Avaliação da Técnica de Levantamento de Peso Olímpico Utilizando Visão Computacional

Keliton Amaral[1], **Alexandre Zamberlan**[1]

[1]Universidade Franciscana (UFN)
Santa Maria – RS

keliton.amaral@ufn.edu.br, alexz@ufn.edu.br

Abstract. *The performance analysis of sports is the practice carried out with the intention of understanding a sport and assist in decision making for better performance of athletes. This project created an application capable of tracking the path of the bar in a snatch lift of olympic weightlifting using processing of videos recorded with the camera of a cell phone. The software was developed with Python language and two of its frameworks: OpenCV for video processing and Matplotlib for graph plotting. The result is a system that can be used to analyze the performance of professional and amateur weightlifters.*

Resumo. *Análise do desempenho esportivo é uma prática realizada com o intuito entender um esporte e auxiliar na tomada de decisões para melhorar o desempenho de atletas. Este projeto criou uma aplicação capaz de rastrear a trajetória da barra em um levantamento snatch do levantamento de peso olímpico (LPO) através do processamento de vídeos gravados com a câmera de um celular. Este software foi desenvolvido com a linguagem Python e dois de seus frameworks: OpenCV para o processamento de vídeo e Matplotlib para plotagem de gráfico. O resultado é um sistema que pode ser usado para analisar o desempenho de levantadores de peso profissionais e amadores.*

1. Introdução

A análise do desempenho é uma área da ciência do esporte e do exercício preocupada com o desempenho esportivo real, em vez do relato subjetivo de atletas ou experimentos em laboratório [O'Donoghue 2010]. O que distingue a análise de desempenho de outras disciplinas é justamente a preocupação com os dados obtidos na execução da atividade desportiva em vez de atividades realizadas em ambientes ideais de laboratório ou dados coletados por meio de questionários, grupos de foco, relatos e entrevistas. Há casos em que exercícios baseados em laboratório podem contar como análise de desempenho se a técnica sob investigação é uma habilidade importante dentro do esporte de interesse ou a ação esportiva pode ser reproduzida fora do ambiente competitivo sem perdas consideráveis à fidelidade dos dados obtidos [O'Donoghue 2010].

A principal razão para fazer a análise de desempenho é desenvolver uma compreensão do esporte e auxiliar a tomada de decisão por aqueles que procuram melhorar o desempenho esportivo - atletas, treinadores e empresários. Muitos profissionais do esporte têm observado e analisado de maneira subjetiva o desempenho para melhorar os resultados em competições, a didática do ensino do esporte ou prevenção de lesões. No entanto,

tais observações possuem limitações [O'Donoghue 2010]. Por exemplo, Ian Franks realizou um estudo com treinadores de ginástica olímpica experientes e inexperientes. Ele solicitou que esses treinadores assistissem aos vídeos de duas apresentações. Treinadores experientes tiveram um grau considerável de falsos positivos, onde relatavam que havia diferenças entre as duas apresentações quando não havia. Além disso, treinadores experientes e inexperientes tiveram resultados parecidos quando solicitados a identificarem as diferenças entre as duas apresentações [Franks 1993]. Esse experimento demonstra a justificativa para usar a análise de desempenho esportivo: superar as limitações impostas pela observação subjetiva, fornecendo informações objetivas para alcançar uma maior compreensão do esporte. Isso pode, portanto, ter um papel vital no aprimoramento de atletas [O'Donoghue 2010].

Naturalmente, ferramentas computacionais são ideais para auxiliar esse ramo da ciência esportiva. Umas das áreas da Ciência da Computação é a Visão Computacional, que é um campo dentro da Inteligência Artificial (IA) que permite que computadores e sistemas obtenham informações relevantes de imagens ou vídeos e realizem ações ou recomendações baseadas nessas informações [IBM 2021]. Dessa forma, sistemas de captura de movimento de atletas utilizando câmeras e posterior processamento e análise dos dados obtidos são usados em diversos esportes.

Portanto, o objetivo geral deste trabalho é projetar, implementar e avaliar uma aplicação que possa auxiliar na análise da técnica da disciplina do *Snatch* (Arranque) do Levantamento de Peso Olímpico. Como objetivos específicos: i) revisão do referencial teórico sobre Levantamento de Peso e Visão Computacional; ii) revisão de trabalhos relacionados e das ferramentas usadas no processamento de vídeo desses trabalhos; iii) modelagem da aplicação; iv) teste da aplicação utilizando vídeos de diferentes praticantes de levantamento de peso olímpico.

Para um melhor entendimento da pesquisa, o texto apresenta a definição do que é o esporte do Levantamento de Peso Olímpico e quais são as disciplinas do *Clean and Jerk* e do *Snatch*. Em seguida, são apresentados conceitos de Visão Computacional no contexto da linguagem Python e seu uso em processamento de vídeo e imagem. Também são discutidos trabalhos relacionados e a relação com esta proposta. Por fim, o cenário do trabalho.

2. Revisão bibliográfica

Nesta seção, são discutidos conceitos e definições relacionados ao esporte contexto do trabalho, processamento de imagem e vídeo na área da Visão Computacional, ferramentas do universo Python para processamento de vídeo e trabalhos relacionados.

2.1. Levantamento de Peso Olímpico

O objetivo do esporte do Levantamento de Peso é levantar um total de peso maior que os oponentes em duas disciplinas: o *Snatch* (Arranque) e o *Clean and Jerk* (Arremesso) [IWF 2020][1]. No *Snatch*, a barra é posicionada horizontalmente na frente do atleta, aos pés dele. O atleta, então, segura na barra com ambas as mãos, com as palmas das mãos voltadas para baixo. Essa barra é puxada para cima em apenas um movimento até que

[1]A aplicação tema deste projeto foi pensada para analisar o Snatch, portanto o *Clean and Jerk* não é discutido neste artigo.

ela esteja acima da cabeça do atleta, com os braços totalmente estendidos, enquanto as pernas são flexionadas ou realizam um *split*, ou seja, uma perna à frente da outra. Durante o movimento, a barra pode deslizar ao longo das pernas e quadril do atleta, mas nenhuma outra parte do corpo além dos pés pode tocar no chão [IWF 2020]. Na posição final, a barra deve se manter acima da cabeça do atleta de maneira estável, os braço e pernas do atleta devem estar estendidos, com os pés posicionados em paralelo no mesmo plano do seu tronco e barra, até que o árbitro dê um sinal para que a barra possa ser solta. O árbitro dá esse sinal assim que o atleta se torna imóvel nessa posição final [IWF 2020]. Todo o movimento, da fase inicial com a barra no chão até o momento final quando o atleta se mantém em pé com a barra acima da cabeça, pode ser dividido em fases ou etapas. Essas fases variam de autor para autor, com algumas variações de nomes e número de fases, porém todos seguem a mesma sequência lógica. Para este projeto é usado, principalmente, o modelo descrito por [Campos et al. 2006] (Figuras 1 e 2): Primeira puxada (*first pull*), Transição (*transition*), Segunda puxada (*second pull*), Virada (*Turnover*), Recepção (*Catching*), Absorção (*Absorption*) e Recuperação até o atleta ficar de pé completamente estabilizado.

