1 Stage: Data exploration

In this part:

- Reading Images taken with the BlackBird and characteristics of the file.
- Plotting the images using different bands.
- Plotting the Image's spectra for chosen bands.
- Reading the cloud file and characteristics of the file.

```
# install hylite

! pip install git+https://github.com/hifexplo/hylite.git

from IPython.display import clear_output
clear_output() # clear output (it isn't easy being clean!)
```

Show hidden output

```
import hylite
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy as sp
import pandas as ps
```

```
from google.colab import drive
drive.mount('/content/drive')
```

Show hidden output

```
from hylite import HyData, HyImage, HyCloud, HyLibrary, HyHeader, HyCollection, HyScene
```

```
# import IO functionality
from hylite import io
```

Reading hdr image:

Data selected: 2023-06-27_22-49-09

```
# open an ENVI image
image = io.load(r'C:\Users\tulas\Desktop\Internship\Blackbird_EN23611\Blackbird_EN23611\2023-06-27_22-49-09.hdr')
```

Description of the hdr file

```
image.header.print()
```

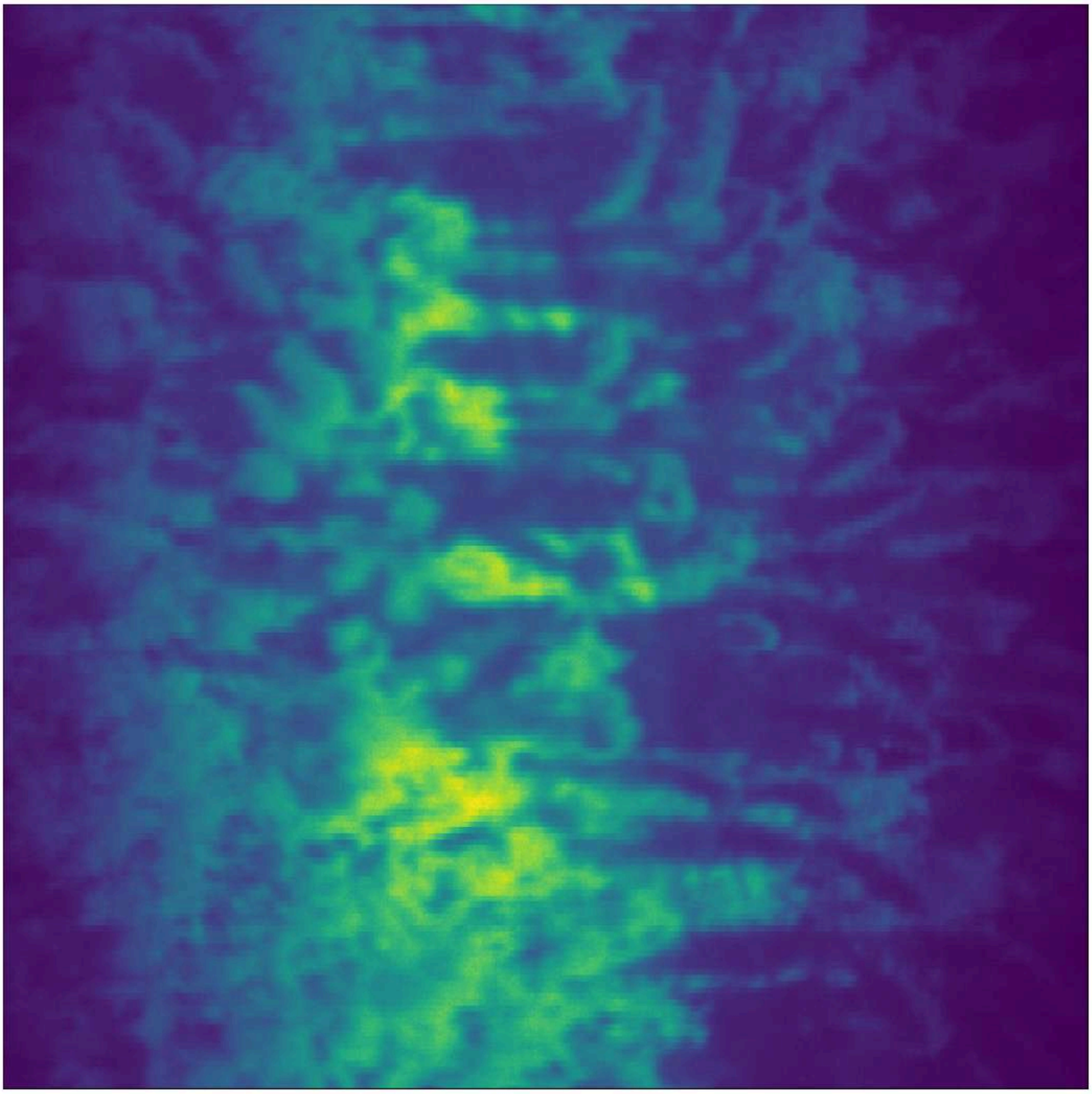
```
file type = { ENVI Standard }
path = { C:\Users\tulas\Desktop\Internship\Blackbird_EN23611\Blackbird_EN23611\2023-06-27_22-49-09.hdr }
description = { Gain (1/10X):50Exposure (Âµs):5000Serialnr: 3015-00003Software Version: v01.00.02.001Captured by HAIP BlackBirdV2 }
samples = { 540 }
lines = { 540 }
bands = { 100 }
header offset = { 0 }
data type = { 12 }
interleave = { bip }
byte order = { 0 }
wavelength units = { nm }
wavelength = { [502. 507. 512. 517. 522. 527. 532. 537. 542. 547. 552. 557. 562. 567.
 572. 577. 582. 587. 592. 597. 602. 607. 612. 617. 622. 627. 632. 637.
 642. 647. 652. 657. 662. 667. 672. 677. 682. 687. 692. 697. 702. 707.
 712. 717. 722. 727. 732. 737. 742. 747. 752. 757. 762. 767. 772. 777.
 782. 787. 792. 797. 802. 807. 812. 817. 822. 827. 832. 837. 842. 847.
 852. 857. 862. 867. 872. 877. 882. 887. 892. 897. 902. 907. 912. 917.
 922. 927. 932. 937. 942. 947. 952. 957. 962. 967. 972. 977. 982. 987.
 992. 997.] }
map info = { UTM, 1, 1, 439296.147, 7183332.583, 0.588, 0.588, 06n, North, WGS-84, units=meters, rotation=-102.900 }
reflectance scale factor = { 1.0 }
```

```
wls = image.header['wavelength']
wls
```

Show hidden output

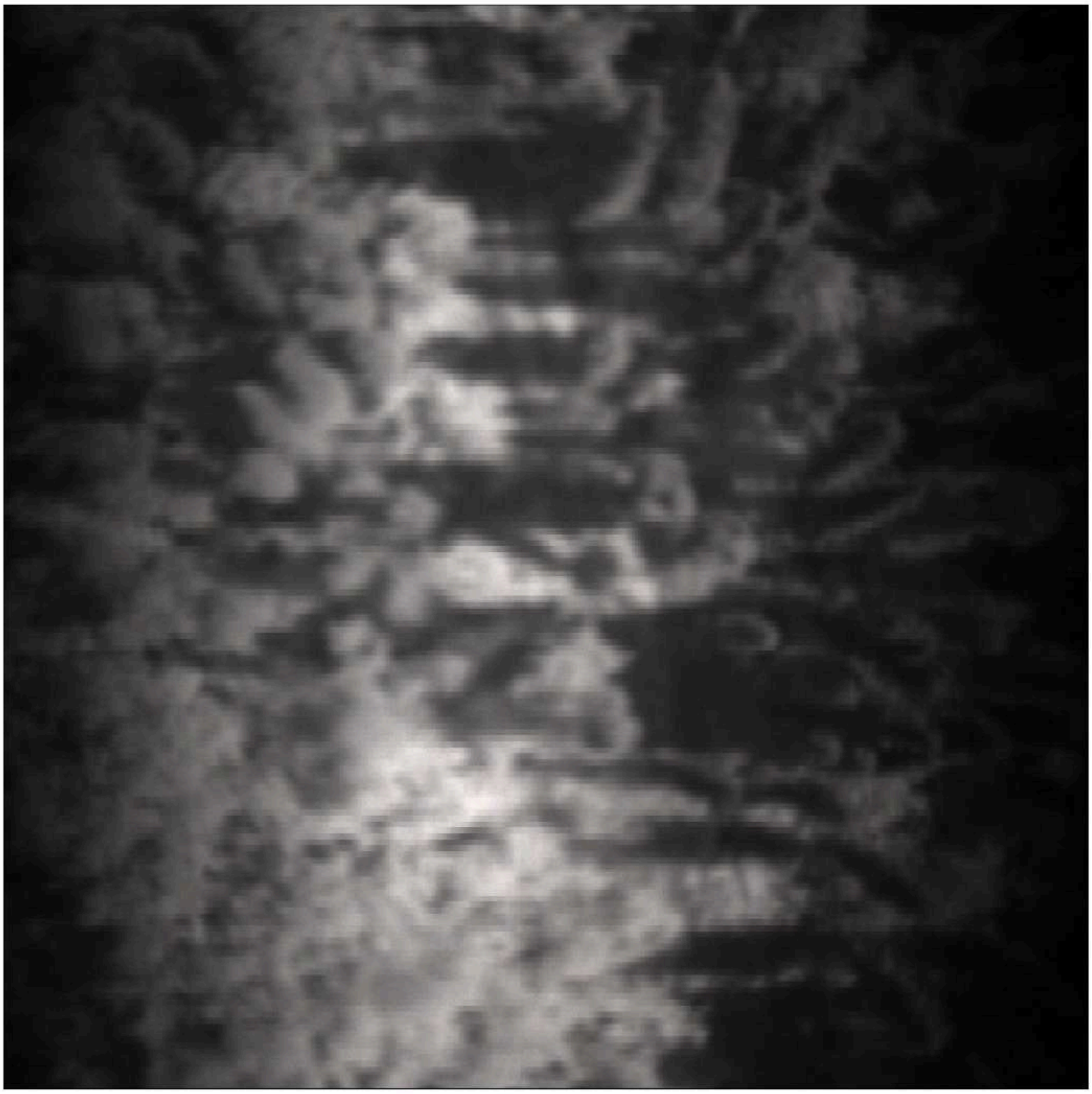
Plotting last band (randomly chosen) :

```
fig,ax = image.quick_plot(99) #Bands: as band (int), or as wv (float)
fig.show()
plt.savefig('image_first_band.png', dpi =300)
```



