<< Back to MAIN notebook

6 Filter stacking

Now that the previous notebooks have covered the basics of the image formats, colorspaces, and filters, we can finally go ahead and perform image enhancement on five sample images presented in the MAIN notebook. To do this, we will inevitably combine different colorspaces, image channels, and filters to create a specific workflow for each example, or an **image enhancement (filtering) stack**.

This notebook is divided into 6 sections: one section concerning a metric for measuring the accentuation of tracer particles, and five sections with examples with proposed enhancement workflows for each of the example images. Each section will aim to explain why a certain procedure is proposed, but the provided strategy will hardly be definitive and only one suitable - feel free to explore (add/remove/modify) different filters and/or adjust filter parameters until you feel satisfied with the final results.

Contents

In [1]:
```
# Necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import cv2
import nbimporter

from axes_tiein import on_lims_change
from image_filtering import lowpass_highpass, intensity_capping, normalize_image, gaussian_lo

# Use [%matplotlib widget] inside JupyterLab, but
# and [%matplotlib notebook] for Jupyter Notebook
%matplotlib widget
```

6.1 Signal-to-noise ratio

To quantify the accentuation of tracer particles relative to the image background, we can adopt a simple signal-to-noise ratio (SNR) metric, which is the ratio of mean image pixel value (defined mostly by the background) and their standard deviation (defined mostly by the tracer particles):

In [2]:
```
def snr(a):
```

```
    # Scale between 0 and 255
    a = ((a - np.min(a)) / (np.max(a) - np.min(a)) * 255).astype(int)

    return np.mean(a)/np.std(a)
```

Image enhancement strategy should generally aim to minimize the SNR score - to increase the standard deviation of pixel values by making the tracer particles "stand out more" in the image. To properly apply this metric, we should never look at the whole image as it contains many non-relevant regions (riverbanks, islands, etc), but should rather choose certain image regions which only contains particles on water surface (background).

> Formal definition of SNR defines the mean value as the "signal" and deviation as "noise".
> In our case, the "noise" is the target of enhancement.

6.2 Image 1

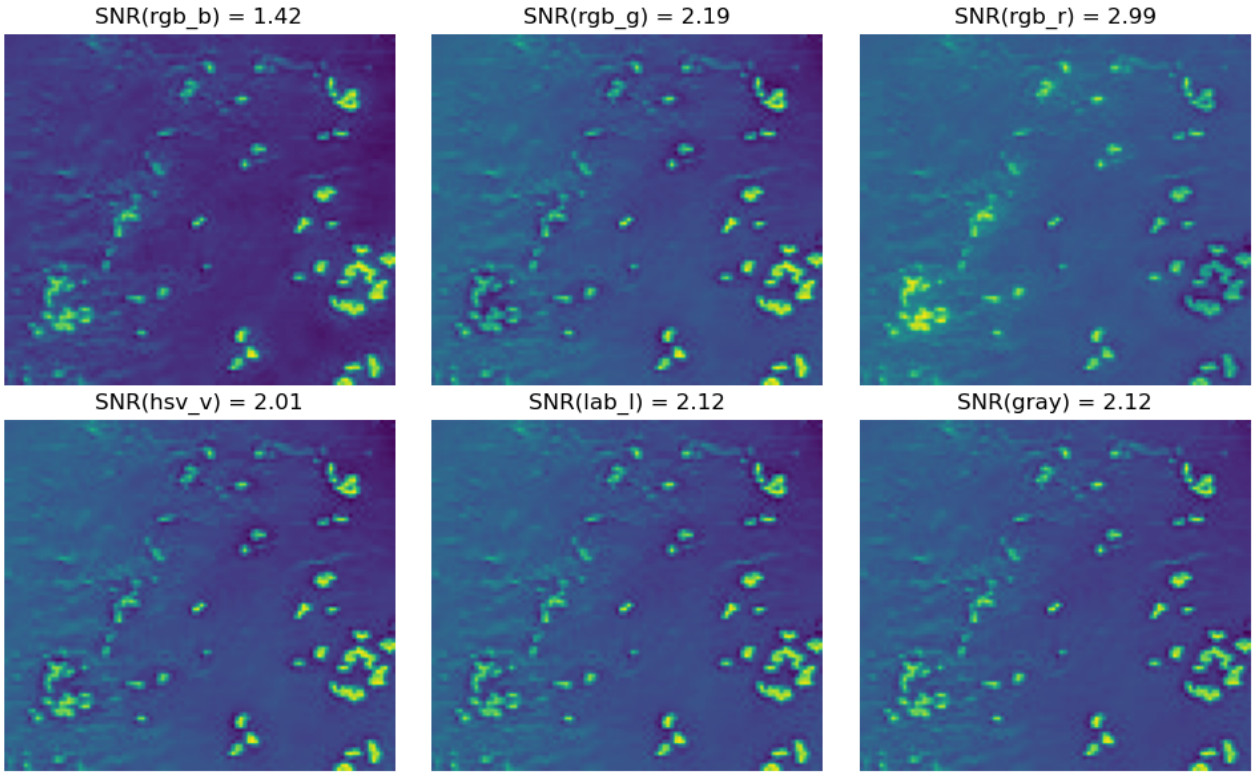
We should first explore the colorspaces using the last code block in the Image colorspaces notebook. By examining the resulting image channels, it is clear from visual inspection that the following channels are viable for image velocimetry: all three RGB channels, HS[V] channel, [L*]a*b* channel, and grayscale model.

