DETECÇÃO DE USUÁRIO E ESTIMAÇÃO DE POSIÇÃO PARA INTERFACE COM REALIDADE VIRTUAL

THACIO GARCIA SCANDAROLI

DOUGLAS DAVID MELO

TRABALHO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

DETECÇÃO DE USUÁRIO E ESTIMAÇÃO DE POSIÇÃO PARA INTERFACE COM REALIDADE VIRTUAL

THACIO GARCIA SCANDAROLI

DOUGLAS DAVID MELO

ORIENTADOR: PROF. RICARDO LOPES DE QUEIROZ, ENE/UNB

TRABALHO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: .

BRASÍLIA/DF: NOVEMBRO – 2009.

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

TRABALHO DE GRADUAÇÃO

Detecção de usuário e estimação de posição para interface com realidade virtual

Thacio Garcia Scandaroli

Douglas David Melo

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro Eletricista

Banca Examinadora

Prof. Ricardo Lopes de Queiroz, ENE/UnB _____
Orientador

Prof. Geovany Araújo Borges, ENE/UnB _____
Examinador Interno

Prof. Alexandre Zaghetto, CIC/UnB _____

AGRADECIMENTOS

Acho que o projeto final é o símbolo mais forte de encerramento da graduação, e ao terminá-lo, não sei bem o sentimento disso, se é felicidade por terminar com sucesso 5 anos de faculdade ou tristeza por estar deixando uma das melhores fases da minha vida pra trás. Seja como for, não posso esquecer todas as pessoas que passaram pela minha vida e contribuíram pelo menos um pouco para o meu crescimento, e que sem elas não teria chegado até aqui.
Agradeço aos meus pais, que me apoiam seja qual for a situação e sempre me incentivam a todo momento. A meu irmão, que quando precisei sempre esteve lá, seja na rua ou em casa, e me deu grandes ajudas desde o início da minha vida acadêmica até nas dificuldades passadas neste trabalho.
Ao Prof. Queiroz por ter me dado não só esta como outras oportunidades. Melhor do que gostar do que faz é fazer o que gosta. Felizmente eu tive esta sorte.
Ao meu amigo e parceiro, não só de trabalho, Douglas, que com a ajuda dele já foram realizados mais projetos além destes.
E a todos os meus amigos que me fizeram companhia nessa jornada, seja estudando para uma prova, conversando ou se divertindo, e tornaram ela, com certeza, a melhor fase da minha vida até então.

Thacio Garcia Scandaroli

Dedico esse trabalho aos meus pais, João Luiz, in memorian, e Jovelina, pelo exemplo de perseverança, determinação e coragem.
A minha irmã, Laís, pelo companheirismo e amizade.
Aos meus professores, pela paciência e dedicação.
Ao meu colega de projeto e parceiro, Thacio, pelas discussões e novas idéias.
Aos meus colegas de estudo, por tudo que passamos juntos.
Às pessoas que, de alguma forma, mesmo sem saber, passaram pela minha vida e me incentivaram a prosseguir.

Douglas David Melo

RESUMO

DETECÇÃO DE USUÁRIO E ESTIMAÇÃO DE POSIÇÃO PARA INTERFACE COM REALIDADE VIRTUAL

Autor: Thacio Garcia Scandaroli
Autor: Douglas David Melo

Orientador: Prof. Ricardo Lopes de Queiroz, ENE/UnB

Departamento de Engenharia Elétrica

Brasília, 30 de novembro de 2009

Neste trabalho é desenvolvida uma interface entre usuário e ambiente virtual utilizando uma câmera. O usuário interage com o ambiente virtual se movendo em frente a tela do computador, e o ambiente virtual muda de perspectiva de acordo com a posição em que o usuário estiver. Para o desenvolvimento do projeto, é considerado que a face do usuário esteja em uma posição que a câmera a capture de maneira frontal. É feito um estudo de métodos para detecção facial e de olhos. Após a detecção, é desenvolvido um método de cálculo para a posição do usuário tendo como referência imagens capturadas por uma câmera, a partir dos olhos detectados na imagem. É utilizado filtro de Kalman na detecção de faces, de olhos e na estimação de posição para predizer e corrigir as variáveis dessas etapas. Como última etapa, são feitos três ambientes virtuais que utilizam a informação de posição do usuário para o desenho de sua perspectiva. Como resultado, é possível detectar a face e os olhos do usuário e estimar aproximadamente a posição dele em relação à câmera, com bons resultados. O programa final opera a uma taxa de 25 frames por segundo e o ambiente virtual se move de acordo com a posição do usuário no espaço.

ABSTRACT

USER DETECTION AND POSITION ESTIMATION FOR A VIRTUAL REALITY INTERFACE

Author: Thacio Garcia Scandaroli
Author: Douglas David Melo

Advisor: Prof. Ricardo Lopes de Queiroz, ENE/UnB

Departamento de Engenharia Elétrica

Brasília, November 30, 2009

This work develops a user interface in a Virtual Reality environment using a camera. The user interacts with the VR environment by moving in front of the monitor and the VR environment changes its perspective according to his/her position. We consider frontal facioal image capture in this project. Methods for facial and eye detection have been studied. After the implementation of face and eye detection, it is developed a method that calculates the user's position, having the camera as reference. The captured imagery is analyzed and faces and eyes are located within the images. We used Kalman filters for prediction and correction of facial detection, eye detection and position estimation. As a last step of this work, three differents VR environments have been made, changing their perspective according to the user's position. As result, it is possible to detect faces and eyes, and estimate the user's aproximate position. The final program runs at a speed of 25 frames per second and the VR ambient moves according to the user's position.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ALGORITMOS

Símbolos Latinos

x	Vetor das variáveis de estado
y	Vetor das variáveis observáveis
A	Matriz do sistema
Q	Matriz de covariância do ruído do sistema
H	Matriz de saída do sistema
R	Matriz de covariância do ruído das medidas
K	Matriz de ganho do filtro de Kalman
P	Matriz de covariância do erro
D_{eyes}	Distância na imagem, em pixels, entre os olhos do usuário
f_x	Foco na direção x, em pixels, da câmera
f_y	Foco na direção y, em pixels, da câmera
X	Distância do usuário até à câmera, na direção x tendo como referência a câmera
Y	Distância do usuário até à câmera, na direção y tendo como referência a câmera
Z	Distância do usuário até à câmera, na direção z tendo como referência a câmera
c_x	Posição horizontal do pixel central da imagem
c_y	Posição vertical do pixel central da imagem

Subscritos

k	índice da amostra
$k-1$	índice da amostra anterior

Sobrescritos

$\hat{}$	estimação
$-$	predição

Siglas

GPDS	Grupo de Processamento Digital de Sinais
VR	Virtual Reality (*Realidade Virtual*)

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO E MOTIVAÇÃO

Com o grande avanço tecnológico, computadores cada vez mais potentes são desenvolvidos, sendo assim possível desenvolver softwares que utilizem cada vez mais cálculos e processamento. Com este avanço, procura-se melhorar a forma de interação entre usuário e computador, tornando-a mais simples e fácil. Essa evolução possibilitou um grande avanço no entretenimento digital, como filmes em 3D e jogos eletrônicos.

Em 1972 foi lançado no mercado o primeiro videogame doméstico que se chamava Magnavox Odyssey. No mesmo ano, foi lançado em versão arcade o jogo *Home PONG*, que pode ser visto na Figura 1.1, conhecido como um dos primeiros jogos a serem lançados e que competiu diretamente com o videogame Odyssey.

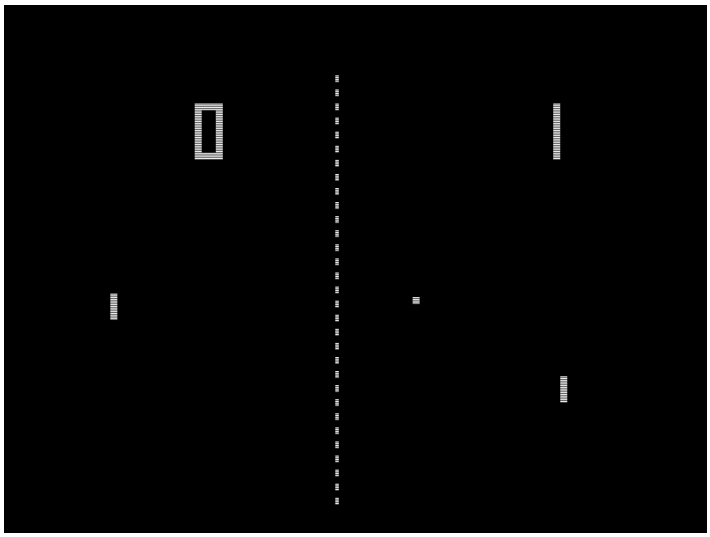


Figura 1.1: Jogo PONG, versão arcade.

Desde então, a tecnologia avançou assim como a computação gráfica e os modos de interação entre usuário e ambiente virtual. Com a evolução da computação gráfica, foi possível criar imagens digitais cada vez mais detalhadas e realistas, levando uma nova experiência visual ao cinema e aos jogos eletrônicos. A Figura 1.2 apresenta uma imagem do jogo crysis, mostrando a evolução gráfica mencionada. Além do avanço gráfico, os jogos eletrônicos sempre tentam inovar na interação entre usuário e ambiente virtual do jogo, tentando torná-los cada vez mais imersivos.

Hoje em dia existe o conceito de realidade virtual, que é uma tecnologia que permite a interação entre um usuário e um ambiente virtual, seja este criado para tentar simular o mundo real ou um mundo imaginário. Com a mescla dos elementos reais e virtuais, é formada uma nova área denominada de Realidade Aumentada. Ela combina tais elementos

Figura 1.2: Crysis, jogo lançado em 2007.

em tempo real para fazer a interação entre esses dois ambientes. Atualmente, a maior parte das linhas de pesquisa em Realidade Aumentada tem como foco utilizar vídeos em tempo real do usuário e acrescentar elementos virtuais.

Como formas de melhorar a interação em questão, imagens, sons e gestos são visados para a mescla dos elementos virtuais e reais. Em novembro de 2006, a empresa Nintendo lançou um videogame doméstico denominado de Wii. Ele possui um controle que captura os gestos do usuário, possibilitando ao usuário uma forma de interação com o mundo virtual ainda pouco explorada nos jogos eletrônicos. A Figura 1.3 mostra duas pessoas jogando tênis de mesa no console wii.

O controle do Wii possui um sensor infravermelho e, utilizando uma barra sensorial que possui dois sensores infravermelhos localizada em cima ou em baixo da TV, é possível, por triangulação, calcular a posição de tal controle. Baseado nisso, Johnny Chung Lee[1] da universidade Carnegie Mellon, desenvolveu um projeto no computador utilizando um wiimote. Neste projeto, ele posicionou o controle do wii a frente da TV e colocou a barra sensorial em um óculos, assim podendo estimar a posição do usuário. Com isso, ele desenvolveu um ambiente virtual que muda de perspectiva de acordo com tal posição, tal rojeto pode ser visto na Figura 1.4

Esta é a inspiração deste projeto, desenvolver um ambiente virtual que mude de perspectiva de acordo com a posição do usuário, utilizando uma câmera caseira USB como sensor.

[1] http://johnnylee.net/, acessado em novembro de 2009.

Figura 1.3: Avô e neta jogando tênis de mesa no videogame Wii, lançado pela Nintendo em 2006.

Figura 1.4: Imagem retirada do vídeo gravado por Johnny Chung Lee para demonstrar seu projeto utilizando um wiimote.

No grupo de Processamento Digital de Sinais (GPDS), vinculado ao Departamento de Engenharia Elétrica da Universidade de Brasília, há uma frente que trabalha com processamento de vídeo. Assim, este projeto visa implementar, por meio de reconhecimento facial e detecção de olhos do usuário, uma forma de interação visual entre usuário e um ambiente virtual no computador. O usuário move o rosto para uma direção enquanto olha para a tela e o cenário virtual muda de acordo com este movimento

1.2 DEFINIÇÃO DO PROBLEMA

Este projeto não será baseado em um hardware específico, mas sim em um desenvolvimento para câmeras genéricas.

O desafio do projeto é fazer um rastreio dos olhos de tal modo que se consiga manter uma experiência de uso do programa agradável para o usuário. Para isso, é necessário que a detecção de face e olhos tenham boas taxas de detecções. Além disso, sabe-se que tais detecções consomem muito processamento. Logo, é preciso que sejam feitas de maneira rápida, para que o programa consiga rodar em tempo real. Também é necessário conseguir calcular a posição do usuário com a utilização de uma câmera USB, esta é a razão da detecção de face e de olhos.

Com a detecção dos olhos do usuário, é feita a estimação da posição deste em relação a câmera e depois é desenhado o cenário virtual de acordo com a sua posição. Este processo deve rodar a uma alta taxa de frames por segundo, caso contrário, o usuário teria uma experiência de interação com o ambiente virtual inadequada. O programa foi desenvolvido na linguagem C++ com o auxílio das bibliotecas OpenCV, para processamento de imagem, OpenGL, para desenho do ambiente virtual, e newmat[1], para cálculo de matrizes. O sistema pode ser visto na Figura 1.5.

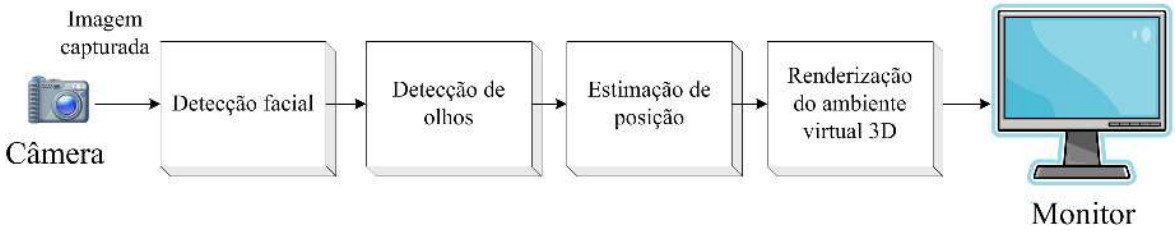


Figura 1.5: Diagrama de blocos do projeto.

1.3 OBJETIVOS DO PROJETO

O projeto tem como objetivo implementar um programa que faça a interação entre o usuário e um ambiente virtual no computador. O resultado do projeto possui um caráter subjetivo, sendo o resultado a interação do usuário e do ambiente 3D. Como principais pontos do projeto para serem implementados, temos:

- Detecção facial
- Detecção de olhos
- Estimação da posição do usuário

[2]http://www.robertnz.net/index.html, acessado em novembro de 2009

- Predição e correção das variáveis de projeto utilizando filtro de Kalman
- Desenho do ambiente 3D de acordo com a posição do usuário

Com todo o sistema pronto, devemos ajustar os parâmetros de projetos de modo à proporcionar ao usuário uma interação agradável. Assim, sendo o resultado de caráter subjetivo, é dada menos atenção à precisão dos dados, desde que o resultado seja visualmente satisfatório.

1.4 APRESENTAÇÃO DO MANUSCRITO

No Capítulo 2, são apresentados os fundamentos teóricos utilizados no projeto, sendo entre estes os assuntos de detecção de olhos e detecção facial, estimação baseadas em filtro de Kalman, calibração de câmera e estimação de posição do usuário em visão monocular. Já o Capítulo 3 é focado no desenvolvimento do trabalho, explicando todas as etapas de implementação do projeto. No Capítulo 4, são mostrados e analisados os resultados do trabalho. Finalmente, o Capítulo 5 apresenta as conclusões do projeto e sugestões para trabalhos futuros.

2 CONCEITOS BÁSICOS

2.1 FILTRO DE KALMAN

Rudolph Emil Kalman publicou em 1960 um artigo descrevendo um conjunto de equações que constituem um processo recursivo eficiente de estimação para solucionar problemas lineares (ou que possam ser linearizados em torno de um ponto) relacionados à filtragem de dados discretos, conhecido como filtro de Kalman [1]. Através da observação de uma variável, denominada variável de observação, pode-se estimar eficientemente outra, não observável, dita variável de estado, sendo possível com isso estimar estados passados e prever estados futuros. Ou seja, se um modelo linear pode ser utilizado para descrever um sistema, e, as incertezas dos dados de entrada e do próprio sistema podem ser modeladas estatisticamente (como ruídos brancos e não correlacionadas), o filtro de Kalman será capaz de encontrar a melhor estimativa baseada na correção de cada medida individual, minimizando os possíveis erros provenientes das imperfeições dos modelos matemáticos e dos sensores de coleta de dados, além dos ruídos provenientes do próprio sistema e do ambiente no qual ele está inserido.

Em uma visão mais moderna e voltada à aplicação, o filtro de Kalman pode ser visto como um algoritmo recursivo ótimo (minimiza o erro) de processamento de dados. Isso porque, apesar da conotação típica de filtro como sendo um dispositivo (caixa preta) composto por circuitos eletrônicos, o fato é que na maioria das aplicações do filtro de Kalman, este é apenas um programa de computador executado por um processador. Ele incorpora toda a informação dada a ele, processa as amostras disponíveis e estima o valor atual das variáveis de interesse, usando o conhecimento do sistema objeto de estudo, da dinâmica dos instrumentos de medição e do comportamento estatístico dos ruídos.

O filtro é dito recursivo porque não precisa que todas as amostras anteriores sejam armazenadas em algum tipo de memória e reprocessadas toda vez que novos dados são obtidos, mas apenas do estado anterior ao atual, permitindo então a atualização dos dados em tempo real. Essa é uma característica vital para a utilização prática do filtro em um grande número de sistemas, de controle e de rastreamento, em especial.

A estimação feita pelo filtro de Kalman usa um controle do tipo realimentação: o filtro estima o estado do processo em algum momento e obtém a realimentação na forma de medidas (de ruído). As equações que descrevem o filtro formam dois grupos: equações de predição e equações de correção [1]. As primeiras são responsáveis pela atualização do tempo, ou seja, pelo avanço das variáveis de estado e das covariâncias no tempo para se obter dessa maneira as estimativas anteriores do próximo estado; já as equações de correção são responsáveis pelas atualizações das medidas, ou seja, pela retroalimentação, incorporando uma nova in-

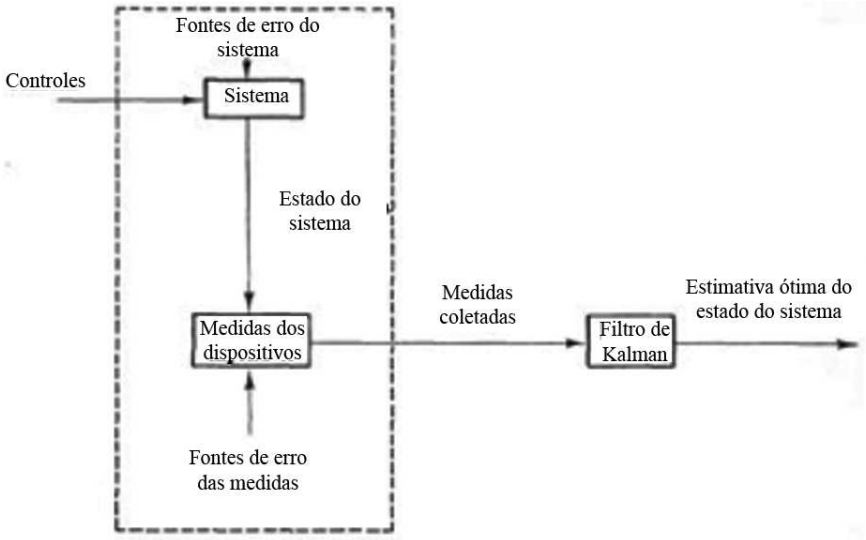


Figura 2.1: Sistema utilizando filtro de Kalman.

formação da variável observável nas estimativas anteriores, obtendo uma melhora (ganho) na estimativa posterior.

