UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

DANILO COLOMBO

Rastreamento visual de objetos utilizando filtro robusto

São Carlos
2012

DANILO COLOMBO

Rastreamento visual de objetos utilizando filtro robusto

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos da Universidade de São Paulo do Curso de Engenharia Mecatrônica.

Orientador: Prof. Dr. Marco Henrique Terra

São Carlos

2012

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica preparada pela Seção de Atendimento ao Usuário do Serviço de Biblioteca – EESC/USP

C718r

Colombo, Danilo
Rastreamento visual de objetos utilizando filtro robusto. / Danilo Colombo ; orientador Marco Henrique Terra. São Carlos, 2012.

Monografia (Graduação em Engenharia Mecatrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2012.

1. Robótica. 2. Filtragem robusta. 3. Kalman. 4. Controle. 5. Rastreamento visual. 6. Acompanhamento de trajetória. 7. Algoritmos genéticos. 8. Filtro de partículas. I. Título.

FOLHA DE AVALIAÇÃO

Candidato(s): Danilo Colombo

Título: Acompanhamento de trajetória de alvos móveis utilizando filtro robusto

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

BANCA EXAMINADORA

Dr. Roberto Santos Inoue

Nota atribuída: nove e meio (9,5)

(assinatura)

Prof. Dr. Valdir Grassi Junior

Nota atribuída: nove e meio (9,5)

Valdir Grassi Junior

(assinatura)

Prof. Dr. Marco Henrique Terra (orientador)

Nota atribuída: nove e meio (9,5)

Marco Henrique Terra

(assinatura)

Média: nove e meio (9,5)

Resultado: aprovado.

Data: 04/07/2012

À minha filha Rafaela que foi minha inspiração
e motivação. Aos meus pais, Ludmila e Dimas,
meu irmão, Marcelo e minha namorada,
Rosemari, com amor.

Agradecimentos

- Aos meus pais, Ludmila e Dimas, e meu irmão Marcelo por toda ajuda e encorajamento ao longo de todos estes anos.
- À minha filha Rafaela por me servir de inspiração e motivação todos os dias da minha vida. A minha namorada Rosemari pelo companheirismo durante esta jornada.
- Ao Prof. Dr. Marco Henrique Terra por toda a orientação e paciência devotada a mim, sem as quais este projeto seria impossível.
- Aos companheiros de laboratório, Roberto Santos Inoue e João Paulo Cerri.
- Aos meus companheiros de classe, em especial ao Matheus Pestana, Daniel Centurion, Henrique Ramos, Danillo Tenório, Luiz Bisan, Péricles Bernardes, Fábio Machuca, Augusto Hirao e Denis Toyoshima.
- À todos os professores da Mecatrônica.
- À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) pelo suporte financeiro.
- À Deus, por toda minha família e amigos.

"Não há nada bom nem mau a não serem estas duas coisas: a sabedoria que é um bem e a ignorância que é um mal".

Platão

Resumo

COLOMBO, D. **Rastreamento visual de objetos utilizando filtro robusto**. 2012. 124 f. Trabalho de Conclusão de Curso (Engenharia Mecatrônica) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2012.

Este projeto de formatura propõe um estudo sobre o rastreamento visual de um objeto alvo utilizando filtro robusto recursivo ótimo para reduzir a área de busca do alvo na imagem. Sendo estudada inicialmente a identificação de objetos em imagens dinâmicas e posteriormente a elaboração de um modelo dinâmico, em espaço de estado, para o movimento do objeto. O filtro de Kalman será estudado e implementado para realizar estimativas de posições futuras do alvo e assim reduzir a área de busca do alvo na imagem. Espera-se reduzir o tempo e o peso computacional com esta redução de área. Serão implementados dois outros filtros robustos baseados no filtro de Kalman, o filtro BDU e o filtro robusto recursivo ótimo, porém com abordagens distintas. Algoritmos genéticos serão usados para a parametrização dos filtros. É construído também um filtro de partículas para o rastreamento visual. Um estudo comparativo do desempenho dos quatro filtros será levantado ao final.

Palavras-chave: Robótica. Filtragem robusta. Kalman. Controle. Rastreamento Visual. Algoritmos genéticos. Filtro de partículas.

Abstract

COLOMBO, D. **Robust filtering for object visual tracking**. 2012. 124 f. Trabalho de Conclusão de Curso (Engenharia Mecatrônica) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2012.

This graduation project proposes a study on visual tracking of a target object using recursive optimal robust filter to reduce the search area in the target image. At first studying how to identify objects in dynamic images, and subsequently the development of a dynamic model in state space for the movement of the target. The Kalman filter will be studied and implemented to make estimates of future positions of the target and thus reduce the search area. Two robust filters will be studied and implemented based on the robust Kalman filter, the BDU and the recursive optimal robust, but with different approaches. Genetic algorithms will be used for parameterization of the filters. A condensation algorithm is also constructed to perform the tracking. A final performance comparison of three filters will be lifted at the end.

Keywords: Robotics. Moving target tracking. Kalman. Robust filtering. Genetic Algorithms. Condensation Algorithm.

Lista de Figuras

Figura 1- Diagrama de blocos para rastreamento visual da trajetória de um alvo 14
Figura 2 - (a) Imagem RGB (b) Imagem HSV (c) Matriz de Hue (d) Thresholding usando hmin= 0.035 e hmax= 0.06 (e) Thresholding usando hmin= 0.05 e hmax= 0.06 31
Figura 3 - (a) Seleção feita pelo usuário usando a função pré-definida roipoly (b) Mascara resultante 33
Figura 4 - (a) Segmentação RGB usando $D = 10$ (b) $D = 32$ (c) $D = 64$ 34
Figura 5 - (a) Resultado obtido na segmentação (b) Erosão utilizando elemento estrutural de tamanho 2, eliminando o ruído sem alterar muito a segmentação (c) Tamanho 10, além do ruído, grande parte da região de interesse foi erodida 37
Figura 6 - Centro de massa da esfera usando segmentação RGB seguida de erosão e calculada usando os momentos invariantes. A posição indicada é $x = 313$ e $y = 434$ 40
Figura 7 - Sequência de resultados obtidos mostrando a posição da esfera e a trajetória percorrida até o momento. Podem ser vistos mais detalhes na Figura abaixo. O número do quadro está indicado em cada figura 42
Figura 8 - Trajetória: A linha em azul marca a trajetória segundo o algoritmo de visão computacional e os pontos mostram a trajetória prevista pelo Filtro 43
Figura 9 - Gráfico mostrando a posição resultante do algoritmo de visão e a posição estimada pelo filtro de Kalman 43
Figura 10 - Circunferências mostrando a posição estimada, vermelha e a posição calculada pelo algoritmo de visão, verde 44
Figura 11 - Erros por iteração e erro médio, com e sem o Filtro de Kalman. 46
Figura 12 - Variação do MSEmed conforme o parâmetro P1 varia. O ponto com melhor MSEmed, ou seja, melhor desempenho do filtro é o ponto $T = 1$.....53
Figura 13 - Resposta do filtro em função de P2(a) e P3(b) 55
Figura 14 - Resposta do filtro em função de P5 encontrado para α foi 1,6.....56
Figura 15 - Resposta obtida variando-se a amplitude da matriz Mf. A linha vermelha indica o desempenho do filtro nominal. Note que há uma faixa de valores para os quais o desempenho do filtro robusto é superior 57
Figura 16 - Esquema de um algoritmo genético 60
Figura 17 - Evolução do desempenho do filtro BDU ao longo das gerações 61
Figura 18- Evolução do desempenho do filtro robusto recursivo ótimo ao longo das gerações 66
Figura 19 - Desempenho do filtro robusto recursivo com relação ao μ 68
Figura 20 - Desempenho do filtro robusto recursivo com relação ao α. Conforme μ aumenta o desempenho do filtro se torna independente do valor de α 69
Figura 21 - Desempenho do filtro robusto recursivo com relação as matrizes M_f e M_h 70
Figura 22 - Oscilação da posição dentro da esfera.....73
Figura 23 - Comparativo entre os diferentes filtros implementados 74
Tabela 2 - Estatísticas dos três filtros mostrando as vantagens de se utilizar o Filtro KRR.....75
Figura 24 – Propagação da função densidade ou difusão para um processo linear com ruídos gaussianos 78
Figura 25 – Difusão para processos não lineares ou com ruídos não gaussianos. Note como há deformações na onda.....79
Figura 26 - Amostragem aleatória. A função contínua é amostrada e cada uma destas amostras recebe um peso igual a densidade de probabilidade do ponto correspondente. As amostras ou partículas com maior área são correspondentes a maiores densidades.....82
Figura 27 - Ilustração do funcionamento do filtro de partículas ou condensation algorithm 83

Tabela 3- Exemplo de amostragem com probabilidade proporcional ao peso de cada partícula 85
Figura 28 - Comparação dos quatro filtros implementados 86
Tabela 4 - Comportamento do filtro de partículas em relação ao número de partículas.....87
Figura 29 - Distribuição dos pesos amostrais referente ao 9º quadro do vídeo. Percebe-se que apenas 3 partículas contribuem significamente para o resultado.....87
Figura 30- Distribuição dos pesos amostrais referente ao 23º quadro do vídeo. Percebe-se que apenas 1 partículas contribuem significamente para o resultado.....88
Figura 31- Tamanho amostral efetivo durante as iterações. Este gráfico mostra que o filtro sofre grande degeneração 89
Figura 32 - Sensor KINECT da Microsoft 91
Figura 33 - Desempenho dos algoritmos de localização dos alvos em relação à velocidade do alvo ...94
Figura 34 - Ilustração do problema na diferenciação temporal. Deve distinguir entre posição atual do objeto e posição passada do objeto 94
Figura 35 - Função limiarização ou thresholding 101
Figura 36 - Identificação do alvo pelo usuário 101
Figura 37 - Função que realiza a segmentação da imagem em RGB (Parte I) 102
Figura 38 - Função que realiza a segmentação da imagem em RGB (Parte II).....103
Figura 39 - Função imstack2vectors 103
Figura 40 - Função que retorna a posição do “centro de massa” de uma imagem binária 104
Figura 41 - Programa principal realiza o rastreamento visual de trajetória do alvo (Parte I).....105
Figura 42 - Programa principal, realiza o rastreamento visual de trajetória do alvo (Parte II) 106
Figura 43 - Função que extrai o alvo da imagem, retornando sua posição e o raio médio 107
Figura 44 - Implementação do filtro BDU (Parte I) 108
Figura 45 - Implementação do filtro BDU (Parte II) 109
Figura 46 - Função gerapop usada para criar uma população 112
Figura 47 - Função fitness que avalia o desempenho do filtro para um determinado conjunto de incertezas.....112
Figura 48 - Esquema do funcionamento de uma roleta, onde pode ser visto que cada área é proporcional ao desempenho relativo do indivíduo correspondente.....113

Lista de Siglas

VANT = Veículo Aéreo não Tripulado

RGB = Red, Green, Blue (Vermelho, Verde, Azul)

HSV= Hue, Saturation, Value (Matriz, Saturação, Valor)

YCbCr = Luminance, Chrominance Blue, Chrominance Red

HSI= Hue, Saturation, Intensity (Matriz, Saturação, Intensidade)

CIE= Commission Internationale de l'Éclairage (Comissão Internacional de Iluminação)

BDU= Bounded Data Uncertaints (Dados com Incertezas Limitadas)

MSE= Mean Square Error (Erro quadrado médio)

AG= Algoritmo Genético

fdp= função distribuição de probabilidade

Lista de símbolos

$\mathbb{R}$	corpo dos números reais
$\mathbb{R}^n$	conjunto dos vetores de dimensão n
$\mathbb{R}^{n\times m}$	conjunto das matrizes de dimensão n × m
A^T	transposta da matriz A
A^{-1}	inversa da matriz A
$\|\bullet\|$	norma euclidiana de um vetor ($\|x\|^2 = x^T x \; para \; x \in \mathbb{R}^n$)
$\|\bullet\|_P$	norma ponderada de um vetor ($\|x\|_P^2 = x^T P x \; para \; x \in \mathbb{R}^n$)
$x^T P(\bullet)$	expressão simplificada para $x^T P x$
E	esperança matemática
A^C	complemento de A
$\hat{x}$	estimativa de x, sendo x o valor real

Sumário

1 Introdução....13
1.1 Justificativas....14
1.2 Objetivos....14
2 Modelagem em espaço de estados....17
2.1 O sistema de rastreamento visual de trajetórias....17
2.2 Modelo dinâmico linear....17
3 Filtro de Kalman....23
3.1 Introdução....23
3.2 Origens do filtro de Kalman....24
3.3 Equações do filtro nominal e ajustes paramétricos....25
4 Visão computacional....27
4.1 Imagens digitais....27
4.2 Segmentação de imagens....28
4.2.1 Limiarização....29
4.2.2 Segmentação de imagens em RGB....31
4.2.3 Subtração do fundo....34
4.3 Melhorias no processo de segmentação....35
4.4 Vizinhança e conectividade de um pixel....37
4.5 Extração de informações....38
5 Resultados parciais....41
6 Interpretação dos resultados pré-liminares....47
7 Rastreamento visual de trajetórias utilizando filtro robusto BDU....49
8 Algoritmos genéticos....59
9 Filtro robusto recursivo....63
9.1 Resultados obtidos com o filtro robusto recursivo....65
10 Redução da área de busca do alvo na imagem....73
11 Filtro de partículas....77
11.1 Propagação de densidade de probabilidades condicionadas....77
11.2 Propagação da densidade em tempos discretos....79
11.3 Amostragem aleatória e o condensation algorithm....81
11.4 Simulação do filtro de partículas....83
11.5 Variações do filtro de partículas....87

12 Aplicação em Tempo Real 91
12.1 Oclusões, múltiplos objetos e ambiente tumultuado 93
13 Conclusão 97
Referências 99
APÊNDICE A 101
APÊNDICE B 111

1 Introdução

Veículos aéreos não tripulados (VANTs) podem ser equipados com câmeras de vídeo. As informações fornecidas pelas imagens dessas câmeras são úteis para interpretações de movimentos dinâmicos. Um importante campo de pesquisa de processamento de vídeo para VANT é o rastreamento visual de alvos móveis, que podem ser, por exemplo, veículos, pessoas ou aeronaves. Porém, não é tarefa fácil a detecção e o rastreamento visual de alvos quando eles se confundem com os fundos das imagens.

Existem várias técnicas para detecção de alvos e rastreamento visual através de imagens, veja [3], [4], [9], [11], [12], [10], [13], [14] e as referências contidas nelas. Quando este tipo de problema é considerado com o fundo da imagem estático ele se torna mais fácil de ser tratado. Em geral, algoritmos de visão são complexos e computacionalmente pesados, o que dificulta a utilização deles em tempo real. Uma alternativa para este problema é fazer a estimativa da posição do alvo utilizando filtros de Kalman.

A redução do espaço de busca da imagem dinâmica completa para uma janela de busca menor tem sido uma estratégia utilizada para aliviar o custo computacional para este tipo de problema. Essa janela é centrada em torno da estimativa de uma posição futura do objeto a ser seguido.

É um fato bastante conhecido que o desempenho do filtro de Kalman se deteriora quando existem incertezas paramétricas no modelo a ser estimado. Neste caso, estimativas baseadas em filtragem robusta limitam a degradação do desempenho dessas estimativas, veja [1] e [5]. Nesse sentido, este projeto de formatura propõe o desenvolvimento de um sistema de rastreamento visual de um alvo em um vídeo utilizando um filtro robusto ótimo para reduzir a área de busca do alvo na imagem, baseado no filtro de Kalman robusto proposto no capitulo 9. A vantagem dessa metodologia, se comparada com o que existe na literatura, é que ela pode ser utilizada em tempo real, pois não requer nenhum ajuste de parâmetro auxiliar. Trata-se de um filtro ótimo, robusto e recursivo. Vale ressaltar que a estabilidade do filtro fica garantida nessas implementações.

O funcionamento do rastreamento visual com redução da área de busca pode ser visto na Figura 1.

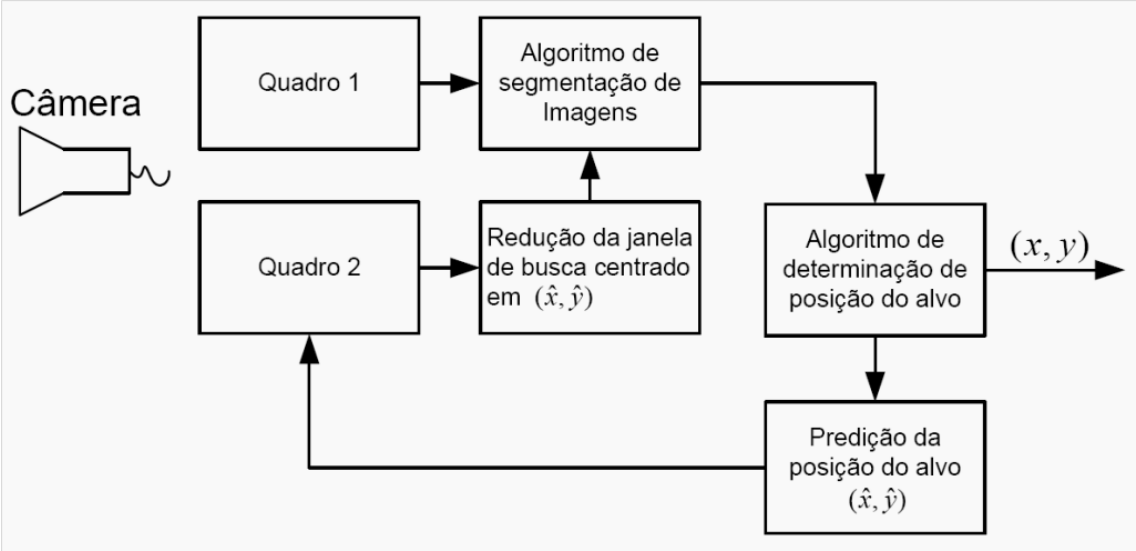


Figura 1- Diagrama de blocos para rastreamento visual da trajetória de um alvo

1.1 Justificativas

Veículos aéreos não tripulados (VANTs) têm sido utilizados para aplicações nos setores agrícola, elétrico, petroleiro e militar. Como exemplo, podemos citar: o monitoramento e mapeamento de regiões, aplicações de defensivos agrícolas, identificação de problemas e possíveis falhas em linhas de transmissão e dutos de petróleo. Assim, é papel fundamental da universidade desenvolver projetos que envolvam tecnologia de VANTs, de maneira a formar engenheiros com habilidades interdisciplinares que possam trabalhar com essa tecnologia.

1.2 Objetivos

O objetivo deste projeto de conclusão de curso é o desenvolvimento de um sistema de rastreamento visual de um alvo móvel em um vídeo utilizando um filtro robusto ótimo para reduzir a área de busca do alvo na imagem.

Procura-se através da construção de filtros robustos melhorar a estimativa de posição futura do alvo. Pretende-se realizar um comparativo de desempenho dos filtros mais amplo do que o encontrado na literatura. Serão adicionados neste trabalho, em relação ao normalmente encontrado nos textos do assunto, o filtro recursivo robusto ótimo, Capitulo 9, e o filtro de partículas, Capitulo *11*.

Ainda será dado início a um estudo de parametrização dos filtros robustos, isto é, a escolha correta das matrizes de incertezas. Existem vários métodos para essa sintonização, o que poderia render vários trabalhos, porém será explorado um destes métodos, os algoritmos genéticos. Além de

possuírem aplicação neste caso, podem ser aplicados em vários outros problemas onde se deseja encontrar variáveis que otimizam um processo, especialmente em inteligência artificial.

2 Modelagem em espaço de estados

2.1 O sistema de rastreamento visual de trajetórias

Uma introdução sobre as várias técnicas utilizadas no rastreamento visual pode ser encontrada em [23]. Nos sistemas típicos de rastreamento visual de alvos móveis, ou *tracking*, há um modelo para o movimento do objeto de interesse e algum conjunto de medidas obtidas a partir de uma sequência de imagens. Estas medidas podem ser a posição de um determinado ponto do objeto, a posição e o momento de uma região do objeto, a textura, a curvatura, ou qualquer outra informação relacionada.

Rastreamento visual pode ser visto como um problema de inferência estatística, visto que a dificuldade chave do problema é manter uma estimativa da posição a priori, dado o conjunto de medidas, e fazer isso de forma computacionalmente eficiente.

Os objetos são modelados como tendo um estado interno, isto é, um conjunto de variáveis que definem o comportamento deles. O estado interno de um objeto no k-ésimo quadro é tipicamente escrito como $\boldsymbol{X_k}$. As medidas obtidas no k-ésimo quadro são componentes de um vetor de medidas $\boldsymbol{Y_k}$.

Tracking é de modo geral um problema difícil, e para ser feito de maneira eficiente, são necessárias algumas hipóteses simplificadoras. A primeira delas é considerar somente o passado imediato do estado:

$$P(\boldsymbol{X_{k+1}}|\boldsymbol{X_0},\ldots,\boldsymbol{X_k}) = P(\boldsymbol{X_{k+1}}|\boldsymbol{X_k});$$

Esta hipótese traz uma enorme simplificação para o problema, facilitando a construção do algoritmo e aliviando o uso de memória. Pode se também dizer que não é uma simplificação muito restritiva se for feita a escolha adequada das variáveis de estado $\mathbf{X}_i$. Uma segunda hipótese simplificadora é que as medidas dependem somente do estado atual do sistema:

$$P(\boldsymbol{Y_k},\boldsymbol{Y_i},\boldsymbol{Y_j}|\,\boldsymbol{X_k}) = P(\boldsymbol{Y_k}|\,\boldsymbol{X_k})P(\boldsymbol{Y_i},\ldots,\boldsymbol{Y_j}|\,\boldsymbol{X_k});$$

Assume-se que o vetor de medidas $\boldsymbol{Y_i}$ é condicionalmente independente de todas as outras medidas dado $\boldsymbol{X_i}$, ou seja, depende apenas do estado atual do sistema.

2.2 Modelo dinâmico linear

Como pode ser notado no tópico anterior, lidando com o rastreamento visual como um problema de inferência estatística, tem-se o inconveniente de trabalhar com as densidades de probabilidade. Trabalhar com essas densidades pode acarretar cálculos de integrais, multiplicação de funções probabilidades, entre outras operações que consumiriam muito tempo de processamento. Solucionar o problema de forma eficiente é fundamental em um sistema de tempo real.

Neste sentido, o filtro de Kalman que será apresentado na próxima seção possui inúmeras vantagens que tornam a solução dos problemas de filtragem computacionalmente viáveis.

O caso mais simples surge quando as dinâmicas dos sistemas são lineares. Para o uso eficiente e acurado do filtro de Kalman adota-se além de um modelo linear para o sistema, um modelo linear para o processo de medição e ainda outra hipótese muito importante, os ruídos envolvidos são gaussianos. Isto significa que os ruídos presentes tanto no modelo quanto nas medidas serão modelados como tendo distribuição normal com média zero e variância conhecida.

Esta hipótese, apesar de nem sempre condizer com a realidade, é uma boa aproximação para a maioria dos sistemas de tracking. É uma propriedade estatística conhecida, que quando se somam os efeitos de várias fontes de ruídos, o sinal resultante tende a ter uma distribuição normal. Caso esta hipótese não se verifique adequada, deve-se buscar outra ferramenta, como por exemplo, o filtro de partículas.

Ainda é necessário mais um passo antes da criação do modelo. Quase sempre não se conhece o movimento do alvo, pelo contrário, se fosse conhecido não faria sentido algum à elaboração de um mecanismo de rastreamento visual. Na verdade, esta é mais uma consideração que deve ser feita antes de gerar o modelo.

O modo mais simples de representar a dinâmica de um alvo é quando seu estado é avançado no tempo multiplicando-se este por uma matriz conhecida, que pode ou não variar com o tempo, e então se adiciona uma variável aleatória com distribuição normal, média zero e covariância conhecida. Do mesmo modo, a medida é obtida multiplicando-se o estado por uma matriz conhecida, que pode ou não variar com o tempo, e novamente adicionando-se uma variável aleatória com média zero e variâncias conhecidas. O número de informações contidas no vetor de estado é que irá determinar o grau do modelo, como os exemplos apresentados a seguir.

Denota-se T a taxa de captura do quadro do vídeo (medida em segundos/quadro), $\{a_{x,k}, a_{y,k}\}$ as acelerações do alvo ao longo das direções de x e y (medidas em pixel/segundos²) e $\{v_{x,k}, v_{y,k}\}$ as velocidades dessas direções durante o k-ésimo quadro (medido em pixels/segundos).

Caracterização de pontos flutuantes: Neste caso assume-se que o vetor de estados encerra somente a posição do alvo. Sendo assim, a nova posição do alvo é dada pela antiga posição somada a um ruído gaussiano. Este modelo, sendo o mais simples possível, dificilmente poderá ser útil, visto que nesta representação o alvo aparenta estar num estado estacionário. Somente é usado quando não há outro modelo ou quando a componente aleatória é muito alta.

Apesar da simplicidade, este modelo satisfaz ao propósito de obter estimativa, ou seja, dado um número suficiente de medidas será possível reconstruir o estado. Diz-se então que o estado é observável e obtido por

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{w_k}$$

sendo $\boldsymbol{w_k}$ o vetor de ruído do sistema.

Velocidade constante: Neste caso, o vetor de estado do sistema encerra além da posição do alvo, também sua velocidade. Desta forma, a nova posição do alvo será da pela sua posição antiga mais o produto da velocidade pelo período de amostragem, acrescida, obviamente, pelo ruído gaussiano. Enquanto isso, a nova velocidade do alvo, será dada pela velocidade anterior mais uma parcela de ruído. A medida constitui-se apenas da posição do alvo, não se medindo sua velocidade. Mas isto não importa, pois como o estado é observável, espera-se que com apenas a posição consiga-se estimar todo o vetor de estado de acordo com o seguinte modelo:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ v_{x,k+1} \\ v_{y,k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x_k \\ y_k \\ v_{x,k} \\ v_{y,k} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} w_k$$

Aceleração constante: nota-se que desde os mais simples até os mais complexos modelos podem ser admitidos para caracterizar o movimento dos alvos. Modelos mais simples funcionam com mais velocidade e exigem menor capacidade computacional, porém são menos acurados. Modelos mais complexos podem descrever sistemas num nível de precisão elevado, mas consomem mais tempo para execução e são computacionalmente mais pesados.

Deve-se adotar então um modelo que seja acurado o suficiente para não “perder o alvo de vista” e ao mesmo tempo possibilite uma implementação em tempo real. O modelo adotado neste trabalho foi um modelo de aceleração constante.

O vetor de estado contém, neste modelo, além da posição e da velocidade, também a aceleração do alvo. A posição e a velocidade são atualizadas temporalmente e a aceleração em princípio seria constante, mas deve-se ressaltar que todas as variáveis são acrescidas do ruído gaussiano a cada iteração. A única variável medida é a posição, e a partir desta medida e do modelo pretende-se estimar todo o estado. Considere

$$v_{x,k+1} \approx v_{x,k} + a_{x,k}T \qquad (1)$$

$$v_{y,k+1} \approx v_{y,k} + a_{y,k}T \qquad (2)$$

$$x_{k+1} \approx x_k + v_{x,k}T + a_{x,k}\frac{T^2}{2} \qquad (3)$$

$$y_{k+1} \approx y_k + v_{y,k}T + a_{y,k}\frac{T^2}{2} \qquad (4)$$

introduzindo os vetores de estado e de medida

$$\boldsymbol{x_k} := [x_k \quad y_k \quad v_{x,k} \quad v_{y,k} \quad a_{x,k} \quad a_{y,k}]^T \qquad (5)$$

$$\boldsymbol{z_k} := [x_k \quad y_k]^T \qquad (6)$$

pode-se definir o modelo no espaço de estado nominal

$$\boldsymbol{x_{k+1}} = F\boldsymbol{x_k} + G\boldsymbol{w_k} \qquad (7)$$

$$\boldsymbol{z_k} = H\boldsymbol{x_k} + \boldsymbol{v_k} \qquad (8)$$

sendo

$$F = \begin{bmatrix} 1 & 0 & T & 0 & O.5T^2 & 0 \\ 0 & 1 & 0 & T & 0 & 0.5T^2 \\ 0 & 0 & 1 & 0 & T & 0 \\ 0 & 0 & 0 & 1 & 0 & T \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \qquad (9)$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \qquad (10)$$

e $\boldsymbol{w_k}$ e $\boldsymbol{v_k}$ são ruídos Gaussianos com média zero e descorrelacionados.

