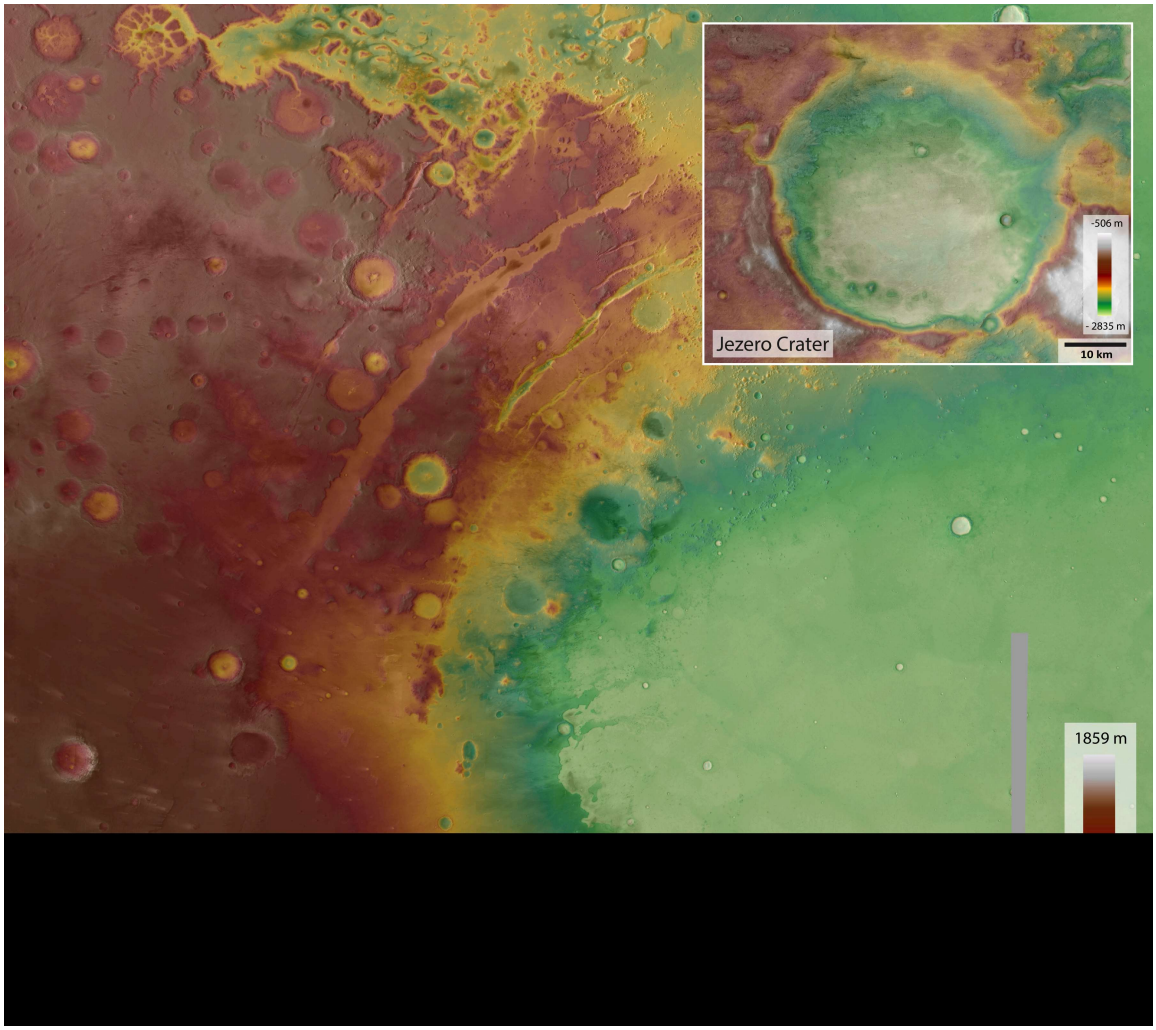
Background

The data used here is from This reference from Astrogeology Gov

- The image that is used is "Mars 2020 Terrain Relative Navigation CTX DTM Mosaic"
- The image is a 2D array of 2048x1940 pixels, each pixel is a 8 bit int value representing the elevation of the terrain
- The image Resulition is **20 meter/pixel**
- Max elevation in the iamge is **-506m** and the minimum is **-2835m**



Summary of the Mars Landing Site Selection Model

The goal of the Mars Landing Site Selection Model is to identify the optimal landing spot on Mars using a computer vision approach, similar to NASA's terrain evaluation methods. The model uses a heightmap image of the Martian surface, where each pixel represents an elevation value, and divides the terrain into smaller, manageable tiles to evaluate the suitability of each tile for landing. Here's a detailed breakdown of the approach:

1. **Loading the Terrain Data**:
    - The heightmap image is loaded in grayscale using OpenCV, where pixel intensity represents the elevation.
2. **Tiling the Terrain Map**:
    - The heightmap is divided into smaller, equally-sized tiles. Each tile represents a fixed area of the Martian surface (e.g., 100x100 pixels).
3. **Calculating Tile Statistics**:
    - For each tile, statistical measures such as the average height, maximum height, minimum height, and standard deviation of heights are computed. The standard deviation is particularly important as it indicates the roughness of the terrain within the tile.
4. **Visualizing Tile Suitability**:
    - The standard deviation values are normalized and mapped to a color scale using a colormap. Green indicates lower standard deviation (smoother, more suitable for landing), while red indicates higher standard deviation (rougher, less suitable for landing).
    - A colored overlay based on the standard deviation is created and blended with the original heightmap image using `matplotlib` to visualize the suitability of each tile.
5. **Decision-Making Algorithm**:
    - Although not implemented in the code provided, the next step would be to develop an algorithm that scores each tile based on its statistics and selects the best landing spot. The algorithm would prioritize tiles with lower roughness (lower standard deviation) and consider additional factors such as proximity to scientific points of interest.

Benefits of the Approach

- **Granular Analysis**: By dividing the heightmap into smaller tiles, the model allows for a detailed, granular analysis of the terrain, ensuring that each potential landing spot is carefully evaluated.
- **Visual Representation**: The use of a colored overlay provides an intuitive visual representation of terrain suitability, making it easier to identify optimal landing spots.
- **Statistical Rigor**: Using statistical measures such as standard deviation helps quantify the roughness of the terrain, which is crucial for safe landing.

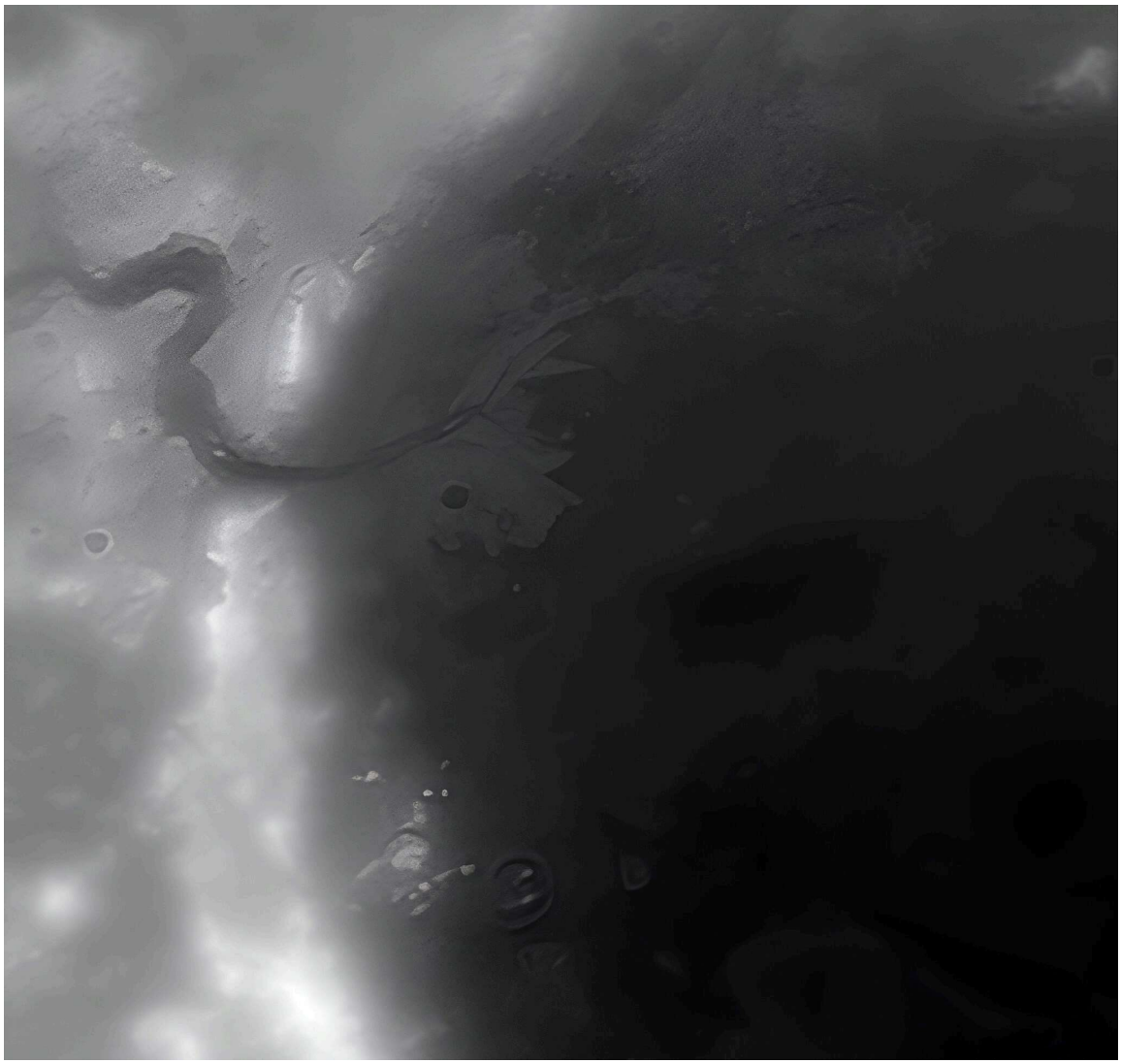
Potential Improvements

- **Enhanced Features**: Incorporate additional features such as slope, presence of obstacles, and proximity to scientific interest points to refine the selection criteria.

