SatPy - Part 2: Exploring Advanced Meteorological Imager (AMI)

For detailed documentation, visit SatPy Documentation.

SatPy is a powerful library for **reading**, **manipulating**, and **displaying** data from remote sensors, primarily related to meteorology. It also provides the capability to **save** this data as images or in various formats.

SatPy excels at generating images with **individual channels or bands** and creating **RGB composites** directly from satellite instrument data.

The pyresample library is used for **resampling data** in different areas with specific projections or uniform grids.

Additionally, Satpy offers various **atmospheric corrections** and **visual enhancements**, either directly within Satpy or through the PySpectral and TrollImage packages.

Advanced Meteorological Imager (AMI):

https://space.oscar.wmo.int/instruments/view/ami

In [25]:
```
urls2dwn = ['https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_am
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_ir09
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_ir10
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_ir11
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_ir12
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_ir13
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_nr01
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_nr01
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_sw03
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_vi00
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_vi00
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_vi00
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_vi00
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_wv06
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_wv06
 'https://noaa-gk2a-pds.s3.amazonaws.com/AMI/L1B/LA/202403/02/16/gk2a_ami_le1b_wv07
```

In [26]:
```
import requests
import os

# Specify the local directory where you want to save the files.
# local_directory = input("Enter the path to the download folder: ")
local_directory = "Output_data"
# Ensure that the local directory exists; create it if it doesn't.
os.makedirs(local_directory, exist_ok=True)
```

```python
# Iterate through the URLs and download files.
for urld in urls2dwn:
    # Extract the filename from the URL.
    ntw = urld.split('/')[-1]

    # Construct the complete path to save the file in the local directory.
    file_path = os.path.join(local_directory, ntw)

    # Send an HTTP GET request to the URL.
    resp = requests.get(urld)

    # Check if the response is successful (status code 200).
    if resp.status_code == 200:
        # Write the content to the file in binary mode.
        with open(file_path, "wb") as file:
            file.write(resp.content)
        print(f"File '{ntw}' downloaded and saved to '{local_directory}'.")
    else:
        print(f"Failed to download '{ntw}' from the URL: {urld}")
```

```
File 'gk2a_ami_le1b_ir087_la020ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_ir096_la020ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_ir105_la020ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_ir112_la020ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_ir123_la020ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_ir133_la020ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_nr013_la020ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_nr016_la020ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_sw038_la020ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_vi004_la010ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_vi005_la010ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_vi006_la005ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_vi008_la010ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_wv063_la020ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_wv069_la020ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
File 'gk2a_ami_le1b_wv073_la020ge_202403021658.nc' downloaded and saved to 'Output_d
ata'.
```

Loading and Visualizing Satellite Data

In [27]:
```
# Importing the warnings module and setting it to ignore all warnings.
# This is useful to prevent unnecessary warning messages from cluttering the notebo
import warnings
warnings.filterwarnings('ignore')
```

In [29]:
```
from satpy.scene import Scene
# Importing Scene from the satpy module. Scene is used to represent satellite data
# and allows for operations like reading, resampling, compositing, and saving data.

from satpy import find_files_and_readers
# Importing find_files_and_readers from satpy. This function is used to automatical
# locate satellite data files and determine the appropriate reader based on the
# metadata and contents of the files.
```

This line imports the debug_on function from the satpy.utils module, which is used to enable detailed debug logging in Satpy. This can be helpful for troubleshooting and understanding the internal workings of Satpy processes.

In [30]:
```
from satpy.utils import debug_on # Importing debug_on from satpy.utils to enable de
```

Searching for AMI Data

In [6]:
```
from datetime import datetime  # Importing datetime for date and time operations
```

In [33]:
```
fMSGn = find_files_and_readers(start_time = datetime(2024,3,2,16,58),  # Set start
                               #end_time = datetime(2019,7,22,12,59),   # Optional: Se
                               base_dir = 'Output_data',  # Set the base directory for
                               reader = 'ami_l1b')  # Specify the file reader
fMSGn  # Display the found files
```

Out[33]:
```
{'ami_l1b': ['Output_data\\gk2a_ami_le1b_ir087_la020ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_ir096_la020ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_ir105_la020ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_ir112_la020ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_ir123_la020ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_ir133_la020ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_nr013_la020ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_nr016_la020ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_sw038_la020ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_vi004_la010ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_vi005_la010ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_vi006_la005ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_vi008_la010ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_wv063_la020ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_wv069_la020ge_202403021658.nc',
  'Output_data\\gk2a_ami_le1b_wv073_la020ge_202403021658.nc']}
```

In [34]:
```
from glob import glob  # Import the glob module to find files using pattern matchin
```

In [35]:
```
fnames = glob('/home/jhbravo/input/AMI/*.nc')
# List all files matching the specified pattern in the directory, useful for handli
```

```
# scn = Scene(reader = 'seviri_l1b_native', filenames = fnames)  # Create a Scene o
```

In [36]:
```
fnames = glob('/home/jhbravo/input/AMI/*.nc')
# The above line uses the glob module to find all files in the specified directory
# that match the given pattern (i.e., all files with a .nat extension from a specif
```

> **SatPy** always expects the original file names!

