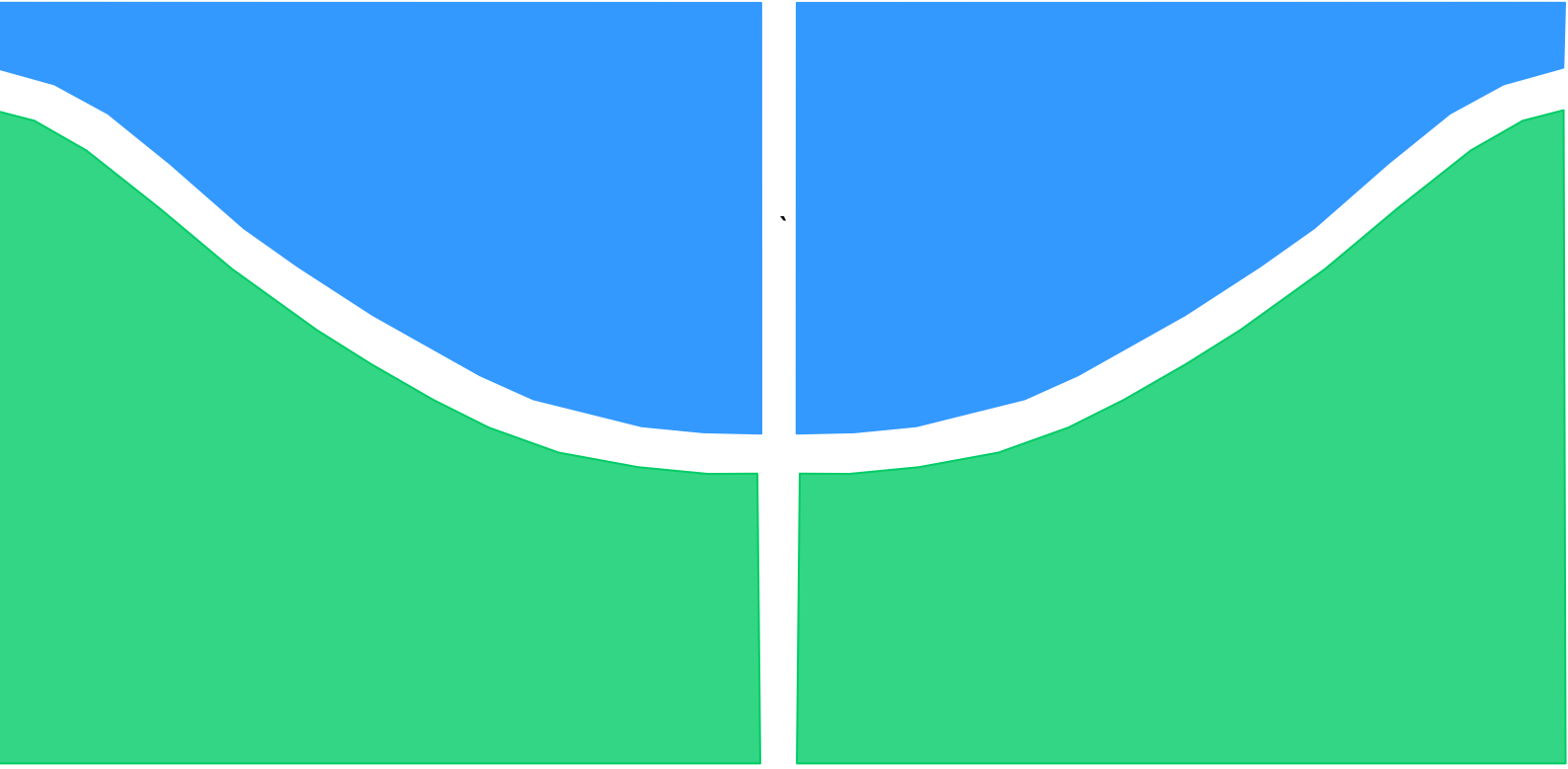
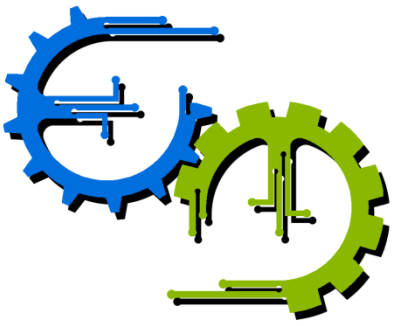


TRABALHO DE GRADUAÇÃO

DESENVOLVIMENTO DE UM SISTEMA SERVO VISUAL PARA CONTROLE DE CÂMERA DE VIGILÂNCIA

NATHAN FABIANO MACHADO

Brasília, dezembro de 2019



ENGENHARIA MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

DESENVOLVIMENTO DE UM SISTEMA SERVO VISUAL PARA CONTROLE DE CÂMERA DE VIGILÂNCIA

NATHAN FABIANO MACHADO

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro de Controle e Automação.

Banca Examinadora

Prof. Walter de Britto Vidal Filho, UnB/ ENM
(Orientador) _____

Eng. Fernando Merege, ANTAQ _____

Prof. José Maurício S. T. da Motta, UnB/ ENM _____

Brasília, dezembro de 2019

FICHA CATALOGRÁFICA

MACHADO, Nathan F.
DESENVOLVIMENTO DE UM SISTEMA SERVO VISUAL PARA CONTROLE DE CÂMERA DE VIGILÂNCIA
[Distrito Federal] 2019.

x, 63p., 210 x 297 mm (FT/UnB, Engenheiro, Controle e Automação, Ano). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. Visão computacional
2. Controle Servo Visual
3. Detecção de Pedestres
4. Câmera PTZ

I. Mecatrônica/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

MACHADO, Nathan F., (2019). Desenvolvimento de um Sistema Servo Visual para controle de câmera de vigilância. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº01 , Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 63p.

CESSÃO DE DIREITOS

AUTOR: Nathan Fabiano Machado

TITULO: Desenvolvimento de um Sistema Servo Visual para controle de câmera de vigilância.

GRAU: Engenheiro ANO: 2019

Nathan Fabiano Machado
CLN 406, bloco B, sala 208 – Asa Norte.
70847-520 Brasília – DF – Brasil.

AGRADECIMENTOS

Agradeço aos meus pais, Nilson e Solange, pelo amor e apoio em cada passo da minha vida, aos meus irmãos, que mesmo longe, sempre estiveram comigo.
Agradeço a minha companheira Aline Bezerra que me apoio durante toda a jornada e ajudou sempre com muito carinho.
Agradeço também ao meu orientador, Professor Walter de Britto Vidal Filho, a orientação e a confiança depositada em mim.

Nathan Fabiano Machado.

RESUMO

Os sistemas de câmeras de vigilância estão ganhando uma importância hoje em dia devido ao aumento da criminalidade. As câmeras podem ser fixas ou móveis e podem capturar as imagens continuamente ou quando existe movimento num setor. Geralmente as câmeras fixas tem o problema de capturar uma imagem ruim quando a atividade criminosa ocorre numa região fora do foco. A câmeras com movimento pan-tilt-zoom, também chamadas PTZ, podem focar em várias regiões a comando de um operador. O Objetivo deste trabalho é desenvolver um sistema que controle uma câmera pan-tilt-zoom para monitorar atividades de forma que possa focar em indivíduos sem a necessidade de um operador humano. Esse sistema vai evitar o que ocorre normalmente, quando indivíduos são filmados, mas as imagens não servem para identificação, pois o indivíduo estava em uma posição fora do foco. Desta forma o sistema deve monitorar uma região, identificar pessoas, acionando e controlando uma câmera, como se fosse um operador humano.

Palavras Chave: Visão Computacional; Controle Servo Visual; Câmera PTZ; Detecção de Pedestres, Rastreamento.

ABSTRACT

Surveillance camera systems are growing up due to the rise in criminality. Cameras can be fixed or mobile and can capture images continuously or when there is movement in a sector. Fixed cameras often have the problem of capturing a bad image when criminal activity occurs in an out-of-focus region. Pan-tilt-zoom cameras, also called PTZ, can focus on various regions at the command of an operator. The objective of this paper is to develop a system that controls a pan-tilt-zoom camera to monitor activities, then it can focus on individuals without the need for a human operator. This system will avoid what normally happens when individuals are filmed, but the images has no details about the suspect, because the subject was in an unfocused position. Thus the system must monitor a region, identify persons, triggering and controlling a camera, as if it were a human operator.

Keywords: Computer Vision; Visual Servoing; PTZ Camera; Pedestrian Detection; Tracking

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

$e(t)$	Erro do sistema em função do tempo	
s^*	Valor de referência do sistema	
s	Valor calculado pelo sistema	
$m(t)$	Posição do objeto na imagem	
a	Parâmetros da câmera	
w	Velocidade angular da câmera PTZ	[rad/s]
v	Velocidade do objeto	[m/s]
L	Distância do objeto à câmera	[m]

Grupos Adimensionais

K_p	Constante proporcional
K_i	Constante integral

Subscritos

max	máxima
m	média
z	zoom

Siglas

PTZ	*Pan, Tilt and Zoom*
FPS	*Frames per second*
IBVS	*Image Based Visual Servoing*
PBVS	*Position Based Visual Servoing*
RGB	*Red, Green and Blue*
HSV	*Hue, Saturation and Value*
HOG	*Histogram of Oriented Gradients*
SVM	*Support Vector Machine*
ASEF	*Average of Synthetic Exact Filters*
MOSSE	*Minimum Output Sum of Squared Error*
KCF	*Kernalized Correlation Filter*
MIL	*Multiple Instance Learning*

CAPÍTULO 1 - INTRODUÇÃO

O uso de câmeras de monitoramento em espaços de circulação pública e em áreas privadas se intensificou nos últimos anos. Segundo dados da Social Progress Imperative, o Brasil é o 11º país mais inseguro do mundo, o que reflete em grande potencial de aplicação de tecnologias de vigilância e segurança (Exposec, 2018).

A utilização desses dispositivos de vigilância, geralmente definidos como um recurso para inibir assaltos, evitar depredações e identificar criminosos, é um fenômeno cada vez mais recorrente, de tal forma que hoje em dia é difícil percorrer ruas, praças, parques, shoppings, aeroportos ou outras áreas de circulação pública, sem deparar-se com sistemas de vigilância. Só na cidade de São Paulo há um milhão de câmeras instaladas – uma para cada 7 habitantes (Exposec, 2018).

Em sistemas de monitoramento e segurança, as câmeras são de grande valia para o reconhecimento de um criminoso, mas muitas vezes a qualidade da imagem, a distância focal da câmera e os pontos cegos da câmera dificultam o entendimento da ação feita pelo indivíduo e a sua correta identificação. Para contornar esses problemas, as câmeras *pan-tilt-zoom* (*PTZ*) são utilizadas geralmente para cobrir grandes áreas, pois possuem um dispositivo motorizado acoplado à câmera em si, possibilitando sua rotação segundo dois eixos distintos, pan e tilt (KIKUCHI, 2007). A rotação de seus eixos e a possibilidade de Zoom fazem da câmera PTZ uma ótima opção para focar em um indivíduo ou objeto suspeito e adquirir imagens mais detalhadas do mesmo, podendo rastreá-lo durante todo o percurso dentro do campo de visão da câmera. No entanto é necessário um operador humano para manusear a câmera e controlar seu movimento para identificar alguma atividade suspeita.

Na maioria dos casos, o operador humano trabalha com múltiplas câmeras simultaneamente e precisa visualizar possíveis atividades suspeitas em cada uma delas separadamente. Ao monitorar múltiplas câmeras, o operador pode não perceber alguma atividade suspeita ou não conseguir focar no alvo desejado a tempo de adiquirir uma imagem nítida e que siga todo o trajeto do suspeito.

1.1 OBJETIVO

O objetivo deste trabalho é automatizar uma câmera *pan-tilt-zoom*, para que o operador humano não precise manipular os controladores de câmeras PTZ para rastrear e focar em indivíduos suspeitos em locais de vigilância. Para isso, técnicas de visão computacional são utilizadas para determinar a localização de atividades a partir das imagens obtidas em conjunto com um sistema de rastreamento de alvo em malha fechada, também chamado de controle servo visual.

1.2 ESTRUTURA DO DOCUMENTO

O Capítulo 2 é dividido em quatro seções. A Seção 2.1 trata conceitos de controle servo visual e as principais abordagens utilizadas na literatura. A Seção 2.2 apresenta conceitos de pré-processamento de imagens, explicitando os algoritmos base para melhorar o desempenho dos processos subsequentes. Na Seção 2.3, são estruturados os detectores e rastreadores e explicitados os principais detectores e rastreadores para realização deste trabalho. Já a Seção 2.4 refere-se as tecnologias das câmeras e como eles são estruturadas.

O Capítulo 3 refere-se à abordagem de controle proposta neste trabalho e está dividido em quatro seções. A Seção 3.1 aborda o problema de forma macro e avalia a estrutura de toda a hierarquia do projeto. A Seção 3.2 define os materiais utilizados e explica a metodologia utilizada para estabelecer comunicação com a câmera. Na seção seguinte, são detalhados os processos de interpretação das imagens, abordando o detector e o rastreador. Por ultimo, a Seção 3.4 discute sobre o controlador utilizado para manter a estabilidade da câmera PTZ ao rastrear o alvo.

O Capítulo 4 avalia os resultados das sobre o detector, rastreador e controlador. Nas duas primeiras seções, são abordados o detector e os rastreadores de forma isolada. Na ultima seção deste capítulo, mostra-se os resultados obtidos para as duas abordagens de controle servo visual desenvolvidas neste trabalho

Por fim, o Capítulo 5 encerra com as conclusões e explicita trabalhos futuros nessa área.

CAPÍTULO 2 – REFERENCIAL TEÓRICO

2.1 CONTROLE SERVO VISUAL

Tradicionalmente, a informação do sensoriamento visual recebido de uma câmera e a manipulação da câmera eram combinadas em uma malha aberta, denominada “ver e mover” (“*look and move*”), que mais tarde foi definida como “ver e mover estático” (“static look and move”) (apud MENEZES, 2013) (MUÑOZ, 2011). Segundo Muñoz (2011), esta abordagem tradicional, embora simples, sofre de alta sensibilidade a erros de calibração da câmera e limitam-se a tarefas bem conhecidas, pois durante uma operação, eventuais erros provenientes do sistema de posicionamento do robô ou do sistema de visão tendem a se acumular, prejudicando a precisão dos seus movimentos. Assim, em 1979, Hill e Park (apud KIKUCHI, 2007) apresentam o conceito de controle servo visual (*visual servoing*) para descrever um controle mais preciso, no qual a informação visual obtida pela câmera é utilizada em um laço de realimentação (Figura 2.1).

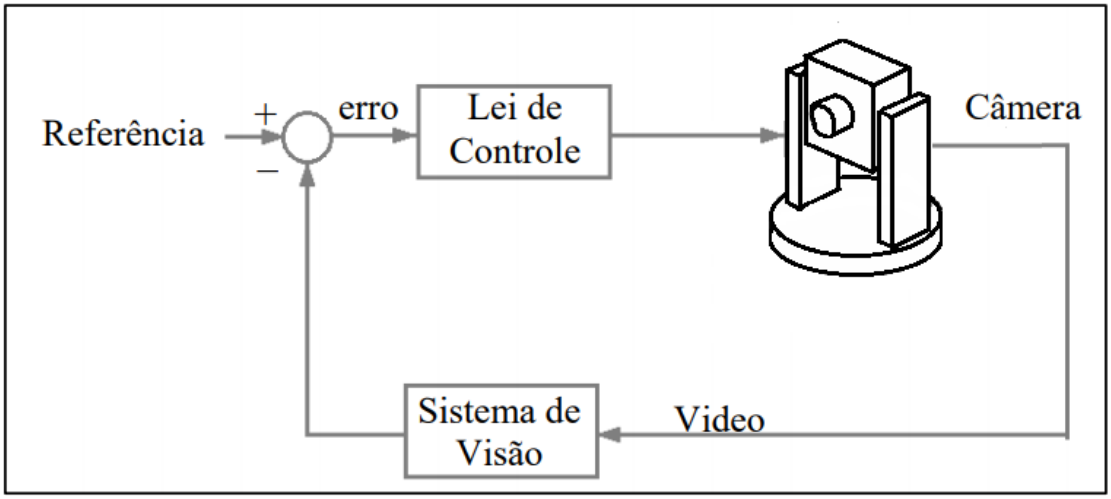


Figura 2.1: Esquemático de um sistema de controle servo visual (adaptado de Munõz (2011) e Kikuchi (2007)).

O termo controle servo visual refere-se ao uso, em malha fechada, dos dados provenientes de visão computacional para determinar a ação de controle de um robô, com o objetivo de realizar uma tarefa. Segundo Bernardes (2009), uso de câmeras como sensores em tarefas de controle tem sido bastante estudado e explorado nas últimas décadas devido a algumas características, dentre as quais se destacam:

● serem sensores de medição que não interferem no ambiente de trabalho, possuindo assim baixo desgaste por não ter contato direto com o ambiente de trabalho;

● amplo campo de cobertura, podendo muitas vezes ter visão de todo o ambiente de

trabalho variando os ângulos de rotação nos atuadores acoplados a câmera;

● baixo custo, se comparado a sensores de varredura como lasers e sensores de ultra-som;

● possui uma taxa de amostragem satisfatória para aplicações em tempo real.

Entretanto existem também inconveniências: o alto atraso devido ao processamento, a baixa taxa de amostragem para casos complexos de processamento de imagens e a alta complexidade dos algoritmos para a interpretação dos dados visuais. Ainda assim, esses tempos de atraso são cada dia menores com a constante otimização dos hardwares para processamento de imagens e novas técnicas de visão computacional, além da complexidade de implementação do controle servo visual estar diminuindo a cada dia, por termos acesso a linguagens de alto nível como o Python e a ajuda de bibliotecas livres como OpenCV.

As aplicações para o controle servo visual são bastante amplas, já que a câmera possibilita uma riqueza de informações sobre o ambiente em que o trabalho é desenvolvido. Bernardes (2009) apresenta um sistema de controle servo visual para robôs omnidirecionais que utilizam um sistema binocular de câmeras para fazer a reconstrução tridimensional do ambiente por Filtro de Kalman Estendido. Pereira (2017) utiliza métodos de rastreamento e perseguição de alvo aplicável a robôs móveis terrestres que possuem restrições de movimento. Menezes (2013) aplica as técnicas de controle servo visual a um Veículo Aéreo Não-Tripulado Quadrirrotor. Kikuchi (2007) produz um sistema de controle servo visual para uma câmera pan-tilt com rastreamento por Região de Referência. Esta aplicação, semelhante à desenvolvida neste trabalho, possui um sistema de controle hierárquico para o máximo ganho de precisão no rastreamento do indivíduo.

Esta seção apresenta os principais conceitos de controle servo visual e as abordagens mais utilizadas na literatura.

2.1.1 Hierarquia na estrutura de controle

A estrutura de controle hierárquico é composta por dois ou mais laços de realimentação. Para a câmera *PTZ*: um laço externo, que é o controle servo visual em si, e um interno, responsável pelo controle de posicionamento da câmera, acionando diretamente os atuadores para o correto posicionamento (Figura 2.2). Em geral, robôs manipuladores tem como entradas apenas sinais de posicionamento para suas articulações, já o controlador interno é responsável por realizar corretamente a movimentação, tendo o controle direto sobre os atuadores do robô (KIKUCHI, 2007). O controle hierárquico servo visual é também chamado de “ver e mover dinâmico” (*“dynamic look-and-move”*) (apud MENEZES, 2013).

Em uma arquitetura não-hierárquica, os sinais de controle são aplicados diretamente na movimentação do robô (Figura 2.3). Este tipo de estrutura é pouco empregado, já que a maioria dos robôs industriais e câmeras *PTZ* apresentam controladores internos, acarretando em estruturas hierárquicas que permitem, em geral, maior estabilidade e precisão do sistema. Especialmente para o controle servo visual, o custo computacional da malha externa costuma ser alto devido ao processamento de imagens, fazendo com que a taxa de amostragem seja baixa se comparado ao controle de manipuladores tradicionais. Para evitar problemas com a estabilidade na malha de controle servo visual, é preferível utilizar controle hierárquico, assim o laço interno continua tendo uma alta taxa de amostragem, mesmo que o laço externo tenha uma baixa taxa de amostragem.

Grande parte dos robôs e câmeras PTZ possuem controle interno de posição implementado, por isso este documento foca na malha externa de controle, ou seja, o laço responsável pela aquisição e interpretação das imagens para o funcionamento pleno do controle.

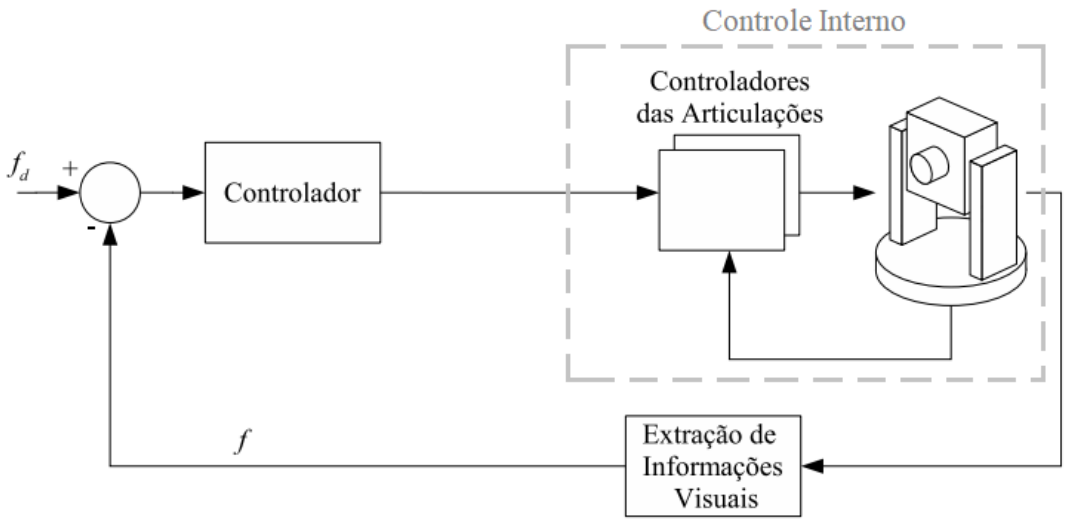


Figura 2.2: Exemplo de controle servo visual hierárquico (adaptado de Kikuchi, 2007).