- **Machine Learning**: Use machine learning algorithms to predict landing suitability based on a larger set of terrain features and historical landing data.
- **Dynamic Adjustments**: Implement dynamic tile sizing to adjust the granularity of the analysis based on the specific region's terrain complexity.

This approach provides a structured, data-driven method for identifying the best landing spots on Mars, ensuring a safe and scientifically valuable landing mission.

Elevation Map



Idea 1

```
1- Make a simple tiling algorithm that
2- Find the height of each point on the map
3- Build a fromula to figure out if the tile is good place for
landing
```

4- Build a fromula to give each tile a score based on the difficulty of landing
5- Build an algorithm to take decision which is the closest best spot to land

---

1- Make a simple tiling algorithm that
2- Find the height of each point on the map

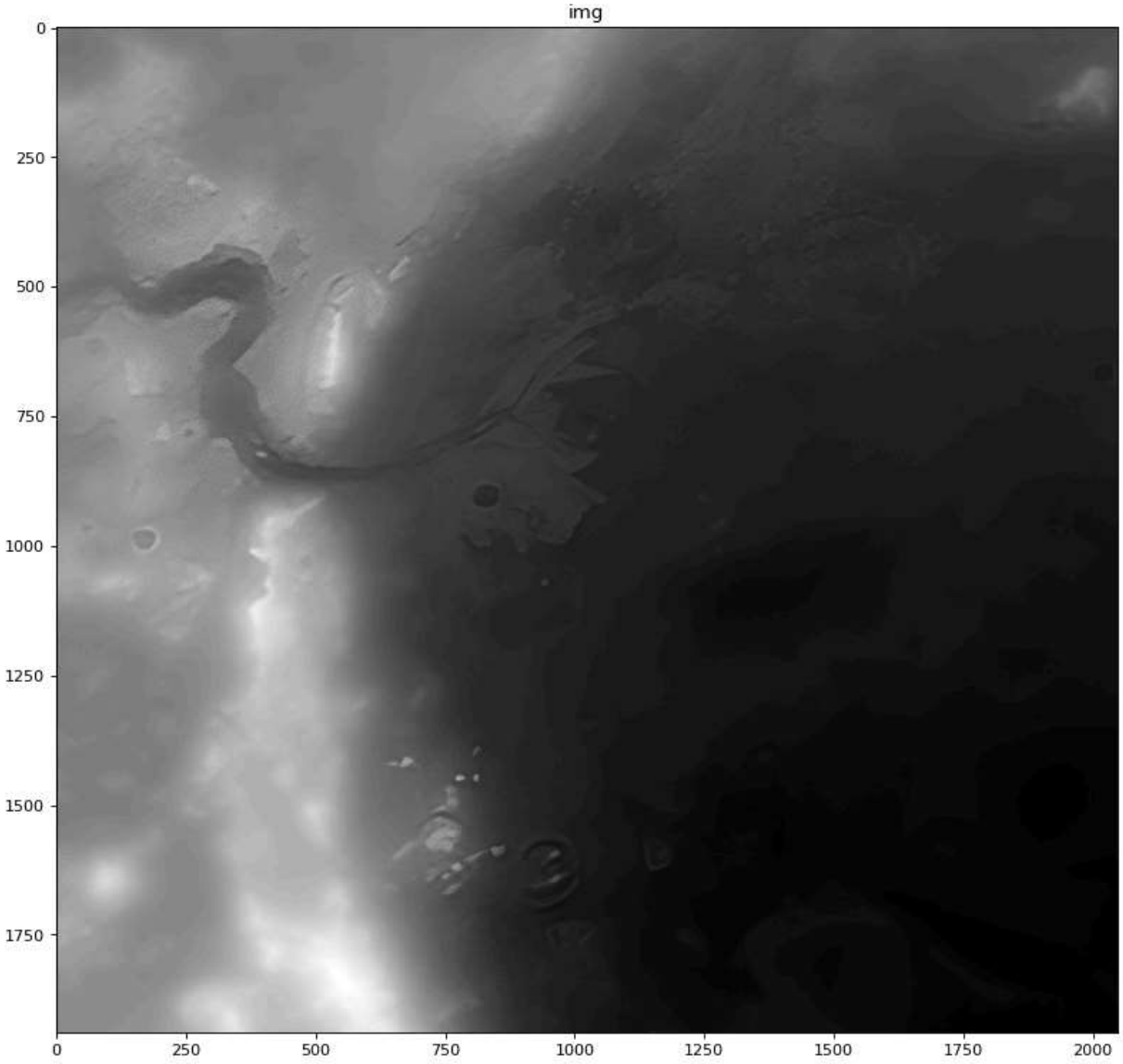
In [ ]:
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure



def load_image(file_path):
    # Load the image in grayscale (heightmap)
    image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
    return image



def tile_image(image, tile_size):
    rows, cols = image.shape
    tile_rows = rows // tile_size
    tile_cols = cols // tile_size

    tiles = []
    for i in range(tile_rows):
        for j in range(tile_cols):
            tile = image[i*tile_size:(i+1)*tile_size, j*tile_size:(j+1)*tile_size]
            tiles.append(tile)
    return tiles

def calculate_tile_statistics(tile):
    avg_height = np.mean(tile)
    max_height = np.max(tile)
    min_height = np.min(tile)
    stddev_height = np.std(tile)
    return avg_height, max_height, min_height, stddev_height

def main_calc(file_path, tile_size):
    image = load_image(file_path)
    tiles = tile_image(image, tile_size)

    tile_statistics = []
    for tile in tiles:
        stats = calculate_tile_statistics(tile)
        tile_statistics.append(stats)

    return tile_statistics
```

```python
def draw_tiles(image, tile_size):
    # Create a plot
    plt.figure(figsize=(10, 10))
    plt.imshow(image, cmap='gray')

    rows, cols = image.shape
    tile_rows = rows // tile_size
    tile_cols = cols // tile_size

    # Draw horizontal lines
    for i in range(1, tile_rows):
        plt.axhline(i * tile_size, color='red', linestyle='--')

    # Draw vertical lines
    for j in range(1, tile_cols):
        plt.axvline(j * tile_size, color='red', linestyle='--')

    # Display the plot
    plt.title(f"Tiling with tile size {tile_size}x{tile_size}")
    plt.show()

def main_draw(file_path, tile_size):
    image = load_image(file_path)
    draw_tiles(image, tile_size)



def showimg(img,typee= "none",size=(10, 10),title = "img",dd= ""):
    """good for dealing with different image types like RGB, BGR, GRAY... with some
    gist_gray = g
    jet = c
    tab20b = t
    viridis = v
    cividis = d
    BGR = bgr
    or leave it and will do defult
    """

    figure(figsize=size, dpi=80)
    try:
        if typee=="g":
            plt.imshow(img,cmap ='gist_gray')
        elif typee=="c" :
            plt.imshow(img,cmap ='jet')
        elif typee=="t" :
            plt.imshow(img,cmap ='tab20b')
        elif typee=="v" :
            plt.imshow(img,cmap ='viridis')
        elif typee=="d" :
            plt.imshow(img,cmap ='cividis')
        elif typee=="greens" :
            plt.imshow(img,cmap ='Greens')
        elif typee=="dd" :
            plt.imshow(img,cmap =dd)
        elif typee.upper() =="BGR":
            plt.imshow(img)
```

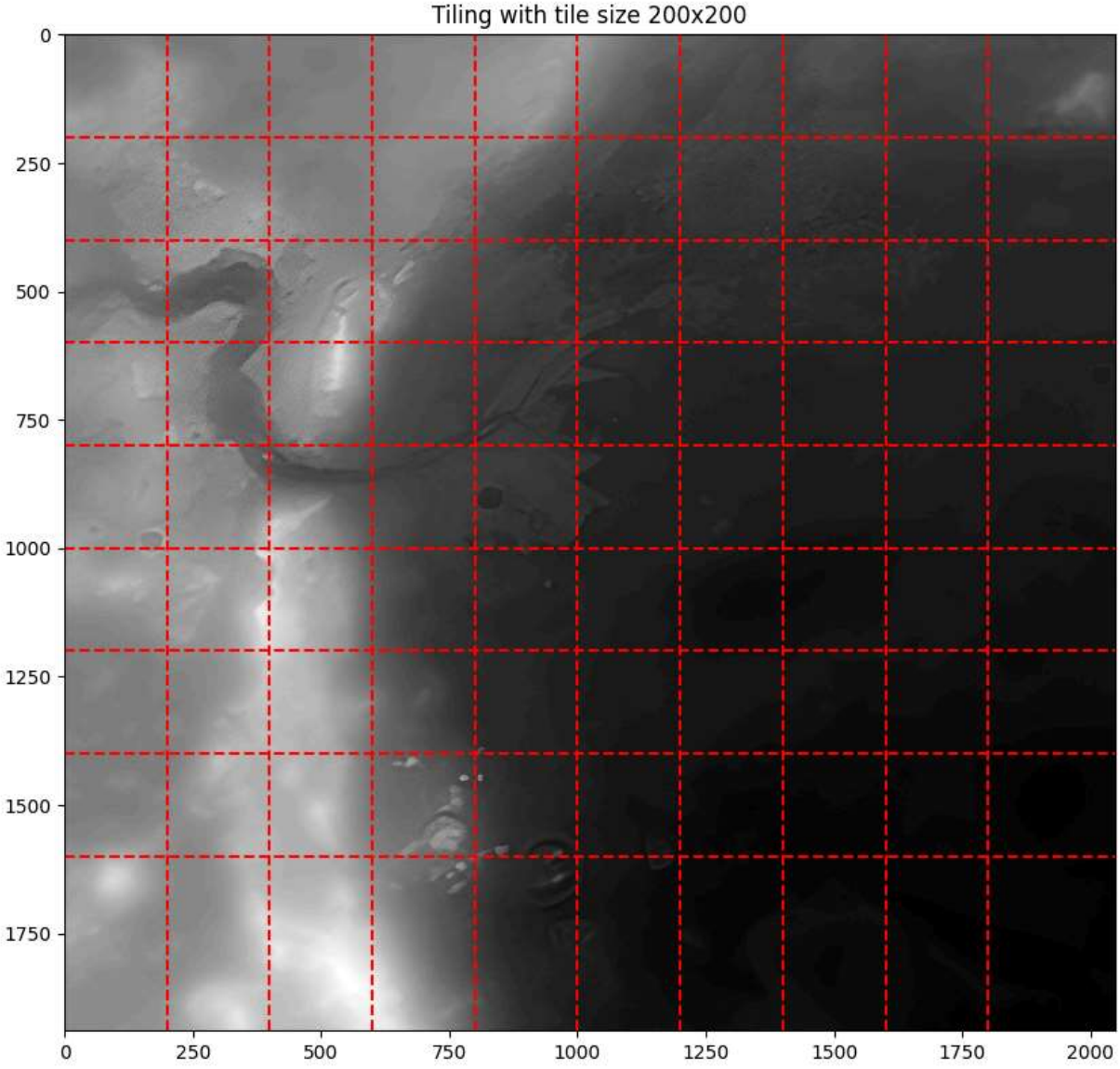
```
        else:
            plt.imshow(img[:,:,[2,1,0]])

