PMR2550- PROJETO DE CONCLUSÃO DE CURSO II

Reconhecimento e Rastreio de Pessoas Utilizando Câmera PTZ

Monografia Final

Alunos

Caio Martine Morgante 7210741
Fábio Naoto Majima 7206558

Orientador

Prof. Marcos Barretto

Caio Martine Morgante

Fábio Naoto Majima

Reconhecimento e Rastreio de Pessoas Utilizando Câmera PTZ

Monografia apresentada no Departamento de Engenharia Mecatrônica e Sistemas Mecânicos da Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro. Área de Concentração: Engenharia Mecatrônica

São Paulo

2015

Caio Martine Morgante

Fábio Naoto Majima

Reconhecimento e Rastreio de Pessoas Utilizando Câmera PTZ

Este relatório é apresentado como requisito parcial para obtenção do título de Engenheiro na Escola Politécnica da Universidade de São Paulo. É o produto do nosso próprio trabalho, exceto onde indicado no texto. O relatório pode ser livremente copiado e distribuído desde que a fonte seja citada.

São Paulo

2015

FICHA CATALOGRÁFICA

Morgante, Caio
Majima, Fábio
Reconhecimento e Rastreio de Pessoas Utilizando Câmera PTZ / C. Morgante, F. Majima -- São Paulo, 2015.
85 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Detecção de pessoas 2.Identificação de pessoas 3.Rastreamento de pessoas I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t.

Sumário

1 INTRODUÇÃO

1.1 Motivação

Em um cenário global de insegurança, com ataques terroristas frequentes, violência urbana, entre inúmeras outras causas, destaca-se uso de câmeras para o controle da segurança. Países como Alemanha, França, Estados Unidos e Brasil, por exemplo, possuem câmeras para monitoramento do espaço público.

Muitos desses sistemas necessitam de pessoas que monitorem as imagens captadas pelas câmeras e identifiquem visualmente suspeitos ou criminosos procurados. Entretanto, em um ambiente com diversas pessoas em movimento, nem sempre os responsáveis pela vigilância conseguem identificar todos os indivíduos. Um sistema automático de identificação se apresenta como alternativa para o problema.

Sob outro aspecto, a automatização de recursos, isto é, o desenvolvimento e a aplicação de robôs é uma tendência real. De acordo com dados publicados pela International Federation of Robotics (IFR) (1), em 2014, a venda de robôs ultrapassou as 229 mil unidades vendidas, o que representa um crescimento de 29% comparado a 2013. Nesse contexto, a implementação de uma câmera inteligente em um robô, proporcionando ao mesmo uma espécie de “visão”, significa um grande avanço dentro da área.

1.2 Objetivo

Este trabalho aborda a elaboração de um projeto de rastreamento de ambiente através de uma ou mais câmeras PTZ. PTZ corresponde à sigla em inglês “pan, tilt, zoom” que, traduzido, significa “deslocamento panorâmico, inclinação e aproximação”. O rastreamento envolve a localização e acompanhamento de qualquer indivíduo parado ou em movimento no espaço coberto pelas câmeras, além de sua identificação.

O objetivo deste trabalho engloba um conjunto de funcionalidades previstas no produto final do projeto, conforme tópicos abaixo.

- Detecção de faces de pessoas.
- Utilização de dados da câmera para reconhecer indivíduo e armazená-lo em um banco de dados.

- Reconhecimento de pessoas com dados previamente armazenados em um banco de dados.
- Automação das funções da câmera (pan, tilt e zoom) para melhor obtenção de imagens.
- Determinação de características marcantes da pessoa durante o rastreamento (como cor de camisa e estrutura física).
- Rastreio de pessoas em um ambiente com uma ou mais câmeras do sistema.
- Utilização e compartilhamento de dados de mais de uma câmera pelo sistema de segurança.

1.3 Estrutura

Este trabalho parte de diversos artigos e ferramentas open source já existentes.

A preparação para elaborar esse projeto partiu da análise do problema em questão e das tecnologias já existentes na área, em conjunto com o estudo de diversos artigos publicados e da análise de patentes já existentes.

Uma vez estruturada a temática do projeto, seguiu-se com o aprofundamento nos estudos sobre as bibliotecas utilizadas neste projeto no desenvolvimento de softwares de manipulação de imagens, OpenCV (2) e JavaCV (3).

A partir daí, o projeto foi dividido em diversas etapas, com aumento gradual de exigência. A primeira envolve a elaboração de um programa de reconhecimento de rostos em fotos. O próximo objetivo foi a detecção de indivíduos em fotos e posteriormente sua identificação. Em seguida, foi implementado o rastreio de pessoas em movimento.

Em paralelo, foram desenvolvidos os comandos para controlar as três funcionalidades da câmera PTZ para, então, combiná-los com as funções de detecção, identificação e rastreamento.

A última etapa do projeto uniu todas as implementações das etapas anteriores dentro da ferramenta JADE (4).

Neste trabalho, o conceito de rastreamento (“tracking”, em inglês) deve ser interpretado como o acompanhamento monitorado e automatizado de indivíduos em um ou mais ambientes.

2 TRABALHOS CORRELATOS

Para o desenvolvimento deste trabalho procuraram-se artigos e teses similares na literatura. Esse estudo alinhou os conhecimentos técnicos adquiridos no curso de engenharia mecatrônica com o conteúdo necessário para iniciar o desenvolvimento deste Trabalho de Conclusão de Curso.

Os três principais temas abordados neste estudo foram: a detecção de faces, a identificação de pessoas e o rastreamento de pessoas ou objetos.

2.1 Detecção

2.1.1 Detecção de faces

Em Bezerra (5), são citadas as abordagens mais conhecidas dos sistemas de detecção facial: detecção baseada na cor da pele humana (color based), detecção por semelhança com molde padrão, no caso um rosto humano, definido manualmente (template based), e detecção através de classificadores binários de características faciais (feature based).

Singh (6) utiliza algoritmos de reconhecimento facial baseado na cor da pele humana, fazendo a comparação de 3 algoritmos: RGB, YCbCr, HSI. O autor cita que a vantagem do processamento de cores decorre do fato desse processamento ser muito mais rápido do que outros algoritmos que utilizam outras características para a detecção de faces. Contudo, os algoritmos que utilizam as cores como parâmetro sofrem com problemas de intensidade de iluminação, além da possibilidade de câmeras diferentes gerarem imagens diferentes para a mesma cena ou pessoa, segundo cada configuração.

O algoritmo RGB consiste de 3 aditivos primários: vermelho, verde e azul, que dão nome ao algoritmo (red, green, blue). Componentes dessas 3 cores são combinadas para produzir a cor resultante. (6)

Embora o algoritmo simplifique o problema do sistema de computação gráfica, ele não é ideal para todas as aplicações. Os componentes vermelho, verde e azul são altamente correlacionados, dificultando a execução de algoritmos de processamento de imagens. Muitos desses algoritmos necessitam de uma medida da intensidade das cores nas imagens, mas o RGB informa apenas as cores presentes, sem dimensionar as suas intensidades.

O algoritmo YCbCr foi desenvolvido em resposta a demanda por algoritmos para lidar com informações digitais dos vídeos. O YCbCr é um sistema digital de cores que separam o RGB (vermelho, verde, azul) em informações de luminância (componente Y) e crominância (componentes Cb e Cr). Essa separação de informações é útil nas aplicações, porém a especificação das cores não é muito intuitiva.

O algoritmo HSI utiliza três propriedades para descrever cores: a matiz, a saturação e a intensidade (Hue, Saturation, Intensity, respectivamente). Ao utilizar o método, não pe necessário saber, por exemplo, a porcentagem de azul e verde necessária para produzir uma cor, simplesmente é feito o ajuste de matiz. Para mudar o vermelho para rosa, ajustar a saturação. Para tornar a cor mais escura ou clara, ajusta-se a intensidade.

O estudo conclui que os algoritmos YCbCr e HSI são mais eficientes quando comparados ao RGB para reconhecer faces, porém nenhuma foi capaz de apresentar resultados satisfatórios. (6)

Baskann Bulut e Atalay propuseram um método de segmentação de rostos humanos. Segunda o artigo, algumas características diferem do resto da face pelos valores de intensidade dos pixels e são determinados pela localização do mínimo valor de cinza nas projeções dos pixels da imagem da face. O sistema limita cada característica facial com o menor retângulo utilizando os resultados da projeção horizontal e vertical dos pixels de valor cinza. Esse retângulo é posteriormente comparado com o retângulo extraído manualmente ao redor da característica facial correspondente. (7)

Nesse método foram processadas as projeções em X e Y considerando a média dos valores dos pixels ao longo do eixo vertical e horizontal. Assim cada característica facial gera um valor mínimo em Y, especificando a característica em X.

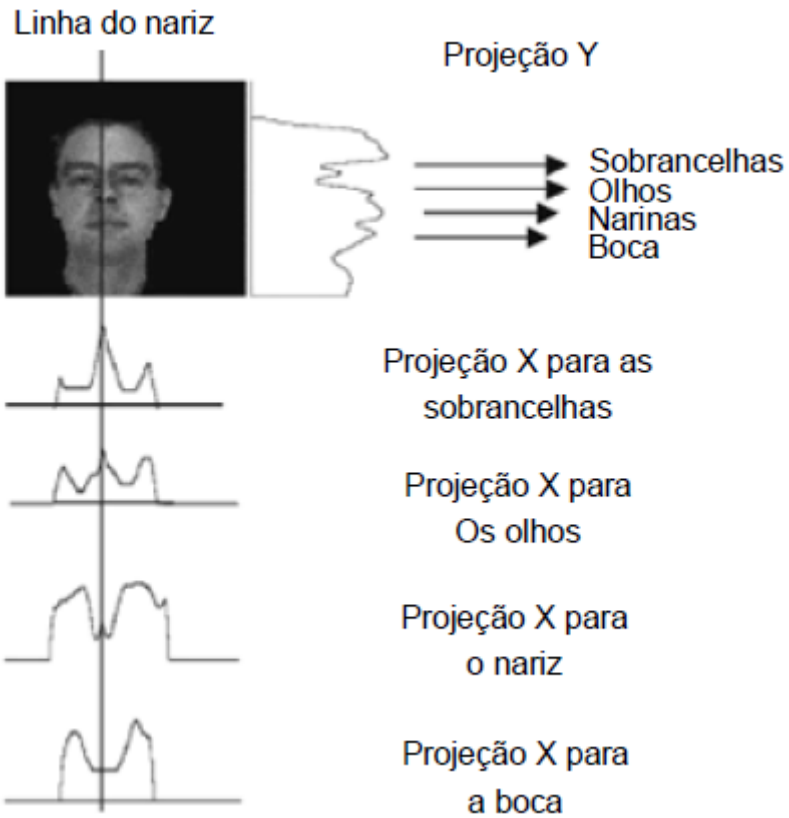

Figura 2.1 – Projeção Y e X para uma imagem. (7)

O resultado da projeção em X reflete a simetria existente no rosto em uma linha vertical que passa pelo nariz. Analisando os mínimos da projeção em Y, percebe-se que o primeiro corresponde à sobrancelha, o segundo aos olhos, o terceiro as narinas, o quarto a boca, e o último ao queixo. Os limites superior e inferior das características faciais são detectados pela projeção, os limites esquerdo e direito, pela projeção X. Estes limites determinam o retângulo para a característica facial. (7)

A extração de pontos característicos da face é um algoritmo baseado nas características visuais humanas, usando uma geometria e simetria facial. O número de pontos característicos deve ser suficiente para permitir a identificação da face. O projeto analisado em (8) localizou nove pontos característicos selecionados de modo a obter invariância com os ângulos. Com dois pontos centrais nos olhos, quatro pontos das extremidades dos olhos, um ponto central no nariz e dois pontos na extremidade da boca, como mostra a figura 2.2.

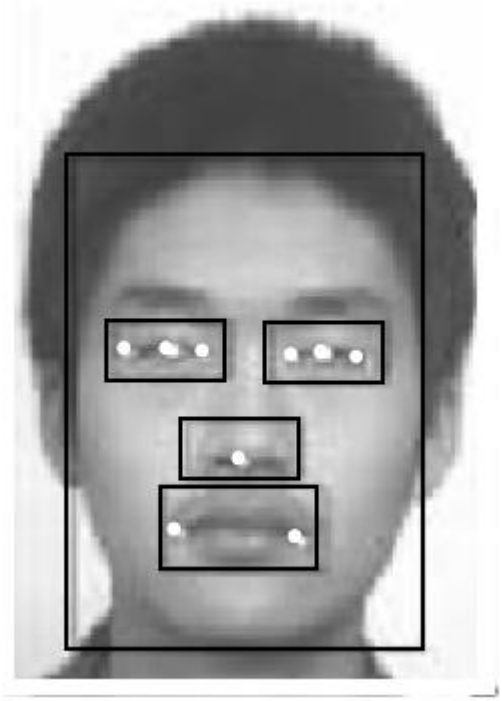

Figura 2.2 – Pontos característicos na face. (8)

O algoritmo analisado apresentou resultados satisfatórios em imagens com rotação de face de até 15º nos sentidos direito e esquerdo. Em imagens com uma rotação acima deste valor, bem como em imagens com forte iluminação lateral, foi verificado uma redução da porcentagem de acerto na localização dos pontos característicos. (8)

Viola e Jones (9) propuseram um algoritmo para detecção de faces. Esse algoritmo está implementado na biblioteca Intel OpenCV e visa localizar em uma imagem características que codifiquem alguma informação do padrão sendo detectado. Esses padrões são baseados nas existências de contrastes orientados entre regiões da imagem (características de Haar).

Quando as características de Haar são aplicadas em uma imagem, são examinados os contrastes naturais devido às características da face (olhos, nariz, boca...), como observado na figura 2.3. Na segunda coluna mostra-se um exemplo de característica de borda formada devido à diferença de contraste da região onde ficam localizados os olhos, região mais escura do que a imediatamente abaixo dela. Na terceira coluna é mostrado um exemplo de característica de linha na região do nariz. Uma região fina e mais clara do que a de seus vizinhos na horizontal, localizada no centro do rosto, próxima a altura dos olhos. (9)

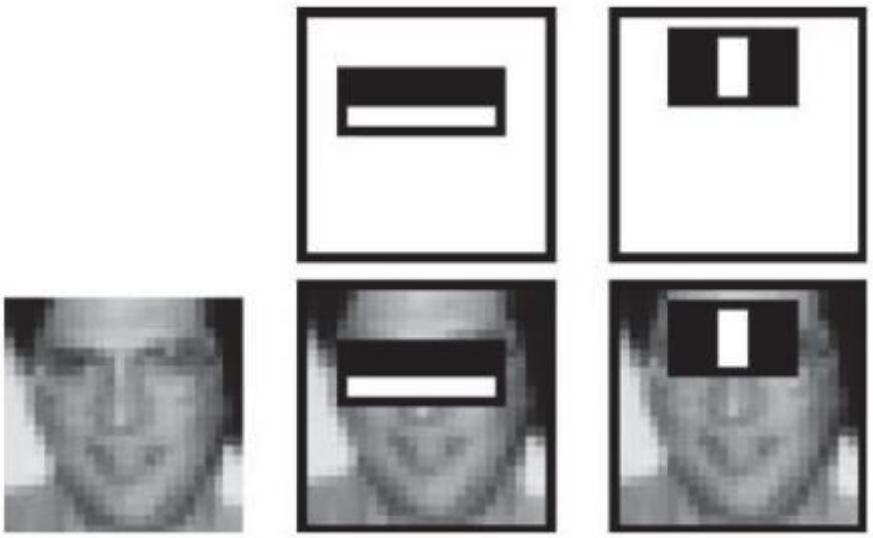

Figura 2.3 – Características de Haar na face. (9)

2.1.2 Detecção de corpos

Brandão (10) sugeriu o uso de “depth images” como alternativa para contornar possíveis variações de luminosidade que interferem na detecção. Essa técnica manipula as imagens obtidas, ignorando características intrínsecas, como cores, e definindo com maior precisão a geometria nos objetos detectados. Para contornar as variações de forma que um corpo humano apresentar em função dos ângulos de captação das imagens, o autor propôs um método híbrido chamado M5AIE. Para a detecção de potenciais partes de corpos, foi proposto um método AGEX (“Accumulative Geosdesic Extrema”), além do algoritmo ASIFT para extração de características marcantes ao corpo. (10)

Figura 2.4 - Imagem em RGB-D. (10)

Figura 2.5 - Depth Image. (10)

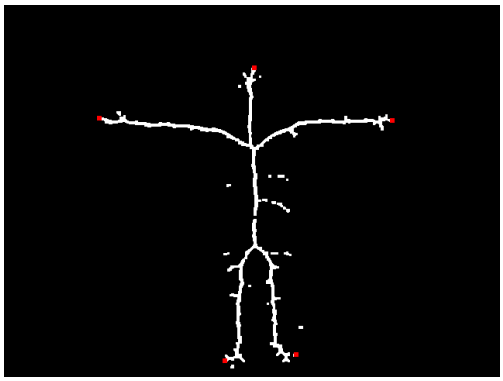

Figura 2.6 - Imagem em AGEX. (10)

Kadim, Daud, Radzi, Samudin e Woon (11) propuseram um método para detectar objetos em movimento durante a movimentação da própria câmera. A detecção é feita utilizando o método de subtração do plano de fundo para quadros sucessivos. Com o movimento da câmera, as imagens obtidas podem ficar embaçadas. Para controlar esse fenômeno, os autores propuseram um método de análise das bordas.

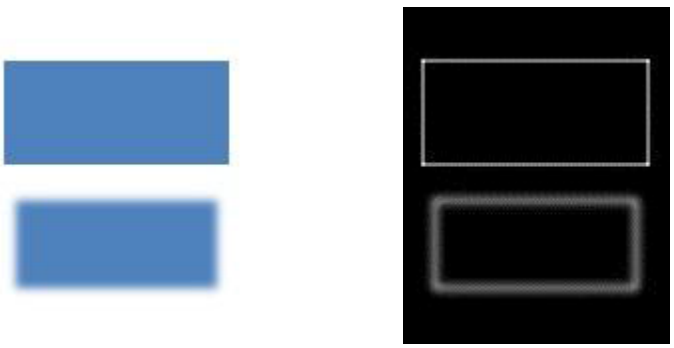

Figura 2.7 - Análise das bordas (11)

Ladonde, Foucher, Gagnon, Pronovost, Derenne e Janelle (12) propuseram um método para identificar seres humanos e veículos utilizando câmeras PTZ. O artigo prevê a alimentação de uma memória baseada nos padrões de atividade armazenados pela câmera (13). Os autores se basearam no método de sobreposição de campos de visão (FOVs) proposto por Javed (14). Basicamente, o sistema é treinado para aprender a topologia dos FOVs, estabelecendo uma interdependência entre as câmeras baseada na predição da velocidade e da relação espaço-temporal dos objetos em movimento.