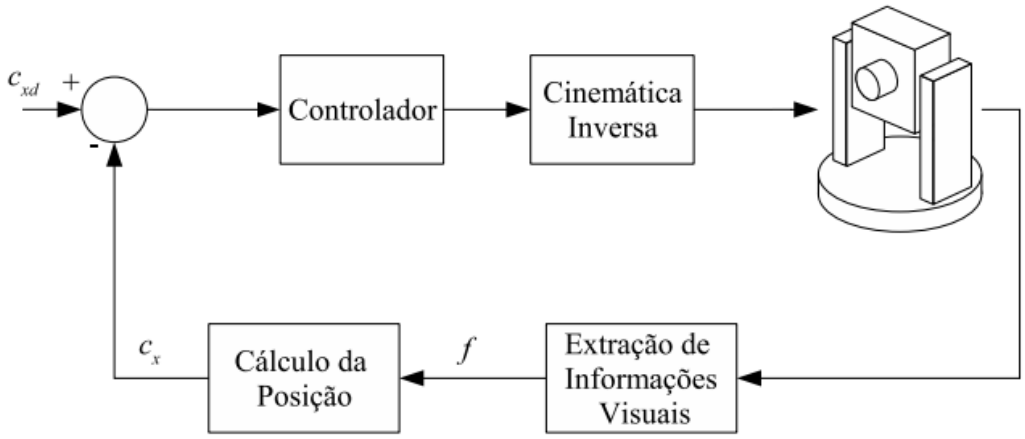
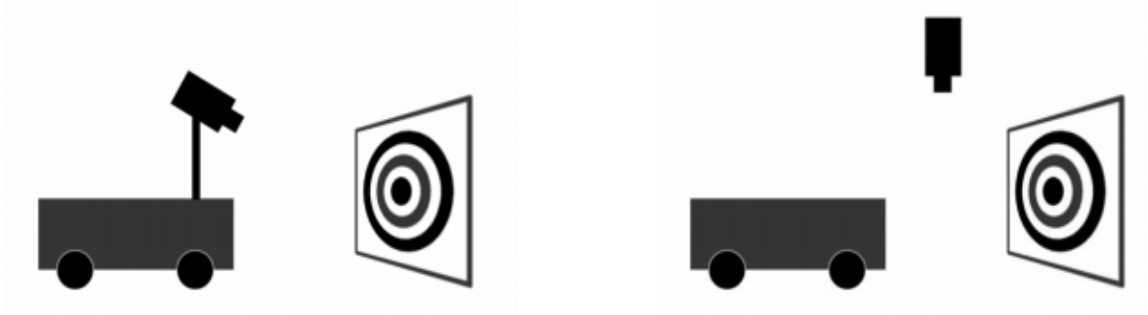


Figura 2.3: Exemplo de controle servo visual não-hierárquico (adaptado de Kikuchi, 2007).

2.1.2 Configuração das câmeras

A configuração do sistema de câmeras pode ser classificada como embarcada ou fixa, também conhecidas como *eye-in-hand* e *eye-to-hand, respectivamente* (Figura 2.4). Na configuração embarcada, a câmera é anexada ao robô e se move junto com ele, ou seja, o movimento do robô atua diretamente no movimento da câmera no espaço. Já na configuração fixa, a câmera fica estática em uma posição no sistema de coordenadas global, de onde adquire as informações visuais do movimento do robô no ambiente de trabalho. A configuração das câmeras não são excludentes, como vê-se em Chang e Shao (2010), é utilizado uma configuração *eye-in-hand* e *eye-to-hand* no mesmo sistema, chamado sistema híbrido.

Neste trabalho, a câmera PTZ possui intrinsicamente a configuração *eye-in-hand* devido aos aspectos construtivos, portanto a configuração *eye-to-hand* não será abordada nas seções subsequentes.



(a) Configuração embarcada ou *eye-in-hand*

(b) Configuração fixa ou *eye-to-hand*

Figura 2.4: Tipos de configuração da câmera (Bernardes, 2009).

Embora a câmera *PTZ* não se movimente pelo ambiente como o robô utilizado por Bernardes (2009), pode-se considerar um sistema *eye-in-hand*, já que a atuação dos motores acoplados à câmera, interferem diretamente na direção e no campo de visão das imagens capturadas.

A configuração da câmera influencia diretamente na escolha e desenvolvimento dos algoritmos para processamentos de imagens. Por exemplo, a detecção de objetos derivada de um detector de movimentos teria dificuldades para detectar corretamente o objeto em movimento em uma configuração *eye-in-hand*, já que o movimento de rotação causado pelos atuadores deixaria todos os componentes da imagem em movimento relativo à câmera.

2.1.3 Controle Baseado em Posição ou em Imagem

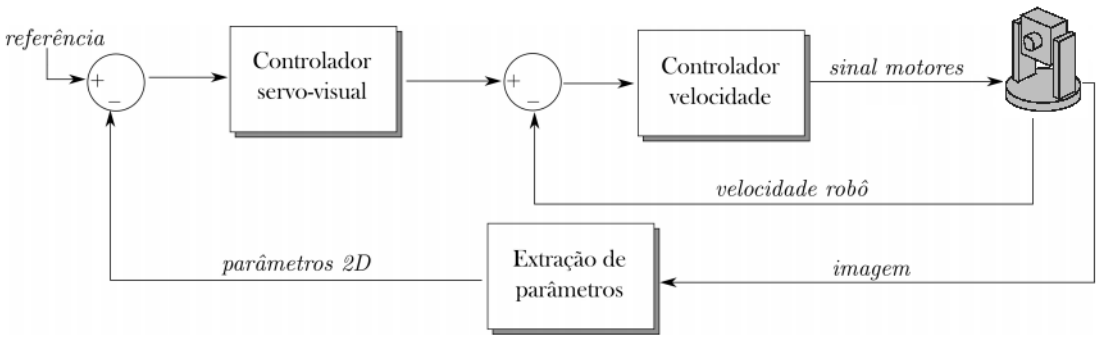
O foco do controle servo visual é a minimização do erro $e(t)$ (CHAUMETTE e HUTCHINSON, 2006), (MUNÕZ, 2011), tipicamente definido por

$$e(t) = s(m(t), a) - s^*$$

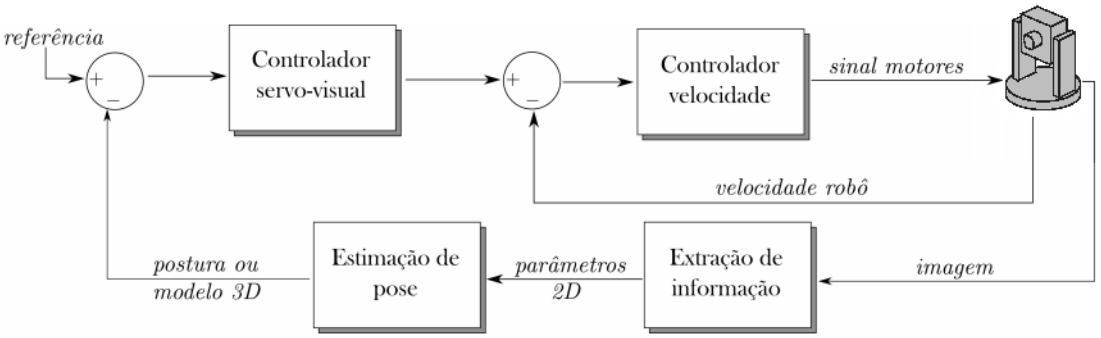
O vetor $m(t)$ é o conjunto de medidas das imagens, por exemplo a coordenada dos pontos de interesse ou as coordenadas do centroide do objeto. O parâmetro a representa os parâmetros intrínsecos da câmera ou os modelos 3D dos objetos alvo, assim pode ser calculado o vetor $s(m(t), a)$. O vetor s^* representa os valores desejados ou valores de referência para o sistema. O design da estrutura de controle pode ser simples, uma vez que s foi selecionado (CHAUMETTE e HUTCHINSON, 2006).

Cada parâmetro $s(m(t), a)$ das características da imagem depende de medidas $m(t)$ de pontos de interesse na imagem e de um grupo de parâmetros a que fornecem informações adicionais sobre o sistema. Para o caso *eye-in-hand*, $m(t)$ está relacionado a postura relativa entre o alvo e a própria câmera, ou seja, $m(t)$ pode ser a posição em píxeis na imagem de um ponto de referência, enquanto s seria a localização desse ponto em coordenadas normalizadas (MENEZES, 2013).

Neste sentido, o controle servo visual pode se diferir entre Controle Baseado em Imagem (*Image-Based Visual Servoing* ou *IBVS*) e Controle baseado em Posição (*"Position-Based Visual Servo* ou *PBVS*) (Figura 2.5).



(a) Sistema de controle baseado em imagem ou *IBVS*



(b) Sistema de controle baseado em posição ou *PBVS*

Figura 2.5: Estratégias de controle hierárquico baseado em imagem e em posição (adaptado de Bernardes (2009)).

No controle baseado em imagem, s é um conjunto de características que estão disponíveis diretamente através da imagem, ou seja, o erro $e(t)$ do controlador é definido como a diferença entre as medições da imagem atual e as medições da imagem final de referência. A abordagem do *IBVS*, também definida como controle 2D, tem menores atrasos computacionais, erros de modelamento do sistema e calibração da câmera. Entretanto o desafio para o controlador é maior, devido aos acoplamentos e não-linearidades da planta.

O procedimento *dynamic look-and-move* define a forma mais simples de implementação de um *IBVS*, pois a partir da imagem mais recente adquirida determina-se os valores dos parâmetros (por exemplo, posição e velocidade) que permitam o alinhamento da câmera com o centroide do alvo. Para estipular tais valores, é necessário determinar o jacobiano de imagem (KIKUCHI, 2007), também denominado matriz de interação (CHAUMETTE e HUTCHINSON, 2006). Esta matriz relaciona o vetor de velocidades dos parâmetros da imagem com a variação de tempo do erro, ou seja, relaciona as velocidades do alvo com as velocidades de rotação *pan* e *tilt* da câmera.

Caso queira estimar a pose ou modelo 3D do alvo em relação a câmera, é necessário a utilização da configuração *PBVS*, também conhecida como controle 3D. O erro, neste caso, é definido como sendo a diferença entre a postura atual do alvo e a postura desejada.

A abordagem *PBVS* pode apresentar maior custo computacional, pois necessita calcular os parâmetros 3D do alvo, como posição e orientação, por isso tem menor robustez aos erros provenientes de modelagem e calibração da câmera. Embora a estratégia *IBVS* seja utilizada com mais frequência devido as características apresentadas pela *PBVS*, a ausência do modelo 3D do alvo em um problema complexo e não-linear, implica maior dificuldade de estabilização no *IBVS*, além de que a existência de mínimos locais podem levar o sinal de controle a zero em posição diferente da desejada (BERNARDES, 2009).

2.2 PROCESSAMENTO DE IMAGENS

O custo computacional envolvido no processamento de informações relevantes em câmeras é elevado, pois esse sensor visual fornece grande quantidade de dados a cada quadro. Por isso, uma alternativa para reduzir a quantidade de dados obtidos da imagem é extrair informações ou características da imagem (*image features*) que possam ser usadas diretamente na lei de controle. Para serem utilizáveis pelo controlador, as *image features* precisam ser rastreadas em cada quadro fornecido pela câmera e é preciso relaciona-las a algum valor numérico (MENEZES, 2013).

Em alguns casos, as imagens capturadas pela câmera são simplificadas para reduzir o tempo de processamento. Uma abordagem comum em técnicas de visão computacional é reduzir as informações das imagens transformando seus dados de *RGB* para imagens apenas com intensidade luminosa ou escalas de cinza. Para casos mais simples, é possível ainda utilizar a limiarização binária, no qual cada píxel da imagem é trocado pelo valor mínimo (equivalente a cor preta) ou valor máximo (equivalente a cor branca), de acordo com um limiar pré-estabelecido para o valor de cada píxel da imagem original.

2.2.1 Imagem Digital

De acordo com Gonzalez e Woods (2010) uma imagem pode ser definida como uma função bidimensional $f(x, y)$ que representa a amplitude f de intensidade do ponto indicado pelo par de coordenadas (x, y). Para uma imagem ser interpretada pelo computador é necessário converter a função $f(x, y)$ para uma forma discreta, este processso é denominado de digitalização.

A digitalização de uma imagem envolve dois passos fundamentais, aquisição e quantização. Na amostragem, discretiza-se o domínio de definição da imagem nas direções x (horizontal) e y (vertical), gerando uma matriz de MxN amostras, respectivamente. Na quantização escolhe-se o valor inteiro I de intensidade para cada ponto da imagem (GONZALEZ e WOODS, 2010). Desta forma, é possível representar a imagem como uma

matriz.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & & f(1,N-1) \\ \vdots & & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

Para cada conjunto de posição (x, y) na matriz, tem-se um *pixel*. A imagem digital possui uma quantidade horizontal e uma quantidade vertical de *pixels,* a esses valores dá se o nome de resolução da imagem (AGREN, 2016). A convenção de coordenadas adotada para imagens é que a origem se encontra no primeiro canto superior esquerdo da imagem.

Em uma imagem digital monocromática ou com apenas um canal, cada valor do *pixel* corresponde a um escalar de intensidade, permitindo a representação das cores em tons de cinza. Tipicamente, em uma imagem colorida cada *pixel* é composto por três valores distintos para representar a cor. Existem vários modelos que definem a forma de decomposição de cores em imagens digitais, como por exemplo os padrões RGB, HSV, CMYK.

Imagens no padrão RGB são matrizes 3D, que podem ser visualizadas como três matrizes 2D distintas, no qual cada matriz 2D corresponde a um dos três canais. A Figura 2.6 mostra a distribuição das cores pelos três canais: *Red, Green* e *Blue*.

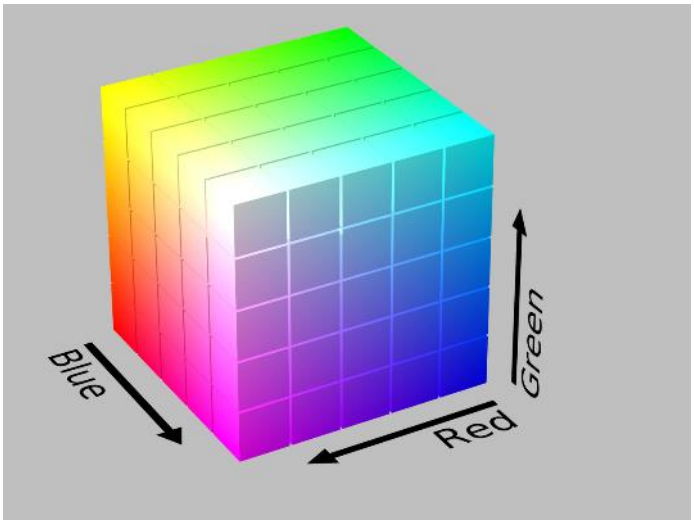


Figura 2.6: Cubo RGB (FRANCO, 2011).

Cada *pixel* possui três bytes, no qual cada byte faz parte de um canal. Com um byte é possível representar o valor de intensidade de uma cor primária em uma faixa de 0 a 255, sendo 0 completamente escuro e 255 completamente intenso (GONZALEZ e WOODS, 2010).

O padrão RGB é comumente utilizado, pois cada canal corresponde a uma das cores primárias aditivas. Assim os monitores e displays podem traduzir de forma direta os valores

de cada byte para exibir uma imagem na tela, por exemplo. No entanto, o padrão RGB consome muita memória quando é preciso processar a imagem, devido aos seus três canais.

Para otimizar o tempo de processamente de cada imagem, primeiro transforma-se a imagem para escalas de cinza, com apenas um canal. A imagem em tons de cinza contém as informações de intensidade da luz em cada pixel, e nos métodos clássicos são calculadas a partir de uma média ponderada entre os três canais do padrão RGB.

Embora exista uma perda de informações na transformação RGB para escalas de cinza, as principais informações de forma dos objetos é preservada e alguns métodos de detecção e rastreamento trabalham melhor com imagens em tons de cinza. Assim pode-se aplicar diferentes processamentos na imagem tendo um custo computacional reduzido e sem perder informações relevantes para este trabalho.

2.2.2 Filtros Digitais

O pré-processamento é uma etapa importante quando trabalhamos com imagens digitais, já que em muitos casos a utilização de fitros para retirar ruídos ou realçar detalhes na imagem podem aumentar o desempenho do sistema. Existem diversos filtros aplicáveis a imagens e esta seção apresenta os principais conceitos sobre filtros digitais.

Os filtros digitais podem suavizar ou realçar detalhes em uma imagem, sendo divididos em filtros passa-baixa, passa-alta e passa-faixa. Ambos modificam o pixel de uma imagem de acordo com os pixeis vizinhos e pode-se categoriza-los em filtros espaciais e espectrais.

Os filtros que veremos são espaciais, já que os filtros espectrais são executados no domínio da frequencia, ao fazer uma Transformada de Fourrier Discreta na imagem e não é necessário a utilização de filtros espectrais para o pré-processamento da imagem.

Os suavizadores de imagem ou filtros passa-baixa, produzem um efeito de borramento nos pixels, reduzindo a intensidade dos pixels de alta frequencia, ou seja, reduzindo a variação de intensidade entre pixels próximos. Dentre os principais suavizadores, temos o filtro de média e o filtro gaussiano, ambos muito utilizados para reduzir ou retirar os ruídos da imagem.

Tanto o filtro de média quanto o filtro gaussiano utilizam máscaras que passam por cada um dos pixels por meio da convolução e suavizam a imagem. A principal diferença entre estes dois filtros são os valores da mascara. Enquanto o filtro de média faz uma média aritmética entre os pixels da máscara, o filtro gaussiano dá maior peso para pixels mais próximos do pixel central da mascara, funcionando como uma distribuição normal.

Os filtros sempre utilizam máscaras de tamanho ímpar, como 3x3, 5x5 e 11x11 por

exemplo. O tamanho da máscara define o quanto aquele filtro vai modificar a imagem, como ilustrado na Figura 2.7.

(a) (b) (c)

Figura 2.7: Imagem original (a) suavizada por um filtro de média 9x9 (b) e 35x35 (c) (Gonzalez e Woods, 2010).

Os filtros passa-alta são utilizados refinar o nível de detalhamento em uma imagem, intensificando a variação dos níveis de intensidade entre píxels vizinhos. No entanto a intensificação do nível de detalhamento pode intensificar tambem os ruídos na imagem.

A principal característica dos filtros passa-alta são os sinais negativos e positivos em suas máscaras, e assim como nos filtros passa-baixa, podem utilizar tamanhos maiores ou menores de máscara para produzir um grande ou pequeno efeito de realçe na imagem.

Neste trabalho não foram utilizados filtros passa-alta, já que este pré-processamento na imagem poderia acarretar em falsas detecções devido a intensificação do ruído.

2.3 DETECÇÃO E RASTREAMENTO DE OBJETOS

A construção de um rastreador de objetos é usualmente dividida em três etapas: representação do objeto, detecção do objeto e rastreamento do objeto (AGREN, 2016). Esta seção revisa diferentes métodos de detecção e rastreamento, comentando sobre os principais passos para a desenvolvimento adequado dos detectores e rastreadores encontrados na literatura.

2.3.1 Representação de Objetos

Para que o rastreamento do objeto seja possível, primeiro deve-se representar o objeto alvo de uma forma que faça sentido para o computador. Para isso, determina-se as características de interesse para a aplicação, que estão diretamente relacionadas com o objetivo do trabalho. No caso deste trabalho, o objeto alvo a ser rastreado é ser um humano que não deveria estar em determinada região, já que estamos lidando com câmeras de

vigilância em sistemas de segurança.

As principais formas de representação dos objetos, segundo Agren (2016), incluem pontos, formas geométricas, silhuetas e contornos, modelos articulados e modelos esqueléticos.

A representação por pontos é a forma mais simples, podendo ser utilizado um único ponto ou múltiplos pontos (Figura 2.8 (a), Figura 2.8 (b)). A abordagem por ponto único é de simples implementação e na maioria das vezes o ponto está localizado no centroide do objeto alvo. Já a representação por múltiplos pontos possibilita uma interpretação mais detalhada do objeto, embora apresente dificuldades no controle quando há oclusão parcial do objeto ou interação entre objetos. Em Munõz (2011), o controle servo visual utiliza a representação do objeto por múltiplos pontos, utilizando o algoritmo Harris.

As retas, retângulos, triângulos e elipses são uma opção para a representação por formas geométricas (Figura 2.8 (c), Figura 2.8 (d)). Bernardes (2009) utilizou a abordagem das retas para a detecção e rastreamento de portas, fazendo um comparativo da abordagem *IBVS* com a *PBVS*. No trabalho de Bernardes (2009) as formas geométricas primitivas se encaixavam bem com as formas dos objetos alvo, entretanto em muitos casos as formas primitivas não são suficientes para descrever bem um objeto, sendo necessário formas mais complexas.

Para a representação da forma do objeto de maneira mais precisa, utiliza-se os modelos por contorno (Figura 2.8 (g), Figura 2.8 (h)) e por silhuetas (Figura 2.8 (i))

Os modelos de articulações (Figura 2.8 (e)) e o modelo esquelético (Figura 2.8 (f)) são utilizados para casos em que não somente a posição do objeto é importante, mas também a sua forma e orientação. Assim, essa abordagem é utilizada quando é necessário o reconhecimento de como o objeto está articulado em cada quadro. O modelo esquelético possui o mesmo padrão do modelo articulado e é bastante utilizado em reconhecimento de objetos, mas não é comum na literatura sua utilização para rastreamento de objetos (AGREN, 2016), devido à complexidade.