Plotting 3 wavelengths (last 3 chosen as random)

```
fig,ax = image.quick_plot((987., 992., 997.)) #Bands: as band (int), or as wv (float)
fig.show()
plt.savefig('image_wv_502_507_512.png', dpi =300)
```

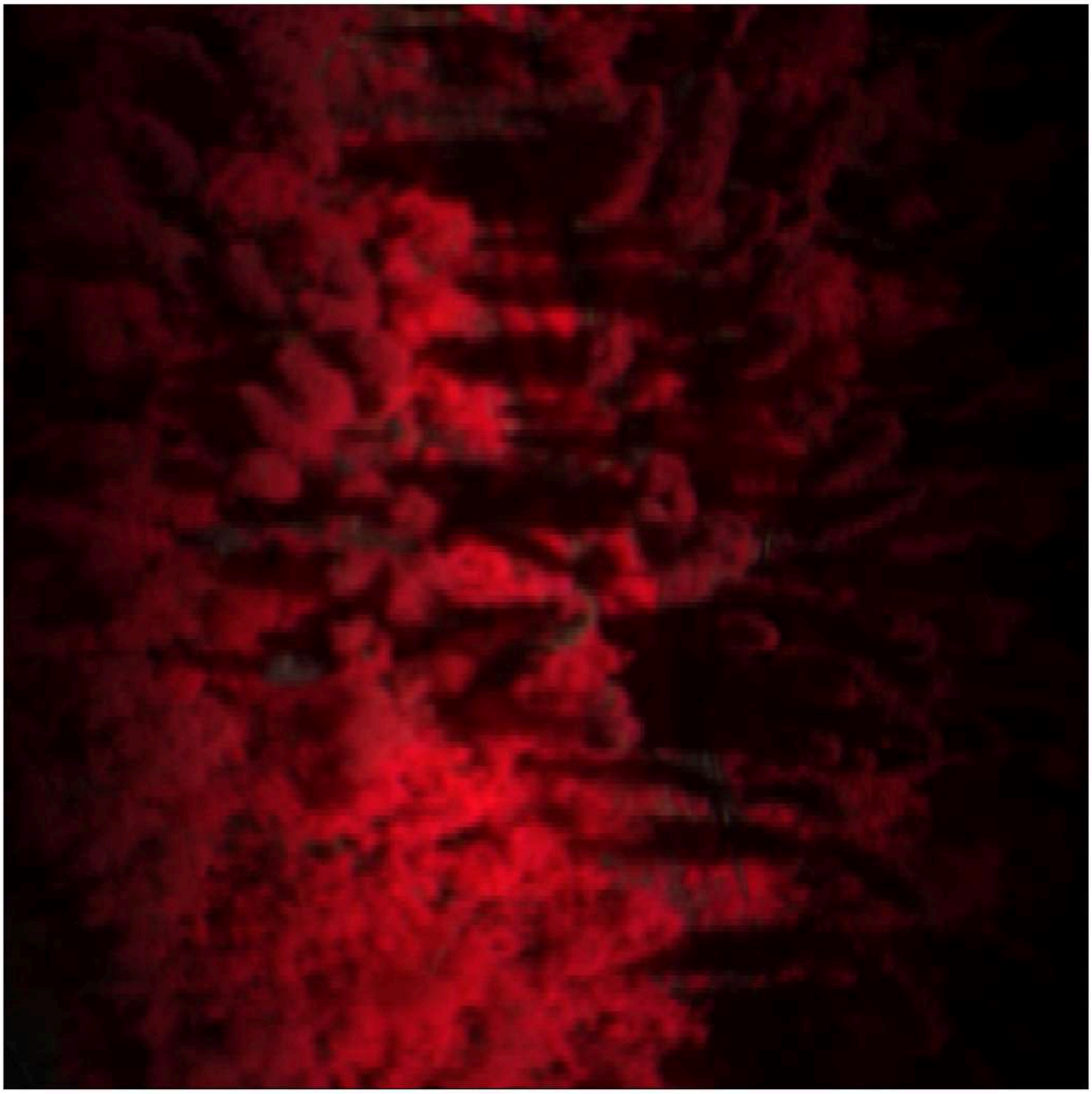


Plotting relevant bands for vegetation analysis

- 562 nm – Green light, used for distinguishing vegetation types and health.

- 667 nm – Red light, useful for vegetation health monitoring (e.g., NDVI).
- 832 nm – Near-infrared (NIR), commonly use for vegetation analysis and remote sensing.

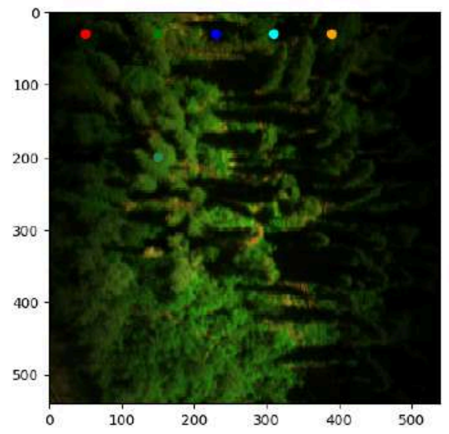
```
fig,ax = image.quick_plot((832., 667., 562.)) #Bands: as band (int), or as wv (float)
fig.show()
plt.savefig('image_wv_832_667_562.png', dpi =300)
```

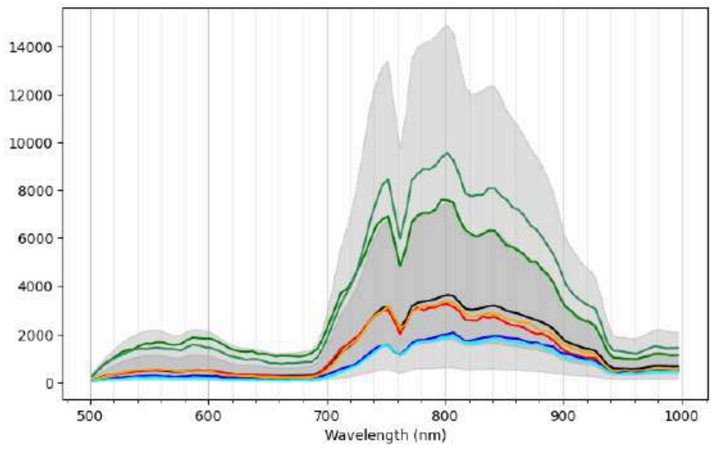


Selecting 3 pixels of the image and plotting their spectra for the first band.

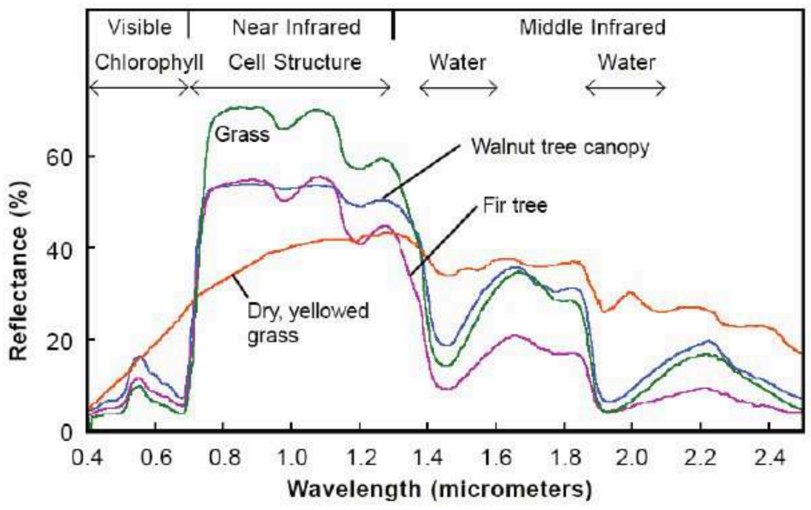
The black spectra is the median, and 5th, 25th, 75th and 95th percentiles of each band (grey envelopes)

```
# plot image and associated spectra
pixels = [(50,30), (150,30), (230,30), (310,30), (390, 30), (150,200)]

fig,ax = plt.subplots(1,2,figsize=(18,5))
image.quick_plot((667.,562.,502.), ax=ax[0], ticks=True) # plot image to existing axes object, and plot x- and y- coords
ax[0].scatter([p[0] for p in pixels], [p[1] for p in pixels], color=['r','g','b', 'cyan', 'orange','seagreen'])

# add a spectral caterpillar
image.plot_spectra(band_range=(500.,1000.), indices=pixels, colours=['r','g','b', 'cyan', 'orange', 'seagreen'], ax=ax[1])
fig.show()
plt.savefig('3_pixels_spectra', dpi =300)
```





A vegetation spectrum (Smith, 2001):