We can now use the SNR metric to compare these channels further by selecting a representative region in the image and calculating the corresponding SNR. The selected region, defined by the coordinate range x=1370..1470 and y=685..775, contains both pink and cyan particles, as well as some surface wave features.

In [3]:
```
img_path = './1080p/1.jpg'

# Image subregion selection
(x1, x2), (y1, y2) = (1370, 1470), (685, 775)

# Conversions
img_bgr = cv2.imread(img_path)
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
img_hsv = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2HSV)
img_lab = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2LAB)

# Split into channels
rgb_b, rgb_g, rgb_r = cv2.split(img_bgr)
hsv_h, hsv_s, hsv_v = cv2.split(img_hsv)
lab_l, lab_a, lab_b = cv2.split(img_lab)
gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)

# Viable image channels
channels = [
    rgb_b, rgb_g, rgb_r,
    hsv_v, lab_l, gray
]

# Prepare a plot
ncols = 3
nrows = np.ceil(len(channels) / ncols)
```

```
fig, ax = plt.subplots(nrows=int(nrows), ncols=ncols, figsize=(9.8, 6))

for i, c in enumerate(channels):
    c_name = [k for k, v in locals().items() if v is c][0]
    subimg = channels[i][y1: y2, x1:x2]
    ax[i//ncols, i % ncols].imshow(subimg)
    ax[i//ncols, i % ncols].set_title('SNR({}) = {:.2f}'.format(c_name, snr(subimg)))

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure



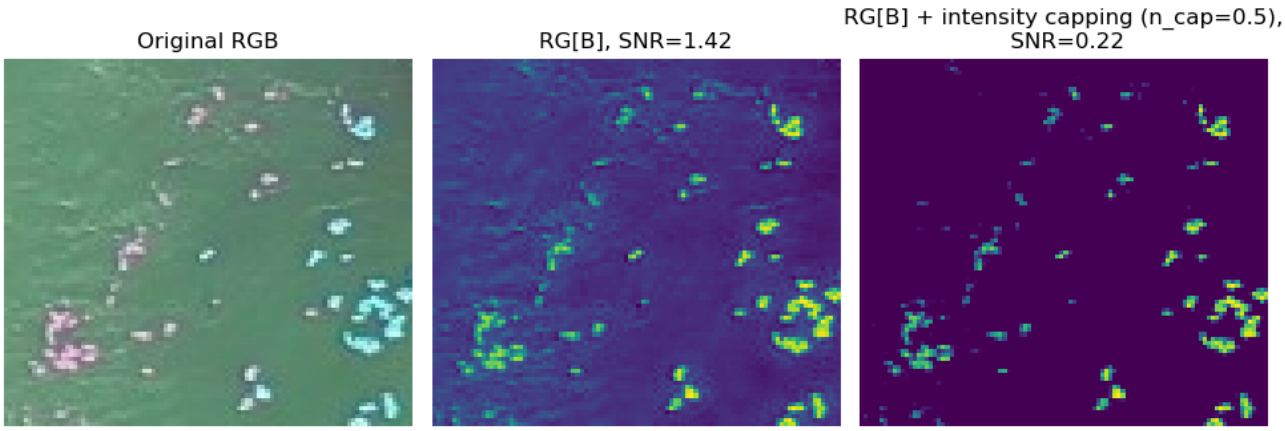
The results of the colorspace analysis indicate that the blue channel from RGB model has the highest SNR value in the selected region, and this channel will be used in the remainder of the analysis.

Since both the pink and cyan tracer particles are already well accentuated, we mostly want to remove the surface wave features, we can achieve this by pixel applying intensity capping:

In [4]:
```
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(9.8, 3.6))