As equações que estabelecem o filtro de Kalman buscam resolver o problema da estimativa das variáveis de estado a partir dos dados provenientes das amostras. Dada uma série temporal multivariada, ou seja, um conjunto de observações feitas seqüencialmente no tempo $\mathbf{y}_k$, sendo este um conjunto finito de pontos ou um intervalo finito, formada por N elementos, denominados de variáveis observáveis e formadores de um vetor, $N \times 1$, $\mathbf{y}_k \in \mathbb{R}^N$, as variáveis aleatórias que representam o ruído das medidas e do sistema devem ser independentes, com ruído branco e com distribuição normal de probabilidade. Para as equações que serão mostradas, as variáveis que possuem sobrescrito $-$ são variáveis preditas e as que possuirem sobrescrito são variáveis estimadas do sistema no instante anterior. Assim estabelecido, as equações de predição utilizadas na resolução do problema proposto são as seguintes:

$$\mathbf{x}_k^- = \mathbf{A}\hat{\mathbf{x}}_{k-1} \tag{2.1}$$

$$\mathbf{P}_k^- = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q} \tag{2.2}$$

Nas equações apresentadas, $\hat{\mathbf{x}}_k$ é o vetor de estados estimados, ou o vetor a *priori*, $\mathbf{x}_k-$ é o vetor de estados predito, ou seja, a *posteriori*, a matriz $\mathbf{A}_{m\times n}$ é a matriz do sistema discretizado e ela relaciona o estado anterior $k-1$ ao estado presente k, enquanto a matriz $\mathbf{Q}_{m\times n}$ representa a matriz de covariância do ruído do sistema e a matriz $\mathbf{P}_k$ é a matriz de covariância do erro. O sinal de negativo sobrescrito representa valores preditos pelo filtro enquanto que o sinal circunflexo representa estados estimados. Logo, na equação 2.1, o vetor de estados $\mathbf{x}_k$, que possui as variáveis de estados do sistema, é obtido pela multiplicação do vetor de estados da amostra anterior $\mathbf{x}_{k-1}$ pela matriz do sistema $\mathbf{A}$. Na etapa de predição,

também é feita o cálculo da da matriz de covariância do erro $\mathbf{P}_k^-$ utilizando a equação 2.2.

Já as equações de correção são as apresentadas a seguir:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}^T (\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R})^{-1} \tag{2.3}$$

$$\hat{\mathbf{x}_k} = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}\hat{\mathbf{x}}_k^-) \tag{2.4}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^- \tag{2.5}$$

$$\tag{2.6}$$

Na etapa de correção, $\mathbf{K}_k$ é a matriz de ganho do filtro de Kalman, $\mathbf{y}_k$ é o vetor que contêm a medição das variáveis observáveis do sistema, e a matriz $\mathbf{I}$ é a matriz identidade. Depois, é gerado o estado próximo através do valor de $\mathbf{y}_k$ e do valor do ganho do filtro, obtido na equação 2.3. Finalmente é feita a estimativa do erro da covariância do estado posterior, $\mathbf{P}_k$. A matriz $\mathbf{H}_{m\times n}$ relaciona o estado à medida $\mathbf{y}_k$ e a matriz $\mathbf{R}_{m\times n}$ representa a matriz de covariância do ruído das medidas. O quadro na figura 2.2 mostra de forma esquemática a operação do filtro de Kalman [1].

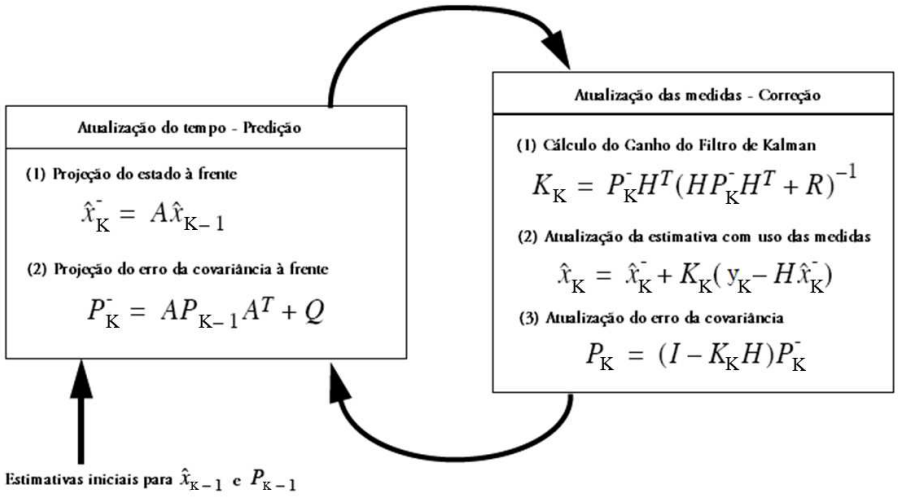


Figura 2.2: Operação do filtro de Kalman.

A idéia é aplicar as equações de atualização do tempo (predição), dados os valores iniciais, e obter assim as predições tanto para o estado a *posteriori* quanto para o seu erro de covariância. Na seqüência, são aplicadas as equações de atualização das medidas (correção) que, caso sejam obtidos dados provenientes das amostras, corrigem a predição feita pelo filtro. Caso contrário, a predição é utilizada como resultado na saída do filtro e o processo é realizado novamente, para um próximo estado.

Para o sistema, é necessário primeiramente definir as variáveis observáveis do sistema que serão medidas, e estas variáveis formam o vetor $\mathbf{y}_k$. Após isso, é possível definidir as variáveis de estado que irão compor o vetor de estado $\mathbf{x}$, que possui variáveis observáveis

e não observáveis. Com tais vetores definidos, é possível definir a matriz de evolução do sistema **A** e a matriz de saída do sistema **H**. As matrizes de covariância de ruído do sistema e das medidas, **Q** e **R**, respectivamente, são inicializadas de acordo com cada sistema, sendo necessário a escolha dos valores para cada matriz que corresponda ao sistema em questão.

2.2 DETECÇÃO FACIAL

A detecção de face tem como objetivo, em uma dada imagem, encontrar faces caso elas existam na imagem. Há vários desafios a serem superados em se tratando dessa detecção pois há muitas variáveis que podem prejudicar a detecção e precisam ser contornadas, como:

- **Orientação da cabeça da pessoa** - Há diversas orientações, de maneira frontal, perfil, olhando para cima ou para baixo, entre outras;
- **Iluminação** - A variação da iluminação pode causar sombras no rosto da pessoa, prejudicando a detecção
- **Oclusão** - O bloquei parcial do objeto a ser detectado na imagem implica em complicações para a detecção;
- **Escala** - Variação do tamanho da face na imagem capturada pode prejudicar a detecção;
- **Presença ou ausência de componentes faciais** - Como barba, bigode, óculos, entre outros, podem prejudicar a detecção,
- **Expressões faciais** - Há diversas expressões possíveis, como chorando, gritando, entre outras, que podem complicar a detecção.

A imagem capturada da face da pessoa varia de acordo com a orientação de sua cabeça, podendo ser capturada da forma frontal, de perfil, inclinada, entre outras posições. Variação de iluminação e de escala, objetos causando oclusão na face, presença de óculos, barbas e bigodes, e expressões faciais dificultam a detecção facial.

A detecção facial pode ser separada em categorias [2]:

Abordagem por características invariantes: Consiste em procurar por características comuns de uma face.

Métodos baseados em modelos: Vários padrões de faces são armazenados e é feita a correlação entre tais padrões e partes da imagem.

Métodos baseados em aparência: É levantado um modelo que aprende, por meio de várias imagens, a identificar faces. Para estes métodos, são utilizados classificadores como redes neurais [3] ou SVM (*support vector machines*) [4].

Cada método tem sua vantagem e desvantagem, assim, cada um é melhor empregado em uma dada situação.

A detecção de faces baseada em AdaBoost desenvolvido por Viola-Jones [5] consegue detectar faces em tempo real em uma imagem. Este é o método mais utilizado atualmente em aplicações que necessitam de velocidade, e ele será utilizado neste projeto.

2.2.1 Detecção baseada em características do tipo Haar utilizando AdaBoost

O método de detecção facial primeiramente apresentado por Paul Viola e Michael Jones [5], comumente chamado de método Viola-Jones, utiliza um algoritmo de aprendizado de máquina AdaBoost, *Adaptative boosting*, apresentado em 1995 [6]. Este algoritmo consiste em vários classificadores fracos em série, obtendo assim um classificador forte. Um classificador fraco é um classificador que não tem grandes probabilidades de acerto, como por exemplo 51% de chance de acerto. O AdaBoost é adaptativo pois ele dá preferência aos classificadores que tiveram mais sucesso, em detrimento dos que falharam em alguma classificação, utilizando pesos.

Em uma imagem, uma característica de dois retângulos (*two-rectangle feature*) é a diferença entre a soma dos pixels de dois retângulos adjacentes. Um característica de três retângulos (*three-rectangle feature*) considera três retângulos adjacentes e é a diferença entre a soma dos pixels dos dois retângulos extremos e do retângulo do meio. Já um característica de quatro retângulos (*four-rectangle feature*) considera quatro retângulos dispostos na forma 2×2 e é a diferença entre a soma dos pixels dos dois retângulos da diagonal principal e a soma dos pixels dos dois retângulos da outra diagonal. Estas características são chamadas de características do tipo Haar (*Haar-like features*) e são mostradas na Figura 2.3.

Apesar de simples a forma de calcular, obter as características de retângulos é um processo lento devido a quantidade de operações matemáticas existentes. Essas características podem ser caculadas de forma rápida utilizando uma representação intermediária, chamada de *integral image*, apresentadas em [5] por Viola e Jones . A *integral image* de um ponto (x, y) pode ser calculada somando todos os pixels acima e a esquerda deste ponto.

Fazer tal soma para cada pixel na imagem é um cálculo proibitivo, mas é possível fazer a soma acumulada, assim, calculando a *integral image* em apenas uma percorrida pela imagem. Calculada a *integral image*, é possível calcular facilmente as características de dois, três ou quatro retângulos.

O código AdaBoost é utilizado em um banco de imagens para decidir quais features serão consideradas para a detecção. São definidos limiares (*theresholds*) para cada característica,

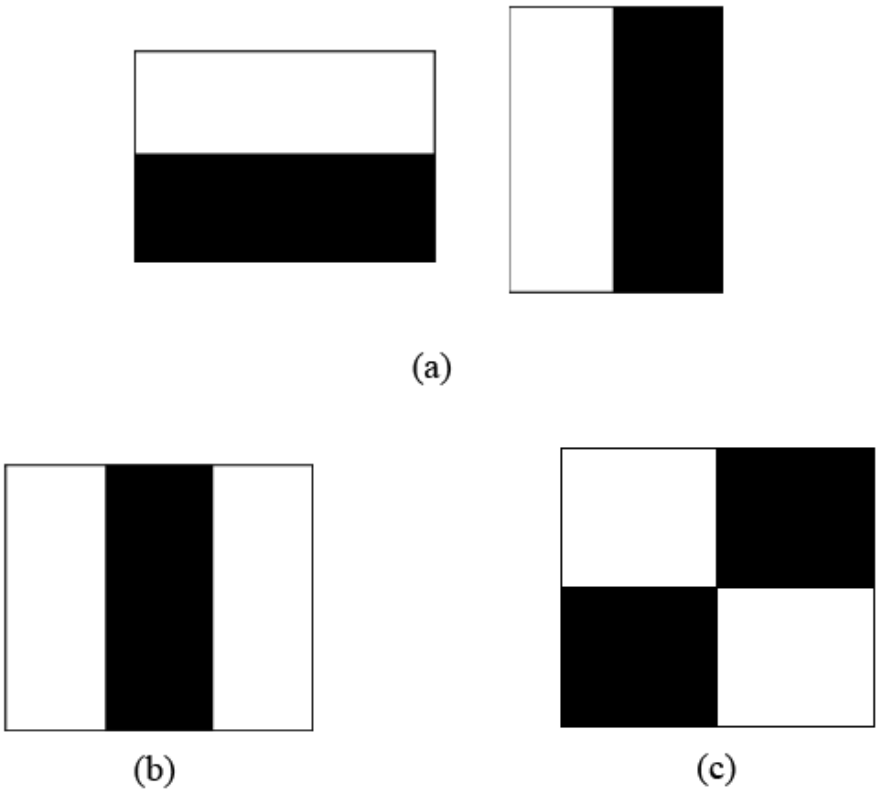


Figura 2.3: Exemplos de características do tipo Haar (a) característica de dois retângulos (b) característica de três retângulos (c) característica de quatro retângulos

caso a característica obtida do retângulo esteja fora desse limite, aquela região é descartada como candidato a possuir uma face. Este trabalho é feito por um classificador que pode classificar uma ou mais features.

Colocando os classificadores em cascata, a detecção começa a eliminar resultados falsos-positivos. O método começa a procura utilizando classificadores mais fracos para eliminar regiões que não são de interesse e nas etapas seguintes utiliza classificadores que consigam uma maior taxa de acertos.

O método originalmente desenvolvido por Viola e Jones foi extendido por LienHart e Maydt em [7], para utilizar features inclinadas em 45 graus.

2.3 DETECÇÃO DE OLHOS

Dada uma imagem, o objetivo da detecção de olhos é determinar na imagem a área de pixels que se encontram possíveis olhos. Existem duas categorias abrangentes para detecção de olhos em uma imagem.

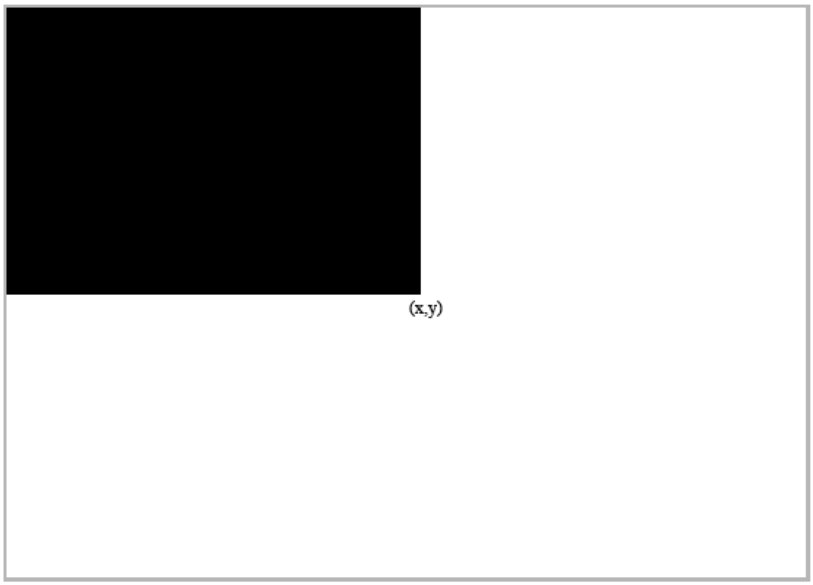


Figura 2.4: A *integral image* de um ponto (x, y) é a soma de todos os pixels acima e a esquerda desse ponto.

Detecção Ativa: Para a detecção ativa, é utilizada iluminação infra-vermelho (*Infrared*, IR) para realçar os olhos do usuário [8], pois a iluminação IR possui propriedades reflexivas na pupila que ajudam na localização dos olhos do usuário e isto é muito vantajoso em casos que haja muita variação de luminosidade no ambiente ou pouca luminosidade. É necessário hardware específico para tal iluminação, tornando assim inviável para aplicações que tenham como objetivo serem simples e portáveis.

Detecção Passiva: Os métodos passivos localizam os olhos em uma imagem a partir do processamento de uma imagem contendo o espectro visível e iluminação ambiente.

Para a cada detecção, ativa ou a passiva, há as mesma categorias apresentadas na seção 2.2, sendo elas: abordagem por características invariantes, métodos baseados em modelos e métodos baseados em aparência.

2.3.1 Detecção baseada em características do tipo Haar utilizando AdaBoots

O método apresentado na seção 2.2.1 serve para detecções além de faces, ele serve para detecção de objetos rígidos em uma determinada perspectiva. Como o classificador é treinado para um banco de imagens, é possível treiná-lo para os olhos, tendo também uma detecção rápida.

Neste projeto, este é o método utilizado para ser feita a detecção de olhos.

2.3.2 Detecção utilizando informação dos pixels nas bordas

Em [9], é proposto um método baseado em características, ou seja, características geométricas. Primeiramente é detectada a face na imagem, pelo método proposto por [5]. Em seguida, é obtida as bordas da imagem utilizando o método de *canny* [10] e é definido um vetor para cada pixel da borda apontando para o pixel da borda mais perto. O comprimento e curvatura destes vetores são associados a cada pixel. Após, é aplicado o PCA (*Principal Component Analysis*) [11] em um banco de imagens de olhos para treinamento e obtenção de autovetores. Para detectar a imagem em interesse, o valor do comprimento e curvatura dos candidatos a olho são projetados e comparados pelos autovetores para a confirmação da detecção dos olhos.

2.3.3 Método baseado nos classificadores Bayesianos

Em [12], é feita a comparação de performance entre três métodos, um baseado em regressão, um baseado no classificadores Bayesianos e outro discriminativo baseado no algoritmo AdaBoost. Como conclusão, o método que obteve melhor performance foi o de classificadores Bayesianos, apesar de sua simplicidade em relação ao AdaBoost. O método utilizando classificador Bayesianos aprende modelos de aparência de olhos e de objetos que não são olhos e aplica a regra de Bayes para produzir a probabilidade de ser um olho por volta da região candidata a detecção.

2.4 CALIBRAÇÃO DE CÂMERA

A calibração de câmera é uma etapa importante no desenvolvimento de projetos que utilizam esse tipo de dispositivo por relacionar as medidas provenientes deste com as medidas reais, do mundo real. A relação entre as unidades naturais da câmera (pixels) e as unidades do mundo físico (metros, por exemplo) é um aspecto crítico na tentativa de reconstruir um ambiente 3D, por exemplo.

O processo de calibração de câmera permite a obtenção de informações relacionadas à geometria interna da mesma e também a distorção produzida pela sua lente, conhecidos como parâmetros intrínsecos. Além disso, a posição e orientação 3D da câmera em relação a determinado sistema de coordenadas, ou seja, a relação espacial entre a câmera e o mundo, formam os parâmetros ditos extrínsecos, que representam a posição da origem do referencial imagem e sua orientação em relação ao referencial objeto.

Um modelo simples para descrição da câmera, conhecido como *pinhole*, considera um único ponto de entrada de luz em um plano e a projeção desse feixe em outro mais afastado. Como resultado, essa imagem está sempre em foco e o tamanho da imagem em relação a distancia do objeto é dada por um único parâmetro da câmera: a distância focal. A Figura

2.5 representa esse modelo.

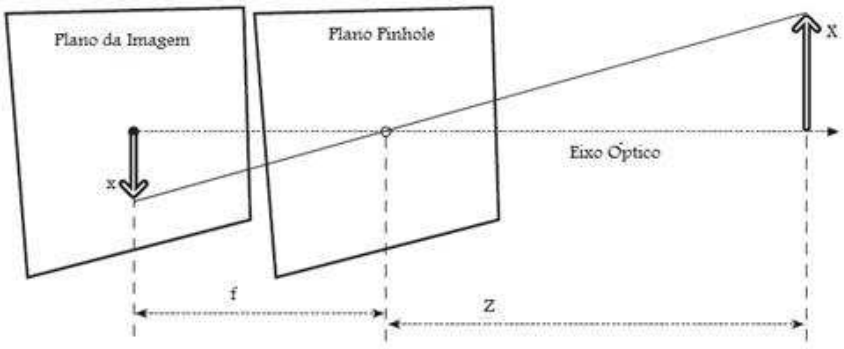


Figura 2.5: Modelo *pinhole*

Onde f é a distância focal da câmera, Z é a distância da câmera ao objeto, X é a altura ou comprimento do objeto e x é a altura ou comprimento da imagem. Assim, por trigonometria:

$$-x = f\frac{X}{Z}$$

Fazendo a expansão dessa idéia para as dimensões da imagem, ou seja, para as coordenadas x e y, têm-se o resultado mostrado na Figura 2.6.