So, do not change them when saving the data on your local machine. Otherwise, SatPy will not be able to open the files.

In [37]:
```
# Creating a Scene object using the file information gathered by find_files_and_rea
scn = Scene(fMSGn)
```

In [38]:
```
# Accessing the attributes of the Scene object to retrieve metadata and other infor
scn.attrs
```

Out[38]:
```
{}
```

In [39]:
```
# Creating a Scene object from the filenames specified in fMSGn
scn = Scene(filenames = fMSGn)
```

In [40]:
```
# Accessing the attributes of the Scene object to view metadata information
scn_attrs = scn.attrs
```

In [41]:
```
# Retrieve and print all available dataset names in the scene
dataset_names = scn.all_dataset_names()
print(dataset_names)
```

```
['IR087', 'IR096', 'IR105', 'IR112', 'IR123', 'IR133', 'NR013', 'NR016', 'SW038', 'V
I004', 'VI005', 'VI006', 'VI008', 'WV063', 'WV069', 'WV073']
```

The `scn.load(['IR_108'], upper_right_corner='NE')` line in ther code bellow is used for loading a specific dataset from a satellite scene in Satpy. Let's break down what each part does:

1. `scn` : This is your Satpy `Scene` object, which contains data from satellite files that you've previously loaded.
2. `.load()` : This method is used to load specific datasets from the satellite files into memory, making them ready for processing and analysis.
3. `['IR_108']` : This is a list containing the names of the datasets you want to load. In this case, you're loading the dataset named `IR_108` , which typically refers to infrared imagery at a wavelength of 10.8 micrometers. This wavelength is often used for cloud imaging, among other applications.
4. `upper_right_corner='NE'` : This parameter specifies how the data should be oriented when loaded. `NE` means that the upper right corner of the data should be in

the northeast. This can be important for getting the geographical orientation correct, especially when dealing with global or hemispherical datasets.

After running this line, the `IR_108` dataset will be loaded into your scene and ready for further processing, such as visualization or analysis.

In [42]:
```
scn.load(['IR105'], upper_right_corner='NE')
```

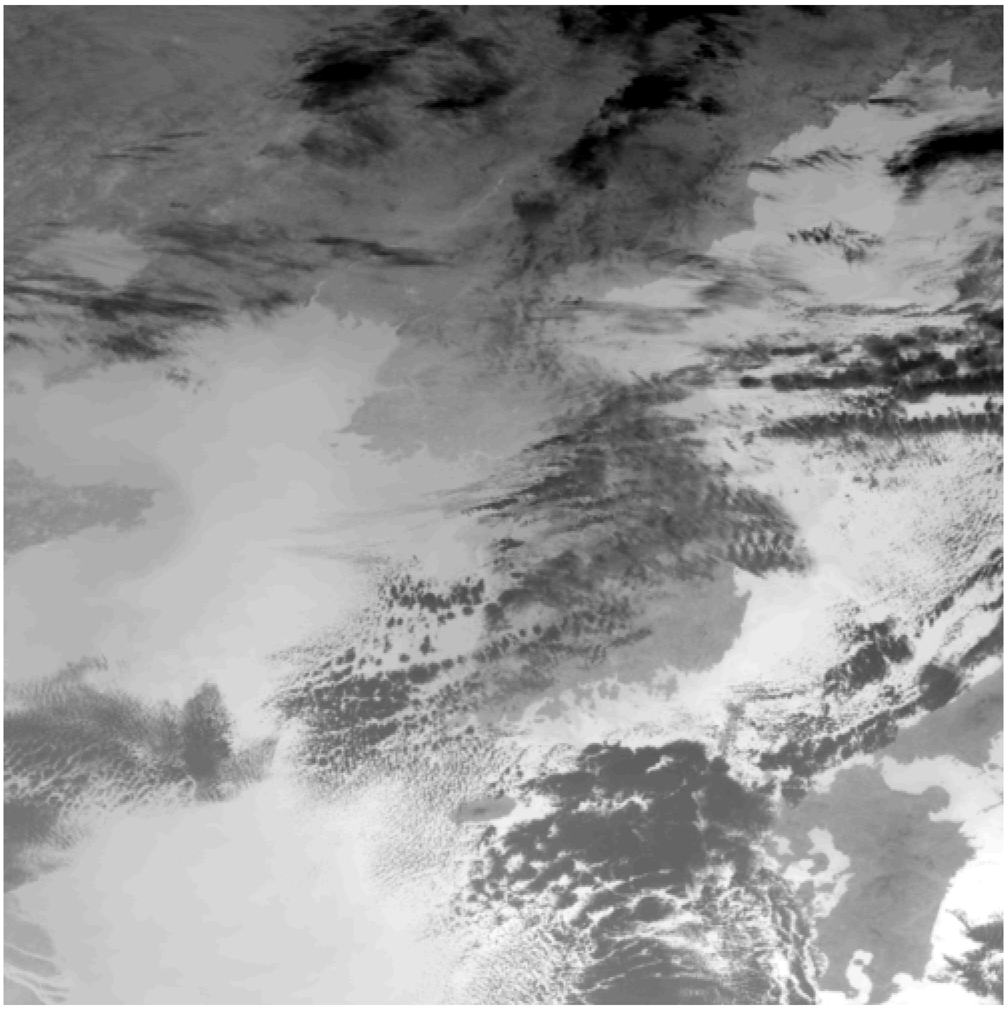
The scn.keys() method in Satpy is used to retrieve a list of all the dataset keys available in the currently loaded Scene object.