Figura 1. Fases do Snatch.

Figura 2. Continuação fases do Snatch.

Por muitos anos, estudiosos do esporte também têm estudado quais são as variáveis de um *Snatch* bem sucedido. Uma das variáveis mais relevante, até o presente, é a trajetória da barra [Noriega 2018]. Um dos primeiros estudiosos a descrever essa trajetória foi Vorobyev [Vorobyev 1978], o qual afirma que a barra descreve um "S"alongado, ao observar o movimento a partir do plano sagital do atleta, isto é, observando o atleta lateralmente (Figura 3).

2.2. Visão Computacional

Visão Computacional é o campo da Ciência da Computação que foca em criar sistemas que possam processar, analisar e entender dados visuais, sejam eles imagens ou vídeos,

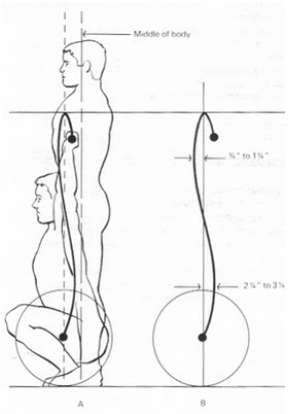

Figura 3. Trajetória ideal da barra no movimento do Snatch [Weightlifting Physics 2021].

da mesma maneira que humanos fazem [Babich 2020]. Na prática, trata-se de ensinar computadores a trabalharem com as imagens ao nível do pixel e interpretá-las utilizando algoritmos especiais.

Três das principais tarefas desse campo são identificação, classificação e rastreamento [Babich 2020]: i) Identificação de objetos. O sistema processa o conteúdo de uma imagem ou vídeo e identifica um objeto particular. Por exemplo, o sistema consegue achar a barra de peso no meio de outros objetos na imagem; ii) Classificação de objetos. O sistema processa o conteúdo de uma imagem ou vídeo e classifica o objeto naquele conteúdo dentro de uma categoria. Por exemplo, o sistema consegue classificar uma barra de peso como masculina (28 milímetros) ou feminina (25 milímetros); iii) Rastreamento de objetos. O sistema processa vídeo para achar um objeto ou mais de um objeto que possuem o mesmo critério de busca e rastreia seus movimentos. Por exemplo, o sistema rastreia a trajetória da barra de peso, da posição inicial (chão) até a posição final (acima da cabeça do atleta). Em um sistema de Visão Computacional, essas três tarefas podem ser interdependentes e utilizadas no mesmo sistema. Por exemplo, um sistema usado em segurança pública pode identificar faces de procurados pela polícia no meio de uma multidão e ao localizá-los, rastrear seus movimentos para informar aonde a pessoa foi. O presente projeto foca na identificação e rastreamento de objetos em vídeo.

2.2.1. Processamento de imagem e vídeo

De acordo com [Gonzalez and Woods 2018], uma imagem pode ser definida como uma função bidimensional *f(x, y)*, onde *x* e *y* são coordenadas cartesianas (plano) e a amplitude de *f* em um determinado par *(x, y)* é a intensidade da imagem naquele ponto. Quando *x*, *y* e *f* possuem valores discretos, a imagem é chamada de imagem digital. Quando esses valores são representados por números contínuos, então tem-se uma imagem analógica[2].

[2]De maneira resumida, valores discretos são valores que só podem ser representados por números do tipo inteiro, 1, 2, 457, 1089 e assim por diante. Enquanto isso, valores contínuos podem assumir qualquer valor, como números com decimais. Por exemplo, o número de pacientes que um hospital atende só pode ser dado por números inteiros (valores discretos). Mas o peso desses pacientes pode ser dado por valores contínuos, como 89,9 kg, 67,65 kg

Uma imagem digital é composta por um número finito de elementos, conhecidos como *pixels*, cada um com uma localização e valor.

Uma das formas de representar uma imagem computacionalmente é utilizando uma matriz (*array* bidimensional) de valores numéricos para *f(x, y)*. Cada elemento *f(x, y)* é um *pixel*. *Pixels* específicos são valores da matriz em uma coordenada [Gonzalez and Woods 2018]. Pode-se dizer que vídeos possuem uma imagem digital em cada *frame* e ao colocar esses *frames* em sequência ocorre a sensação de movimento [Gonzalez and Woods 2018]. Em aplicações de vídeo, a função é dada por *f(x, y, t)*, onde *t* é um dado discreto que indica o tempo quando o *pixel* ocorre [Gonzalez and Woods 2018].

2.2.2. Linguagem Python e Framework OpenCV

A linguagem e seus *frameworks* têm destaque nas áreas de Inteligência Artificial, Ciência de Dados, Mineração de Dados e Programação de Alto Desempenho, por exemplo. Um dos *frameworks* de destaque é o OpenCV, que é uma biblioteca de código aberto que dispõem de ferramentas para realizar diversos tipos de processamento de imagem e vídeo [OpenCV 2021]. Uma das principais funções do OpenCV é o rastreamento de objetos [Granatyr 2020]. Porém, ressalta-se uma diferenciação importante na codificação entre rastreamento e detecção (ou identificação) de objetos. Como visto anteriormente, um vídeo é uma sequência de imagens digitais e quando vários *frames* são colocados em sequência, eles dão a sensação de movimento. Para detectar um objeto, um algoritmo necessita saber as principais características do objeto e analisar no *frame* qual conjunto de pixels melhor demonstra essas características. No entanto, rastreamento de objetos implica identificar esse conjunto de *pixels* em cada *frame* do vídeo, já que se o objeto se mover, os elementos da matriz que contém o dados significantes do objeto também vão mudar.

Algoritmos específicos de rastreamento de objetos costumam trabalhar com um certo nível de predição (por exemplo, CSRT). Considere a tarefa de rastrear um corredor em uma prova de corrida. É possível identificar esse corredor sempre executando o algoritmo de detecção de objeto do zero a cada *frame* de um vídeo. Contudo, se forem utilizados certos algoritmos específicos de rastreamento, que usam predição, eles identificam o objeto (corredor) nos *frames* iniciais e utilizam as características desse objeto para prever o próximo movimento do corredor nos *frames* seguintes. Em outras palavras, o esforço computacional para rastreamento de objetos sem técnicas de predição acaba sendo maior porque o algoritmo precisa realizar toda a tarefa de identificação do objeto em todos os *frames*, ao invés de identificar nos primeiros *frames* e prever a próxima posição[Granatyr 2020]. Dessa forma, utilizar algoritmos com predição gera economia de recursos computacionais. Entre os algoritmos de rastreamento existentes, nativamente, no OpenCV encontram-se o *Kernel Correlation Filters* (KCF) e o *Channel and Spatial Reliability of Discriminative Correlation Filter* (CSRT). O KCF é mais rápido para o processamento de vídeo, porém ele é menos preciso especialmente quando objetos fazem mudanças bruscas de trajetória, enquanto o CSRT apesar de mais lento, consegue rastrear objetos com mais precisão.