    except:
        plt.imshow(img)


    legend_elements = [plt.Line2D([0], [0], marker='o', color='w', label='Best Land
                                  markerfacecolor='g', markersize=10),
                       plt.Line2D([0], [0], marker='o', color='w', label='Worst Lan
                                  markerfacecolor='y', markersize=10)]

    plt.legend(handles=legend_elements, loc='upper right')

    plt.title(title)
    plt.show()
```

Showing the image:

In [ ]:
```
file_path = 'data\JEZ_ctx_B_soc_008_DTM_MOLAtopography_DeltaGeoid_20m_Eqc_latTs0_lo

showimg(load_image(file_path),'g',size=(12,12))
```



In [ ]:
```
tile_size = 200
tile_statistics = main_calc(file_path, tile_size)


main_draw(file_path, tile_size)

# Print statistics for each tile
for idx, stats in enumerate(tile_statistics):
    print(f"Tile {idx+1}: Avg Height = {stats[0]}, Max Height = {stats[1]}, Min Hei
```


Tiling with tile size 200x200
0
250
500
750
1000
1250
1500
1750
0
250
500
750
1000
1250
1500
1750
2000

Tile 1: Avg Height = 142.040925, Max Height = 166, Min Height = 116, Std Dev = 14.26832681656735
Tile 2: Avg Height = 131.3493, Max Height = 162, Min Height = 118, Std Dev = 9.801315703006408
Tile 3: Avg Height = 127.70065, Max Height = 135, Min Height = 118, Std Dev = 2.9358371169906547
Tile 4: Avg Height = 138.599925, Max Height = 152, Min Height = 129, Std Dev = 4.557791679571918
Tile 5: Avg Height = 148.433225, Max Height = 169, Min Height = 100, Std Dev = 10.080463337534391
Tile 6: Avg Height = 100.085575, Max Height = 169, Min Height = 59, Std Dev = 21.941954149969757
Tile 7: Avg Height = 74.628025, Max Height = 103, Min Height = 49, Std Dev = 8.657257625794385
Tile 8: Avg Height = 71.525075, Max Height = 88, Min Height = 49, Std Dev = 7.078843213716136
Tile 9: Avg Height = 66.888325, Max Height = 85, Min Height = 52, Std Dev = 7.241201812846746
Tile 10: Avg Height = 75.3537, Max Height = 125, Min Height = 53, Std Dev = 10.95203845455265
Tile 11: Avg Height = 138.360225, Max Height = 188, Min Height = 99, Std Dev = 9.944300525897987
Tile 12: Avg Height = 143.55565, Max Height = 207, Min Height = 78, Std Dev = 12.885144666533629
Tile 13: Avg Height = 131.78985, Max Height = 174, Min Height = 94, Std Dev = 7.174171518544842
Tile 14: Avg Height = 130.57805, Max Height = 147, Min Height = 83, Std Dev = 10.886625197805792
Tile 15: Avg Height = 92.36725, Max Height = 152, Min Height = 46, Std Dev = 23.450097759974359
Tile 16: Avg Height = 61.111225, Max Height = 98, Min Height = 38, Std Dev = 10.510390287680805
Tile 17: Avg Height = 57.9323, Max Height = 96, Min Height = 34, Std Dev = 6.899138838289892
Tile 18: Avg Height = 53.664625, Max Height = 82, Min Height = 29, Std Dev = 7.09821094427145
Tile 19: Avg Height = 47.451025, Max Height = 74, Min Height = 36, Std Dev = 4.683177494968026
Tile 20: Avg Height = 47.0009, Max Height = 67, Min Height = 38, Std Dev = 5.274954899333264
Tile 21: Avg Height = 127.579425, Max Height = 198, Min Height = 74, Std Dev = 13.180978403342257
Tile 22: Avg Height = 118.32545, Max Height = 202, Min Height = 45, Std Dev = 25.608785256186987
Tile 23: Avg Height = 142.4024, Max Height = 236, Min Height = 28, Std Dev = 18.658958283891412
Tile 24: Avg Height = 104.048325, Max Height = 207, Min Height = 58, Std Dev = 25.70878331804862
Tile 25: Avg Height = 65.880975, Max Height = 95, Min Height = 41, Std Dev = 8.379985563792756
Tile 26: Avg Height = 49.1688, Max Height = 69, Min Height = 33, Std Dev = 4.876136437795808
Tile 27: Avg Height = 46.035325, Max Height = 77, Min Height = 35, Std Dev = 4.12183540966582
Tile 28: Avg Height = 43.3751, Max Height = 60, Min Height = 30, Std Dev = 3.784144551943014

Tile 29: Avg Height = 39.500825, Max Height = 60, Min Height = 32, Std Dev = 4.16698923917197
Tile 30: Avg Height = 37.270375, Max Height = 44, Min Height = 32, Std Dev = 1.6650142219737942
Tile 31: Avg Height = 152.576575, Max Height = 210, Min Height = 118, Std Dev = 10.301021127508427
Tile 32: Avg Height = 122.731275, Max Height = 203, Min Height = 62, Std Dev = 20.341862301037608
Tile 33: Avg Height = 145.253875, Max Height = 237, Min Height = 75, Std Dev = 28.328335152005934
Tile 34: Avg Height = 74.4848, Max Height = 139, Min Height = 43, Std Dev = 12.925930100383493
Tile 35: Avg Height = 56.89415, Max Height = 87, Min Height = 34, Std Dev = 6.3939186558400944
Tile 36: Avg Height = 41.69965, Max Height = 68, Min Height = 33, Std Dev = 5.427166837079914
Tile 37: Avg Height = 35.0849, Max Height = 42, Min Height = 30, Std Dev = 2.352305675289672
Tile 38: Avg Height = 32.81445, Max Height = 40, Min Height = 29, Std Dev = 2.2141750602651094
Tile 39: Avg Height = 32.14625, Max Height = 37, Min Height = 27, Std Dev = 1.6819217988658093
Tile 40: Avg Height = 30.5426, Max Height = 37, Min Height = 27, Std Dev = 1.8206688990588047
Tile 41: Avg Height = 149.10725, Max Height = 187, Min Height = 123, Std Dev = 8.697352898296124
Tile 42: Avg Height = 139.588875, Max Height = 213, Min Height = 74, Std Dev = 16.852711984555334
Tile 43: Avg Height = 113.4319, Max Height = 236, Min Height = 57, Std Dev = 35.912263339831562
Tile 44: Avg Height = 67.17355, Max Height = 94, Min Height = 40, Std Dev = 7.027028561027769
Tile 45: Avg Height = 53.739375, Max Height = 76, Min Height = 29, Std Dev = 6.168484385112359
Tile 46: Avg Height = 36.09, Max Height = 59, Min Height = 28, Std Dev = 4.937458860588106
Tile 47: Avg Height = 31.001, Max Height = 44, Min Height = 19, Std Dev = 2.47801311538095
Tile 48: Avg Height = 26.819475, Max Height = 31, Min Height = 18, Std Dev = 3.4833009810200153
Tile 49: Avg Height = 26.44645, Max Height = 31, Min Height = 19, Std Dev = 2.633748734693574
Tile 50: Avg Height = 23.678475, Max Height = 30, Min Height = 18, Std Dev = 2.909741685162963
Tile 51: Avg Height = 152.5636, Max Height = 192, Min Height = 126, Std Dev = 14.24339513739614
Tile 52: Avg Height = 168.55885, Max Height = 242, Min Height = 139, Std Dev = 19.296910029263753
Tile 53: Avg Height = 164.207525, Max Height = 232, Min Height = 88, Std Dev = 33.2208775978958
Tile 54: Avg Height = 64.567675, Max Height = 103, Min Height = 43, Std Dev = 14.143691883464339
