

# Advanced Spectral Imaging: Homework 3

Yuya Haga

December 26, 2024

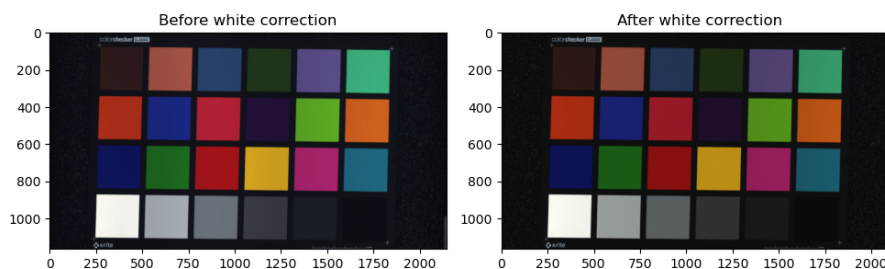
## 1 Tasks #1. White correction

### 1.1 Colorchecker [from Specim scanner]

White correction of specim scanner is shown in Figure 1. The white correction is done by using a white reference and a dark reference.

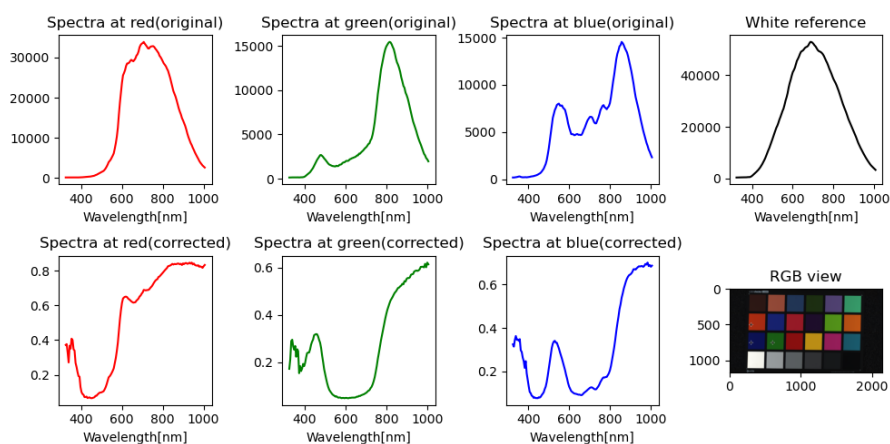
Script for white correction is shown in Code 1.

Figure 1: White correction of specim scanner



Spectra of white corrected image is shown in Figure 2.

Figure 2: Spectra of white corrected image

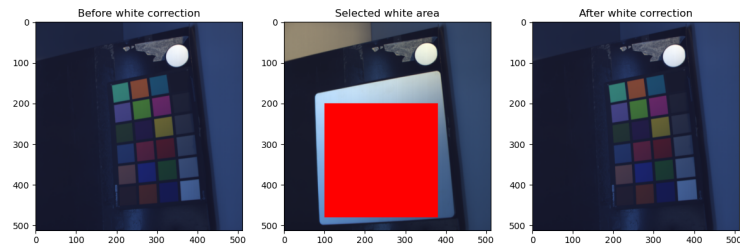


## 1.2 Color Checker 2 lamps + White Sample 2 lamps

### 1.2.1 SpecimIQ

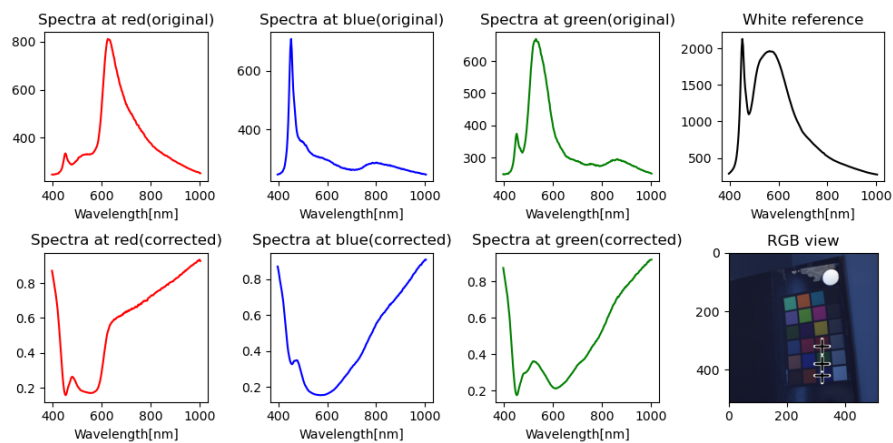
White correction of SpecimIQ is shown in Figure 3.

Figure 3: White correction of SpecimIQ with large reference



Spectra of white corrected image is shown in Figure 4.

