

Rastreamento de Múltiplos

Objetos em Tempo Real

THIAGO DA ROSA DE BUSTAMANTE

¹UFMG - Universidade Federal de Minas Gerais, Av. Antônio Carlos, 667 – Pampulha - Belo Horizonte, MG, Brasil
trb@dcc.ufmg.br

Resumo. Este trabalho concentra-se no problema de rastreamento de múltiplos objetos, a partir de imagens. Ele mostra uma implementação simples, que considera muito o tempo como um fator para análise dos resultados.

1 Introdução

Rastreamento de objetos é um problema clássico em visão computacional e possui aplicação em diversas áreas. Neste trabalho, estamos preocupados com a captura de movimentos realizados por pessoas (tipicamente exercícios executados por atletas).

Os movimentos que despertam o nosso interesse são movimentos muito complexos (não descrevem nenhuma função matemática conhecida) e muito rápidos. Possuem grandes variações de velocidade ao longo de sua trajetória.

Em função destas características, estes movimentos são de grande interesse, pois não se consegue, sem uma ferramenta poderosa, obter informações sobre o movimento, de forma que se possa fazer uma análise de sua correitude.

Isto torna particularmente difícil a vida dos treinadores e dos atletas, que precisam de um feedback para os exercícios realizados. Este feedback precisa ser o mais rápido possível, enquanto o atleta ainda possui na memória o exercício que acabou de realizar.

Em alguns casos, o feedback precisa ser em tempo real, para que a pessoa que está executando possa ser corrigida pelo treinador (ou médico, etc.).

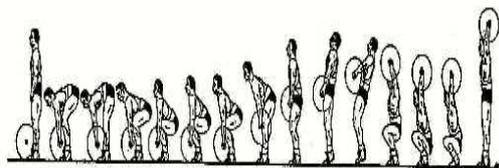
2 Estudo de caso: Arranco

Trabalharemos com movimentos esportivos complexos. Utilizaremos para nosso estudo o arranco, que é um dos movimentos executados em competições de levantamento de pesos.

Escolhemos este movimento porque ele se enquadra perfeitamente nas características levantadas anteriormente. É considerado como um dos mais complexos movimentos

esportivos. Possui uma duração média de 1,2 segundos e uma trajetória bastante complexa.

As Figuras 1, 2 e 3 [4] nos ajudam a entender o arranco. A Figura 1 mostra a execução do arranco e a Figura 2 mostra o que seria a trajetória ideal para a barra.



A Figura 3 mostra a velocidade da barra durante a execução do mesmo.

Figura 1 O arranque.

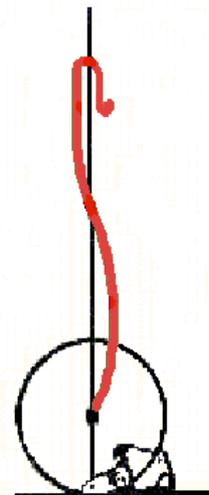


Figura 2 Trajetória da barra no arranque.

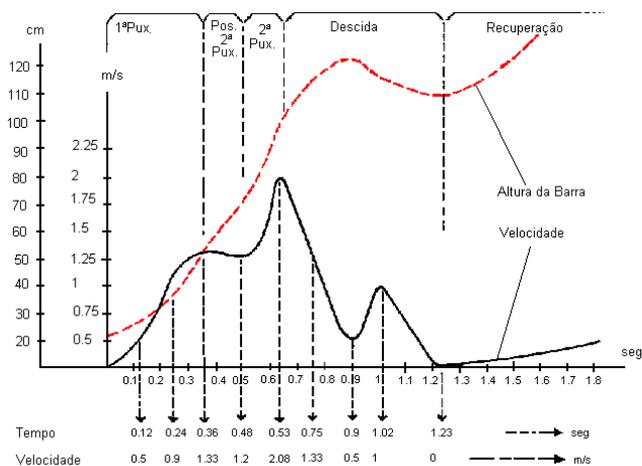


Figura 3 Velocidade e altura da barra no arranco.

Podemos observar na Figura 3 as grandes mudanças na velocidade e na posição da barra em pequenos espaços de tempo.

Definimos, assim, alguns requisitos adicionais para que o nosso sistema de rastreamento possa ser aplicado em movimentos com essas características apresentadas:

1. Alta Precisão: Vários dos parâmetros utilizados para análise do arranco possuem valores na ordem de poucos centímetros, de forma que um erro de mais de um centímetro poderia comprometer a análise dos dados coletados.
2. Alta Taxa de Quadros/Segundo: A barra se move com uma velocidade muito grande em alguns trechos do movimento (veja na Figura 3), de modo que uma baixa taxa de quadros faria com que tivéssemos poucos pontos coletados nestes trechos, dificultando a reconstrução do movimento depois.

