IMPLEMENTAÇÃO DE SOFTWARE PARA TOTAL SKY IMAGER

SOFTWARE IMPLEMENTATION FOR TOTAL SKY IMAGER

Bruna Raquel Alves Silva
brunahouse9@gmail.com

Ygo Neto Batista
ygo@pesqueira.ifpe.edu.br

RESUMO

Este trabalho apresenta o desenvolvimento e implementação de um aplicativo no sistema operacional Android para Total Sky Imager. O equipamento comercial possui custo elevado de modo que se torna inviável a sua aquisição pelo IFPE campus Pesqueira. Esta pesquisa faz parte do projeto intitulado “Desenvolvimento e aplicação de tecnologias para monitoramento e controle de plantas de energia fotovoltaica – Total Sky Imager (TSI)” que está em desenvolvimento no IFPE campus Pesqueira. O TSI analisa imagens do céu, resultando em uma estimativa ou previsão da irradiância solar, que são relevantes informações para a geração da energia solar fotovoltaica. O modelo em desenvolvimento no IFPE fundamenta-se no baixo custo. Neste trabalho de TCC foram implementados em Android métodos de (i) aquisição de imagens, realizando capturas de forma automática, extraindo metadados e armazenando as informações em um banco de dados; (ii) remoção da distorção gerada por uma lente do tipo olho de peixe através de um novo algoritmo desenvolvido neste TCC, o qual resultou em uma redução no tempo de processamento desta etapa em 46,5% em relação ao algoritmo original de referência; e, (iii) segmentação, distinguindo os pixels entre céu, nuvens claras e nuvens escuras. As demais etapas do projeto, necessárias para a conclusão do TSI, fazem parte de outros planos de trabalho em andamento.

Palavras-chave: Total Sky Imager. Energia fotovoltaica. Previsão. Android.

ABSTRACT

This work presents the development and implementation of an application in the Android operating system for Total Sky Imager. Commercial equipment has a high cost so that its acquisition by the IFPE campus Pesqueira is unfeasible. This research is part of the project entitled "Development and application of technologies for monitoring and control of photovoltaic energy plants - Total Sky Imager (TSI)" which is under development at the IFPE campus Pesqueira. TSI analyzes images of the sky, resulting in an estimate or prediction of solar irradiance, which are relevant information for the generation of photovoltaic solar energy. The model under development at IFPE is based on low cost. In this work, methods of (i) image acquisition were implemented in Android, performing captures automatically, extracting metadata and storing the information in a database; (ii) removal of the distortion generated by a fisheye lens through a new algorithm developed in this work, which resulted in a 46.5% reduction in the processing time of this step in relation to the original reference algorithm; and, (iii) segmentation, distinguishing the pixels between sky, light clouds and dark clouds. The remaining stages of the project, necessary for the completion of the TSI, are part of other work plans in progress.

Keywords: Total Sky Imager. Photovoltaic Energy. Forecast. Android.

1 INTRODUÇÃO

A geração solar fotovoltaica cresceu 55,9% entre 2020 e 2021, conforme aponta o Balanço Energético Nacional (2022), ganhando espaço no mercado de energia. A partir dessa constatação, é importante ressaltar que “a conversão direta da energia solar em energia elétrica ocorre pelos efeitos da radiação sobre determinados materiais, particularmente os semicondutores” (ANEEL, 2016). Essa radiação solar, que atinge a superfície terrestre, varia em curto prazo, principalmente devido à movimentação de nuvens na atmosfera. Em fração de segundos, tal variação pode reduzir a geração de uma usina fotovoltaica. Portanto, é importante para usinas ter meios de prever a radiação incidente em curto prazo com o objetivo de otimizar sua geração uma vez que a previsão da radiação possibilita alocação adequada dos recursos energéticos, facilitando a gestão de geração em uma usina fotovoltaica.

Existem algumas formas de prever a irradiância solar, tais como método da persistência (*persistence method*), método das séries temporais (*times series method*), método da regressão (*regression method*), método de previsão numérica do tempo ou NWP (*numerical weather prediction method*), método aprendizado de máquina ou ML (*machine lerning method*) e método da imagem total do céu ou TSI (*total sky imager*). Esses métodos são descritos no trabalho intitulado “Estado da arte para previsão da radiação solar” (BOAVENTURA, DOMINGOS, OLIVEIRA, 2020). Dentre os modelos citados, o Total Sky Imager apresenta maior acurácia para previsão a curto prazo. Tal equipamento utiliza do processamento de imagens do céu para fazer a estimativa e previsão, em um curto período de tempo, da disponibilidade de irradiância daquela área.

A empresa *Yankee Environmental System* desenvolveu o modelo comercial TSI-880, que pode ser visto na Figura 1-(a). O modelo capta imagens do céu em formato JPEG, com resolução de 352 x 288 pixels, através de um sensor CCD. Em seguida, as imagens são processadas e os resultados são apresentados em tempo real através de um servidor web, por meio de vistas panorâmicas e animações. O maior problema do modelo é seu alto custo. As empresas que comercializam este equipamento solicitam cadastro e contato prévio para fornecer um orçamento, contudo, Lopes (2015) informa em seu artigo que equipamentos como o TSI-880 passam de USD 35k, em 2015, tornando inviável sua aquisição para algumas usinas de menor porte ou para fins educacionais.

Como alternativa aos modelos comerciais, pesquisas acadêmicas têm sido desenvolvidas. Echer (2005) desenvolveu um sistema de mapeamento da cobertura de nuvens do céu. O modelo desenvolvido conta com uma câmera digital e uma lente angular. A aquisição de imagem é feita através do sistema de cores RGB e depois transformada para o espaço de atributos matiz, saturação, valor (HSV). O pesquisador observou que neste espaço de cores é possível realizar a separação dos elementos da imagem (nuvens e céu) de forma mais precisa. O modelo foi implementado em um software para Windows que recebe a imagem capturada pela câmera. No entanto, o programa que captura a imagem não está associado automaticamente com o Windows, ou seja, o processamento da imagem só acontece após o usuário ativar manualmente o sistema. Esse processo é desvantajoso e deverá ser melhorado em atualizações futuras. A estrutura do modelo pode ser observada na Figura 1-(b).

Lopes (2015) também desenvolveu um protótipo com estrutura de software semelhante às das literaturas disponíveis. No entanto, diferente da maioria das propostas apresentadas, o autor desenvolveu uma estrutura física original, tornando o custo do projeto inferior a 300 €(euros). O problema do modelo é a não utilização de uma lente de ampliação da imagem, o que o limita o ângulo de visão. Na Figura 1-(c) é possível observar a estrutura do modelo. As imagens da Figura 1 foram retiradas de seus artigos originais, algumas estão em baixa resolução.

Figura 1: (a)- TSI 880 modelo comercial. (b)- Modelo desenvolvido por Echer (2016).
(c)- Modelo desenvolvido por Lopes (2015).

Fonte: Referidas pesquisas.

Nos últimos anos, o grupo de pesquisa em fontes de energia renováveis do IFPE campus Pesqueira tem se dedicado ao desenvolvimento de tecnologias para otimização de sistemas de geração fotovoltaico, dentre elas o desenvolvimento de um TSI.

Como principal diferencial em relação aos concorrentes estudados, este trabalho propõe um TSI de mais baixo custo visto que propomos utilizar um smartphone de entrada, eliminando a necessidade da aquisição de periféricos externos, tais como câmeras, GPS, módulos GPRS, wifi, Bluetooth e USB. De fato, o trabalho foi implementado e testado em um smartphone LG k40s, o qual possui processador de 2 GHz 8 cores, 3 GB de memória RAM e câmera de 13 Mp, e apresentou capacidade de processamento e memória adequados para a execução do algoritmo. Por fim, a presença de um sistema operacional Android facilita a implementação do algoritmo visto a vasta oferta de bibliotecas e ferramentas para programação.

As etapas de desenvolvimento do TSI são descritas no projeto “Desenvolvimento e aplicação de tecnologias para monitoramento e controle de plantas de energia fotovoltaica – total sky imager.” (BATISTA, 2021). Na figura 2 é possível observar o diagrama do projeto, que é composto por as etapas de aquisição da imagem, pré-

processamento, extração de características, interpretação da imagem e sistema mecânico. O objetivo deste trabalho é apresentar o desenvolvimento e implementação em Android dos algoritmos da aquisição de imagem e pré-processamento (segmentação e remoção da distorção).