3 Filtro de Kalman

3.1 Introdução

No capítulo anterior foi descrito o sistema dinâmico para o rastreamento visual de alvos móveis, representado em espaço de estados. O vetor de estado possui uma certa quantidade de variáveis que do ponto de vista do modelo são suficientes para uma completa descrição do sistema. Se fosse possível medir todas estas variáveis, sem a presença de ruídos e interferências, poder-se-ia ter a todo o momento uma representação ideal do sistema. Embora, às vezes seja possível medir todas as variáveis de um sistema dinâmico, o caso mais comum é quando apenas algumas variáveis são mensuráveis, seja por impossibilidade ou inviabilidade de se medir todas. Além disso, é pouco provável que uma medida seja totalmente confiável. Desta maneira o objetivo do filtro de Kalman é, a partir de um conjunto de medidas corrompidas, estimar todo o estado do sistema dinâmico, desde que este seja observável, e fazê-lo de forma eficaz. O filtro é dito ótimo no sentido de minimizar o erro quadrado médio entre o valor estimado e o valor real do vetor de estado.

O Filtro de Kalman permite que sejam feitas estimativas preditoras, filtradas e suavizadas. O filtro que vamos considerar neste trabalho é baseado em predição e filtragem. Estimativas preditoras são obtidas quando se tem um conjunto de medidas até o k-ésimo passo e deseja-se obter uma estimativa do vetor de estado no passo $k + 1$. Quando finalmente se obtém a medida no passo $k + 1$, pode-se usá-la para corrigir a predição, processo chamado de filtragem ou correção. O algoritmo usado para o rastreamento visual consiste de dois conjuntos de equações, as preditoras e as corretoras. As equações preditoras, são responsáveis pelo avanço temporal do filtro. A predição é usada para centrar a janela de busca no próximo quadro. As equações corretoras são responsáveis pela realimentação da nova medida corrigindo a estimativa feita a priori. Esses dois conjuntos de equações são usados recursivamente.

Muito embora o filtro de Kalman tenha várias características favoráveis ao seu uso, ele também possui suas limitações. Uma delas, já mencionada, é quando os ruídos não podem ser modelados como gaussianos, e o filtro de partículas torna-se uma boa opção. Outra premissa central do filtro de Kalman é que os parâmetros do modelo sejam precisamente conhecidos, quando essa premissa é violada o desempenho do filtro se deteriora e surge a necessidade da substituição por um filtro robusto. Existem vários textos introdutórios a respeito do filtro de Kalman, um deles pode ser encontrado em [22].

3.2 Origens do filtro de Kalman

Pode-se definir $\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}}$ como sendo a estimativa de estado a priori no passo $k+1$ tendo-se o conhecimento do processo até o passo k, e $\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k+1}}$ como sendo a estimativa de estado a posteriori dada agora o conhecimento da medida $\boldsymbol{z}_{\boldsymbol{k+1}}$. Desta forma, também pode-se definir os erros a priori e a posteriori como

$$\boldsymbol{e}_{\boldsymbol{k+1|k}} = \boldsymbol{x}_{\boldsymbol{k+1}} - \hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}}$$
$$\boldsymbol{e}_{\boldsymbol{k+1|k+1}} = \boldsymbol{x}_{\boldsymbol{k+1}} - \hat{\boldsymbol{x}}_{\boldsymbol{k+1|k+1}}$$

As matrizes covariância dos erros são respectivamente

$$\boldsymbol{P}_{\boldsymbol{k+1|k}} = E\left[\boldsymbol{e}_{\boldsymbol{k+1|k}}\boldsymbol{e}_{\boldsymbol{k+1|k}}{}^{\boldsymbol{T}}\right]$$
$$\boldsymbol{P}_{\boldsymbol{k+1|k+1}} = E\left[\boldsymbol{e}_{\boldsymbol{k+1|k+1}}\boldsymbol{e}_{\boldsymbol{k+1|k+1}}{}^{\boldsymbol{T}}\right]$$

As equações do filtro de Kalman surgem da tentativa de encontrar uma relação para calcular a estimativa de estado a posteriori $\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k+1}}$ como uma combinação linear da estimativa a priori $\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}}$ e um fator de correção dado pela diferença ponderada entre a medida atual $\boldsymbol{z}_{\boldsymbol{k+1}}$ e o valor esperado desta, $\boldsymbol{H}\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}}$. A explicação para isso pode ser encontrada nas teorias estatísticas que estão por trás do filtro de Kalman e nas hipóteses adotadas. Esta primeira equação pode ser vista abaixo

$$\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k+1}} = \hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}} + K\left[\boldsymbol{z}_{\boldsymbol{k+1}} - H\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}}\right]$$

A discrepância entre o valor esperado para a medida, $\boldsymbol{H}\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}}$, e o valor realmente medido, $\boldsymbol{z}_{\boldsymbol{k+1}}$, é chamada de resíduo. Se o resíduo for pequeno, tendendo a zero, os dois valores estão em concordância e a estimativa corrigida é próxima da estimativa a priori.

A matriz K de ponderação é conhecida como ganho de Kalman e é calculada de tal forma que minimize a covariância do erro a posteriori. Uma das soluções para esse problema de minimização é dada por

$$K_k = P_{k+1|k}H^T\left(HP_{K+1|K}H^T + R\right)^{-1}$$

As equações, sob a óptica estatística, são derivadas a partir do conhecimento das funções distribuição de probabilidades condicionadas e o uso da regra de Bayes. De modo geral os estimadores de mínimos quadrados podem ser funções não lineares complicadas. As hipóteses de ruídos gaussianos garantem que a distribuição de probabilidade do estado será normal e consequentemente o problema se reduz a estimativas lineares. Desta forma é suficiente manter os dois primeiros momentos da distribuição do estado para uma caracterização completa

$$E[\boldsymbol{x_{k+1}}] = \hat{\boldsymbol{x}}_{\boldsymbol{k+1|k+1}}$$
$$E\left[\left(\boldsymbol{x_{k+1}} - \hat{\boldsymbol{x}}_{\boldsymbol{k+1|k+1}}\right)\left(\boldsymbol{x_{k+1}} - \hat{\boldsymbol{x}}_{\boldsymbol{k+1|k+1}}\right)^{\boldsymbol{T}}\right] = \boldsymbol{P_{k+1|k+1}}$$

Maiores detalhes sobre as origens probabilísticas do filtro podem ser encontradas em, [16], [18], [19] e [20].

3.3 Equações do filtro nominal e ajustes paramétricos

As equações do filtro surgem da minimização da matriz covariância do erro a posteriori e por isso o filtro é chamado de ótimo. Além disso, há soluções não recursivas e recursivas para este problema. As soluções não recursivas perdem mérito conforme o número de medidas cresce. Kalman foi o primeiro a propor uma solução recursiva para este problema, utilizando dois conjuntos de equações e realimentação

Equações de Predição:

$$\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}} = F_k \hat{\boldsymbol{x}}_{\boldsymbol{k|k}} \quad (11)$$
$$P_{k+1|k} = F_k P_{k|k} F_k^T + G_k Q_k G_k^T \quad (12)$$

Equações de Correção:

$$K_{k+1|k} = P_{k+1|k} H_{k+1}^T \left[H_{k+1} P_{k+1|k} H_{k+1}^T + R_{k+1}\right]^{-1} \quad (13)$$
$$\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k+1}} = \hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}} + K_{k+1}\left[\boldsymbol{z_{k+1}} - H_k \hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}}\right] \quad (14)$$
$$P_{k+1|k+1} = [I - K_{k+1} H_{k+1}] P_{k+1|k} \quad (15)$$

Um grande desafio na implementação do filtro é o ajuste dos parâmetros Q e R. Q está relacionado com o processo a ser estimado e R está relacionado com o processo de medição. Há várias

maneiras de ajustar estes parâmetros, seja com testes on-line ou outros filtros conhecidos como identificadores de sistemas. Muito embora estas opções estejam disponíveis, optou-se neste projeto pelo ajuste empírico destes valores. Desta maneira serão escolhidos valores para Q e R que garantam um bom desempenho do filtro.

4 Visão computacional

Os algoritmos de filtragem exploram as questões dinâmicas do sistema, como posição, velocidade e aceleração. Estas características dinâmicas são utilizadas para auxiliar na busca do alvo em um quadro de vídeo. Porém, a busca do alvo, propriamente dita, é realizada por algoritmos de visão computacional. Estes algoritmos são responsáveis pela extração das características estáticas do alvo nas imagens, por exemplo, posição de um ponto do alvo, área do alvo, textura, etc. Os princípios e as técnicas para o processamento de imagens serão abordados nesta seção, com grande ênfase na área de segmentação de imagens. Quando se trabalha com os algoritmos de rastreamento visual é muito importante o conhecimento do ambiente e dos objetos que estão envolvidos. O objetivo é desenvolver um conjunto de técnicas de segmentação de imagens e localização de alvos que posteriormente ficarão à disponibilidade para serem utilizadas conforme a situação. Uma excelente referência para o conteúdo de processamento de imagens é [17].

4.1 Imagens digitais

A aquisição de imagens se dá através de câmeras ou sensores específicos conectados a computadores. O modelo matemático de uma imagem pode ser implementado de várias maneiras: função de imagem, modelo geométrico, modelo radiométrico, modelo de cor, modelo de frequência espacial e imagens digitais. O interesse aqui se encontra nas imagens digitais, que são representadas por funções de duas variáveis, onde $f(x, y)$ representa a intensidade do nível de cinza da imagem na coordenada (x, y). Apesar de a intensidade luminosa ser uma variável contínua, nas imagens digitais essa intensidade está representada por um certo número de bits, ou seja, está quantizada. Essa quantização é conhecida como a resolução da imagem em relação ao nível de cinza. Os intervalos de amostragem usados pelas câmeras ou sensores definem o que se chama resolução espacial da imagem.

Esse par de coordenadas e sua respectiva intensidade luminosa são organizados em uma matriz. Uma imagem em preto e branco, ou em escala de cinza, fica completamente representada por essa matriz. Imagens coloridas, por outro lado, necessitam normalmente de três matrizes para estarem modeladas. Há diversos sistemas de codificação de cores, ou mais formalmente, espaços de cores. RGB e HSV são os mais comuns, mas existem outros como HSI, YCbCr, etc.

O sistema de cores RGB, adotado pela CIE (em francês, Commission Internationale de l'Eclairage), em 1931, consiste na utilização de um espaço tridimensional onde as cores são obtidas a partir das cores primárias vermelha, verde e azul, respectivamente baixa, média e alta frequências do espectro visível pelo olho humano. É muito usado em monitores coloridos, câmeras e é um dos

modelos de cores mais importantes no processamento de imagens, onde cada cor primária é representada por uma matriz.

HSV, ou Hue, Saturation, Value é um espaço de cores não linear muito comum e análogo ao modo que as pessoas usam para selecionar cores. Um exemplo de aplicação deste sistema são as paletas de cores, como a do *Paint Brush*. O Hue(H) indica a matiz (tipo da cor), o Value(V) especifica a quantidade de luz, e a Saturation(S) indica quanto de luz branca esta misturado na cor pura. É muito usado em tratamento de imagens, pois através da matriz H consegue-se identificar a cor de um objeto sem grande influência da iluminação e intensidade desta cor, que fica a cargo dos outros parâmetros. Isso significa que um verde claro e um verde escuro podem ter o mesmo Hue.

Após a aquisição da imagem, esta pode passar por etapas de pré-processamento. Nestas etapas as imagens são trabalhadas de forma a serem utilizadas nos algoritmos de segmentação. Algumas tarefas comuns nesta etapa são o redimensionamento e o realce das imagens adquiridas. Esta etapa de pré-processamento, apesar de muitas vezes necessária, não será abordada neste projeto, pois o maior interesse são os algoritmos de segmentação.

4.2 Segmentação de imagens

A segmentação de imagens está na transição entre processamentos de baixo nível e alto nível. Os processamentos de baixo nível são aqueles que recebem como entradas imagens e retornam na saída imagens. Os processamentos de alto nível são aqueles que recebem imagens e retornam atributos extraídos destas imagens. A segmentação é um processo intermediário; apesar de receber e retornar apenas imagens, as imagens segmentadas são fundamentais para a extração de características.

Podemos definir a segmentação como um processo que subdivide uma imagem em suas regiões constituintes. O nível de detalhe em que a subdivisão é realizada depende do problema a ser resolvido, ou seja, a segmentação estará concluída quando os objetos de interesse forem detectados. É o processo mais importante na visão computacional, e também o mais complicado. Quando se trabalha com a busca autônoma de alvos, nem sempre se tem conhecimento sobre o ambiente envolvido e isso leva a duas consequências principais: Primeiro, deve-se possuir uma gama de algoritmos a disposição, para que possam ser aplicados nas mais diversas situações. Segundo, a abordagem deve ser centrada na seleção de características que realcem os objetos de interesse, diminuindo a ineficiência do ambiente e outros objetos irrelevantes. A confecção de um algoritmo robusto o suficiente para ser aplicado à vários ambientes e objetos é uma busca contínua de quem estuda este tema.

De maneira geral, os algoritmos de segmentação baseiam-se em duas propriedades das imagens: descontinuidade e similaridade. A primeira classe de algoritmos subdivide uma imagem com base em mudanças bruscas das propriedades ao longo da imagem, ou seja, detectam variações abruptas na intensidade de pixels adjacentes. Exemplos de algoritmos pertencentes a essa classe são os detectores de borda. A segunda classe, por sua vez, subdivide uma imagem em regiões que sejam semelhantes segundo algum critério, ou seja, os pixels em uma determinada região devem ter certas propriedades em comum. Exemplos desta abordagem são os algoritmos de limiarização e algoritmos de segmentação em RGB.

4.2.1 Limiarização

A técnica de segmentação de imagens mais simples é a limiarização, ou tresholding.
Ambos os termos são usados neste trabalho. Possui fácil implementação e alta velocidade computacional. A limiarização consiste em nada mais do que percorrer os pixels de uma imagem classificando-os, segundo algum critério, em diferentes categorias.

O caso mais comum é realizar um teste que separe os pixels pertencentes ao objeto de interesse, ou *foreground*, dos pixels pertencentes ao plano de fundo da imagem, ou background. Matematicamente, um exemplo disso seria

$$g(x,y) = \begin{cases} 1 \; se \; f(x,y) \in T \\ 0 \; se \; f(x,y) \in T \end{cases}$$

sendo que (x,y) é um ponto da imagem, $f(x,y)$ é a intensidade deste ponto, T é uma faixa de valores de intensidade e $g(x,y)$ é a função que descreve a imagem segmentada.

Na verdade, quaisquer valores de intensidade podem ser escolhidos para a imagem segmentada, porém a vantagem de se escolher “1” (objeto) e “0” (plano de fundo) é que o resultado passa a ser uma imagem binária.

Há vários tipos de limiarização; se T é uma constante aplicável a toda imagem o processo é conhecido como limiarização global, o contrário seria a limiarização variável. Se T for uma função de (x,y) então a segmentação é dita dinâmica ou adaptativa. Existem as limiarizações múltiplas, que dividem as imagens em mais de duas classes. Entretanto, sendo a grande vantagem da limiarização sua simplicidade, normalmente não se usam mais do que duas classes.

Um tipo muito comum de limiarização para imagens coloridas é a segmentação em HSV. Como a limiarização envolve normalmente apenas uma matriz enquanto imagens coloridas possuem três matrizes, uma matriz deve ser selecionada e as outras não serão utilizadas. Isso é inviável para imagens RGB porque não se pode simplesmente descartar duas cores e escolher uma. A maneira de solucionar esse problema é trabalhar com imagens HSV, pois apresentam a matriz H que representa a cor sem levar em consideração a sua tonalidade ou brilho. Desta forma a limiarização em HSV é utilizada quando um objeto possui uma cor que se destaca na imagem.

O sucesso ou fracasso da limiarização é determinado por três fatores principais: O primeiro fator é que o objeto deve se destacar na cena, ou seja, a intensidade dos pixels contidos no objeto está em uma faixa de valores que não é encontrada no restante da imagem. O segundo e terceiro fatores são a iluminação e refletância. A forma como a iluminação varia no ambiente, se é uniforme ou não, e a maneira como o objeto reflete a luz podem alterar significantemente a intensidade dos pixels na imagem. Como nem sempre é possível controlar estas variáveis, deve-se tomar cuidado na hora de optar por esse método.

A função escrita no Matlab para fazer a limiarização de imagens HSV está contida no Apêndice A.

A função recebe a imagem F e dois parâmetros, h_{min} e h_{max}, que determinam a faixa de valores T. A primeira linha da função isola a primeira matriz da imagem F, ou seja, a matriz Hue. Percorrem-se então todos os pixels desta matriz e para cada um verifica-se se sua intensidade está entre os parâmetros mínimo e máximo. No caso de verdadeiro, a intensidade do pixel recebe o valor *255*, ou seja, branco, indicando que pertence ao objeto. Caso contrário, a intensidade recebe o valor “0”, preto, e indica o plano de fundo. A seguir, tem-se uma imagem de uma pessoa segurando uma esfera laranja, considerada como objeto de interesse e espera-se isolar através da limiarização.

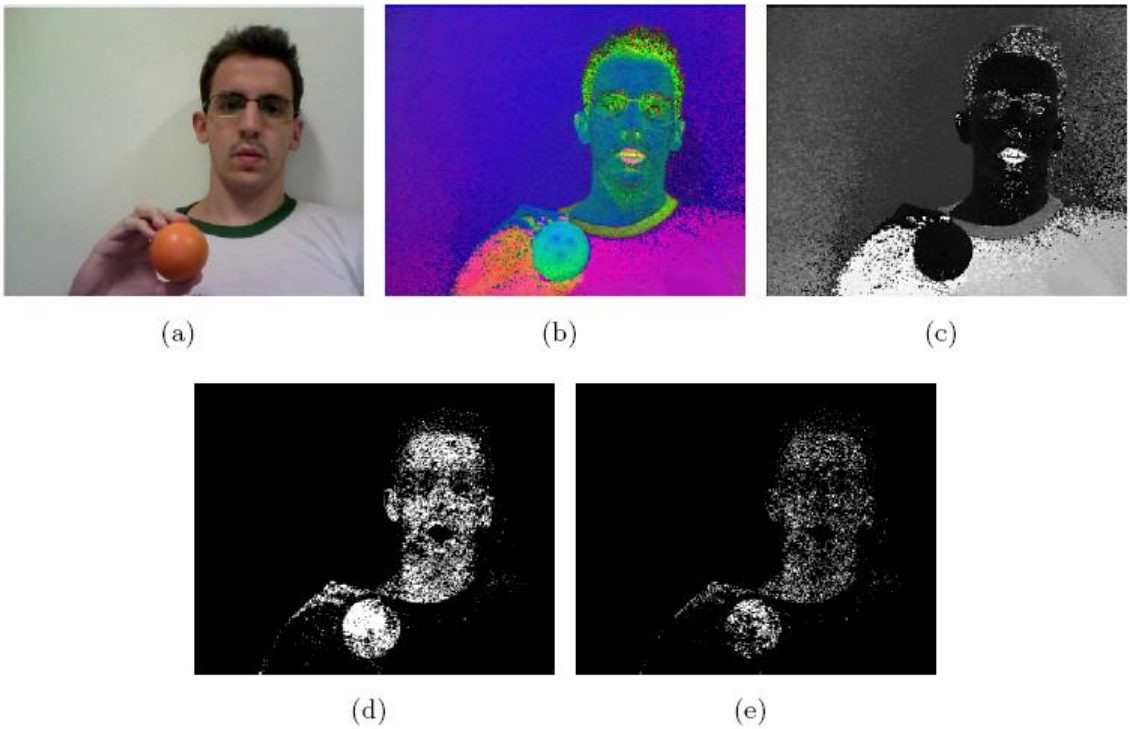


Figura 2 - (a) Imagem RGB (b) Imagem HSV (c) Matriz de Hue (d) Thresholding usando $\boldsymbol{h_{min}}$= 0.035 e $\boldsymbol{h_{max}}$= 0.06 (e) Thresholding usando $\boldsymbol{h_{min}}$= 0.05 e $\boldsymbol{h_{max}}$= 0.06

A primeira Figura *2-a* mostra a imagem capturada da câmera cujo formato é RGB. Após uma conversão usando a função pré-definida do Matlab, *rgb2hsv*, obtém-se a Figura *2-b*. A matriz de Hue isolada pode ser vista na Figura *2-c*. Usando-se a paleta de cores do "*Paint Brush*" ou o próprio visualizador de imagens do Matlab pode-se determinar a faixa de Hue correspondente a cor laranja. Aplicou-se a função duas vezes usando valores diferentes para h_{min} e h_{max}, e os resultados são mostrados na Figura *2-d e 2-e*, respectivamente. Note como a iluminação não é uniforme, gerando na esfera algumas regiões de maior brilho e outras de sombra. Essa variação de iluminação e refletância causa uma variação do Hue ao longo da esfera o que dificulta a limiarização. Além disso, a cor da pele humana, neste caso também possui um Hue parecido com a do objeto e interfere na segmentação. Diminuindo o intervalo T, diminui-se este efeito, porém o resultado perde qualidade. Em casos assim, deve-se procurar outro método de segmentação, como o que será descrito a seguir.

4.2.2 Segmentação de imagens em RGB

Após algum uso dos algoritmos de limiarização chega-se a conclusão de que apesar de ser um dos métodos mais simples e comum na segmentação de imagens, essa simplicidade implica em baixo poder de segmentação. Isso ocorre principalmente porque se usa apenas uma das matrizes da imagem HSV. As duas outras matrizes por não terem sido consideradas acabam acarretando uma perda de informação. Outra desvantagem é que a maioria das imagens é obtida em formato RGB, e a conversão

de RGB para HSV é computacionalmente pesada e envolve um tempo de execução que poderia ser um limitante para aplicações em tempo real.

Como alternativa, pode-se segmentar imagens diretamente no espaço RGB. O uso da segmentação por meio dos vetores de cores RGB produz excelentes resultados, com a desvantagem de ser mais complexa que a limiarização. Suponha que o intuito seja segmentar objetos de uma determinada cor em uma imagem. Coleta-se uma amostra representativa das cores de interesse e calcula-se como estimativa o valor médio e os desvios padrões deste conjunto. Dado o ponto RGB que representa essa média, o objetivo passa a ser classificar os pixels como pertencentes ou não a uma região próxima a esse ponto. Uma das medidas mais simples para definir quais pontos estão próximos, ou não, é a norma euclidiana. Os pontos que possuem uma distância, no espaço RGB,em relação a este ponto, inferior a um certo parâmetro D são considerados como pertencentes ao objeto, caso contrário não pertencem ao objeto. Resumindo, os pontos serão considerados como pertencentes ao objeto se estiverem contidos em uma esfera de raio D centrada no ponto representativo da cor do objeto. Devido ao alto custo computacional para calcular as distâncias euclidianas, que envolve quadrados e raízes, pode-se ao invés de usar uma esfera centrada no valor médio, usar um cubo. Determinar se um ponto está ou não no cubo é muito mais simples do que na esfera. Pode-se também usar uma elipsoide, cujos eixos principais são orientados na direção de máxima dispersão de dados, sendo esta uma das razões que podem levar a preferência por este método. É comum escolher valores para D que sejam múltiplos do maior desvio padrão encontrado. A imagem obtida é uma imagem binária com o objeto desejado destacado do fundo da imagem.

Denominando-se $\boldsymbol{a}$ o vetor RGB que expressa a cor média e $\boldsymbol{p}$ um ponto arbitrário no espaço RGB. Diz-se que $\boldsymbol{z}$ é similar a $\boldsymbol{a}$ se a distância entre eles for menor que um parâmetro D. Usando-se a distância euclidiana

$$d(\boldsymbol{z},\boldsymbol{a}) = \|\boldsymbol{z}-\boldsymbol{a}\|$$

$$= [(\boldsymbol{z}-\boldsymbol{a})^T(\boldsymbol{z}-\boldsymbol{a})]^{\frac{1}{2}}$$

$$= [(z_R - a_R)^2 + (z_G - a_G)^2 + (z_B - a_B)^2]^{\frac{1}{2}}$$

Pode-se generalizar a equação anterior para se obter a elipsoide

$$d(\boldsymbol{z},\boldsymbol{a}) = [(\boldsymbol{z}-\boldsymbol{a})^T C^{-1}(\boldsymbol{z}-\boldsymbol{a})]^{\frac{1}{2}}$$

na qual C é a matriz de covariância da amostra coletada. Esta distância é também conhecida como *distância Mahalanobis*.

No Apêndice A encontra-se um trecho do programa em Matlab que permite o usuário identificar o objeto a ser seguido e então se calcula o vetor RGB, correspondente ás cores médias, e também os desvios padrões.

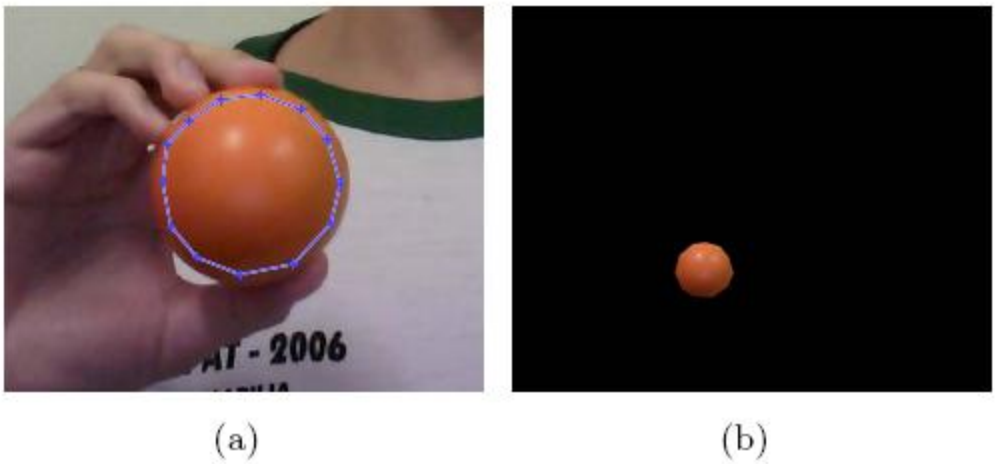


Figura 3 - (a) Seleção feita pelo usuário usando a função pré-definida roipoly (b) Mascara resultante

Faz-se a leitura de uma imagem qualquer que tenha o objeto de interesse e pede-se ao usuário para criar a máscara selecionando o objeto a ser seguido, ver Figura 3 (partes (a) e (b)). Depois separá-se o objeto em suas três componentes de cor e através de uma reordenação destas componentes gera-se a matriz I. A partir da matriz I é possível gerar a matriz de covariância, que resulta nos valores médios e desvios padrões procurados.