Reading another images

```
# open an img
Data_img = io.load( r'C:\Users\tulas\Desktop\Internship\Blackbird_EN23611\Blackbird_EN23611\2023-06-27_22-49-09.img' )
Data_jpg = io.load( r'C:\Users\tulas\Desktop\Internship\Blackbird_EN23611\Blackbird_EN23611\2023-06-27_22-49-09.jpg' )
Data_png =io.load( r'C:\Users\tulas\Desktop\Internship\Blackbird_EN23611\Blackbird_EN23611\2023-06-27_22-49-09.png' )
Data_tiff = io.load( r'C:\Users\tulas\Desktop\hylite for Tree classification\Blackbird_EN23611\Blackbird_EN23611\RGB full size\2023-06-2


fig,ax = Data_png.quick_plot((0,1,2)) #It has only 3 bands: RGB
fig.show()
```

```
fig,ax = Data_jpg.quick_plot((0,1,2)) #It has only 3 bands: RGB
fig.show()
```

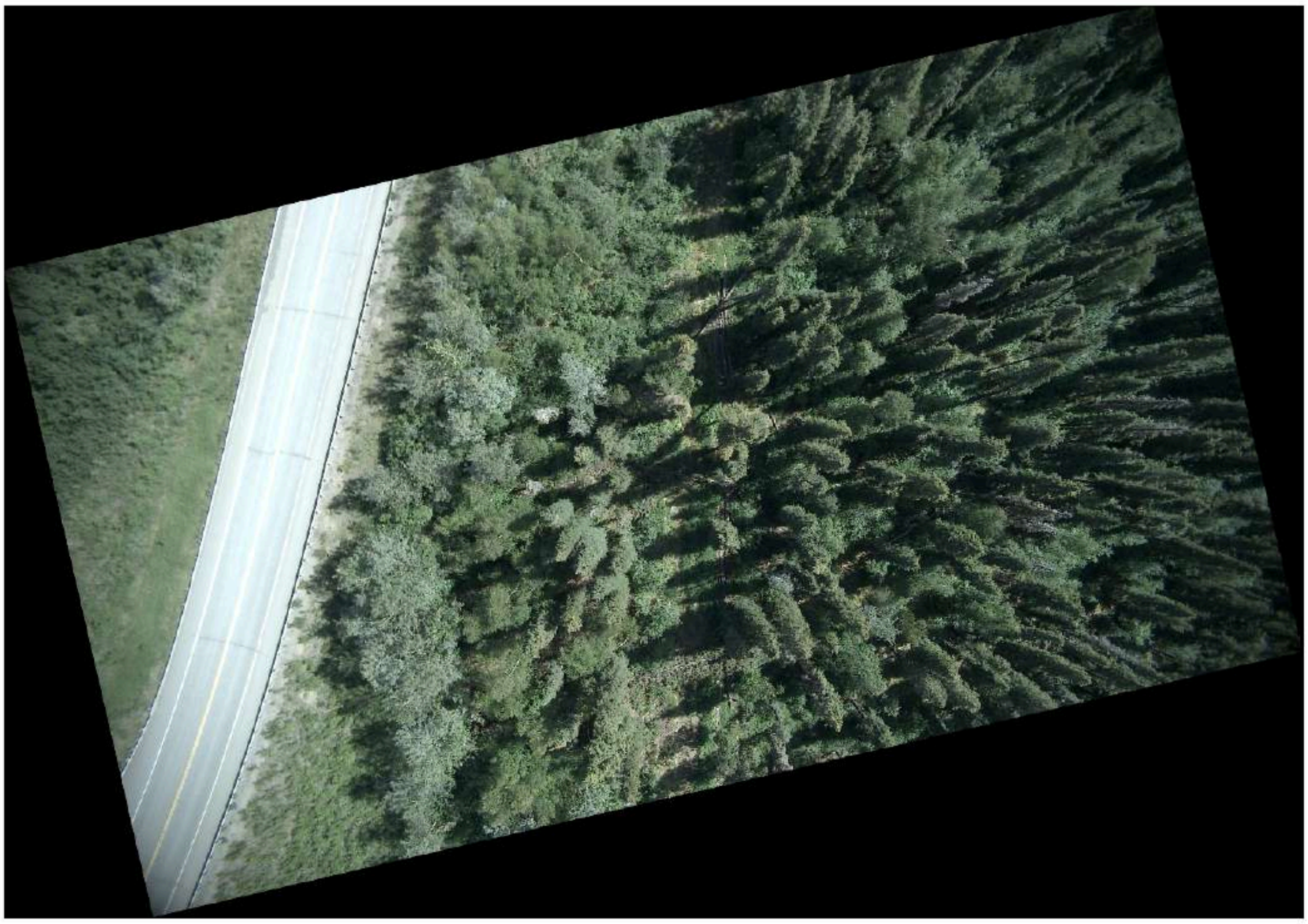
Show hidden output

TIF Image

```
Data_tiff.header.print()
```

```
file type = { ENVI Standard }
reflectance scale factor = { 1.0 }
```

```
fig,ax = Data_tiff.quick_plot((0,1,2)) #It has only 3 bands: RGB
fig.show()
```



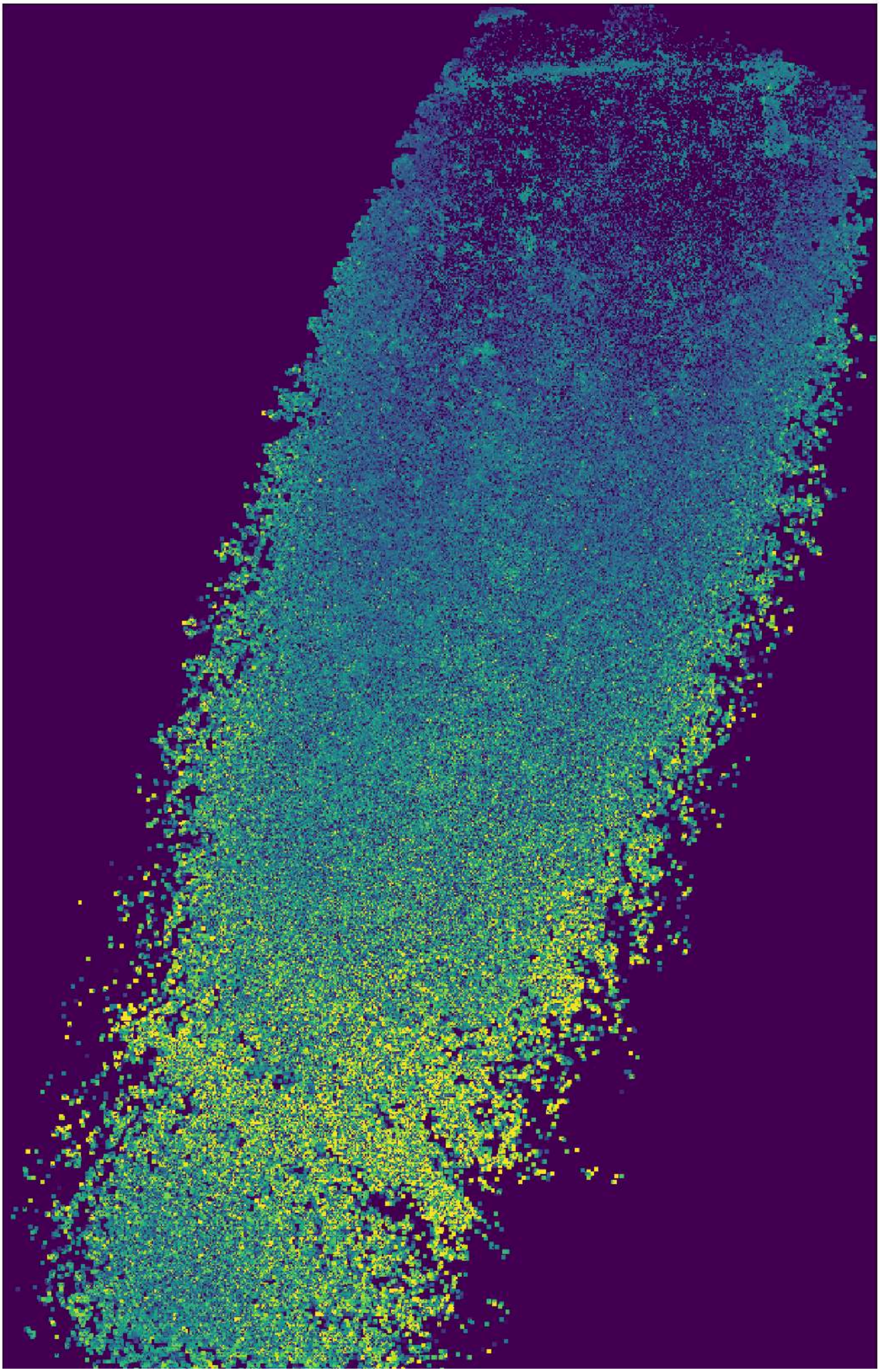
Reading Cloud

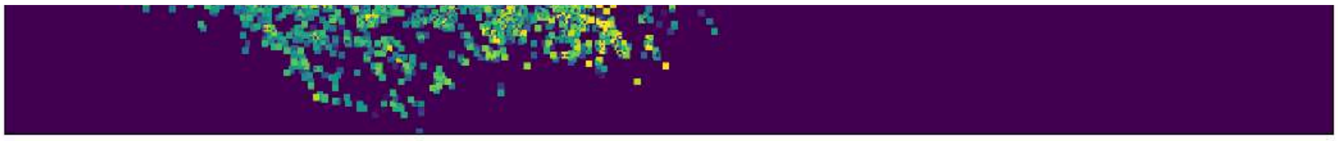
```
# open a .ply point cloud
cloud = io.load(r"C:\Users\tulas\Desktop\Internship\EN23611_pointcloud_2024-05-15_12h09_02_173.ply")
```

```
cloud.header.print()
```

```
file type = { Hypercloud }
```

```
fig,ax = cloud.quick_plot(0, fill_holes=True)
fig.show()
plt.savefig('cloud_band_1.png', dpi=300)
```





```
wl = [cloud.get_wavelengths()[i] for i in range(8)]
wl
```

```
[0, 1, 2, 3, 4, 5, 6, 7]
```

```
cloud.header['bands'] = 8
```

```
cloud.header.print()
```

```
file type = { Hypercloud }
bands = { 8 }
```