ax[0].imshow(img_rgb[y1: y2, x1:x2])
ax[0].set_title('Original RGB')

ax[1].imshow(rgb_b[y1: y2, x1:x2])
ax[1].set_title('RG[B], SNR={:.2f}'
                .format(snr(rgb_b[y1: y2, x1:x2])))

# Accentuate light particles on darker background from blue channel
n_cap = 0.5
img1_cap = intensity_capping(rgb_b, n_std=n_cap, mode='LoD')
```

```
ax[2].imshow(img1_cap[y1: y2, x1:x2])
ax[2].set_title('RG[B] + intensity capping (n_cap={:.1f}),\n SNR={:.2f}'
                .format(n_cap, snr(img1_cap[y1: y2, x1:x2])))

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure

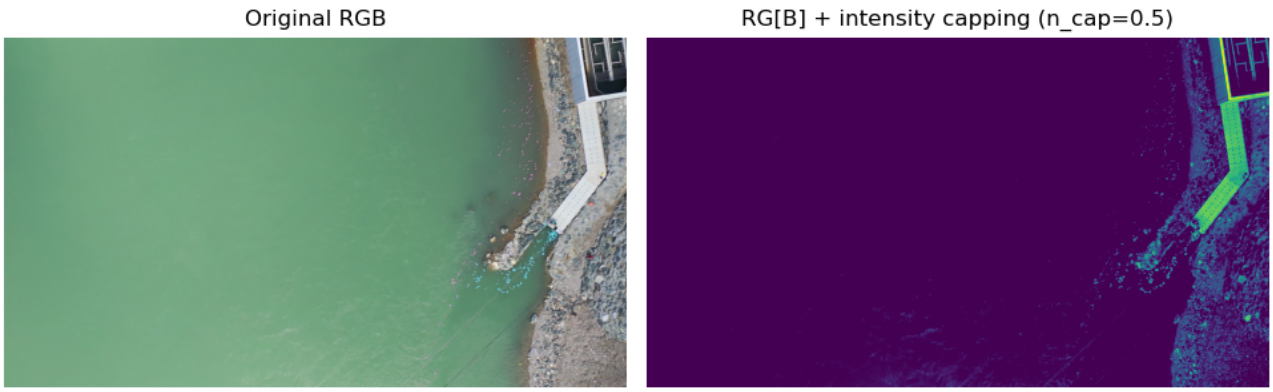


The resulting SNR is far lower than that of the raw blue channel, and surface wave features are almost completely removed. We can increase the capping parameter `n_cap` to further filter such features, but one should also be careful to not also lose information about tracer particles as well - finding a right balance is often a matter of experience through trial and error.