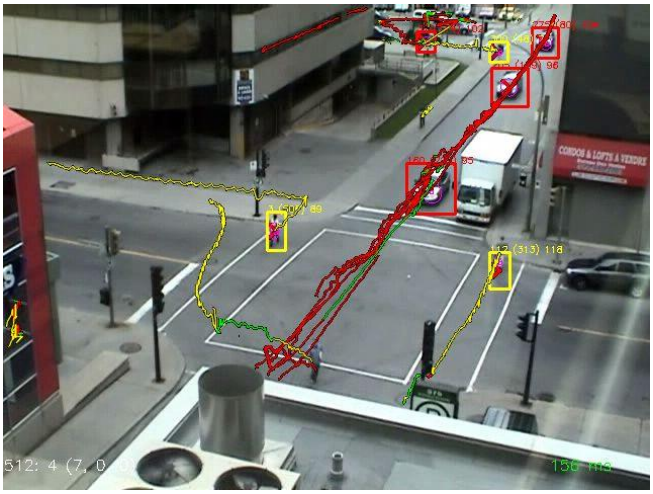

Figura 2.8 - Padrão de atividades. (12)

Kang, Paik, Koschan, B. Abidi e M. Abidi (15) propuseram um método de detecção baseado em um modelamento dinâmico do plano de fundo da imagem. Basicamente, caso um pixel não altere durante N quadros consecutivos, ele é utilizado para atualizar a imagem base do plano de fundo. Baseado nisso, qualquer alteração significativa da imagem pode ser detectada como um objeto em movimento.

Figura 2.9 - Modelamento de plano de fundo com subtração. (15)

Caso sejam utilizadas mais de uma câmera, o autores sugeriram o algoritmo “8-pontos”, que parte de, pelo menos, 4 pontos correspondentes de duas imagens diferentes que serão utilizados para criar uma relação entre as imagens gerando, assim, uma matriz homogênea. No momento em que as câmeras mudarem de posição, elas deverão obter a mesma matriz homogênea para um par de imagens.

Ng e Delp (16) propuseram um método de atualização de modelo de plano de fundo adaptável a cenas dinâmicas. O artigo prevê a implementação de um taxa de aprendizado a cada pixel de acordo com dois parâmetros distintos. O primeiro parâmetro depende da diferença entre as intensidades dos pixels do modelo de plano de fundo e do quadro atual. O segundo depende da duração que o pixel segue classificado como pertencendo ao plano de fundo. Os autores também introduziram um método que detecta mudanças de iluminação repentinas, segmentando os objetos em movimento durante essas mudanças. O objeto em movimento será, então, obtido pelo método de subtração do plano de fundo.

Figura 2.10 - Método de subtração. (16)

Buljan (17) propôs um método de detecção de humanos utilizando câmeras em movimento. Em sua análise, ele dividiu o processo de detecção em 4 etapas: pré-processamento, segmentação, extração de característica e classificação do objeto. A primeira etapa consiste na preparação da imagem a ser processada nas etapas seguintes. Na etapa de segmentação, as bordas dos objetos presentas na imagem são localizados. Buljan utilizou o método de segmentação de visão estérea proposto por Geronimo (18) para executar essa etapa. Na etapa de extração de característica, a imagem é transformada em uma configuração reduzida de características contendo apenas informações relevantes como bordas, cantos, gradientes de intensidade, entre outros. Finalmente, baseado nas etapas anteriores, o objeto é classificado como sendo um ser humano ou não. Buljan concluiu que os melhores métodos para esta etapa são os baseados no aprendizado da própria máquina, como o proposto por Geronimo (18).

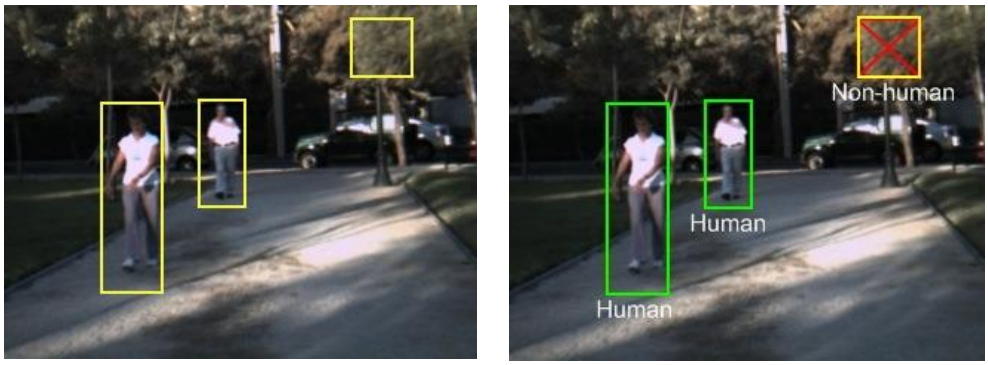

Figura 2.11 - Etapas de segmentação e classificação. (17)

Dutta e Chaudhuri (19) definiram “borda” como sendo a fronteira entre o objeto e o plano de fundo. A partir dessa definição, eles concluíram que, se todas as bordas de uma imagem puderem ser detectadas precisamente, todos os objetos podem ser localizados e suas características mensuradas. No artigo, eles propuseram a detecção das bordas utilizando um algoritmo de detecção baseado em cores (RGB). O método proposto consiste em 4 passos. No primeiro, a imagem passa por um filtro para suprimir ruídos indesejados. Depois, são atribuídos valores para diferentes escalas de cinza (RGB) para cada pixel. No terceiro passo, as imagens são limitadas de acordo com os valores para que, então, as bordas sejam definidas e o resultado refinado.

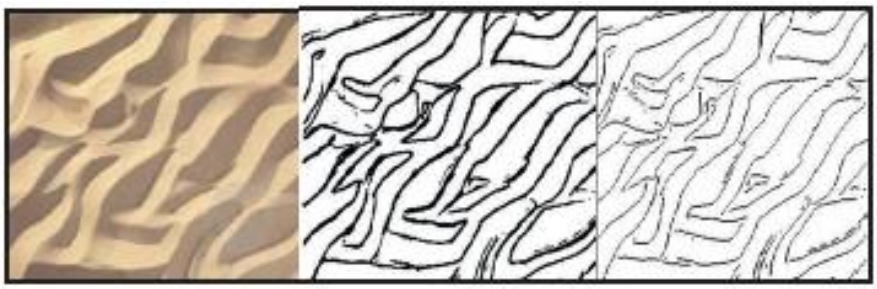

Figura 1.12 - Representação das bordas (meio) e refinamento (dir.). (19)

Pan, Fei, Ni e Zou (20) propuseram o uso de um algoritmo de rápido reconhecimento de padrões de cores baseado em técnicas de correlações quarténias.

O algoritmo de correlação de quarternions foi aplicado a partir do uso de transformadas de Fourier, que contribui para aumentar a eficiência dos algoritmos disponíveis.

Figura 2.13 – Padrão a ser procurado em uma imagem. (20)

Figura 2.14 – Imagem a ser localizado o padrão. (20)

Figura 2.15 – Resultado da detecção. (20)

2.2 Identificação

Gaspar (21) faz o reconhecimento facial usando redes neurais do tipo perceptron multicamadas (MLP). Esse tipo de rede aprende a transformar os dados de entrada na resposta desejada. Suas principais aplicações incluem o reconhecimento de padrões, aproximador universal de funções, processamento de imagens e de sinais e previsão de séries temporais.

Composto por uma ou mais camadas escodidas (hidden layers) a MLP pode implementar qualquer mapeamento de entrada e saída. (21)

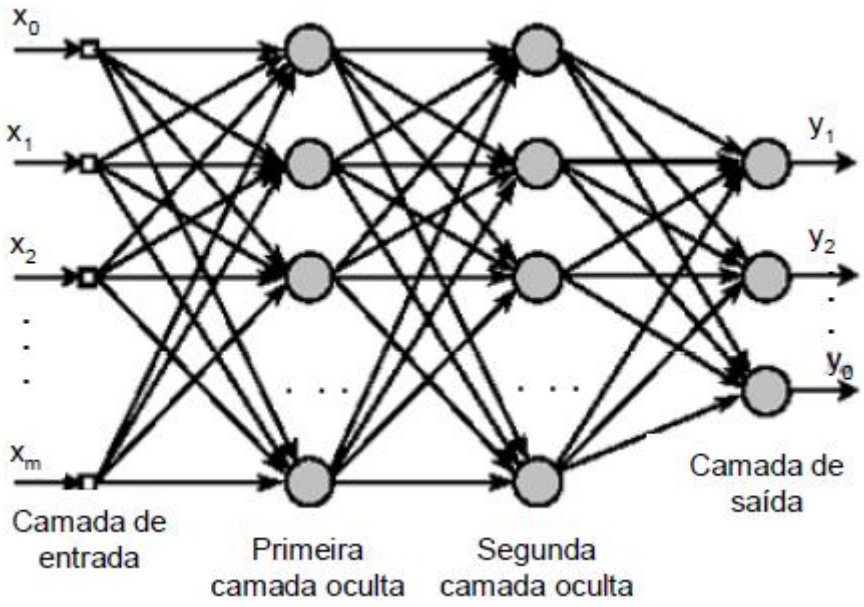

Figura 2.16 – Arquitetura da rede neural multicamadas. (21)

A figura 2.4 mostra uma rede perceptron multicamadas em forma de grafo, que tem o vetor X como entrada, e o vetor Y como saída.

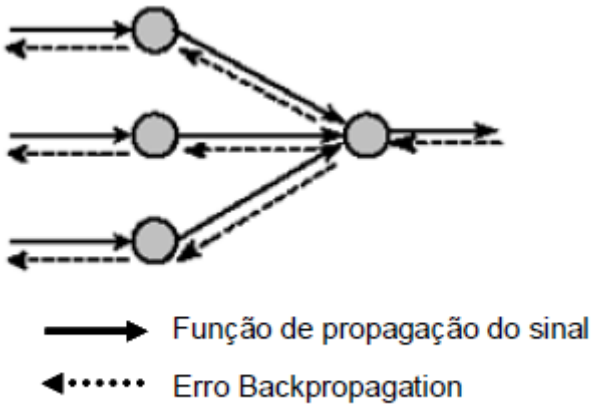

Figura 2.17 – Direção da propagação da função do sinal. (21)

Já a figura 2.5 mostra a direção de dois fluxos de sinal em um MLP. Os sinais funcionais são estímulos recebidos pelos sinais de entrada que se propagam de neurônio para neurônio para frente e manisfesta-se como um sinal de saída. Os sinais de erro são gerados pelos neurônios de saída e se propagam para trás, camada por camada. (21)

O algoritmo de reconhecimento PCA (Principal Componet Analysis) tem como principal característica a capacidade de comprimir dados presentes na imagem, que pode ser feita analisando a variância de cada uma de suas dimensões. O método conserva a dimensão de maior variância, resultando na compressão dos dados sem perdas significativas de informação, o que minimiza o esforço computacional.

O PCA identifca e representa um subespaço do espaço vetorial de uma imagem, utilizando o conceito de auto-valores e auto-vetores, a partir da correlação existente dos valores dos pixels de uma face.

Um conjunto de imagens selecionadas é usado para a criação do espaço de faces. Imagens de faces de usuários são coletadas e projetadas nesse espaço, gerando vetores característicos de cada face, que vão sendo armazenados. Uma amostra de uma face é projetada no espaço de faces para obter um vetor de características que é comparado com os vetores armazenados. A distância da base de vetores da face amostrada com a base de vetores das faces cadastradas é medida e comparado a um limiar, caso seja maior ou igual o usuário é autenticado. (22)

Outra técnica de reconhecimento analisada em (22) é baseada em padrões binários locais (LBP), conhecido também como Transformada Census.

A figura 2.6 mostra o funcionamento do LBP. A cada pixel da imagem, o LBP atribui um rótulo comparando o valor do tom de cinza do pixel com os da vizinhança. Começando pelo pixel acima e à esquerda do pixel central, em um movimento no sentido horário, o valor do tom de cinza do pixel central é comparado com cada um dos seus vizinhos. Caso o valor for maior ou igual ao do pixel central, o pixel é rotulado com valor 1, e 0 caso contrário. Ao alinhar-se os resultados na mesma sequencia em que foram calculados é criado um número na base 2, que é depois convertido para a base decimal e associado ao pixel central.

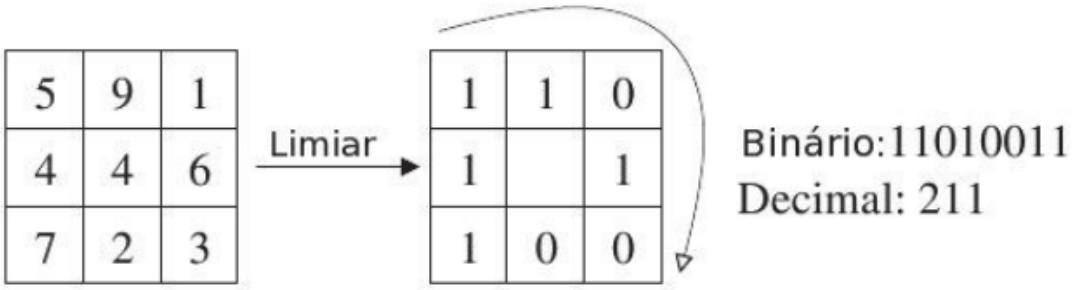

Figura 2.18 – Representação da operação do LBP. (22)

Uma vez calculado o operador LBP para cada pixel, é gerado um histograma. A distribuição de frequências dos valores dos histogramas são usados como descritores faciais. (22)

Kami (23) compara diferentes técnicas de reconhecimento facial a partir de vídeos para determinar as que apresentam um melhor compromisso entre tempo de processamento e precisão.

Os resultados obtidos em (23) mostram que o uso dos classificadores de padrões (Bayesiano, Redes Neurais Artificiais Multicamadas, Redes Neurais Artificiais baseadas em Mapas Auto-Organizáveis de Kohonen, K Vizinhos mais Próximos, Maquinas de Vetores de Suporte, com as variantes sem Kernel, Kernel linear, Kernel RBF (Radial Basis Function) e Kernel Sigmoide, e OPF (Floresta de Caminhos Ótimos)) e do HMM (Hidden Markov Model) apresentaram resultados significativamente melhores.

Kami (23) também notou uma oclusão bastante recorrente, a mão cobrindo parte da face, que interferiu negativamente no desempenho das técnicas estudadas. No caso do reconhecimento facial a partir de imagens estáticas, a chance de o reconhecimento falhar é bastante grande.

Segundo Bezerra (5), o fato de não haver uma norma técnica brasileira para circuitos fechados de televisão (CFTV) responsáveis pela segurança de um ambiente, gera problemas como a baixa resolução e mau posicionamento das câmeras dos circuitos dificultando o reconhecimento dos indivíduos criminosos.

Bezerra (5) também propõem critérios para que se obtenha uma melhor qualidade na captura das imagens. No critério de posicionamento foram avaliadas a altura e o ângulo de trabalho das câmeras. No critério de iluminação foi aconselhado o uso do mesmo tipo de lâmpada em todo o ambiente, e se necessário o uso de câmeras/lentes com recursos especiais (ex: visão noturna, lentes auto íris). No critério de resolução espacial, para que o reconhecimento facial seja funcional é citado que deve-se ter no mínimo 40 pixels na largura da face capturada.

2.3 Rastreamento

Vigus (24) propôs um método, segundo o autor, confiável para rastreamento de trajetórias de objetos em um vídeo a partir do filtro de Kalman e o método de fusão e divisão

de regiões. O artigo (24) afirma que a melhor maneira de se proporcionar o rastreamento de algum objeto em tempo real é antecipando onde o objeto estará no instante de tempo seguinte. A função do filtro de Kalman e de sua característica de busca espiral é justamente minimizar o erro entre a posição calculada para o instante seguinte e a posição em que de fato o objeto se encontrará no ponto seguinte. O método de fusão e divisão de regiões complementará o processo de rastreamento. Uma vez determinada a região a ser rastreada, o programa irá verificar se 80% dos pixels estão de acordo com o critério de homogeneidade pré-programado. Caso contrário, a região sofrerá uma divisão ou uma fusão (figura. 2.8). Esse processo se repete até que seja encontrada a região adequada que se submeterá, então, ao filtro de Kalman.

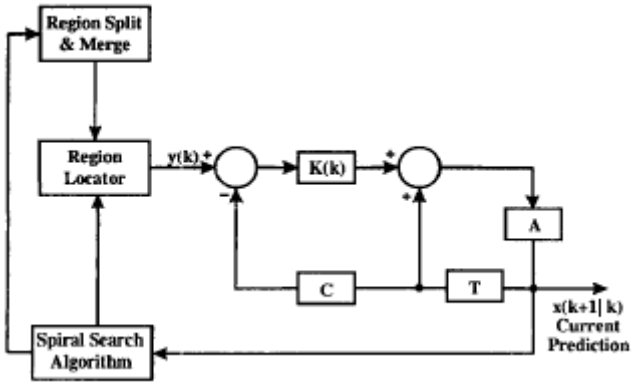

Figura 2.7 – Diagrama de blocos do sistema. (24)

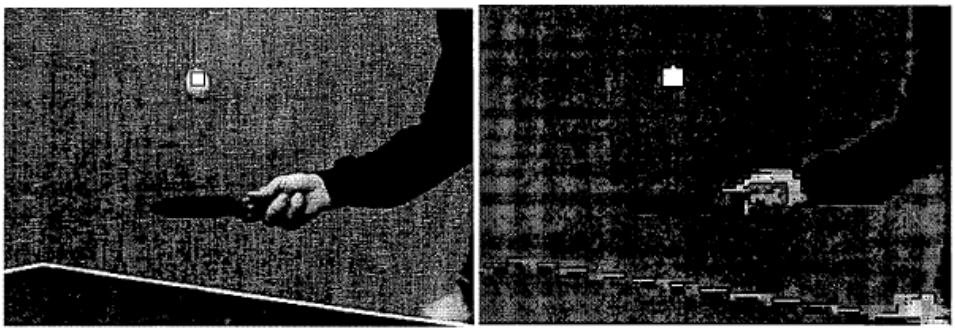

Figura 2.8 – Inicial segmentação (esquerda) e divisão/fusão da região (direita) (24)

Nessa mesma linha, Verma, Schmid e Mikolajczyk (25) escreveram sobre a detecção facial e rastreamento em um vídeo através da determinação da probabilidade de detecção propagada. Eles definem o conceito de rastreamento facial como sendo, em cenas dinâmicas, o acompanhamento da face durante as sequencias de quadros. De acordo com esse artigo, pode-se dividir o rastreamento em duas grandes categorias. A primeira é o

rastreamento da cabeça como um todo, baseado em formas e cores, e a segunda é o rastreamento de características faciais, como movimentos de lábios e olhos.

