PIBIC-PIBITI/CNPq/INPE
RELATÓRIO TÉCNICO DE ATIVIDADES

Número do Processo Institucional:138924/2021-0

Número do Processo Individual: 111912/2022-9

Bolsista: Júlia Maria Santos Barroso

Orientador: Claúdio Clemente Faria Barbosa

Coorientador: (quando houver)

Área: Desenvolvimento de rotinas em ambiente PYTHON para o processamento de imagens ópticas na plataforma MAPAQUALI de monitoramento de sistemas aquáticos continentais por Sensoriamento Remoto.

Vigência original da bolsa: 01/09/2022 a 31/08/2023

Modalidade da bolsa: PIBIC

RELATÓRIO TÉCNICO

Resumo do Projeto

Iniciação científica no Laboratório de Instrumentação de Sistemas Aquáticos do INPE (http://www.dpi.inpe.br/labisa/ - LabISA), que desenvolvem pesquisas usando sensoriamento remoto, com a codificação de algoritmos em Python, com a finalidade de processar imagens de satélite na plataforma em desenvolvimento MAPAQUALI, com o objetivo de: 1)Desenvolver algoritmos para monitoramento de corpos d'água; 2) Estudo da estrutura e arquitetura da plataforma MAPAQUALI (GeoServer, rotinas Python, algoritmos de Machine Learning).

Objetivo

Implementar rotinas em Python, para processamento e tratamento de imagens (conversão, recorte e etc) para serem utilizadas no ambiente da plataforma de monitoramento e sistemas aquáticos por satélites MAPAQUALI. Codificar algoritmos em ambiente Anaconda para estimativa de indicadores de qualidade da água, produzindo gráficos para melhor visualização.

Atividades Desenvolvidas durante o período da bolsa

Foram exercidas as seguintes atividades em conjunto com os outros bolsistas de iniciação científica:

1)Realização do curso de Introdução à Programação para Sensoriamento Remoto;

2) Estudo das rotinas, bibliotecas e funções usadas nos projetos existentes;

3) Organização de dados e criação de jsons das pesquisas de campo no fillezila;

4) Codificação em Python de rotinas para recorte de imagem e plotagem de gráficos para comparação dos dados extraídos dos satélites.

```python
def copy(path_origin, path_destiny):
    # Verifica se o diretório de origem existe
    if not os.path.exists(path_origin):
        print(f"Diretório de origem '{path_origin}' não encontrado.")
        return

    # Verifica se o diretório de destino existe, caso contrário, cria
    if not os.path.exists(path_destiny):
        os.makedirs(path_destiny)

    try:
        xfdumanifest_origin = os.path.join(path_origin,'xfdumanifest.xml')
        xfdumanifest_destiny  = os.path.join(path_destiny, 'xfdumanifest.xml')
        shutil.copy2(xfdumanifest_origin, xfdumanifest_destiny)

        tie_geometries_origin = os.path.join(path_origin, 'tie_geometries.nc')
        tie_geometries_destiny  = os.path.join(path_destiny, ' tie_geometries.nc')
        shutil.copy2(tie_geometries_origin,  tie_geometries_destiny)

        geo_coordinates_origin = os.path.join(path_origin, 'geo_coordinates.nc')
        geo_coordinates_destiny  = os.path.join(path_destiny, 'geo_coordinates.nc')
        shutil.copy2(geo_coordinates_origin, geo_coordinates_destiny)

    except Exception as e:
        print(f"Ocorreu um erro durante a cópia dos arquivos: {str(e)}")
```