A função *colorseg* faz a segmentação usando os parâmetros adquiridos anteriormente. Além disso, essa função ainda permite o usuário escolher entre dois métodos de segmentação, usando *distâncias euclidianas* ou *distâncias de Mahalanobis*. A função *imstack2vectors*, mostrada na sequência, extrai vetores de uma pilha de imagens. Isto significa que para cada pixel da imagem, um vetor é criado contendo as três componentes de cor. A vantagem é que pode-se trabalhar, desta forma, com vetores médios e matrizes de covariância de vetores aleatórios. A explicação das partes do algoritmo pode ser encontrada nos comentários do código.

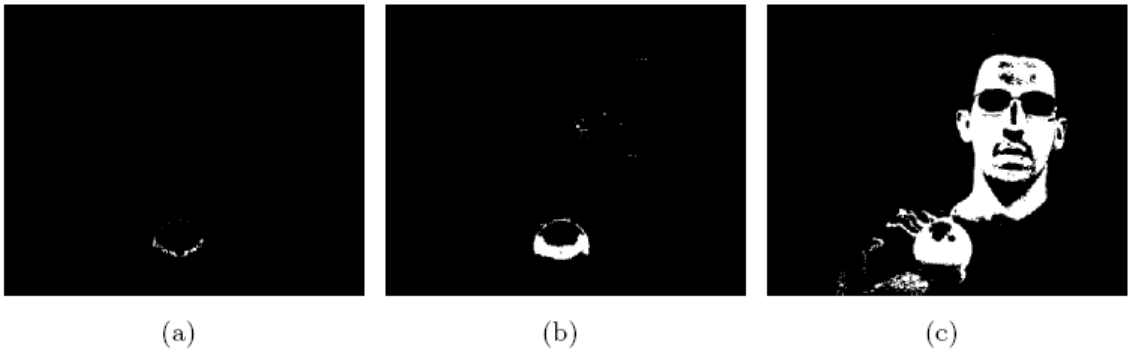


Figura 4 - (a) Segmentação RGB usando $\boldsymbol{D = 10}$ (b) $\boldsymbol{D = 32}$ (c) $\boldsymbol{D = 64}$

Como pode ser visto na figura *4-a* e *b*, o problema de iluminação ainda é o fator limitante da qualidade dos resultados obtidos. Com o valor de $D = 10$ o processo é bem restritivo e, embora não haja ruídos, poucos pixels são classificados como pertencentes ao objeto. Com o valor de *64*, a seletividade é baixa fazendo com que, além do objeto de interesse, sejam destacadas outras partes da cena. Utilizando um valor intermediário, *32*, tem-se o melhor resultado, ainda com presença de ruídos. Não é o resultado ideal, mas, devido às condições de iluminação, que não puderam ser controladas, é o melhor que se pode obter e é superior ao obtido pelo thresholding. A região segmentada pertencente à esfera é simétrica e está próxima do seu centro geométrico, assim, reduzindo-se os efeitos dos ruídos será possível obter uma boa estimativa de posição. Este resultado será mostrado adiante, após os processos de melhoria da segmentação e os algoritmos de extração de informações serem abordados.

4.2.3 Subtração do fundo

Os algoritmos anteriores são usados para selecionar objetos pela cor, mas nem sempre essa é a melhor maneira. Quando o fundo é estático, uma estratégia interessante para isolar o alvo é a subtração do fundo. Algoritmos que realizam esta subtração podem variar desde os mais simples aos mais complexos e elaborados.

A detecção de objetos pode ser feita construindo-se uma representação da cena chamada de modelo do plano de fundo, “background model". Para cada novo quadro adquirido, mede-se o desvio em relação a esse modelo. Define-se uma quantidade mínima para esse desvio ser significativo, e qualquer região da nova imagem cujo desvio ultrapasse esse valor será considerada pertencente ao objeto, caso contrário, pertencente ao plano de fundo. Esta técnica, embora simples, está sujeita a vários problemas como, mudanças na iluminação, falta de atualização do modelo de fundo e presença de ruídos. Uma melhoria significativa poderia ser obtida utilizando-se várias imagens na construção do modelo do fundo, calculando-se assim a média de um conjunto de quadros. Isso diminui o efeito do ruído. Existem, por outro lado, abordagens mais completas, que podem trabalhar com atualização do

modelo de fundo em tempo de execução. Algumas das inúmeras vantagens de se atualizar o modelo são que objetos que não estavam no modelo inicial, mas permanecem em repouso durante muito tempo podem ser incorporados ao modelo do plano de fundo e mudanças de iluminação são também incorporadas ao modelo. Existem ainda abordagens que decompõem a cor em suas componentes de cromaticidade e brilho, fazendo com que o resultado da segmentação seja menos suscetível a problemas de iluminação, veja [15].

De maneira geral, quando se trata de algoritmos de subtração de fundo, há três questões fundamentais: Como os objetos são distinguidos do plano de fundo? Como o plano de fundo é mantido ao longo do tempo? Como os objetos são pós-processados para rejeitar falsos positivos? Respostas para estas perguntas e uma variedade de algoritmos de subtração de fundo podem ser encontradas em [21].

4.3 Melhorias no processo de segmentação

Iluminação não uniforme, mudanças na posição e rotações dos alvos, além de outros objetos se movendo ao fundo com cores parecidas são problemas que dificultam e às vezes impossibilitam o uso dos algoritmos apresentados. Como já mostrado nas figuras *2-d*, *2-e* e *4-b* as imagens binárias resultantes desses algoritmos de segmentação possuem, além do objeto isolado, algum ruído. A primeira tentativa para reduzir este ruído seria diminuir o intervalo de limiarização ou o parâmetro *D* na segmentação em RGB. Isto a princípio reduziria de fato o ruído, mas como consequência haveria perda de qualidade nas imagens do alvo.

Existem, por outro lado, algumas operações morfológicas que podem ser aplicadas, antes ou depois da segmentação, para melhorar a qualidade dos resultados obtidos. Entre elas estão a suavização, erosão, algoritmos que trabalham com conceitos de conectividade e até mesmo o filtro de Kalman. O uso do filtro para redução da área de busca além de reduzir os custos computacionais, como já mencionado, reduz a interferência de outros objetos no resultado.

Se ao invés de considerar informações a respeito de cada pixel individualmente, os algoritmos trabalhassem com informações relacionas ao pixel e sua vizinhança, muito provavelmente isso acarretaria em uma melhor segmentação. A suavização é considerada um passo nesse sentido, pois modifica o valor de cada pixel com base nos pixels ao seu redor. O objetivo desta técnica é diminuir os efeitos espúrios nas imagens. A grande vantagem destes algoritmos é que reduzem os ruídos e removem descontinuidades e detalhes antes da extração dos objetos de interesse. Um exemplo desta classe de algoritmos é o filtro de média (ou filtro passa-baixa), que troca o valor de intensidade de um determinado pixel pela média de intensidades da vizinhança do pixel. Isso é feito através de uma

máscara, cujo tamanho deve ser escolhido cautelosamente para não causar borramento nas imagens. Matematicamente, temos

$$g(x,y) = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)f(x+s,y+t)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)}$$

sendo que $f(x,y)$ representa a imagem original e $g(x,y)$ a imagem suavizada. Os parâmetros a e b delimitam o tamanho da máscara e $w(s,t)$ representa os parâmetros da máscara.

Outro procedimento utilizado é a erosão. Trata-se de uma operação morfológica fundamental no processamento de imagens. É um processo que contrai ou estreita objetos em imagens binárias. A maneira e a extensão em que isso ocorre são determinadas por um molde conhecido como elemento estrutural, que na verdade nada mais é que uma matriz de "*0*"*s* e "*1*"*s*.

O processo de erosão consiste em transladar o elemento estrutural através da imagem e checar os pontos onde o elemento encaixa inteiramente na imagem. A imagem resultante terá valor "*1*" em todas as localizações da origem do elemento estrutural, na qual este sobrepõe somente pixels com valor "*1*". A erosão de A por B, é definida como

$$A \ominus B = \{z|(B)_z \subseteq A\}$$

ou,

$$A \ominus B = \{z|(B)_z \cap A^c = \emptyset\}$$

Nessas equações A representa a imagem original e B o elemento estrutural. A erosão de A por B é o conjunto de pontos z de tal maneira que B, transladado por z, está contido em A. Outra maneira equivalente é definir o conjunto de pontos em que B não tem elementos comuns com o fundo A^c, que indica o complemento de A.

Na prática, a erosão tende a eliminar os ruídos e pequenas manchas na imagem segmentada. Deve-se, contudo tomar cuidado com a dimensão do elemento estrutural, pois, além do ruído ele pode retirar partes do objeto de interesse.

```
se = streal ('diamond',lenght); %structural element
E = imerode (imagem,se);
```

Trecho 1- Erosão

O trecho de programa acima é usado na erosão. O Matlab possui estas duas funções pré-definidas no *"Image Processing Toolbox"*. A primeira função, *streal*, cria o elemento estrutural, *se*, recebendo como parâmetro a forma do elemento estrutural, no caso, diamante. O outro parâmetro, *lenght*, é passado para a função determinando o tamanho do elemento estrutural. A função *imerode*, que efetua a erosão, recebe a imagem original e o elemento estrutural. Essas funções foram aplicadas sob a figura *4-b* e os resultados são mostrados a seguir.

(a) (b) (c)

Figura 5 - (a) Resultado obtido na segmentação (b) Erosão utilizando elemento estrutural de tamanho **2**, eliminando o ruído sem alterar muito a segmentação (c) Tamanho **10**, além do ruído, grande parte da região de interesse foi erodida

4.4 Vizinhança e conectividade de um pixel

A suavização e erosão são processos que levam em consideração a vizinhança de um pixel. Porém, torna-se necessário usar uma abordagem mais elaborada para descrever a vizinhança de um pixel, um caminho digital ou uma região conectada.

Um pixel *p* na coordenada *(x, y)* tem ao seu redor oito vizinhos. Quatro destes estão dispostos na horizontal e vertical, além disso, dividem uma borda com o pixel *p*; são os pixels nas coordenadas *(x+1,y)*, *(x-1,y)*, *(x,y+1)*, *(x,y-1)*. Este conjunto de pixels é chamado de *vizinhança-4* de *p*, e é representado por $N_4(p)$. Outros quatros destes vizinhos estão dispostos nas diagonais e dividem apenas um vértice com o pixel *p*, são os pixels nas coordenadas *(x+1,y+1)*, *(x+1,y-1)*, *(x-1,y+1)*, *(x-1;y-1)* representados por $N_D(p)$. Os dois conjuntos unidos são chamados de *vizinhança-8* de *p*, $N_8(p)$. Caso o pixel esteja na borda da imagem, alguns destes vizinhos não existirão.

Tendo definido a vizinhança de um pixel, outro passo muito importante é definir adjacência. Para tal definição é necessário escolher um conjunto de valores de intensidade *V* que é usado para avaliar se um pixel é ou não adjacente a outro. Por exemplo, dois pixels p e q com valores pertencentes a *V* são *adjacentes-4* se q estiver no conjunto $N_4(p)$. Da mesma maneira, p e q com

valores pertencentes a *V* são *adjacentes-8* se q estiver no conjunto $N_8(p)$. Quando se usa a *adjacência-8* em uma imagem corre-se o risco de ocorrer ambiguidades, ou seja, dois pixels podem ser ligados por dois ou mais caminhos diferentes. Para evitar ambiguidades, defini-se a *adjacência-m* ou mista. Dois pixels p e q com valores em *V* são *adjacentes-m* se: q estiver em $N_4(p)$, ou q estiver em $N_D(p)$ e o conjunto $N_4(p) \cap N_4(q)$ não contiver nenhum pixel cujos valores pertencem a *V*. A partir de uma sequência de pixels adjacentes cria-se um caminho ou curva digital. O caminho digital tem um inicio e um fim que podem ser coincidentes no caso de um caminho fechado.

Finalmente, é possível definir o conceito de conectividade: Se *S* for um subconjunto de pixels em uma imagem, diz-se que dois pontos p e q são conexos em *S* se houver um caminho entre eles consistindo inteiramente de pixels em *S*. Este conceito pode ser aplicado nos processos de segmentação para eliminar ruídos. Existem algoritmos que percorrem a imagem segmentada (binária) localizando os componentes conectados, outro algoritmo cria uma matriz de rótulos para estes componentes. Desta forma é possível posteriormente ordená-los por tamanho, sendo normalmente o maior deles referente ao objeto de interesse e os outros são ruídos. No Toolbox de processamento de imagens do Matlab ainda é possível encontrar a função *regionprops*, que mede propriedades de regiões conectadas em uma imagem binária. Entre estas propriedades, está o centroide, área, perímetro, entre outras.

4.5 Extração de informações

Como o resultado dos processos de segmentação são imagens binárias, torna-se simples a extração de informações. A principal informação desejada é a posição do alvo, que pode ser dada tanto por um ponto específico pré-determinado ou por um ponto genérico do objeto. No primeiro caso, conhece-se o ponto que deve ser identificado no objeto, por exemplo, a ponta do dedo indicador de uma mão, e para isso deve-se conhecer alguma característica extra como a curvatura da ponta do dedo.

O segundo caso é mais abrangente, pois permite que qualquer ponto seja usado como referência para a localização do objeto, desde que isto seja feito em todos os quadros. Sendo assim, uma maneira prática para encontrar a posição do alvo em um determinado quadro é encontrar seu “centro de massa". Aqui, “centro de massa" se refere ao ponto obtido considerando-se a intensidade de cinza como equivalente a densidade de um objeto, que pode ser calculado utilizando os primeiros momentos invariantes da imagem.

O momento *2-D* de ordem $(p+q)$ de uma imagem digital $f(x,y)$ de tamanho MxN pode ser definido como

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x,y) \qquad (16)$$

em que p e q são inteiros. Assim é possível definir o “centro de massa" como o ponto de coordenadas $(\bar{x}, \bar{y})$ em que

$$\bar{x} = \frac{m_{10}}{m_{00}}$$

e

$$\bar{y} = \frac{m_{01}}{m_{00}}$$

sendo m_{00} a área da região considerada. Segue, no Apêndice A, a função escrita no Matlab para realizar esta tarefa.

Este método pode ser usado com tranquilidade, pois verificou-se que os valores absolutos dos sete primeiros momentos são invariantes em relação a translação, escala, espelhamento (apenas inversão de sinal) e rotação. O cálculo da área do objeto também é importante quando se usa o método de rotular e classificar regiões conectadas em uma imagem.

A seguir mostra-se o resultado da posição obtida por meio da função posição, usando a imagem erodida *5-b*.

Figura 6 - Centro de massa da esfera usando segmentação RGB seguida de erosão e calculada usando os momentos invariantes. A posição indicada é $\boldsymbol{x = 313}$ e $\boldsymbol{y = 434}$

Como foi dito na seção *5.2.2*, embora o resultado da segmentação não tenha sido o desejado, ainda assim obteve-se uma boa estimativa do centro da esfera.

5 Resultados parciais

Tendo como base os conceitos estudados até aqui, uma primeira simulação utilizando o filtro de Kalman nominal foi realizada. Nesta simulação o alvo é a esfera de cor laranja que será movida por uma pessoa. Obviamente não é um caso de fundo estático, pois o braço da pessoa se move junto com a esfera, porém a cor da esfera se destaca do resto da imagem e pode-se assim usar o algoritmo de limiarização ou a segmentação em RGB. Os algoritmos de limiarização se mostraram poucos eficientes, enquanto o uso dos algoritmos de segmentação RGB produziram melhores resultados. O vídeo utilizado possui 50 quadros. O código do programa que faz o rastreamento visual do alvo ao longo de sua trajetória está transcrito no Apêndice A.

A primeira parte do código apenas inicializa as matrizes do sistema, aloca espaço para os vetores de estado a serem armazenados, e inicializa a matriz P. A função *extrairalvo*, que também está no Apêndice A, é usada para determinar o centroide do alvo e também um raio médio que serve apenas como referência na hora de determinar o tamanho da janela de redução de busca. As informações obtidas no primeiro quadro são utilizadas para inicializar o Filtro de Kalman e a partir daí começam as iterações até que todos os quadros sejam percorridos.

A posição obtida em um determinado quadro é usada para estimar a posição futura, que é usada para centrar a janela de busca. Após a leitura da posição do próximo quadro, a estimativa é corrigida e usada para gerar a próxima estimativa. A função *tic* e *toc* é usada para determinar o tempo de execução do algoritmo para avaliar o desempenho do Filtro, conforme Tabela *1*. Além disso, os vetores de posição $z(i)$ e $y(i)$, o vetor de estados $x(i,:)$ e o vetor $prev(:,i)$ são armazenados para a confecção dos gráficos e análise de desempenho.

A seguir são mostradas algumas figuras indicando o rastreamento visual do alvo pelo programa. A primeira sequência de imagens dá a noção de trajetória e ainda mostra o resultado da filtragem.

Figura 7 - Sequência de resultados obtidos mostrando a posição da esfera e a trajetória percorrida até o momento. Podem ser vistos mais detalhes na Figura abaixo. O número do quadro está indicado em cada figura

As figuras *8* e *9* mostram um detalhamento maior da trajetória, mostrando um aspecto interessante do filtro. O desempenho do filtro está diretamente relacionado com o movimento do alvo. Sendo assim, se a trajetória for suave e os parâmetros não tiverem saltos de um quadro para o outro, o desempenho é alto; caso contrário, o desempenho do filtro decresce à medida que os parâmetros se alteram com maior rapidez. Esse resultado justifica em parte o uso do filtro para o rastreamento visual de trajetória e também demonstra a importância de considerar para a próxima etapa o projeto do filtro robusto.

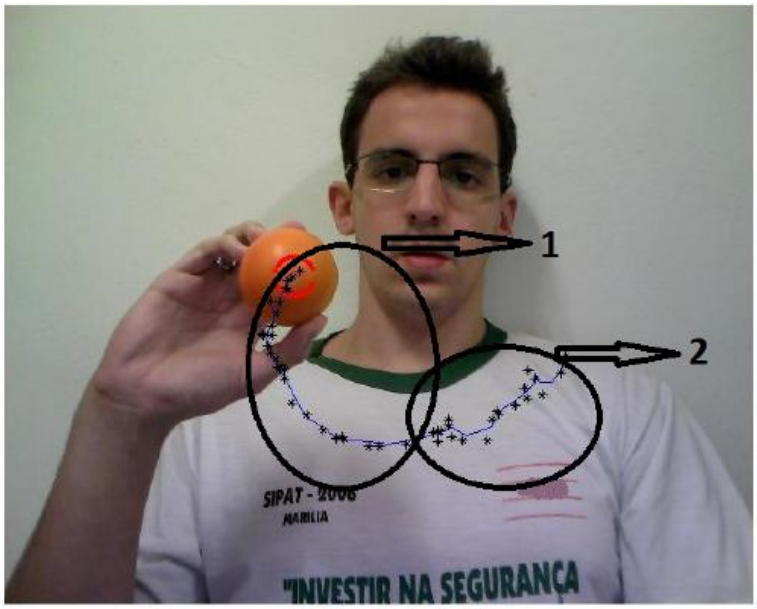


Figura 8 - Trajetória: A linha em azul marca a trajetória segundo o algoritmo de visão computacional e os pontos mostram a trajetória prevista pelo Filtro

Nesta figura podem ser vistas duas regiões distintas da trajetória. A região 1 corresponde a uma trajetória suave e assim o filtro consegue acompanhar o alvo com pouco desvio. Na região 2 há mudanças bruscas na trajetória e nestas mudanças há um maior desvio entre estimativa e valor real.

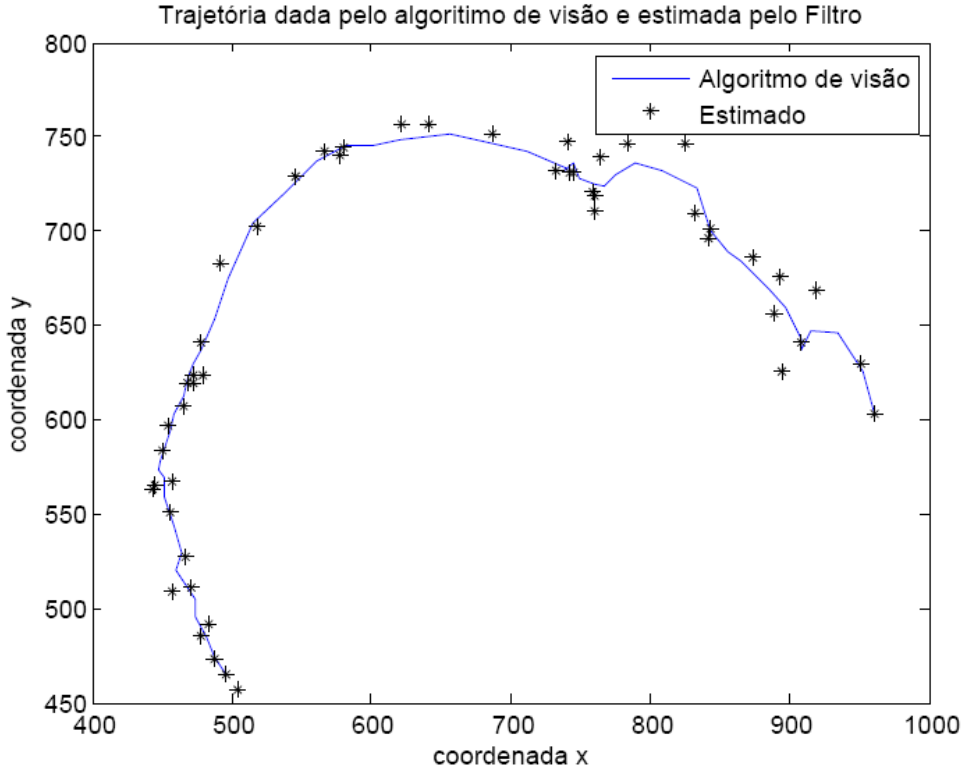


Figura 9 - Gráfico mostrando a posição resultante do algoritmo de visão e a posição estimada pelo filtro de Kalman

A figura *10* mostra duas circunferências, uma verde, indicando a posição dada pelo algoritmo de visão computacional, e uma vermelha, indicando a posição predita pelo filtro.

Figura 10 - Circunferências mostrando a posição estimada, vermelha e a posição calculada pelo algoritmo de visão, verde

Para uma justificativa completa para o uso de filtragens em rastreamento visual de trajetórias, a tabela a seguir mostra a redução do tempo de execução do programa conforme a janela de busca se reduz. O tempo de execução mostrado na tabela, não condiz com o tempo que será encontrado na prática, pois aqui usa-se o Matlab como plataforma de execução, além disso os dados são salvos durante a execução para posterior análise. Contudo, como todos os casos foram executados sob as mesmas condições, pode se dizer que o tempo relativo mostra a vantagem do uso de estimativas para redução da área de busca.

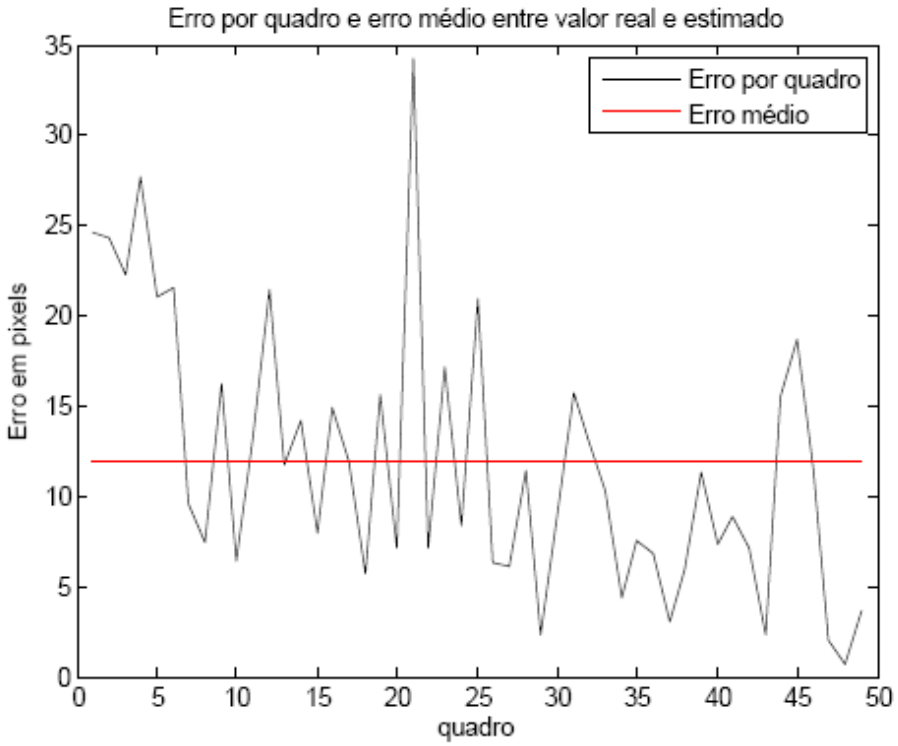
Tabela 1 - Tempo de execução com a redução da área da janela de busca do alvo

Tamanho da janela de busca do alvo	Tempo	Tempo relativo
Toda imagem (sem redução da área de busca)	117.50s	100 %
96 pixels x 96 pixels	57.88s	49.3%
77 pixels x 77 pixels	54.76s	46.6%
58 pixels x 58 pixels	50.79s	43.2%

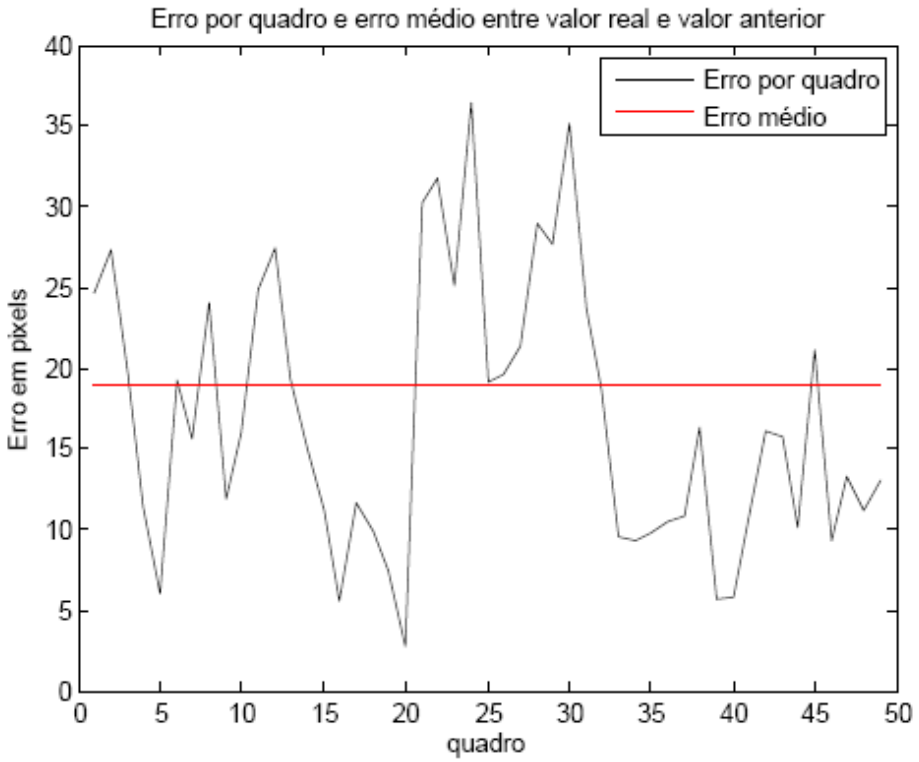
Finalmente, mesmo com todos esses resultados, alguém poderia se perguntar o seguinte: E se fosse usada a própria posição do alvo no quadro atual para centrar a janela de busca no próximo quadro, isso poderia funcionar? A resposta é: sim, poderia funcionar. Mas a ideia deste projeto é

justamente mostrar a vantagem no desempenho utilizando o filtro. Percebe-se através dos gráficos mostrados na figura *11* que o desempenho do filtro de Kalman nominal é maior do que quando se usa simplesmente a posição anterior como estimativa. O erro médio com o uso do filtro diminui aproximadamente *30%*. Isso implica em duas coisas: Primeiro, como o erro é menor com uso do filtro de Kalman, pode-se ter uma maior redução da janela de busca e segundo, existem situações em que usando simplesmente a posição como estimativa não será possível fazer o rastreamento visual. Da mesma maneira, existirão situações em que o filtro nominal não conseguirá realizar o rastreamento visual e então será usado o filtro robusto.

