UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

EDUARDO STEIN BRITO

Transcrição Musical Automática do Instrumento de Bateria a partir de Vídeos

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Bruno Castro da Silva

Porto Alegre
2019

“A man provided with paper, pencil, and rubber,
and subject to strict discipline, is in effect a universal machine.”

— Alan Mathison Turing

AGRADECIMENTOS

Agradeço o apoio da minha família — principalmente aos meus pais —, amigos e a minha noiva Andressa, que esteve ao meu lado durante todo o desenvolvimento deste trabalho.

RESUMO

Métodos para transcrição automática de música consistem em algoritmos para automaticamente extrair informações de tom e ritmo a partir de áudios e/ou vídeos. Diversas pesquisas e trabalhos desenvolvidos na área propões técnicas eficazes, mas ainda existe espaço para melhorias e algoritmos mais sofisticados. Este trabalho propõe implementar uma nova solução para a transcrição musical automática do instrumento de bateria a partir da análise de vídeos de pessoas tocando este instrumento, e propõe atingir tal objetivo combinando técnicas de processamento de imagem e informações espaciais sobre a movimentação do corpo do músico e as peças da bateria que podem estar sendo tocadas a cada momento. O presente trabalho foca na transcrição do instrumento de bateria a partir *apenas* de vídeo, mas não áudio; essa suposição é importante pois a bateria é um instrumento cujos sons são frequentemente sobrepostos e não possuem notas bem definidas, como no caso de instrumentos de corda, o que pode dificultar o uso de técnicas baseadas em áudio e análise de frequência para transcrição. Em particular, uma das dificuldades de transcrever automaticamente músicas deste instrumento é que até mesmo as partituras não descrevem as músicas por notas musicais, e sim por qual tambor ou prato deve ser tocado em dado instante. Além disso, ao contrário de instrumentos de corda, nos quais o número de cordas é fixo, a quantidade de tambores e pratos em uma bateria é variável. Nossa implementação supera tais obstáculos com o uso de informação espacial, tais como a determinação de onde estão as peças da bateria e a verificação automática de qual peça foi tocada em cada momento. Além disso, se o áudio estiver disponível, nossa solução poderá ser usada em conjunto com técnicas atuais de transcrição baseadas em áudio. Neste trabalho, através do uso de filtros e máscaras, com auxílio de algoritmo de estimação de pose, foi possível realizar a transcrição de trechos de vídeos de músicos tocando o instrumento de bateria. Iremos apresentar, neste trabalho, as tecnologias empregadas, assim como resultados experimentais, limitações do trabalho e possíveis aplicações do método proposto.

Palavras-chave: Transcrição automática de música. aprendizado de máquina. processamento de imagens. instrumento de bateria. estimação de pose.

Automatic Music Transcription of the Drum Instrument from Videos

ABSTRACT

Methods for automatic music transcription consist of algorithms to automatically extract tone and rhythm information from audios and/or videos. Several research developed in the field propose effective techniques, but there is still room for improvement and more sophisticated algorithms. This work proposes to implement a new solution for the automatic musical transcription of the drum instrument from the analysis of videos of people playing this instrument, and proposes to achieve this goal by combining image processing techniques and spatial information about the movement of the musician's body and the parts of the drum that may be being played at any moment. The present work focuses on the transcription of the drum instrument from video only, but not audio; this assumption is important because the drum is an instrument whose sounds are often superimposed and do not have well-defined notes, contrary to the case of string instruments, making it difficult to use audio-based techniques and frequency analysis for transcription. In particular, one of the difficulties of automatically transcribing songs of this instrument is that even the music sheets do not describe the songs by musical notes, but by which drum or cymbal should be played at a given moment. In addition, unlike stringed instruments, in which the number of strings is fixed, the number of drums and cymbals in a drum kit is variable. Our implementation overcomes such obstacles with the use of spatial information, such as determining where the drum parts are and the automatic checking of which part has been touched at each time. In addition, if audio is available, our solution can be used in conjunction with current audio-based transcription techniques. In this work, through the use of filters and masks, with the aid of a pose estimation algorithm, it was possible to perform transcriptions of videos of musicians playing the drums instrument. We will show, in this work, the technologies employed, limitations of our technique, as well as experimental results and possible applications of the proposed method.

Keywords: automatic music transcription, machine learning, image processing, drum instrument, pose estimation.

LISTA DE ABREVIATURAS E SIGLAS

AMT	Automatic Music Transcription
HMM	Hidden Markov Model
SVM	Support Vector Machine
BG	Background
FG	Foreground
HSL	Hue Saturation Luminance
RGB	Red Green Blue
FN	False Negative
FP	False Positive
TP	True Positive
TN	True Negative
FPS	Frames Por Segundo

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Transcrição automática de música (AMT, do inglês *automatic music transcription*) é o processo de converter um sinal musical acústico ou um vídeo em alguma forma de notação musical. Considere o seguinte cenário: um aprendiz de algum instrumento musical deseja aprender a tocar um trecho de sua música favorita a partir da observação de um músico tocando-a. Sem ter muita experiência, e ainda tendo o desafio de aprender a música visualizando os movimentos rápidos de um músico, é necessário que o aprendiz perceba quais notas estão sendo tocadas, o intervalo entre as notas e ainda conseguir transcrever isso a fim de poder reproduzir tais notas musicais posteriormente. Este é um cenário no qual é possível a aplicação da técnica proposta para transcrição automática de música, sem necessidade de intervenção por parte do usuário.

Algumas abordagens para este problema já foram desenvolvidas, tais como, o uso de Aprendizado Profundo (*Deep Learning*) para identificar notas a partir de arquivos de áudio (SARNATSKYI et al., 2017). No entanto, muitas destas soluções são focadas apenas no uso de áudio para a extração dos dados, possuindo limitações, como pouca eficácia para casos com múltiplas notas tocadas simultaneamente. Pensando nisso, propomos e desenvolvemos uma abordagem, com o uso apenas de vídeo, para a transcrição de músicas tocadas em baterias. A técnica pode ser considerada complementar às existentes, pois não exclui a possibilidade do uso conjunto com técnicas para AMT baseadas na análise de áudio.

Para atingir o objetivo de automaticamente transcrever músicas tocadas em bateria, serão usadas — para obter informações de um vídeo — técnicas que permitam obter informação espacial sobre os movimentos do músico de forma a extrair-se as notas tocadas em uma bateria com diversos tambores e pratos. Dois problemas principais serão abordados usando técnicas/algoritmos de aprendizado de máquina (ou *machine learning*, em inglês) para construir a solução proposta: rastreamento de objeto e estimação de pose.

O trabalho restringe-se a obter as notas musicais extraídas do tocar de uma bateria a partir de um vídeo. Isso se deve ao fato de que os movimentos envolvidos ao se tocar este instrumento são mais explícitos se comparados com vídeos de pessoas tocando instrumentos de corda, por exemplo. Além disso, existe uma ampla gama de vídeos que poderão ser usados para a validação da solução proposta. Enquanto métodos baseados em áudio analisam frequências para identificar qual nota está sendo tocada, no caso de baterias isto não é possível, uma vez que não se afina uma peça — como um prato, por

exemplo — para que ele tenha uma frequência especifica.

Desta forma, o trabalho diferencia-se da maioria dos trabalhos de AMT encontrados na literatura, pois diferente de métodos que buscam realizar a transcrição através de áudio (GOWRISHANKAR; BHAJANTRI, 2016), nosso método busca apenas através de vídeo transcrever o que foi tocado. Algo semelhante já foi feito para instrumentos cordados (GOLDSTEIN; MOSES, 2018) e também para um instrumento de percussão (MARENCO et al., 2015), no entanto existem diferenças importantes entre estes trabalhos e o nosso apresentado. Enquanto para instrumentos de corda a extração da música pode ser feita através da frequência das notas tocadas, não é possível fazer isto para o instrumento de bateria, uma vez que devemos extrair qual peça foi tocada e não qual nota. Já a diferença do nosso trabalho para os já existentes na literatura que executam a transcrição de instrumentos de percussão, temos o diferencial de propor uma técnica que funciona de forma flexível independente do número de peças de uma bateria através do uso de rótulos que seriam dadas a regiões selecionadas na imagem que representariam a localização dos instrumentos. Ademais, não é necessário nenhum treinamento no nosso algoritmo proposto.

Nosso processo vai combinar métodos de processamento de imagens para subtrair o fundo de cada quadro do vídeo e remover informações irrelevantes nas imagens, combinando com o uso de *pose estimation* para saber a localização próxima das baquetas — pois estarão próximas as mãos e braços do músico — de forma a tentar identificar a localização espacial das baquetas e depois, utilizando métodos como filtro de Kalman e heurísticas próprias, realizar o rastreamento temporal das baquetas a fim de estimar quando elas colidem com algumas das peças da bateria. Detectada as colisões nas peças, as transcrevemos em uma saída que representa a música tocada. Neste trabalho não focamos da transcrição do bumbo da bateria, uma vez que em geral, este fica escondido nas imagens analisadas.

O método será avaliado testando em pequenos trechos de vídeos de bateristas tocando o instrumento, em que será executada nossa solução e que ao final da execução do algoritmo será gerado um arquivo de saída. Este arquivo de saída, não é a partitura da música em si, mas a informação de qual peça da bateria foi tocada em dado instante de tempo em qual *frame*. As colisões detectadas serão comparadas com uma saída ideal, manualmente anotada, descrevendo em quais *frames* exatos ocorreram as colisões. Será analisado principalmente a acurácia, precisão, sensibilidade e F1 Score obtidas em cada vídeo. Através desses experimentos, demonstramos a eficácia do nosso método e também discutimos suas características e limitações, assim como identificamos trabalhos futuros.

1.1 Estrutura

O trabalho está estruturado da seguinte forma. No Capítulo 2, revisamos a base teórica necessária para compreensão do método proposto neste trabalho. No Capítulo 3, revisamos publicações anteriores que também visam realizar AMT, principalmente através do uso de vídeo. Já no Capítulo 4, encontram-se as validações e experimentos que foram realizados a fim de identificar quais técnicas serão combinadas para compor a técnica proposta. No Capítulo 5, mostramos a maneira que a solução proposta foi desenvolvida, baseada nas técnicas validadas do Capítulo 4. Finalmente, o Capítulo 6 apresenta os resultados experimentais obtidos através das técnicas propostas e compara diferentes configurações das implementações. O Capítulo 7 conclui este trabalho delineando seus principais resultados, as características do método proposto, e discutindo trabalhos futuros.

2 EMBASAMENTO TEÓRICO

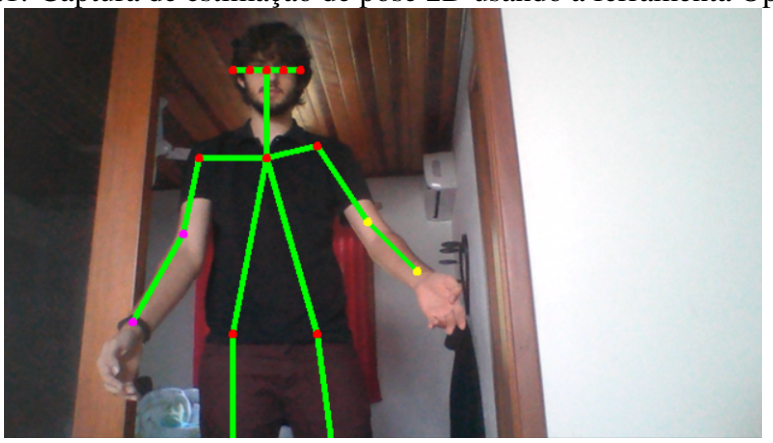
Neste capítulo, revisamos as principais duas formas de uso de aprendizado de máquina aplicados a nosso problema e também o uso de processamento de imagens, as quais serão posteriormente combinadas para compor o método sendo proposto neste trabalho. Nós dividimos este capítulo em três abordagens com seus respectivos algoritmos. Realizamos uma comparação das características gerais das técnicas citadas, bem como a intuição básica de seus funcionamentos.

2.1 Estimação de Pose

Estimação de pose (*pose estimation*, em inglês) consiste na identificação de como um objeto (ou pessoa) está configurado no espaço 3D, tanto em seus ângulos, quanto nas orientações de suas juntas (SIMAS et al., 2007). A estimativa de pose humana é um problema difícil porque o corpo humano tem muitos graus de liberdade e articulações. Também é de suma importância conseguir superar algumas das dificuldades relacionadas a variação da pose devido roupas, formato do corpo, tamanho, iluminação, entre outros. Ademais, é preciso que os resultados sejam eficazes mesmo que partes do corpo se sobreponham, como, por exemplo, a mão de uma pessoa cobrindo parcialmente a sua perna ou algum outro objeto cobrindo parcialmente um membro do individuo.

Os problemas de *pose estimation* podem ser divididos em dois tipos: 2D *pose estimation* e 3D *pose estimation*. O primeiro estima uma posição 2D (x, y) de coordenadas para cada junta no espaço na qual a pessoa se situa, a partir de uma imagem. Já o segundo estima uma posição 3D (x, y, z) de coordenadas neste mesmo espaço métrico a partir de uma imagem.

Figura 2.1: Captura de estimação de pose 2D usando a ferramenta OpenPose



Fonte: obtido pelo autor, 2019.

A Figura 2.1 apresenta um exemplo no qual se detecta as juntas do corpo da pessoa, incluindo a posição dos braços, de forma que se possa mais facilmente determinar os movimentos por ela executados — por exemplo, ao tocar um instrumento musical, como a bateria.

2.1.1 Algoritmos de Estimação de Pose

O foco de nosso trabalho é solucionar o problema de transcrição de música utilizando apenas *input* visual. Algumas abordagens e diferentes algoritmos já foram desenvolvidos justamente com o propósito de estimar a posição de um objeto ou pessoa em um espaço 3D. Foram estudados diferentes algoritmos, alguns com código *open source*, os quais são passíveis de serem utilizados em em nossa implementação ou servirem como base de inspiração.

DensePose é um trabalho desenvolvido por pesquisadores do Facebook que tem como objetivo obter, a partir de imagens RGB, uma representação da pose humana baseada na superfície do corpo humano, com um resultado que estes pesquisadores chamam de *dense human pose estimation* (GÜLER; NEVEROVA; KOKKINOS, 2018). Nesta técnica foi utilizado o *COCO dataset* — um conjunto de dados de detecção, segmentação e anotações de objetos em grande escala — e a partir dele foi introduzido um novo *pipeline* de anotações que explora informações de superfície 3D durante a anotação para 50 mil imagens do *dataset*, gerando um novo conjunto de dados chamado de *DensePose-COCO*. Usando o conjunto de dados resultante, sistemas de redes convolucionais foram treinados, fornecendo uma correspondência de densidade na estimação de pose — isto é, o *output*

não é composto apenas por “palitos” com juntas, como o exemplo da Figura 2.1, e sim uma espécie de *grid* que envolve toda a superfície dos membros do indivíduo. Foram usadas arquiteturas convolucionais e também sistemas baseados em regiões, sendo que a segunda arquitetura obteve desempenho superior. O sistema fornece resultados altamente precisos em tempo real e é capaz de lidar com escala, variações de pose além de poder “alucinar” as juntas do o corpo humano por trás de roupas como vestidos ou saias.

Para nossa proposta, o uso desta tecnologia poderia ser utilizada para realizar o *tracking* das mãos do individuo tocando o instrumento para, de alguma forma, extrair qual o tambor sendo tocado. O problema desta técnica é que esta não faz o rastreamento das baquetas do músico, dificultando transcrição. Além disso, para o nosso problema, a informação de “densidade” extraída pela ferramenta é irrelevante. Extrair apenas os braços e mãos como palitos com suas juntas já seria o suficiente.

Outro trabalho de base lidando com o problema de *pose estimation* é o **OpenPose**, que é um sistema capaz de estimar a pose de múltiplas pessoas em tempo real, detectando os pontos principais do corpo humano, como mãos, face e pés. O código é *open source* e é disponibilizado no *GitHub* [https://github.com/CMU-Perceptual-Computing-Lab]. Este trabalho foi baseado inicialmente no artigo *Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields* (CAO et al., 2017). A abordagem usa uma técnica que os autores denominaram de Part Affinity Fields (PAFs) para aprender a associar partes do corpo a indivíduos na imagem. Utilizando redes convolutivas com uma arquitetura multi-estágio de dois *branches* (ramos), inicialmente é feita uma análise que gera um mapa de *features*. Essas *features* são usadas em cada um dos ramos, que refinam as previsões ao longo de etapas sucessivas, com supervisão intermediária em cada etapa. Cada etapa no primeiro ramo prevê mapas de confiança e cada etapa no segundo ramo prevê *affinity fields*. Desta forma, um dos ramos acaba prevendo todos mapas de confiança, uma representação 2D da confiança de que uma determinada parte do corpo ocorre em cada localização de pixel e o outro prevê os *affinity fields*, que codificam o grau de associação entre as partes do corpo. Este trabalho, desenvolvido em 2017, conseguiu um aumento de 13% no MAP (do inglês *Mean Average Precision*) — que é a média das precisões máximas em diferentes valores de *recall* — se comparado com métodos anteriores considerados estado da arte. Para que esta solução seja utilizada para rastrear as baquetas do músico é necessário que seja indicado de alguma forma um *offset* da mão do músico, sinalizando a posição na qual a ponta da baqueta deve estar. Com esta informação, aliado aos rótulos colocados nas peças da bateria, seria possível detectar o que foi tocado.

Na solução proposta neste trabalho foi utilizado OpenPose, não só pelo bom desempenho em pequenos testes realizados, mas também pela sua integração com a principal biblioteca utilizada na solução final, descrita no capítulo 5.

2.2 Object Tracking

O rastreamento de objetos (ou *object tracking* em inglês), pode ser definido como o problema de estimar a trajetória de um objeto no plano da imagem enquanto ele se move em uma cena (YILMAZ; JAVED; SHAH, 2006). Outra possível definição é que o objetivo do rastreamento de objetos é segmentar uma região de interesse a partir de uma cena de vídeo e acompanhar movimentos de objetos, seus posicionamentos e oclusões (PAREKH; THAKORE; JALIYA, 2014). Existem diversos desafios associados ao rastreamento de objetos, por exemplo, conseguir lidar com movimentos bruscos do objeto, possível movimento da câmera, objetos que não são rígidos e mudanças na cena. De forma geral, a localização do objeto é feita a cada quadro. Realizando um comparativo com algoritmos de detecção de objetos — que têm apenas o interesse de identificar e localizar todos os objetos conhecidos em uma cena — os algoritmos de rastreamento, em geral, são mais rápidos. Isso porque quando se está rastreando um objeto que foi detectado no quadro anterior, sabe-se muito sobre a aparência do objeto, além da direção e velocidade de seu movimento. Logo, é possível usar este tipo de informação para prever melhor a localização do objeto no próximo quadro.