Figura 1 – Implementação de rotina que faz cópia de arquivos de uma pasta de origem para uma pasta de destino, ela foi adicionada a um conjunto de outras rotinas, contendo códigos para corte de regiões de interesse, conversão de valores de radiância e reflectância, em imagens de satélite.

```python
## Library:
import pandas as pd
import geopandas as gpd
from shapely.geometry import MultiPoint, Point
import ast
import os
import rasterio
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import *
import seaborn as sns

## Point in the pixel center:
full_data = gpd.read_file(r"./imagens/pontos/pontos.shp")
# points = ['Mutum_08_2019_Ponto_01','Mutum_08_2019_Ponto_02','Curape_08_2019_Ponto_01','Curape_08_2019_Ponto_02',
#           'Curape_08_2019_Ponto_03','Concordia_08_2019_Ponto_01','Concordia_08_2019_Ponto_02','Concordia_08_2019_Ponto_03',
#           'Concordia_08_2019_Ponto_04','Branco_08_2019_Ponto_01','Branco_08_2019_Ponto_02','Branco_08_2019_Ponto_03',
#           'Cerrado_08_2019_Ponto_01','Cerrado_08_2019_Ponto_02','Cerrado_08_2019_Ponto_03','Cerrado_08_2019_Ponto_04']

points = [i for i in range(0, 30)]

contribution = ['AWP_Method']
geometry = []
for point in points:
    point = full_data.loc[full_data['id'] == point]
    geometry_ = point['geometry']
    geometry.append(geometry_)

lista = [400, 412, 442, 490, 510, 560, 620, 665, 674, 681, 708, 753, 761, 764, 767, 778, 865, 885, 900, 940, 1020]

## Functions:

def _open_image_(path_image):
    return (rasterio.open(path_image))

def _extract_point_value_(image, ind, point_coors):
    ''' Extracts the pixel' values to the transect' points coordinates. '''

    # Extract:
    point_coors = point_coors.to_crs(epsg=4326)
    x = np.round((point_coors.x), 3)
    y = np.round((point_coors.y), 3)
    row, col = image.index(x, y)
    value = image.read(ind)[row, col]
    return ( value )

def _reflectance_retrieval_(path_image, point_coors):
    data_ = []
    open_image = _open_image_(path_image)
    for band in range(1,22):
        extract_point_value = _extract_point_value_(open_image, band, point_coors)
        x = pd.DataFrame(extract_point_value).transpose()
        data_.append(x)
    return (data_)
```

Figura 2 – Rotina de plotagem de gráficos para comparação de dados de outras duas rotinas trabalhadas. Cada rotina faz importação de diversas bibliotecas necessárias, nesse caso são usadas bibliotecas de geoprocessamento e plotagem de gráficos, essa rotina lê um arquivo shapefile com dados de pontos geográficos, define uma lista de pontos de interesse e valores de banda, a biblioteca rasterio abre a imagem raster e o código possui funções para extrair os valores das coordenadas dos pontos e extrair os valores de reflectância das bandas.

```
    return (data_)

def _ds_merge_(data_, aod_type, process_type, proporcion, point):
    bands = [i for i in range(1, 22)]
    tapa = data_[0].transpose()
    for band in range(1,20):
        tapa = tapa.merge(data_[band].transpose(), left_index=True, right_index=True)
    tapaf = tapa.merge(data_[20].transpose(), left_index=True, right_index=True).set_axis(bands, axis='columns')
    # Adjust:
    transpose_ = tapaf.transpose()
    filter_0 = transpose_.set_axis(['Reflectance'], axis=1, inplace=False)
    filter_0['band'] = filter_0.index
    filter_0['point'] = point
    filter_0['lista'] = lista
    output = filter_0.set_index(pd.DataFrame({'id': [j for j in 21 * [0, ]]})['id'])
    return ( output )

## Process:
path_main = r"C:\Users\kauar\Downloads\mosaico_rrs_atmospy.tif"

points = [str(i) for i in range(0, 30)]

# Surface Reflectance WITH AE -> p_with_AE:
list_p_with_AE = []
for point, name_point in zip(geometry, points):
    # path_image_01 = path_main
    p_retrieved_01 = _reflectance_retrieval_(path_image=path_main, point_coors=point)
    merge_01 = _ds_merge_(data_=p_retrieved_01,
                          aod_type='Default',
                          process_type='p_with_AE',
                          proporcion=name_point[:-30],
                          point=name_point)
    list_p_with_AE.append(merge_01)
p_with_AE = pd.concat(list_p_with_AE)
p_with_AE.index = [i for i in range(0, len(p_with_AE.index))]

path_main_ics = r"./imagens/outputimg.tif"

# # Surface Reflectance WITH AE -> p_with_AE:
list_p_with_AE_ics = []
for point, name_point in zip(geometry, points):
    # path_image_01 = path_main_ics
    p_retrieved_02 = _reflectance_retrieval_(path_image=path_main_ics, point_coors=point)
    merge_02 = _ds_merge_(data_=p_retrieved_02,
                          aod_type='Default',
                          process_type='p_with_AE',
                          proporcion=name_point[:-30],
                          point=name_point)
    list_p_with_AE_ics.append(merge_02)
p_with_AE_ics = pd.concat(list_p_with_AE_ics)
p_with_AE_ics.index = [i for i in range(0, len(p_with_AE_ics.index))]
```