Verna (25) elaborou um algoritmo de detecção baseado justamente no reconhecimento de características visuais a partir de um banco de dados formado por diferentes atributos visuais, por sua vez, determinados a partir de características visuais marcantes de cada parte do rosto. Probabilisticamente, o algoritmo procura encaixar esses atributos em regiões da imagem escolhidas empiricamente, determinando a chance de cada região selecionada ser, de fato, um rosto. Dessa forma, a probabilidade de detecção é definida para cada posição da imagem em diversas escalas e ângulos.

Para acompanhar a propagação temporal das faces, Verna (25) utilizou uma adaptação do método da Propagação Condicional de Densidade (“Condensation”), cujo algoritmo foi elaborado por Isard e Blake (26). As probabilidades de detecção são propagadas através do tempo utilizando um filtro “Condensation” e prevê a posição da imagem a partir de atualizações geradas pelo mapa de probabilidades produzido ao longo da rotina de detecção.

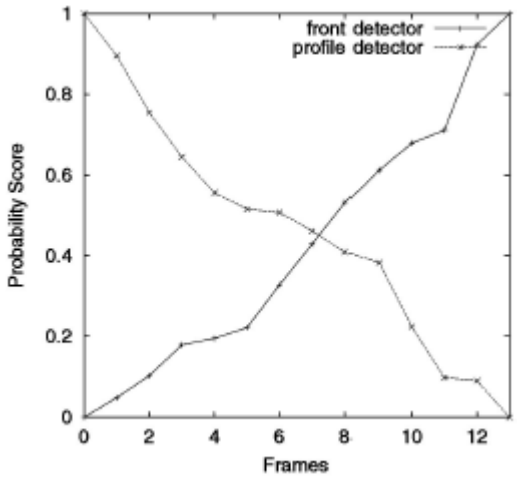

Figura 2.9 – Variação da probabilidade conforme a face é virada. (25)

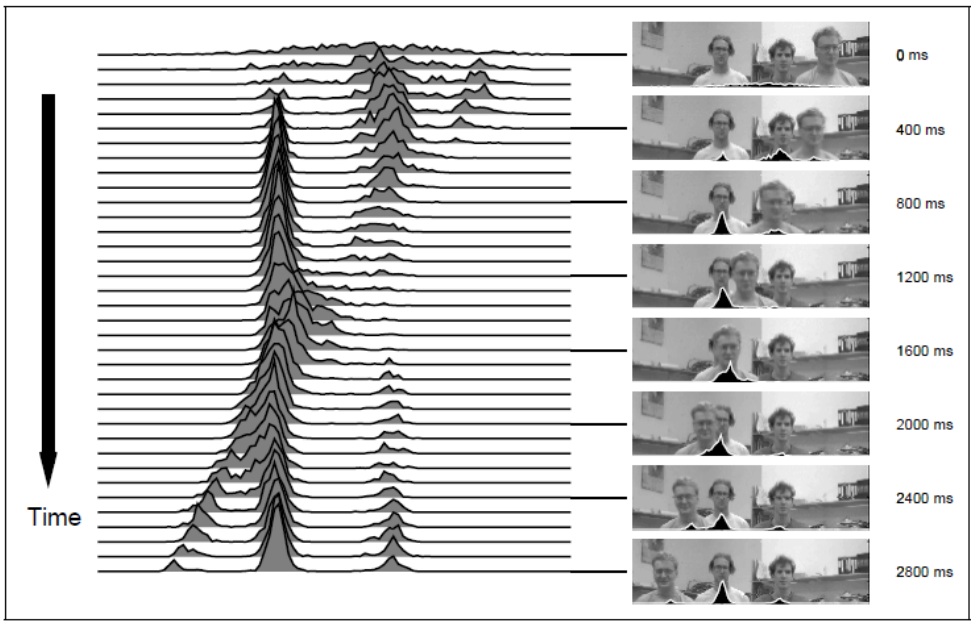

Figura 2.10 – Rastreio de uma distribuuição multi-modal. (26)

Mihaylova, Bull, Angelova e Canagarajah (27) sugeriram um método de rastreamento por meio de redes de comunicação sem fio baseados em medidas recebidas de indicadores de intensidade de sinal. Esse método envolve uma estimativa on-line de posição e variação de velocidade da unidade móvel e é formulado como um problema de estimativa de um sistema híbrido que consiste de um vetor de estado base e um vetor de estado modal. O comando é modelado como um processo Markoviano de primeira ordem e utiliza um filtro sequencial Monte Carlo para minimizar as imprecisões das medidas recebidas.

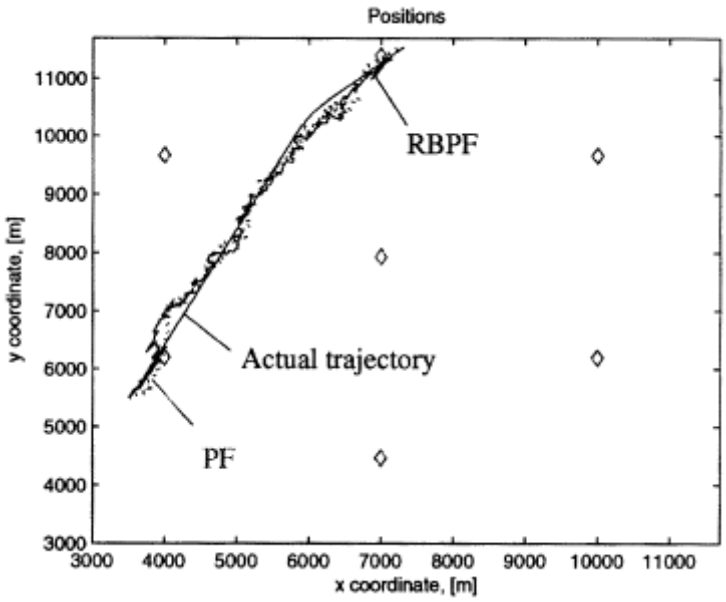

Figura 2.11 – Trajetórial real da unidade móvel e as trajetórias estimadas pelo PF e RBPF. (27)

Dornaika e Orozco (28) propuseram o rastreamento de rostos e características faciais a partir de um modelo computacional 3D. Eles sugerem o uso de modelos de aparência online para rastreamento simultâneo através da detecção da cabeça, dos lábios, das sobrancelhas e das pálpebras, sendo um simples piscar uma grande referência de rastreamento. De acordo com eles, é possível desenvolver algoritmos precisos de rastreamento baseados apenas em um modelo de aparência. Evidentemente, podem ocorrer mudanças de aparência da imagem dos indivíduos rastreados. Para resolver esses casos, eles se basearam dos trabalhos de Ahlberg, Cascia e Matthews (28) que propõem alternativas estatísticas e determinísticas para contornar o problema.

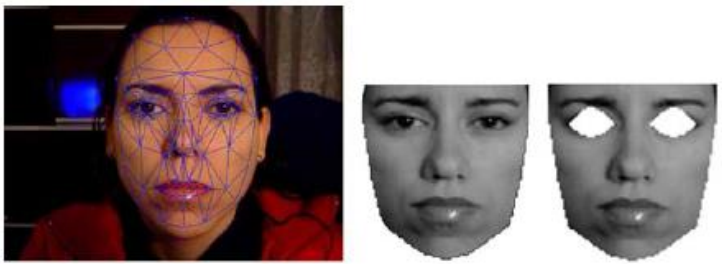

Figura 2.12 – Imagem de entrada com a adaptação correta, extração da face e extração da face sem os olhos. (28)

Para aumentar a precisão do rastreamento das faces, Dornaika e Orozco assumiram o piscar dos olhos como características faciais extremamente marcantes e, uma vez conhecidos a frequência média de piscadas e a detecção dos olhos do indivíduo, podem ajudar a estabilizar os modelos faciais 3D.

Germa e Lerasle (29) apresentaram em seu artigo um modelo de reconhecimento facial e rastreamento aplicado em robôs acompanhantes. Em seu projeto, eles desenvolveram um algoritmo para determinar qual o melhor método de reconhecimento a ser aplicado em cada situação, além de idealizar uma maneira de reconhecer pessoas previamente identificadas unindo características físicas marcantes associadas a ele como sua forma e as cores de sua roupa.

Figura 2.13 – Progresso de uma peer-to-peer H/R sessão de interação. O retângulo representa o template da pessoa-alvo. (29)

Seguindo uma linha parecida, Takala e Pietikäinen (30) propuseram um rastreador em tempo real baseado em informações de cor, textura e movimento. Eles propõem o uso de histogramas de cor RGB (do inglês, red-vermelho, green-verde, blue-azul) para interpretar as cores e padrões locais binários (LBP) para representar as propriedades da textura. A movimentação é levada em conta através de sua trajetória e localização instantânea. Essas medidas são utilizadas para compor o algoritmo de rastreamento.

O trabalho desenvolvido por Passarinho, Salles e Sarcinelli (31) propõe um framework para detecção e seguimento de faces humanas em imagens coloridas sob condições não uniformes de iluminação e diferentes poses da fase. Ao contrário do método anterior, por exemplo, eles não utilizaram modelos estatísticos de cor ou partes deformáveis da face para detectá-la. Este método obtém as respostas de Gabor do recorte da imagem que corresponde à face e as utilizam como atributo a ser seguido, além de utilizar um filtro Kalman modificado para situações de ausência de observações.

Finalmente, Yin, Dimitrios e Velastin (32) elaboraram um artigo sobre avaliação de performance de algoritmos de tracking. Eles usaram seis sequências de vídeos com

diferentes características. A ideia era identificar possíveis fragilidades em cada um dos métodos. Eles compararam dois trackers, o BARCO e o OpenCV sendo que o primeiro levou vantagem na maioria dos critérios. Ainda assim, ficou comprovado pelo artigo que o OpenCV possui maior precisão da estimativa da posição dos objetos em movimento.

3 TÉCNICAS E TECNOLOGIAS

3.1 Tecnologias

3.1.1 JADE

O nome JADE vem de Java Agent DEvelopment Framework, e é um software implementado na linguagem JAVA que simplifica a implementação de sistemas multi-agentes por meio de um midware com ferramentas gráficas que auxiliam a codificação. Um sistema em JADE pode ser utilizado em máquinas cujas configurações serão controladas com uma interface como usuário remoto.

A arquitetura de comunicação possibilita a criação e gerenciamento de uma fila (queue) de mensagens de comunicação de agentes. Agentes podem acessar suas filas via uma combinação de vários módulos. A comunicação é feita segundo o padrão FIPA, com a maior parte dos protocolos de interação já disponíveis e instanciados após a definição do comportamento da aplicação em cada estado do protocolo. (4)

A figura abaixo mostra os principais elementos da arquitetura JADE. Suas aplicações são realizadas através de um conjunto de elementos chamados **Agentes**, cada um com um nome único. Agentes executam tarefas e interagem através de troca de mensagens. Eles estão em **Plataformas** que os providenciam serviços básicos, como o envio de mensagens. Uma Plataforma contém um ou mais **Contêineres.** Contêineres podem ser executados em um ou mais hosts e podem conter zero ou mais agentes. No exemplo da figura, o contêiner “Container 1” no host “Host 3” contém os agentes “A2” e “A3”. Um contêiner especial é o **Contêiner Principal**, que difere dos outros por:

1- Precisar ser o primeiro contêiner da plataforma e todos os outros contêineres são registrados a ele.
2- Incluir 2 agentes especiais: o primeiro é o AMS, que representa a autoridade na plataforma e é o único agente capaz de executar ações de gerenciamento da

plataforma, tais como acionar ou parar agentes ou desligar toda a plataforma. O segundo é o DF (Directory Facilitator), que providencia o serviço de divulgação, onde agentes podem informar o serviço que eles providenciam, e encontrar outros agentes que fornecem os serviços que eles precisam. (4)

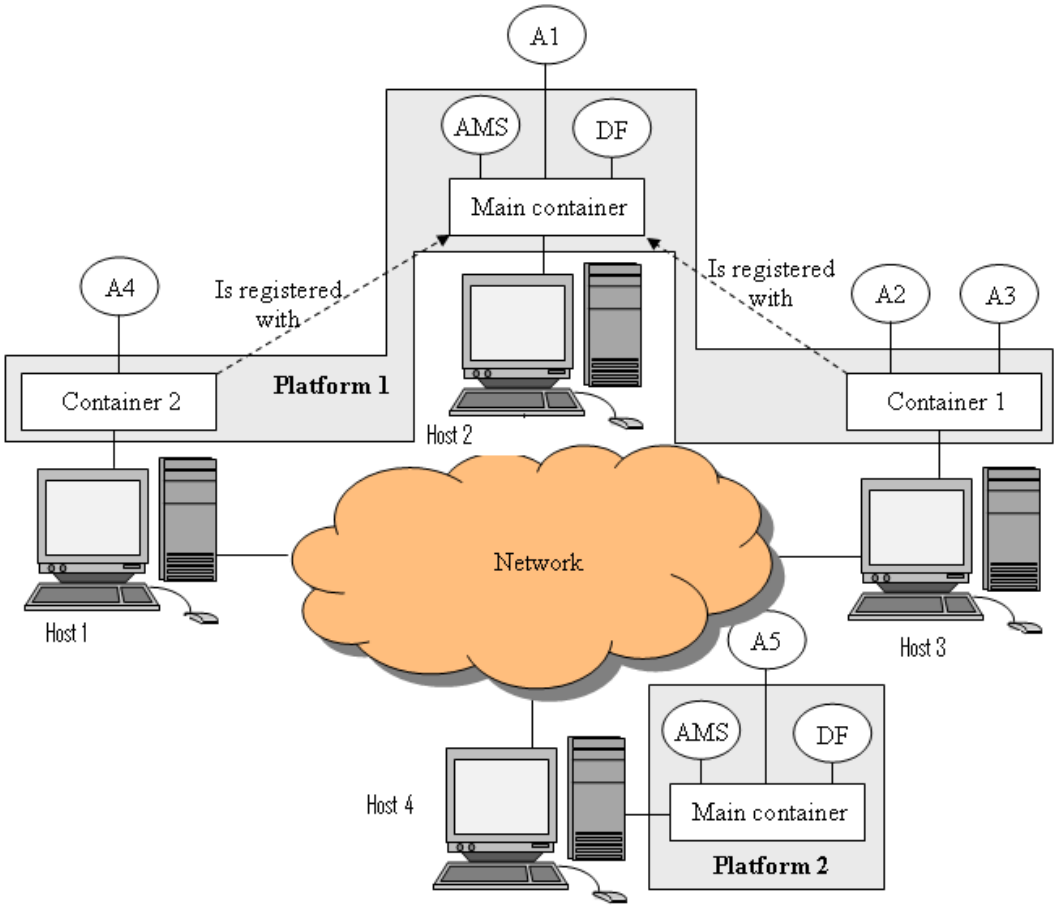

Figura 3.1 – Arquitetura JADE. (4)

3.1.2 OpenCV

OpenCV (Open Source Computer Vision Library) é uma biblioteca de visão computacional e aprendizado de máquina (machine learning). (33)

Possui mais de 2500 algoritmos otimizados, clássicos ou do estado da arte. Esses algoritmos podem ser usados para detectar e reconhecer faces, identificar objetos, classificar ações humanas em vídeos, rastrear o movimento de câmeras, rastrear objetos que se movem, extrair modelos 3D de objetos, juntar imagens produzindo uma imagem de maior resolução de toda a cena, encontrar imagens similares de um banco de dados de imagens, remover o vermelho dos olhos de imagens tiradas com flash, acompanhar o

movimento dos olhos, etc. A biblioteca é usada extensivamente em grupos de pesquisa, empresas e entidades governamentais. (33)

OpenCV possui uma estrutura modular, o que significa que os pacotes (packages) incluem várias bibliotecas compartilhadas ou estáticas. Seus principais módulos são:

- **core** – um modulo compacto que define estruturas básicas, e funções básicas usadas nos outros módulos.
- **imgproc** – um módulo de processamento de imagem que inclui um filtro de imagem, transformação geométrica de imagens (ex: alterar as dimensões, mudar de perspectiva), conversor de cores, histogramas, entre outros.
- **video** – um modulo de analise de vídeo que inclui um algoritmo de rastreamento de objetos.
- **calib3d** – algoritmos básicos de geometria com múltiplas perspectivas, calibração de câmeras simples e estéreas, estimador de posição de objetos, algoritmos de correspondência estérea, elementos de reconstrução 3D.
- **features2d** – detector de saliências, descritores.
- **objdetect** – detecção de objetos e instancias de classes pré-definidas (ex. faces, olhos, pessoas, carros).
- **highgui** – uma interface de captura de vídeo, com codecs de imagens e vídeos.
- **gpu** – algoritmos de aceleração de GPUs de diferentes módulos do OpenCV. (2)

3.1.3 JavaCV

Para se entender a relação entre o OpenCV e o JavaCV deve-se primeiro estudar as tecnologias por trás dessas bibliotecas.

Originalmente, o OpenCV era escrito em linguagem C, ainda que atualmente a biblioteca esteja formulada em C++ com suporte a diversas linguagens de programação através de wappers. JavaCV é um desses wrappers. Dessa forma, pode-se dizer que sempre que se roda um programa com JavaCV estará sendo utilizada também a biblioteca OpenCV, mesmo que chamada em outra interface. (3)

3.2 Técnicas

3.2.1 Classificador Haar em cascata

Este detector de objetos foi proposto inicialmente por Paul Viola e melhorado por Rainer Lienhart. (34)

Um classificador é treinado com algumas centenas de amostras de imagens de um objeto particular (ex: uma face, um carro), chamadas de exemplos positivos, que são escalados para o mesmo tamanho (ex: 40x40), e exemplos negativos (imagens arbitrárias do mesmo tamanho).

Após o treinamento de um classificador, ele pode ser aplicado nas regiões de interesse (do mesmo tamanho usado no treinamento) de uma imagem. O classificador retorna 1 se a região é propensa a ter o objeto, e 0 caso contrário. Para procurar o objeto em toda a imagem, a janela de busca é movida por toda a imagem e a verifica em todas as posições usando o classificador. O classificador é feito de modo que o seu tamanho pode ser facilmente alterado, de modo a ser capaz de encontrar os objetos de interesse em diferentes tamanhos, o que é mais eficiente do que alterar o tamanho da imagem em si. Portanto, para achar um objeto de tamanho desconhecido, o processo de escaneamento é feito diversas vezes em escalas diferentes.