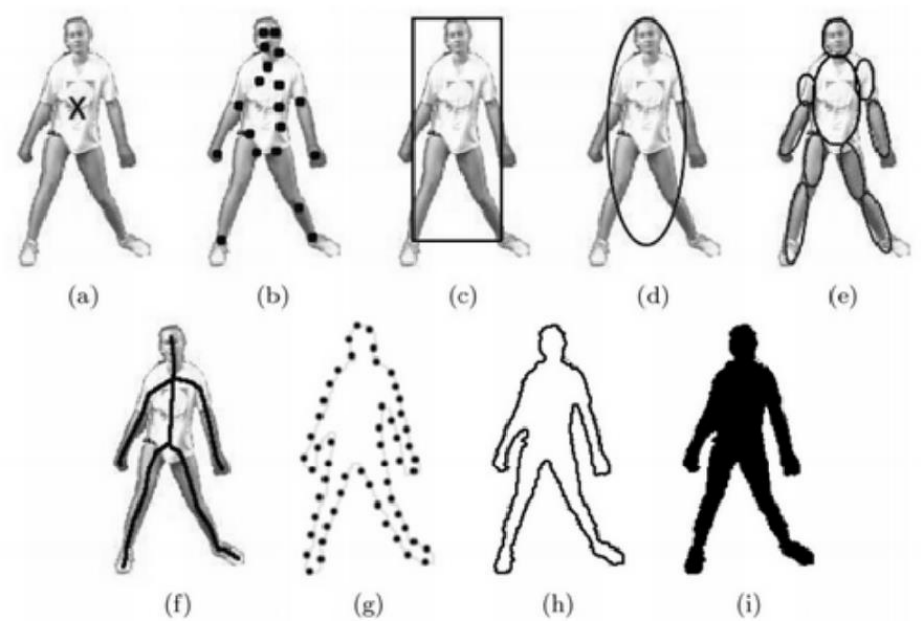


Figura 2.8: Tipos de representação de um objeto (Agren, 2016).

Talvez a parte mais importante em um rastreador de objetos utilizado em controle servo visual seja a seleção correta de características do objeto para que ele seja distinto de outros objetos e do plano de fundo da imagem. Sabendo disso, faz necessário uma correta escolha da representação de forma do objeto e suas características, como por exemplo, bordas utilizadas quando existe grande variação na luminosidade do ambiente de trabalho, já que as bordas são menos sensíveis a luminosidade que a característica cor.

O rastreamento de objetos empregado em controle servo visual, muitas vezes utiliza a representação por ponto único ou formas geométricas primitivas e principalmente características de distinção por arestas ou por cores. Henz (2014) utiliza representação por retângulo, com seleção da cor característica do objeto para identifica-lo e rastreá-lo. Bernardes (2009) define seu rastreamento por identificação de bordas a partir da intensidade das cores, representando os objetos por retângulos.

Vê-se que as características do objeto a ser rastreado são de grande importância para sua detecção na imagem, e cada característica possui vantagens e desvantagens associadas.

2.3.2 Detecção de Objetos

Para que seja possível rastrear um objeto, primeiro é necessário detectar tal objeto na sequência de imagens capturadas pela câmera. Existem duas diferentes abordagens para isso, a primeira, no qual a detecção é feita em cada quadro do vídeo e a segunda, onde a detecção ocorre somente no primeiro quadro em que o objeto aparece (AGREN, 2016). Em ambas abordagens, existem vários métodos de detecção de objetos. Nesta seção veremos

os principais detectores empregados na literatura, como a detecção por pontos, a subtração do plano de fundo e a segmentação, seguida da aprendizagem de maquina ou classificador.

No detector de pontos, pontos de interesse são detectados a partir de características definidas, como por exemplo, textura, intensidade das cores, cantos e pontos de máxima e mínima luminosidade. Pontos de interesses bons são em geral pontos que continuam detectáveis ao ocorrer mudanças de luminosidade no ambiente. Esses pontos costumam fazer parte das bordas de um objeto e existem métodos específicos para a detecção desses pontos, denominado Detector de Harris (*Harris Corner Detector*).

Muñoz (2011) utiliza o detector de cantos de Harris para detectar o objeto e rastreá-lo, indicando sua posição e orientação com alta precisão em um ambiente semiestruturado. Macedo e Amaral (2017) utilizam uma abordagem por detecção de máximos e mínimos pontuais, utilizando filtro de partículas e consegue rastrear objetos com oclusão parcial e com pequenas deformações de rotação, translação e escala. No entanto o custo computacional envolvido é alto e este tipo de detector não é muito utilizado para detectar seres humanos na literatura, devido a baixa flexibilidade de deformação do objeto para que ele seja detectado e rastreado.

Já a subtração do plano de fundo tem um baixo custo computacional e é capaz de detectar objetos em movimento. Este algoritmo isola o objeto alvo e tem alto desempenho em trabalhos no qual há pouco ou nenhuma variação luminosa e o plano de fundo continua sem mudanças durante todo o tempo de captura das imagens. Neste caso, cada quadro é subtraído com o quadro de referência, pixel a pixel. Assim, os valores dos pixels referentes ao objeto serão os únicos pixels não-nulos na imagem.

Em casos de controle servo visual com câmera embarcada, a própria movimentação da câmera faz o plano de fundo variar de quadro a quadro, prejudicando a detecção do objeto. O método de diferenciação temporal é utilizado para que a câmera detecte apenas o objeto, mesmo que exista um pequeno movimento relativo da câmera ou mudanças no plano de fundo ao longo do tempo. Neste método, a imagem atual é subtraída da imagem anterior e existe um compensador para pequenos deslocamentos, assim é possível que o detector encontre o objeto sem problemas de interferência do plano de fundo. Chen, Cheng e Tsai (2002) utilizam este princípio, mas com um modelo baseado nas diferenças obtidas a partir de três imagens consecutivas.

Esta técnica pode ser problemática quando o movimento do objeto é muito lento ou caso ele pare de se movimentar, pois a diferenciação temporal consegue apenas identificar objetos em movimento.

Existem também técnicas mais robustas para a subtração do plano de fundo. Piccardi (2004) analisa a performance de sete diferentes algoritmos de subtração de plano de fundo. Sua análise vai de algoritmos mais simples como o filtro de média temporal, no qual é feito uma média das ultimas imagens capturadas para a subtração, até algoritmos mais complexos que utilizam etapas de aprendizagem e classificação para a subtração do plano de fundo com alta precisão. Embora as técnicas de diferenciação temporal sejam simples e tenha um bom desempenho, nas câmeras *PTZ* essa técnica estaria restrita a detectar com sucesso apenas enquanto a câmera não se movimentasse em nenhum eixo.

Para detectar um ser humano em uma câmera de vigilancia que se movimenta em *pan, tilt* e *zoom*, pode-se utilizar técnicas de segmentação ou extração de características unicas do objeto trabalhando em conjunto com técnicas de aprendizagem de máquina para detectar tais objetos.

A segmentação é o processo de particionar uma imagem em vários segmentos, ou regiões. Os segmentos, juntos, cobrem toda a imagem e são comumente gerados a partir de características de cada segmento, que pode ser baseado em cor ou intensidade. Esta técnica é útil para localizar objetos e limites entre diferentes áreas (AGREN, 2016).

Os segmentos são comumente definidos a partir da intensidade ou da variação de intensidade entre os pixels vizinhos, formando uma espécie de contorno nos objetos. Ainda assim, esses algoritmos podem fazer uma segmentação sem restrição de tamanho dos segmentos, ou podem efetuar uma normalização na extensão das áreas segmentadas. Essas técnicas de detecção são utilizadas em vários tipos de rastreadores que utilizam classificadores e podem detectar e identificar múltiplos tipos de objetos na cena quando aliados a redes neurais, por exemplo.

A aprendizagem supervisionada possibilita a detecção de objetos a partir do treinamento do sistema com o auxílio de um banco de dados com muitas imagens de referência, semelhantes ao objeto que se deseja detectar. Esta técnica evita que seja necessária uma comparação da imagem capturada com as imagens de referência, já que o sistema funciona como um filtro construído para passar apenas características que sejam semelhantes as imagens de referência utilizadas na aprendizagem supervisionada. Existem diversas abordagens para extrair características do objeto a ser detectado e veremos as principais abordagens.

Entre as técnicas clássicas para a detecção de seres humanos, estão: *Histogram of Oriented Gradients (HOG)* e *Haar Cascade*. Ambas as técnicas extraem características da imagem e utilizam aprendizagem de máquina para treinar o sistema a identificar tais características, classificando o objeto.

No HOG, os Vetores de Gradiente são vetores que além de direção e sentido, nos entregam o maior valor a respeito de alguma grandeza (no nosso caso o gradiente equivale para as bordas). Dessa forma o vetor de gradiente possui direção, sentido e magnitude apontando para onde há o maior contraste relativo as bordas. Então a imagem é dividida em celulas de mesmo tamanho e são calculadas os vetores de gradiente para cada célula (Figura 2.9). Após calcular os gradientes, os vetores são organizados em um histograma normalizado de acordo com o angulo e a magnitude do vetor.

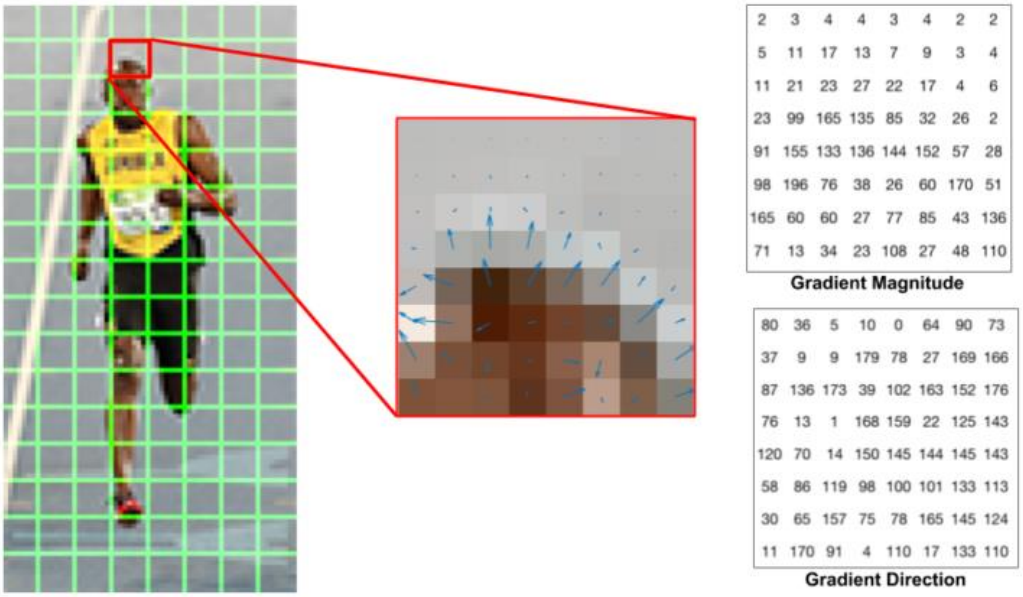


Figura 2.9: Calculo da direção e magnitude do vetor de gradiente em cada célula (MALLICK, 2016).

Na biblioteca OpenCV, o Histograma de Gradientes Orientados é dividido em 9 camadas correspondendo a divisões nos angulos dos vetores de zero a 180 graus. Ou seja, as camadas são distribuidas de 20 em 20 graus, correspondendo a uma faixa de valores no histograma, e cada célula dessa camada representa a distribuição de magnitude naquele ponto. Atráves de técnicas de aprendizagem supervisionada, como a *Support Vector Machine* (SVM), as amostras são separadas em verdadeiras e falsas de acordo com os rótulos e suas características. Assim é possível detectar objetos semelhantes a aqueles treinados pelo sistema.

Outra alternativa classica seria a detecção de objetos por *Haar Cascade*, no qual são utilizados filtros de Haar para extrair caracteristica do objeto, como cantos, linhas e circulos. As características extraidas em cada janela deslizante alimentam a cascata, que consiste de uma arvore de decisões onde os níveis mais profundos possuem um maior nivel de detalhamento sobre as características do objeto. Caso haja uma saída negativa em qualquer um dos níveis aquela janela é rejeitada (Viola e Jones, 2001).

Viola e Jones (2001) propõem a utilização do Adaboost para treinar o classificador. Devido ao Adaboost reforçar os resultados anteriores tidos como falsos, o classificador *Haar Cascade* possui pequenas taxas de falsos postivos.

Existem ainda as redes neurais convolucionais que possuem bons resultados na detecção de pedestres, no entanto estas abordagens costumam ser complexas computacionalmente e possuem alto custo computacional.

2.3.3 Rastreamento de Objetos

Os rastreadores são utilizados para obter a informação de posição do objeto em uma sequencia de frames e costumam ter um custo computacional menor que o de detectores (BOLME, 2010). Os detectores podem ser utilizados em apenas uma imagem e estimarem o local do objeto detectado, já os rastreadores precisam de uma sequencia de imagens e a posição inicial do objeto a ser rastreado para que o algoritmo persiga o alvo atualizando sua posição durante os frames.

Devido à grande variedade de algoritmos utilizados para o rastreamento de objetos, é comum separa-los em três grandes grupos denominados rastreamento por pontos (*point tracking*), rastreamento por kernel (*kernel tracking*) e rastreamento por silhuetas (*silhouette tracking*) (YILMAZ, JAVED e SHAH, 2006) (AGREN, 2016), além disso, cada um dos grupos possui subcategorias (Figura 2.10).

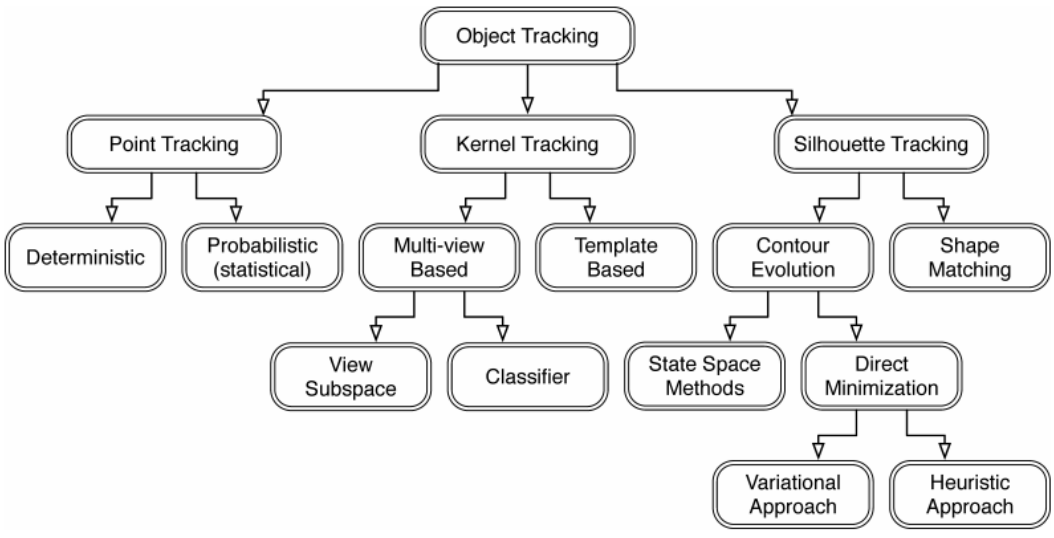


Figura 2.10: Estruturação dos tipos de rastreadores de objetos (AGREN, 2016).

Outra forma de categorizar os métodos de rastreamento de objetos é por rastreamento baseado em contornos e bordas, rastreamento baseado em região de referência e rastreamento por segmentação (KIKUCHI, 2007). No entanto, esta forma de categorizar as

técnicas de rastreamento está entrando em desuso, pois todas essas categorias podem ser consideradas subcategorias de *silhouette tracking* (apud AGREN, 2016).

O rastreamento de pontos é dividido em determinístico e estatístico. Ambos detectam e representam os objetos por pontos, e a cada quadro são associados os pontos do quadro atual com os pontos do quadro anterior. Um dos rastreadores de pontos revisados neste trabalho foi o de Macedo e Amaral (2017), no qual fazem a detecção e rastreamente de objetos planares utilizando filtro de particulas.

Muñoz (2011) rastreia objetos planos em um ambiente semiestruturado, por meio dos pontos de cantos definidos pelo algoritmo de Harris. A associação dos pontos do quadro anterior com o atual é feita por casamento robusto, utilizando algumas restrições citadas por Agren (2016) para a otimização de desempenho do processo.

Segundo Agren (2016), as restrições sobre os pontos podem ser separadas em: proximidade dos pontos correspondentes; pequenas mudanças de velocidade; orientação comum do movimento para pontos próximos; e rigidez, que assume que as distâncias entre dois pontos será a mesma. Essas limitações permitem que o rastreamento por pontos tenha um custo reduzido de processamento. Ao retirar essas restrições, em alguns casos, o custo computacional pode ser elevado a ponto de ser inviável para sistemas em tempo real.

As restrições propostas por Agren (2016) reduz o leque de aplicações para o qual o rastreamento por pontos é eficaz. Portanto esta categoria de rastreadores não possui bons resultados no rastreamento de seres humanos, que podem estar em diversas poses e possuir variações de tamanho e forma.

Considerado menos complexo que outras abordagens, o rastreamento por kernel utiliza a semelhança entre regiões próximas ao objeto para rastreá-lo. Esta abordagem utiliza regiões definidas por formas geométricas primitivas como quadrados e elipses para representar a área no qual o objeto está contido. Podem ser utilizadas diferentes características, por exemplo cor e intensidade, para a comparação e verificação de semelhança entre as regiões do quadro atual, próximas a região do objeto no quadro anterior. Assim é possível encontrar o objeto, assumindo que na passagem de um quadro para outro, haverá uma pequena mudança na posição e no formato do objeto.

O rastreamento de kernel é popular devido a sua baixa complexidade e alta robustez (AGREN, 2016), já que pode utilizado para o rastreamento de objetos desconhecidos e possui versões de rastreamento de múltiplos objetos e interações entre objetos.

A representação de objetos por formas primitivas ou pontos pode não ser o suficiente,

devido à baixa acurácia para objetos complexos. O rastreamento de silhueta possibilita uma alta precisão do formato do objeto (apud AGREN, 2016).

Rastreadores de silhueta utilizam a comparação de características de regiões próximas à região onde se encontrava o objeto no quadro anterior. Esta técnica é semelhante ao rastreamento de kernel, no entanto as características principais utilizadas para o casamento das regiões são as cores e o histograma de arestas. Outra abordagem desta técnica é encontrar a correspondência a partir da silhueta do objeto, possibilitando uma visualização mais completa do objeto. O casamento de silhueta se difere do rastreamento de pontos, porque além de considerar o movimento e posição, utiliza também características de aparência do objeto (apud AGREN, 2016).

Para este trabalho, testou-se dois rastreadores de kernel: o *Kernalized Correlation Filters* (KCF) (HENRIQUES, 2015) e o *Minimum Output Sum of Squared Error* (MOSSE) (BOLME, 2010), ambos utilizando filtros correlacionais para o rastreamento.

Segundo Bolme (2010), rastreadores que utilizam filtros correlacionais possuem uma performance equivalente a de rastreadores mais complexos, mas exigindo menor custo computacional.

O algoritmo MOSSE encontra um filtro que minimiza a soma do erro quadrático entre a saída real e a saída desejada da convolução. O rastreamento utilizando MOSSE chega a 669 frames por segundo nos testes executados por Bolme (2010), possuindo um dos melhores desempenho entre os rastreadores atuais.

O desenvolvimento do rastreador MOSSE se deu através de modificações realizadas no rastreamento por ASEF (Average of Synthetic Exact Filters), no qual ele melhorava os filtros correlacionais para uma determinada tarefa. No entanto eram necessárias muitas imagens para treinar o ASEF. O rastreamento por MOSSE utilizou princípios do ASEF, mas com aprendizagem online, ou seja, o algoritmo é treinado enquanto os códigos estão sendo executados. Devido a característica de aprendizagem online do rastreamento MOSSE, ele é tolerante a variações e deformações na rotação, escala e iluminação, além de rastrear objetos com oclusão parcial.

Assim como o filtro MOSSE, o rastreador KCF possui treinamentos online. O treinamento se dá com amostras positivas e negativas do objeto a ser rastreado, no qual as amostras positivas são o proprio objeto alvo em cada frame e as amostras negativas são recortes aleatorios das partes da imagem em que o objeto rastreado não se encontra.

Os filtros correlacionais kernalizados (KCF) são uma técnica para treinar online um

classificador, focando em obter uma quantidade massiva de amostras negativas aleatórias ao redor do objeto a ser rastreado (HENRIQUES, 2015), tendo assim uma boa validação do classificador para rastrear o objeto.

Os testes de performance realizados por Henriques (2015) mostra que KCF obteve uma taxa de 172 frames por segundo enquanto MOSSE obteve 615 frames por segundo. No entanto não é necessário uma taxa de frames tão alta, já que as câmeras tradicionais no mercado possuem uma taxa de captura de imagens entre 25 e 60 FPS.

2.4 TECNOLOGIAS

As características intrínsecas da câmera são de extrema importância para determinar as condições do projeto de controle servo visual, já que características como resolução de imagem, quadros por segundo e velocidade de rotação dos eixos podem afetar diretamente na escolha do algoritmo de rastreamento e na estratégia de controle utilizada.

Nesta seção serão listados os principais tipos de câmeras de segurança disponíveis no mercado, a estrutura geral das câmeras PTZ e os principais softwares utilizados para o desenvolvimento sistemas de visão computacional.

