PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Utilizando OCR para imagens de streetview

Eduardo Siqueira Torres da Silva

Projeto Final de Graduação

CENTRO TÉCNICO CIENTÍFICO - CTC
DEPARTAMENTO DE INFORMÁTICA
Curso de Graduação em Ciência da Computação

Rio de Janeiro, junho de 2024

Eduardo Siqueira Torres da Silva

Utilizando OCR para imagens de streetview

Relatório de Projeto Final, apresentado ao programa Bacharelado em Ciência da Computação da PUC-Rio como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Alberto Barbosa Raposo

Rio de Janeiro
Junho de 2024.

Resumo

Siqueira, Eduardo. Raposo, Alberto. Utilizando OCR para imagens de streetview. Rio de Janeiro, 2024. 31p. Relatório Final de Estágio Supervisionado II – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho consiste em analisar as dificuldades que OCR (Optical Character Recognition, ou Reconhecimento Ótico de Caracteres) tem quando é utilizado em imagens de streetview, listando as características deste tipo de imagem que gera estas dificuldades para OCR. O trabalho também propõe um pipeline para tratar as dificuldades identificadas, com detalhamento do que cada etapa do pipeline faz para o OCR atingir resultados favoráveis com imagens de streetview.

Palavras-chave

OCR, imagem, streetview, equiretangular, pipeline, cubemap, texto, caractere detecção, reconhecimento

Abstract

Siqueira, Eduardo. Raposo, Alberto. Utilizing OCR on streetview images. Rio de Janeiro, 2024. 31p. Relatório Final de Estágio Supervisionado II – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

This project consists of analyzing the difficulties OCR (Optical Character Recognition) has when utilized with streetview images, listing the characteristics about this type of image that creates these difficulties for OCR. It also proposes a pipeline to address the identified difficulties, detailing what each step of the pipeline does for OCR to achieve favorable results with streetview images.

Keywords

OCR, image, streetview, equirectangular, pipeline, cubemap, text, character detection, recognition

Sumário

1. Introdução

Nos dias de hoje, ferramentas para a leitura, processamento e geração de imagens estão ganhando foco. Utilizar do poder de processamento computacional ao invés dos olhos humanos para analisar em massa um número de imagens resultou tanto na agilização de atividades que requerem um alto nível de foco e conhecimento prévio de um ser humano para serem realizadas, quanto na melhora dos resultados, por uma máquina não ter tantas limitações quanto um ser humano para realizar um mesmo trabalho um enorme de vezes.

1.1 OCR

Com este surgimento de ferramentas e procedimentos que trabalham com imagens, um dos principais procedimentos que surgiu foi o OCR (“optical character recognition” em inglês, ou “reconhecimento óptico de caracteres" em português). Antes do surgimento de OCR, as ferramentas e procedimentos existentes para extração de texto só conseguiam processar arquivos de texto ou os dados de texto dentro do arquivo, mas não era possível processar arquivos de imagem com textos. Então, OCR tornou possível processar estes arquivos de imagens e retornar em seu output, em formato de texto, toda informação de texto que foi encontrada na imagem [1] [2].

Um OCR tem 3 etapas principais:

1. Etapa de detecção: identificar quais regiões da imagem há possíveis textos.
2. Etapa de reconhecimento: a partir das regiões da imagem com possíveis textos, OCR tenta identificar quais caracteres estão presentes em cada região.
3. Etapa de output: retornar todas as instâncias de texto que foram detectadas e reconhecidas.

Figura 1. Imagem executada no PaddleOCR; domínio público

Na figura 1 temos um exemplo do output de uma imagem analisada por um OCR, em específico o escolhido para este trabalho foi o PaddleOCR (link em: https://github.com/PaddlePaddle/PaddleOCR). Ambas as instâncias de texto presentes na imagem (“NEXT” e “4 MILES”) foram corretamente detectadas e reconhecidas pelo OCR e transcritas no output, junto de um valor de 0 a 1 que condiz ao grau de certeza que este OCR tem destes textos estarem corretos e de uma caixinha envolvendo a região que o OCR detectou alguma instância de texto.

1.2 Imagens de streetview

Imagens equiretangulares são imagens geradas por câmeras 360, que capturam todo o ambiente ao seu redor numa única imagem [3]. Este tipo de imagem ficou mais comumente conhecido como imagem de streetview, devido ao seu uso mais comum para mapeamento de regiões, como na plataforma do Google Maps. Pela visão de streetview é possível visualizar diferentes imagens que a câmera 360 tirou como se o próprio usuário estivesse se locomovendo no ambiente.

Por questões de conveniência, neste trabalho será usado o termo de streetview sempre para se referir à imagens de uma câmera 360.

Imagens de streetview são muito eficientes para mapear grandes regiões, porque essencialmente elas são um conjunto de várias imagens regulares que uma câmera normal tira. Mas nas plataformas como o Google Maps só é mostrada uma projeção de parte da imagem de streetview. Imagens de streetview inteiras apresentam algumas distorções e borramentos (Figura 2).

Figura 2. Fonte: https://commons.wikimedia.org/wiki/File:360%C2%B0_Panorama_vor_Stiftskirche_St.Maria_Magdalena_in_Flaesheim.jpg / image author: DerMische; under license: https://creativecommons.org/licenses/by-sa/4.0/deed.en

1.3 Motivações

OCR é quase puramente usado com imagens regulares. Enquanto imagens de streetview são uma forma mais eficiente de mapear um local, abrangendo quase 100% da região ao redor de uma câmera 360. Então seria um grande ganho se imagens de streetview fossem rodadas em OCR. Entretanto, há certos obstáculos que impedem que OCR tenha resultados bons quando rodado em imagens de streetview.

Este trabalho tem o intuito de identificar quais são os principais obstáculos que OCR tem com imagens de streetview e propor uma solução para estes problemas, criando uma ponte entre OCR e imagens de streetview, possibilitando que trabalhos futuros de OCR possam ser exercidos com conjuntos de imagens de streetview.

2. Situação atual

Os primeiros passos para o desenvolvimento de OCR com imagens de streetview ainda não foram tomados. O desenvolvimento e funcionamento de OCR é, praticamente, todo voltado para trabalhar com imagens regulares. As características que diferenciam imagens regulares de imagens de streetview são fatores que precisam ser levados em conta por um OCR para ele ter resultados bons.

Muitas das dificuldades mais comuns que OCR tem para detectar e reconhecer texto em imagens são amplificadas numa imagem de streetview [4]. Portanto, durante o desenvolvimento deste trabalho, não foram encontrados quaisquer exemplos de outros trabalhos realizados com OCR em imagens de streetview.

3. Objetivos

Os objetivos deste trabalho são definir quais são os obstáculos que OCR tem para produzir bons resultados com imagens de streetview e propor um pipeline de processamento das imagens de streetview para ultrapassar estes obstáculos. Para identificar os obstáculos serão realizados testes passando imagens de streetview diretamente num OCR, sem qualquer pré-processamento, e a partir dos resultados identificar quais características das imagens de streetview acarretam nestes obstáculos com OCR e porque elas têm um efeito negativo na performance de OCRs.

4. Tecnologias usadas

O trabalho todo foi realizado com a linguagem python e em jupyter notebook. O OCR escolhido para este trabalho foi o “PaddleOCR”. Nada foi testado com qualquer outro OCR, mas é esperado que qualquer OCR possa atingir os mesmos resultados encontrados neste trabalho. Para auxiliar na formação do pipeline, foram utilizadas as bibliotecas “py360convert” e “pyEquirectRotate”, cujas as funções serão mencionadas quando for explicado cada etapa do pipeline.

5. Avaliação de performance

Antes da formação do pipeline, foram feitos testes com OCR tanto em imagens regulares como em imagens de streetview. Estes testes servem para correlacionar as dificuldades que OCR tem, até mesmo, com imagens regulares e imagens de streetview, identificando as características de imagens de streetview que atrapalham na performance de OCR's.