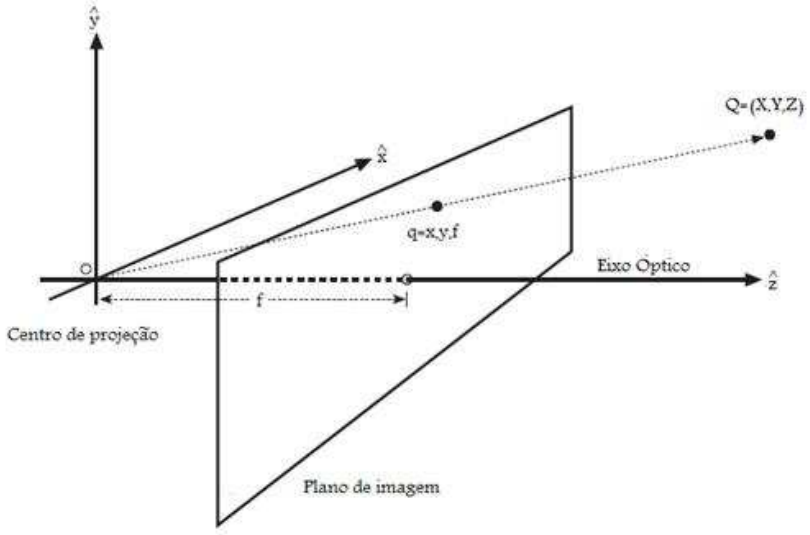


Figura 2.6: Representação de uma imagem capturada.

Na figura 2.6 f continua sendo a distância focal, Q é o ponto representado e q é a representação de Q no plano de imagem. Porém, por ser uma interpretação em duas dimensões, a distância focal f deve ser considerada em duas dimensões, f_x e f_y, por estas poderem apresentar valores diferentes (o que geralmente ocorre para câmeras simples). Além disso, uma consideração importante é que, na maioria das vezes, a câmera não está posicionada no

centro da imagem de exibição, ou seja, o centro de coordenadas localizado no centro da tela não coincide com o eixo óptico. Sendo assim, dois valores de *offset* devem ser considerados para adequação dessas referências. A partir dessas considerações, duas equações podem ser estabelecidas:

$$x_{tela_{(pixels)}} = f_{x_{(pixels)}}(\frac{X}{Z}) + c_{x_{(pixels)}} \tag{2.7}$$

$$y_{tela_{(pixels)}} = f_{y_{(pixels)}}(\frac{Y}{Z}) + c_{y_{(pixels)}} \tag{2.8}$$

Onde $x_{tela_{(pixels)}}$ e $y_{tela_{(pixels)}}$ são as coordenadas na imagem apresentada pela câmera, $f_{x_{(pixels)}}$ e $f_{y_{(pixels)}}$ são as distâncias focais em cada uma das direções, $c_{x_{(pixels)}}$ e $c_{y_{(pixels)}}$ são os parâmetros de *offset*, todos esses em pixels, e X,Y e Z são as coordenadas do objeto real em relação a tela.

Para a obtenção dos focos f_x e f_y, foi utilizado um processo de calibração que fornece além desses dois parâmetros, vários outros que podem ser de interesse para outras aplicações. O *software* utilizado foi o Matlab 6.0, que fornece um *toolbox* de calibração de câmeras baseado no método proposto por *Heikkilä* e que utiliza um procedimento de quatro passos para calibração: o primeiro consiste na determinação de uma solução analítica que proporcione uma aproximação dos parâmetros intrínsecos e extrínsecos da câmera; no segundo é feita uma estimação não-linear dos parâmetros através do método dos mínimos quadrados para minimização dos resíduos entre o modelo e as observações; no terceiro, é feita a correção e posterior extração de pontos de calibração na imagem devido à distorção do padrão de calibração causada pela projeção perspectiva; e finalmente no quarto é aplicado um modelo empírico que compensa distorções radiais e tangenciais da lente.

Em primeiro lugar, são tiradas várias fotos de um tabuleiro de xadrez, em diferentes posições, que serão utilizadas na calibração, como as apresentadas na Figura 2.7

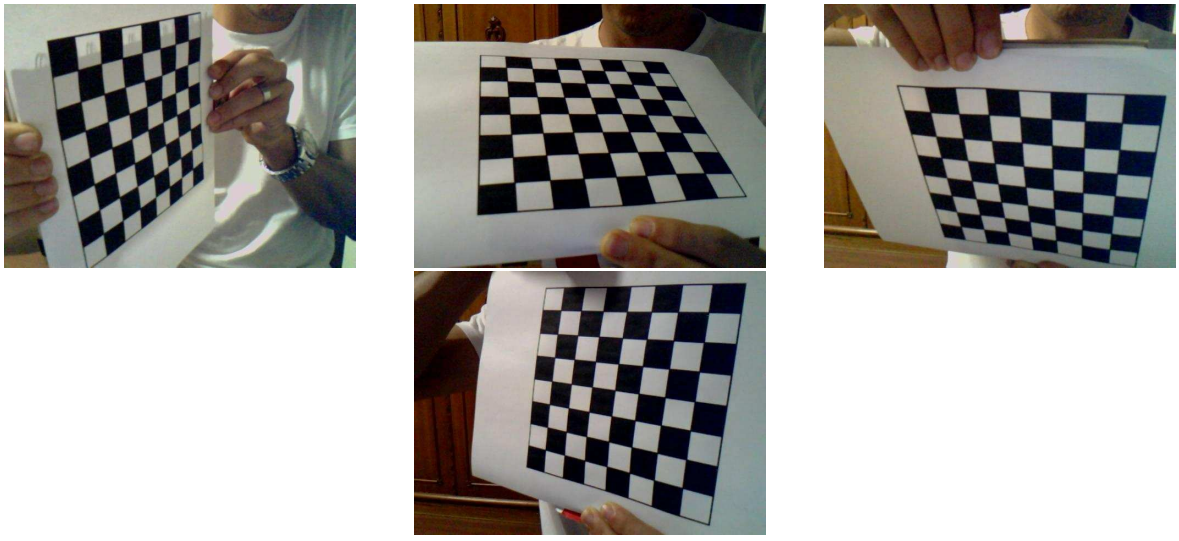


Figura 2.7: Fotos do tabuleiro em diferentes posições.

Primeiro será feito o carregamento de todas as imagens e a partir daí a ferramenta pedirá ao usuário que delimite a região de interesse para cada figura, como mostrado na Figura 2.8.

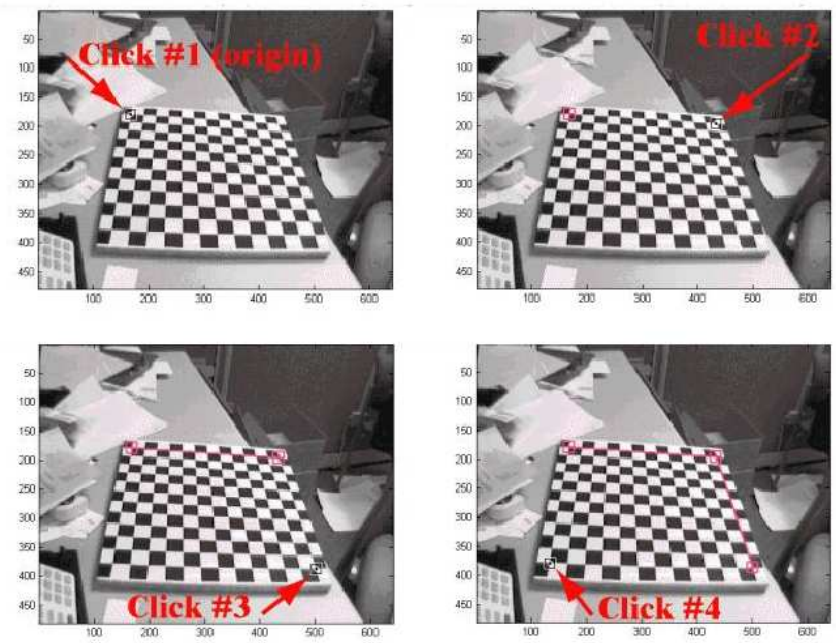


Figura 2.8: Detecção de limites do quadriculado.

O programa apresenta então a predição da grade de quadrados, como mostrado na Figura 2.9.

Figura 2.9: Seleção de limites do tabuleiro

Nesse momento é feito o questionamento se há necessidade de aplicar um fator de distorção da lente para correção da predição. É possível fazer várias tentativas até o melhor ajuste, se necessário. Depois de realizado todo esse processo para cada uma das imagens carregadas, o programa apresenta o resultado final da calibragem.

[1] Figuras 2.8 e 2.9 retiradas de http://www.vision.caltech.edu/bouguetj/calib$_doc/htmls/example.html$

2.5 ESTIMAÇÃO DE POSIÇÃO DO USUÁRIO EM RELAÇÃO À CÂMERA

Para estimar a posição do usuário em relação à câmera, é necessário algumas informações a *priori*. É possível fazer essa estimação utilizando visão estereoscópica, ou seja, utilizando duas câmera, ou utilizando visão monocular, com apenas uma câmera.

Utilizando um modelo simplificado, a câmera considerada não possui distorções no eixo x e y. Nestas condições, temos a seguinte equação matricial:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.9}$$

Onde X,Y e Z são as posições de um ponto qualquer no espaço, x,y e w são os pontos na imagem, f_x e f_y são os focos, em pixels, da imagem e (c_x,c_y) é o pixel central dela. Desenvolvendo a Equação matricial (2.9), temos as equaçôes (2.7) e (2.8) e que $w = Z$.

As equações (2.7) e (2.8) mostram que, para um ponto qualquer (X, Y, Z) no espaço há um correspondente (x,y) na imagem.

Com os olhos do usuário detectados, há a informação de 2 pontos da imagem. Caso a distância dos olhos D_{eyes} seja conhecida, é possível determinar a distância do usuário em relação a câmera com algumas considerações.

Isolando a váriavel Z em (2.7), temos:

$$Z = f_x \frac{X}{(x - c_x)} \tag{2.10}$$

Na Equação 2.10, f_x e c_x são parâmetros de calibração da câmera e x é uma informação que pode ser obtida pelo processamento de imagem.

Neste caso, sabemos a posição dos olhos na imagem em pixels, caso a informação da distância horizontal entre um dos olhos do usuário e a câmera seja conhecida, é possível saber a distância Z do usuário à câmera utilizando a Equação 2.10. Como esta informação não é conhecida, é desejável utilizar parâmetros que sejam conhecidos e que possuam uma pequena variância.

Para poder calcular a distância do usuário até a câmera, é necessário ter em mente o seguinte: desde que o usuário se mantenha à uma distância no eixo Z fixa da câmera e mantenha o rosto na mesma posição, não há variação na distância, em pixel, D_{eyes}. Isto, de fato, pode ser mostrado pela Equação (2.7), calculando as posições dos olhos de uma pessoa em diferentes posições no eixo x, nas condições em que a distância Z é constante e varia-se apenas X, como a distância entre os dois olhos da pessoa, em cm, é sempre constante, logo a distância em pixels, D_{eyes}, também será constante, isto é ilustrado na Figura 2.11.

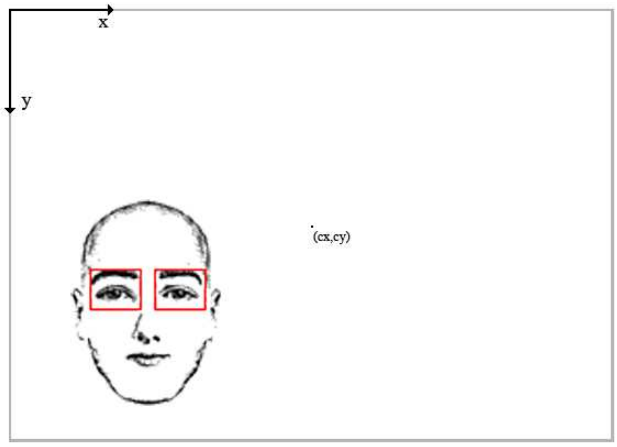


Figura 2.10: Detecção de olhos do usuário.

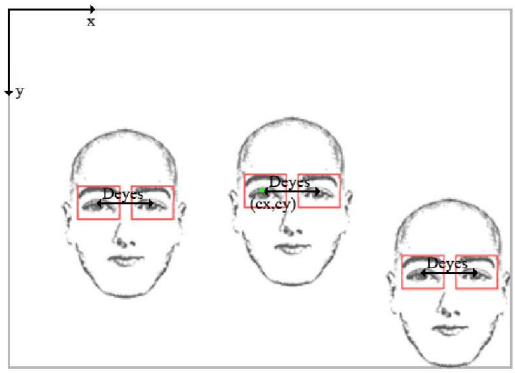


Figura 2.11: A distância D_{eyes} se mantem constante, considerando que a distância Z do usuário à câmera seja também constante

Com essa consideração, independente de qual lugar da imagem em que esteja o rosto do usuário, podemos transladar o rosto e considerar que o centro do olho direito detectado está sobreposto ao ponto central da imagem (c_x,c_y) para facilitar o cálculo. O ponto a ser calculado na Equação 2.10 é o olho esquerdo.

Com isso, temos o seguinte:

$$X_{right} = c_x$$
$$X_{left} = c_x + D_{eyes}$$
$$X_{screen} = X_{left}$$

Onde X_{right} é a posição X do centro do olho direito e X_{left} é a posição X do centro do olho esquerdo. Logo, substituindo na Equação (2.10):

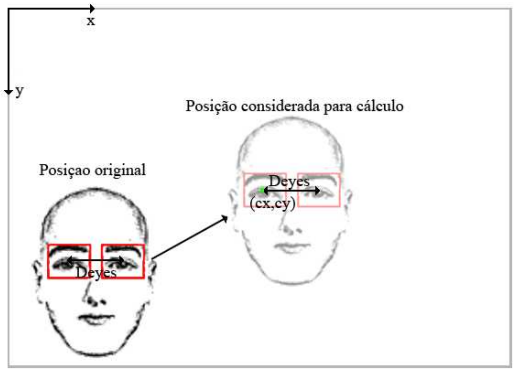


Figura 2.12: Rosto transladado.

$$Z = \frac{f_x X}{c_x + D_{eyes} - c_x}$$

$$Z = \frac{f_x X}{D_{eyes}} \tag{2.11}$$

Como uma média de várias pessoas, a distância entre os olhos de um indivíduo é aproximadamente 6,5 cm, a Equação (2.12) poderá ser reescrita da forma:

$$Z = f_x \frac{0,065}{D_{eyes}} \tag{2.12}$$

Sabendo a distância Z do usuário à câmera, podemos obter a distância X e Y da mesma forma, manipulando as equações (2.7) e (2.8) de forma a isolar as variáveis X e Y. Então temos:

$$X = \frac{(x - c_x)Z}{f_x} \tag{2.13}$$

$$Y = -\frac{(y - c_y)Z}{f_y} \tag{2.14}$$

Assim, com a detecção dos olhos do usuário, é possível estimar a posição do usuário em relação à câmera.

Este cálculo de posição é aproximado. Ele não leva em consideração a rotação da cabeça do usuário. Para uma boa estimação é necessário que o usuário fique com a face de frente para a câmera. Logo, caso o usuário esteja com a cabeça inclinada, virada pra cima, pra baixo ou pros lados, a estimação não será precisa e ocasionará erros. Para os cálculos, foi considerado um caso ideal.

3 DESENVOLVIMENTO

3.1 INTRODUÇÃO

O projeto visa desenvolver um programa que crie uma realidade virtual, visando uma interação visual entre o usuário e o ambiente virtual parecida com a interação visual que uma pessoa possui com o mundo real. No ambiente real, em diferentes posições que uma pessoa esteja, ela terá diferentes perspectivas do mundo ao seu redor. Neste projeto, tenta-se dar a mesma sensação descrita, mas em um ambiente virtual, onde o cenário, mostrado na tela de um computador, mudará de acordo com a posição na qual o usuário estiver.

Com a teoria já apresentada, é possível desenvolver um programa que faça o que foi descrito. Para ter uma boa sensação, é necessário que o programa processe uma quantidade suficiente de quadros por segundo. Tal quantidade não é exatamente determinada, pois a experiência têm caráter subjetivo.

Para a implementação, o programa foi escrito na linguagem C++, utilizando as bibliotecas de apoio OpenCV, OpenGL e newmat. Para testes, foram utilizadas uma câmera de um celular ligada ao PC por cabo USB, uma webcam USB e uma câmera já embutida em um *notebook*. Para o desenvolvimento do projeto, foi considerado que o usuário manterá a face de forma que a câmera a capture frontalmente.

Primeiramente, foi feito a detecção facial e em seguida a detecção dos olhos do usuário. Após isso, foi desenvolvido um método de cálculo para a distância do usuário tendo como referência a câmera, utilizando como informação a posição de cada olho do usuário na imagem obtida pela câmera. Depois, foi implementado o filtro de Kalman para fazer a correção e predição da posição da face na imagem, posição dos olhos na imagem e da distância do usuário até a câmera. Em seguida, foram desenhados três ambientes virtuais 3D que se adaptam a partir da informação da posição da câmera virtual do ambiente, esta posição sendo a distância em X, Y e Z do usuário em relação à câmera real.

O diagrama de blocos do programa pode ser visto na Figura 3.1, onde podemos ver todos os processos feitos pelo programa, estes processos serão explicados no decorrer do capítulo.

3.2 CALIBRAÇÃO DA CÂMERA

Como primeiro passo para o andamento do projeto, foi feita a calibração de câmera, explicada na seção 2.4, para obtermos os parâmetros de foco da câmera que serão utilizados mais a frente no projeto. A calibração foi utilizando por um programa para MATLAB obtido na Internet. A calibração consiste em tirar fotos várias de uma imagem que possui quadrados

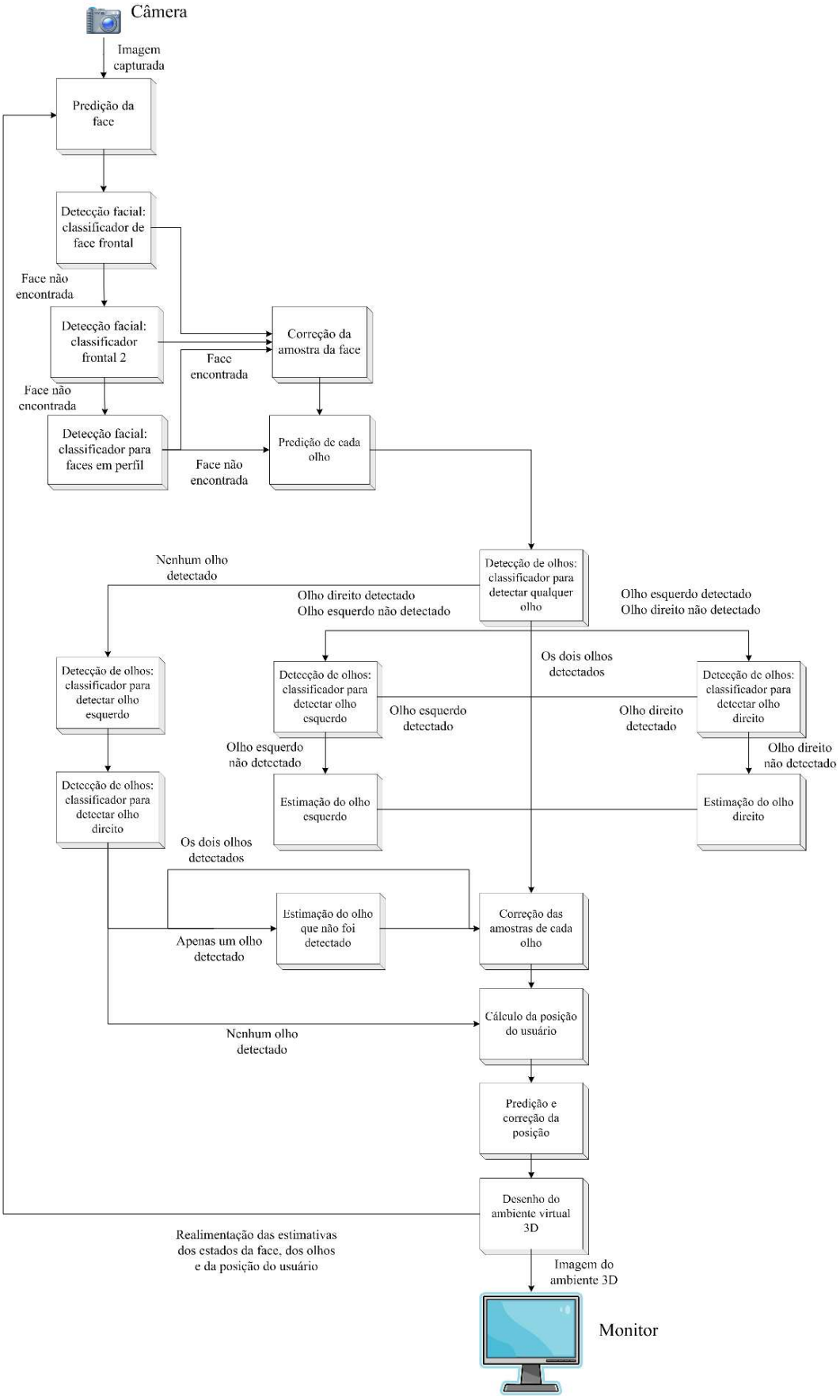


Figura 3.1: Diagrama de blocos do projeto.