Figure 4: Spectra of white corrected image

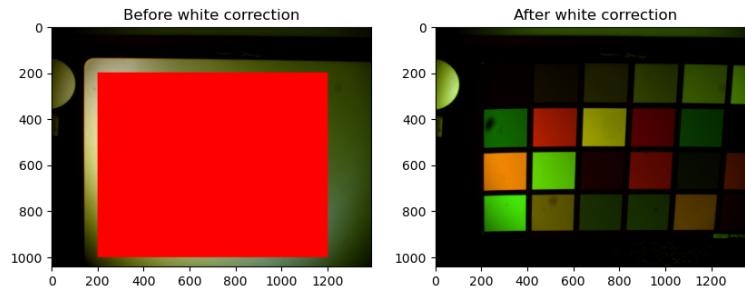


Figures were generated with the script shown in Code 7.

### 1.2.2 Nuance camera

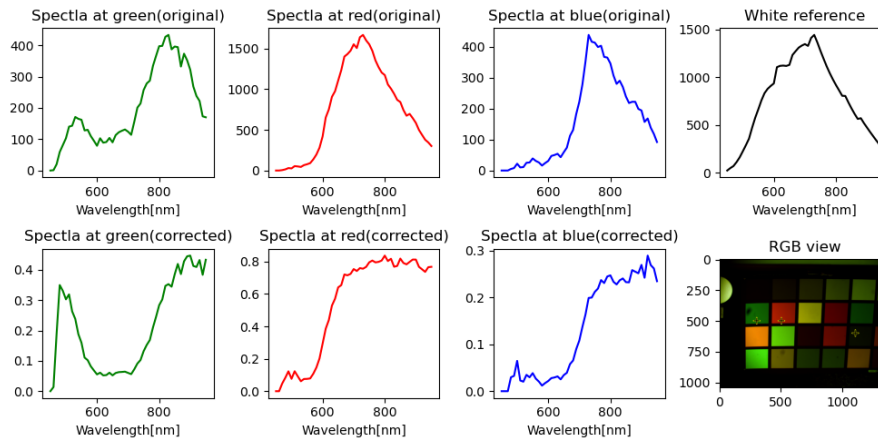
White correction of nuance camera is shown in Figure 5.

Figure 5: White correction of nuance camera with large reference



Spectra of white corrected image is shown in Figure 6.

Figure 6: Spectra of white corrected image



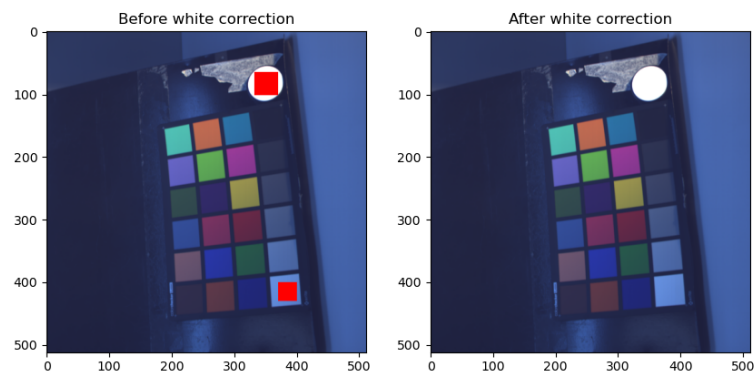
Nuance image was loaded with the script shown in Code 4. Figures were generated with the script shown in Code 3.

### 1.3 Color Checker 2 lamps using left and right white samples inside the image

#### 1.3.1 SpecimIQ

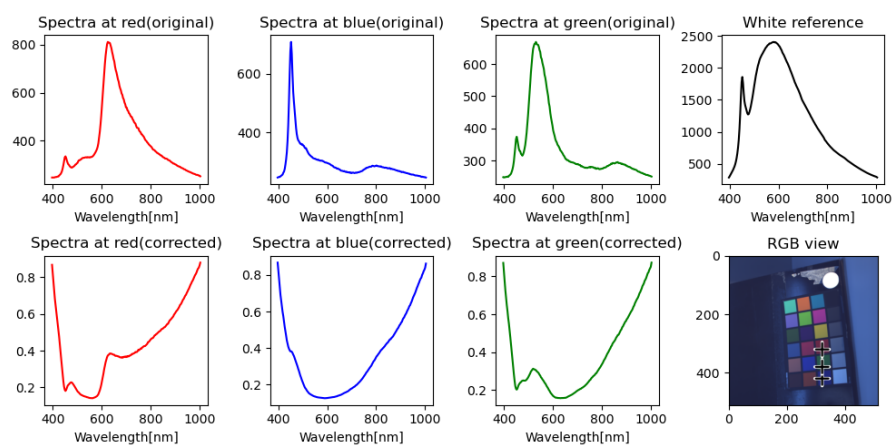
White correction of SpecimIQ is shown in Figure 7.

Figure 7: White correction of SpecimIQ



Spectra of white corrected image is shown in Figure 8.

Figure 8: Spectra of white corrected image

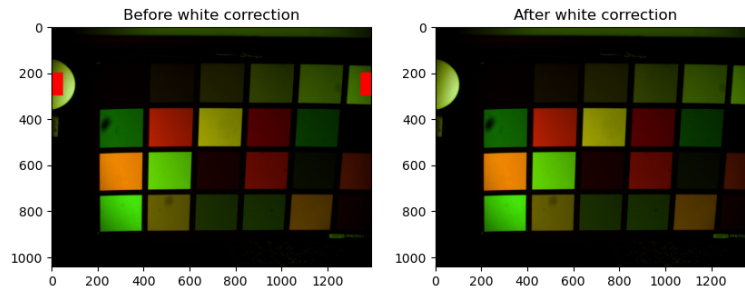


Figures were generated with the script shown in Code 6.