Tile 55: Avg Height = 39.4839, Max Height = 77, Min Height = 29, Std Dev = 5.087778571243053
Tile 56: Avg Height = 29.78975, Max Height = 39, Min Height = 21, Std Dev = 3.243693101620435

Tile 57: Avg Height = 18.1824, Max Height = 36, Min Height = 12, Std Dev = 4.1909999
09329514
Tile 58: Avg Height = 17.545425, Max Height = 24, Min Height = 12, Std Dev = 2.08741
14518644857
Tile 59: Avg Height = 20.920975, Max Height = 25, Min Height = 17, Std Dev = 1.10757
8461949762
Tile 60: Avg Height = 18.821125, Max Height = 22, Min Height = 14, Std Dev = 1.79476
42559330739
Tile 61: Avg Height = 129.321175, Max Height = 143, Min Height = 124, Std Dev = 2.58
81695499667328
Tile 62: Avg Height = 157.8925, Max Height = 217, Min Height = 125, Std Dev = 22.046
175263523605
Tile 63: Avg Height = 166.727675, Max Height = 215, Min Height = 99, Std Dev = 24.89
5159451073514
Tile 64: Avg Height = 76.306875, Max Height = 132, Min Height = 43, Std Dev = 20.289
574976681376
Tile 65: Avg Height = 39.307225, Max Height = 109, Min Height = 29, Std Dev = 5.3848
89766687429
Tile 66: Avg Height = 26.38885, Max Height = 33, Min Height = 22, Std Dev = 2.380240
6763812773
Tile 67: Avg Height = 20.537425, Max Height = 25, Min Height = 12, Std Dev = 2.31361
17585660303
Tile 68: Avg Height = 17.1438, Max Height = 24, Min Height = 9, Std Dev = 3.51089327
09497167
Tile 69: Avg Height = 15.12345, Max Height = 22, Min Height = 10, Std Dev = 2.951831
651280269
Tile 70: Avg Height = 12.521075, Max Height = 17, Min Height = 6, Std Dev = 2.660968
9671950325
Tile 71: Avg Height = 134.274475, Max Height = 174, Min Height = 122, Std Dev = 9.45
1761130835617
Tile 72: Avg Height = 163.980625, Max Height = 192, Min Height = 127, Std Dev = 17.8
58205386022835
Tile 73: Avg Height = 170.639275, Max Height = 213, Min Height = 112, Std Dev = 20.8
07211549709752
Tile 74: Avg Height = 105.302375, Max Height = 200, Min Height = 49, Std Dev = 19.61
0378485877703
Tile 75: Avg Height = 44.679425, Max Height = 188, Min Height = 16, Std Dev = 13.484
752747802794
Tile 76: Avg Height = 22.978175, Max Height = 38, Min Height = 12, Std Dev = 3.95595
7364453641
Tile 77: Avg Height = 11.344025, Max Height = 24, Min Height = 3, Std Dev = 4.176574
170223127
Tile 78: Avg Height = 9.2048, Max Height = 15, Min Height = 4, Std Dev = 1.992311963
5237851
Tile 79: Avg Height = 9.74115, Max Height = 14, Min Height = 6, Std Dev = 1.41412399
65080856
Tile 80: Avg Height = 5.29295, Max Height = 9, Min Height = 2, Std Dev = 1.458554180
5157598
Tile 81: Avg Height = 162.71495, Max Height = 218, Min Height = 135, Std Dev = 17.79
9626302186795
Tile 82: Avg Height = 159.744175, Max Height = 226, Min Height = 133, Std Dev = 24.0
05767610500918
Tile 83: Avg Height = 198.2494, Max Height = 248, Min Height = 137, Std Dev = 22.596
051638284067
Tile 84: Avg Height = 113.32335, Max Height = 216, Min Height = 54, Std Dev = 31.032
274244365333

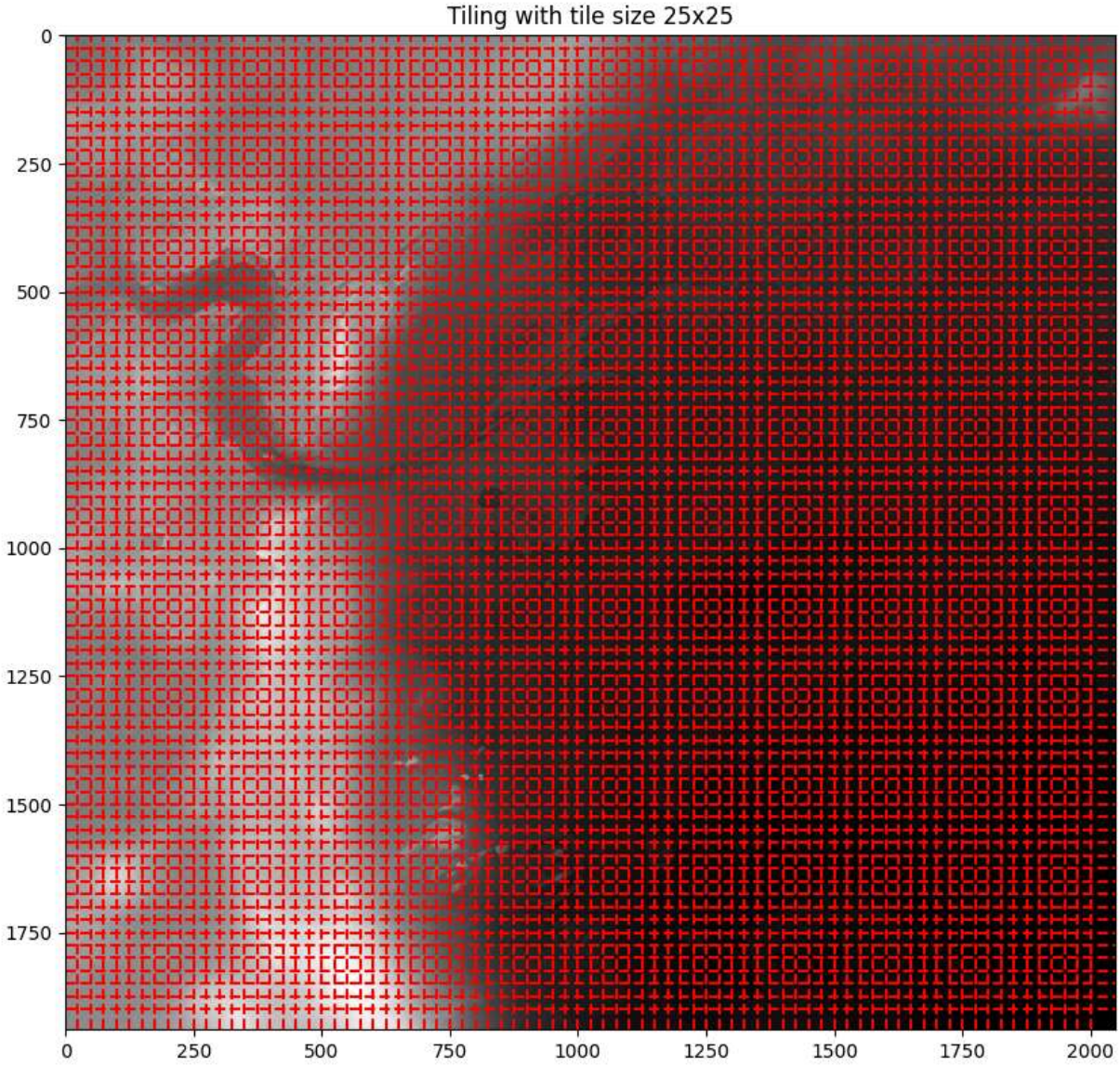
```
Tile 85: Avg Height = 40.7668, Max Height = 140, Min Height = 13, Std Dev = 12.17470
8118061805
Tile 86: Avg Height = 19.19775, Max Height = 49, Min Height = 10, Std Dev = 5.876639
765844083
Tile 87: Avg Height = 10.3448, Max Height = 23, Min Height = 5, Std Dev = 3.26348938
40795622
Tile 88: Avg Height = 10.8634, Max Height = 18, Min Height = 4, Std Dev = 3.17556143
69745704
Tile 89: Avg Height = 6.879525, Max Height = 17, Min Height = 1, Std Dev = 2.8428349
88945894
Tile 90: Avg Height = 5.55025, Max Height = 9, Min Height = 0, Std Dev = 2.074132333
6518334
```

Tile size = 200 meaning that we have 200 pixel * 20 meter/pixel = 4km each tile

The reasonable landing area for safe landing is **500m**, but the uncertainty of the landing spot will mean that we need more than one tile

this means that we need tile size of 500/20 = 25 pixels

In [ ]:
```
tile_size = 25  # Example tile size, this would mean each tile is 100x100 pixels
main_draw(file_path, tile_size)
```

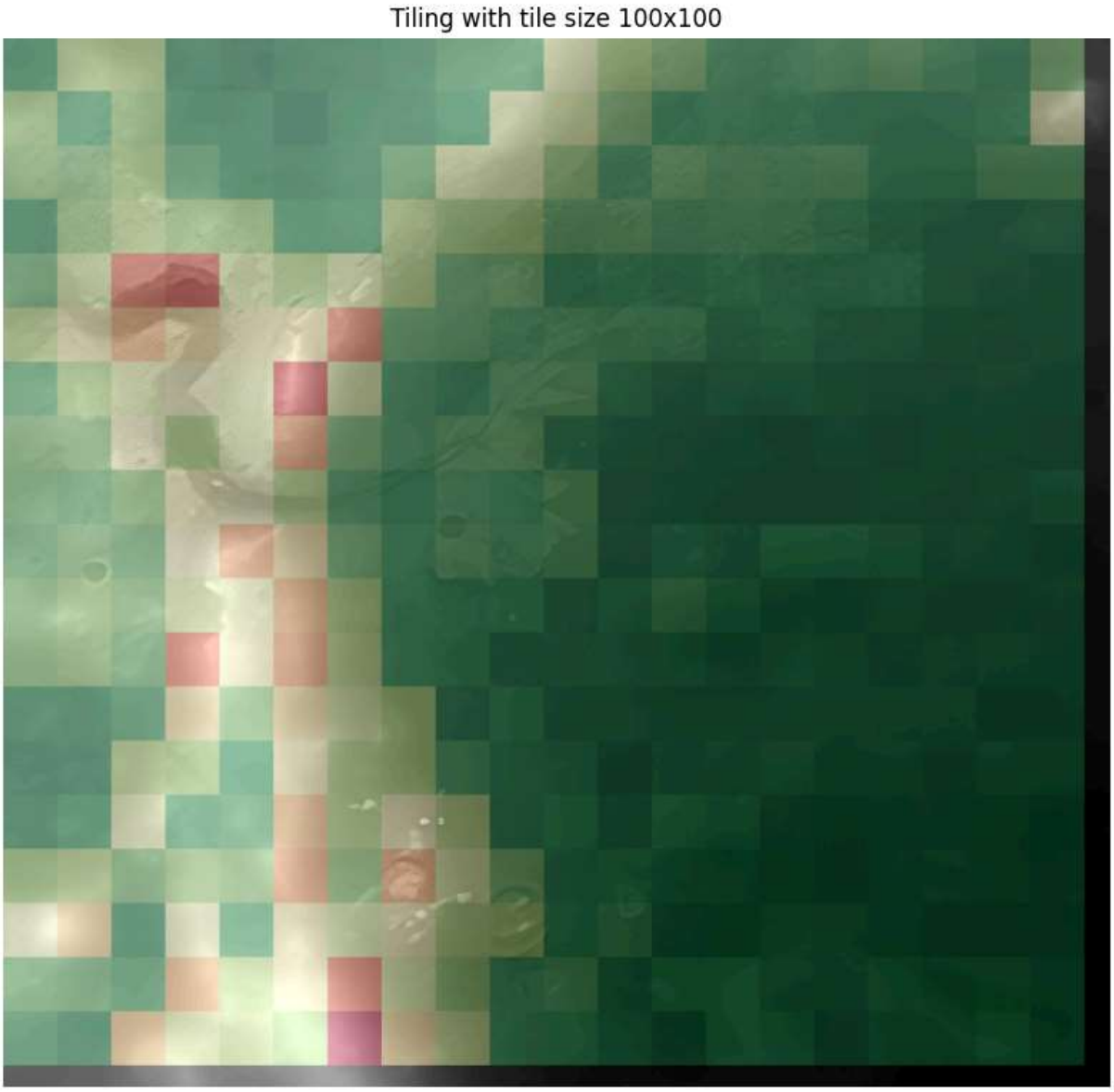


3- Build a fromula to figure out if the tile is good place for landing

In [ ]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

def load_image(file_path):
    # Load the image in grayscale (heightmap)
    image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
    return image

def tile_image(image, tile_size):
    rows, cols = image.shape
    tile_rows = rows // tile_size
    tile_cols = cols // tile_size
```

```
    tiles = []
    for i in range(tile_rows):
        for j in range(tile_cols):
            tile = image[i*tile_size:(i+1)*tile_size, j*tile_size:(j+1)*tile_size]
            tiles.append(tile)
    return tiles, tile_rows, tile_cols

def calculate_tile_statistics(tile):
    stddev_height = np.std(tile)
    return stddev_height

def create_colored_overlay(image, tile_size):
    tiles, tile_rows, tile_cols = tile_image(image, tile_size)
    stddevs = [calculate_tile_statistics(tile) for tile in tiles]

    # Normalize standard deviations to range [0, 1] for coloring
    norm = mcolors.Normalize(vmin=min(stddevs), vmax=max(stddevs))
    cmap = plt.get_cmap('RdYlGn_r')  # Red to green colormap

    # Create an overlay image
    overlay = np.zeros((image.shape[0], image.shape[1], 3), dtype=np.uint8)

    index = 0
    for i in range(tile_rows):
        for j in range(tile_cols):
            tile_std = stddevs[index]
            color = cmap(norm(tile_std))[:3]  # Get RGB color
            color = [int(c * 255) for c in color]  # Convert to 0-255 range

            overlay[i*tile_size:(i+1)*tile_size, j*tile_size:(j+1)*tile_size] = col
            index += 1

    return overlay

def main(file_path, tile_size,a1=0.7 ,a2= 0.4):
    image = load_image(file_path)
    overlay = create_colored_overlay(image, tile_size)

    # Blend the original image and the overlay
    blended = cv2.addWeighted(cv2.cvtColor(image, cv2.COLOR_GRAY2BGR),a1 , overlay,

    # Display the result
    plt.figure(figsize=(10, 10))
    plt.imshow(blended)
    plt.title(f"Tiling with tile size {tile_size}x{tile_size}")
    plt.axis('off')
    plt.show()



tile_size = 100
main(file_path, tile_size)
```