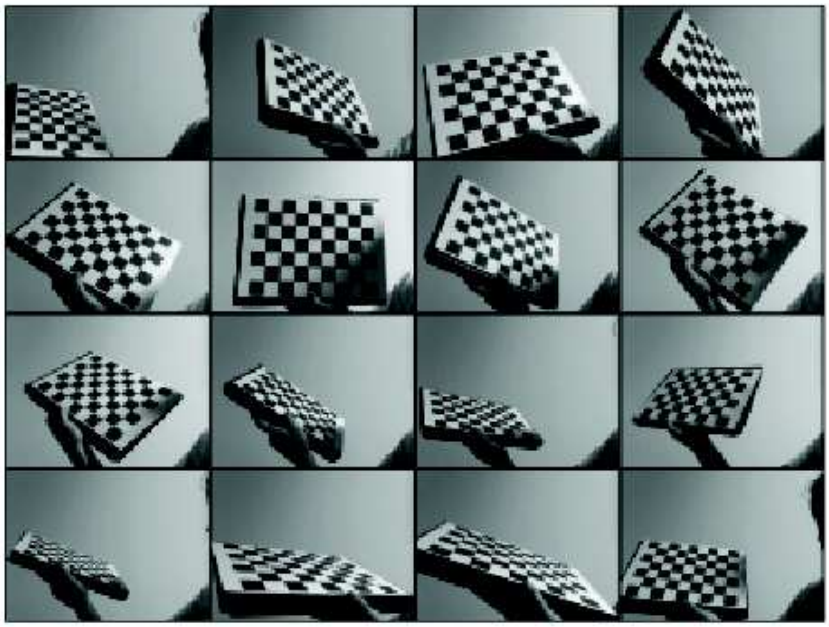


Figura 3.2: Fotos utilizadas para calibração da câmera.

preto e brancos alternados, como um tabuleiro de xadrez, em diferentes posições para fazer a calibração. As imagens utilizadas para a calibração podem ser vistas na Figura 3.2.

O programa necessita da ajuda para achar os extremos do tabuleiro de xadrez, em seguida, ele identifica as quinas dentro desses extremos e se encarrega dos cálculos e retorna os parâmetros de calibração da câmera.

Utilizando uma câmera USB do modelo Microsoft Lifecam que grava vídeos a uma resolução de 640×480 pixels, foi obtido por meio da calibração:

$$f_x = 373$$
$$f_y = 378$$

Onde f_x e f_y são os focos, expressos em unidade de pixel, da câmera, sendo f_x referente ao eixo x e f_y referente ao eixo y. Os focos f_x e f_y variam linearmente de acordo com a resolução da imagem.

Tais parâmetros são utilizados para o cálculo da posição do usuário no espaço, tendo como referência a câmera. Para o projeto não é necessário obter um valor preciso de f_x ou f_y, pois, quando a informação da posição do usuário é convertida para o ambiente virtual, os valores da posição são divididos por um valor arbitrário para se obter o efeito visual desejado, devido a diferença de escala. Então, mesmo utilizando estes focos pertencentes à câmera

mencionada, para os cálculos, pode ser utilizado qualquer outra câmera e considerado os mesmos valores de fx e fy, ainda assim será obtido bons resultado, mesmo que esta possua parâmetros de calibração diferentes.

3.3 OPENCV

OpenCV é uma biblioteca desenvolvida pela Intel em 2000, nas linguagens de programação C/C++, Python e Visual Basic, para usuários de várias plataformas, que permite a criação e desenvolvimento de aplicativos principalmente na área de visão computacional. Identificação de objetos, sistemas de reconhecimento facial e de movimentos, gravação de vídeos, rastreio de robôs, construção de ambientes 3D e de realidade virtual podem ser apontadas como algumas das áreas de aplicação, proporcionando um melhor aproveitamento das oportunidades de utilização da visão computacional nos ambientes em que seja necessário.

Os módulos de processamento de imagem e vídeo I/O, estrutura de dados, álgebra linear, GUI (Interface Gráfica do Usuário) básica com sistema de janelas independentes, controle de mouse e teclado, além de mais de 350 algoritmos de visão computacional como filtros de imagem, calibração de câmera, reconhecimento de objetos e análise estrutural, fazem do OpenCV uma possível solução para problemas de desenvolvimento de softwares direcionados. Outro ponto que pode levar universidades e empresas a começarem a empregar soluções baseadas nessa ferramenta computacional é por este se tratar de um *software* aberto ao uso acadêmico e comercial.

3.3.1 Detecção de objetos utilizando OpenCV

Foi utilizado o método de Viola e Jones [5], apresentado na seção 2.2.1, para a detecção facial e de olhos por ser um método de rápido processamento. A biblioteca OpenCV já possui este método implementado e ele pode ser utilizado por meio da função cvHaarDetectObjects. Esta função possui 5 parâmetros de entrada, como resultado retorna uma sequência de retângulos e cada um destes retângulos representa uma face. Esta função pode ser utilizada para vários tipos de objetos, desde que o classificador seja treinado para tal. Como entrada da função, temos:

image: A imagem na qual será feita a procura.

cascade: O classificador do tipo Haar cascade da forma lida pelo OpenCV.

storage: Espaço alocado na memória para armazenar a sequência de resultados da detecção do objeto, sendo estes resultados em forma de retângulos.

[1] http://opencv.willowgarage.com/wiki/: Site Wiki do OpenCV acessado em novembro 2009

scale factor: O fator pelo qual a janela de procura da função será aumentada para percorrer novamente a imagem.

min neighbors: Faz o agrupamento de retângulos vizinhos, sendo este parâmetro utilizado para mesclar retângulos similares. O valor 0 é utilizado para retornar todos os retângulos encontrados, e acima disso torna a mescla de retângulos maior.

flags: Faz um processamento específico disponibilizado pela biblioteca OpenCV.

min size: Menor janela de procura, valor em pixels, largura e altura da janela.

Como retângulo de saída, possuímos os dados do vértice superior direito do retângulo, da largura e da altura dele. Assim, temos o retângulo definido.

3.4 DETECÇÃO FACIAL

O objetivo de detectar a face do usuário na imagem é definir uma região para procura dos olhos e, além disso, a face é utilizada para saber se as amostras encontradas na procura dos olhos são referentes ao olho direito ou esquerdo.

3.4.1 Detecção facial na imagem

Para a detecção facial, queremos que seja encontrada apenas a face do usuário que utiliza o programa. Como a face será utilizada para ser feita a procura dos olhos, então é mais interessante obtermos um retângulo igual ou maior que a face real do usuário do que um retângulo menor que a face dele. Como parâmetro de entrada para a função, foi utilizado o *scale fator* = 1.3, *min neighbors* = 8, *min size* =(20,20), este limitando o tamanho mínimo da face de 20×20 e a *flag* utilizada foi *CV HAAR FIND BIGGEST OBJECT*, esta flag faz com que a função só retorne o maior retângulo encontrado, assim, o método de escolha da face é feito pelo próprio OpenCV. O parâmetro *min size*(20,20) limita a distância máxima do usuário à câmera que ele pode ser detectado.

Foram utilizados três classificadores para a detecção facial, todos eles acompanham os arquivos do OpenCV. Para a primeira detecção, foi utilizado um classificador treinado para detectar faces frontais, caso este não retorne nenhuma face, é utilizado outro classificador também treinado para faces frontais que, como este foi treinado com imagens diferentes do primeiro classicador, aumenta a chance de detectar uma face e se novamente não for encontrado outra face na imagem, é utilizado um terceiro classificador treinado para faces em perfil. O esquema pode ser visto na Figura 3.3.

Por limitação do método e dos classificadores treinados, há muitos casos em que o rosto não pode ser identificado. Exemplos são os casos em que o usuário não esteja com a cabeça

Figura 3.3: Detecção de face.

disposta frontalmente a câmera, esteja com a cabeça inclinada, com algum objeto estranho na frente do rosto, quando a iluminação ambiente estiver fraca, entre outros casos.

3.4.2 Predição e correção da amostra da face

Obtida a amostra face, é feita a predição e a correção dessa amostra utilizando o filtro de Kalman, apresentado na Seção 2.1. O OpenCV, para processamento interno, define um retângulo utilizando a posição (x,y) do vértice superior esquerdo, largura e altura. Então, estas variáveis farão parte do sistema. Com a informação da velocidade e aceleração destas variáveis, é possível fazer uma predição linear das mesmas.

Para a face, o estado do sistema possui 12 variáveis:

- p_x - Posição x do vértice superior esquerdo do retângulo
- p_y -Posição y do vértice superior esquerdo do retângulo
- $width$ -Largura do retângulo
- $height$ -Altura do retângulo
- v_x -Velocidade da posição x do vértice
- v_y -Velocidade da posição y do vértice
- v_w -Velocidade da variação da largura do retângulo
- v_h -Velocidade da variação da altura do retângulo
- a_x -Aceleração da posição x do vértice
- a_y -Aceleração da posição y do vértice

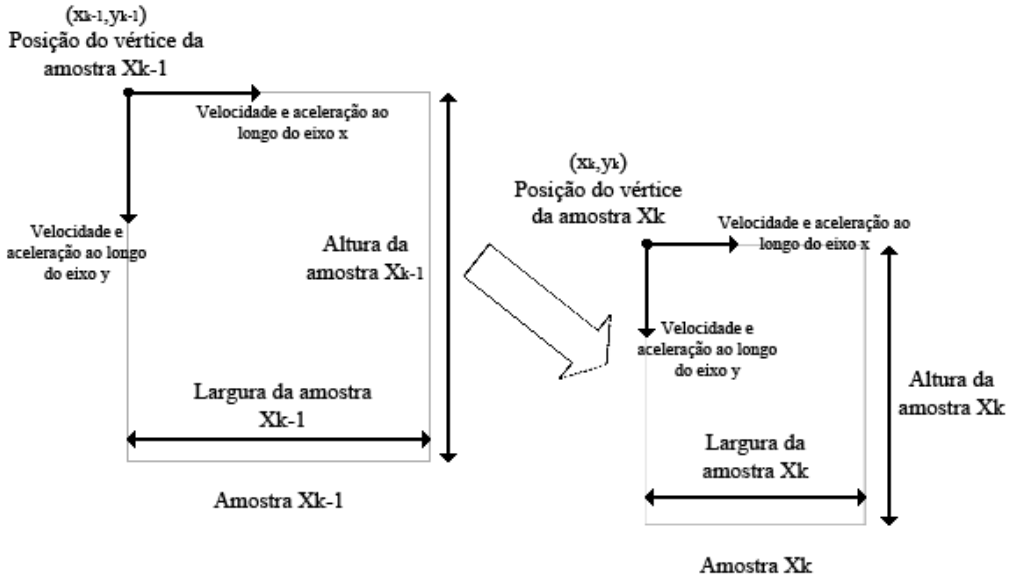


Figura 3.4: Estados da face em uma amostra $\mathbf{x}_{k-1} e \mathbf{x}_k$

- a_w -Aceleração da variação da largura do retângulo
- a_h -Aceleração da variação da altura do retângulo

Todas as variáveis possuem como dimensão pixels. A Figura 3.4 mostra as variáveis mencionadas.

Logo, o vetor de estados $\mathbf{x}_k$ do sistema possui a seguinte forma:

$$\mathbf{x}_k = \begin{bmatrix} p_x & p_y & width & heigth & v_x & v_y & v_w & v_h & a_x & a_y & a_w & a_h \end{bmatrix}^T \tag{3.1}$$

Para o sistema em questão, foi fixado um tempo de amostragem, T, de 40 milisegundos. Logo, uma nova imagem é capturada a cada 40 ms, assim, o programa roda a 25 quadros por segundo (fps). Para o dado sistema, A matriz **A** possui a forma mostrada em (3.2) pela característica do sistema, onde temos a variável, a velocidade dessa variável e a aceleração da mesma. Na multiplicação da matriz **A** pelo vetor de estados, mostrada na equação (2.1), temos o incremento das variáveis de estados pelas equações de movimento uniformement variado, mostradas em (3.4), (3.5) e (3.6). Nestas equações, $\mathbf{x}_k$ e $\mathbf{x}_{k-1}$ correspondem ao vetor de estados da amostra atual e da amostra anterior, respectivamente, $\mathbf{v}_k$ são as velocidades referentes ao vetor de estado $\mathbf{x}_k$, e $\mathbf{a}_k$ são as acelerações referentes também ao vetor de estados $\mathbf{x}_k$.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & T & 0 & 0 & 0 & \frac{T^2}{2} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & T & 0 & 0 & 0 & \frac{T^2}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & T & 0 & 0 & 0 & \frac{T^2}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & T & 0 & 0 & 0 & \frac{T^2}{2} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & T & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & T & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & T & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & T \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.3}$$

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{v}_{k-1}T + \mathbf{a}\frac{T^2}{2} \tag{3.4}$$

$$\mathbf{v}_k = \mathbf{v}_{k-1} + \mathbf{a}_{k-1}T \tag{3.5}$$

$$\mathbf{a}_k = \mathbf{a}_{k-1} \tag{3.6}$$

Já a matriz **H** possui tal forma mostrada na Equação 3.10, pois, a partir das *features* dada pelo OpenCV, é possível retirar as linhas referentes a variáveis de velocidade e aceleração.

A matriz **Q**, matriz de covariância dos ruídos de processo, e a matriz **R**, matriz de covariância de medição, precisam ser inicializadas. Estas matrizes foram ajustadas a partir de um valor inicial considerado plausível, obtido a partir de várias tentativas, sendo que elas possuem esses valores pois são considerados apenas movimentos suaves de face do usuário.

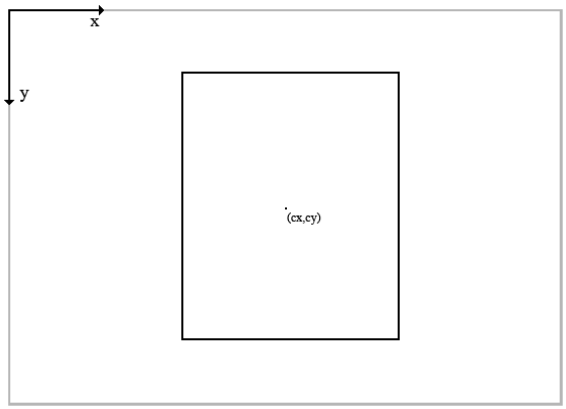


Figura 3.5: Valor inicial das variáveis observáveis do filtro de Kalman.

$$\mathbf{Q} = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 \end{bmatrix} \tag{3.7}$$

$$\mathbf{R} = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 80 \end{bmatrix} \tag{3.8}$$

A estimativa inicial, valor necessário para o filtro, foi arbitrada sendo um retângulo no meio da imagem, de acordo com a Figura 3.5.

Como não é garantido que, ao iniciar a câmera, a face do usuário seja detectada, foi atribuído valores arbitrários para a estimativa inicial. A matriz de covariância do erro, $\mathbf{P}_k$, foi inicializada com valores altos, pois inicialmente não se sabe onde se encontra a face do usuário. Desta maneira, a convergência do filtro para a real face do usuário é rápida, não implicando em prejuízos devido aos valores arbitrados para a estimativa inicial.

Com o sistema definido, é possível fazer a predição de uma amostra utilizando as Equações 2.1 e 2.2, e, após obter-se a amostra, é possível fazer a correção utilizando as equações 2.3, 2.4 e 2.5.

Caso a face seja detectada, os valores corrigidos da posição (x,y) do vértice já mencionado do retângulo, largura e altura, serão definidos como a face para ser utilizada na próxima etapa do programa.

Para os valores do filtro, temos valores reais, mas é necessário utilizar valores inteiros pois a imagem só possui valores inteiros, sendo estes os pixels. Essa adaptação é feita convertendo uma variável em C++ do tipo *double* para uma variável do tipo *int*, ou seja, desconsidera a parte fracionária e só considera a parte inteira.

Para a detecção facial completa, temos o algoritmo mostrado na tabela de algoritmo 3.1.

3.4.3 Simulação

Para compreender melhor o funcionamento da detecção facial, foi feita uma simulação em MATLAB.

Na simulação, foram criados dois eixos, x e y, que representam a imagem, tendo o eixo x o valor limite de 640 pixels e o eixo y de 480 pixels, simulando uma imagem de 640x480 pixels. Foi criado um retângulo que representa a face do usuário, este foi inicializado no meio da imagem e ele se move em linhas retas nas direções x e y e, quando chega a algum limite da imagem, é calculado uma nova direção para sua trajetória. Assim, o retângulo que

Algoritmo 3.1 Algoritmo de detecção de face

1: **Captura da imagem da câmera**
2: **Inicialização de variável de controle**
   número de faces = 0
3: **Procura por faces com primeiro classificador**
   Se *encontrar*, **então**
      número de faces = 1
4: **Se** *número de faces = 0*, **então**
      Procura por faces com segundo classificador
      Se *encontrar*, **então**
         número de faces = 1
5: **Se** *número de faces = 0*, **então**
      Procura por faces com terceiro classificador
      Se *encontrar*, **então**
         número de faces = 1
6: **Predição da face utilizando as equações 2.1 e 2.2**:
7: **Correção da face utilizando as equações 2.3, 2.4 e 2.5** :

é referente a face usuário se move pela imagem.

A face detectada foi simulada somando valores aleatórios, que representam um ruído gaussiano, a posição (x,y) da face, largura e altura, assim, mesmo se a face se mantiver imóvel, a detecção retornará diferentes retângulos a cada novo quadro assim como o nosso sistema real.

A predição e correção foram feitas utilizando filtro de Kalman de primeira ordem, considerando movimento uniforme, diferentemente do implementado no programa. Sendo assim o estado do sistema possui 8 variáveis:

- Posição x do vértice superior esquerdo do retângulo
- Posição y do vértice superior esquerdo do retângulo
- Largura do retângulo
- Altura do retângulo
- Velocidade da posição x do vértice
- Velocidade da posição y do vértice
- Velocidade da variação da largura do retângulo
- Velocidade da variação da altura do retângulo

Na prática, foi implementado um filtro de Kalman que contêm as variáveis de aceleração, mas a simulação não contém tais variáveis.

Para a simulação, o filtro de Kalman possui as seguintes matrizes:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & T & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & T & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & T & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & T \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.10}$$

A matriz **Q**, matriz de covariância dos ruídos de processo, e a matriz **R**, matriz de covariância de medição, foram inicializadas com o seguintes valores:

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 33.33 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 33.33 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.11}$$

$$\mathbf{R} = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 400 \end{bmatrix} \tag{3.12}$$

Estas matrizes foram ajustadas com um certo valor incial definido a partir de várias tentativas.

3.5 DETECÇÃO DOS OLHOS

Na detecção de olhos, é necessário fazer alguns refinamentos pois, para termos um bom rastreio do olho, temos que saber qual é o olho direito e qual é o esquerdo. Além disso, é interessante limitar a região de procura dos olhos para diminuir a quantidade de processamento e evitar a detecção de objetos parecidos com um olho que estejam fora de uma face, como já temos a informação da face do usuário, isto é possível. Após feita a predição dos parâmetros olhos, e caso os olhos sejam detectados, é feita a correção destes dados.

Os cálculos de predição e correção são feitos para cada olho, um independente do outro, já que pode haver uma rotação da face do usuário. Algumas restrições *ad-hoc* também foram implementadas.