In [43]:
```
scn.keys()
```

Out[43]:
```
[DataID(name='IR105', wavelength=WavelengthRange(min=10.115, central=10.35, max=1
0.585, unit='µm'), resolution=2000, calibration=<2>, modifiers=())]
```

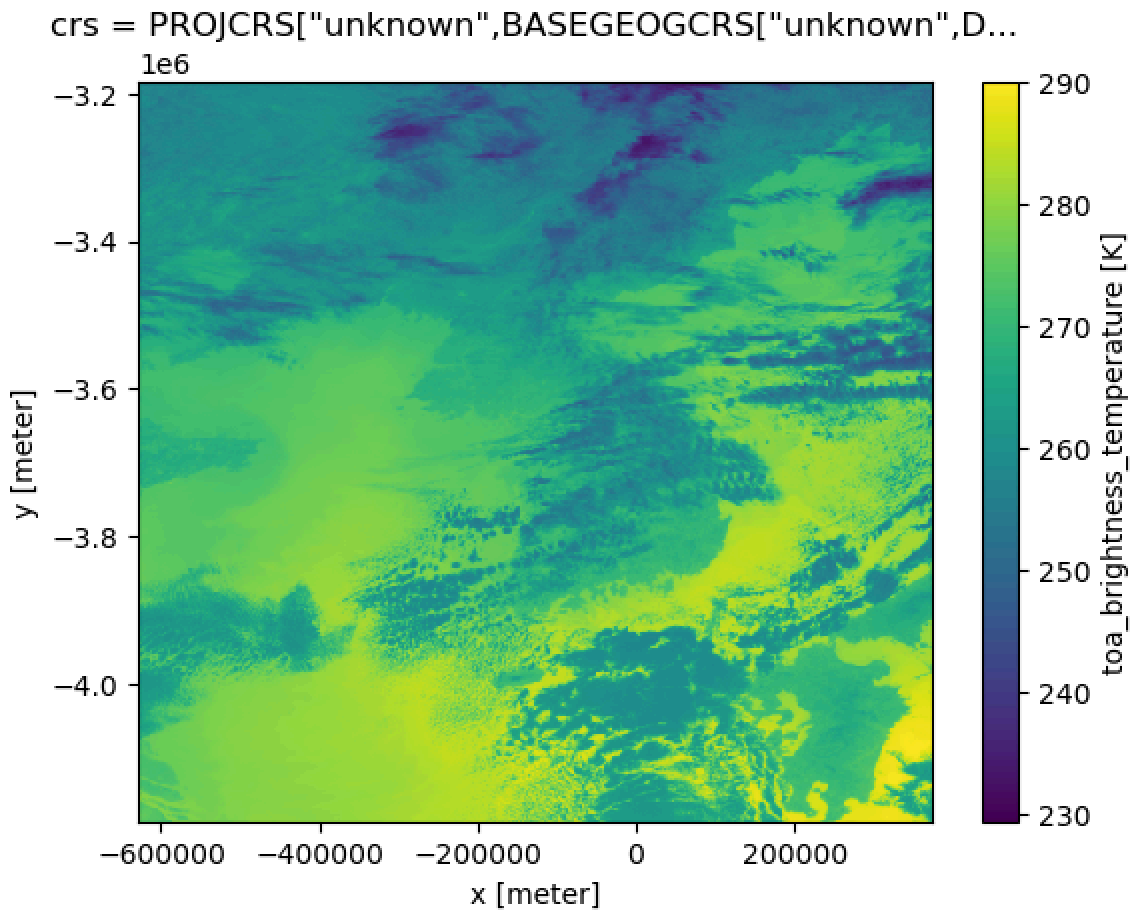
In [44]:
```
# Show the 'IR_108' dataset using Satpy's built-in visualization capabilities
scn.show('IR105')
```