Figura 3 – Continuação da rotina. Nesse trecho os valores de reflectância das bandas são mesclados em um dataframe e os resultados são armazenados em uma lista, o processo é realizado para as duas imagens e os resultados são concatenados em um dataframe, resultando em duas listas para plotagem de gráficos.

```
p_with_AE = pd.concat(list_p_with_AE)
p_with_AE.index = [i for i in range(0, len(p_with_AE.index))]

path_main_ics = r"./imagens/outputimg.tif"

# # Surface Reflectance WITH AE -> p_with_AE:
list_p_with_AE_ics = []
for point, name_point in zip(geometry, points):
    # path_image_01 = path_main_ics
    p_retrieved_02 = _reflectance_retrieval_(path_image=path_main_ics, point_coors=point)
    merge_02 = _ds_merge_(data_=p_retrieved_02,
                          aod_type='Default',
                          process_type='p_with_AE',
                          proporcion=name_point[:-30],
                          point=name_point)
    list_p_with_AE_ics.append(merge_02)
p_with_AE_ics = pd.concat(list_p_with_AE_ics)
p_with_AE_ics.index = [i for i in range(0, len(p_with_AE_ics.index))]

# Output:
list_per_point = []
# p_with_AE:
p_with_AE_00 = pd.DataFrame({'point': p_with_AE['point'],
                             'band': p_with_AE['band'],
                             'mosaico_rrs_atmospy00': p_with_AE['Reflectance'],
                             'outputimg': p_with_AE_ics['Reflectance'],
                             'lista': p_with_AE['lista']
                             })
list_per_point.append(p_with_AE_00)

# Append:
error = pd.concat(list_per_point)
error.to_csv(r'./results/JURUA_Error_AWP_METHOD.txt', sep='\t')

# ## Data:
x = []
for i in range(0, 10):
    for j in range(0, 3):
        x.append(i)
y = []
for i in range(0, 10):
    for j in range(0, 3):
        y.append(j)

fig, ax = plt.subplots(ncols=3, nrows=10, figsize=(15, 28))
fig.tight_layout()
# constrained_layout = True
# palette = ['#249225', '#104E79', '#40715F', '#BCAA86']
for i in range(0, 30):
    ax[x[i], y[i]].set_title(i, y=0.9)
    ax[0 + x[i], 0 + y[i]].set_ylim((-0.006, 0.03))
    filter = error.loc[error['point'] == str(i)]
    print(filter)
    sns.pointplot(data=filter, x="lista", y="mosaico_rrs_atmospy00", color="red", ax=ax[x[i], y[i]], markers="")
    sns.pointplot(data=filter, x="lista", y="outputimg", color="blue", ax=ax[x[i], y[i]], markers="")
    ax[x[i], y[i]].set_xticklabels(['400' , '412', '442', '490', '510', '560', '620', '665', '674', '681', '708', '753', '761', '764', '767', '778', '865', '885', '900', '940', '1020'],rotation=45)
    ax[x[i], y[i]].tick_params(axis='both', which='major', labelsize=8)
    plt.setp(ax[x[i], y[i]].collections, alpha=1, sizes=[80])
    ax[x[i], y[i]].set_ylabel('RRS', fontsize=8)
    ax[x[i], y[i]].set_xlabel('bandas', fontsize=8)

fig.tight_layout()
plt.savefig('./results/graphics02.png', dpi=400, bbox_inches='tight')
```

Figura 4 – Parte final da rotina. É criado um subplot com várias figuras para gráficos, contendo as informações sobre os pontos de interesse.

