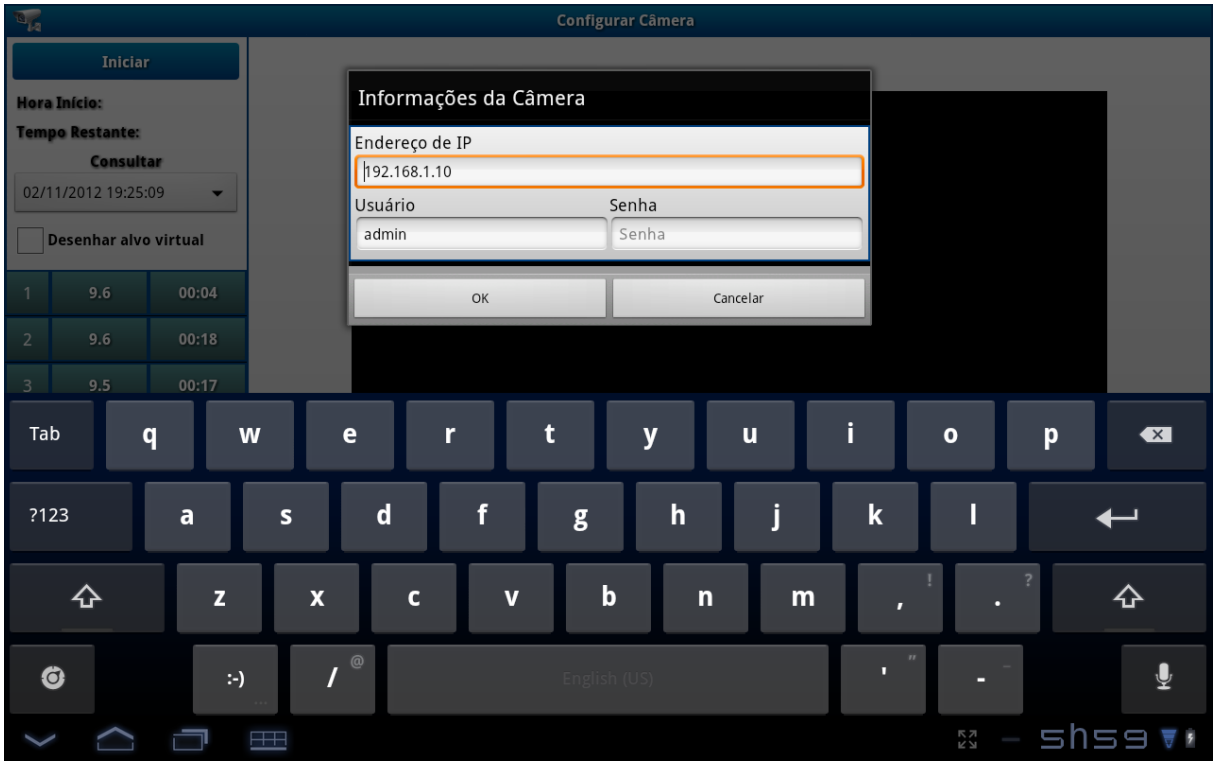


Figura 22 - Acesso remoto

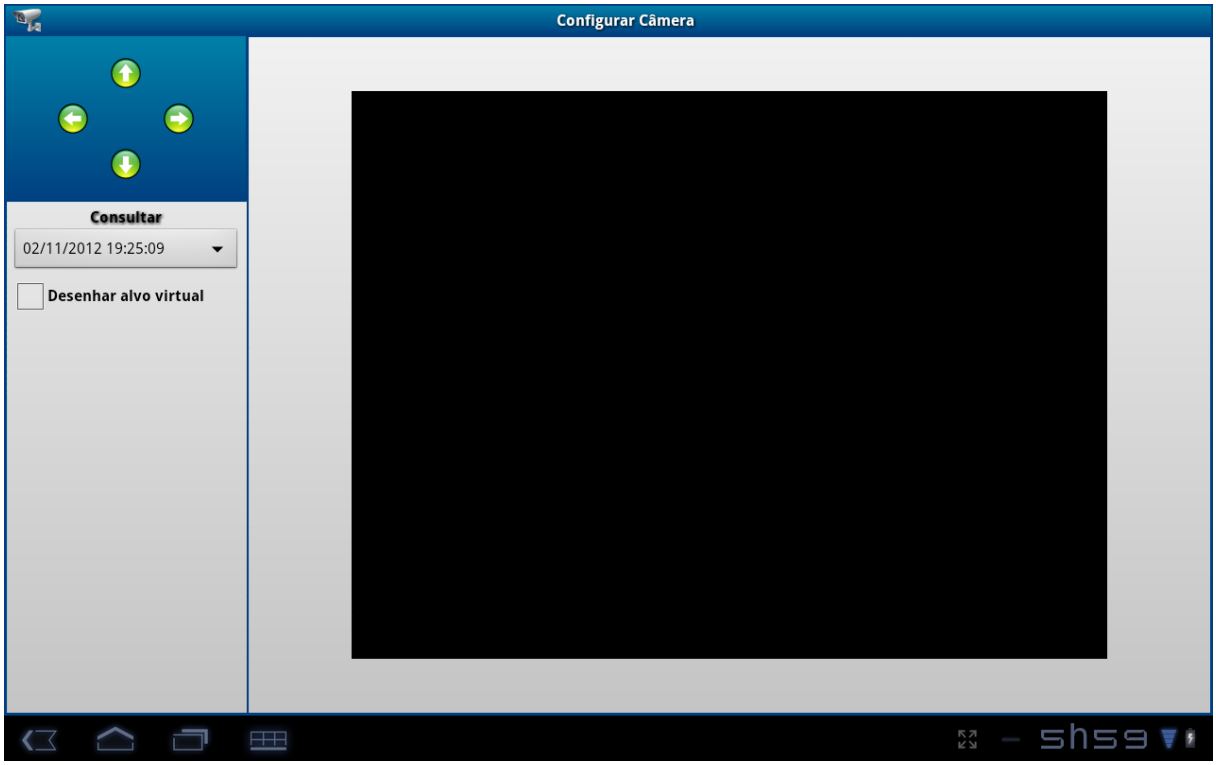


Figura 23 - Controle remoto

Após a configuração da câmera externa, as imagens capturadas pela câmera serão mostradas no centro da tela principal do sistema, como pode ser visto na Figura 24.