As principais técnicas de *object tracking* são três: *Point tracking*, *Kernel Tracking* e *Silhouette Tracking*. Na primeira técnica, o rastreamento pode ser feito a partir da correspondência de objetos detectados sendo representados por pontos nos quadros. Já *Kernel tracking* é baseado no movimento do objeto. Normalmente, este processo é realizado calculando-se o movimento do objeto, que é representado por uma região, de um quadro para outro. Os algoritmos usados deste tipo de técnica podem diferir no número de objetos rastreados e no método usado para estimar o movimento do objeto. Por fim, *Silhouette Tracking* é usado quando a região completa de um objeto é necessária. Objetos de formas complexas, por exemplo, as mãos, podem ser descritas com precisão por este método. O objetivo da técnica é encontrar a região do objeto em cada quadro por meio de um modelo de objeto gerado usando os quadros anteriores. Este modelo é justamente a previsão de para onde o objeto está e irá se mover no futuro.

Um problema comum que pode acontecer ao se rastrear objetos é o desvio do

modelo. Isto consiste no fenômeno em que o alvo — objeto de interesse que está sendo feito o rastreamento — gradualmente se afasta do modelo no rastreamento. Este desvio é atribuído ao acúmulo de pequenos erros introduzido na localização do modelo cada vez que o modelo é atualizado. Uma atualização mais rápida do modelo resulta em um desvio de modelo maior. Desta forma é necessário saber equilibrar a renovação frequente do modelo — pois é necessária para mantê-lo atualizado com a mudança da aparência do alvo — com a integridade do mesmo em face de erros de desvio causados justamente pela a atualização rápida (PAN; HU, 2007).

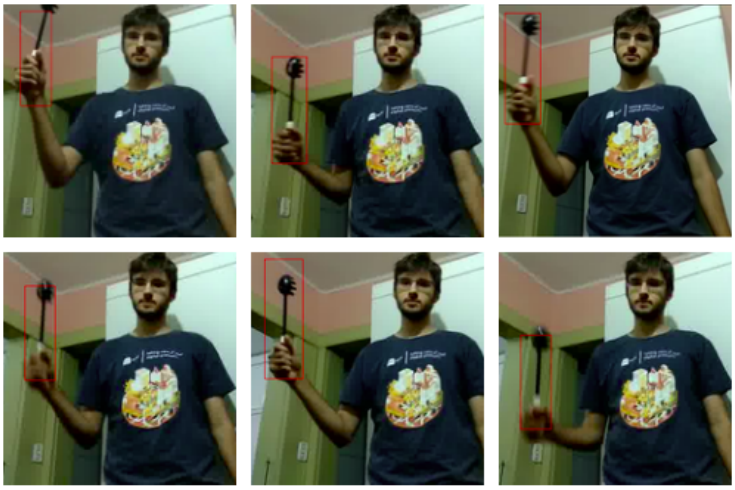
2.2.1 Algoritmos de Object Tracking

Os algoritmos aqui descritos foram avaliados para encontrar o melhor para rastrear as baquetas do músico e posteriormente ser feita a transcrição da música tocada. O resultado desta experimentação está descrito no capítulo 4.

Dentre as algoritmos de *object tracking* que encontramos na literatura, e que são implementadas em bibliotecas *open source*, alguns se destacam. Um destes é o vencedor do Visual Object Tracking Challenge de 2014, que teve seu código de implementação disponibilizado. Neste trabalho foi utilizada uma abordagem com o uso do aprendizado de filtros de correlação discriminativa — usados para detecção de padrões. Hoje, é possível encontrar em bibliotecas um *correlation tracker* baseado na implementação deste trabalho de 2014 (DANELLJAN et al., 2014), que faz uma separação de filtros para que seja possível estimar de forma independente a escala do objeto alvo após a tradução de onde está o alvo ser encontrada. A Figura 2.3 mostra um exemplo de um rastreamento de um utensilio doméstico usando o algoritmo de *object tracking* previamente descrito.

Temos também o trabalho de pesquisadores da *Graz University of Technology* (GRABNER; GRABNER; BISCHOF, 2006). Neste trabalho foi desenvolvido um algoritmo baseado no algoritmo AdaBoost (FREUND; SCHAPIRE; ABE, 1999). O classificador usa o plano de fundo na etapa de atualização, que tenta assim evitar o problema do desvio. Outro algoritmo é o MIL apresentado no trabalho *Visual tracking with online multiple instance learning*, que treina um classificador para separar o objeto do seu fundo em tempo real (BABENKO; YANG; BELONGIE, 2009). A aprendizagem do classificador tenta evitar o problema de desvio. Existe também o algoritmo de Median Flow (KALAL; MIKOLAJCZYK; MATAS, 2010). Neste, o rastreamento é adequado para movimentos muito suaves e previsíveis do objeto e quando o mesmo é visível em toda a sequência de

Figura 2.2: Teste de rastreamento de objeto usando a biblioteca Dlib



Fonte: obtido pelo autor, 2018.

frames. É bastante preciso para esse tipo de problema superando o algoritmo MIL.

Existem também alguns *frameworks* — ou seja, uma implementação que junta algumas soluções já prontas –, sendo um destes o TLD (KALAL et al., 2012). Este é um *framework* de rastreamento que decompõe explicitamente a tarefa de rastreamento de longo prazo em rastreamento, aprendizado, e detecção. O rastreador segue o objeto de quadro a quadro, enquanto o detector localiza o objeto e corrige o rastreador, se necessário. Já na etapa de aprendizado, são calculados os erros do detector e este é atualizado para evitar estes erros no futuro. O algoritmo Median Flow (KALAL; MIKOLAJCZYK; MATAS, 2010) é usado como o componente de rastreamento nesta implementação. Com isso, esta implementação é capaz de lidar com movimentos rápidos, oclusões parciais, ausência de objetos entre outros problemas.

2.3 Filtros e Máscaras

Além das técnicas usadas para resolver os problemas de *pose estimation* e *object tracking*, descritas nas seções anteriores, a compreensão do presente trabalho também se beneficia de uma apresentação rápida de técnicas de processamento de imagens baseadas em filtros e máscaras. Estes são importantes para conseguir remover detalhes indesejados nas imagens e também destacar aquilo que importa. Dado o problema de extrair a informação do que está sendo tocado em uma bateria, por exemplo, é de suma importância remover informações irrelevantes, que podem atrapalhar a detecção correta dos instrumentos tocados.

2.3.1 Filtros para Subtração de Fundo

Subtração de fundo é uma técnica utilizada para gerar uma máscara de primeiro plano (ou seja, uma imagem binária contendo os pixels pertencentes a objetos em movimento na cena) usando-se câmeras estáticas. Está técnica é usada para conseguir remover fundos estáticos em um vídeo. Isto é de interesse neste trabalho, pois com isso é possível, por exemplo, apenas filtrar pelo o que está se movendo na imagem e, por consequência, deixar na imagem apenas a movimentação do músico e de suas baquetas. Esta técnica funciona calculando a máscara através de uma subtração entre o quadro atual e um modelo de fundo, o qual contém a parte estática da cena dadas as características da cena observada.

2.3.1.1 Mistura Gaussiana - MOG

O modelo de mistura gaussiana é dado pela soma ponderada de densidades gaussianas que descreve a probabilidade de cada pixel pertencer ao fundo da imagem. Os pesos desta soma são proporções de tempo que cores permanecem na cena. Ou seja, nesta técnica — baseada em pixels — basicamente necessita a decisão de se dado pixel pertence ao fundo (*background* em inglês, BG) ou algum objeto em primeiro plano (*foreground* em inglês, FG). As cores de fundo prováveis são as que ficam por mais tempo em cena e de forma mais estática. Por consequência disso, esta é uma técnica que **usa o histórico dos pixels** para determinar se dado pixel pertence ou não ao fundo.

Para exemplificar, um pixel no tempo pode ser denotado por $\vec{x}^{(t)}$, que representa um vetor das cores básicas que compõe aquele pixel, naquele instante de tempo. Em um caso geral, não sabemos nada sobre os objetos em primeiro plano nem quando e com que frequência eles estarão presentes. Para decidir que um pixel pertence ao plano de fundo temos que ter que:

$$p(\vec{x}^{(t)}|BG) > cthr \tag{2.1}$$

onde cthr é um valor de *threshold*, um valor limite. O modelo de fundo é estimado a partir de um conjunto de treinamento indicado como χ, o qual é composto de amostras de pixels que pertencem ou não ao fundo.

Sabendo que, na prática, a iluminação na cena pode mudar, um novo objeto pode ser trazido para a cena ou um objeto presente pode ser removido, é necessário se adaptar a mudanças. Assim, para se adaptar, o conjunto de treinamento χ deve ser atualizado,

adicionando novas amostras e descartando as antigas. Dada essas informações, no modelo de Mistura Gaussiana, se escolhe um período de tempo razoável τ, e assim, dado o tempo *t* temos que $\chi\tau = \{\vec{x}^{(t)}, ..., \vec{x}^{(t-\tau)}\}$. Para cada nova amostra atualiza-se o conjunto de dados de treinamento $\chi\tau$ e se estima novamente a probabilidade de um pixel no tempo pertencer ou não ao fundo, considerando o período de tempo do conjunto de dados de treinamento. Essa estimativa é feita por um somatório de componentes Gaussianos. (ZIVKOVIC et al., 2004).

2.3.2 Filtros para Detecção de Linhas

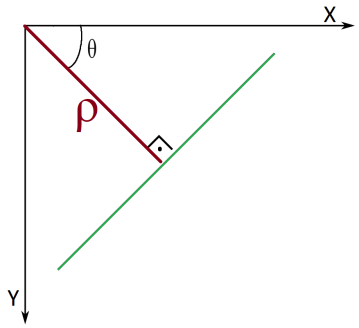
Um detector de linhas é um algoritmo que detecta linhas em uma dada imagem. Este é um dado interessante, no contexto do presente projeto, pois pode ser possível por exemplo, detectar onde é possível que esteja a baqueta do músico, uma vez que a baqueta é possivelmente, na imagem, uma linha reta. Assim, utilizando da informação de linhas retas pode-se aplicar uma máscara sobre a imagem para apenas aparecer estas linhas, e por consequência, as baquetas do músico.

2.3.2.1 Transformada de Hough Probabilística

Uma linha pode ser representada como *y = mx + c* ou em forma paramétrica, como $\rho = xcos\theta + ysin\theta$ onde ρ é a distância perpendicular da origem até a linha, e θ é o ângulo formado por essa linha perpendicular e eixo horizontal. Dessa forma, qualquer linha pode ser representada em dois termos (ρ, θ). A Transformada de Hough funciona usando um acumulador para manter contadores que são representados por dois parâmetros (ρ, θ). O método usa uma matriz destes acumuladores em que as linhas representam os possíveis ρ e as colunas denotam o θ. Assim, o tamanho da matriz depende da precisão desejada. Para uma precisão de 1 grau, são necessárias 180 colunas. Para ρ, a distância máxima possível é o comprimento diagonal da imagem. Então, para ter a precisão de um pixel, o número de linhas pode ser o comprimento diagonal da imagem.

Para entender esta transformada, é importante saber o que é o espaço Hough. Cada ponto (ρ, θ) no espaço Hough (que é representado pela matriz) corresponde a uma linha de ângulo θ e a distância ρ da origem no espaço de dados original. O valor de uma função no espaço Hough fornece a densidade do ponto ao longo de uma linha no espaço de dados. Então, para cada pixel, considera-se todas as linhas que passam por esse ponto em

Figura 2.3: Representação em (ρ, θ) de uma linha reta



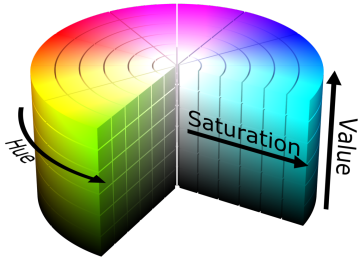
Fonte: confeccionado pelo autor, 2019.

um determinado conjunto discreto de ângulos; ou seja, se a precisão for de 1 grau, são 180 valores de ângulos testados. Para cada ângulo θ, é usada a equação $\rho = xcos\theta + ysin\theta$ para determinar a distância ρ da origem até a linha, ou seja, o valor de ρ. Para cada linha considerada, incrementamos uma contagem (inicializada em zero) no acumulador de Hough no ponto (ρ, θ) da matriz. Depois de considerar todas as linhas através de todos os pontos, um acumulador de Hough pertencente a matriz que possui um valor alto provavelmente corresponderá a uma linha de pontos. A Transformada de Hough Probabilística é uma otimização da Transformada de Hough. Ele não leva todos os pontos em consideração; em vez disso, leva apenas um subconjunto aleatório de pontos, o suficiente para a detecção de linha (MATAS; GALAMBOS; KITTLER, 2000).

2.3.3 Filtros Baseados na Análise de Cores

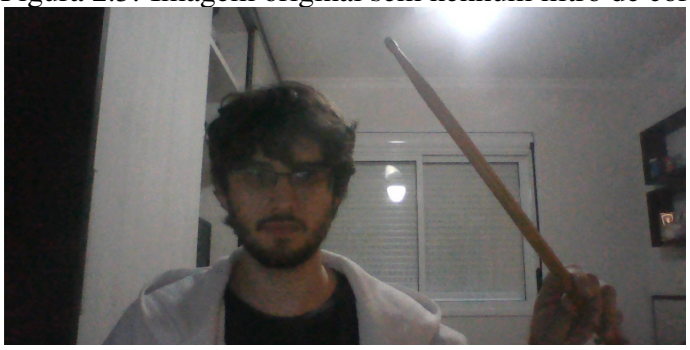
Um filtro de cor serve para remover ou destacar cores em uma dada imagem. Grande parte dos filtros de cores funcionam no sistema de cor HSL — *hue* (matiz, tonalidade), *saturation* (saturação) e *value* (valor) — em oposição ao sistema mais comumente conhecido, o RGB — *red* (vermelho), *green* (verde), *blue* (azul). Esse esquema de cores é preferencialmente utilizado uma vez que é considerado mais compatível com a percepção humana de cor (Sobottka; Pitas, 1996). Nestes filtros, ajustando os valores máximos e mínimos de tonalidade, saturação e valor, é possível destacar apenas um conjunto especifico de cores em uma dada imagem.

Figura 2.4: Modelo de cores HSV mapeado para um cilindro



Fonte: obtido online (Wikimedia Commons, 2013).

Figura 2.5: Imagem original sem nenhum filtro de cor



Fonte: obtido pelo autor, 2019.

Figura 2.6: Aplicado filtro na Figura 2.5 com HSV mínimo (0,35,54) e HSV máximo de (42, 254, 107)

Fonte: obtido pelo autor, 2019.

2.3.4 Filtros para Estimativa de Localização de um Objeto

Diversos métodos existem para tentar estimar, com base em amostras coletas ao longo do tempo, a posição mais provável de um objeto em movimento. Nesta seção, focamos no Filtro de Kalman. Este filtro é um conjunto de equações matemáticas que fornece uma solução computacional eficiente do método de mínimos quadrados, que é uma técnica que procura encontrar o melhor ajuste para um conjunto de dados tentando minimizar a soma dos quadrados das diferenças entre um valor estimado e os dados reais. O filtro suporta estimativas de estados passados, presentes e futuros — estados como posição e velocidade, por exemplo —, podendo fazê-lo mesmo quando o sistema modelado é desconhecido (WELCH; BISHOP et al., 1995). O filtro é útil para, por exemplo, dado um conjunto de pontos em que estimamos que a baqueta do baterista está, prever onde possivelmente estará a baqueta no próximo instante de tempo.

Esta técnica funciona estimando o estado do processo em um dado momento e obtém *feedback* na forma de medições ruidosas. Assim, este filtro aplica equações para projetar para frente — no tempo — as estimativas de estado e erro atuais para obter as estimativas para o próximo intervalo de tempo. Outras equações são responsáveis por incorporar novas medidas na estimativa para obter uma estimativa melhorada (WELCH; BISHOP et al., 1995).

3 TRABALHOS RELACIONADOS

Nesta seção, descrevemos alguns trabalhos relacionados ao nosso — i.e., técnicas para execução de AMT de forma automática. Foram feitas pesquisas em trabalhos que transcrevem música tanto a partir de áudio quanto de vídeo, com o foco principal em instrumentos de percussão.

3.1 Soluções Baseadas em Transcrição de Áudio

Muitas das soluções existentes para extrair notas musicais a partir de áudios trabalham apenas com áudios de notas isoladas, algo menos complexo que uma música com múltiplos instrumentos e vocais misturados. Os métodos utilizados para a transcrição variam no entanto, a maioria dos métodos de transcrição empregam o Modelo Oculto de Markov (HMM, *Hidden Markov Model*) na extração de notas (GOWRISHANKAR; BHAJANTRI, 2016). A ideia deste modelo é determinar parâmetros desconhecidos/ocultos a partir dos parâmetros observáveis. Em alguns trabalhos desenvolvidos usando HMM, em um dado instrumento cordado, a corda na qual a nota é tocada é considerada um estado oculto que precisa ser estimado, dadas *features* observadas e a frequência fundamental obtida (MAEZAWA et al., 2012). Já na classificação das notas de piano, por exemplo, o método de Support Vector Machine (SVM) é o classificador mais utilizado (GOWRISHANKAR; BHAJANTRI, 2016). SVM é um método supervisionado que necessita de um conjunto de treinamento de entradas e saídas correspondentes, que no caso de piano, seria necessário treiná-lo com *datasets* de notas do instrumento para poder classificar.