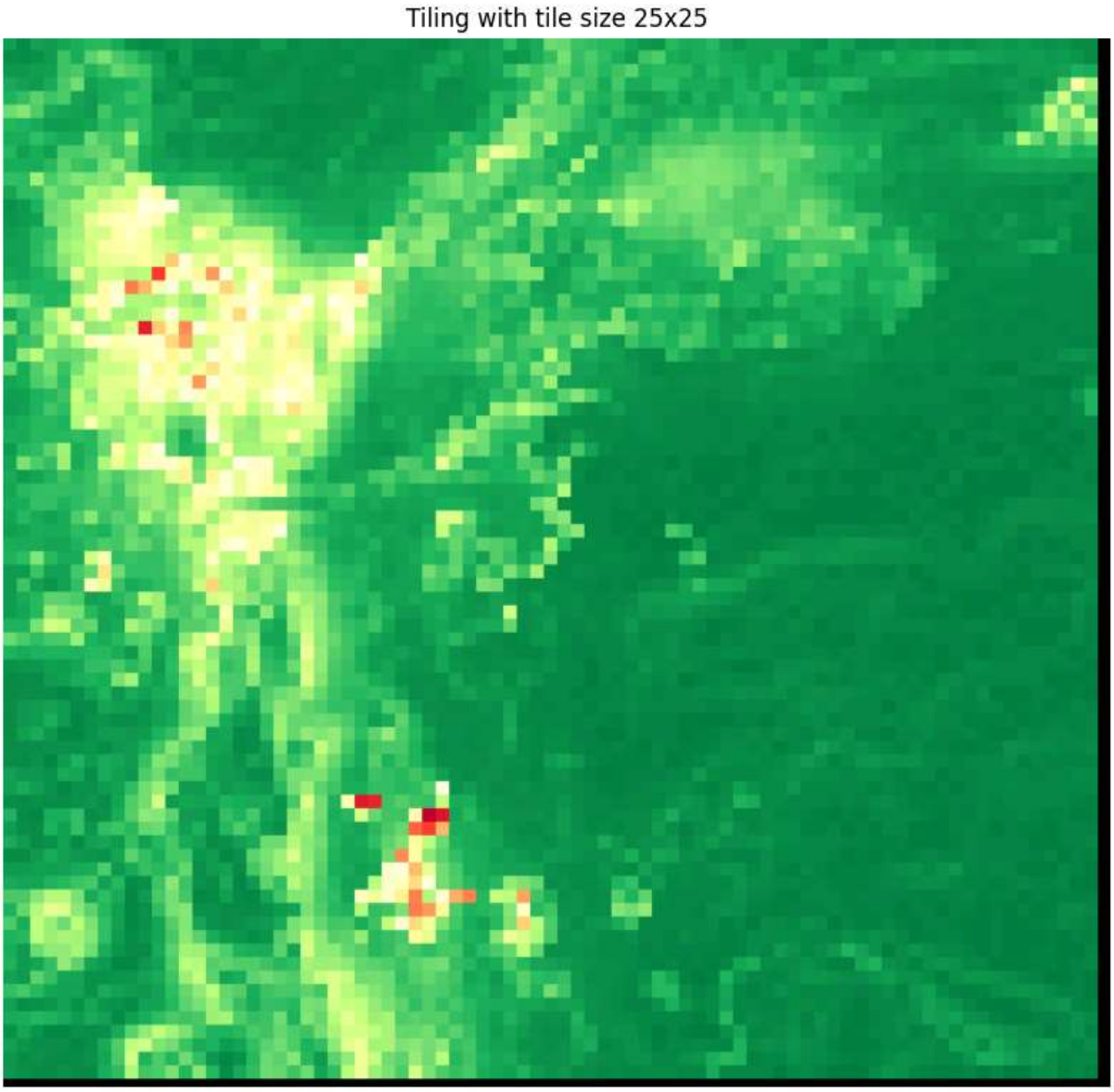
Any yellowish or redish tiles means that this area is bad for landing

With the right tile size

In [ ]:
```
tile_size = 25
main(file_path,tile_size)
```

To show the effect more, increasing the alpha of the effect

In [ ]:
```
tile_size = 25
main(file_path,tile_size,a1= 0,a2= 1.2)
```



In [ ]:

```
from matplotlib import pyplot as plt

plt.style.use('ggplot')
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.serif'] = 'Ubuntu'
plt.rcParams['font.monospace'] = 'Ubuntu Mono'
plt.rcParams['font.size'] = 14
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.labelweight'] = 'bold'
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
plt.rcParams['legend.fontsize'] = 12
plt.rcParams['figure.titlesize'] = 12
plt.rcParams['image.cmap'] = 'jet'
plt.rcParams['image.interpolation'] = 'none'
plt.rcParams['figure.figsize'] = (12, 10)
plt.rcParams['axes.grid']=True
```

```
plt.rcParams['lines.linewidth'] = 2
plt.rcParams['lines.markersize'] = 8
colors = ['xkcd:pale orange', 'xkcd:sea blue', 'xkcd:pale red', 'xkcd:sage green',
'xkcd:scarlet']
```

In [ ]:
```
image = cv2.imread("data\JEZ_ctx_B_soc_008_DTM_MOLAtopography_DeltaGeoid_20m_Eqc_la

tiles, tile_rows, tile_cols = tile_image(image, tile_size)
stddevs = [calculate_tile_statistics(tile) for tile in tiles]
```