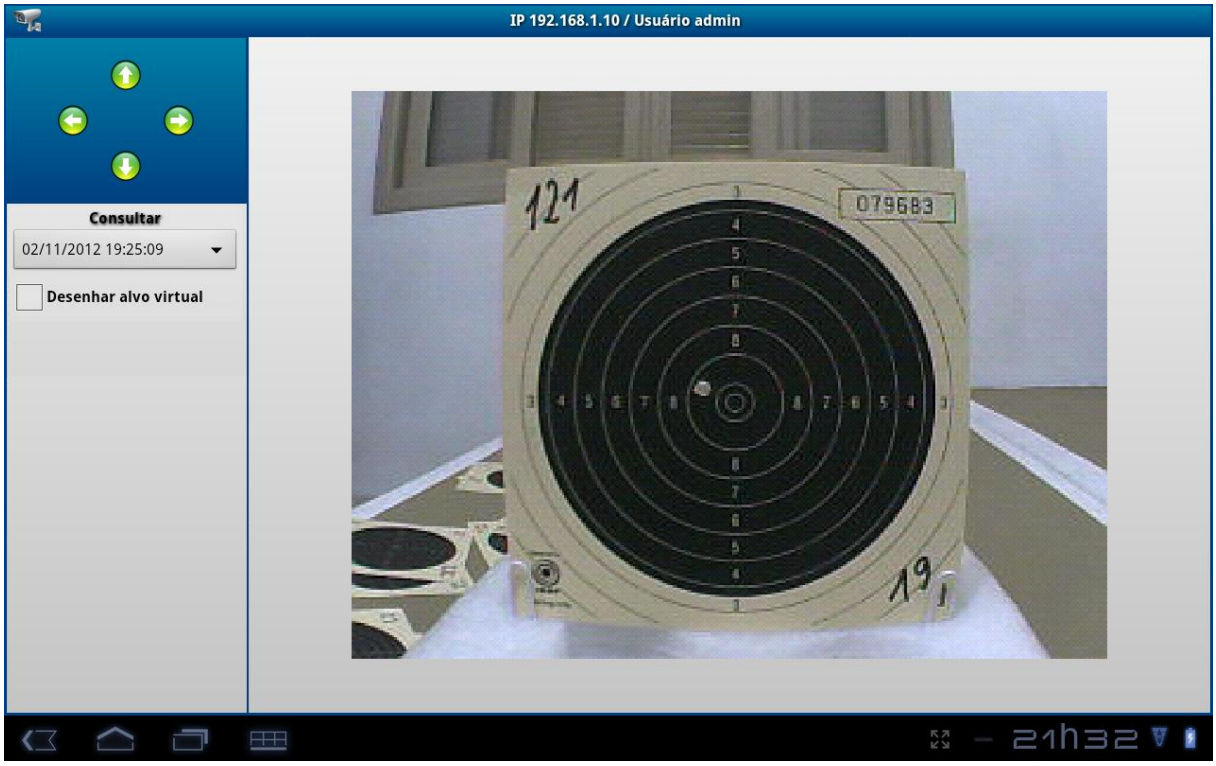


Figura 24 - Imagem capturada pela câmera

3.3.6.2 Configurar filtro

Com a câmera IP configurada é possível configurar o filtro Canny para ser possível identificar os alvos e os disparos com maior precisão. O modo de configuração de filtro é acessado através do botão `Menu` presente nos equipamentos Android. Este botão irá mostrar dois botões, dentre eles está o botão `Configurar Filtro` que irá habilitar os controles de configuração do filtro na tela principal, conforme pode ser visto na Figura 25. As opções de configuração disponíveis são a visualização do filtro Canny através do botão `Ativar`, a configuração do incremento e decremento da intensidade do filtro Canny pelos botões `Mais` e `Menos`, que se encontram logo abaixo do botão `Ligar` do filtro Canny e por fim o incremento e decremento do tamanho do raio externo dos alvos pelos botões `Mais` e `Menos`, que se encontram abaixo dos botões de configuração do filtro Canny. A intensidade do filtro Canny e tamanho do raio podem ser visualizados entre os respectivos botões `Mais` e `Menos`. As configurações informadas são aplicadas automaticamente e podem ser visualizadas nas imagens capturadas pela câmera e apresentadas no meio da tela principal, como pode ser visto na Figura 26.