3 Filtro de partículas

Filtro de partículas é um método para rastreamento de objetos que consiste em utilizar um conjunto de amostras (partículas) para estimar a posição do objeto rastreado.

A busca pelo objeto é feita a partir das posições estimadas, de forma que não se percorre a imagem inteira em busca do objeto.

3.1 Processo de Markov

Um processo de Markov de primeira ordem é um processo probabilístico no qual o estado do objeto no instante X_k depende apenas do estado anterior (X_{k-1}).

3.2 Modelo

Podemos modelar um filtro de partículas tomando X como um vetor que representa o estado do objeto rastreado em cada instante de tempo K , com K variando de 0 a N .

X é um processo de Markov de primeira ordem onde $X_k | X_{k-1} \sim P_{X_k | X_{k-1}}(X | X_{k-1})$, com uma distribuição inicial $p(X_0)$.

Seja Y um vetor com as observações feitas em cada instante de tempo K . Temos que Y_k pode ser aproximado por: $Y_k | X_k \sim P_{Y_k | X_k}(Y | X_k)$, dependendo apenas de X_k :

$$X_k = f(X_{k-1}) + V_k$$

$$Y_k = h(X_k) + W_k$$

Onde f e h são funções conhecidas.

3.3 Sampling Importance Resampling (SIR)

É um filtro de partículas amplamente usado, que aproxima a função de distribuição $P(X_k | Y_0 \dots Y_k)$ através de um conjunto de amostras (samples) às quais são atribuídos pesos.

$$\{(W_k^{(L)}, X_k^{(L)}) : L = 1, \dots, P\}$$

Onde $W_k^{(L)}$ são aproximações para a probabilidade relativa a posteriori de forma que:

$$\sum_{L=1}^P w_k^{(L)} = 1$$

3.4 Estratégia adotada

A estratégia adotada para este trabalho foi utilizar um filtro de partículas com as características mostradas acima.

Cada objeto a ser rastreado possui um vetor X de estados. Inicialmente, a posição de cada objeto é calculada percorrendo-se toda a primeira imagem do vídeo para determinar os estados iniciais.

Cada posição K do vetor de estados X representa o estado no frame de ordem K no vídeo sendo processado.

Desta forma, a cada novo quadro recebido, é realizada a seguinte seqüência de passos:

- 1 **obtain_observations(k):** observa na imagem a posição onde cada partícula do conjunto está apontando.
- 2 **predict_new_bases(k):** Com base na observação feita e na posição anterior da partícula, estima qual será a próxima posição para cada partícula.
- 3 **calculate_base_weights(k):** Para cada partícula, calcula um novo peso para a medição feita. Este peso é uma aproximação para a probabilidade de o objeto rastreado estar na posição prevista.
- 4 **update_after_iterating(k):** Exibe resultados obtidos e atualiza o vetor de estado.

4 Implementação

Como o objetivo é calcular um número variável de objetos em tempo real, fizemos uma simplificação importante, com o intuito tornar menos custoso o rastreamento. Marcamos os objetos a serem rastreados com pontos pequenos e com cores únicas para cada objeto.

Esta simplificação torna mais fácil o processo de observação e de cálculo de pesos das partículas, como veremos mais tarde.

Para estimar a posição das partículas, o deslocamento real do objeto observado na iteração anterior é usado.

A cada iteração é armazenado o deslocamento real nas coordenadas x e y do objeto. Cada partícula é então deslocada deste valor, como estimativa para a próxima iteração.

Para etapa de calcular um peso para cada partícula, foi feita uma função que verifica as posições estimadas de cada partícula. Caso a posição pesquisada corresponda ao objeto marcado é atribuído peso 1. Caso não corresponda, é atribuído peso 0.

Desta forma, a etapa de obter uma nova observação pode ser resumida a procurar nas partículas com peso igual a 1.

Para cada objeto rastreado foi definido um conjunto de partículas próprio. Cada conjunto faz suas estimativas e observações de acordo com o objeto associado a ele.

Uma implementação paralela foi feita também, para tentar aumentar a taxa de quadros processados por segundo.