O erro no gráfico da figura *11-a* foi levantado usando a distância no plano entre a posição calculada pelo algoritmo de visão e a posição estimada pelo filtro a priori, ou seja, sem correção. No caso sem o filtro, figura *11-b*, foi calculada simplesmente a distância entre as posições em dois quadros consecutivos. Vale ressaltar ainda que os erros envolvidos estão todos abaixo de *20%* se comparados ao tamanho da esfera.



(a)



(b)

Figura 11 - Erros por iteração e erro médio, com e sem o Filtro de Kalman**.**

6 Interpretação dos resultados pré-liminares

Até este ponto o projeto foi uma oportunidade de familiarização com as técnicas de processamento de imagens e visão computacional. O estudo do Filtro de Kalman consolidou os conteúdos de sistemas dinâmicos, aprendidos nas disciplinas de modelagem e controle, além de propiciar uma compreensão dos conceitos de estimativas e método dos mínimos quadrados.

Foram obtidos bons resultados, que validam a proposta inicial do projeto e espera-se que com o desenvolvimento do filtro robusto seja possível realizar comparações de desempenho com o filtro nominal. A maior dificuldade até o momento se encontra nos algoritmos de segmentação, que são suscetíveis a mudanças de iluminação, outros objetos que se movem ao fundo do vídeo, variações do ângulo relativo entre o alvo e a câmera e mudanças bruscas no movimento do alvo.

Nesta primeira etapa da pesquisa foi realizado o rastreamento visual de trajetória de alvos móveis utilizando o filtro nominal de Kalman. Porém, como já mencionado, na presença de incertezas há uma deterioração das estimativas do filtro de Kalman. Neste caso, estimativas baseadas em filtragem robusta limitam a degradação do desempenho dessas estimativas, veja [5] e [30]. O objetivo da segunda etapa desta pesquisa é realizar o rastreamento visual de trajetórias com base em filtragem robusta. Dois modelos de filtros serão abordados e implementados. Com os resultados será possível avaliar cada um deles em relação ao filtro de Kalman.

7 Rastreamento visual de trajetórias utilizando filtro robusto BDU

As estimativas obtidas por meio do filtro de Kalman proposto na primeira etapa deste projeto foram descritas com base nos modelos estatísticos. Estas estimativas podem ser obtidas a partir de uma conveniente utilização determinística do método dos mínimos quadrados. Isso significa que a estimativa para $\boldsymbol{x_{i+1}}$ pode ser obtida a partir da estimativa anterior $\hat{\boldsymbol{x}}_{\boldsymbol{i|i}}$ e do conhecimento da nova medida $\boldsymbol{z_{i+1}}$, através da minimização da seguinte função custo, [25]

$$J_i = \min_{x_i, x_{i+1}} \left[\left\| x_i - \hat{x}_{i|i} \right\|^2_{P^{-1}_{i|i}} + \| x_{i+1} - F_i x_i \|^2_{Q_i^{-1}} + \| z_{i+1} - H_{i+1} x_{i+1} \|^2_{R^{-1}_{i+1}} \right] \quad (17)$$

A solução deste problema, [16], pode ser encontrada através da técnica dos mínimos quadrados regularizados. Segue que

$$\min_{\boldsymbol{x}_i} \left[\boldsymbol{x}_i^T Q \boldsymbol{x}_i + (A\boldsymbol{x} - \boldsymbol{b})^T W (A\boldsymbol{x} - \boldsymbol{b}) \right] \qquad (18)$$

tem como solução

$$\hat{\boldsymbol{x}} = [Q + A^T W A]^{-1} A^T W \boldsymbol{b} \qquad (19)$$

Escolhendo-se adequadamente as identificações dos parâmetros desta solução com os parâmetros do sistema e através de alguns recursos algébricos podem-se obter as equações do filtro de Kalman.

Na prática, porém, os termos nominais A, b estão sujeitos a distúrbios ou incertezas. Essas incertezas podem degradar as estimativas, ou seja, um estimador projetado com base em A pode não produzir boas estimativas quando na realidade está sujeito a $(A + \delta A)$. Esta motivação leva ao projeto de um novo filtro que considere estas incertezas.

O primeiro passo é encontrar uma nova função custo que incorpore estas incertezas. Segundo, [30], esse novo funcional poderia ser

$$J(x, y) = \boldsymbol{x}^T Q \boldsymbol{x} + R(x, y) \qquad (20)$$

onde o resíduo $R(x, y)$ é definido por

$$R(x, y) = (A\boldsymbol{x} - b + H\boldsymbol{y})^T W (A\boldsymbol{x} - b + H\boldsymbol{y}) \qquad (21)$$

Neste caso, o termo adicional $H\boldsymbol{y}$ representa as possíveis fontes de incertezas do modelo. A vantagem desta abordagem é que a matriz H permite limitar $\boldsymbol{y}$ a um certo espaço imagem. Porém, a incerteza $\boldsymbol{y}$ não é conhecida. O que se pode fazer neste caso, é assumir que se conhece um limite para esta incerteza, ou seja, $\|\boldsymbol{y}\| \leq \emptyset(x)$. O novo problema passa a ser

$$\hat{\boldsymbol{x}} = \arg \min_{x} \max_{\|\boldsymbol{y}\| \leq \emptyset(x)} J(x, y) \qquad (22)$$

Este problema é como um jogo em que um dos adversários tenta minimizar o funcional de custo e o outro tenta maximizar as incertezas e, portanto maximizar o funcional. Essa questão pode ser resolvida a partir do momento em que se limita o funcional através de $\phi(x)$. Com uma escolha apropriada de $\phi(x)$ pode-se chegar a seguinte especialização do problema

$$\min_{x} \max_{\delta A, \delta \boldsymbol{b}} \left[\|\boldsymbol{x}\|_Q^2 + \|(A + \delta A)\boldsymbol{x} - (\boldsymbol{b} + \delta \boldsymbol{b})\|_W^2 \right] \qquad (23)$$

onde δA e δb representam as perturbações do sistema de forma que

$$[\delta A, \delta b] = H\Delta[N_a N_b] \qquad (24)$$

Os parâmetros H, N_a, N_b aparecem com frequência nos problemas de filtragem robusta e surgem das especificações de tolerância dos parâmetros físicos do sistema. Métodos para identificar estes parâmetros é um assunto que poderia ocupar vários trabalhos. Vamos considerar nesta pesquisa um método baseado em algoritmos genéticos, que será mostrado mais adiante.

É um resultado conhecido que o seguinte estimador de $\boldsymbol{x}$ é a solução para a equação *23*

$$\hat{x} = \left[\hat{Q} + A^T \widehat{W} A\right]^{-1} \left[A^T \widehat{W} b + \hat{\lambda} N_a^T N_b\right] \qquad (25)$$

Quando comparada com a solução nominal, notam-se duas diferenças básicas:

- As matrizes de ponderação Q, W são trocadas pelas versões corrigidas $\hat{Q}$, $\widehat{W}$.

- No lado direito da equação há um termo adicional $\hat{\lambda} N_a^T N_b$.

Tanto os parâmetros corrigidos quanto o termo adicional dependem da variável $\hat{\lambda}$ que surge da minimização da função G(λ), veja para maiores detalhes [30].

A primeira vista pode-se pensar em resolver o problema de minimização da função $G(\lambda)$ a cada iteração, o que dificultaria a aplicação em tempo real. Porém uma solução subótima pode ser obtida com a seguinte expressão

$$\hat{\lambda} = (1 + \alpha)\lambda_l \ \ para\, \alpha > 0 \qquad (26)$$

$$\lambda_l = \|H^T W H\| \qquad (27)$$

O filtro robusto sob esta abordagem mantém a forma do filtro nominal de Kalman, com a diferença que agora serão usados os parâmetros corrigidos ao invés dos parâmetros nominais.

Considere para tanto, o sistema com incertezas

$$\boldsymbol{x}_{i+1} = (F_i + \delta F_i)\boldsymbol{x}_i + (G_i + \delta G_i)\boldsymbol{w}_i \qquad (28)$$

$$\boldsymbol{y_i} = H_i \boldsymbol{x_i} + \boldsymbol{v_i} \qquad (29)$$

onde as perturbações são modeladas como em *24*

$$[\delta F_i, \delta G_i] = M_i \Delta_{\mathrm{i}} [N_{f,i} N_{g,i}] \qquad (30)$$

e $\boldsymbol{w}_i$ e $\boldsymbol{v}_i$ são ruídos Gaussianos não correlacionados com média zero e variâncias Q_k e R_k, respectivamente. Assumindo que se possui uma estimativa inicial $\hat{\boldsymbol{x}}_{i|i}$ pode-se atualizar a estimativa resolvendo-se

$$\min_{\{x_i, w_i\}} \max_{\delta F_i, \delta G_i} \left[\left\| \boldsymbol{x}_i - \boldsymbol{x}'_{i|i} \right\|^2_{P^{-1}_{i|i}} + \|\boldsymbol{w}_i\|^2_{Q_i^{-1}} + \|\boldsymbol{y}_{i+1} - H_{i+1}\boldsymbol{x}_{i+1}\|^2_{R^{-1}_{i+1}} \right] \qquad (31)$$

Através de alguns recursos pode-se mostrar que este é um problema de mínimos quadrados regularizados e com algumas manipulações consegue-se gerar o conjunto de equações que compõe o filtro robusto, veja [30].

As equações do filtro robusto foram usadas no lugar das equações do filtro nominal e implementadas no Matlab para o rastreamento visual feito na primeira etapa da pesquisa. Esta implementação pode ser visualizada no Apêndice A.

Para avaliar a melhora no desempenho do filtro robusto em relação ao filtro nominal, define-se o seguinte método de avaliação

$$MSE_x = \sum_{i=1}^{I} \frac{(x_i - x'_i)(x_i - x'_i)}{I} \quad (32)$$

$$MSE_y = \sum_{i=1}^{I} \frac{(y_i - y'_i)(y_i - y'_i)}{I} \quad (33)$$

$$MSE_{\Delta x} = \sum_{i=1}^{I} \frac{(\Delta x_i - \Delta x'_i)(\Delta x_i - \Delta x'_i)}{I} \quad (34)$$

$$MSE_{\Delta y} = \sum_{i=1}^{I} \frac{(\Delta y_i - \Delta y'_i)(\Delta y_i - \Delta y'_i)}{I} \quad (35)$$

$$MSE_{\text{med}} = \frac{MSE_x + MSE_y + MSE_{\Delta x} + MSE_{\Delta y}}{4} \quad (36)$$

onde i é o índice representando cada quadro do vídeo. I é o número total de quadros na imagem, e Δx_i e Δy_i deslocamentos.

Após a implementação do filtro é realizado um ajuste paramétrico para se obter o melhor resultado com relação ao avaliador proposto acima. Para maior simplicidade consideram-se inicialmente as matrizes de incertezas M_f, N_f e N_g como tendo os valores usados em [5]. Os outros parâmetros deverão ser ajustados com base na minimização do MSE_{med}, usado como medidor de desempenho do filtro.

$$N_f = [0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1]$$
$$N_g = [1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$$
$$M = [0.1 \quad 0.2 \quad 0.1 \quad 0.1 \quad 0.1 \quad 0.1]^T$$

Para o início dos testes consideram-se os valores usados na versão nominal do filtro. *"P1"*, *"P2"*, *"P3"* e *"P4"* serão determinados através de uma busca em um intervalo previamente definido. Observe que os MSE_s (Mean Square Errors), Apêndice A, são calculados do *2º* ao *49º* quadro por dois

motivos: no primeiro quadro não há uma estimativa preditora, mas apenas um "chute" inicial; e no último quadro não é possível calcular o deslocamento dx e dy, pois seria necessário o conhecimento da posição no quadro *51*.

O primeiro parâmetro *"P1"* é referente ao T, taxa de captura dos quadros do vídeo, medidos em *segundos/quadro*. Em uma aplicação em tempo real este parâmetro seria determinado pela configuração da câmera, porém, no vídeo pré-processado usado no exemplo, se refere ao intervalo que o programa de captura dos quadros leva entre uma aquisição e outra. Sabe-se previamente que este valor deve ser próximo de $1s$ e utilizando isso como partida estima-se o melhor valor para este parâmetro. Para tanto usa-se um laço de controle de fluxo que faz com que T varie de *0,02* a *2* com um passo de *0:02*. O gráfico da figura *12* mostra a curva do desempenho em função do parâmetro *"P1"*. Conclui-se assim o valor subótimo do parâmetro *"P1"* é $1s$. Subótimo porque não foi realizado um processo formal de minimização do MSE_{med} em função deste parâmetro.

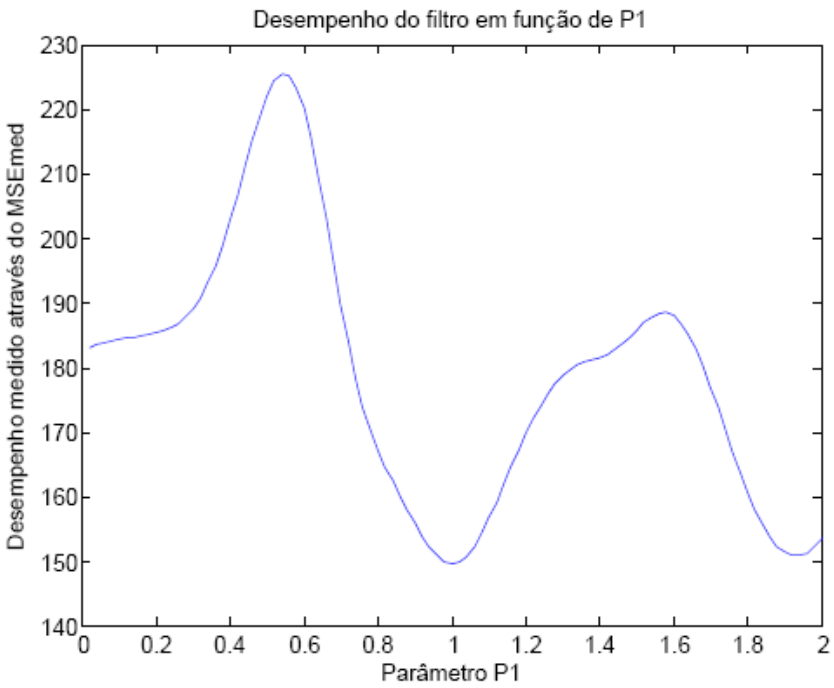
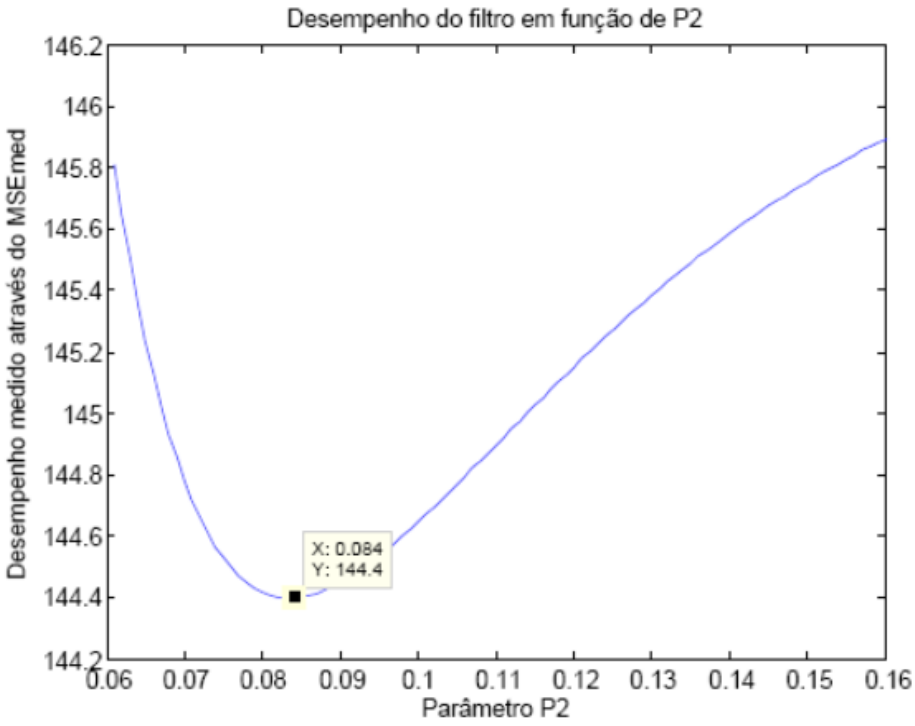


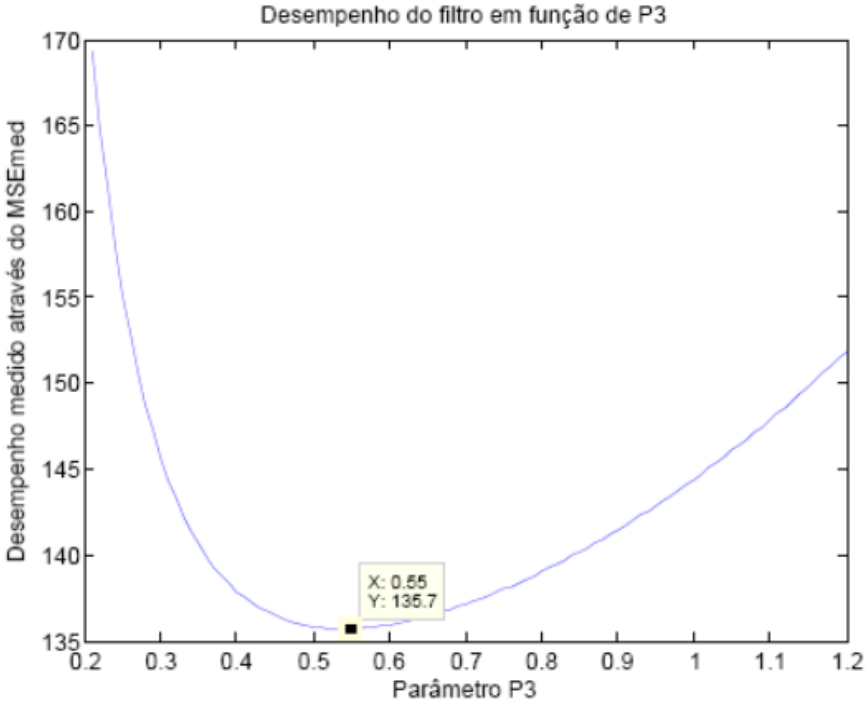
Figura 12 - Variação do $\boldsymbol{MSE_{med}}$ conforme o parâmetro $\boldsymbol{P1}$ varia. O ponto com melhor $\boldsymbol{MSE_{med}}$, ou seja, melhor desempenho do filtro é o ponto $\boldsymbol{T = 1}$

Para as matrizes de covariância dos ruídos existem vários métodos on-line para este ajuste. Porém aqui adotou-se um método prático que consiste em supor que a variância é praticamente a mesma para todos os termos do vetor e portanto podem ser escritas como um parâmetro escalar multiplicado pela matriz identidade. Desta forma os parâmetros *"P2"* e *"P3"*, o primeiro referindo-se á dinâmica do sistema e o segundo ao processo de medição, podem ser ajustados da mesma maneira

que o parâmetro anterior. Foram levantados os gráficos da figura 13-a e b que permitem determinar seus valores, novamente de forma subótima. Os pontos onde ocorre o menor MSE_{med} são $P2 = 0{,}084\ e\ P3 = 0{,}55$.



(a)



(b)

Figura 12 - Resposta do filtro em função de P2(a) e P3(b)

O parâmetro "*P4*" é usado para configurar a área da janela de busca do alvo e será abordado somente no final como um dos critérios de avaliação. O último parâmetro, *"P5"*, refere-se ao α. Este valor é usado na determinação do λ. Pode-se demonstrar que normalmente o valor ótimo de λ encontra-se perto do seu limite inferior, que é dado pela equação *26*. Um valor aproximado para α seria *0,5*, como citado em [5] e [30]. A figura *14* mostra o comportamento do filtro com a variação do parâmetro "*P5*". O valor sub-ótimo encontrado para α foi de 1,6.

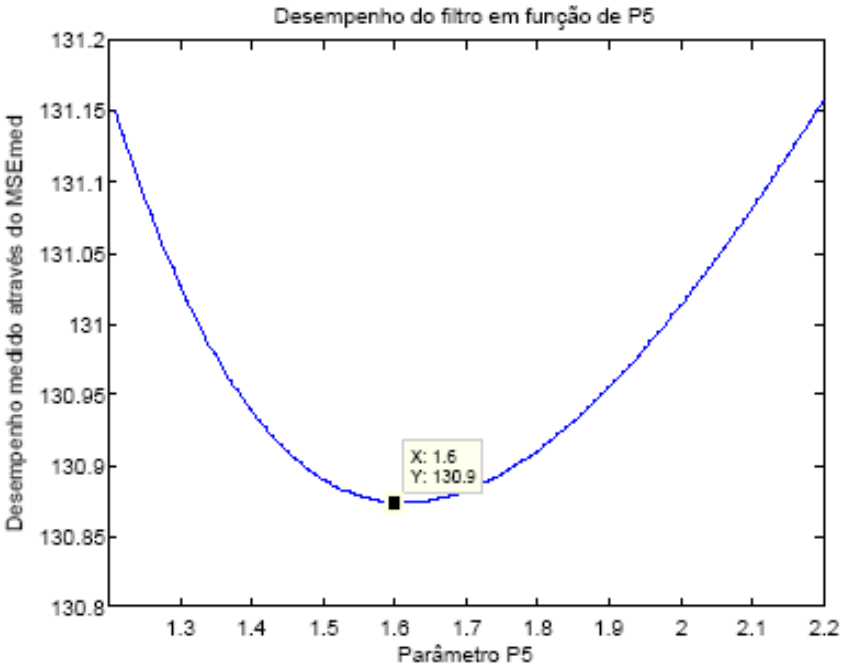


Figura 13 - Resposta do filtro em função de $\boldsymbol{P5}$ encontrado para $\boldsymbol{\alpha}$ foi $\mathbf{1,6}$

Até o momento, apenas o MSE_{med} foi usado como fator de medida de desempenho do filtro robusto BDU (*bounded data uncertainties*), porém, seria conveniente comparar com o desempenho do filtro nominal. Para isso propõe-se uma nova medida de desempenho dada por:

$$I = \frac{MSE_{med_K} - MSE_{med_R}}{MSE_{med_K}} \times 100\% \qquad (37)$$

Com os parâmetros ajustados, calcula-se a melhoria I do filtro robusto em relação ao filtro de Kalman nominal

$$MSE_{med_K} = 116.6$$
$$MSE_{med_R} = 130.9$$
$$I = -12.26\%$$

O valor obtido foi $-12{,}6\%$, ou seja, na verdade houve uma deterioração do desempenho do filtro. Mesmo maximizando o desempenho do filtro robusto, este ainda encontra-se aquém do filtro nominal. A única explicação para isso seria a escolha inadequada das matrizes de incertezas. A escolha realizada garantiu a estabilidade do filtro, contudo, o superdimensionamento das incertezas possivelmente pode ter acarretado em uma deterioração do desempenho. Para verificar esta possibilidade será feita uma avaliação da incerteza em M_f multiplicando-a por um fator entre 0 e 1, ver Figura 15. Um método mais elaborado para o cálculo das matrizes de incertezas será abordado adiante.

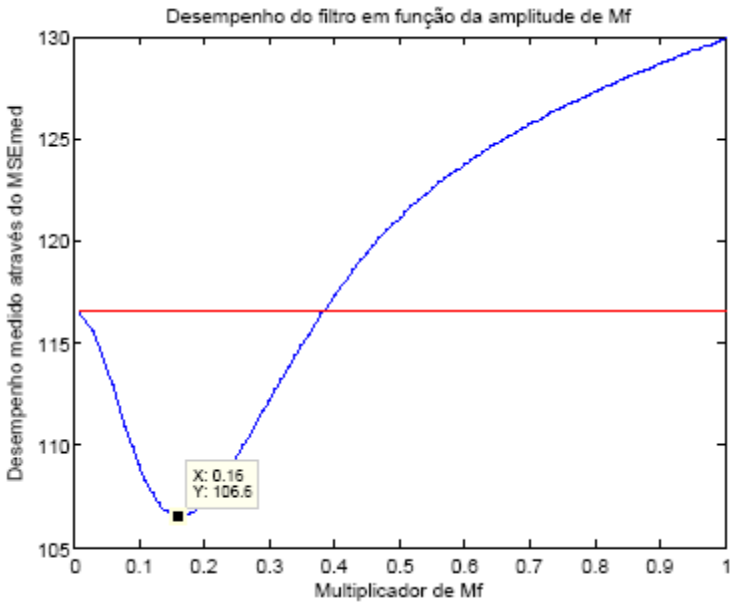


Figura 14 - Resposta obtida variando-se a amplitude da matriz $\mathbf{M_f}$. A linha vermelha indica o desempenho do filtro nominal. Note que há uma faixa de valores para os quais o desempenho do filtro robusto é superior

Como dito no parágrafo anterior, limitando-se a amplitude da incerteza em M_f, foi possível obter uma melhora de até *8,58%*. Esta manobra foi feita simplesmente com intuito ilustrativo. As matrizes que modelam as incertezas serão, na verdade, obtidas através de algoritmos genéticos.

8 Algoritmos genéticos

Pode-se notar que os valores de M_f, N_f e N_g foram mantidos constantes e iguais ao valor usado em [12]. Estes valores são empíricos e dependem dos parâmetros físicos do problema, sendo assim, devem ser ajustados de acordo com a aplicação. Este ajuste, entretanto, não pode ser feito através do mesmo método utilizado anteriormente, pois tratam-se de matrizes. Um método mais sistemático será usado para o ajuste das matrizes de incerteza. Para isso será usado um algoritmo genético.

Os algoritmos genéticos são ferramentas muito utilizadas em processos de otimização por duas razões, por vasculharem várias regiões do espaço de busca de uma vez e por usarem informações históricas para experimentar novos pontos de busca. Sendo assim, são muitas vezes chamados de métodos de busca paralelas randômicas direcionadas.

Estes algoritmos fazem uso da teoria evolutiva muito conhecida na biologia. O princípio é tratar cada possível solução como um individuo que possui genes que determinam suas características. Sendo cada individuo uma possível solução para o problema, basta selecionar os indivíduos que apresentem um melhor desempenho, ou seja, estão mais próximos do ponto ótimo, processo também conhecido como seleção natural, e gerar descendentes a partir destes. Os descendentes podem ser gerados através de cruzamentos e mutações. Cada passo do algoritmo genético permite que sejam usadas várias variações e técnicas diferentes.

Os principais passos em um algoritmo genético são: gerar a população inicial, selecionar os indivíduos mais aptos (melhores soluções), gerar descendentes (cruzamento e mutação). Um critério de parada pode ser adicionado para limitar o número de gerações que serão analisadas. A figura *16* mostra a estrutura do algoritmo.