3.5.1 Detecção dos olhos na imagem

Antes de ser feita a detecção dos olhos, é definida uma região de interesse na imagem para ser feita a procura dos olhos. O OpenCV possui uma função que determina essa região de interesse, assim, os olhos não serão procurados em toda a imagem, diferentemente da face. Para um retângulo encontrado que contém uma face, vamos determinar uma certa

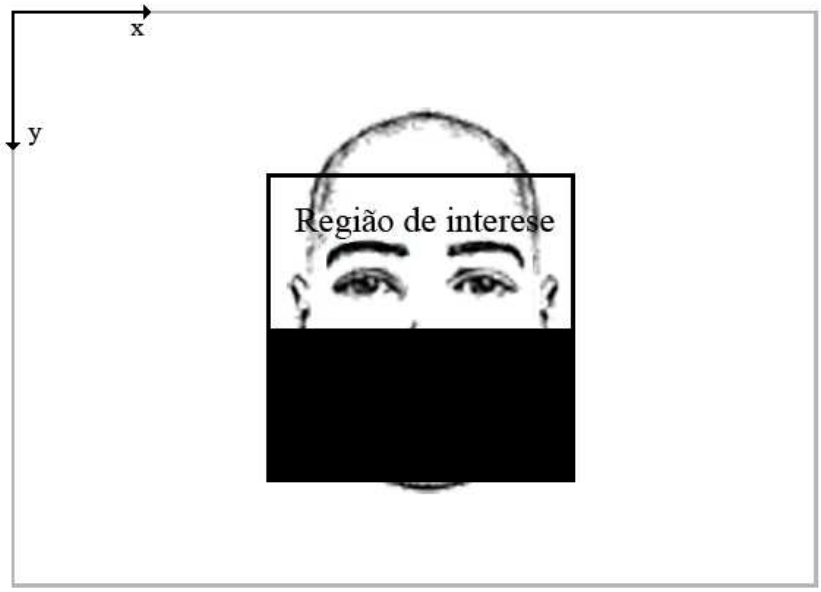


Figura 3.6: Região de interesse para detecção dos olhos

região dentro dela onde será feita a procura. Como o retângulo encontrado varia de tamanho, mesmo que as imagens sejam parecidas, é preciso escolher uma região um pouco maior que os olhos. Esta região também é um retângulo, sua largura é a largura da face e sua altura é a altura da face dividida por 1.8, compreendendo assim só a parte superior da face. O vértice superior direito da região está localizado no mesmo ponto que o vértice superior direito da face. Podemos ver um exemplo da região de interesse de procura dos olhos na Figura 3.6.

Após a definição da região de interesse, é feita a procura por olhos. Esta procura só acontecerá caso a face seja encontrada ou, caso ela não tenha sido encontrada, a predição dos parâmetros da face estejam dentro dos limites da imagem, já que a predição pode estimar que a face esteja fora dos limites mencionados. Os parâmetros utilizados na função de procura por olhos foi escolhido de forma a retornar todos os olhos encontrados, pois em seguida será feito a separação entre olhos esquerdos e direitos além de ser eliminados dados considerados improváveis de serem olhos. Foi utilizado o *scale fator* = 1.3, *min neighbors* = 0, *min size* =(15,15) e não foi utilizada nenhuma *flag*, de forma a se obter várias amostras de olhos. Para a primeira procura por olhos, será utilizado o classificador treinado para olhos direitos e esquerdos que acompanha o OpenCV. Para a segunda procura, são utilizados dois classificadores baixados na internet[1], um treinado para detectar olhos direitos e o outro treinado para detectar olhos esquerdos.

Após a procura por olhos, é feita a separação entre olho direito e olho esquerdo. Uma amostra é considerada como olho esquerdo se o centro do retângulo desta amostra for maior

[1] Classificadores treinados por Shiqi Yu, site: http://yushiqi.cn/research/eyedetection

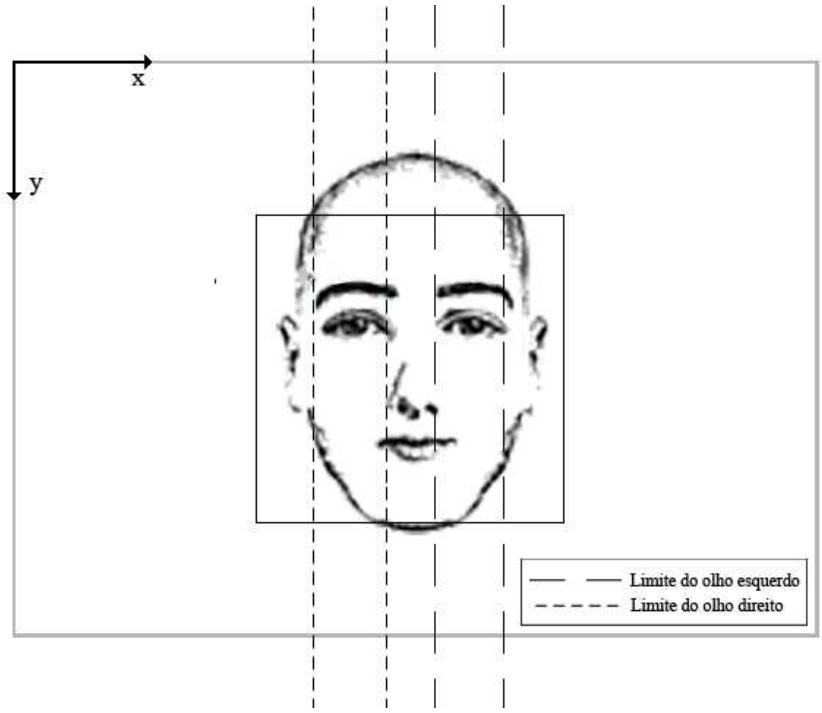


Figura 3.7: Limites para saber de qual olho a amostra se refere

que a largura da face divida por 1.7 e ao mesmo tempo for menor que a largura da face dividida por 0.8. Da mesma maneira, uma amostra é considerada como olho direito caso o centro do retângulo dela seja maior que a largura da face dividida por 5 e também seja menor que a largura da face dividida por 2.6. Os valores que estivem fora dessas regiões são descartados. Na Figura 3.7, podemos visualizar essas condições.

Tendo separado todas as amostras, o olho direito será a média de todas as amostras de olhos direitos e o olho esquerdo será a média de todas as amostras de olhos esquerdos. Isto pode ser considerado pois as amostras da detecção pelo método Viola-Jones são descorrelacionadas e a matriz de covariância de cada amostra é igual.

Assim, depois da primeira procura por olhos com um dado classificador, é feita a separação das amostras. Caso não seja encontrado algum olho esquerdo, será feita uma nova procura, mas agora com um classificador treinado especificamente para olhos esquerdos. Da mesma maneira, caso não seja encontrado algum olho direito, será feita a procura com um outro classificador treinado para olhos direitos. Após estas procuras, será feita novamente a separação de amostras e o cálculo das amostras a serem consideradas.

Após estes procedimentos, pode haver 4 casos: os dois olhos são encontrados, apenas o olho esquerdo é encontrado, apenas o olho direito é encontrado ou nenhum dos olhos são encontrados. O caso ideal é quando os dois olhos são encontrados.

Caso apenas um dos olhos seja encontrado, temos a situação em que a face foi encontrada e um olho também, assim, é provável que houve uma variação em um dos olhos onde não foi possível detectá-lo. Causas podem ser problemas na iluminação, *blur* na imagem devido

a movimento, cabelo na frente do olho, entre outras possibilidades que acontecem frequentemente. Nestas situações, será utilizada uma solução *ad-hoc*, será estimada uma posição para o olho e esta será considerada como uma amostra. Ela é feita da seguinte forma: caso o olho esquerdo seja encontrado e o olho direito não, a amostra deste será a amostra do olho esquerdo transladada no eixo x da imagem. Para o translado, será utilizado o valor predito da distância entre os olhos do usuário em pixels. A Figura 3.1 mostra que o programa possui uma predição e estimação de posição do usuário. Se for utilizada a predição da posição Z do usuário tendo como referência a câmera, é possível calcular a distância entre os olhos do usuário, em pixels, isolando a variável D_{eyes} na equação (2.12). Então, temos:

$$D_{eyes} = \frac{f_x 0,065}{Z} \tag{3.13}$$

Então, a variável D_{eyes} é a distância predita, em pixels, entre os olhos do usuário.

Assim, no caso em que o olho esquerdo é encontrado e o olho direito não, a amostra considerada deste será a amostra do olho esquerdo com sua posição no eixo x subtraída de D_{eyes}. Da mesma maneira, no caso em que o olho direito é encontrado e o olho esquerdo não, a amostra considerada para este será a amostra do olho direito com sua posição no eixo x somada a D_{eyes}.

Com estes procedimentos, é possível ter uma boa detecção de olhos, que elimina as amostras absurdas, as separam entre olho direito e esquerdo e continuam estimando a posição do olho mesmo se apenas um olho for encontrado.

3.5.2 Predição e correção das amostras dos olhos

Como as amostras dos olhos são obtidas na forma de retângulos assim como a amostra da face, o procedimento de predição e correção é parecido com aquele.

A estimação dos olhos é feita separadamente, ou seja, é feita uma estimação para o olho direito e outra para o olho esquerdo.

Para cada olho, o estado do sistema possui 12 variáveis:

- p_x - Posição x do vértice superior esquerdo do retângulo
- p_y -Posição y do vértice superior esquerdo do retângulo
- $width$ -Largura do retângulo
- $height$ -Altura do retângulo
- v_x -Velocidade da posição x do vértice
- v_y -Velocidade da posição y do vértice

- v_w -Velocidade da variação da largura do retângulo
- v_h -Velocidade da variação da altura do retângulo
- a_x -Aceleração da posição x do vértice
- a_y -Aceleração da posição y do vértice
- a_w -Aceleração da variação da largura do retângulo
- a_h -Aceleração da variação da altura do retângulo

As variáveis do retângulo possuem como dimensão pixels, as de velocidade possuem pixels por segundo e as de aceleração possuem pixels por segundo ao quadrado. Na Figura 3.4 pode ser visto as variáveis do estado do sistema.

Como já mencionado anterior, o tempo de amostragem, T, do sistema é de 40 milisegundos.

Para o dado sistema, o vetor de estados pode ser visto em (3.14), a matriz **A** em (3.2) e a matriz **H** em (3.10)

A matriz **A** possui tal forma pela característica do sistema, onde temos a variável, a velocidade dessa variável e a aceleração dela, como já mencionado na seção de predição da face, e igualmente, a matriz **H** possui tal forma, pois, a partir das *features* dada pelo OpenCV, é possível retirar as linhas referentes a variáveis de velocidade e aceleração.

A matriz **Q**, matriz de covariância dos ruídos de processo, e a matriz **R**, matriz de covariância de medição, foram inicializadas com valores iniciais plausíveis, obtidos a partir de várias tentativas. A matriz **Q** pode ser vista em (3.11) e a matriz **R** em (3.12), foram utilizados os mesmos valores que os utilizados na detecção de face.

A estimativa inicial, valor necessário para o filtro, de cada olho foi arbitrada de forma a corresponder ao olho direito ou esquerdo da estimativa inicial da face arbitrada na seção 3.4.2, sendo que a forma com que foi arbitrada pode ser vista na Figura 3.8.

Como também não é garantido que, ao iniciar a câmera, os olhos do usuário sejam detectados, foram atribuídos valores arbitrários para seus valores iniciais e a matriz de covariância do erro, $\mathbf{P}_k$, foi inicializada com valores altos, obtendo-se uma rápida convergência.

Com o sistema definido, é possível fazer a predição de uma amostra utilizando as equações (2.1) e (2.2), e, após obter-se a amostra, é possível fazer a correção utilizando as equações (2.3), (2.4) e (2.5).

Da mesma forma da amostra da face, as amostras dos olhos precisam ser convertidas de valores reais para valores inteiros, e isto é feito desconsiderando a parte decimal do número.

Para a detecção de olhos, temos o algoritmo mostrado na tabela de algoritmo 3.2.

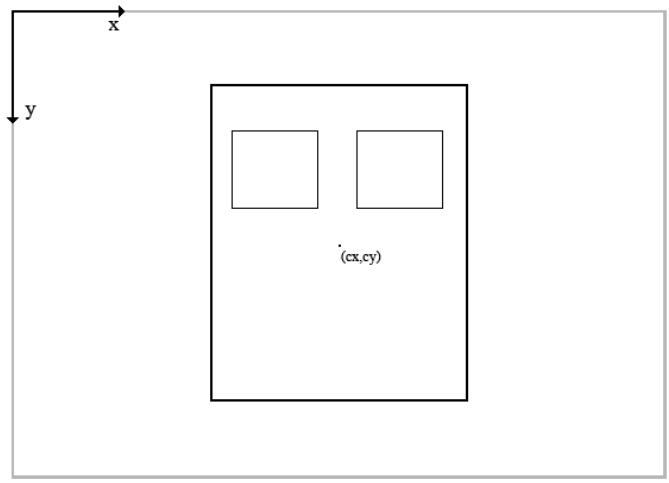


Figura 3.8: Estimativas iniciais dos olhos

Algoritmo 3.2 Algoritmo de detecção de olhos

1: **Define a região de interesse na parte superior da imagem**
2: **Procurar por olhos**
3: **Separar as amostras e calcular a amostra a ser considerada para cada olho**
4: **Se** *o olho esquerdo não for encontrado*, **então**
   Procurar pelo olho esquerdo com o classificador específico
   Se *encontrar*, **então**
      calcular amostra a ser considerada
5: **Se** *o olho direito não for encontrado*, **então**
   Procurar pelo olho esquerdo com o classificador específico
   Se *encontrar*, **então**
      calcular amostra a ser considerada
6: **Se** *detectou olho direito e não detectou olho esquerdo*, **então**
   Igualar a amostra do olho esquerdo à amostra do direito com sua posição somada de D_{eyes}
7: **Se** *detectou olho esquerdo e não detectou olho direito*, **então**
   Igualar a amostra do olho direito à amostra do esquerdo com sua posição subtraída de D_{eyes}
8: **Calcular predição de cada olho utilizando as equações 2.1 e 2.2**:
9: **Calcular correção de cada olho utilizando as equações 2.3, 2.4 e 2.5** :

3.6 ESTIMAÇÃO DE POSIÇÃO

A posição do usuário é utilizada para mover a câmera no ambiente virtual 3D, assim, é necessário achar uma relação entre um ponto na imagem e um ponto no espaço. O ponto que

será considerado para posição do usuário é o centro de seus dois olhos, pois trata-se de uma experiência visual.

3.6.1 Cálculo da posição do usuário

Para ser feito o cálculo da posição, é necessário obter algumas informações da imagem detectada. Neste passo, os olhos do usuário já foram detectados, assim, facilmente conseguimos saber a posição do centro de cada olho, a distância, em pixels, entre os dois olhos e a posição do centro dos dois olhos.

Como centro de referência para os valores mencionados, será utilizado o centro da imagem, pois ele corresponde a um ponto no espaço que esteja de frente a câmera.

A distância entre os dois olhos D_{eyes} considerada é a distância entre eles no eixo x da imagem. Então, é desconsiderada a distância entre os dois no eixo y da imagem, pois, para simplificar o sistema, como já mencionado, é considerado que o usuário fique com sua face de maneira que a câmera a capture frontalmente. Com esta informação, é possível calcular a posição Z do usuário como referência a câmera utilizando a equação (2.12).

Para a posição X e Y do usuário utilizando a mesma referência, o ponto da imagem que corresponderá ao usuário será o centro dos dois olhos. Este ponto pode ser calculado sendo a média simples da posição (x,y) do olho direito e do olho esquerdo. Tendo calculado tal ponto, utilizando as equações (2.14) e (2.14), é possível calcular a posição X e Y real do usuário.

No ambiente virtual, queremos que a pessoa olhe a tela e não a câmera. A posição Y que se obtêm com esses cálculos tem como referência a câmera, então, será somado a essa posição a distância entre a tela do monitor e a câmera, assim nossa referência passa a ser o centro da tela do monitor. Está sendo considerado que o centro da tela do monitor e a câmera estão alinhados verticalmente, caso contrário, é necessário somar a posição X do usuário a distância horizontal entre tal centro e a câmera.

3.6.2 Predição e estimação da posição

Tendo como referência de coordenadas a câmera, no espaço, o usuário possui posição (X, Y, Z). Desta forma, é interessante usar estas três variáveis para serem estimadas. Como o modelo utilizado para a estimação da face e dos olhos, será utilizado um sistema que tenha como variáveis de estado os valores da posição do usuário em relação à câmera, a velocidade de cada uma das componentes da posição e suas acelerações.

Assim, o sistema terá 9 variáveis:

- *X* - Posição x do usuário

- *Y* - Posição y do usuário
- *Z* - Posição z do usuário
- v_X - Velocidade da posição x
- v_Y - Velocidade da posição y
- v_Z - Velocidade da variação z
- a_X - Aceleração da posição x
- a_Y - Aceleração da posição y
- a_Z - Aceleração da variação z

As variáveis de posição possuem como dimensão cm, e as variáveis de velocidade possuem como dimensão cm/s e da aceleração possuem cm/s^2.

Como vetor de estados **x'**$_k$, temos:

$$\mathbf{x'}_k = \begin{bmatrix} p_X & p_Y & p_Z & v_X & v_Y & v_Z & a_X & a_Y & a_Z \end{bmatrix}' \tag{3.14}$$

Como já mencionado anteriormente, o tempo de amostragem, T, do sistema é de 40 milisegundos.

Para o dado sistema, temos as seguintes matrizes:

$$\mathbf{A'} = \begin{bmatrix} 1 & 0 & 0 & T & 0 & 0 & \frac{T^2}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 & 0 & \frac{T^2}{2} & 0 \\ 0 & 0 & 1 & 0 & 0 & T & 0 & 0 & \frac{T^2}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 & T & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

$$\mathbf{H'} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.16}$$

A matriz **A'** possui tal forma pela característica do sistema em questão, que possui as variáveis de posição, velocidade e aceleração. Já a matriz **H'** possui tal forma, pois, devido as variáveis utilizadas pelo programa, as linhas referentes a velocidade e aceleração podem ser eliminadas

A matriz **Q'**, matriz de covariância dos ruídos de processo, e a matriz **R'**, matriz de covariância de medição, foram inicializadas com valores iniciais plausíveis, definidos a partir de várias tentativas.

$$\mathbf{Q'} = \begin{bmatrix} 0.01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.004 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 \end{bmatrix} \tag{3.17}$$

$$\mathbf{R'} = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 20 \end{bmatrix} \tag{3.18}$$

O valor inicial do estado utilizado para o filtro foi obtido pelo processo do programa até a parte de predição da posição.

Com o sistema definido, é possível fazer a predição de uma amostra utilizando as equações (2.1) e (2.2), e, após obter-se a amostra, é possível fazer a correção utilizando as equações (2.3), (2.4) e (2.5).

Feita a correção da posição, têm-se o valor desejado que será utilizado no ambiente virtual.

3.7 DESENHO DO AMBIENTE VIRTUAL 3D

Para o ambiente virtual, foram desenhados vários objetos 3D, utilizando a biblioteca gráfica OpenGL, e foi colocada uma câmera virtual dentro do ambiente. Esta câmera virtual é a perspectiva na qual será desenhada os objetos, assim, caso a câmera mova, toda a perspectiva do ambiente mudará. A câmera virtual se move de acordo com a posição do usuário, que foi considerada sendo a posição do centro dos olhos do usuário tendo como referência a câmera, e essa posição foi dividida por um valor de escala.

Se o usuário estiver com o centro dos olhos de frente ao centro da tela do monitor e ele se mover para a direita, então a câmera virtual também se moverá para a direita. No caso em que houvesse um cubo 3D na tela, quando o usuário se movesse para a direita, ele conseguiria ver o lado do cubo.

Como foram calculadas a posição (X, Y, Z) no espaço do usuário, é repassada à câmera virtual estas três posições, logo, é possível que o ponto de vista se mova verticalmente, horizontalmente, se aproxime ou se afaste dos objetos do cenário.

Foram desenhados três ambientes para que se conseguisse passar da melhor forma a experiência visual do projeto.

3.7.1 OpenGL

Para a parte de renderização 3D foi utilizado a API (Application Programming Interface) gráfica OpenGL (Open Graphics Library), utilizada na computação gráfica para desenvolvimento de aplicativos gráficos, ambientes virtuais 3D, jogos 2D ou 3D, entre várias outras aplicações.

O OpenGL possui centenas de funções que fornecem acesso a praticamente todos os recursos do *hardware* de vídeo e assim esconde a complexidade de geração de gráficos 2D e 3D por meio de uma biblioteca uniforme.