Out[44]:



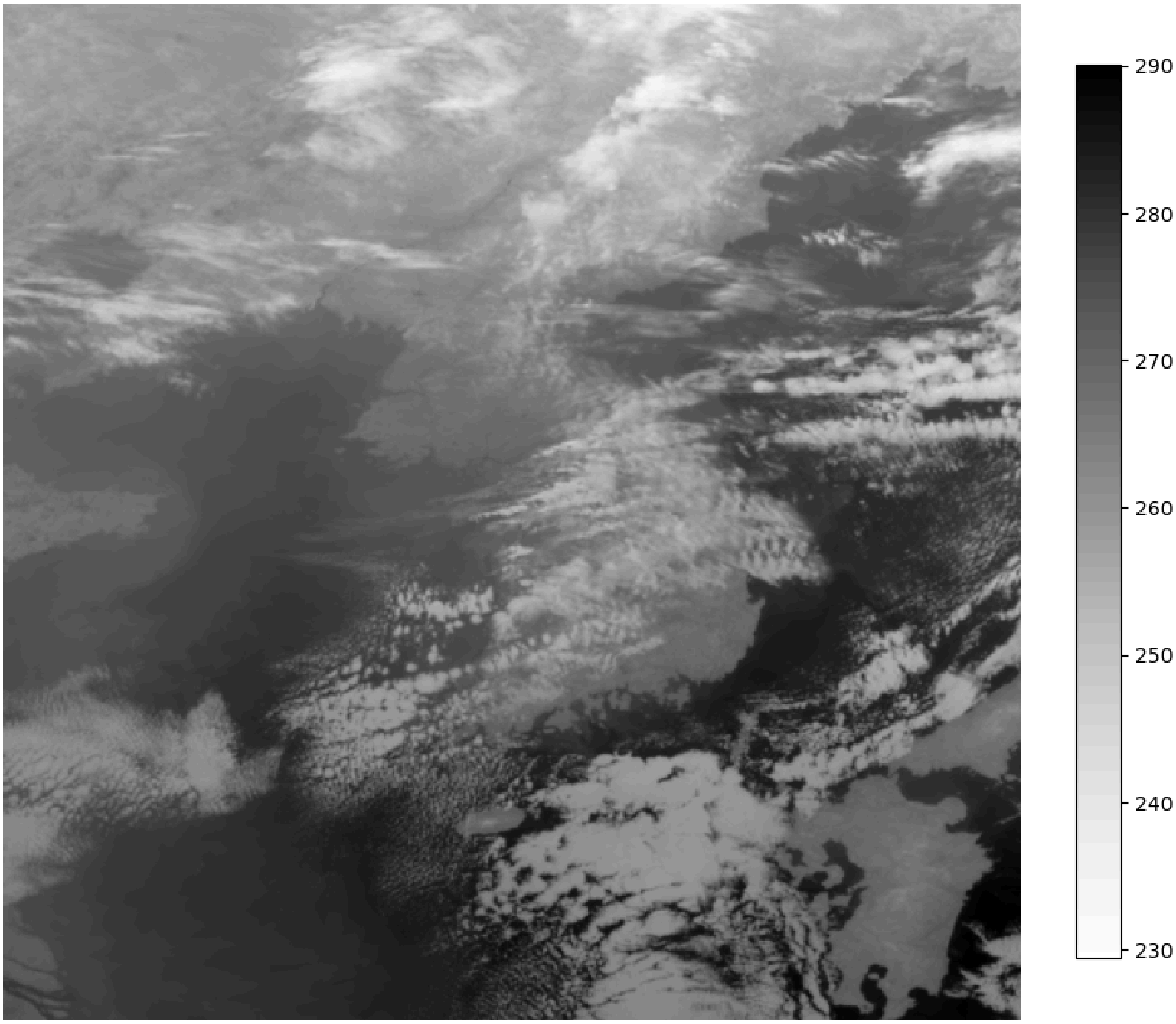
In [45]:
```
%matplotlib inline
# Plot the 'IR_108' channel using matplotlib's imshow function
scn['IR105'].plot.imshow()
```

Out[45]: <matplotlib.image.AxesImage at 0x27ded748bc0>



In [46]:
```
# Import the matplotlib library for creating visualizations in Python
import matplotlib.pyplot as plt
```