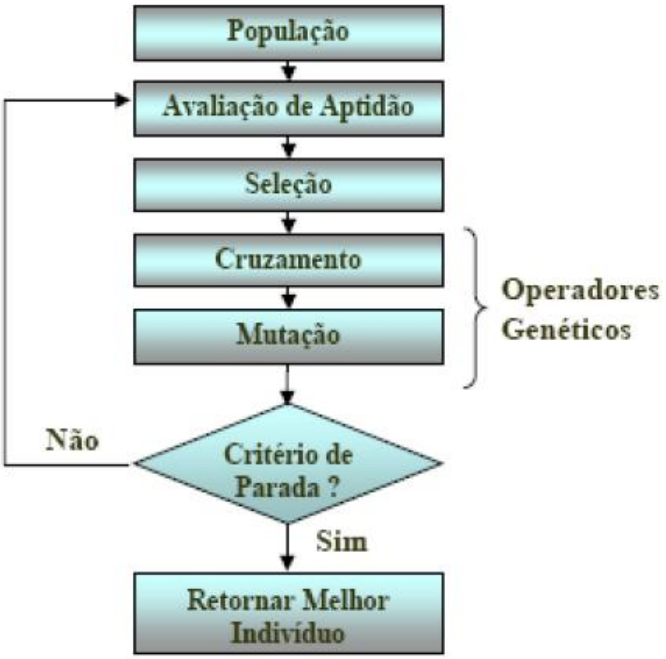


Figura 15 - Esquema de um algoritmo genético

Na Figura *17* pode ser visto o resultado do algoritmo genético ao longo das gerações. Neste caso foi aplicado o algoritmo genético no filtro BDU para calcular M_f, N_f e N_g com valores entre *0* e *20* e uma precisão de *0,001*. Cada valor em uma destas matrizes era tratado como sendo binário, sendo cada bit um gene. A cada geração do algoritmo genético foi gerada uma população de *400* indivíduos. Destes *400* indivíduos, eram selecionados os *50* melhores e eram colocados em uma roleta, onde cada "fatia" era proporcional á sua aptidão relativa. Destes *50* indivíduos, *25* eram selecionados, através da roleta, para gerarem descendentes. Os descendentes eram gerados através de cruzamento ou mutação. Para esta reprodução foram definidas duas variáveis, a probabilidade de cruzamento e a probabilidade de mutação, sendo variáveis importantes no controle da convergência e variabilidade.

É importante fazer algumas observações sobre o AG. Primeiro, nada garante que a solução encontrada seja a solução ótima, pois trata-se de um método heurístico e está sujeito a encontrar mínimos locais. Segundo, a mudança nas matrizes de incertezas acarreta mudanças nos demais parâmetros, como, por exemplo, o deslocamento α do ótimo de *1,6* para *0,8*.

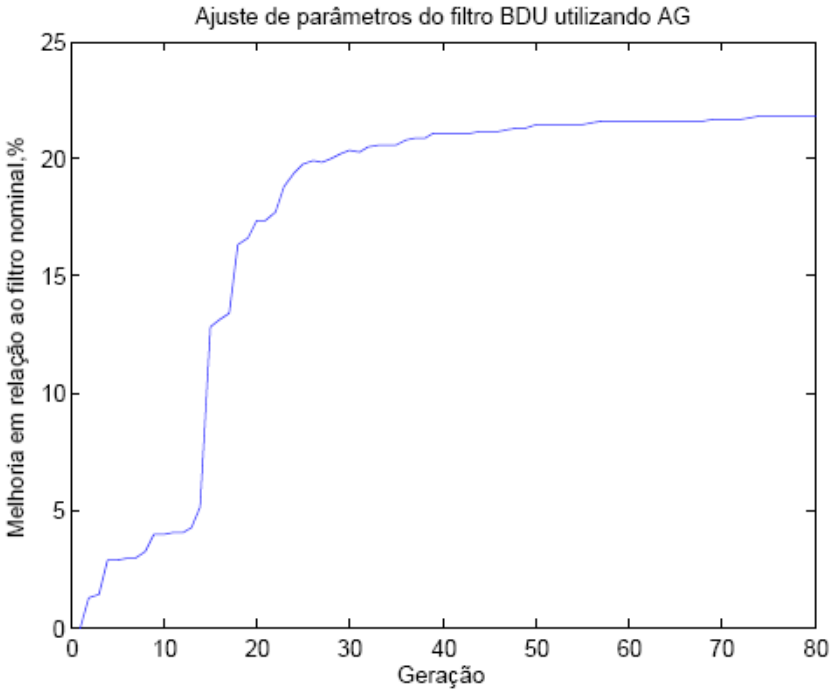


Figura 16 - Evolução do desempenho do filtro BDU ao longo das gerações

O importante aqui é notar que com o filtro BDU e um ajuste razoável das matrizes de incertezas foi possível obter *21,8%* de melhora em relação ao filtro nominal. Vale ainda ressaltar que o resultado depende de vários fatores como o tamanho da população, taxa de mutação e cruzamento, número de gerações avaliadas, entre outros.

Uma explicação mais detalhada do funcionamento do algoritmo aqui proposto pode ser encontrada no Apêndice B.

As matrizes obtidas através do algoritmo genético são:

$$M_f = 0.01 \times [16.115 \quad 0.226 \quad 4.971 \quad 8.730 \quad 14.417 \quad 12.411]^T;$$
$$N_f = 0.1 \times [0.000 \quad 0.002 \quad 0.478 \quad 4.116 \quad 3.414 \quad 16.302];$$
$$N_g = 0.1 \times [0.853 \quad 0.014 \quad 16.099 \quad 12.031 \quad 0.011 \quad 16.310].$$

9 Filtro robusto recursivo

As estimativas obtidas pelo uso do filtro BDU surgem de uma conveniente utilização do método dos mínimos quadrados regularizados. Uma solução alternativa para o problema de filtragem robusta pode ser obtida a partir da utilização dos métodos dos mínimos quadrados regularizados em conjunto com funções penalidades, veja [24].

Esta nova abordagem traz algumas vantagens, entre elas, estruturas em blocos matriciais simétricos e uma formulação mais geral, permitindo a inserção de incertezas em todos os parâmetros.

Considere o seguinte modelo incerto

$$\boldsymbol{x_{k+1}} = (F_k + \delta F_k)\boldsymbol{x_k} + (G_k + \delta G_k)\boldsymbol{w_k} \qquad (38)$$

$$\boldsymbol{z_{k+1}} = (H_{k+1} + \delta H_{k+1})\boldsymbol{x_{k+1}} + (K_{k+1} + \delta K_{k+1})\boldsymbol{v_{k+1}} \qquad (39)$$

cujas incertezas paramétricas são modeladas por

$$\begin{bmatrix} \delta F_k & \delta G_k & 0 & 0 \\ 0 & 0 & \delta K_k & \delta H_{k+1} \end{bmatrix} := \begin{bmatrix} M_{1,k} & 0 \\ 0 & M_{2,k} \end{bmatrix} \begin{bmatrix} \Delta_1 & 0 \\ 0 & \Delta_2 \end{bmatrix} \begin{bmatrix} N_{F_k} & N_{G_k} & 0 & 0 \\ 0 & 0 & N_{K_k} & N_{H_{k+1}} \end{bmatrix}$$

sendo $\|\Delta_j\| \leq 1$, $j = 1,2$, $\boldsymbol{x}_k \in \Re^n$ o estado; $\boldsymbol{x}_k \in \Re^p$ a medida de saída, $\boldsymbol{w_k}$ e $\boldsymbol{v_{k+1}}$ são ruídos Gaussianos não correlacionados com média zero e variâncias Q_k e R_k, respectivamente. As matrizes $\mathrm{M}_{1,\mathrm{k}}$, $\mathrm{M}_{2,\mathrm{k}}$, N_{F_k}, N_{G_k}, N_{K_k} e $N_{H_{k+1}}$ são conhecidas.

Novamente o problema consiste em resolver um problema de otimização $min\, max$, desta vez usando funções penalidades

$$\min_{x_k, x_{k+1}} \max_{\delta_k} J_k \qquad (40)$$

sendo $\delta_k := \{\delta F_k, \delta G_k, \delta K_k, \delta H_k\}$ e a função penalidade $\mathrm{J_k}$ associada definida como

$$J_k := \begin{bmatrix} x_k - \hat{x}_{k|k} \\ w_k \\ v_{k+1} \\ x_{k+1} \end{bmatrix}^T \begin{bmatrix} P_{k|k}^{-1} & 0 & 0 & 0 \\ 0 & Q_k^{-1} & 0 & 0 \\ 0 & 0 & R_{k+1}^{-1} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k - \hat{x}_{k|k} \\ w_k \\ v_{k+1} \\ x_{k+1} \end{bmatrix}$$

$$+ \left(\begin{bmatrix} F_{\delta,k} & G_{\delta,k} & 0 & -I \\ 0 & 0 & K_{\delta,k} & H_{\delta,k} \end{bmatrix} \begin{bmatrix} x_k - \hat{x}_{k|k} \\ w_k \\ v_{k+1} \\ x_{k+1} \end{bmatrix} - \begin{bmatrix} -F_{\delta,k}\hat{x}_{k|k} \\ z_{k+1} \end{bmatrix} \right)^T \mu I([\bullet]),$$

$$F_{\delta,k} = (F_k + \delta F_k), G_{\delta,k} = (G_k + \delta G_k), H_{\delta,k} = (H_{k+1} + \delta H_{k+1}),\ K_{\delta,k} = (K_k + \delta K_k).$$

Note que duas variáveis faltam ser definidas: $P_{k|k}$ representa a variância do erro de estimativa $x_k - \hat{x}_{k|k}$ e o parâmetro μ vem da teoria de programação não linear, mais especificamente de funções penalidades, veja [29] e as referências contidas nela. O objetivo deste parâmetro é fazer com que os termos relacionados a ele, entre parênteses à esquerda e à direita, convirjam para zero quando μ tender para o infinito.

Como resultado, o filtro robusto parametrizado por variáveis de penalidade tem uma semelhança impressionante com o filtro de Kalman convencional, que não é robusto. Em virtude disto, as provas da convergência e da estabilidade podem ser obtidas facilmente. Detalhes dessas provas, feitas para a versão em blocos matriciais descrita logo a seguir, podem ser vistas em [28] e no artigo [27] apresentado no *18th IFAC World Congress*.

Para a apresentação do filtro robusto é necessária a definição de algumas variáveis auxiliares

$$\mathcal{E}_k := \begin{bmatrix} -I \\ H_{k+1} \end{bmatrix}, \mathcal{F}_k := \begin{bmatrix} F_k \\ 0 \end{bmatrix}, \mathcal{G}_k := \begin{bmatrix} G_k & 0 \\ 0 & K_k \end{bmatrix}, \mathcal{Z}_{k+1} := \begin{bmatrix} 0 \\ z_{k+1} \end{bmatrix},$$

$$N_{\mathcal{E},k} := \begin{bmatrix} 0 \\ N_{H_{k+1}} \end{bmatrix}, N_{\mathcal{F},k} := \begin{bmatrix} N_{F_k} \\ 0 \end{bmatrix}, N_{\mathcal{G},k} := \begin{bmatrix} Q_k & 0 \\ 0 & R_{k+1} \end{bmatrix},$$

$$M_k := \begin{bmatrix} M_{1,k} & 0 \\ 0 & M_{2,k} \end{bmatrix}, \mathcal{R}_k := \begin{bmatrix} Q_k & 0 \\ 0 & R_{k+1} \end{bmatrix}. \qquad (41)$$

Teorema 1: A estimativa preditora robusta ótima para o sistema (23) e a equação recursiva de Riccati são dadas por

$$\begin{bmatrix} \hat{x}_{k+1|k} & P_{k+1|k} \end{bmatrix} =$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ I \end{bmatrix}^T \begin{bmatrix} P_{k|k-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 \\ 0 & \mathcal{R}_k & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & \hat{\lambda}_k^{-1} I & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\hat{\lambda}_k^{-1} I & 0 & 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu^{-1} I & 0 & 0 & M_k & \mathcal{F}_k & \mathcal{G}_k & \mathcal{E}_k \\ 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & N_{\mathcal{F},k} & N_{\mathcal{G},k} & N_{\mathcal{E},k} \\ 0 & 0 & I & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & M_k^T & 0 & 0 & 0 & 0 & 0 & 0 \\ I & 0 & 0 & 0 & \mathcal{F}_k^T & N_{\mathcal{F},k}^T & 0 & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & \mathcal{G}_k^T & N_{\mathcal{G},k}^T & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathcal{E}_k^T & N_{\mathcal{E},k}^T & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} \hat{x}_{k|k-1} & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \mathcal{Z}_k & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -I \end{bmatrix}, \quad (42)$$

onde o parâmetro lambda $\hat{\lambda}_K$ é obtido de maneira equivalente ao filtro BDU sobre o intervalo $\hat{\lambda}_K \geq \left\|\mu M_k^T M_k\right\|$, para um dado valor de μ.

9.1 Resultados obtidos com o filtro robusto recursivo

O filtro robusto recursivo ótimo mostrado anteriormente foi implementado no *Matlab*. O ajuste paramétrico para este filtro é muito mais complexo, pois além de possuir mais matrizes de incertezas, também possui o parâmetro μ. Para este ajuste adotou-se o algoritmo genético com algumas modificações. Este novo algoritmo inclui: todas as matrizes de incertezas *(=6)*, um multiplicador para cada matriz de incerteza *(=6)*, a matriz Q e R *(=2)*, o parâmetro μ e α *(=2)*. No total são *16* parâmetros com precisões e intervalos diferentes, o que torna o algoritmo genético complexo e pesado.

A partir de uma estimativa inicial realizada apenas com as matrizes de incertezas, o algoritmo genético ficou em funcionamento durante mais de uma semana, totalizando *2783* gerações. A figura *18* mostra a evolução do desempenho até o valor máximo de *32,69%*.

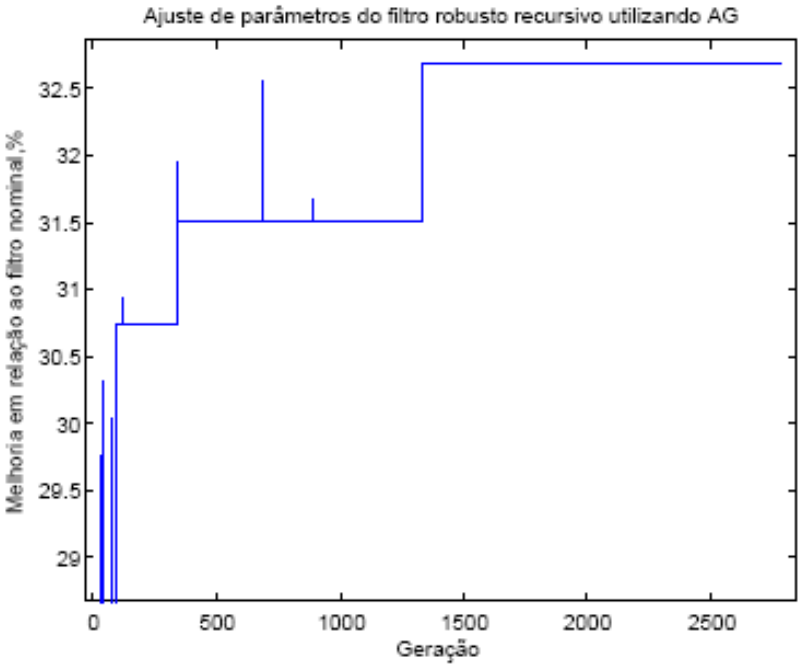


Figura 17- Evolução do desempenho do filtro robusto recursivo ótimo ao longo das gerações

Vale ressaltar que foram feitas várias tentativas com o filtro BDU para tentar igualar este desempenho, todas sem sucesso. Estas tentativas não constam neste documento, porém os resultados divergiram muito pouco dos resultados mostrados na seção anterior. Além disso, são apresentadas, no apêndice A, as funções utilizadas para as simulações, o que permite que qualquer um interessado possa reproduzir os experimentos.

Os valores encontrados foram:

$$M_f = (10^{-50}) \times [6.045 \quad 31.825 \quad 7.427 \quad 52.156 \quad 26.964 \quad 26.668]^T;$$

$$N_f = (10^{-1.7}) \times [0.276 \quad 0.78 \quad 57.212 \quad 0.976 \quad 21.139 \quad 6.171];$$

$$N_g = (10^{-1.4}) \times [18.55 \quad 12.332 \quad 23.007 \quad 1.87 \quad 1.573 \quad 32.272 \quad 13.781 \quad 63.878];$$

$$M_h = (10^{-48}) \times [43.899 \quad 16.406]^T;$$

$$N_h = (10^{-2.2}) \times [0.687 \quad 389.51 \quad 22.461 \quad 4822.2 \quad 38.074 \quad 312690];$$

$$N_k = (10^{-0.9}) \times [38.555 \quad 19.303 \quad 48.692 \quad 33.995 \quad 12.363 \quad 54.596 \quad 29.853 \quad 49.182];$$

$$Q = 2.41; \;\; R = 0.01$$

$$\alpha = 1.22 \; e \; \mu = 10^{42}$$

Algumas análises interessantes podem ser feitas que relacionem os dados obtidos com a teoria do filtro. O parâmetro μ garante que os termos do funcional tendem a zero quando o mesmo tende ao infinito, logo o μ esperado na prática deve ser extremamente grande, ver gráfico da figura *19*. O valor encontrado foi de 10^{42}, mas pode-se constatar que para 10^{300} o desempenho do filtro é o mesmo.

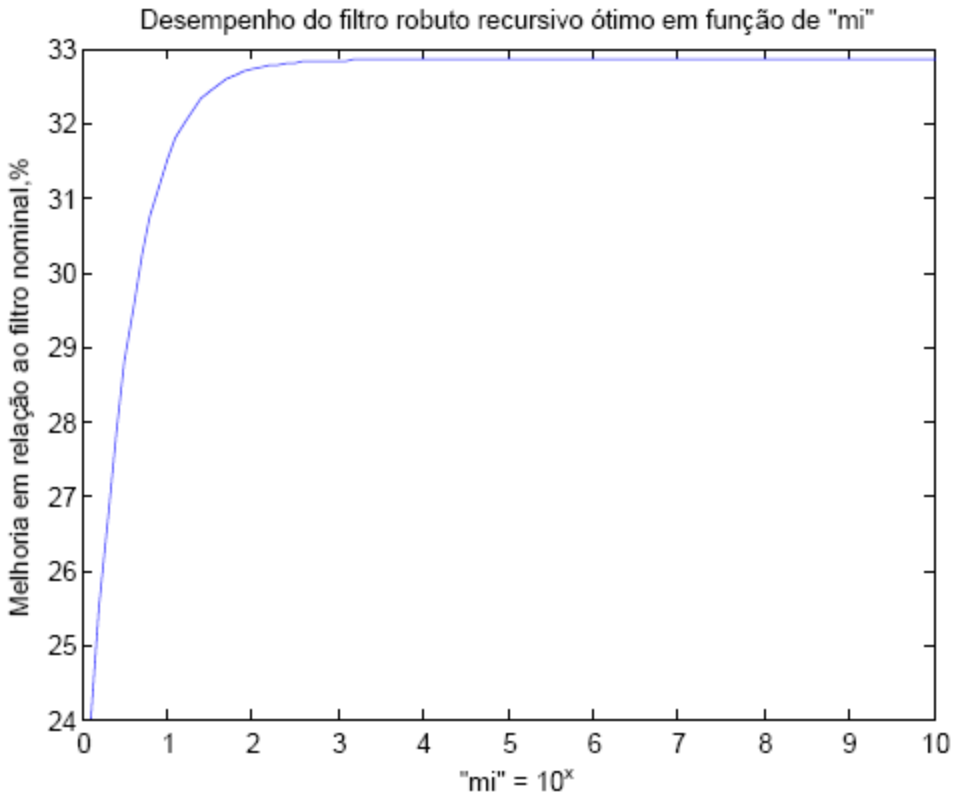


Figura 18 - Desempenho do filtro robusto recursivo com relação ao μ

Outra constatação é com relação ao parâmetro α. Observando-se a figura *20* pode-se notar que o desempenho do filtro é independente deste parâmetro para valores de μ suficientemente grandes. Esta é uma das maiores vantagens desta metodologia, se comparada com o que existe na literatura, pois o filtro não depende do ajuste deste parâmetro auxiliar, sendo indicado para aplicações em tempo real.

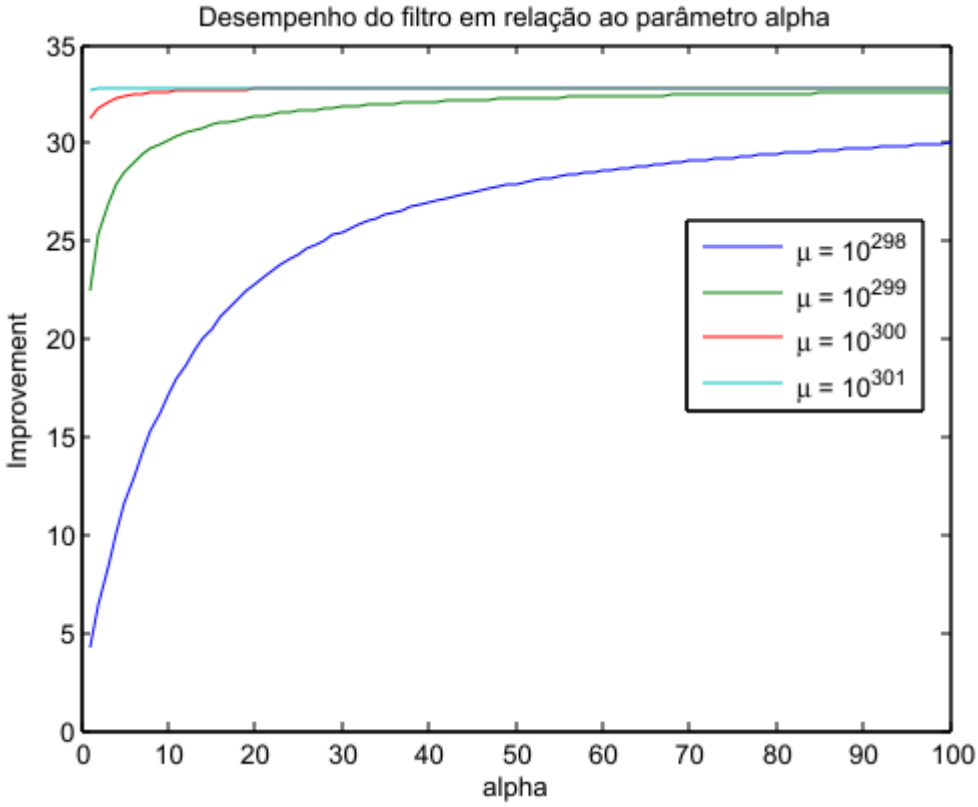


Figura 19 - Desempenho do filtro robusto recursivo com relação ao α. Conforme **μ** aumenta o desempenho do filtro se torna independente do valor de α

Em relação as matrizes M_f e M_h foram levantados os gráficos das figuras *21-a* e *b*. Pode-se perceber que as matrizes M_f e M_h tendem a zero sem modificar em nada o desempenho do filtro, ou seja, elas perdem a função no modelo das incertezas, que fica a cargo das matrizes N_f , N_g, N_h, N_k.

Aplicando-se a teoria de Funções Penalidade a fim de encontrar a solução ótima, consideram-se os limites

$$\mu \to \infty, \lambda_i^{-1} \to 0 \; e \; W_i^{-1} \to 0 \qquad (43)$$

Desta forma, uma nova versão do filtro pode ser definida

$$\begin{bmatrix} \hat{x}_{i+1|i} & P_{i+1|i} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ I \end{bmatrix}^T \begin{bmatrix} P_{i|i-1} & 0 & 0 & 0 & I & 0 & 0 \\ 0 & \mathcal{R}_i & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & \mathcal{F}_i & \mathcal{G}_i & \mathcal{E}_i \\ 0 & 0 & 0 & 0 & N_{\mathcal{F},i} & N_{\mathcal{G},i} & N_{\mathcal{E},i} \\ I & 0 & \mathcal{F}_i^T & N_{\mathcal{F},i}^T & 0 & 0 & 0 \\ 0 & I & \mathcal{G}_i^T & N_{\mathcal{G},i}^T & 0 & 0 & 0 \\ 0 & 0 & \mathcal{E}_i^T & N_{\mathcal{E},i}^T & 0 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} \hat{x}_{i|i-1} & 0 \\ 0 & 0 \\ \mathcal{Z}_i & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -I \end{bmatrix}$$

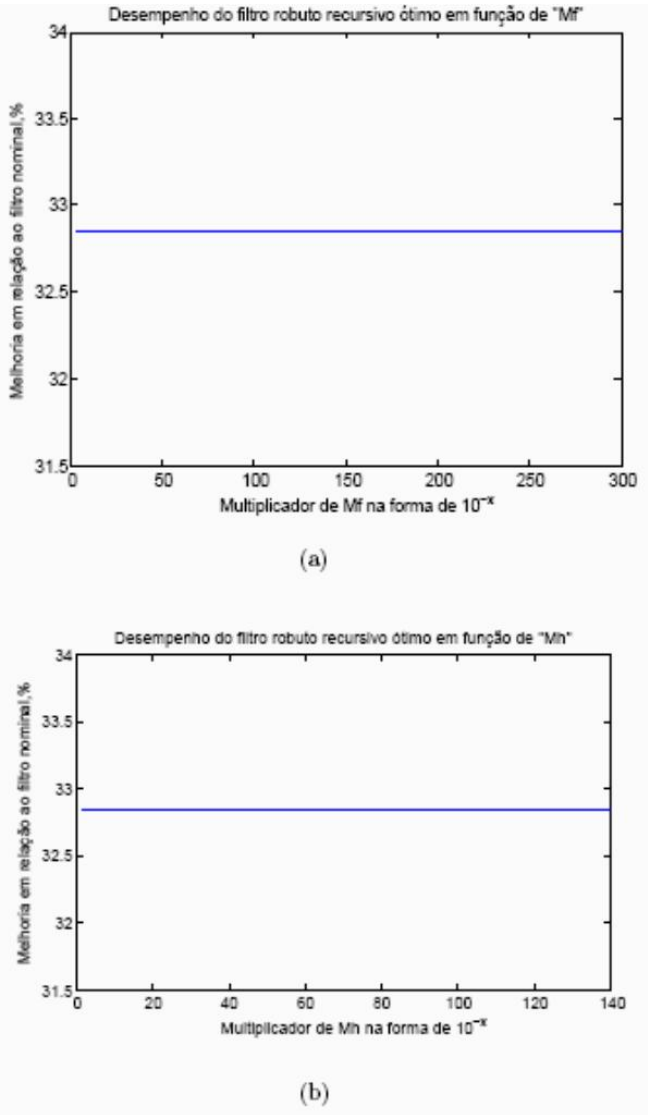


Figura 20 - Desempenho do filtro robusto recursivo com relação as matrizes M_f e M_h

para $\mathcal{E}_i, \mathcal{F}_i, \mathcal{G}_i, N_{\mathcal{G},i}, N_{\mathcal{E},i}, N_{\mathcal{F},i}$ definidas em 41.

Como esperado, com esta versão do filtro implementada obteve-se o mesmo resultado da versão anterior quando μ tendia para o infinito e as matrizes $\mathrm{M_f}$ e $\mathrm{M_h}$ tendiam a zero. Vale ressaltar mais uma vez que esse resultado é extremamente importante para aplicações em tempo real, já que não requer ajuste adicional de parâmetros.

10 Redução da área de busca do alvo na imagem

Voltando ao objetivo principal da construção dos filtros robustos, que era o rastreamento visual de trajetória de alvos móveis, deve-se analisar qual a vantagem prática da construção de filtros robustos. A primeira questão é analisar a redução da área de busca na imagem com os três filtros implementados (nominal, BDU e o recursivo ótimo). A segunda questão a ser analisada é o comportamento das estimativas em relação à movimentação dos alvos, ou seja, qual filtro fornece um resultado mais suave e estável (isso significa que o filtro estará menos sujeito a variações bruscas no movimento do alvo).