A palavra “cascata” no nome do classificador mostra que o classificador resultante consiste de vários classificadores mais simples que são aplicados na região de interesse até o ponto em que o candidato é rejeitado ou todos os estágios de busca são percorridos. As características de Haar são as entradas para os classificadores básicos, e o algoritmo utiliza:

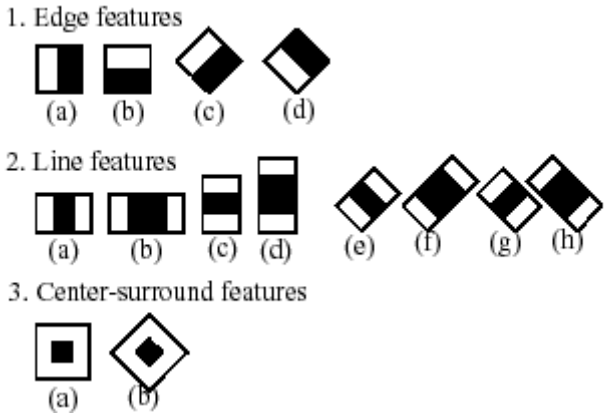

Figura 3.2 – Característcas de Haar

As características usadas em um classificador particular são especificadas pela forma, posição dentro da região de interesse, e a escala. Por exemplo, no caso da terceira característica de linha (item 2c), a resposta é calculada como a diferença entre o número de pixels da imagem sobre o retângulo que cobre toda a característica (incluindo as listras brancas e a listra preta no meio) e o número de pixels da imagem sobre a listra preta multiplicada por 3 para compensar a diferença do tamanho entre as áreas. (34)

3.2.2 Algoritmo Eigenfaces

O algoritmo Eigenfaces é o primeiro método bem sucedido de reconhecimento facial (35) Este método procura captar variações em um banco de dados formado por imagens de rostos humanos e, a partir desta informação, codificar e comparar imagens de faces individuais de maneira holística.

Especificamente, os Eigenfaces são as componentes principais de uma distribuição de faces, ou ainda, os autovetores de uma matriz de covariância do conjunto de imagens faciais, no qual uma imagem com N pixels é considerada um vetor em um espaço N-dimensional (36) com média μ.

Os pares de autovalores e autovetores são ordenados de forma decrescente de acordo com o autovalor para que sejam determinados os k primeiros autovetores para representar o sistema. Esses representam as direções de maior variância dos dados. Forma-se, então, uma matriz A, com as colunas sendo cada um dos k autovetores principais do conjunto para, então, projetar uma imagem de vetor x_{orig} qualquer nesse subespaço para que se possa, então, determinas as componentes principais de seu vetor.

Assim, determina-se qual das imagens de treinamento projetada é a de menor distância para o vetor x_{pca} (I) e esta será a imagem mais próxima dentre as que estão no banco de dados (37).

$$x_{pca} = A^T(x_{orig} - \mu) \quad \text{(I)}$$

3.2.3 Algoritmo Fisherfaces

Diferentemente do método das Eigenfaces, o método das Fisherfaces é específico à classe, ou seja, ele parte do fato de que os dados de treinamento já estão divididos em classes (38)

Sendo X um vetor de c classe no qual cada classe é formada por X_i sendo x_i uma imagem pertencente à classe, μ_i é definida com a média de uma classe e μ como a média entre as classes. Definem-se, então, a matriz de covariância entre as classes SB e a matriz de covariância interclasse S_w. (37)

A matriz de autovalores W_{opt} é escolhida como a que maximiza a razão entre as matrizes de covariância, sendo ela limitada a c - 1, em que c é o número de classes.

A identificação da imagem de teste é análoga ao método das Eigenfaces. A imagem de teste é projetada e comparada com cada uma das faces de treinamento também projetadas, sendo a primeira associada à imagem de treinamento mais próxima.

No caso desse projeto, o método das Fisherfaces foi o que apresentou melhores resultados com menores amostragens de imagens de treinamento.

3.2.4 HOG e Reconhecimento de Cores

Para executar a detecção de corpos em movimento, este trabalho seguiu a proposta de Cosmo, Salles e Ciarelli (39), alinhando o descritor HOG (40) e a técnica CCS (41), além do método de reconhecimento de cores proposto por Stachowicz e Lemke. (42)

HOG é a sigla em inglês para “Histogram of Orieted Gradients”. O método parte do princípio que, em geral, diferentes corpos humanos possuem a mesma forma quando estão em movimento e, portanto, a orientação dos gradientes nos limites corpo/fundo segue um padrão específico (40). Cosmo, Salles e Ciarelli resumiram como ocorre o cálculo do descritor da seguinte maneira: a imagem é dividida em células, cada uma contendo um histograma dos gradientes orientados, calculados em cada canal do modelo de cor RGB, onde o gradiente final é escolhido como sendo o do canal de cor com maior intensidade do gradiente. As células então são unidas em blocos, onde acontece a normalização local dos histogramas. O descritor final é a concatenação de todos os histogramas normalizados pela norma L2, e tem dimensão igual a 3780 (39).

CCS é a sigla em inglês para “Color Self Similarities”. A técnica assume que corpos humanos possuem similaridade em seu padrão de cores. Nesse método, Walk, Majer, Schindler e Schiele propuseram a normalização local dos histogramas, tal qual foi feito no descritor HOG. Através da medida de distância entre os histogramas é possível identificar a similaridade entre as cores em determinada região (41).

O método de reconhecimento de cores propõe que as milhares de tonalidades de cores presentes em determinada imagem sejam reduzidas para apenas 8: vermelho, verde,

azul, ciano, magenta, amarelo, branco e preto. Esse método reduz a complexidade da imagem, mantendo informações suficientes para ainda ser possível identificar todos os elementos da imagem. Essas informações são armazenadas em um vetor de dimensão igual a 8 e guardadas em um banco de dados. Conforme novas imagens são detectadas, elas passam pelo mesmo procedimento e são comparadas com imagens já armazenadas no banco de dados de tal maneira que são medidas as distâncias entre os vetores novos e armazenados. O método retorna a menor dessas distância, indicando o reconhecimento. O método foi citado no artigo (42) como desenvolvido pelo próprio Stachowicz em conjunto com Zhiyu Zhan (43).

3.2.5 Filtro de Kalman

O filtro de Kalman é um método matemático criado por Rudolf Kalman. (44) Neste trabalho, ele foi implementado com o intuito de aumentar a precisão do rastreamento em tempo real.

O filtro de Kalman é composto por um conjunto de equações matemáticas que constitui um processo recursivo eficiente de estimação através da minimização do erro quadrático. Ele é aplicável quando os modelos estão escritos sob a forma espaço-estado. Além disso, o filtro de Kalman permite a estimação dos parâmetros desconhecidos do modelo através da maximização da verossimilhança via decomposição do erro de previsão. (45)

A derivação do filtro de Kalman apoia-se no fato de que tanto os ruídos das equações de medição e transição como o vetor inicial de estado, são normalmente distribuídos. Isto significa que apenas os dois primeiros momentos são suficientes para descrever todos os estados em qualquer instante de t = 1 a t = T. (45)

4 SOLUÇÃO PROPOSTA

Neste capítulo será discutido o protótipo desenvolvido com base nos estudos de projetos similares, das técnicas e tecnologias utilizadas, bem como a análise dos resultados obtidos.

4.1 Visão Geral

O protótipo é composto por duas câmeras PTZ, que enviam imagens de um ambiente para um processador de dados. O processamento da imagem é feito em tempo real e exibido ao usuário por uma interface gráfica.

O processamento da imagem é o uso dos algoritmos estudados e desenvolvidos de reconhecimento facial e rastreamento de pessoas, e a automação das funções da câmera. Um banco de dados é usado para armazenar imagens para auxiliar o processo.

O rastreamento de pessoas é feito utilizando as duas câmeras, integradas no mesmo sistema, de modo que uma pessoa rastreada por uma câmera, ao sair da área coberta pela primeira câmera, deve continuar sendo rastreada ao entrar na área coberta pela segunda câmera.

4.2 Diagramas de Casos de Uso

Os diagramas de casos de uso abaixo mostram as principais ações que o sistema a ser desenvolvido pode executar, bem como os agentes envolvidos.

Como mostrado na figura 5.1, o usuário pode treinar o sistema para o reconhecimento de pessoas. Esse treinamento é feito através de imagens de pessoas conhecidas armazenadas em um banco de dados. O usuário também pode movimentar as câmeras do sistema manualmente, de modo a focar uma região de interesse. A imagem que sai como resultado dos dados colhidos e processados pode ser visualizada através de uma interface gráfica. Nessa imagem há a imagem recebida da câmera, junto com um indicador dos resultados dos algoritmos de reconhecimento (nome do indivíduo no sistema) e de rastreamento (um indicador que acompanha a pessoa enquanto ela se movimenta no vídeo).

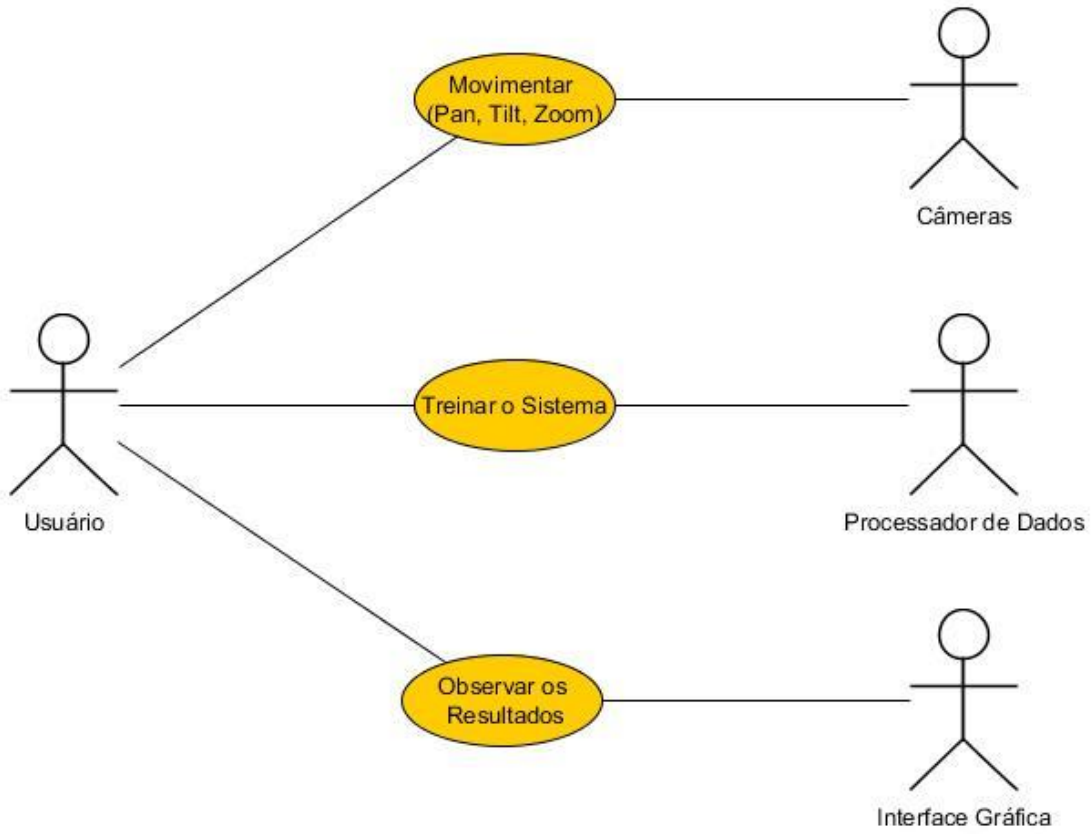

Figura 4.1 – Casos de uso do usuário

O diagrama de casos de uso do processador de dados é mostrado na figura 5.2. O processador de dados é responsável pelas ações automatizadas do sistema. As imagens são recebidas pelas câmeras, processadas e alteradas de modo a mostrar os resultados desejados (reconhecimento e rastreamento), e enviadas para a interface gráfica. As funções das câmeras (pan, tilt e zoom) estão automatizadas para que sejam colhidas melhores imagens para o sistema, de modo a aperfeiçoar os resultados obtidos.

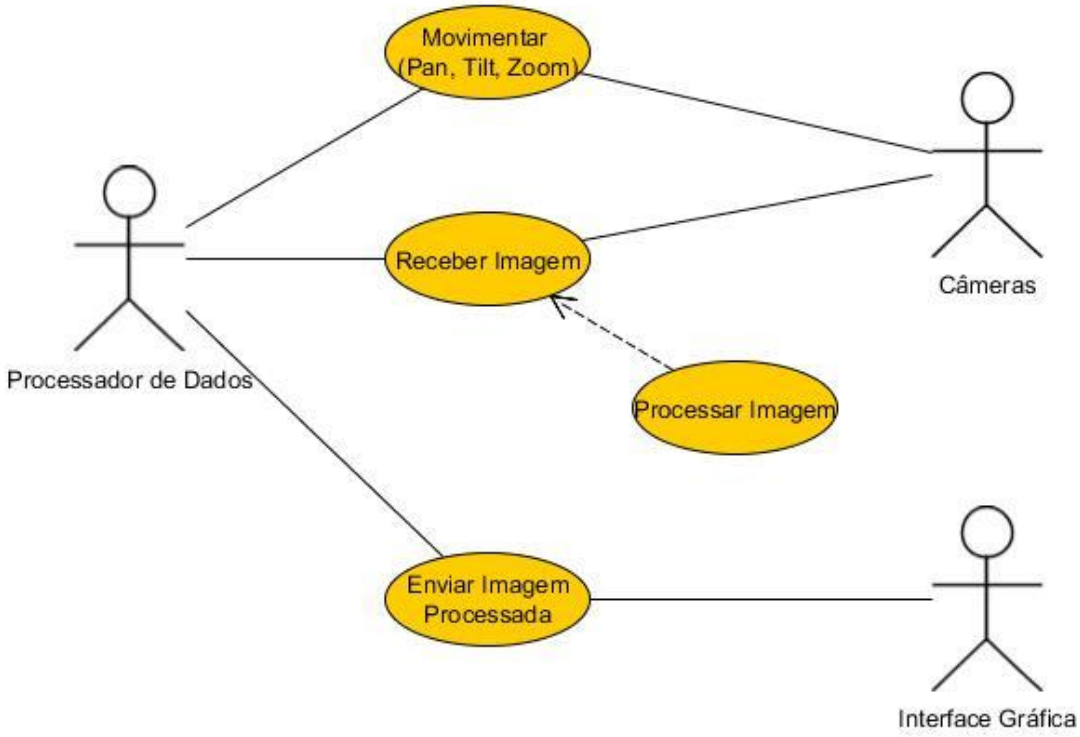

Figura 4.2 – Casos de uso do processador de dados

4.3 Detalhamento do Projeto

O projeto utiliza dois modelos de câmera PTZ. O primeiro modelo utilizado é o NIP-09L2J 3X Opital Zoom da NEO Coolcam (46), desenhado para vídeos de sistemas de vigilância em ambientes interno e externos. Possui conexão sem fio, mecanismo de rotação horizontal de 350º e vertical de 120º e lente com zoom de até 3x. O segundo modelo é o NIP-OZX (47) que também possui conexão sem fio e mesmo grau de rotação vertical e horizontal, mas que não possui zoom. Ambos são P2P IP.

Figura 4.3a: Modelo NIP-09L2J (46)

Figura 4.3b: Modelo NIP-09L2J (46)

Figura 2.4: Modelo NIP-OZX (47)

A interface gráfica e o processamento dos dados foram feitos usando um computador doméstico com processador Intel® Core™ i7-4510U CPU @ 2.00GHz 2.60GHz, com 16,0GB de memória.

Os programas utilizados são: um programa de banco de dados (MySQL) para armazenar e dar acesso a informações importantes para o sistema, ferramenta JADE (versão 4.3.3) que possibilita o processamento assíncrono das funções do protótipo, biblioteca de códigos JavaCV 0.7, compatível com o OpenCV 2.4.8.

O diagrama de componentes da figura 5.3 mostra a organização do projeto, baseada no trabalho de conclusão de curso desenvolvido em 2014 (37). Novos agentes e pacotes

foram adicionados ao projeto existente conforme a necessidade, de modo a alcançar os objetivos deste trabalho.

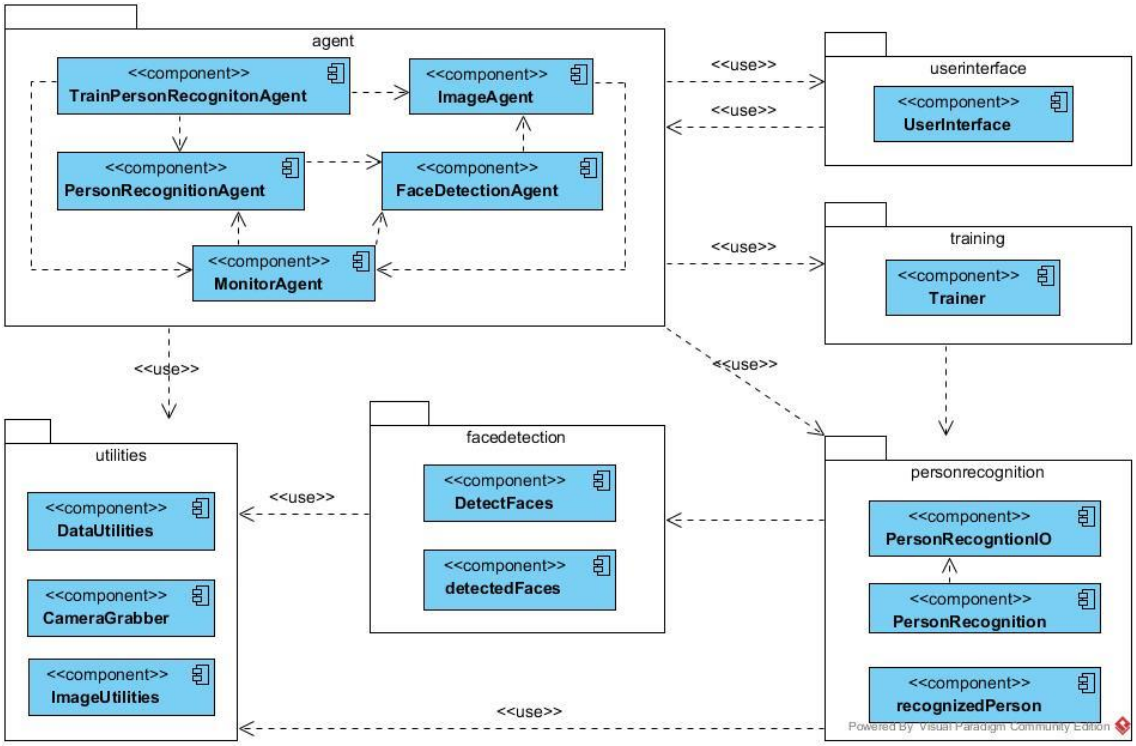

Figura 4.3 – Diagrama de componentes do sistema

Cada componente corresponde a uma classe em Java, as relações representam o uso de funções de outras classes. No caso de agentes, essas relações representam o processo de envio de mensagem. Cada agente é responsável por determinada função, executando-as através do acesso aos pacotes de funções, além de serem capazes de comunicarem entre si. Abaixo uma descrição de cada agente:

ImageAgent - responsável pela obtenção das imagens, portanto, ele é o responsável para fazer a interação com as câmeras.