2.3. Trabalhos Relacionados

Nesta seção são mostrados alguns trabalhos que utilizaram ferramentas computacionais para fazer avaliações biomecânicas e posturais do corpo humano.

No trabalho realizado por [de Moraes and Cantarelli 2015] foi implementado um protótipo para avaliação postural utilizando o dispositivo Microsoft Kinect para captura da postura do paciente, a linguagem de programação C# para criação do software de avaliação e o MySQL como gerenciador do banco de dados. O protótipo tinha a capacidade de realizar a análise da visão anterior (frente) do corpo humano, mas devido a limitações do próprio Kinect, não foi possível a utilização das visões lateral e posterior [de Moraes and Cantarelli 2015]. O trabalho de [Sohani et al. 2018] propôs uma técnica para analisar o movimento de pulo vertical de um atleta utilizando análise de vídeo quadro a quadro (*frame by frame*). Segundo [Sohani et al. 2018], os métodos mais comuns para analisar movimentos atléticos são observações feitas por um especialista humano/treinador ou valores são capturados por algum dispositivo ou traje no corpo do atleta. O objetivo foi criar heurísticas e algoritmos que diminuíssem a dependência do especialista, proporcionando um método mais acessível para analise do movimento de atletas, utilizando um *smartphone* para captura do vídeo com o movimento. O sistema processava vídeos quadro a quadro aplicando uma combinação das técnicas *Histogram of Oriented Gradient* (HOG) e *Support Vector Machine* (SVC) para achar o humano no quadro e então rastreá-lo do início ao fim do vídeo. A partir daí, foi possível calcular a altura em pixels do pulo e fazer uma conversão para a altura pulada no mundo físico real. O algoritmo foi escrito utilizando a biblioteca OpenCV [Sohani et al. 2018]. Já no trabalho de [Kusuma et al. 2018], foi investigado o aumento no desempenho do *Snatch* de um atleta, via análise de vídeos para melhoria posterior do movimento. Para isso, foram observados os movimentos e ângulos dos tornozelos, joelhos, quadril, velocidade e trajetória da barra utilizando dois pesos diferentes. Seis atletas masculinos tiveram os vídeos de seus levantamentos analisados por um software de análise de movimento chamado SIMI[3]. Os vídeos com a realização dos movimentos foram capturado com uma câmera Sony NEX-VG30EH. Os resultados foram relatados em três partes. A primeira dividiu o levantamento em várias fases. A segunda parte comparou levantamentos bem sucedidos e levantamentos fracassados. A terceira analisou o melhor atleta em relação aos demais. Concluiu-se que a análise cinemática[4] pode ser utilizada para observar insuficiências técnicas que não podem ser observadas em tempo da execução, permitindo ao treinador criar exercícios corretivos com maior precisão. Por fim, é citado o trabalho de [Ng 2020] que não apenas identificou os exercícios de musculação sendo realizados nos vídeos, como também avaliou as respectivas técnica. Foi avaliada a postura do atleta bem como o ângulo entre articulações chave do seu corpo. O trabalho foi feito utilizando Python e sua biblioteca OpenCV. O sistema foi capaz de detectar e reconhecer alguns dos exercícios normalmente realizados em academias, tais como rosca de biceps e elevação de ombros com uma precisão variável entre 60 e 98 dependendo do exercício sendo realizado no vídeo analisado pela aplicação.

A Tabela 1 apresenta um comparativo dos trabalhos correlatos estudados, com

[3]Site do fabricante.

[4]Conforme [Whittaker 1937], cinemática é o ramo da física que se ocupa da descrição dos movimentos de pontos, corpos ou sistemas de corpos (grupos de objetos), sem se preocupar com a análise de suas causas.

seus objetivos, linguagens, hardware e ferramentas utilizadas.

Trabalho	Objetivo	Linguagens	Hardware	Outras ferramentas
[de Moraes and Cantarelli 2015]	Avaliação postural.	C#, SQL	Microsoft Kinect	SDK do Kinect
[Sohani et al. 2018]	Avaliação do movimento de salto do atleta.	Python	Celular com câmera	OpenCV
[Kusuma et al. 2018]	Avaliação da técnica de Snatch do LPO.	Linguagem proprietária	Câmera Sony NEX-VG30EH	Software proprietário (SIMI)
[Ng 2020]	Reconhecimento do exercício de musculação e avaliação da técnica	Python	Celular com câmera	OpenCV

Tabela 1. Comparativo dos trabalhos relacionados.

3. Proposta do sistema

Este trabalho é um projeto de Visão Computacional apoiado por uma revisão bibliográfica tanto dessa área, quanto da área da ciência esportiva. Para avaliação da proposta, um protótipo foi projetado e implementado para capturar vídeos de Levantamento de Peso Olímpico e processá-los para detectar a qualidade técnica do movimento.

3.1. Materiais e métodos

O projeto utilizou ferramentas de desenvolvimento de *software* e sistema de versionamento de arquivos, bem como métodos de gestão de projetos de *software*: i) Trello - ambiente para gestão de atividades via técnica Kanban; ii) Overleaf - editor *online* Latex; iii) Astah - ambiente para diagramação de aspectos funcionais e estruturais do protótipo; Github - repositório e versionamento de códigos construídos; iv) Linguagem Python; v) *Framework* OpenCV - para processamento de vídeos [OpenCV 2021]; e Matplotlib - para criação dos gráficos da trajetória da barra [MatPlotLib 2021].

Este trabalho faz uso do Scrum, que é uma metodologia ágil, iterativa e incremental para desenvolvimento de software [Wykowski and Wykowska 2019]. Dentro do contexto deste trabalho, há papéis como *Scrum master* (gestor), *Product owner* ('dono' ou responsável), *Dev team* (equipe de desenvolvimento). Dessa forma, os autores deste trabalho (aluno e orientador) foram os responsáveis por todo o processo de criação, desenvolvimento e testes do sistema (*Dev Team*). O delineamento e as definições de aspectos funcionais (planejamento, execução, revisão e retrospectiva do *sprint*) do sistema, os *layouts* e a dinâmica de funcionamento geral, além de fornecer todo o conteúdo relacionado ao processo LPO, ficaram sob responsabilidade do aluno, que é o especialista da área. O orientador do trabalho comportou-se como *Scrum Master*, que gerenciou o desenvolvimento como um todo (prazos, metas, responsáveis, controle de versões do sistema, reuniões, etc). E o treinador de LPO é o *Product Owner*.

