UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS DE CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MURILLO ANDRÉ MALESKI

DESENVOLVIMENTO DE UM JOGO COM REALIDADE AUMENTADA PARA AUXILIAR NA APRENDIZAGEM DE PIANO

CHAPECÓ
2021

MURILLO ANDRÉ MALESKI

DESENVOLVIMENTO DE UM JOGO COM REALIDADE AUMENTADA PARA AUXILIAR NA APRENDIZAGEM DE PIANO

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.
Orientador: Prof. Dr. Fernando Bevilacqua

CHAPECÓ
2021

Maleski, Murillo André

Desenvolvimento de um jogo com Realidade Aumentada para auxiliar na aprendizagem de piano / Murillo André Maleski. – 2021.
58 f.: il.

Orientador: Prof. Dr. Fernando Bevilacqua.
Trabalho de conclusão de curso (graduação) – Universidade Federal da Fronteira Sul, curso de Ciência da Computação, Chapecó, SC, 2021.

1. Realidade Aumentada. 2. Piano. 3. Detecção. 4. Jogos. I. Bevilacqua, Prof. Dr. Fernando, orientador. II. Universidade Federal da Fronteira Sul. III. Título.

E-mail: murillomaleski@gmail.com

MURILLO ANDRÉ MALESKI

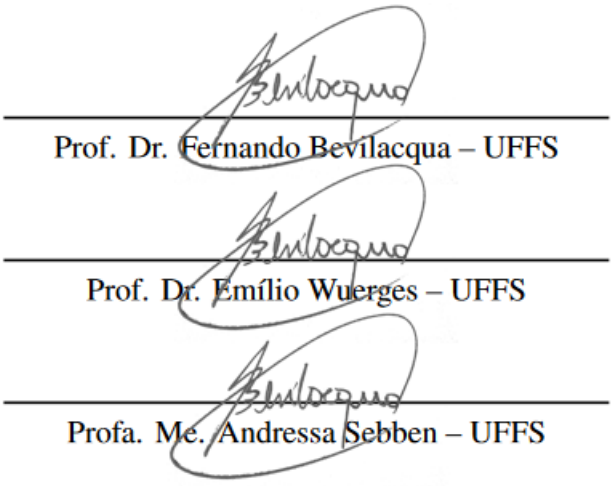
DESENVOLVIMENTO DE UM JOGO COM REALIDADE AUMENTADA PARA AUXILIAR NA APRENDIZAGEM DE PIANO

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Dr. Fernando Bevilacqua

Este trabalho de conclusão de curso foi defendido e aprovado pela banca avaliadora em: 20/05/2021.

BANCA AVALIADORA



RESUMO

Aprender piano é uma experiência lenta e exaustiva sendo necessário dedicar muito tempo e entusiasmo para conseguir pequenos avanços e, por este motivo, muitos estudantes ficam desmotivados e acabam por desistir. Para contornar a situação, existem ferramentas musicais que auxiliam no aprendizado, e estão cada vez mais robustas, apostando em áreas inovadoras como a realidade virtual e a realidade aumentada. Neste trabalho, são apresentados estudos sobre diferentes modelos de detecção de vídeo para identificar o modelo ideal que possibilite o desenvolvimento um de um jogo com *Head-Mounted-Display* (HMD) para auxiliar no aprendizado de piano, no qual o jogador pode utilizar o próprio teclado ou piano para realizar as ações do jogo, assim melhorando suas habilidades de piano de maneira intuitiva e divertida.

Palavras-chave: Realidade Aumentada. Piano. Detecção. Jogos.

ABSTRACT

Learning piano is a slow and exhaustive experience being needed to dedicate a lot of time and enthusiasm to get small advances and, for this reason, many students get unmotivated and end up giving up. To work around the situation, there are musical tools that assist in learning, and are increasingly robust, investing in innovative areas such as virtual reality and augmented reality. In this work, studies are presented on different models of video detection to identify the ideal model that allows the development of a game with Head-Mounted-Display (HMD) to assist in piano learning, in which the player can use the keyboard or piano to perform the actions of the game, thus improving your piano skills intuitively and fun.

Keywords: Augmented Reality. Piano. Detection. Games.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 APRESENTAÇÃO

Aprender piano pode ser uma experiência lenta e exaustiva, necessitando muito esforço e dedicação para produção de algumas simples notas. Durante o estudo de piano, o aprendizado pode ocorrer de maneira formal (ou tradicional) e de maneira informal (SILVA, 2010), utilizando outras técnicas e ferramentas para o aprendizado. A forma tradicional envolve o estudo da história da música, das técnicas de execução e de aprendizagem além do estudo do repertório pianístico. Já o aprendizado informal pode ocorrer culturalmente pelo convívio familiar ou também em grupos com a música, ouvindo, discutindo, além da escolha do repertório com o qual o estudante mais se identifica. De acordo com Silva (2010), há uma tensão entre as duas práticas principalmente no que diz respeito ao interesse do aluno. Em suma, as práticas informais são complementares às práticas formais, devendo ser utilizadas como formas de incentivo e aprimoramento dos conhecimentos formais.

As tecnologias mais recentes trazem propostas interessantes de ferramentas e recursos auxiliares para o aprendizado, como é o caso da Realidade Virtual (RV) e da Realidade Aumentada (RA). Esses recursos permitem a inclusão de objetos e ideias fictícias em um ambiente teórico. Há possibilidade, então, de criar o próprio mundo virtualmente e também acrescentar elementos ao mundo real vistos pelo plano virtual. Pode-se visualizar objetos, animais e até visitar incontáveis lugares sem precisar sair de casa. Inclusive é possível vivenciar a experiência de filmes e jogos através do olhar do personagem, como se o próprio usuário estivesse presente na cena.

Vários jogos apresentam a ideia de que o usuário é o próprio controle, de modo que não há necessidade de possuir um controle físico para jogar. Basta apenas mover alguma parte do corpo para que as ações dentro do jogo sejam executadas. A RA e RV aproveitam esse conceito para transformar o próprio jogador no personagem do jogo, em que todas as ações e decisões são visualizadas e tomadas da perspectiva de quem está jogando.

O jogo Synthesia, ilustrado na Figura 1, oferece a experiência de diversão enquanto auxilia no aprendizado de piano, principalmente na memorização das tablaturas[1]. O jogo mostra um teclado virtual e as teclas a serem pressionadas em seu tempo correto. O Synthesia pode ainda ser conectado a um teclado eletrônico, tornando a experiência ainda mais realista.

Em luz do disposto, seria interessante unir a ideia de incentivo dos jogos e a RA como uma ferramenta de aprendizagem musical no estilo Synthesia, dizendo para o jogador o que ele deve fazer e se esta cumprindo o objetivo corretamente. Para isso é necessário estudar um modelo de detecção que possibilite a criação de um jogo de RA, utilizando um teclado musical Casio CTK-496. Este trabalho irá fazer um estudo sobre diferentes modelos de detecção aplicados sobre o teclado, verificando se as teclas poder ser detectadas e utilizadas como objeto de *tracking*

[1] A tablatura é um sistema de notação que mostra a posição dos dedos ao tocar e o ritmo.

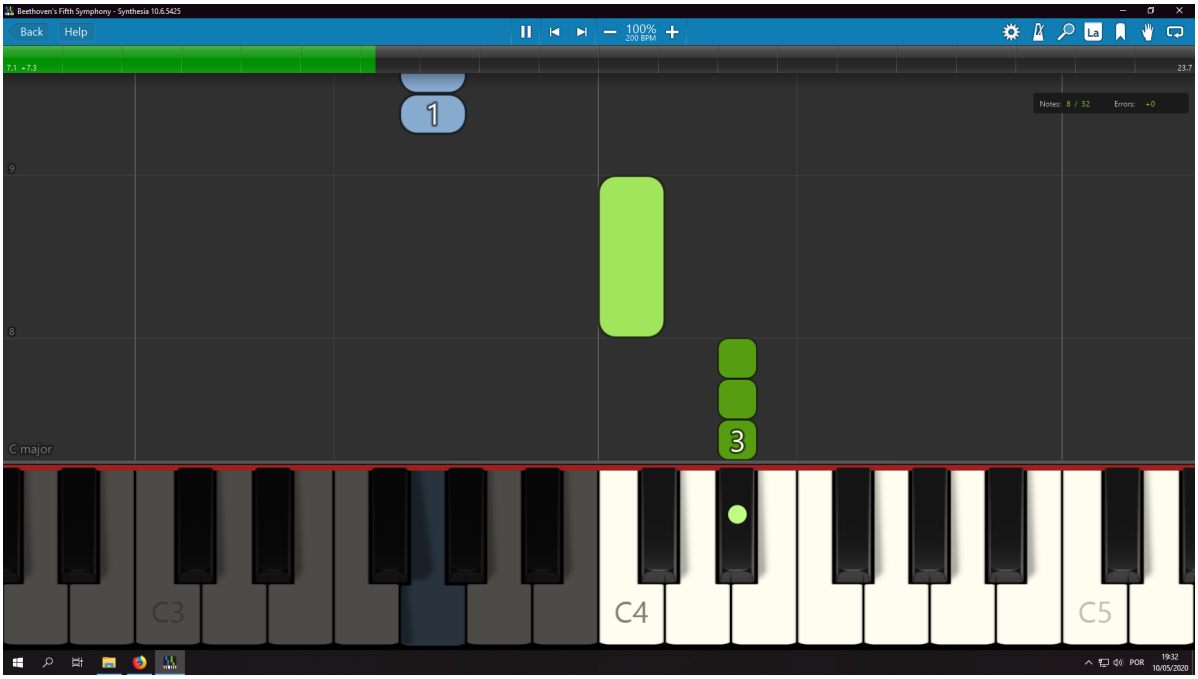


Figura 1 – Demonstração do jogo Synthesia.

para RA. Dentre os modelos será apontada a opção ideal para servir como base do jogo, no qual o objetivo é conseguir um desempenho próximo a 30 FPS possibilitando uma boa experiência com RA. Por fim serão demonstrados os passos para criação de um jogo de ritmo, carregando uma música e exibindo as notas em direção as teclas no estilo Synthesia.

Na seção a seguir será apresentada a problemática que motivou este projeto. Após, será realizada uma análise das abordagens, aplicações e sistemas de aprendizagem de piano de acordo com Seidler (2019), assim como uma breve explicação sobre alguns conceitos de piano utilizados neste trabalho. Em seguida será analisado o fator desempenho de cada aplicação e sua abordagem. Por fim, será proposta uma nova abordagem com o intuito de obter uma aplicação com melhor desempenho em aprendizagem de piano.

1.2 PROBLEMÁTICA

Dentro do cenário global e das inovações tecnológicas pode-se notar a grande busca por soluções e ferramentas para auxiliar e acelerar o processo de aprendizagem. É notória a dificuldade de percepção de resultados quando o assunto é música, devido à grande demanda de tempo e dedicação para adquirir as habilidades desejadas. Diante dessa complexidade para a evolução do aprendizado, muitas pessoas desistem da música, quando apenas precisariam de incentivo ou ferramentas para facilitar esta jornada.

Dessa forma, a utilização do Synthesia para aprender piano torna o processo mais simples e prático, pois a visualização da tablatura em tempo real é mais intuitiva e até mesmo mais interessante do que fazer a leitura da partitura. Um ponto que poderia ser aprimorado é a tradução da tablatura do Synthesia para o teclado real do piano, uma vez que é necessário

observar na tela do jogo qual tecla pressionar e em seguida mudar o foco para o teclado físico. Essa transição de foco acaba gerando uma carga cognitiva maior, o que pode ser mais oneroso do que benéfico no aprendizado.

Um dos grandes problemas apresentado nos trabalhos relacionados é o desempenho da aplicação. Criar uma aplicação que detecta objetos do mundo real demanda muito custo computacional e recursos de processamento. O principal desafio deste projeto é detectar as teclas do piano obtendo uma boa taxa de quadros para a exibição, de modo a trazer a RA o mais próximo possível da visão humana, reduzindo assim os sintomas causados devido ao uso prolongado dessas tecnologias, em que os usuários apresentam sintomas adversos como náusea e dor de cabeça.

Em meio a esse contexto, há uma lacuna para a pesquisa e exploração de novas ferramentas para auxilixar no ensino de música, principalmente aquelas contextualizadas com RA e RV. Somam-se a isso as dificuldades em termos de Interface Humano-Computador (IHC), visto que a mera utilização de novos recursos não implica, necessariamente, em um melhor aprendizado. A exploração de técnicas de visão computacional e IHC no contexto de aprendizado de piano podem criar novas oportunidades para aprimorar materiais de ensino musical. O presente trabalho pretende contribuir com essa lacuna, através da exploração do estado da arte de RA e RV aplicados ao ensino musical de piano. Adicionalmente, pretende-se desenvolver um aplicativo para que os conceitos e ideias sejam testados.

2 OBJETIVOS

2.1 OBJETIVOS GERAIS

Explorar as técnicas de detecção de vídeo para permitir o uso de realidade aumentada em um sistema Head-Mounted Display (HMD) na plataforma Android para um jogo para o auxílio de técnicas musicais no piano.

2.2 OBJETIVOS ESPECÍFICOS

- Explorar as técnicas de detecção de superficie para detectar as teclas do piano;
- Propor um novo modelo de detecção de teclas em um piano combinando as técnicas analisadas;
- Analisar e comparar diferentes metodologias para a detecção de teclas em um piano/teclado.
- Apontar a abordagem mais indicada para trabalhar com detecção de teclas em um piano/-teclado.

3 JUSTIFICATIVA

O processo de aprendizagem se torna mais fácil e eficiente quando são utilizadas técnicas, recursos e ferramentas que contribuem na prática e exercício da aprendizagem. A teoria da carga cognitiva se preocupa com as maneiras com as quais cada recurso cognitivo da mente humana está focado ou é utilizado durante a aprendizagem ou resolução de problemas (CHANDLER; SWELLER, 1991). Mais formas de aprendizagem possibilitam o engajamento e a estimulação da pessoa que está em processo de aprendizagem. Nesse aspecto, o ponto principal é fazer a dosagem da quantidade de conteúdo em diferentes formas de aprendizagem, implicando assim na automação do aprendizado (SWELLER, 1988). Nesse sentido a quantidade de elementos a serem processados cognitivamente torna justificável o uso de ferramentas de aprendizagem, uma vez que as ferramentas servem para dividir o foco e a forma de aquisição das informações.