Para música com apenas uma nota tocando em dado momento (monofônica) temos um caso que já possui soluções diversas, principalmente com técnicas baseadas em áudio quando esse é livre de ruídos. Todavia, na música polifônica, temos uma situação muito mais complicada, em particular para a transcrição em situações nas quais existe a presença de múltiplos instrumentos do mesmo tipo ou quando há ruído de fundo significativo. Em casos de transcrição de instrumentos com corda, outro desafio é que a mesma nota pode ser produzida por várias cordas. Assim, extrair a informação de qual corda foi tocada, que é outro aspecto importante da transcrição, não é trivial (GOLDSTEIN; MOSES, 2018). Nossa proposta possibilita a superação de tais problemas uma vez que o vídeo fornece uma visão espacial de como a música é executada, ao invés de apenas informação do som

produzido por sua execução; isso facilita a identificação do que está sendo tocado, uma vez que extrair qual peça foi tocada por dada baqueta é independente do que está sendo tocado pela a outra baqueta.

Nos casos de transcrição de notas tocadas por um instrumento de bateria, em particular, como proposto neste trabalho, existem alguns outros problemas associados. Os instrumentos de percussão são bem diferentes dos instrumentos com notas musicais bem definidas, como os instrumentos cordados. O toque dos tambores muitas vezes apresentam espectros de ruído. Isto, aliado ao fato de que as peças não são afinadas em notas específicas, fazem com que certos algoritmos adaptados a outros tipos de instrumento não sejam aplicáveis nos de percussão (WU et al., 2018). Outro problema é que os sons de bateria são normalmente sobrepostos uns aos outros, ou seja, diferentes peças da bateria são tocadas simultaneamente. Isso pode levar a situações ambíguas em que é difícil classificar automaticamente os eventos de som de bateria ou suas combinações (WU et al., 2018). Novamente, uma abordagem utilizando vídeo permite identificar de forma facilitada o toque de múltiplos tambores simultâneos, devido ao *tracking* que é possível realizar nas baquetas, através do uso da informação espacial obtidas por técnicas visão computacional. Por estas limitações focamos nesta revisão de literatura mais em métodos baseados em vídeo, dando menos atenção a métodos baseados na transcrição baseada em áudio — visto que o objeto de nosso trabalho é transcrição baseada em vídeos.

3.2 Soluções Baseadas em Transcrição de Vídeo

Um trabalho desenvolvido para a extração de música a partir de vídeo focou no uso da vibração de cordas de uma guitarra para a extração de áudio da mesma (GOLDSTEIN; MOSES, 2018). Usando vídeo sem áudio, capturado através de uma câmera montada no instrumento, é realizada a análise de cada corda e suas vibrações, de forma a obter-se como *output* uma partitura que permite que músicos toquem a música capturada visualmente. O grande problema com o uso deste tipo de técnica para extrair informação de uma bateria, que é o instrumento foco de nosso trabalho, é que este instrumento não é afinado da mesma forma que instrumentos de corda. As partituras inclusive não referenciam as músicas por notas musicais (dó, ré, mi, fá, sol, lá e si) e sim por qual tambor ou prato deve ser tocado naquele momento (chimbal, caixa, bumbo, pratos, etc), não existindo um padrão de afinação para o instrumento. Além disso, não se sabe a quantidade de tambores e pratos que terá a bateria, isso porque o músico que monta o instrumento pode, por exem-

plo, mudá-lo de uma música para a outra. Queremos propor algo mais flexível, que seja capaz de funcionar independentemente de quantas peças tenha a bateria e de sua afinação. Em particular, queremos extrair qual peça, não qual a nota tocada, caso contrário seria necessário saber a afinação de cada tambor de antemão para conseguir inferir a peça.

Dentre as soluções que usam vídeo, poucos trabalhos foram encontrados nos quais tem como objetivo transcrever música em instrumento de percussão usando informação visual, que justamente são trabalhos que enfrentam os problemas semelhantes aos que temos no nosso trabalho desenvolvido. Um destes trabalhos descreve e avalia uma nova abordagem na qual os sinais de vídeo gravados por uma câmera filmando um baterista são analisados a fim de melhorar o desempenho da transcrição de música (GILLET; RICHARD, 2005). O trabalho é um *follow-up* de um estudo anterior realizado na transcrição de bateria onde apenas *features* de áudio foram usados (GILLET; RICHARD, 2004). O sistema baseado apenas em áudio possuía um módulo de extração de *features* e um módulo de classificação para o qual vários classificadores (Modelo Oculto de Markov, Support Vector Machines) foram testados. Foi usado um *dataset* próprio que consiste em 35 sequencias contendo 2170 batidas no instrumento. Duas áreas das imagens de vídeo foram definidas e usadas no trabalho: uma área na qual o movimento está associado ao gesto executado pelo baterista para acertar o instrumento e a outra em que movimento está associado à vibração do próprio instrumento. Isto resultou em um conjunto de 18 *features* computadas. O sistema pode ser calibrado de forma manual — são definidas regiões da imagem correspondente a cada peça da bateria. A fusão de informações de vídeo e áudio é realizada por três abordagens diferentes: vetores de *features* tanto de áudio quanto vídeo, melhor dos especialistas — isto é, dois classificadores são treinados, um usando as *features* de áudio, o outro as *features* de vídeo e o que fornece o melhor score é mantido — e fusão — como o anterior, dois classificadores são treinados, exceto que estes classificadores produzem duas probabilidades sendo a classificação baseada na classe que maximiza o produto dessas duas. O resultado do trabalho foi um aumento na taxa de reconhecimento obtido com uma combinação de *features* de áudio e vídeo. O desempenho usando a combinação de ambos foi melhor do que com o uso apenas de áudio ou vídeo. Isso pode ser explicado pelo fato de que o processamento dos dados de áudio e vídeo no mesmo classificador permite tirar proveito de sua correlação.

Comparando este trabalho com a solução que iremos propor, este extrai *features* do vídeo para usar em um classificador. A diferença é que, em princípio, a nossa solução não requer treinamento com *datasets*. A ideia é usar informação espacial, como por exemplo a

localização das peças da bateria, assim como a posição em que se encontra as baquetas do músico, para que possamos determinar com exatidão qual peça foi tocada. A exploração do uso de informação espacial não foi explorada no trabalho existente descrito, além de que o mesmo funciona apenas para as peças usadas no treinamento, sendo menos flexível que nossa solução proposta.

Outro trabalho desenvolvido usa também uma abordagem multimodal — usa tanto áudio quanto vídeo — para a transcrição de música de percussão a partir de gravações (MARENCO et al., 2015). O foco do trabalho é para a transcrição do tambor do Candombe, um ritmo popular afro do Uruguai. Neste trabalho, informações de alto nível obtidas de uma modalidade são usadas para direcionar o processamento de ambas as modalidades, como por exemplo combinar a saída de um classificador monomodal com o outro, ou então, combinar as *features* de cada modo. Para obter informações significativas do vídeo, objetos relevantes na cena precisam ser detectados, ou seja, o bastão, a pele e a mão. Os eventos detectados são classificados em um conjunto de diferentes tipos de batidas — usando *dataset* próprio — combinando as informações de ambas as modalidades em um esquema de fusão em nível de *features*, semelhante ao trabalho citado anteriormente. Para detectar o tambor, usa-se filtragem de cores da imagem. A detecção do bastão é também a partir de filtragem de cor que posteriormente passa por um detector de segmento de linha. A detecção da mão que atinge o tambor é abordada pela segmentação da pele acima da cabeça do tambor.

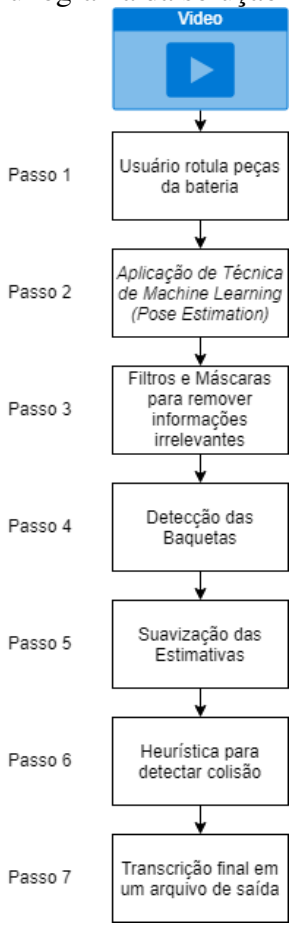
Com base na posição destas partes, várias *features* são criadas para descrever o tipo de movimento desses objetos durante um evento de áudio. O resultado deste trabalho foi que a abordagem multimodal supera os sistemas monomodais — só áudio ou só vídeo. Além disso, a vantagem também é perceptível para cada tipo de batida. Ainda, a baixa taxa de classificação alcançada pela modalidade de áudio para um caso em que foi usada uma afinação diferente do tambor, é compensada efetivamente no método que usa informações também de vídeo (MARENCO et al., 2015). Uma limitação deste trabalho é que ele foi testado apenas com o instrumento Candombe, que consiste em um único tambor. Nossa solução proposta é mais flexível neste sentido, pois permite que múltiplas peças, sejam estas tambores ou pratos, possam ser tocadas e detectadas individualmente através do uso de rótulos previamente fornecidos pelo usuário; para mais detalhes, ver Seção 5.1.

4 METODOLOGIA

A fim de desenvolver a solução proposta, iremos avaliar diversas técnicas e métodos que podem ser combinados e compostos a fim de construir um sistema completo para transcrição automática de músicas de bateria, com base em vídeos. Neste capítulo, iremos descrever a metodologia de testes que foram efetuados a fim de determinar quais algoritmos e heurísticas melhor se adaptam para compor a nossa solução. Resultados experimentais e análises da qualidade da solução resultante são discutidas no Capítulo 6.

A solução final proposta está descrita em mais detalhes no Capítulo 5, mas, de maneira breve, propomos que ela seja feita seguindo o fluxograma da Figura 4.1.

Figura 4.1: Fluxograma da solução final proposta



Fonte: obtido pelo autor, 2019.

Neste capítulo iremos investigar em mais detalhes quais técnicas/heurísticas vão ser utilizadas para implementar cada um dos passos/caixas no fluxograma, assim como a importância e necessidade de alguns passos desta solução. Vale ressaltar que algumas etapas de alguns passos, como por exemplo o uso de subtração de fundo no passo 3,

podem ser implementados diretamente por algoritmos de conhecida alta performance, e que portanto não exigem muita experimentação. Outros passos, como o passo 1, 5 e 7, não exigem validações, apenas serem implementados e serão descritos no próximo capítulo em detalhe. Alguns passos, entretanto, como o passo 2 e principalmente o passo 6 têm um impacto maior na qualidade da solução que está sendo desenvolvida, e iremos analisar, nesse capítulo, hipóteses e métodos de implementação.

4.1 Avaliação e Escolha de Algoritmos/Heurísticas

Iremos iniciar a análise sistemática sobre quais as melhores heurísticas e algoritmos a serem utilizados para compor nossa solução final pelo estudo de diferentes técnicas para detecção de colisões entre dois objetos. Nesta seção focamos na avaliação e escolhas dos algoritmos/heurísticas necessárias para os passos 2, 3, 4 e 6. Inicialmente acreditava-se que o passo 3 não seria necessário e que apenas Técnicas de *Machine Learning* usadas no passo 2 seriam suficientes para extraírem diretamente as informações do posicionamento das baquetas substituindo também o passo 4. As subseções 4.1.1 e 4.1.2 mostram a necessidade da existência e separação dos passos 3 e 4.

4.1.1 Validação de Técnicas Para Rastreamento da Baqueta: Problema de *Object Tracking*

Neste trabalho, consideramos duas possíveis formas de ser possível realizar o rastreamento temporal da posição das baquetas do músico: o uso técnicas para resolver o problema de *object tracking* e de técnicas para resolver o problema de *pose estimation*. Nesta seção, avaliamos primeiramente a performance de técnicas para resolver o problema de *object tracking*. Para validarmos esta hipótese realizamos testes utilizando a biblioteca OpenCV, que está um pouco mais detalhada no Capítulo 5. A ferramenta disponibilizava todos os algoritmos descritos na subseção 2.2.1. Realizamos os testes e em todos ocorreram algum tipo de problema sendo o problema do desvio de modelo o principal fator dos erros de rastreamento. Um exemplo de erro pode ser visto na Figura 4.2. Em todos os algoritmos uma determinada região era demarcada como o objeto de interesse e este era rastreado nos *frames* seguintes. Uma das principais causas de problema era o fato de que a cor da baqueta era muito semelhante a do braço do baterista, além de que o formato da

baqueta ao se mover muda, pois dependendo da perspectiva o tamanho da baqueta diminui, muda de ângulo, ou até mesmo quando a ponta está direcionada para a câmera pode se tornar apenas um ponto na imagem. Alguns dos algoritmos testados também não conseguiam se recuperar de oclusão. Por fim, os movimentos rápidos das baquetas causavam borrões no objeto em si, também prejudicando o rastreamento do objeto.

Principalmente por se tratar de algoritmos genéricos de *tracking* estes não estavam adaptados para rastrear especificamente um bastão — no caso, as baquetas. Por todos estes motivos, descartamos o uso dos algoritmos já existentes de *object tracking* na ferramenta para realizar o rastreamento na baqueta e decidimos que um algoritmo próprio para o rastreamento seria necessário. Portanto, este algoritmo próprio não irá diretamente aplicar *object tracking*, e sim de uma combinação de métodos de *pose estimation* (descritos na seção seguinte) com heurísticas de processamento de imagens.

Figura 4.2: Na sequência: imagem original, trecho selecionado para realizar o *tracking* e por fim o resultado após alguns poucos *frames* rastreando, apresentando o problema do desvio de modelo

Fonte: obtido pelo autor, 2019.

4.1.2 Validação de Técnicas Para Rastreamento da Baqueta: Problema de *Pose Estimation*

Inicialmente o objetivo do uso de algoritmos que resolviam o problema de *pose estimation* seria para conseguir diretamente inferir onde estão as baquetas do músico. Nesta seção isso se mostrou não viável e assim, o uso de algoritmos para *pose estimation* é de forma mais auxiliar, ajudando na obtenção de estimativas sobre a posição da ponta da baqueta. Os motivos pelos quais algoritmos de *pose estimation* não resolvem o problema de rastrear completamente as baquetas estão descritos nesta seção.

Usando a ferramenta OpenCV (descrita em detalhes no Capítulo 5), foi possível realizar a estimação de pose, usando-a em conjunto com a biblioteca OpenPose (CAO et al., 2018). Não houve a necessidade de treinarmos o algoritmo, pois foi usada uma rede neural pronta, já treinada usando o *framework* TensorFlow, pois OpenCV fornece um módulo de Rede Neural Profunda que permite carregar diretamente um modelo de rede neural deste *framework*. Esta rede neural utilizada, é uma rede mais simples se comparada com outras redes neurais que estão disponíveis para uso. Esta rede permite processamento em tempo real na CPU ou dispositivos embarcados de baixo consumo de energia (KIM, 2013), e isto já atendia nossas necessidades e não exigia muito da máquina utilizada para os experimentos. Nestes experimentos, foi possível obter a informação da localização dos pulsos do baterista, que estão próximos das baquetas.

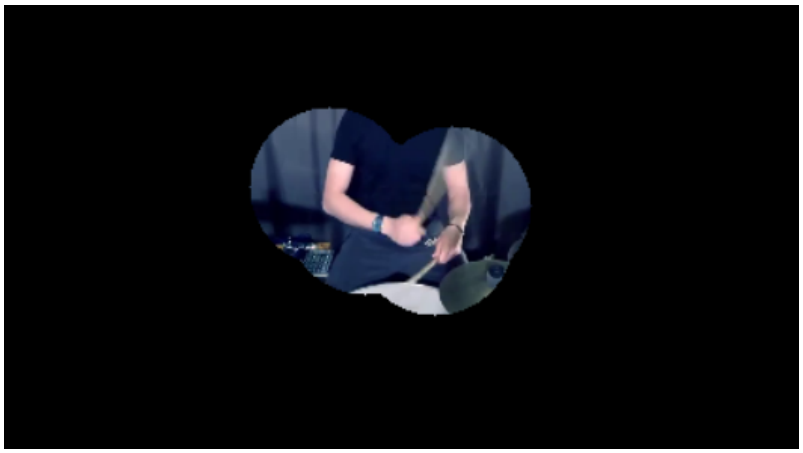
Figura 4.3: Estimação de pose do baterista obtida via OpenCV com OpenPose e TensorFlow

Fonte: obtido pelo autor, 2019.

Desta forma, com a posição dos pulsos do baterista, podemos criar uma máscara que pode filtrar a imagem original, como uma forma de auxiliar o *tracking* das baquetas,

uma vez que é possível aplicar uma máscara na imagem para que apenas as regiões próximas as mãos e braços sejam analisadas para o *tracking*. Pelos experimentos realizados, a conclusão é que o algoritmo de *pose estimation* será usado apenas no auxílio para a obtenção da informação espacial das baquetas, ou seja, uma forma de ajudar em descobrir em que posição encontram-se as mesmas, mas o método por si só não consegue definir onde as baquetas estão. Isso porque mesmo sabendo onde estão os pulsos do baterista fica difícil inferir para em qual posição encontra-se a baqueta. Ademais, em testes realizados as vezes as informações da localização dos pulsos eram perdidas por alguns momentos, de forma que dificultaria o uso apenas disto para saber onde estão as baquetas. Assim, não só as informações dos pulsos são utilizadas, mas também as informação sobre as posições dos braços, criando uma máscara que filtra as regiões próximas a estas partes do corpo, como pode ser visto na Figura 4.4.

Figura 4.4: Máscara aplicada usando informações obtidas pela estimação de pose visto na Figura 4.3



Fonte: obtido pelo autor, 2019.

Assim, se faz necessário o uso de mais filtros e máscaras para que detecte-se de fato onde que a baqueta está, novamente reforçando a necessidade do desenvolvimento de uma algoritmo próprio — que será descrito no capítulo 5 — que rastreie as baquetas, usando assim, o auxílio do algoritmo para *pose estimation*.