Para realizar a comparação entre os três filtros em relação à redução da janela de busca utilizou-se o parâmetro *"P4"*, ver capitulo *6*. Para cada filtro, foram feitas simulações reduzindo a janela de busca através de *"P4"*. O processo foi repetido até que o rastreamento visual falhou, devido à redução excessiva da janela de busca. Para o filtro nominal foi obtida uma janela mínima de busca com lados medindo *17,29* pixels, para o BDU, *14,61* pixels e para o robusto recursivo, *19,83* pixels.

Apesar de o filtro BDU ter permitido uma redução um pouco maior na área de busca em relação ao filtro nominal, percebe-se que o filtro recursivo ótimo teve um pior desempenho neste sentido. Na prática, pouca diferença foi notada entre as reduções na janela de busca. Uma explicação para este comportamento inesperado pode ser visto mais adiante. Vale ressaltar que este tipo de analise é incompleta, pois não leva em consideração se é o centro do alvo que está sendo localizado, ou outro ponto qualquer do alvo. Isto significa que o ponto encontrado como referência para a posição do alvo pode estar oscilando dentro do próprio alvo, ver Figura *22* para o filtro nominal.

(a) (b) (c) (d)

Figura 21 - Oscilação da posição dentro da esfera

Uma maneira mais adequada de avaliar o desempenho dos filtros pode ser feita ao colocar em um gráfico os erros entre a posição real do objeto, encontrada manualmente através da sobreposição da imagem à uma circunferência de centro conhecido, e a posição estimada ao longo dos quadros. Desta forma pode-se chegar às seguintes conclusões: O filtro nominal tem um desempenho bem inferior aos robustos, tanto em termos de erros de posição quanto com relação à suavidade do rastreamento visual.

O filtro BDU, apesar de ter um erro de posição ligeiramente menor que o robusto recursivo ótimo, é mais suscetível às incertezas, nota-se pelo formato de “dente de serra" do gráfico *11.2*. O filtro robusto recursivo ótimo, como já visto anteriormente, é o que melhor incorpora as incertezas no modelo, possui uma curva mais suave, mostrando uma boa estimativa tanto na posição quanto na velocidade. Os dados que demonstram isso podem ser encontrados na tabela *2*.

Através deste gráfico pode-se também notar que nos primeiros quadros o erro do filtro recursivo ótimo é maior, ou seja, ele necessita de um tempo de ajuste e convergência maior. Isso explica o porquê do rastreamento visual com este filtro ter uma redução de janela menor: ele “perde" a localização do alvo no começo da simulação. Era de esperar que um filtro mais complexo tivesse um tempo de resposta maior. O que importa na prática é o funcionamento em regime destes filtros e neste caso o filtro robusto ótimo sob funções penalidades tem um melhor desempenho,

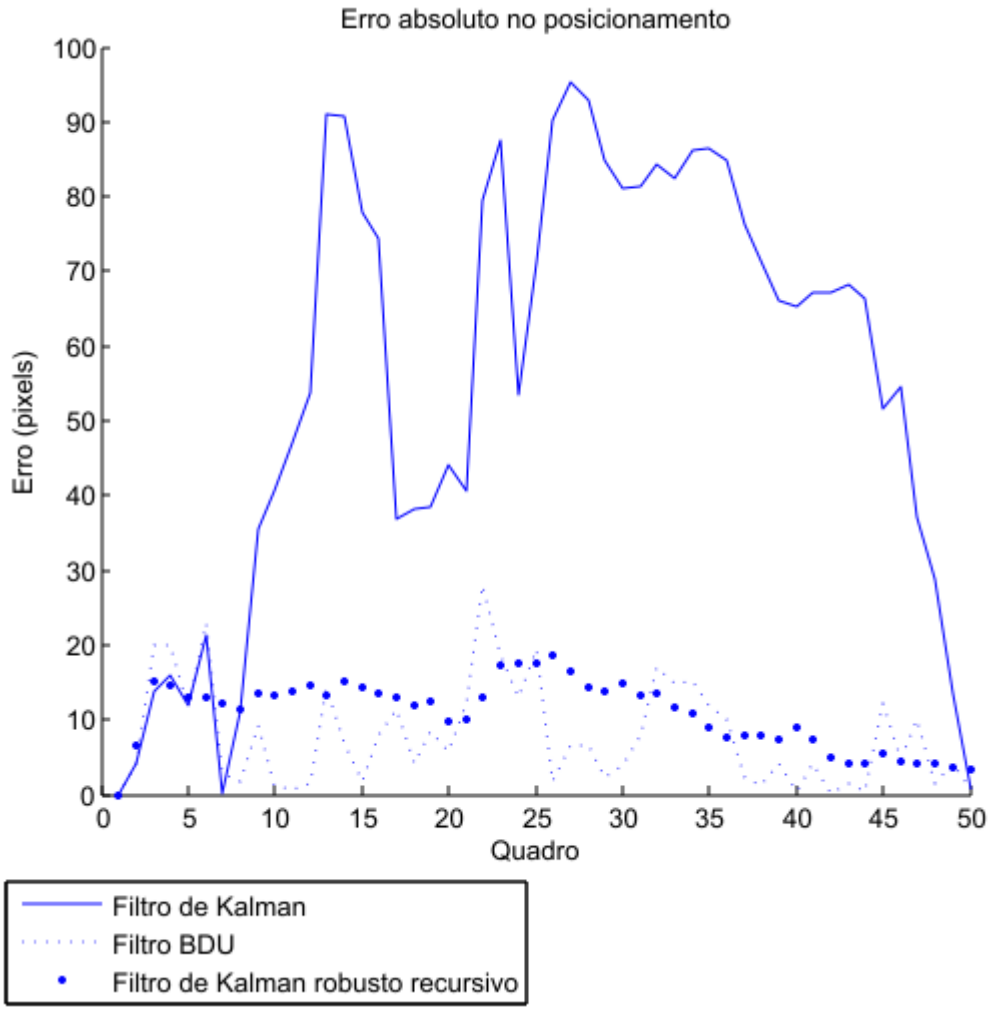


Figura 22 - Comparativo entre os diferentes filtros implementados

Tabela 2 - Estatísticas dos três filtros mostrando as vantagens de se utilizar o Filtro KRR

Estatísticas dos filtros (Erros de posição em pixels)			
Tipo de erro	**Filtro nominal**	**Filtro BDU**	**Filtro RR**
máx	95,37	28,08	18,57
médio	55,27	7,87	10,84
desvio padrão	29,39	6,95	4,52
amplitude	95,37	28,08	18,57

Melhorias em relação ao filtro nominal		
Tipo de erro	**Filtro BDU**	**Filtro KRR**
máximo	71%	81%
médio	86%	80%
desvio padrão	76%	85%
amplitude	29%	19%

O filtro de Kalman robusto recursivo apesar do erro médio ligeiramente superior ao BDU, possui menor erro máximo, menor desvio e uma menor amplitude de variação.

11 Filtro de partículas

Os filtros baseados em Kalman deixam de ser ótimos quando aplicados a sistemas não lineares e/ou com ruídos não gaussianos. Neste caso filtro de partículas podem apresentar melhor resultados. Neste trabalho o termo filtro de partículas será usado com o mesmo significado que o termo *CONDENSATION algorithm – **Con**ditional **Dens**ity Propag**ation.*** Existem várias referências para este filtro, sendo [31] uma das mais completas.

Para entender um pouco sobre a propagação de densidade de probabilidade condicionada em um sistema, deve-se considerar os quatro elementos em um sistema, ou melhor, as quatro funções de densidade de probabilidade (fdp). Três destes elementos formam a especificação do sistema e um deles descreve a solução.

1. A fdp a priori $p(\boldsymbol{x})$ para o estado x.
2. A fdp do processo $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ que descreve a dinâmica do sistema (parte determinística).
3. A fdp da observação (ou medição) $p(\boldsymbol{z}|\boldsymbol{x})$.

Para maiores detalhes destas distribuições de probabilidade pode-se consultar o Apêndice A da referência [31]. O objetivo do filtro é gerar a estimativa $p_t(\boldsymbol{x})$ que é a fdp para o estado a posteriori, isto é $p_t(\boldsymbol{x}) \equiv p(\boldsymbol{x}_t|\boldsymbol{Z}_t)$, onde $\boldsymbol{Z}_t$ é o conjunto de todas as medidas até o momento *t*. Basta apenas uma das fdps descritas acima não ser gaussiana para que a solução também não o seja. Desta forma, apenas quando todas as fdps são gaussianas é que se pode utilizar o filtro de Kalman.

Algumas soluções para os casos em que não ocorrem distribuições gaussianas, embora não façam parte do escopo deste trabalho, podem ser citadas a titulo de curiosidade: Quando a densidade do estado a priori é não gaussiana pode-se utilizar uma aproximação com um somatório de densidades gaussianas com pesos constantes, resultando em um conjunto de filtros de Kalman independentes. Quando a dinâmica é não gaussiana pode-se aproximar o processo por uma expansão de Taylor com pesos variando no tempo, resultado no filtro de Kalman estendido (FKE).

A solução abordada aqui, entretanto, será o filtro de partículas.

11.1 Propagação de densidade de probabilidades condicionadas

O filtro de Kalman foi apresentado no capítulo *4* de uma forma estocástica, isto é, baseada em probabilidades. Uma abordagem probabilística mais geral pode ser feita para se entender como funcionam todos os filtros Bayesianos. A ideia gira em torno da densidade de probabilidade do estado $p(\boldsymbol{x})$ e sua propagação ao longo dos processos. A figura abaixo ilustra esta propagação quando $p(\boldsymbol{x})$ é gaussiana, ver [31]

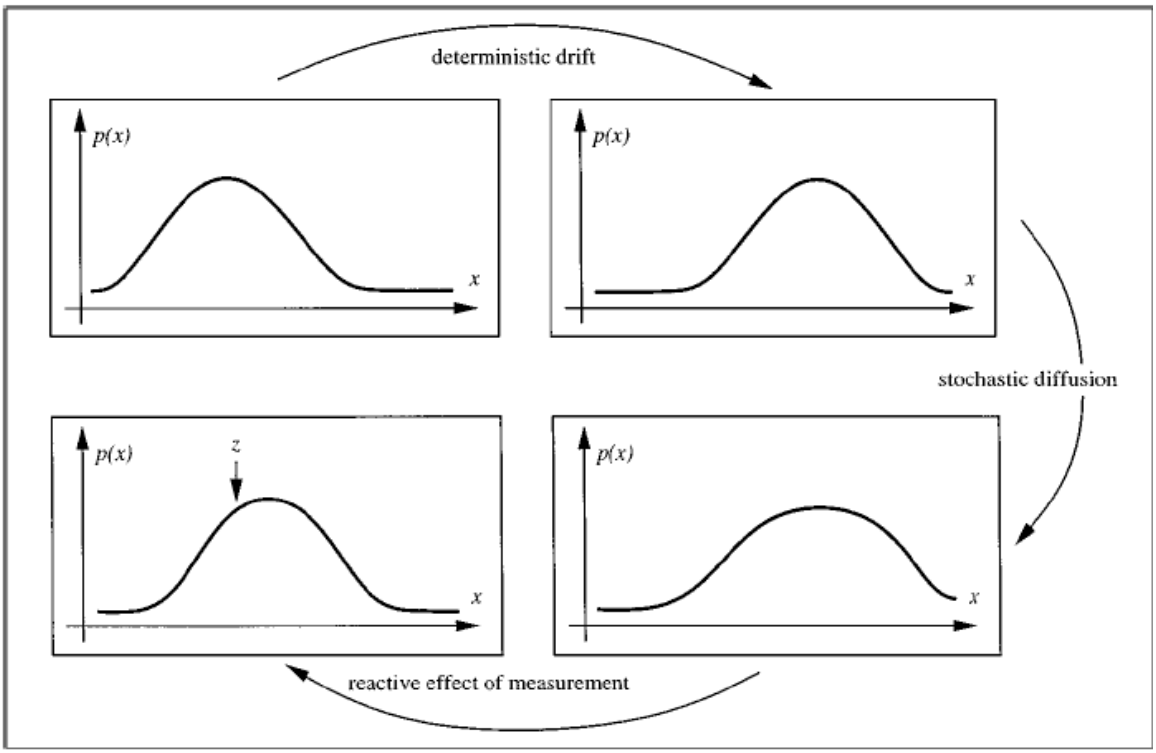


Figura 23 – Propagação da função densidade ou difusão para um processo linear com ruídos gaussianos

Neste caso, o filtro de Kalman foi aplicado. O processo é conhecido como difusão, e é governado pela equação de "Fokker-Planck". Para processos gaussianos isto envolve apenas um pulso que se translada, se espalha e se afina novamente, mas sempre permanecendo gaussiano. A dinâmica do sistema, faz com que o pulso se translade; enquanto os efeitos aleatórios fazem com que ela se espalhe, ou seja, faz com que haja um aumento na incerteza. A observação faz com que haja a formação de um pico em torno da medida, funcionando como um *feedback*, diminuindo a incerteza. O próximo gráfico ilustra o que ocorre em sistemas não gaussianos

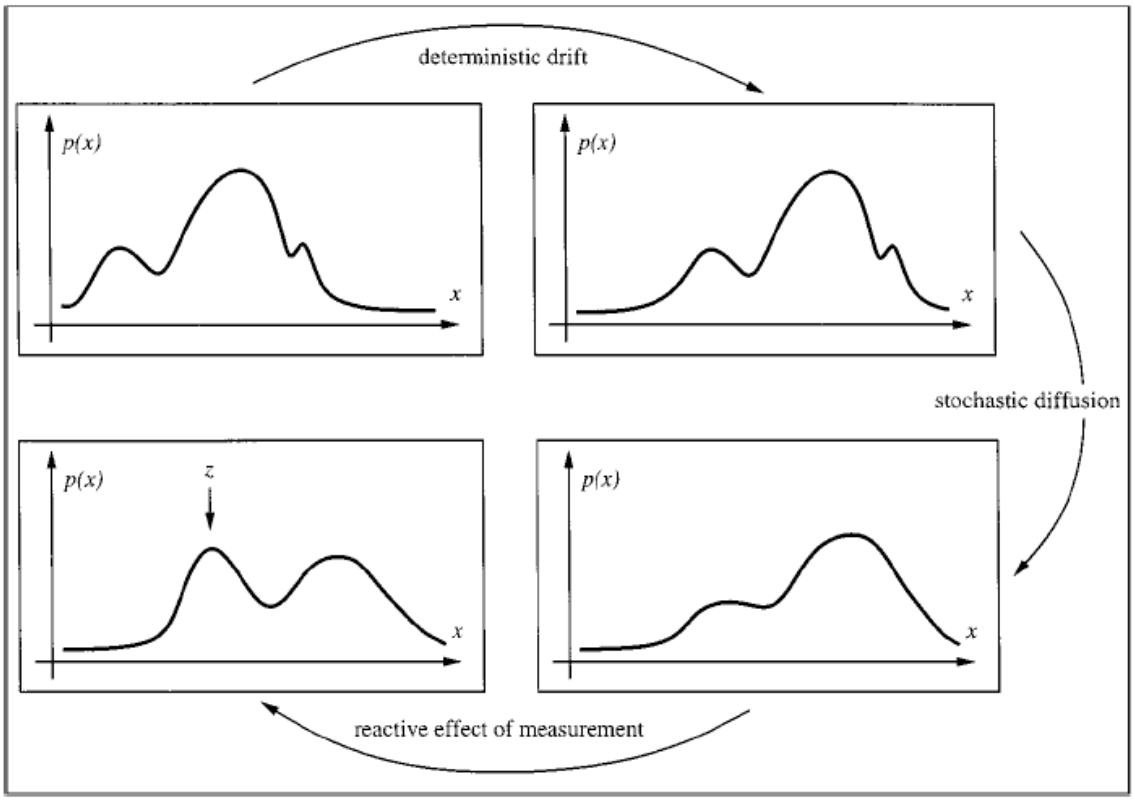


Figura 24 – Difusão para processos não lineares ou com ruídos não gaussianos. Note como há deformações na onda

Neste caso, devido as deformações na onda, a estimativa pode se deteriorar. Para atender a situações como esta, utiliza-se o filtro de partículas. É um algoritmo muito mais simples que o filtro de Kalman, com a desvantagem de ser baseado em amostragem aleatória o que prejudica um pouco a eficiência computacional. Apesar de os gráficos ilustrarem o que ocorre nos filtros, é interessante uma formalização matemática para sistemas em tempo discreto, embora uma completa dedução não seja o objetivo aqui.

11.2 Propagação da densidade em tempos discretos

Para uma melhor compreensão do texto segue uma tabela de notação complementar

$\boldsymbol{\chi}_t$ → Histórico de todos os vetores de estados até o tempo t $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_t\}$.

$\mathcal{Z}_t$ → Histórico de todas as observações até o tempo t $\{\boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_t\}$.

$p(\boldsymbol{x}_t|\mathcal{Z}_t)$ → Densidade a posteriori.

$p(\boldsymbol{x}_t|\mathcal{Z}_{t-1})$ → Densidade a priori.

$p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ → Densidade da observação.

$p(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{t-1})$ → Densidade de inicialização.

Como já feito para a dedução do filtro de Kalman, supõe-se que a dinâmica do sistema forma uma cadeia Markoviana, isto é, cada estado é condicionado ao seu passado imediato

$$p(\boldsymbol{x}_t|\boldsymbol{\chi}_{t-1}) = p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$$

As observações são consideradas independentes umas das outras e também independentes do processo:

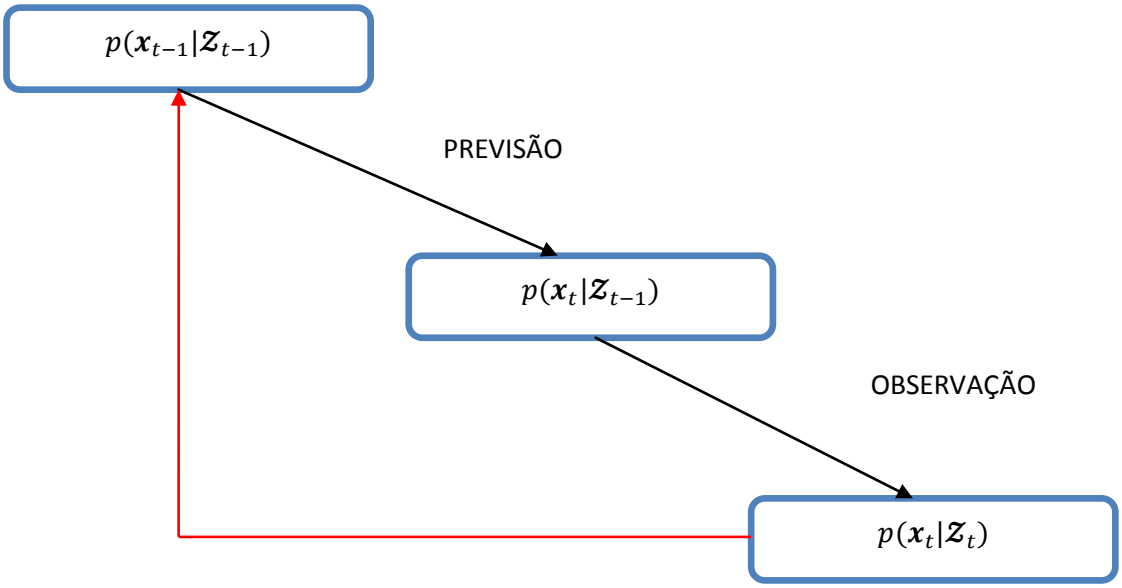
$$p(\mathcal{Z}_{t-1,}\boldsymbol{x}_t|\boldsymbol{\chi}_{t-1}) = p(\boldsymbol{x}_t|\boldsymbol{\chi}_{t-1})\prod_{i=1}^{t-1} p(\boldsymbol{z}_i|\boldsymbol{x}_i)$$

onde

$$\prod_{i=1}^{t-1} p(\boldsymbol{z}_i|\boldsymbol{x}_i) = p(\mathcal{Z}_t|\boldsymbol{\chi}_t)$$

Sendo assim, o estado atual, ou seja, toda a informação que se disponibiliza pode ser representado pelas densidades da forma

$$p_t(\boldsymbol{x}_t) \equiv p(\boldsymbol{x}_t|\mathcal{Z}_t)$$



A previsão e correção pela observação são feitas através das seguintes propagações

$$p(\boldsymbol{x}_t|\mathcal{Z}_{t-1}) = \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})\, p(\boldsymbol{x}_{t-1}|\mathcal{Z}_{t-1}) \qquad (44)$$

$$p(\boldsymbol{x}_t|\mathcal{Z}_t) = kp(\boldsymbol{z}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\mathcal{Z}_{t-1}) \qquad (45)$$

k é uma constante de normalização independente do tempo t. A estimação a priori $p(\boldsymbol{x}_t|\mathcal{Z}_{t-1})$ é feita tomando-se o conhecimento passado $p(\boldsymbol{x}_{t-1}|\mathcal{Z}_{t-1})$ e avançando temporalmente baseado na equação de difusão (Fokker-Planck). A atualização ou correção através da observação é feita similarmente a regra de Bayes, ou seja, a estimativa a priori é multiplicada pela probabilidade da observação ter sido causada por aquele estado. Fica claro que trabalhar com estas funções densidades de probabilidade, além de multiplicações e integrais acarretaria um esforço computacional enorme e não atenderia requisitos de aplicação em tempo real. Várias abordagens tratam deste problema, uma delas, usada pelo filtro de partículas, é a amostragem aleatória.

11.3 Amostragem aleatória e o condensation algorithm

Amostragem aleatória é um método para aproximar funções densidade de probabilidade. Encontrar $p(\boldsymbol{x}|\boldsymbol{z})$ é extrair todo conhecimento a respeito de **x** a partir de **z**. Na impossibilidade de extrair todo o conhecimento, procura-se obter amostras e pesos para estas de forma a ter uma distribuição aproximada.

Passos:

- Em primeiro lugar, é feito uma amostragem $S = \{\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(N)}\}$ aleatória a partir da fdp a priori de **x**.
- Para cada elemento de S, $\boldsymbol{s}^{(i)}$, é atribuído um peso π_i proporcional a densidade de observação $p(\boldsymbol{z}|\boldsymbol{x} = \boldsymbol{s}^{(i)})$

$$\pi_i = \frac{p_z(\boldsymbol{s}^{(i)})}{\sum_{j=1}^{N} p_z(\boldsymbol{s}^{(j)})}$$

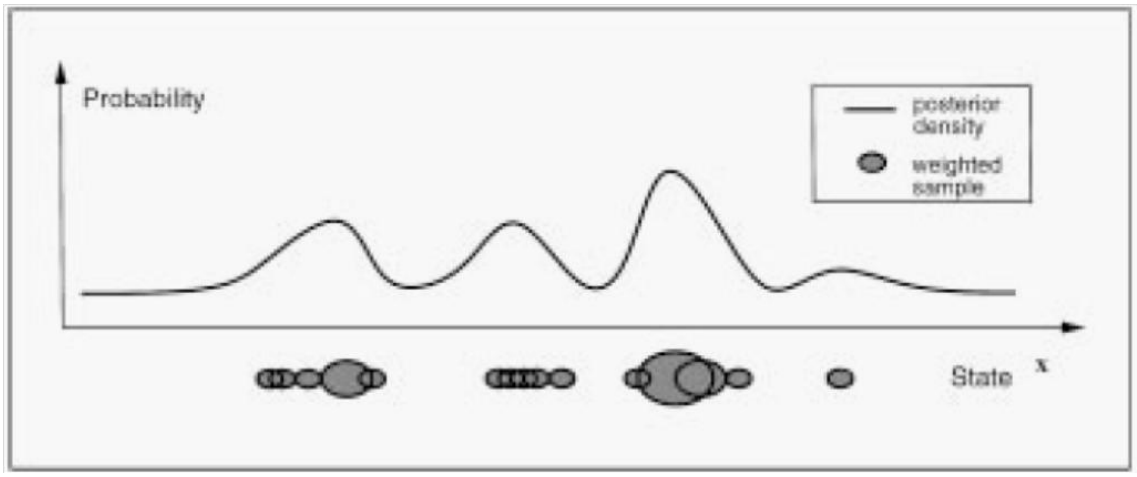


Figura 25 - Amostragem aleatória. A função contínua é amostrada e cada uma destas amostras recebe um peso igual a densidade de probabilidade do ponto correspondente. As amostras ou partículas com maior área são correspondentes a maiores densidades

O filtro de partículas faz isso recursivamente. O resultado obtido a partir e cada iteração é um conjunto de amostras ponderadas que servem de base para a próxima iteração. O ideal é que seja mantido um número fixo de partículas em cada iteração, garantindo assim um tempo de execução constante.

Os passos podem ser descritos como a seguir:

- O primeiro passo é a amostragem com reposição N vezes. Nesta etapa corre-se o risco de amostras com alto peso serem escolhidas várias vezes e amostras com baixos pesos nunca serem escolhidas.
- O segundo passo é avançar o conjunto através da dinâmica determinística do sistema, desta forma, elementos idênticos terão o mesmo resultado.
- Outro avanço é realizado pela componente aleatória do sistema, o que faz com que elementos iguais agora se divirjam realizando cada um o seu próprio movimento Browniano.
- Neste estágio a nova amostra foi gerada, porém, ainda sem pesos para cada partícula, isto é, tem-se apenas uma estimativa preditora.
- O passo da observação gera os pesos a partir da probabilidade condicionada $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$.

A Figura 27 ilustra estes passos:

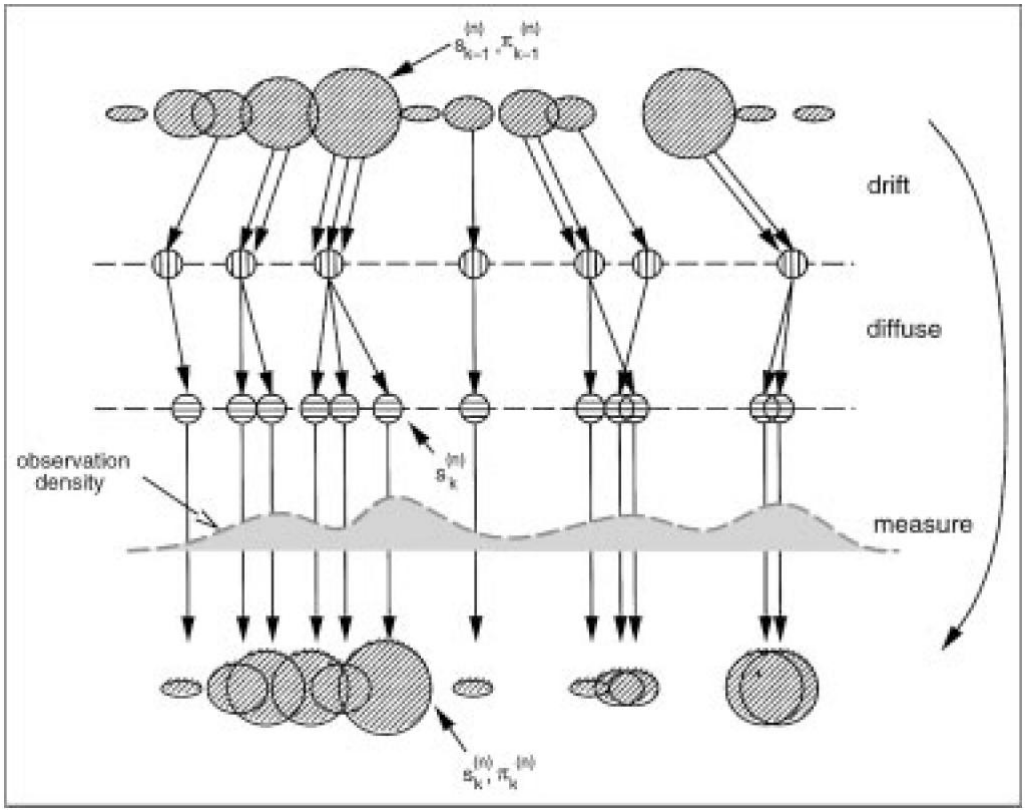


Figura 26 - Ilustração do funcionamento do filtro de partículas ou condensation algorithm

É possível observar que uma das características atrativas no filtro de partículas é sua simplicidade em relação aos outros filtros estudados, além de sua generalidade. Pode-se dizer que isto é uma consequência da ausência da equação de Riccati que é computacionalmente pesada. Estas e mais informações podem ser consultadas no texto já citado.