5.1 Testes com imagens regulares

Os testes com imagens regulares consistiram em verificar o grau de sucesso tanto da detecção quanto do reconhecimento de textos quando os textos eram, de certa forma, afetados por quatro fatores escolhidos que têm chance de alterar os resultados de OCR. Estes fatores são:

- Iluminação: imagens com grau de iluminação baixo, tanto afetando o texto inteiro, como afetando apenas trechos do texto (quando parte do texto está com algum sombreamento por cima dele) - Figura 3.
- Distância: a distância física entre o texto e a câmera que capturou a imagem é grande. Como imagens são objetos 2D, neste caso, distância também pode ser entendida como o tamanho do texto em relação ao restante da imagem, então textos pequenos na imagem (Figura 4).
- Fonte: fontes de textos que se diferenciam significativamente das fontes padrões (Figura 5).
- Ângulo: se um texto está escrito de cima pra baixo, de baixo pra cima ou na diagonal (Figura 6).

Para os testes, foram passadas pelo PaddleOCR dez imagens para cada um dos fatores, onde a presença do fator na imagem era perceptível a olho nu, e mais dez imagens "mixed", com presença de dois ou mais destes fatores nas imagens, em todas as imagens sendo notável a influência do fator em um ou mais textos da imagem (Figura 7). Todas as imagens testadas são de domínio público.

Figura 3. Imagem para teste de Iluminação; domínio público

Figura 4. Imagem para teste de Distância; domínio público

Figura 5. Imagem para teste de Fonte; domínio público

Figura 6. Imagem para teste de Ângulo; domínio público

Figura 7. Imagem para teste de “Mixed”; domínio público

5.2 Resultados com imagens regulares

Para as imagens da categoria de iluminação, os resultados foram surpreendentemente positivos (Figura 8). A maioria dos textos presentes nas imagens o OCR conseguiu detectar e reconhecer corretamente com um grau de acurácia alto. Os casos onde não houve sucesso foram quando o texto, ou trechos do texto, não eram legíveis a olho nu, sem dar um zoom ou editar a imagem de alguma forma. Nestes casos o OCR falhou na etapa de detecção de texto, ou seja, ele não identificou a presença de qualquer texto nas regiões com forte sombreamento. Entretanto, quando OCR pode detectar a presença de texto nas áreas sombreadas ele também foi capaz de reconhecer quais são os caracteres corretos daquela instância de texto. Conclui-se que, mesmo se um texto esteja afetado por iluminação baixa ou sombreamento, se a olho nu uma pessoa consegue identificar o texto na imagem, o OCR também consegue, e o fator de iluminação afeta principalmente a etapa de detecção de texto mais do que a etapa de reconhecimento de texto.

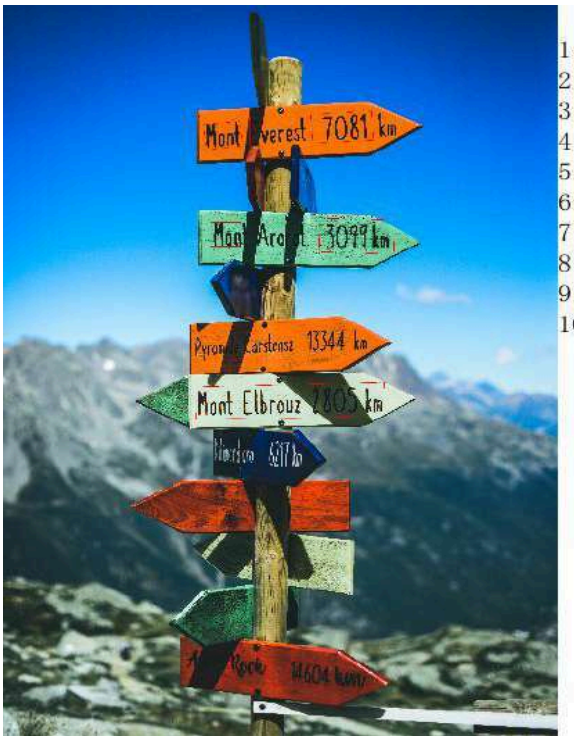

Figura 8. Imagem gerada pelo PaddleOCR da imagem de teste de Iluminação

Para as imagens da categoria de distância, os resultados também foram surpreendentemente positivos (Figura 9). No geral, os resultados obtidos se assemelham muito com os resultados da categoria de iluminação, o OCR consegue detectar e reconhecer os textos corretamente, com uma acurácia alta, até a distância atingir um limite que o texto não é mais legível a olho nu. Se o texto fica suficientemente distante (ou suficientemente pequeno) o OCR não consegue detectar a presença de qualquer instância de texto, mas se ele consegue ao menos detectar o texto, a etapa de reconhecimento retorna resultados bons. Conclui-se para a questão da distância os mesmo pontos para a questão da iluminação, se um texto é legível a olho nu, OCR não terá problemas, e esta questão tem maior impacto na etapa de detecção de texto do que na etapa de reconhecimento de texto.

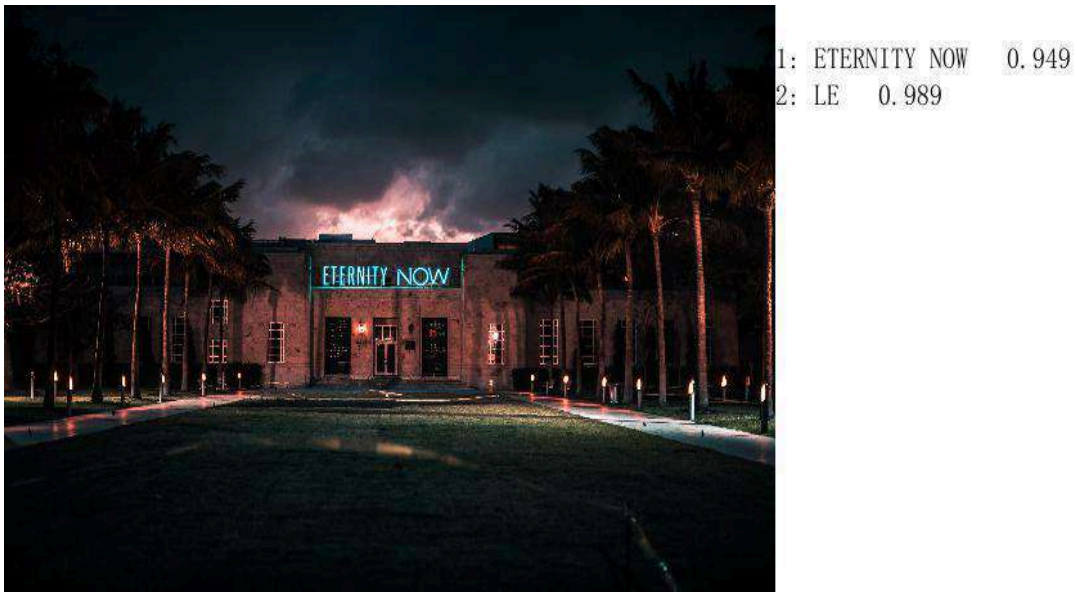

Figura 9. Imagem gerada pelo PaddleOCR da imagem de teste de Distância

Para as imagens da categoria de fonte, os resultados, em grande parte, não foram positivos, com falhas tanto na detecção quanto no reconhecimento de texto (Figura 10). O PaddleOCR teve bastante problemas detectando e reconhecendo texto com fontes que são significativamente distantes das padrões, além de ter valores de acurácia inferiores a 0.8, analogamente, as fontes mais próximas das padrões deram resultados melhores no geral. A fonte apresenta ser um fator de alta influência para os resultados de OCR, mas assim como a iluminação e a distância, tem como estabelecer um certo limite que o texto não ultrapasse para que os resultados não sejam afetados fortemente. Neste caso, este limite seria dado pela proximidade da fonte do texto para as fontes padrões. Outra solução possível para este problema seria adicionar à base de dados de treinamento do OCR fontes mais distantes das padrões, tendo em vista que poderá ter possíveis conflitos entre caracteres de fontes diferentes. Conclui-se que a fonte é a principal causadora de problemas tanto com a detecção e reconhecimento de textos, mas ela também é a que possui a solução mais clara (adicionar novas fontes para o treinamento do OCR), entretanto, não teria como esta solução cobrir todo e qualquer tipo de fonte que existe, muito menos espera-se que não surjam conflitos entre caracteres de fontes diferentes.