Figura 25 - Configurar filtro

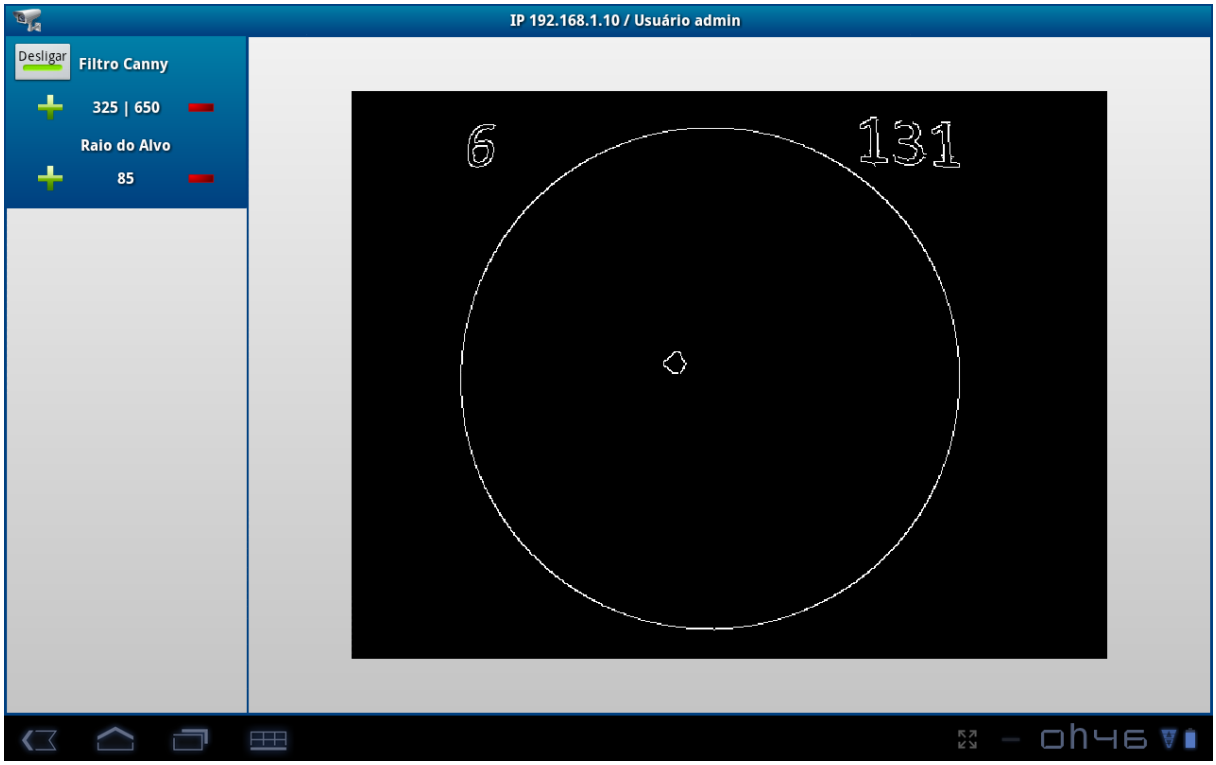


Figura 26 - Filtro ativo e configurado

3.3.6.3 Análise dos alvos e disparos

Com a câmera configurada e com o filtro configurado é possível iniciar a análise de

alvos. Para iniciar uma análise é necessário estar no modo de visualização de imagens. Este modo é acessado através do botão `Menu` presente nos equipamentos Android e depois no botão `Visualizar` que irá habilitar o botão de processamento, que pode ser visto no canto esquerdo superior na Figura 27. O botão `Iniciar` começa o processamento das imagens capturadas pela câmera tentando localizar um alvo, também habilita o botão `Finalizar` e o botão `Finalizar Troca`, além de apresentar informações como a hora de início do processamento e o tempo restante para o final da prova.

Figura 27 - Iniciar processamento

Após a detecção do alvo (Figura 28) o botão `Finalizar Troca` pode ser utilizado para encerrar a detecção do alvo e iniciar a procura e análise dos possíveis disparos. Cada disparo encontrado é inserido na lista de disparos e sua localização, pontuação e distância em relação ao centro do alvo serão desenhadas sobre o alvo com uso de objetos virtuais (Figura 29).