Como citado anteriormente, um dos desafios deste projeto será realizar a integração visual da tablatura com o teclado do piano. Diante disso, será utilizada RA para exibir a tablatura em tempo real sobre o teclado físico do piano. A realidade virtual e aumentada já provaram serem eficientes quando combinadas a uma ferramenta computacional ou a um jogo interativo. Na medicina, por exemplo, a RV foi capaz de melhorar significativamente o quadro de pacientes com AVCs (ARAÚJO et al., 2014), graças ao incentivo promovido pela tecnologia. Os trabalhos relacionados apresentam modelos de aplicações para auxiliar na memorização das tablaturas de piano, dentre eles percebe-se que a visualização da tablatura movendo-se em direção à tecla que deve ser pressionada pode ser considerada a mais eficiente em questão de aprendizagem. Em parte, isso pode ser explicado pelo fato de que técnicas de RA sobrepõem objetos virtuais sobre o mundo real, reduzindo a troca de contexto cognitivo do usuário. Isso acontece, principalmente, porque o usuário de RA não precisa focar em dois contextos completamente isolados, como a tela do computador e o piano. O foco está inteiramente no piano, que é melhorado com elementos digitais sobrepostos.

Isso justifica a utilização de RA no contexto de ensino musical, visto que o ensino não está mais limitado ao mundo físico, incorporando-se novos elementos, como técnicas IHC. De maneira a aprimorar a experiência em RA, a aplicação pode ser, por exemplo, gamificada. A gamificação tem potencial para influenciar, engajar e motivar pessoas (KAPP, 2012), aumentando assim o nível de interesse nos jogos, uma vez que estes são construídos de forma a fixar a atenção do usuário por longos períodos.

Neste trabalho será proposto um modelo de aplicação Android, sem necessidade de marcações fiduciais, apenas a área natural do teclado ou piano. De acordo com Seidler (2019), um algoritmo de deteção de características naturais tende a ter melhor precisão em toda largura do teclado do que utilizando marcas fiduciais. Pretende-se obter um desempenho com taxa de quadros próximo ou superior a 30 quadros por segundo, com o intuito de deixar a experiência de RA o mais agradável possível. Por ter uma ampla variedade de bibliotecas de RA e RV disponíveis, a plataforma Android foi escolhida como base do projeto. Isso reduzirá custos

além de permitir que a tecnologia produzida possa ser disponibilizada a outros pesquisadores (e usuários) sem ou com custo reduzido.

4 REVISÃO BIBLIOGRÁFICA

4.1 GAMIFICAÇÃO

A gamificação fornece um contexto para a construção de um sentido mais amplo para a interação, tanto nas escolas como em outros ambientes de aprendizagem. Isso potencializa a participação e a motivação dos indivíduos inseridos nesses ambientes (FARDO, 2013). Games também são ferramentas valiosas para criar experiências significativas, impactando positivamente na aprendizagem do indivíduo.

No estudo de Chang, Chen e Huang (2011), utilizando um console Xbox 360 com Kinect, alguns pacientes são submetidos a um período de reabilitação motora virtual. O resultado é um aumento significativo no número de movimentos corretos quando submetidos a seções de jogos, conforme ilustrado na Figura 2.

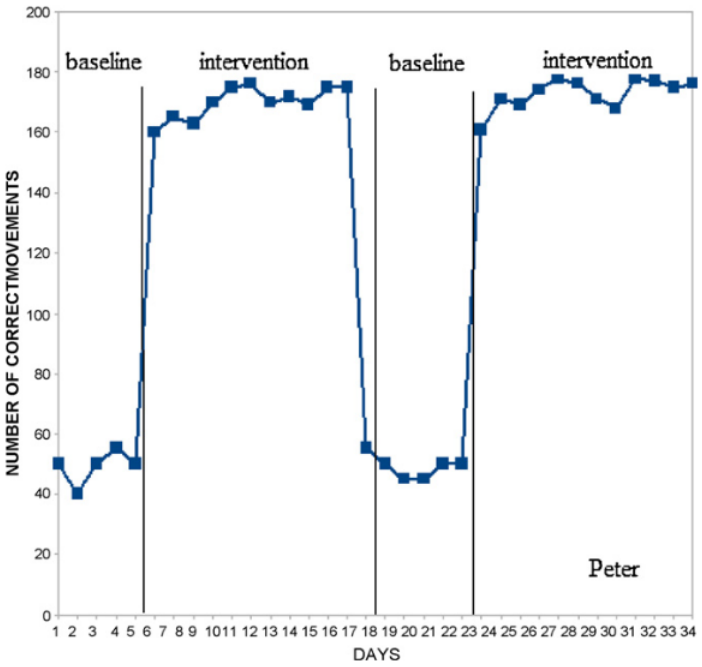


Figura 2 – Números dos movimentos corretos na reabilitação do paciente Peter durante o estudo de Chang, Chen e Huang (2011). *Intervention* se refere ao período na qual o paciente foi submetido a seções de jogos.

É importante observar que quanto maior o número de seções de jogos a que o paciente era submetido, maior era o crescimento do número de movimentos corretos. Isso significa que o jogo atuou de maneira incentivadora no tratamento do paciente. A pesquisa realizada por Araújo et al. (2014) utilizou da RV para reabilitação do membro superior parético por AVC de um paciente. Novamente foi notável a melhora gerada pelo jogo de RV. O incentivo gerado pela proposta dos jogos age como um recurso terapêutico na recuperação desses pacientes.

4.2 CONCEITOS PARA PIANO

O piano é um instrumento musical composto normalmente por 88 teclas, cada uma representando uma nota musical. Alguns conceitos importantes sobre o piano utilizados neste projeto estão descritos a seguir.

4.2.1 Oitavas

Na música existe o conceito de **oitavas**, cada oitava representa um intervalo entre uma nota musical e outra com metade ou dobro de sua frequência, assim funcionando com uma distância entre diferentes notas. O processo é ilustrado na Figura 3. Uma oitava é comumente representada por uma sequência de notas na forma: Dó, Ré, Mi, Fá, Sol, Lá, Si, Dó. A sequência começa e termina com o *Dó*, assim uma oitava segue a partir do término de outra.

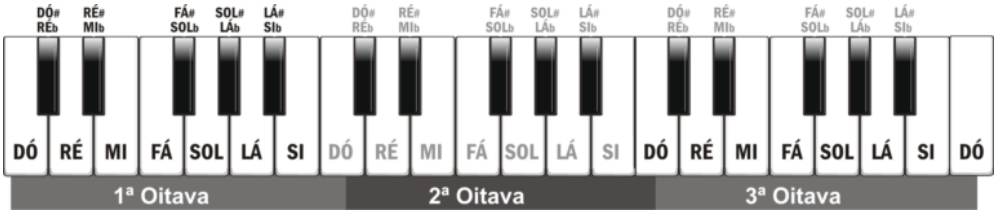


Figura 3 – Oitavas de um piano. Fonte: blog.teclasmagicas.com

4.2.2 Tablaturas

A tablatura é um sistema de notação que mostra ao instrumentista a posição dos dedos ao tocar, juntamente com o ritmo, ou o tempo correto a serem pressionadas as teclas. A tablatura não indica, porém, a duração de uma nota, apenas o tempo de início. As tablaturas são mais comumente encontradas em instrumentos com cordas, de modo que cada linha representa uma corda.

No piano a tablatura possui uma linha para cada oitava do piano. Cada tecla é representada por uma *Cifra*[1], as teclas pretas são representadas por letras maiúsculas e as teclas brancas por letras minúsculas. As cifras seguem a ordem: C, D, E, F, G, A, B referenciando às notas Dó, Ré, Mi, Fá, Sol, Lá, Si respectivamente. O processo é ilustrado na Figura 4.

Na leitura da tablatura podem aparecer alguns símbolos auxiliares como o símbolo | (barra horizontal), separando a tablatura em partes, e os simbolos R e L representando as mãos direita e esquerda respectivamente. Um exemplo de tablatura pode ser escrito da seguinte forma:

```
R 3|-g-a-b-|-g-a-b-|
L 2|-a-g---|-a-g---|
L 1|-a-g---|-a-g---|
```

[1] Cifra é a nota musical a ser interpretada, comumente representada por letras.

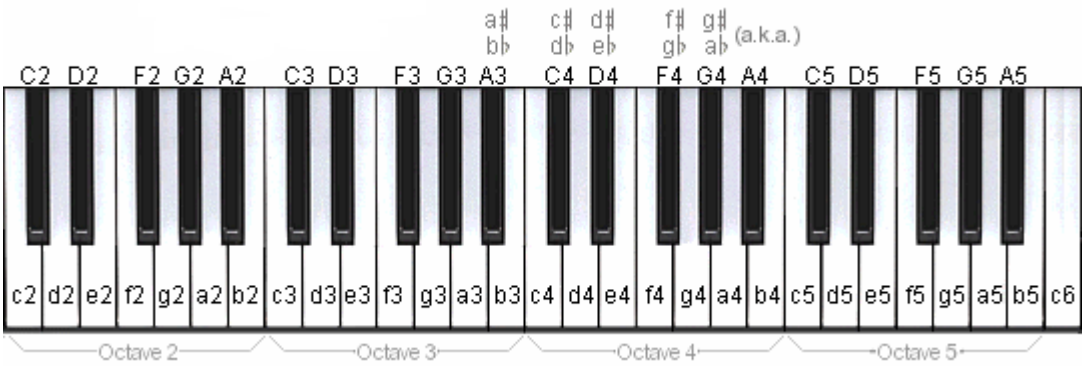


Figura 4 – Tablaturas de um piano. Fonte: tabnabber.com

4.2.3 Partituras

Uma partitura é uma representação escrita de música padronizada mundialmente. Tal como outro sistema de escrita, dispõe de símbolos próprios (notas musicais) que se associam a sons. A partitura é a representação mais utilizada para piano.

A partitura é representada por 5 linhas desenhadas, sendo que cada linha e cada espaço representam uma nota musical diferente. As notas que aparecem fora do espaço desenhado são representadas com um pequeno traço, devendo ser interpretado como uma linha imaginária.

Em uma partitura de piano as oitavas são divididas em duas partes chamadas *Claves*. As mais utilizadas são a clave de Sol e de Fá, primeira representa as notas mais altas (agudas), e a segunda as notas mais baixas (graves). A clave de Sol aponta o posicionamento da nota Sol nas linhas partitura, indicando como deve ser feita a leitura das notas e seu posicionamento no teclado do piano. A clave de Fá, do mesmo modo, aponta a nota Fá na partitura. As Figuras 5 e 6 ilustram a relação entre a clave de Sol e Fá, respectivamente, e as teclas no teclado. Estas claves são usualmente utilizadas em conjunto em partituras de piano, de modo que a clave de Sol representa a mão direita e a de Fá a mão esquerda.

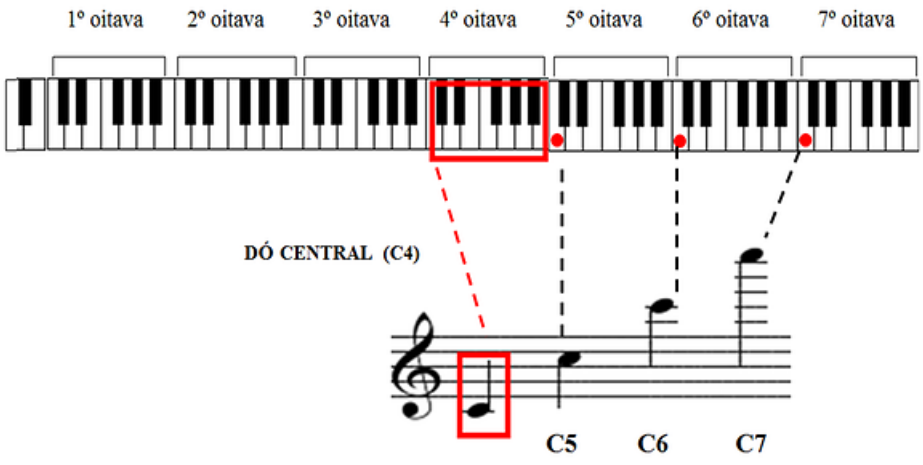


Figura 5 – Clave de Sol no piano. Fonte: www.descomplicandoamusica.com

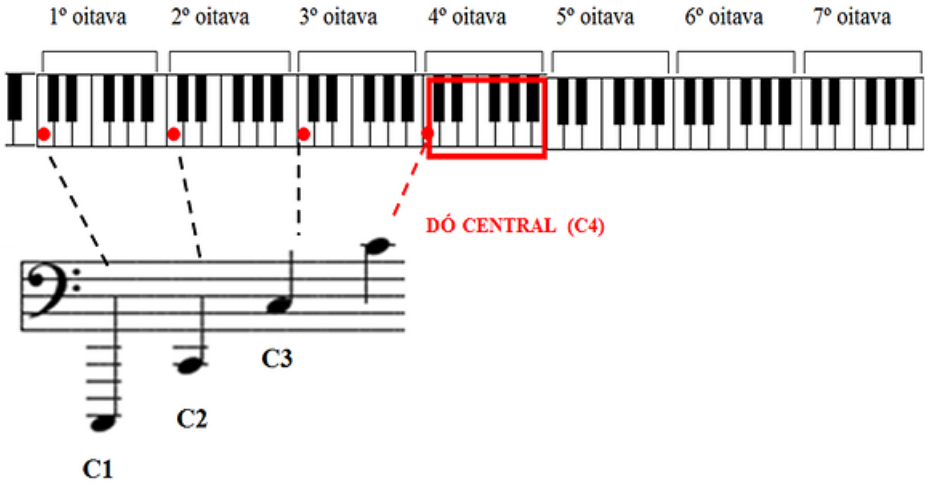


Figura 6 – Clave de Fa no piano. Fonte: www.descomplicandoamusica.com

Existe uma vasta área de estudo sobre partituras, porém esse tópico está fora do escopo deste projeto. Esta introdução ao assunto teve o intuito de clarificar a diferença entre partitura a tablatura. No contexto deste projeto, o termo tablatura será usado para referenciar a área do piano na qual as notas irão aparecer em direção à sua respectiva tecla. Essa é uma forma visual mais prática de enxergar uma tablatura.