FaceDetectionAgent – responsável por encontrar um rosto de uma pessoa na imagem fornecida pela câmera. A utilização dessas informações é importante para o processo de reconhecimento da pessoa e o seu rastreamento, já que as principais informações de uma pessoa no sistema são definidas pelo seu rosto.

PersonRecognitonAgent – implementa o algoritmo de reconhecimento de pessoas. O resultado obtido depende das imagens utilizadas para o treinamento do reconhecedor.

TrainPersonRecognitionAgent – responsável por treinar o reconhecedor.

MonitorAgent – responsável pela interface gráfica com o usuário. Exibe os resultados em tempo real das imagens capturadas pelas câmeras após o seus processamentos.

Os pacotes de funções, que contém os algoritmos usados pelos agentes, são descritos abaixo:

facedetection - contém o algoritmo de classificadores Haar em cascata, responsável por detectar rostos ou outros objetos importantes em imagens.

personrecognition - contém o algoritmo de reconhecimento de pessoas, além de guardar os resultados obtidos.

training - contém as funções de treinamento do sistema.

userinterface - contém as funções necessárias para fazer a interface com o usuário

utilities - contém diversos tipos de funções: tratamento de imagens, tratamento da estrutura de dados do sistema, interface de obtenção de dados da câmera.

O algoritmo de rastreamento desenvolvido utiliza de cálculos de probabilidades para determinar, caso o algoritmo de detecção não encontre a pessoa, se ela continua ou não na imagem. Isso é importante, pois durante o processo de detecção, alguns dos frames capturados pela câmera podem não reconhecer um indivíduo, mesmo que ele esteja parado, ou caso ele vire de costa, por exemplo.

O rastreamento está acoplado ao sistema mostrado no diagrama de componentes da imagem 5.3 e foi desenvolvido baseado no estudo de projetos similares publicados em teses e artigos científicos além de outras fontes auxiliares conforme a necessidade.

5 RESULTADOS

5.1 Detecção

O protótipo criado detecta faces em uma imagem e desenha retângulos ao redor delas utilizando o Classificador de Haar em Cascata. A estrutura do programa é mostrada na figura 5.1.

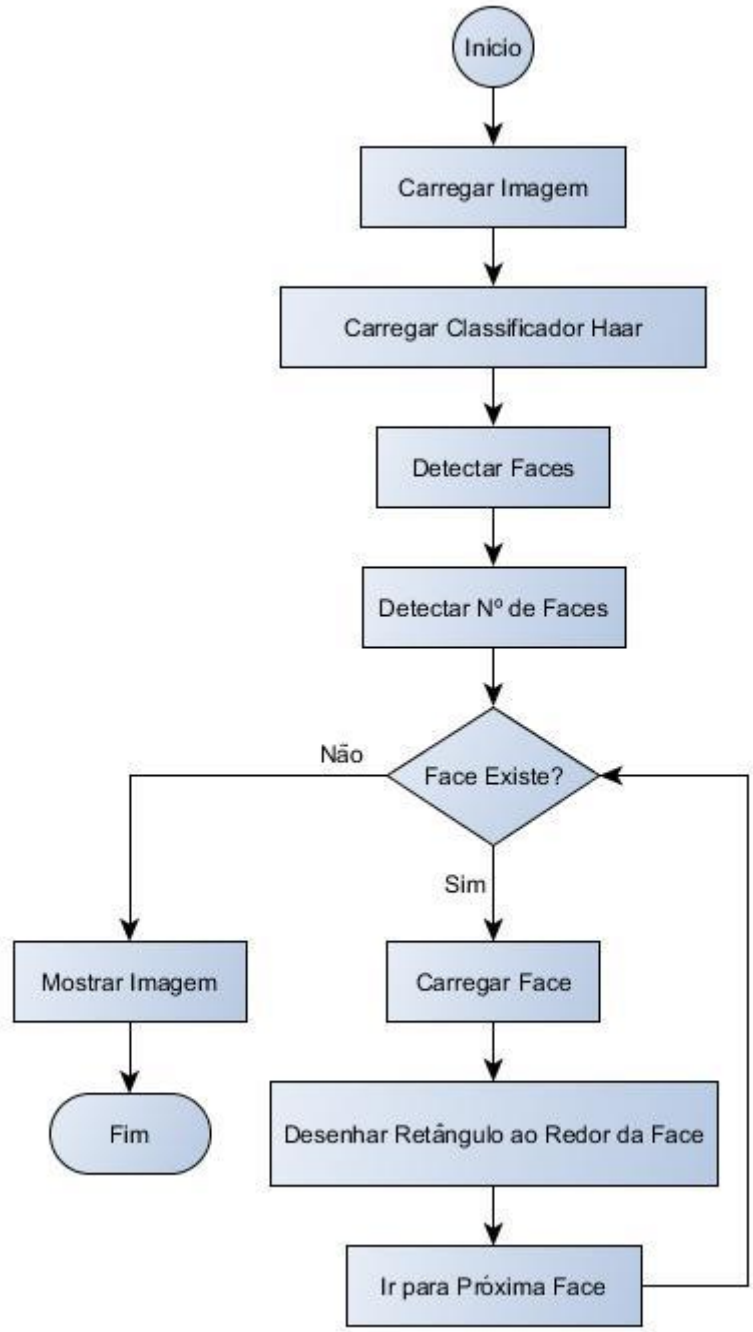

Figura 5.1 - Diagrama de atividades do protótipo de detecção de faces

Ao selecionar uma imagem para ser feita a detecção de faces, é carregado o classificador de Haar que identifica rostos frontais de pessoas, que é utilizado na imagem, gerando como resultado o número de rostos encontrados. Para cada rosto é desenhado um retângulo ao seu redor, a ser mostrado junto com a imagem inicial. A figura 5.2 mostra um dos resultados obtidos.

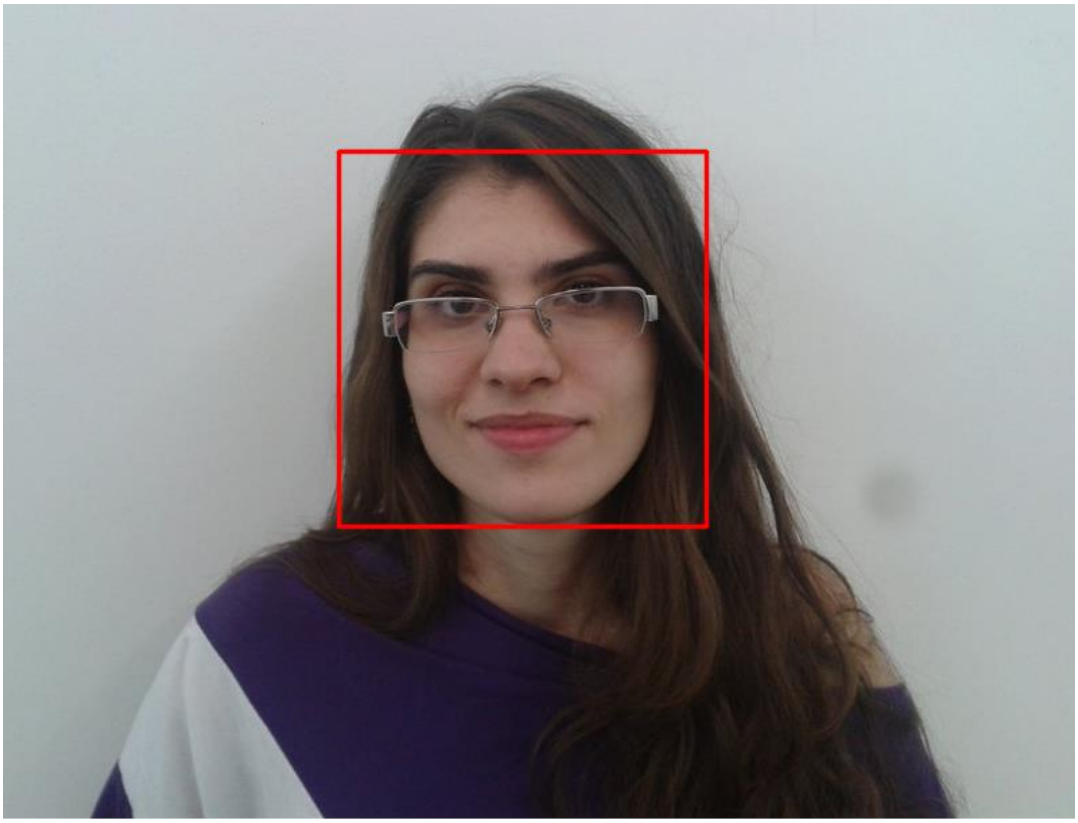

Figura 5.2 – Resultado da detecção de faces

Ao analisar os resultados obtidos observou-se que o algoritmo obteve resultados satisfatórios na detecção de faces frontais. Porém ele é sensível a inclinações da face, devido à natureza do algoritmo, com erros acentuados a partir de aproximadamente 15º de inclinação. Rostos de perfil não são reconhecidos pois não existe um classificador de Haar para este tipo de rosto.

Outro fato a ser mencionado é que o algoritmo procura na imagem qualquer tipo de padrão que se assemelhe a um rosto humano (olhos, nariz e boca), e certas formas de objetos, bem como condições de luminosidade, podem levar a falsos positivos, ou seja, detectar um rosto inexistente na imagem.

5.2 Reconhecimento em fotos

O reconhecimento de pessoas foi feito utilizando o algoritmo fisherfaces. Esse algoritmo necessita de um treinamento com imagens de pessoas conhecidas antes de ser usado. Para realizar o treinamento, deve-se ter o mesmo número de amostras para cada indivíduo e as imagens devem ter a mesma dimensão de pixels.

Utilizando o banco de dados Yale Face (48), as imagens utilizadas para o treinamento são mostradas na figura 5.3. A primeira linha são fotos do indivíduo identificado como “Subject 1” pelo sistema, a segunda linha, do “Subject 2”, a terceira, “Subject 3”.

Figura 5.3 – Imagens utlizadas para o treinamento do reconhecedor

A estrutura do protótipo é mostrada na figura 5.4.

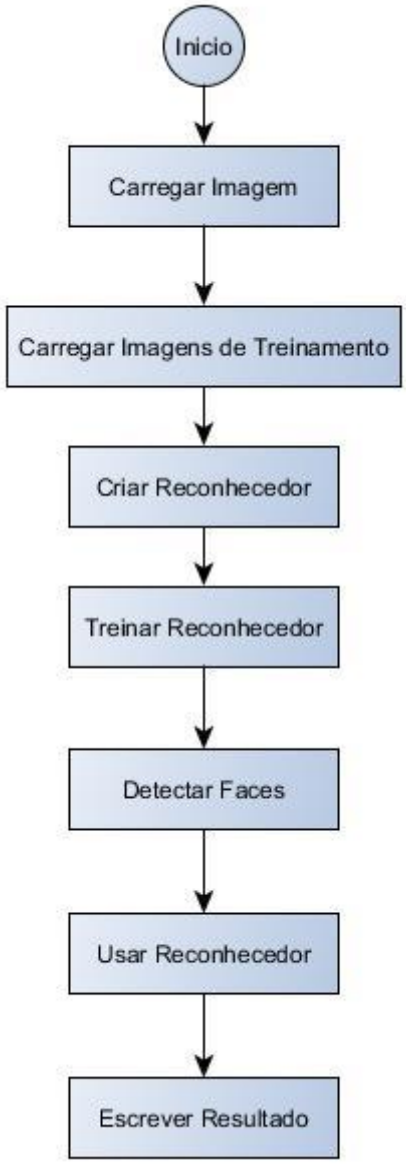

Figura 5.4 – Diagrama de atividades do protótipo de reconhecimento em fotos

Após escolher a imagem a ser usada pelo algoritmo de reconhecimento, é efetuado o treinamento do reconhecedor. Rostos são detectados na imagem utilizando o algoritmo de detecção de faces. Em cada rosto é feito o processo de reconhecimento, ou seja, é obtido o resultado de identificação dos indivíduos pelo sistema. Um retângulo e uma string associada ao resultado da identificação são desenhados em cada rosto na imagem original e exibidos para o usuário. Alguns dos resultados obtidos são mostrados na figura 5.5.

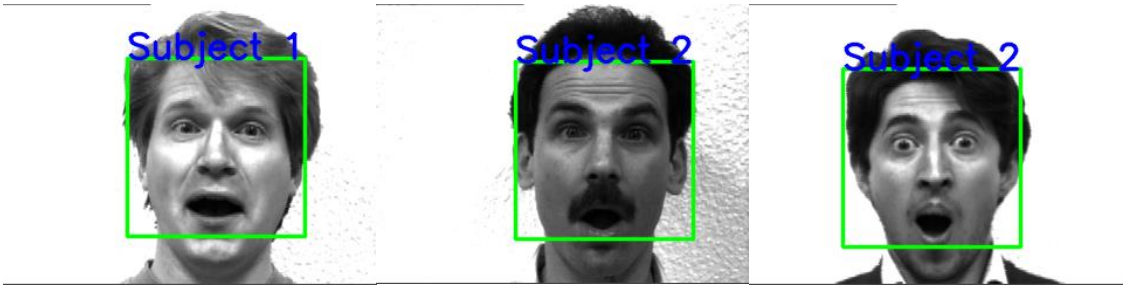

Figura 5.5 – Resultados do reconhecimento em fotos

Analisando os resultados acima, percebe-se que, apesar da pequena quantidade de imagens para o treinamento, os indivíduos 1 e 2 foram reconhecidos com sucesso. Porém o indivíduo 3 foi reconhecido pelo sistema erroneamente como o indivíduo 2. De modo corrigir esse erro, pode ser feito uma calibração do “limiar de decisão” do algoritmo de reconhecimento, o que resultaria no indivíduo 3 não sendo mais confundido como o indivíduo 2, porém o sistema ainda não conseguiria reconhecê-lo, pois o resultado seria dado como indivíduo não reconhecido. Outra abordagem a ser tomada seria aumentar o número de imagens de treinamento, e/ou a alteração das imagens de treinamento por outras que tenham as diferenças visuais entre os indivíduos em maior evidência.

Durante o trabalho como um todo, foi verificado que o protótipo obtém resultados satisfatórios caso o indivíduo a ser testado está submetido às mesmas condições de iluminação e na mesma posição angular em relação à câmera quando comparo às imagens de treinamento. Mudanças nessas condições e na aparência da pessoa (uso de óculos, mudança do penteado e dos pelos faciais) impactaram negativamente de modo acentuado nos resultados do protótipo.

5.3 Reconhecimento em vídeo

O programa desenvolvido utiliza a câmera do computador para filmagem. A cada frame o programa detecta as faces presentes no frame e prediz o reconhecimento das faces.

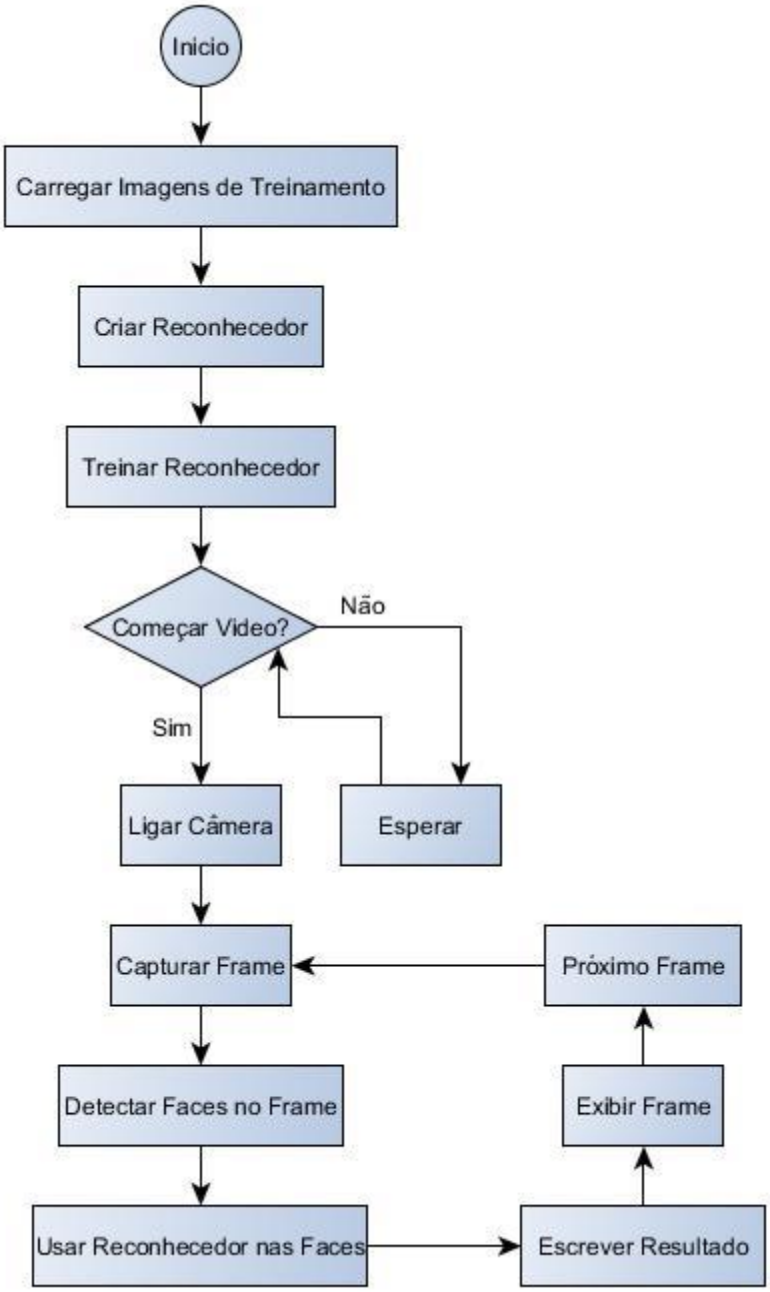

Figura 5.6: Diagrama de atividades do protótipo de reconhecimento em vídeos

Os resultados do reconhecimento em vídeo mostraram-se piores que os de reconhecimento em imagens. Isto se deve a qualidade das imagens obtidas pela câmera de computador, muito inferior a da câmera usada para obtenção das imagens, somado ao fato que a imagem é tomada por um ângulo diferente do usado para o treinamento do reconhecedor.

5.4 Rastreamento

Abaixo são mostradas imagens do sistema de rastreamento implementado. O retângulo vermelho representa os dados recebido pelo reconhecedor Haar de faces, enquanto o retângulo verde é o estado do filtro de Kalman, ponderado entre as medidas observadas do reconhecedor de faces, e do modelo dinâmico adotado.