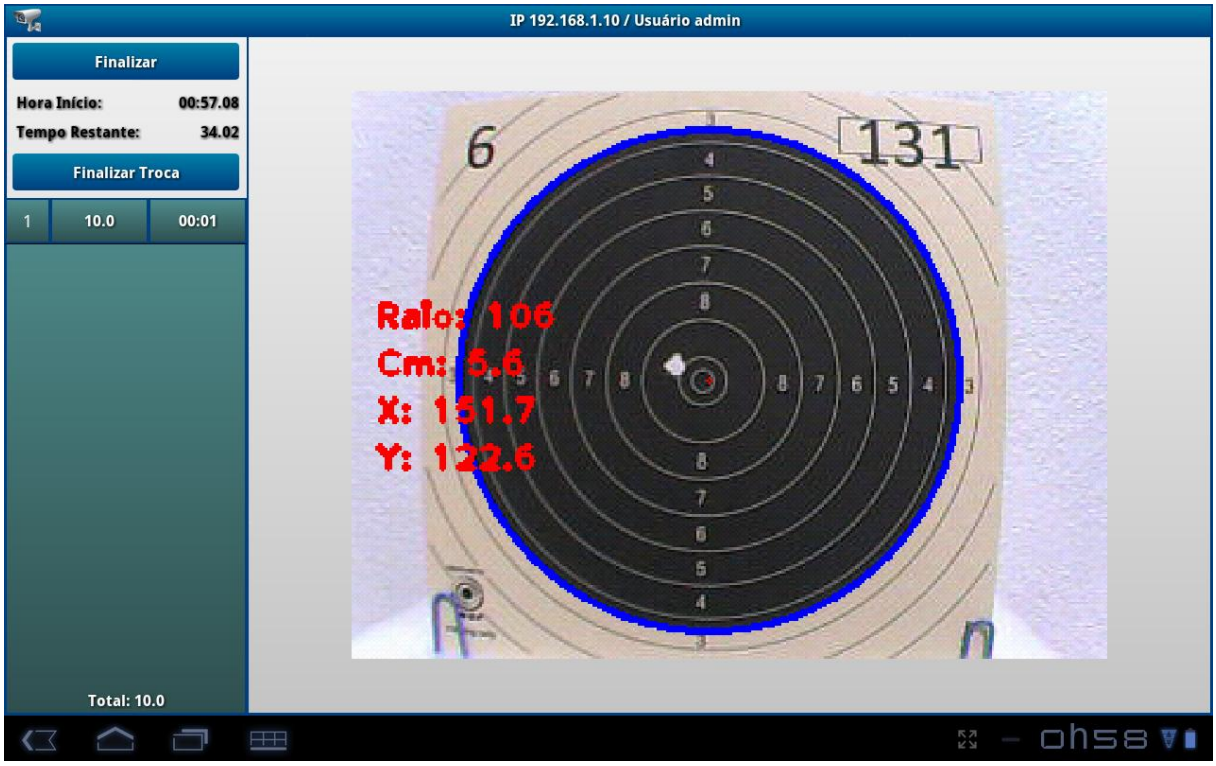


Figura 28 - Detecção do alvo

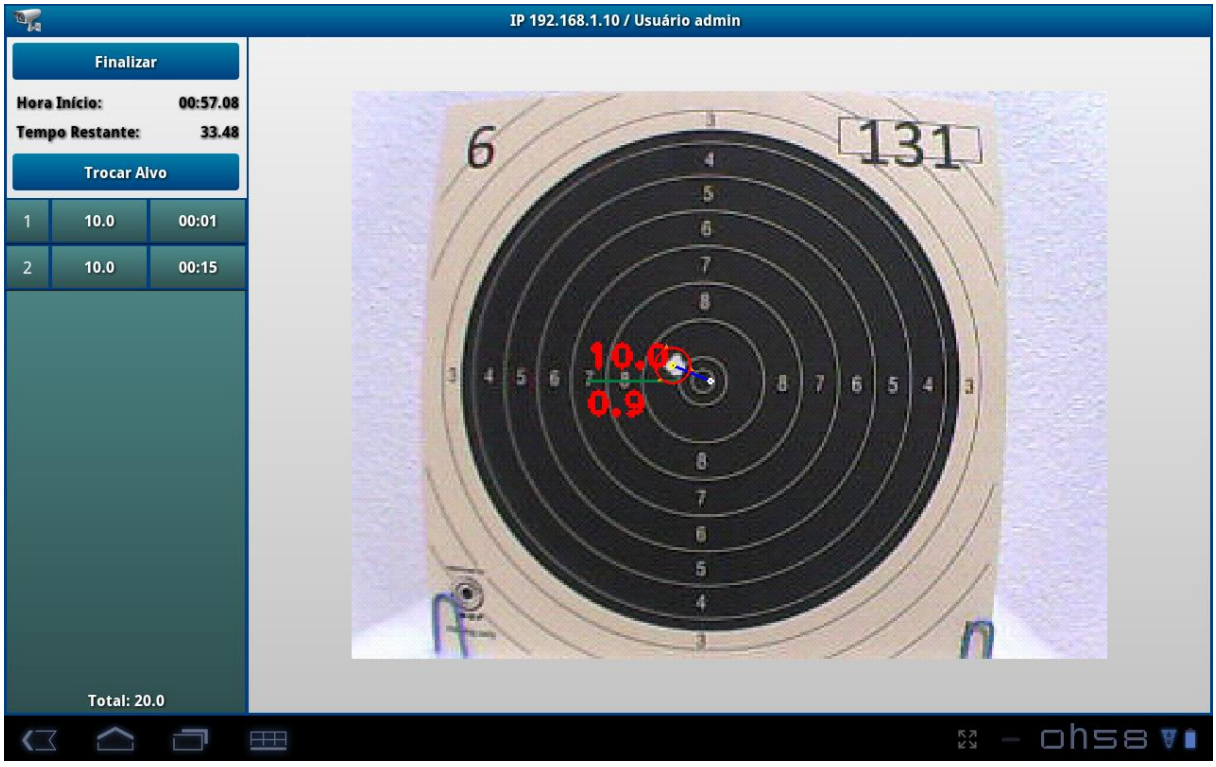


Figura 29 - Disparo localizado

O botão `Trocar Alvo` tem por finalidade permitir uma troca de alvo. Após utilizar este botão a análise de disparos é finalizada e a aplicação volta a procurar um alvo nas imagens capturadas pela câmera. Caso em algum momento o alvo não seja reconhecido será necessário finalizar a análise através do botão `Finalizar` e voltar ao modo de configuração do filtro para

modificar a regulagem do filtro e depois recomeçar a análise.

3.3.6.4 Consulta de análises

A consulta das informações das análises dos alvos e disparos está acessível através do botão `Consulta`. Este botão abre uma lista (Figura 30) com as datas de todas as análises efetuadas. A seleção de uma das datas fará a aplicação carregar todos os disparos analisados naquela data e apresente os mesmos na lista de disparos. Um simples toque sobre um disparo na lista desenhará as informações do disparo sobre a imagem atual apresentada pela câmera. A opção `Desenhar alvo virtual` habilita o desenho de um alvo virtual sobre a imagem, como pode ser visto na Figura 31.