4.2.4 MIDI

Musical Instrument Digital Interface (MIDI) é o protocolo de comunicação entre instrumentos musicais e computadores. Surgiu nos anos 80 para padronizar os hardwares de música. Conforme Back (2016), um arquivo MIDI armazena dados em sua estrutura, e não sinais musicais como outros formatos para armazenar música, por exemplo, o MP3[2]. Entre esses dados pode-se citar:

- Qual tecla é pressionada ou solta;
- Quão forte a tecla é pressionada;
- Tempo (ou BPM) em que a tecla deve permanecer pressionada;
- Volume;

Um teclado eletrônico ou um piano digital possuem conectividade MIDI para interagir com softwares computacionais. Neste trabalho, o conceito de MIDI será utilizado apenas para carregar músicas e as transformar em tablaturas virtuais, não tendo necessidade de conectar em um piano ou teclado.

[2] http://www.mpgedit.org/mpgedit/mpeg_format/mpeghdr.htm

4.3 TRABALHOS RELACIONADOS

Um dos grandes desafios da RA é detectar os objetos do mundo real, ou reconhecer o meio onde serão inseridos os objetos virtuais. Mais desafiador ainda é conseguir um desempenho aceitável nas técnicas de reconhecimento, de modo que o resultado seja uma imagem fluida e boa o suficiente para ser utilizada em um HMD sem causar mal-estar ao usuário. Neste trabalho, será necessário detectar a posição de um piano ou teclado no ambiente e criar a disposição virtual das teclas, assim, criando objetos virtuais para representar a tecla a ser pressionada.

Existem diversas formas de detecção das teclas, diferindo entre si pelo desempenho e pela finalidade da aplicação. Nos trabalhos analisados, predominou o uso de marcações fiduciais como modelo de detecção. As marcações fiduciais são imagens situadas no mundo real com carcterísticas visuais fáceis de serem identificadas e reconhecidas (HOFMAM et al., 2006).

Chow et al. (2013) apresenta uma abordagem utilizando múltiplas marcações para detectar as teclas, separando cada uma em distâncias diferentes e assim reconhecendo em qual divisão do piano está posicionado. Segundo o autor a utilização de várias marcações idênticas aumentou drasticamente o desempenho da detecção. Esta técnica está ilustrada na Figura 7.

Figura 7 – Abordagem de (CHOW et al., 2013).

As marcações estão dispostas em distâncias únicas, assim o algoritmo é capaz de reconhecer qual parte do teclado a câmera está capturando apenas medindo as distâncias entre as marcações.

Outra abordagem com múltiplas marcações é apresentada por Zeng, He e Pan (2019). Nesse caso, percebe-se a dificuldade de mapear as teclas usando apenas uma marca, devido às teclas serem muito parecidas e o campo de visão estar limitado a apenas um quadrante do piano. Nesta abordagem os autores adotaram uma marcação diferente em cada oitava do piano, permitindo assim, que a aplicação reconheça qual parte do teclado está no quadro da câmera. A

disposição das mútiplas marcações está ilustrada na Figura 8.

Figura 8 – Abordagem de (ZENG; HE; PAN, 2019)

Também é possível utilizar uma abordagem sem o emprego de marcas fiduciais, conhecida como *Natural Feature Tracking* (NFT), reconhecendo as teclas através de técnicas de visão computacional. Segundo Huang et al. (2011), a utilização deste método de maneira geral é a forma mais popular, uma vez que produz uma experiência mais natural sem necessidade de assistência de marcações.

Em sua abordagem, Huang et al. (2011) captura a imagem inteira de um teclado para fazer o reconhecimento das teclas, conforme ilustrado na Figura 9. Utilizando um *threshold* binário o algoritmo reconhece a área das teclas e em seguida faz uma série de cálculos vetoriais para definir a posição de cada tecla. A aplicação consegue uma precisão de detecção muito boa atingindo uma média de 15 FPS. Esse é um resultado considerável para o porte da aplicação, porém, para uma experiência de jogo com realidade aumentada, é necessário otimização para melhorar a taxa de quadros.

Figura 9 – Abordagem de (HUANG et al., 2011)

Goodwin e Green (2013) apresenta um reconhecimento de teclas por visão computacional, ilustrado na Figura 10. A câmera captura uma parte do teclado e o algoritmo identifica as linhas da área das teclas e em seguida calcula a medida das mesmas de acordo com as descrições de JG Savard (2011). A aplicação atingiu uma média de 25.1 FPS utilizando uma webcam de

notebook convencional. Segundo o autor o resultado está longe do ideal, pois muitas vezes as bordas das teclas detectadas ficam fora do teclado real. Além de problemas com sombras e baixa iluminação que afetam significativamente o resultado.

Figura 10 – Abordagem de (GOODWIN; GREEN, 2013)

A aplicação HoloKeys (HACKL; ANTHES, 2017), ilustrada na Figura 11, utiliza uma marca fiducial simples para reconhecimento da posição do piano ou teclado. Nesse caso, uma configuração de tamanho das teclas informa à aplicação como deve ser a disposição das teclas. A aplicação gera um teclado virtual de 88 teclas onde o plano de teclas será posicionado. Esse teclado virtual fica aproximadamente posicionado sobre o teclado físico, oferecendo a noção de qual tecla deve ser posicionada no momento correto.

Figura 11 – Abordagem de (HACKL; ANTHES, 2017).

Seidler (2019) apresenta um modelo misto de técnicas para detecção de teclas. O autor faz uma análise dos trabalhos realizados até o presente momento e propõe um modelo que mistura visão computacional com a detecção de marcas fiduciais. De acordo com as análises do autor, o modelo de Huang et al. (2011) necessita que a câmera capture todas as teclas no mesmo quadro, o que exige um campo de visão (*Field Of View* - FOV) muito abrangente e de difícil

obtenção utilizando um HMD. Logo é necessário utilizar as marcas fiduciais para reconhecer quais oitavas do piano estão no quadro, como descrito nas abordagens de Hackl e Anthes (2017) assim como a de Chow et al. (2013), que empregaram marcações em pontos externos às teclas. Seidler (2019) propõe utilização de pequenas marcas fiduciais nas próprias teclas, pois nem todos os teclados ou pianos possuem espaço ou local para posicionar as marcações fora do espaço das teclas.

Nesse modelo, duas marcas fiduciais em cor vermelha são posicionadas em cima de teclas brancas de diferentes oitavas, conforme ilustrado na Figura 12. A distância entre as marcas foi calculada pelo autor de modo que uma delas sempre esteja dentro do FOV, permitindo saber qual oitava está sendo exibida. Através da visão computacional detecta-se a exata posição das teclas e mostra-se o tempo certo que cada uma deveria ser pressionada. A aplicação tem uma taxa de quadros média de 234 FPS, resultando em poucas falhas. Isso demonstra que o projeto é promissor para auxiliar no aprendizado de piano.

Figura 12 – Abordagem de (SEIDLER, 2019).

Por fim, Seidler (2019) também aponta que um algoritmo de detecção capaz de rastrear as características naturais de um teclado poderia apresentar uma melhor precisão em toda largura de um teclado. Também seria possível fazer a captura de quadros mais estreitos. Dessa forma, sua abordagem se focou em tentar eliminar as limitações tecnológicas atuais explorando pontos naturais no teclado.

5 IMPLEMENTAÇÃO

O presente trabalho consiste em descrever os passos para construção de um jogo, com a utilização de RA, para plataforma Android, no estilo Synthesia. O foco maior deste trabalho está em determinar os melhor modelo de detecção para obter um boa experiência com RA, permitindo assim que posso ser desenvolvido um jogo onde o jogador visualiza as teclas a serem tocadas e o tempo correto para pressioná-las em tempo real. Com a RA, as tablaturas do piano poderiam ser posicionadas sobre o teclado/piano utilizado, melhorando a intuitivadade da aplicação e facilitando a conversão da tablatura para as teclas. Este capítulo descreve o processo de implementação bem como os desafios e abordagens de cada etapa na construção do jogo.

Em uma aplicação de RA, é necessário um ponto de *tracking* para unir os planos do mundo real e do virtual. Normalmente são utilizadas marcações para determinar a posição onde os objetos do mundo virtual serão posicionados no mundo real. Um dos objetivos deste trabalho é determinar o ponto de *tracking* sem utilizar marcações, aproveitando as características únicas dos objetos atuando nesse meio.

A primeira etapa foi encontrar uma Região de Interesse (ROI), por onde serão extraídos os pontos de *tracking*. A ROI deste trabalho é determinada pela área de teclas do piano/teclado, sendo um ponto em comum entre pianos e teclados possuírem teclas brancas ao redor de teclas pretas.

Inicialmente tentou-se detectar a área das teclas de um teclado através de uma foto do teclado. Com o sucesso de detecção através de uma foto, partiu-se para a tentativa utilizando vídeo da câmera. Nas seções a seguir serão relatadas as tentativas de detecção bem como seus resultados observados.

5.1 DETECÇÃO DE TECLAS NA IMAGEM DE UM TECLADO

A primeira tentativa foi utilizar uma foto de um teclado Casio CTK-496 e detectar a área das teclas na imagem. A opção factível foi através da utilização do Python 3.9.1 [1] e do pacote do OpenCV 4.5.1 para Python.

Após carregar a foto do teclado com o OpenCV, foi preciso estudar uma maneira de criar traços na área das teclas que pudessem ser identificados com contornos. Esses traços precisavam separar a área das teclas do restante do teclado além de permitir a identificação de cada tecla, garantindo assim que todas as teclas do teclado estejam dentro da área detectada. Para isso, foi convertida a imagem em escala de cinza e testado um *threshold* binário com parâmetro 80 de *thresh*, de modo a segmentar as partes claras e escuras do teclado. Aplicando contornos sobre o *threshold* binário, as bordas do teclado ficavam grudadas nas teclas brancas, devido a coloração ser muito parecida e a incidência de sombras na imagem tornarem as teclas brancas um pouco mais escuras.

[1] https://www.python.org/

Haviam muitas possibilidades para combinar os parâmetros e obter um bom resultado de segmentação das bordas das teclas brancas para esta abordagem. Levando isso em consideração, foi feito um algoritmo de *adaptiveThreshold* interativo, com objetivo de identificar os melhores parâmetros combinados. O funcionamento do algoritmo ocorre da seguinte forma:

- Redimensionar imagem original antes de aplicar os filtros;
- Aplicar *adaptiveThreshold*;
- Aplicar morfologia, para separar partes grudadas da imagem e reduzir ruídos;
- Redimensionar imagem para tamanho original;

Para as combinações, os parâmetros utilizados foram: porcentagem de redimensionamento da imagem em 25, 50, 75 e 100 porcento; tamanho do bloco de pixels da vizinhança usando para calcular o valor do *threshold*, ou *block_size*, nos valores 3, 9, 11, 23, 31; a constante de subtração da média usada no cálculo do *threshold*, chamada de *sub_mean*, nos valores 2, 6, 12, 24 e 30; tamanho do Kernel usado na morfologia, sendo 3x3, 6x6, 9x9 ou 12x12; tipo de morfologia aplicada, sendo *cv2.MORPH_OPEN* ou *cv2.MORPH_CLOSE*.

Combinando todos os parâmetros, foram realizados 3200 testes. Para cada teste foi atribuído manualmente uma nota de 0 a 5, indicando do menos útil ao mais útil respectivamente, sendo o foco principal obter uma forma de separar teclas brancas e pretas do restante do teclado. Os parâmetros testados foram escolhidos devido à variação de resultados durante as combinações, entretanto poderia ter sido feito com outros parâmetros e utilizando um algoritmo diferente e mais otimizado. Analisando as notas de todos os testes, não foi possível identificar um padrão de combinação que retorne um resultado melhor. Mas foi possível perceber que, na maioria dos testes onde os parâmetros *block_size* e *sub_mean* estavam com valores mais altos, o resultado foi uma tela branca com pequenos riscos pretos, sendo considerado falho com avaliação 0.

Dentre todas as combinações obtidas, foram selecionados os *thresholds* visualmente mais úteis para seguir com a detecção, normalmente com nota 4 ou 5. Para a área das teclas brancas, o *threshold* escolhido redimensiona a imagem em 50% e como parâmetros do *adaptiveThreshold* utiliza *block_size*=31 e *sub_mean*=6, após aplica morfologia *MORPH_OPEN* com kernel 12x12, o resultado está demonstrado na Figura 13.

Para calcular os contornos foi utilizado *findContour*, em modo *RETR_TREE* para recuperar todos os contornos e reconstruir uma hierarquia dos contornos aninhados [2], e a aproximação *CHAIN_APPROX_TC89_L1* da cadeia Teh-Chin (TEH; CHIN, 1989). Ao observar todos os contornos detectados na imagem, não foi possível encontrar nenhum contorno que envolvesse toda área das teclas, devido a primeira e última tecla branca do teclado ficarem unidas à borda do teclado. Para contornar o problema, foi utilizado o algoritmo detector de bordas *Sobel* [3].