Como descrito acima, cada objeto possui o seu conjunto próprio de partículas e de observações. Desta

forma, foi feita uma implementação onde cada objeto é rastreado por um processo diferente. Cada processo precisa consultar apenas dados relacionados ao seu objeto, não havendo a necessidade de muitos controles para sincronização de informações.

4.1 Complexidade

A complexidade deste algoritmo, como veremos, depende do número de partículas utilizado.

Em cada iteração do filtro, como podemos observar nos passos mostrados na seção anterior, chama-se as funções `obtain_observations`, `predict_new_bases`, `calculate_base_weights` para cada partícula no conjunto. A função `update_after_iterating` é chamada também uma vez a cada iteração.

Desta forma, constata-se facilmente que a complexidade total do algoritmo será $O(N * (\text{obs} + \text{pred} + \text{weight}) + \text{upd})$, onde:

- N = número de partículas;
- obs = Complexidade da função `obtain_observations`;
- pred = Complexidade da função `predict_new_bases`;
- weight = Complexidade da função `calculate_base_weights`;
- upd = Complexidade da função `update_after_iterating`.

A função `update_after_iterating` apenas imprime o resultado da posição detectada na iteração corrente, tendo complexidade constante ($O(1)$).

A função `predict_new_bases`, como descrito acima, estima a posição de cada partícula somando-se ao posicionamento da mesma o deslocamento observado do objeto na iteração anterior. Este deslocamento é calculado uma vez e é somado para cada partícula. Desta forma, o custo desta etapa também é constante ($O(1)$).

Na função `calculate_base_weights`, a partícula verifica a sua posição na imagem checando se a sua cor coincide com a do objeto associado, o que também possui custo constante ($O(1)$).

A função `obtain_observations` apenas precisa verificar o peso da partícula. Caso ela possua peso 1, a partícula está na posição correta. Esta verificação também pode ser feita em tempo constante ($O(1)$).

Desta forma, a complexidade total para cada iteração é $O(N)$.

4.2 Observações

Um erro que pode acontecer neste tipo de filtro é o de todas as partículas estarem longe do objeto. Neste caso, o filtro “perde” o objeto na imagem e precisa procurá-lo novamente na imagem inteira.

Procurar o objeto novamente é uma operação muito cara e tem custo $O(\text{height} \times \text{width})$, onde height é a altura e width a largura da imagem.

A chance deste tipo de erro acontecer diminui na medida em que se aumenta o número de partículas utilizado.

Entretanto, como pudemos ver no item anterior, a complexidade de cada iteração depende diretamente deste valor, de modo que é preciso achar um valor adequado que não fique muito grande e que não perca com frequência o objeto.

Outro ponto importante é a forma como as partículas estão distribuídas inicialmente. A implementação feita busca na imagem inteira pela posição dos objetos na fase de inicialização. Neste momento, inicializa-se também as posições e pesos das partículas, de modo que as partículas posicionadas em cima dos objetos são iniciadas com 1 e as demais são posicionadas nos pixels em volta do objeto e iniciadas com peso 0.

Para implementação feita, o número de partículas usadas pode ser configurado. Para os teste realizados, o número que proporcionou o melhor resultado foi 1600 partículas. Com este número, temos como observar, por exemplo, um quadrado de 40×40 pixels.

4.3 Código gerado

O código foi implementado na linguagem C, com o compilador gcc 3.4, no sistema operacional windows XP. Foi utilizada a biblioteca IM (3.1), do Tecgraf da PUC-RIO para captura e representação interna das imagens.

5 Resultados

Os testes foram executados em uma máquina com processador Intel Centrino Duo Core de 2,16 Ghz, 2 Gb de Ram e 512Mb de Vídeo.

Foi utilizado um vídeo com uma pessoa realizando os movimentos de um arranco. Neste vídeo foram escolhidos três objetos para rastreamento: o quadril do atleta, o cotovelo e a posição onde estaria a barra com os pesos. No vídeo, a pessoa realiza os movimentos sem a barra propriamente dita.

Para os testes, as marcações dos objetos foram feitas nos quadros da imagem e não na cena. Veja a Figura 4 com alguns quadros obtidos do vídeo.

O vídeo de teste tinha uma resolução de 352×288 pixels e possuía 278 quadros, contendo 2 execuções repetidas do arranco. Os quadros foram adquiridos a uma taxa de 30 quadros por segundo.

O arquivo de saída possui uma lista com as coordenadas (x, y), o instante K e o número do objeto lido, para cada objeto em cada iteração.

Para medir o desempenho do algoritmo, foi calculado o tempo de execução para o vídeo acima. Foram feitos experimentos variando o número de objetos rastreados.