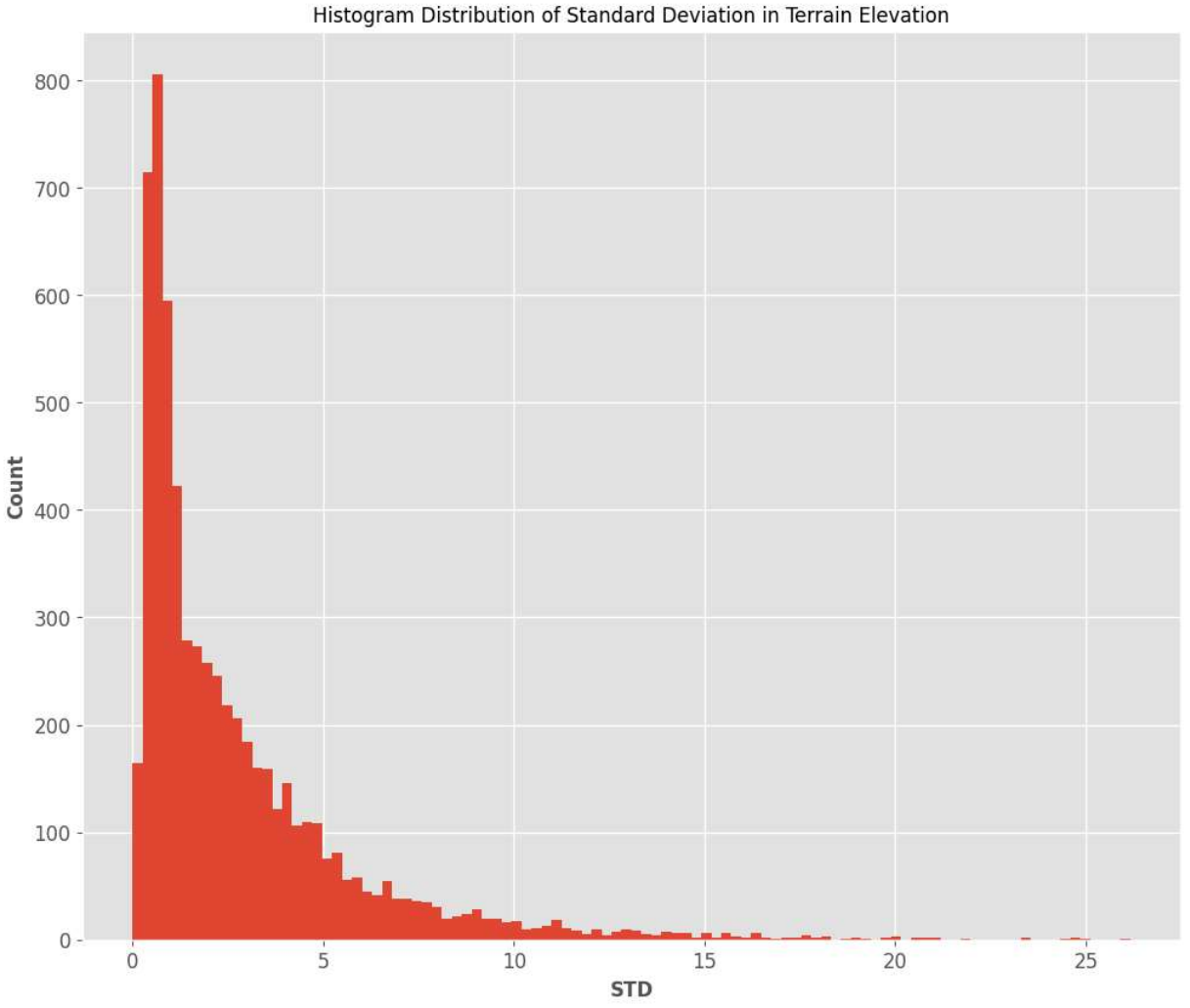
In [ ]:
```
len(stddevs)
```

Out[ ]:
```
6237
```

In [ ]:
```
stddevs_n =np.array(stddevs)
np.mean(stddevs_n)
```

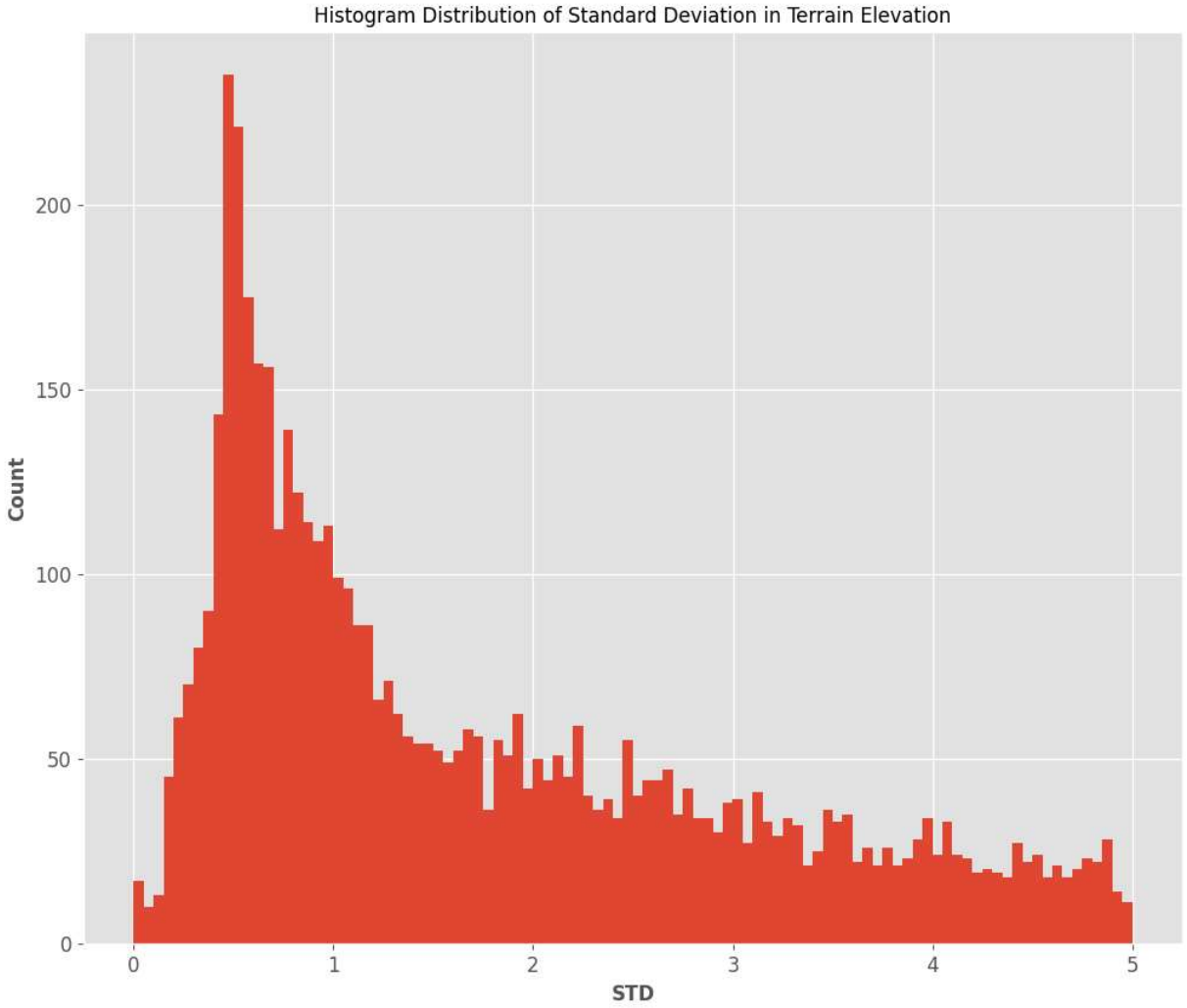
Out[ ]:
```
2.7588799518055347
```

In [ ]:
```
plt.hist(stddevs_n,bins=100);
plt.title(label= "Histogram Distribution of Standard Deviation in Terrain Elevation
plt.ylabel( "Count");
plt.xlabel( "STD");
```



Ignoring the outliers

In [ ]:
```
plt.hist(stddevs_n[stddevs_n<5],bins=100);
plt.title(label= "Histogram Distribution of Standard Deviation in Terrain Elevation
plt.ylabel( "Count");
plt.xlabel( "STD");
```



filtering the tiles based on the STD

In [ ]:
```
def find_suitable_tiles(image, max_allowed_std, tile_size):
    tiles, tile_rows, tile_cols = tile_image(image, tile_size)
    suitable_tiles = []

    for i in range(tile_rows):
        for j in range(tile_cols):
            tile = tiles[i * tile_cols + j]
            stddev = calculate_tile_statistics(tile)
            if stddev <= max_allowed_std:
                suitable_tiles.append((i, j))

    return suitable_tiles
```

```
def create_colored_overlay(image, tile_size, suitable_tiles):
    tiles, tile_rows, tile_cols = tile_image(image, tile_size)
    stddevs = [calculate_tile_statistics(tile) for tile in tiles]

    # Normalize standard deviations to range [0, 1] for coloring
    norm = mcolors.Normalize(vmin=min(stddevs), vmax=max(stddevs))
    cmap = plt.get_cmap('coolwarm')  # Red to white colormap

    # Create an overlay image
    overlay = np.zeros((image.shape[0], image.shape[1], 3), dtype=np.uint8)

    index = 0
    for i in range(tile_rows):
        for j in range(tile_cols):
            tile_std = stddevs[index]
            if (i, j) in suitable_tiles:
                color = [0, 255, 0]  # Green for suitable tiles
            else:
                color = cmap(norm(tile_std))[:3]  # Get RGB color
                color = [int(c * 255) for c in color]  # Convert to 0-255 range
                # pass

            overlay[i*tile_size:(i+1)*tile_size, j*tile_size:(j+1)*tile_size] = col
            index += 1

    return overlay



def filter(image, tile_size, max_allowed_std, a1=0.5, a2=0.5,t=""):

    suitable_tiles = find_suitable_tiles(image, max_allowed_std, tile_size)
    overlay = create_colored_overlay(image, tile_size, suitable_tiles)

    # Blend the original image and the overlay
    blended = cv2.addWeighted(cv2.cvtColor(image, cv2.COLOR_GRAY2BGR), a1, overlay,

    return blended
```