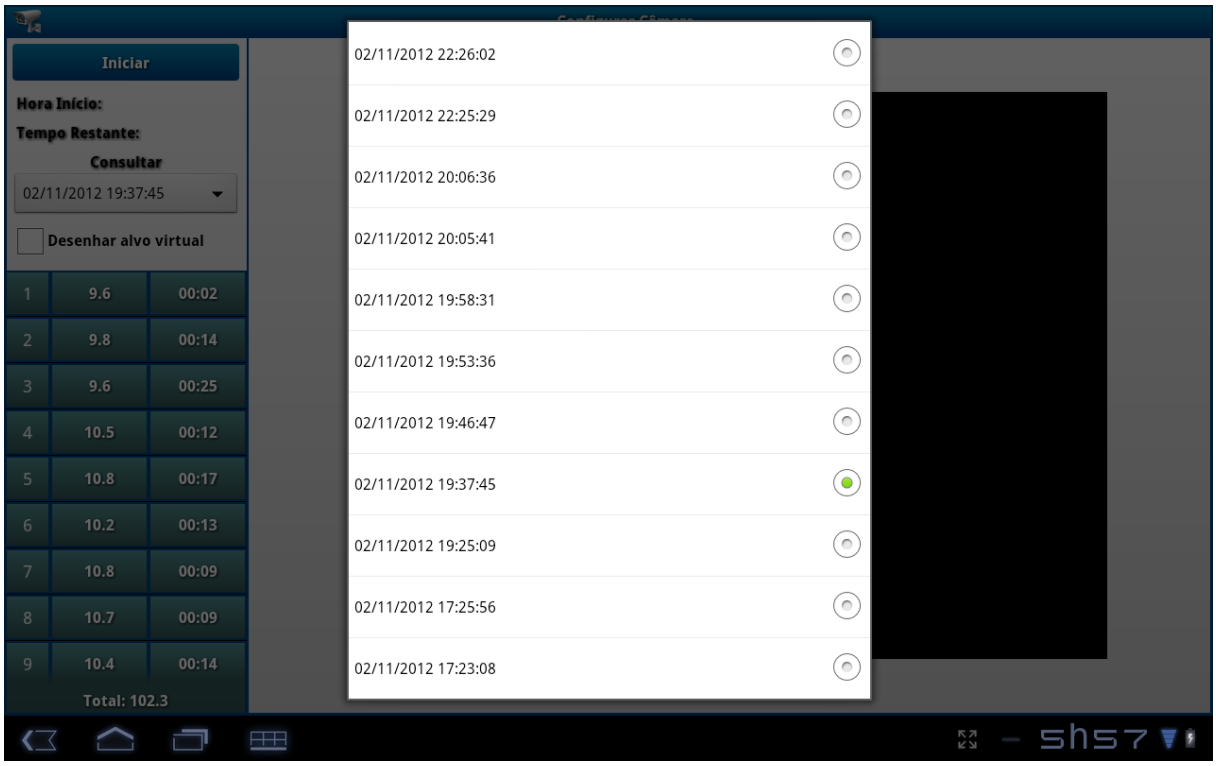


Figura 30 - Lista de análises

Figura 31 - Alvo virtual

3.4 RESULTADOS E DISCUSSÃO

O reconhecimento dos alvos e dos disparos foi realizado utilizando a linguagem Java e a biblioteca gráfica OpenCV para o desenvolvimento da aplicação. Os resultados obtidos pela aplicação foram satisfatórios dentro de um ambiente controlado.

A aplicação se utilizou de uma câmera IP de baixo custo para a obtenção das imagens dos alvos. Devido à ausência de suporte nativo a zoom na câmera, a mesma foi posicionada a uma distância aproximada de 15 centímetros do alvo, objetivando enquadrar toda a área do alvo na imagem capturada. Além disso, como a aplicação não possui a implementação de uma rotina de calibragem de câmera, foi necessário colocar a câmera de frente para o alvo, impossibilitando os testes numa situação real, pois a câmera ficaria na frente da linha de tiro. Além da alteração da resolução da câmera durante os testes, nenhum outro ajuste foi feito nas configurações da câmera.

Todos os testes foram efetuados num ambiente interno iluminado através de uma lâmpada fluorescente e sem influência de luz exterior. Também foi constatado que um

excesso de luminosidade na parte da frente ou na parte detrás do alvo dificulta a localização dos disparos.

A modalidade escolhida para servir de base para os testes da aplicação foi a modalidade de carabina apoiada. Numa competição desta modalidade não existe atualmente um tempo mínimo ou máximo estipulado para divulgação dos resultados obtidos ao competidor. Ao final dos 20 disparos válidos efetuados pelo competidor, os alvos são levados para serem processados por uma máquina da FCCTE. Esta máquina efetua a leitura de cada alvo e imprime uma pontuação sobre o mesmo com base no disparo. O tempo em que os resultados são disponibilizados varia de acordo com a quantidade de alvos que estão na fila aguardando apuração e também varia com outros fatores, como disponibilidade de fiscais para alimentarem a máquina com os alvos e problemas que podem ocorrer na própria máquina.

Para mostrar ao competidor os resultados obtidos em tempo real, a aplicação foi desenvolvida buscando aperfeiçoar o tempo de processamento das imagens, estipulando o tempo máximo de 1 (um) segundo para detectar e reconhecer o alvo e eventuais disparos. Portanto durante a fase de desenvolvimento e fase inicial dos testes a câmera foi configurada e utilizada na resolução 320x240, visando alcançar uma performance maior. Na parte final dos testes a câmera foi configurada também na resolução 640x480, visando verificar o desempenho nesta resolução e fornecer um comparativo com a resolução mais baixa de 320x240 nos resultados obtidos. Na resolução de 640x480 nenhuma modificação foi efetuada no método de processamento das imagens.

Na otimização das imagens utilizou-se a função `GaussianBlur` para diminuição do ruído. A diminuição dos ruídos possibilitou uma detecção de bordas mais precisa pelo operador Canny, o que facilitou a localização das bordas de maior contraste. Por fim a utilização da técnica da transformada de Hough possibilitou a detecção do alvo e dos disparos.

No uso da transformada de Hough encontrou-se um problema em relação à biblioteca OpenCV versão 2.31, que não suportava corretamente a parametrização dos raios mínimo e máximo, ocasionando a detecção incorreta dos círculos. Este problema foi resolvido com o uso da versão 2.42 da biblioteca em questão. Durante os testes de detecção, notou-se ainda uma pequena oscilação na detecção dos círculos pela transformada de Hough, o que interferiu nos cálculos das pontuações dos disparos.

A biblioteca gráfica OpenGL ES 1.0 não foi utilizada para desenhar os textos com as informações de distância e pontuação dos disparos, devido a uma limitação de sua especificação. Para contornar este problema, optou-se pela utilização dos recursos da biblioteca gráfica OpenCV, que possui um método para o desenho de textos sobre as imagens.

A interface JavaCV não foi utilizada, pois a mesma fornece apenas um invólucro para várias bibliotecas gráficas, inclusive a biblioteca OpenCV que foi utilizada no desenvolvimento deste trabalho e que supriu todas as necessidades.

Ao final do desenvolvimento da aplicação foram desenvolvidos testes de desempenho na captura e análise de imagens nas duas resoluções utilizadas, como demonstrado na Figura 32.