Figura 5.6: Resultado do rastreamento em diferentes frames

O modelo adotado tem como variáveis de estado a posição de um dos vértices do retângulo, sua velocidade e suas dimensões. Foi adotada a hipótese de que a velocidade entre cada frame não apresenta grandes variações, sendo considerada constante e reavaliada a cada frame segundo a diferença da posição do retângulo.

Observa-se que os dois retângulos são praticamente congruentes em quase todos os frames. Isso mostra a influência do sensor sobre a atualização dos estados.

5.5 Detecção de corpos

Utilizando o HOG para detecção de corpos em uma imagem de pedestres foi obtido o resultado abaixo.

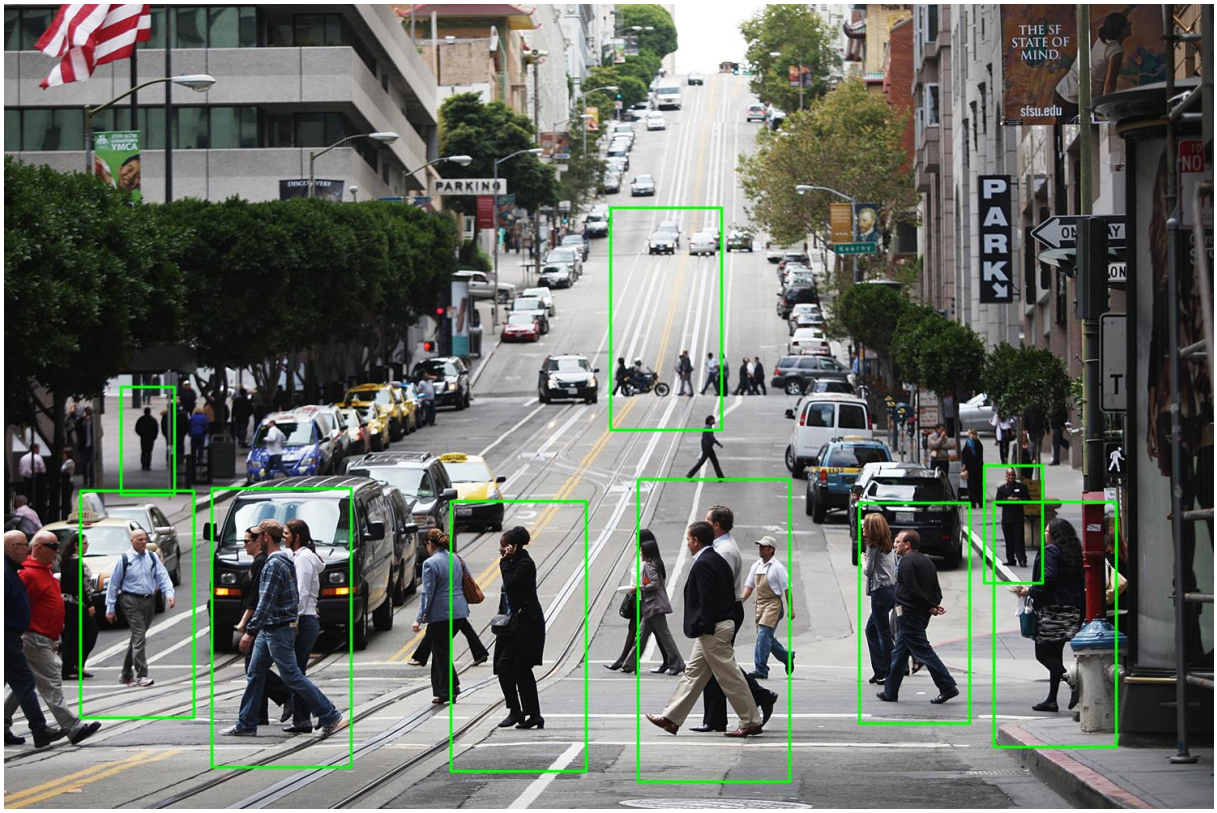

Figura 5.7: Método HOG aplicado em uma imagem de pedestres

Observa-se que o detector localizou a maioria dos pedestres mais próximos da imagem. Para pessoas próximas umas das outras, o detector não conseguiu distinguir que havia mais de uma pessoa na região e acabou agrupando-as em um único retângulo. Isso se deve a própria natureza do identificador, que agrupa automaticamente detecções muito próximas, pois uma pessoa normalmente apresenta várias detecções de diferentes características, e o identificador suaviza esses resultados agrupando eles.

6 CONCLUSÃO

Durante o desenvolvimento deste projeto, percebeu-se que os algoritmos de detecção de faces mostraram-se bem sólidos quando se conhece a orientação da face a ser testada, embora alguns falsos positivos acabem se manifestando. Os algoritmos de reconhecimento de pessoas deixaram a desejar no quesito robustez, pois eles são fortemente influenciados pelas condições das imagens de teste (iluminação, ângulo da câmera, posição e orientação do rosto, entre outros), mostrando resultados satisfatórios apenas quando o teste é feito em condições muito similares a das imagens utilizadas em seu treinamento, não sendo capaz de identificar uma pessoa quando ela, por exemplo, cobre parte do rosto com a mão.

Devido à natureza da forma de captação de imagens que este projeto se dispõe (câmeras de segurança), é impossível captar com precisão uma característica que é única e pode definir quem é um indivíduo (ex: impressão digital, íris). Erros de reconhecimento eram esperados, mas foram atenuados com o uso das funções da câmera (pan,tilt e zoom) de modo automatizado para obter melhores imagens: caso o programa detectasse um rosto mas não conseguisse identifica-lo, ainda poderia focá-lo e aproximá-lo através do PTZ.

O algoritmo de rastreamento desenvolvido apresentou resultados satisfatórios. Sua implementação em conjunto com o algoritmo de detecção de faces foi aprimorada através do uso de funções de probabilidade em imagens retiradas em tempo real pela câmera, ainda que tenha, eventualmente, resultado em falsos positivos em imagens estáticas.

Para auxiliar o rastreamento, foi implementado outro classificador Haar em cascata, desta vez de corpos humanos, em conjunto com o primeiro, sendo possível assim rastrear uma pessoa mesmo se ela virar de costas durante a filmagem.

7 REFERÊNCIAS

1. IFR International Federation of Robotics. [Online] 2015. http://www.ifr.org/industrial-robots/statistics/.

2. OpenCV Documentation. [Online] [Citado em: 16 de Junho de 2015.] http://docs.opencv.org/.

3. JavaCV. [Online] [Citado em: 20 de Junho de 2015.] https://github.com/bytedeco/javacv.

4. JADE. [Online] JADE, 16 de Junho de 2015. http://jade.tilab.com/.

5. *Proposta de Critérios para Câmeras de Vigilância em Aplicações de CFTV Indoor para Fins de Identificação Forense de Suspeitos.* **Bezerra, Rodrigo Albernaz.** Brasília : s.n., 2012.

6. *A Robust Skin Color Based Face Detection Algorithm.* **Singh, Sanjay Kr., et al., et al.** 2003, Tamkang Journal of Sciece and Engineering, Vol. 6, pp. 227-234.

7. **BASKAN, S., BULUT, M.M. e VOLKAN, A.** Projection based method for segmentation of human face and its evaluation. *Pattern Recognition Letters.* 2002, Vol. 23, 1623-1629.

8. **GU, H., SU, G. e DU, C.** FeaturePoints Extraction from Faces. *Image and Vision Computing.* 2003, 154-158.

9. **VIOLA, P. e JONES, M.J.** Robust Real-Time Face Detection. *International Journal of Computer Vision.* 2, 2004, Vol. 57, p 137-154.

10. **BRANDAO, A., FERNANDES, L. e CLUA, E.** A Method for Body Part Detection and Tracking using RGB-D Images. 2014.

11. **Kadim, Z., et al., et al.** Method to Detect and Track Moving Object in Non-static PTZ Camera. *Proceedings of the International MultiConference of Engineers and Computer Scientists 2013.* 2013, Vol. 1.

12. **LALONDE, M., FOUCHER, S., GAGNON, L., PRONOVOST, E., DERENNE, M., JANELLE, A.** *A system to automatically track humans and vehicles with a PTZ camera, Canada, 2007.*

13. **STAUFFER, C., GRIMSON, W. E.** *Learning patterns of activity using real-time tracking", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 22, No. 8, pp. 747-757, 2000.*

14. **JAVED, O., RASHEED, Z., ALATAS, O., SHAH, M. KnightM.** *A Real Time Surveillance System for Multiple Overlapping and Non-Overlapping Cameras, IEEE conf. on Multimedia and Expo, Special Session on Multi-Camera Surveillance Systems, Baltimore, 2003.*

15. **KANG, S., PAIK, J., KOCHAN, A., ABIDI, B., ABIDI, M. A.** *Real-time video tracking using PTZ cameras, Proc. of SPIE 6th International Conference on Quality Control by Artificial Vision, Vol. 5132, pp. 103-111, Gatlinburg, 2003.*

16. **NG, K. K., DELP, E. J.** *Background subtraction using a pixel-wise adaptive learning rate for object tracking initialization, School of Electrical and Computer Engineering, Indiana.*

17. **BULJAN, S.A.M.** *Human detection using uncontrolled moving cameras and novel features derived from a visual saliency mechanism, Santiago do Chile, 2008.*

18. **GERONIMO D., LOPEZ, A., SAPPA A.** *Computer vision approaches to pedestriandetection: Visible spectrum survey. LNCS, Pattern Recognition and Image Analysis, 547–554.*

19. **DUTTA, S., CHAUDHURI B. B.** *Color Edge Detection Algorithm in RGB Color Space, International Conference on Advances in Recent Technologies in Communication and Computing 978-0-7695-3845-7/09, 2009.*

20. **PAN, J., FEI, S., NI, Y., ZOU, W.** *Color Pattern Recognition Using Fast Quaternion Correlation Algorithm.*

21. *Reconhecimento de Faces Humanas Usando Redes Neurais MLP.* **Gaspar, Thiago Lombardi.** São Carlos : s.n., 2006.

22. *BioMobile: Sistema de Identificação de Usuários em Dispositivos Móveis na Plataforma Android Utilizando Reconhecimento de Faces a Partir de Vídeo.* **Farina, André Marcelo.** São José do Rio Preto : s.n., Fevereiro de 2012.

23. *Análise de Técnicas de Reconhecimento de Padrões para a Identificação Biométrica de Usuários em Aplicações WEB Utilizando Faces a Partir de Vídeos.* **Kami, Guilherme José da Costa.** Bauru : s.n., Agosto de 2011.

24. *Video Object Tracking Using Region Split and Merge and a Kalman Filter Tracking Algorithm.* **Vigus, S A, Bull, D R e Canagarajah, C N.** Bristol : s.n., 2001, IEEE, Vol. 1, pp. 650-653.

25. *Face Detection and Tracking in a Video by Propagating Detection Probabilities.* **Verma, R G, Schmid, C e Mikolajczyk, K.** 2003, IEEE, Vol. 25, pp. 1215-1228.

26. *Contour Tracking by Stochastic Propagation of Conditional Density.* **Isard, M e Blake, A.** Cambridge : s.n., 1996, In Proc. European Conf. Computer Vision, pp. 343-356.

27. *Mobility Tracking in Cellular Networks with Sequential Monte Carlo Filters.* **Mihaylova, L S, Bull, D R e Canagarajah, C N.** Philadelphia : s.n., 2005, IEEE, Vol. 1, pp. 107-114.

28. *Real time 3D Face and Facial Feature Tracking.* **Dornaika, F e Orozco, J.** 2007, J Real Time Image Proc, Vol. 2, pp. 35-44.

29. *Video-Based Face Recognition and Tracking form a Robot Companion.* **Germa, T e Lerasle, F.** Toulouse : s.n., 2009, International Journal of Pattern Recognition, Vol. 23, pp. 591-616.

30. *Multi-Object Tracking Using Color, Texture and Motion.* **Takala, V e Pietikäinen, M.** Oulu : s.n., Infotech Oulu and Dept. of Electrical and Information Engineering.

31. *Face Tracking in Unconstrained Color Videos with the Recovery of the Location of Lost Faces.* **Passarinho, C J P, Salles, E O T e Filho, M S.** 2015, IEEE, Vol. 13, pp. 307-314.

32. *Performace Evaluation of Object Tracking Algorithms.* **Yin, F, Makris, D e Velastin, S.** Kingston : s.n., Digital Imaging Research Centre.

33. *OpenCV.* [Online] [Citado em: 16 de Junho de 2015.] http://opencv.org/about.html.

34. [Online] [Citado em: 16 de Junho de 2015.] http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html.

35. **SEO, N.** Eigenfaces and Fisherfaces. *University of Maryland.* ENEE633 Pattern Recognition Project 2-1, Project 2-1.

36. **TURK e SHENG.** [Online] [Citado em: 19 de 06 de 2015.] http://www.scholarpedia.org/article/Eigenfaces.

37. *Sistema de Identificação de Emoções por Expressões Faciais com Operação ao Vivo.* **Tinen, Bruno.** São Paulo : s.n., 2014.

38. *Estudo comparativo de algoritmos para reconhecimento facial.* **KINUTA, C., et al., et al.** São Caetano do Sul : Universidade IMES.

39. **COSMO, D.L., SALLES, E.O.T e CIARELLI, P.M.** Pedestrian Detection Utilizing Gradient Orientation Histograms and Color Self Similarities Descriptors. *IEEE Latin America Transactions.* 7, 2015, Vol. 13.

40. **DALAL, N. e TRIGGS, B.** Histograms of Oriented Gradients for Human Detection. *INRIA Rhone-Alps.* Montbonnot : s.n.

41. **WALK, S., et al., et al.** New features and insights for pedestrian detection. *Computer vision and pattern recognition (CVPR).* 2010. p 1030-1037.

42. **STACHOWICZ, M. e LEMKE, D.** Color Recognition. s.l. : Department of Electrical and Computer Engineering - University of Minnesota.

43. **ZHAN, Z.** Fuzzy mathematical approach to color recognition. University of Minnesota Master's Thesis : s.n., 1998.

44. **KALMAN, R.E.** A new approach to linear filtering and prediction problems. *Journal of Basic Engineering .* 1960.

45. Filtro de Kalman. [Online] [Citado em: 4 de 11 de 2015.] http://www.maxwell.vrac.puc-rio.br/7604/7604_5.PDF.

46. [Online] NEO Coolcam. [Citado em: 18 de 11 de 2015.] http://www.szneo.com/en/products/show.php?id=134.

47. [Online] NEO Coolcam. [Citado em: 18 de 11 de 2015.] http://www.szneo.com/en/products/show.php?id=147.

48. Yale Faces Database. [Online] [Citado em: 28 de Março de 2015.] vision.ucsd.edu/content/yale-face-database.

49. *Dome Camera Owner's Manual Model LT703.*

50. **GERONIMO D., LOPEZ, A., SAPPA A.** *Computer vision approaches to pedestriandetection: Visible spectrum survey. LNCS, Pattern Recognition and Image Analysis, 547–554.*

ANEXO 1 - SUMÁRIO EXECUTIVO

26/09/2014 – Reunião com o orientador sobre primeira ideia de projeto

30/09/2014 – Reunião com orientador sobre propostas de projetos sugeridas por ele

05/11/2014 – Escolha do projeto atual

11/12/2014 – Primeiro esboço de artigo e cronograma

18/12/2014 – Reunião com orientador sobre artigo e cronograma

10/03/2015 – Reunião com o orientador sobre OpenCV

19/03/2015 – Primeiras tentativas de simulação com OpenCV

20/03/2015 – Elaboração da Proposta de Projeto

23/03/2015 – Revisão da Proposta de Projeto

25/03/2015 – Entrega da Proposta de Projeto

10/04/2015 – Elaboração de software de detecção facial concluído

16/04/2015 – Reunião com o orientador sobre Relatório Intermediário

22/04/2015 – Entrega do Relatório Intermediário do Primeiro Semestre

07/05/2015 – Reunião com o orientador sobre Relatório Final do Primeiro Semestre

10/05/2015 – Sequência das simulações de detecção e identificação com OpenCV

20/05/2015 – Conversa com o orientador sobre sequência do projeto e perspectivas para o segundo semestre

05/06/2015 – Início da elaboração do pôster do projeto

16/06/2015 – De acordo do orientador sobre estrutura e conteúdo do pôster

17/06/2015 – Entrega do pôster

20/06/2015 – Finalização do Relatório Final do Primeiro Semestre

24/06/2015 – Apresentação do pôster e entrega do Relatório Final do Primeiro Semestre

13/08/2015 – Reunião com o orientador sobre resultados do primeiro semestre e expectativas e cronograma para o segundo semestre

02/09/2015 – Reunião com o orientador sobre métodos de detecção e rastreamento

08/09/2015 – Reunião sobre artigos científicos e pesquisa

20/09/2015 – Início da elaboração do Relatório Intermediário

01/10/2015 – Entrega do Relatório Intermediário do Segundo Semestre

10/10/2015 – Estudos sobre reconhecimento de cores, HOG e filtro de Kalman

22/10/2015 – Reunião com o orientador sobre detecção de corpos e funcionamento da câmera

25/10/2015 – Câmera instalada e funcionando

30/10/2015 – Primeiros comandos de controle para a câmera e testes de detecção de corpos

08/11/2015 – Aperfeiçoamento do rastreamento através do filtro de Kalman

12/11/2015 – Algoritmo para movimentar a câmera

17/11/2015 – Reunião com o orientador sobre captura de imagem da câmera usando Java e últimos detalhes do projeto

19/11/2015 – Monografia e artigo

ANEXO 2 – CÓDIGOS

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hog;

import static com.googlecode.javacv.cpp.opencv_core.*;
import static com.googlecode.javacv.cpp.opencv_highgui.*;
import static com.googlecode.javacv.cpp.opencv_objdetect.*;

import com.googlecode.javacpp.annotation.ByVal;
import com.googlecode.javacpp.annotation.StdVector;
import com.googlecode.javacv.CanvasFrame;
import com.googlecode.javacv.OpenCVFrameGrabber;
import com.googlecode.javacv.cpp.opencv_core;
import com.googlecode.javacv.cpp.opencv_core.CvArr;
import static com.googlecode.javacv.cpp.opencv_core.CvMat.create;
import com.googlecode.javacv.cpp.opencv_core.CvMemStorage;
import com.googlecode.javacv.cpp.opencv_core.CvRect;
import com.googlecode.javacv.cpp.opencv_core.CvSize;
import com.googlecode.javacv.cpp.opencv_core.InputMat;
import com.googlecode.javacv.cpp.opencv_imgproc;
```