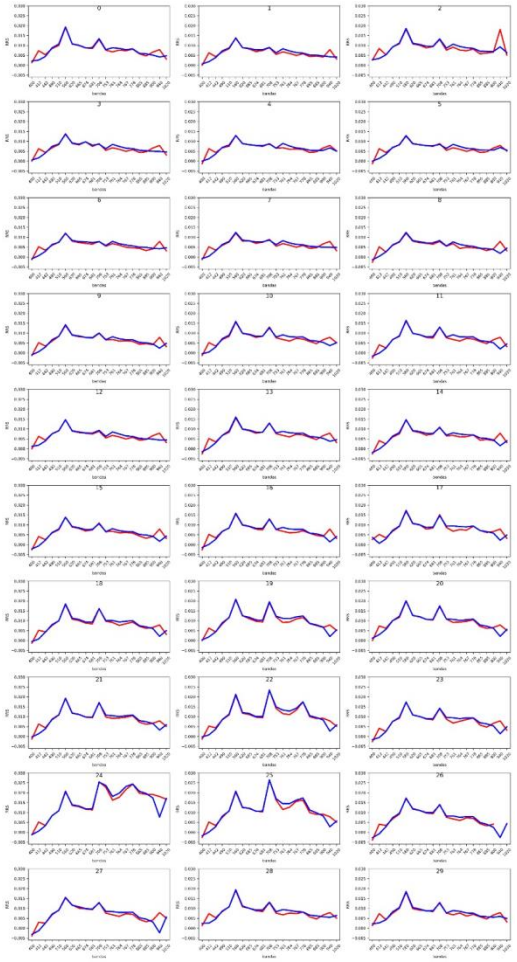


Figura 5 - Resultado da rotina. A saída é um arquivo de imagem com os dados de reflectância dos pontos geográficos de interesse das duas imagens raster para comparações.

1) Resultados Obtidos em função do Plano de Trabalho proposto

I. Aprendizado em Python para sensoriamento remoto;

II. Conhecimento de uso dos diferentes tipos de dados de sensoriamento remoto usados nas pesquisas científicas;

III. Aprendizado de codificação de dados científicos;

IV. Aprendizado e criação de rotinas para estimar indicadores de qualidade de água.

2) Conclusões Gerais

Durante a bolsa de iniciação científica, tive a oportunidade de me envolver em diversas atividades relacionadas ao sensoriamento remoto e programação. Inicialmente, participei de um curso de Introdução à Programação para Sensoriamento Remoto, que me proporcionou uma base para entender os princípios e conceitos fundamentais nessa área.

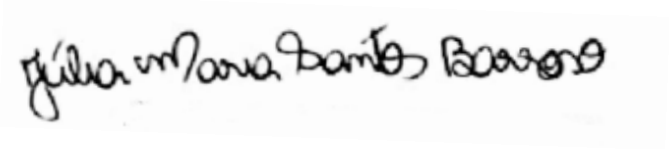
Um dos pontos mais relevantes da bolsa foi o estudo das rotinas, bibliotecas e funções utilizadas nos projetos existentes. Através dessa imersão, pude compreender como aplicar o conhecimento adquirido durante o curso e utilizar as ferramentas disponíveis para solucionar problemas reais. Essa experiência me permitiu expandir minha compreensão sobre a programação aplicada ao sensoriamento remoto, assim como aprender boas práticas.

Outra atividade importante foi a organização de dados e a criação de JSONs a partir das pesquisas de campo, utilizando o FileZilla, pois percebi a importância da organização e do armazenamento adequado dos dados, garantindo sua integridade e disponibilidade para análises futuras.

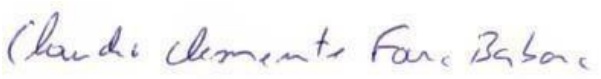
Também realizei em conjunto com os outros bolsistas a codificação em Python de rotinas para recorte de imagem e plotagem de gráficos, com o objetivo de comparar os dados extraídos dos satélites. Essa parte do trabalho exigiu um maior estudo para compreensão, pois envolvia o uso de bibliotecas específicas e a aplicação de algoritmos complexos.

Em suma, a bolsa de iniciação científica foi uma oportunidade valiosa de aprendizado e crescimento profissional. Através das atividades realizadas, pude aprofundar meu conhecimento em programação para sensoriamento remoto, adquirir novas habilidades técnicas e fortalecer minha capacidade de resolver problemas complexos com trabalho em equipe.

07, de junho de 2023



Bolsista: (colocar nome completo e assinar)



Orientador(a): (colocar nome completo e assinar)