4.1.3 Detectando Colisões: Cálculo Vetorial

Nesta seção iremos validar qual a melhor forma de execução do passo 6 do fluxograma previamente apresentado, ou seja, como a partir dos dados do posicionamento da baqueta podemos inferir que ocorreu uma colisão em uma dada peça da bateria. Iremos

assumir que a posição de cada objeto, ao longo do tempo, será representada por suas coordenadas em 2D, na imagem relativa àquele momento no tempo. A detecção de colisões envolve o desafio de que as medições da coordenadas dos objetos ao longo do tempo se dá de forma ruidosa, e que as observações 2D correspondem a projeções em duas dimensões das trajetórias de objetos se movendo em 3D, o que dificulta a detecção de colisões. Uma hipótese de nossa solução é que partindo de uma informação perfeita da localização espacial das baquetas em vídeo, principalmente de suas pontas, e dada a localização das peças pré-rotuladas em cena pelo usuário, conforme passo 1, seria possível detectar as colisões baseando-se no em cálculos vetoriais, verificando quando ocorre uma variação brusca no deslocamento das baquetas ao longo do tempo. Isso ocorre pois, havendo uma colisão entre a baqueta e uma peça da bateria, há uma alteração abrupta na trajetória 2D sendo seguida por ela.

Esta ideia baseia-se em que calculando o vetor que indica a direção de movimento da baqueta em três momentos consecutivos no tempo, e verificando-se o ângulo entre esses vetores, pode-se verificar, por exemplo, que se o ângulo for zero, não houve alteração na direção. Já se o ângulo for 180 graus, houve inversão completa da direção sendo seguida pela baqueta. Ou seja, se para um tempo t, temos a ponta da baqueta no ponto (x_1, y_1) e depois, em um tempo $t + 1$ temos esta em um outro ponto (x_2, y_2) podemos calcular o vetor, considerando o primeiro ponto a origem e o segundo ponto calculado com a seguinte equação:

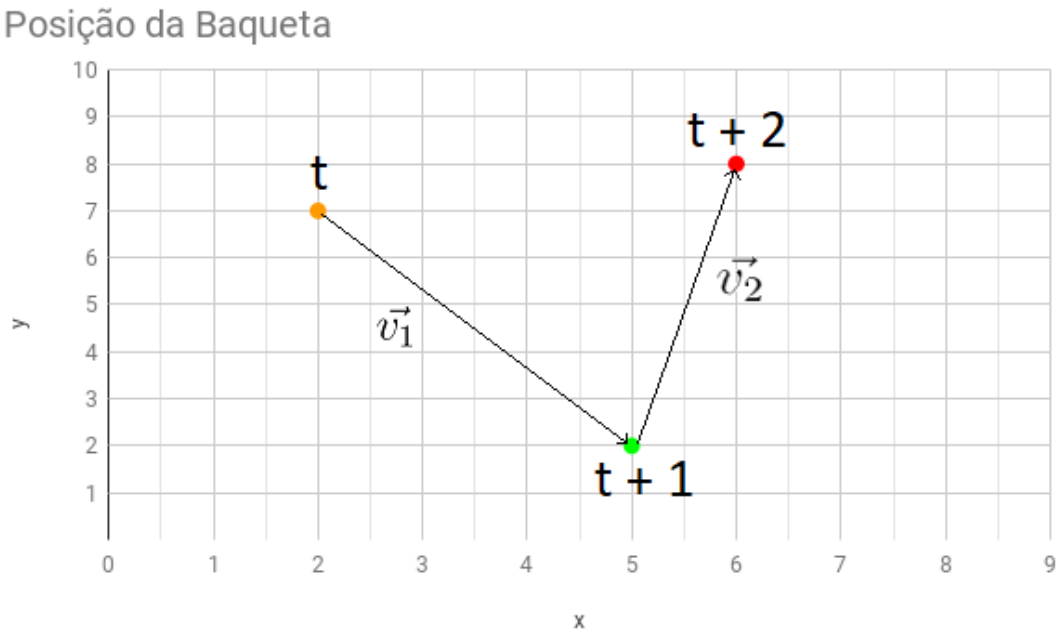
$$\vec{d} = (x_2 - x_1, y_2 - y_1) \tag{4.1}$$

Dado três pontos (x,y) que representam as posições da ponta de uma das baquetas nos instantes de tempo t, $t + 1$, $t + 2$, podemos obter dois vetores — $\vec{v_1}$ e $\vec{v_2}$, por exemplo –, um constituído pelos pontos dos instantes t e $t+1$ e o outro pelos instantes $t+1$ e $t+2$, conforme método previamente descrito. Com estes vetores calculados é possível calcular o ângulo entre eles. Considerando um vetor partindo da origem até o ponto (x_1, y_1) e o outro vetor partindo da origem até o ponto (x_2, y_2) temos um ângulo θ dado por:

$$\theta = \arccos \frac{((x_2.x_1) + (y_1.y_2))}{\sqrt{((x_1)^2 + (y_1)^2)}.\sqrt{((x_2)^2 + (y_2)^2)}} \tag{4.2}$$

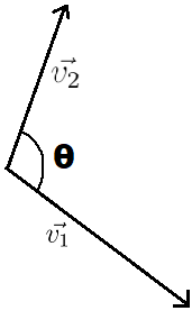
Estes vetores podem ser vistos melhor na figura 4.5. Na Figura 4.6 é mostrado o ângulo entre estes vetores θ.

Figura 4.5: Vetores $\vec{v_1}$ e $\vec{v_2}$ entre pontos dos instantes de tempo t, $t+1$, $t+2$



Fonte: obtido pelo autor, 2019.

Figura 4.6: Ângulo entre os vetores $\vec{v_1}$ e $\vec{v_2}$ obtidos conforme a Figura 4.5



Fonte: obtido pelo autor, 2019.

Partimos de uma hipótese que caso ocorra uma variação brusca do ângulo denotado por estes vetores que serão obtidos através dos pontos que identificam a localização da baqueta no tempo, podemos definir um *threshold* que caso ultrapassado caracterize o acontecimento como uma possível colisão. Se esta possível colisão ocorreu sobre uma área previamente definida na qual se encontra uma peça do instrumento, classificamos o evento ocorrido como uma colisão nesta peça. A partir desta hipótese foram utilizados três trechos de vídeos com a câmera posicionada de frente para o baterista e outros três trechos com a câmera em posição lateral para validar se era de fato possível a extração da informação da colisão por esta heurística descrita. Cada trecho corresponde a um clipe de no máximo 3 segundos em que o baterista toca o instrumento musical. Para a validação,

foram manualmente anotadas, *frame* por *frame* de cada clipe, as posições (x, y) em que estava a ponta da baqueta direita — e apenas direita, já que é o suficiente para validar a detecção da colisão via cálculo vetorial — simulando assim uma situação em que teríamos a informação perfeita da localização espacial da mesma. Também, em cada trecho, foi mapeada a região em que se encontrava cada uma das peças da bateria que eram tocadas em dado trecho. Esta etapa de mapear as regiões das peças seria justamente o passo 1 do fluxograma, em que o usuário rotula as peças da bateria previamente, sendo assim possível detectar em qual peça da bateria ocorreu uma colisão.

Figura 4.7: *Frames* dos dois vídeos analisados

Fonte: obtido pelo autor, 2019.

De posse da localização da ponta da baqueta da mão direita em cada trecho do vídeo, foram realizados os cálculos dos vetores, nos quais verificamos os ângulos entre eles; vetores que tiveram uma variação maior que um determinado *threshold* sugerem que, naquele momento no tempo, pode ter havido uma colisão. Caso essa tenha sido em uma região onde se encontra uma peça da bateria, consideramos o evento como uma colisão e, dada a posição original da baqueta, conseguimos extrair em qual tambor ou prato tal evento ocorreu.

No Apêndice A encontram-se os resultados, em detalhe, para cada um dos seis trechos analisados. As tabelas neste apêndice mostram os cálculos das variações dos ângulos resultantes dos cálculos vetoriais para que caso ocorra uma variação relevante — ou seja, maior que um *threshold* definido — seja verificado se ocorreu colisão usando a informação da posição original da baqueta naquele instante.

Sumarizando os resultados obtidos em cada um dos trechos analisados, temos a seguinte matriz de confusão que descreve a distribuição de falsos positivos (FP, *false positive*), falsos negativos (FN, *false negative*), verdadeiro positivo (TP, *true positive*), verdadeiro negativo (TN, *true negative*) nas colisões detectadas pela heurística do cálculo vetorial.

Tabela 4.1: Matriz de confusão com o resultado das detecções de colisão via heurística do cálculo vetorial

		Diagnóstico verdadeiro	
		Colidiu	Não colidiu
Dado obtido via heurística	Colidiu	34	2
	Não colidiu	6	241

Fonte: obtido pelo autor, 2019.

A acurácia desta heurística — que é a proporção de resultados verdadeiros entre o número total de casos examinados — é de:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} = \frac{34 + 241}{34 + 241 + 2 + 6} = 0.9717 \tag{4.3}$$

Também podemos calcular os valores de precisão — é a fração de instâncias recuperadas que são relevantes — e sensibilidade — que é a fração de instâncias relevantes que são recuperadas.

$$Prec = \frac{TP}{TP + FP} = \frac{34}{34 + 2} = 0.9444 \tag{4.4}$$

$$Sens = \frac{TP}{TP + FN} = \frac{34}{34 + 6} = 0.85 \tag{4.5}$$

Com estes dados, podemos calcular por fim a métrica chamada de F1 *score*, que seria a média harmônica da precisão e da sensibilidade, sendo uma métrica que equilibra sensibilidade e precisão:

$$F1 = 2.\frac{Prec.Sens}{Prec + Sens} = 2.\frac{0.9444.0.85}{0.9444 + 0.85} = 0.8947 \tag{4.6}$$

Obtivemos uma acurácia alta e também um ótimo F1 Score de maneira que **confir-**

mamos o funcionamento da heurística — além de verificarmos que ela funciona tanto para vistas laterais quanto frontais. Isso indica que o problema agora pode ser **reduzido ao problema de se obter informação mais precisa possível sobre as coordenadas da ponta da baqueta**, a fim de fornecê-las para a heurística acima.

4.2 Determinação de Métricas de Performance

O segundo grande passo metodológico necessário para o desenvolvimento deste trabalho diz respeito à escolha de quais experimentos serão conduzidos, como serão conduzidos, e quais métricas de performance iremos utilizar para determinar a qualidade da solução proposta.

Para os experimentos da solução desenvolvida serão selecionados clipes retirados do YouTube em que músicos tocam o instrumento de bateria. Pela maneira como os dados são analisados, foi possível perceber que o tamanho ótimo para ser analisado por vez é de trechos contíguos de 5 segundos. Esta limitação pode ser estendida/melhorada conforme será discutido na seção de trabalhos futuros 7.1. No total serão analisados pelo menos dois vídeos, cada qual com no mínimo três clipes de diferentes momentos da música. Ao menos um clipe terá a câmera posicionada com uma visão lateral do baterista e um clipe com visão frontal do baterista, de forma que seja possível verificar em quais cenários obtivemos melhores resultados.

Para cada clipe será rotulado manualmente em que posição se encontram as peças da bateria para ser possível verificar se houve colisão ou não em dada peça. Além disso, manualmente — em cada clipe — será necessário verificar em quais *frames* houve de fato colisão com peças da bateria e em qual peça. Esta informação é essencial, pois constitui o *ground truth*, que representa a informação considerada verdadeira sobre as colisões no instrumento para o clipe analisado. Assim, serão comparadas estas informações ideais com os resultados da nossa solução ao analisarmos os mesmos clipes, para verificando a qualidade do algoritmo desenvolvido.

As métricas utilizadas para aferir a qualidade da solução final desenvolvida serão as mesmas utilizadas na validação da heurística do cálculo vetorial descrita na Seção 4.1.3. Para isso, após executarmos nosso algoritmo para cada um dos clipes, usaremos as informações de *output* e — comparando com as informações ideais de colisão para cada clipe — faremos a contagem de TP, TN, FP e FN para cada um dos vídeos. É possível que as detecções encontradas estejam defasadas por alguns poucos *frames*. Por este motivo,

consideramos uma margem aceitável de 3 *frames* a mais ou a menos ao comparar com o resultado ideal. Consideramos esta quantidade de *frames* para vídeos de 30 FPS, em que este erro na detecção significa um atraso ou adiantamento de apenas 0.1 segundos na detecção.

Pela própria forma como o instrumento é tocado, sabemos que existem diferenças nas quantidades de movimentos que cada uma das mãos/baquetas executam ao tocar uma música. Por este motivo a análise é separada entre os resultados obtidos no rastreamento da baqueta segurada pela mão dominante e pela mão não dominante, sendo considerada a mão dominante aquela que se movimenta mais e dita o ritmo da música tocada. Além disso, serão executados 3 vezes o algoritmo para cada clipe em cada mão, pois é possível que existam variações nos resultados devida a marcação manual do *ground truth* que pode variar um pouco de uma execução para a outra e também por causa do rotulamento dos instrumentos que pode também variar um pouco devido a ser um processo manual.

4.3 Considerações

Primeiramente, o uso de *pose estimation* não resolve o problema da detecção das baquetas e colisões como um todo, mas ao menos indica aonde a baqueta provavelmente está e isto já auxilia no processo que os métodos de *object tracking*, por si só, não conseguiram resolver. Por consequência, no método final proposto será obtido essa estimativa grosseira da região da baqueta via estimação de pose e processar aquela região com métodos de processamento de imagem, descritos no capítulo 2.3, a fim de estimar a localização da baqueta ao longo do tempo. Desta forma, essencialmente estamos construindo um método próprio para *object tracking*, especificamente ajustado para rastreamento de baquetas, e que é constituído de uma combinação de *pose estimation* com filtros de imagem.

Portanto, o algoritmo próprio para rastrear a baqueta e sua ponta será baseado no uso de detector de linhas, subtração de fundo e filtro de cores, entre outras técnicas, além do uso da estimação de pose. O método proposto e sua implementação estão descritos em detalhes no capítulo 5.

5 MÉTODO PROPOSTO

Neste capítulo descrevemos o método utilizado na nossa solução. É feito o detalhamento das etapas utilizadas no processamento das imagens, realizadas em combinação com o uso algoritmos de *pose estimation*, e também como, a partir das informações obtidas, geramos uma saída no algoritmo que descreve as colisões da baqueta com as diferentes peças da bateria.

Para o desenvolvimento da solução escolhemos a biblioteca *Open Computer Vision Library*, ou OpenCV, que é uma ferramenta *open source* de visão computacional e *machine learning*. Esta ferramenta possui milhares de algoritmos já prontos e parametrizáveis, incluindo algoritmos de subtração de fundo, filtro de cores, detectores de linhas, rastrear objetos em movimento, entre outros. A biblioteca possui interfaces para as linguagens C++, Python, Java e MATLAB, suportando Windows, Linux, Android e Mac OS. Devido a facilidade e recursos úteis para o desenvolvimento de nossa solução, optamos pelo seu uso.

Para que seja possível a geração de uma saída do algoritmo que descreva as colisões nas peças da bateria são necessários diversos passos que compõe o processamento para se chegar no resultado final da solução. Esta sequência de etapas constitui-se em: 1) rotular áreas no vídeo analisado indicando a localização das peças de interesse para estimar possíveis momentos em que há colisão da baqueta com uma peça; 2) usar técnicas de processamento de imagens para identificar a localização das baquetas; feito através dos seguintes sub-passos: 2.1) subtração de fundo; 2.2) *pose estimation* para identificação da localização do punho do músico, e identificação da área na qual a baqueta pode se encontrar; 2.3) detector de linhas para detectar a baqueta em si a partir do resultado dos processamentos anteriores; 2.4) filtro de cores para remoção de dados restantes irrelevantes; 2.5) extração da posição das pontas das baquetas; 2.6) suavização dos resultados obtidos; 3) aplicar heurística para detecção da colisão usando como base a localização da baqueta e as áreas rotuladas das peças de interesse; 4) geração de um arquivo de saída com os dados sobre as colisões detectadas. Nas seções e subseções que seguem serão detalhados cada um dos passos de processamento que a solução proposta utiliza.

5.1 Seleção das Peças da Bateria

Para conseguir detectar a colisão usando a heurística já validada do cálculo vetorial, é necessário que o usuário indique, de antemão, as regiões da imagem na qual as peças de interesse se encontram. Por este motivo, através da ferramenta OpenCV, disponibilizamos para o usuário a possibilidade de selecionar regiões de interesse na imagem, permitindo que o usuário rotule um número variável de peças — e.g., de tons, caixas, pratos, etc.

Figura 5.1: Seleção de peças da bateria

Fonte: obtido pelo autor, 2019.

Estas informações são armazenadas e utilizadas somente quando detecta-se uma possível colisão devido a uma mudança brusca do ângulo entre os vetores obtidos via cálculo vetorial. Esta constitui a primeira etapa para que se possa aplicar nosso algoritmo proposto.

5.2 Estimando a Posição as Baquetas ao Longo do Tempo Utilizando Técnicas de Processamento de Imagens

Esta seção descreve os passos necessários para aplicar filtros de imagem nos *frames* originais do vídeo, para que restem apenas as informações essenciais necessárias à identificação/estimativa da localização das baquetas; essas informações serão, então, utilizadas pelos algoritmos descritos na Seção 5.2.6, os quais efetuam a análise de possíveis momentos no tempo nos quais há uma colisão da baqueta com uma peça da bateria.

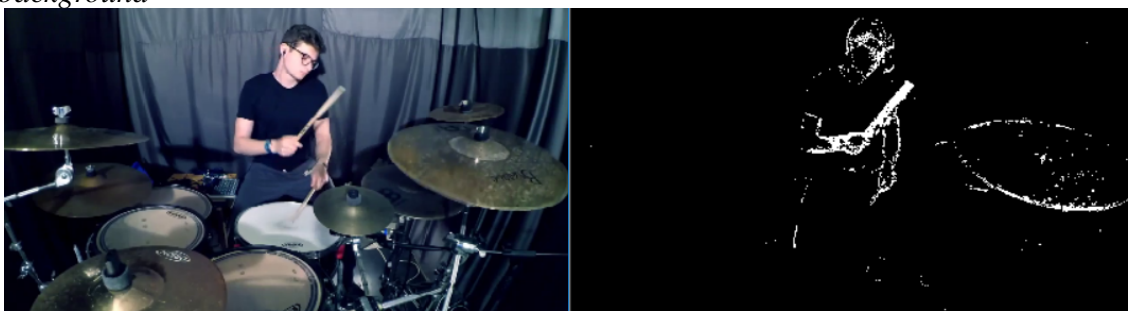
5.2.1 Subtração de Fundo

A primeira técnica de processamento de imagens usada no algoritmo desenvolvido é a aplicação da subtração de fundo. Isso irá permitir que apenas o que está se movimentando no vídeo fique em destaque — esse processo irá, portanto, destacar as baquetas e o corpo do músico, quando movimentados. Com um simples método usando a biblioteca OpenCV é possível criar um subtrator de fundo que usa o algoritmo de Mistura Gaussianas, descrito anteriormente. São necessários 3 parâmetros para a sua configuração que consiste em uma parâmetro que descreve a quantidade de *frames* usados no histórico do subtrator, um *threshold* que determina o quão sensível a mudança nos pixels deve ser para que considere ou não o mesmo pertencente ao fundo da imagem e por fim um parâmetro que indica se deve ser feita a detecção de sombras — que é uma informação completamente irrelevante em nossa aplicação:

```
1          cv2.createBackgroundSubtractorMOG2(history=100,
2          varThreshold=30, detectShadows=False)
```

Com este parâmetros é possível aplicar o subtrator *frame* a *frame* obtendo resultados como os da Figura 5.2. A imagem final serve como uma máscara que pode ser aplicada novamente na imagem original, para que mantenham-se as cores originais. No entanto, tal passo será apenas executado após mais filtros serem aplicados sobre os *frames* resultantes da subtração do fundo.