### 1.3.2 Nuance camera

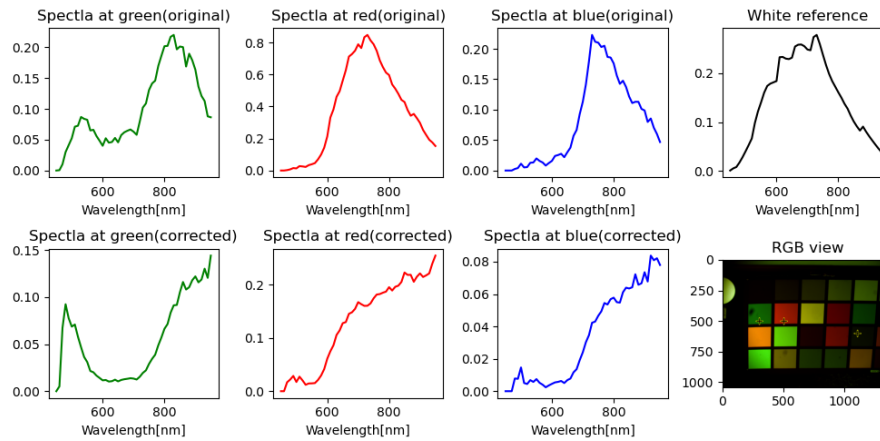
White correction of nuance camera is shown in Figure 9.

Figure 9: White correction of nuance camera



Spectra of white corrected image is shown in Figure 10.

Figure 10: Spectra of white corrected image

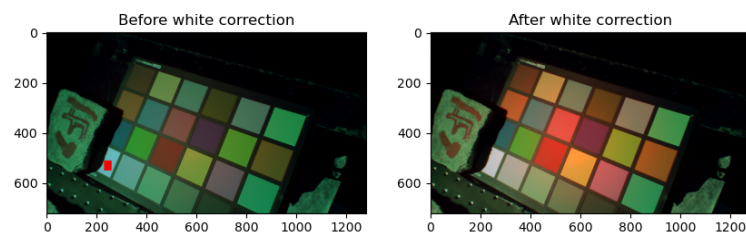


Nuance image was loaded with the script shown in Code 4. Figures were generated with the script shown in Code 2.

#### 1.4 Tunable light source

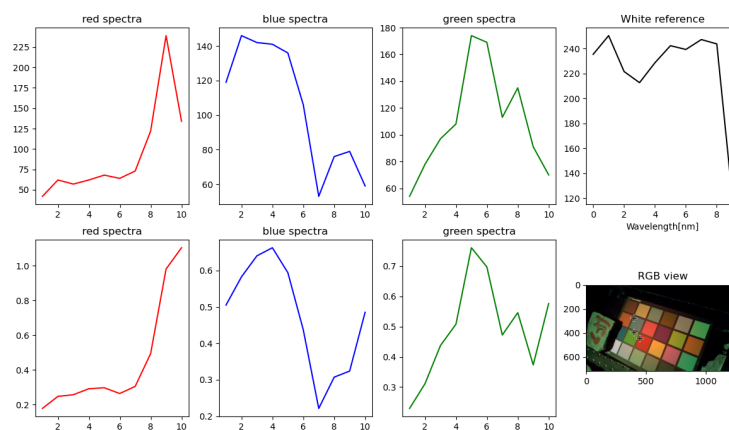
White correction of tunable light camera is shown in Figure 11.

Figure 11: White correction of tunable light source



Spectra of white corrected image is shown in Figure 12

Figure 12: Spectra of white corrected image



Nuance image was loaded with the script shown in Code 8. Figures were generated with the script shown in Code 9.