2.4.1 Tipos de câmeras

A vasta gama de câmeras de vigilância disponíveis no mercado se dá ao aumento de demanda para diferentes tipos de aplicações. A utilização de câmeras em sistemas de segurança é indispensável e seus tipos definem as características de construção da câmera. Segundo a *Tyco International*, os principais tipos de câmeras de segurança são:

● câmera box, geralmente utilizadas em ambientes externos, devido a sua capacidade de customização das lentes para cada necessidade;

● câmera dome, muito utilizada para ambientes internos por sua estética agradável;

● câmera bullet, que geralmente inclui iluminadores infravermelho para visão noturna ou com pouca iluminação;

● câmera IP, que trabalham com sinal digital, possibilitam uma fácil comunicação com outros dispositivos por meio de protocolo de internet e podem ter conexão wireless;

● câmera termal, utilizada para áreas com nenhuma iluminação;

● câmera PTZ, utilizada para visualizar grandes áreas com a movimentação pan e tilt, além de ser possível a visualização de detalhes do alvo com o zoom;

Para cada uma dessas características construtivas, existe uma vasta gama de marcas e modelos que determinam a resolução, a taxa de quadros do vídeo e a entrada/saída de sinal que pode ser analógico ou digital. Neste trabalho, estamos interessados unicamente nas câmeras PTZ devido ao seu grande campo de visão dado pelos atuadores no eixo vertical e horizontal.

2.4.2 Estrutura de uma câmera PTZ

As câmeras *pan-tilt-zoom* são formadas pelo corpo da câmera e sua base. A base da câmera possui dispositivos motorizados que permitem a rotação da câmera no eixo horizontal (*pan*) e vertical (*tilt*), além do controle de zoom para detalhamento de um objeto alvo. Os movimentos de *pan* e *tilt* permitem que a câmera possua um campo de visão muito vasto ao seu redor. No entanto, as câmeras *PTZ* possuem restrições de movimento devido aos aspectos construtivos.

A restrição angular do eixo vertical e em alguns casos também do eixo horizontal faz com que a câmera tenha alguns pontos inalcançáveis. Na prática, essa restrição influencia o controle servo visual quando um objeto alvo se desloca para um ponto cego da câmera, no qual a variação nos ângulos *pan* e *tilt* não conseguem acompanhar o objeto.

A velocidade angular da câmera PTZ também é um fator importante no sistema. Em alguns casos onde a velocidade de rotação é fixa e não pode ser alterada, é necessário utilizar sinais de pulsos com comandos sequenciais de *move* e *stop* para os atuadores, no entanto esta estratégia pode causar vibrações no sistema e afetar muito a qualidade da imagem, dificultando o rastreamento do objeto. Câmeras PTZ utilizadas em grandes áreas costumam ter um controle de velocidade embutido, diminuindo significativamente problemas com vibrações na movimentação. Ainda assim, existem casos em que a velocidade máxima de rotação da câmera PTZ não é o suficiente para conseguir acompanhar o movimento do alvo, fazendo com que o sistema perca o objeto a ser rastreado.

Além das restrições de posição e velocidade angular, existe ainda o sistema de comunicação entre o software de controle servo visual e a câmera. O meio mais comum para recepção e transmissão de dados a uma câmera PTZ é utilizando protocolo IP. Com uma rede local ou por meio da internet, é possível receber as informações visuais e transmitir os comandos de movimentação para a câmera. No entanto é preferível utilizar redes locais, pois a comunicação via internet está sujeita a maiores latências, podendo afetar diretamente o sistema de controle servo visual.

Para evitar a possibilidade de alta latência na comunicação câmera-software, Kikuchi (2007) utiliza uma plataforma previamente construída para funcionar como a base da câmera

PTZ. Assim é fixada uma câmera tradicional no topo da base, possibilitando os movimentos *pan* e *tilt*, com a facilidade de se ter um controle direto por sinal digital dos servo-motores que rotacionam a base.

CAPÍTULO 3 – MATERIAIS E MÉTODOS

Neste trabalho se apresentam duas metodologias de controle para a câmera PTZ, a primeira com o detector enviando diretamente informações de referencia para o controlador, e a segunda no qual um detector e um rastreador trabalham em conjunto para extrair informações e utiliza-las no controlador. A Figura 3.1 esquematiza o sistema, no qual a segunda metodologia utilizou todos os blocos para o controle servo visual e a outra não utilizou o bloco de rastreamento (bloco II).

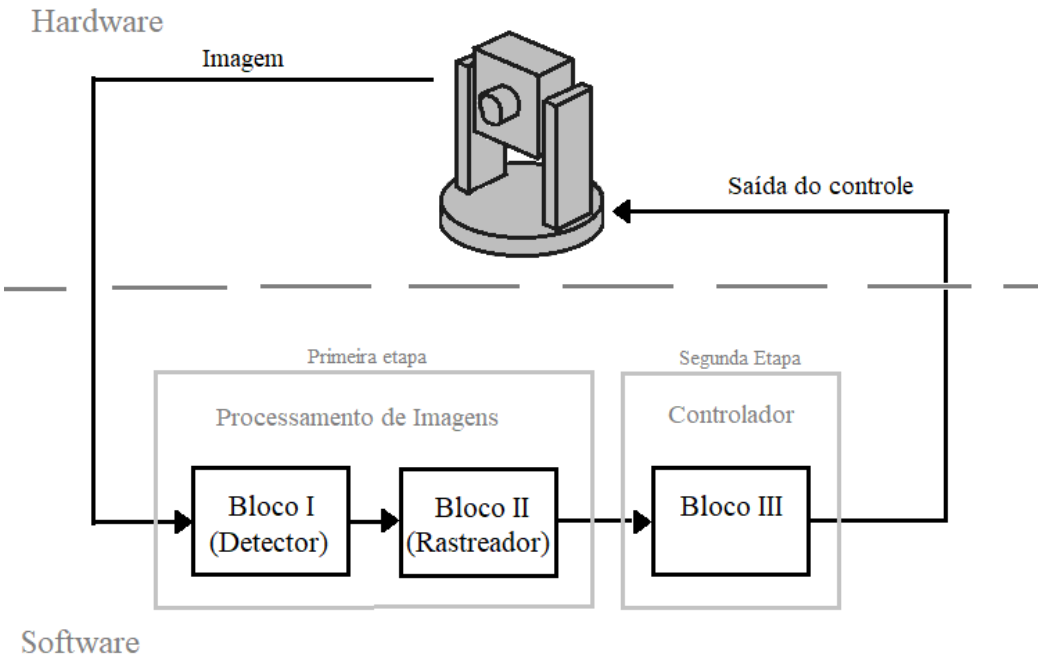


Figura 3.1: Estrutura do sistema de controle servo visual.

Para ambas as metodologias, o controle servo visual foi dividido em duas etapas. A primeira etapa preocupa-se com a correta interpretação dos dados capturados pelo sensor visual, assim é possível transformar tais dados em informações, como posição e velocidade do objeto, por exemplo.

A outra etapa envolve o controle dos atuadores acoplados a movimentação em *pan*, *tilt* e *zoom*. Esta etapa trata da lei de controle e estabilidade do sistema, sendo crucial para que o sistema consiga focar no invividuo alvo sem grandes atrasos e com o mínimo de características vibratórias, que poderiam comprometer a qualidade da imagem capturada e desestabilizar o sistema.

O software para o controle servo visual foi feito de forma modular, assim é possível substituir blocos de código ou retira-los a fim de modificar apenas uma parte sem prejudicar o todo. Este modo de trabalho facilitou a produção das duas metodologias, já que elas possuem os blocos de detecção (bloco I) e de controle (bloco III) em comum, e se diferenciam

apenas na utilização do rastreador, no qual no qual são extraidas as informações de posição do alvo na imagem.

O controle servo visual de uma câmera *PTZ* para sistemas de segurança deve considerar diversas restrições. Um dos principais problemas do uso de sensores visuais é o custo de processamento, que pode causar uma resposta lenta e comprometer o sistema de controle. Isso se deve ao fato de que a imagem, por si só, não pode ser, de forma simples, convertida no sinal de controle para a atuação. Em geral, as propriedades das imagens exigem muitas etapas de processamento para que, a partir do sensor visual, estime-se os dados úteis para o controlador. Dessa forma, deve-se considerar todos os requisitos e limitações físicas do projeto no desenvolvimento do software.

Neste capitulo são explicitados os hardwares, a escolha de toda a estrutura do controle servo visual e o seu desenvolvimento. Assim como o definição da planta do sistema, o desenvolvimento do software para processamento das imagens e a escolha da lei de controle aplicada na câmera em cada metodologia, criando uma comparativo entre elas.

3.1 ESTRUTURA DO SISTEMA DE CONTROLE SERVO VISUAL

Antes de definir os requisitos e restrições do sistema, é necessário entender a estruturação do controle servo visual utilizado. Como observado na seção 2.1, as câmeras *PTZ* são por definição *eye-in-hand*, ou seja, o acionamento dos atuadores do sistema interfere diretamente no campo de visão da câmera.

Sabendo do modelo físico do sistema, é importante determinar qual é a melhor abordagem de controle para a solução do objetivo proposto neste trabalho: o controle servo visual baseado em imagem (*IBVS*) ou o controle servo visual baseado em posição (*PBVS*).

Muñoz (2011) faz um comparativo entre a abordagem *IBVS* e *PBVS* para o rastreamento de objetos por pontos, utilizando o algoritmo de Harris adaptado. Foi identificado que para condições ideais, sem erros de calibração da câmera e erros de modelagem do objeto, a abordagem *PBVS* rastreou o objeto possuindo um menor tempo de execução da tarefa e melhor deslocamento relativo da câmera. No entanto, ao adicionar erros na calibração da câmera e na modelagem do objeto para o sistema *PBVS*, a abordagem *IBVS* se mostrou muito mais eficaz.

Segundo Chaumette e Hutchinson (2006), a abordagem *PBVS* depende de parâmetros intrínsecos da câmera e do objeto alvo, fornecendo dados de posição, velocidade e orientação, por exemplo. No entanto, estamos interessados exclusivamente nos dados de posição, relativa à câmera, do indivíduo.

Devido a abordagem *PBVS* necessitar parâmetros pré-estabelecidos do objeto alvo e

ter falhas de desempenho ao se introduzir erros na calibração da câmera, optou-se pela utilização da abordagem *IBVS,* no qual o erro do controlador é calculado diretamente através da imagem.

A partir da Figura 3.1, pode-se analisar a hierarquia do sistema. Ela consiste de três malhas de controle, distribuidas em duas camadas distintas. A primeira etapa ou primeira malha de controle envolve exclusivamente os elementos do software de detecção e rastreamento do objeto, no qual o programa utiliza uma realimentação das características do objeto para o rastreamento do mesmo. Assim como a primeira malha, a segunda malha também é definida na camada de software, sendo responsável por calcular e enviar os comandos para os controladores internos de *pan*, *tilt* e *zoom* da câmera, com base nas informações de posição do alvo, extraidas da primeira malha. Na camada de hardware, encontra-se a malha de controle responsáveis por atuar cada um dos graus de liberdade da câmera *PTZ*, sendo eles o eixo de rotação horizontal "*pan"*, o eixo de rotação vertical *"tilt"* e o dispositivo de deslocamento relativo entre as lentes, para o ajuste do *zoom*.

Nosso foco será no desenvolvimento da camada de software, já que as principais câmeras *PTZ* no mercado possuem controladores internos para cada grau de liberdade. Uma importante característica adotada no desenvolvimento do software do controle baseado em imagem foi a capacidade de encapsulamento de diferentes partes do software. Como observa-se na figura 3.1, os blocos I, II e III são independentes, e podem ser substituídos por outros algoritmos que possuam o mesmo objetivo, detectar o objeto, rastrear o objeto e controlar a câmera, respectivamente. A capacidade de modulação e encapsulamento será discutida em mais detalhes na seção 3.3.

Para a camada de software, é de vital importância o conhecimento das restrições físicas do sistema e a imposição de limitações no sistema para que o controle servo visual seja eficiente. Kikuchi (2007), conhecendo as restrições do seu projeto, assume em seus algoritmos de rastreamento do objeto, que a posição do objeto a ser rastreado sempre se inicializaria no centro da imagem capturada pela câmera. Este tipo de limitação imposta ao sistema diminui sua usabilidade, no entanto é necessário em alguns casos, tanto para evitar problemas muito complexos computacionalmente, quanto para evitar restrições físicas que o sistema não consegue lidar.

As principais restrições físicas do sistema deste trabalho estão relacionadas a velocidade angular de *pan* e *tilt,* a taxa de captura de imagens, o tempo de processamento de cada imagem para extração das informações de posição do objeto e a velocidade de deslocamento entre lentes para ampliar ou reduzir a imagem.

3.2 CAMERA PTZ

Neste trabalho foram testadas 2 câmeras para o desenvolvimento do controle servo visual. O Anexo I contém todas as especificações de cada uma das câmeras.

Os primeiros testes foram na câmera IP doméstica de baixo custo, modelo IPC-Z06H. Apesar de conseguir acessar o video e os comandos de *pan* e *tilt* através do software disponibilizado pela distribuidora da câmera, não foi possível ter acesso através do endereço IP da câmera.

Foram utilizados outros softwares e bibliotecas *open source*, como o ONVIF Device Manager e o OpenCV, para conseguir acessar o stream de video da camera e os comandos de *pan* e *tilt*, mas ambos tiveram problemas de acesso. Já que os testes iniciais não eram possíveis na camera IPC-Z06H (Figura 3.2a), optou-se pela aquisição da câmera Speed Dome SL-130IPC851 (Figura 3.2b).

(a) Câmera IPC-Z06H (b) Câmera SL-130IPC851

Figura 3.2: Câmeras Pan-Tilt e PTZ utilizadas nos testes.

Com a câmera *Speed Dome* e ajuda do software ONVIF Device Manager, foi possível identificar de forma rápida o IP da câmera e acessá-la através do browser. Em seus manuais não constava nenhuma informação sobre a estrutura da aplicação e qual seria o comando para cada ação *pan*, *tilt* e *zoom*.

Tendo como objetivo capturar as imagens da câmera e controla-la, foi necessário monitorar a rede para encontrar cada uma dos comandos e seu funcionamento. Ao inspecionar a rede com a câmera conectada via cabo, observou-se um padrão nas mensagens enviadas a câmera para controlar seus graus de liberdade. Este padrão possuia o sentido e a velocidade do movimento.

Ao inspecionar a rede, não foi encontrado nenhuma forma de obter a posição angular

absoluta de *pan*, *tilt* ou *zoom* e tambem não foi possível utilizar comandos de posição angular, somente foi possível determinar o sentido do movimento e sua velocidade. Esta limitação na comunicação entre câmera e software influencia diretamente na escolha do controlador e no seu desenvolvimento, que é abordado em detalhes na seção 3.4.

O padrão de endereçamento para os comandos encontrados na inspeção da rede foi:

$$\boldsymbol{URL} = \boldsymbol{http}://\boldsymbol{user}:\boldsymbol{pw}@\mathbf{192.168.15.173}:\mathbf{80}/\boldsymbol{web}/\boldsymbol{cgi}-\boldsymbol{bin}/\boldsymbol{hi3510}/\boldsymbol{ptzctrl.cgi}?$$

$$\boldsymbol{POST} = -\boldsymbol{step} = \mathbf{0}\& - \boldsymbol{act} = \boldsymbol{sentido}\& - \boldsymbol{speed} = \boldsymbol{velocidade}$$

Com a concatenação do $\boldsymbol{URL}$ e do $\boldsymbol{POST}$ tinhamos um comando reconhecido pela câmera PTZ, no qual $\boldsymbol{sentido}$ e $\boldsymbol{velocidade}$ eram as váriaveis do comando para alterar o sentido e a velocidade do movimento. A Tabela 3.1 contem todas as palavras-chave utilizadas como a direção e sentido do movimento da câmera.

Tabela 3.1 – Mapa de comandos para controlar movimentos de pan, tilt e zoom da câmera Speed Dome SL-130IPC851.

Movimento da câmera	**Comando de sentido do movimento**
Rotação horizontal para a esquerda	*left*
Rotação horizontal para a direita	*right*
Rotação vertical para cima	*up*
Rotação vertial para baixo	*down*
Parar movimentos	*stop*
Rotação diagonal esquerda superior	*leftup*
Rotação diagonal esquerda inferior	*leftdown*
Rotação diagonal direita superior	*rightup*
Rotação diagonal direita inferior	*rightdown*
Redução do campo ótico da câmera	*zoomin*
Ampliação do campo ótico da câmera	*zoomout*
Movimento em todos os eixos necessários para volta a câmera ao seu estado de posição inicial	*home*

Na velocidade de rotação definida no $\boldsymbol{POST}$ são permitidos números inteiros, maiores ou iguais a 10 e menores ou iguais a 63. Ao verificar empiricamente, descobriu-se que esses números de velocidade correspondiam a velocidade de rotação em graus por segundo.

Para controlar a câmera, primeiramente estabeleu-se conexão com endereço URL através da biblioteca PyCurl e foram enviados as mensagens no formato POST neste mesmo

endereço.

Além das limitações de comunicação, foi necessário definir as limitações fisicas da câmera para a correta implementação do projeto. As principais limitações fisicas são a máxima e mínima velocidade angular em *pan* e *tilt*, os angulos máximos de rotação e a capacidade máxima de *zoom*.

A máxima velocidade angular $\boldsymbol{w_{max}}$ impacta diretamente a capacidade da câmera *PTZ* perseguir o individuo alvo durante seu trajeto. Então, assumimos que os objetos a serem rastreados não estão em alta velocidade, para não ultrapassarem a velocidade angular máxima da câmera *PTZ*. Caso contrário, o objeto pode sair do campo visual da câmera, fazendo o controle servo visual perder o rastreamento. A velocidade máxima do objeto pode ser dada por:

$$\boldsymbol{v_{max} = L * w_{max}}$$

No qual $\boldsymbol{v_{max}}$ é a componente de velocidade tangencial à câmera e $\boldsymbol{L}$ é a distância do objeto à câmera. Sabendo que a velocidade angular máxima $\boldsymbol{w_{max}}$ permitida para a câmera PTZ é de 63º/s ou 1,1 rad/s, calcula-se que a partir de 5 metros de distância, a câmera poderia rastrear uma pessoa correndo a 5,5 m/s. A partir de 15 metros de distancia da câmera, a velocidade angular da câmera seria capaz de rastrear qualquer indivíduo sem precisar atuar em sua velocidade máxima de rotação.

No entanto, existem limitações de velocidade para o objeto além das limitações impostas pelos motores de *pan* e *tilt*, já que os algoritmos de detecção utilizados neste trabalho possuem melhor desempenho para detectar e rastrear objetos que não estão em alta velocidade.

Existe também a limitação de velocidade angular mínima em que a câmera *PTZ* pode operar. Sabendo que existe uma zona morta entre a velocidade zero e a velocidade mínima, é preciso adaptar o controlador da câmera para que os atuadores não fiquem ligando e desligando em pequenas frações de segundos. Esta restrição será tratada na seção 3.4, por poder causar vibrações e instabilidade no sistema de controle.

Além das restrições relacionadas a velocidade angular dos atuadores do sistema, a taxa de obtenção de quadros é um fator limitante da câmera e influencia diretamente o desempenho dos algoritmos de rastreamento. Segundo Kikuchi (2006), o controle servo visual se beneficia com maiores taxas de captura de quadros, ao custo de possuir mais ruídos por ter um tempo de exposição mais curto para a captura de cada imagem. O benefício das altas taxas de *frames* vem da ideia de que quanto menor o intervalo de tempo entre dois quadros consecutivos, mais semelhante o objeto alvo será entre um quadro e outro. Isso é especialmente verdade para o algoritmo de rastreamento utilizado neste trabalho, pois é assumido que não haverá mudanças bruscas no formato do objeto e na sua posição entre

frames consecutivos.

3.3 INTERPRETAÇÃO DAS IMAGENS

A interpretação dos dados visuais capturados pela câmera detém o maior custo computacional do controle servo visual, ou seja, é a principal preocupação de custo computacional e deve-se evitar abordagens lentas e que consumam muita memória.

O Software foi dividido em três blocos, sendo os blocos I e II responsáveis pelo processamento de imagens. Existem várias abordagens e algoritmos distintos de detecção e rastreamento de objetos. Esta seção se preocupa em explicar a escolha da abordagem e dos algoritmos utilizados para detecção e rastreamento do objeto, assim como a exposição das vantagens e desvantagens dos algoritmos utilizados em relação aos principais algoritmos da literatura.

Segundo Agren (2016), a escolha das características do objeto a ser rastreado define a robustez do algoritmo de rastreamento. Existem diversas características e para que haja uma detecção eficiente do objeto, é necessário que o objeto tenha características únicas. Caso a característica para detecção seja, por exemplo, a cor azul do objeto, e no plano de fundo existam outros objetos azuis, o algoritmo encontrará problemas em detectar corretamente o objeto alvo. Para contornar este tipo de problema, algumas vezes são inseridas no algorítmo de detecção outras características, como forma do objeto e textura, por exemplo.

Henz (2014) e Franco (2011) utilizam o rastreamento por cor, uma abordagem com baixo custo computacional em ambientes controlados. No entanto para o nosso objetivo que é a detecção e rastreamento de objetos que podem variar a forma e as cores entre eles, como os seres humanos por exemplo, este rastreador se torna problemático. Além de que, em geral, cores são muito sensíveis a mudanças na iluminação e este seria outro problema para este trabalho, já que o intuito é a aplicação do controle servo visual em câmeras de segurança que estão monitorando ambientes não estruturados.

As câmeras *PTZ* aplicadas em sistemas de segurança podem ser instaladas em locais abertos ou fechados, que possuem uma grande área de vigilância, como campos de futebol, shoppings, universidades e empresas privadas, por exemplo. Assim, faz-se necessário a preocupação com mudanças na iluminação, principalmente em locais abertos, para que a detecção e rastreamento do objeto no controle servo visual não seja deficiente.

O código do sistema de controle servo visual foi desenvolvido de modo modular, ou seja, cada bloco de código pode ser substituido por outro bloco com mesmas entradas e saídas, desde que possuam objetivos semelhantes na estrutura geral do código. A estrutura modular permitiu a substituição de blocos que não possuiam uma boa taxa de detecção ou falhava

muito no rastreamento, por exemplo.