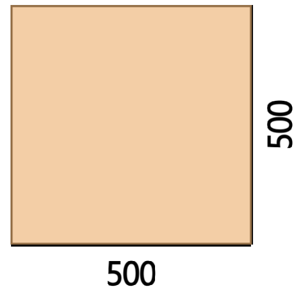
In [47]:
```
# Create a figure and an axes object with specified figure size
fig, ax = plt.subplots(figsize=(10,10))

# Display the data from the 'IR_108' channel of the satellite scene using a graysca
plt.imshow(scn['IR105'].values, cmap="Greys")

# Hide the axis labels and ticks to focus on the image only
ax.set_axis_off()

# Add a colorbar to the plot with a fraction size of the plot, useful for scale/ref
plt.colorbar(fraction=.04)

# Display the figure with all its components
plt.show()
```



In [48]:
```
# Accessing the 'IR_108' dataset from the Scene object 'scn'
# - 'scn' is a Scene object, used for handling satellite data.
# - 'IR_108' specifies an infrared channel at 10.8 micrometers.
# This dataset is used for cloud imaging, surface temperature, and atmospheric anal
scn['IR105']
```

Out[48]: xarray.DataArray 'image_pixel_values' (**y**: 500, **x**: 500)

	Array	Chunk
Bytes	0.95 MiB	0.95 MiB
Shape	(500, 500)	(500, 500)
Dask graph	1 chunks in 11 graph layers	
Data type	float32 numpy.ndarray	



▼ Coordinates:

crs	()	object	PROJCRS["unknown",BASEGEOGCRS["u...
y	(y)	float64	-3.186e+06 ... -4.186e+06
x	(x)	float64	-6.279e+05 -6.259e+05 ... 3.721e+05

► Indexes: (2)

► Attributes: (30)

Radiance Calibration

In [49]:
```
# Load the data for the 10.8μm band.
# The parameter [10.8] specifies the wavelength of the band in micrometers.
scn.load([10.5],

         # Specify the calibration to radiance values.
         # "radiance" calibration converts the data to radiometric units (mW m-2 sr
         calibration=["radiance"],

         # Set the orientation of the image.
         # "upper_right_corner='NE'" aligns the image with its upper right corner t
         upper_right_corner='NE')
```

Brightness Temperature Calibration

In [50]:
```
# Load the data for the 10.8μm band again, this time for a different calibration.
scn.load([10.5],

         # Specify the calibration to brightness temperatures.
         # "brightness_temperature" calibration converts the data to temperature un
         calibration=["brightness_temperature"],

         # Maintain the same orientation as before.
         upper_right_corner='NE')
```

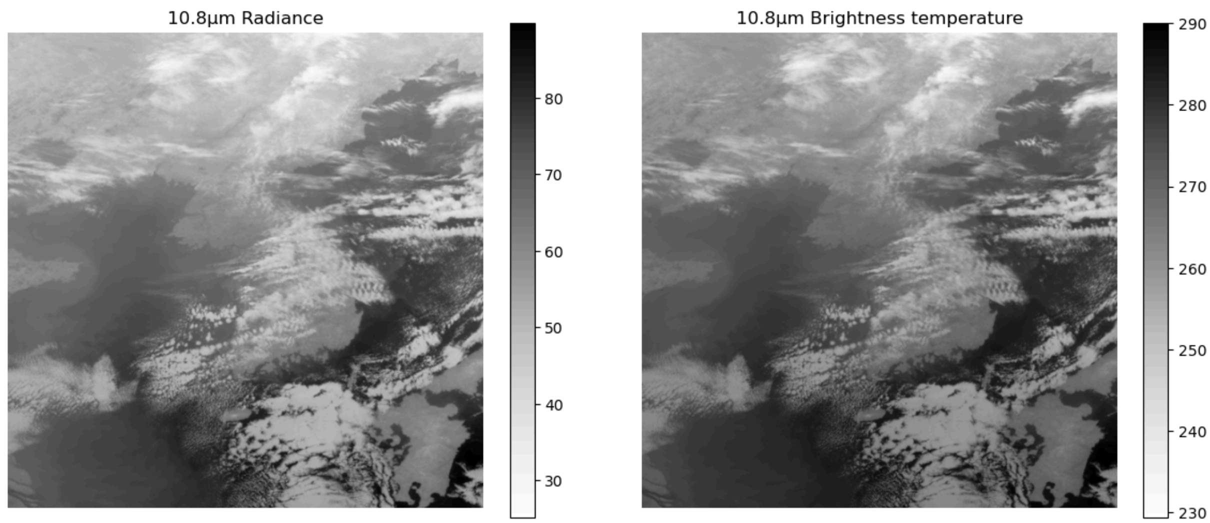
In [51]:
```
# List all the datasets currently loaded into the Scene object 'scn'
available_datasets = scn.keys()
print(available_datasets)
```

```
[DataID(name='IR105', wavelength=WavelengthRange(min=10.115, central=10.35, max=10.5
85, unit='µm'), resolution=2000, calibration=<2>, modifiers=()), DataID(name='IR10
5', wavelength=WavelengthRange(min=10.115, central=10.35, max=10.585, unit='µm'), re
solution=2000, calibration=<3>, modifiers=())]
```