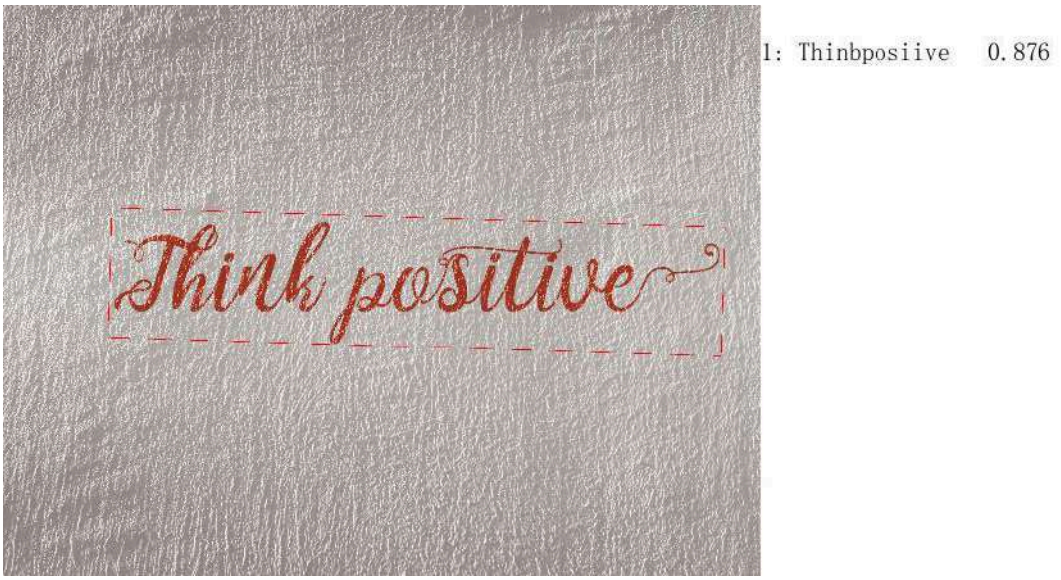

Figura 10. Imagem gerada pelo PaddleOCR da imagem de teste de Fonte

Para as imagens da categoria de ângulo, os resultados foram bons no geral, com algumas inconsistências (Figura 11). Em certos casos o OCR conseguiu extrair corretamente o texto de uma imagem, com um grau de acurácia bom, mas em alguns, até mesmo quando a angulação do texto era igual à de casos onde teve sucesso, o texto não era detectado. Todavia, o ângulo não aparenta ter afetado significativamente a etapa de reconhecimento de texto, somente a de detecção. Em ângulos menores o OCR apresentou resultados muito positivos, mas quando a angulação chegava por volta de 45 graus a inconsistência nos resultados aumentou. Conclui-se que a angulação do texto influencia principalmente na etapa de detecção de texto, e quando a angulação se aproxima dos 45 graus, a consistência da detecção de texto cai.

Para esta categoria de testes não foram utilizadas imagens com textos escritos na vertical (de cima para baixo ou de baixo para cima), nem textos com angulação próxima de 90 graus, pois foi um pressuposto para estes testes que o PaddleOCR não iria detectá-los. Entretanto, utilizando o PaddleOCR em casos fora deste trabalho foi notado que ele é capaz de detectar e reconhecer textos na vertical. A falta de testes com estes casos de textos não terá repercussão significativa para este trabalho, mas seria um trabalho futuro verificar se existe alguma relação da performance de OCR com textos na vertical, ou próximos da vertical, com imagens de streetview.

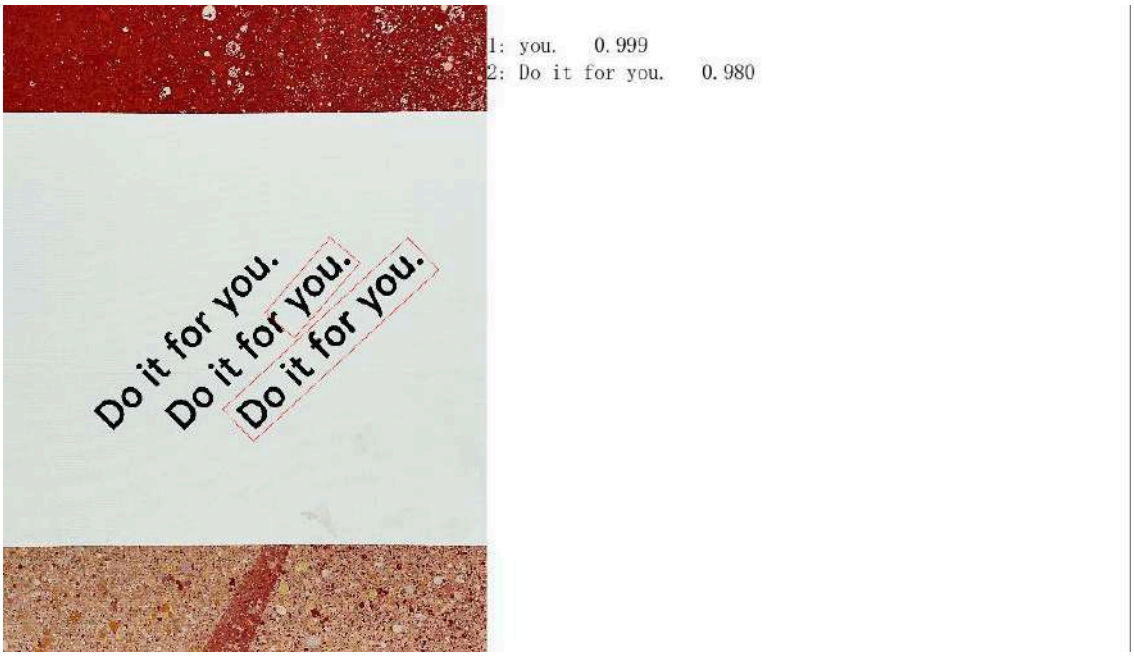

Figura 11. Imagem gerada pelo PaddleOCR da imagem de teste de Ângulo

Para as imagens da categoria “mixed” os mesmos problemas apresentados pelas categorias anteriores se repetem (Figura 12). Devido à etapa de detecção preceder a etapa de reconhecimento de texto, os fatores que afetam a detecção de texto têm maior influência nos resultados do OCR, somente quando a detecção é sucedida que é possível ver as influência destes fatores no reconhecimento. Somente nos casos onde estes fatores não atingem seus limites para interferir fortemente na detecção e reconhecimento de texto que os resultados foram positivos, mas quando qualquer um destes fatores atinge seu limite, os resultados pioram. Conclui-se que quando mais de um dos fatores testados está presente no texto os resultados têm maior risco de piorarem, sendo cada vez mais difícil de detectar e reconhecer corretamente qualquer instância de texto na imagem.

Figura 12. Imagem gerada pelo PaddleOCR da imagem de teste de “Mixed”

5.3 Testes com imagens de streetview

Para estes testes, pelo OCR, foram passadas 23 imagens de streetview (exemplo na Figura 13). Os resultados esperados eram de que quase nenhum texto seria detectado, apenas trechos de textos e alguns poucos textos posicionados em certas regiões da imagem estavam previstos a serem detectados.

Figura 13. Imagem de streetview usada para teste

5.4 Resultados com imagens de streetview

Os resultados obtidos condisseram com os resultados esperados dos testes, poucos foram os casos que o OCR detectou qualquer presença de texto. Mas estes testes também serviram para verificar algum vínculo entre as imagens de streetview e os quatro fatores que afetam textos em imagens vistos nos testes anteriores. Assim, é possível identificar uma relação entre imagens de streetview com os fatores de distância e ângulo em textos.