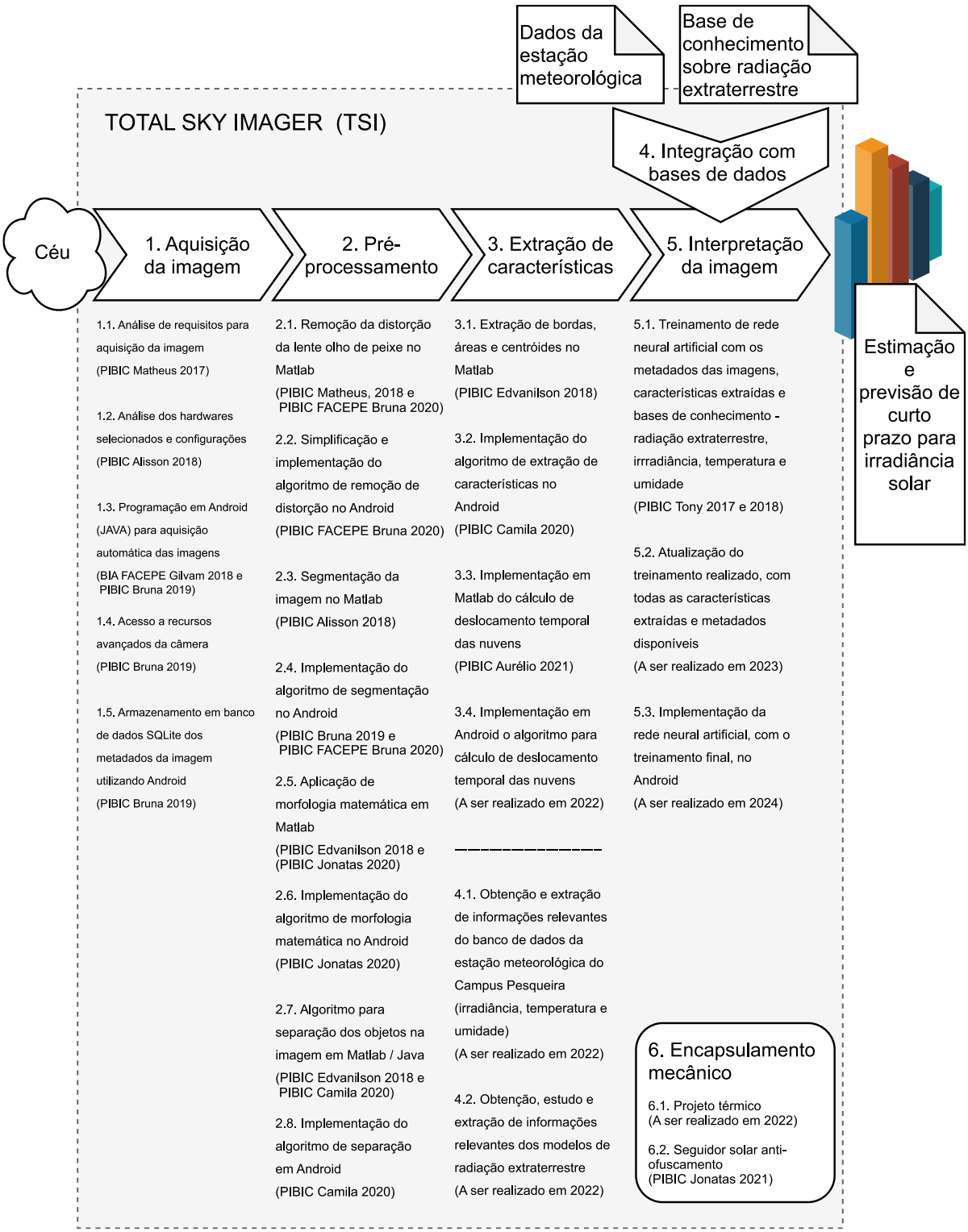
O tópico 1 (Figura 2) aborda o desenvolvimento do modelo para captura de imagens a partir da câmera embarcada no smartphone, associada a uma lente do tipo “olho de peixe”. O tópico 2 (Figura 2) é composto pelos algoritmos de segmentação, remoção da distorção gerada pela lente “olho de peixe”, morfologia matemática e separação dos objetos, conforme descritos a seguir.

A segmentação, realiza a distinção dos pixels que compõem o céu, nuvem clara e nuvem escura, presentes em uma imagem da abóbada celeste. A próxima etapa retira a distorção de uma imagem capturada com uma lente “olho de peixe”, que são lentes bicôncavas que ampliam o ângulo de abertura da câmera, a fim de capturar a maior área da abóbada celeste possível. A morfologia matemática é aplicada para filtrar ruídos na imagem e suavizar as bordas das nuvens.

O tópico 3 (Figura 2) aborda as etapas para implementação do método de extração de características relevantes das imagens, tais como tamanho, centro, direção e velocidade de deslocamento das nuvens. O tópico 4 (Figura 2) é responsável pelo mecanismo de análise dos bancos de dados disponíveis, tais como das estações meteorológicas e de modelos matemáticos da radiação solar extraterrestre. O tópico 5 (Figura 2) apresenta as etapas para desenvolvimento de uma rede neural artificial para realizar a estimativa e previsão da radiação solar. E por fim o tópico 6 (Figura 2) aborda a elaboração do sistema mecânico do TSI que deve agrupar o smartphone e a lente olho de peixe a um sistema antiofuscamento, responsável pelo bloqueio da incidência solar direta sobre a lente, evitando danos à lente e perda de informações das imagens.

Vale ressaltar que o objetivo deste trabalho é apresentar o desenvolvimento e implementação em Android dos algoritmos da aquisição de imagem e pré-processamento (segmentação e remoção da distorção).

Figura 2: Diagrama de implementações do TSI.



Fonte: Adaptação do projeto de pesquisa do TSI (BATISTA, 2021).

2 DESENVOLVIMENTO

Essa pesquisa aborda o estudo dos subtópicos 2.1 e 2.3 (pré-processamento) e implementação em Android dos tópicos 1.3 ao 1.5(aquisição de imagem), 2.2 e 2.4 (pré-processamento), que foram apresentados na Figura 2.

2.1 Algoritmo desenvolvido para remoção da distorção causada por uma lente olho de peixe (LIMEIRA, 2019)

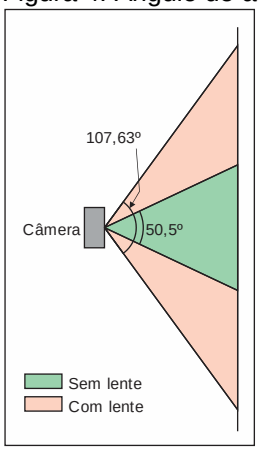
A utilização de uma lente olho de peixe em processos de aquisição de imagem resulta em uma maior área de captura da imagem. Neste projeto, Limeira (2019) utilizou uma lente sem especificação de fabricante, conforme Figura 3, cedida pelo professor orientador.

Figura 3: Modelo da lente olho de peixe utilizada.

Fonte: Limeira (2019).

Visando observar a ampliação do ângulo de abertura na captura da imagem, foi medido o ângulo de abertura da câmera do smartphone Samsung S9+ e, em seguida, foi medido o ângulo com a lente acoplada. Sem a lente, o ângulo de abertura foi de 50,5º, enquanto com a lente esse ângulo foi de 107.63º, um aumento de 113% no ângulo. A Figura 4 apresenta estes ângulos de abertura. O aumento do ângulo tem como fator negativo o embaçamento da imagem (visto que a lente é de baixo custo), porém, fotos de nuvens tem bordas suaves e não necessitam de capturar detalhes. Na Figura 5 é possível observar as imagens dos dois casos.

Figura 4: Ângulo de abertura da imagem.



Fonte: Autor (2022).

Figura 5: (a)-Captura de imagem sem a lente olho de peixe. (b)- Captura de imagem com a lente olho de peixe.

(a) (b)

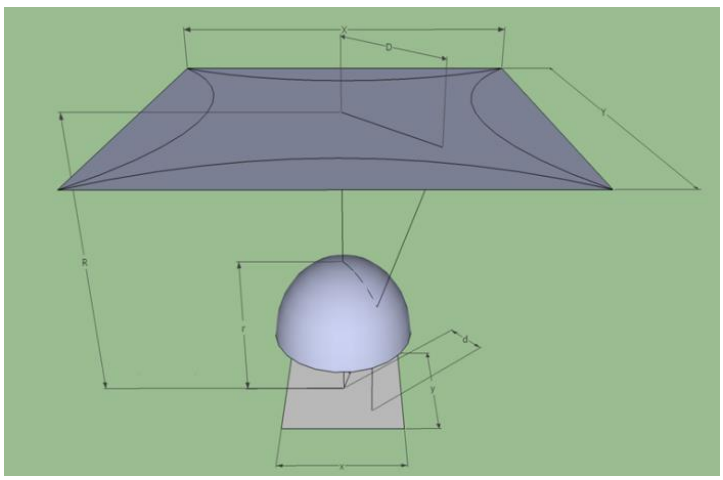
Fonte: Autor (2022).

No entanto, é gerada uma distorção na imagem que não é interessante para algumas aplicações, inclusive para o TSI. Se a imagem distorcida for processada da forma que capturada, as distâncias entre os elementos do céu (nuvens, sol) na imagem não serão lineares. Quanto mais lineares forem as entradas da rede neural, mais fácil será o treinamento da rede para obter previsões precisas.