In [52]:
```
# Convert the keys view to a list to enable indexing
keys_list = list(scn.keys())

# Access the second dataset key
second_dataset_key = keys_list[1]

# Extract the central wavelength from the 'wavelength' attribute of the second data
central_wavelength = second_dataset_key['wavelength'][1]

# Print the central wavelength and the key for the second dataset
print("Central Wavelength:", central_wavelength, "µm")
print("Second Dataset Key:", second_dataset_key)
```

```
Central Wavelength: 10.35 µm
Second Dataset Key: DataID(name='IR105', wavelength=WavelengthRange(min=10.115, cent
ral=10.35, max=10.585, unit='µm'), resolution=2000, calibration=<3>, modifiers=())
```

In [53]:
```
# Create a figure and two subplot axes, arranged horizontally, with specified size
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14,6))

# Display the values of the second dataset in the Scene on the first subplot (ax1)
im1 = ax1.imshow(scn[scn.keys()[1]].values, cmap="Greys")
# Remove the axis labels and ticks for ax1
ax1.set_axis_off()

# Display the values of the first dataset in the Scene on the second subplot (ax2)
im2 = ax2.imshow(scn[scn.keys()[0]].values, cmap="Greys")
# Remove the axis labels and ticks for ax2
ax2.set_axis_off()

# Set the title for ax2
ax2.set_title("10.8µm Brightness temperature")
# Set the title for ax1
ax1.set_title("10.8µm Radiance")

# Add a colorbar next to ax1, adjusting its size
fig.colorbar(im1, ax=ax1, fraction=.05)
# Add a colorbar next to ax2, adjusting its size
fig.colorbar(im2, ax=ax2, fraction=.05)

# Display the figure
plt.show()
```



In [54]:
```
# Access the x-coordinates (longitude values) of the 'IR_108' data array in the Sce
scn['IR105'].x
```

Out[54]: xarray.DataArray 'x' (**x**: 500)

array([-627862.965097, -625858.956769, -623854.948441, ..., 368129.173919,
370133.182247, 372137.190575])

▼ Coordinates:

crs	()	object	PROJCRS["unknown",BASEGEOGCRS["u...
x	(x)	float64	-6.279e+05 -6.259e+05 ... 3.721e+05

► Indexes: (1)

▼ Attributes:

units : meter

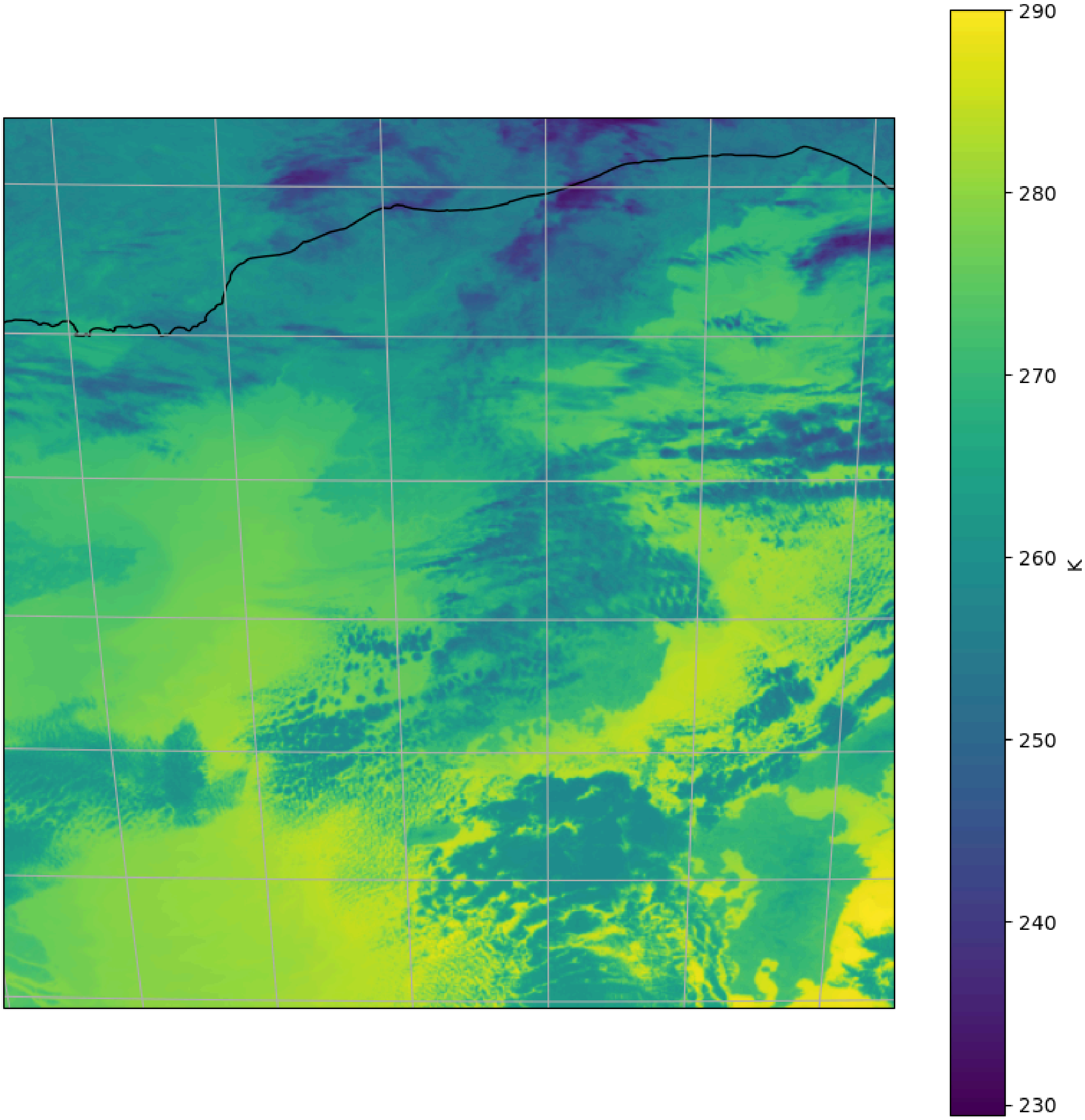
In [55]:
```
# define palette (matplotlib style)
cmap = ['#ffffff', '#ffffff', '#ffffff', '#ffffff', '#ffffff', '#b6ffb6', '#79ff79'
        '#0028a2', '#000079', '#fbfb00', '#e7e700', '#d2d200', '#baba00', '#a6a600'
        '#aaaaaa', '#a6a6a6', '#9e9e9e', '#969696', '#8e8e8e', '#868686', '#7d7d7d'
        '#313131', '#282828', '#202020', '#181818', '#141414', '#000000', '#000000'
```

In [56]:
```
# Retrieve the area definition (spatial reference) associated with the 'IR_108' dat
area_def = scn['IR105'].attrs['area']
```

In [57]:
```
# Convert the area definition to a Cartopy Coordinate Reference System (CRS) object
cartopy_crs = area_def.to_cartopy_crs()
```