Imagens de streetview guardam uma grande quantidade de informação, de certa forma elas são um conjunto de várias outras imagens regulares menores que cobrem uma parte da imagem de streetview. Devido a isto, toda a informação da imagem tem de ficar comprimida para caber nas dimensões da imagem de streetview sem perda de informação, isto gera as distorções na imagem, diferenciando-a claramente de uma imagem regular. Por consequência destas distorções, qualquer texto que está presente na imagem fica distorcido, tendo seu tamanho em relação às dimensões da imagem muito pequeno, e sua angulação dificilmente permanecerá na horizontal. Portanto, textos nas imagens de streetview estão sujeitos aos mesmos problemas que textos com uma distância alta (ou tamanho pequeno) e uma angulação não horizontal e não uniforme numa imagem regular, e os resultados dos testes também mostraram o mesmo comportamento que os testes de distância e ângulo, quando um texto ultrapassa um certo limite em um destes quesitos o OCR falha em detectar a presença de texto (Figura 14).

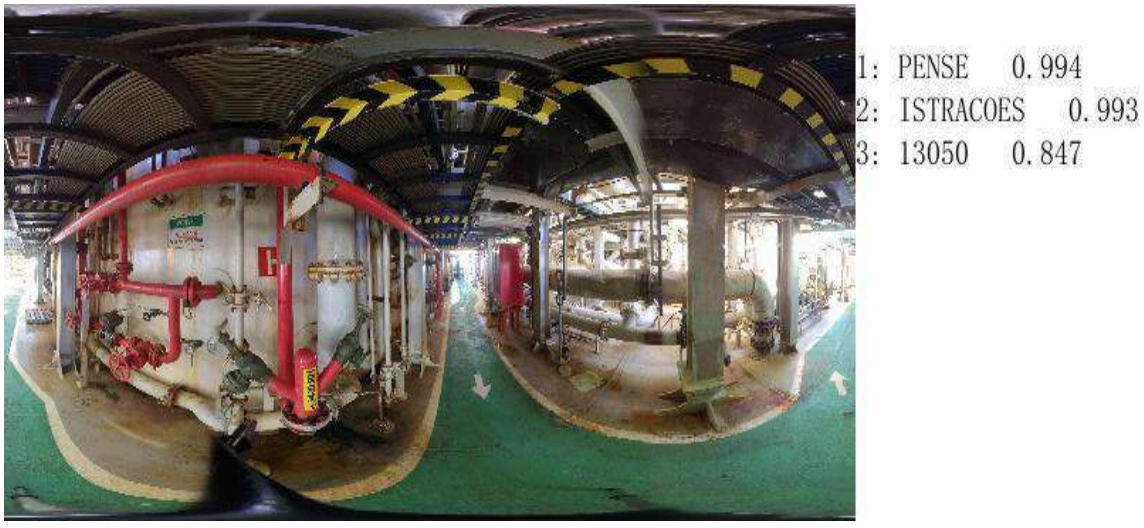

Figura 14. Imagem gerada pelo PaddleOCR da imagem de streetview

6. O problema

Após analisar os resultados dos testes, é possível concluir que a causa da má performance de OCR com imagens de streetview é o fato deste tipo de imagem ter muita informação que precisa ser comprimida para caber nas dimensões da imagem, por consequência desta compressão, a informação de textos da imagem tem seu tamanho reduzido e sua angulação fica distorcida. Somente textos que estão fisicamente muito próximos da câmera e em certas posições específicas têm uma boa chance de serem identificados por OCR, mas não seria uma solução viável estipular precisamente como uma imagem de streetview tem de ser capturada para depois ser analisada por OCR. Além disso, imagens de streetview são geralmente capturadas em grandes conjuntos, para mapear um local, então um mesmo texto pode estar presente em várias imagens diferentes, porém, o OCR pode ser capaz de identificar o texto em algumas imagens (por causa de um posicionamento específico entre a câmera e o texto) mas não em outras.

7. Solução proposta

Para solucionar este problema de baixa performance de OCR com imagens de streetview, este trabalho propõe transformar a imagem de streetview em seis imagens de cubemap que compõem a imagem original (Figuras 15 e 16) [5].

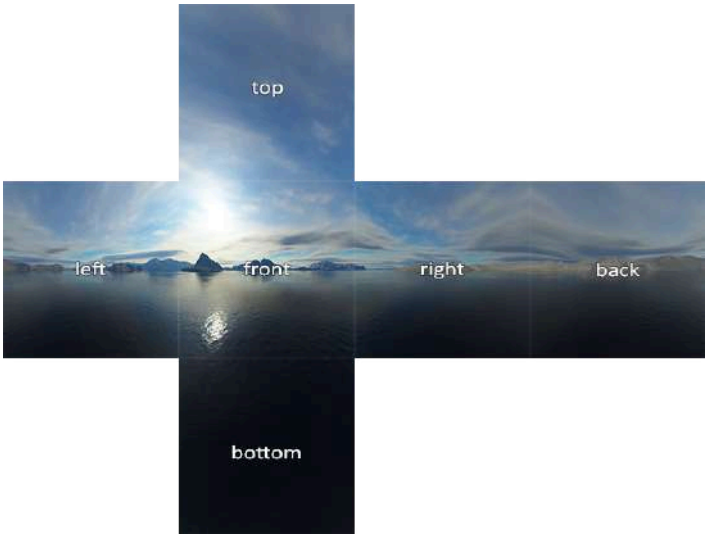

Figura 15. Exemplo de Cubemap

Figura 16. Faces do cubemap numa imagem de streetview

Como já foi destacado, o principal problema das imagens de streetview é que elas comprimem muita informação para dimensões que não suportam perfeitamente grandes quantidades de informação, o que resulta na compressão e distorção da informação da imagem. Mas, ao separar esta informação em seis partes com dimensões comuns para imagens regulares a informação fica mais bem tratada para o OCR, se assemelhando a uma imagem regular qualquer (exemplo na Figura 17).

Figura 17. Face “Front” do Figura 13

Também é importante destacar o detalhe de que no processo de transformar uma imagem de streetview nas seis imagens de cubemap é possível que as informações de textos fiquem partidas em duas imagens diferentes. Este processo irá gerar seis imagens sem overlapping de informação, mas qualquer informação posicionada na região onde duas das faces do cubemap se separam, naturalmente, também será separada em duas partes, logo, ao passar ambas as faces com um pedaço da informação de texto em cada no OCR, não seria identificada a instância de texto inteira, apenas suas partes. Para resolver este problema foi preciso integrar a função de rotacionar imagens de streetview no processo de leitura de imagens de streetview com OCR.

8. Pipeline

O pipeline que este trabalho propõe para utilizar OCR com imagens de streetview segue os seguintes passos (Figura 18):

1. Transformar a imagem de streetview em seis imagens que compõem o seu cubemap.
2. Passar as imagens de cubemap no OCR.
3. Salvar a informação das instâncias de texto encontradas pelo OCR.
4. Verificar se as coordenadas de bounding box de qualquer instância de texto estão próximas das bordas da imagem onde a instância de texto foi encontrada.
5. Caso haja coordenadas próximas das bordas, rotacionar a imagem de streetview original e repetir os passos 1, 2 e 3, até um máximo de três vezes.

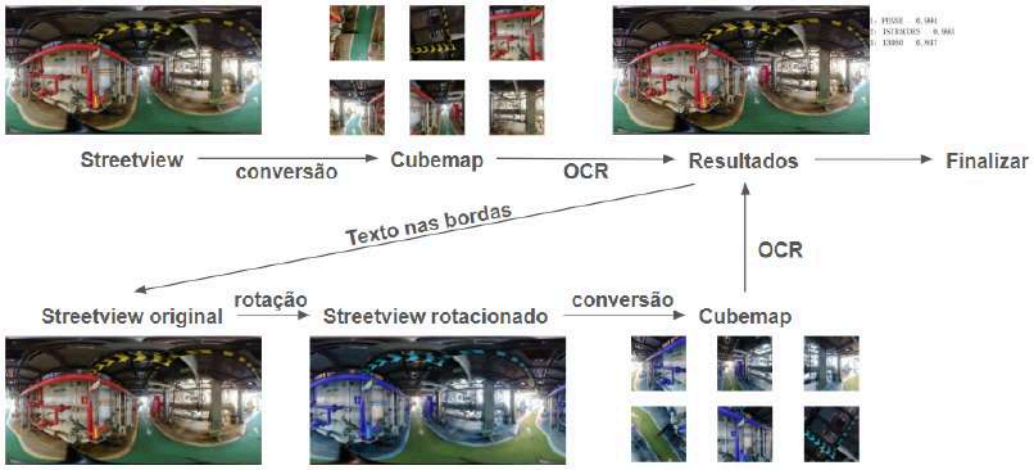

Figura 18. Pipeline proposto.