Info cloud

```
# open a .ply point cloud
cloud = io.load(r"C:\Users\tulas\Desktop\Internship\EN23611_pointcloud_2024-05-15_12h09_02_173.ply")
```

```
cloud.header.print()
```

```
file type = { Hypercloud }
```

```
!pip install laspy
```

```
Requirement already satisfied: laspy in c:\users\tulas\miniconda3\lib\site-packages (2.5.3)
Requirement already satisfied: numpy in c:\users\tulas\miniconda3\lib\site-packages (from laspy) (1.26.4)
```

```
!pip install pyproj
from pyproj import CRS
```

Show hidden output

```
import laspy

# Load the LAS file
las = laspy.read(r"C:\Users\tulas\Desktop\Internship\EN23611_pointcloud.las")

# Access VLRs
vlrs = las.vlrs

# Iterar sobre cada VLR para inspeccionar su contenido
for vlr in vlrs:
    print(f"Description: {vlr.description}")
    print(f"Record ID: {vlr.record_id}")
    print(f"User ID: {vlr.user_id}")
    #print(f"Record Length: {vlr.record_length_after_header}")
    #print(f"Data: {vlr.record_data}")
    print("-" * 40)
```

```
Description: LASzip DLL 3.4 r3 (191111)
Record ID: 34735
User ID: LASF_Projection
----------------------------------------
```

```
# Inicializar variables para almacenar los GeoKeys
geo_key_directory = None
geo_double_params = None
geo_ascii_params = None

# Iterar sobre cada VLR para encontrar los GeoKeys
for vlr in vlrs:
    if isinstance(vlr, laspy.vlrs.known.GeoKeyDirectoryVlr):
        geo_key_directory = vlr
    elif isinstance(vlr, laspy.vlrs.known.GeoDoubleParamsVlr):
        geo_double_params = vlr
    elif isinstance(vlr, laspy.vlrs.known.GeoAsciiParamsVlr):
        geo_ascii_params = vlr

# Mostrar la información de los GeoKeys
if geo_key_directory:
    print("GeoKeyDirectoryVlr data:")
    for key_entry in geo_key_directory.geo_keys:
        print(key_entry)
    print("-" * 40)
else:
    print("No se encontraron GeoKeyDirectoryVlr")

if geo_double_params:
    print("GeoDoubleParamsVlr data:")
    for item in geo_double_params.doubles:
        print(item)
    print("-" * 40)
else:
    print("No se encontraron GeoDoubleParamsVlr")

if geo_ascii_params:
    print("GeoAsciiParamsVlr data:")
    print(geo_ascii_params.string)
    print("-" * 40)
else:
    print("No se encontraron GeoAsciiParamsVlr")

# Crear un objeto CRS (ejemplo básico)
if geo_ascii_params:
    wkt = geo_ascii_params.string
    crs = CRS.from_wkt(wkt)
    print(f"Proyección interpretada: {crs.to_proj4()}")
else:
    print("No se pudo crear el objeto CRS sin los parámetros ASCII.")
```

```
GeoKeyDirectoryVlr data:
<GeoKey(Id: 3072, Location: 0, count: 1, offset: 32606)>
----------------------------------------
No se encontraron GeoDoubleParamsVlr
No se encontraron GeoAsciiParamsVlr
No se pudo crear el objeto CRS sin los parámetros ASCII.
```

Start coding or generate with AI.

2 Stage: Hypercloud generation

In this stage:

- **Alignement:** Individual pixels were manually selected for each individual images and colored points are shown in them. After that, using CloudCompare, the points corresponding to those pixels were estimated. Using the align_to_cloud_manual function, the cloud and 3 images were aligned.
- **Blend scenes:** After the alignment of some individual images to the cloud, it is possible to use the function Blend_scenes, to project a wider image in the cloud. This last part didn't work

```
# install hylite

! pip install git+https://github.com/hifexplo/hylite.git

from IPython.display import clear_output
clear_output() # clear output (it isn't easy being clean!)
```

```
import hylite
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy as sp
import pandas as ps
```

```
import hylite
from hylite import HyData, HyImage, HyCloud, HyLibrary, HyHeader, HyCollection, HyScene
```

```
# import IO functionality
from hylite import io
```

```
# put these in Camera instances
from hylite.project import Camera
```

Get individual pixels

Image 1

```
im_png = io.load(r'C:\Users\tulas\Desktop\Internship\Blackbird_EN23611\Blackbird_EN23611\2023-06-27_22-49-09.png')
```

```
# open a .ply point cloud
cloud = io.load(r"C:\Users\tulas\Desktop\Internship\EN23611_pointcloud_2024-05-15_12h09_02_173.ply")
```

```
  # keypoints as cloud IDs
points = np.array([24538033, 25510259, 25678553, 558196, 23341312, 102600] )

# keypoints as image pixel coordinates
pixels = [(196,225),(500, 420), (1050, 225), (500, 800), (1500, 600), (300, 1500)]
```

```
# define new camera object to store results in (and define sensor properties)
cam2 = Camera(np.zeros(3), np.zeros(3),
             'persp', # this is a tripod-mounted (panoroamic) sensor. Set to 'persp' for frame sensors.
             fov = 33, # vertical sensor field of view
             dims = (im_png.xdim(), im_png.ydim()),
             step = 0.084 # angular step [ provided by manufacture, or assume square pixels ]
             )
```

pnp function:

It was possible to import it, but it was an error in some part, so I just wrote the code again here.

```
points = np.array(points).astype(np.uint)
kxyz = cloud.xyz[ points, : ]

kxyz
```

```
array([[4.3912419e+05, 7.1832065e+06, 4.1014099e+02],
       [4.3911962e+05, 7.1832035e+06, 4.1067401e+02],
       [4.3911550e+05, 7.1831945e+06, 4.2316400e+02],
       [4.3910662e+05, 7.1832095e+06, 4.0804901e+02],
       [4.3910166e+05, 7.1831840e+06, 4.1959900e+02],
       [4.3909031e+05, 7.1832235e+06, 4.2001801e+02]], dtype=float32)
```

```
kxy = np.array( pixels ).astype(float) - 0.5
kxy
```

```
array([[ 195.5,  224.5],
       [ 499.5,  419.5],
       [1049.5,  224.5],
       [ 499.5,  799.5],
       [1499.5,  599.5],
       [ 299.5, 1499.5]])
```

```
print(im_png.xdim(), im_png.ydim())
```

```
1800 1800
```

```
def pnp(kxyz, kxy, fov, dims, ransac=False, **kwds):
    """
    Solves the pnp problem to locate a camera given keypoints that are known in world and pixel coordinates using opencv.

    Args:
        kxyz: Nx3 array of keypoint positions in world coordinates.
        kxy: Nx2 array of corresponding keypoint positions in pixel coordinates.
        fov: the (vertical) field of view of the camera.
        dims: the dimensions of the image in pixels
        ransac: true if ransac should be used to filter outliers. Default is True.
        **kwds: keyword arguments are passed to cv2.solvePnP(...) or cv2.solvePnPRansac(...).
                Other arguments can include:

            - optical_centre = the pixel coordinates (cx,cy) of the optical centre to use. By default the
                 middle pixel of the image is used. Additionally a custom optical center can be passed as (cx,cy)
    Returns:
        A tuple containing:

         - p = the camera position in world coordinates
         - r = the camera orientation (as XYZ euler angle).
         - inl = list of Ransac inlier indices used to estimate the position, or None if ransac == False.
    """
    import cv2 # import this here to avoid errors if opencv is not installed properly

    # normalize keypoints so that origin is at mean
    mean = np.mean(kxyz, axis=0)
    kxyz = kxyz - mean

    # flip kxy coords to match opencv coord system
    # kxy[:, 1] = dims[1] - kxy[:, 1]
    # kxy[:, 0] = dims[0] - kxy[:, 0]

    # compute camera matrix
    tanfov = np.tan(np.deg2rad(fov / 2))
    aspx = dims[0] / dims[1]
    fx = dims[0] / (2 * tanfov * aspx)
    fy = dims[1] / (2 * tanfov)
    cx, cy = kwds.get("optical_centre", (0.5 * dims[0] - 0.5, 0.5 * dims[1] - 0.5))
    if 'optical_centre' in kwds: del kwds['optical_centre']  # remove keyword
    cameraMatrix = np.array([[fx, 0, cx],
                             [0, fy, cy],
                             [0, 0, 1]])

    # distortion free lens
    dist_coef = np.zeros(4)

    # use opencv to solve pnp problem and hence camera position
    if ransac:
        suc, rot, pos, inl = cv2.solvePnPRansac(objectPoints=kxyz[:, None, :3].copy(),  # points in world coords
                                                imagePoints=kxy[:, None, :2].copy(),  # points in img coords
                                                cameraMatrix=cameraMatrix,  # camera matrix
                                                distCoeffs=dist_coef,
                                                **kwds)
    else:
        inl = None
        suc, rot, pos = cv2.solvePnP(objectPoints=kxyz[:, None,  :3].copy(),  # points in world coords
                                     imagePoints=kxy[:, None, :2].copy(),  # points in img coords
                                     cameraMatrix=cameraMatrix,  # camera matrix
                                     distCoeffs=dist_coef,
                                     **kwds)
    assert suc, "Error - no solution found to pnp problem."

    # get first solution
    if not inl is None:
        inl = inl[:, 0]
    pos = np.array(pos[:, 0])
    rot = np.array(rot[:, 0])

    # apply rotation position vector
    p = -np.dot(pos, spatial.transform.Rotation.from_rotvec(rot).as_matrix())

    # convert rot from axis-angle to euler
    r = spatial.transform.Rotation.from_rotvec(rot).as_euler('xyz', degrees=True)

    # apply dodgy correction to r (some coordinate system shift)
    r = np.array([r[0] - 180, -r[1], -r[2]])

    return p + mean, r, inl
```

```
import numpy as np
from scipy import spatial
from numba import jit, prange

p_est, r_est, inl = pnp(kxyz, kxy, cam2.fov, cam2.dims, ransac=False)
```

```
est = Camera(p_est, r_est, cam2.proj, cam2.fov, cam2.dims, cam2.step)
```

```
def proj_persp( xyz, C, a, fov, dims, normals=None):
    """
    Project 3d point xyz based on a pinhole camera model.

    Args:
        xyz (ndarray): a nx3 numpy array with a position vector (x,y,z) in each column.
        C (ndarray): the position of the camera.
        a (ndarray): the three camera rotatation euler angles (appled around x, then y, then z == pitch, roll, yaw). In DEGREES.
        fov (float): the vertical field of view.
        dims (tuple): the image dimensions (width, height)
        normals (ndarray): per-point normals. Used to do backface culling if specified. Default is None (not used).

    Returns:
        A tuple containing:

        - pp = a (n,3) array containg the project point coordinates (x,y) and distance from the camera (z).
        - vis = a (n,) array scored with True if the corresponding point is visible from the camera.
    """

    #transform origin to camera position
    xyz = xyz - C[None,:]

    # backface culling
    vis = np.full(xyz.shape[0],True,dtype=bool)
    if not normals is None:
        ndv = np.einsum('ij, ij->i', normals, xyz) #calculate dot product between each normal and vector from camera to point
                                                   #(which is the point position vector in this coordinate system)
        vis = ndv < 0 #normal is pointing towards the camera (in opposite direction to view)

    #apply camera rotation to get to coordinate system
    #  where y is camera up and z distance along the view axis
    R = spatial.transform.Rotation.from_euler('XYZ',-a,degrees=True).as_matrix()
    #xyz = np.dot(xyz, R)
    xyz = xyz@R

    #calculate image plane width/height in projected coords
    h = 2 * np.tan( np.deg2rad( fov / 2 ) )
    w = h * dims[0] / float(dims[1])

    #do project and re-map to image coordinates such that (0,0) = lower left, (1,1) = top right)
    px = ((w/2.0) + (xyz[:,0] / -xyz[:,2])) / w
    py = ((h/2.0) + (xyz[:,1] / xyz[:,2])) / h
    pz = -xyz[:,2] #return positive depths for convenience

    #calculate mask showing 'visible' points based on image dimensions
    vis = vis & (pz > 0) & (px > 0) & (px < 1) & (py > 0) & (py < 1)

    #convert to pixels
    px *= dims[0]
    py *= dims[1]
    return np.array([px,py,pz]).T, vis
```