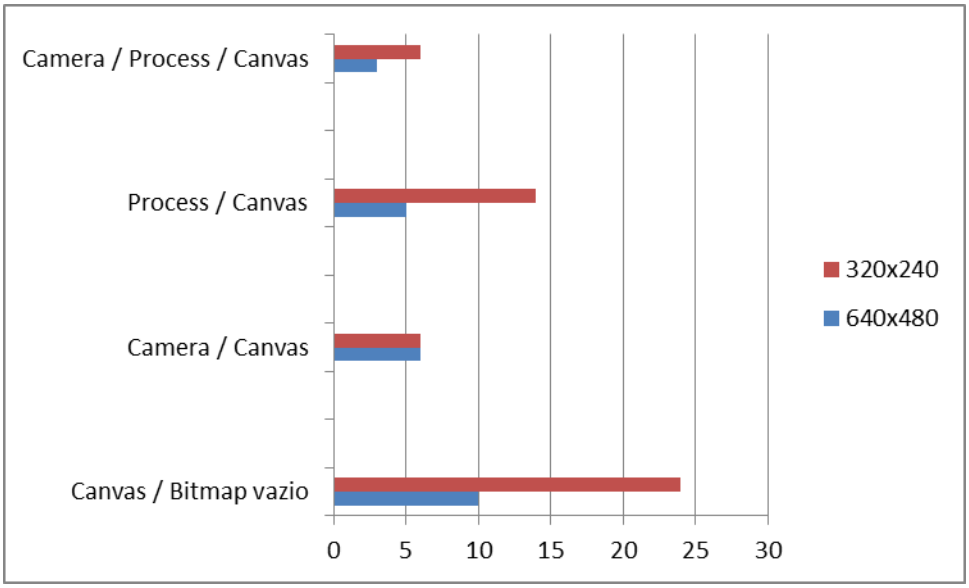


Figura 32 – Desempenho

Através da Figura 32 é possível observar que a taxa de FPS cai durante a captura da imagem pela câmera. Também é possível verificar que o processamento da imagem na resolução 320x240 *pixels* não chega a causar impacto no desempenho. Em ambas as resoluções foi alcançado o requisito estipulado de analisar pelo menos uma imagem por segundo.

Na validação dos resultados obtidos pela aplicação, foram comparados os resultados obtidos através de 10 alvos oficiais da modalidade carabina apoiada, que tiveram suas pontuações geradas por uma medição manual, por uma medição pela máquina da FCCTE e por uma medição através da aplicação desenvolvida. A Figura 33 mostra as medições na resolução 320x240 e a Figura 34 mostra as medições na resolução 640x480. Como foram utilizados os mesmos alvos nos testes nas duas resoluções, os resultados da medição manual e da medição pela máquina da FCCTE foram os mesmos nas duas figuras.

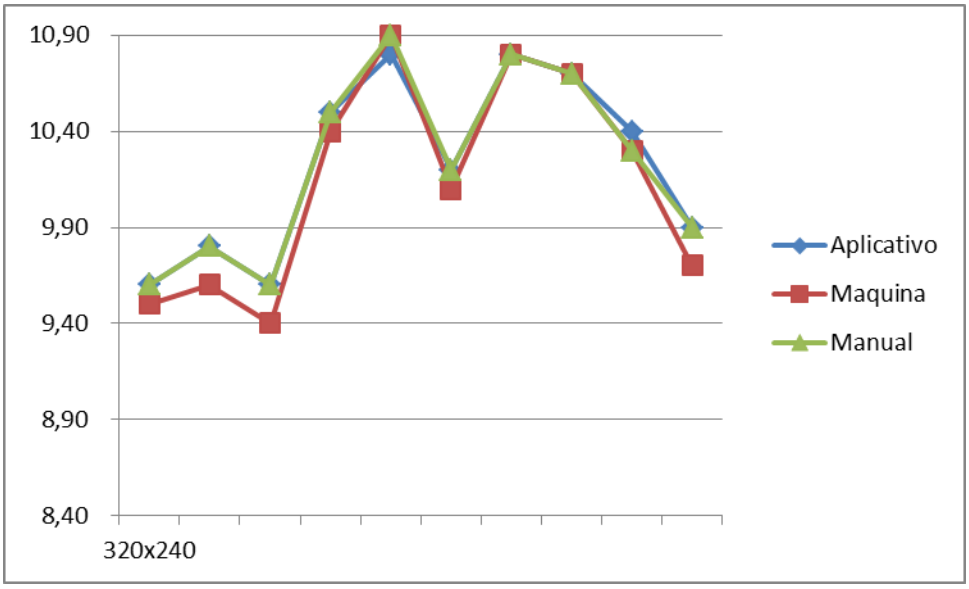


Figura 33 - Medição 320x240

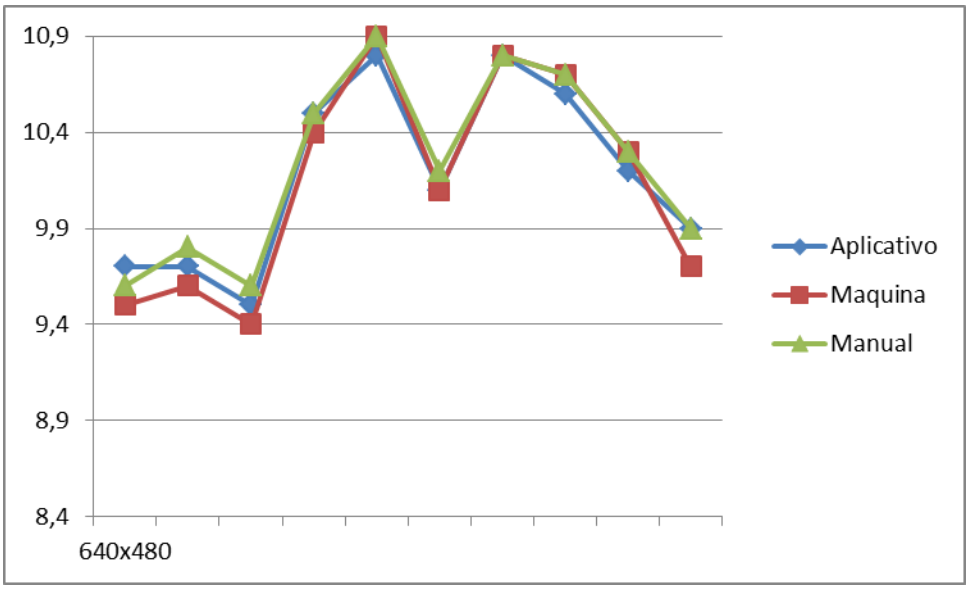


Figura 34 - Medição 640x480

Como se pode observar nas figuras acima em ambas as resoluções ocorrem diferenças entre os resultados obtidos pelos diferentes métodos. Comparando os resultados do método manual com os resultados obtidos pela aplicação na resolução 320x240, foram constatados resultados praticamente iguais, com poucas variações. Já na resolução 640x480 ocorreram mais diferenças nas pontuações, sendo neste caso então necessário um refinamento maior no tratamento da imagem. Durante os testes nas duas resoluções utilizadas não ocorreram problemas na detecção do alvo, não sendo necessária a finalização da análise e reconfiguração do operador Canny ou do raio dos alvos.