Neste trabalho, para a etapa de “conversão” foi utilizada a biblioteca “py360convert” (link em: https://github.com/sunset1995/py360convert) para realizar a conversão das imagens de streetview para as imagens de cubemap, mas qualquer outro método de conseguir as imagens de cubemap também pode ser usada, desde que não haja muito overlapping de informação entre duas ou mais faces do cubemap, e/ou não haja perda significante de informação da imagem original. Na etapa “OCR”, como já foi estipulado anteriormente, o OCR escolhido foi o “PaddleOCR”, mas qualquer outro OCR também pode ser usado, desde que o OCR retorne as informações das coordenadas de bounding box das instâncias de texto encontradas, senão, não terá como identificar textos que atravessam a interseção entre duas faces do cubemap. Antes de passar pela etapa “Texto nas bordas”, é feita uma simples comparação entre as coordenadas de bounding box de todos os textos encontrados pelo OCR com as coordenadas das bordas das imagens de cubemap, se alguma instância de texto estiver suficientemente próxima de uma borda da imagem, é pressuposto que esta instância de texto possa ser uma metade de uma instância de texto maior que foi partida em duas pelo conversão da imagem de streetview para seis imagens de cubemap, portanto, não foi identificada pelo OCR, logo passa para a etapa “Texto nas bordas”. Caso nenhum texto esteja próximo das bordas, o pipeline finaliza diretamente. São necessárias, no máximo, três rotações na imagem de streetview original, pois, com três rotações específicas já é possível cobrir todas as vizinhanças entre duas faces de um cubemap, as especificações de como estas rotações têm de ser feitas serão dadas no capítulo sobre esta etapa do pipeline. Na etapa “rotação” foi utilizada a biblioteca “pyEquirectRotate” (link em: https://github.com/BlueHorn07/pyEquirectRotate) para rotacionar as imagens de streetview, novamente, qualquer outra forma de realizar a função de rotacionar imagens de streetview pode ser utilizada, desde que consiga chegar os mesmos resultados que as ferramentas escolhidas para este trabalho chegaram.

9. Código e testes

Para verificar se o pipeline proposto é funcional, neste trabalho também foi feita a implementação e testagem do pipeline usando a linguagem Python. As especificações de como foram implementadas e testadas cada etapa do pipeline também estão descritas para evitar e/ou contornar futuros problemas que poderiam surgir.

9.1 Etapa de conversão

Para esta etapa utiliza-se da biblioteca “py360convert” para converter as imagens de streetview em cubemap, utilizando a função “e2c(e_img, face, mode, cube_format)” desta biblioteca. Sobre os parâmetros desta função, “e_img” é a imagem de streetview a converter, mas preciso passar esta imagem no formato “numpy array”, portanto pode ser necessário transformar a imagem para este tipo, o parâmetro “face” são as dimensões (altura e largura) em pixels das imagens de cubemap a serem geradas e os parâmetros “mode” e “cube_format” são as especificações de como a conversão tem de ser feita, neste trabalho foram usados os parâmetros “bilinear” e “dict”, respectivamente, para os parâmetros, tentativas com os outros valores para estes parâmetros foram feitas, mas somente com estes valores que a função não retornou erro.

É importante destacar que esta função de conversão em cubemap não tem tempo de processamento insignificante, portanto, um grupo de poucas imagens são convertidas relativamente com rapidez, mas um grupo de muitas imagens irá tomar bastante tempo. Outro detalhe importante é que naturalmente as faces “back” e “right” geradas no cubemap estão espelhadas (Figura 19), logo, é preciso espelhar elas novamente para não enviar para o OCR ler uma imagem em que todos os textos estão espelhados. Para resolver este problema foi utilizada a função “ImageOps.mirror()” da biblioteca “PIL” (Figura 20), mas qualquer outra forma de realizar a função de espelhar uma imagem também servirá.

Após espelhar as faces “back” e “right” tudo o que resta é certificar-se de salvar todas as seis faces de cubemap de cada uma das imagens convertidas em algum local para serem, em seguida, passadas para o OCR.

Figura 19. Exemplo de face espelhada do Cubemap

Figura 20. Figura 19 após ser espelhada com a função “ImageOps.mirror()”

9.2 Etapa de OCR

Nesta etapa só é preciso passar todas as imagens de cubemap geradas na etapa anterior em qualquer OCR. O único requerimento é que seja possível acessar os dados das coordenadas dos bounding boxes dos textos encontrados pelo OCR escolhido, já que estes dados são necessários para a etapa seguinte. É sugerido que tenha um padrão fixo na ordem que todas as imagens serão passadas para o OCR, principalmente se forem passadas um grupo grande de imagens, por exemplo, passar em seis por seis as imagens de cubemap, sendo as seis imagens que compõem uma única imagem de streetview, e que estas seis imagens das faces também sempre sigam a mesma ordem para todo grupo de seis imagens (a ordem escolhida durante a codificação e testagem do pipeline deste trabalho foi alfabética; “back” - “bottom” - “front” - “left” - “right” - “top”). Não é estritamente necessário que haja um padrão na ordem que as imagens são passadas para o OCR, mas isto também irá facilitar a implementação da etapa seguinte (Figura 21).

Figura 21. Resultados do PaddleOCR para uma das faces de um cubemap

9.3 Etapa de checar bordas

Esta etapa não é estritamente necessária para todas as situações que se busca encontrar todas as instâncias de texto em imagens de streetview. Se o conjunto de imagens a serem passadas para o OCR não for muito grande, ou busca-se apenas uma rápida leitura de OCR numa imagem de streetview esta etapa pode ser destacada, substituindo a função de checagem automática das bordas desta etapa por uma checagem pessoal sem ter grande perda de tempo.

A forma como foi implementa esta etapa para a testagem do pipeline para este trabalho não tem custo de tempo significativo, só haverá grande custo de tempo caso esta etapa identifique a presença de texto próximo das bordas que ativará a próxima etapa (rotação) e poderá mais que dobrar o tempo de processar uma única imagem. Mas, caso seja utilizado um conjunto grande de imagens de streetview, é recomendado implementar esta e a seguinte etapa, mesmo que elas aumentem significativamente o tempo para processar todas as imagens, é inviável verificar pessoalmente um grande conjunto de imagens pessoalmente, buscando quaisquer textos que foram cortados pelas bordas, a melhora da eficácia com estas etapas pode valer a pena um tempo de execução maior. Fica a rigor da pessoa implementando este pipeline proposto se serão implementadas ambas estas etapas ou não.

Esta etapa serve para verificar se há presença de textos próximos das bordas e acionar a etapa seguinte, caso encontre-se textos próximos das bordas, ou finalizar o processo, caso não encontre-se textos próximos das bordas. Para esta verificação é necessária as coordenadas de bounding box de todos os textos encontrados pelo OCR, assim pode-se comparar todas elas com as coordenadas das bordas (eixo X: 0 e valor da largura; eixo Y: 0 e valor da altura) da imagem de cubemap a qual o texto do bounding box referido foi encontrado, se a distância entre uma das coordenadas de bounding box de um texto e as de uma das bordas da imagem for menor que um limite estipulado, a verificação retorna positiva para a instância de texto verificada. Caso, alguma verificação retorne positiva é importante que seja guardada tanto a informação qual das quatro bordas da imagem o texto está próximo e em qual face do cubemap aquele texto pertence, pois estas informações ditam qual rotação deve ser feita na etapa seguinte. Mesmo que haja uma instância de texto que retornou positiva na verificação, é necessário verificar todas as instâncias de texto restantes ainda, porque outras instâncias de textos podem demandar uma rotação diferente de outras instâncias de texto na etapa seguinte. O overlapping

de rotações demandadas também deve ser levado em conta, se duas instâncias de texto precisam de uma mesma rotação, não será necessário realizar esta rotação duas vezes, apenas uma irá satisfazer a demanda de ambas as instâncias de textos.