Finally, we should also present and compare the RGB blue channel and the capped image to verify that the desired features are accentuated across the entire image:

In [5]:
```
fig, ax = plt.subplots(nrows=1, ncols= 2, figsize=(9.8, 3.2))

ax[0].imshow(img_rgb)
ax[0].set_title('Original RGB')

ax[1].imshow(img1_cap)
ax[1].set_title('RG[B] + intensity capping (n_cap={:.1f})'.format(n_cap))

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure



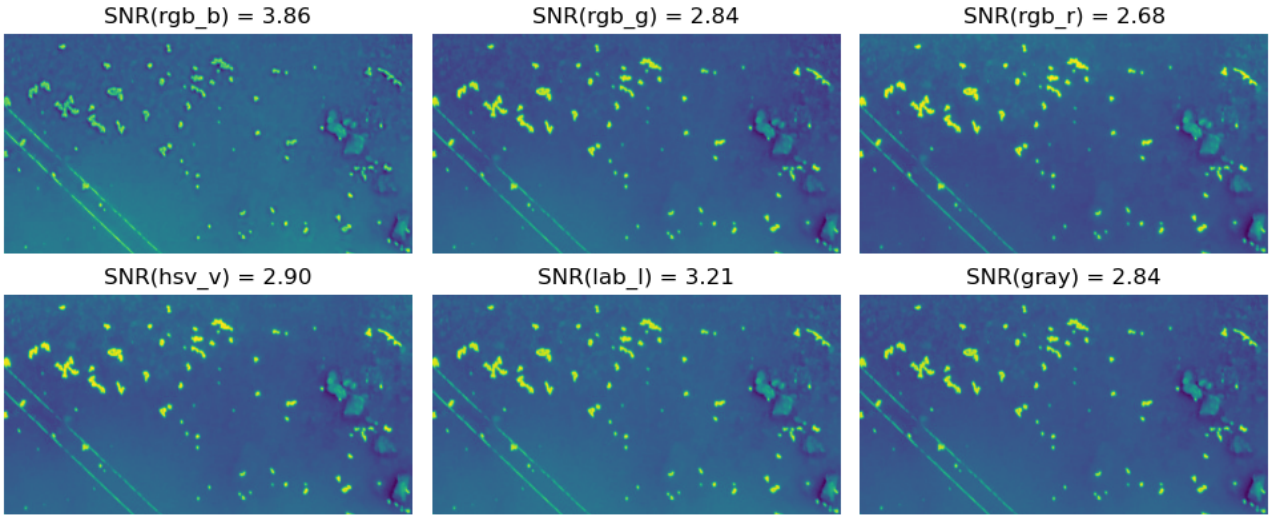
6.3 Image 2

Second example is similar to **Image 1**, but contains some regions with visual disturbances such as visible riverbed, powerlines, chroma changes near riverbanks, etc. As a representative region, we can select the top-left area on the water surface close to the riverbank. Same as with **Image 1**, viable colorspace models and channels are: all three RGB channels, HS[V] channel, [L*]a*b* channel, and grayscale model. For this example, RGB red channel has the highest SNR value in the selected region, but will fail to properly present cyan-colored particles. Because of this, we can select grayscale model for further enhancement.

In [6]:
```
img_path = './1080p/2.jpg'

# Image subregion selection
(x1, x2), (y1, y2) = (0, 335), (350, 530)

# Conversions
img_bgr = cv2.imread(img_path)
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
img_hsv = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2HSV)
img_lab = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2LAB)

# Split into channels
rgb_b, rgb_g, rgb_r = cv2.split(img_bgr)
hsv_h, hsv_s, hsv_v = cv2.split(img_hsv)
lab_l, lab_a, lab_b = cv2.split(img_lab)
gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)

# Viable image channels
channels = [
    rgb_b, rgb_g, rgb_r,
    hsv_v, lab_l, gray
]

ncols = 3
nrows = np.ceil(len(channels) / ncols)
fig, ax = plt.subplots(nrows=int(nrows), ncols=ncols, figsize=(9.8, 4.2))

for i, c in enumerate(channels):
    c_name = [k for k, v in locals().items() if v is c][0]
    subimg = channels[i][y1: y2, x1:x2]
```

```
    ax[i//ncols, i % ncols].imshow(subimg)
    ax[i//ncols, i % ncols].set_title('SNR({}) = {:.2f}'
                                      .format(c_name, snr(subimg)))