Limeira (2019) criou um algoritmo para remoção da distorção. A implementação do algoritmo foi desenvolvida no artigo original em oito etapas, que podem ser resumidas em: (i) leitura da imagem distorcida; (ii) definição dos parâmetros da imagem distorcida; (iii) processamento da geometria da lente; (iv) criação de matrizes auxiliares para armazenamento temporário de imagens; (v) mapeamento das informações da imagem original na imagem de saída.

A Figura 6 mostra a geometria básica utilizada pelo algoritmo. O plano inferior representa a imagem original (com distorção causada pela lente). A calota acima representa a geometria da lente. Quanto menor o raio da calota, maior será a distorção eliminada pelo algoritmo. O plano superior representa a imagem final, sem distorção.

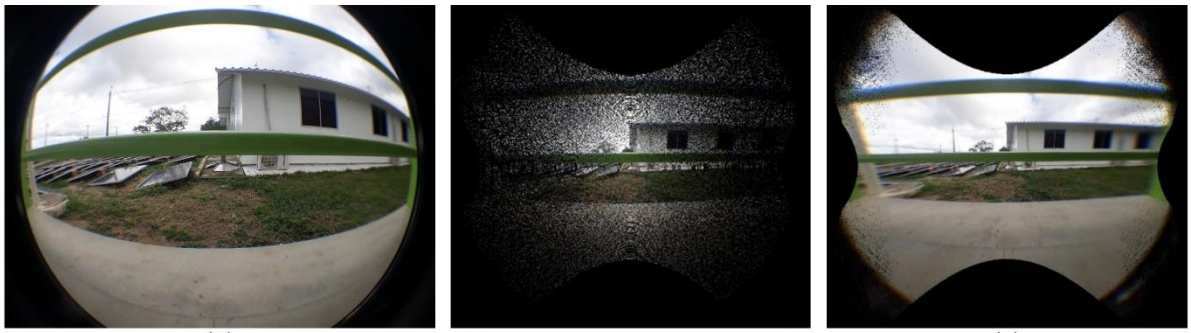
Figura 6: Dimensões da imagem em perspectiva.



Fonte: Limeira (2019).

Observe que a imagem final possui áreas sem informação da imagem original. Além disso, observe que a imagem original é “esticada” pelos vértices. O distanciamento dos pixels da imagem original devido esse “esticamento” causa pixels sem informações na imagem final. Para solucionar esta questão, Limeira et al (2019) fez o processo de cálculo da mediana dos pixels vizinhos aos vazios (sem informação da imagem original), os copiando e assim os preenchendo. O processo não gera perdas de informações. As imagens original, “esticada” e “preenchida” podem ser vistas na Figura 7.

Figura 7: Resultado obtido com algoritmo de remoção da distorção desenvolvido por Limeira (2019). (a)- Imagem com distorção. (b)-Imagem esticada com buracos. (c)-Imagem final com os buracos preenchidos.



Fonte: Limeira (2019).

Apesar de eficiente na remoção da distorção, o algoritmo proposto por Limeira (2019) necessita de alto poder computacional. O processo completo de remoção da distorção é feito em aproximadamente dois minutos em um processador i3 de 9ª geração. Ou seja, inviável para aplicação imediata em smartphone de baixo poder computacional.

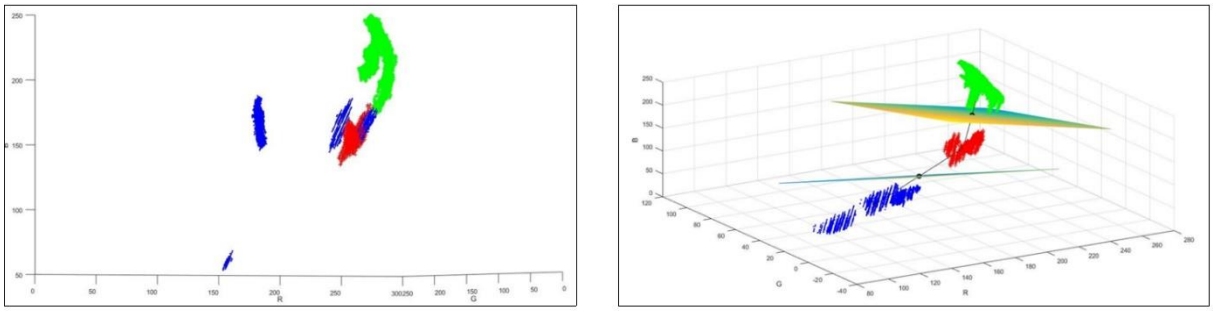
2.2 Algoritmo desenvolvido para Segmentação de imagens (TAVARES, 2019)

Utilizando o Matlab como ferramenta, Tavares (2019) desenvolveu um algoritmo capaz de separar os componentes de uma imagem da abóbada celeste, entre céu, nuvem clara e nuvem escura.

A primeira etapa foi definir a melhor configuração de câmera, especificamente a temperatura de cor da imagem capturada, e a melhor configuração de filtros ópticos, especificamente um filtro passa-faixa no visível e um filtro passa-faixa no infravermelho próximo.

O pesquisador capturou 22 fotos, com duas câmaras distintas (uma com e uma sem filtro passa-faixa no visível). E para cada câmera, foram variadas as temperaturas de cor. Para cada foto capturada, Tavares (2019) plotou a distribuição dos pixels no espaço RGB. Pixels de cores próximas tendem a ficar agrupados. Ou seja, pixels referentes às nuvens claras, às nuvens escuras e ao céu tendem a se agruparem. Chamamos estes agrupamentos de *clusters*. Quanto mais distantes forem estes *clusters* um dos outros, mais fácil será separar as regiões. Na Figura 8-a é possível observar sobreposição dos clusters, ou seja, imagens que apresentaram essas características foram descartadas por ser difícil de separar o céu e as nuvens, enquanto na Figura 8-b é possível separar os *clusters* utilizando planos.

Figura 8: (a)- Gráfico 3D da distribuição dos pixels das sub-regiões com sobreposição destas. (b)- Gráfico 3D dos clusters com planos de separação. As cores atribuídas pelo autor para os clusters (vermelho, verde e azul) não tem relação com as cores das componentes RGB dos pixels.



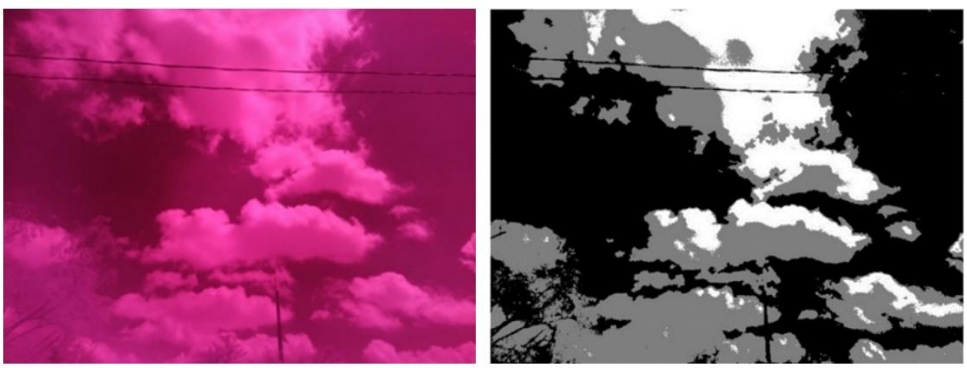
(a) (b)

Fonte: Tavares (2019).

Para as imagens que não apresentaram sobreposição entre os *clusters*, Tavares (2019) calculou a distância entre os seus centros de massa. Quanto maior a distância, mais separados estarão os clusters e mais fácil será classificá-los. Na Figura 8-b é possível observar as características das imagens sem sobreposição, os planos traçados definem de que região cada pixel faz parte, sendo: (i) região do céu se o pixel estiver abaixo do plano inferior; (ii) nuvem escura se estiver entre os dois planos; e (iii) nuvem clara se estiver acima do plano superior.

A configuração que resultou nos melhores resultados foi com a lente passa-faixa no visível, com a lente passa-faixa no infravermelho próximo e com a temperatura de cor da câmera ajustada para 5500 K. A imagem adquirida com esta configuração pode ser vista na Figura 9-a. No processamento, Tavares definiu os tons de interesse para cada região da imagem como sendo preto para o céu, cinza escuro para as nuvens escuras e branco para as nuvens claras. Na figura 9-b é possível observar o resultado da segmentação da imagem.

Figura 9: (a)- Imagem capturada com a melhor configuração dentre as 22 testadas; (b)- Imagem segmentada.



(a) (b)

Fonte: Tavares, 2019.

2.3 Android Studio

O Android Studio é uma plataforma de desenvolvimento para dispositivos Android, que disponibiliza uma vasta biblioteca de classes e métodos. Esta foi a plataforma utilizada para o desenvolvimento desta pesquisa.

2.3.1 Thread

O funcionamento de um aplicativo é baseado em processos. “Quando um aplicativo é aberto no Android, um processo dedicado no sistema operacional é criado para executá-lo.” (LACHETA, 2016). A classe *Thread* é a responsável pelo gerenciamento desses processos, como também pela atualização da tela, trabalhando em conjunto com a classe *Handler*, responsável por fazer a entrega de mensagens (processos que devem ser executados) de forma segura. A classe *Handler* permite gerenciar atividades de tempo em tempo.