[2] Disponível em: https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html
[3] Disponível em: https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html

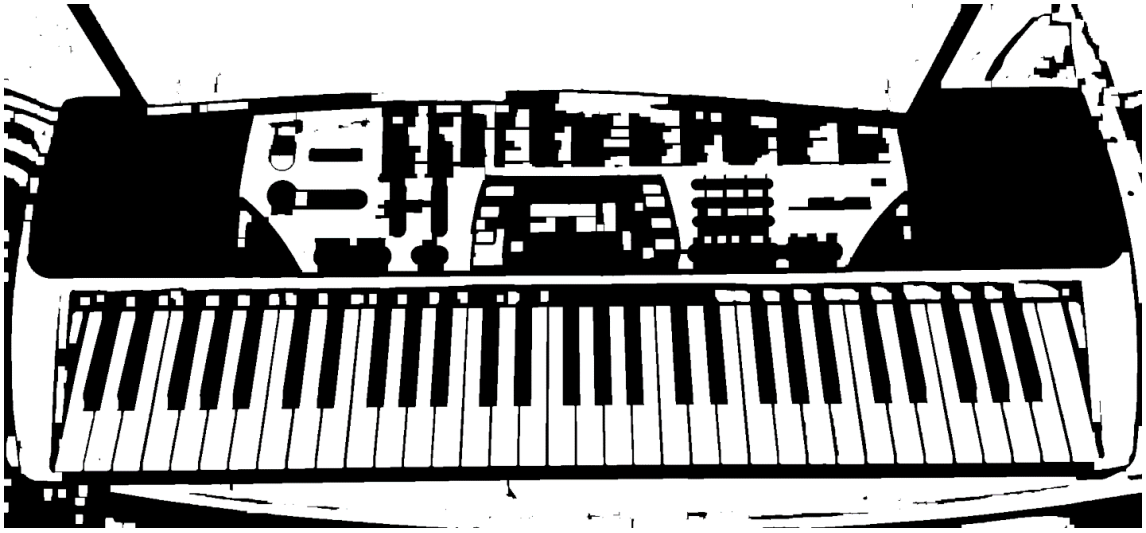


Figura 13 – Threshold selecionado para separar área das teclas brancas.

O operador Sobel calcula uma aproximação do gradiente da imagem nos eixos X e Y, ou seja, uma aproximação da derivada dos eixos. Para fazer o cálculo o operador utiliza um kernel ímpar, por exemplo de tamanho 3. Para calcular a derivada horizontal, no eixo X, o cálculo é feito de acordo com a equação:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * Imagem$$

Para calcular a derivada vertical, no eixo Y, o cálculo se da por:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * Imagem$$

Após, cada ponto da imagem calcula uma aproximação do gradiente combinando os resultados anteriores, na forma: $G = \sqrt{G_x^2 + G_y^2}$.

De acordo com a documentação, um kernel de tamanho 3 pode apresentar imprecisões visíveis. Optou-se, então, por um kernel de tamanho 21, o que resulta em uma boa aproximação da imagem. Como o objetivo era identificar os principais contornos da imagem, sem a necessidade de detectar todas as arestas da imagem, não foi executado o cálculo da aproximação do gradiente, mas sim um somatório da derivada no eixo X e Y. Como resultado, apenas as arestas principais foram selecionadas da imagem com threshold aplicado, e assim a função *findContours* foi capaz de identificar contornos englobando a área total das teclas.

Para exibir o contorno correto em meio aos outros contornos produzidos pela imagem, foi calculado um percentual de área entre o frame total da imagem e o contorno. Na prática, a área do contorno e do frame foi dada pela função *contourArea*, em seguida foi calculado a porcentagem que a área do contorno representa do frame total, dado por:

$$percArea = areaContorno * 100/areaFrame.$$

Apenas os contornos que representam mais 8% da área do frame foram exibidos, resul-

tando em apenas o contorno da área das teclas, como ilustra a Figura 14.

Figura 14 – Área de teclas detectadas demonstradas por um contorno em vermelho.

A próxima etapa foi utilizar a área de teclas como região de interesse (ROI), fazendo com que a detecção de teclas pretas atue apenas na área delimitada pelo ROI da etapa anterior. Como as teclas pretas são partes facilmente identificáveis na imagem, bastava encontrar um *threshold* contendo as partes mais escuras do teclado. O *threshold* escolhido redimensiona a imagem em 25% e como parâmetros do *adaptiveThreshold* utiliza *block_size*=31 e *sub_mean*=6, após aplica morfologia *MORPH_CLOSE* com kernel 6x6, o resultado está demonstrado na Figura 15.

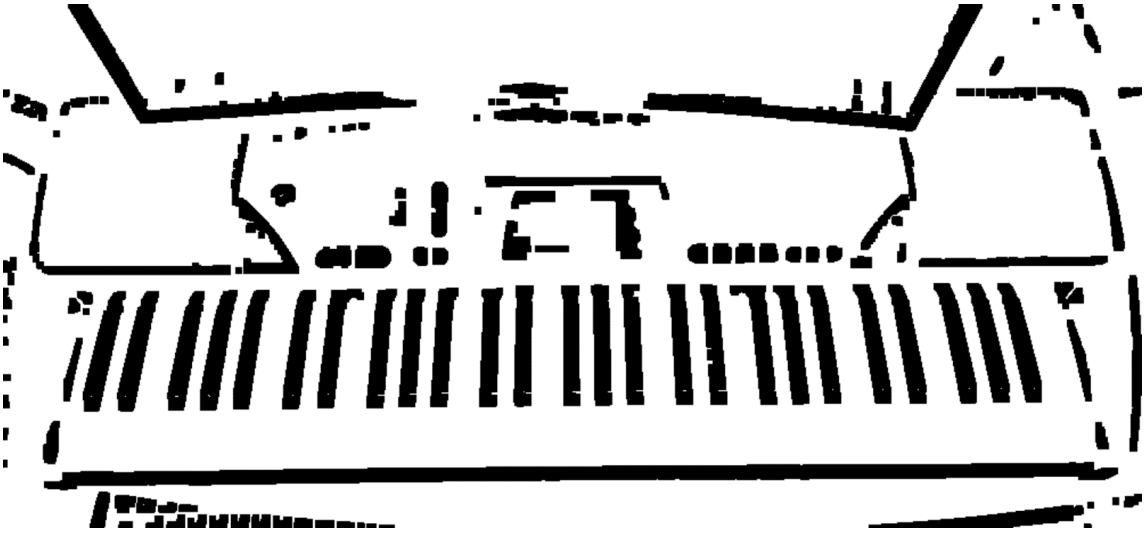


Figura 15 – Threshold selecionado para identificar as teclas pretas.

Foi aplicado um filtro negativo no *threshold* com *bitwise_not*, e assim como no anterior foi feito extração de arestas. Desta vez foi utilizado o operador *Canny* [4]. Este operador utiliza o mesmo cálculo de gradiente de aproximação por derivada do *Sobel*, e de modo incremental

[4] Disponível em: https://docs.opencv.org/master/da/d22/tutorial_py_canny.html

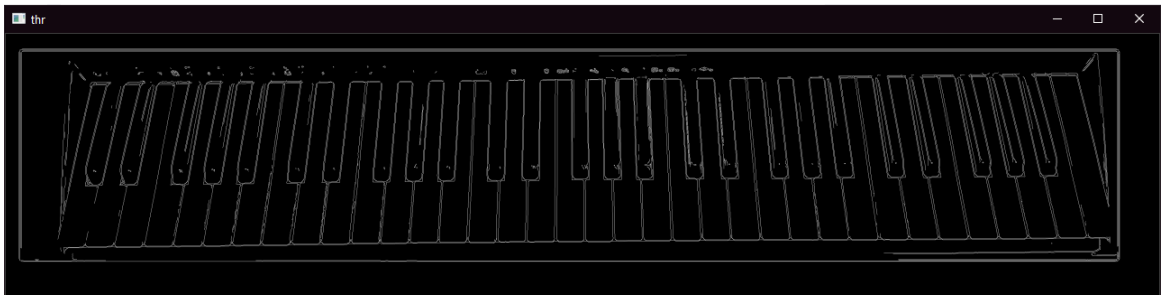
calcula o ângulo da direção do gradiente de forma: $Angle(\theta) = \tan^{-} 1(\frac{G_y}{G_x})$. Utilizando o grandiente e o ângulo, é removido os pixels que não são pontos máximos das bordas, o chamado *Non-max Suppression*. Por fim, aplica-se a etapa de identificação dos pixels fracos e fortes, utilizando *Double Threshold*, sendo máximo e mínimo, e assim aplicando o processo de *Hysteresis*, onde pixels com valor de gradiente acima do *threshold* máximo são considerados arestas, e os valores de gradiente abaixo do *threshold* mínimo são descartados. Na prática, o *Canny* serviu para reduzir alguns ruídos da imagem, realçando os contornos realmente importantes para a etapa *findContours*. O resultado é demonstrado na Figura 16.

Figura 16 – Contornos das teclas pretas identificados por Canny.

Os contornos das teclas pretas não pareciam ter formato de teclas. Haviam sombras nas partes superiores das teclas pretas que faziam com que a silhueta de tecla ficasse deformada. Como tentativa de solucionar o problema, foi descartado o uso do *adaptiveThresold*, e foi aplicado um filtro de sombras na imagem em escala de cinza. Uma função removeSombras foi criada para fazer a redução de sombras e normalização, o funcionamento está descrito em tópicos:

- Aplicar *dilate* com kernel 7x7;
- Aplicar e guardar *medianBlur*, com parâmetro ksize=21, para subtrair posteriormente;
- Subtrair da cor branca (255) o valor guardado do *medianBlur*;
- Aplicar *normalize* na imagem subtraída, com alpha=0 e beta=255;

Também foi utilizado *Canny* sobre a imagem normalizada para realçar as arestas e reduzir pequenos ruídos. Como resultado, os pequenos espaços escuros entre as teclas brancas também foram destacados, resultando em contornos bem definidos das teclas brancas e pretas. A Figura 17 ilustra em (a) o resultado do *Canny* e em (b) os contornos identificados.



(a) Arestas extraídas por Canny na ROI.

(b) Contornos encontrados na ROI, após Canny.

Figura 17 – Visualização das arestas reconhecidas por Canny e os contornos encontrados na ROI, durante a etapa de detecção das teclas pretas.

Com isso, o resultado da detecção de teclas utilizando uma imagem foi satisfatório. Na seção a seguir será descrito como foi a tentativa de aplicar os mesmos passos fazendo a leitura de vídeo da câmera de um smartphone. Os frames de vídeo não possuem a mesma qualidade focal de um foto. Consequentemente, apareceram ruídos significativos nos *thresholds*, dificultando na separação das teclas.

5.2 DETECÇÃO DE TECLAS DE UM TECLADO USANDO VÍDEO DA CÂMERA

5.2.1 Configuração do ambiente no Unity

Para a detecção com vídeo da câmera, foi escolhido o Unity como plataforma e também como Engine para futura produção do jogo. Utilizando a versão 2018.3.0f2 do Unity com módulo para Android, foi compilado a biblioteca OpenCVforUnity versão 2.4.2 da EnoxSoftware [5] com a flag *OPENCV_ENABLE_NONFREE* desabilitada (para utilizar os recursos pagos deve-se

[5] https://github.com/EnoxSoftware/OpenCVForUnity

marcar esta flag como habilitada). Como um dos princípios do projeto é custos reduzidos, o foco será as bibliotecas gratuitas do OpenCV.

Na cena principal do Unity, foi mantido a Main Camera e a Directional Light instanciadas por padrão. Foi adicionado um *Raw Image* em GameObject > UI > Raw Image. Um objeto *Canvas* é criado com a *Raw Image* na sua hierarquia, nas propriedades do *Canvas*, foi alterado o componente *Canvas Scaler*, parametrizado a resolução X=1280 e Y=800, também o *Screen Match Mode* configurado para *Match Width or Height*. Na *Raw Image*, foi adicionado o componente *Aspect Ratio Fitter*, com *Aspect Mode = Envelope Parent*, com isso o *Aspect Ratio* (ou AR, define a proporção entre a largura e altura) deve ser calculado automaticamente de acordo com o *Canvas*. Por fim, foi criado um script em C# vinculado à *Main Camera* onde será aplicado a lógica e programação.

5.2.2 Aplicação do modelo de deteção de imagem em vídeo

Seguindo o modelo da detecção por imagem, foi convertido o frame em escala de cinza e aplicado a mesma função de *adaptiveThreshold* iterativo descrita na seção anterior, para detecção da área das teclas brancas. O resultado é completamente diferente do encontrado na detecção por imagem o *threshold* resultante estava tão fragmentado que nenhum contorno podia ser identificado, como demonstra a Figura 18, em comparação com a detecção de uma foto, por exemplo, ilustrada na Figura 13.

Figura 18 – Resultado do *adaptiveThreshold* para teclas brancas aplicado sobre o frame de vídeo.

Além da dificuldade de lidar com sombras causadas pela iluminação, quando se trabalha com vídeo no OpenCV, é preciso lidar com ruídos na imagem, pois no frame de vídeo não ocorre as diversas otimizações que a câmera em modo fotografia consegue usufruir. Adicionalmente objetos em movimentos deixam rastros na imagem. Todos esses fatores, entre outros, provocaram o problema com o *threshold*, inviabilizando o modelo logo na primeira etapa de detecção.

Como a aplicação de reconhecimento de teclas foi projetada desde o início para trabalhar com vídeo, deveria ter sido optado por imediantamente começar os testes de detecção utilizando vídeo ao invés de testar com um foto do teclado. Como aprendizado para futuros projetos, será levado em consideração a utilização do modelo correto para cada abordagem, de modo a não provocar atrasos no projeto devido à falha do modelo.

5.2.3 Novo modelo de detecção de teclas do teclado por vídeo

Ao observar vários modelos de pianos e teclados, é possível perceber que comumente todos possuem teclas pretas envolvidas por teclas brancas ao redor. Partindo desse princípio, foi feito um novo modelo de detecção cujo foco é identificar partes pretas em meio a partes brancas no frame de vídeo do teclado.

Para reduzir os ruídos do vídeo, foi aplicado *GaussianBlur* com kernel 9x9 e três interações no frame em escala de cinza, em seguida aplicado um *threshold* binário invertido, com parâmetro *thresh*=80, subtraindo a parte branca e cinza da imagem, sobrando apenas as partes mais escuras. Por fim foi utilizado a morfologia *erode* para eliminar pontos pequenos na imagem o resultado é exibido na Figura 19. Com isso as teclas pretas são destacadas, sendo possível delimitar os contornos na próxima etapa.