Foram estipulados apenas três tipos de rotação para este trabalho, cada uma destas rotações consegue cobrir a região de quatro das doze arestas de um cubemap (Figura 22), cobrindo todas as possíveis regiões onde uma instância de texto pode ter sido partida em duas.

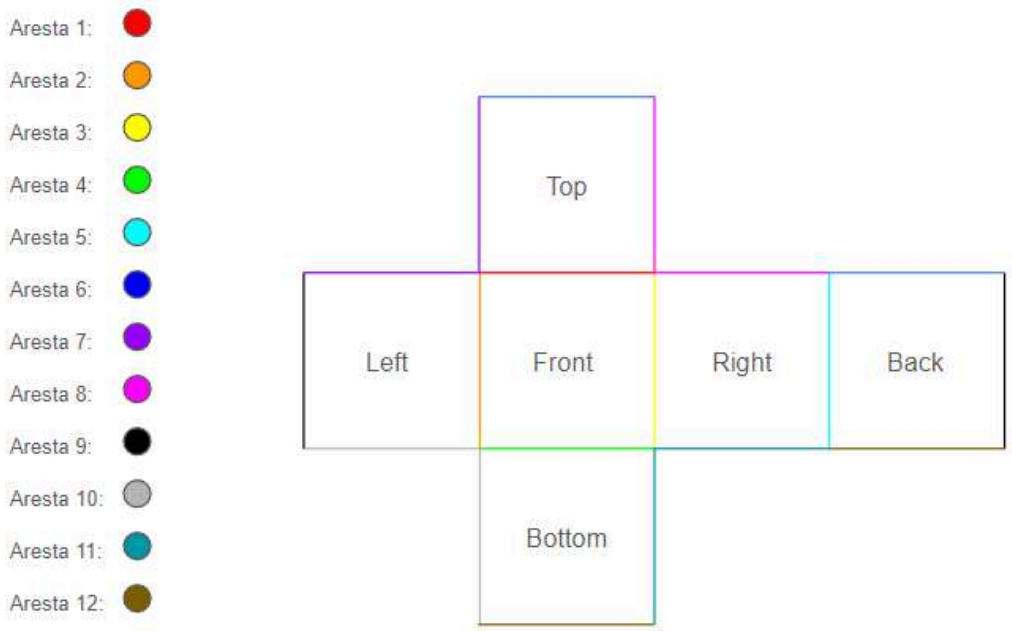

Figura 22. Arestas de um Cubemap.

Para simplificar, os três tipos de rotação presentes neste trabalho serão cada um referidos como R1, R2 e R3. Referindo-se à figura 22 acima, a R1 cobre as arestas 2, 3, 5 e 9. A R2 cobre as arestas 1, 4, 6 e 12. E a R3 cobre as arestas 7, 8, 10 e 11. Portanto, durante esta etapa, no máximo serão solicitadas três rotações para a etapa seguinte. No caso, se uma instância de texto está próxima da aresta 1 e outra da aresta 4, como ambas as arestas são cobertas pela R1, apenas uma solicitação de rotação precisa ser feita para diminuir o tempo de processamento da imagem, vista que, mesmo uma rotação só aumenta por volta de um minuto o tempo para finalizar o processamento de um conjunto de imagens.

9.4 Etapa de rotação

Para esta etapa será utilizada a informação de quais rotações são necessárias para ser rodado o OCR novamente na imagem de streetview rotacionada, mas a região onde foi identificada a chance de ter um texto partido em dois estará no centro de uma das faces do cubemap. Ao rotacionar uma imagem de streetview as imagens de cubemap também serão rotacionadas, logo, as rotações precisam ser bem definidas para que as regiões das antigas arestas do cubemap fiquem no foco/centro da imagem após a rotação. Inerentemente, rodar o OCR pela segunda, terceira e quarta vez, mesmo após a rotação da imagem, retornará resultados repetidos entre todas as iterações, somente um pequeno número de novas instâncias de texto aparecem em cada instância depois da primeira, além de que instâncias de texto que foram totalmente e corretamente identificadas em iterações anteriores podem ser movidas para a região das novas arestas após a rotação da imagem, retornando mais instâncias de texto partidas em duas. Entretanto, neste trabalho foi estipulado que não devem ocorrer mais rotações do que foi previsto pela etapa anterior, principalmente para evitar um aumento maior ainda no tempo de processamento e geração de instância de texto já identificadas, como também para evitar do programa entrar em loop, caso sempre haja uma instância de texto próxima das bordas após cada rotação.

Figura 23. Face “front” e “right” do cubemap da primeira iteração

Figura 24. Face “front” do cubemap, após rotação da segunda iteração

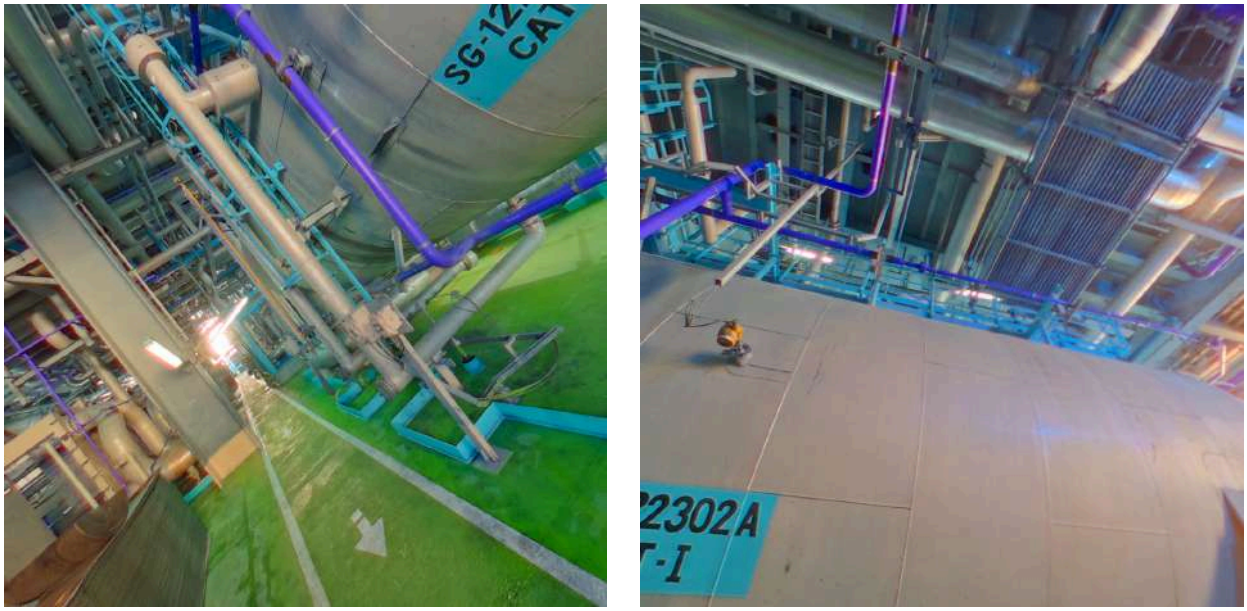

Figura 25. Faces “left” e “top” do cubemap, após rotação da terceira iteração

Como já foi dito anteriormente, as três rotações devem seguir um padrão fixo para maximizar a eficiência delas e evitar longos períodos de processamento para uma única imagem. É importante destacar novamente que a rotação é feita na imagem de streetview, não nas de cubemap, após rotacionar a streetview ela será decomposta nas suas imagens de cubemap. Nem todas as rotações ocorreram numa imagem, além do mais, dependendo de qual é a região de interesse nas imagens, nem todas as rotações propostas neste trabalho precisam ser implementadas, e outras rotações podem substituir as propostas neste trabalho dependendo da região das imagens de streetview se busca cobrir. As três rotações propostas neste trabalho têm o intuito de retificar o problema que surge ao decompor uma imagem de streetview em seis imagens de cubemap, todo texto que não se enquadra na situação problema descrita é assumido que o OCR conseguirá detectar e reconhecer corretamente.