Figura 4 Imagens de um arranco.

Cada experimento foi realizado 20 vezes e a média dos tempos foi calculada.

A Tabela 1 mostra os resultados obtidos para este teste:

Número de Objetos	Tempo Médio (seg.)	Desvio Padrão	Quadros/seg.
1	1,389	0,06836	200,14
2	1,527	0,07445	182,06
3	1,602	0,06895	173,53

Tabela 1 Testes.

Neste teste foi computado o tempo total de processamento do programa. Está incluído neste tempo carregar o arquivo com as imagens para memória, processar o filtro em todos os quadros da imagem e imprimir os resultados.

O último campo na tabela (quadros/seg.) é apenas uma estimativa. Apenas se dividiu o número total de quadros na imagem pelo tempo total de processamento. Para calcularmos a taxa real precisaríamos ler um vídeo com esta taxa de quadros e processá-lo à medida que os quadros vão sendo lidos.

Este número serve apenas para nos dar um indicativo que não teríamos problema para ler um vídeo com a taxa desejada.

Dadas as características do movimento escolhido, como podemos observar na seção 2, precisamos de uns 50 quadros por segundo para que tenhamos uma cobertura boa de quadros em todas as fases do movimento.

Testamos também a influência do número de partículas usadas. A Tabela 2 mostra os resultados caso troquemos o número de partículas de 1600 para 900. A Tabela 3 mostra os mesmos resultados para uma quantidade de partículas igual a 2500.

Número de Objetos	Tempo Médio (seg.)	Desvio Padrão	Quadros/seg.
1	1,416	0,08032	196,33
2	1,531	0,07442	181,58
3	1,648	0,06935	168,69

Tabela 2 Testes com janela de 30x30.

Número de Objetos	Tempo Médio (seg.)	Desvio Padrão	Quadros/seg.
1	1,451	0,06912	191,59
2	1,598	0,07011	173,97
3	1,710	0,07302	162,57

Tabela 3 Testes com janela de 50x50.

Como podemos observar, o uso de um número menor de partículas fez com que o tempo aumentasse, apesar da complexidade estar em função deste número. Isto se deve

ao fato de que o filtro passou a perder o objeto e teve que buscar mais vezes pelo objeto na imagem inteira, o que é muito caro.

Caso aumentemos mais a janela o resultado também piora, pois, apesar de diminuir a chance de perder o objeto, aumenta o processamento em cada iteração.

Caso o número de quadros por segundo fosse mais alto no momento da aquisição do vídeo (a câmera utilizada conseguia capturar apenas 30 quadros por segundo), talvez pudéssemos utilizar um número menor de partículas, uma vez que os objetos estariam se deslocando menos de um quadro para outro.

Por fim, fizemos um teste comparando a implementação utilizando um processo com a implementação utilizando vários processos.

A Tabela 4 mostra os resultados obtidos.

Número de quadros do vídeo	Tempo Médio Paralelo (seg.)	Tempo Médio Um Processo (seg.)
278	1,812	1,531
5560	23,312	26,125
8340	36,159	45,243

Tabela 4 Testes com janela de 40x40 e dois objetos rastreados paralelamente.

A melhora deve-se ao uso dos dois núcleos da máquina, que executam os processos simultaneamente. O algoritmo paralelo gastou 79,92% do tempo que o algoritmo com um processo gastou.

6 Conclusão

O algoritmo implementado teve um desempenho muito bom para o tipo de movimento escolhido, dando grandes indícios de não ser o gargalo em um sistema de tempo real.

Uma massa de testes mais intensa precisa ser realizada a fim de testar o comportamento do algoritmo com outros tipos de entrada: mais ruído, objetos ao fundo, vídeos de tamanhos diferentes e com diferentes taxas de quadro por segundo, etc.

Referências

- [1] M. Isard, A. Blake, CONDENSATION -- conditional density propagation for visual tracking, 1998.
- [2] David A. Forsyth and Jean Ponce, Computer Vision - A Modern Approach, Prentice Hall, New Jersey, 2003
- [3] E. Trucco, A. Verri, Introductory Techniques for 3-D Computer Vision, Prentice Hall, New Jersey, 1998.
- [4] Federacion Española de Halterofilia, Halterofilia Básica.
- [5] A. Medvedev, Film – reel prepared on the request of the IWF Executive Committee.
- [6] P. S. Maybeck , Introduction from Stochastic, estimation and control, Academic Press, 1979.