## 2 Tasks #2. Nuance camera

## 3 Tasks #3. Build ENVI spectral image from Tunable light source

## 4 Code

```
1 import matplotlib.pyplot as plt
2
3 from asi import path_config
4 from asi.draw import draw_multi_crosss, reconstruct_rgb_envi
5 from asi.io.load_envi import load_spectral_image
```

```

6 from asi.preprocess import load_white_corrected
7 from asi.utils import get_wavelengths
8
9 session1_root = path_config.measurements / "Session1"
10 spec_path = session1_root / "Specim_scanner/Color_checker_8_binning/capture"
11
12 fig, axes = plt.subplots(1, 2, figsize=(10, 5))
13
14 colorchecker_path = spec_path / "solutions_scan_0110"
15 whiteref_path = spec_path / "WHITEREF_solutions_scan_0110"
16 darkref_path = spec_path / "DARKREF_solutions_scan_0110"
17
18 spectral_image, envi_header = load_spectral_image(colorchecker_path)
19 rgb_view = reconstruct_rgb_envi(spectral_image, envi_header)
20 axes[0].imshow(rgb_view)
21 axes[0].set_title("Before white correction")
22
23 spectral_image, envi_header = load_white_corrected(
24     colorchecker_path,
25     whiteref_path,
26     darkref_path,
27 )
28 rgb_view = reconstruct_rgb_envi(spectral_image, envi_header)
29
30 axes[1].imshow(rgb_view)
31 axes[1].set_title("After white correction")
32 plt.show()
33
34 # Show spectra
35 colors = ["r", "g"]
36 positions = [(300, 500), (400, 1000)]
37 canvas = draw_multi_crosss(rgb_view, positions)
38
39 plt.rcParams["figure.dpi"] = 100
40 fig, axes = plt.subplots(1, 3, figsize=(10, 5), tight_layout=True)
41
42 wavelength = get_wavelengths(envi_header)
43 for pos, color, ax in zip(positions, colors, axes[:2]):
44     ax.plot(wavelength, spectral_image[pos[1], pos[0], :], color=color)
45     ax.set_title(f"Spectrum at {pos[0]}th pixel")
46     ax.set_xlabel("Wavelength[nm]")
47
48 axes[2].imshow(canvas)
49 axes[2].set_title("RGB view of spectral image")
50
51 plt.show()

```

Code 1: White correction for Specium Scanner

```

1 import numpy as np
2 from matplotlib import pyplot as plt
3
4 from asi import path_config
5 from asi.draw import draw_multi_crosss, reconstruct_rgb, select_area
6 from asi.io.load_nuance import load_nuance_image
7
8 session1 = path_config.measurements / "session1"
9 nuance = session1 / "Nuance"

```

```

10
11
12 # White correction with small reference
13
14 fig, axes = plt.subplots(1, 2, figsize=(10, 5))
15 root = nuance / "colorchecker 2lights"
16
17 white_pos_list = [
18     (slice(200, 300), slice(0, 50)),
19     (slice(200, 300), slice(-50, -1)),
20 ]
21
22
23 spectral_image, wavelengths = load_nuance_image(root)
24 spectral_image = spectral_image.astype(np.float64)
25 spectral_image /= spectral_image.max()
26
27 original_rgb = reconstruct_rgb(spectral_image, wavelengths)
28 for white_pos in white_pos_list:
29     original_rgb = select_area(original_rgb, white_pos)
30 original_rgb /= original_rgb.max()
31 original_rgb = original_rgb.clip(0, 1)
32 axes[0].imshow(original_rgb)
33 axes[0].set_title("Before white correction")
34
35 white_sq_list = []
36
37 # White correction with selected area
38 for white_pos in white_pos_list:
39     white_sq = spectral_image[white_pos]
40     a, b, band = white_sq.shape
41     white_sq = white_sq.reshape(a * b, band)
42
43     # replace nonzero elements with minimum value
44     nonzero_elements = white_sq[white_sq != 0]
45     min_elm = nonzero_elements.min()
46     white_sq_list.append(white_sq)
47
48 white_sq = np.vstack(white_sq_list)
49 whiteref = white_sq.mean(axis=0)
50
51 # spectral_image *= 1.5
52 # spectral_image = spectral_image.clip(0, 1)
53 white_corrected = spectral_image / whiteref
54 white_corrected /= white_corrected.max()
55 white_corrected = white_corrected.clip(0, 1)
56 del original_rgb
57
58 white_corrected_rgb_view = reconstruct_rgb(white_corrected, wavelengths)
59 axes[1].imshow(white_corrected_rgb_view)
60 axes[1].set_title("After white correction")
61
62 plt.show()
63
64 # Show spectra
65 colors = ["g", "r", "b"]
66 color_names = ["green", "red", "blue"]
67 positions = [(300, 500), (500, 500), (1100, 600)]

```



```

68
69
70 plt.rcParams["figure.dpi"] = 100
71 fig, axes = plt.subplots(2, 4, figsize=(10, 5), tight_layout=True)
72 print(axes)
73
74 axes1, axes2 = axes
75 for pos, color, color_name, ax in zip(positions, colors, color_names, axes2):
76     print(ax)
77     ax.plot(wavelengths, white_corrected[pos[1], pos[0], :], color=color)
78     ax.set_title(f"Spectra at {color_name}(corrected)")
79     ax.set_xlabel("Wavelength[nm]")
80
81 for pos, color, color_name, ax in zip(positions, colors, color_names, axes1):
82     print(ax)
83     ax.plot(wavelengths, spectral_image[pos[1], pos[0], :], color=color)
84     ax.set_title(f"Spectra at {color_name}(original)")
85     ax.set_xlabel("Wavelength[nm]")
86
87
88 axes1[-1].plot(wavelengths, whiteref, color="k")
89 axes1[-1].set_title("White reference")
90 axes1[-1].set_xlabel("Wavelength[nm]")
91
92 canvas = draw_multi_crosss(white_corrected_rgb_view, positions)
93 axes2[-1].imshow(canvas)
94 axes2[-1].set_title("RGB view")
95
96 plt.show()