Figura 5.2: Na esquerda o *frame* original. Na direita com a aplicação da subtração de *background*



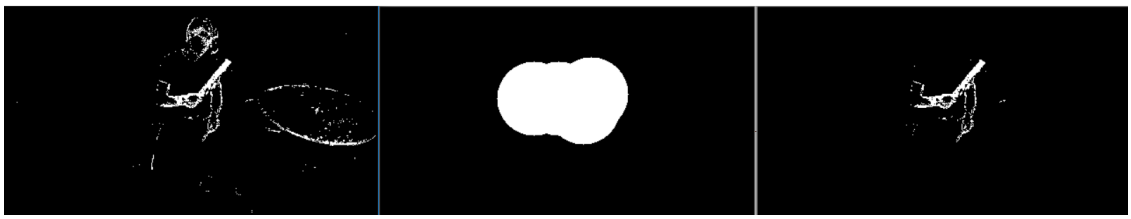
Fonte: obtido pelo autor, 2019.

5.2.2 Pose Estimation

Com a máscara de subtração de fundo pronta realizamos a junção com a máscara retornada pelo algoritmo de *pose estimation*, já descrita na seção 4.1.2, a qual, como

descrito naquela seção, corresponde a uma região selecionada da imagem na qual a baqueta provavelmente se encontra. Ao fazer isso, informações como o prato se mexendo e outros dados irrelevantes para o rastreamento da ponta da baqueta são removidos. Para efetuar a junção das duas máscaras, aplicamos uma operação booleana "AND" nas imagens resultantes dos filtros, nas quais "1" indica um pixel branco e "0" caso contrário; isso resulta em uma terceira imagem semelhante à apresentada na Figura 5.3. Nota-se claramente nesta figura que com a combinação de *pose estimation* e um filtro básico de processamento de imagens, já é possível reduzir o ruído na imagem e filtrar melhor as informações necessárias para identificação da posição da baqueta.

Figura 5.3: Aplicação da subtração de *background* com máscara de *pose estimation*



Fonte: obtido pelo autor, 2019.

5.2.3 Detector de Linhas

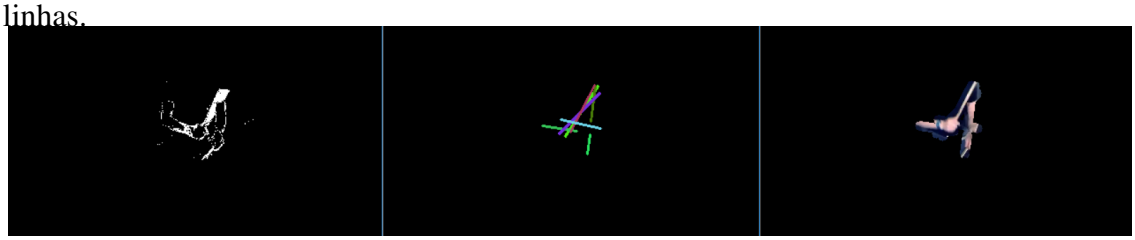
Dado que as informações agora estão filtradas apenas em regiões próximas as baquetas, aplicamos então um detector de linhas para identificar a baqueta propriamente dita em cada quadro do vídeo. O algoritmo utilizado é baseado na transformação de Hough Probabilística e está disponível na biblioteca OpenCV. Ao utilizar este recurso, é possível configurar o *threshold* do algoritmo, tornando este mais sensível para detectar as linhas; o máximo espaçamento entre pontos, para considerar linhas formada por pontos não perfeitamente contíguos; e também o tamanho mínimo de uma linha. Também é regulada a precisão da medida através de um parâmetro *rho*, o qual corresponde à resolução de distância do acumulador em pixels, e *theta*, que corresponde à resolução angular do acumulador em radianos. Parâmetros diferentes foram testados de forma empírica e encontrou-se a seguinte configuração, considerada adequada para alguns dos testes realizados:

```
1     minLineLength = 5
2     maxLineGap = 10
```

```
3       thr = 40
4       lines = cv2.HoughLinesP(frame, rho=1, theta=PI/180,
5       threshold=thr, minLineLength=minLineLength,
6       maxLineGap=maxLineGap)
```

Aplicando o detector de linhas à imagem resultante do processo descrito na Seção 5.2.2, obtivemos diversas linhas candidatas na região filtrada. Detectada estas linhas, é aplicado um filtro sobre a imagem original nos pontos próximos às linhas, de forma que se obtenha uma terceira imagem, exemplificada à direita na Figura 5.4.

Figura 5.4: Uso de detector de linhas para filtrar as baquetas do baterista. A figura à esquerda é a imagem previamente filtrada apenas com remoção do fundo e mantendo apenas a região identificada pelo *pose estimation*; a imagem do meio apresenta as linhas detectadas; e a terceira imagem corresponde ao resultado de se aplicar um filtro de detecção de linhas.

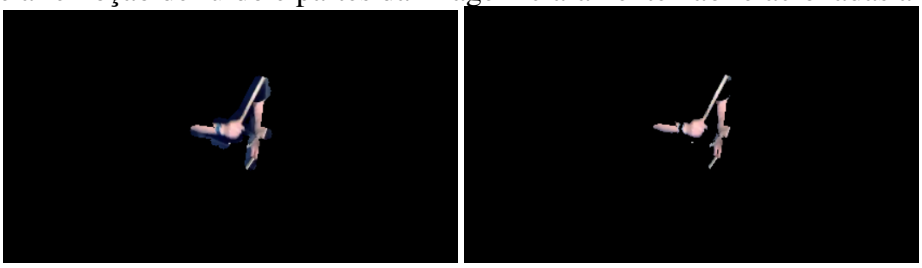


Fonte: obtido pelo autor, 2019.

5.2.4 Filtro de Cores

Com as baquetas filtradas através das técnicas anteriores, aplicamos mais um filtro à imagem. Neste caso, trata-se de um filtro opcional e que possivelmente deve ser ajustado de acordo com o vídeo em si. Este filtro é justamente um filtro de cores, conforme descrito na seção 2.3.3 e tem como objetivo filtrar informações irrelevantes que ainda podem ter restado mesmo após a aplicação dos demais filtros. Desta maneira, é possível remover, por exemplo, a camiseta do músico que está se movimentando e que tem cor diferente da baqueta. Ao aplicarmos o filtro de cores, de forma que sejam mantidas apenas regiões da imagem contendo cores próximas às cores de uma baqueta típica, temos o resultado apresentado na Figura 5.5.

Figura 5.5: À esquerda imagem sem o filtro de cores aplicado; e à direita, com o filtro de cor. Note a remoção de ruído e partes da imagem claramente não relacionadas à baqueta.



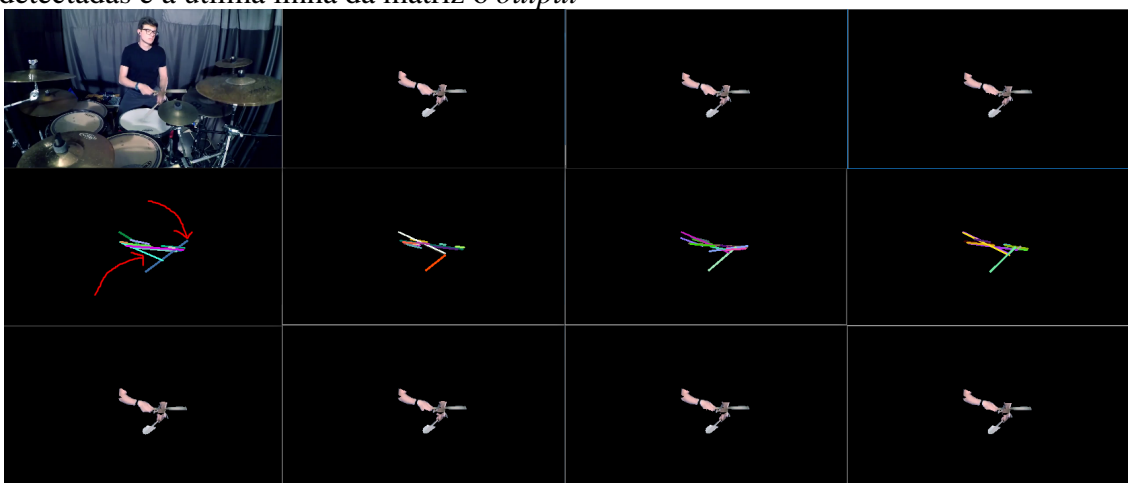
Fonte: obtido pelo autor, 2019.

5.2.5 Iterações

Com todos os filtros aplicados, sobra na imagem apenas a informação das baquetas e dos braços do músico — esses objetos sobram pois ambos estão em movimento, ambos possuem cores semelhantes, e ambos correspondem aproximadamente a linhas retas. Os braços ainda remanesceram uma vez que a cor é muito similar a da baqueta; caso contrário, o filtro de cores o teria eliminado da imagem. Depois destas filtragens, o *output* pós-filtros é usado novamente como *input* destes mesmos filtros, re-aplicando a sequencia de passos descrita anteriormente para que seja possível obter um conjunto melhorado de linhas candidatas; a melhoria se deve ao fato de que o processo é agora aplicado a uma imagem já fortemente filtrada e pré-processada. Este processo fornece um nova imagem de saída. É possível realizar múltiplas iterações como a descrita acima, de forma que as imagens resultantes podem ser novamente usadas como entradas à sequencia de filtros.

Apresentamos, na Figura 5.6, uma matriz de 12 imagens para demostrar o uso de filtros sendo aplicados de forma iterativa, conforme descrito acima. Cada coluna representa uma iteração, e, portanto, temos quatro iterações representadas na imagem. A primeira linha é o *input* da iteração. A segunda linha apresenta as linhas detectadas na imagem, e a última linha é o *output* que pode ser fornecido à próxima iteração. Em vermelho — na segunda linha, primeira coluna — temos flechas apontando para linhas erroneamente detectadas. Nota-se que estas linhas com problema desaparecem já na segunda iteração. Não é claro o benefício a partir da segunda iteração, e portanto, para a solução final foi escolhido o uso de apenas duas iterações de filtragem, uma vez que isso torna o algoritmo mais computacionalmente eficiente.

Figura 5.6: Matriz das iterações do algoritmo. Cada coluna representa uma iteração do algoritmo, sendo a primeira linha a imagem de *input* da iteração, segunda linha as linhas detectadas e a útlima linha da matriz o *output*



Fonte: obtido pelo autor, 2019.

5.2.6 Detectando a Ponta da Baqueta

Esta subseção descreve os passos necessários para, a partir da filtragem e da detecção de linhas, resultantes do processo descrito nas anteriormente, extrair a informação da localização estimada da baqueta e de sua ponta. Com os dados obtidos nesta etapa se torna viável a aplicação de uma heurística para determinar se houve colisão da baqueta com uma peça da bateria.

Com base em estimativas (possivelmente ruidosas) da posição da baqueta em um conjunto de tempos t, $t + 1$, ..., $t + k$, podemos utilizar diferentes métodos para estimar a posição mais provável da baqueta em $t + k + 1$. Caso $k = 0$, teríamos o caso de tentar prever a próxima posição da baqueta apenas com base na sua posição atual — o que pode ser difícil. Nessa seção, discutimos uma heurística que usa informação da posição em vários *frames* passados para que assim seja possível inferir onde possivelmente está a baqueta no novo instante de tempo $t + k + 1$.

Nossa solução baseou-se nesta informação temporal de posições estimadas prévias da posição da baqueta para conseguir extrair onde a baqueta em cada possível *frame*. Para tal, é necessário que sejam definidos em alguns poucos *frames* anteriores onde está a baqueta para que, a partir desta informação, nos próximos *frames*, seja possível inferir onde está a baqueta baseado no histórico de informações e nas linhas correntes detectadas.

Inicialmente, para o histórico não iniciar vazio, o que dificultaria muito para iniciar a inferir as posições da baqueta, se faz necessário que manualmente sejam populados em

alguns poucos *frames* esta informação. Alguns testes foram feitos e o número de *frames* que deve ser feita esta marcação manual para que a solução funcione varia entre 5 até 15 *frames*. Estas marcações chamaremos de *ground truth*, que representam uma informação verdadeira, não inferida, sobre onde estão as baquetas. Além disso, como já mencionado, existe um histórico da localização da baqueta que é salvo e que justamente inicia com os pontos do *ground truth*. Note que após o inicio do algoritmo, pós informado o *ground truth*, não é mais necessário informar manualmente a posição das baquetas.

Assim, de posse do *ground truth*, o que é feito é que detecta-se todas as linhas para um determinado *frame* para o qual não se conhece a posição das baquetas, conforme o processo de aplicação de filtros sucessivos citado na seção anterior. Então, dado o histórico de linhas, é verificada qual linha detectada no quadro atual está mais próxima das mesmas. Para esta escolha é calculada a distância entre os conjuntos de dois pontos para cada linha do histórico e os dois pontos que constituem cada uma das retas detectadas no *frame* atual.

Exemplificando: se usarmos um histórico de *5 frames*, dada as linhas detectadas em um *frame* corrente, será verificada a distância destas linhas até as linhas do histórico (formadas por dois pontos). Desta forma, a predição de reta — de todas obtidas pelo detector de linhas no *frame* atual — mais próxima de uma das linhas do histórico é considerada a baqueta detectada neste *frame*. E então esta linha é adicionada no histórico para ser usada na predição da baqueta no *frame* seguinte. Note que esta técnica assume que a baqueta no tempo $t + k + 1$ estará próxima de um dos pontos de t até $t + k$.

Existiam algumas variações nesta técnica proposta que poderiam ser utilizadas, como utilizar uma heurística que desse preferência aos pontos mais recentes do histórico e não considerar todos os pontos deste histórico com pesos iguais para inferir a baqueta no *frame* atual. No entanto, dado o histórico pequeno utilizado não foi aplicado tal heurística, e, sendo assim, foi assumido que todos os pontos do histórico são igualmente válidos para determinar onde deve estar a baqueta no *frame* analisado. Ademais, em nossa técnica existe também um processo de extrapolação/predição da posição futura da ponta da baqueta que será mais detalhado na Seção 5.2.7.

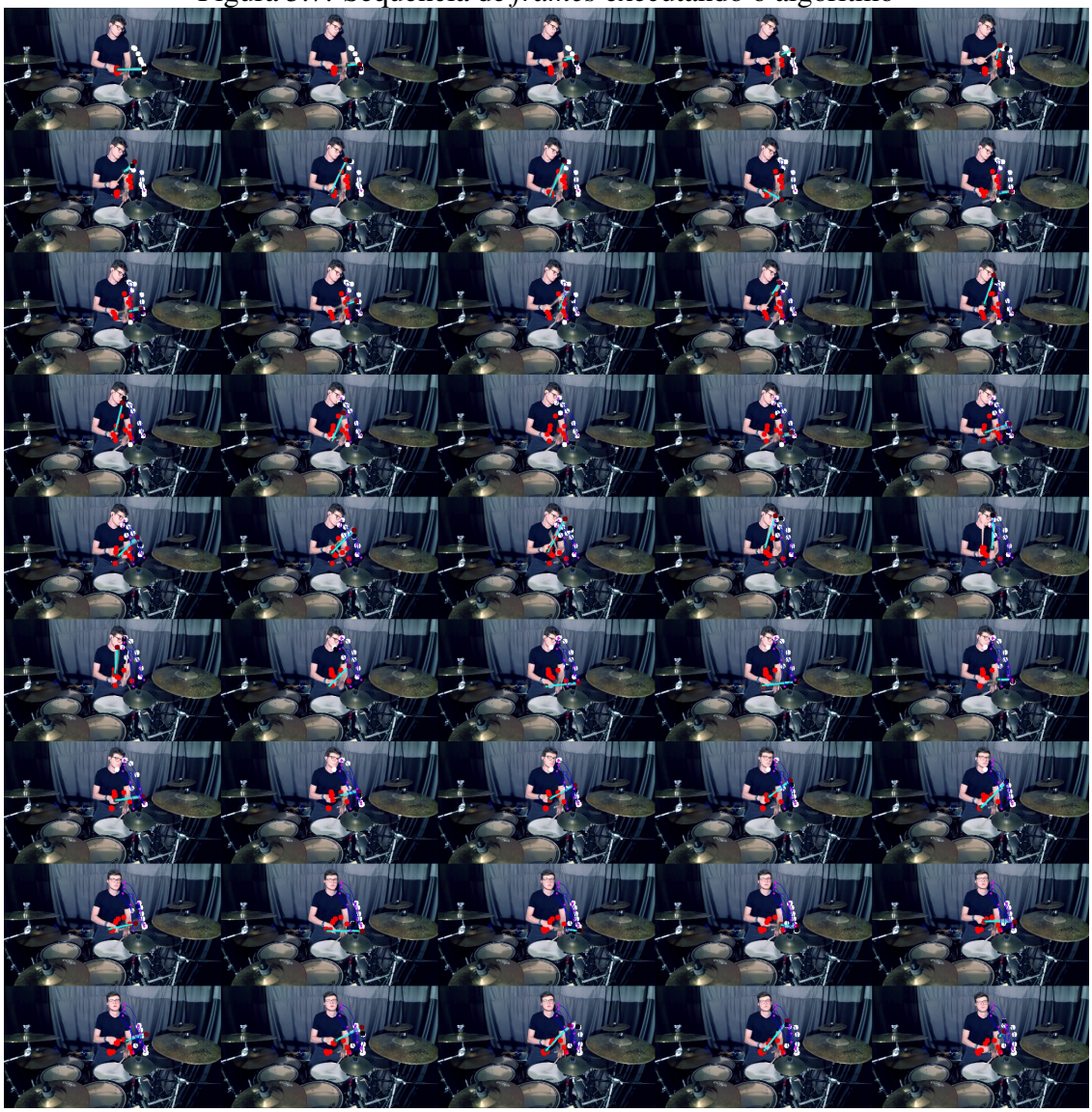
Foram feitos alguns testes para que dadas as linhas mais próximas, alguns filtros fossem aplicados: um destes filtros iria remover, dentre as linhas restantes, as que não tivessem tamanho próximo do tamanho médio das outras linhas; outro filtro iria identificar as linhas com ângulo próximo as últimas linhas do histórico. No final, estes filtros foram implementados, mas foram desativados, uma vez que notamos que podia se perder detec-

ções da baqueta, as vezes resultando em um conjunto de linhas vazio, não conseguindo inferir a baqueta. Isso porque no primeiro filtro poderia se perder uma detecção por ter ocorrido uma detecção segmentada da reta correspondente à baqueta, de forma que o seu tamanho fosse menor que a média, filtrando a linha. Já no segundo filtro, no caso de uma variação brusca da posição da baqueta, as linhas detectadas poderiam acabar sendo removidas.