A Figura 35 apresenta um comparativo dos resultados obtidos pela aplicação entre as duas resoluções testadas.

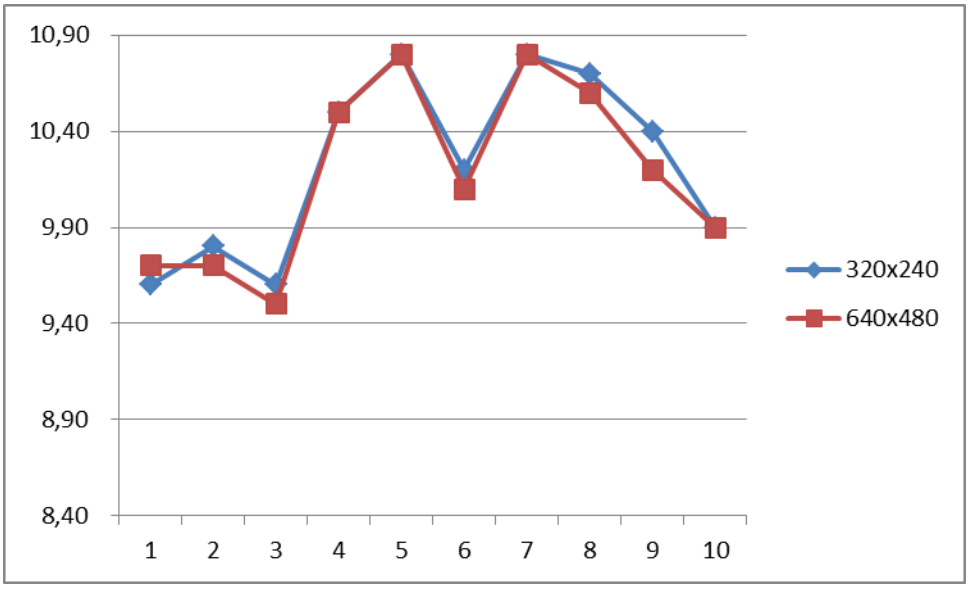


Figura 35 - Comparativo 320x240 e 640x480

Estas diferenças podem ser ocasionadas pelas oscilações na detecção dos círculos pela transformada de Hough ou pelo posicionamento da câmera. Porém este problema não interfere na validação dos resultados obtidos e nem no alcance dos objetivos propostos.

Por fim verificamos a necessidade da realização de mais testes em ambientes controlados, aumentando o número de alvos analisados e também variando a iluminação durante a detecção dos alvos e de seus disparos.

4 CONCLUSÕES

Este trabalho apresentou o projeto e desenvolvimento de uma aplicação para a plataforma Android capaz de detectar e reconhecer em tempo real, no caso 6 (seis) imagens por segundo na resolução de 320x240 e 3 (três) imagens por segundo na resolução de 640x480, alvos de competições de tiro e dos disparos sofridos pelos mesmos, calculando suas respectivas pontuações e disponibilizando os resultados ao competidor. A aplicação também se utilizou de uma câmera de baixo custo para efetuar a captura das imagens dos alvos analisados.

Através dos resultados obtidos pode-se perceber ainda a necessidade de um refinamento no processamento das imagens, para diminuir as oscilações na detecção dos círculos pela transformada de Hough. Outra necessidade é o desenvolvimento de uma calibragem automática da câmera, para possibilitar o posicionamento da mesma num ângulo que permita a utilização da aplicação numa situação real. Ambas as necessidades podem influenciar diretamente num cálculo mais preciso das pontuações dos disparos.

Em relação às tecnologias utilizadas, o ambiente de desenvolvimento Eclipse juntamente com os *plugins* do SDK da plataforma Android, mostram-se adequados e facilitaram o desenvolvimento. O uso da ferramenta EA 7.5 para a elaboração das especificações do trabalho atendeu todas as necessidades do projeto, não apresentando problemas ou dificuldades durante sua execução.

Por fim, em relação aos trabalhos correlatos pode-se perceber que as soluções propostas por Poffo (2010), que utiliza o processamento de imagens para a detecção de círculos nas placas de trânsito, por Starosky (2003), que calcula a linha de impedimento e a distância entre dois pontos através do processamento de imagens, e por Vasselai (2010), que utiliza dispositivos móveis da plataforma Android para auxiliar o usuário através do uso do conceito da realidade aumentada e do OpenGL ES, serviram de base para a realização deste trabalho. Pois o mesmo se utiliza da combinação de recursos e técnicas utilizados por estes três trabalhos.

4.1 EXTENSÕES

Como sugestões de extensões para continuidade do presente trabalho têm-se:

a) utilizar uma câmera externa com zoom, preferencialmente óptico, ou o uso de um zoom digital para aproximar o alvo, o que poderá facilitar a análise dos alvos e disparos, gerando resultados mais precisos;
b) desenvolver uma rotina para recortar e descartar a área externa da região do alvo, minimizando a região da imagem e otimizando assim o processo de análise dos disparos;
c) utilizar a biblioteca OpenCV para desenhar também todos objetos gráficos sobre as imagens, o que eliminaria a necessidade da utilização do OpenGL ES;
d) armazenar as imagens dos alvos analisados no banco de dados SQLite;
e) desenvolver uma rotina de calibragem automática da câmera, além de permitir o uso da câmera em ângulos diferentes;
f) desenvolver um aplicação servidor, para computadores de mesa, que irá receber as imagens, efetuar a análise e enviar as imagens já analisadas para o dispositivo móvel, possibilitando assim um tratamento maior nas imagens que pode influenciar na precisão no cálculo das pontuações;
g) utilizar um número maior de alvos nos testes para validar melhor os resultados obtidos pela aplicação;
h) testar a aplicação em ambientes controlados variando a iluminação para verificar o comportamento da detecção dos alvos e de seus disparos.

REFERÊNCIAS BIBLIOGRÁFICAS

BUENO, Marcelo L. **Visão computacional**: detecção de bordas através de algoritmo Canny. [S.l.], 2000. Disponível em: <http://www.inf.ufsc.br/~visao/2000/Bordas/>. Acesso em: 14 nov. 2012.