```

Code 2: White correction for Nuance Camera with small reference

```

1 import numpy as np
2 from matplotlib import pyplot as plt
3
4 from asi import path_config
5 from asi.draw import draw_multi_crosss, reconstruct_rgb, select_area
6 from asi.io.load_nuance import load_nuance_image
7
8 session1 = path_config.measurements / "session1"
9 nuance = session1 / "Nuance"
10
11 # White correction with large reference
12
13 # load white image
14
15 root = nuance / "white 2lights"
16
17 white_image, wavelengths = load_nuance_image(root)
18 white_image = white_image.astype(np.float64)
19
20
21 fig, axes = plt.subplots(1, 2, figsize=(10, 5))
22
23 white_pos = (slice(200, 1000), slice(200, 1200))
24
25
26 white_rgb = reconstruct_rgb(white_image, wavelengths)

```

```

27 white_rgb = select_area(white_rgb, white_pos)
28 white_rgb /= white_rgb.max()
29 white_rgb = white_rgb.clip(0, 1)
30 axes[0].imshow(white_rgb)
31 axes[0].set_title("Before white correction")
32
33
34 # White correction with selected area
35 white_sq = white_image[white_pos]
36
37 # replace nonzero elements with minimum value
38 nonzero_elements = white_sq[white_sq != 0]
39 min_elm = nonzero_elements.min()
40 white_sq = white_sq.clip(min_elm, None)
41
42
43 # load spectral image
44 root = nuance / "colorchecker 2lights"
45 spectral_image, wavelengths = load_nuance_image(root)
46
47 # apply white correction
48 whiteref = white_sq.mean((0, 1))
49 white_corrected = spectral_image / whiteref
50 white_corrected /= white_corrected.max()
51 white_corrected = white_corrected.clip(0, 1)
52 del white_rgb
53
54 white_corrected_rgb_view = reconstruct_rgb(white_corrected, wavelengths)
55 axes[1].imshow(white_corrected_rgb_view)
56 axes[1].set_title("After white correction")
57
58 plt.show()
59
60 # Show spectra
61 colors = ["g", "r", "b"]
62 color_names = ["green", "red", "blue"]
63 positions = [(300, 500), (500, 500), (1100, 600)]
64
65
66 plt.rcParams["figure.dpi"] = 100
67 fig, axes = plt.subplots(2, 4, figsize=(10, 5), tight_layout=True)
68 print(axes)
69
70 axes1, axes2 = axes
71 for pos, color, color_name, ax in zip(positions, colors, color_names, axes2):
72     print(ax)
73     ax.plot(wavelengths, white_corrected[pos[1], pos[0], :], color=color)
74     ax.set_title(f"Spectra at {color_name}(corrected)")
75     ax.set_xlabel("Wavelength[nm]")
76
77 for pos, color, color_name, ax in zip(positions, colors, color_names, axes1):
78     print(ax)
79     ax.plot(wavelengths, spectral_image[pos[1], pos[0], :], color=color)
80     ax.set_title(f"Spectra at {color_name}(original)")
81     ax.set_xlabel("Wavelength[nm]")
82
83
84 axes1[-1].plot(wavelengths, whiteref, color="k")

```

```

85 axes1[-1].set_title("White reference")
86 axes1[-1].set_xlabel("Wavelength[nm]")
87
88 canvas = draw_multi_crosss(white_corrected_rgb_view, positions)
89 axes2[-1].imshow(canvas)
90 axes2[-1].set_title("RGB view")
91
92 plt.show()

```

Code 3: White correction for Nuance Cmaera with large reference

```

1 from pathlib import Path
2
3 import numpy as np
4 import tifffile as tiff
5 from natsort import natsorted
6
7
8 def load_nuance_image(tiff_root: Path) -> tuple[np.ndarray, list[float]]:
9     wavelengths: list[float] = []
10    imgs = []
11    tiff_list = list(tiff_root.glob("*.tif"))
12    tiff_list = natsorted(tiff_list, reverse=False)
13    for tiff_path in tiff_list:
14        img = tiff.imread(tiff_path)
15        wavelength = float(tiff_path.stem.split("_")[-1])
16        imgs.append(img)
17        wavelengths.append(wavelength)
18
19    spectral_image = np.stack(imgs, axis=-1)
20    spectral_image = spectral_image.astype(np.float64)
21    return spectral_image, wavelengths

```

Code 4: Load Nuance Cmaera

```

1 from pathlib import Path
2
3 import numpy as np
4
5
6 def parse_envi_header(lines: list) -> dict[str, str]:
7     """
8     Parses ENVI file content into a structured dictionary
9     This code was written with Github Copilot
10    """
11    envi_data = {}
12    in_block_key = None
13
14    for line_org in lines:
15        line = line_org.strip()
16
17        # Skip empty lines
18        if not line:
19            continue
20
21        # Handle multiline blocks
22        if in_block_key:
23            if line.endswith("}"):