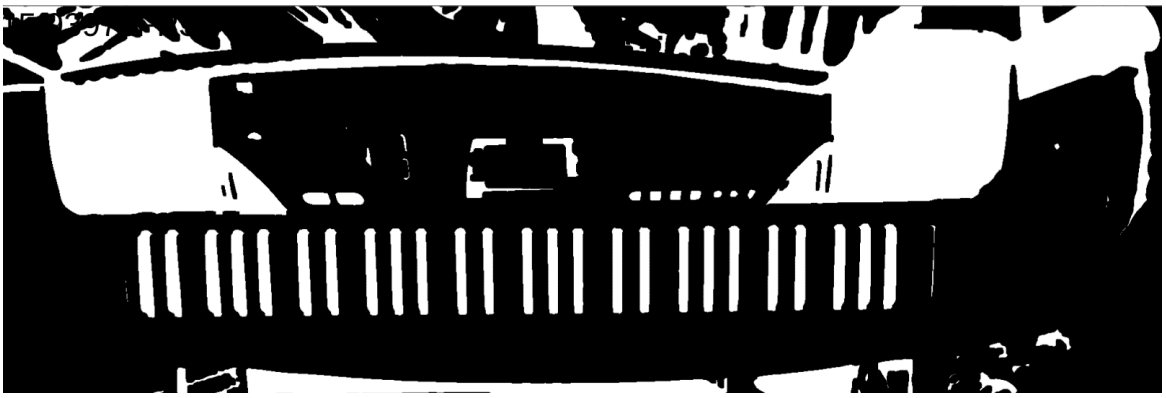


Figura 19 – Threshold binário invertido aplicado sobre o frame de vídeo.

Foi aplicado detecção de contornos, utilizando *findContour* em modo *RETR_TREE* e aproximação *CHAIN_APPROX_TC89_L1* , configuração de contornos aplicada em todos os casos de testes deste trabalho. Alguns contornos eram muito pequenos para serem identificados, então foi feito o cálculo de porcentagem relativa de área para removê-los, utilizando o mesmo cálculo do modelo anterior: $percArea = areaContorno * 100/areaFrame$. Todos os contornos com menos de 0,05% da área do frame original foram ignorados, eliminando assim pequenos detalhes na imagem que são desnecessários.

Para selecionar os contornos referentes às teclas pretas, estudou-se uma abordagem com *Aspect Ratio* (AR), ou seja, calcular a proporção entre a altura e a largura da tecla, sendo: $AR = altura/largura$. As medidas da tecla preta do teclado Casio CTK-496 são: 8,5cm de largura e 1,4cm para base maior e 1,0cm para a base menor da tecla, em formato de prisma

trapezoidal. Fazendo o cálculo, obteve-se AR=6,07 considerando a base maior da tecla e, AR=8,05 considerando a base menor.

Transformando os contornos em *RotatedRect* com a função *minAreaRect* do OpenCV, foi aplicado o cálculo AR para todos os contornos. Foram selecionados todos os contornos cujo valor do *aspect ratio* estivesse no intervalo entre 3 e 12, pois conforme o ângulo da câmera podiam haver variação no AR para cada tecla, causado pela perpectiva da câmera. Nesta etapa, todos os contornos referentes às teclas pretas foram identificados, no entanto algumas sombras e botões do teclado também foram selecionados pelo intervalo de AR assim, foi necessário mais uma etapa para selecionar apenas os contornos desejados em uma região do frame.

A solução para o problema acima foi ignorar uma parte do frame da câmera durante a detecção. Geralmente os leitores de código de barras possuem uma área em linha que ignora a parte superior e inferior da imagem, reconhecendo apenas um retângulo ao centro da imagem. Utilizando a mesma ideia de um leitor de código de barras, foi criada uma região de interesse (ROI) considerando o *aspect ratio* da região de teclas do teclado Casio CTK-496. A região de teclas brancas que envolve também as teclas pretas possui: 84cm de largura e 14cm de altura. Foi calculado o AR de largura para a região das teclas: $AR_L = largura/altura$, resultando em AR=6. No frame de vídeo foi criado um retângulo de largura igual à largura total do frame e altura: $altura = larguraFrame/AR_L$, assim ignorando a parte do frame superior a altura do retângulo e a parte do frame inferior a origem do retângulo. Levando em consideração que o OpenCV renderiza o frame do ponto superior esquerdo para o ponto inferior direito. Após isso, o resultado é uma ROI que, ao posicionar as teclas do teclado dentro desse retângulo de detecção, apenas as teclas pretas são reconhecidas, como demonstrado na Figura 20.

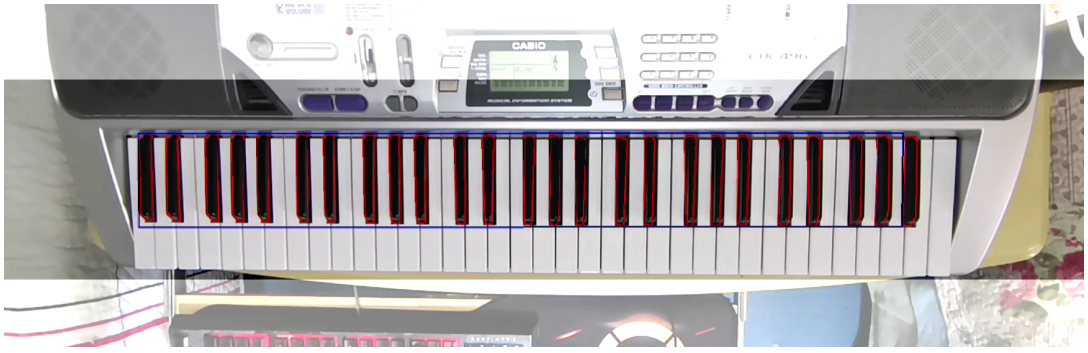


Figura 20 – Área de teclas pretas detectadas contornadas em azul e teclas pretas contornadas em vermelho.

Para desenhar um retângulo sobre a área das teclas pretas, gravou-se os pontos máximos e mínimos de cada contorno da tecla detectada em seguida foi criado um retângulo, usando os 4 pontos gravados, ilustrado também pela Figura 20.

Para identificar se todas as teclas estão detectadas e presentes no frame de vídeo, foi criado um parâmetro de **Quantidade de Teclas**. No teclado Casio CTK-496, existem 61 teclas, sendo 25 pretas e 36 brancas. No parâmetro de quantidade de teclas foi configurado 61 para o teste, e através de um cálculo de proporção de teclas pretas por quantidade total de teclas:

$qtdPretas = qtdKeys * 5/12$, foi obtido valor 25. Então quando a quantidade de teclas detectadas fosse igual ao valor obtido pela equação acima, a *flag* **detected** booleana recebe valor True, útil na próxima etapa de *tracking*.

Para encontrar a região das teclas brancas, poderia ter sido feito uma ampliação da região das teclas pretas, uma vez que sabendo a posição das teclas pretas também é possível saber a posição das teclas brancas. No escopo do presente trabalho, a área de teclas pretas foi considerada suficiente.

5.3 MÉTRICAS DE DESEMPENHO

Para medir o desempenho da aplicação, utilizou-se a taxa média de quadros por segundo (FPS), testando os resultados no player do Unity e no aplicativo para Android. O computador utilizado para rodar o Unity possui processador i7 2600K Quad-core, 16GB de memória RAM DDR3, placa de vídeo NVIDIA GeForce GTX 770 com 2GB de memória de vídeo GDDR5. O smartphone onde serão realizados os testes é um Huawei nova 5T com processador HUAWEI Kirin 980 Octa-core, 8GB de memória RAM, processador gráfico Mali-G76 720 MHz. A câmera do smartphone também é utilizada nos testes do computador, sendo conectada através do aplicativo Irium Webcam [6], a câmera grava vídeos com resolução 4K a 30fps, e em 1080p a 30/60fps.

Para calcular o FPS no Unity, foi inserido a seguinte função dentro do bloco Update():

```
frameCount++;
dt += Time.unscaledDeltaTime;
if (dt > 1.0 / updateRateSeconds)
{
    fps = frameCount / dt;
    frameCount = 0;
    dt -= 1.0F / updateRateSeconds;
}
```

Para cada frame processado pelo *Update()* do Unity, é incremetado o contador de frames *frameCount* e o contador de tempo *dt*, que recebe o tempo entre o último frame e o frame atual através do método *unscaledDeltaTime*. No bloco *Start()* do Unity, foi configurado a constante *updateRateSeconds* para 4 segundos, isso implica que quando o *dt* for maior que 1.0/4.0 = 0.25 segundos, será calculado o FPS dividindo o total de frames contados até o momento pelo tempo em segundos que passou, e em seguida reiniciado os contadores *frameCount* e *dt*. O FPS selecionado em cada teste é o valor mais frequente retornado pela função.

Testando a aplicação de detecção de teclas do teclado através de vídeo, pelo Unity player foram feitos dois testes com resoluções diferentes. O primeiro em uma resolução 960x540,

[6] https://iriun.com/

obtendo 77,14 FPS, no entanto teve-se um pouco de dificuldade para detectar corretamente as teclas do teclado, pois quanto menor a resolução, maior a dificuldade de separar as sombras da imagem. O segundo teste foi executado com resolução 1920x1080 (Full-HD), obteve 17,92 FPS, detectando as teclas sem maiores problemas.

Compilando o aplicativo para Android e executando no smartphone, a resolução da câmera padrão é 2336x1080, foi mantido nessa resolução pois o objetivo não é comparar smartphone com computador, mas sim o desempenho da aplicação de acordo com o modelo de detecção. Foi obtido 18,35 FPS no smartphone.

5.4 TRACKING DA ÁREA DETECTADA

Fazer detecção de imagens e frames é custoso no quesito processamento, e isso leva a aplicação a apresentar baixa taxa de quadros por segundo durante o funcionamento. Os valores encontrados abaixo de 20 FPS para resolução Full-HD, criam atrasos perceptíveis entre a captura e exibição do frame de vídeo, sendo desconfortável para visualização e um resultado não satisfatório para este trabalho. Porém, pode-se utilizar técnicas de *tracking* para contornar o problema. Aproveitando a área de teclas detectada no processo descrito anteriormente, é possível transformá-la em um recurso adaptativo, que não necessita ser novamente processada com *tresholds* e contornos, apenas recalculando sua posição de acordo com o movimento, ou seja, utilizando a área detectada como ponto de *tracking*.

5.4.1 CamShift

A primeira tentativa foi utilizar o CamShift, uma variação melhorada do MeanShift [7]. Primeiro é calculado o histograma da região de interesse, no caso a área de teclas detectada. Em seguida calcula a *Back Propagation* dos pixels do frame, por onde se encontra o ponto de maior densidade de pixels indicando qual será a posição de movimento que o objeto irá receber.

Quando a área de teclas é encontrada na primeira etapa da detecção, é setado a *flag* de nome *detected* para *True*. Com isso foi colocado uma condicional que ao verificar o valor verdadeiro para *detected*, a aplicação ignora os passos de detecção e entra no bloco de código do CamShift.

Fazendo o teste pelo Unity player, com resolução 960x540 foi obtido 110,11 FPS. Com resolução 1920x1080 (Full-HD), obteve 23,17 FPS. No smartphone, em resolução 2336x1080 obteve-se 25,67 FPS.

Apesar do ganho de performance da aplicação, o problema do método CamShift é a aplicabilidade do mesmo. O CamShift consegue identificar traços de movimentos que ocorrem na variação de algum ponto do frame, mas isso requer que apenas esse ponto seja alterado de um frame para outro para que seja possível fazer o *tracking*. Neste projeto, a câmera se move

[7] https://docs.opencv.org/master/d7/d00/tutorial_meanshift.html

livremente com os objetos no cenário estáticos, fazendo com que todos, ou pelo menos a grande maioria dos pontos entre um frame e outro sejam diferentes e, por consequência, o CamShift não consegue identificar o movimento corretamente.

5.4.2 Features Detector ORB

Devido a mobilidade da câmera, foi pensado em uma solução para considerar a movimentação da câmera e realizar o *tracking* compensando a perspectiva da câmera para o plano tridimensional dos objetos. Para isso, foi desenvolvido um modelo de detecção de *features*, extraindo os pontos mais importantes da área detectada e realizando um *match* com os pontos mais importantes do frame de vídeo. Nesse contexto, o algoritmo ORB (RUBLEE et al., 2011) foi escolhido para uma tentativa de *tracking*.

O ORB utiliza o método FAST (ROSTEN; DRUMMOND, 2006) para encontrar pontos chaves na imagem, em seguida aplica o método de Harris para detecção de cantos na imagem [8]. Utilizando uma pirâmide de *features* multiescalares, o ORB calcula para cada canto, a orientação em vetor de intensidade, para na etapa seguinte utilizar as orientações e vetores com o descritor BRIEF (CALONDER et al., 2010). No construtor do método *ORB()*[9], há varios parâmetros que podem ser modificados. Um parâmetro importante é o *nFeatures* que indica o número máximo de *features* que podem ser encontrados, cujo valor padrão é 500.

Como um teste inicial, foi configurado o ORB para detectar as *features* apenas do frame de vídeo, utilizando o método *detectAndCompute()* para calcular os pontos mais importantes da imagem e *drawKeypoints()* para exibir na tela os pontos detectados. Feito um teste pelo Unity player, usando parâmetro *nFeatures*=500 no construtor do *ORB()*, com resolução 960x540 foi obtido 25,53 FPS e com resolução 1920x1080 (Full-HD) obteve-se 5,71 FPS. Executando pelo smartphone em resolução 2336x1080 obteve-se 5,17 FPS. Na Figura 21 pode ser visualizado as *features* detectadas.

Utilizar o recurso ORB tende a ser muito custoso para ser calculado em grandes resoluções de tela. Para um caso de uso real a imagem computada deve ser calculada na menor resolução possível, para assim melhorar o desempenho da aplicação como um todo. Com isso em mente foi redimensionado o frame de entrada para 640x480, utilizando o método *resize* do OpenCV, e aplicado o método *detectAndCompute()* da mesma forma que foi testado logo acima, e ao final o frame é redimensionado novamente para a resolução de entrada. Na prática foi testado pelo Unity player com a resolução de entrada 1920x1080 (Full-HD), obtendo assim 10,07 FPS. Com resolução de entrada 960x540 obteve-se 30,12 FPS, um ganho considerável porém não atingiu o esperado, que seria mesmo com resolução de entrada em Full-HD, conseguir atingir uma taxa superior a 20 quadros por segundo. Testando no smartphone, a uma resolução de entrada de 2336x1080, foi obtido 12,52 FPS.