2.3.2 Activity

“A classe *Activity* é um componente crucial de um aplicativo para Android” (DEVELOPERS, s.d.). Ela representa uma atividade, ação ou funcionalidade que pode ser realizada dentro da aplicação. De forma mais simples é o elemento que agrupa a interface do usurário (tela da aplicação) e suas funcionalidades.

2.3.3 Intent

A classe *Intent* (intenção) é utilizada quando há a necessidade de interagir com componentes do sistema Android. “Uma *Intent* fornece o recurso para realizar a vinculação entre os códigos em diferentes aplicativos.” (DEVELOPERS, s.d.). Ao estanciar a classe *Intent* é possível interagir com componentes da aplicação, como também com componentes de aplicativos existentes no dispositivo. Há duas formas de se trabalhar com a classe *Intent*: implícita e explicita. As *Intents* explícitas definem claramente o componente que deve ser chamado, normalmente são usadas dentro da própria aplicação. Intentes implícitas especificam a ação a ser realizada, e por opção o programador pode fornecer dados para ação. Na aplicação em questão as *Intents* foram empregadas das duas maneiras, explicitamente para circular entre telas da aplicação e implicitamente para o acesso a câmera.

2.2.4 ExifInterface

A classe *ExifInterface* é utilizada para ler e escrever *tags Exif* em diversos formatos de imagem. Os dados Exif (*Exchangeable Image File Format*) são “uma especificação que permite que fabricantes de câmeras digitais gravem informações sobre quais foram as condições técnicas na hora da captura junto ao arquivo da foto, como um metadado”.(COSSETTI, 2018).

2.2.5 Layout

Um dos pontos de destaque no desenvolvimento de um software é a interface gráfica. Existe algumas formas de gerenciar o *layout* das aplicações, organizando os componentes de forma adequada para cada aplicativo. A classe *View* gerencia e reapresenta os componentes interativos visuais do sistema, tais como: botões, caixas

de texto, etc. O gerenciador de *layout* é a forma como os componentes são organizados na tela.

2.3.6 Banco de dados

A implementação do banco de dados teve como base o conteúdo apresentado no livro “Android essencial” (LECHETA, 2016), que apresenta instruções para criação e manuseio de um banco de dados *SQlite*. Essa ferramenta é integrada ao Android Studio, sendo uma alternativa leve e eficaz.

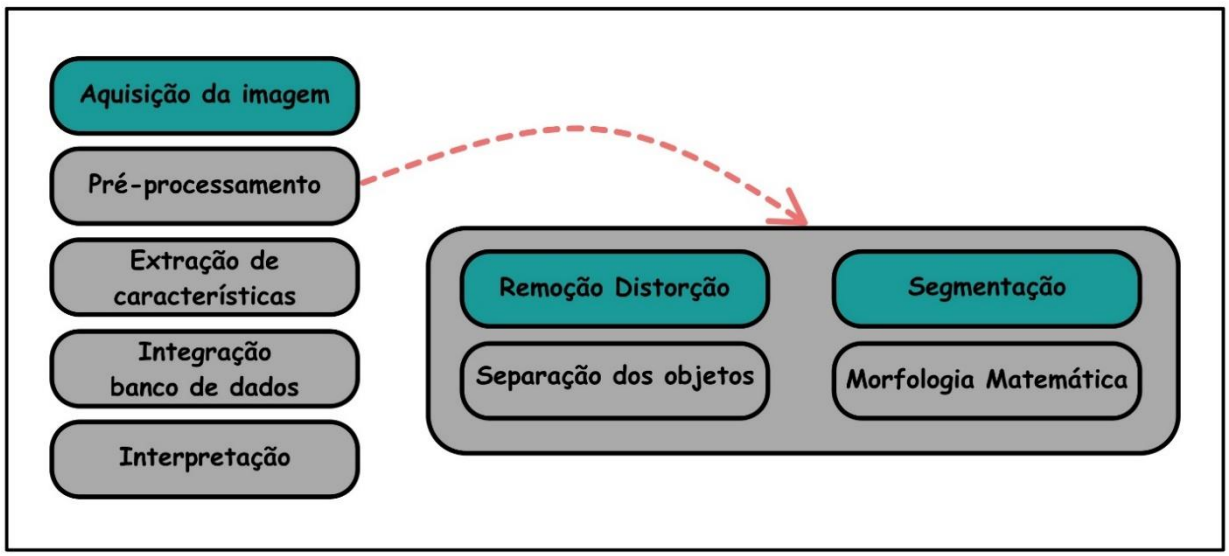
Para a criação de um banco de dados *SQlite* é necessário utilizar a classe *SQliteOpenHelper*, que possui três métodos obrigatórios: *onCreate()*, responsável por criar a estrutura da tabela de armazenamento; *onUpgrade()*, para atualizações da estrutura do banco; e *onDowngrade()*, utilizado quando existe uma nova versão do banco. O método *onCreate()* é o mais relevante para a aplicação do TSI, pois, ele é responsável por criar a estrutura das tabelas (linhas e colunas).

3 METODOLOGIA

Para o desenvolvimento desse trabalho foi adotado o método teórico experimental, que teve início com o estudo das pesquisas anteriores, realizadas pelo grupo de pesquisa do IFPE Campus Pesqueira, referentes ao desenvolvimento do TSI de baixo custo, seguido do estudo e implementação em Android, através da plataforma Android Studio.

A Figura 10 apresenta um diagrama com as *activities* que compõem a aplicação TSI. Para cada *activity*, há uma classe relacionada. As implementações a serem descritas nesse trabalho são referentes às classes *Aquisicao* (aquisição de imagem), *Distorcao* (remoção da distorção causada pela lente olho de peixe) e *Segmentacao* (segmentação – separação céu / nuvem). Estas duas últimas fazem parte do pré-processamento da imagem.

Figura 10-Diagrama das *activities* que compõem a aplicação TSI. As *activities* em verde são as que foram desenvolvidas neste trabalho.



Fonte: Autor (2022)

3.1 Classe Aquisicao (Aquisição de imagens)

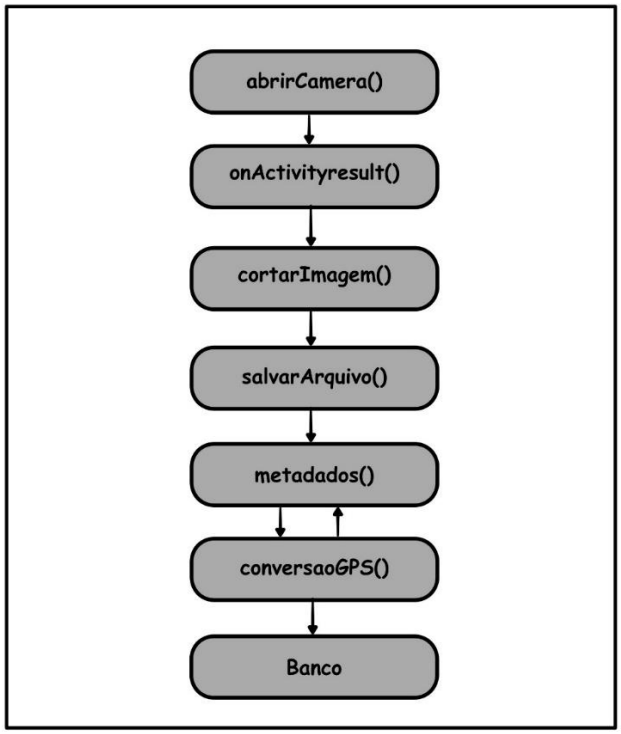
Na estrutura de programação do software foi criada a classe *Aquisição,* a qual é responsável pela captura, armazenamento e extração de informações das imagens (metadados do arquivo).

A captura de imagens pode ser feita de forma única (uma única imagem) ou de forma sucessiva (captura de imagens de forma automática, sem a interação direta do usuário para adquirir imagens). Ambos métodos de captura utilizam da mesma estrutura de acesso à câmera (será melhor detalhada à frente). A captura única acontece com um clique no botão de “aquisição única”, no qual a câmera do dispositivo é ativada e, posteriormente, é feita a captura de uma única foto.

Na estrutura de aquisição de imagens de forma sucessiva é pedido ao usuário que ele informe o tempo desejado entre capturas consecutivas. Após o preenchimento do tempo, o botão que inicia o processo de captura deve ser acionado. Em seguida, o método tempo*()* converte o tipo de dados e o valor digitado pelo usuário para um valor inteiro, representando os segundos da temporização. Logo após, é feita a chamada do método *cameraTemp()*, que aciona todo o mecanismo de captura, armazenamento e extração dos metadados das imagens. O método *newPostdelayed()* pertencente à classe *Handler* e é responsável pela temporização entre as capturas consecutivas. O *loop* chega ao fim quando detectado o acionamento do botão *stop*.