```
import static com.googlecode.javacv.cpp.opencv_imgproc.CV_BGR2HSV;
import static com.googlecode.javacv.cpp.opencv_imgproc.CV_HIST_ARRAY;
import com.googlecode.javacv.cpp.opencv_imgproc.CvHistogram;
import static com.googlecode.javacv.cpp.opencv_imgproc.cvCreateHist;
import static com.googlecode.javacv.cpp.opencv_imgproc.cvCvtColor;
import com.googlecode.javacv.cpp.opencv_video;
import com.googlecode.javacv.cpp.opencv_video.CvKalman;
import static com.googlecode.javacv.cpp.opencv_video.cvKalmanCorrect;
import static com.googlecode.javacv.cpp.opencv_video.cvKalmanPredict;

public class HOG {

    private static OpenCVFrameGrabber grabber = null;
    private static IplImage currentFrame;
    private static double trackerThreshold = 100;

    private static boolean queryFrame() throws Exception {
        IplImage frame = grabber.grab();

        if (frame != null) {
            cvConvertImage(frame, currentFrame, 0);
            return true;
        } else {
            return false;
```

```
    }
  }

  public static void main(String arg[]) throws Exception {

    currentFrame = cvLoadImage("resources/body9.jpg");
    HOGDescriptor hog = new HOGDescriptor();
    hog.setSVMDetector(HOGDescriptor.getDefaultPeopleDetector());
     //CvMemStorage storage = CvMemStorage.create();
    //cvClearMemStorage(storage);
    CvRect found = new CvRect();
    hog.detectMultiScale(currentFrame, found, 0, cvSize(8, 8), cvSize(64, 64), 1.1, 2);
    IplImage img = currentFrame;
    double test_result = Double.POSITIVE_INFINITY;
    double test_aux = 0;
    double test_threshold = Double.POSITIVE_INFINITY;
    IplImage recog = null;

    for (int i = 0; i < found.sizeof(); i++) {
      int x = found.position(i).x(), y = found.position(i).y(), w = found.position(i).width(), h =
found.position(i).height();
      if (x > 0 && y > 0 && w > 0 && h > 0 && x <= img.width() && img.height() >= 0 && y
<= img.height() && x + w >= 0 && y + h >= 0) {
        cvRectangle(currentFrame, cvPoint(x, y), cvPoint(x + w, y + h), CvScalar.GREEN,
2, CV_AA, 0);
```

```
        cvSetImageROI(img, cvRect(x, y, w, h));
        IplImage roi = cvCreateImage(cvGetSize(img), img.depth(), img.nChannels());
        cvCopy(img, roi, null);
        cvResetImageROI(img);

        cvShowImage("Result", roi);
        cvWaitKey(0);
        cvDestroyWindow("Result");
        //cvReleaseImage(roi);

        test_aux = Histogram(cvLoadImage("resources/test.jpg"), roi);
        if (test_aux < test_result && test_aux < test_threshold) {
            test_result = test_aux;
            recog = cvCreateImage(cvGetSize(roi), roi.depth(), roi.nChannels());
            cvCopy(roi, recog, null);
        }

    }

}
System.out.println(test_result);
cvShowImage("Test", cvLoadImage("resources/test.jpg"));
cvWaitKey(0);
```

```
cvShowImage("Result", recog);
cvShowImage("Result", currentFrame);
cvWaitKey(0);

 //IplImage result = NormalizeImage(currentFrame);
//cvShowImage("Result", result);
//cvReleaseImage(result);

/*
 grabber = new OpenCVFrameGrabber(0);
 grabber.start();
 CanvasFrame canvasFrame = new CanvasFrame("HOG-Test");
 canvasFrame.setCanvasSize(grabber.getImageWidth(), grabber.getImageHeight());
 CvSize frameSize = new CvSize(grabber.getImageWidth(), grabber.getImageHeight());
 currentFrame = cvCreateImage(frameSize, IPL_DEPTH_8U, 3);
 //HOGDescriptor hog = new HOGDescriptor();
 //hog.setSVMDetector(HOGDescriptor.getDefaultPeopleDetector());

 //Detect objects
 /*
 CvMemStorage storage = CvMemStorage.create();
 while (queryFrame()) {
 cvClearMemStorage(storage);
 CvRect found = new CvRect();
```

```
CvSeq sign = cvHaarDetectObjects(currentFrame, cascade, storage, 1.5, 3, CV_HAAR_DO_CANNY_PRUNING);

//hog.detectMultiScale(currentFrame, found, 0, cvSize(8, 8), cvSize(64, 64), 1.1, 2);

for (int i = 0; i < found.sizeof(); i++) {

int x = found.position(i).x(), y = found.position(i).y(), w = found.position(i).width(), h = found.position(i).height();

cvRectangle(currentFrame, cvPoint(x, y), cvPoint(x + w, y + h), CvScalar.GREEN, 2, CV_AA, 0);

}

canvasFrame.showImage(currentFrame);

}

grabber.stop();

canvasFrame.dispose();

*/

/*

grabber = new OpenCVFrameGrabber(0);

grabber.start();

CanvasFrame canvasFrame = new CanvasFrame("HOG-Test");

canvasFrame.setCanvasSize(grabber.getImageWidth(), grabber.getImageHeight());

CvSize frameSize = new CvSize(grabber.getImageWidth(), grabber.getImageHeight());

currentFrame = cvCreateImage(frameSize, IPL_DEPTH_8U, 3);


CvHaarClassifierCascade cascade = new CvHaarClassifierCascade(cvLoad("resources/haarcascade_frontalface_default.xml"));

CvMemStorage storage = CvMemStorage.create();
```

```
int numberOfTrackers = 0;

CvKalman tracker[] = new CvKalman[10];

Kalman kalman = new Kalman();

CvMat state[] = new CvMat[10];

CvMat measure[] = new CvMat[10];

int counter[] = new int[10];

int newFaceCounter = 0;

double x0[] = new double[10];

double y0[] = new double[10];


while (queryFrame()) {

cvClearMemStorage(storage);

CvSeq sign = cvHaarDetectObjects(currentFrame, cascade, storage, 1.5, 3,
CV_HAAR_DO_CANNY_PRUNING);


int total_Faces = sign.total();

CvRect r[] = new CvRect[10];


//Draw rectangles around detected objects

for (int i = 0; i < total_Faces; i++) {


r[i] = new CvRect(cvGetSeqElem(sign, i));


cvRectangle(currentFrame, cvPoint(r[i].x(), r[i].y()), cvPoint(r[i].width() + r[i].x(),
```

```
r[i].height() + r[i].y()), CvScalar.RED, 2, CV_AA, 0);

int index = -1;
boolean trackExist = true;
int faceNumber = 0;

for (int j = 0; j < numberOfTrackers; j++) {
index = j;
if (total_Faces > numberOfTrackers) {
System.out.println("faces " + total_Faces);
newFaceCounter++;

if (newFaceCounter > 10) {

System.out.println("Distance= " + distance(r[i].x(), r[i].y(), (double) state[index].get(0, 0),
(double) state[index].get(1, 0)));
if (distance(r[i].x(), r[i].y(), (double) state[index].get(0, 0), (double) state[index].get(1, 0))
> trackerThreshold) {

trackExist = false;
faceNumber = i;
}
}
} else {
newFaceCounter = 0;
```

```
}

}

if (!trackExist || numberOfTrackers == 0 && numberOfTrackers < 9) {

tracker[index + 1] = kalman.createNewTracker();

state[index + 1] = create(6, 1, CV_32F);

state[index + 1].put(0, 0, r[faceNumber].x());

state[index + 1].put(1, 0, r[faceNumber].y());

state[index + 1].put(2, 0, 0);

state[index + 1].put(3, 0, 0);

state[index + 1].put(4, 0, r[faceNumber].width());

state[index + 1].put(5, 0, r[faceNumber].height());

measure[index + 1] = create(6, 1, CV_32F);

x0[index + 1] = r[faceNumber].x();

y0[index + 1] = r[faceNumber].y();

numberOfTrackers++;

System.out.println("Create tracker");

System.out.println("Trackers " + numberOfTrackers);

}

}
```

```
for (int trackNumber = 0; trackNumber < numberOfTrackers; trackNumber++) {
cvKalmanPredict(tracker[trackNumber], state[trackNumber]);
tracker[trackNumber].state_pre(state[trackNumber]);

if (trackNumber < total_Faces) {
double result;
double aux = 0;
int faceNumber = 0;

for (int i = 0; i < total_Faces; i++) {
result = distance((double) r[i].x(), (double) r[i].y(), (double) state[trackNumber].get(0, 0),
(double) state[trackNumber].get(1, 0));

if (result < aux || i == 0) {
faceNumber = i;
aux = result;
}
}
measure[trackNumber].put(0, 0, r[faceNumber].x());
measure[trackNumber].put(1, 0, r[faceNumber].y());
double vx = (r[faceNumber].x() - x0[faceNumber]) / counter[trackNumber];
double vy = (r[faceNumber].y() - y0[faceNumber]) / counter[trackNumber];
x0[faceNumber] = r[faceNumber].x();
```

```
y0[faceNumber] = r[faceNumber].y();

measure[trackNumber].put(2, 0, vx);

measure[trackNumber].put(3, 0, vy);

measure[trackNumber].put(4, 0, r[faceNumber].width());

measure[trackNumber].put(5, 0, r[faceNumber].height());


cvKalmanCorrect(tracker[trackNumber], measure[trackNumber]);

tracker[trackNumber].state_post(state[trackNumber]);

counter[trackNumber] = 1;


} else {

//System.out.println("no measure");

counter[trackNumber]++;

//System.out.println("conter= " + counter[index]);


if (counter[trackNumber] > 20) {

numberOfTrackers--;

System.out.println("Destroy tracker");

System.out.println("Trackers " + numberOfTrackers);

}

//tracker[index].state_post(state[index]);

}

//System.out.println(r[index].x());
```

```
        cvRectangle(currentFrame, cvPoint((int) state[trackNumber].get(0, 0), (int) state[trackNumber].get(1, 0)),
        cvPoint((int) state[trackNumber].get(4, 0) + (int) state[trackNumber].get(0, 0),
        (int) state[trackNumber].get(5, 0) + (int) state[trackNumber].get(1, 0)), CvScalar.GREEN, 2, CV_AA, 0);
        //cvRectangle(currentFrame, cvPoint(10, 10),cvPoint(100, 100), CvScalar.GREEN, 2, CV_AA, 0);
        }
        canvasFrame.showImage(currentFrame);
        }*/
    }

    static double distance(double x1, double y1, double x2, double y2) {
        double result = Math.sqrt(Math.pow((x1 - x2), 2) + Math.pow((y1 - y2), 2));

        return result;
    }

    static IplImage NormalizeImage(IplImage theimg) {

        IplImage redchannel = cvCreateImage(cvGetSize(theimg), 8, 1);
        IplImage greenchannel = cvCreateImage(cvGetSize(theimg), 8, 1);
        IplImage bluechannel = cvCreateImage(cvGetSize(theimg), 8, 1);

        IplImage redavg = cvCreateImage(cvGetSize(theimg), 8, 1);
```

```
IplImage greenavg = cvCreateImage(cvGetSize(theimg), 8, 1);
IplImage blueavg = cvCreateImage(cvGetSize(theimg), 8, 1);

IplImage imgavg = cvCreateImage(cvGetSize(theimg), 8, 3);
cvSet(imgavg, cvScalar(255, 255, 255, 0));

cvSplit(theimg, bluechannel, greenchannel, redchannel, null);

for (int x = 0; x < theimg.width(); x++) {
    for (int y = 0; y < theimg.height(); y++) {
        double redValue = cvGetReal2D(redchannel, y, x);
        double greenValue = cvGetReal2D(greenchannel, y, x);
        double blueValue = cvGetReal2D(bluechannel, y, x);
        double sum = redValue + greenValue + blueValue;

        cvSetReal2D(redavg, y, x, redValue / sum * 255);
        cvSetReal2D(greenavg, y, x, greenValue / sum * 255);
        cvSetReal2D(blueavg, y, x, blueValue / sum * 255);
    }
}
cvMerge(blueavg, greenavg, redavg, null, imgavg);
cvReleaseImage(redchannel);
cvReleaseImage(greenchannel);
cvReleaseImage(bluechannel);
```

```
    cvReleaseImage(redavg);
    cvReleaseImage(greenavg);
    cvReleaseImage(blueavg);

    return imgavg;

}

static double Histogram(IplImage src_base, IplImage src_test) {

    IplImage hsv_base = cvCreateImage(src_base.cvSize(), src_base.depth(), 3);
    IplImage hsv_test = cvCreateImage(src_test.cvSize(), src_test.depth(), 3);

    /// Convert to HSV
    cvCvtColor(src_base, hsv_base, CV_BGR2HSV);
    cvCvtColor(src_test, hsv_test, CV_BGR2HSV);

    IplImageArray hsvChannels_base = splitChannels(hsv_base);
    IplImageArray hsvChannels_test = splitChannels(hsv_test);

    /// Using 50 bins for hue and 60 for saturation
    int h_bins = 50;
    int s_bins = 60;
    int histSize[] = {h_bins, s_bins};
```

```
    // hue varies from 0 to 179, saturation from 0 to 255
    float h_ranges[] = {0, 180};
    float s_ranges[] = {0, 256};
    float[] ranges[] = {h_ranges, s_ranges};

    /// Histograms
    CvHistogram hist_base = cvCreateHist(2, histSize, CV_HIST_ARRAY, ranges, 1);
    CvHistogram hist_test = cvCreateHist(2, histSize, CV_HIST_ARRAY, ranges, 1);

    /// Calculate the histograms for the HSV images
    opencv_imgproc.cvCalcHist(hsvChannels_base, hist_base, 0, null);
    opencv_imgproc.cvNormalizeHist(hist_base, 1);

    opencv_imgproc.cvCalcHist(hsvChannels_test, hist_test, 0, null);
    opencv_imgproc.cvNormalizeHist(hist_test, 1);

    /// Apply the histogram comparison methods
    double base_test = opencv_imgproc.cvCompareHist(hist_base, hist_test, 1);

    return base_test;
}

private static IplImageArray splitChannels(IplImage hsvImage) {
```

```
        CvSize size = hsvImage.cvSize();
        int depth = hsvImage.depth();
        IplImage channel0 = cvCreateImage(size, depth, 1);
        IplImage channel1 = cvCreateImage(size, depth, 1);
        IplImage channel2 = cvCreateImage(size, depth, 1);
        cvSplit(hsvImage, channel0, channel1, channel2, null);

        return new IplImageArray(channel0, channel1, channel2);
    }
}
```

```
package hog;

import static com.googlecode.javacv.cpp.opencv_core.CV_32F;
import com.googlecode.javacv.cpp.opencv_core.CvMat;
import static com.googlecode.javacv.cpp.opencv_core.CvMat.create;
import com.googlecode.javacv.cpp.opencv_video.CvKalman;

import static com.googlecode.javacv.cpp.opencv_video.cvCreateKalman;
import static com.googlecode.javacv.cpp.opencv_video.cvKalmanCorrect;
import static com.googlecode.javacv.cpp.opencv_video.cvKalmanPredict;

public class Kalman {

    CvMat state;
    CvMat meas;

    public CvKalman createNewTracker() {
        int stateSize = 6;
        int measureSize = 6;
        int controlSize = 0;

        CvKalman kf = cvCreateKalman(stateSize, measureSize, controlSize);

        state = create(stateSize, 1, CV_32F); // [x, y, v_x, v_y, w, h]
```

```
meas = create(measureSize, 1, CV_32F); // [z_x, z_y, vz_x, vz_y, z_w, z_h]


//cv::Mat procNoise(stateSize, 1, type)
// [E_x,E_y,E_v_x,E_v_y,E_w,E_h]
CvMat transition = create(6, 6, CV_32F);
// Transition State Matrix A
// Note: set dT at each processing step!
// [ 1 0 1 0 0 0 ]
// [ 0 1 0 1 0 0 ]
// [ 0 0 1 0 0 0 ]
// [ 0 0 0 1 0 0 ]
// [ 0 0 0 0 1 0 ]
// [ 0 0 0 0 0 1 ]


transition.put(0, 0, 1);
transition.put(0, 1, 0);
transition.put(0, 2, 1);
transition.put(0, 3, 0);
transition.put(0, 4, 0);
transition.put(0, 5, 0);


transition.put(1, 0, 0);
transition.put(1, 1, 1);
transition.put(1, 2, 0);
```

```
transition.put(1, 3, 1);

transition.put(1, 4, 0);

transition.put(1, 5, 0);


transition.put(2, 0, 0);

transition.put(2, 1, 0);

transition.put(2, 2, 1);

transition.put(2, 3, 0);

transition.put(2, 4, 0);

transition.put(2, 5, 0);


transition.put(3, 0, 0);

transition.put(3, 1, 0);

transition.put(3, 2, 0);

transition.put(3, 3, 1);

transition.put(3, 4, 0);

transition.put(3, 5, 0);


transition.put(4, 0, 0);

transition.put(4, 1, 0);

transition.put(4, 2, 0);

transition.put(4, 3, 0);

transition.put(4, 4, 1);

transition.put(4, 5, 0);
```

```
transition.put(5, 0, 0);

transition.put(5, 1, 0);

transition.put(5, 2, 0);

transition.put(5, 3, 0);

transition.put(5, 4, 0);

transition.put(5, 5, 1);


kf.transition_matrix(transition);


CvMat measureMatrix = create(6, 6, CV_32F);

// Measure Matrix H

// [ 1 0 0 0 0 0 ]

// [ 0 1 0 0 0 0 ]

// [ 0 0 1 0 0 0 ]

// [ 0 0 0 1 0 0 ]

// [ 0 0 0 0 1 0 ]

// [ 0 0 0 0 0 1 ]


measureMatrix.put(0, 0, 1);

measureMatrix.put(0, 1, 0);

measureMatrix.put(0, 2, 0);

measureMatrix.put(0, 3, 0);

measureMatrix.put(0, 4, 0);
```

```
measureMatrix.put(0, 5, 0);


measureMatrix.put(1, 0, 0);
measureMatrix.put(1, 1, 1);
measureMatrix.put(1, 2, 0);
measureMatrix.put(1, 3, 0);
measureMatrix.put(1, 4, 0);
measureMatrix.put(1, 5, 0);

measureMatrix.put(2, 0, 0);
measureMatrix.put(2, 1, 0);
measureMatrix.put(2, 2, 1);
measureMatrix.put(2, 3, 0);
measureMatrix.put(2, 4, 0);
measureMatrix.put(2, 5, 0);

measureMatrix.put(3, 0, 0);
measureMatrix.put(3, 1, 0);
measureMatrix.put(3, 2, 0);
measureMatrix.put(3, 3, 1);
measureMatrix.put(3, 4, 0);
measureMatrix.put(3, 5, 0);

measureMatrix.put(4, 0, 0);
```

```
measureMatrix.put(4, 1, 0);

measureMatrix.put(4, 2, 0);

measureMatrix.put(4, 3, 0);

measureMatrix.put(4, 4, 1);

measureMatrix.put(4, 5, 0);


measureMatrix.put(5, 0, 0);

measureMatrix.put(5, 1, 0);

measureMatrix.put(5, 2, 0);

measureMatrix.put(5, 3, 0);

measureMatrix.put(5, 4, 0);

measureMatrix.put(5, 5, 1);


kf.measurement_matrix(measureMatrix);