CBTE. **História e curiosidade**. [S.l.], 2012. Disponível em: <http://www.cbte.org.br>. Acesso em: 17 dez. 2012.

CONCI, Aura. **Canny**: detecção de borda. [S.l.], [2010?]. Disponível em: <http://www.ic.uff.br/~aconci/canny.pdf>. Acesso em: 14 nov. 2012.

COHEN, Marcelo; MANSSOUR, Isabel H. **OpenGL**: uma abordagem prática e objetiva. São Paulo: Novatec, 2006.

FCCTE. **Alvos**. [S.l.], 2012a. Disponível em: <http://www.fct.com.br/alvos.asp>. Acesso em: 17 dez. 2012.

_____. **Armas**. [S.l.], 2012b. Disponível em: <http://www.fct.com.br/armas.asp>. Acesso em: 17 dez. 2012.

_____. **História**. [S.l.], 2012c. Disponível em: <http://www.fct.com.br/historia.asp>. Acesso em: 17 dez. 2012.

_____. **Regulamentos**: carabina apoiada. [S.l.], 2012d. Disponível em: <http://www.fct.com.br/reg_apoiada_2012.asp>. Acesso em: 17 dez. 2012.

FOSCAM. **IP wireless FI8918W**. [S.l.], 2010. Disponível em: <http://www.foscam.com.br/produtos/detalhes/1/>. Acesso em: 05 nov. 2012.

_____. **What is Android?** [S.l.], 2012b. Disponível em: <http://developer.android.com/guide/basics/what-is-android.html>. Acesso em: 17 abr. 2012.

_____. **Writing real time games for Android**. [S.l.], 2009. Disponível em: <http://dl.google.com/io/2009/pres/WritingRealTimeGamesforAndroid.pdf>. Acesso em: 11 nov. 2012.

JAMUNDÁ, Teobaldo. **Visão computacional**: reconhecimento de formas. [S.l.], 2000. Disponível em: <http://www.inf.ufsc.br/~visao/2000/Hough/index.html>. Acesso em: 14 nov. 2012.

KHRONOS GROUP. **OpenGL ES**: the standard for embedded accelerated 3D graphics. [S.l.], 2012. Disponível em: <http://www.khronos.org/opengles/>. Acesso em: 19 abr. 2012.

_____. **OpenGL ES**: commom / commom-lite profile specification. [S.l.], 2004. Disponível em: <http://www.khronos.org/registry/gles/specs/1.0/opengles_spec_1_0.pdf>. Acesso em: 06 nov. 2012.

OPENCV. **Introduction**. [S.l.], 2012a. Disponível em: <http://docs.opencv.org/modules/core/doc/intro.html>. Acesso em: 17 abr. 2012.

_____. **OpenCV**. [S.l.], 2012b. Disponível em: <http://opencv.willowgarage.com/wiki/>. Acesso em: 17 abr. 2012.

_____. **OpenCV**: basic structures. [S.l.], 2012c. Disponível em: <http://docs.opencv.org/modules/core/doc/basic_structures.html#mat>. Acesso em: 14 nov. 2012.

_____. **OpenCV**: feature detection. [S.l.], 2012d. Disponível em: <http://docs.opencv.org/modules/imgproc/doc/feature_detection.html>. Acesso em: 14 abr. 2012.

_____. **OpenCV**: filtering. [S.l.], 2012e. Disponível em: <http://docs.opencv.org/modules/imgproc/doc/filtering.html>. Acesso em: 14 nov. 2012.

_____. **OpenCV**: miscellaneous transformations. [S.l.], 2012f. Disponível em: <http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor>. Acesso em: 14 nov. 2012.

PEDRINI, Hélio; SCHWARTZ, William R. **Análise de imagens digitais**: princípios, algoritmos e aplicações. São Paulo: Thomson Learning, 2008.

PISTORI, Hemerson; PISTORI, Jeferson; COSTA, Eduardo R. Hough-circles: um módulo de detecção de circunferências para o ImageJ. In: WORKSHOP SOFTWARE LIVRE, 6., 2005, Porto Alegre. **Anais eletrônicos...** Porto Alegre: UCDB, 2002. Não paginado. Disponível em: <http://www.gpec.ucdb.br/pistori/publicacoes/pistori_wsl2005.pdf>. Acesso em: 10 nov. 2012.

POFFO, Fernando. **Visual autonomy – protótipo para reconhecimento de placas de trânsito.** 2010. 54 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2010/344092_1_1.pdf>. Acesso em: 16 abr. 2012.

SCHEMBERGER, Elder E.; FREITAS, Ivonei; VANI, Ramiro. Plataforma Android. **Jornal Tech**, [S.l.], n. 1, não paginado, ago. 2009. Disponível em: <http://www.jornaltech.com.br/wp-content/uploads/2009/09/Artigo_Android.pdf>. Acesso em: 18 abr. 2010.

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. Disponível em: <http://www.bc.furb.br/docs/MO/2003/278826_1_1.pdf>. Acesso em: 16 abr. 2012.

TECHMUNDO. **Motorola XOOM**: um tablet dual-core com Android HoneyComb. [S.l.], 2011. Disponível em: <http://www.techtudo.com.br/ces-2011/noticia/2011/01/motorola-xoom-um-tablet-dual-core-com-android-honeycomb.html>. Acesso em: 05 nov. 2012.

TRINDADE, Fernando. **Técnicas de visão computacional para rastreamento de olhar em videos**: visão computacional. [S.l.], 2009. Disponível em: <http://almerindo.devin.com.br/index.php?option=com_content&view=article&id=78%3Atecnicas-de-computacao-visual-para-rastreamento-de-olhar-em-videos&catid=43%3Atrabalhos-de-alunos&Itemid=18&limitstart=2>. Acesso em: 06 nov. 2012.

VASSELAI, Gabriela T. **Um estudo sobre realidade aumentada para a plataforma Android.** 2010. 102 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2011/346536_1_1.pdf>. Acesso em: 16 abr. 2012.

WANGENHEIM, Aldo von. **Visão computacional**: seminário introdução à visão computacional. Florianópolis, [1998?]. Disponível em: <http://www.inf.ufsc.br/~visao/>. Acesso em: 06 nov. 2012.