3.2. Modelagem do sistema

Todo processo de modelagem passa pelo levantamento de requisitos, que é justamente identificar aspectos funcionais e estruturais necessários ao funcionamento adequado do sistema. O sistema tem como requisitos funcionais:

1. Vídeo lateral que começa com o atleta na posição inicial do *snatch* e termina na posição final com o atleta em pé, pés paralelos e barra acima da cabeça;
2. Vídeo em formato *MP4*;
3. Vídeo com resolução 1080p;

4. Vídeo lateral mostrando o atleta realizando o movimento do *Snatch*;
5. Processamento do vídeo e análise do rastreio da trajetória da barra;
6. Visualização de gráfico com plotagem da trajetória da barra em um plano cartesiano.

Como requisitos não-funcionais:

1. Biblioteca de tratamento e análise de vídeos;
2. Câmera de gravação de vídeo.

Dessa forma, para uma compreensão dos atores e das funcionalidades planejadas para o sistema, a Figura 12, presente na seção "Apêndices ", ilustra os requisitos funcionais distribuídos nos diversos casos de uso. Também nos "Apêndices", é possível encontrar o diagrama de atividades (Figura 13) que apresenta a dinâmica de funcionamento do sistema, como uma modelagem de processo.

Acredita-se, que atendendo os requisitos funcionais 1, 11 e 12 é possível testar e avaliar o cerne do sistema proposto, uma vez que é possível visualizar se a trajetória seguiu um padrão definido pelos autores citados em "Revisão Bibliográfica", por meio de gráfico da trajetória.

3.3. Resultados e discussões

Esta seção mostra os resultados alcançados, trechos de código, principais dificuldades encontradas e explicações sobre as ações tomadas durante o desenvolvimento da aplicação.

3.3.1. Gravação dos vídeos e Classes de serviços

Para gravar os vídeos usados na geração dos gráficos, foi utilizada a câmera do *iPhone 13 Pro*. O celular foi posicionado sobre uma plataforma de 75 cm de altura, distante da barra por aproximadamente 2,5 metros. O vídeo estava no formato *MP4*, resolução 1080p a 30 *frames* por segundo. Ele também foi editado para que mostrasse apenas do início do levantamento até o momento final, antes do atleta largar a barra, aproximadamente 6 segundos.

O processamento de vídeo da aplicação depende de três classes principais - "Lift", "Analysis"e "Athlete". A Classe "Lift"manipula as informações do levantamento da barra e cada vídeo com levantamento é tratado como uma instância dessa classe, que possui os seguintes atributos: i) "file_path": caminho para o arquivo de vídeo demonstrando o levantamento; ii) "bar_weight": peso em quilogramas que o atleta levantou; iii) "lift_date": data e hora da realização do levantamento.

A importância do caminho para o vídeo é evidente já que a aplicação precisa de um arquivo no formato *MP4* para realizar o processamento. O peso em quilogramas que o atleta levantou é usado no cálculo de força relativa da classe "Analysis". Por fim, a data e hora do levantamento podem prover dados importantes para análise do progresso do atleta no decorrer de sua carreira ou comparativos entre treinos realizados em diferentes períodos (este último, como um trabalho futuro).

A classe "Lift"possui ainda o método "find_trajectory()", onde ocorre o rastreamento da barra e a extração das coordenadas da mesma em cada *frame* para posterior

análise na classe “Analysis”. O método inicia com a leitura do arquivo de vídeo utilizando uma instância da classe “VideoCapture”do OpenCV, na variável “cap”. A classe “VideoCapture”é usada para capturar video de arquivos de video, sequências de imagens ou câmeras [OpenCV 2022]. Essa classe possui uma função pública que recebe uma *string* com o caminho para o vídeo e que neste caso, foi utilizado o atributo “file_path”.

Como apresentado na seção “2.2.2. Linguagem Python e Framework OpenCV”, dois dos principais algoritmos de rastreamento de objeto utilizados em Visão Computacional são o CSRT e o KCF. A implementação de ambos no OpenCV é feita da mesma forma, mudando apenas o nome do método que cria o rastreador. Para este trabalho, foram criados os rastreadores de ambos, através do “TrackerCSRT_create()”e “TrackerKCF_create()”, que herdam da classe “Tracker”métodos mais gerais e comuns a todos os algoritmos de rastreamento disponíveis no OpenCV, como os métodos para inicializar o rastreador ou para atualizar o *frame* sendo processado [OpenCV 2022].

Após testar a implementação do KCF e CSRT, notou-se que o KCF tinha um tempo de processamento melhor, porém perdia a localização do objeto nos *frames*, onde a barra acelerava ou desacelerava bruscamente, como na fase de Transição (quando a barra passa da altura dos joelhos e vai até mais ou menos a altura da cintura do atleta) e Virada (quando a barra pára de subir e o atleta inicia o movimento para ir para debaixo dela). Por outro lado, o CSRT, apesar de mais lento, não perdia a barra em nenhum *frame*. Por causa disso, ele foi escolhido para o desenvolvimento deste projeto.

A Figura 4, trecho de código, apresenta a criação e inicialização de uma nova instância do rastreador do CSRT. Um dos conceitos utilizados para auxiliar na precisão desses algoritmos é o conceito de *Region Of Interest*, ROI ou região de interesse. ROI é uma amostra dentro de um conjunto de dados, selecionado para um propósito [OpenCV 2022]. Em Visão Computacional, ROI define as bordas do objeto da imagem que será rastreado [Granatyr 2020]. No OpenCV, a ROI é selecionada através de um *bounding box*, literalmente uma caixa desenhada pelo usuário. Na linha 47, é utilizado o método “read()”para ler o primeiro *frame* do vídeo e guardar seus dados na variável “frame”. Por fim, essa variável é passada para o método “selectROI()”que abre uma janela com o primeiro *frame* do video para o usuário desenhar o *bounding box* com o *mouse*, segurando e arrastando o *cursor* até criar uma caixa que envolva e mantenha a anilha da barra no centro dessa caixa. Também na Figura 4, imagem de levantamento de peso, é possível visualizar o desenho de um *bounding box* ao redor da anilha de peso.

Na sequência, no método “find_trajectory()”, é feito um laço de repetição do tipo *while* (Figura 5). A sua função principal dessa rotina é atualizar a aplicação com o próximo *frame* do vídeo, que é uma sucessão de imagens estáticas (*frames*). Por causa disso, os rastreadores do OpenCV vão comparar a posição do objeto no *frame* atual com o *frame* anterior para verificar a movimentação do objeto. Isso é feito por toda a sequência de imagens que compõem o vídeo, até que todos os *frames* tenham sido lidos.