Para inferir a coordenada da ponta da baqueta, é exigido que o usuário manualmente insira, como parte do *ground truth* por ele fornecido indicando os dois pontos que formam a linha da baqueta em *frames* iniciais do vídeo, qual deles corresponde à ponta e a medida que mais pontos vão entrando no histórico continua-se salvando qual dos dois pontos é a ponta da baqueta. Desta forma, ao encontrar a linha que se encaixa melhor no histórico, o ponto que está mais próximo das pontas das baquetas no histórico é considerado a ponta da baqueta. Dado que foi inferida a posição da baqueta no *frame* atual, esta posição entra no histórico e no próximo *frame* fará parte dos pontos considerados para inferir a nova posição da baqueta.

A Figura 5.7 mostra uma sequência de 45 *frames* consecutivos — a sequência é da esquerda para direita, de cima para baixo — nos quais é possível verificar o funcionamento básico do algoritmo para identificação da posição mais provável da baqueta em cada *frame*. Os pontos brancos indicam o histórico de pontas da baqueta, os pontos vermelhos indicam o histórico da parte inferior da baqueta (cabo), o ponto preto é a predição do filtro de Kalman (que será descrito na próxima seção), em azul mostramos o tracejado da ponta da baqueta, e em azul claro indicamos onde que a baqueta foi inferida naquele *frame*. O algoritmo desenvolvido apenas rastreia uma baqueta por vez, sendo necessário rodá-lo uma vez para cada uma das duas baquetas. Em uma seção posterior serão descritos com maiores detalhes as limitações da solução desenvolvida.

Figura 5.7: Sequência de *frames* executando o algoritmo



Fonte: obtido pelo autor, 2019.

Note que no primeiro quadro é possível observar o *ground truth* manualmente fornecido pelo usuário. A medida que se avança nos quadros, se obtém mais pontos a partir da inferência da localização da baqueta, assim sendo substituído o histórico pelas informações novas obtidas. Outra observação é que em momentos em que a baqueta se movimentou muito rápida e ficou borrada no quadro, a detecção da baqueta é afetada. Além disso, em alguns quadros é possível verificar que foi detectada uma baqueta de tamanho diferente do correto, uma vez que, em alguns casos, a linha da baqueta foi incorretamente identificada como um conjunto de segmentos de retas menores que o ideal.

5.2.7 Filtro de Kalman

Devido as variações que podem ocorrer nas medidas obtidas da ponta da baqueta — inclusive por erros — foi adicionado um filtro de Kalman que é aplicado ao histórico das pontas da baqueta. O objetivo disso é obter uma predição da posição atual da baqueta que suavize variações bruscas em medições de posições anteriores, as quais podem ocorrer devido a falta de alguma detecção de ponta ou detecção em local incorreto em algum *frame*. O filtro de Kalman introduz esta suavização nas predições, ao mesmo tempo que preserva a tendência do movimento da baqueta, uma vez que utiliza o histórico de posições passadas preditas para obter as predições. Assim, a cada *frame* — após determinar onde estaria a baqueta baseado no histórico — o filtro é aplicado para obter a informação suavizada da ponta que é a informação utilizada para a detecção de colisão descrita na próxima seção.

5.3 Detectando Colisões

Com as regiões de interesse selecionadas, quando é detectada uma variação angular brusca da ponta da baqueta — obtida pós a aplicação do filtro de Kalman — e que seja maior que um *threshold* definido é verificado onde a mesma se encontrava no momento da suposta colisão; caso o local da colisão esteja dentro da região de interesse associada a uma peça da bateria, é considerado que houve uma colisão com aquela peça. A heurística aplicada é exatamente a descrita em 4.1.3, sendo apenas necessário a cada *frame* realizar os cálculos da heurística usando como a informação da ponta da baqueta a posição predita pelo filtro de Kalman.

5.4 Saída do Algoritmo

Quando uma colisão é detectada, se salva com qual peça ela ocorreu, o tempo em milissegundos em relação ao início do vídeo analisado, e o *frame* no qual ocorreu a colisão, além do rótulo da peça tocada. Ao final, se gera um arquivo de saída com estes dados, além da informação da duração total do vídeo analisado e o número total de *frames*.

Arquivo 5.1 – Exemplo de *output*. Aqui, “instrument 0” corresponde a uma das peças manualmente indicadas pelo usuário — no caso de vídeo analisado, um dos pratos da bateria


```
{
   "drums":[
      {
         "label":"instrument 0",
         "time":1400.0,
         "frame":41
      },
      {
         "label":"instrument 0",
         "time":2000.0,
         "frame":59
      },
      {
         "label":"instrument 0",
         "time":2033.3333333333333,
         "frame":60
      },
      {
         "label":"instrument 0",
         "time":2300.0,
         "frame":68
      }
   ],
   "duration":2466.6666666666665,
   "frames": 74
}
```

Com esta saída é possível — em um trabalho futuro — traduzir a informação de colisão com peças individuais (e o tempo em que tais colisões ocorrem) para produzir um arquivo MIDI ou até mesmo uma partitura, uma vez que temos a informação de tempo de cada colisão e em qual peça.

6 RESULTADOS

Para a obtenção dos resultados experimentais, foram separados diferentes trechos/-clipes de cinco segundos de diferentes vídeos obtidos no YouTube; processamos estes vídeos com o método descrito na Seção 4.2 e analisamos os resultados obtidos quanto a detecção correta da colisão com as peças da bateria. Os vídeos escolhidos foram de bateristas tocando com câmeras filmando de frente para músico, ou em posição lateral. Além disso, os vídeos escolhidos possuíam 30 *frames* por segundo (FPS), que consideramos um número razoável de *frames*. Escolhidos os vídeos, os clipes de cinco segundos foram separados da seguinte forma: três trechos por vídeo, um clipe do inicio do vídeo, um do meio e um do final.

Para conseguir realizar a verificação sobre se o *output* gerado por nossa técnica corretamente detectou as colisões, é feita a análise dos vídeos de forma manual, rotulando manualmente em quais *frames* ocorreram as colisões e em quais peças da bateria, a fim de obter um conjunto de validação com o qual a saída de nossa técnica é comparada. De posse destas informações, mensuramos o quanto o método desenvolvido consegue detectar em termos de colisões com as peças corretas. É importante lembrar novamente que foi considerada uma margem de erro de *frames* para considerarmos se está correta a detecção da colisão, conforme descrito em 4.2.

Em cada teste, utilizamos o método proposto para fazer o rastreamento tanto da baqueta da mão dominante quanto da mão não-dominante. Para um mesmo clipe, para uma dada baqueta, foram feitas sempre três execuções do algoritmo, a fim de minimizar o efeito de ruído na rotulação manual do *ground truth* nas nossas métricas de performance. Após executar o nosso método em cada trecho de vídeo, utilizamos a saída por ele produzida e a informação de considerada verdadeira sobre as colisões — manualmente obtida — para montar montada uma matriz de confusão para cada um dos clipes analisados. A matriz contempla os dados dos três *outputs* associados a cada clipe, comparados ao resultado esperado caso a detecção fosse perfeita, e separados por qual baqueta foi rastreada — a da mão dominante ou não. No apêndice B é possível visualizar um exemplo de teste realizado e em quais quadros idealmente deveriam ser detectadas as colisões [1]. Os resultados obtidos podem ser vistos, de forma sumarizada, nas tabelas e imagens das próximas subseções.

[1] Todo o *dataset* usado, incluíndo os vídeos, *frames* que ocorreram as colisões e *outputs* podem ser visualizados no link: <https://drive.google.com/open?id=19QUbz14tRd_WcJmA_hRZrYTUzsQ_pQkx>

6.0.1 Análise de Vídeo: Vídeo 1 - Câmera Frontal - 30 FPS

Neste vídeo, a câmera encontrava-se posicionada de forma fixa frontalmente ao baterista (AGUZZI, 2016). Em nossos testes, este é o melhor cenário, pois deixa mais claros os movimentos do baterista, uma vez que câmeras laterais podem às vezes esconder uma da baquetas atrás do corpo do músico.

Ao processar o primeiro clipe extraído deste vídeo, observamos que as colisões da baqueta da mão dominante foram detectadas com excelente performance, como pode ser visto na Matriz de Confusão nas tabelas 6.1 e 6.2. Com uma acurácia de 96.22% e um F1 Score de 0.79, obtivemos um resultado que não diferiu muito do que foi obtido via heurística do cálculo vetorial apresentada na Seção 4.1.3, responsável por detectar colisões, na qual havíamos usado o *ground truth* de todas posições diretamente; ou seja, o nosso método obteve performance bastante semelhante justamente ao que considerávamos ser um cenário em que teríamos a informação perfeita da ponta da baqueta. Vale ressaltar que houve uma queda no F1 score em relação a validação da heurística da Seção 4.1.3, devido a um aumento de Falsos Positivos. No entanto, os resultados foram um pouco piores para os outros dois clipes. Isto pode ser visto na tabela 6.2. Ambos os clipes tiveram proporcionalmente um número maior de Falsos Positivos o que baixou o valor da métrica de sensibilidade que por sua vez fez que o F1 Score fosse mais baixo em ambos os casos se comparados com o F1 Score do primeiro clipe. Como em cada clipe os parâmetros de *threshold* de colisão, *threshold* de detecção da linha e entre outros, tiveram que ser ajustados, é possível que o ajuste tenha deixado o algoritmo um pouco mais sensível a colisões gerando mais Falsos Positivos, no entanto, tal medida foi necessária para que fosse possível também obter as colisões que de fato aconteceram (True Positives).

Considerando os resultados de acurácia obtidos ao se analisar os três trechos do primeiro vídeo, pudemos observar que nosso método obteve bons resultados finais: uma acurácia total — ou seja, considerando o somatório de todos FNs, TPs, FPs e TNs dos três clipes — de 93.26% e 0.69 de F1 Score, ao se validar nossa técnica quando aplicada para rastrear os movimentos da mão dominante. A seguir, iremos analisar a performance do algoritmo ao se analisar os mesmos 3 clipes deste vídeo, mas quando nossa técnica é utilizada para rastrear os movimentos da mão não dominante.

Tabela 6.1: Matriz de confusão: Vídeo 1 - **Mão dominante**

		Diagnóstico verdadeiro	
		Colidiu	Não colidiu
Dado obtido via algoritmo	Colidiu	103	14
	Não colidiu	77	1156

Fonte: obtido pelo autor, 2019.

Tabela 6.2: Todos resultados obtidos por clipe: Vídeo 1 - **Mão dominante**

	Clipe 1	Clipe 2	Clipe 3	Total
FN	15	42	20	77
TP	33	42	28	103
FP	2	10	2	14
TN	400	356	400	1156
Total	450	450	450	1350
ACC	0.9622	0.8844	0.9511	0.9326
Prec	0.9429	0.8077	0.9333	0.8803
Sens	0.6875	0.5	0.5833	0.5722
F1	0.7952	0.6176	0.7179	0.6936

Fonte: obtido pelo autor, 2019.

Apresentamos, agora, nas tabelas 6.3 e 6.4, a performance de nossa técnica quando analisando o comportamento da mão não dominante do baterista nos 3 trechos do primeiro vídeo. Podemos observar que o algoritmo tem dificuldades em rastrear a baqueta na mão não dominante. O motivo pelo qual isso ocorre é melhor discutido na Seção 6.1.1. Neste caso mesmo conseguindo uma boa acurácia total de 94.07%, a métrica de F1 Score foi ruim, uma vez que tivemos não só um número elevado de colisões não detectadas, mas também Falsos Positivos. Os Falsos Negativos aumentaram justamente por causa que em alguns momentos a baqueta na mão dominante poderia não ser detectada, principalmente por ela se movimentar menos do que a mão não dominante, sendo necessário configurar parâmetros que conseguissem ser sensíveis os suficientes para capturar os pequenos movimentos da baqueta. Em contra partida, ao deixar mais sensível o rastreamento da baqueta, era possível que Falsos Positivos acontecessem, em que o algoritmo acaba confundindo um movimento que não houve colisão com uma colisão. Desta forma, foi necessário tentar encontrar um equilíbrio entre os parâmetros para que fosse sensível o suficiente para detectar alguma colisão de forma correta, mas não tão sensível ao ponto de acontecer muitos Falsos Positivos — e mesmo após diversas tentativas de configuração este resultado que encontramos não conseguiu ser muito satisfatório. Este é um padrão que se repete para todos os vídeos nos testes da mão não dominante, e se deve justamente ao fato dos movimentos serem menos acentuados na mão não dominante, que pode fazer com que o

filtro de subtração de fundo remova a região que encontra-se a baqueta.

Tabela 6.3: Matriz de confusão: Vídeo 1 - **Mão não dominante**

		Diagnóstico verdadeiro	
		Colidiu	Não colidiu
Dado obtido via algoritmo	Colidiu	27	30
	Não colidiu	50	1243

Fonte: obtido pelo autor, 2019.

Tabela 6.4: Todos resultados obtidos por clipe: Vídeo 1 - **Mão não dominante**

	Clipe 1	Clipe 2	Clipe 3	Total
FN	17	22	11	50
TP	11	6	10	27
FP	7	20	3	30
TN	415	402	426	1243
Total	450	450	450	1350
ACC	0.9467	0.9067	0.9689	0.9407
Prec	0.6111	0.2308	0.7692	0.4737
Sens	0.3929	0.2143	0.4762	0.3506
F1	0.4783	0.2222	0.5882	0.4030

Fonte: obtido pelo autor, 2019.

6.0.2 Análise de Vídeo: Vídeo 2 - Câmera em Diferentes Posições - 30 FPS

No segundo vídeo que analisamos, a câmera se encontra às vezes em uma posição lateral (em dois clipes; clipes 1 e 3) e às vezes (no clipe 2) em uma posição frontal levemente superior ao baterista (VADRUM, 2019). A diversidade de diferentes ângulos de filmagem auxilia na análise da performance do nosso algoritmo em situações diversas de vídeos que apenas sejam feitos a partir de uma posição "canônica" — e.g., sempre perfeitamente alinhados frontalmente com o músico.

Primeiro analisamos, na tabela 6.6, o desempenho do algoritmo ao analisar o comportamento da mão dominante do músico. Podemos perceber que houve, aqui, desempenho pior do que no vídeo 1. Mesmo a acurácia aumentando de 93.26% para 95.78% tivemos uma queda do F1 Score de 0.6936 para 0.6076; valorizamos mais a métrica de F1 Score pois ela leva em consideração Falsos Positivos e Falsos Negativos, enquanto a acurácia em nossos testes será geralmente muito alta, pelo motivo que existem muitos *frames* nos quais não existem de fato colisões, e portanto, os Verdadeiros Negativos elevam a acurácia a níveis bem altos e que não necessariamente refletem um bom desempenho.

O clipe 2, por exemplo, mesmo sendo frontal (assim como os clipes do vídeo 1), por ter sido filmado com uma câmera posicionada em local levemente superior ao músico, teve seu desempenho prejudicado, apresentando dificuldades em detectar colisões. Isto porque com uma câmera posicionada de forma um pouco elevada, faz que o movimento da baqueta seja menos acentuado — basta imaginar que em uma situação mais extrema, com a câmera posicionada exatamente acima do baterista teríamos, na visualização 2D da imagem, uma baqueta que não estaria basicamente se movendo, pois seu deslocamento na vertical não seria notável para um observador que encontra-se vendo-a em uma visão superior, sendo possível observar apenas uma pequena diminuição no tamanho da baqueta ao se afastar do observador. Exatamente por este motivo, pela câmera estar em posição levemente superior, foi necessário deixar o algoritmo parametrizado de uma forma mais sensível ao rastreamento e a colisões o que possivelmente contribuiu para o aumento de Falsos Positivos tornando assim menos preciso o algoritmo.

No entanto, mesmo com um desempenho pior do que o desempenho total do vídeo 1, o resultado no geral ainda foi satisfatório. Em particular, nosso método teve uma acurácia total de 95.41% e 0.6076 de F1 Score. Tivemos um resultado inferior ao do primeiro vídeo, mas ainda sim os resultados sugerem que o algoritmo consegue detectar colisões de maneira satisfatória ao se analisar o comportamento da mão dominante do baterista.

Tabela 6.5: Matriz de confusão: Vídeo 2 - **Mão dominante**

		Diagnóstico verdadeiro	
		Colidiu	Não colidiu
Dado obtido via algoritmo	Colidiu	48	22
	Não colidiu	40	1240

Fonte: obtido pelo autor, 2019.