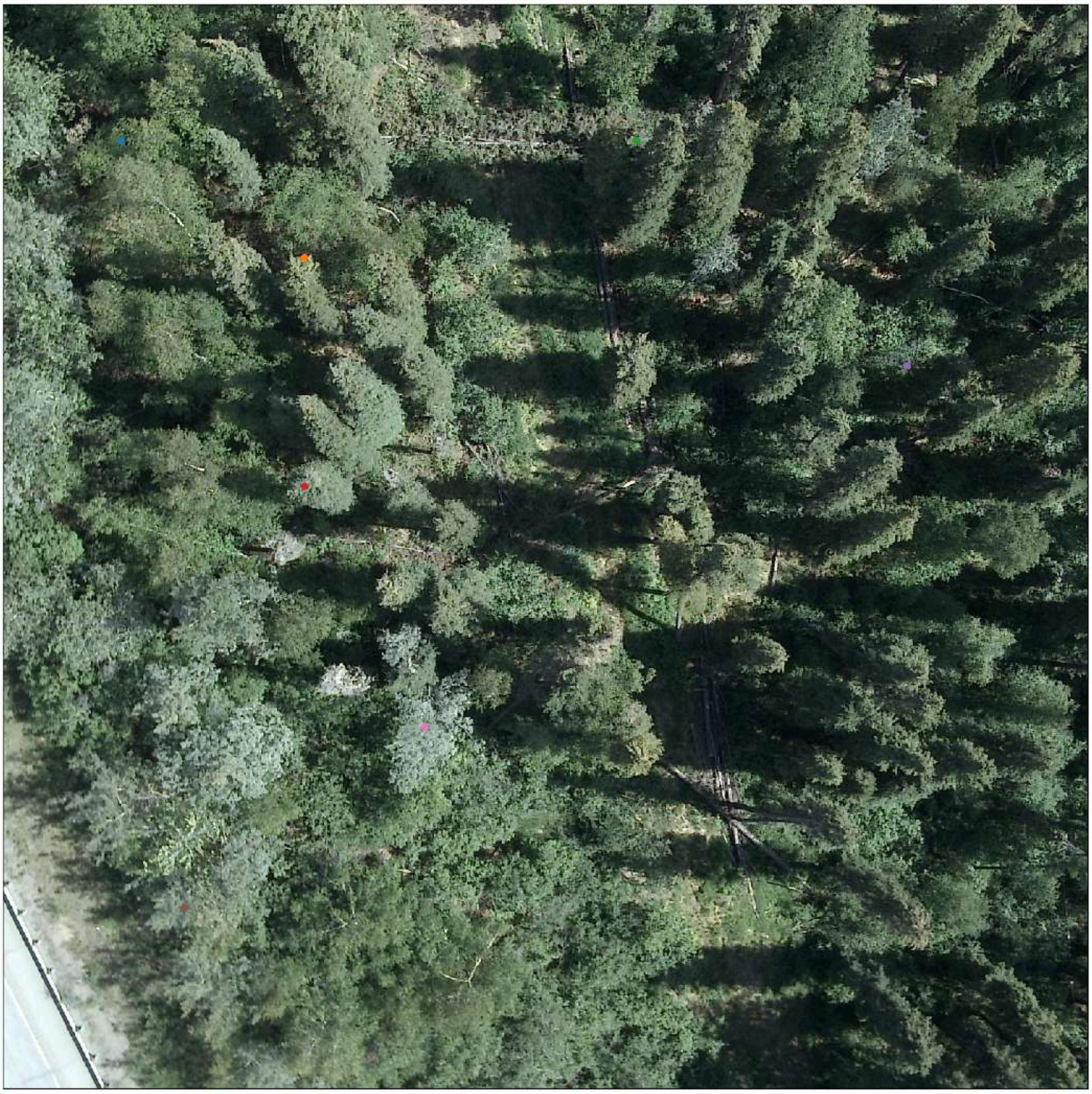
Calculations

```
# calculate final correspondances
if 'pano' in cam2.proj.lower():
    pp, vis = proj_pano(kxyz[inl, :], C=p_est, a=r_est,
                        fov=cam2.fov, dims=cam2.dims, step=cam2.step)
else:
    pp, vis = proj_persp(kxyz[inl, :], C=p_est, a=r_est,
                          fov=cam2.fov, dims=cam2.dims)
#print(pp)
#print(pp[:, 0:2])
#print(kxy[inl, :])
#err = np.linalg.norm(pp[:, 0:2] - kxy[inl, :], axis=1)
#err = np.mean(err)

print(est)
```