O retorno do método “find_trajectory()”é uma lista aninhada (*nested list*) de nome “trajectory”. Cada elemento dela é uma sub-lista “[x, y]”, onde x e y são as coordenadas do *pixel* do centro do objeto rastreado dentro do *bounding box*. A cada passagem do laço de repetição, é feita a leitura de um novo *frame* e com ele, o objeto assume um novo x e y, que é adicionado como novo elemento no fim da lista “trajectory”. O principal problema

```
# Create CSRT tracker instance
tracker = cv2.TrackerCSRT_create()
# Read first frame
ok, frame = cap.read()
# Create bounding box with region of interest
bbox = cv2.selectROI(frame)
# Initialize the tracker with a
# known bounding box that surrounded the target.
ok = tracker.init(frame, bbox)
```

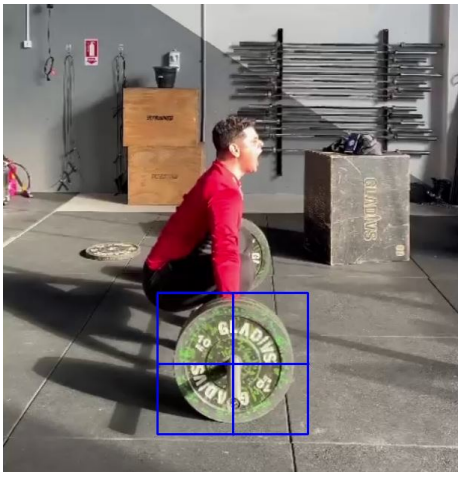

Figura 4. Criação do rastreador e do *bounding box*

```
while True:
    ok, frame = cap.read()
    if not ok:
        break
    # Add gaussian blurring to frame
    frame_blurred = cv2.GaussianBlur(frame, (5, 5), 0)
    # Convert to gray scale
    frame = cv2.cvtColor(frame_blurred, cv2.COLOR_BGR2GRAY)
    ok, bbox = tracker.update(frame)
    if ok:
        (x, y, bbox_w, bbox_h) = [int(f) for f in bbox]
        # Draw a box to follow the object being tracked
        cv2.rectangle(frame, (x, y), (x + bbox_w, y + bbox_h),
        (0, 255, 0), 2, 1)
        new_y = video_height - y
        trajectory.append([(x+bbox_w)/2, new_y+bbox_h/2])
    else:
        cv2.putText(frame, "Error", (100, 80), cv2.FONT_HERSHEY_SIMPLEX, 1,
        (0, 0, 255), 2)       You, now • Uncommitted changes

    cv2.imshow('Video', frame)
    # 'ESC' key to stop
    if cv2.waitKey(1) & 0XFF == 27:
        break
```

Figura 5. Laço de repetição do método que acha a trajetória da barra.

durante o desenvolvimento da aplicação foi a precisão dessas coordenadas.

Um primeiro desafio foi o fato do sistema de coordenadas do OpenCV ser diferente do Matplotlib. O gráfico da trajetória da barra é gerado utilizando Matplotlib, biblioteca da linguagem Python para criação de visualizações estáticas, animadas e interativas [MatPlotLib 2021]. Essa biblioteca por padrão utiliza o sistema de coordenadas cartesianas tradicional, onde cada ponto no gráfico é dado pelo par (x, y) onde x é o deslocamento horizontal e y é o deslocamento vertical. Nesse sistema, o ponto origem[5] é no canto inferior esquerdo e conforme a trajetória se desloca para cima, o elemento responsável pelo deslocamento vertical (y) aumenta. O OpenCV no entanto trata imagens como uma matriz matemática onde ponto de origem é no canto superior esquerdo. Ao percorrer verticalmente uma imagem digital, o y diminui se o ponto subir e aumenta se descer, o inverso do sistema cartesiano tradicional. A Figura 6 apresenta a diferença entre os dois sistemas de coordenadas.

Embora o MatPlotLib possa criar um gráfico com a origem no canto superior esquerdo, optou-se por utilizar o sistema cartesiano tradicional por uma questão de ex-

[5]Par de coordenadas (x, y) onde os elementos são (0, 0)

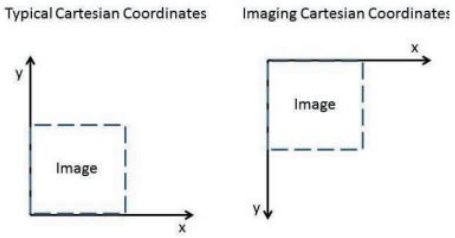

Figura 6. Sistema cartesiano tradicional (esquerda) e sistema utilizado em processamento de imagens (direita) [Lumenera 2022].

periência do usuário, visto que representar altura de objetos utilizando o sistema do OpenCV é menos comum. Foi necessário então tratar essa diferença na representação de deslocamento vertical. O OpenCV possui métodos para fazer transformações matriciais, mas esses métodos também invertem a imagem. Como o software precisa apresentar o primeiro *frame* para o usuário delimitar a ROI, não seria prático mostrar um *frame* "de cabeça para baixo"para o usuário. Também não seria prático inverter a imagem verticalmente pois o video é apresentado ao usuário durante o processamento da demanda de rastreamento. Durante a execução do laço de repetição demonstrado na Figura 5 a variável do *bounding* box (bbox) é atualizada a cada *frame* através do método "tracker.update(frame)"na linha 52, guardando o novo x e y do canto superior esquerdo do *bounding* box que acompanha e rastreia as anilhas da barra. Esse x e y é usado entre outras coisas como argumentos para o método "cv2.rectangle()"que vai desenhar um retângulo ao redor das anilhas a cada *frame*, apenas para demonstrar ao usuário que o rastreamento está sendo realizado. Por causa disso a imagem também não foi invertida para não comprometer essa apresentação ao usuário.

Optou-se por uma solução mais simples, pois como o deslocamento vertical no *frame* é finito por conta de se saber o tamanho da imagem, existe sempre uma proporção entre a altura final da imagem até a altura do pixel sendo analisado e depois até a altura inicial da imagem. Considerando a altura total como um segmento de reta, a altura onde o pixel está sempre vai dividir esse segmento em outros dois segmentos proporcionais. Pode-se representar essa proporção com uma relação algébrica do tipo $h = y1 + y2$, onde h é o tamanho do segmento de reta que dá a altura total da imagem, y1 é a distância da origem até a altura do pixel e y2 é a distância da altura do pixel até o final do segmento de reta h. Realizando a operação algébrica h - y1 portanto é possível saber y2 e como consequência a altura do pixel no sistema cartesiano que começa com a origem no canto inferior esquerdo. A Figura 7 ajuda a explicar essa proporção utilizando uma imagem ou *frame* que possui 1000 px de altura e um objeto que está no y de valor 300 no OpenCV. Ao subtrair 300 da altura total, o resto da operação é 700, isto é, faltam 700 linhas de pixel para alcançar o final da imagem. Fazendo o caminho inverso, ou seja, utilizando o plano cartesiano tradicional, significa colocar o objeto no y de valor 700.