In [ ]:

```
tile_size = 100
max_allowed_std = 10.0

image = load_image(file_path)
suitable_tiles = find_suitable_tiles(image, max_allowed_std, tile_size)
print("Suitable tiles (row, col):", suitable_tiles)
```

```
Suitable tiles (row, col): [(0, 0), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8),
(0, 9), (0, 11), (0, 12), (0, 13), (0, 14), (0, 15), (0, 16), (0, 17), (0, 18), (0,
19), (1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 1
1), (1, 12), (1, 13), (1, 14), (1, 15), (1, 16), (1, 17), (1, 18), (2, 0), (2, 1),
(2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 10), (2, 11), (2, 12), (2, 13),
(2, 14), (2, 15), (2, 16), (2, 17), (2, 18), (2, 19), (3, 0), (3, 1), (3, 3), (3,
4), (3, 5), (3, 6), (3, 8), (3, 9), (3, 10), (3, 11), (3, 12), (3, 13), (3, 14), (3,
15), (3, 16), (3, 17), (3, 18), (3, 19), (4, 0), (4, 8), (4, 9), (4, 10), (4, 11),
(4, 12), (4, 13), (4, 14), (4, 15), (4, 16), (4, 17), (4, 18), (4, 19), (5, 7), (5,
8), (5, 9), (5, 10), (5, 11), (5, 12), (5, 13), (5, 14), (5, 15), (5, 16), (5, 17),
(5, 18), (5, 19), (6, 0), (6, 1), (6, 7), (6, 8), (6, 9), (6, 10), (6, 11), (6, 12),
(6, 13), (6, 14), (6, 15), (6, 16), (6, 17), (6, 18), (6, 19), (7, 0), (7, 1), (7,
6), (7, 7), (7, 8), (7, 9), (7, 10), (7, 11), (7, 12), (7, 13), (7, 14), (7, 15),
(7, 16), (7, 17), (7, 18), (7, 19), (8, 0), (8, 1), (8, 2), (8, 6), (8, 7), (8, 8),
(8, 9), (8, 10), (8, 11), (8, 12), (8, 13), (8, 14), (8, 15), (8, 16), (8, 17), (8,
18), (8, 19), (9, 0), (9, 1), (9, 2), (9, 6), (9, 7), (9, 8), (9, 9), (9, 10), (9, 1
1), (9, 12), (9, 13), (9, 14), (9, 15), (9, 16), (9, 17), (9, 18), (9, 19), (10, 0),
(10, 1), (10, 2), (10, 7), (10, 8), (10, 9), (10, 10), (10, 11), (10, 12), (10, 13),
(10, 14), (10, 15), (10, 16), (10, 17), (10, 18), (10, 19), (11, 0), (11, 1), (11,
2), (11, 7), (11, 8), (11, 9), (11, 10), (11, 11), (11, 12), (11, 13), (11, 14), (1
1, 15), (11, 16), (11, 17), (11, 18), (11, 19), (12, 0), (12, 1), (12, 2), (12, 4),
(12, 8), (12, 9), (12, 10), (12, 11), (12, 12), (12, 13), (12, 14), (12, 15), (12, 1
6), (12, 17), (12, 18), (12, 19), (13, 0), (13, 1), (13, 3), (13, 4), (13, 8), (13,
9), (13, 10), (13, 11), (13, 12), (13, 13), (13, 14), (13, 15), (13, 16), (13, 17),
(13, 18), (13, 19), (14, 0), (14, 1), (14, 3), (14, 4), (14, 9), (14, 10), (14, 11),
(14, 12), (14, 13), (14, 14), (14, 15), (14, 16), (14, 17), (14, 18), (14, 19), (15,
0), (15, 2), (15, 3), (15, 4), (15, 6), (15, 9), (15, 10), (15, 11), (15, 12), (15,
13), (15, 14), (15, 15), (15, 16), (15, 17), (15, 18), (15, 19), (16, 2), (16, 4),
(16, 10), (16, 11), (16, 12), (16, 13), (16, 14), (16, 15), (16, 16), (16, 17), (16,
18), (16, 19), (17, 0), (17, 1), (17, 2), (17, 8), (17, 9), (17, 10), (17, 11), (17,
12), (17, 13), (17, 14), (17, 15), (17, 16), (17, 17), (17, 18), (17, 19), (18, 0),
(18, 1), (18, 5), (18, 9), (18, 10), (18, 11), (18, 12), (18, 13), (18, 14), (18, 1
5), (18, 16), (18, 17), (18, 18), (18, 19)]
```

In [ ]:
```
from tqdm import tqdm_notebook
```

Visualize

In [ ]:
```
def showframes_add(frames,ccmap= None,labels=[],label_font_size=8,nrows=0,ncols=0):
    """Sub function don't call it alone
    """

    plot_di = int(frames.shape[0]**0.5)

    if nrows ==0 or ncols ==0 :
        nrows, ncols = plot_di,plot_di

    fig, axes = plt.subplots(nrows=nrows, ncols=ncols)

    if len(labels)!=0:
        plt.subplots_adjust(left=0.1,bottom=0.02,right=0.9,top=0.99,wspace=0.3,hspa
        for ind ,ax in enumerate( axes.flat):

            im = ax.imshow(frames[int(ind)],cmap =ccmap) #### Plots the frame
            ax.set_title(f"lab: {labels[ind]}", fontstyle='italic',fontsize =label_
```

```
            ax.set_xticks([])#### Turn of Ticks
            ax.set_yticks([])#### Turn of Ticks




    else:
        plt.subplots_adjust(left=0.1,bottom=0.02,right=0.9,top=0.9,wspace=0.3,hspac
        for ind ,ax in enumerate( axes.flat):

            im = ax.imshow(frames[int(ind)],cmap =ccmap) #### Plots the frame
            ax.set_title(f"{ind}", fontstyle='italic',fontsize = label_font_size, p
            ax.set_xticks([])#### Turn of Ticks
            ax.set_yticks([])#### Turn of Ticks




    fig.subplots_adjust(right=0.85)
    cbar_ax = fig.add_axes([0.9, 0.15, 0.02, 0.7])
    fig.colorbar(im, cax=cbar_ax)



#     fig.tight_layout(pad=3.0)



def showframes(frames,typee= None,fig_s = (10,10),labels: list =[],label_font_size:

    """good for dealing with many frames with different image types like RGB, BGR,
    with some types "cmaps":
    gist_gray = g
    jet = c
    tab20b = t
    viridis = v
    cividis = d
    BGR = bgr
    or leave it and will do defult

    labels are used to make titles for each image like the model prediction for thi

    label_font_size takes int

    """
    plt.rcParams['figure.figsize'] = fig_s
    figure(figsize=fig_s, dpi=100)


    if typee=="g":
        showframes_add(frames,ccmap ='gist_gray',labels=labels,label_font_size= lab
    elif typee=="c" :
        showframes_add(frames,ccmap ='jet',labels=labels,label_font_size= label_fon
    elif typee=="t" :
```

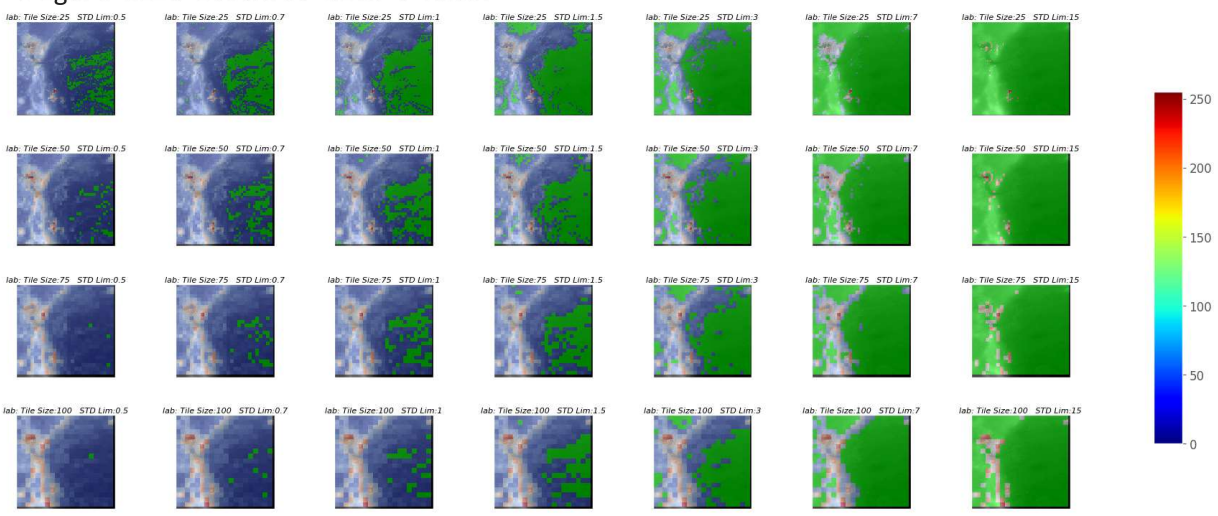
```
        showframes_add(frames,ccmap ='tab20b',labels=labels,label_font_size= label_
    elif typee=="v" :
        showframes_add(frames,ccmap ='viridis',labels=labels,label_font_size= label
    elif typee=="d" :
        showframes_add(frames,ccmap ='cividis',labels=labels,label_font_size= label

    elif typee=="RGB"or typee=='rgb':
        showframes_add(frames[:,:,:,[2,1,0]],ccmap ='cividis',labels=labels,label_f
    else:
        showframes_add(frames,labels=labels,label_font_size= label_font_size,nrows=



    plt.show()
```

In [ ]:
```
tile_size = 25
max_allowed_std = 5.0

tile_sizes = range(25,101,25)
max_allowed_stds = [0.5,.7,1,1.5,3,7,15]

image = load_image(file_path)

ims = []
labs = []
for tile_size in tqdm_notebook(tile_sizes):
    for max_allowed_std in max_allowed_stds:
        labs.append(f'Tile Size:{tile_size}   STD Lim:{max_allowed_std}')
        ims.append(filter(image, tile_size, max_allowed_std))
```

```
C:\Users\Mario\AppData\Local\Temp\ipykernel_10468\3884034434.py:11: TqdmDeprecationW
arning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for tile_size in tqdm_notebook(tile_sizes):
  0%|          | 0/4 [00:00<?, ?it/s]
```