Tabela 6.6: Todos resultados obtidos por clipe: Vídeo 2 - **Mão dominante**

	Clipe 1	Clipe 2	Clipe 3	Total
FN	11	16	13	40
TP	17	15	16	48
FP	8	5	9	22
TN	414	414	412	1240
Total	450	450	450	1350
ACC	0.9578	0.9533	0.9511	0.9541
Prec	0.6800	0.7500	0.6400	0.6857
Sens	0.6071	0.4839	0.5517	0.5455
F1	0.6415	0.5882	0.5926	0.6076

Fonte: obtido pelo autor, 2019.

Discutimos agora, nas tabelas 6.8 e 6.7, o comportamento do algoritmo ao analisar o movimento da mão não dominante do músico. Neste caso, os resultados foram péssimos para a baqueta na mão não dominante. Para casos como o clipe 3, clipe no qual a câmera estava posicionada de forma lateral, não foi detectada nenhuma colisão — e ainda por cima, foram detectados vários Falsos Positivos. Isto refletiu diretamente na precisão e sensibilidade que zeraram. Isto ocorreu porque neste terceiro clipe, além dos problemas já descritos sobre a mão não dominante, existe uma parte do vídeo na qual a baqueta fica parcialmente escondida, fora do *frame*. Aqui, novamente fica evidente que não podemos analisar o desempenho apenas pela acurácia, pois mesmo com uma acurácia alta, os Falsos Negativos e Falsos Positivos foram elevados. Portanto, isto mostra a importância da precisão e sensibilidade como uma métrica para apontar estes problemas, e por consequência, o F1 Score sendo uma medida da média destas métricas, também reflete claramente o desempenho ruim do algoritmo para este caso. No geral, isso mostra que de fato é necessário que o algoritmo seja melhorado futuramente para detectar de forma melhor a baqueta que encontra-se na mão não dominante.

Tabela 6.7: Matriz de confusão: Vídeo 2 - **Mão não dominante**

		Diagnóstico verdadeiro	
		Colidiu	Não colidiu
Dado obtido via algoritmo	Colidiu	48	22
	Não colidiu	40	1240

Fonte: obtido pelo autor, 2019.

Tabela 6.8: Todos resultados obtidos por clipe: Vídeo 2 - **Mão não dominante**

	Clipe 1	Clipe 2	Clipe 3	Total
FN	8	21	15	44
TP	13	3	0	16
FP	14	12	8	34
TN	415	414	427	1256
Total	450	450	450	1350
ACC	0.9511	0.9267	0.9556	0.9422
Prec	0.4815	0.2	0.	0.32
Sens	0.619	0.125	0.	0.2667
F1	0.5417	0.1538	NA	0.2909

Fonte: obtido pelo autor, 2019.

6.1 Discussão Sobre os Resultados

A acurácia manteve-se alta em todos os testes, mas isso se deve principalmente ao fato de que, na maior parte do tempo, não há colisões nos *frames*, o que acaba contribuindo muito para uma acurácia alta. Por este motivo é importante a análise do F1 Score que — em casos como do terceiro clipe do segundo vídeo — será baixo sempre quando houver poucas detecções de colisão ou problemas como falsos positivos e/ou falsos negativos.

É importante destacar que, para análise de cada clipe dos vídeos, foram necessários ajustes em parâmetros de configuração do método, através da variação dos seus meta-parâmetros. Entretanto, essa análise foi feita de maneira manual e empírica, e não temos garantias que a melhor configuração foi encontrada e utilizada. Os parâmetros que poderiam ser configurados consistiam nas configurações previamente descritas do detector de linhas e do subtrator de fundo, além do número de pontos de *ground truths* a serem utilizados pelo processo da detecção da ponta da baqueta descrito na Seção 5.2.6, o tamanho do histórico, o tamanho da área próxima dos braços identificados pelo passo de *pose estimation*, e o *threshold* utilizado na análise da variação angular pela heurística do cálculo vetorial para identificação de colisões. Para os testes com a mão não dominante, no geral, percebemos que o algoritmo ficava sensível demais (de forma que detectava muitos Falsos Positivos) ou então ficava pouco sensível (perdendo muitas detecções aumentando os Falsos Negativos). Em certos casos, tanto Falsos Positivos e Falsos Negativos ocorriam, pois conforme descrito anteriormente, a pouca movimentação da mão não dominante é um problema, pois esta acaba sendo filtrada pelo subtrator de fundo, e assim, com pouca movimentação uma colisão poderia não ser detectada, mas então ao mover a baqueta de uma forma um pouco mais brusca, devido o quão sensível foi configurado o algoritmo, ocorriam também Falsos Positivos.

Notamos também em nossos testes que nossa suposição original de que a baqueta seria sempre uma linha reta não é totalmente verdadeira, uma vez que dependendo do movimento da baqueta, a mesma pode estar borrada em *frames* e até mesmo ser levemente curvada na imagem. O resultado disso, é que nestes *frames* existe a possibilidade que sobre a baqueta sejam detectadas múltiplas linhas segmentadas. Isto pode dificultar um pouco o processo de detectar a posição da baqueta, no entanto, ainda sim é melhor termos a informação das linhas segmentadas em alguns poucos *frames* do que filtrar este tipo de informação.

6.1.1 Limitações da Solução Desenvolvida

Um dos problemas encontrados na solução proposta é que em algumas situações há um movimento brusco por parte do músico — mover a baqueta de um extremo da imagem para o outro, por exemplo — o que completamente muda a região da bateria que está sendo tocada. Neste caso, pode ser que não haja *frames* suficientes para amostrar esta transição de posição, e o algoritmo pode não conseguir atualizar corretamente o histórico sendo mantido. Sendo assim, a solução depende diretamente dos *frames* obtidos por segundo; quanto maior a resolução temporal, melhor. Caso o vídeo analisado tenha poucos *frames* por segundo, muitas das posições intermediarias recentes da baqueta podem não terem sido capturadas, o que resulta em tentativas de detecção de colisões com base em históricos incompletos; ao tentar inferir próximas posições da baqueta com base neste histórico, há introdução de erros.

Em nossos testes, no geral, observamos que o problema descrito acima (*frame rate* baixo) resultava em erros que se acumulavam no histórico, de forma que o algoritmo não mais conseguia corretamente prever as posições sucessivas da baqueta — o histórico inteiro era incrementalmente substituído por informações equivocadas. Desta forma, seria necessário redefinir um histórico na nova posição da baqueta, ou seja, uma possível solução seria de tempos em tempos solicitar a intervenção do usuário para que ele rotulasse (assim como fez nos *frames* iniciais) o *ground truth* de localização das baquetas. Este problema também ocorre porque o algoritmo apenas olha para um histórico passado de posições estimadas da baqueta e não para *frames* futuros, os quais estão disponíveis para o algoritmo, visto que não se trata de um algoritmo de tempo-real. Seria possível, portanto, em uma versão melhorada do algoritmo, executá-lo uma primeira vez para obter localizações possíveis da baqueta (ao longo de todo o vídeo) e então executar uma segunda passada, desta vez analisando um histórico que contempla tanto possíveis localizações em quadros passados recentes quando em quadros futuros próximos, dessa forma melhor inferindo a posição futura da baqueta.

Outra limitação do método proposto é que ele não lida com informação sobre o bumbo. Infelizmente, no geral, tal peça da bateria fica escondida nos vídeos, de forma que não é possível inferir (apenas com base em informações visuais) o toque no mesmo. Além disso, mesmo que o vídeo apresentasse uma visão do bumbo, devido a este não estar próximo dos braços do baterista, ele seria removido pelo filtro de *pose estimation*; a análise desta peça exigiria tratamento especial por parte do método.

Existe também uma dificuldade intrínseca à análise do toque de uma bateria, que é o fato de que existe uma mão (a mão dominante) que se move muito mais e mais rapidamente do que a outra mão, e que (no geral) dita o ritmo da música; enquanto a outra mão (a mão **não** dominante) se move pouco, com movimentos menos acentuados. O problema com o algoritmo desenvolvido é que o mesmo encontra dificuldades no rastreio da mão não dominante, uma vez que a sua pouca movimentação faz com que o filtro de *background* por vezes a remova da imagem. Além disso, quando as baquetas se cruzam, é possível que o rastreamento acabe "confundindo" as estimativas de posição de cada uma das baquetas, e incorretamente atribuindo à baqueta da mão não dominante posições que, na verdade, pertencem à baqueta da mão dominante, a qual se move mais e tem mais linhas detectadas. Nos resultados apresentados neste capítulo, este problema fica evidente.

Outra limitação do método (conhecida de antemão, antes mesmo de implementarmos a solução) é que ele não conseguiria obter a informação da intensidade com a qual uma dada peça do instrumento foi tocada. Esta é uma informação que poderia possivelmente ser inferida a partir da velocidade do movimento executado pelo músico; no entanto, não fez parte do foco do nosso trabalho. Outro fator limitante é que, na bateria, a posição exata na qual foi atingido um prato ou um tambor, por exemplo, afeta o som que o mesmo irá produzir. Nossa solução apenas detecta qual peça foi atingida; portanto, se for desejado saber uma região específica que a peça foi atingida, seria necessário que o usuário informasse mais rótulos, no início do algoritmo, não sobrepostos, para cada uma das partes das peças de interesse. Uma vez que o algoritmo depende de vídeos nos quais se consiga visualizar o movimento claro das baquetas, também temos limitações sobre a posição da câmera. A câmera deve estar em frente ao músico ou em uma posição lateral. Caso esteja em uma visão superior, se torna muito difícil de detectar a batida da baqueta quando a mesma se desloca verticalmente, sem mudar seu ângulo ou inclinação, e se distanciando da câmera, para colidir com uma peça da bateria. Além disso, é também necessário que a câmera não se mova durante os vídeos, caso contrário pode haver problemas no filtro de subtração de fundo.

Por fim, também não fornecemos ou analisamos informações sobre a afinação na qual as peças do instrumento se encontram. Todavia, como discutido anteriormente, a informação mais importante para transcrição de músicas de bateria é qual peça foi tocada, e não qual nota que ela produz. Esta observação reforça o fato de que a nossa abordagem tem como objetivo ser um método complementar na transcrição automática de música, e que, se usada em conjunto com outras soluções que analisam áudio, por exemplo, pode

resultar em um desempenho melhor, possivelmente superando parcialmente as limitações aqui discutidas.

7 CONCLUSÃO

Existem diversas formas de realizar AMT em música; no entanto poucos trabalhos foram encontrados utilizando como base para a transcrição informações visuais. Neste trabalho, focando em AMT para o instrumento de bateria, mostramos que apenas com base no processamento de vídeos, é possível realizar transcrição de músicas de forma aproximada. Este trabalho contribui ao combinar elementos como *pose estimation* com técnicas de processamento de imagens, mostrando que, com uma sequência de etapas bem definida e ajustada, é possível obter as informações necessárias para que se possa transcrever o que foi tocado no instrumento de percussão. O método proposto consegue atingir resultados interessantes que evidenciam o fato de existe informação visual muito relevante que pode ser explorada por técnicas de AMT. Entretanto, percebemos também, no capítulo em que estudamos o estado-da-arte, que isso raramente é feito, uma vez que a maioria das soluções existentes apenas analisa o áudio das músicas. Ademais, a técnica aqui proposta não exclui a possibilidade de ser utilizada em conjunto com técnicas baseadas em áudio, o que torna nosso método uma alternativa possível e complementar, passível de prover melhorias a sistemas nos quais se usa apenas informação sonora; mesmo em casos em que tais sistemas tenham informações visuais disponíveis.

As principais dificuldades encontradas na solução proposta estão relacionadas com a qualidade das imagens analisadas — que precisam ocorrer um número de *frames* suficientes (por segundo) para que as colisões sejam propriamente capturadas pela câmera. Outra dificuldade está na configuração dos parâmetros utilizados por nosso algoritmo. A configuração destes, para que se obtenha o melhor resultado possível, é um problema desafiador, o qual tratamos, até o momento, de forma empírica. Além disso, encontramos dificuldades também no rastreamento de baquetas que não se encontram na mão dominante do músico, o que evidencia que existe espaço para mais estudos e melhorias na técnica proposta.

7.1 Trabalhos Futuros

Uma melhoria no trabalho desenvolvido seria adicionar ao histórico de posições estimadas da baqueta também predições sobre a posição da baqueta nos próximos n *frames*, a fim de que a decisão de posicionamento da baqueta em um dado momento no tempo não seja baseada apenas no passado, mas também em estimativas de para onde ela

possivelmente se moveu em *frames* subsequentes. Tal alteração iria contribuir para que o método conseguisse ter melhor performance ao analisar trechos maiores que 5 segundos, como feito atualmente; tal decisão, como discutido anteriormente, se deve justamente a esta limitação da solução atual — i.e., de que o histórico inclui apenas estimativas do comportamento passado da baqueta. Assim, uma mudança abrupta da posição baqueta para uma região bem distante do histórico (mover a baqueta da extrema esquerda para a extrema direita da imagem, por exemplo) faz com que o método não encontre linhas viáveis para considerar como a baqueta, ou então, detecta regiões erradas, como o braço do músico como uma baqueta, forçando que sejam segmentados os vídeos analisados. Outra melhoria possível de ser explorada em trabalhos futuros é realizar o rastreamento das duas baquetas simultaneamente. Isso auxiliaria na performance no método pois, além de tornar mais prático o uso do algoritmo, traria a vantagem de que, ao rastrearmos ambas as baquetas, poderíamos evitar os problemas ocasionados quando há o cruzamento entre as baquetas conforme descrito na Seção 6.1.1. Com tal alteração, diminuiríamos também o problema de conseguir rastrear de forma mais precisa a baqueta posicionada na mão não dominante do músico. Por fim, poderíamos, em um trabalho futuro, utilizar o *output* produzido por nosso método (para um exemplo, verificar Apêndice B) para geração automática de arquivos MIDI e/ou partituras correspondentes à transição da música presente nos vídeos analisados.

REFERÊNCIAS

AGUZZI, P. **Paolo Aguzzi - Xiaomi Yi - Drum Cam Test**. 2016. Available from Internet: <https://www.youtube.com/watch?v=a1-xDsUnu5c>.

BABENKO, B.; YANG, M.-H.; BELONGIE, S. Visual tracking with online multiple instance learning. In: IEEE. **Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on**. [S.l.], 2009. p. 983–990.

CAO, Z. et al. OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. In: **arXiv preprint arXiv:1812.08008**. [S.l.: s.n.], 2018.

CAO, Z. et al. Realtime multi-person 2d pose estimation using part affinity fields. In: **CVPR**. [S.l.: s.n.], 2017.

DANELLJAN, M. et al. Accurate scale estimation for robust visual tracking. In: BMVA PRESS. **British Machine Vision Conference, Nottingham, September 1-5, 2014**. [S.l.], 2014.

FREUND, Y.; SCHAPIRE, R.; ABE, N. A short introduction to boosting. **Journal-Japanese Society For Artificial Intelligence**, JAPANESE SOC ARTIFICIAL INTELL, v. 14, n. 771-780, p. 1612, 1999.

GILLET, O.; RICHARD, G. Automatic transcription of drum loops. In: IEEE. **2004 IEEE International Conference on Acoustics, Speech, and Signal Processing**. [S.l.], 2004. v. 4, p. iv–iv.

GILLET, O.; RICHARD, G. Automatic transcription of drum sequences using audiovisual features. In: **Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.** [S.l.: s.n.], 2005. v. 3, p. iii/205–iii/208 Vol. 3. ISSN 1520-6149.

GOLDSTEIN, S.; MOSES, Y. Guitar music transcription from silent video. 2018.

GOWRISHANKAR, B. S.; BHAJANTRI, N. U. An exhaustive review of automatic music transcription techniques: Survey of music transcription techniques. In: **2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)**. [S.l.: s.n.], 2016. p. 140–152.

GRABNER, H.; GRABNER, M.; BISCHOF, H. Real-time tracking via on-line boosting. In: **Bmvc**. [S.l.: s.n.], 2006. v. 1, n. 5, p. 6.

GÜLER, R. A.; NEVEROVA, N.; KOKKINOS, I. Densepose: Dense human pose estimation in the wild. **CoRR**, abs/1802.00434, 2018. Available from Internet: <http://arxiv.org/abs/1802.00434>.

KALAL, Z.; MIKOLAJCZYK, K.; MATAS, J. Forward-backward error: Automatic detection of tracking failures. In: IEEE. **Pattern recognition (ICPR), 2010 20th international conference on**. [S.l.], 2010. p. 2756–2759.

KALAL, Z. et al. Tracking-learning-detection. **IEEE transactions on pattern analysis and machine intelligence**, v. 34, n. 7, p. 1409, 2012.

KIM, I. **tf-pose-estimation**. [S.l.]: GitHub, 2013. <https://github.com/ildoonet/tf-pose-estimation>.

MAEZAWA, A. et al. Automated violin fingering transcription through analysis of an audio recording. **Computer Music Journal**, MITP, v. 36, n. 3, p. 57–72, 2012.

MARENCO, B. et al. A multimodal approach for percussion music transcription from audio and video. In: SPRINGER. **Iberoamerican Congress on Pattern Recognition**. [S.l.], 2015. p. 92–99.

MATAS, J.; GALAMBOS, C.; KITTLER, J. Robust detection of lines using the progressive probabilistic hough transform. **Computer Vision and Image Understanding**, Elsevier, v. 78, n. 1, p. 119–137, 2000.

PAN, J.; HU, B. Robust object tracking against template drift. In: IEEE. **Image Processing, 2007. ICIP 2007. IEEE International Conference on**. [S.l.], 2007. v. 3, p. III–353.

PAREKH, H. S.; THAKORE, D. G.; JALIYA, U. K. A survey on object detection and tracking methods. **International Journal of Innovative Research in Computer and Communication Engineering**, v. 2, n. 2, p. 2970–2978, 2014.

SARNATSKYI, V. et al. Music transcription by deep learning with data and"artificial semantic"augmentation. **arXiv preprint arXiv:1712.03228**, 2017.

SIMAS, G. M. et al. Utilizando visão computacional para reconstrução probabilística 3d e rastreamento de movimento. **VETOR-Revista de Ciências Exatas e Engenharias**, v. 17, n. 2, p. 59–77, 2007.

Sobottka, K.; Pitas, I. Face localization and facial feature extraction based on shape and color information. In: **Proceedings of 3rd IEEE International Conference on Image Processing**. [S.l.: s.n.], 1996. v. 3, p. 483–486 vol.3.