3.3.1 Processamento de imagens utilizando detector e rastreador

O primeiro programa desenvolvido (Anexo II) teve como intuito detectar uma pessoa a partir de algoritmos de diferenciação temporal e segmentação por contorno utilizando sobel. Assim era possível detectar qualquer movimento e separar o objeto a partir de seus contornos. No entanto, tinhamos a restrição de não podermos movimentar a câmera em *pan*, *tilt* ou *zoom* com essa abordagem, pois quando a câmera se movimentava, o algoritmo de diferenciação temporal falhava, devido ao plano de fundo mudar a cada frame. Devido a esta grande restrição, iniciou-se a construção de um novo software utilizando uma abordagem clássica da detecção de objetos.

Na nova abordagem, escolheu-se utilizar a técnica de Histograma de Gradientes Orientados (HOG) (DALLAL e TRIGGS, 2005). Nesta clássica técnica de detecção de objetos, os Vetores de Gradiente nos indicam, em cada ponto, onde há os maiores contrastes relativos a bordas. Assim tem-se uma menor influencia da variação de luminosidade no ambiente, já que as caracteristicas de borda são menos sensíveis a luminosidade (AGREN, 2016).

Como vimos na seção 2.3.2, aliando HOG a um algoritmo de aprendizagem de maquina, pode-se criar um detector de objetos a partir da aprendizagem supervisionada. Neste trabalho, utilizou-se este detector através da biblioteca OpenCV, no qual não foi necessário treinar o sistema, pois o OpenCV já possui um sistema padrão treinado para detectar ou classificar pedestres.

Ao utilizar o HOG com SVM disponibilizado como uma função pronta para uso pela biblioteca OpenCV, iniciou-se testes em imagens de pessoas capturadas em bases de dados na internet. Após os testes em imagens, desenvolveu-se a primeira abordagem para o controle servo visual da câmera PTZ, que é esquematizado na Figura 3.3.

Observando a Figura 3.3, vê-se que logo após a captura da imagem, faz-se um pré-processamento da mesma, antes de utiliza-la na detecção. Este pré-processamento consiste em transformar a imagem RGB capturada pelo câmera em uma imagem com apenas um canal. Com a imagem em tons de cinza, aplica-se um filtro Gaussiano para suavizar a imagem retirando ruidos de alta frequencia. Veremos que na segunda abordagem, ilustrada pela Figura 3.4, a etapa de pré-processamento é melhorada para que haja uma diminuição na detecção de falsos positivos. Esta etapa de pré-processamento é imprescindível para uma boa taxa de acertos do detector, já que estamos trabalhando com uma câmera *PTZ*

capturando imagens em locais sem estrutura propícia para uma detecção ótima.

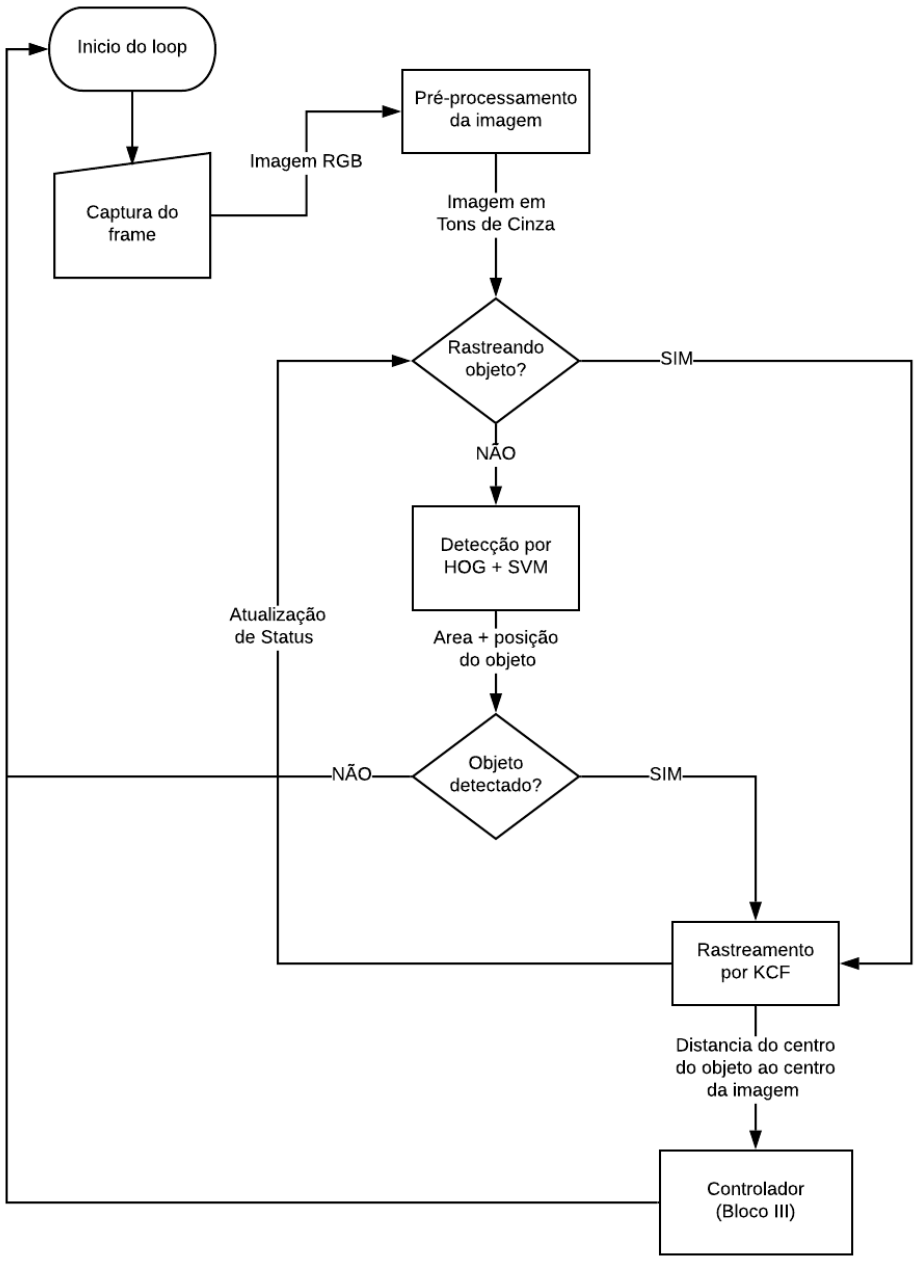


Figura 3.3: Fluxograma do software de detecção e rastreamento.

Após o pré-processamento, verifica-se se já estavamos rastreando algum individuo e em caso negativo, passamos a imagem pelo detector HOG com SVM (DALLAL e TRIGGS, 2005). Este bloco de detecção produz o histograma de gradientes orientados da imagem e classifica a imagem em pedestre ou não-pedestre, de acordo com a aprendizagem de maquina decorrida pelo SVM implementada no OpenCV. Como resposta desta função, tem-

se todas as posições e áreas (em formatos retangulares) dos individuos detectados.

Caso nenhum individuo seja detectado, voltamos ao inicio do loop e trabalhamos com o próximo frame. Caso detecte-se um invividuo, iremos iniciar um rastreamento deste por meio do *Minimum Output Sum of Squared Error* ou MOSSE.

A utilização do bloco de rastreamento (bloco II) foi adicionada para que não fosse necessário fazer uma detecção a cada frame após encontrar o alvo, ou seja, o rastreamento foi adicionada para melhorar a performance do software e para refinar a precisão da posição do alvo.

Segundo Bolme (2010), os filtros correlacionais MOSSE possuem alta performance se comparado a outros rastreadores como MIL e ASEF, e é menos complexo e mais fácil de ser utilizado que rastreadores baseados em Redes Neurais Convolucionais como o GOTURN.

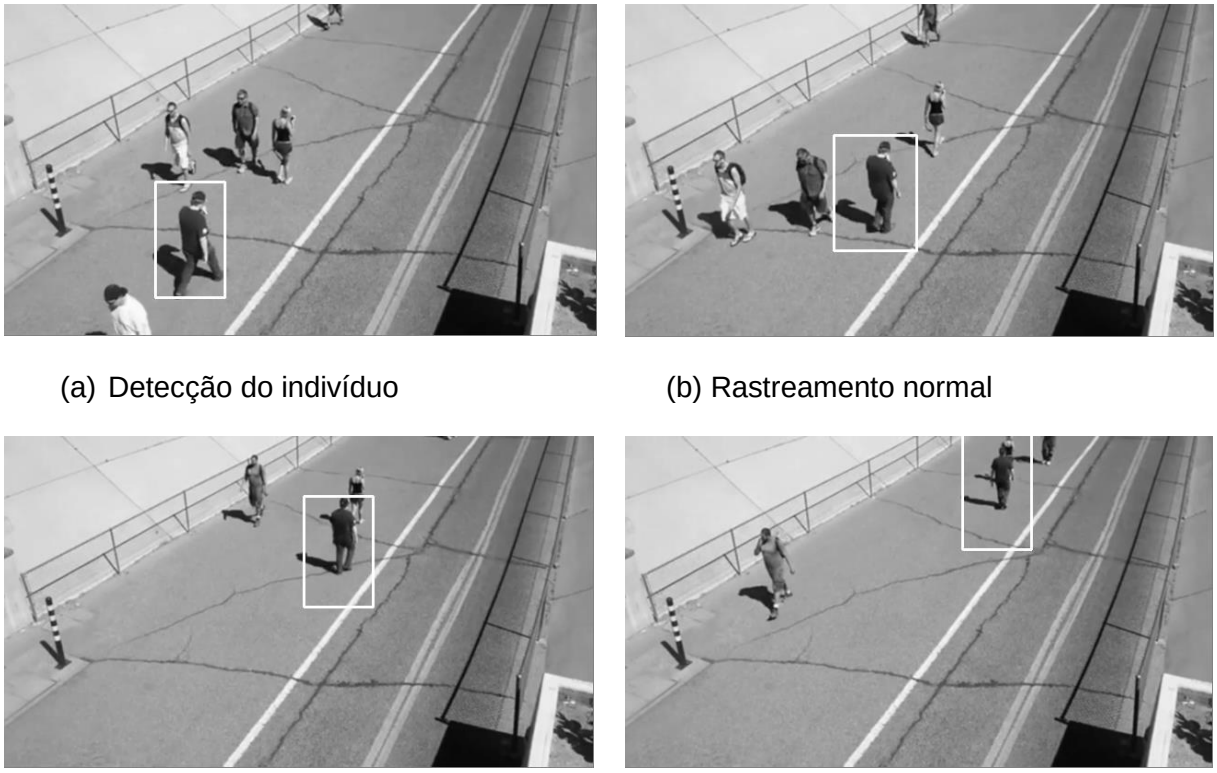
Sabendo que os filtros correlacionais possuem um bom custo beneficio de rastreamento e desempenho, testamos o bloco II com dois rastreadores que utilizam filtros correlacionais disponibilizados pelo OpenCV: MOSSE e KCF.

Após fazer testes de robustez do rastreamento e do desempenho de cada um deles, escolheu-se o algoritmo com maior precisão, o KCF.

Ao utilizar o filtro KCF como rastreador, o sistema atualizava a posição do alvo a cada novo frame. Essa informação de posição era referenciada de acordo com o centro da imagem e essa informação de erro entre posição do alvo e o centro da câmera é passada para o controlador.

Esta abordagem teve exito em detectar e rastrear o individuo, no entanto o rastreador não atualizava a área do objeto a cada frame. Ou seja, após o HOG com SVM detectar o individuo e passar a posição e área inicial para o rastrador KCF, a atualização de posição era efetuada com sucesso, mas a area do retângulo contendo o individuo era constante, independente se o individuo estava se afastando (diminuindo seu tamanho na imagem) ou se

aproximando (aumentando seu tamanho na imagem) da câmera.



(a) Detecção do indivíduo (b) Rastreamento normal

(c) Alvo se afastando da câmera (d) Rastreamento impreciso

Figura 3.4: Teste realizado para verificar a imprecisão do rastreador quando o objeto alvo se afasta da câmera (adaptado de BriefCam, 2009).

Na Figura 3.4(a) o HOG detecta a individuo na imagem e passa essa informação para o rastreador. Na Figura 3.4(b) pode-se ver que o rastreador persegue o individuo corretamente enquanto o individuo não muda sua escala. Já nas Figuras 3.4(c) e 3.4(d) observa-se que o rastreador começa a ter imprecisão e caso isso se extenda por um longo período o rastreador não é capaz de perseguir o objeto e atualizar sua área delimitante.

A informação da area do retangulo contendo o objeto é de extrema importancia para o controle do zoom, pois não é possível focar no indivíduo sem saber o seu tamanho em relação ao tamanho da imagem. Visando uma solução para tal problema, optou-se pelo desenvolvimento da segunda abordagem, no qual utilizaremos apenas o detector.

3.3.2 Processamento de imagens utilizando apenas detector

Visando solucionar o problema que ocorreria ao ampliar ou reduzir a imagem para focar no indivíduo, optou-se por retirar o bloco II. A Figura 3.5 mostra o esquemático do novo

algoritmo de processamento de imagens.

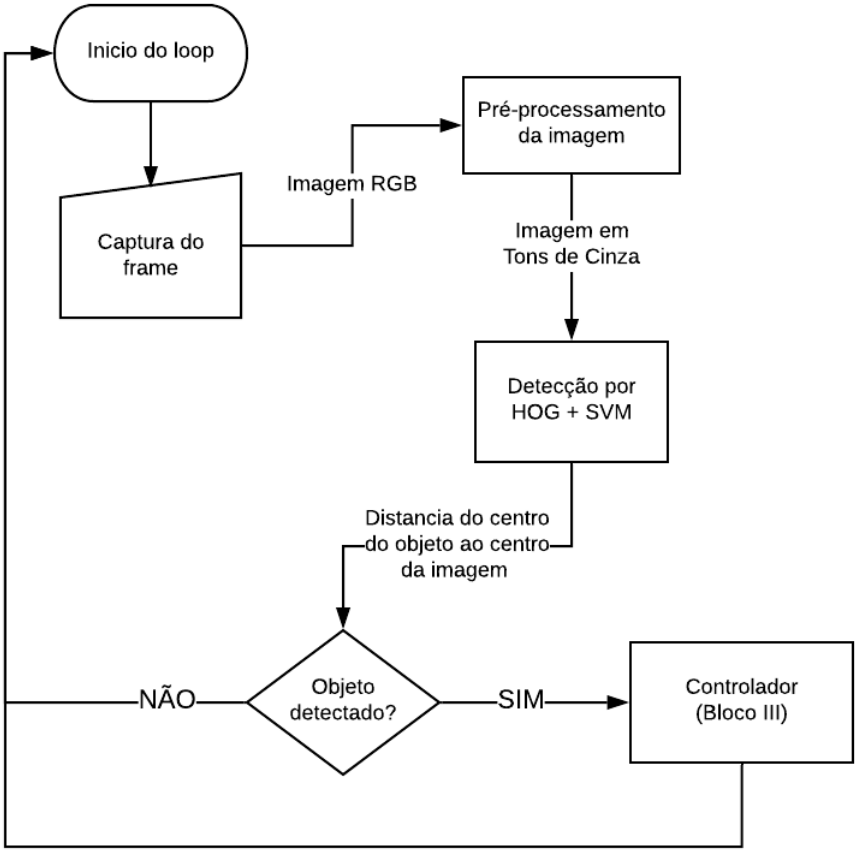


Figura 3.5: Fluxograma do software com detecção em cada frame.

A primeira mudança nessa abordagem foi a adição de uma etapa no pré-processamento, no qual é ajustado o gama da imagem e posteriormente utiliza-se o filtro Gaussiano. O Ajuste de gamma foi importante para obtermos imagens com iluminação semelhantes, mesmo que tenham sido capturadas em momentos distintos com mais ou menos luminosidade no local. Esse ajuste foi feito manualmente modificando o valor do gama sempre que fosse necessário para obter uma resultado satisfatório.

Embora a mudança no pré-processamento tenha impacto na detecção do individuo, a principal diferença entre o algoritmo da Figura 3.4 e da Figura 3.5 é o rastreador. Nesta abordagem sem rastreador, o bloco de detecção é executado em todos os frames, diminuindo o desempenho do sistema, já que os detectores de forma geral possuem um custo computacional maior que os rastreadores.

A informação de distancia e tamanho da caixa do individuo alvo é calculada no bloco de detecção e serve de referencia para o bloco do controlador. Devido a detecção ocorrer em todos os frames, o tamanho do individuo é atualizado em cada frame caso o individuo se aproxime ou se afaste da câmera. A Figura 3.6 mostra dois frames de uma pessoa se

aproximando da câmera e o detector atualizando corretamente o tamanho da caixa de detecção.

Figura 3.6: Detecção de individuo se aproximando e sua área sendo atualizada a cada frame (adaptado de BriefCam, 2009).

Na Figura 3.6 podemos observar dois frames de um video, no qual o tamanho da caixa de contenção ou área do individuo é atualizada. Assim podemos utilizar a relação entre tamanho da caixa do individuo detectado com o tamanho da imagem para obtermos um variavel de controle do zoom.

Sabendo que na realidade de sistemas de vigilância o objeto a ser rastreado não estará perfeitamente no centro da imagem, utilizou-se a união de algoritmos de detecção por movimento com o rastreador por contorno (Figura 3.1, bloco I).

O bloco I é responsável por detectar o objeto quando o mesmo entra no campo de visão da câmera e encontrar o contorno equivalente ao objeto em movimento. Assim, é possível calcular a área e centroide da referência inicial e passar esta referência para o bloco II, que utiliza somente algoritmo de contorno para efetivar o rastreamento.

A mesclagem do algoritmo de detecção de movimento com a detecção de silhueta no bloco I, causa um maior tempo de processamento em relação ao bloco II. Por isso, caso o objeto já tenha sido encontrado na imagem, os dados da imagem em tons de cinza vão diretamente para o bloco II, não sendo necessário qualquer processamento no bloco I. Ou seja, a abordagem utilizada detecta o objeto apenas a primeira vez em que ele se movimenta, depois disso o bloco II faz todo o processamento de imagens até que o rastreador perca a posição do objeto ou que o objeto saia do campo de visão coberto pelo câmera *PTZ*.

Para a simplificação do problema de detecção inicial (bloco I), assumimos que o objeto alvo estará completamente dentro do enquadro da imagem antes de começar a se movimentar. Caso contrário, seria necessário o tratamento de oclusão parcial do objeto.

Assumiu-se também que o objeto não se movimentará em alta velocidade, assim podemos concluir que não haverá grandes mudanças na silhueta do objeto e na posição do mesmo, entre dois quadros consecutivos. Esta imposição é de extrema importância para que o algoritmo consiga rastrear o objeto, pois na fase de casamento entre área e centroide de

referência com a área e centroide do *frame* atual, assumimos que as áreas e posições dos centroides serão bastante semelhantes entre si, em um curto período de tempo. Reforçando novamente a necessidade de uma taxa de obtenção de *frames* adequada.

Neste trabalho, tanto a câmera IP quanto a câmera Speed Dome capturam 30 quadros por segundo, possuindo assim um tempo entre quadros de aproximadamente 34 ms. Então podemos assumir que as diferenças entre área e centroide de dois quadros consecutivos são de fato pequenas, dado que o período entre captura é suficientemente pequeno.

Após a ultima tarefa de cada quadro, o bloco II calcula a informação da distância entre o centroide do objeto e o centro da imagem. Este vetor distância é transmitido para o bloco III, que utiliza esta informação para convergir o centro da imagem com o centro do objeto. Está distância é o erro que o controlador utilizará para calcular a nova orientação da câmera.

3.4 CONTROLADOR

Existem muitas técnicas de controle utilizadas para desenvolver um controle servo visual. Pereira (2017) utiliza uma técnica de controle PID com algumas adaptações para controlar um robô móvel terrestre e orienta-lo em direção ao objeto alvo. Franco (2011) aborda o controle *IBVS* em um robô Pionner P3-AT com base em três características de referência. Kikuchi (2007) utiliza três diferentes abordagens, um controlador Linear Quadrático, Filtro de Kalman e controle proporcional integral. Por fim, Kikuchi (2007) utilizou o controlador PI em conjunto com filtro de kalman e obteve resultados satisfatórios para o controle servo visual de uma câmera *pan-tilt*.

Para definirmos qual é a melhor abordagem no desenvolvimento do controlador, é necessário primeiramente entender a planta do sistema. A Figura 3.7 ilustra a relação entre a distância angular e os pixeis da imagem capturada pela câmera *PTZ*.

Para relacionar a distancia angular entre pontos na imagem e os píxeis na imagem, foram feitas três medições para estimar o angulo de abertura das lentes da câmera sem o zoom. Utilizou-se uma fita métrica para fazer as medidas à 120 cm de distância da câmera e obteve-se os valores de distância $\boldsymbol{L}$ entre o extremo esquerdo da imagem e o extremo direito, em linha horizontal.

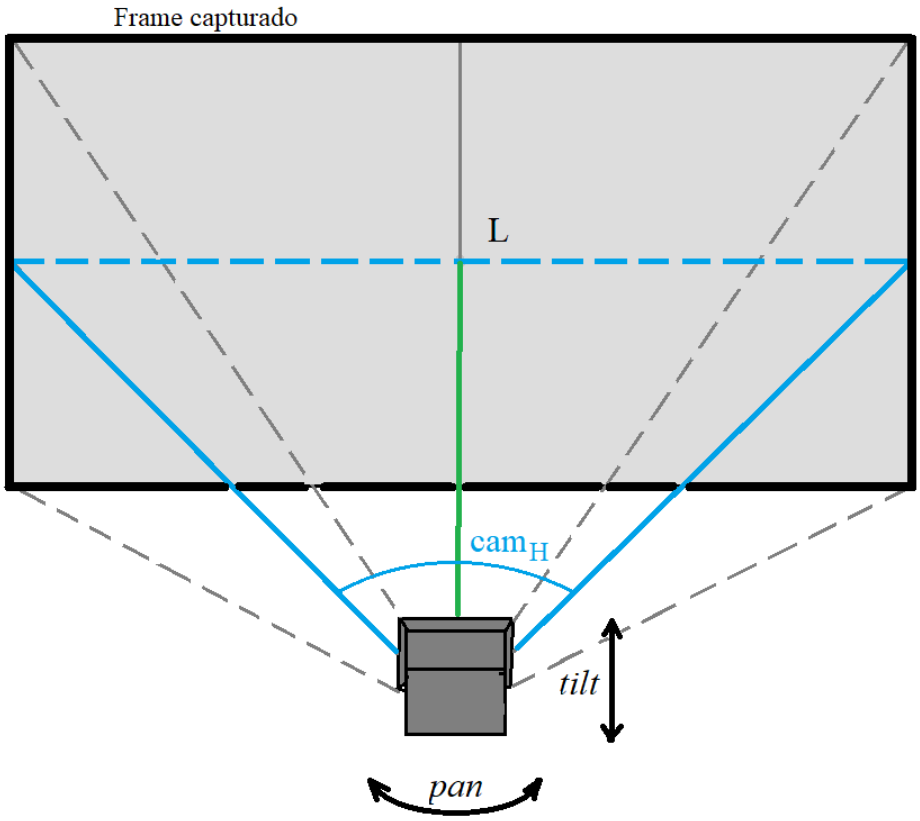


Figura 3.7: Esquemático da relação entre abertura das lentes da câmera e tamanho da imagem.