A captura da imagem, armazenamento e extração dos metadados ocorrem a partir da chamada dos métodos *abrirCamera(), onActivityResult(), cortarImagem(), salvarArquivo(), metadados(), conversaoGPS() e Banco()*. Um fluxograma dessa etapa poder ser visto na Figura 1.

Figura 11- fluxograma etapa de captura de imagem



Fonte: Autor (2022).

3.1.1 Acesso à câmera

Existem disponíveis na plataforma Android modelos de código que possibilitam acesso simplificado à câmera do dispositivo. No entanto, por questões de aplicabilidade, foi utilizada *Intent* implícita, solicitando acesso à câmera nativa do próprio dispositivo, com acesso a todos os recursos da câmera. Essa *Intent* é disparada no método *abrirCamera()*. O retorno da *Intent* (imagem capturada) é recuperado através do método *onActivityResult()*. Este método faz parte da classe *Intent*, sendo responsável por retornar os resultados da *Intent* instanciada, no caso da aplicação, um *bitmap*. O código 1 apresenta a estrutura do método.

Código 1: Método *abrirCamera()*

```
public void abrirCamera() {

    String sobreNome = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new
Date());
    String nome = "skyImager_" + sobreNome + ".jpg";//Nome do arquivo
    File pastaArquivos =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES +
"/Total Sky Imager/");//cria pasta no armazenamento interno
    pastaArquivos.mkdirs();//verifica se a pasta já existe

    caminhoImagem = pastaArquivos.getAbsolutePath() + "/" + nome;//Recebe o nome
do caminho do arquivo
    File file = new File(caminhoImagem);//cria um arquivo com caminho
    setFileFoto(file);
    Uri uriImagem = Uri.fromFile(file);//cria um Uri para imagem através do
arquivo
    //Intent para acesso a câmera do dispositivo, retornando a imagem capturada
através do Uri da imagem
    Intent cameraIntent = new
Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
    cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, uriImagem);
    startActivityForResult(cameraIntent, 1);}
```

Fonte: Autor (2021).

3.1.2 Redimensionamento da imagem

Visando reduzir o esforço computacional e, considerando que as imagens das nuvens no céu não possuem bordas nítidas, é possível reduzir a resolução da imagem sem perder qualidade das características extraídas das nuvens.

O redimensionamento é feito através do método *cortarImagem()*, que é chamado sempre que uma nova imagem for capturada. O método cria uma imagem vazia (pixels ainda não definidos) com dimensões pré-estabelecidas (400 x 400 pixels) e a preenche com os pixels centrais da imagem original (capturada). A resolução foi definida após testes com a câmera do smartphone Samsung S9+ e com a lente olho de peixe. O código 2 apresenta a estrutura do método.

Código 2: Método *cortarImagem ()*

```
private Bitmap cortarImagem(Bitmap bitmap) {
    Bitmap imagemCort = Bitmap.createBitmap(getImagemT(), 0, 0, 400, 400);//Cria
um Bitmap 400x400
    //Preenche o bitmap com os pixels da imagem capturada
    int wq = (getImagemT().getWidth() / 2 - 200);
    int hq = (getImagemT().getHeight() / 2 - 200);

    for (int x = 0; x < imagemCort.getWidth(); x++) {
        for (int y = 0; y < imagemCort.getHeight(); y++) {
            imagemCort.setPixel(y, x, bitmap.getPixel(x + wq, y + hq));
        }
    }

    imagemCapTela.setImageBitmap(imagemCort);//Mostra a imagem na Tela da
aplicação
    salvarArquivo2(imagemCort);
    return imagemCort;}
```

Fonte: Autor (2021).

3.1.3 Armazenamento da imagem

Após redimensionamento, é chamado o método *salvarArquivo()*, o qual cria um nome para o arquivo no qual será salvo o *bitmap*, seguindo a estrutura: “TSI_400X400_” + “yyyyMMdd_HHmmss” + “.jpg”. Em seguida, o *bitmap* é armazenado na memória interna do dispositivo com o nome definido conforme descrito. O código 3 apresenta a estrutura do método.

Código 3: Método *salvarArquivo ()*

```
public File salvarArquivo(Bitmap Bitmap2) {
    File pastaArquivos2 =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES +
"/Total Sky Imager 400x400/");
    pastaArquivos2.mkdirs();
    String sobreNome = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new
Date());//nome ao arquivo
    String nome = "TSI_400X400_" + sobreNome + ".jpg";
    setFileFoto(new File(pastaArquivos2, nome));
    setFileImaCort(getFileFoto());
    if (getFileFoto().exists())
        getFileFoto().delete();
    try {
        FileOutputStream salvar = new FileOutputStream(getFileFoto(), true);
        Bitmap2.compress(Bitmap.CompressFormat.JPEG, 100, salvar);
        salvar.flush();
        salvar.close();
    } catch (Exception e) {
        e.printStackTrace();}
        return getFileFoto();}
```

Fonte: Autor (2021).

3.1.4 Metadados

O software desenvolvido armazena e monitora os metadados das imagens, tais como data, hora, ISO, velocidade de exposição, longitude e latitude. Essas informações são importantes para melhorar o desempenho da rede neural artificial. Entretanto, esse trabalho não terá como foco essa discussão.

A extração dos metadados é realizada no método *metadados()*, o qual utiliza a classe *ExifInterface* (nativa do Android). Os metadados são lidos através de *Tags*, como a "*TAG_ISO_SPEED_RATINGS*", apresentada no Código 4, que retorna o valor do ISO. A leitura dos demais metadados é feita de forma similar, fazendo uso das inúmeras *Tags* que a classe *ExifInterface* possui.

Código 4: Método *metadados ()*

```
public void metadados(String filename) {
    try {
        ExifInterface exif = new ExifInterface(filename);

        setIso(exif.getAttribute(exif.TAG_ISO_SPEED_RATINGS));
        setVelExp(exif.getAttribute(exif.TAG_EXPOSURE_TIME));
        setDataTime(exif.getAttribute(exif.TAG_DATETIME));
        setLatitude(convGPS(exif.getAttribute(exif.TAG_GPS_LATITUDE)));
        setLongitude(convGPS(exif.getAttribute(exif.TAG_GPS_LONGITUDE)));

        Toast.makeText(Aquisicao.this, "Dados adicionados com sucesso!",
Toast.LENGTH_SHORT).show();
        db.exMetadados();
    } catch (IOException e) {
        e.printStackTrace();
        Toast.makeText(this, "Error!",
                Toast.LENGTH_LONG).show();}}
```

Fonte: Autor (2021).

3.1.5 Conversão GPS

O metadado que contém a informação de posicionamento global, a partir do sinal de GPS, retorna o valor no formato de graus, minutos e segundos. Contudo, em alguns modelos matemáticos para a estimativa da irradiância solar extraterrestre, a latitude deve ser indicada apenas em graus. Adicionalmente, mostrou-se mais simples utilizar a informação apenas em graus no Google Maps. Portanto, foi necessária a implementação de um método de conversão de graus, minutos e segundos para apenas graus, denominado *conversaoGPS()*.O código 5 apresenta a estrutura do método.

Código 5: Método *convGPS ()*

```
public String convGPS(String entradaGps){
        String[] texGPS= entradaGps.split("[,]");
        String[] tgraus= texGPS[0].split("[/]");
        String[] tminutos= texGPS[1].split("[/]");
        String[] tsegundos= texGPS[2].split("[/]");
        Double graus= -
(Double.valueOf(tgraus[0])+((Double.valueOf(tminutos[0]))/60)
+(Double.valueOf(tsegundos[0]))/10000/60/60);
        String sGraus= String.valueOf(graus);
        return sGraus; }}
```

Fonte: Autor (2021).

3.1.6 Banco de dados

Um banco de dados armazena os metadados extraídos da imagem de forma organizada, possibilitando sua utilização em outras situações. Na aplicação foi criada uma classe, a qual nomeamos de *Banco*, que é extensão da classe *SQLiteOpenHelper*. Esta classe possui métodos e atributos, tais como nome do banco, nome da tabela e versão do banco, esses parâmetros são referência para o armazenamento dos dados. Para a aplicação, a classe *Banco* inicializa os atributos com os valores nome do banco = *Banco de dados TSI*, nome da tabela = *Metadados* e versão do banco = *1*. O banco armazena as informações em formato de tabelas.

Em seguida, é necessário instanciar um objeto da classe *Banco*, o qual nomeamos *db*. No método *onCreate()* de *Banco* a estrutura da tabela é informada através da String *SQL_CREATE_ENTRIES*, a qual define a ordem em que as informações serão postas nas colunas da tabela. Para a aplicação, foi definida a ordem data, hora, latitude longitude, ISO e tempo de exposição. As linhas são autoincrementadas através de um ID à medida que uma nova imagem for capturada.

Os dados devem ser inseridos em *db* pelo método *insert()* e devem ser objetos da classe ContentValues, os quais possuem atributos que recebem os metadados da imagem. O código 6 apresenta a estrutura do método.