[8] https://docs.opencv.org/3.4/d4/d7d/tutorial_harris_detector.html
[9] https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html

Figura 21 – Visualização das *features* identificadas por *ORB(500)*.

Para comparar com as *features* encontradas no frame, deve-se utilizar um *target*, ou uma imagem de alvo para fazer as validações. No momento que a variável booleana *detected* for verdadeira, foi gravado uma imagem da área de teclas detectadas e dado o nome de **featureImage**, ilustrado pela Figura 22.

Figura 22 – Imagem da área de teclas pretas detectada chamada de featureImage.

Aplicando o método *detectAndCompute()* na imagem, para detectar os pontos e computar os descritores, o ORB não foi capaz de identificar boas *features* para detecção, na maioria das vezes não detectou nenhuma *feature*. Na tentativa de conseguir capturar bons pontos para prosseguir com a detecção, foi feita uma amplicação da área de teclas pretas detectadas, aumentando o contorno para abranger as teclas brancas.

Para calcular a ampliação da área de teclas detectadas, foi feito um processo em duas etapas. A primeira etapa consistia em aumentar a altura das teclas, uma vez que as teclas pretas e brancas possuem origem no mesmo ponto e alturas diferentes. No teclado Casio CTK-496, as teclas pretas possuem 9 cm de altura, enquanto as brancas 14 cm, em uma regra de três simples é descoberto que a altura das teclas deve aumentar em aproximadamente 55,556%. Para calcular a altura do contorno, basta obter o maior ponto Y e subtrair pelo menor ponto Y. A segunda etapa, aumentar a largura da área de teclas, é um pouco mais complicada.

Conforme as medidas de John J. G. Savard (2020), o espaçamento de teclas pretas e brancas ocorre em proporções diferentes, medindo as teclas do teclado Casio CTK-496 a tecla branca possui 2 cm de largura, e a tecla preta ocupa 0,8 cm do espaço da largura da tecla branca. Em uma fração de tecla branca de 5 partes, a tecla preta ocupa $\frac{2}{5}$ do espaço da tecla branca. Entre duas oitavas, o espaço entre as teclas pretas é de 2,4 cm ($\frac{6}{5}$ de tecla branca). Sabendo essas

proporções e sabendo também que na direita do teclado não existe a tecla preta ocupando espaço na tecla branca, e na esquerda do teclado a primeira tecla branca possui $\frac{2}{5}$ do espaço ocupado por uma tecla preta, é possível ampliar a área proporcionalmente a quanto de tecla branca está ao redor das teclas pretas.

Inicialmente obtêm-se a largura total da área de teclas pretas, subtraindo o maior ponto X do menor ponto X do contorno. Em seguida utiliza-se o parâmetro de quantidade de teclas para subtrair $\frac{11}{5}$ de teclas brancas, correspondente ao espaço da primeira tecla branca e as duas últimas teclas brancas do teclado, obtendo assim quantas teclas brancas representam a largura da área de teclas pretas. Por regra de três determinar a largura da área de teclas brancas, sendo o parâmetro de quantidade de teclas seu valor correspondente, utilizando a largura da área de teclas pretas e a quantidade de teclas brancas que representam essa largura. Obtendo a largura da área de teclas brancas, calcula-se a procentagem de aumento com relação à largura da área de teclas pretas, incrementando assim os dois lados do teclado, ponderando $\frac{3}{5}$ do aumento no lado esquerdo, e $\frac{8}{5}$ no lado direito.

Extraindo as *features* da nova imagem ampliada da área de teclas, observa-se o êxito em extrair alguns pontos na imagem featureImage, como exposto na Figura 23.

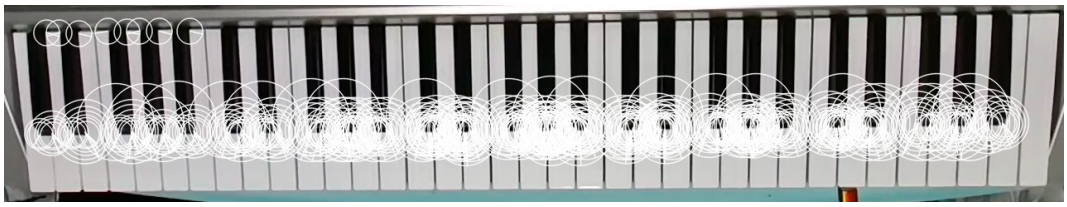


Figura 23 – *Features* da área de teclas ampliada.

O próxima passo é fazer o *match* entre a featureImage usada como *target*, ou objeto da detecção, e o frame de vídeo como cena de detecção. Para isso, foi utilizado o *BFMatcher*[10], que testa cada um dos descritores da featureImage com cada um dos descritores do frame de vídeo. Aplicando modo *BRUTEFORCE_HAMMINGLUT* no BFMatcher, vários pontos em comum entre o frame e a featureImage foram identificados, e podem ser visualizados no método *drawMatches()*, como mostrado na Figura 24.

Apesar de vários pontos em comum serem identificados, alguns pontos indesejáveis também foram identificados. Partes do frame fora da área das teclas foram identificadas como *matches* e, apesar da câmera estar capturando o mesmo frame que gerou a featureImage, várias linhas se cruzaram, ao invés de ficarem paralelas, algo que indica que os *features* do *target* não são bons para detecção.

Como tentativa de selecionar apenas as linhas retas e ignorar linhas cruzadas, foi aplicado uma seleção de *good matches*, ou seja, identificar os melhores *matches* que irão fazer parte da próxima etapa. Para isso foi calculado a distância máxima e mínima de cada *match*. Percorrendo um vetor com todos os *matches* foi selecionado todos os pontos onde a distância estiver próximos

[10] https://docs.opencv.org/master/d3/da1/classcv_1_1BFMatcher.html

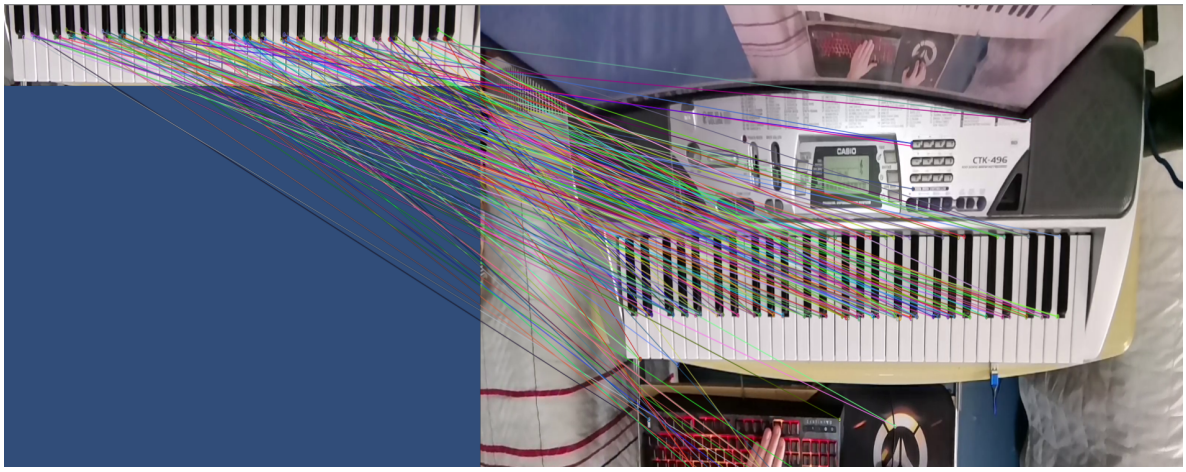


Figura 24 – *Matches* entre featureImage e o frame de vídeo, encontrado com BFMatcher.

da distância média dos pontos de *match*. O resultado reduziu vários cruzamentos, mas ainda existiam linhas cruzadas e *matches* fora da área das teclas.

Mesmo sem ter encontrado os *matches* ideais, foi passado para próxima etapa. Nesta última etapa o objetivo é fazer a transformação da perspectiva entre o frame de vídeo e a featureImage. Utilizando o método *findHomography()* para encontrar as transformações entre os pontos de *match* do frame e da featureImage, e em seguida aplicando o método *perspectiveTransform()* para aplicar o mapeamento das transformações. Foi tentado desenhar linhas utilizando o resultado do mapeamento, mas as linhas não formaram nenhum contorno de objeto. Foi testando o *findHomography()* nos modos: *RANSAC*, *LMEADS* e *RHO*, porém nenhum teste conseguiu desenhar corretamente o mapeamento.

Testando o desempenho no Unity player, em resolução 1920x1080 (Full-HD), obteve-se 3,68 FPS, já em 960x540 obteve-se 10,21 FPS. No smartphone, a uma resolução de 2336x1080, verificou-se 5,74 FPS.

Devido ao baixo desempenho apresentado pelo método ORB, e por não ter sido possível identificar boas *features* para detecção, foi tentando uma abordagem diferente para descobrir se o teclado poderia oferecer bons recursos para aplicar *tracking*. Na seção seguinte será trabalhado com o Vuforia como uma alternativa para a detecção das teclas.

5.5 TRACKING COM OPENCV E VUFORIA

O *Software Development Kit* (SDK) Vuforia Engine [11] é uma plataforma para criação de realidade aumentada também suportada pelo Unity, podendo ser baixada pela gerenciador de pacotes interno do motor de jogos. A grande vantagem do Vuforia é que não há restrição de dispositivos para utilização de seus recursos, sendo limitado apenas pela capacidade do hardware.

[11] https://library.vuforia.com/

Para utilizar o Vuforia, foi criado um novo projeto com as bibliotecas atualizadas. Foi atualizado o Unity para versão 2021.1.3f1 e o pacote do OpenCV foi atualizado para a versão 2.4.4, também utilizando apenas os recursos gratuitos do OpenCV. Foi feito o download do Vuforia pela Asset Store do Unity, importando no projeto a versão 9.8.5 do Vuforia.

No novo projeto do Unity, foram criadas duas cenas. A primeira cena chamada **DetectionScene** foi configurada de acordo os passos descritos na seção 5.2.1, e o código de deteção descrito na seção 5.2.3. A segunda cena chamada de **ARScene**, teve sua *Main Camera* removida e adicionada a câmera de realidade aumenta do Vuforia: *ARCamera*. Também foi adicionado um *Image Target* e alterado o bloco *Image Target Behaviour* para o tipo *From Image*. O *Image Target* é responsável por adicionar os objetos no plano tridimensional, assim qualquer objeto vinculado neste componente irá aparecer no momento que a *ARCamera* detectar a imagem vinculado ao *Image Target Behaviour*.

Na *ARCamera*, foi criado um novo arquivo em C# para criar um *tracker* via linhas de código, que será utilizado após ocorrer a detecção da área das teclas. No momento que a cena DetectionScene reconhecer a área das teclas pretas do teclado, a *flag* booleana *detected* fica com valor verdadeiro, e quando isso ocorre é gravado a imagem da área detectada para posteriormente ser usada no *Image Target*, como visto na Figura 22, e a cena é trocada para ARScene.

Ao iniciar o Vuforia na cena ARScene, é criado um *tracker* com a imagem da área detectada salva anteriormente usando o recurso *RuntimeImageSource*. Em seguida foi criado um novo *DataSet* vinculando o *tracker* criado ao contexto de detecção do Vuforia. Por último foi criado um objeto para seguir o *tracker*, no caso um simples plano, vinculado ao *DefaultTrackableEventHandler* através do método *SetParent()*.

Em um primeiro teste, executando no Unity player, após detectar a área das teclas e abrir a ARScene, o Vuforia não conseguiu identificar no frame de vídeo a imagem vinculada ao *tracker*, o *Status* exibido pelo Vuforia permanecia *UNKNOW*. Acredita-se que dos possíveis fatores para isso ocorrer é a baixa resolução de imagem que o Vuforia renderiza, apenas 640x480 pixels. Como a imagem foi gravada utilizando um frame de vídeo em Full-HD, é provável que em baixa resolução não identificou os mesmo pontos. Para verificar esse pressuposto, a imagem antes de ser salva teve seu tamanho reduzido em 3 vezes ($1920/640 = 3$), e assim foi realizado o teste novamente. O resultado foi o mesmo, levando a entender que a imagem das teclas, ilustrada pela Figura 22, não possui boas *features* para realizar o *tracking* com o Vuforia.

Observando o teclado Casio CTK-496, há uma tela com vários pontos únicos na imagem como mostra a Figura 25, que poderiam servir para realizar o *tracking* com o Vuforia.

Alterando a parte do código onde carrega a imagem da área de teclas detectada, e configurado para carregar uma foto da tela do teclado como mostra a Figura 25, foi testado no Unity player com resolução 640x480, e o Vuforia conseguir fazer o *tracking* da tela do teclado, atingindo uma taxa de quadros de 302,73 FPS, como exibido na Figura 26. Executando no smartphone, a resulução fica 720x540, devido à configuração do Vuforia estar no modo *MODE_OPTMIZE_SPEED*, alcançando uma taxa de quadros de 29,98 FPS. Alterando

Figura 25 – Tela do teclado Casio CTK-496.

a configuração do Vuforia para *MODE_OPTMIZE_QUALITY*, a resolução de tela muda para 1280x720, atingindo uma taxa de quadros de 28,31 FPS.