```

```

24         # Closing multiline
25         envi_data[in_block_key] += line[:-1].strip()
26         in_block_key = None
27     else:
28         # Continue multiline
29         envi_data[in_block_key] += line
30     continue
31
32     # Key-value pair parsing
33     if "=" in line:
34         key, value = map(str.strip, line.split("=", 1))
35         key = key.lower().replace(" ", "_") # Normalize key format
36
37         # Handle block values
38         if value.startswith("{"):
39             # Handle multiline block
40             in_block_key = key
41             # Remove opening '{'
42             value = value[1:].strip()
43             if value.endswith("}"):
44                 # Single-line block
45                 envi_data[key] = value[:-1]
46                 in_block_key = None
47             else:
48                 # Start multiline block
49                 envi_data[key] = value
50         else:
51             # Single-line value
52             envi_data[key] = value
53     return envi_data
54
55
56 def load_envi_header(hdr_file: Path) -> dict[str, str]:
57     """Loads ENVI header file."""
58     with hdr_file.open(encoding="utf-8") as f:
59         header_content = f.readlines()
60         envi_header = parse_envi_header(header_content)
61     return envi_header
62
63
64 def load_spectral_image(file_stem: Path) -> tuple[np.ndarray, dict[str, str]]:
65     """Loads spectral image from ENVI format."""
66     # Load ENVI header
67     hdr_file = file_stem.with_suffix(".hdr")
68     envi_header = load_envi_header(hdr_file)
69
70     # Load parameters
71     interleave = str(envi_header["interleave"])
72     lines = int(envi_header["lines"])
73     samples = int(envi_header["samples"])
74     bands = int(envi_header["bands"])
75     data_type = int(envi_header.get("data type", 12))
76
77     # Map ENVI data type to NumPy dtype
78     data_type_map = {
79         1: np.uint8,
80         2: np.int16,
81         3: np.int32,

```

```

82         4: np.float32,
83         5: np.float64,
84         6: np.complex64,
85         9: np.complex128,
86         12: np.uint16,
87         13: np.uint32,
88         14: np.int64,
89         15: np.uint64,
90     }
91
92     if data_type not in data_type_map:
93         msg = f"Unsupported data type: {data_type}"
94         raise ValueError(msg)
95
96     dtype = data_type_map[data_type]
97
98     # Load raw data
99     raw_file = file_stem.with_suffix(".raw")
100     with open(raw_file, "rb") as f:
101         raw = np.fromfile(f, dtype=dtype)
102
103     # define shape and transpose order by interleave method
104     if interleave.upper() == "BIL":
105         new_shape = (lines, bands, samples)
106         axis_order = (0, 2, 1)
107     elif interleave.upper() == "BIP":
108         new_shape = (lines, samples, bands)
109         axis_order = (0, 1, 2)
110     elif interleave.upper() == "BSQ":
111         new_shape = (bands, samples, lines)
112         axis_order = (0, 2, 1)
113     else:
114         msg = f"Interleave {interleave} not supported."
115         raise ValueError(msg)
116
117     spectral_image = raw.reshape(new_shape)
118     # change axis order to 'lines, samples, bands'
119     spectral_image = np.transpose(spectral_image, axis_order)
120
121     return spectral_image, envi_header

```

Code 5: Load ENVI format images

```

1 import numpy as np
2 from matplotlib import pyplot as plt
3
4 from asi import path_config
5 from asi.draw import draw_multi_crosss, reconstruct_rgb_envi, select_area
6 from asi.io.load_envi import load_spectral_image
7 from asi.preprocess import load_white_corrected
8 from asi.utils import get_wavelengths
9
10 specim_iq_root = path_config.measurements / "Session1" / "SpecimIQ"
11
12 path_404 = specim_iq_root / "404" / "capture"
13
14
15 # White correction with small reference

```

```
16 fig, axes = plt.subplots(1, 2, figsize=(10, 5))
17
18 image_path = path_404 / "404"
19 spectral_image, envi_header = load_spectral_image(image_path)
20
21
22 white_pos_list = [
23     (slice(65, 102), slice(332, 370)),
24     (slice(400, 430), slice(370, 400)),
25 ]
26
27 original_rgb = reconstruct_rgb_envi(spectral_image, envi_header)
28 for white_pos in white_pos_list:
29     original_rgb = select_area(original_rgb, white_pos)
30
31 original_rgb *= 1.5
32 original_rgb = original_rgb.clip(0, 1)
33
34 axes[0].imshow(original_rgb)
35 axes[0].set_title("Before white correction")
36
37 # White correction with selected area
38 white_sq_list = []
39 for white_pos in white_pos_list:
40     white_sq = spectral_image[white_pos]
41     a, b, band = white_sq.shape
42     white_sq = white_sq.reshape(a * b, band)
43
44     # replace nonzero elements with minimum value
45     nonzero_elements = white_sq[white_sq != 0]
46     min_elm = nonzero_elements.min()
47     white_sq_list.append(white_sq)
48
49 white_sq = np.vstack(white_sq_list)
50 whiteref = white_sq.mean(axis=0)
51
52 white_corrected = spectral_image / whiteref
53
54 white_corrected_rgb_view = reconstruct_rgb_envi(white_corrected, envi_header)
55 white_corrected_rgb_view *= 1.5
56 white_corrected_rgb_view = white_corrected_rgb_view.clip(0, 1)
57
58
59 axes[1].imshow(white_corrected_rgb_view)
60 axes[1].set_title("After white correction")
61
62 plt.show()
63
64 # Show spectra
65 colors = ["r", "b", "g"]
66 color_names = ["red", "blue", "green"]
67 positions = [(320, 320), (320, 420), (320, 380)]
68
69 wavelengths = get_wavelengths(envi_header)
70
71
72 fig, axes = plt.subplots(2, 4, figsize=(10, 5), tight_layout=True)
73
```

```

74 axes1, axes2 = axes
75 for pos, color, color_name, ax in zip(positions, colors, color_names, axes2):
76     ax.plot(wavelengths, white_corrected[pos[1], pos[0], :], color=color)
77     ax.set_title(f"Spectra at {color_name}(corrected)")
78     ax.set_xlabel("Wavelength[nm]")
79
80 for pos, color, color_name, ax in zip(positions, colors, color_names, axes1):
81     ax.plot(wavelengths, spectral_image[pos[1], pos[0], :], color=color)
82     ax.set_title(f"Spectra at {color_name}(original)")
83     ax.set_xlabel("Wavelength[nm]")
84
85
86 axes1[-1].plot(wavelengths, whiteref, color="k")
87 axes1[-1].set_title("White reference")
88 axes1[-1].set_xlabel("Wavelength[nm]")
89
90 canvas = draw_multi_crosss(white_corrected_rgb_view, positions)
91 axes2[-1].imshow(canvas)
92 axes2[-1].set_title("RGB view")
93
94 plt.show()