3.7.2 Cubo 3D

Neste ambiente, foi desenhado um cubo 3D e cada um de seus lados possue uma cor diferente. A câmera foi colocada de forma que uma das arestas do cubo fique de frente para a câmera, ou seja, se o usuário estiver olhando de frente para o monitor, ele verá dois lados do cubo e a aresta mencionada estará no centro da tela. A Figura 3.9 mostra uma imagem do cubo.

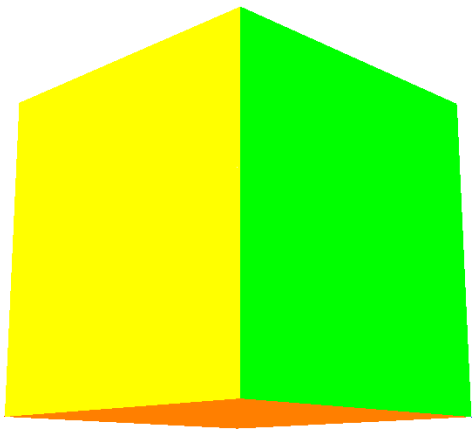


Figura 3.9: Cubo 3D, cada lado do cubo possui uma cor

3.7.3 Geometrias

Para este ambiente, foram desenhados vários tipos de objetos 3D. A câmera foi colocada atrás de um cubo, para que o usuário consiga avistar o resto dos objetos, ele precisa se mover

de forma a sair de trás do cubo. A Figura 3.10 mostra o ambiente completo.

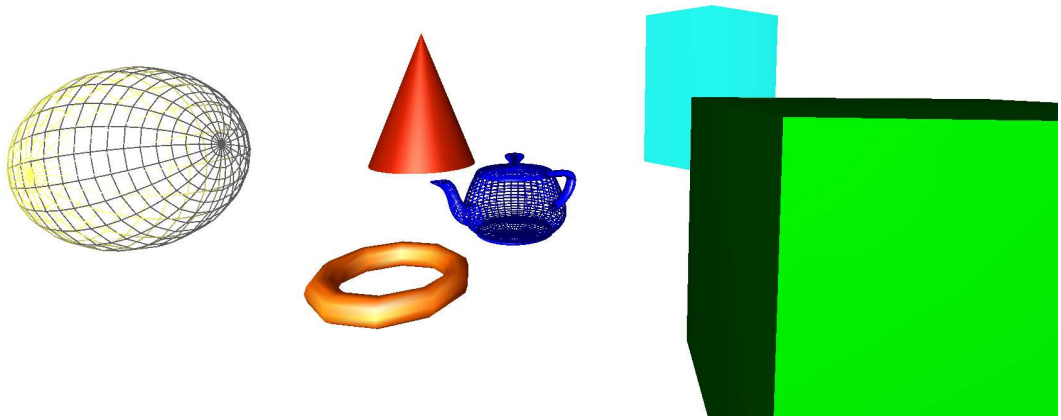


Figura 3.10: Ambiente virtual com várias geometrias espalhadas pelo cenário

3.7.4 Alvos

Este cenário foi baseado no trabalho Johnny Chung Lee. No ambiente, foi desenhado vários alvos saindo do fundo da tela até perto da câmera virtual, cada alvo em uma distância diferente da câmera virtual. Este é o ambiente que proporciona a melhor experiência ao usuário, e ele pode ser visto na Figura 3.11.

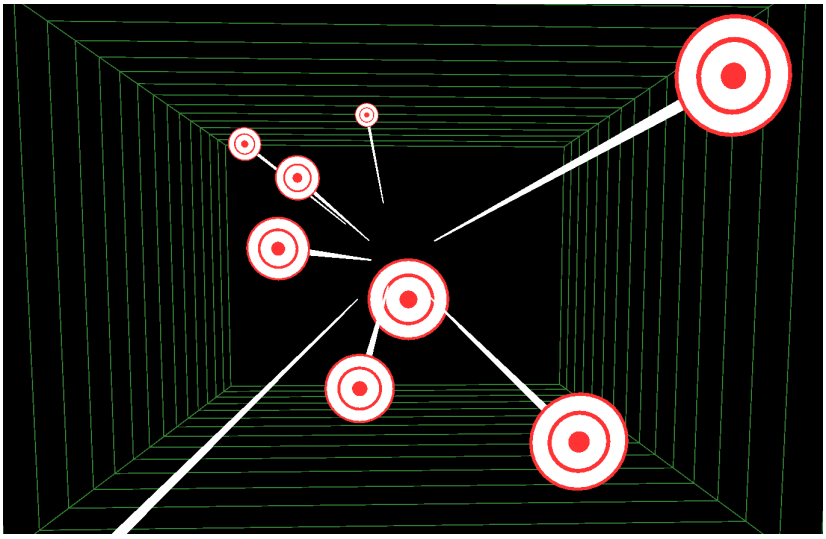


Figura 3.11: Ambiente virtual, alvos espalhados pelo cenário em diferentes profundidades

4 RESULTADOS

O projeto pode ser dividido em quatro etapas: detecção facial, detecção de olhos, estimação de posição e renderização do ambiente virtual. Estas etapas podem ser vistas na Figura 4.1.

Como resultado geral do projeto, se quer uma interativadade entre o usuário e o ambiente virtual que seja agradável, tendo o resultado um caráter subjetivo. Os parâmetros de projeto foram decididos de forma a se obter uma melhor experiência. Como o projeto consiste na interação entre um objeto virtual e um usuário, é interessante tentar alcançar um processamento em tempo real dos dados para dar uma experiência agradável. Caso haja um tempo grande para o processamento, a freqüência com que a imagem do ambiente é desenhada será muito reduzida e o usuário verá quadro a quadro o movimento do objeto, em vez de ver um movimento fluído do objeto quando ele se move.

O programa consegue rodar a 30 quadros por segundo, em média, caso não seja definido um tempo de execução para o programa. Como é necessário uma taxa de amostragem constante para os cálculos do filtro de Kalman, esta taxa é de 40 milisegundos. Logo, o programa roda a 25 quadros por segundo, utilizando uma imagem de vídeo capturada pela câmera de resolução 640 x 480 pixels.

Este capítulo apresentará os resultados obtidos em cada etapa do projeto mencionada. Como a coleta da imagem para processamento é feita por meio de uma câmera USB, as condições de coleta de dados são:

- Iluminação ambiente controlada
- Iluminação direta na face do usuário, eliminando sombras na face e nos olhos
- Plano de fundo branco e uniforme

Nestas condições, podemos observar os resultados do projeto da melhor forma, já que com iluminação controlada e ausência de objetos ao fundo da imagem, a detecção de face e olhos é melhor sucedida.

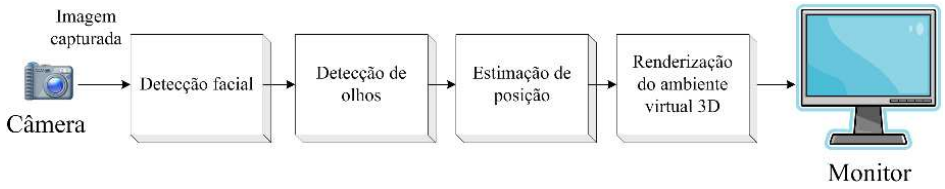


Figura 4.1: Etapas do projeto.

4.1 SIMULAÇÃO DA DETECÇÃO DE FACE

Na simulação, foram capturas imagens da face, detecção da face e correção feita pelo filtro de Kalman, e foi levantada uma tabela com o erro de estimação.

A Figura 4.2 apresenta quadros em que a face, representada pelo retângulo preto, está se movimentando e a cada quadro há uma nova face detectada, representada pelo retângulo vermelho, e o retângulo resultante da estimativa feita pelo filtro de Kalman é representado pelo retângulo azul.

A tabela 4.1 mostra o erro de estimação em pixels, variável real subtraída da variável estimada pelo conjunto detecção facial e filtro de Kalman. Como a detecção retorna retângulos que contêm faces, as variáveis de estado que foram obtidas o erro de estimação foram: posição x, posição y, largura e altura do retângulo. Todas essas variáveis referentes as imagens da Figura 4.2.

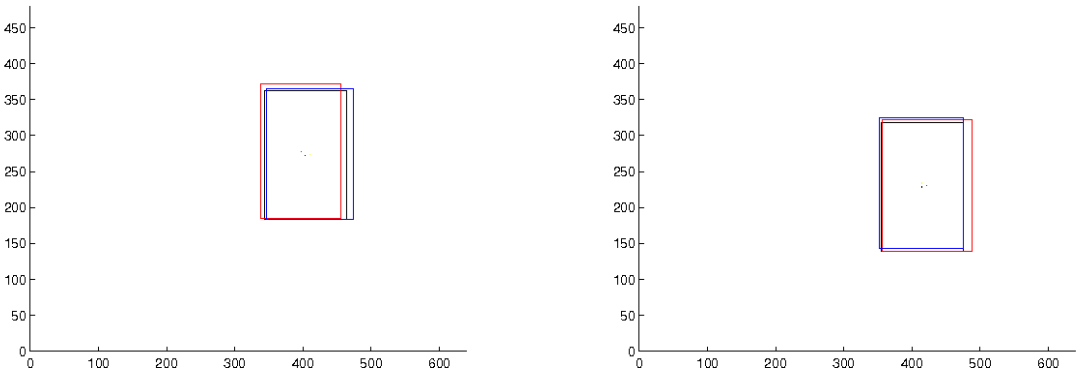
Tabela 4.1: Erro de estimação da simulação

Figura 4.2	Posição x	Posição y	Largura	Altura
(a)	-6,3	-1,2	-7,2	-2
(b)	0,8	-5,3	-,3	-1
(c)	-1,7	-1,8	-5	-2
(d)	15,1	17	-2	2
(e)	-10,5	7,7	3	0
(f)	-2,9	3,9	-1	-1

4.2 DETECÇÃO FACIAL

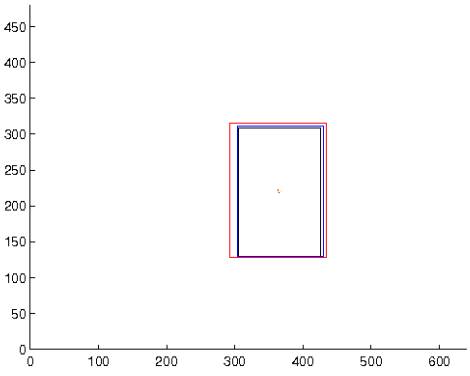
De forma a aumentar a chance de detectar uma face na imagem, foram utilizados três classificadores, como mencionado na Seção 3.4.1. Como espera-se que só uma pessoa estará utilizando o programa, não é preciso ter uma grande preocupação com os resultados falso-positivos dos classificadores. Considerando a face o maior retângulo resultante da detecção de face, a maior parte dos falsos-positivos são eliminados. Como limitação do método Viola-Jones, em ambientes de baixa iluminação a detecção é prejudicada, assim, a iluminação ambiente foi controlada.

O filtro de Kalman, além de fazer a predição para os casos em que a face não for encontrada e a correção para as amostras, também serve com um filtro passa-baixa, suavindo as pequenas oscilações, em alta freqüência, das váriaveis de estado do sistema. A detecção para vários quadros diferentes, com o usuário imóvel, retorna resultados diferentes para a face, como pode ser visto na Figura 4.3. Observar tal variação é difícil em duas imagens estáticas, mas com o programa funcionamento, é imediata a percepção de tal variação. Essa variação

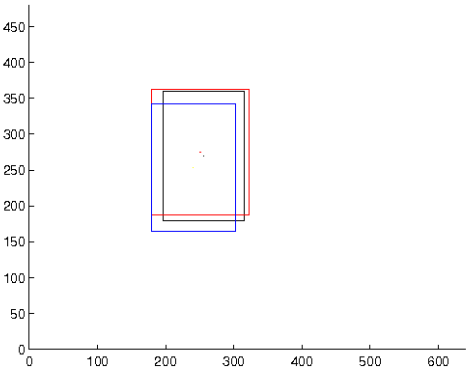


(a)

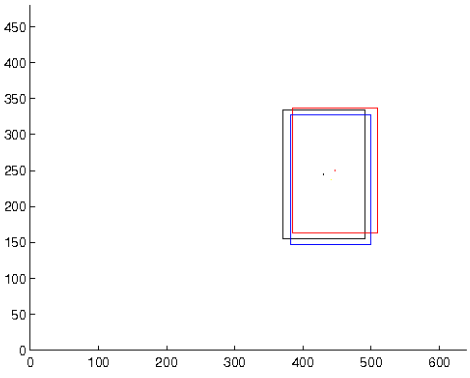
(b)



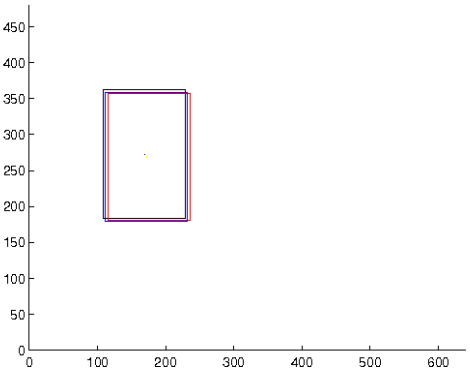
(c)



(d)



(e)



(f)

Figura 4.2: Simulação, sendo o retângulo preto a face do usuário, o retângulo vermelho a face detectada e o retângulo azul a correção feita pelo filtro de Kalman.

no resultado, para a detecção facial, não traz problemas para o resultado final.

Figura 4.3: Variação da detecção para diferentes quadros, nos quais a face se mantém imóvel

Deseja-se um rastreio rápido para a face, mas também é desejado que a estimativa da face não varie bruscamente e isto não ocorre com o resultado dados pelos classificadores. O filtro de Kalman para a face, apresentado na Seção 3.4.2, foi projetado de tal maneira que se tenha o melhor custo benefício entre rapidez e suavidade para o rastreio, e esta é uma decisão subjetiva. Caso a resposta do filtro seja muito suave, no resultado final do programa, no ambiente 3D, a resposta ao movimento do usuário será lenta. Caso a resposta do filtro seja rápida e pouco suave, o usuário mantendo seu rosto imóvel, o ambiente 3D ficará tremendo.

A Figura 4.4 apresenta quadros em que a face não foi encontrada e, para processamento nas etapas seguintes do programa, foi utilizado a predição do filtro de Kalman.

Na Figura 4.5, os retângulos verdes representam as faces corrigidas pelo filtro de Kalman e os retângulos pretos representam a face detectada, e no último quadro nela podemos ver uma má detecção que foi corrigida pelo filtro.

A Figura 4.6 apresenta quadros em que a face não foi detectada, assim o retângulo verde representa a predição do filtro de Kalman.

Para a detecção da face, são utilizados dois classificadores treinados para detectar faces frontais e um para detectar faces em perfil, este para os casos em que o usuário inclinar a cabeça. Ainda assim, há muitos casos em que os classificadores não são capazes de detectar a face e o programa precisa trabalhar com a predição feita pelo filtro. Casos como o usuário com a cabeça inclinada, virada para algum lado ou em movimento rápido, e neste último, ocorre um blur na imagem devido a rapidez do movimento.

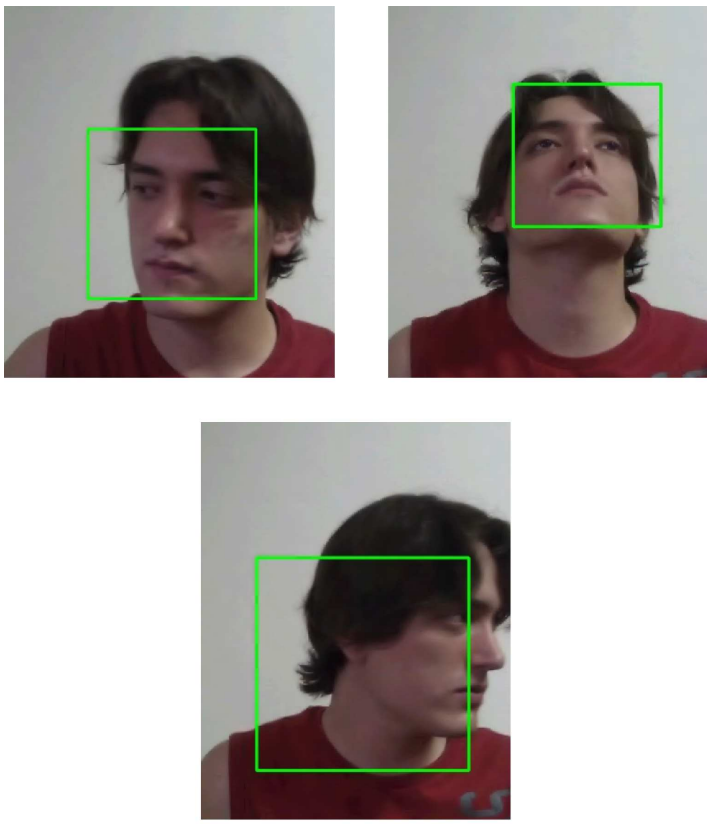


Figura 4.4: Frames em que a face não foi detectada, assim, foi feita a predição

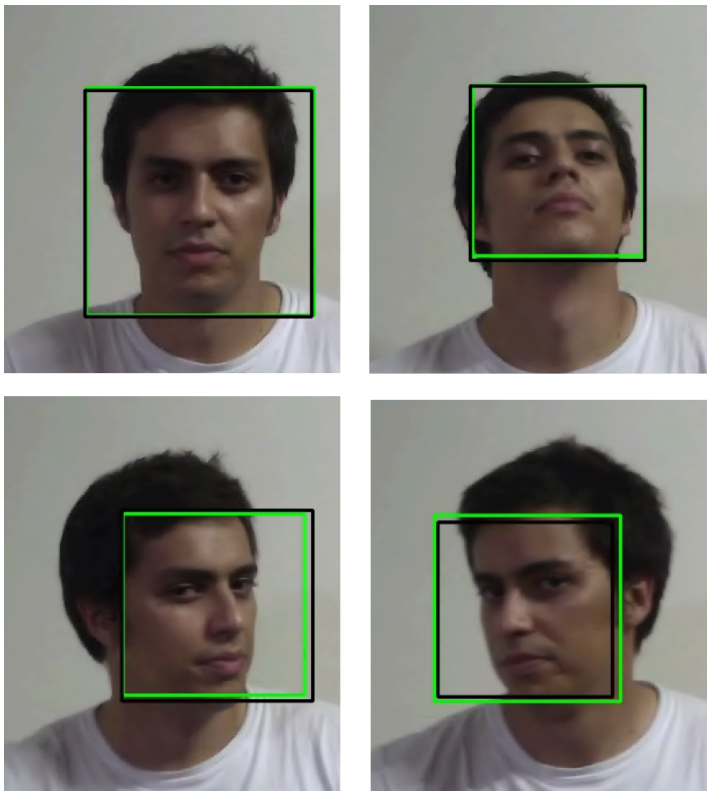


Figura 4.5: Faces detectadas e suas amostras corrigidas

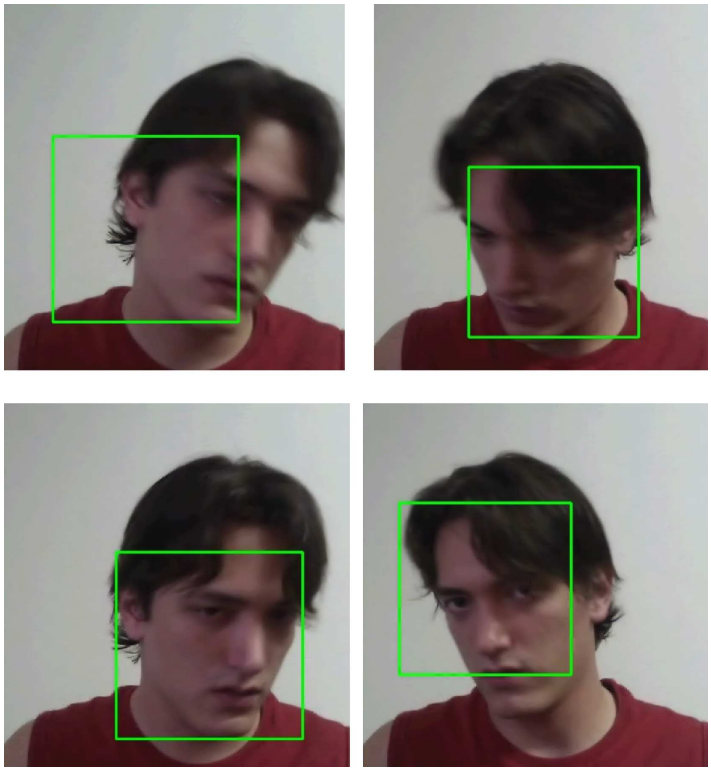


Figura 4.6: Faces preditas

4.3 DETECÇÃO DE OLHOS

Na detecção de olhos, foram utilizados três classificadores, um treinado para detectar qualquer olho, outro treinado para detectar olhos direitos e outro treinado para detectar olhos esquerdos. Caso só um olho seja encontrado, foi apresentado na seção 3.5.1 uma estimativa para o olho que não foi detectado. Após a detecção e a estimação mencionada, é feita a correção das amostras pelo filtro de Kalman.