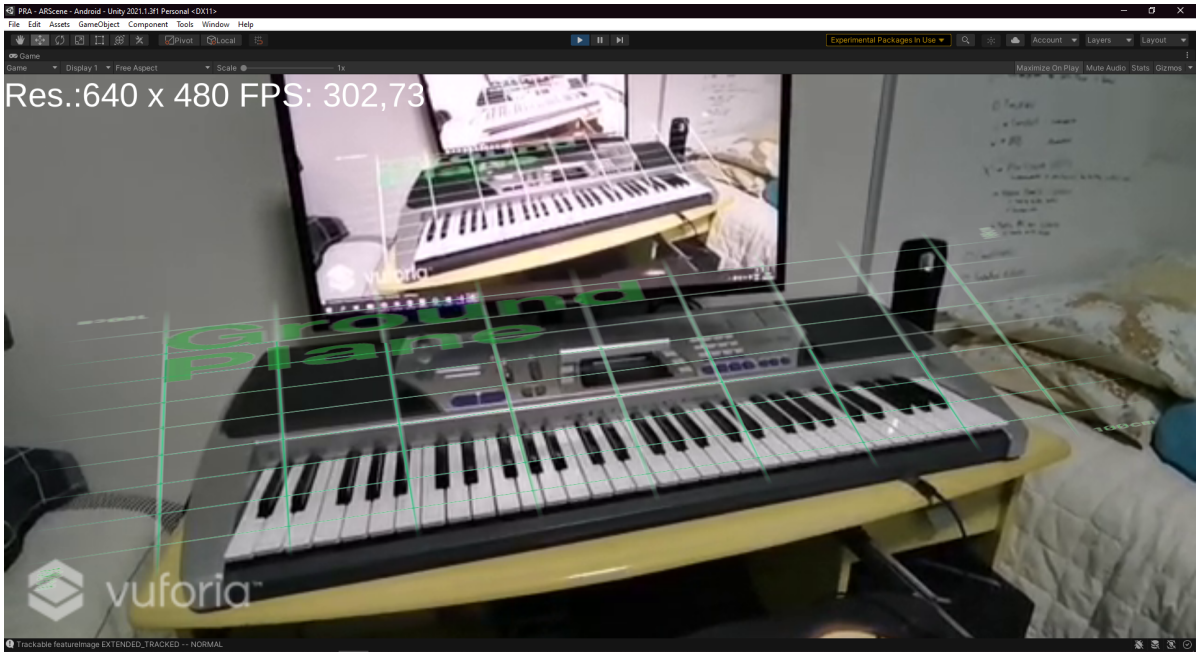


Figura 26 – Resultado da Realidade Aumentada utilizando Vuforia com tracking na tela do teclado Casio CTK-496.

Mesmo utilizando um foto salva, o modelo permaneceu sem uso de marcações para fazer a a deteção das teclas. Utilizando apenas o Vuforia foi possível realizar o *tracking* do teclado sem precisar da etapa de detecção com OpenCV. Futuramente poderia se pensar em um modelo de detecção substituindo o carregamento de uma foto salva, para uma foto tirada em tempo real durante o uso da aplicação, assim criando um ambiente de realidade aumentada sem a necessidade de utilizar marcações.

Foi feito uma tentativa de utilizar a tela do teclado Casio CTK-496 da Figura 25 com o modelo de deteção com ORB da seção anterior. Como resultado foi possível detectar e contornar a área da tela do teclado, aplicando transformação de perspectiva para fazer o *tracking*. Porém no modelo com ORB, quando a câmera está muito longe ou em um ângulo muito ruim para

detectar a tela do teclado, ou até mesmo se a tela do teclado sair do frame da câmera, o *tracking* é interrompido. O Vuforia possui um recurso de *Extended Tracking*, que quando os 3 casos citados acima acontecem, o Vuforia consegue extender a área detectada sem perder os pontos de *tracking*, o que torna o modelo muito mais interessante do ponto de vista prático do desenvolvimento da aplicação.

5.6 MARKERLESS AR EXAMPLE

A EnoxSoftaware, desenvolvedora do módulo OpenCVforUnity, fez um exemplo de RA utilizando um modelo sem marcações, chamado MarkerLess AR Example [12]. No modelo, inicialmente precisa ser definido um *target* para detecção, para isso utiliza um botão para tirar uma foto da região interessada. Em seguida a aplicação calcula os pontos de detecção e adiciona um objeto 3D fazendo o *tracking* do mesmo enquanto o *target* está identificável no frame da câmera. Foi feito um teste com a tela do teclado Casio CTK-496, conseguindo realizar o *tracking* de um objeto 3D acima do *target* com desempenho similar ao modelo com ORB, alcançando 27,39 FPS no Unity player em resolução 640x480. Ao tentar compilar para Android, o modelo trava e fecha o aplicativo sem exibir nada. O modelo poderia ser estudado e melhorado para conseguir melhores taxas de quadros por segundo e para conseguir persistir o *tracking* mesmo após o *target* deixar o frame da câmera.

5.7 JOGO DE RITMO MIDI

Ao conseguir obter um plano sobre a área das teclas e um *tracking* bem definido, o próximo passo é construir o jogo de ritmo com MIDI. É preciso utilizar um módulo de leitura de um arquivo MIDI, como a biblioteca *smflite* criado por Keijiro Takahashi [13], escrita em C# e permitindo sua importação no Unity. Para cada evento MIDI carregado pela biblioteca, a aplicação será responsável por criar a disposição das notas na tablatura virtual. É um biblioteca antiga, e é necessário tratar cada nota como um conjunto hexadecimal de dados, o que dificulta o processo. Neste projeto foi utilizado a biblioteca DryWetMIDI [14], também em C# e com muito mais recursos implementados prontos para uso.

Inicialmente foi criado um plano para representar a trilha por onde as notas irão percorrer até as teclas, a tablatura virtual. Em seguida foi criado cubos azuis para representar as notas de teclas brancas, e cubos verdes para representar as notas de teclas pretas. Estes cubos que representam as notas foram salvos como *Prefab* no Unity, permitindo que sejam instanciados em qualquer lugar da aplicação sem a necessidade de existirem inicialmente na cena. Os cubos possuem propriedade *RigidBody* sem gravidade, para permitir aplicação de física e movimentação na tablatura.

[12] https://github.com/EnoxSoftware/MarkerLessARExample
[13] https://github.com/keijiro/smflite
[14] https://github.com/melanchall/drywetmidi

Criando um script em C# e importando os módulos do DryWetMIDI, foi carregado cada evento MIDI existente em um arquivo de teste. Para cada evento, foi identificado qual a nota musical, sua duração e posição na execução da música, e assim poder transformar essas informações na tablatura onde as notas irão percorrer até as teclas.

Para definir qual a posição da nota em relação a tecla, foi identificado a oitava que a nota faz parte e qual o número da nota, sendo definido por um *enum* na seguinte ordem: 0 - C; 1 - C#; 2 - D; 3 - D#; 4 - E; 5 - F; 6 - F#; 7 - G; 8 - G#; 9 - A; 10 - A#; 11 - B.

O valor no *enum* foi chamado de **noteOffset**, assim somando à posição da primeira tecla o valor da oitava e o noteOffset da nota, considerando cada tecla com tamanho de 1 unidade, como mostra a equação: $-42.0f + ((note.Octave - 1) * 12 + (noteOffset + 1))$. Como cada oitava possui 12 teclas, ao multiplicar por 12 obtém-se a posição onde o início da oitava está. Depois foi calculado o tempo que cada nota deveria surgir na trilha, atribuindo a posição de cada uma das notas no eixo Y. Em seguida calculado a duração de cada nota através do atributo *Lenght* de cada nota. Por fim foi instanciado cada uma das notas vinculadas à trilha, e aplicado as transformações de posição, rotação e escala para obter a posição correta na trilha. Para as notas percorrerem a trilha foi atribuido o valor de *velocity* da nota para o parâmetro *velocity* do *RigidBody* de cada tecla. Por fim foi adicionado uma imagem em escala real das teclas de um piano para verificar se a posição das notas estavam corretas. O resultado é exibido na figura 27.

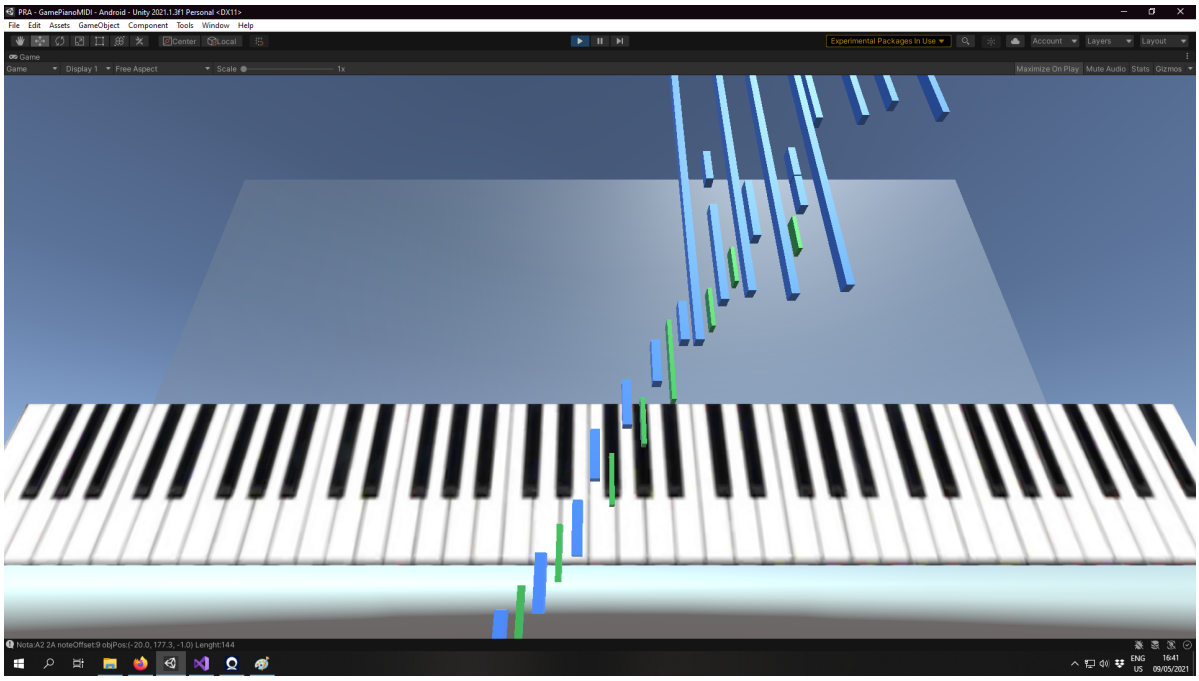


Figura 27 – Jogo de ritmo importando arquivo MIDI, criando uma trilha de notas em direção a cada tecla do piano.

Não foi possível definir com precisão o tempo, tamanho e velocidade de cada nota, mas a posição referente a cada tecla do piano ficou muito bem definida. Com mais algum tempo seria possível definir com precisão cada posição das notas e substituir o plano na cena de detecção pelo módulo do jogo de ritmo em MIDI.

6 RESULTADOS

Para cada etapa de detecção de teclas de um teclado musical, utilizando vídeo, foram feitos testes de desempenho a fim de obter um modelo que possa ser utilizado para desenvolver um jogo de realidade aumentada.

Nas Tabelas 1 e 2 foram anotados os valores de taxa de quadros por segundo mais frequentes dos testes realizados no Unity player e no smartphone, respectivamente. Algumas resoluções não são suportadas pelo Irium Webcam, aplicativo que transforma o smartphone em webcam para o computador, e por isso não há valores para essas resoluções na Tabela 1. O mesmo pode ser dito para o aplicativo Android executando direto no smartphone, onde o Vuforia somente oferece suporte para as resoluções 720x540 e 1280x960.

	Unity Player					
	640x480	**720x540**	**960x540**	**1280x960**	**1920x1080**	**2336x1080**
OpenCV Contornos teclas pretas	84,95 FPS	-	77,14 FPS	34,44 FPS	17,92 FPS	-
OpenCV CamShift	122,29 FPS	-	110,11 FPS	42,26 FPS	22,64 FPS	-
OpenCV ORB	33,31 FPS	-	25,53 FPS	11,35 FPS	5,71 FPS	-
Vuforia	302,73 FPS	-	-	-	-	-
MarkerLessARExample	27,39 FPS	-	-	-	-	-

Tabela 1 – Tabela de quadros por segundo de acordo com a resolução da tela no Unity player.

	Smartphone					
	640x480	**720x540**	**960x540**	**1280x960**	**1920x1080**	**2336x1080**
OpenCV Contornos teclas pretas	59,31 FPS	58,24 FPS	58,03 FPS	31,24 FPS	21,35 FPS	18,35 FPS
OpenCV CamShift	58,95 FPS	58,23 FPS	57,22 FPS	38,64 FPS	22,51 FPS	18,63 FPS
OpenCV ORB	26,43 FPS	24,34 FPS	19,73 FPS	9,89 FPS	6,23 FPS	5,17 FPS
Vuforia	-	29,98 FPS	-	28,31 FPS	-	-
MarkerLessARExample	-	-	-	-	-	-

Tabela 2 – Tabela de quadros por segundo de acordo com a resolução da tela no Smartphone.

Analisando o FPS de cada modelo para cada resolução, é possível concluir que as tecnologias atuais estão começando a apresentar bons resultados mas ainda não oferecem performance em resoluções mais altas como 1080p e Full-HD. Quando se trabalha com IHC como um jogo de RA, é desejável ao menos obter uma taxa de quadros por segundo que os olhos humanos consigam acompanhar sem ocorrência de atrasos. De acordo com Chen e Thropp (2007), o olho humano consegue diferenciar os quadros a uma taxa média de 15 FPS, lembrando que cada pessoa é diferente da outra, e acima desse valor o cérebro transforma automaticamente as imagens em movimento, fazendo com que a transição entre um frame e outro passe despercebida perante a visão humana. Na prática, os testes que rodaram abaixo de 15 FPS causaram atrasos no movimento, algo não recomendado para uma aplicação de RA pois provoca enjôos durante um uso prolongado da aplicação.

Os algoritmos de detecção de *features* funcionam bem na tarefa de detectar pontos importantes da imagem, mas são muito pesados e lentos, sendo necessário estudar mais a fundo e melhorar a forma de trabalhar com esse tipo de algoritmo.