VADRUM. **Les Toréadors (Classical Drumming)**. 2019. Available from Internet: <http://www.youtube.com/watch?v=9ZOX6oTitIU>.

WELCH, G.; BISHOP, G. et al. An introduction to the kalman filter. 1995.

Wikimedia Commons. **Modelo de cores HSV mapeado para um cilindro.** 2013. [Online; acessado dia 1 Junho de 2019]. Available from Internet: <https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder.png>.

WU, C. et al. A review of automatic drum transcription. **IEEE/ACM Transactions on Audio, Speech, and Language Processing**, v. 26, n. 9, p. 1457–1483, Sept 2018. ISSN 2329-9290.

YILMAZ, A.; JAVED, O.; SHAH, M. Object tracking: A survey. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 38, n. 4, dec. 2006. ISSN 0360-0300. Available from Internet: <http://doi.acm.org/10.1145/1177352.1177355>.

ZIVKOVIC, Z. et al. Improved adaptive gaussian mixture model for background subtraction. In: CITESEER. **ICPR (2)**. [S.l.], 2004. p. 28–31.

APÊNDICE A — RESULTADOS DOS TRECHOS ANALISADOS PARA VALIDAÇÃO DA HEURÍSTICA DO CÁLCULO VETORIAL

Tabela A.1: **Trecho 1 (Frontal)**: Tabela com o resultado final da heurística usando a variação angular para verificar se houve colisão — Threshold de variação angular: 60

Variação angular	*Relevância*	*Colisão*	*Validação*
36.65610842	Ignorar		OK
-47.51252176	Ignorar		OK
67.16634582	Mudança grande	Não colidiu	OK
-21.45745791	Ignorar		OK
-119.7014095	Mudança grande	Não colidiu	OK
148.5521102	Mudança grande	Colidiu em TAMBOR 1	ACERTOU COLISAO
7.865745825	Ignorar		OK
-1.506242082	Ignorar		OK
3.131398719	Ignorar		OK
-3.018357338	Ignorar		OK
2.468915208	Ignorar		OK
-52.20298509	Ignorar		OK
53.26160413	Ignorar		OK
-3.357078679	Ignorar		OK
-16.15733986	Ignorar		OK
-24.32073824	Ignorar		OK
24.97647051	Ignorar		OK
-0.7119413232	Ignorar		OK
-90.4848246	Mudança grande	Não colidiu	OK
36.83103329	Ignorar		OK
-31.39352293	Ignorar		OK
86.0307052	Mudança grande	Colidiu em PRATO 1	ACERTOU COLISAO
-5.49954083	Ignorar		OK
5.599339337	Ignorar		OK
-37.35959324	Ignorar		OK
26.46287424	Ignorar		OK
12.16018162	Ignorar		OK
-10.72356657	Ignorar		OK

Fonte: obtido pelo autor, 2019.

Tabela A.2: **Trecho 2 (Frontal)**: Tabela com o resultado final da heurística usando a variação angular para verificar se houve colisão — Threshold de variação angular: 60

Variação angular	*Relevância*	*Colisão*	*Validação*
-0.2997594855	Ignorar		OK
-27.21340097	Ignorar		OK
10.19594843	Ignorar		OK
2.312544031	Ignorar		OK
-11.71164915	Ignorar		OK
1.839108728	Ignorar		OK
17.74467163	Ignorar		OK
-35.27523117	Ignorar		OK
39.2072035	Ignorar		OK
-7.548649733	Ignorar		OK
-50.50121744	Ignorar		OK
-105.8422171	Mudança grande	Não colidiu	OK
162.7685058	Mudança grande	Colidiu em PRATO 1	ACERTOU COLISAO
0.6539321841	Ignorar		OK
-9.124874749	Ignorar		OK
-6.979129172	Ignorar		OK
20.31322587	Ignorar		OK
1.218875235	Ignorar		OK
-180	Mudança grande	Não colidiu	OK
180	Mudança grande	Não colidiu	OK
0	Ignorar		OK
0	Ignorar		OK
0	Ignorar		OK
-30.25643716	Ignorar		OK
13.53794032	Ignorar		OK
-15.97534143	Ignorar		OK
7.947653856	Ignorar		OK
-72.01136003	Mudança grande	Não colidiu	OK
69.80817855	Mudança grande	Colidiu em PRATO 1	ACERTOU COLISAO
15.60471573	Ignorar		OK
-64.00151408	Mudança grande	Não colidiu	OK
64.36434692	Mudança grande	Não colidiu	OK
-59.1904344	Ignorar		OK
47.3545047	Ignorar		OK
3.815460698	Ignorar		OK
-100.5789971	Mudança grande	Colidiu em TAMBOR 2	ERROU COLISAO - FALSO POSITIVO
-51.2209401	Ignorar		ERROU COLISAO - NAO DETECTOU
3.458112883	Ignorar		OK
158.9341977	Mudança grande	Colidiu em TAMBOR 2	ACERTOU COLISAO
2.9728551	Ignorar		OK
-37.93879665	Ignorar		OK
14.86657289	Ignorar		OK
-25.95304066	Ignorar		OK
-93.15700009	Mudança grande	Não colidiu	OK
138.9695847	Mudança grande	Colidiu em PRATO 1	ACERTOU COLISAO
-18.39898378	Ignorar		OK
-14.6334975	Ignorar		OK
41.68221883	Ignorar		OK
0	Ignorar		OK
0	Ignorar		OK
0	Ignorar		OK

Fonte: obtido pelo autor, 2019.

Tabela A.3: **Trecho 3 (Frontal)**: Tabela com o resultado final da heurística usando a variação angular para verificar se houve colisão — Threshold de variação angular: 60

Variação angular	*Relevância*	*Colisão*	*Validação*
-11.19020827	Ignorar		OK
9.547206861	Ignorar		OK
-6.545082088	Ignorar		OK
-139.0169661	Mudança grande	Não colidiu	OK
16.48278842	Ignorar		OK
110.1308002	Mudança grande	Não colidiu	OK
-112.2908916	Mudança grande	Não colidiu	OK
147.9696609	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-22.77895702	Ignorar		OK
-132.6475343	Mudança grande	Não colidiu	OK
150.2096579	Mudança grande	Não colidiu	OK
-136.0963128	Mudança grande	Não colidiu	OK
141.6792061	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-166.4554532	Mudança grande	Não colidiu	OK
-1.422834593	Ignorar		OK
166.2798533	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-26.17528998	Ignorar		OK
-125.9285024	Mudança grande	Não colidiu	OK
-17.21710004	Ignorar		OK
178.3535649	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-164.3340301	Mudança grande	Não colidiu	OK
135.1491618	Mudança grande	Não colidiu	OK
-138.650511	Mudança grande	Não colidiu	OK
99.44858823	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-40.95218241	Ignorar		OK
88.7473568	Mudança grande	Não colidiu	OK
-154.0722878	Mudança grande	Não colidiu	OK
172.0001902	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-171.3506401	Mudança grande	Não colidiu	OK
26.20437089	Ignorar		OK
118.6177971	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
4.069252857	Ignorar		OK
-137.1793297	Mudança grande	Não colidiu	OK
-8.79741071	Ignorar		OK
169.2173661	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-171.4630125	Mudança grande	Colidiu em PRATO 2	ERROU COLISAO - FALSO POSITIVO
154.6879561	Mudança grande	Não colidiu	OK
-147.5288077	Mudança grande	Não colidiu	OK
141.782888	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-138.949868	Mudança grande	Não colidiu	OK
153.4709225	Mudança grande	Não colidiu	OK
-163.8556612	Mudança grande	Não colidiu	OK
167.7336201	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-166.4401732	Mudança grande	Não colidiu	OK
34.69850673	Ignorar		OK
101.3699379	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
20.46518941	Ignorar		OK
-148.6472383	Mudança grande	Não colidiu	OK
-7.125016349	Ignorar		OK
164.7448813	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-171.8698976	Mudança grande	Não colidiu	OK
178.1523897	Mudança grande	Não colidiu	OK
-142.2875781	Mudança grande	Não colidiu	OK
119.996473	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-131.4886268	Mudança grande	Não colidiu	OK
134.4149681	Mudança grande	Não colidiu	OK
-145.238691	Mudança grande	Não colidiu	OK
156.0624139	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-168.0553913	Mudança grande	Não colidiu	OK
78.11134196	Mudança grande	Não colidiu	OK
FALTOU PONTOS	FALTOU PONTOS	FALTOU PONTOS	ERROU COLISAO - NAO DETECTOU
FALTOU PONTOS	FALTOU PONTOS	FALTOU PONTOS	FALTOU PONTOS

Fonte: obtido pelo autor, 2019.

Tabela A.4: **Trecho 1 (Lateral)**: Tabela com o resultado final da heurística usando a variação angular para verificar se houve colisão — Threshold de variação angular: 80

Variação angular	*Relevância*	*Colisão*	*Validação*
-16.45739766	Ignorar		OK
15.74381164	Ignorar		OK
-7.861756927	Ignorar		OK
9.944412177	Ignorar		OK
-34.0538893	Ignorar		OK
33.77924832	Ignorar		OK
-18.1014097	Ignorar		OK
-8.933162107	Ignorar		OK
-45.63451622	Ignorar		OK
48.86648627	Ignorar		OK
-12.25303087	Ignorar		OK
-127.4990604	Mudança grande	Não colidiu	OK
164.3719994	Mudança grande	Colidiu em PRATO 1	ACERTOU COLISAO
-10.03820137	Ignorar		OK
1.925241223	Ignorar		OK
0.5065016307	Ignorar		OK
-1.025061966	Ignorar		OK
-0.623173124	Ignorar		OK
-8.119394149	Ignorar		OK
-4.499233063	Ignorar		OK
-27.95900216	Ignorar		OK
-41.7615319	Ignorar		OK
69.09716104	Ignorar		OK
-123.1827408	Mudança grande	Não colidiu	OK
136.4457269	Mudança grande	Colidiu em PRATO 1	ACERTOU COLISAO
2.503627285	Ignorar		OK
-44.04292159	Ignorar		OK
22.98857561	Ignorar		OK
17.11891491	Ignorar		OK
-31.60822312	Ignorar		OK
25.83647748	Ignorar		OK
7.30656895	Ignorar		OK
-9.635906007	Ignorar		OK
-17.63291229	Ignorar		OK
-61.35916796	Ignorar		OK
72.0251457	Ignorar		OK
24.73885239	Ignorar		OK
-131.2284487	Mudança grande	Não colidiu	ERROU COLISAO - NAO DETECTOU
111.5064816	Mudança grande	Não colidiu	OK
14.01316379	Ignorar		OK
2.121152083	Ignorar		OK
-2.698833649	Ignorar		OK
-3.735994141	Ignorar		OK
-0.3724615211	Ignorar		OK
-8.743109734	Ignorar		OK
-21.04232631	Ignorar		OK
-6.327260062	Ignorar		OK
-29.57627023	Ignorar		OK
39.06089339	Ignorar		OK
-114.9247044	Mudança grande	Não colidiu	OK
142.3148884	Mudança grande	Colidiu em PRATO 1	ACERTOU COLISAO

Fonte: obtido pelo autor, 2019.

Tabela A.5: **Trecho 2 (Lateral)**: Tabela com o resultado final da heurística usando a variação angular para verificar se houve colisão — Threshold de variação angular: 80

Variação angular	*Relevância*	*Colisão*	*Validação*
147.9022012	Mudança grande	Colidiu em TAMBOR 1	ACERTOU COLISAO
-18.32848487	Ignorar		OK
13.02673069	Ignorar		OK
1.487867529	Ignorar		OK
-42.95652205	Ignorar		OK
-90.43959729	Mudança grande	Não colidiu	OK
140.6482474	Mudança grande	Não colidiu	OK
-22.8161505	Ignorar		OK
-104.4701132	Mudança grande	Não colidiu	OK
-42.28107629	Ignorar		ERROU COLISAO - NAO DETECTOU
1.254165224	Ignorar		OK
164.2580022	Mudança grande	Colidiu em TAMBOR 1	ACERTOU COLISAO
-128.7971363	Mudança grande	Não colidiu	OK
11.86190813	Ignorar		OK
-41.1314579	Ignorar		OK
149.6200851	Mudança grande	Não colidiu	OK
-151.7694057	Mudança grande	Não colidiu	OK
158.4630164	Mudança grande	Colidiu em TAMBOR 1	ACERTOU COLISAO
-12.41810508	Ignorar		OK
-2.47049497	Ignorar		OK
-10.48970547	Ignorar		OK
-40.62099894	Ignorar		OK
-13.75313317	Ignorar		OK
47.99228323	Ignorar		OK
-129.3606035	Mudança grande	Não colidiu	OK
163.3757673	Mudança grande	Colidiu em TAMBOR 1	ACERTOU COLISAO
-8.523858315	Ignorar		OK
-3.6373234	Ignorar		OK
-18.12439142	Ignorar		OK
-10.32019186	Ignorar		OK
-76.95326638	Ignorar		OK
83.34672367	Mudança grande	Não colidiu	OK
-110.9510781	Mudança grande	Não colidiu	OK
140.6158097	Mudança grande	Colidiu em TAMBOR 1	ACERTOU COLISAO
-5.710593137	Ignorar		OK
1.063918972	Ignorar		OK
-0.5574555677	Ignorar		OK
2.944068981	Ignorar	Não colidiu	OK
-1.048081288	Ignorar		OK
-20.38800595	Ignorar		OK
13.23594244	Ignorar		OK
-160.5278757	Mudança grande	Não colidiu	OK
172.675753	Mudança grande	Não colidiu	OK
0.5601364234	Ignorar		OK
-137.1848635	Mudança grande	Não colidiu	OK
106.4259874	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
28.8073848	Ignorar		OK
5.119368978	Ignorar		OK
-167.9258603	Mudança grande		OK
162.9467626	Mudança grande		OK
-33.46769815	Ignorar	Não colidiu	OK
-114.0842642	Mudança grande	Não colidiu	OK
106.1540839	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO

Fonte: obtido pelo autor, 2019.

Tabela A.6: **Trecho 3 (Lateral)**: Tabela com o resultado final da heurística usando a variação angular para verificar se houve colisão — Threshold de variação angular: 80

Variação angular	*Relevância*	*Colisão*	*Validação*
23.90763465	Ignorar		OK
-139.1640802	Mudança grande	Não colidiu	OK
148.7410017	Mudança grande	Colidiu em TAMBOR 1	ACERTOU COLISAO
14.95542882	Ignorar		OK
-79.39421094	Ignorar		OK
-36.32359689	Ignorar		OK
100.2986293	Mudança grande	Não colidiu	OK
-140.4306224	Mudança grande	Não colidiu	OK
148.7181003	Mudança grande	Colidiu em TAMBOR 1	ACERTOU COLISAO
-1.095458966	Ignorar		OK
10.68616031	Ignorar		OK
-105.1144419	Mudança grande	Não colidiu	OK
39.08170474	Ignorar		OK
-110.5919704	Mudança grande	Não colidiu	OK
177.8230907	Mudança grande	Colidiu em PRATO 2	ACERTOU COLISAO
-15.28717507	Ignorar		OK
-2.331207414	Ignorar		OK
-122.410409	Mudança grande	Não colidiu	OK
71.06350189	Ignorar		OK
20.85241549	Ignorar		OK
-88.40315318	Mudança grande	Não colidiu	OK
134.0779333	Mudança grande	Colidiu em PRATO 3	ACERTOU COLISAO
-11.0141168	Ignorar		OK
6.121578932	Ignorar		OK
-9.578864858	Ignorar		OK
-49.6030942	Ignorar		OK
47.18072151	Ignorar		OK
-147.646974	Mudança grande	Não colidiu	OK
160.0040121	Mudança grande	Não colidiu	ERROU COLISAO - NAO DETECTOU
-5.600050162	Ignorar		OK
-140.3327355	Mudança grande	Não colidiu	OK
129.3617073	Mudança grande	Não colidiu	OK
20.59313833	Ignorar		OK
-158.2300726	Mudança grande	Não colidiu	OK
151.1607545	Mudança grande	Não colidiu	ERROU COLISAO - NAO DETECTOU
0.0787628948	Ignorar		OK
-86.79154453	Mudança grande	Não colidiu	OK
68.14238312	Ignorar		OK
-140.0980077	Mudança grande	Não colidiu	OK

Fonte: obtido pelo autor, 2019.

APÊNDICE B — EXEMPLO DE RESULTADOS DOS TESTES REALIZADOS PARA OBTER AS MÉTRICAS SOBRE A SOLUÇÃO FINAL

B.1 Exemplo de informação real sobre as colisões em um clipe

Tabela B.1: Vídeo 1 - Clipe 1 - Mão dominante - Tabela com *frames* nos quais existem colisões em instrumentos - Considerada a informação real sobre as colisões

Frame	Rótulo da Peça Colidida
4	0
12	0
22	0
30	0
40	0
49	0
58	0
67	0
77	0
86	0
94	0
103	0
113	0
121	0
131	0
140	1

Fonte: obtido pelo autor, 2019.

B.2 Exemplo de saída de aplicações do algoritmo sobre os clipes selecionados

Arquivo B.1 – Arquivo de saída do algoritmo: Vídeo 1 - Clipe 1 - Mão dominante - Execução 1


```
{
   "drums":[
      {
         "label":"instrument 0",
         "time":500.0,
         "frame":14
      },
      {
         "label":"instrument 0",
         "time":1100.0,
         "frame":32
      },
      {
         "label":"instrument 0",
         "time":1433.3333333333333,
         "frame":42
      },
      {
         "label":"instrument 0",
         "time":2000.0,
         "frame":59
      },
      {
         "label":"instrument 0",
         "time":2300.0,
         "frame":68
      },
      {
         "label":"instrument 0",
         "time":2700.0,
         "frame":80
      },
      {
         "label":"instrument 0",
         "time":2966.6666666666665,
         "frame":88
      },
      {
         "label":"instrument 0",
         "time":3233.3333333333335,
         "frame":96
      },
      {
         "label":"instrument 0",
         "time":3533.3333333333335,
```

```
            "frame":105
        },
        {
            "label":"instrument 0",
            "time":4100.0,
            "frame":122
        },
        {
            "label":"instrument 0",
            "time":4466.666666666667,
            "frame":133
        }
    ],
    "duration":5000.0,
    "frames":150
}
```