Código 6: Método *exMetadados ()*

```
public void exMetadados() {
    SQLiteDatabase db = getWritableDatabase(); //Cria ou abri o banco de dados
    ContentValues valores = new ContentValues(); /*ARMAZENAR O CONJUNTO DE
VALORES*/
    valores.put(DATA_TIME, Aquisicao.getDataTime());
    valores.put(ISO,Aquisicao.getIso());
    valores.put(TEMPO_EX, Aquisicao.getVelExp());
    valores.put(LATITUDE, Aquisicao.getLatitude());
    valores.put(LONGITUDE, Aquisicao.getLongitude());
    db.insert(TAB_METADADOS,null,valores);/* INSERIR TABELA*/
    db.close(); /* FECHAR O BANCO DE DADOS*/  }
```

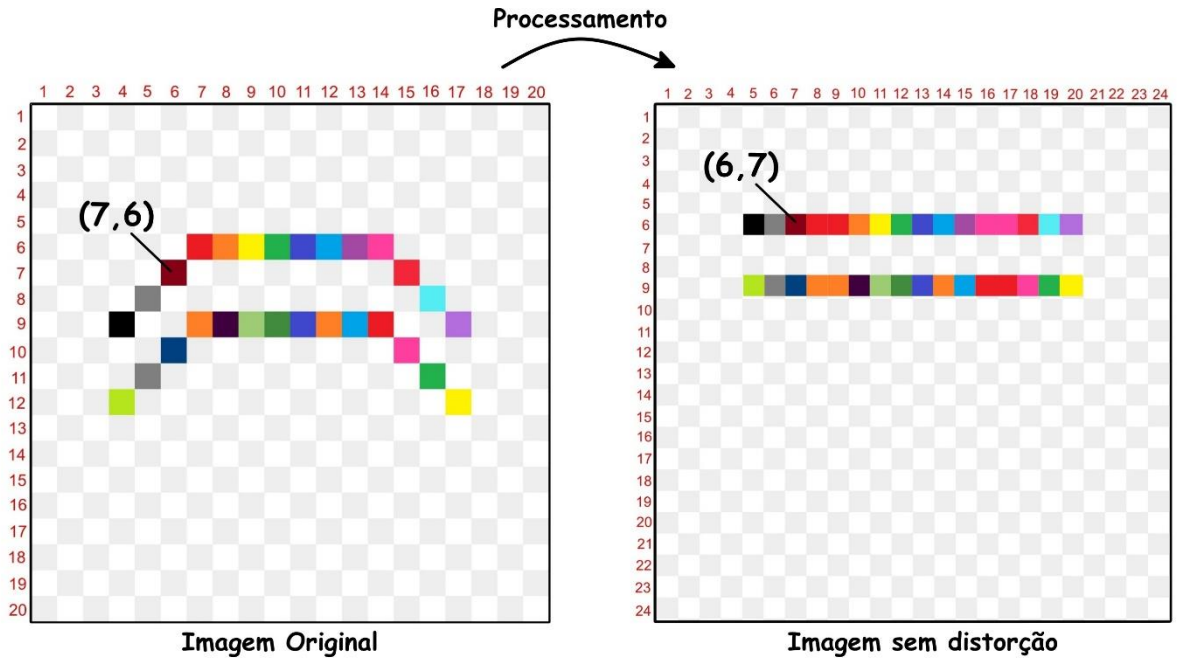
Fonte: Autor (2021).

3.2 Remoção da distorção causada por uma lente olho de peixe

Para a implementação no Android do método de remoção da distorção, foi feita a simplificação do modelo de remoção desenvolvido por Limeira (2019). A simplificação foi feita através da técnica de *Look Up Table* (LUT), que teve como objetivo converter o processamento, outrora desenvolvido, por uma simples estrutura de repetição. A obtenção da LUT se deu através do mapeamento dos pixels que continham informações de referência, para uma imagem após o processamento utilizando o algoritmo de Limeira (2019).

Na Figura 12, a “imagem original” representa a imagem de entrada, com distorção. A mesma imagem é submetida ao processamento desenvolvido por Limeira (2019), que resulta na “Imagem sem distorção”. A imagem sem distorção tem dimensões superiores à imagem original, portanto, pixels sem informações são criados. Estes pixels, no algoritmo de Limeira (2019), são preenchidos com as informações dos pixels vizinhos. Portanto, na LUT, pode ser que uma informação de um pixel da imagem de entrada seja mapeada para mais de um pixel na imagem de saída.

Figura 12- Ilustração do mapeamento. A informação marrom contida no pixel indicado na imagem, inicialmente alocada no pixel (4, 5), será realocada no pixel (3,4).



Fonte: Autor (2022).

Na seção 3.2.1 foi visto que a dimensão padrão para a imagem de entrada estabelecida foi de 400 x 400 pixels. Após processar a imagem com o algoritmo de Limeira et al (2019), descrito na seção 2.1, a resolução da imagem de saída foi de 548 x 548 pixels. A LUT tem a mesma dimensão da imagem de saída e armazena as coordenadas da informação na imagem original que preencherá cada pixel na imagem de saída. Esse mapeamento será sempre o mesmo para qualquer imagem de entrada, visto que a distorção depende apenas da geometria da lente, do conjunto óptico e do sensor, e não depende das informações na imagem capturada.

Processando ambos algoritmos em Matlab, em um computador com processador Intel(R) Core (TM) i5-7200U CPU @ 2.50GHz 2.71 GHz, 8GB de memória RAM, obtivemos os seguintes tempos de processamento:

Algoritmo de Limeira *et al*. (2019): 1110 ms;

Conversão por LUT: 590 ms,

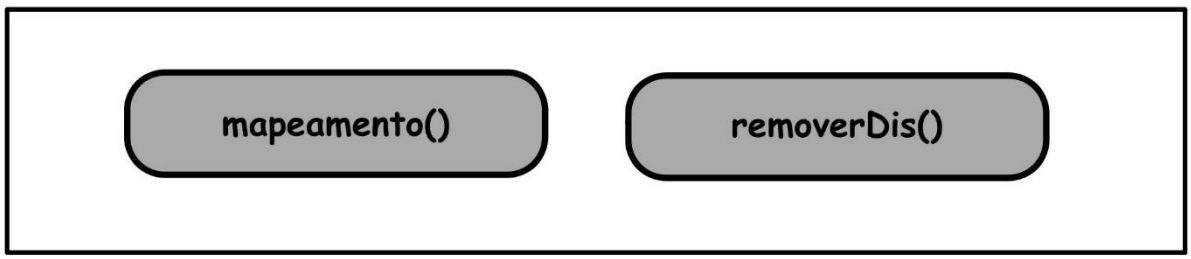
ou seja, obtivemos uma redução de 46,9% no tempo de processamento. Era esperada uma redução desta magnitude visto que no algoritmo original há diversas operações trigonométricas (para cada pixel da imagem), enquanto utilizando a LUT é somente são operações de “cópia e cola” de informações entre matrizes.

Como desvantagem, esta técnica necessita armazenar a LUT na memória. Nesta configuração, a LUT possui 900 kB. Contudo, não consideramos este armazenamento como problema visto que smartphones atuais tem capacidade de armazenamento na RAM acima de 2 GB.

3.2.1 Implementação do algoritmo de remoção da distorção no Android

A classe *Distorcao* é responsável por realizar o processo de remoção da distorção. Os métodos chamados nesta etapa estão no fluxograma da Figura 13.

Figura 13 – Fluxograma da etapa de remoção da distorção.



Fonte: Autor (2022).

A LUT foi levada para o Android através de um arquivo de texto (extensão .txt) e o arquivo é armazenado junto aos demais arquivos da aplicação. Para realização da leitura do arquivo de texto, conversão em tipo de dado numérico inteiro e armazenamento em matrizes de inteiros (LUT), foi criado o método *mapeamento()*.

O método *removerDis()* é responsável por copiar as informações da imagem original para uma nova imagem, seguindo o mapeamento indicado na LUT. O código 7 apresenta o método em questão.

Código 7: Método *removerDis()* (remoção da distorção).

```
public void removeDis(Bitmap bmp){
    Bitmap bmp2 = Bitmap.createBitmap(544, 544, Bitmap.Config.ARGB_8888);//Bitmap
a ser preenchido com os pixels da LUT
    for(int x=0;x<bmp2.getWidth();x++){
        for(int y=0;y<bmp2.getHeight();y++){
            bmp2.setPixel(x, y, bmp.getPixel(mapL[x][y],mapC[x][y])); //Preenche a
imagem final com a LUT
        }
    }
    setImagemSdis(bmp2);
    imgSemDis.setImageBitmap(bmp2);}
```

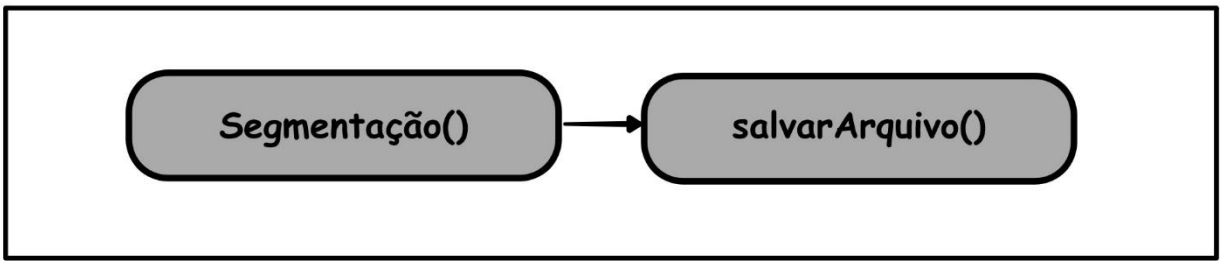
Fonte: Autor (2021).