In [ ]:
```
showframes(np.array(ims),fig_s=(20,7),labels=labs,ncols=len(max_allowed_stds),nrows
```

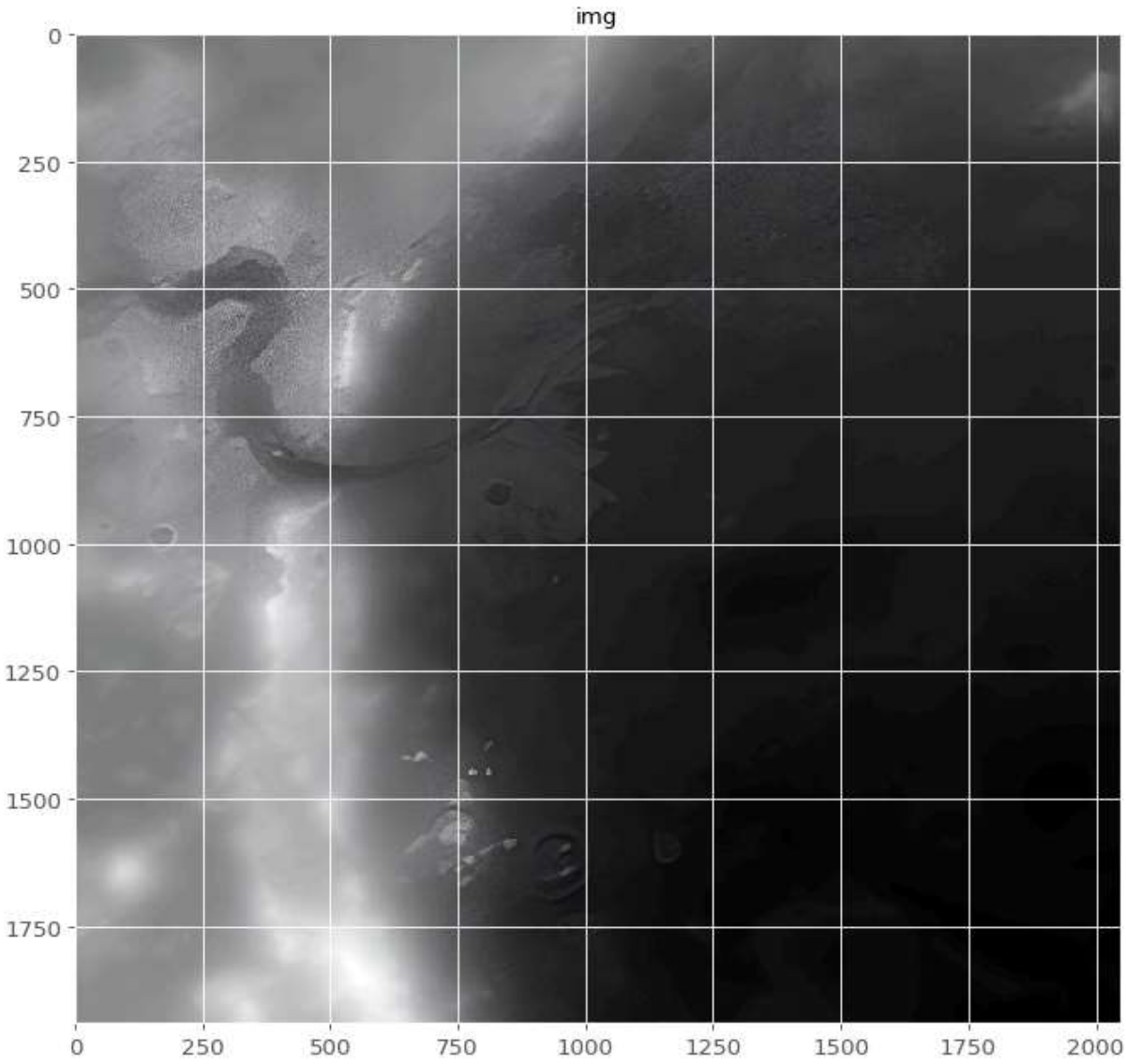
```
<Figure size 2000x700 with 0 Axes>
```



stacking the good tiles

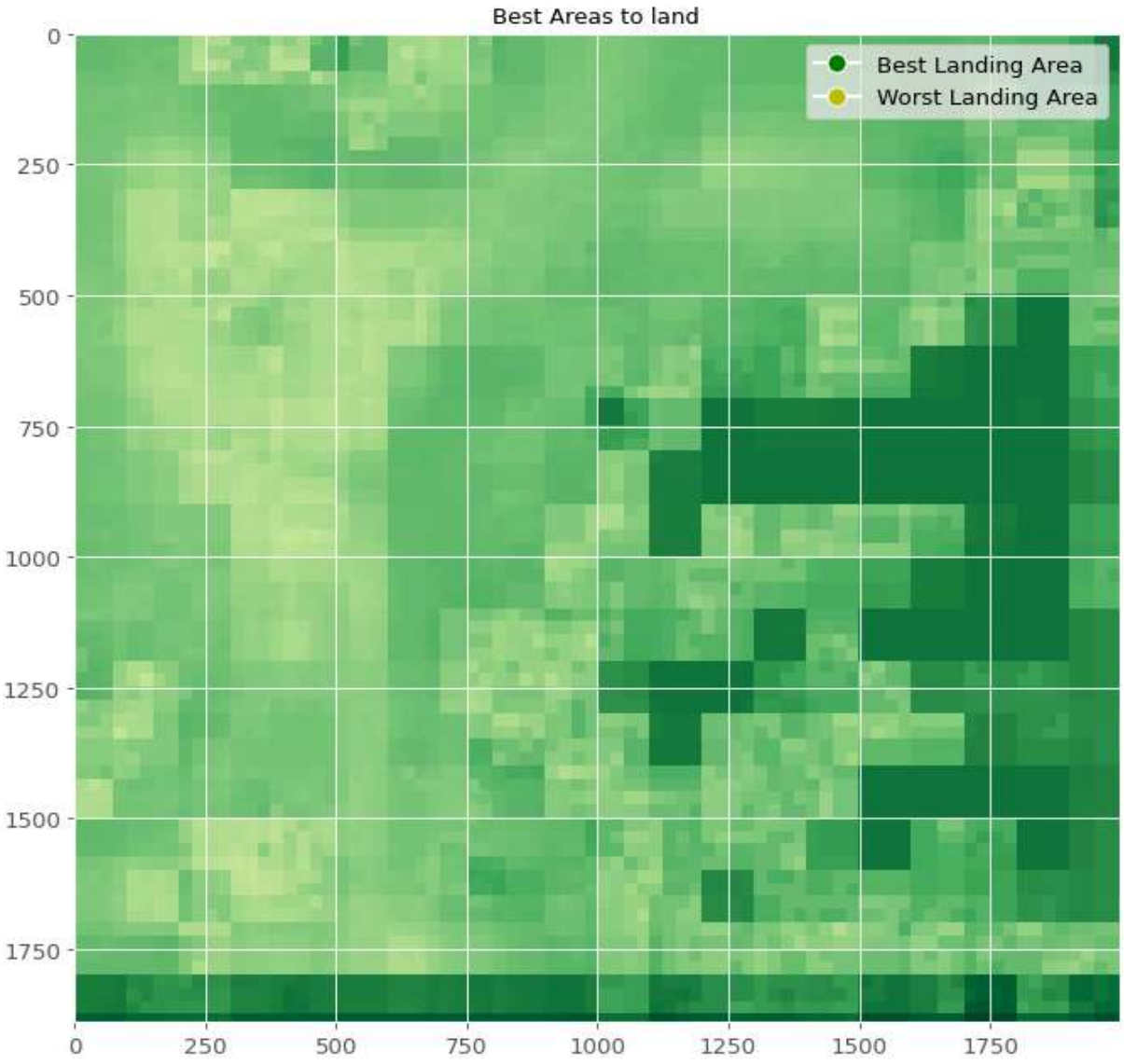
In [ ]:
```
def tile_image(image, tile_size):
    rows, cols,_ = image.shape
    tile_rows = rows // tile_size
    tile_cols = cols // tile_size

    tiles = []
    for i in range(tile_rows):
        for j in range(tile_cols):
            tile = image[i*tile_size:(i+1)*tile_size, j*tile_size:(j+1)*tile_size]
            tiles.append(tile)
    return tiles, tile_rows, tile_cols




def filter(image, tile_size, max_allowed_std, a1=0.5, a2=0.5,t=""):

    suitable_tiles = find_suitable_tiles(image, max_allowed_std, tile_size)
    overlay = create_colored_overlay(image, tile_size, suitable_tiles)

    # Blend the original image and the overlay
    blended = cv2.addWeighted(image, a1, overlay, a2, 0)

    return blended
```

In [ ]:
```
tile_size = 25
max_allowed_std = 1

tile_sizes = range(25,101,25)


image = cv2.imread(file_path)

img_f = image.copy()

showimg(img_f)

ims = []
labs = []
for tile_size in tqdm_notebook(tile_sizes):
    img_f = filter(img_f, tile_size, max_allowed_std,a1=0.5,a2=0.5)
```




```
C:\Users\Mario\AppData\Local\Temp\ipykernel_10468\1444362737.py:15: TqdmDeprecationW
arning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for tile_size in tqdm_notebook(tile_sizes):
  0%|          | 0/4 [00:00<?, ?it/s]
```

In [ ]:
```
img_f[0,0]=[0,0,0]
img_f[0,0]=[255,255,255]
showimg(255-(img_f[:-50,:-50,1]**1.9)-30,'dd',title="Best Areas to land",dd="YlGn")
```


Best Areas to land
Best Landing Area
Worst Landing Area
0
250
500
750
1000
1250
1500
1750