11.4 Simulação do filtro de partículas

O filtro de partículas foi implementado no Matlab. Para esta implementação foi escolhido um algoritmo simples através de amostragem com reposição. Outros métodos serão discutidos na próxima seção, porém não serão implementados. Segue a seguir um resumo do passo-a-passo executado pelo algoritmo:

1. Criar todas as matrizes do sistema, F, G, H, R, Q.
2. Criar uma variável que será o número de partículas que serão mantidas a cada iteração, NPAR.
3. Criar uma matriz de estados **x** (NPAR x MAXVZS x 6), para garantir que todas as partículas em todas as iterações sejam salvas.
4. Criar uma matriz de pesos (NPAR x MAXVZS) para associar o peso de cada partícula.
5. Criar uma matriz de covariância (NPAR x MAXVZS x 6 x 6) e iniciá-la como uma matriz diagonal.

6. **xs** → Estado selecionado
7. PE → Covariância estimada a priori
8. Ler o quadro e extrair a imagem (posição em x, posição em y e raio médio)
9. Caso seja a primeira iteração, deve-se criar as NPAR partículas através de uma amostragem aleatória dos pixels na tela. Utiliza-se um *rand* para gerar um número aleatório entre 0 e 1 e multiplica-se pelo tamanho da tela.
10. Como não há informações a respeito destes pixels, ou seja, podem ser tanto pontos do objeto ou do fundo, coloca-se o mesmo peso para cada partícula igual à 1/ NPAR. Netse caso, não há mais nada o que fazer e deve-se voltar para linha 8
11. Se não for a primeira iteração, o novo conjunto de partículas deve ser obtido através de amostragem no conjunto anterior. Neste caso, embora ainda seja uma amostragem aleatória, é baseada no peso de cada partícula, ou seja, partículas com maior peso têm mais chances de serem escolhidas. O trecho a seguir ilustra como isso funciona:

```
if i ~= 1
    AMOSTRAS=10;
    ident=zeros(100*AMOSTRAS,1);
    idcont=0;
    for j = 1 : NPAR
      num= floor(AMOSTRAS*100*pesos(j,i-1));
      if num > 0
        ident(idcont+1:idcont+num) = j*ones(1,num);
        idcont=idcont+num;
      end
    end
end

  for j = 1 : NCON

    if i==1
      xs = [floor(MC*rand(1)),floor(MR*rand(1)),0,0,0,0]';
    else
      k = ident(ceil(idcont*rand(1)));
      xs(1) = x(k,i-1,1);
      xs(2) = x(k,i-1,2);
      xs(3) = x(k,i-1,3);
      xs(4) = x(k,i-1,4);
      xs(5) = x(k,i-1,5);
      xs(6) = x(k,i-1,6);
    end
  end
```

AMOSTRAS indica o número médio de amostras que serão retiradas por partícula, ou seja, o vetor *indent* terá AMOSTRAS*NPAR amostras. Este vetor *ident* é um vetor auxiliar que contém amostras que serão selecionadas aleatóriamente depois. Apesar de AMOSTRAS indicar um número médio, o número de amostras que serão retirados de cada partícula é proporcional ao peso da partícula, como pode ser visto pela multiplicação na sexta linha do código. Exemplo para um caso em que NPAR=4 é mostrado a seguir:

Tabela 3- Exemplo de amostragem com probabilidade proporcional ao peso de cada partícula

Partículas	Pesos	Amostras
1	0,2	8
2	0,11	4
3	0,6	24
4	0,09	4
TOTAL	1*	40**

*Sempre será 1, visto que os pesos são normalizados
**Sempre será AMOSTRAS*NPAR

12. O próximo passo após determinar **xs** que é o estado selecionado é avança-lo no tempo através da dinâmica do sistema, ou seja, **xp = F*xs**, gerando assim a estimativa a priori. Isto deve ser feito para todos os estados selecionados.
13. Atribui peso a cada um dos estados de acordo com a observação, isto é, os estados que são mais prováveis de terem gerado a observação são atribuídos pesos maiores. Para realizar isto foi calculado um **desvio**, dado pela diferença entre a medida e a posição estimada (**H*xp**); e o peso da amostra é dado por 1/(**desvio*desvio'**). Lembrando que **desvio** não é um escalar e sim um vetor.
14. Os pesos atribuídos às partículas devem ser normalizados.
15. É identificado o maior peso, e a partícula correspondente é escolhida como a estimativa a posteriori do vetor de estado.
16. O processo recomeça em 8.

Da mesma forma que os filtros robustos implementados, foi realizada a simulação no rastreamento visual da trajetória da esfera laranja e o resultado obtido foi:

$$MSE_{med} = 48.4778$$

$$Improvement = \frac{116.6 - MSE_{med}}{116.6} \times 100\% = 58.4239\%$$

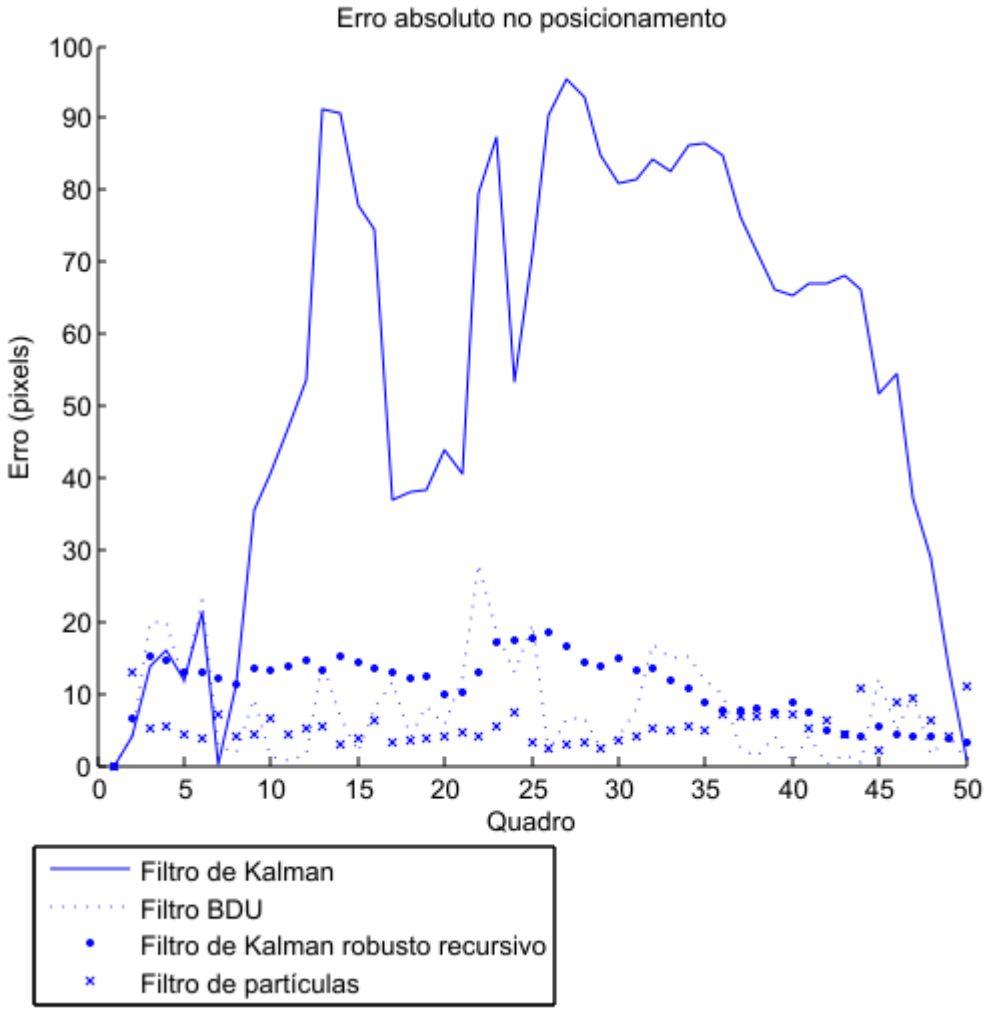


Figura 27 - Comparação dos quatro filtros implementados

O valor *116.6* refere-se ao erro quadrado médio do filtro de Kalman nominal. Nota-se que o filtro de partículas teve um resultado superior aos filtros robustos implementados com a grande vantagem de não precisar de ajuste paramétricos nem de hipótese para incertezas.

O desempenho do filtro de partículas varia conforme o número de partículas que são mantidas ao longo do tempo. No caso anterior, a simulação foi realizada com 100 partículas por iteração. Para avaliar a influência do número de partículas no desempenho do filtro foi realizado um teste com este número variando e para cada valor de NPAR foram realizados 10 simulações. A tabela a seguir mostra alguns dados obtidos deste teste:

Tabela 4 - Comportamento do filtro de partículas em relação ao número de partículas

Melhorias x Número de partículas (10 simulações cada)										
Melhoria	**10**	**20**	**40**	**50**	**80**	**100**	**160**	**320**	**500**	**640**
min	-225,60	-79,81	-39,14	12,01	38,60	-1928,00	39,51	42,70	41,21	45,68
máx	47,66	45,18	59,39	59,52	59,00	58,81	59,41	59,13	60,39	60,44
médio	-67,50	-8,09	13,05	37,33	49,65	34,00	52,78	50,30	55,22	54,81
desvio padrão	107,70	40,09	31,48	17,28	7,40	19,76	6,65	6,30	6,15	4,501
amplitude	273,20	125,00	98,53	47,51	20,40	60,74	19,90	16,43	19,18	14,76

Conforme o número de partículas aumenta o desempenho médio vai aumentando. Outra consequência deste aumento é a redução no desvio padrão e amplitude entre os resultados, o que indica um aumento de estabilidade do filtro e uma maior confiança na sua implementação.

Este resultado faz com que, embora não seja o objetivo inicial deste trabalho, cresça o interesse em estudar este filtro mais afundo e aprimorar sua aplicação. Esta implementação é uma das mais simples e algumas outras serão discutidas abaixo.

11.5 Variações do filtro de partículas

O algoritmo implementado na seção anterior pode ser chamado de SIS (Sequencial Importance Sampling- Amostragem Sequencial por Importância). No modo mais genérico pode-se ter uma função $q(x)$ que é conhecida como densidade de importância e determina a importância de cada partícula. No caso anterior foi usado simplesmente o peso de cada partícula como importância. Desta forma é recursivamente propagado um conjunto de partículas e seus respectivos pesos. Agora, observe os gráficos da Figura 29 e 30:

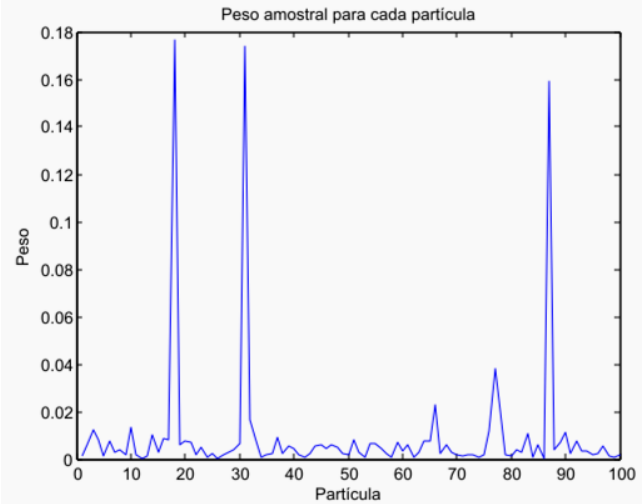


Figura 28 - Distribuição dos pesos amostrais referente ao 9º quadro do vídeo. Percebe-se que apenas 3 partículas contribuem significamente para o resultado

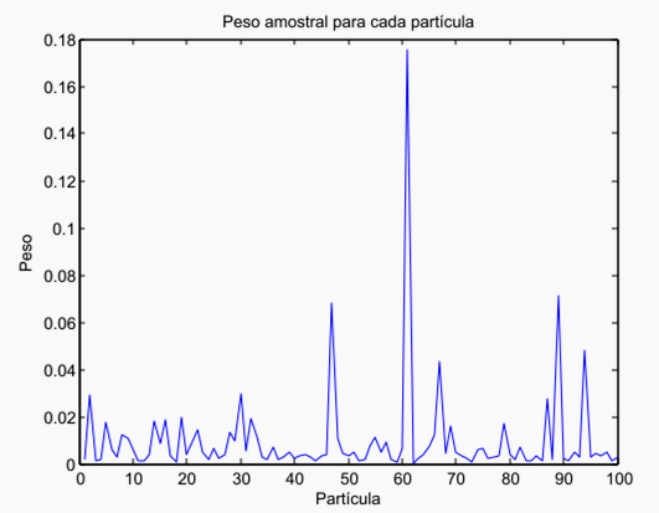


Figura 29- Distribuição dos pesos amostrais referente ao 23º quadro do vídeo. Percebe-se que apenas 1 partículas contribuem significamente para o resultado

Ambos indicam a variação do peso amostral ao longo das 100 amostras. Pode-se perceber que os gráficos apresentam alguns picos, o que pode causar degeneração. O problema de degeneração ocorre quando poucas partículas têm peso muito alto e várias outras têm peso baixo, fazendo com que muito esforço computacional seja feito para atualizar amostras que pouco contribuem para o resultado. Uma forma quantitativa para se caracterizar este efeito é dado pelo tamanho amostral efetivo:

$$N'_{ef} = \frac{1}{\sum_{i=1}^{NPAR}(p_k(i))^2}, p_k \; indica \; o \; peso \; da \; amostra \quad (46)$$

Quanto menor o valor de N'_{ef} maior a degeneração da amostra. Um valor razoável para o tamanho amostral efetivo seria superior à $^{2N}/_{3}$. Para avaliar o filtro implementado calculou-se o N'_{ef} para todos os quadros e comparou-se com o valor de *66*, segue o gráfico:

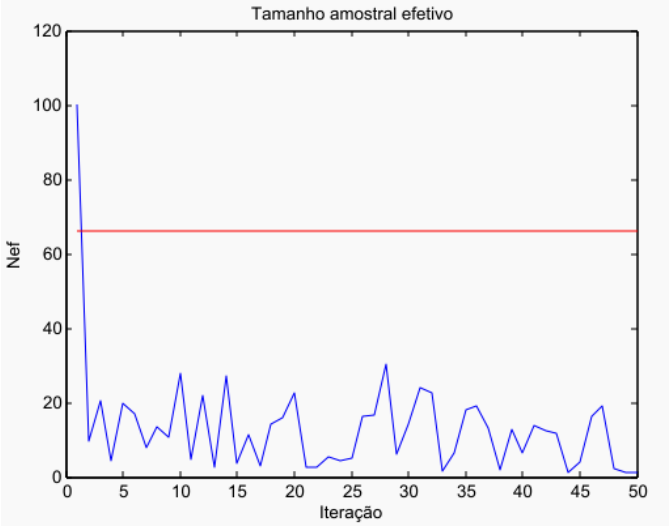


Figura 30- Tamanho amostral efetivo durante as iterações. Este gráfico mostra que o filtro sofre grande degeneração

Percebe-se que neste caso ocorre grande degeneração. Para aliviar este problema pode-se aumentar o número de amostras. Na maioria dos casos isso pode só aumentar o tempo computacional sem conduzir a nenhuma melhoria. A necessidade de reduzir a degeneração levou a composição de um novo filtro, o SIR (Sampling Importance Resampling, ou, Amostragem e Reamostragem por Importância). A ideia, neste caso, é eliminar as amostras que possuem baixo peso, através da escolha específica da função $q(x)$. A reamostragem deve ser feita toda iteração. Maiores detalhes podem ser encontrados em [32].

Eliminando-se as amostras de menor peso, e fazendo isso a cada iteração, pode acarretar uma redução da diversidade do filtro. Quando isso se torna um problema uma variação deste filtro pode ser implementada. O filtro ASIR (Auxiliary Sampling Importance Resampling, ou, Amostragem e Reamostragem por Importância Auxiliar) pode ser considerado igual ao SIR com uma reamostragem no instante de tempo anterior através de uma distribuição de probabilidade auxiliar. Para pequenos ruídos esta versão pode funcionar melhor, caso contrário se recomenda-se o uso do SIR.

12 Aplicação em Tempo Real

Um dos objetivos da continuação deste trabalho é implementá-lo em um sistema de tempo real para uma aplicação prática.

O primeiro passo para uma aplicação em tempo é gerar um código enxuto e escrito em C++, com a utilização do OPEN CV para o processamento de imagens. O Matlab, usado até o momento, é uma plataforma amigável para o estudo e desenvolvimento dos programas, mas não para uma aplicação em tempo real.

Deve ser definido o tipo de aplicação ao qual o sistema se destina, ou seja, o tipo de ambiente ao qual será inserido (outdoor, indoor), ao tipo de objeto que será seguido (formato, cor, textura, ou outra característica que defina como será a segmentação), o equipamento utilizado para o sensoriamento (câmeras, scanners, sensores infravermelhos,...).

Na continuidade deste projeto espera-se fazer a aquisição de imagens utilizando o sensor KINECT, que será instalado no helicóptero autônomo que está em funcionamento no Laboratório de Sistemas Inteligentes (LASI) na USP em São Carlos.

Figura 31 - Sensor KINECT da Microsoft

O KINECT é um periférico desenvolvido pela Microsoft para o Xbox 360 e tem como colaboradora a empresa Prime Sense. Possui *3* sensores principais:

1. Câmera RGB que permite o reconhecimento de objetos,
2. Sensor de profundidade que permite um scaneamento do ambiente em *3D*,
3. Microfone embutido

Além disso, possui processador e software próprio. Consegue detectar *48* pontos de articulação no corpo humano, o que demonstra a capacidade e precisão desta ferramenta.

Determinadas as características do sistema, o próximo passo é calcular os deadlines de cada processo do programa, isto é, os prazos para a execução de cada tarefa. Este parâmetro é o mais importante numa aplicação em tempo real e deve ser levado em consideração na hora de elaborar cada uma das funções que vão compor o software. O deadline é determinado pela frequência com que o sensor adquire imagens do alvo e deve ser respeitado correndo-se o risco de uma falha no sistema.

A frequência de captura dos quadros de vídeo, a velocidade média do alvo que se deseja seguir, a distância deste até o sensor, a velocidade média do sensor e do helicóptero devem ser levadas em consideração para se determinar o pior cenário possível. Com estes dados consegue-se estimar qual a máxima distância na imagem que o objeto pode percorrer entre um quadro e outro. Esta distância influenciará o tamanho da janela de busca do alvo.

$$\text{Área de busca}_{\text{janela}} = \text{f (distância percorrida pelo alvo entre quadros, erro de estimativa do filtro)}$$

Existem duas possíveis abordagens para determinar a área de busca do alvo na imagem. A primeira é a abordagem estática: Com base no conhecimento das distâncias envolvidas, erros nas estimativas e um fator de segurança, estabelece-se o tamanho da janela, que será constante ao longo do tempo. A abordagem dinâmica é muito semelhante à anterior, contudo, baseia-se num histórico dos últimos N quadros, calculando-se a média dos erros de estimativa e atualizando este valor constantemente. A vantagem da primeira é que não há overhead no processamento, ou seja, não há necessidade de efetuar cálculos para a atualização do tamanho da janela, enquanto no segundo, a principal vantagem é que quando o movimento for mais suave, ou mais homogêneo, pode-se obter reduções ainda maiores na área de busca, enquanto durante movimentos bruscos, há uma compensação com o aumento desta área.

Outra questão importante é a “perda” da localização do alvo. Como foi demonstrado o filtro robusto consegue manter as estimativas para incertezas dentro de uma faixa predefinida. Quando há

algum desvio que vai além desta faixa, o programa fica sujeito a perder o alvo. O que fazer nesta situação? A primeira solução é utilizar um *flag* para indicar que o alvo não foi localizado. Após a detecção da falha, o programa deve eliminar a redução da área de busca, e trabalhar com toda a imagem, facilitando a localização do alvo novamente. Se o alvo for localizado desta maneira, o programa reinicia e volta a reduzir a área de busca, caso contrário deve-se determinar outro método para encontrar o alvo, como uma varredura 360º.

Ainda resta o ajuste paramétrico do filtro, ou também chamado de *tuning*. Existem muitos parâmetros que são função do sistema e do tipo de alvo que será seguido, porém outros parâmetros são fortemente influenciados pelo ambiente. Um trabalho futuro poderia ser, através de cálculos de derivadas parciais, encontrar a sensibilidade do desempenho do filtro em relação a cada uma das matrizes envolvidas e quais são mais susceptíveis ao ambiente. Assim, pode-se alocar mais processamento no ajuste de parâmetros que irão influenciar mais o resultado do filtro.

O problema destas ações, como a área de busca dinâmica e o procedimento de relocalização do alvo, e o ajuste on-line das incertezas é que os ciclos de execução do programa passam a ter períodos diferentes entre si, deixando de ser um sistema determinístico. Fica impossível assim considerar previsões dos tempos de execuções e deadlines de cada tarefa.

12.1 Oclusões, múltiplos objetos e ambiente tumultuado

Um problema comum em aplicações de tempo real são as oclusões, isto é, quando o alvo se esconde atrás de um outro objeto. Outro problema é realizar o rastreamento visual em ambientes tumultuados, ou seja, com altos níveis de ruídos e vários objetos se confundindo com o alvo. Além do desenvolvimento de um filtro robusto, é interessante desenvolver um algoritmo de identificação do alvo que também seja robusto. Com o avanço tecnológico dos sensores e câmeras é importante desenvolver sistemas capazes de aproveitar ao máximo o grande número de informações que estes dispositivos oferecem e ao mesmo tempo serem computacionalmente eficientes.

Foram abordadas algumas técnicas de segmentação de imagens, identificação de objetos e extração de informação. Não serão introduzidos novos conceitos aqui, apenas refinados os já existentes e com uma abordagem que mostra uma maneira de conciliar técnicas distintas de forma a obter melhores resultados. Assim torna-se possível combinar as ferramentas de forma à alcançar o resultado de que o parágrafo anterior comenta.

Em relação as ferramentas de identificação e localização de objetos desenvolvidas até o momento, podem ser feitas algumas observações: trabalham com imagens isoladas e não com um vídeo, deve-se aproveitar as características de movimento do alvo para auxiliar a sua própria

identificação; necessitam de um modelo detalhado do alvo para efetuar a segmentação; precisam que vários pixels estejam no alvo.

De forma a superar estes desafios pode-se combinar técnicas antagônicas para obter um algoritmo robusto de alto desempenho, ver [33] e as referências contidas nela. Um resumo do trabalho é mostrado aqui.

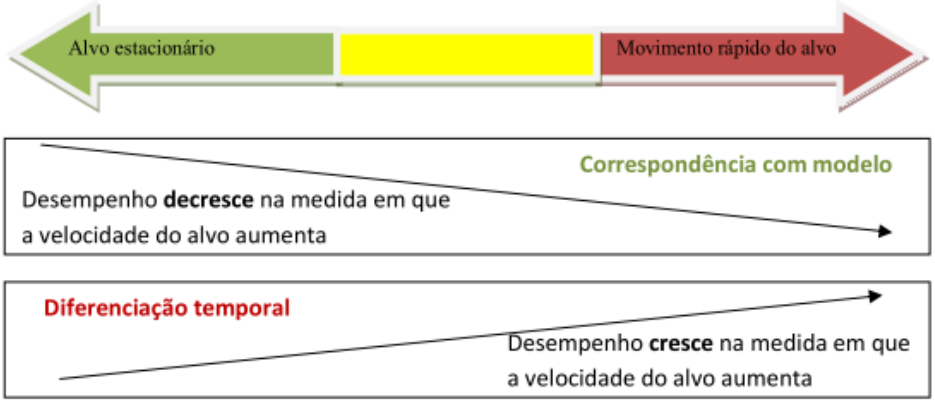


Figura 32 - Desempenho dos algoritmos de localização dos alvos em relação à velocidade do alvo

Correspondência com modelo consiste em comparar o alvo encontrado com um modelo já conhecido do alvo. Quando usado sozinho pode falhar quando a aparência do alvo varia conforme a luminosidade e ângulo de visão. Não é robusto em relação a orientação e tamanho do objeto. Sua aplicação é interessante quando o alvo está a baixas velocidades.

Diferenciação temporal é bem similar a subtração de fundo já abordada. Trata-se da comparação de um quadro num instante $t + \delta t$ com o quadro anterior, ou seja, no instante t. O método mais simples é uma simples subtração das intensidades dos pixels correspondentes e uma limiarização para classificar o pixel como pertencente ou não ao objeto. Limitado quando há movimento da câmera, fundos não estático e oclusões do alvo. Outro problema comum com esta técnica é que a comparação dos quadros gera duas regiões, uma onde o alvo está localizado e uma onde o alvo estava localizado. Com o modelo é possível eliminar este problema.

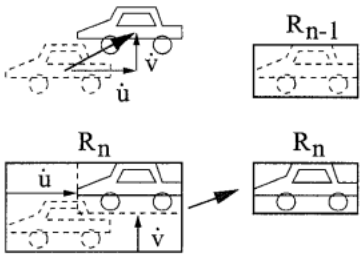


Figura 33 - Ilustração do problema na diferenciação temporal. Deve distinguir entre posição atual do objeto e posição passada do objeto

Combinando as duas técnicas, tem-se o seguinte algoritmo:

- Aplica-se a diferenciação temporal
- Identifica regiões em movimento: Utiliza-se o conceito de componentes conectados ou regiões conectados, ver capitulo *4*.
- Classificação: É importante para o rastreamento visual de múltiplos objetos. É necessário obter alguma chave identificadora do objeto. Viu-se que a cor pode ser um bom parâmetro se o objeto se destacar do fundo da imagem e houver pouca variação de luminosidade. Em aplicações práticas são requisitos difíceis de se obter. Os sete primeiros momentos de uma imagem são normalmente um bom identificador para o objeto pois são invariantes quanto a escala, rotação, translação... porém também são sensíveis à variação de luminosidade e mais importante, necessitam que muitos pixels estejam no alvo. O identificador proposto em [33] é conhecido como dispersão e é igual à razão do perímetro ao quadrado pela área. Com esse identificador o autor pretende classificar objetos em categorias como veículos, humanos e ruídos.
- Criação de modelos: após identificar as regiões em movimento, cria-se um modelo para esta região em armazena num vetor de potenciais alvos. O identificador de cada região é armazenado num vetor de hipóteses.
- Correspondência por modelo: Compara-se o vetor de potenciais alvos com o vetor anterior. Se existirem regiões que estavam no vetor e não se encontram mais, são considerados ruídos ou transientes e são eliminados. Se existir regiões que não estavam no vetor e agora estão, são tratados como potenciais alvos e são adicionados ao vetor.

O algoritmo proposto desta forma lembra um filtro de partículas, onde se geram a cada iteração possíveis alvos e através da medida selecionam as mais prováveis. Consegue-se assim lidar com oclusões e múltiplos alvos de forma satisfatória.