Nessa detecção, a região em que se procura os olhos é mostrada na Figura 3.6. Se fosse feita a procura por olhos na imagem inteira, a detecção retornaria muitos resultados falsos-positivos, então é necessário ser feita a detecção da face. Se a procura fosse feita na face inteira, ainda haveria muitos falsos positivos, então é feita a procura de olhos apenas na região superior da face, reduzindo muito os resultados falsos-positivos, mas ainda é necessário um processo de refinamento e um processo para saber qual é a amostra do olho, e estes processos foram apresentados na seção 3.5.1. A Figura 4.7 apresenta a detecção de olhos em diferentes regiões para um mesmo quadro: em (a) há a procura por olhos na face inteira e em (b) apenas na região mostrada na Figura 3.6, o retângulo verde é a face detectada, logo é a região de interesse de procura de (a).

Com todos os resultados obtidos da detecção, considerando o caso em que os dois olhos são encontrados, eles são separados em olho direito e olho esquerdo, é feita a média desses resultados e a amostra considerada é a média do respectivo olho. Este processo foi apresen-

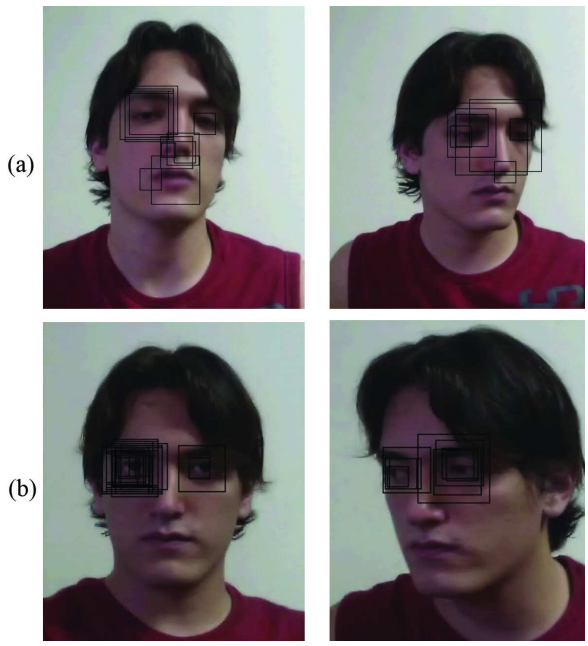


Figura 4.7: Detecção de olhos em diferentes regiões (a) face inteira (b) parte superior da face

tado com mais detalhes na seção 3.5.1.

No caso em que apenas um olho é encontrado, o outro é estimado de acordo com a predição da distância entre os dois olhos. Com esta estimativa, mesmo sendo encontrado apenas um olho durante uma certa quantidade de quadros, é possível ser feito o tracking dos dois olhos. Isto é mostrado na Figura 4.8, onde há o movimento da face com um olho tampado.

Após as detecções, é feita a correção das amostras pelo filtro de Kalman para serem utilizadas na etapa seguinte do programa. A Figura 4.9 mostra os resultados retornados pelos classificadores da detecção de olhos, sendo estes os retângulos pretos, e a amostra corrigida pelo filtro de Kalman representada pelo retângulo verde.

A Figura 4.10 apresenta dois quadros que foram feitos detecção, predição e correção das amostras da face e dos olhos, os retângulos verdes são referentes as amostras da face a dos olhos corrigidas.

Com as mesmas restrições da detecção de face, a detecção de olhos também possui muitos casos em que os olhos não são detectados, já que os classificadores foram treinados para detectar olhos frontais. Dependendo da posição do usuário, não é possível detectar os olhos, assim como em casos que haja movimento rápido, sombras em algum dos olhos ou cabelo na frente dos olhos. Na Figura 4.11 é possível ver quadros em que não foi possível detectar algum dos olhos, lembrando que os retângulos pretos são referentes a detecção e os verdes a predição ou correção.

Figura 4.8: Frames em que apenas um olho foi encontrado, assim o outro foi estimado

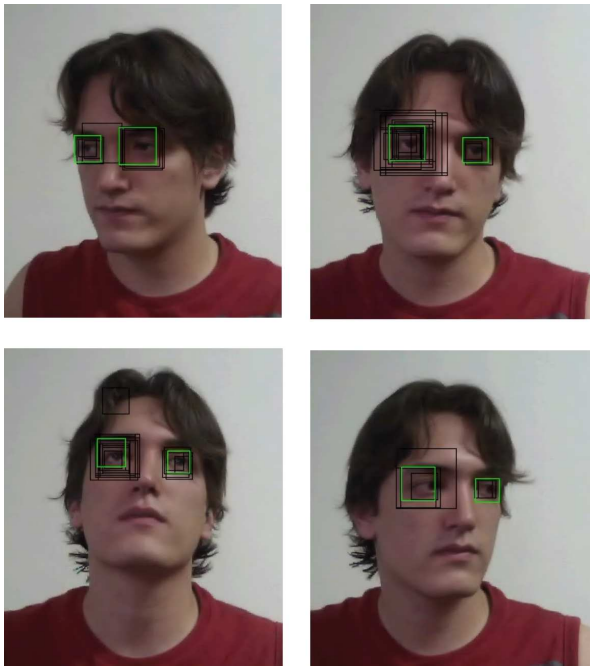


Figura 4.9: Olhos detectados e suas amostras corrigidas

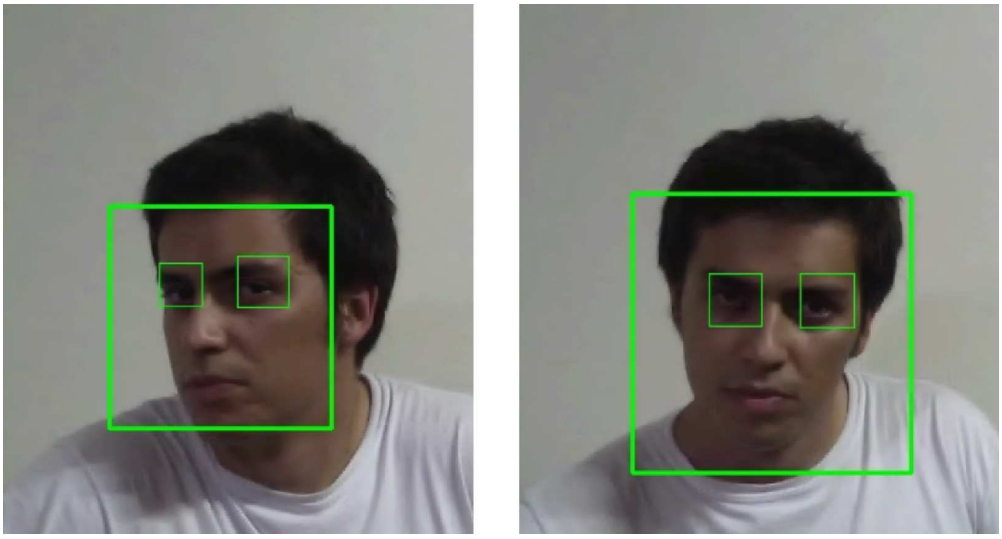


Figura 4.10: Face e olhos detectados, preditos e corrigidos

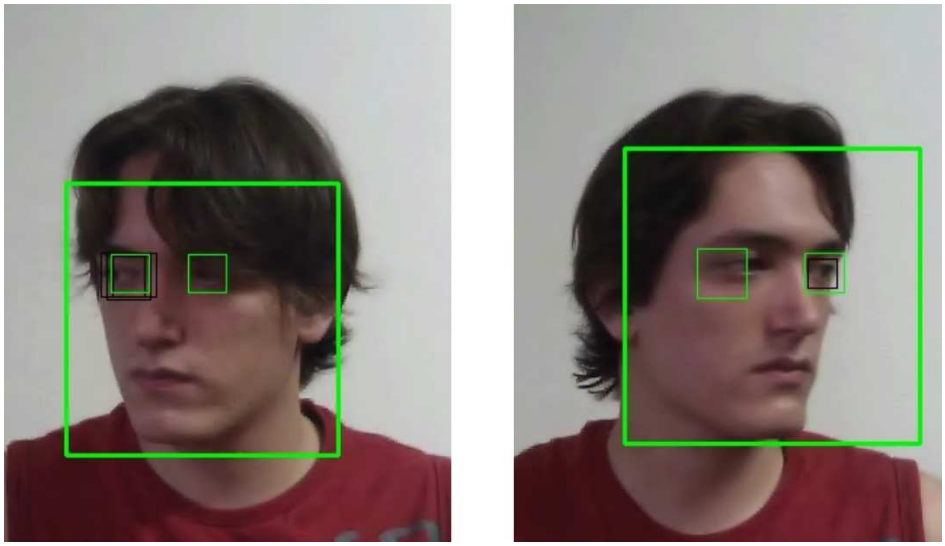


Figura 4.11: Algum dos olhos não detectado, logo ocorrendo a estimação do olho não detectado

4.4 ESTIMAÇÃO DE POSIÇÃO

Possuindo a estimativa dos dois olhos, é possível calcular a posição do usuário, como apresentado na Seção 3.6.1, utilizando a distância em pixels entre os dois olhos estimados.

O método de cálculo desenvolvido neste trabalho e apresentado na Seção 2.5, necessita que o usuário esteja com a face de forma que a câmera a capture de maneira frontal. A coleta de dados para traçar gráficos que mostram o comportamento do método em relação à posição real do usuário é compicada devido à dificuldade de medição da posição do usuário em relação à câmera.

Considerando a posição em que o usuário fique com o centro dos dois olhos de frente ao meio da câmera e esteja à 50 centímetros de distância dela, o usuário possui posição (x,y,z) de (0,0,50).

Na Figura 4.12, o gráfico de *posição estimada x posição real* mostra a variação da distância do usuário ao longo do eixo x da câmera, no sentido horizontal, mantendo a altura dele em relação ao solo constante, logo, a posição no eixo y constante. A distância dele à câmera também é constante, logo, sua posição no eixo z é constante. Caso a estimação fosse precisa, o resultado seria uma reta e para curva obtida, observamos que os dados coletados se aproximam de uma reta. O erro máximo dos dados colhidos foi de 1.5 cm e o erro médio de 1.02 cm, ainda assim a estimação foi boa para os objetivos do projeto. O erro estimado para a posição X do usuário tendo como referência a câmera pode ser visto na Figura 4.13. Como já foi explicado nas seções anteriores, há variação da distância estimada para uma face imóvel, além da dificuldade de medição que causa erros da ordem de centímetros, sendo o erro máximo aproximadamente 10% da medida real. Mesmo assim os valores obtidos continuam se aproximando dos valores reais.

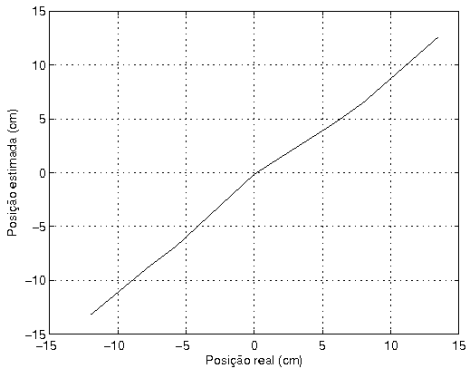


Figura 4.12: Curva da *posição estimada x posição real* do usuário ao longo do eixo x

É apresentado na Figura 4.14 o gráfico de *posição estimada X posição real*, em que há a variação na diferença de altura do usuário tendo como referência a câmera, assim tendo variação no eixo y, e mantendo a posição no eixo x e z constantes. O erro máximo ocorrido é de 1.6 cm e o erro médio é de 0.96 cm. A Figura 4.15 apresenta o erro estimado da posição

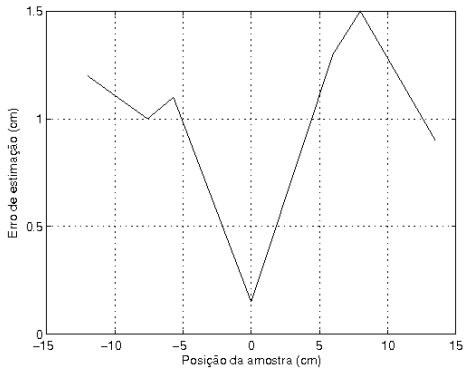


Figura 4.13: Erro de estimação para a posição x do usuário

y do usuário tendo como referência a câmera. Houve uma diferença média razoável entre as medidas reais e as estimativas, mas ainda assim a aproximação é suficiente para os objetivos do projeto.

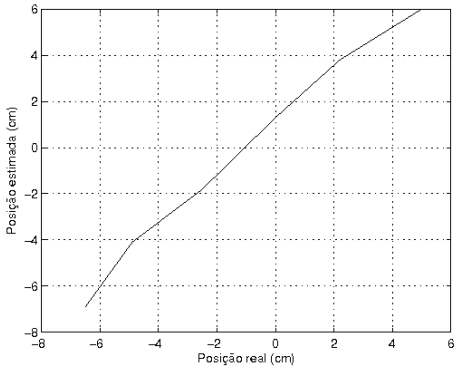


Figura 4.14: Curva da *posição estimada x posição real* do usuário ao longo do eixo y

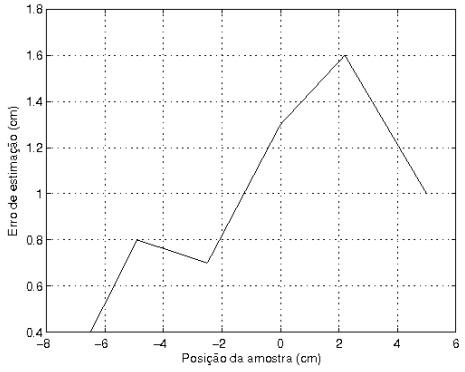


Figura 4.15: Erro de estimação para a posição y do usuário

Como há variações nas estimativas dos olhos, após o cálculo, teremos uma variação no cálculo da posição mesmo que o usuário se mantenha imóvel.

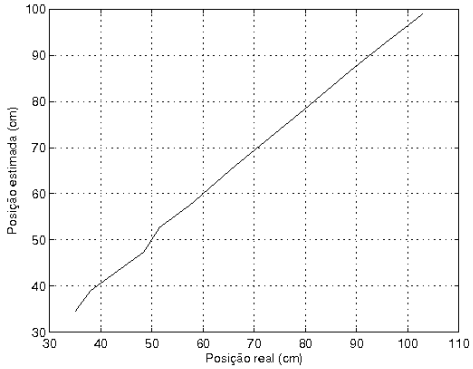


Figura 4.16: Curva da *posição estimada x posição real* do usuário ao longo do eixo z

Na Figura 4.16, há o gráfico de *posição estimada x posição real*, desta vez, a variação ocorre no eixo z, ou seja, na distância do usuário até a câmera, com a posição (x,y) do usuário constante. Podemos observar como a curva se aproxima de uma reta, mostrando que, apesar da variação, os resultados estão próximos. O erro médio é de 1.47 cm e o erro máximo é de 4 cm, sendo que o erro é maior quanto maior for a distância. O gráfico do erro estimado da distância do usuário até a câmera pode ser visto na Figura 4.17.

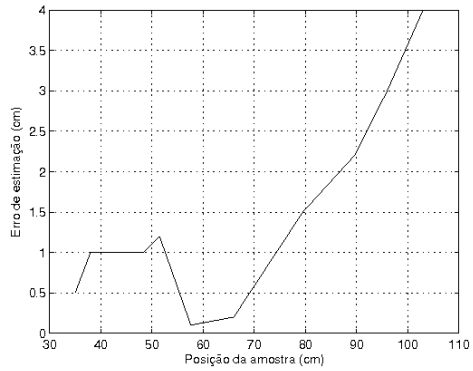


Figura 4.17: Erro de estimação para a posição z do usuário

Pelos gráficos, podemos ver que os resultados são próximos das medições reais. Para o projeto, não é necessária uma grande precisão da posição, então o método de cálculo utilizado supre as necessidades do projeto.

4.5 AMBIENTE VIRTUAL 3D

Nesta seção é mostrado o resultado final do projeto, e este resultado possui um caráter subjetivo e para realmente apreciar tal resultado, é necessário utilizar o programa. As fotos desta seção foram tiradas da perspectiva do usuário.

Utilizando o cenário do cubo, temos o resultado apresentado na Figura 4.18, com o usuá-

rio em diferentes posições. Pode-se ver que, quando o usuário se levanta como se fosse observar a parte de cima do cubo, o ambiente é redesenhado de forma que o usuário possa ver isso. Assim acontece quando ele tenta ver o cubo por baixo. O cubo também se torna menor a medida que o usuário se afasta e se torna maior a medida que o usuário se aproxima.

Utilizando o cenário das geometrias, o resultado é apresentado na Figura 4.19, com o usuário em diferentes posições. Caso o usuário se mantenha de forma que o centro de seus olhos fique perto do meio da tela, no ambiente 3D, ele estará atrás do cubo. O usuário consegue ver o resto do cenário caso ele tente sair de trás do cubo. O cenário pode ser visto de várias perspectivas, já que o cubo está obstruindo a vista do cenário.

A Figura 4.20 mostra o resultado obtido no cenário dos alvos, com o usuário em diferentes posições. Este é o cenário de melhor resultado, os alvos estão dispostos em diferentes profundidades, então, quando o usuário se move horizontalmente ou verticalmente, os alvos são redesenhados de forma a se ter a noção de profundidade. Quando o usuário se aproxima, há alvos que saem da tela, e quando ele se afasta, tais alvos reaparecem.

Podemos ver pelas imagens que o objetivo do projeto foi atingido, tendo-se uma experiência muito agradável com o programa. Há um problema no programa, que não pode ser visto em imagens e este problema já foi mencionada várias vezes neste manuscrito. Mesmo o usuário ficando imóvel, há variação nas amotras obtidas da face e dos olhos, assim, acarretando numa variação da posição (x,y,z) do usuário em torno de um certo ponto. Assim, com o usuário parado, o ambiente virtual dá a impressão de ficar tremendo, pois o ambiente virtual sente diferenças na escala de centésimos, que é a grandeza de variação da posição do usuário. Este problema não impede o bom funcionamento do programa ou torna a experiência para o usuário ruim, mas o resultado poderia melhorar caso tal problema não existisse.