A detecção das teclas de um teclado musical utilizando OpenCV foi concluída, porém não foi muito asseertiva. Apesar da identificação correta das teclas, e o contorno da área de teclas ficar bem definido, o modelo não conseguia detectar somente teclas, e acabava contornando botões do teclado e algumas sombras como se fossem teclas, precisando de uma ROI para capturar apenas uma parte do frame para conseguiur detectar com sucesso. Já o algoritmo ORB não conseguiu identificar *features* suficientes na área de teclas para diferenciar do restante do frame, e da mesma forma o Vuforia não identificou nenhum ponto para conseguir fazer o *tracking* da área do contorno das teclas. No trabalho de Silva e Reis (2020), também foi percebida a dificuldade dos algoritmos de detecção de *features* na definição de bons pontos de detecção, resultando em falha na comparação com o *target*.

Observando a baixa efetividade de continuar com a área de teclas do teclado como objeto de *tracking*, foi testado a captura de imagem da tela do teclado Casio CTK-496, exibida na Figura 25, como *target* do modelo de detecção. A tela consegue definir bons pontos no algoritmo ORB e no Vuforia, sendo a opção ideal para o teclado musical em questão. Para outros modelos de teclados e pianos seria interessante aplicar a mesma metodologia e identificar se as teclas conseguem definir bons pontos de detecção, sem a utilização de marcações. Caso o resultado de tais testes seja falho, poderia ser concluído que as teclas de um piano/teclado não são eficientes para detecção, e um modelo com marcações seria mais adequado ao trabalhar com esse tipo de instrumento musical.

O Unity se mostrou uma ferramenta muito eficaz e uma poderosa *Engine* para desenvolvimento de jogos. Existem bastante recursos desenvolvidos pela comunidade que podem ser incluídos como pacotes para desenvolvimento, o que tornou possível a realização deste trabalho. Um dos grandes problemas do Unity é uma série de bugs em cada versão lançada, como a maioria dos pacotes utilizados no trabalho não são desenvolvidos pela empresa responsável pelo Unity, ocorreram várias situações em que o programa parou de funcionar e fechou sem solução. É preciso destacar um bug de desempenho na versão 2018.3.0f2, na qual o Unity player mostrava uma taxa de quadros muito abaixo do esperado. Para contornar o problema era preciso executar o player maximizado e em seguida alterar para modo minimizado, assim a taxa de quadros era normalizada, possibilitando fazer as aferições de FPS para as métricas de desempenho da aplicação desenvolvida. Outro fator que tomou muito tempo do projeto foi a manipulação da *WebcamTexture* para fazer a leitura do frame da câmera, no momento da troca entre cenas do Unity: quando uma cena que estava utilizando a câmera trocava para outra, a câmera parava de funcionar no Unity, sendo necessário reiniciar o Unity inúmeras vezes para testes. Depois de um tempo significativo, encontrou-se uma solução simples e bastante óbvia, que era parar o uso da câmera antes de trocar as cenas. Isso reforça como problemas relativamente simples consomem um tempo considerável além do planejado para o desenvolvimento do projeto.

A biblioteca do OpenCV possui muitos recursos bons para trabalhar com detecção de imagens e vídeo, estando presente na grande maioria das linguagens de programação e com documentação muito bem escrita. O OpenCVforUnity da EnoxSoftware foi desenvolvido com

base na biblioteca em Java e foi escolhido por ter a maior parte dos recursos necessários para o projeto e por conseguir aproveitar a documentação em Java para desenvolver no OpenCVforUnity. Não foram encontrados grandes problemas para se trabalhar com a biblioteca. Havia duas documentações com versões diferentes do OpenCVforUnity, sendo necessário ficar atento à flag *OPENCV_ENABLE_NONFREE* que por padrão vem desabilitada, então a versão da documentação 2.4.4 era a correta para buscar as referências.

Ao trabalhar com o OpenCV, foi cometido um grande erro de projeto que tomou muito tempo de desenvolvimento. O objetivo deste trabalho foi de fazer a detecção de teclas usando vídeo da câmera, mas para fazer os testes iniciais foi utilizado uma foto do teclado onde se desejava detectar as teclas. Mesmo obtendo sucesso na detecção de uma foto, ao tentar detectar usando vídeo foi percebido que não funcionaria da mesma forma. No vídeo existem ruídos, rastros, sombras, borrões e vários outros fatores que são otimizados para uma fotografia. Por consequência, toda a detecção com imagens foi descartada e foi necessário iniciar o processo de detecção do ínicio para trabalhar com vídeo, resultando em atraso no projeto.

Outro recurso que merece destaque foi a utilização do Vuforia para automatizar o *tracking* e permitir o ambiente de RA no Unity. A plataforma Vuforia possui muitos recursos para trabalhar com realidade virtual, aumentada e mista, ainda funcionando em qualquer dispositivo, inclusive os dispositivos que o ARCore da Google não oferece suporte. O Vuforia oferece um recurso de *Extended Tracking* que produz um exelente resultado quando o *target* não está posicionado no frame de vídeo, mantendo o objeto da RA fixo no plano 3D e compensando as rotações da câmera. Apesar das qualificações, o Vuforia possui várias limitações quanto ao uso, como por exemplo a resolução da tela durante a visualização. Foi constatado que o Vuforia opera apenas na resolução 640x480 no Unity player, e compilado para Android no smartphone funciona nas resoluções 720x540 e 1280x960, além de ter a marca d'água sobre o frame de vídeo o tempo todo. Um dos grandes pontos fracos do Vuforia é a documentação mínima e vários artigos antigos estarem fora do ar na internet, tornando difícil a manipulação das bibliotecas dentro do Unity.

7 CONCLUSÃO

Neste trabalho foram apresentados modelos de detecção, sem utilização de marcações, para serem utilizados em um jogo de realidade aumentada que auxilia no aprendizado de piano. As etapas descritas neste trabalho possibilitaram a realização de um estudo sobre a detecção de teclas em um teclado musical, trazendo dados importantes sobre desempenho e falhas a serem evitadas ao realizar um projeto de deteção com imagens e vídeos. Devido à pandemia do Cornonavírus, o projeto precisou ser adaptado e reduzido para ser trabalhado em um menor espaço de tempo, tomando como foco principal a aplicação de algoritmos para detecção de teclas do teclado musical sem a utilização de marcações.

Todos os modelos que tentaram detectar as teclas do teclado não obtiveram um bom resultado. Utilizando apenas o modelo de detecção de contornos, não foi encontrada uma forma melhor de selecionar apenas o contorno das teclas em meio aos contornos de outros objetos na cena, sendo recortada uma parte do frame de modo que somente as teclas ficassem sobre uma ROI. No trabalho de Goodwin e Green (2013) a detecção de teclas também teve um resultado longe do ideal, onde a presença de sombras e baixa iluminação dificultavam na tarefa de detecção. Utilizando detecção de *features* as teclas não foram suficientes para realizar a detecção no teclado em questão (Casio CTK-496), não se pode afirmar que isso se aplica a todos os teclados, oferecendo uma oportunidade de pesquisa a ser feita para determinar se em outros modelos de teclados e pianos o resultado observado é validado. Desta forma, utilizar a tela do teclado foi uma alternativa de adaptação ao modelo de detecção possibilitando o uso da RA sem utilizar marcações, como era proposto neste projeto.

Por fim, conclui-se que foi possível determinar um modelo de detecção onde a taxa de quadros por segundo fique próxima de 30 FPS, desempenho desejado para este trabalho. O modelo com Vuforia utilizando a tela do teclado Casio CTK-496 foi capaz de adicionar um plano sobre a área de teclas e mantê-lo com precisão, considerado o modelo ideal para desenvolver o jogo de realidade aumentada. O jogo seria constituído pela tablatura virtual obtida no jogo de ritmo MIDI, no entanto não foi possível determinar corretamente a posição das nota musicais em tempo hábil para a conclusão deste projeto, deixando aberto uma lacuna para um trabalho futuro.

7.1 TRABALHOS FUTUROS

Para futuros projetos, há algumas propostas e tentativas válidas a serem feitas com base nos modelos de detecção apresentados. Os algoritmos de detecção de *features* conseguiram entregar um bom resultado em baixas resoluções, algo que pode ser explorado em uma análise mais a fundo do modelo. Como exemplo poderiam ser calculadas as *features* em resolução reduzida e utilizar algum algoritmo para ampliar a detecção e aplicar sobre o frame de vídeo em alta resolução, exibindo assim uma imagem de alta qualidade com taxa de quadros aceitável

para a aplicação. Outro algoritmo válido para ser testado é o AKAZE, que de acordo com a ExnoxSoftware, desenvolvedora do OpenCVforUnity, tem um desempenho melhor que o ORB e o SIFT, que é uma biblioteca paga.

Da mesma forma, para determinar a rotação e posição da área detectada sobre o frame da câmera, poderia ser construída uma função *findHomography()* própria baseada nos pontos obtidos pelo modelo de detecção de contornos das teclas pretas, discutido na seção 5.2.3 deste trabalho. O modelo em questão obteve um desempenho acima do ORB, abrindo a possibilidade de incrementar o modelo de detecção de contornos e aplicar as transformações de perspectiva para obter um ambiente de RA.

Por fim, é preciso citar o Vuforia como a melhor alternativa observada para construir o jogo de RA, sendo que obteve uma boa taxa de quadros em uma resolução de 1280x960, parecendo suficiente para uma boa experiência de realidade aumentada. Melhorando o jogo de ritmo MIDI para identificar corretamente o tempo e velocidade da música, pode ser utilizado o modelo de detecção com Vuforia, susbtituindo o plano pela trilha do jogo de ritmo MIDI, mostrando em um Head-Mounted Display (HMD) a tecla e o tempo correto para pressioná-la no teclado ou piano.

REFERÊNCIAS

ARAÚJO, Mônica et al. Realidade virtual: efeitos na recuperação do membro superior de pacientes hemiparéticos por acidente vascular cerebral. **Arq. Catarinenses Med**, v. 43, n. 1, p. 15–20, 2014.

BACK, D. **Standard MIDI-file format spec. 1.1, updated, 1999**. [S.l.: s.n.], 2016.

CALONDER, Michael et al. Brief: Binary robust independent elementary features. In: SPRINGER. EUROPEAN conference on computer vision. [S.l.: s.n.], 2010. p. 778–792.

CHANDLER, Paul; SWELLER, John. Cognitive load theory and the format of instruction. **Cognition and instruction**, Taylor & Francis, v. 8, n. 4, p. 293–332, 1991.

CHANG, Yao-Jen; CHEN, Shu-Fang; HUANG, Jun-Da. A Kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities. **Research in developmental disabilities**, Elsevier, v. 32, n. 6, p. 2566–2570, 2011.

CHEN, Jessie YC; THROPP, Jennifer E. Review of low frame rate effects on human performance. **IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans**, IEEE, v. 37, n. 6, p. 1063–1076, 2007.

CHOW, Jonathan et al. Music education using augmented reality with a head mounted display. In: PROCEEDINGS of the Fourteenth Australasian User Interface Conference-Volume 139. [S.l.: s.n.], 2013. p. 73–79.

FARDO, Marcelo Luis. A gamificação aplicada em ambientes de aprendizagem. **RENOTE-Revista Novas Tecnologias na Educação**, v. 11, n. 1, 2013.

GOODWIN, Adam; GREEN, Richard. Key detection for a virtual piano teacher. In: IEEE. 2013 28th International Conference on Image and Vision Computing New Zealand (IVCNZ 2013). [S.l.: s.n.], 2013. p. 282–287.

HACKL, Dominik; ANTHES, Christoph. HoloKeys-An Augmented Reality Application for Learning the Piano. In: FORUM Media Technology. [S.l.: s.n.], 2017. p. 140–144.

HOFMAM, Maurıcio et al. Um estudo sobre marcas fiduciais em realidade aumentada: combinando detecção de linhas com calibração de câmera. In: VIII symposium on virtual reality–SVR. [S.l.: s.n.], 2006. p. 337–348.

HUANG, Feng et al. Piano ar: A markerless augmented reality based piano teaching system. In: IEEE. 2011 Third International Conference on Intelligent Human-Machine Systems and Cybernetics. [S.l.: s.n.], 2011. v. 2, p. 47–52.

KAPP, Karl M. **The gamification of learning and instruction: game-based methods and strategies for training and education**. [S.l.]: John Wiley & Sons, 2012.

ROSTEN, Edward; DRUMMOND, Tom. Machine learning for high-speed corner detection. In: SPRINGER. EUROPEAN conference on computer vision. [S.l.: s.n.], 2006. p. 430–443.

RUBLEE, Ethan et al. ORB: An efficient alternative to SIFT or SURF. In: IEEE. 2011 International conference on computer vision. [S.l.: s.n.], 2011. p. 2564–2571.

SAVARD, JG. **The size of the piano keyboard**. [S.l.: s.n.], 2011.

SAVARD, John J. G. **The Size of the Piano Keyboard**. Disponível em: <http://www.quadibloc.com/other/cnv05.htm>. Acesso em: 24 mai. 2020.

SEIDLER, Linus. Using AR to Help Learn Songs on a PhysicalPiano. **Department of Informatics Technical University of Munich**, p. 1–65, 2019.

SILVA, Everton da; REIS, Dalton Solano dos. USO DA REALIDADE AUMENTADA COM MARCADORES DINÂMICOS, 2020.

SILVA, Juliana Rocha De Faria. "Algumas coisas não dá pra ensinar, o aluno tem que aprender ouvindo": A prática docente de professores de piano popular do Centro de Educação Profissional – Escola de Música de Brasília (CEP/EMB). **Dissertação (Mestrado em Música)-Universidade de Brasília**, p. 1–168, 2010.

SWELLER, John. Cognitive load during problem solving: Effects on learning. **Cognitive science**, Elsevier, v. 12, n. 2, p. 257–285, 1988.

TEH, C-H; CHIN, Roland T. On the detection of dominant points on digital curves. **IEEE Transactions on pattern analysis and machine intelligence**, IEEE, v. 11, n. 8, p. 859–872, 1989.

ZENG, Hong; HE, Xingxi; PAN, Honghu. FunPianoAR: a novel AR application for piano learning considering paired play based on multi-marker tracking. In: IOP PUBLISHING, 1. JOURNAL of Physics: Conference Series. [S.l.: s.n.], 2019. v. 1229, p. 012072.