Conhece-se a altura total da imagem ou *frame* através da linha 29, que possui a expressão "video_height = int(cap.get(4))", e a altura do canto superior esquerdo do *bounding box* na linha 54. Calcula-se na linha 58 da Figura 4 a relação algébrica para descobrir o y no sistema cartesiano. Essa operação é realizada em cada *frame*. Note na linha 59 é realizado o *append* que recebe o x e y da trajetória da barra, porém é realizada uma operação de soma e divisão por 2 para adicionar à lista o x e o y do centro do *bounding*

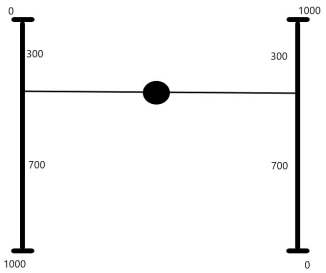

Figura 7. A diferença entre a altura total da imagem e a altura do pixel no OpenCV (esquerda) sempre vai dar a altura do pixel no sistema cartesiano (direita).

box.

Outro desafio foi a precisão da linha que desenha a trajetória do gráfico. Um vídeo a cores e com muita nitidez parece afetar a precisão do objeto rastreador mais do que um video preto e branco e com um pouco de desfoque. Nas linhas 49 e 51 da Figura 4, foi realizada a conversão de cores (BGR) para tons de cinza, além de adicionado desfoque gaussiano em cada *frame*, o que contribuiu com a diminuição do ruído no vídeo e mais precisão.

A classe "Analysis"possui dois métodos, mostrados na Figura 8 (lado esquerdo). O método "plot_graph()"recebe a lista aninhada com as coordenadas *frame-a-frame* da barra que é retorno de "find_trajectory()"da classe "Lift". O "plot_graph()"vai realizar algumas operações da bliblioteca MatPlotLib, que ao ser importada para o projeto recebeu o *alias* "plt". Especial atenção entre as linhas 6 e 10. É feito um laço de repetição que vai percorrer toda a lista de coordenadas e separá-la em duas novas listas, uma com os x e outra com os y. Isso foi feito porque o método da linha 17 que plota os dados no gráfico, "plot()", precisa dos valores de x e y separadamente. Por fim, também há o método "calculate_relative_strength()"que calcula o índice de força relativa do atleta. É um operação de divisão simples que divide o peso que o atleta levanta com o seu peso corporal. É um dos indicadores de desempenho mais utilizados em esportes de força e também serve como critério de desempate em competições de levantamento de peso [IWF 2020]

A última classe é a "Athlete", na Figura 8 (lado direito), que é composta de um único método construtor que recebe atributos do atleta: peso corporal, altura e idade. Peso corporal é utilizado no cálculo da força relativa, porém altura e idade não são utilizados pelo sistema no momento. Como parte de trabalhos futuros será possível avaliar outros indicadores de desempenho baseados nesses dados.

3.3.2. Análise dos gráficos gerados

Na Figura 9, é possível ver o gráfico da trajetória da barra gerado pela aplicação ao lado de um desenho da trajetória originalmente descrita por [Vorobyev 1978]. Note que no gráfico gerado pela aplicação é descrito um "S"alongado, semelhante a interpretações artísticas da trajetória, como na imagem da direita. Começando de baixo para cima, existe um pequeno deslocamento negativo (para a esquerda) no eixo x, seguido de um deslocamento positivo (para a direita) no mesmo eixo. Após alcançar uma determinada altura,

```python
def plot_graph(self, coordinates):
    x = []
    y = []
    for a,b in coordinates:
        x.append(a)
        y.append(b)

    plt.xlabel("Eixo x")
    plt.ylabel("Eixo y")
    plt.title("Coordenadas cartesianas da trajetória da barra")
    plt.xlim(0, 1000)
    plt.ylim(0, 1000)
    plt.plot(x, y, label="Trajetória")

    plt.show()

def calculate_relative_strength(self, body_weight, bar_weight):
    return round(bar_weight / body_weight, 2)
```

```python
class Athlete:

    def __init__(self, body_weight, body_height, age):
        self.body_weight = body_weight
        self.body_height = body_height
        self.age = age
```

Figura 8. Métodos para plotagem do gráfico e cálculo da força relativa (lado esquerdo). Construtor da classe Athlete (lado direito).

a barra pára de subir e produz uma pequena descida. Um dos principais pesquisadores, [Vorobyev 1978], originalmente encerra a descrição da trajetória nesse ponto, mas como visto em [Noriega 2018] e [Campos et al. 2006], o atleta ainda precisa ficar de pé com a barra acima da cabeça. Por causa disso, no gráfico gerado pela aplicação, a linha ainda sobe.

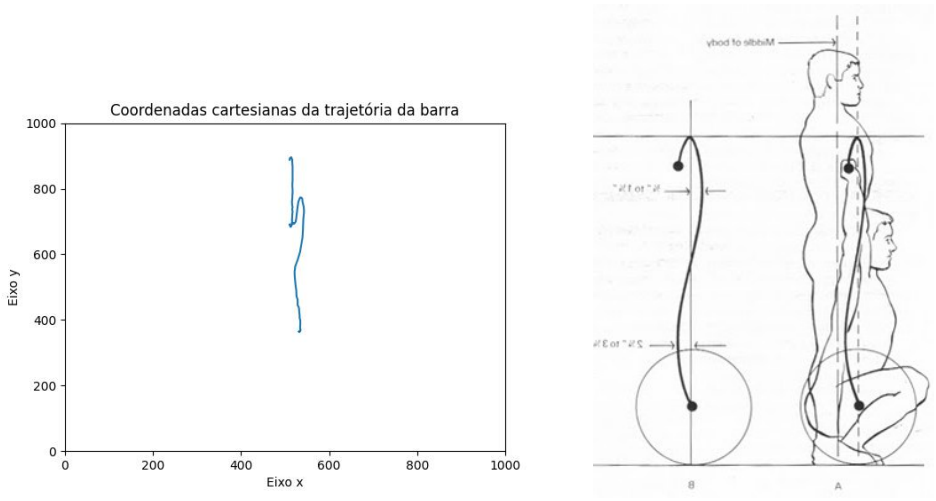

Figura 9. Comparação dos gráficos.

Com o intuito de ser mais didático e demonstrar a precisão da aplicação, pode-se ainda mostrar algumas fases do *Snatch* utilizando *frames* específicos do vídeo juntamente com o gráfico (Figuras 10 e 11, respectivamente).