```

Code 6: White correction for SpecimIQ with small reference

```

1 from matplotlib import pyplot as plt
2
3 from asi import path_config
4 from asi.draw import draw_multi_crosss, reconstruct_rgb_envi, select_area
5 from asi.io.load_envi import load_spectral_image
6 from asi.utils import get_wavelengths
7
8 specim_iq_root = path_config.measurements / "Session1" / "SpecimIQ"
9
10
11 # White correction with small reference
12 fig, axes = plt.subplots(1, 3, figsize=(15, 5))
13
14 image_path = specim_iq_root / "404" / "capture" / "404"
15 spectral_image, envi_header = load_spectral_image(image_path)
16
17
18 original_rgb = reconstruct_rgb_envi(spectral_image, envi_header)
19
20 # original_rgb *= 1.5
21 # original_rgb = original_rgb.clip(0, 1)
22
23
24 axes[0].imshow(original_rgb)
25 axes[0].set_title("Before white correction")
26
27 white_path = specim_iq_root / "405" / "capture" / "405"
28 white_image, white_envi_header = load_spectral_image(white_path)
29
30 white_pos = (slice(200, 480), slice(100, 380))
31 white_rgb_view = reconstruct_rgb_envi(white_image, white_envi_header)
32 white_rgb_view = select_area(white_rgb_view, white_pos)
33
34 axes[1].imshow(white_rgb_view)

```

```

35 axes[1].set_title("Selected white area")
36
37 white_sq = white_image[white_pos]
38
39 # White correction with selected area
40 whiteref = white_sq.mean(axis=(0,1))
41
42 white_corrected = spectral_image / whiteref
43
44 white_corrected_rgb_view = reconstruct_rgb_envi(white_corrected, envi_header)
45 # white_corrected_rgb_view *= 2.5
46 # white_corrected_rgb_view = white_corrected_rgb_view.clip(0, 1)
47
48
49 axes[2].imshow(white_corrected_rgb_view)
50 axes[2].set_title("After white correction")
51
52 plt.show()
53
54 # Show spectra
55 colors = ["r", "b", "g"]
56 color_names = ["red", "blue", "green"]
57 positions = [(320, 320), (320, 420), (320, 380)]
58
59 wavelengths = get_wavelengths(envi_header)
60
61
62 fig, axes = plt.subplots(2, 4, figsize=(10, 5), tight_layout=True)
63
64 axes1, axes2 = axes
65 for pos, color, color_name, ax in zip(positions, colors, color_names, axes2):
66     ax.plot(wavelengths, white_corrected[pos[1], pos[0], :], color=color)
67     ax.set_title(f"Spectra at {color_name}(corrected)")
68     ax.set_xlabel("Wavelength[nm]")
69
70 for pos, color, color_name, ax in zip(positions, colors, color_names, axes1):
71     ax.plot(wavelengths, spectral_image[pos[1], pos[0], :], color=color)
72     ax.set_title(f"Spectra at {color_name}(original)")
73     ax.set_xlabel("Wavelength[nm]")
74
75
76 axes1[-1].plot(wavelengths, whiteref, color="k")
77 axes1[-1].set_title("White reference")
78 axes1[-1].set_xlabel("Wavelength[nm]")
79
80 canvas = draw_multi_crosss(white_corrected_rgb_view, positions)
81 axes2[-1].imshow(canvas)
82 axes2[-1].set_title("RGB view")
83
84 plt.show()

```

Code 7: White correction for SpecimIQ with large reference

```

1 from pathlib import Path
2
3 import cv2
4 import numpy as np
5