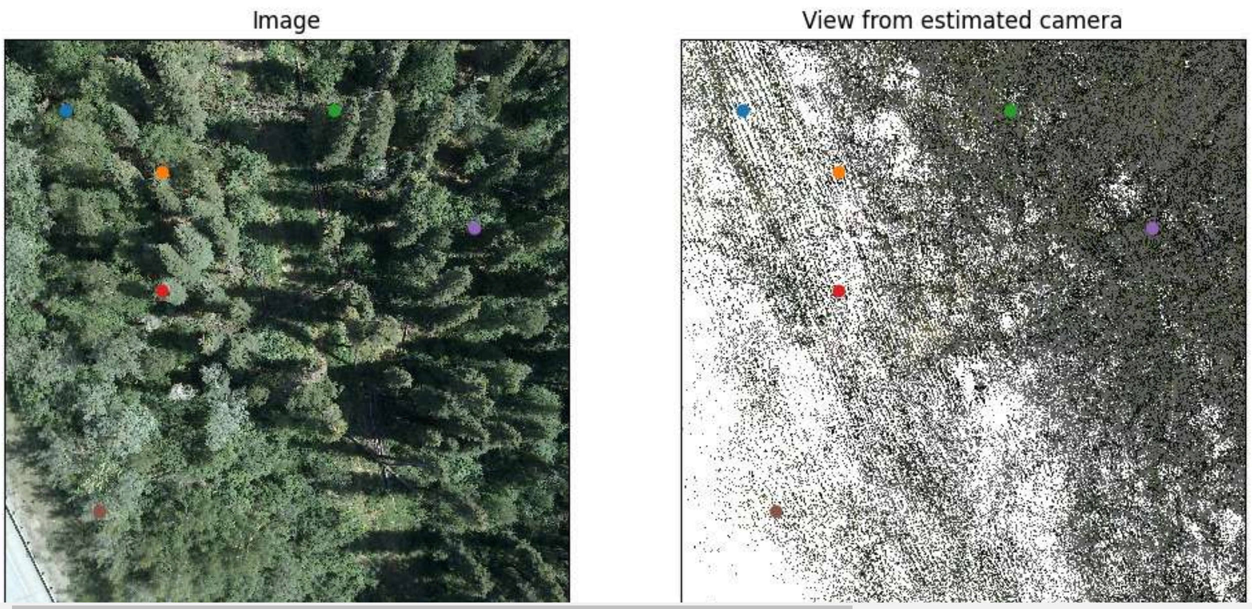
```
<hylite.project.camera.Camera object at 0x000001A92F45D8B0>
```

```
fig = plt.figure(figsize = (10,7))
im_png.quick_plot((0,1,2)) #It has only 3 bands: RGB

for px,py in pixels:
    plt.scatter(px,py)

fig.show()
```

```
<Figure size 1000x700 with 0 Axes>
```



```
fig,ax = plt.subplots(1,2,figsize=(12,10))
im_png.quick_plot((0,1,2),ax=ax[0])
cloud.quick_plot('rgb', est, ax=ax[1])
ax[0].set_title("Image")
ax[1].set_title("View from estimated camera")
for px,py in pixels:
    ax[0].scatter(px,py)
    ax[1].scatter(px,py)
#for a in ax:
    #a.set_ylim(350,100) # zoom in a bit
#fig.tight_layout()
fig.show()
```



```
from hylite.project.align import align_to_cloud
est2, kp, r = align_to_cloud( im_png, cloud, est, bands=(0,1,2),
                             method='sift', # which keypoint extractor to use
                             sf=2, # supersample rendered point cloud to improve matching (sometimes)
                             s=2, # size of points when rendering cloud
                             recurse=1, # repeatedly render cloud to improve/update matching based on new pose
                             gf=True) # display graphical QAQC plots
```

Show hidden output

Image 2

```
im_2 = io.load(r"C:\Users\tulas\Desktop\Internship\Blackbird_EN23611\Blackbird_EN23611\2023-06-27_22-49-31.png")
```

```
# keypoints as cloud IDs
#points = np.array([47841697, 48088645, 47091845, 25345624, 25019950, 48040694, 48064632, ])
points_2 = np.array([25615050, 24674969, 337308, 25003931, 25428822, 47498412, ])

# keypoints as image pixel coordinates
#Azul, naranja, verde, rojo, violeta, marron, rosa
pixels_2 = [(1100, 1230), (350, 1290), (1150,1700), (900, 730), (550, 450), (600,250 )]


# define new camera object to store results in (and define sensor properties)
cam2 = Camera(np.zeros(3), np.zeros(3),
             'persp', # this is a tripod-mounted (panoroamic) sensor. Set to 'persp' for frame sensors.
             fov = 33, # vertical sensor field of view
             dims = (im_2.xdim(), im_2.ydim()),
             step = 0.084 # angular step [ provided by manufacture, or assume square pixels ]
             )


points_2 = np.array(points_2).astype(np.uint)
kxyz = cloud.xyz[ points_2, : ]

kxyz
```

```
array([[4.3912094e+05, 7.1831935e+06, 4.2853900e+02],
       [4.3912462e+05, 7.1832060e+06, 4.1036099e+02],
       [4.3910719e+05, 7.1832005e+06, 4.1538101e+02],
       [4.3913625e+05, 7.1831915e+06, 4.2768799e+02],
       [4.3914222e+05, 7.1832020e+06, 4.0937500e+02],
       [4.3914812e+05, 7.1831970e+06, 4.3356400e+02]], dtype=float32)
```

```
kxy = np.array( pixels_2 ).astype(float) - 0.5
kxy
```

```
array([[1099.5, 1229.5],
       [ 349.5, 1289.5],
       [1149.5, 1699.5],
       [ 899.5,  729.5],
       [ 549.5,  449.5],
       [ 599.5,  249.5]])
```

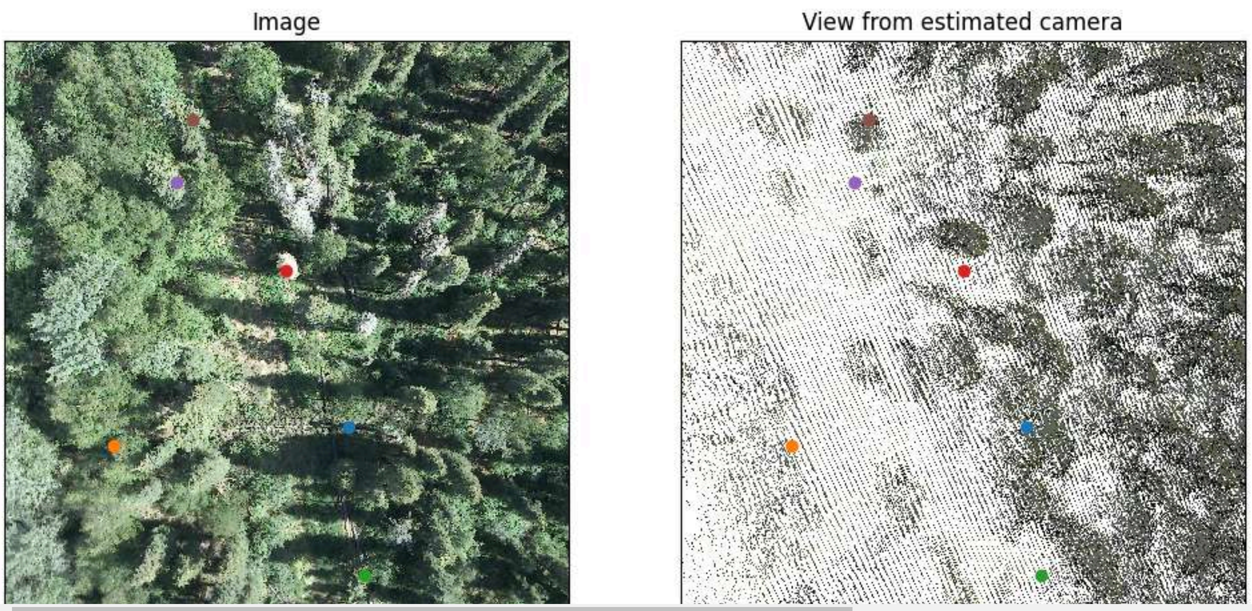
```
import numpy as np
from scipy import spatial
from numba import jit, prange

p_est, r_est, inl = pnp(kxyz, kxy, cam2.fov, cam2.dims, ransac=False)



est_= Camera(p_est, r_est, cam2.proj, cam2.fov, cam2.dims, cam2.step)


# calculate final correspondances
if 'pano' in cam2.proj.lower():
    pp, vis = proj_pano(kxyz[inl, :], C=p_est, a=r_est,
                        fov=cam2.fov, dims=cam2.dims, step=cam2.step)
else:
    pp, vis = proj_persp(kxyz[inl, :], C=p_est, a=r_est,
                         fov=cam2.fov, dims=cam2.dims)
#print(pp)
#print(pp[:, 0:2])
#print(kxy[inl, :])
#err = np.linalg.norm(pp[:, 0:2] - kxy[inl, :], axis=1)
#err = np.mean(err)

print(est_)
```

```
<hylite.project.camera.Camera object at 0x000001A93FAB8C20>
```

```
fig = plt.figure(figsize = (8,5))
im_2.quick_plot((0,1,2)) #It has only 3 bands: RGB

for px,py in pixels_2:
    plt.scatter(px,py)

fig.show()
```

Show hidden output

```
fig,ax = plt.subplots(1,2,figsize=(12,10))
im_2.quick_plot((0,1,2),ax=ax[0])
cloud.quick_plot('rgb', est_, ax=ax[1])
ax[0].set_title("Image")
ax[1].set_title("View from estimated camera")
for px,py in pixels_2:
    ax[0].scatter(px,py)
    ax[1].scatter(px,py)
#for a in ax:
    #a.set_ylim(350,100) # zoom in a bit
#fig.tight_layout()
fig.show()
```



```
from hylite.project.align import align_to_cloud
est_2, kp, r = align_to_cloud( im_2, cloud, est_, bands=(0,1,2),
                              method='sift', # which keypoint extractor to use
                              sf=2, # supersample rendered point cloud to improve matching (sometimes)
                              s=2, # size of points when rendering cloud
                              recurse=1, # repeatedly render cloud to improve/update matching based on new pose
                              gf=True) # display graphical QAQC plots
```

```
Projecting scene... Done.
Gathering matches....r=342....g=464....b=328..(1134).
Solved PnP problem using 11 inliers (residual error = 1.1 px).
```

Reprojected keypoint residuals.