// Process Noise Covariance Matrix Q

// [ Ex 0  0    0 0    0 ]

// [ 0  Ey 0    0 0    0 ]

// [ 0  0  Ev_x 0 0    0 ]

// [ 0  0  0    1 Ev_y 0 ]

// [ 0  0  0    0 1    Ew ]

// [ 0  0  0    0 0    Eh ]

//cv::setIdentity(kf.processNoiseCov, cv::Scalar(1e-2));

/*
```

```
CvMat noiseCovariance = create(4, 4, CV_32F);

noiseCovariance.put(1, 1, 1e-2);

noiseCovariance.put(1, 2, 0);

noiseCovariance.put(1, 3, 0);

noiseCovariance.put(1, 4, 0);


noiseCovariance.put(2, 1, 0);

noiseCovariance.put(2, 2, 1e-2);

noiseCovariance.put(2, 3, 0);

noiseCovariance.put(2, 4, 0);


noiseCovariance.put(3, 1, 0);

noiseCovariance.put(3, 2, 0);

noiseCovariance.put(3, 3, 1.0);

noiseCovariance.put(3, 4, 0);


noiseCovariance.put(4, 1, 0);

noiseCovariance.put(4, 2, 0);

noiseCovariance.put(4, 3, 0);

noiseCovariance.put(4, 4, 1.0);


kf.measurement_noise_cov(noiseCovariance);


kf.process_noise_cov(noiseCovariance);
```

```
        // Measures Noise Covariance Matrix R
        */

        return kf;
    }

    public CvMat updateTracker(CvKalman kf, boolean found) {
        /*
         kf.transitionMatrix.at(2) = dT;
         kf.transitionMatrix.at(9) = dT;
         */
        cvKalmanPredict(kf, state);

        if (found) {
            cvKalmanCorrect(kf, meas);
        } else {
            kf.state_post(state);
        }

        return state;
    }
}
```

```
package tcc;

import static com.googlecode.javacv.cpp.opencv_core.CV_32F;
import static com.googlecode.javacv.cpp.opencv_core.CV_AA;
import com.googlecode.javacv.cpp.opencv_core.CvMat;
import static com.googlecode.javacv.cpp.opencv_core.CvMat.create;
import com.googlecode.javacv.cpp.opencv_core.CvScalar;
import com.googlecode.javacv.cpp.opencv_core.IplImage;
import static com.googlecode.javacv.cpp.opencv_core.cvPoint;
import static com.googlecode.javacv.cpp.opencv_core.cvRectangle;
import com.googlecode.javacv.cpp.opencv_video.CvKalman;
import static com.googlecode.javacv.cpp.opencv_video.cvCreateKalman;
import static com.googlecode.javacv.cpp.opencv_video.cvKalmanCorrect;
import static com.googlecode.javacv.cpp.opencv_video.cvKalmanPredict;
import static java.awt.geom.Point2D.distance;
import java.util.ArrayList;

public class Tracker {

    int numberOfTrackers = 0;
    CvKalman tracker[] = new CvKalman[10];
    CvMat state[] = new CvMat[10];
    CvMat measure[] = new CvMat[10];
    int counter[] = new int[10];
```

```
int newFaceCounter = 0;
double x0[] = new double[10];
double y0[] = new double[10];
private static double trackerThreshold = 100;

IplImage faceTrack(ArrayList<DetectedFace> faceArray, IplImage currentFrame) {

    int total_Faces = faceArray.size();
    for (int i = 0; i < total_Faces; i++) {

        int x = faceArray.get(i).getXinitialPoint();
        int y = faceArray.get(i).getYinitialPoint();
        int width = faceArray.get(i).getWidth();
        int height = faceArray.get(i).getHeight();

        cvRectangle(currentFrame, cvPoint(x, y), cvPoint(width + x,
                height + y), CvScalar.RED, 2, CV_AA, 0);

        int index = -1;
        boolean trackExist = true;
        int faceNumber = 0;

        for (int j = 0; j < numberOfTrackers; j++) {
            index = j;
```

```
            if (total_Faces > numberOfTrackers) {
                System.out.println("faces " + total_Faces);
                newFaceCounter++;

                if (newFaceCounter > 10) {

                    System.out.println("Distance= " + distance(x, y, state[index].get(0, 0), state[index].get(1, 0)));
                    if (distance(x, y, state[index].get(0, 0), state[index].get(1, 0)) > trackerThreshold) {

                        trackExist = false;
                        faceNumber = i;
                    }
                }
            } else {
                newFaceCounter = 0;
            }
        }

        if (!trackExist || numberOfTrackers == 0 && numberOfTrackers < 9) {

            tracker[index + 1] = createNewTracker();
            state[index + 1] = create(6, 1, CV_32F);
            state[index + 1].put(0, 0, x);
```

```
        state[index + 1].put(1, 0, y);
        state[index + 1].put(2, 0, 0);
        state[index + 1].put(3, 0, 0);
        state[index + 1].put(4, 0, width);
        state[index + 1].put(5, 0, height);

        measure[index + 1] = create(6, 1, CV_32F);
        x0[index + 1] = faceArray.get(faceNumber).getXinitialPoint();
        y0[index + 1] = faceArray.get(faceNumber).getYinitialPoint();

        numberOfTrackers++;
        System.out.println("Create tracker");
        System.out.println("Trackers " + numberOfTrackers);
    }
}

for (int trackNumber = 0; trackNumber < numberOfTrackers; trackNumber++) {
    try {
        cvKalmanPredict(tracker[trackNumber], state[trackNumber]);
        tracker[trackNumber].state_pre(state[trackNumber]);

        if (trackNumber < total_Faces) {
            double result;
            double aux = 0;
```

```
int faceNumber = 0;


for (int i = 0; i < total_Faces; i++) {
   result = distance(faceArray.get(i).getXinitialPoint(),
faceArray.get(i).getYinitialPoint(),
      state[trackNumber].get(0, 0), state[trackNumber].get(1, 0));


   if (result < aux || i == 0) {
      faceNumber = i;
      aux = result;
   }
}


measure[trackNumber].put(0, 0, faceArray.get(faceNumber).getXinitialPoint());
measure[trackNumber].put(1, 0, faceArray.get(faceNumber).getYinitialPoint());
double vx = (faceArray.get(faceNumber).getXinitialPoint() - x0[faceNumber]); //
counter[trackNumber];
double vy = (faceArray.get(faceNumber).getYinitialPoint() - y0[faceNumber]); //
counter[trackNumber];
x0[faceNumber] = faceArray.get(faceNumber).getXinitialPoint();
y0[faceNumber] = faceArray.get(faceNumber).getYinitialPoint();
measure[trackNumber].put(2, 0, vx);
measure[trackNumber].put(3, 0, vy);
measure[trackNumber].put(4, 0, faceArray.get(faceNumber).getWidth());
measure[trackNumber].put(5, 0, faceArray.get(faceNumber).getHeight());
```

```
                cvKalmanCorrect(tracker[trackNumber], measure[trackNumber]);
                tracker[trackNumber].state_post(state[trackNumber]);
                counter[trackNumber] = 1;
            } else {
                //System.out.println("no measure");
                counter[trackNumber]++;
                //System.out.println("conter= " + counter[index]);

                if (counter[trackNumber] > 20) {
                    numberOfTrackers--;
                    System.out.println("Destroy tracker");
                    System.out.println("Trackers " + numberOfTrackers);
                }
                 tracker[trackNumber].state_post(state[trackNumber]);
            }
            //System.out.println(r[index].x());
            cvRectangle(currentFrame, cvPoint((int) state[trackNumber].get(0, 0), (int) state[trackNumber].get(1, 0)),
                    cvPoint((int) state[trackNumber].get(4, 0) + (int) state[trackNumber].get(0, 0),
                            (int) state[trackNumber].get(5, 0) + (int) state[trackNumber].get(1, 0)), CvScalar.GREEN, 2, CV_AA, 0);
        } catch (Exception e) {
            numberOfTrackers--;
        }
```

```
        //cvRectangle(currentFrame, cvPoint(10, 10),cvPoint(100, 100), CvScalar.GREEN, 2,
CV_AA, 0);
      }
      //canvasFrame.showImage(currentFrame);
      return currentFrame;
   }

   IplImage faceTrack(DetectedFace face, IplImage currentFrame) {

      int total_Faces = 1;
      for (int i = 0; i < total_Faces; i++) {

         int x = face.getXinitialPoint();
         int y = face.getYinitialPoint();
         int width = face.getWidth();
         int height = face.getHeight();

         cvRectangle(currentFrame, cvPoint(x, y), cvPoint(width + x,
               height + y), CvScalar.RED, 2, CV_AA, 0);

         int index = -1;
         boolean trackExist = true;
         int faceNumber = 0;
```

```
for (int j = 0; j < numberOfTrackers; j++) {
    index = j;
    if (total_Faces > numberOfTrackers) {
        System.out.println("faces " + total_Faces);
        newFaceCounter++;

        if (newFaceCounter > 10) {

            System.out.println("Distance= " + distance(x, y, state[index].get(0, 0), state[index].get(1, 0)));
            if (distance(x, y, state[index].get(0, 0), state[index].get(1, 0)) > trackerThreshold) {

                trackExist = false;
                faceNumber = i;
            }
        }
    } else {
        newFaceCounter = 0;
    }
}

if (!trackExist || numberOfTrackers == 0 && numberOfTrackers < 9) {

    tracker[index + 1] = createNewTracker();
```

```
        state[index + 1] = create(6, 1, CV_32F);
        state[index + 1].put(0, 0, x);
        state[index + 1].put(1, 0, y);
        state[index + 1].put(2, 0, 0);
        state[index + 1].put(3, 0, 0);
        state[index + 1].put(4, 0, width);
        state[index + 1].put(5, 0, height);

        measure[index + 1] = create(6, 1, CV_32F);
        x0[index + 1] = face.getXinitialPoint();
        y0[index + 1] = face.getYinitialPoint();

        numberOfTrackers++;
        System.out.println("Create tracker");
        System.out.println("Trackers " + numberOfTrackers);
    }
}

for (int trackNumber = 0; trackNumber < numberOfTrackers; trackNumber++) {
    cvKalmanPredict(tracker[trackNumber], state[trackNumber]);
    tracker[trackNumber].state_pre(state[trackNumber]);

    if (trackNumber < total_Faces) {
        double result;
```

```
double aux = 0;
int faceNumber = 0;

for (int i = 0; i < total_Faces; i++) {
    result = distance(face.getXinitialPoint(), face.getYinitialPoint(),
            state[trackNumber].get(0, 0), state[trackNumber].get(1, 0));

    if (result < aux || i == 0) {
        faceNumber = i;
        aux = result;
    }
}

measure[trackNumber].put(0, 0, face.getXinitialPoint());
measure[trackNumber].put(1, 0, face.getYinitialPoint());
double vx = (face.getXinitialPoint() - x0[faceNumber]); // counter[trackNumber];
double vy = (face.getYinitialPoint() - y0[faceNumber]); // counter[trackNumber];
x0[faceNumber] = face.getXinitialPoint();
y0[faceNumber] = face.getYinitialPoint();
measure[trackNumber].put(2, 0, vx);
measure[trackNumber].put(3, 0, vy);
measure[trackNumber].put(4, 0, face.getWidth());
measure[trackNumber].put(5, 0, face.getHeight());
```

```
            cvKalmanCorrect(tracker[trackNumber], measure[trackNumber]);
            tracker[trackNumber].state_post(state[trackNumber]);
            counter[trackNumber] = 1;

        } else {
            //System.out.println("no measure");
            counter[trackNumber]++;
            //System.out.println("conter= " + counter[index]);

            if (counter[trackNumber] > 20) {
                numberOfTrackers--;
                System.out.println("Destroy tracker");
                System.out.println("Trackers " + numberOfTrackers);
            }
            //tracker[index].state_post(state[index]);
        }
        //System.out.println(r[index].x());
        cvRectangle(currentFrame, cvPoint((int) state[trackNumber].get(0, 0), (int) state[trackNumber].get(1, 0)),
                cvPoint((int) state[trackNumber].get(4, 0) + (int) state[trackNumber].get(0, 0),
                        (int) state[trackNumber].get(5, 0) + (int) state[trackNumber].get(1, 0)), CvScalar.GREEN, 2, CV_AA, 0);
        //cvRectangle(currentFrame, cvPoint(10, 10),cvPoint(100, 100), CvScalar.GREEN, 2, CV_AA, 0);
    }
```

```
    //canvasFrame.showImage(currentFrame);
    return currentFrame;
}

public CvKalman createNewTracker() {
    int stateSize = 6;
    int measureSize = 6;
    int controlSize = 0;
    CvMat trackState;
    CvMat trackMeas;

    CvKalman kf = cvCreateKalman(stateSize, measureSize, controlSize);

    trackState = create(stateSize, 1, CV_32F); // [x, y, v_x, v_y, w, h]
    trackMeas = create(measureSize, 1, CV_32F); // [z_x, z_y, vz_x, vz_y, z_w, z_h]

    //cv::Mat procNoise(stateSize, 1, type)
    // [E_x,E_y,E_v_x,E_v_y,E_w,E_h]
    CvMat transition = create(6, 6, CV_32F);
    // Transition State Matrix A
    // Note: set dT at each processing step!
    // [ 1 0 1 0 0 0 ]
    // [ 0 1 0 1 0 0 ]
    // [ 0 0 1 0 0 0 ]
```

```
// [ 0 0 0 1 0 0 ]

// [ 0 0 0 0 1 0 ]

// [ 0 0 0 0 0 1 ]


transition.put(0, 0, 1);

transition.put(0, 1, 0);

transition.put(0, 2, 1);

transition.put(0, 3, 0);

transition.put(0, 4, 0);

transition.put(0, 5, 0);


transition.put(1, 0, 0);

transition.put(1, 1, 1);

transition.put(1, 2, 0);

transition.put(1, 3, 1);

transition.put(1, 4, 0);

transition.put(1, 5, 0);


transition.put(2, 0, 0);

transition.put(2, 1, 0);

transition.put(2, 2, 1);

transition.put(2, 3, 0);

transition.put(2, 4, 0);

transition.put(2, 5, 0);
```

```
transition.put(3, 0, 0);

transition.put(3, 1, 0);

transition.put(3, 2, 0);

transition.put(3, 3, 1);

transition.put(3, 4, 0);

transition.put(3, 5, 0);


transition.put(4, 0, 0);

transition.put(4, 1, 0);

transition.put(4, 2, 0);

transition.put(4, 3, 0);

transition.put(4, 4, 1);

transition.put(4, 5, 0);


transition.put(5, 0, 0);

transition.put(5, 1, 0);

transition.put(5, 2, 0);

transition.put(5, 3, 0);

transition.put(5, 4, 0);

transition.put(5, 5, 1);


kf.transition_matrix(transition);
```

```
CvMat measureMatrix = create(6, 6, CV_32F);

// Measure Matrix H

// [ 1 0 0 0 0 0 ]

// [ 0 1 0 0 0 0 ]

// [ 0 0 1 0 0 0 ]

// [ 0 0 0 1 0 0 ]

// [ 0 0 0 0 1 0 ]

// [ 0 0 0 0 0 1 ]


measureMatrix.put(0, 0, 1);

measureMatrix.put(0, 1, 0);

measureMatrix.put(0, 2, 0);

measureMatrix.put(0, 3, 0);

measureMatrix.put(0, 4, 0);

measureMatrix.put(0, 5, 0);


measureMatrix.put(1, 0, 0);

measureMatrix.put(1, 1, 1);

measureMatrix.put(1, 2, 0);

measureMatrix.put(1, 3, 0);

measureMatrix.put(1, 4, 0);

measureMatrix.put(1, 5, 0);


measureMatrix.put(2, 0, 0);
```

```
measureMatrix.put(2, 1, 0);

measureMatrix.put(2, 2, 1);

measureMatrix.put(2, 3, 0);

measureMatrix.put(2, 4, 0);

measureMatrix.put(2, 5, 0);


measureMatrix.put(3, 0, 0);

measureMatrix.put(3, 1, 0);

measureMatrix.put(3, 2, 0);

measureMatrix.put(3, 3, 1);

measureMatrix.put(3, 4, 0);

measureMatrix.put(3, 5, 0);


measureMatrix.put(4, 0, 0);

measureMatrix.put(4, 1, 0);

measureMatrix.put(4, 2, 0);

measureMatrix.put(4, 3, 0);

measureMatrix.put(4, 4, 1);

measureMatrix.put(4, 5, 0);


measureMatrix.put(5, 0, 0);

measureMatrix.put(5, 1, 0);

measureMatrix.put(5, 2, 0);

measureMatrix.put(5, 3, 0);
```

```
measureMatrix.put(5, 4, 0);

measureMatrix.put(5, 5, 1);


kf.measurement_matrix(measureMatrix);


// Process Noise Covariance Matrix Q

// [ Ex 0  0    0 0    0 ]

// [ 0  Ey 0    0 0    0 ]

// [ 0  0  Ev_x 0 0    0 ]

// [ 0  0  0    1 Ev_y 0 ]

// [ 0  0  0    0 1    Ew ]

// [ 0  0  0    0 0    Eh ]

//cv::setIdentity(kf.processNoiseCov, cv::Scalar(1e-2));

/*

 CvMat noiseCovariance = create(4, 4, CV_32F);

 noiseCovariance.put(1, 1, 1e-2);

 noiseCovariance.put(1, 2, 0);

 noiseCovariance.put(1, 3, 0);

 noiseCovariance.put(1, 4, 0);


 noiseCovariance.put(2, 1, 0);

 noiseCovariance.put(2, 2, 1e-2);

 noiseCovariance.put(2, 3, 0);

 noiseCovariance.put(2, 4, 0);
```

```
        noiseCovariance.put(3, 1, 0);
        noiseCovariance.put(3, 2, 0);
        noiseCovariance.put(3, 3, 1.0);
        noiseCovariance.put(3, 4, 0);

        noiseCovariance.put(4, 1, 0);
        noiseCovariance.put(4, 2, 0);
        noiseCovariance.put(4, 3, 0);
        noiseCovariance.put(4, 4, 1.0);

        kf.measurement_noise_cov(noiseCovariance);

        kf.process_noise_cov(noiseCovariance);
        // Measures Noise Covariance Matrix R
        */
       return kf;
    }
}
```