No código 7 é criada uma imagem vazia. Seu preenchimento é feito através da estrutura de repetição (*for*) que varre a imagem de entrada (distorcida) e realoca os pixels na imagem vazia, com base nos valores da LUT, que tem seus dados armazenados nas matrizes *mapL[ ][ ]* e *mapC[ ][ ]*. Para cada informação armazenada em um pixel da imagem original, é necessário indicar a linha e a coluna do pixel na qual aquela informação será armazenada na imagem final, por isso são necessárias duas matrizes de mapeamento (ou seja, a LUT é na verdade composta por duas matrizes). O método *getPixel()* (próprio do Android) faz o acesso ao pixel da imagem de entrada e o *setPixel()* (próprio do Android) altera o valor do pixel na imagem "vazia". Ao fim da varredura, a imagem sem distorção é retornada.

3.3 Segmentação da imagem

O processo de segmentação da imagem utilizado é o descrito na seção 2.2. A classe *Segmentacao* é responsável por realizar a segmentação da imagem. Os métodos chamados nesta etapa estão no fluxograma da Figura 14.

Figura 14– Fluxograma da etapa de segmentação de imagens.



Fonte: Autor (2022)

3.3.1 Implementação segmentação da imagem no Android

No Android, cada pixel da imagem capturada pela câmera é armazenado em 4 bytes, os quais armazenam as informações ARGB (alpha, red, green e blue) daquele pixel. Considerando que todas as imagens adquiridas possuem como transparência padrão o valor 255, ou seja, imagem totalmente opaca, este parâmetro pode ser ignorado nas próximas etapas.

O algoritmo desenvolvido por TAVARES et al. (2019), originalmente em Matlab, foi implementado em Android e que pode ser visto no código 8. Diferente do Matlab,

que possuem comandos simplificados para operações com matrizes, no Android alguns comandos de processamento de matriz devem ser implementados com laços de repetição e testes condicionais.

Código 8: Método *segmentacao ()*

```
public void segmentacao(File f, BitmapFactory.Options options ) {
    Bitmap imR = Bitmap.createBitmap(imagemO.getWidth(), imagemO.getHeight(),
Bitmap.Config.ARGB_8888);
    for (int linha = 0; linha < alt; linha++) {
        for (int coluna = 0; coluna < larg; coluna++) {
            imR.setPixel(coluna, linha, (matrizR[coluna][linha] << 16) |
0xFF000000);}}

    int R, G, B = 0;
    for (int linha = 0; linha < alt; linha++) {
        for (int coluna = 0; coluna < larg; coluna++) {
            R = matrizR[coluna][linha];
            G = matrizG[coluna][linha];
            B = matrizB[coluna][linha];

            if (B > ((11288 - 41.7524 * R - 17.1889 * G) / 34.2023)) {
                //preenche nuvem escura
                matrizResR[coluna][linha] = 80;
                matrizResG[coluna][linha] = 90;
                matrizResB[coluna][linha] = 100;
            } else {
               // preenche céu
                matrizResR[coluna][linha] = 110;
                matrizResG[coluna][linha] = 160;
                matrizResB[coluna][linha] = 190;
            }
            if (B > (16097 - 31.2814 * R - 28.7598 * G) / 33.9229) {
               //preenche nuvem clara
                matrizResR[coluna][linha] = 230;
                matrizResG[coluna][linha] = 230;
                matrizResB[coluna][linha] = 220;
            }}}

         Bitmap imB = Bitmap.createBitmap(imagemO.getWidth(), imagemO.getHeight(),
Bitmap.Config.ARGB_8888);// cria a imagem final
    for (int linha = 0; linha < alt; linha++) {
        for (int coluna = 0; coluna < larg; coluna++) {
            imB.setPixel(coluna, linha, (matrizResB[coluna][linha] |
(matrizResG[coluna][linha] << 8) | (matrizResR[coluna][linha] << 16)) |
0xFF000000); } }

    //mostra na tela e salvar
    imagemOG.setImageBitmap(imagemO);//imagem original
    imagemOB.setImageBitmap(imB);//imagem final
    this.salvarArquivo(imB);}
```

Fonte: Autor (2021).

4 RESULTADOS E ANÁLISE

4.1 Aquisição de imagens

Na interface de Aquisição de imagens existe um *ImageView* responsável por apresentar a imagem capturada na tela. Durante o processo de aquisição, os metadados da imagem são extraídos, armazenados no banco de dados e é apresentado um informativo para o usuário. Na Figura 15 é apresentada a interface da classe de aquisição de imagens, com o resultado obtido.

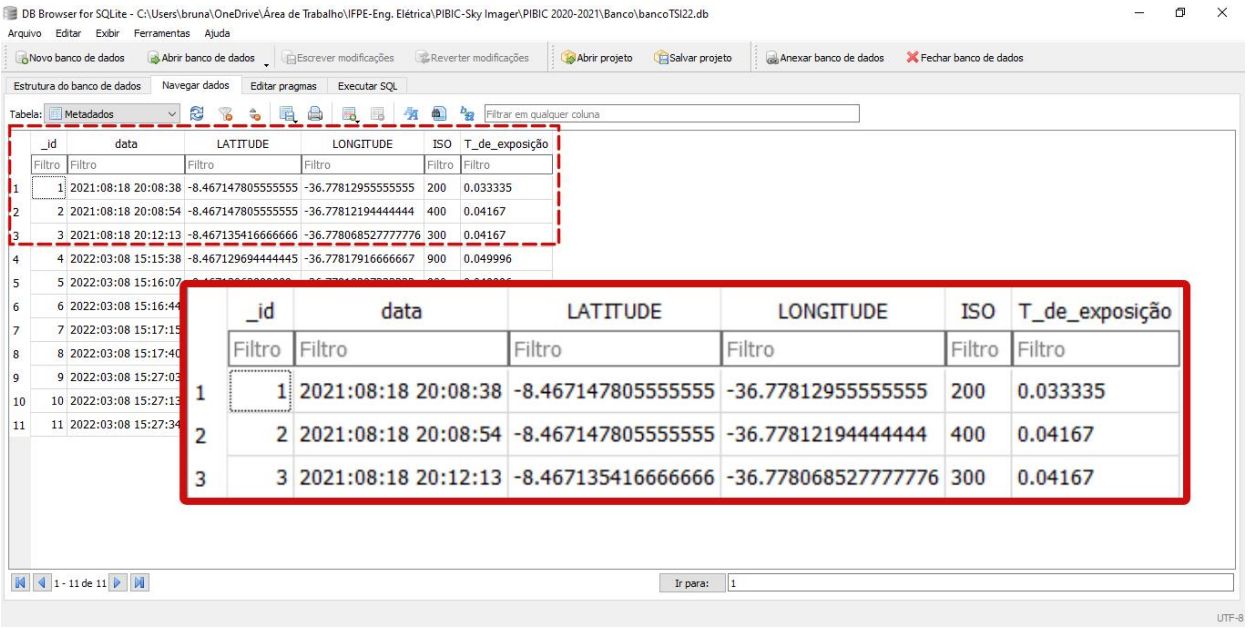
Figura 15– Tela da aquisição da imagem.

Fonte: Autor (2022)

Para navegar no banco de dados da aplicação, faz-se necessária a utilização de uma plataforma externa compatível com o banco de dados SQlite. Foi utilizada a

DB browser sqlite, a qual permite o fácil acesso e manuseio do Banco de dados. A Figura 16 apresenta a interface da plataforma com alguns dados.

Figura 16: Banco de dados apresentado na *DB browser sqlite*



Fonte: Autor (2022).

Os dados apresentados na Figura 16 são referentes às imagens capturadas com a aplicação, sendo assim, foi validada a funcionalidade dos métodos de extração e armazenamento de metadados. A disponibilização do banco possibilita o avanço do projeto em relação ao treinamento da rede neural artificial, a ser desenvolvido em trabalhos futuros.

4.2 Remoção da Distorção

A Figura 17 apresenta a interface da classe de remoção da distorção no Android, com o objetivo alcançado. Na interface existem dois botões, (i) Remover Distorção: ao ser acionado apresenta através das *ImageView* a imagem capturada e o resultado da remoção da distorção; e (ii) Validação: apresenta na interface uma imagem de referência, distorcida e previamente armazenada junto com o aplicativo e o resultado da aplicação do método de remoção, validando o processo.