Image 3:

```
im_3 = io.load(r"C:\Users\tulas\Desktop\Internship\Blackbird_EN23611\Blackbird_EN23611\2023-06-27_22-50-14.png")


# keypoints as cloud IDs
points_3 = np.array([300730,24515700,1523628,24829406,25987940,25109826])

# keypoints as image pixel coordinates
#Azul, naranja, verde, rojo, violeta, marron,
pixels_3 = [(250, 1250), (150, 700), (870,1320), (410, 1590), (1250, 610), (650,650)]


fig = plt.figure(figsize = (8,5))
im_3.quick_plot((0,1,2)) #It has only 3 bands: RGB

for px,py in pixels_3:
    plt.scatter(px,py)

fig.show()
```

Show hidden output

```
# put these in Camera instances
from hylite.project import Camera



# define new camera object to store results in (and define sensor properties)
cam2 = Camera(np.zeros(3), np.zeros(3),
             'persp', # this is a tripod-mounted (panoroamic) sensor. Set to 'persp' for frame sensors.
             fov = 33, # vertical sensor field of view
             dims = (im_3.xdim(), im_3.ydim()),
             step = 0.084 # angular step [ provided by manufacture, or assume square pixels ]
             )


#points_3 = np.array(points_3).astype(np.uint)
kxyz = cloud.xyz[ points_3, : ]

kxyz
```

```
array([[4.3909025e+05, 7.1832035e+06, 4.0937299e+02],
       [4.3911231e+05, 7.1832035e+06, 4.1724100e+02],
       [4.3909631e+05, 7.1831860e+06, 4.2049301e+02],
       [4.3910569e+05, 7.1831995e+06, 4.6682199e+02],
       [4.3912041e+05, 7.1831755e+06, 4.2348199e+02],
       [4.3911050e+05, 7.1832055e+06, 4.3077499e+02]], dtype=float32)
```

```
kxy = np.array( pixels_3 ).astype(float) - 0.5
kxy
```

```
array([[ 249.5, 1249.5],
       [ 149.5,  699.5],
       [ 869.5, 1319.5],
       [ 409.5, 1589.5],
       [1249.5,  609.5],
       [ 649.5,  649.5]])
```

```
print(im_3.xdim(), im_3.ydim())
```

```
1800 1800
```

```
import numpy as np
from scipy import spatial
from numba import jit, prange

p_est, r_est, inl = pnp(kxyz, kxy, cam2.fov, cam2.dims, ransac=False)



est__ = Camera(p_est, r_est, cam2.proj, cam2.fov, cam2.dims, cam2.step)


# calculate final correspondances
if 'pano' in cam2.proj.lower():
    pp, vis = proj_pano(kxyz[inl, :], C=p_est, a=r_est,
                        fov=cam2.fov, dims=cam2.dims, step=cam2.step)
else:
    pp, vis = proj_persp(kxyz[inl, :], C=p_est, a=r_est,
                         fov=cam2.fov, dims=cam2.dims)
```

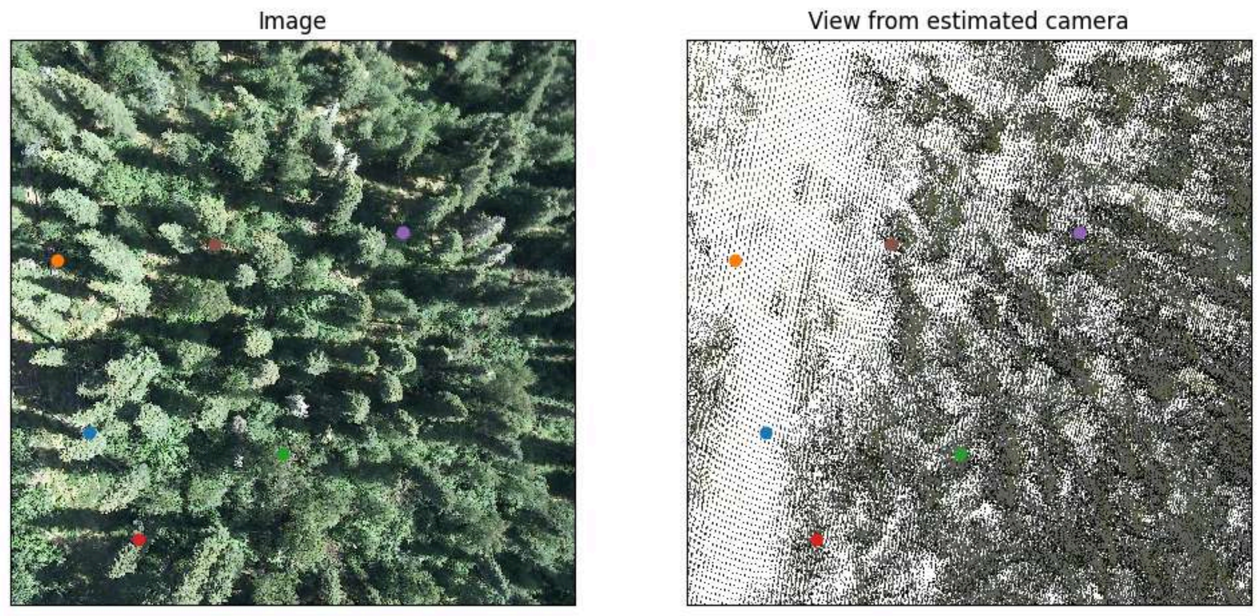
```
#print(pp)
#print(pp[:, 0:2])
#print(kxy[inl, :])
#err = np.linalg.norm(pp[:, 0:2] - kxy[inl, :], axis=1)
#err = np.mean(err)

print(est__)
```

```
<hylite.project.camera.Camera object at 0x000001A92AB6E2A0>
```

```
cloud.quick_plot('rgb', est__)
```

Show hidden output

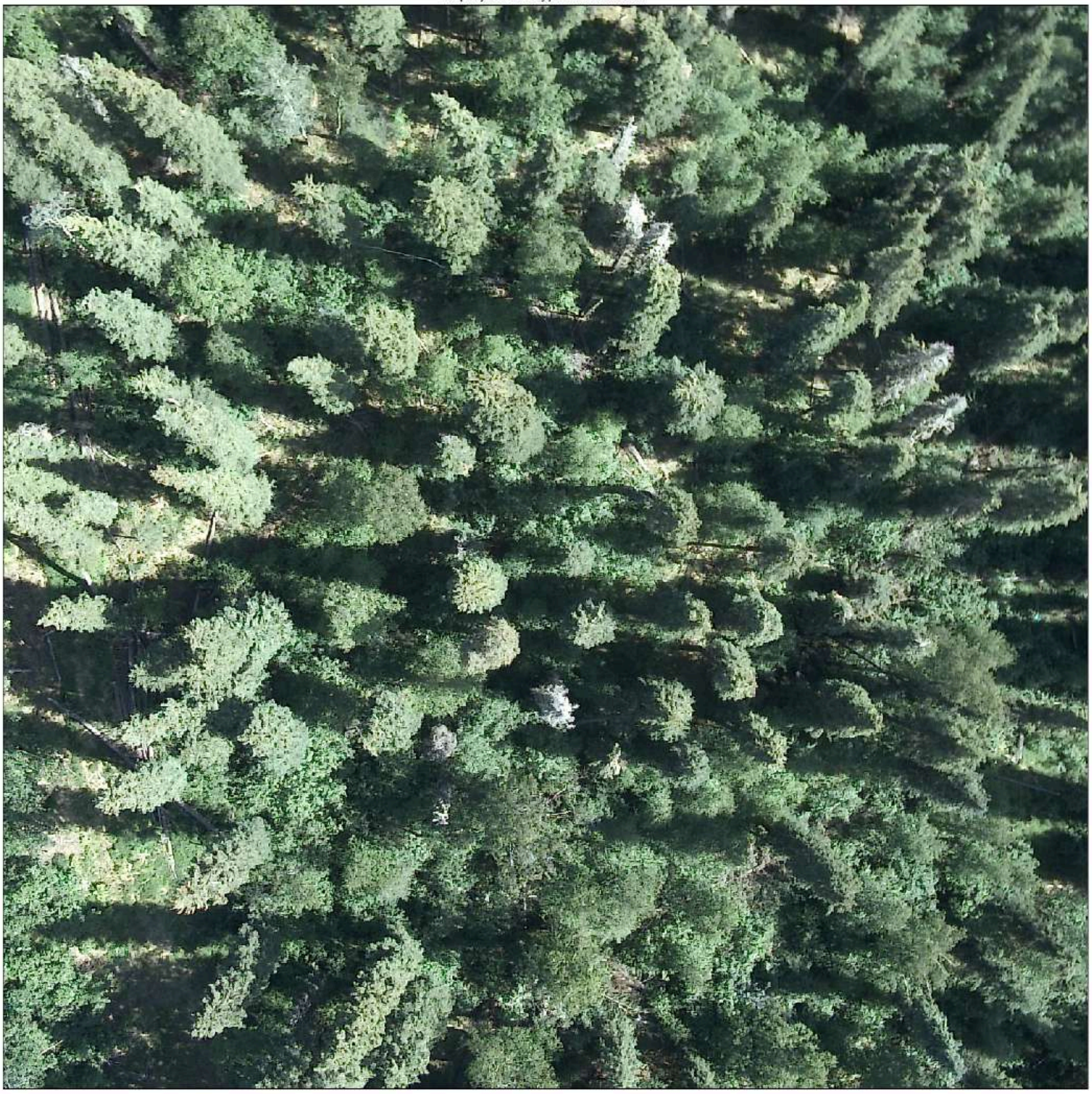
```
fig,ax = plt.subplots(1,2,figsize=(12,10))
im_3.quick_plot((0,1,2),ax=ax[0])
cloud.quick_plot('rgb', est__, ax=ax[1])
ax[0].set_title("Image")
ax[1].set_title("View from estimated camera")
for px,py in pixels_3:
    ax[0].scatter(px,py)
    ax[1].scatter(px,py)
#for a in ax:
    #a.set_ylim(350,100) # zoom in a bit
#fig.tight_layout()
fig.show()
```



```
from hylite.project.align import align_to_cloud
est__2, kp, r = align_to_cloud( im_3, cloud, est__, bands=(0,1,2),
                             method='sift', # which keypoint extractor to use
                             sf=2, # supersample rendered point cloud to improve matching (sometimes)
                             s=2, # size of points when rendering cloud
                             recurse=1, # repeatedly render cloud to improve/update matching based on new pose
                             gf=True) # display graphical QAQC plots
```

```
Projecting scene... Done.
Gathering matches....r=347....g=394....b=397..(1138).
Solved PnP problem using 10 inliers (residual error = 0.7 px).
```

Reprojected keypoint residuals.