Então estimou-se que o campo de visão horizontal $\boldsymbol{cam_H}$ da câmera Speed Dome a partir da equação:

$$\boldsymbol{cam_H = 2 * tan^{-1}\left(\frac{Lm}{2 * 120}\right)}$$

onde $\boldsymbol{Lm}$ é a média das três medidas de distancia $\boldsymbol{L}$ entre os extremos horizontais.

Sabendo que tinhamos a informação de posição do objeto em um sistema proporcional, decidiu-se utilizar a abordagem clássica e desenvolver um controlador PI. O controlador teria como entrada a distancia, em pixeis, do centro do alvo detectado ao centro da imagem e como saída, deveria enviar os comandos para os atuadores da câmera através de protocolo HTTP.

A vantagem de um controlador PI é que ele possui a propriedade de promover o modelo para tipo 1 (KIKUCHI, 2007), assim os ruidos e perturbações constantes são suprimidas devido ao fator integrador do controlador. Ou seja, o controlador PI permite a estabilidade em torno de seu valor de referencia, com erro nulo em regime estacionário.

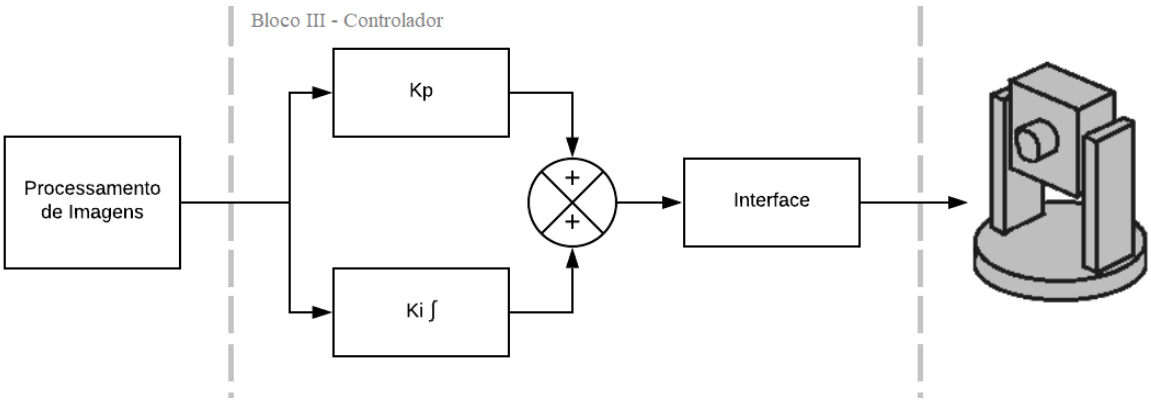


Figura 3.8: Esquemático do bloco do controlador.

A partir da Figura 3.8, observa-se que a informação de distância do centro do objeto detectado ao centro da imagem é a variável de entrada para o bloco do controlador, e após a somatória do controlador entre a fator proporcional e integral, tem-se o código de interface para transcrever as informações do controlador para os comandos HTTP que a câmera *PTZ* reconhece.

Devido a câmera *PTZ* possuir os três grais de liberdade *pan*, *tilt* e *zoom*, optou-se por termos um controlador PI para cada grau de liberdade, de forma indenpendente.

Com o vetor de distancia do objeto ao centro da imagem foi possível extrair o componente horizontal e vertical do vetor, que corresponde a amplitude angular do movimento em *pan* e *tilt* para alinhar o centro da imagem com o centro do alvo detectado. Devido a semelhança entre o movimento *pan* e *tilt,* as constantes proporcionais e integrais foram as mesmas para ambos.

A constante proporcional K_p para o movimento em *pan* foi calculada a partir da razão entre a quantidade de pixels horizontais da imagem capturada img_H e o campo de visão horizontal da câmera em graus cam_H, como pode-se ver na equação:

$$K_p = \frac{cam_H}{img_H}$$

Para estimar a constante de integração K_i para o controlador em *pan* e *tilt*, utilizou-se inicialmente o valor de 10% do K_p e a partir desse ponto, aumentou-se o valor de K_i aos poucos até acharmos experimentalmente um valor para a constante integrativa que alcançava um erro nulo em regime estacionário com uma resposta rápida.

Após definir o melhor valor de K_i empiricamente, o controlador estava causando pequenas vibrações em regime estácionario devido a zona morta de velocidade do atuador comentada na seção 3.2. A zona morta de velocidade entre 0º/s e 15º/s aliada a uma taxa de

frames de aproximadamente 15 FPS quando utilizando apenas o detector, nos dava uma resolução de movimento de aproximadamente um grau por frame. Ou seja, a resolução do movimento angular em *pan* e *tilt* era maior que o valor de K_p.

Para solucionar o problema de vibrações em regime estácionário, criou-se um limitador na bloco de interface do controlador. Este limitador fazia com distancias entre o centro de imagem e do objeto detectado fossem ignoradas se fossem menor do que a resolução do movimento em *pan* e *tilt*. Na prática, a resolução de movimento de 1º equivale a 7,36 pixels e no software o valor do limitador foi arredendado para 10 pixels. Ou seja, erros menores que 10 pixels entre o centro do objeto e o centro da câmera eram ignorados e nenhum movimento era efetuado pelos atuadores de *pan* e *tilt* até que este erro fosse maior que 10 pixels.

Após o correto funcionamento do controlador dos movimentos horizontais e verticais, fez-se a implementação do controle de zoom. Diferente de *pan* e *tilt*, o *zoom* só precisa das informações de tamanho do objeto para relacionar com o tamanho da imagem e calcular o zoom que deve ser aplicado para focar no indivíduo alvo. A Figura 3.9 ilustra as relações entre o tamanho da imagem e o tamanho da caixa de detecção do objeto.

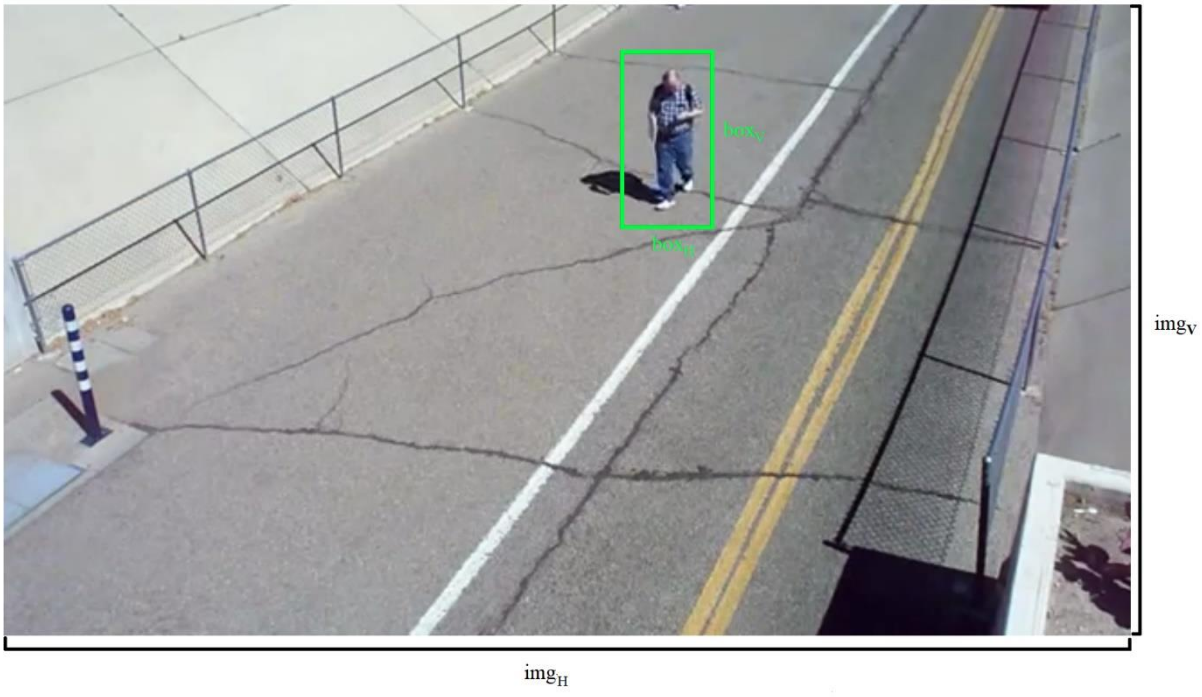


Figura 3.9: Relação entre tamanho da imagem e tamanho do objeto para estimativa do zoom (adaptado de BriefCam, 2009).

Para encontrar a relação de zoom necessária para focar no indíviduo, temos:

$$zoom = mín\left(\frac{img_V}{box_V} ; \frac{img_H}{box_H}\right)$$

No qual img_V e img_H são o tamanho da imagem em pixels na vertical e na horizontal, respectivamente. E o tamanho do objeto detectado é representado por box_V para vertical e box_H para horizontal.

Esta etapa de calculo foi adicionada ao controlador de zoom, e o valor do $zoom$ foi utilizado como referência para o controlador, assim a constante proporcional do zoom K_{pz} pode ser unitária.

O mesmo método utilizado para encontrar empiricamente o valor da constante integral para o movimento de *pan* e *tilt* foi utilizado para encontra a constante integral do zoom K_{iz}, no entanto iniciou-se com K_{iz} sendo 5% do valor de , K_{pz}. O melhor valor foi estimado em 10% do K_{pz}, ou seja, $K_{iz} = 0{,}10$.

Comparando as constantes integrais de *pan* e *tilt* com a do controlador de zoom, pode-se ver que proporcionamente o K_{iz} é menor que o K_i. Isso se deve ao fato de um zoom afetar a imagem de uma forma mais impactante ao poder desfocar completamente a imagem caso não seja suave. Caso o foco automático da câmera seja desregulado enquanto o zoom esta sendo efetuado, o sistema poderia se tornar instável durante este período.

Ainda com o intuito de evitar que o foco automático da câmera *PTZ* encontrasse problemas ao ativar o zoom, colocou-se um limitador para o fator de zoom, semelhante aos limitadores de 10 pixels para *pan* e *tilt* implementados no bloco de interface. Assim o controlador do zoom só poderia ser executado caso os controladores de *pan* e *tilt* não fossem executados naquele instante.

Esta abordagem para o controlador de zoom evita que a câmera atue nos três graus de liberdade ao mesmo tempo. Esta restrição foi necessária, já que ao não foram encontrados comandos para movimentar a câmera em três graus de liberdade simultaneamente.

A principal responsábilidade do bloco de interface mostrado na Figura 3.8 é de receber os valores de saídas dos controladores PI para *pan*, *tilt* e *zoom* e transforma-los em comandos HTTP com o sentido do movimento e a velocidade, estabelecendo comunicação através da biblioteca PyCurl.

CAPÍTULO 4 – RESULTADOS EXPERIMENTAIS

Este capítulo agrega os principais resultados obtidos para a detecção, rastreamento e controle. Para a detecção e o rastreamento, foram feitos testes isolados em para avaliar a qualidade e precisão da detecção e do rastreamento. Posteriormente, obteve-se resultados empiricos para as duas abordagens do controle servo visual, uma utilizando o detector HOG com SVM (DALLAL e TRIGGS, 2005) trabalhando em conjunto com o rastramento por filtros correlacionais KCF (HENRIQUES, 2015), e a outra utilizando apenas o detector HOG com SVM sendo executado a cada frame.

4.1 DETECTOR

Inicialmente, realizou-se testes de detecção da técnica HOG e SVM implementadas por padrão na biblioteca OpenCV. Estes testes foram divididos em duas amostras, cada uma com 50 imagens de pedestres. A primeira amostra contém imagens em baixa resolução (abaixo de 0,3 megapixels) e a segunda amostra possui imagens de resolução próxima a da câmera PTZ (aproximadamente 0,9 megapixels).

Para obtenção de dados relativos ao teste, executamos o detector em cada imagem e contamos quantas pessoas estavam na imagem (Referência), quantas delas foram detectadas corretamente (Acertos), quantas não foram detectadas (Falsos Negativos) e quantas falsas detecções ocorreram (Falsos Positivos). A Tabela 4.1 e 4.2 sintetiza os dados relativos a primeira amostra.

Tabela 4.1: Taxas de detecção em cada imagem de baixa resolução.

Total de Imagens	**Imagens com ao menos um acerto**	**Imagens com ao menos um falso negativo**	**Imagens com ao menos um falso positivo**
50	21	30	7
100%	42,0%	60,0%	14,0%

Tabela 4.2: Taxas de detecção dos pedestres, em imagens de baixa resolução.

Total de Pessoas	**Quantidade de Acertos**	**Quantidade de Falso Negativo**	**Quantidade de Falso Positivo**
83	30	53	10
100%	36,1%	63,9%	12,0%

Observando as Tabela 4.1 e 4.2, vemos que a taxa de acertos do detector é pequena

para imagens de baixa resolução. Isso se deve a dificuldade do descritor HOG em extrair as características do objeto em uma resolução muito reduzida. Observa-se que de 83 pedestres distribuidos nas 50 imagens, apenas 30 pedestres foram detectados corretamente, equivalente a 36,1% do total de pedestres.

Devido ao valor reduzido na taxa de acertos, executou-se os testes com imagens de maior resolução para verificar se as taxas de detecção melhorariam. As Tabelas 4.3 e 4.4 resumem os dados da segunda amostra.

Tabela 4.3: Taxas de detecção em cada imagem de alta resolução.

Total de Imagens	**Imagens com ao menos um acerto**	**Imagens com ao menos um falso negativo**	**Imagens com ao menos um falso positivo**
50	45	18	25
100%	90,0%	36,0%	50,0%

Tabela 4.4: Taxas de detecção dos pedestres, em imagens de alta resolução.

Total de Pessoas	**Quantidade de Acertos**	**Quantidade de Falso Negativo**	**Quantidade de Falso Positivo**
106	82	23	36
100%	77,4%	21,7%	34,0%

Através da Tabela 4.3, pode-se observar que 90% das imagens obtiveram pelo menos um acerto na detecção. Se compararmos aos testes efetuados em imagens de baixa resolução, conseguiu-se aumentar drasticamente a quantidade de acertos, já que passamos de uma taxa de 36,1% para 77,4% de acertos. No entanto, o aumento na taxa de acertos nas imagens de alta resolução acarretou em um aumento na quantidade de falsos positivos.

Pode-se inferir da Tabela 4.3 que ao utilizar imagens de alta resolução, aproximadamente metade dos frames podem ter uma detecção falso positivo. Esta alta taxa de falsos positivos podem fazer com que o controle servo visual detecte e rastreie objetos irrelevantes. Assim, faz-se necessário estabelecer uma resolução adequada para utilizar no controle servo visual, equilibrando em um valor no qual a taxa de acertos não é muito baixa, mas que também não existam muitos falsos positivos.

Existem outros fatores, além da resolução da imagem, que influenciam no processo de detecção, como luminosidade, amostras utilizadas para treinar o SVM, a pose do ser humano na imagem e até mesmo os parâmetros utilizados para o descritor HOG, por exemplo.

4.2 RASTREADOR

O motivo de testarmos o rastreamento separado do resto do programa é para obter informações sobre a qualidade de cada um dos dois rastreadores testados e ao fim escolher o melhor método de rastreamento para o projeto do controle servo visual.

Os testes foram realizados em 10 vídeos de curta duração e cada video possui pelo menos um ser humano a ser rastreado do primeiro frame até o ultimo, sem oclusão total do invivíduo a ser rastreado. Dos dez vídeos de teste, quatro capturavam imagens de pessoas se aproximando ou se afastando da câmera e foi categorizado como video “A”, dois vídeos capturavam pessoas que efetuavam movimentos bruscos (“B”) e os ultimos quatro possuiam pessoas sem movimentos bruscos e sem se aproximar ou se afastar da câmera (“C”). A Tabela 4.5 apresenta os dados obtidos no rastreamento por KCF e por MOSSE.

Tabela 4.5: Dados extraídos de testes no rastreamento por KCF e MOSSE (** frames com alta imprecisão no rastreamento).

Video	**Total de frames**	**Frames rastreados por KCF**	**Frames rastreados por MOSSE**
A1	216	210	215
A2	272	88	1
A3	108	22	108
A4	170	102	119
B1	225	139	224
B2	154	60	142
C1	700	541	492 **
C2	120	7	106 **
C3	215	16	215
C4	470	127 **	141 **

O rastreamento por KCF obteve sucesso em 49,5% do total de frames, enquanto o rastreamento por MOSSE obteve 63,4% de acertos. No entanto, pode-se observar na Tabela 4.5 que existem três vídeos da categoria “C” que tiveram uma grande imprecisão da informação de posição do indivíduo no rastreamento por MOSSE. Além disso, foi observado durante os testes que a principal causa de falha no rastreamento por KCF foi em situações

onde havia oclusão parcial do indivíduo. Já a técnica MOSSE possui alta tolerancia a oclusão parcial, mas com um precisão menor e com maior falso positivo, como por exemplo, rastrear partes do plano de fundo (Figura 4.1).

Figura 4.1: Rastreamento por filtro MOSSE com falso positivo (adaptado de BriefCam, 2009).

Embora o filtro MOSSE tenha rastreado os indivíduos por mais tempo em cada uma das categorias da amostra, percebeu-se que o algoritmo KCF possui uma melhor precisão da posição do indíviduo e contem menores taxas de falsos positivos. Assim optou-se por utilizar o rastreamento por KCF em uma das abordagens do controle servo visual desenvolvido neste trabalho.

4.3 CONTROLE SERVO VISUAL

Primeiramente, fez-se medições de alguns parâmetros para calcular a constante $\boldsymbol{K_p}$ do controlador. Colocou-se a câmera PTZ em um local fixo e uma fita métrica a 120 cm de distância da câmera, de forma tângencial ao raio de visão da câmera. Com a imagem obtida pela câmera, pode-se obter a distancia horizontal em centímetros que a câmera consiguia capturar do extremo esquerdo da imagem ao extremo direito. Assim, pode-se relacionar a quantidade de pixels horizontais na imagem ($\boldsymbol{img_H}$) com a abertura angular correspondente ao campo de visão da câmera PTZ ($\boldsymbol{cam_H}$). As três medidas $\boldsymbol{L}$ foram tomadas:

$$\boldsymbol{L_1 = 225\ cm}$$

$$\boldsymbol{L_2 = 228\ cm}$$

$$\boldsymbol{L_3 = 228\ cm}$$

Sendo $\boldsymbol{L_m}$ a média das medidas $\boldsymbol{L}$ e $\boldsymbol{cam_H}$ o campo de visão da câmera em graus, temos:

$$\boldsymbol{L_m = 227\ cm}$$

$$\boldsymbol{cam_H = 2 * tan^{-1}\left(\frac{L_m}{2*120}\right) = 86,8}$$

Sabendo que a abertura horizontal e vertical das lentes da câmera são proporcionais ao tamanho da imagem, pôde-se admitir a mesma constante K_p para os controladores em *pan* e *tilt*. E sabendo que a imagem capturada possui 640 pixels de largura e seu campo de visão horizontal medido foi de aproximadamente 86,8 graus, temos:

$$K_p = \frac{86,8}{640} = 0,13563$$

Após a obtenção da constante proporcional $\boldsymbol{K_p}$, obteve-se a constante integral $\boldsymbol{K_i}$ experimentalmente como sendo 15% do valor da constante proporcional.

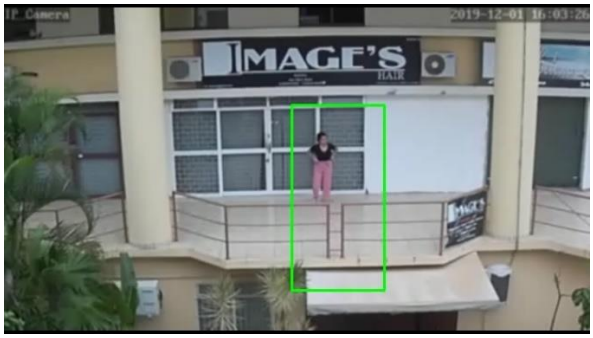
$$\boldsymbol{K_i = 0,02034}$$

Então, iniciou-se os testes com a câmera SL-130IPC851. Os testes foram realizados para as duas abordagens de detecção e rastreamento no controle servo visual, discutidas na seção 3.3.

A câmera PTZ foi instalada em um local alto, para capturar imagens sem oclusão do indivíduo e os testes aconteceram em um local aberto, com iluminação constante.

Os primeiros testes foram realizados com a abordagem utilizando detecção HOG e rastreamento por KCF trabalhando em conjunto para detectar e rastrear o indivíduo enquanto ele estivesse no campo de visão da câmera. A Figura 4.2 mostra a detecção ocorrendo no primeiro quadro e nos 3 quadros posteriores o rastreador atualizando a posição do indivíduo e aplicando zoom optico para capturar detalhes do individuo.