A R1 deve rotacionar a imagem em 45º pelo eixo do “yaw”. A R2 deve rotacionar a imagem em 45º pelo eixo do “pitch”. E a R3 deve rotacionar a imagem primeiro em 90º pelo eixo do “yaw” depois em 45º pelo eixo do “pitch, caso a ordem dessas duas operações seja invertida a imagem resultante não será a mesma e não haverá garantia que ela cobre as arestas que a R3 deve cobrir.

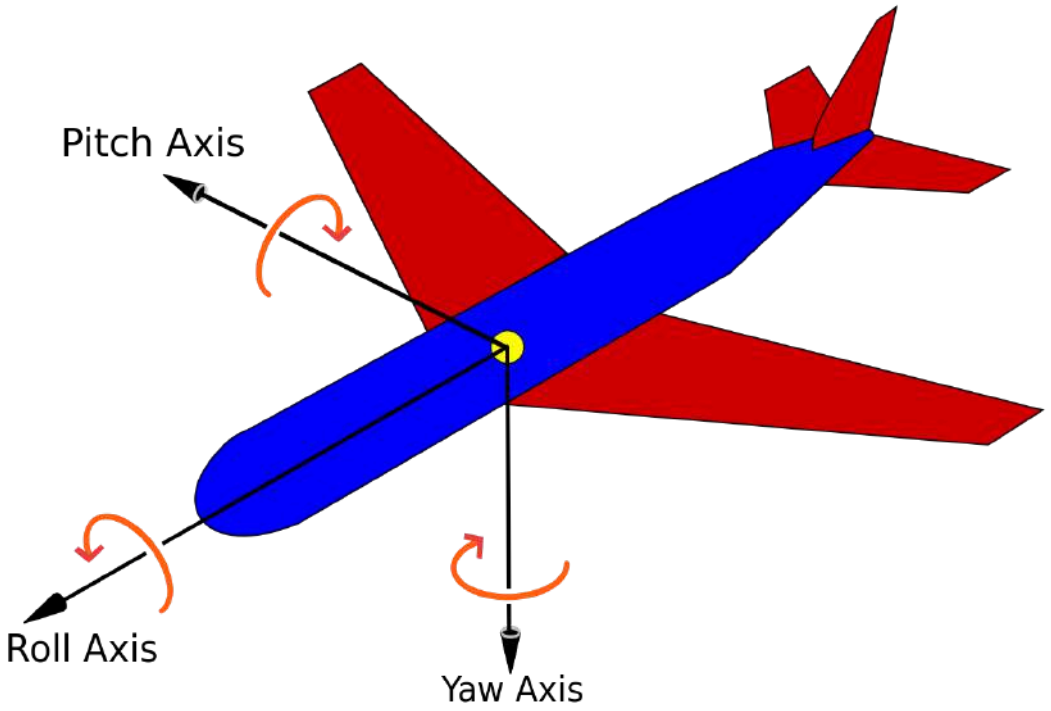

Figura 26. Eixos do roll, pitch e yaw

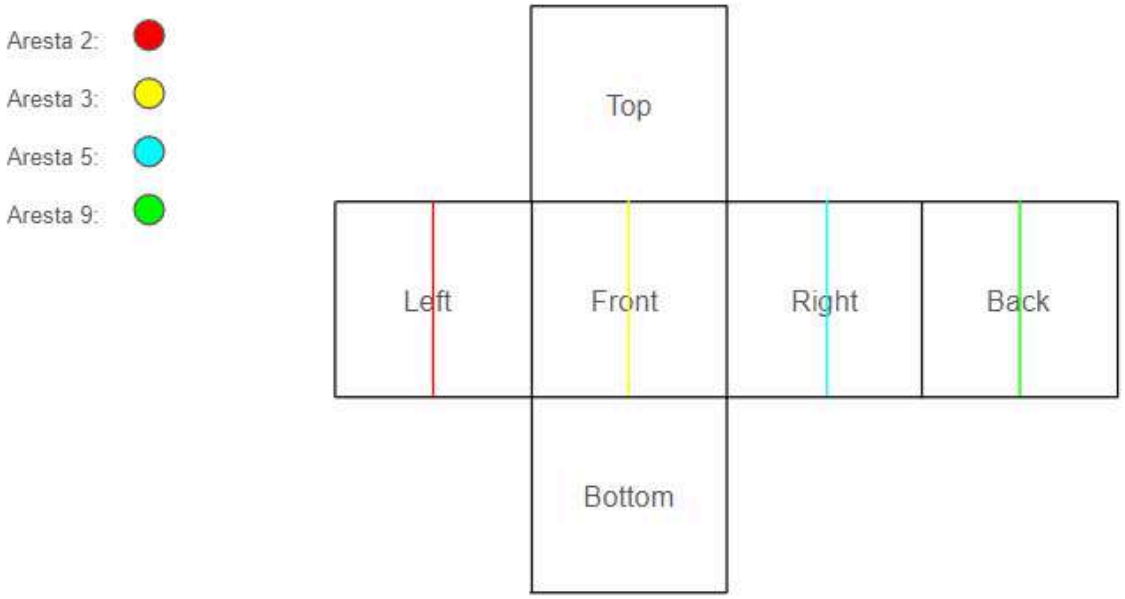

Figura 27. Cubemap após a R1, com a localização das arestas do cubemap original

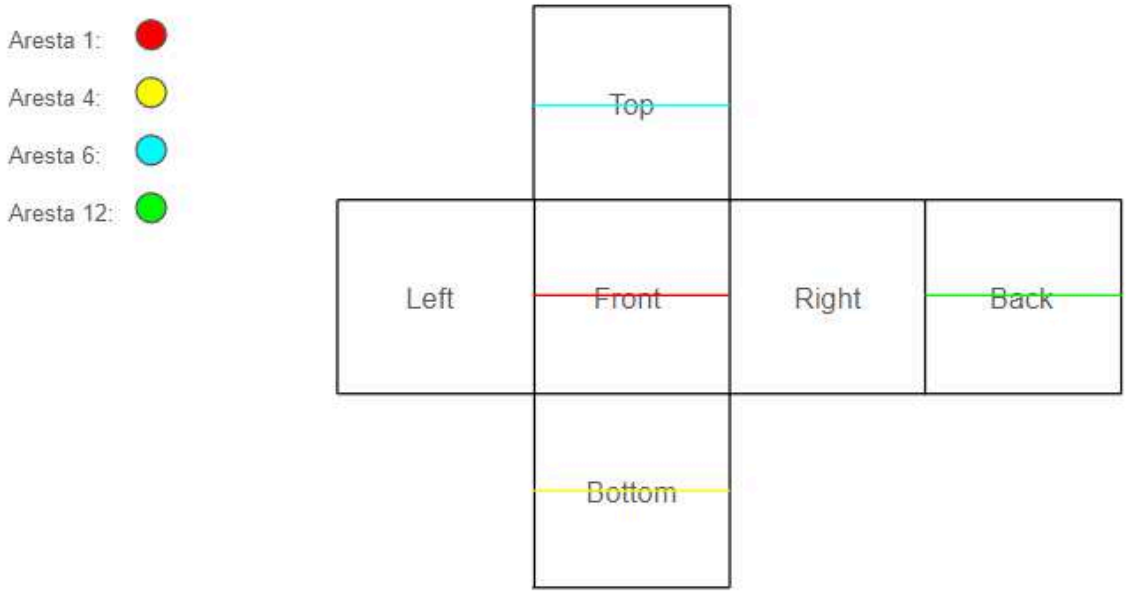

Figura 28. Cubemap após a R2, com a localização das arestas do cubemap original

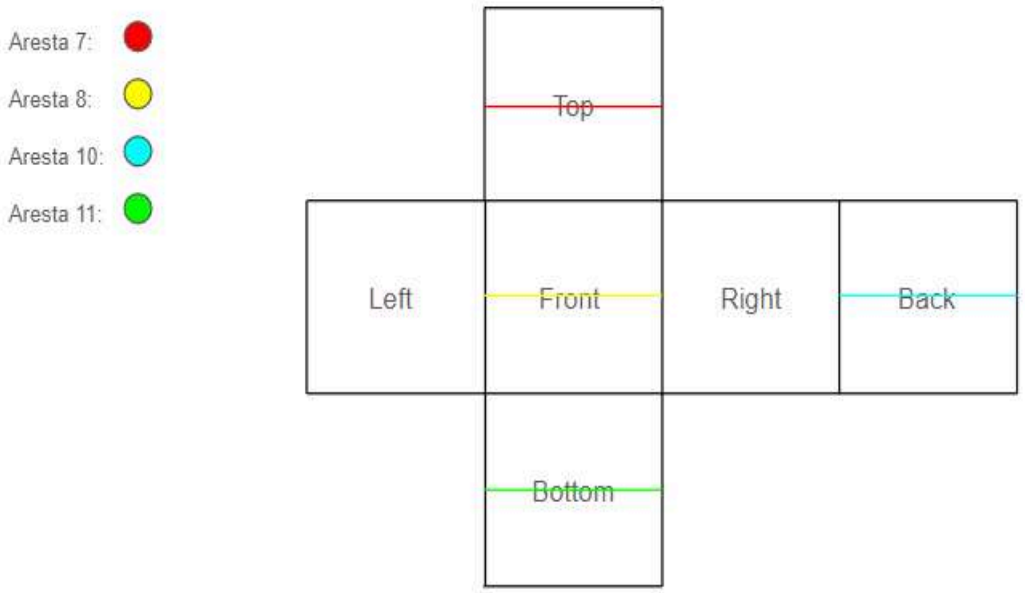

Figura 29. Cubemap após a R3, com a localização das arestas do cubemap original

Para realizar a função de rotacionar as imagens de streetview foi utilizada a biblioteca “pyEquirectRotate”. Uma características das imagens rotacionadas por esta biblioteca é que a imagem de retorno perde um fator de cor (como dá para ver na diferença entre as figuras 23 e 24), não foi estudado a fundo porque e como isto ocorre, mas, nos testes realizados esta questão não apresentou uma queda aparente na performance do OCR. Por não haver disponibilidade e/ou fácil acesso de outras ferramentas que têm a função de rotacionar imagens, foi decidido permanecer utilizando o “EquirectRotate” mesmo que exista algum problema não identificado com esse descoloramento da imagem. Novamente, qualquer outra ferramenta que realize as funções desejadas pode ser uma substituta viável.

Durante a codificação desta etapa, é preciso saber quais das rotações precisam ser feitas e executá-las sempre na imagem de streetview original. Após cada rotação o fluxo de execução é o mesmo que foi seguido nas primeiras etapas do pipeline, converter a imagem de streetview rotacionada num cubemap, passar estas imagens de cubemap pelo OCR e obter os resultados. A classe da biblioteca “pyEquirectRotate” que realiza a rotação é a “EquirectRotate” com a função “rotate()”. O parâmetro que a função “rotate()” precisa é a própria instância da imagem de streetview que será rotacionada, enquanto para instância um objeto da classe “EquirectRotate” é preciso passar as informações

de altura e largura da imagem a ser rotacionada e quantos graus a imagem será rotacionada pelo eixo do “yaw”, “pitch” e “roll” respectivamente. Para pegar a instância da imagem, assim como a informação de altura e largura dela pode-se usar qualquer biblioteca ou função própria dedicada para tratar imagens, neste trabalho foi utilizada a biblioteca “cv2”, enquanto as informações dos graus de rotação para os três eixos é dado pelo tipo de rotação que será feito. Um último detalhe é, quando a imagem de streetview rotacionada for convertida nas seis faces do cubemap, a face “bottom” precisa ser espelhada e invertida (de cabeça para baixo) para os textos da imagem ficarem bem tratados. Este detalhe é especialmente importante para a R2 e R3, pois nelas a face “bottom” é um ponto de interesse por ela cobrir uma aresta onde possivelmente uma instância de texto atravessa.

9.5 Resultados dos testes