Figura 17: Interface da classe de remoção da distorção

Fonte: Autor (2022)

A Figura 18 apresenta os resultados do processo de remoção da distorção. Conforme esperado, o resultado, obtido com o processamento no Matlab do algoritmo de LIMEIRA (2019) e o resultado obtido pelo desenvolvimento e aplicação da LUT no Android foram semelhantes numericamente ao compararmos as matrizes de saída dos dois algoritmos. A diferença é na precisão do ponto flutuante dos dois sistemas. Ou seja, não houve nenhum prejuízo em relação à qualidade do resultado obtido.

Figura 18: (a)- Imagem original (adquirida com a lente olho de peixe). (b)- Resultado da remoção utilizando o processamento no Matlab – LIMEIRA et al (2019). (c)- Resultado da implementação da Look Up Table no Android.

Fonte: Autor (2021)

No Android, utilizando um Smartphone LG K40S com Android 10, o processamento para remover a distorção (*removeDis()*) foi realizado em 790 ms. Este tempo de processamento torna viável a remoção da distorção em Android para o TSI.

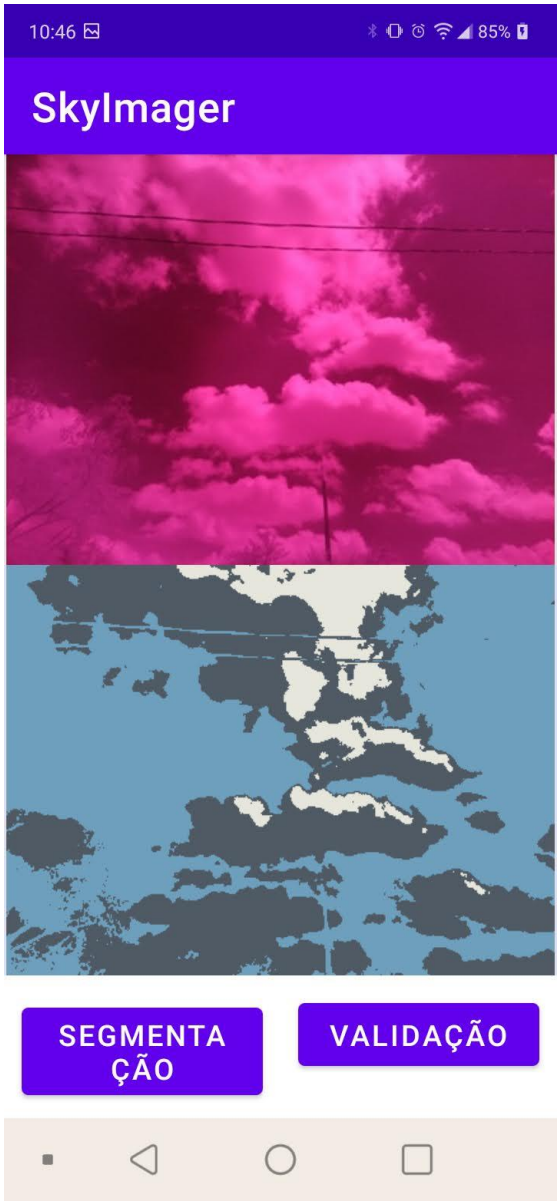
4.3 Segmentação de imagens

A Figura 19 apresenta a interface da classe de segmentação de imagens, com o resultado obtido. Semelhantemente a interface de remoção da distorção, a interface de segmentação possui dois botões, (i) Segmentação: ao ser acionado apresenta através das *ImageView* a imagem capturada e o resultado da segmentação para mesma; e (ii) Validação: apresenta na interface o resultado para uma imagem de referência previamente armazenada junto ao aplicativo para validação do processo.

Na imagem inferior da interface (Figura 19) é possível observar a distinção entre céu, nuvem clara e nuvem escura. O céu foi representado por uma tonalidade azulada (escolhida deliberadamente), e as nuvens se dividindo em dois tons de cinza para as nuvens claras e escuras. Por questões de estética, as cores adotadas para o preenchimento da imagem segmentada têm como base as cores do céu e das nuvens em um dia nublado, mas, posteriormente, podem sofrer alterações. Essas cores não são relevantes nas próximas etapas do processamento.

O tempo de processamento da segmentação de imagens em um Smartphone LG K40S Android 10 foi de 650 ms. Portanto, é viável implementar o algoritmo de segmentação proposto por Tavares et al (2019) no Android.

Figura 19: Interface classe de aquisição de imagens



Fonte: Autor (2022)

5 CONSIDERAÇÕES FINAIS

Os resultados obtidos alcançaram êxito, comprovando a viabilidade e validando os algoritmos desenvolvidos para o TSI, agora melhorados e implementados em Android, visto que a utilização do smartphone é o grande diferencial deste projeto.

Quando todos os requisitos funcionais forem alcançados, o modelo se tornará uma alternativa econômica para ser implantada e explorada como uma importante ferramenta de otimização para a usina solar localizada no IFPE Campus Pesqueira e para o desenvolvimento de novas pesquisas acadêmicas.

Do ponto de vista acadêmico, foram realizados estudos da linguagem de programação Java e criação de aplicativos para sistema operacional Android,

utilizando a interface de desenvolvimento Android Studio. Adicionalmente, o estudo de conceitos sobre processamento de imagens, física óptica (lentes e luz) e câmeras digitais foram necessários para o desenvolvimento da pesquisa. Portanto, nota-se que o conhecimento adquirido nesta pesquisa vai além daquele visto em sala de aula durante o curso de Bacharelado em Engenharia Elétrica.

Em relação a continuidade desta pesquisa, novos planos de trabalho sempre são elaborados pelo professor orientador a fim de concluir as etapas de extração de características da imagem, interpretação da imagem através de redes neurais artificiais, integração com bancos de dados da estação solarimétrica do IFPE Campus Pesqueira e construção da estrutura mecânica, incluindo o sistema anti-ofuscamento.

REFERENCIAS

AGÊNCIA NACIONAL DE ENERGIA ELÉTRICA. **Atlas de energia elétrica do Brasil**. Disponível em: https://www.aneel.gov.br/documents/656835/14876406/2005_AtlasEnergiaEletricaBrasil2ed/06b7ec52-e2de-48e7-f8be-1a39c785fc8b. Acesso em: 07 de dezembro de 2021.

BATISTA, Y. N. **Desenvolvimento e aplicação de tecnologias para monitoramento e controle de plantas de energia fotovoltaica – total sky imager.** 2021. Disponível em: https://www.ifpe.edu.br/o-ifpe/pesquisa-pos-graduacao-e-inovacao/grupos-de-pesquisa. Acesso em: 18 de junho de 2022.

BOAVENTURA, W.C; DOMINGOS, S.F.; OLIVEIRA, L. G. **Estado da arte para previsão da radiação solar.** VIII Congresso Brasileiro de Energia Solar. 01 a 05 de junho de 2020. Fortaleza.

COSSETTI, M. S. **O que são dados EXIF de fotos. Tecnoblog, 2018.** Disponível em: https://tecnoblog.net/responde/o-que-sao-dados-exif-de-fotos-e-como-encontra-los-ou-esconde-los/#:~:text=O%20Exchangeable%20Image%20File%20Format,da%20foto%2C%20como%20um%20metadado.Acesso em: 28 de maio de 2022.

DEVELOPERS. **Guias do desenvolvedor**. Disponível em: https://developer.android.com/guide. Acesso em: 12 de agosto de 2021.

ECHER, **Mariza. Desenvolvimento de um Sistema de Superfície para Mapeamento Automático da Fração de Cobertura de Nuvens**. Tese (Doutorado em Geofísica Espacial). Instituto Nacional de Pesquisas Espaciais, São José dos campos, 2005.

EMPRESA DE PESQUISA ENERGÉTICA. **Balanço Energético Nacional (BEN) 2021: Ano base 2020**. Disponível em: https://www.epe.gov.br/pt/publicacoes-dados-abertos/publicacoes/balanco-energetico-nacional-2021 .Acesso em: 04 de junho de 2022.

LECHETA.R. Android essencial.1. ed. São Paulo: Novatec, 2016.

LIMEIRA, M. **Pesquisa e desenvolvimento de algoritmos, com implementação em MatLab, para remover a distorção causada pela lente "olho de peixe" na aquisição de**

imagens. XIV Congresso de Iniciação Científica do IFPE - CONIC, 2019, Pesqueira. Anais eletrônico do XIV congresso de iniciação cientifica-CONIC, 2019.

LOPES, M. G. **Desenvolvimento de um sistema de baixo custo para a previsão da irradiância solar a curto prazo.** Dissertação (mestrado em engenharia física tecnológica). Instituto Superior Técnico da Universidade de Lisboa, Lisboa, 2015.

TAVARES, A. **Pesquisa e desenvolvimento de um sistema de processamento de imagens, baseado na análise espectral da radiação solar, para caracterização do céu e das nuvens, com aplicação em Sky Imager de baixo custo.** XIV Congresso de Iniciação Científica do IFPE - CONIC, 2019, Pesqueira. Anais eletrônico do XIV congresso de iniciação cientifica-CONIC, 2019.