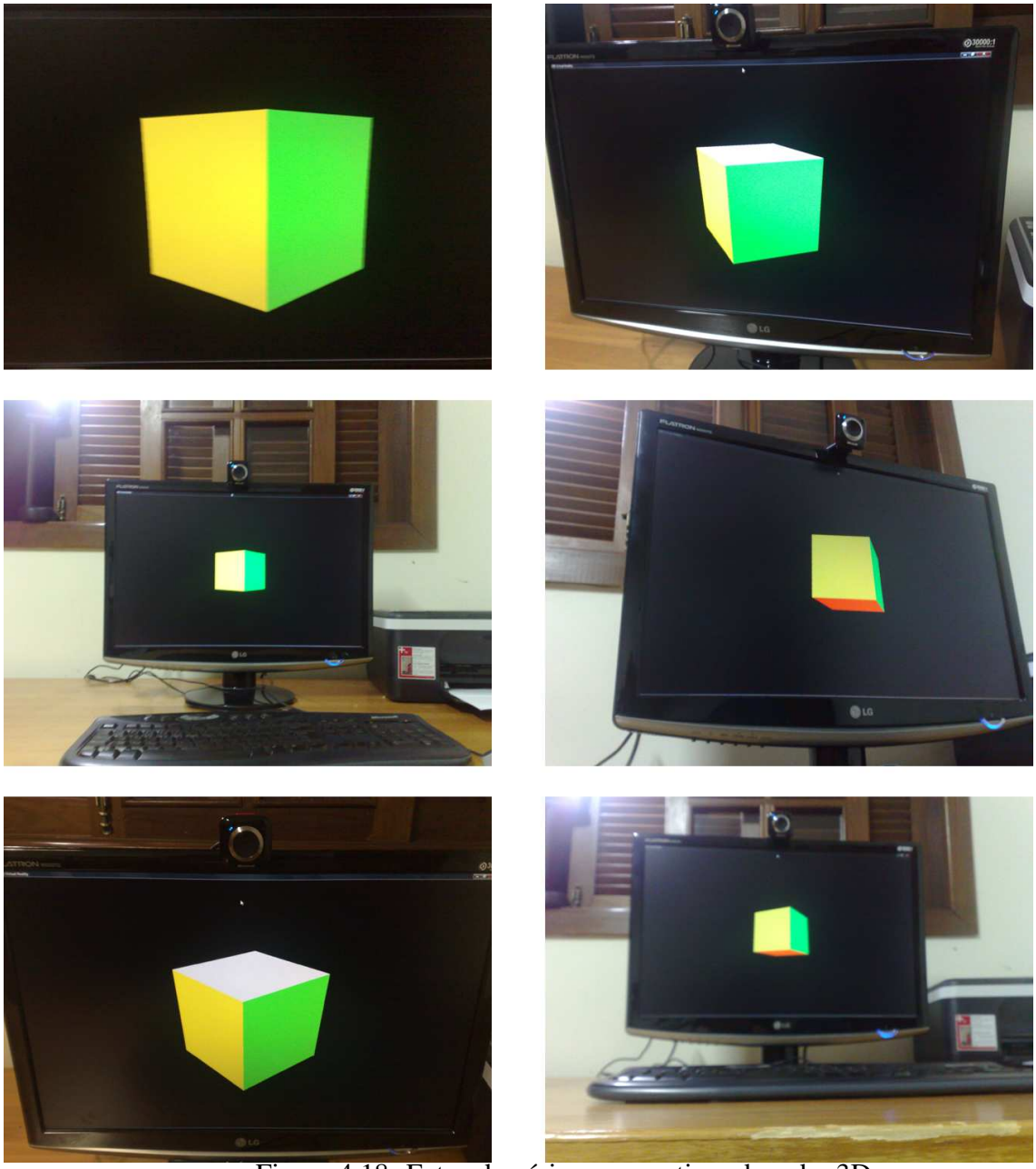


Figura 4.18: Fotos de várias perspectivas do cubo 3D

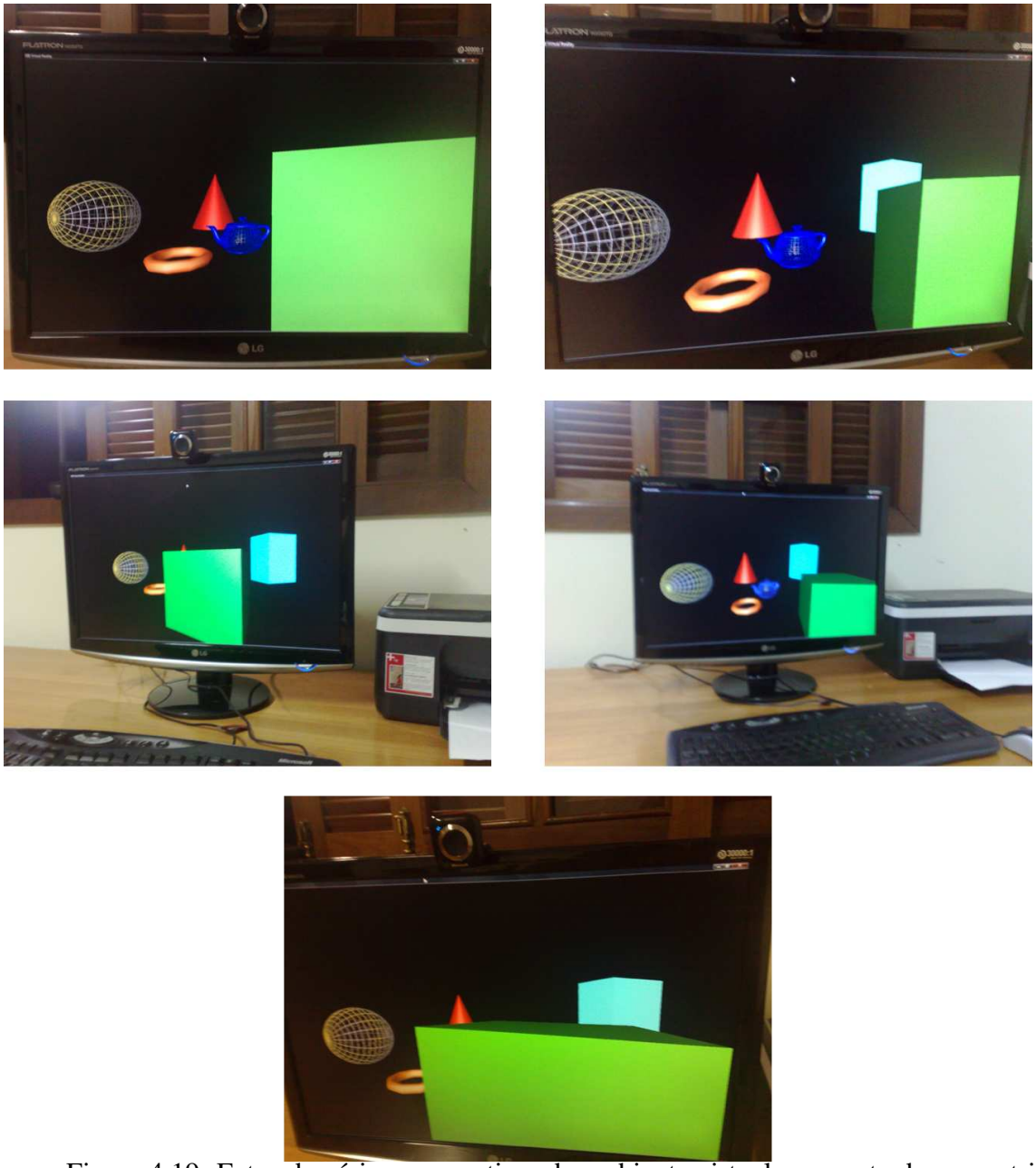


Figura 4.19: Fotos de várias perspectivas do ambiente virtual composto de geometrias

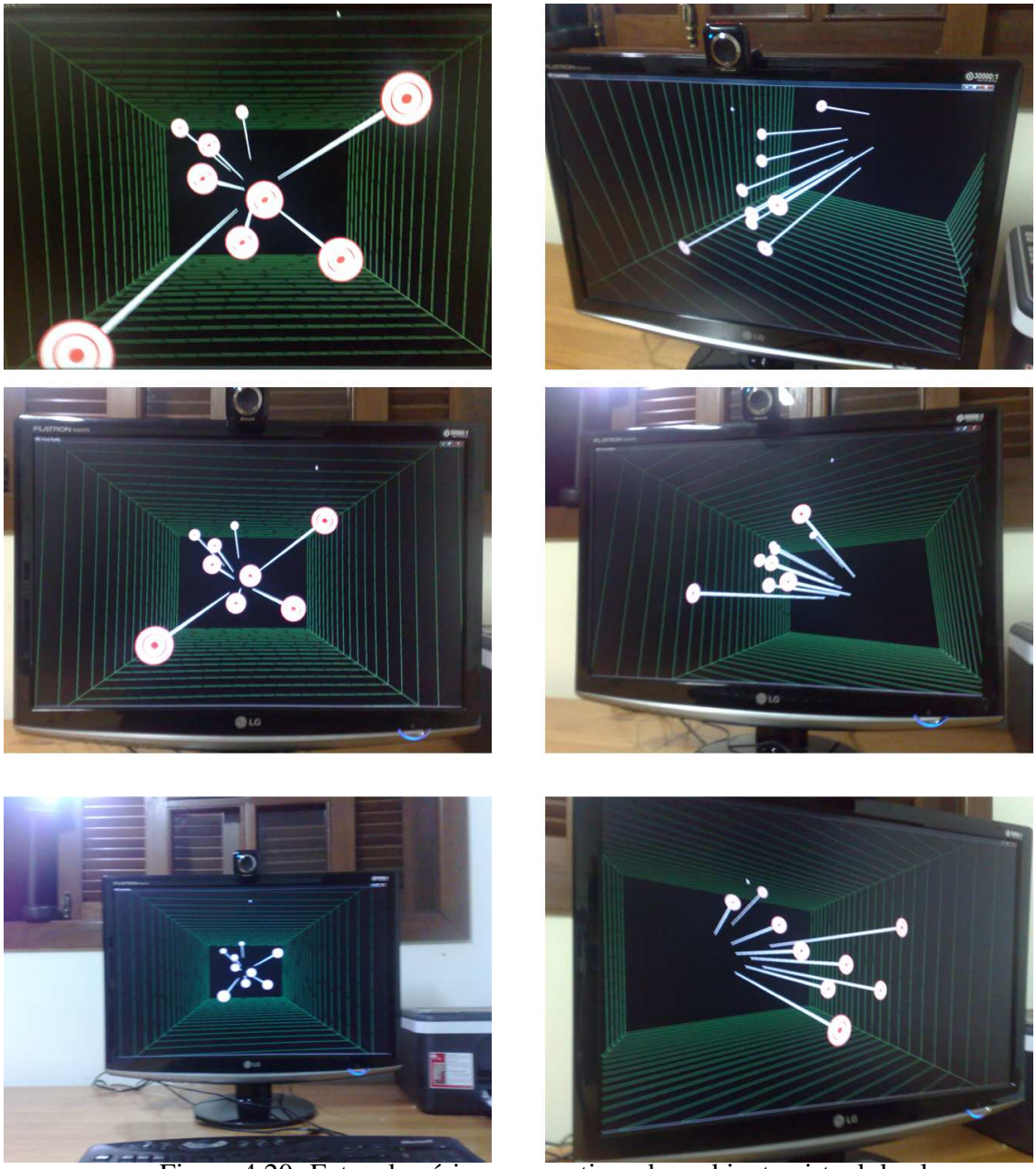


Figura 4.20: Fotos de várias perspectivas do ambiente virtual de alvos

5 CONCLUSÕES E PROPOSTAS DE TRABALHOS FUTUROS

Este projeto teve como objetivo fazer uma interface de interação visual entre uma pessoa e um ambiente virtual. Utilizando uma câmera USB no computador, o usuário pode interagir visualmente com um objeto no ambiente virtual assim como ele interage visualmente com um objeto real. O objeto virtual muda de perspectiva de acordo com o movimento do usuário.

Para o desenvolvimento do projeto, foi necessário o estudo e implementação de detecção facial, detecção de olhos, estimação baseada em filtro de Kalman, cálculo de distância de uma pessoa à câmera e programação em OpenGL. O programa desenvolvido consiste em um ambiente virtual que é redesenhado de acordo com a posição do usuário em relação à tela do computador. Tendo calculado o ponto de vista do usuário, os objetos virtuais reagem visualmente à mudança de posição dele como se fossem objetos reais dentro da tela do computador. No projeto, é considerado que o usuário estará com sua face de forma que a câmera a capture de maneira frontal, pois para ao usuário ver a tela, é nessa posição que ele provavelmente estará. Assim, é considerado que o usuário mantenha tal disposição da sua cabeça para a detecção facial, de olhos e cálculo de distância do usuário à câmera.

Para a detecção facial, foram estudados diferentes métodos de detecção e foi escolhido o método Viola-Jones, devido à sua rapidez. Com o suporte da biblioteca OpenCV, a detecção, utilizando tal método, foi implementada com ótimos resultados, sendo possível detectar a face do usuário em várias posições e dificilmente retornando resultados falso-positivos. A detecção facial no projeto é utilizada para restringir a área de procura dos olhos. Tal face, quando não é encontrada, é predita e quando encontrada, sua forma é corrigida para diminuir erros de medidas.

A detecção de olhos também foi implementada utilizando o método Viola-Jones. A procura por olhos ocorre apenas na parte superior da face estimada na etapa anterior, reduzindo assim o número de resultados falsos-positivos. A detecção utiliza classificadores que encontram olhos em posição frontal para a câmera. Após a detecção, é feito um refinamento e separação de resultados em olho direito e olho esquerdo, para depois ser feita a correção de tais resultados. Quando algum olho não for encontrado, tal olho é predito. Com a face do usuário sendo capturada pela câmera de maneira frontal, caso não haja sombras no olho do usuário, o resultado é muito bom, mas quando o usuário se move rápidamente ou inclina a cabeça, o detector de olhos começa a ter problemas e os olhos muitas vezes não são detectados. Isto acontece por limitações do método Viola-Jones. Além disso, tal método retorna vários candidatos a olhos, e de quadro a quadro, a posição e largura desses candidatos, considerando que eles são retângulos, variam. Assim, há uma variabilidade nos olhos encontrados a cada quadro, que faz com que a posição do usuário calculada varie e, no resultado final do programa, faz com que o ambiente virtual fique tremendo.

Com tais etapas concluídas, foi desenvolvido um método de cálculo da posição, em três dimensões, de uma pessoa em relação à câmera. Não foi encontrada nenhuma bibliografia referente a esse tipo de cálculo, por isso ele foi desenvolvido. Embora aproximados e dentro da limitação de que o usuário tenha que ficar com o rosto virado frontalmente para a câmera, os resultados são bons o suficiente para serem utilizados no projeto, já que não é necessário uma grande precisão pois os valores da posição do usuário são convertidos para a escala do ambiente virtual 3D. Como o cálculo considera uma certa postura para a cabeça do usuário, caso ele a incline, o resultado do cálculo de sua distância é alterado, aumentando assim o erro de estimação de sua posição.

Para poder ter um melhor *tracking* da face, dos olhos e posição do usuário, com correção dos valores e predição de tais quando necessário, foi utilizado o filtro de Kalman. O filtro dá mais robustez ao sistema e o torna mais estável, diminuindo variações e erros nos resultados. Além disso, o filtro de Kalman também é um filtro passa-baixa, diminuindo assim as pequenas variações de alta freqüência, já mencionadas, devido a detecção de olhos. Com o valor da posição do usuário em relação à câmera, é possível obter a posição do usuário em relação ao centro da tela, que é utilizado na etapa seguinte.

Como última etapa do projeto, foram desenhados três ambientes virtuais distintos: um cubo com cada uma das faces de uma cor diferente, um ambiente no qual várias geometrias diferentes foram espalhadas pelo cenário e um ambiente com alvos saindo do fundo do cenário. Nesses ambientes, é inserido uma câmera virtual em cada um em uma determinada posição do cenário, que representa a perspectiva do usuário. É utilizada a informação de posição do usuário em relação à tela para mover tal câmera virtual, sendo essa posição ajustada por um fator de escala, já que cada ambiente virtual possui sua própria escala.

O resultado geral do projeto é satisfatório, sendo possível ver o ambiente virtual de diferentes perspectivas. Devido à forma com que a detecção de olhos foi implementada, há vezes em que os olhos não são encontrados. Problemas na variação da iluminação, sombras ou mudança na inclinação da cabeça do usuário alteram bastante o resultado de tal detecção. Além disso, os olhos detectados, em forma de retângulos, variam suas formas de quadro a quadro, mesmo que o usuário mantenha sua face imóvel, alterando assim o resultado do cálculo da posição do usuário. Isto faz com que a câmera virtual também varie sua posição, dando uma sensação leve de tremedeira na imagem da tela. Isto pode ser resolvido no projeto do filtro de Kalman da posição, mas há uma troca entre tal sensação de tremedeira e o tempo de resposta do filtro ao movimento do usuário. Quanto menor esta variação de posição, maior é o tempo de resposta do filtro. Com um tempo de resposta alto, o resultado do projeto pode não ser tão satisfatório. Então optou-se por uma leve variação e um pequeno atraso na resposta do filtro, este atraso sendo pouco perceptível.

Com intuito de melhorar e enriquecer o trabalho feito neste projeto, são feitas algumas propostas de trabalhos futuros para melhorar o desempenho do programa e trazer uma experiência visual mais agradável ao usuário. Estas propostas são:

1. Implementar um método de reconhecimento de orientação da cabeça do usuário.

2. Modificar o método de cálculo de distância de uma pessoa à câmera, incluindo a informação da orientação da cabeça do usuário.

3. Implementar outro método para detecção de olhos, que seja mais preciso e continue sendo rápido ou melhorar a detecção de olho feita no programa, de forma a identificar a iris do olho e assim obter uma medida mais precisa.

4. Desenhar os ambientes 3D de forma que a imagem na tela do computador seja 3D utilizando 3D anaglífico, para visualização com óculos que possuam uma lente de cada cor, ou tornar o programa compatível com o *NVIDIA 3D vision*, método da NVIDIA em que é possível ver programas gráficos, feitos com OpenGL, em 3D utilizando óculos do tipo *shutter*. Para estes casos, novos classificadores teriam de ser treinados para a detecção facial e detecção de olhos.

5. Aplicar este projeto em um jogo simples que seja *open source*. Um jogo que rode a 60 quadros por segundo, se for adicionado o processamento deste projeto, rodará a 20 quadros por segundo, sem grandes prejuízos à experiência do jogo.

[1] G. Welch e G. Bishop, "An introduction to the kalman filter," July 2006.

[2] N. A. Ming-Hsuan Yang, David. J. Kriegman, "Detecting faces in images: A survey," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, January 2002.

[3] H. Rowley, S. Baluja, e T. Kanade, "Neural network-based face detection," *IEEE Transactions On Pattern Analysis and Machine intelligence*, vol. 20, pp. 23–38, 1996.

[4] I. Kukenys e B. McCane, "Support vector machines for human face detection," in *NZCSRSC 2008: Proceedings of the New Zealand Computer Science Research Student Conference*, 2008.

[5] P. Viola e M. J. Jones, "Robust real-time face detection," *Internation Journal of Computer Vision*, pp. 137–154, July 2003.

[6] Y. Freund e R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, pp. 119–139, August 1997.

[7] R. Lienhart e J. Maydt, "An extended set of haar-like features for rapid object detection," *IEEE ICIP 2002*, vol. 1, pp. 900–903, September 2002.

[8] Z. Zhu, K. Fujimura, e Q. Ji, "Real-time eye detection and tracking under various light conditions," in *Proceedings of ETRA: Eye Tracking Research Applications Symposium*. ACM Press, 2002, pp. 139–144.

[9] S. Asteariadis, N. Nikolaidis, A. Hajdu, e I. Pitas, "An eye detection algorithm using pixel to edge information," in *Proceedings of Second IEEE-EURASIP International Symposium on Controle, Communications and Signal Processing*, 2006.

[10] E. Nadernejad, S. Sharifzadeh, e H. Hassanpour, "Edge detection techniques: Evaluations and comparisons," *Appl. Math. Sci*, vol. 2, no. 29-32, 1507-1520, 2008.

[11] I. T. Jolliffe, *Principal Component Analysis*, 2^o ed., ser. Springer Series in Statistics. Springer, 2002.

[12] M. Everingham e A. Zisserman, "Regression and classification approaches to eye localization in face images," in *Proceedings of the 7th Internacional Conference on Automatic Face and Gesture Recognition (FG2006)*, April 2006, pp. 441–446.

[13] J. N. T. D. Dave Shreiner, Mason Woo, *OpenGL Programming Guide*. Adisson-Wesley, 2008.

[14] G. Bradsky e A. Kaehler, *Learning OpenCV*. O'Reilly Media, Inc, 2008.

[15] P. Campadelli, R. Lanzarotti, e G. Lipori, "Eye localization: a survey," in *Fundamentals of verbal and non verbal communication and the biometric issue*. IOS Press, 2007, pp. 234–245.

[16] S. Amarnag, R. S. Kumanaran, e J. N. Gowdy, "Real time eye tracking for human computer interfaces," in *Proceedings of the International Conference on Multimedia and Expo (ICME)*, vol. 3, july 2003, pp. 557–560.