Os testes mostraram os resultados esperados, uma forte melhora na identificação de textos comparado ao passar imagens de streetview diretamente no OCR. Pequenos grupos de por volta de dez imagens foram testados por vez e após uma verificação física nas imagens de cubemap geradas pode-se verificar que muitas das instâncias de texto que não eram identificadas na imagem de streetview eram identificadas pela imagem cubemap da região onde o texto estava. O tempo de processamento ficou por volta de um minuto para dez imagens quando não havia qualquer rotação numa imagem de streetview, mas cada rotação realizada aumentava o tempo de processamento por volta de um minuto. Foram poucos os casos que uma imagem precisou de uma rotação e menos ainda os casos de precisar de duas ou três rotações, mas estes casos dependem muito do dataset sendo usado e da posição de cada texto na imagem. No geral, mesmo com um tempo de processamento bem maior, o ganho na performance extremamente significativo e pouquíssimas foram as instâncias de texto não identificadas durante todo o processo do pipeline.

10. Conclusão

Utilizar imagens de streetview diretamente com OCR não retornará resultados bons, é preciso de algum pré-processamento da imagem para

minimizar ou neutralizar os fatores das imagens de streetview que interferem com a performance do OCR. Idealmente, todas as funções do pipeline proposto neste trabalho já poderiam ser integradas num OCR, mas até os OCR's evoluírem para tratar imagens de streetview com eficiência é um trabalho muito maior.

Muitas das propostas deste trabalho servem mais como sugestões, passos que podem ou não ser seguidos, dependendo do contexto do dataset de imagens sendo trabalhado e dos objetivos estipulados para utilizar OCR com imagens de streetview. As etapas de checar bordas e rotação podem ser desconsideradas se foram julgadas muito custosas pelos ganhos que elas retornam ou se existirem alternativas melhores para solucionar o problema de textos partidos ao meio pelas arestas do cubemap. Até algumas faces do cubemap podem ser desconsideradas quando passadas no OCR, em muitos contextos as faces "top" e "bottom" das imagens de streetview dão apenas visão do teto/céu e do chão respectivamente, podendo ser desconsideradas na primeira iteração do OCR, como também após uma rotação nem todas as faces do cubemap rotacionado têm informação das arestas de interesse, como pode ser visto nas figuras 27, 28 e 29. Assim, podem ser passadas no OCR apenas quatro faces do cubemap em cada iteração sem alguma perda significativa aparente dos resultados.

Existem poucos trabalhos que combinam imagens de streetview e OCR, este trabalho busca servir como um pontapé inicial para que o potencial desta combinação seja reconhecido e futuramente cresça para possibilitar métodos mais simples, rápidos e eficientes do método proposto pelo pipeline neste trabalho.

11. Referências bibliográficas

[1] Sargur N. Srihari, Ajay Shekhawat, and Stephen W. Lam. 2003. Optical character recognition (OCR). Encyclopedia of Computer Science. John Wiley and Sons Ltd., GBR, 1326–1333.

[2] Van Strien, D., Beelen, K., Ardanuy, M. C., Hosseini, K., McGillivray, B., & Colavizza, G. (2020). Assessing the impact of OCR quality on downstream NLP tasks.

[3] Chen, H., Liu, J., Li, M., Jiang, K., Xu, Z., Sun, R., & Sui, Y. (2023). Equirectangular image construction method for standard CNNs for Semantic Segmentation. *arXiv preprint arXiv:2310.09122*.

[4] Orji, E. Z., Haydar, A., Erşan, İ., & Mwambe, O. O. (2023). Advancing OCR Accuracy in Image-to-LaTeX Conversion—A Critical and Creative Exploration. *Applied Sciences*, *13*(22), 12503.

[5] Han, S. W., & Suh, D. Y. (2020). PIINET: A 360-degree panoramic image inpainting network using a cube map. *arXiv preprint arXiv:2010.16003*.