13 Conclusão

A adição de incertezas no modelo em espaço de estado para o rastreamento visual de trajetória de alvos móveis resultou em uma queda no erro médio quadrático entre estimativas e valores reais de posição e velocidade. Foi possível comparar os dois filtros robustos implementados e perceber as vantagens do filtro robusto sob a metodologia das funções penalidades. Esta versão, em relação ás apresentadas comumente na literatura, apresenta várias vantagens como a possibilidade de considerar incertezas em todas as matrizes, organização em blocos matriciais e a mais importante de todas, não requer ajuste de parâmetros adicional.

Além de explorar o campo da filtragem robusta, foi possível entrar em contato com os algoritmos genéticos, que se mostraram eficientes na otimização dos parâmetros e incertezas, que de outra forma seria inviável. Além disso, permitem variações na implementação que podem melhorar a busca, como variar as taxas de cruzamento e mutação, usar o método do torneio ao invés do método da roleta, alterar o tamanho da população, entre outros.

Quanto ao rastreamento visual de trajetórias, unem-se dois importantes resultados. O primeiro obtido na primeira etapa da pesquisa, que mostrou a vantagem do uso da filtragem no rastreamento visual do movimento dos alvos, e o segundo, obtido nesta etapa, que mostra a vantagem de se considerar filtros robustos para tal tarefa. Para uma aplicação, o uso de filtros robustos implica em movimentos mais suaves de uma câmera e menores riscos de se “perder" a localização do alvo.

Outro resultado promissor foi o obtido com o uso dos filtros de partículas. Apesar de não ter sido o foco deste trabalho, o primeiro teste com o uso deste filtro mostrou resultados superiores ao encontrado com os demais filtros baseados em Kalman. É necessário um estudo mais aprofundado destes filtros para obter resultados ainda melhores.

Do ponto de vista geral, este projeto possibilitou uma comparação entre diferentes métodos de filtragem, mais abrangente do que a encontrada na literatura, que compara apenas a versão nominal do filtro de Kalman com o filtro BDU. Espera-se em trabalhos futuros, acrescentar-se incertezas às matrizes Q e R e gerar um resultado ainda mais abrangente.

Referências

[1] BIANCO, A.; ISHIHARA, J. Y.; TERRA, M. H. Optimal robust prediction for general discrete-time singular systems. In: INTERNATIONAL CONFERENCE ON CONTROL, AUTOMATION, ROBOTICS AND VISION, 10., 2008, Hanoi. **Proceedings…** New York: IEEE, 2008.

[2] BROWN, R. G.; HWANG, P. Y. C. **Introduction to random signal and applied Kalman filtering**: with Matlab exercises and solutions. 3. ed. New York: John Wiley & Sons, 1997.

[3] COLLINS, R.T.; YANXI LIU; LEORDEANU, M. Online selection of discriminative tracking features. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, New York, v. 27, n.10, p. 1631-1643, 2005.

[4] COMANICIU, D.; RAMESH, V.; MEER, P. Kernel-based object tracking. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, New York, v. 25, p. 564-577, 2003.

[5] DOMINGUEZ, S. M.; KEATON, T.; SAYED, A. H. A robust tracking method for multimodal wearable computer interfacing. **IEEE Transactions on Multimedia**, New York, v. 8, n. 5, p. 956-972, 2006.

[8] LUENBERGER, D. G. **Linear and Nonlinear Programming**. Dordrecht: Kluwer Academic Publishers, 2003.

[9] SHEN Y.; GONG H.; XIONG Y. Application of adaptive extended kalman filter for tracking a moving target. **Computer Simulation**, Beijing, v. 24, p. 210-213, 2007.

[10] TAO M.; LU, Z.; YUAN, W. Geometrically invariant feature points detection based on scale space theory. **Acta Electronica Sinica**, Beijing, v. 12, p. 2564-2568, 2006.

[11] TISSAINAYAGAMA, P.; SUTER, D. Object tracking in image sequences using point features. **Pattern Recognition**, Amsterdam, v. 38, n. 1, p. 105-113, 2005.

[12] YU, W. et al. A new framework of moving target detection and tracking for uav video application. In: THE INTERNATIONAL ARCHIVES OF THE PHOTOGRAMMETRY, REMOTE SENSING AND SPATIAL INFORMATION SCIENCES, 37., 2008, Beijing. **Proceedings…** [S. n.]: ISPRS, 2008. p. 609-614.

[13] ZENG, W.; ZHU, G.; LI, Y. Point matching estimation for moving object tracking based on kalman _lter. In: IEEE/ACIS INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION SCIENCE, 8., Shanghai, 2009. **Proceedings…** Shanghai: IEEE, 2009.

[14] ZHAO, L.; CHEN, Q. Implementation of vehicle detection and tracking based on Kalman filter. **Electronic Measurement Technology**, Beijing, v. 2, p. 165-168, 2007.

[15] http://www.tecgraf.puc-rio.br/~mgattass/ra/trb09/Guilherme/VisaoComputacional%20-%20Trabalho%201.html

[16] R. G. Brown and P.Y.C. **Hwang. Introduction to Random Signals and Applied Kalman Filtering**. 2. ed. New York: John Wiley & Sons, 1992.

[17] GONZALES, R. C.; WOODS, R. E. **Processamento Digital de Imagens**. 3. ed. São Paulo: Pearson, 2010.

[18] JACOBS, O. L. R. **Introduction to Control Theory**. 2. ed. Oxford: Oxford University Press, 1993.

[19] KALMAN, R. E. A new approach to linear filtering and prediction problems. **Journal of Basic Engineering**, New York, p. 35-45, mar. 1960.

[20] MAYBECK, P. S. **Stochastic models, estimation, and control**. Amsterdam: Academic Press, 1979. 1 v.

[21] MCIVOR, A. M. **Background subtraction techniques**. Reveal Ltd PO Box 128-221, Remunera, Auckland, New Zeland.

[22] WELCH, G.; BISHOP, G. **An introduction to the Kalman filter**. TR 95-041. Department of Computer Science University of North Carolina at Chapel Hill, NC 27599-3175, page 16, july, 24 2006.

[23] YILMAZ, A.; JAVED, O.; SHAH, M. Object tracking: a survey. **ACM Computing Surveys**, New York, v. 38, n. 4, art. 13, dec. 2006.

[24] BIANCO, A. F. **Filtros de Kalman robustos para sistemas dinâmicos singulares em tempo discreto**. 129 f. 2009. Tese (Doutorado em Engenharia Elétrica) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2009

[25] CAMPOS, J. C. T. **Filtragem robusta para sistemas singulares discretos no tempo**. 137 f. 2004. Tese (Doutorado em Engenharia Elétrica) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2004.

[27] INOUE, R. S.; TERRA, M. H. Robust recursive kalman filtering for attitude estimation. In: WORLD CONGRESS OF THE INTERNATIONAL FEDERATION OF AUTOMATIC CONTROL, 18., 2011, Milano. **Proceedings…** Laxenburg: IFAC, 2011.

[28] ISHIHARA, J. Y.; TERRA, M. H.; BIANCO, A. F. Recursive linear estimation for general discrete-time descriptor systems. **Automatica**, Amsterdam, v. 46, n. 4, p. 761-766, apr. 2010.

[29] SAYED, A. H. A framework for state-space estimation with uncertain models. **IEEE Transaction on Automatic Control**, New York, v. 46, n. 7, p. 998-1013, jul. 2001.

[30] RAMAN K. MEHRA. On the identification of variances and adaptive Kalman Filtering. **IEEE Transaction on Automatic Control**, v.15, n. 2, p. 175-184, apr. 1970.

[31] M. Isard and A. Blake. Condensation–conditional density propagation for visual tracking. Int. J. Computer Vision, 29(1):5–28, 1998.

[32] CAMPOS, V. A. F. ; SANTANA, D. D. S. ; FURUKAWA, C. M. ; MARUYAMA, N. . **FILTROS DE PARTÍCULAS APLICADOS À ESTIMAÇÃO DE TRAJETÓRIAS**. In: XV Congresso Brasileiro de Automática, 2004, Gramado, RS. Anais do CBA 2004, 2004.

[33] Alan Lipton, Hironobu Fujiyoshi, and Raju Patil, "Moving target classification and tracking from real-time video," Proc. of the Workshop on Application of Computer Vision, October, 1998, pp. 8 - 14.

APÊNDICE A

FUNÇÕES DO MATLAB

```
function [ G ] = thresholding( F , hmin, hmax  )
%thresholding transforma uma imagem HSV em uma imagem binária.
%   A cor branca representa o objeto desejado e a cor preta
%   todo o resto da imagem.

H = double (F(:,:,1)); % H é o hue da imagem
[x,y]= size (H);
for i=1:x
    for j=1:y
        if ((hmin < H(i,j))&&(H(i,j)<hmax))
            H(i,j)= 255;
        else
            H(i,j)= 0;
        end
    end
end
G = H;
end
```

Figura 34 - Função limiarização ou thresholding

```
%Este pequeno trecho permite o usuário criar uma máscara que
%contenha o alvo. A partir dessa máscara calcula-se o vetor
%RGB médio representativo do alvo e também os desvios
%padrões para cada componente.

f = frame(:,:,:); % Toma o primeiro quadro como referência
mask = roipoly (f); % Permite ao usuário criar a máscara
red = immultiply (mask,f(:,:,1)); % Isola a cor vermelha
green = immultiply(mask,f(:,:,2)); % Isola a cor verde
blue = immultiply(mask,f(:,:,3)); % Isola a cor azul
g = cat(3,red,green,blue); % concatena os três componentes e
figure,imshow(g); %exibe o alvo para o usuário

[M,N,K]= size(g); % Encontra as dimensões da máscara
I = reshape(g,M*N,3); % Redimensionamento
idx = find (mask); % Encontra os elementos não-nulos da máscara
I = double (I(idx,1:3));
C = cov (I); % Calcula a matriz covariância

d = diag(C); % Isola a diagonal da matriz covariância
sd = sqrt (d)'; % Cálcula o desvio padrão
m = mean(I); % Calcula os valores médios para cada componente.
```

Figura 35 - Identificação do alvo pelo usuário

```
function I = colorseg(varargin)
% Esta função foi retirada do livro: Gonzales and Woods,
% Digital Image Processing, 2nd Ed.
% S= COLORSEG('EUCLIDEAN", F, T, M) realiza a segmentação
% da imagem colorida F usando a medida de distância
% euclideana. M é o vetor [1x3] representando a média de
% cores da imagem. T é o parâmetro usado para
% comparar com as distâncias encontradas.
% S= COLORSEG('MAHALANOBIS', F, T, M, C) realiza a
% segmentação da imagem colorida F usando a medida de
% distância Mahalanobis. C é a matriz covariância [3x3]
% da amostras de cores do objeto de interesse.

f = varargin{2}; % Recebe a imagem
if (ndims(f) ~= 3)||(size(f,3)~= 3)
    error('input image must be RGB.');
end

M = size(f,1); % Cálcula as dimensões
N = size(f,2);
f = imstack2vectors (f);
f = double (f);

I = zeros(M*N,1);
T = varargin{3};
m = varargin{4};
m = m(:)';

if length(varargin)==4
    method = 'euclidean';
elseif length(varargin)==5
    method = 'mahalanobis';
else
    error('Wrong number of inputs.');
end

switch method
% Cálcula a distância entre todas as linhas de X e m.

    case 'euclidean'
        p = length(f);
        D = sqrt(sum(abs(f-repmat(m,p,1)).^2,2));
    case 'mahalanobis'
        C = varargin{5};
        D = mahalanobis(f,C,m);
    otherwise
        error('unknow segmentation method.')
```

Figura 36 - Função que realiza a segmentação da imagem em RGB (Parte I)

```
end

% Verifica os pontos onde a distância é menor que T.
J = find(D<=T);
I(J) = 1 ;
I = reshape(I,M,N);

end
```

Figura 37 - Função que realiza a segmentação da imagem em RGB (Parte II)

```
function [ X, R ] = imstack2vectors( S, MASK )

[M,N,n]= size(S);
if nargin==1
    MASK = true(M,N);
else
    MASK = MASK ~= 0 ;
end

R = find (MASK);

Q = M*N;
X = reshape(S,Q,n);
MASK = reshape(MASK,Q,1);
X = X(MASK,:);
end
```

Figura 38 - Função imstack2vectors

```
function [xbar,ybar] = posicao(F)
% Esta função retorna a posição do centróide de uma imagem
% em escala de cinza ou binária, utilizando os momentos
% m00, m10, m01.
% Foi baseada no livro: Gonzales and Woods, Digital Image
% Processing, 2nd Ed.

if (ndims(F) ~= 2)|| issparse(F) || ~isreal(F)|| ...
        ~(isnumeric(F) || islogical(F))
    error (['F must be a 2-D, real, nonsparse, numeric or'...
        'logical matrix.']);

end
F = double(F);


m = compute_m(F);
xbar = m.m10/m.m00;
ybar = m.m01/m.m00;

end
%-----------------------------------------------------------------
    function m = compute_m(F)

    [M,N] = size(F);
    [x,y] = meshgrid(1:N, 1:M);

    % Transforma x,y and F em um vetor coluna para deixar a
    % somatória um pouco mais fácil de ser efetuada
    x = x(:);
    y = y(:);
    F = F(:);

    % Cálcula a área
    m.m00 = sum(F);

    % Não permite a divisão por zero.
    if (m.m00 == 0)
        m.m00 = eps;
    end

    %Os outros momentos centrais são:

    m.m10 = sum (x .* F);
    m.m01 = sum (y .* F);


    end
    %-----------------------------------------------------------------
```

Figura 39 - Função que retorna a posição do "centro de massa" de uma imagem binária

```
tic


% Definições do sistema

R = 0.1*eye(2);
H = [1 0 0 0 0 0
     0 1 0 0 0 0];
G = eye(6);
Q = 2.3*eye(6);
P = eye(6);
T =1;
F= [1  0  T  0  0.5*(T^2)  0,
     0  1  0  T  0  0.5*(T^2),
     0  0  1  0  T  0  ,
     0  0  0  1  0  T  ,
     0  0  0  0  1  0  ,
     0  0  0  0  0  1  ];
x=zeros(50,6);

[X,Y,DIM] = size(framergb(:,:,:,1));


% Extrair alvo
  [z(1),y(1),rm]= extrairalvo(framergb(:,:,:,1),3,m);
  rmax = 2.5*rm;

% Inicializa o filtro de Kalman
    x(1,:) = [z(1),y(1),0,0,0,0];
    xp = F*x(1,:)';
    prev(:,1) = xp;
    PP = F*P*F' + G*Q*G';

% Percorre os outros quadros

for i = 2 : 50

  % Extrai alvo com redução da janela de busca

  a = round(max(xp(1)- rmax,0));
  b = round(min(xp(1)+ rmax,X));
  c = round(max(xp(2)- rmax,0));
  d = round(min(xp(2)+ rmax,Y));

  [z(i),y(i),rm]= extrairalvo(framergb(c:d,a:b,:,i),3,m);
  z(i)= z(i)+a;
  y(i)= y(i)+c;


  % Kalman: Equações de correção
```

Figura 40 - Programa principal realiza o rastreamento visual de trajetória do alvo (Parte I)

```
  K = PP*H'*inv(H*PP*H'+R);
  x(i,:) = (xp + K*([z(i),y(i)]' - H*xp))';
  P = (eye(6)-K*H)*PP;

  % Kalman: Equações de avanço temporal

  xp= F*x(i,:)';
  prev(:,i) = xp;
  PP = F*P*F' + G*Q*G';


end

toc
```

Figura 41 - Programa principal, realiza o rastreamento visual de trajetória do alvo (Parte II)

```
function [ x, y,rm ] = extrairalvo( im,ee,m )
% Esta função extrai a posição do alvo e o raio médio
% O raio médio pode ser usado como referência para o cálculo
% do tamanho da janela de redução de busca

% Realiza a segmentação
E32 = colorseg('euclidean',im,32,m);
% Realiza a erosão
se = strel ('diamond',ee);
E = imerode (E32,se);
% Cálcula as propriedades básicas dos componentes conectados
% na imagem binária E.
prop = regionprops(E,['basic']);
  [N,W] = size(prop);

% Cria-se um vetor e classifica as regiões conectadas
% por ordem de tamanho neste vetor
  id = zeros(N);
  for i = 1 : N
    id(i) = i;
  end
  for i = 1 : N-1
    for j = i+1 : N
      if prop(i).Area < prop(j).Area
        tmp = stats(i);
        prop(i) = prop(j);
        prop(j) = tmp;
        tmp = id(i);
        id(i) = id(j);
        id(j) = tmp;
      end
    end
  end

  % O método usado para conseguir o centróide
  % é identico ao algoritmo de momentos.
   centroid = prop(1).Centroid;

  % A área é obtida contando-se os pixels que
  % pertencem ao objeto de interesse

  rm = sqrt(stats(1).Area/pi);
  x = centroid(1);
  y = centroid(2);

end
```

Figura 42 - Função que extrai o alvo da imagem, retornando sua posição e o raio médio

```
% Código do Filtro BDU no Matlab

P1 =;
P2 =;
P3 =;
P4 =;
P5 =;

T = P1; %Taxa de captura dos frames no vídeo em seg/quadro
F = [1  0  T  0  0.5*(T^2)  0,
     0  1  0  T  0  0.5*(T^2),
     0  0  1  0  T  0  ,
     0  0  0  1  0  T  ,
     0  0  0  0  1  0  ,
     0  0  0  0  0  1  ];
G =  eye (6);
H =  [1 0 0 0 0 0
     0 1 0 0 0 0];
Q =  P2*eye(6);
R =  P3*eye(2);
Mf = [0.1 0.2 0.1 0.1 0.1 0.1]';
Nf = [0 0 1 1 1 1];
Ng = [1 1 0 0 0 0];


%Condições iniciais

[X,Y,DIM] = size(framergb(:,:,:,1));
tic
[s(1),y(1),rm]= extrairalvo(framergb(:,:,:,1),3,m);
rmax = P4*rm;
z = [s(1);y(1)];
Pp = Q;
P = inv(inv(Pp) + H'*inv(R)*H);
x_hat = [z(:,1) ; 0 ; 0; 0 ; 0];
  i=1;
  x(:,i)= x_hat;
  xp(:,i) = x_hat;
alpha = P5;

%Passo1
if (H*Mf) == 0
    lambda = 0;
else
    lambda = (1+ alpha )*norm(Mf'*H'*inv(R)*H*Mf);
end

%Percorre as imagens

for i = 2 : 50
%Cálculo das matrizes corrigidas levando-se em conta o lambda
```

Figura 43 - Implementação do filtro BDU (Parte I)

```
Q_hat = inv(inv(Q) + lambda*Ng'* ...
                          (inv(eye(1)+ lambda*Nf*P*Nf'))*Ng);
R_hat = R - inv(lambda)*H*Mf*Mf'*H';
P_hat = inv(inv(P) + lambda*Nf'*Nf);
G_hat = G - lambda*F*P_hat*Nf'*Ng;
F_hat = (F - lambda*G_hat*Q_hat*Ng'*Nf)*...
                              (eye(6)- lambda*P_hat*Nf'*Nf);
%Equação de predição
xp_hat = F_hat*x_hat;
xp(:,i) = xp_hat;
Pp = F*inv(P_hat)*F' + G_hat*Q_hat*G_hat';
Re = R_hat + H*Pp*H';
%Extração da posição utilizando redução da área de busca
  a = round(max(xp_hat(1)- rmax,0));
  b = round(min(xp_hat(1)+ rmax,X));
  c = round(max(xp_hat(2)- rmax,0));
  d = round(min(xp_hat(2)+ rmax,Y));
  [s(i),y(i),rm]= extrairalvo(framergb(c:d,a:b,:,i),3,m);
  s(i)= s(i)+a;
  y(i)= y(i)+c;
  z(:,i) = [s(i);y(i)];

%Equação de correção
e = z(:,i) - H*xp_hat;
P = Pp - Pp*H'*inv(Re)*H*Pp;
x_hat = xp_hat + P*H'*inv(R_hat)*e;
 x(:,i) = x_hat;

end

toc

%Medidores de Desempenho
MSEx = 0;
MSEy = 0;
MSEdx = 0;
MSEdy = 0;

%real é uma vetor que contém as posições verdadeiras da esfera
for i=2:49
    MSEx = MSEx + (real(1,i)-xp(1,i))^2;
    MSEy = MSEy + (real(2,i)-xp(2,i))^2;
    MSEdx = MSEdx + ((real(1,i+1)-real(1,i))- xp(3,i)*T)^2;
    MSEdy = MSEdy + ((real(2,i+1)-real(2,i))- xp(4,i)*T)^2;
end

MSEx = MSEx/48;
MSEy = MSEy/48;
MSEdx = MSEdx/48;
MSEdy = MSEdy/48;
MSEmed = (MSEx + MSEy + MSEdx + MSEdy)/4;
```

Figura 44 - Implementação do filtro BDU (Parte II)

APÊNDICE B

DETALHAMENTO DO ALGORITMO GENÉTICO

O algoritmo genético foi implementado usando um recurso do Matlab conhecido como *"varargin"*. É uma lista de argumentos com tamanho variável, isto permite que o código seja usado tanto para o filtro BDU quanto para o robusto recursivo ótimo, e além disso, pode ser usado para testar apenas algumas matrizes incertezas ou outros parâmetros do filtro.

O programa é composto por várias funções como, *gerapop()*, *fitness()*, *improvement()*, *selecao()*, *prole()*, *crossover()*, *mutacao()*.

A primeira função, *gerapop()*, tem como objetivo criar uma população de *n* indivíduos aleatórios conforme as regras especificadas. O resultado desta função é uma cadeia de caracteres *1s* e *0s* (genes) e espaços em branco. Os espaços em branco separam o trecho do "DNA" que codifica um parâmetro de outro, de tal forma que cada indivíduo é formado por um número variável de trechos em que cada um é responsável por um parâmetro do problema a ser otimizado. Sendo assim, a função *gerapop()* recebe um argumento indicando o tamanho da população, ou seja, quantos indivíduos aleatórios serão gerados. Para cada parâmetro a ser otimizado, esta função deve receber mais *4* argumentos. O primeiro indica o número de genes que este parâmetro terá, ou seja, no caso do parâmetro M_f serão *6* termos, pois é uma matriz *6x1*. Os outros dois argumentos definem a faixa de valores em que cada um destes termos poderá variar e o último define a precisão. Para cada parâmetro, estes *4* argumentos, servem apenas para mostrar ao programa qual é a faixa de valores em binário que este parâmetro vai ocupar.

Um exemplo do uso da função *gerapop()* é o seguinte:

```
gerapop(5,2,0,1,0.1,1,1,5,0.2)
Será gerada uma população com 5 indivíduos. Cada indivíduo terá codificado 2 parâmetros
(note que para cada parâmetro possui 4 argumentos). O primeiro parâmetro possui 2 termos
que vão de 0 a 1 com precisão de 0.1, já o segundo parâmetro possui apenas 1 termo
que vai de 1 a 5 com precisão de 0.2. O resultado é:

ans =

10001001 00111
10010110 00110
00100101 11000
10010001 11000
10010100 10101
```

Figura 45 - Função gerapop usada para criar uma população

A segunda função, *fitness()*, transforma a codificação binária nos respectivos valores de cada termo e os organiza de acordo com cada parâmetro. Para isso ela recebe os mesmos parâmetros que a função *gerapop()* mais a população propriamente dita. Ainda em *fitness()* é feita uma chamada para a função *improvement()* que avalia o desempenho deste indivíduo em termos da melhora em relação ao filtro nominal. A função *improvement()* é na verdade o filtro robusto, que recebe as matrizes de incertezas e compara o erro com o nominal. Usando-se este desempenho como fator de seletividade a função *selecao()* escolhe os indivíduos mais aptos entre a população.

```
fitness(ans,5,2,0,1,0.1,1,1,5,0.2)
Transforma os valores binários em decimais
0.8 0.9     1.4
0.9 0.6     1.2
0.2 0.5     4.8
0.9 0.1     4.8
0.9 0.4     4.2
e chama a função improvement()
```

Figura 46 - Função fitness que avalia o desempenho do filtro para um determinado conjunto de incertezas

A seleção é feita através do método da roleta. Para isso a função *selecao()* recebe, além da população e o desempenho de cada individuo, dois parâmetros *"nr"* e *"ns"*. No método da roleta um certo número de indivíduos, *"nr"*, são colocados em uma faixa de um disco com área proporcional ao seu desempenho relativo. A roleta é girada *"ns"* vezes, e em cada uma destas um indivíduo é selecionado. Assim os melhores indivíduos são mais prováveis de serem escolhidos e ao mesmo

tempo mantém-se uma certa variabilidade, á medida que indivíduos não tão bons podem ser selecionados com uma probabilidade menor.

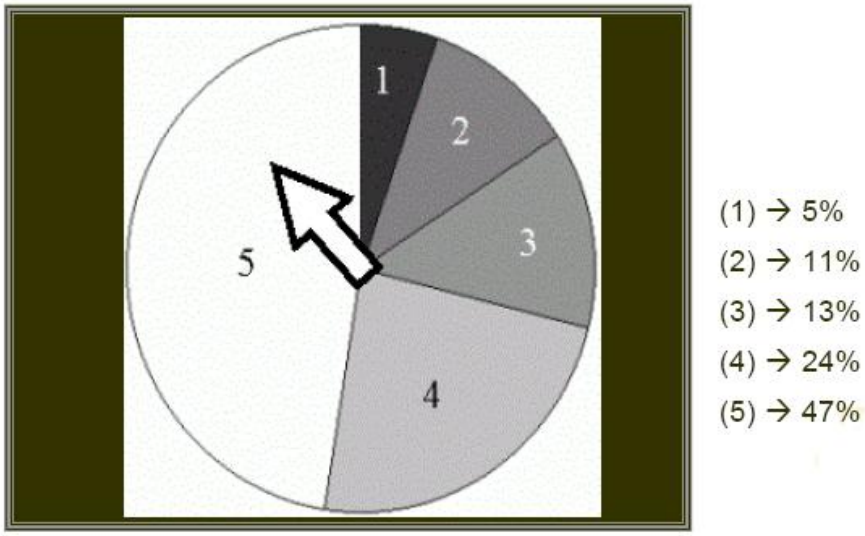


Figura 47 - Esquema do funcionamento de uma roleta, onde pode ser visto que cada área é proporcional ao desempenho relativo do indivíduo correspondente

Com esses indivíduos selecionados será construída uma nova geração com o mesmo número de indivíduos da geração atual. A função responsável por isso é a função *prole()*, que recebe como argumentos, os indivíduos selecionados, o número de indivíduos selecionados, a taxa de cruzamento e mutação e o número de indivíduos que serão gerados.

Esta função funciona da seguinte maneira:

- Sorteia-se um “pai” entre os indivíduos.

- Sorteia-se um número aleatório de *0* a *1* e compara-se este número com a probabilidade de cruzamento.

- Caso seja menor: soteia-se outro “pai” para realizar o cruzamento. O cruzamento é realizado através da função *crossover()*.

- Caso seja maior: não há cruzamento, sendo o novo indivíduo gerado através da mutação dos genes do “pai".

A função *crossover()*, recebe os dois pais e através de uma máscara de *0s* e *1s* gera um descendente com parte do material genético do primeiro “pai” e parte do segundo. A função *mutacao()* recebe o “pai” e o percorre, sendo que para cada bit é sorteado um número aleatório que sendo menor que a probabilidade de mutação irá trocar o bit. Caso contrário o bit continuara inalterado.

Desta forma a nova população é gerada e o processo se repete. Pode ser definido um critério de parada para o programa, ou quando não há mais evolução no desempenho entre uma geração e outra, ou quando um certo número de gerações é atingido. Em cada geração o melhor individuo é tomado como referência para avaliar o comportamento do algoritmo genético ao longo do tempo.