```



```

6 # CONSTS
7 MAX_PIXEL_VALUE = 255
8 MIN_PIXEL_VALUE = 0
9
10
11 def parse_png_path(path: Path) -> tuple[str, int, float]:
12     # parse path of tunable files
13     path_parts = path.stem.split(",")
14     name = path_parts[0]
15     ch_info = path_parts[1]
16     exp_info = path_parts[2]
17     ch_id = int(ch_info.strip().split(" ")[1])
18     exp_id = float(exp_info.strip().split(" ")[1])
19     return name, ch_id, exp_id
20
21
22 def gen_template(name: str, ch: int) -> str:
23     return f"{name}, ch {ch}, exp * ms.png"
24
25
26 def get_score(im: np.ndarray) -> int:
27     nb_max = (im == MAX_PIXEL_VALUE).sum()
28     nb_min = (im == MIN_PIXEL_VALUE).sum()
29     return nb_max + nb_min
30
31
32 def load_tunable_image(
33     tunable_root: Path, name: str, white_pos: tuple[slice, slice]
34 ) -> tuple[np.ndarray, list[int]]:
35     png_list = list(tunable_root.glob("*.png"))
36     channels = {parse_png_path(p)[1] for p in png_list}
37     channels = sorted(channels)
38     imgs = []
39     # chose minimum score image
40     for ch in channels:
41         best_im = None
42         best_score = 1e9
43         template = gen_template(name, ch)
44         for png_path in tunable_root.glob(template):
45             # Select minimum score image. Score is calculated from number of
unvalid pixels
46             im = cv2.imread(str(png_path), cv2.IMREAD_GRAYSCALE)
47             if im[white_pos].max() == MAX_PIXEL_VALUE:
48                 continue
49             score = get_score(im)
50             if score < best_score:
51                 best_score = score
52                 best_im = im
53             if best_im is None:
54                 msg = f"Channel {ch} not found."
55                 raise ValueError(msg)
56             imgs.append(best_im)
57
58     spectral_image = np.stack(imgs, axis=-1)
59     return spectral_image, channels

```

Code 8: Load Tunable Light Sources images

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 from asi import path_config
5 from asi.draw import draw_multi_crosss, select_area
6 from asi.io import load_tunable_image
7
8 # Configurations
9 WHITE_POS = (slice(510, 550), slice(230, 260))
10 SELECT_CHANNELS = [9, 4, 0]
11 # LOAD IMAGE
12 session1 = path_config.measurements / "session1"
13 tunable_root = session1 / "Tunable light sorces" / "ImagesASI"
14
15 spectral_image, channels = load_tunable_image(
16     tunable_root, name="colorchecker", white_pos=WHITE_POS
17 )
18 spectral_image = spectral_image.astype(np.float64)
19
20 # Make RGB view
21 rgb_view = spectral_image[..., SELECT_CHANNELS]
22 # Postprocess for preview
23 rgb_view /= rgb_view.max()
24 rgb_view *= 0.8
25 rgb_view = rgb_view.clip(0, 1)
26
27 # Select white area for correction
28 rgb_view = select_area(rgb_view, WHITE_POS)
29
30 # Apply white correction
31 white_sq = spectral_image[WHITE_POS]
32 whiteref = white_sq.mean(axis=(0, 1))
33 white_corrected = spectral_image / whiteref
34
35 # RGB view after white correction
36 white_corrected_rgb_view = white_corrected[..., SELECT_CHANNELS]
37 white_corrected_rgb_view /= white_corrected_rgb_view.max()
38 white_corrected_rgb_view *= 1.2
39 white_corrected_rgb_view = white_corrected_rgb_view.clip(0, 1)
40
41
42 # Plot images
43 fig, axes = plt.subplots(1, 2, figsize=(10, 5))
44 axes[0].imshow(rgb_view)
45 axes[0].set_title("Before white correction")
46
47 axes[1].imshow(white_corrected_rgb_view)
48 axes[1].set_title("After white correction")
49
50 plt.show()
51
52 # show spectra
53 red_pos = (450, 450)
54 blue_pos = (400, 280)
55 green_pos = (400, 400)
56
57
58 pos_list = (red_pos, blue_pos, green_pos)

```

```
59 colors = ["r", "b", "g"]
60 color_names = ["red", "blue", "green"]
61
62 fig, axes = plt.subplots(2, 4, figsize=(15, 8))
63
64 org_axes, white_axes = axes
65 for axes, image in zip([org_axes, white_axes], [spectral_image, white_corrected
66 ]):
67     for pos, color_name, color, ax in zip(pos_list, color_names, colors, axes):
68         ax.plot(channels, image[pos[1], pos[0], :], color=color)
69         ax.set_title(f"{color_name} spectra")
70
71 org_axes[-1].plot(whiteref, color="k")
72 org_axes[-1].set_title("White reference")
73 org_axes[-1].set_xlabel("Wavelength[nm]")
74
75 canvas = draw_multi_crosss(white_corrected_rgb_view, pos_list)
76 white_axes[-1].imshow(canvas)
77 white_axes[-1].set_title("RGB view")
78
79 plt.show()
```

Code 9: White correction for Tunable Light Sources with small white reference