Save scenes:

It is used to use the blend_scenes function and combine all the scenes created

```
from hylite import HyScene
```

```
#The camera position is saved in:
est2
```

```
S = HyScene('scene1', r'C:\Users\tulas\Desktop\Internship\Scenes') # initialise a scene just like any HyCollection
S.construct(im_png, cloud, est) # do projections and construct scene
```

```
S.print()
```

```
Attributes stored in RAM:
          - <class 'hylite.project.pmap.PMap'> called pmap
          - <class 'numpy.ndarray'> called xyz
          - <class 'numpy.ndarray'> called depth
          - <class 'numpy.ndarray'> called view
          - <class 'hylite.hycloud.HyCloud'> called cloud
          - <class 'NoneType'> called normals
          - <class 'hylite.hyimage.HyImage'> called image
          - <class 'bool'> called vb
          - <class 'int'> called occ_tol
          - <class 'hylite.project.camera.Camera'> called camera
          - <class 'int'> called maxf
Attributes stored in header:
Attributes stored on disk:
```

```
S_2 = HyScene('scene2', r'C:\Users\tulas\Desktop\Internship\Scenes') # initialise a scene just like any HyCollection
S_2.construct(im_2, cloud, est_2) # do projections and construct scene
```

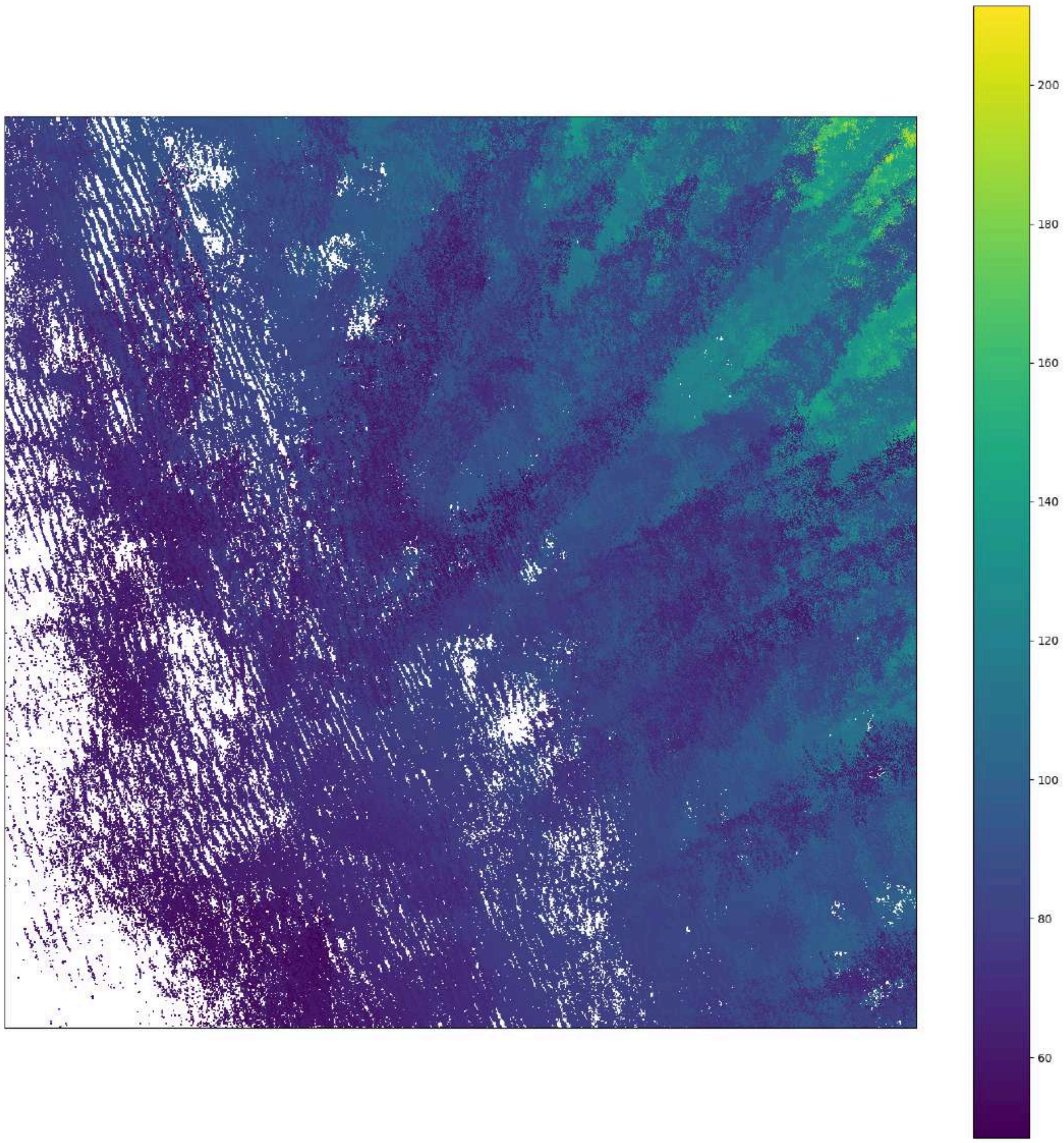
```
S_3 = HyScene('scene3',r'C:\Users\tulas\Desktop\Internship\Scenes') # initialise a scene just like any HyCollection
S_3.construct(im_3, cloud, est__2 ) # do projections and construct scene
```

```
S.save() # save HyScene for later use
S_2.save() # save HyScene for later use
S_3.save() # save HyScene for later use
```

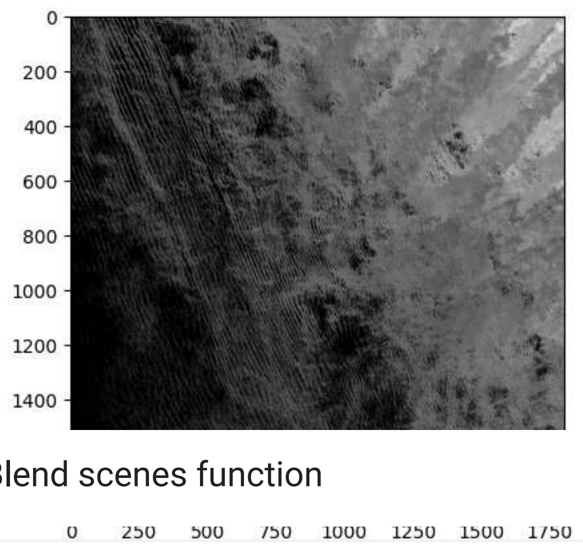
Show scenes

```
depths = cloud.render(est, bands='xyz',fill_holes=True ) # render point position
depths.set_as_nan(0) # remove zeros
depths.data -= est.pos # put camera at origin
depths.data = np.dstack( [ depths.data , np.linalg.norm(depths.data, axis=-1 ) ] ) # compute distance from sensor
depths.set_band_names(['x','y','z','depth']) # add band names
```

```
# plot depth
fig,ax = depths.quick_plot('depth')
fig.colorbar(ax.cbar, fontsize = 20) # N.B. colorbar information is added to the relevant axes object as the .cbar attribute
fig.show()
```



```
# plot per-pixel depths
plt.imshow(S.depth.T, cmap='gray')
plt.show()
```



Blend scenes function

```
def get_blend_weights(scenes, method, ascloud=True):
    """
    Compute weights for doing scene blending based on geometric attributes computed using
    push_geomattr( ... ).

    Args:
        scenes: a list of scenes to compute blend weights for.
        method: the weighting method. Options are:

             - 'equal' = all values are weighted equally.
             - 'distance' = closer points are given higher weight.
             - 'gsd' = lower ground-sampling values are given higher weight.
             - 'obliquity' = less oblique points are given higher weight.

        ascloud: True if weights should be returned as a HyCloud instance. Otherwise a numpy array is returned.
    Returns:
        a numpy array containing the normalised blending weight for each (point,scene).
    """

    weights = np.ones((scenes[0].cloud.point_count(), len(scenes)))
    if 'equal' not in method.lower():
        for i, s in enumerate(scenes):
            # compute geometric features
            geom = push_geomattr(s)
            if 'dist' in method.lower():
                weights[:, i] = geom.data[:, 0]
            elif 'gsd' in method.lower():
                weights[:, i] = geom.data[:, 1]
            elif 'obl' in method.lower():
                weights[:, i] = geom.data[:, 2]
            else:
                assert False, "Error - %s is an unknown weighting method" % method
        # invert and normalise
        mn, mx = np.nanpercentile(weights, (0, 100))
        weights = mx - weights  # lower values should get higher weight, so flip
    weights /= np.nansum(weights, axis=-1)[..., None]  # sum to 1

    if ascloud:
        weights = hylite.HyCloud(scenes[0].cloud.xyz, bands=np.nan_to_num(weights, nan=0, posinf=0))
        weights.set_band_names([s.name for s in scenes])
    return weights


import os
import numpy as np
from scipy.sparse import coo_matrix, csc_matrix, csr_matrix
import hylite
from tempfile import mkdtemp
import shutil
from tqdm import tqdm
from pathlib import Path


def blend_scenes(scenes, weights, bands=(0, -1), chunksize=25, trim=False, ooc=True, vb=True):
    """
    Blend together collections of scenes that reference the same underlying cloud to create a fused
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.