3.3.3. Discussões e trabalhos futuros

Durante o desenvolvimento desta primeira versão, foi testado apenas o formato de video *MP4*. A classe "VideoCapture()"do OpenCV trabalha com diversos formatos de arquivo de video, porém cada um utiliza um *codec*[6] diferente. Não foi testada a precisão do rastreador CSRT ou da função que cria a lista com coordenadas da barra em diferentes for-

[6]Um *codec* é um *software* que permite fazer a codificação ou decodificação de um fluxo de dados digitais, com o intuito de fazer transmissão ou armazenamento. Neste caso, o OpenCV utiliza um *codec*

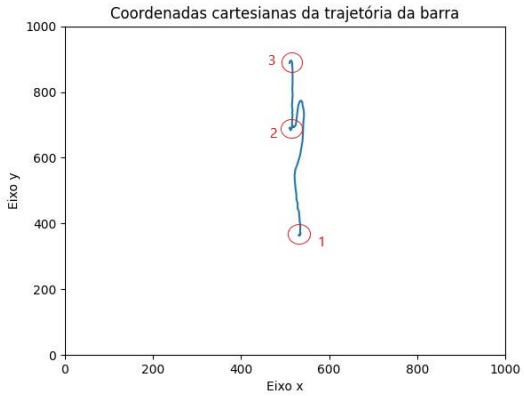

Figura 10. Marcação do Início da primeira puxada (1); Final da Recepção (2); Final da Recuperação (3)

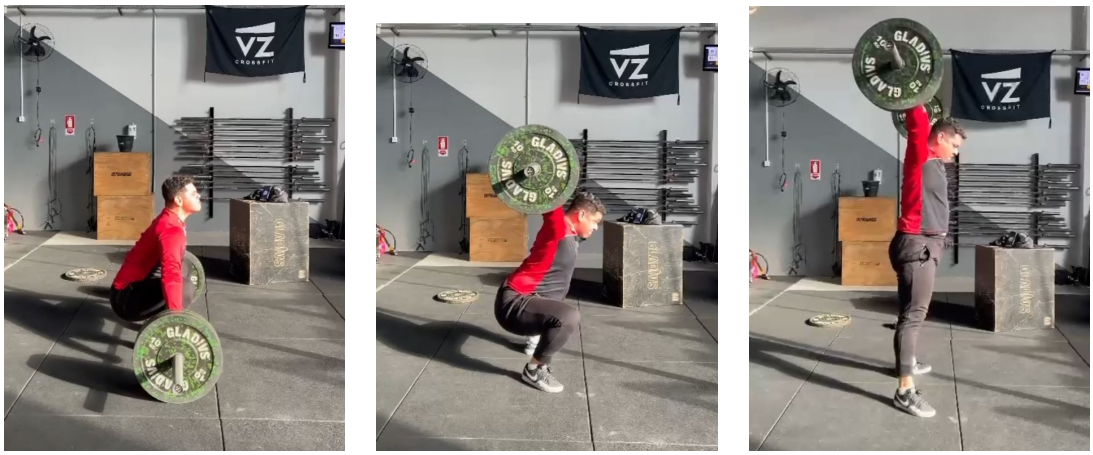

Figura 11. Ínicio da primeira puxada (1); Final da Recepção (2); Final da Recuperação (3).

matos de vídeo. Mas levando em consideração que *codecs* utilizam algoritmos distintos de decodificação de dados, é possível que um mesmo levantamento gravado ou convertido para diferentes formatos seja rastreado de maneira ligeiramente diferente devido à mudança de qualidade gráfica.

O mesmo comentário é feito em relação à resolução do vídeo. Em tese, um vídeo com maior resolução possui mais *pixels* que a aplicação pode usar para rastrear a barra, influenciando na precisão. Foram testadas resoluções maiores que 1080p, porém a janela aberta pelo OpenCV para que o usuário desenhe o *bounding box* também muda de tamanho. Resoluções maiores abrem janelas maiores, que podem não caber completamente na tela do dispositivo do usuário. O OpenCV possui métodos para mudar o tamanho do vídeo mas que exigem recriá-lo, então retorna-se à consideração feita no parágrafo anterior sobre perda de qualidade com a decodificação.

Entende-se que há alguns trabalhos a realizar para que o sistema entre em operação

---

para decodificar o arquivo de vídeo *MP4* em dados que possam ser lidos e armazenados em uma variável ou instância de classe.

e fique disponível a atletas e a seus treinadores. Para isso, funcionalidades precisam ser implementadas para que atendam outros requisitos funcionais e não funcionais, como por exemplo: Funcionamento do sistema na Web (portabilidade); Gestão de usuários com controle de *login*; Gestão de atletas e equipes; Gestão de treinamentos; Gestão de vídeos (*upload*, identificação e deleção); Banco de dados Web.

4. Conclusões

Este trabalho desenvolveu um protótipo baseado em Visão Computacional para auxiliar na avaliação da técnica do esporte de Levantamento de Peso Olímpico. Foi realizada a revisão bibliográfica sobre o esporte, sobre a Visão Computacional e algumas de suas principais ferramentas, como a linguagem Python e o *framework* OpenCV. Sobre este, foi realizada uma pesquisa a cerca dos algoritmos de detecção e rastreamento de objetos presentes no *framework*. Também foi realizada uma análise de trabalhos correlatos na área de Visão Computacional aplicada ao esporte e à detecção do corpo humano. Por fim, foi feita a modelagem e planejamento geral de sistema.

Tudo isso culminou com o desenvolvimento de um protótipo que é capaz de receber um vídeo lateral de um *Snatch*, plotar um gráfico da trajetória da barra e informar a razão de força relativa do atleta para aquele levantamento. Esse software possui uma base para contínua expansão e acréscimo de novas funcionalidades para continuar contribuindo com o desenvolvimento do esporte do Levantamento de Peso Olímpico e o campo da Visão Computacional.

Referências

Babich, N. (2020). What is computer vision how does it work? an introduction. https://xd.adobe.com/ideas/principles/emerging-technology/what-is-computer-vision-how-does-it-work/. Acessado em outubro de 2021.

Campos, J., Poletaev, P., Cuesta, A., Pablos, C., and Carratalá, V. (2006). Kinematical analysis of the snatch in elite male junior weighlifters of different weight categories. In *Journal of Strength and Conditioning Research*, pages 843–850. National Strength Conditioning Association.

de Moraes, G. R. and Cantarelli, G. (2015). *Protótipo de aplicação para avaliação postural utilizando o Microsoft Kinect*. Trabalho Final de Graduação Curso Ciência Da Computação - Centro Universitário Franciscano, Santa Maria, RS, Brasil.

Franks, I. M. (1993). The effects of experience on the detection and location of performance differences in gymnastic technique. In *Research Quarterly for Exercise and Sport Vol. 64 No. 2*, pages 227–231. The American Alliance for Health, Physical Education, Recreation and Dance.