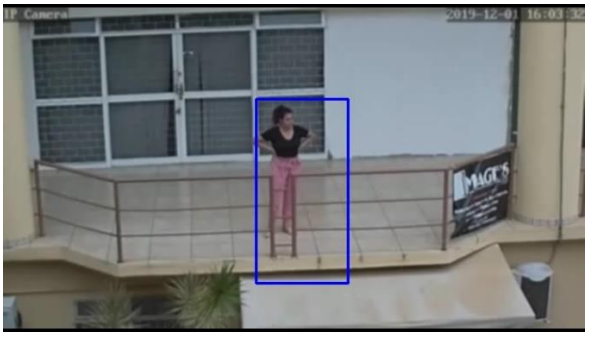
Observa-se que o resultado apresentado na Figura 4.2(a) possui uma grande imprecisão no tamanho da caixa de detecção, mas mesmo assim o rastreamento consegue seguir o alvo corretamente e dar zoom o suficiente para extrair mais informações do indivíduo, intuito principal deste trabalho.



(a) Indivíduo detectado

(b) Posição passada para o rastreador

(c) Zoom no indivíduo



(d) Foco no indíviduo

Figura 4.2: Sequencia de frames do video de teste utilizando detector e rastreador para encontrar e focar no indivíduo.

No entanto, em todos os casos foi necessário ajustar o zoom inicial da câmera para reduzir o campo visual da câmera, tentando retirar as árvores e plantas do cenário. Caso não fizessemos essa adaptação no cenário, o detector sempre tinha problemas com detecções de falsos positivos, como apresentado na Figura 4.3.

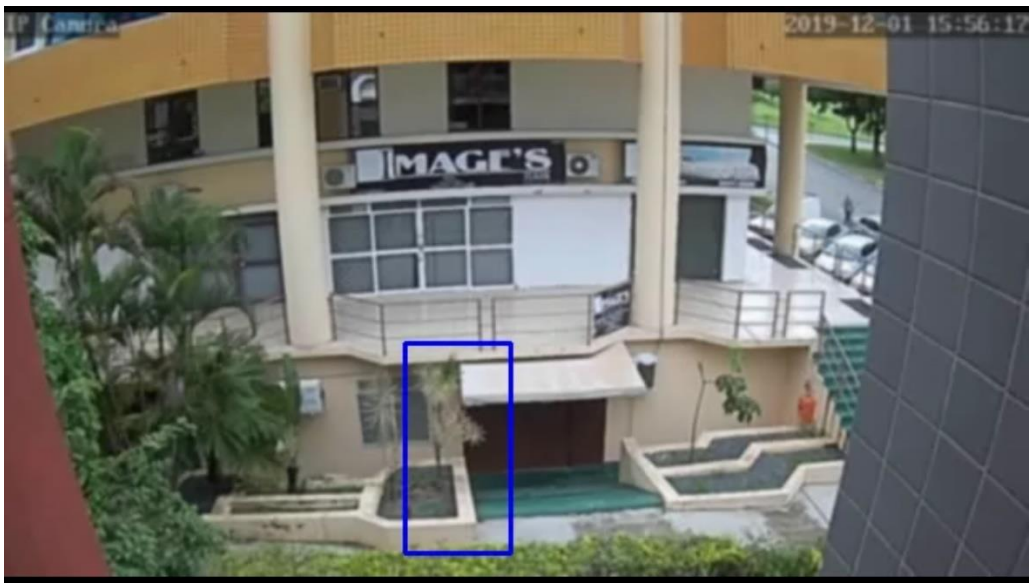
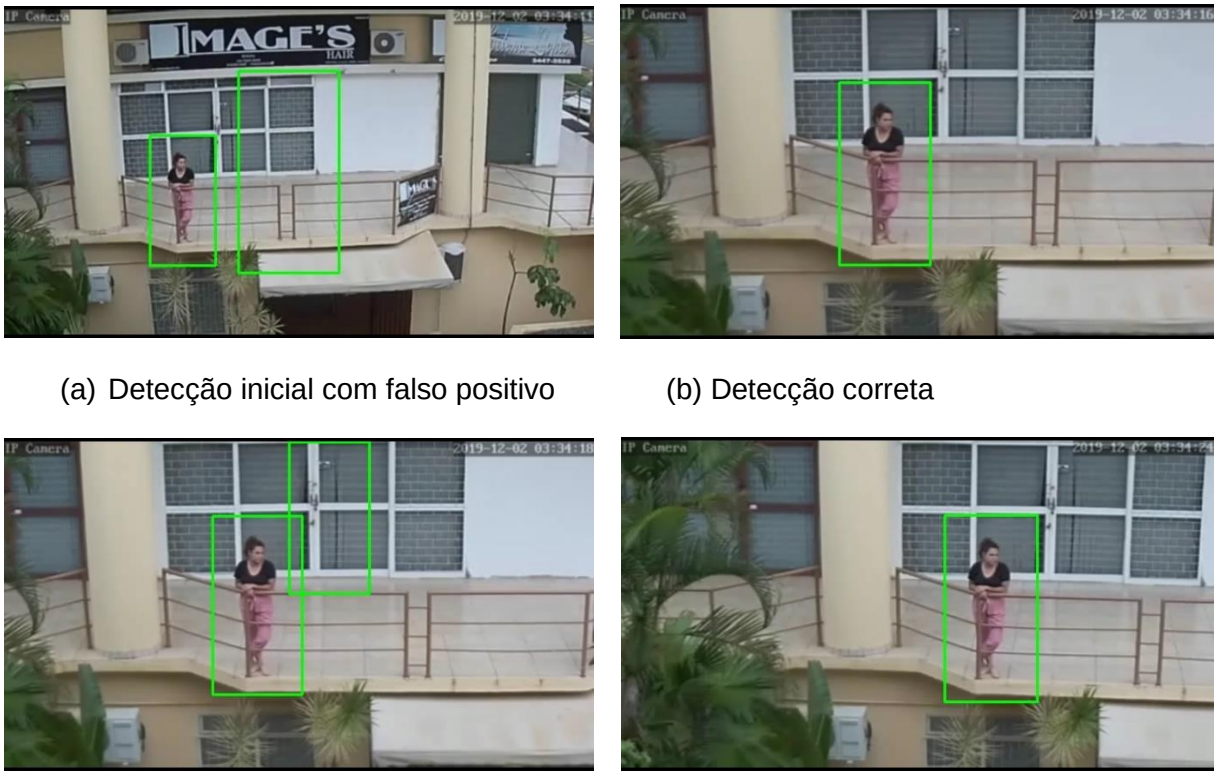


Figura 4.3: Planta sendo detectada como um ser humano, levando o sistema a falhar.

Para comparar as duas metodologias propostas, fez-se os testes da abordagem utilizando apenas detecção a cada frame no mesmo cenário e condições de iluminação. No entanto o controle servo visual mostrou característica oscilatórias em regime permanente para alguns casos nesta abordagem, devido a detecção de multiplos alvos em cada frame, no qual apenas um alvo era o correto. A Figura 4.4 ilustra o controlador tentando focar no individuo, mas oscilando devido as falsas detecções.



(a) Detecção inicial com falso positivo (b) Detecção correta

(c) Detecção com falso positivo (d) Detecção correta

Figura 4.4: Sequencia de frames do vídeo de teste utilizando apenas detector para encontrar e focar no individuo.

Na Figura 4.4(c) e 4.4(d) pode-se constatar que em um frame o indivíduo está na metade esquerda da imagem, enquanto no outro frame o indivíduo está a direita da imagem. Este efeito oscilatório ocorreu por mais de 10 segundos durante a gravação do vídeo e só parou após o individuo se movimentar pela sacada e a detecção diminuir a quantidade de falsos positivos na imagem.

Nos 8 vídeos gravados nesta abordagem, apenas dois tiveram características oscilatórias que fizeram o sistema falhar. Embora todas as gravações tenham detectado falsos positivos em determinados frames, geralmente era uma detecção pontual que não persistia no mesmo local ao decorrer do vídeo. Sendo assim, o algoritmo ainda conseguia focar com sucesso no individuo.

Para verificar as características oscilatórias e a estabilidade do controlador, fez-se testes com um alvo fixo para observarmos a resposta transitória e a resposta em regime permanente.

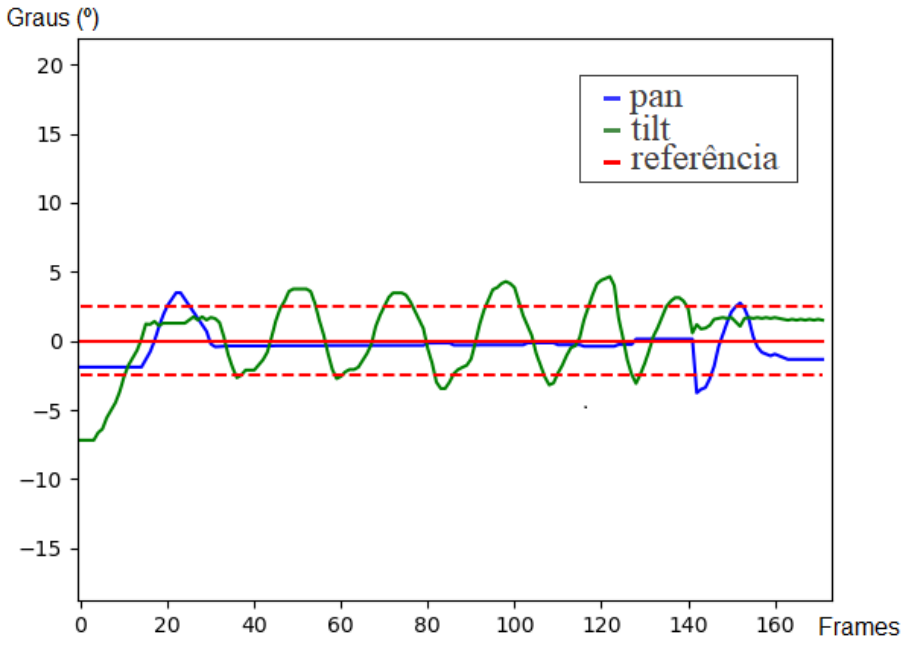


Figura 4.5: Resposta do controlador com alvo fixo, no eixo horizontal e vertical.

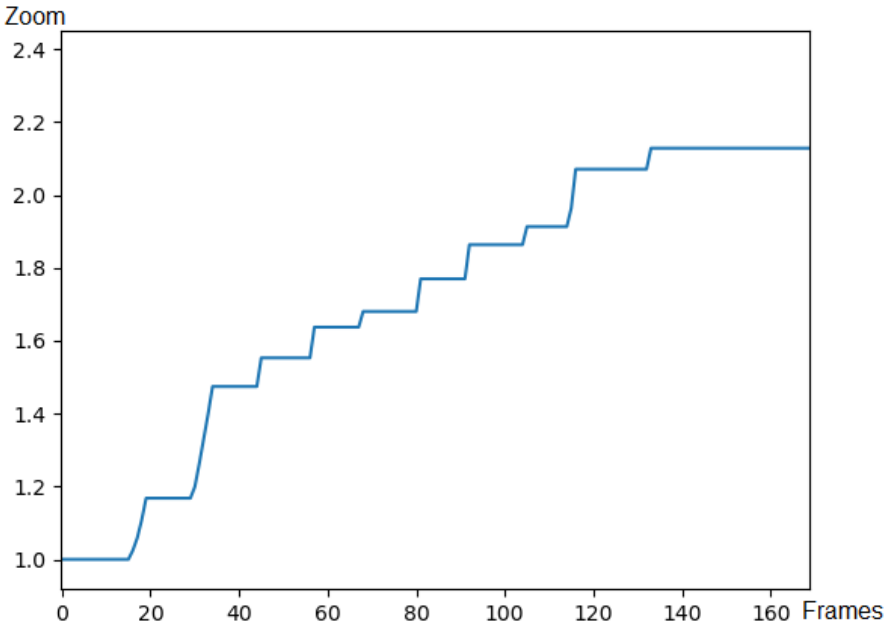


Figura 4.6: Fator de Zoom aplicado ao alvo fixo em cada frame.

As Figura 4.5 mostra o valor do erro, em graus, entre o angulo da câmera e o angulo em que a câmera deveria estar para que o centro do alvo se alinhe com o centro da imagem.

Nota-se que o angulo em *pan* e *tilt* para de oscilar após todo o zoom ser aplicado, obtendo um erro menor que 2,5 graus em cada eixo de rotação.

Pode-se perceber que, embora o controle servo visual se estabilize com um erro aceitável para capturar detalhes do indivíduo alvo, durante todo o tempo de zoom existe uma oscilação sobre o alvo. Isso se deve a restrição dos movimentos da câmera e ao atraso no tempo de resposta para um comando da câmera.

Os movimentos nos três graus de liberdade da câmera devem ser executados separadamente, ou seja, ao movimentar a câmera horizontalmente, não é possível executar o zoom simultaneamente, por exemplo. Essa restrição de comandos discutida na Seção 3.2 faz o sistema ter uma resposta mais lenta, por isso a resposta de *pan* e *tilt* apresentada na Figura 4.5 demora a entrar em regime permanente.

Metade dos testes realizados utilizando o detector a cada frame conseguiram focar no indivíduo, embora ocorresse maior oscilação em alguns casos. Já a abordagem com detector e rastreador trabalhando em conjunto, obteve 62% de resultados positivos, ou seja, esta abordagem conseguiu ampliar a imagem e focar no individuo alvo com mais frequencia. Além disso, a abordagem com rastreador possui um desempenho mais alto, chegando a 18 FPS, enquanto a abordagem com apenas detector não passava de 12 FPS.

Embora o método utilizando detector e rastreador tenham uma taxa de sucesso maior que a abordagem utilizando o detector, as falhas neste método eram piores, já que levavam o algoritmo a rastrear o plano de fundo em alguns casos. E devido ao plano de fundo não mudar, o rastreador sempre focava no plano de fundo após ocorrer uma falsa detecção, como vimos na Figura 4.3.

Logo, o controle servo visual utilizando apenas detector HOG/SVM tem melhor desempenho quando o algoritmo é executado por muito tempo, já que caso aconteça uma falsa detecção, na maior parte será pontual e eventualmente o controlador voltará para a posição inicial. A abordagem utilizando detector e rastreador possui um menor custo computacional e maior precisão ao seguir o alvo, no entento pode não conseguir voltar a posição inicial casa haja um falsa detecção e o rastreador comece a rastrear alguma parte do plano de fundo.

CAPÍTULO 5 – CONCLUSÕES

Como resultado final desse trabalho, foi desenvolvido dois softwares de controle servo visual para a câmera Speed Dome SL-130IPC851, com a finalidade de estabelecer uma comparação de desempenho entre eles. Com base na literatura, foram analisados algoritmos de processamento de imagens e de controle. Então efetuou-se testes individuais em cada bloco de código, para garantir a eficiencia de cada componente do controle servo visual isoladamente. Depois integrou-se esses blocos de códigos para a formação dos dois softwares finais.

Uma etapa importante na metodologia foi na escolha de utilizar algoritmos de aprendizagem de máquina para melhorar a detecção, pois inicialmente seria construido um software com detector feito por diferenciação temporal. Ao utilizar aprendizagem de máquina, abriu-se uma gama de opções para detectores e rastreadores, como na detecção por descritor HOG associado ao SVM ou os rastreadores MOSSE e KCF.

Após escolher individualmente o detector, rastreador e controlador, foram implementados dois softwares. Um contendo apenas o detector e o controlador, e o outro contendo detector, rastreador e controlador. Em ambos os softwares conseguiu-se alcançar resultados positivos ao perseguir e focar no individuo alvo, no entanto foram necessárias restrições para diminuir a quantidade de detecção de falsos positivos nas imagens. As restrições incluem utilizar cenários com poucos objetos e plantas, não ter grandes variações de luminosidade em alguma região da imagem e possuir uma resolução minima de 0,2 megapixels.

Embora o software com rastreador tenha obtido melhor precisão do alvo e menor custo computacional, a abordagem que não utilizou rastreador obteve uma maior tolerância a falhas na detecção.

Um problema encontrado no desenvolvimento do projeto deve-se a falta de informações sobre a comunicação com a câmera PTZ, já que os endereços URL e a estrutura da aplicação da câmera é diferente para cada marca. Devido a essa falta de informações tanto nos manuais da câmera quanto em projetos semelhantes, não foi possível identificar pela rede formas de obter informações de posição absoluta de rotação em *pan* e *tilt* ou o nível de zoom.

5.1 TRABALHOS FUTUROS

O trabalho apresentado pôde, através de uma câmera PTZ, rastrear e ampliar a imagem com zoom otico para pegar detalhes sobre o individuo. Esta linha de pesquisa possui uma gama de aplicações reais, já que os sistemas de vigilância estão em ascensão e novas tecnologias de hardware e software são lançadas com frequencia nessa área. Os trabalhos que se propõem são:

- Desenvolver um sistema que utiliza um detector facial, após a efetuação do zoom no individuo, para capturar fotos do rosto do indivíduo e cruzar com informações externas para realizar reconhecimento facial do individuo.
- Estudo de outras abordagens para os detectores e rastreadores utilizando redes neurais e fazendo um comparativo para melhorar o desempenho do sistema e diminuir a taxa de falhas na detecção.
- Adaptar este trabalho para detecção e rastreamento de grupos de pessoas em presídios, integrando o sistema em várias câmeras, com o intuito de identificar grupos e gangues em sistemas prisionais.

REFERÊNCIAS BIBLIOGRÁFICAS

AGREN, Sanna. **Object tracking methods and their areas of application: A meta-analysis**: A thorough review and summary of commonly used object tracking methods. 2016. 37 p. Master Thesis (Computer Science)- UMEA Universitet, [S.l.], 2016.

BERNARDES, Mariana Costa. **Controle servo visual para aproximação de portas por robôs móveis equipados com duas câmeras** . 2009. 82 p. Dissertação de mestrado (Mestrado em Engenharia Elétrica)- Universidade de Brasília, Brasília - DF, 2009.

BOLME, David S. **Visual Object Tracking using Adaptative Correlation Filters**. 2010. 10 p. Paper – Colorado State University, Fort Collins – CO, USA, 2010.

BriefCam Youtube Videos. **BriefCam video surveillance to actionable Intelligence**. 10 de Agosto de 2009. Disponível em: < https://www.youtube.com/user/BriefCamVS/videos>. Acesso em: 15 de Novembro de 2019.

CHANG, Wen-Chung; SHAO, Chia-Kai. **Hybrid fuzzy control of an eye-to-hand robotic manipulator for autonomous assembly tasks**. IEEE Proceedings of SICE Annual Conference 2010, National Taipei University of Technology, Taiwan, Republic of China, 2010.

CHAUMETTE, François; HUTCHINSON, S. **Visual servo control, Part I: Basic approaches**. IEEE Robotics and Automation Magazine, Institute of Electrical and Electronics Engineers, 2006, 13 (4), pp.82-90.

DALLAL, Navneet; TRIGGS, Bill. **Histogram of Oriented Gradients for Human Detection**. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. France.

Exposec. Mercado de segurança eletrônica projeta crescimento de 8% em 2018. **Exposec – Feira Internacional de Segurança**, São Paulo, 20 de Fevereiro de 2018. Disponível em: <https://www.learnopencv.com/histogram-of-oriented-gradients>. Acesso em: 4 de Junho de 2019.

FRANCO, David Bevilaqua de Sales Duarte. **Desenvolvimento de sistema de visão para movimentação de robô móvel PIONEER P3-AT** . 2011. 52 p. Dissertação (Graduado em Engenharia de Controle e Automação)- Universidade de Brasília, Brasília – DF, 2011.

GONZALEZ, Rafael C.; WOODS, Richard E. **Digital Image Processing 3º edition**. Prentice Hall, Upper Saddle River, New Jersey, USA, 2010.

HENRIQUES, João F.; CASEIRO, Rui; MARTINS, Pedro; BATISTA, Jorge. **High-Speed Tracking with Kernelized Correlation Filters**. 2015. 14 p. IEEE Transactions on Pattern Analysis and Machine Intelligence – University of Coimbra, Portugal, 2015.

HENZ, Renato Marino. **Controle Servo-visual de robôs** . 2014. 9 p. Artigo (Graduado em Engenharia de Controle e Automação)- Universidade de Brasília, [S.l.], 2014.

KIKUCHI, Davi Yoshinobu. **Sistema de controle servo visual de uma câmera pan-tilt com rastreamento de uma região de referência** . 2007. 106 p. Dissertação de mestrado (Mestrado em Engenharia)- Escola Politécnica da Universidade de São Paulo, São Paulo, 2007.

MACEDO, Marlon S.; AMARAL, Eduardo M. A. **Sistema de controle servo visual de uma câmera pan-tilt com rastreamento de uma região de referência** . 2017. 5 p. Artigo Científico - Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo, Rodovia ES 010, Km 6.5, Manguinhos CEP 29.173-084 - Serra - ES, 2017.

MALLICK, Satya. Histogram of Oriented Gradients. **Learn OpenCV**, San Diego, December 6, 2016. Disponível em: <https://www.learnopencv.com/histogram-of-oriented-gradients>. Acesso em: 27 de Novembro de 2019.

MENEZES, Rafael Perrone Bezerra de. **Controle servo visual de veículos aéreos multirrotores** . 2013. 144 p. Dissertação de mestrado (Mestrado em Mecatrônica)- Universidade Federal da Bahia, Salvador, 2013.

MUÑOZ, Gloria Liliana López. **Análise comparativa da técnicas de controle servo-visual de manipuladores robóticos baseadas em posição e em imagem** . 2011. 128 p. Dissertação de mestrado (Mestrado em Sistemas Mecatrônicos)- Universidade de Brasília, Brasília - DF, 2011.

PEREIRA, Rafael Cardoso. **Técnica de rastreamento e perseguição de alvo utilizando o algoritmo Haar cascade aplicada a robôs terrestres com restrições de movimento** . 2017. 105 p. Dissertação de mestrado (Mestrado em Engenharia Mecatrônica)- Universidade Federal do Rio Grande do Norte, Natal - RN, 2017.