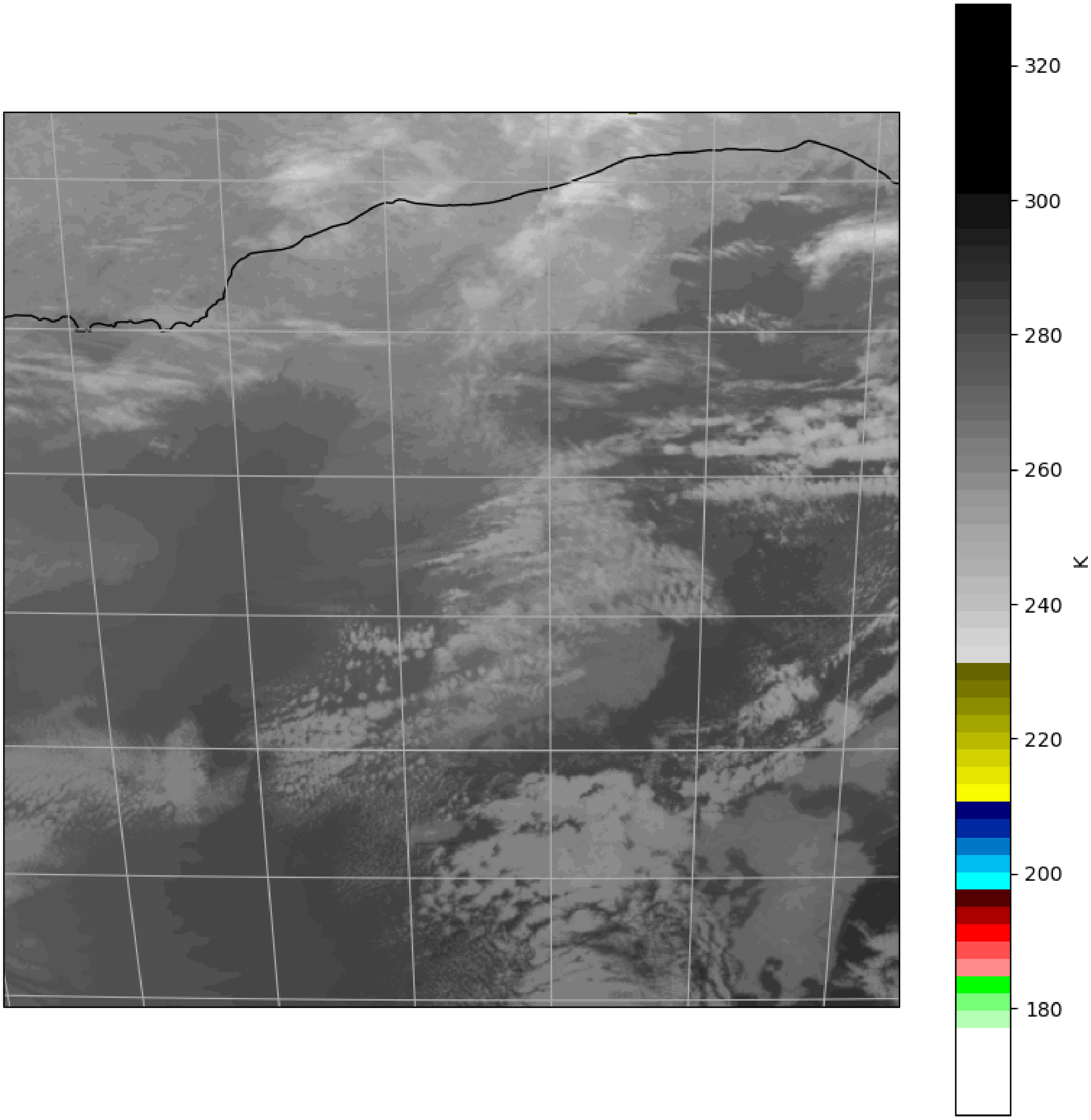
In [58]:
```
# Convert the area definition to a Cartopy CRS (Coordinate Reference System) object
crs = area_def.to_cartopy_crs()

# Create a figure with specific size
fig = plt.figure(figsize=(10,10))
```

```
# Add axes to the figure with the specified projection (crs)
ax = plt.axes(projection=crs)

# Draw coastlines on the map for reference
ax.coastlines()

# Add gridlines to the map
ax.gridlines()

# Display the 'IR_108' data as an image, using the converted Cartopy CRS for correc
plt.imshow(scn['IR105'], transform=crs, extent=crs.bounds, origin='upper')

# Add a color bar to the plot, labeling it with the data's unit of measurement
plt.colorbar(label=scn['IR105'].attrs['units'])

# Display the plot
plt.show()
```



In [59]:
```
# Import the NumPy library, which provides support for large, multi-dimensional arr
import numpy as np
```

In [60]:
```
# Create an array of levels for the color map, spanning from -109 to 56, with the s
levels = np.linspace(-109, 56, num=len(cmap))

# Create a BoundaryNorm object for the color map to ensure proper coloring based on
norm = plt.cm.colors.BoundaryNorm(levels, len(levels))

# Create a ListedColormap using the 'cmap' colors and the custom norm
irmap = plt.cm.colors.ListedColormap(cmap)

# Display results

# Convert the area definition to a Cartopy CRS (Coordinate Reference System) object
crs = area_def.to_cartopy_crs()

# Create a figure with specific size
fig = plt.figure(figsize=(10,10))

# Add axes to the figure with the specified projection (crs)
ax = plt.axes(projection=crs)

# Draw coastlines on the map for reference
ax.coastlines()

# Add gridlines to the map
ax.gridlines()

# Display the 'IR_108' data as an image, using the converted Cartopy CRS for correc
# Set the minimum and maximum values for the color map (vmin and vmax)
# Use the custom 'irmap' color map and apply the 'norm' for color scaling
plt.imshow(scn['IR105'], transform=crs, extent=crs.bounds, origin='upper', vmin=-10

# Add a color bar to the plot, labeling it with the data's unit of measurement
plt.colorbar(label=scn['IR105'].attrs['units'])

# Display the plot
plt.show()
```


320
300
280
260
240
220
200
180
K