Gonzalez, R. C. and Woods, R. E. (2018). *Digital Image Processing*. Pearson, New York, 4th edition.

Granatyr, J. (2020). Curso visão computacional: o guia completo. https://iaexpert.academy/trilhas/trilha-visao-computacional. Acessado em outubro de 2021.

IBM (2021). What is computer vision? https://www.ibm.com/topics/computer-vision. Acessado em agosto de 2021.

IWF (2020). Weightlifting: The Two Lifts. https://iwf.sport/weightlifting_/the-two-lifts. Acessado em setembro de 2021.

Kusuma, M. N. H., Rilastia, D., Syafei, M., Nugroho, R., and Budihardjo, B. (2018). Biomechanical analysis of snatch technique in conjunction to kinematic motion of olympic weightlifters. In *Advances in Health Science Research Volume 12*, pages 132–137. Atlantis Press.

Lumenera (2022). Behind the scenes of today's image processing. https://www.infopedia.pt/dicionarios/lingua-portuguesa/refatorar. Acessado em maio de 2022.

MatPlotLib (2021). Matplotlib. https://matplotlib.org. Acessado em maio de 2022.

Ng, J. (2020). *Posture evaluation for variants of weight-lifting workouts recognition*. PhD thesis, UTAR.

Noriega, C. L. (2018). *Análise comparativa da execução técnica do Levantamento de Peso Olímpico entre praticantes com e sem experiência*. Programa de Pós-Graduação em Neurociências e Comportamento da Universidade de São Paulo, São Paulo, Brasil.

O'Donoghue, P. (2010). *Research Methods For Sports Performance Analysis*. Routledge, 270 Madison Avenue, New York, NY 10016.

OpenCV (2021). About. https://opencv.org/about. Acessado em agosto de 2021.

OpenCV (2022). Opencv modules. https://docs.opencv.org/4.x/index.html. Acessado em janeiro de 2022.

Sohani, A., Ullah, R., Rai, A., Karni, O., et al. (2018). Performing an'athletic movement assessment'for sports jump using state of the art video analysis and heuristics techniques like body detection and displacement assessment. *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*, 45(1):171–184.

Vorobyev, A. (1978). *A Textbook on Weightlifting*. International Weightlifting Federation.

Weightlifting Physics (2021). The physics of the snatch. https://thephysicsofsnatching.weebly.com/. Acessado em outubro de 2021.

Whittaker, E. T. (1937). *A treatise on the analytical dynamics of particles and rigid bodies*. CUP Archive. Disponível em https://books.google.com.br/books?id=epH1hCB7N2MC&printsec=frontcover&hl=pt-BR&source=gbs_ge_summary_r&cad=0v=onepage&q&f=false.

Wykowski, T. and Wykowska, J. (2019). Lessons learned: Using Scrum in non-technical teams. https://www.agilealliance.org/resources/experience-reports/lessons-learned-using-scrum-in-non-technical-teams. Acessado em agosto de 2020.

5. Apêndices

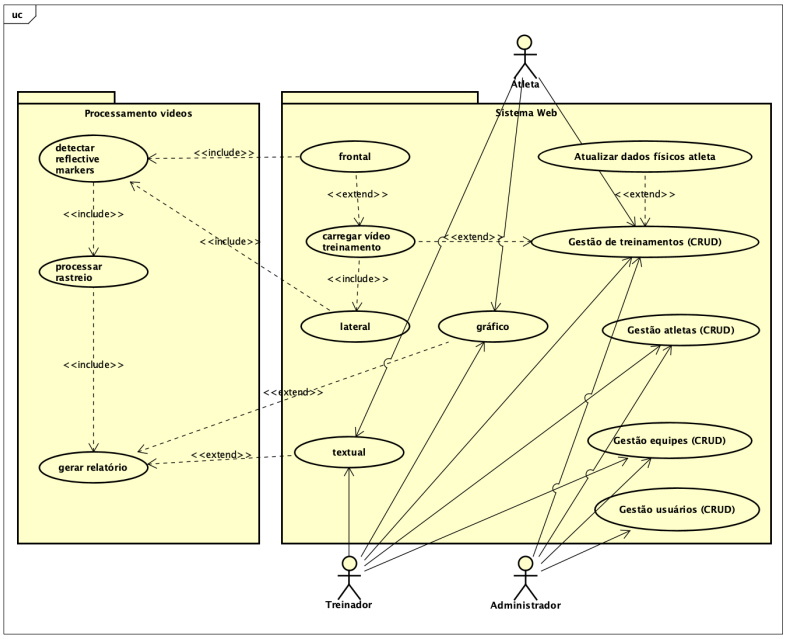

Figura 12. Diagrama Casos de Uso com os atores e as principais funcionalidades.

Note, que a proposta do sistema contém dois subsistemas: um para gerenciar usuários, atletas, treinadores, equipes e seus treinos (versão Web); outro para processar os vídeos carregados, ou seja, realizar os rastreios de movimento (Python-OpenCV) e gerar relatório (textual ou gráfico) sobre a trajetória da barra.

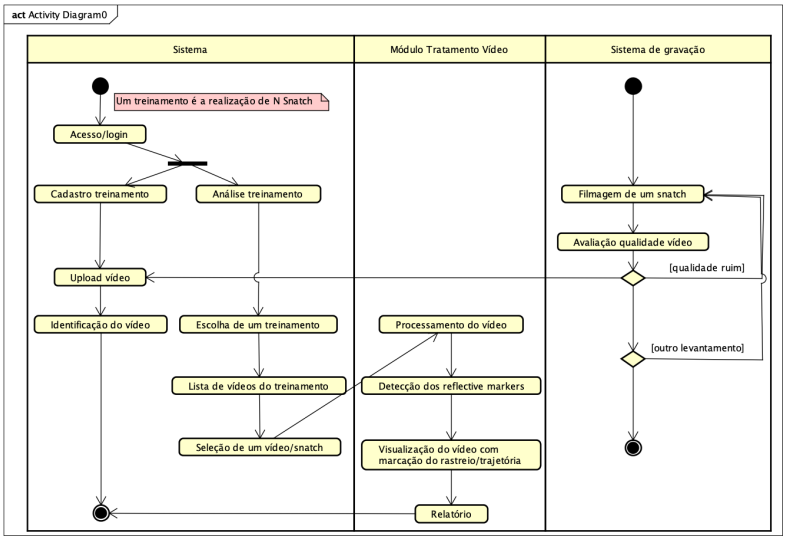

Figura 13. Digrama de Atividades com a visão geral de funcionamento do sistema.

Perceba que o diagrama ilustra a dinâmica de funcionamento de um sistema Web completo, não só o protótipo apresentado no texto. Nesse diagrama é possível visualizar os papéis do sistema, do módulo de tratamento do vídeo e do sistema de gravação.