VIOLA, Paul; JONES, Michael. **Rapid Object Detection using a Boosted Cascade of Simple Features** 2001. 9 p. Article – Conference on Computer Vision and Pattern Recognition 2001, [S.l.], 2001.

YILMAZ, Alper; JAVED, Omar; SHAH, Mubarak. **Object Tracking: A Survey**. 2006. 45 p. Article - Ohio State University, University of Central Florida, ObjectVideo, Inc, [S.l.], 2006.

ANEXOS

Anexo I – Especificações das câmeras utilizadas

Anexo II – Fluxograma da primeira abordagem para detectar e rastrear objetos

Anexo III – Código em python do controle servo visual utilizando detector e rastreador

Anexo IV – Código em python do controle servo visual utilizando apenas detector

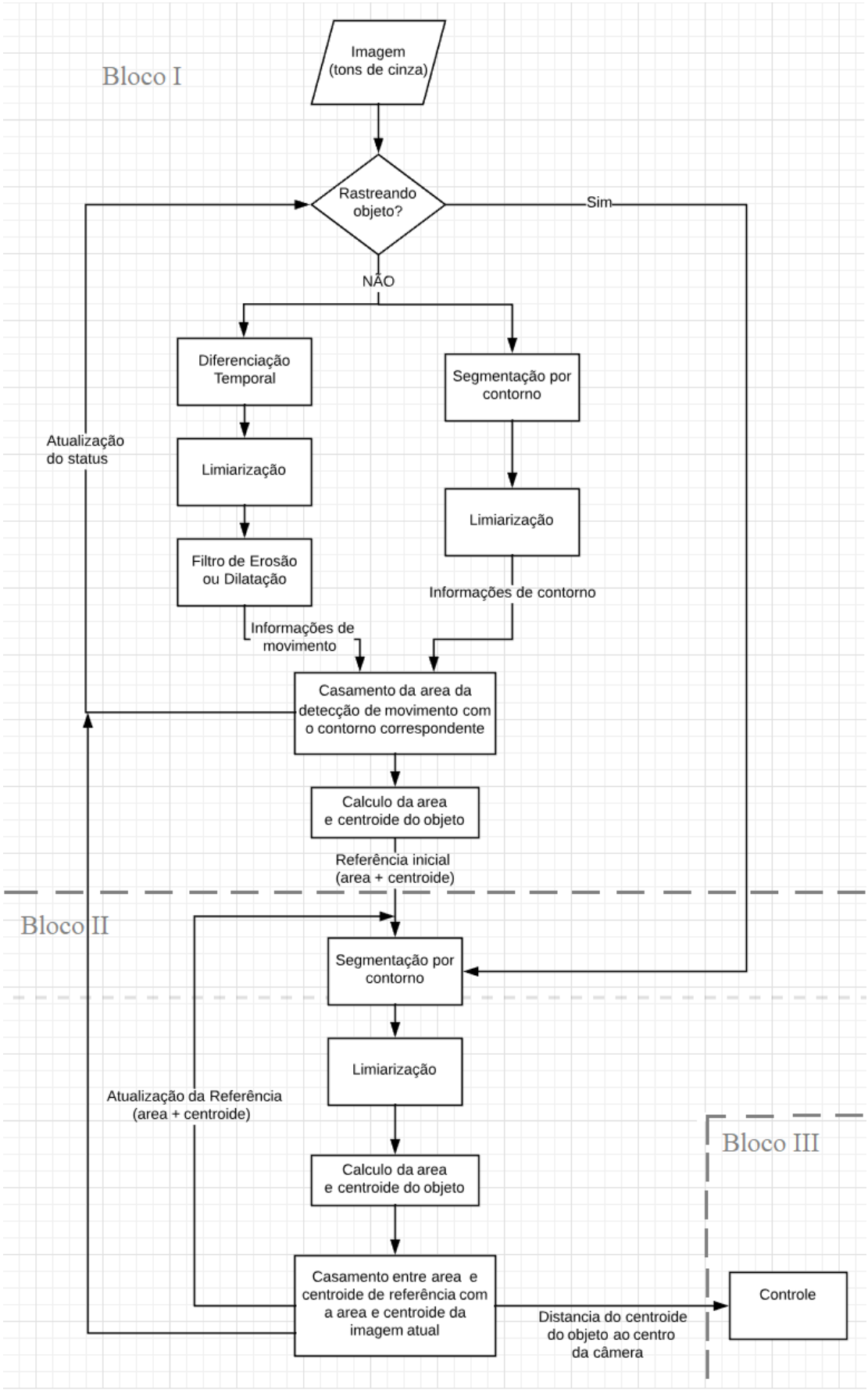
Câmera IPC-Z06H:

Sensor de Imagem:	1,0 Megapixels CMOS
Resolução de vídeo:	720p, 360p, 240p
Lentes:	2,8 mm
Rotação Pan:	355º
Rotação Titl:	120º
Iluminação Mínima:	0,5 lux
Distância Infravermelho:	10 m
Protocolo Wifi:	IEEE 802.11 b/g/n
Alimentação:	DC 5V / 2A
Protocolos suportados:	ONVIF, HTTP, TCP/IP, DDNS, DHCP, P2P, FTP

Câmera SL-130IPC851

Sensor de Imagem:	1,3 Megapixels CMOS
Resolução de vídeo:	960p, 720p, 360p
Lentes:	2,8 – 12 mm
Rotação Pan:	355º
Rotação Titl:	90º
Iluminação Mínima:	0,01 lux
Distância Infravermelho:	50 m
Protocolo Wifi:	IEEE 802.11 b/g/n
Alimentação:	DC 12V / 2A
Protocolos suportados:	ONVIF, HTTP, TCP/IP, DDNS, DHCP, P2P, FTP

ANEXO II: Fluxograma da primeira abordagem para detectar e rastrear objetos



```
import numpy as np
import cv2
import time
import pycurl
import math

# Constantes e Variaveis Globais
KP = 0.13563        # Calculado -> 640 <=> 86,8º = 0,13563 (valor antigo = 0.0703125)
KI = 0.02034        # Calculado -> 0.2*KP = 0,027186 (valor antigo = 0.035)
KPZ = 1.0           # Calculado -> 1.0 por não precisar de conversao
KIZ = 0.10          # Calculado -> 0.2*KPZ = 0.2
integralX = 0.0
integralY = 0.0
integralZ = 0.0
limitVect2 = 2.0    # Equivale a aproximadamente 15 pixeis de erro
limitZin = 1.5      # limite de zoomin: tela > limitZin*bbox
limitZout = 1.2     # limite de zoomout: tela < limitZout*bbox
zoomGlobal = 1.0

# Funcão de interface entre saida do controlador e o comando para executar a ação de controle
def interface(outx, outy, outz):
    # Contantes e Variaveis Globais
    global limitVect2
    global limitZin
    global limitZout
    global zoomGlobal

    # Calcula modulo do vetor (outx, outy)
    vect2 = outx*outx + outy*outy
    vect2 = math.sqrt(vect2)
    velAng = vect2 # velocidade angular é o modulo do vetor
    extraLimit = 0.2 * zoomGlobal

    # Verifica se o erro está acima de um limiar para então executar o controle
    if (vect2 < (limitVect2 + extraLimit)): # Abaixo do limiar - pode dar zoom caso precise
        direcao = "stop"
        velAng = 1
        if (outz > limitZin): # Se relacao de zoom (tela/box) for maior que limitZin, da zoomin
            direcao = "zoomin"
        elif (outz < limitZout): # Se relacao de zoom (tela/box) for menor que limitZout, da zoomout.
            direcao = "zoomout"
    elif(abs(outx) > abs(outy)): # Acima do limiar - precisa mover pan e/ou tilt
        if (outx > 0): # Ação de controle em X é maior que em Y, prioriza movimento em X
            direcao = "left"
        else:
            direcao = "right"
    else:
        if (outy > 0): # Ação de controle em Y é maior que em X, prioriza movimento em Y
            direcao = "up"
        else:
            direcao = "down"

    # Restringe velocidade maxima e minima para efetuar ações de controle
    if (velAng > 40):
        velAng = 40
    elif (velAng < 10):
        velAng = 10

    # retorna comando de controle para a câmera IP (direção e velocidade)
    return (direcao, velAng)


# Função responsavel pelos calculos do controlador PI
def controlador(erroX, erroY, erroZ, instanteAnterior):
    global zoomGlobal
    # Calcula variação de tempo entre frames
    instanteAtual = time.time()
    dt = instanteAtual - instanteAnterior

    # Constantes e Variaveis Globais
    global KP
    global KI
```

```
    global KZ
    global integralX
    global integralY
    global integralZ

    # Termos proporcionais
    px = KP * erroX
    py = KP * erroY
    pz = KPZ * erroZ

    # Termos integrais
    integralX += erroX * dt
    integralY += erroY * dt
    integralZ += erroZ * dt
    ix = KI * integralX
    iy = KI * integralY
    iz = KIZ * integralZ

    # Calcula o valor de saida
    outx = px + ix
    outy = py + iy
    outz = pz + iz

    # Retorna valores de saida do controlador
    return (outx, outy, outz, instanteAtual, dt)


# Função que envia mensagem para a CameraIP com o comando estabelecido pela interface
def comando(c, direcao, velocidade, intervaloTempo):
    global zoomGlobal

    if direcao == 'zoomin':
        multplicador = 1 + (intervaloTempo/3)
        zoomGlobal *= multplicador
    elif direcao == 'zoomout':
        multplicador = 1 - (intervaloTempo/3)
        zoomGlobal *= multplicador

    if zoomGlobal < 1.0:
        zoomGlobal = 1.0
    elif zoomGlobal > 4.0:
        zoomGlobal = 4.0
    #pqp = time.time()
    chave = '-step=0&-act=' + direcao + '&-speed=' + str(velocidade)
    c.setopt(c.POSTFIELDS, chave)
    c.perform()
    #pqp2 = time.time() - pqp
    #print(pqp2) aprox 0,02 s


#------------------------- INICIO ----------------------------------------------------
# Cria o classificador histograma de gradientes orientados (HOG) + maquinas de vetores de suporte (SVM)
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

# Inicializando variaveis da biblioteca PycURL para estabelecer conexao com CameraIP
c = pycurl.Curl()
c.setopt(c.URL, 'http://admin:admin@192.168.15.183:80/web/cgi-bin/hi3510/ptzctrl.cgi?')

# Inicializando variaveis auxiliares
detected = False
trackerInit = False

#Gravar Video
fourcc = cv2.VideoWriter_fourcc(*'DIVX')
out = cv2.VideoWriter('teste6/F01_009.avi',fourcc, 14.0, (640,352))

#Gravar informações do gráfico
dataset = open('graficos/dadosEstatico_005.txt', 'w')
intervalo = 0

# Loop principal do programa
while True:
    # Captura imagem da CameraIP
    cap = cv2.VideoCapture('http://admin:admin@192.168.15.183/tmpfs/auto.jpg?')
    ret, image = cap.read()

    # Efetua pre-processamentos na imagem
```

```
            image = cv2.GaussianBlur(image, (3,3), 0)
            orig = image.copy()

            # Caso nada tenha sido detectado, roda o algoritmo para detectar
            if not detected:
                trackerInit = False
                tracker = None

                # Pass frame to our body classifier
                (rects, weights) = hog.detectMultiScale(image, winStride=(4, 4),
                    padding=(8, 8), scale=1.1)

                # Extract bounding boxes for any bodies identified
                #for (x, y, w, h) in rects:
                #    cv2.rectangle(orig, (x, y), (x + w, y + h), (0, 0, 255), 2)

                # apply non-maxima suppression to the bounding boxes using a
                # fairly large overlap threshold to try to maintain overlapping
                # boxes that are still people
                rects = np.array([[x, y, x + w, y + h] for (x, y, w, h) in rects])
                pick = non_max_suppression(rects, probs=None, overlapThresh=0.2)

                # draw the final bounding boxes
                counter = 0
                for (xA, yA, xB, yB) in pick:
                    counter += 1
                    cv2.rectangle(image, (xA, yA), (xB, yB), (0, 255, 0), 2)
                    detected = True
                    tempo = time.time()
                    if counter == 1:
                        initBB = (xA, yA, xB-xA, yB-yA)
            # Caso um humano tenho sido detectado, roda o algoritmo de rastreamento no alvo detectado
            else:
                # Atualiza Rastreador para localizar o alvo
                if trackerInit:
                    (detected, box) = tracker.update(orig)
                    # check to see if the tracking was a success
                    if detected:
                        (x, y, w, h) = [int(v) for v in box]
                        cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)

                        # Calcula erros X, Y e Z (pan, tilt, zoom) para feedback do controlador
                        height, width, channels = image.shape
                        centroX = (width/2)
                        boxX = (x + w/2)
                        erroX = centroX - boxX          # CentroX - BoxX

                        centroY = (height/2)
                        boxY = (y + h/2)
                        erroY = centroY - boxY          # CentroY - BoxY

                        erroZ = min((width/w), (height/h))  # Minimo entre [LarguraImagem/LarguraBox ; AlturaImagem/AlturaBox]

                        #Escreve dados no txt
                        #dataset.write(str(centroX) + "\t" + str(centroY) + "\t" + str(boxX) + "\t" + str(boxY) + "\t" + str(erroZ) + "\t" +
str(zoomGlobal) + "\n")
                        dataset.write(str(erroX) + "\t" + str(erroY) + "\t" + str(erroZ) + "\t" + str(zoomGlobal) + "\n")

                        # Calcula e afetua Ações de controle
                        saidaX, saidaY, saidaZ, tempo, intervalo = controlador(erroX, erroY, erroZ, tempo)
                        direcao, velocidade = interface(saidaX, saidaY, saidaZ)
                        comando(c, direcao, velocidade, intervalo)
                        # print(zoomGlobal)

                # Inicializa rastreador caso ainda nao tenha se inicializado
                else:
                    trackerInit = True
                    tracker = cv2.TrackerKCF_create()
                    tracker.init(orig, initBB)

            if cv2.waitKey(1) == 13: #13 is the Enter Key
                break
            cv2.imshow('BLOCOS 1/2/3', image)
        dataset.close()
        out.release()
        cap.release()
        cv2.destroyAllWindows()
```

```
import numpy as np
import cv2
import time
import pycurl
import math

# Constantes e Variaveis Globais
KP = 0.07593        # Calculado -> 640 <=> 87º = 0,13593 (valor antigo = 0.0703125)
KI = 0.027186       # Calculado -> 0.2*KP = 0,027186 (valor antigo = 0.035)
KPZ = 1.0           # Calculado -> 1.0 por não precisar de conversao
KIZ = 0.10          # Calculado -> 0.2*KPZ = 0.2
integralX = 0.0
integralY = 0.0
integralZ = 0.0
limitVect2 = 2.0    # Equivale a aproximadamente 20 pixeis de erro
limitZin = 1.7      # limite de zoomin: tela > limitZin*bbox
limitZout = 1.3     # limite de zoomout: tela < limitZout*bbox
imgCounter = 0      # Conta quantidade de fotos tiradas na execução
zoomAtual = 1.00000 # Estima o zoom atual

def screenShot(img, bbox):
    # ae
    x, y, w, h = bbox
    ss_img = img[y:h, x:w]
    cv2.imshow("cropped", ss_img)

# Funcão de interface entre saida do controlador e o comando para executar a ação de controle
def interface(outx, outy, outz):
    # Contantes e Variaveis Globais
    global limitVect2
    global limitZin
    global limitZout
    global imgCounter

    # Calcula modulo do vetor (outx, outy)
    vect2 = outx*outx + outy*outy
    vect2 = math.sqrt(vect2)
    velAng = vect2 # velocidade angular é o modulo do vetor

    # Verifica se o erro está acima de um limiar para então executar o controle
    if (vect2 < limitVect2): # Abaixo do limiar - pode dar zoom caso precise
        direcao = "stop"
        velAng = 20
        if (outz > limitZin): # Se relacao de zoom (tela/box) for maior que limitZin, da zoomin
            direcao = "zoomin"
        elif (outz < limitZout): # Se relacao de zoom (tela/box) for menor que limitZout, da zoomout. Senão, nao mexe no
zoom
            direcao = "zoomout"
        else:
            direcao = "screenshot"
    elif(abs(outx) > abs(outy)): # Acima do limiar - precisa mover pan e/ou tilt
        if (outx > 0): # Ação de controle em X é maior que em Y, prioriza movimento em X
            direcao = "left"
        else:
            direcao = "right"
    else:
        if (outy > 0): # Ação de controle em Y é maior que em X, prioriza movimento em Y
            direcao = "up"
        else:
            direcao = "down"

    # Restringe velocidade maxima e minima para efetuar ações de controle
    if (velAng > 40):
        velAng = 40
    elif (velAng < 10):
        velAng = 10

    # retorna comando de controle para a câmera IP (direção e velocidade)
    return (direcao, velAng)


# Função responsavel pelos calculos do controlador PI
```

```
def controlador(erroX, erroY, erroZ, instanteAnterior):
    # Calcula variação de tempo entre frames
    instanteAtual = time.time()
    dt = instanteAtual - instanteAnterior

    # Constantes e Variaveis Globais
    global KP
    global KI
    global KZ
    global integralX
    global integralY
    global integralZ

    # Termos proporcionais
    px = KP * erroX
    py = KP * erroY
    pz = KPZ * erroZ

    # Termos integrais
    integralX += erroX * dt
    integralY += erroY * dt
    integralZ += erroZ * dt
    ix = KI * integralX
    iy = KI * integralY
    iz = KIZ * integralZ

    # Calcula o valor de saida
    outx = px + ix
    outy = py + iy
    outz = pz + iz

    # Retorna valores de saida do controlador
    return (outx, outy, outz, instanteAtual)


# Função que envia mensagem para a CameraIP com o comando estabelecido pela interface
def comando(c, direcao, velocidade, tpi, zoomAtual):
    #global zoomAtual

    # Pega a variação de tempo entre o ultimo comando e o comando atual
    tempoAtual = time.time()
    dt = tempoAtual - tpi

    # Atualiza o valor de zoom com base no tempo em q ele ficou acionado
    multiplicador = zoomAtual/10
    if direcao == "zoomin":
        zoomAtual += (multiplicador * dt )
    #elif direcao == "zoomout":
    #    zoomAtual = zoomAtual / multiplicador
    #print(zoomAtual)

    # Bloqueia valor de zoom para que: 1 < zoomAtual < 4
    if zoomAtual > 4.0:
        zoomAtual = 4.0
    elif zoomAtual < 1.0:
        zoomAtual = 1.0

    # Envia as informações de direção e velocidade do movimento para a câmera PTZ
    chave = '-step=0&-act=' + direcao + '&-speed=' + str(velocidade)
    c.setopt(c.POSTFIELDS, chave)
    c.perform()

    return tempoAtual, zoomAtual


#------------------------- INICIO ----------------------------------------------------
# Cria o classificador histograma de gradientes orientados (HOG) + maquinas de vetores de suporte (SVM)
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

# Inicializando variaveis da biblioteca PycURL para estabelecer conexao com CameraIP
c = pycurl.Curl()
c.setopt(c.URL, 'http://admin:admin@192.168.15.183:80/web/cgi-bin/hi3510/ptzctrl.cgi?')

# Inicializando variaveis auxiliares
detected = False
trackerInit = False
movel = False
```

```
#Grvar Video
fourcc = cv2.VideoWriter_fourcc(*'DIVX')
out = cv2.VideoWriter('teste5/F02_008.avi',fourcc, 12.0, (640,352))

tp = time.time()
tempo = time.time()
zp = 1
# Loop principal do programa
while True:
    # Captura imagem da CameraIP
    cap = cv2.VideoCapture('http://admin:admin@192.168.15.183/tmpfs/auto.jpg?')
    ret, image = cap.read()
    detected = False
    # Efetua pre-processamentos na imagem
    #image = cv2.resize(image, None,fx=0.7, fy=0.7, interpolation = cv2.INTER_LINEAR)
    #image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.GaussianBlur(image, (3,3), 0)
    orig = image.copy()

    # Caso nada tenha sido detectado, roda o algoritmo para detectar

    #gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Passa a imagem pelo classificador para detectar e retornar os rects
    (rects, weights) = hog.detectMultiScale(image, winStride=(4, 4),
        padding=(8, 8), scale=1.05)

    # Retira Overlaping de detecções com o algoritmo non_max_supression
    rects = np.array([[x, y, x + w, y + h] for (x, y, w, h) in rects])
    pick = non_max_suppression(rects, probs=None, overlapThresh=0.6)

    # Desenha as bbox's finais
    counter = 0
    for (xA, yA, wA, hA) in pick:
        counter += 1
        #cv2.rectangle(image, (xA, yA), (xA+wA, yA+hA), (0, 255, 0), 2)
        cv2.rectangle(image, (xA, yA), (wA, hA), (0, 255, 0), 2)
        detected = True
        #tempo = time.time()
        if counter == 1:
            initBB = (xA, yA, wA, hA)
    if detected:
        # Calcula erros X, Y e Z (pan, tilt, zoom) para feedback do controlador
        x, y, wp, hp = initBB
        w = wp-x
        h = hp-y
        height, width, chan = image.shape
        erroX = (width/2) - (x + w/2)       # CentroX - BoxX
        erroY = (height/2) - (y + h/2)      # CentroY - BoxY
        erroZ = min((width/w), (height/h))  # Minimo entre [LarguraImagem/LarguraBox ; AlturaImagem/AlturaBox]

        # Calcula e afetua Ações de controle
        saidaX, saidaY, saidaZ, tempo = controlador(erroX, erroY, erroZ, tempo)
        direcao, velocidade = interface(saidaX, saidaY, saidaZ)
        if direcao == "screenshot":
            screenShot(image, initBB)
        else:
            tp, zp = comando(c, direcao, velocidade, tp, zp)
            movel = True
    else:
        if movel == True:
            tp, zp = comando(c, "stop", 10, tp, zp)
            movel = False


    if cv2.waitKey(1) == 13: #13 é a tecla ENTER
        break
    #cv2.imshow('Pedestrians', orig)
    cv2.imshow('BLOCOS 1/2/3', image)
    out.write(image)

out.release()
cap.release()
cv2.destroyAllWindows()
```