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure

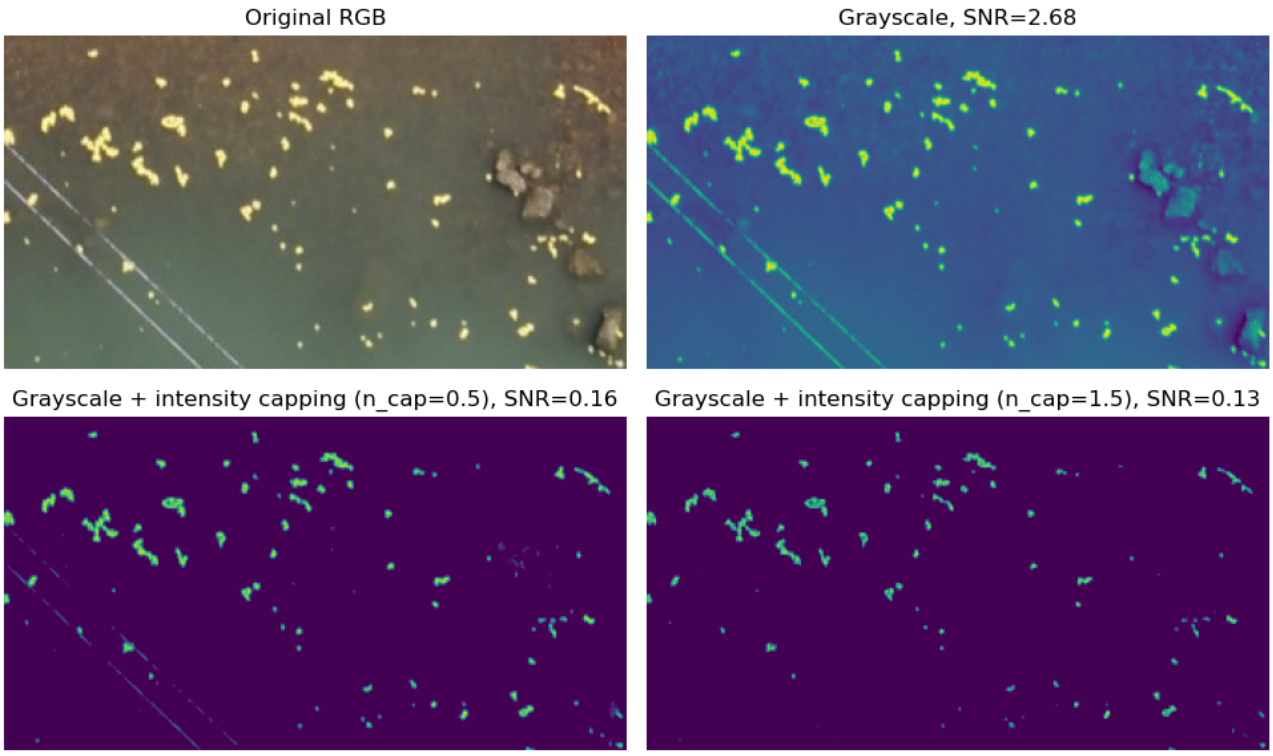


Same as with **Image 1**, we can try to apply pixel value intensity capping to accentuate the tracers on the water surface. As a side effect, intensity capping can also be successfully used to remove some of the riverbed features (right side in images below), as well as to remove powerlines from the bottom-left corner when the intensity capping parameter is high enough (N > 1.0):

In [7]:
```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(9.8, 6.0))

ax[0][0].imshow(img_rgb[y1: y2, x1:x2])
ax[0][0].set_title('Original RGB')

ax[0][1].imshow(gray[y1: y2, x1:x2])
ax[0][1].set_title('Grayscale, SNR={:.2f}'.format(snr(rgb_r[y1: y2, x1:x2])))

# Intensity capping
n_cap = 0.5
img2_cap1 = intensity_capping(gray, n_std=n_cap, mode='LoD')

ax[1][0].imshow(img2_cap1[y1: y2, x1:x2], vmin=0, vmax=255)
ax[1][0].set_title('Grayscale + intensity capping (n_cap={:.1f}), SNR={:.2f}'
                   .format(n_cap, snr(img2_cap1[y1: y2, x1:x2])))

# Intensity capping
n_cap = 1.5
img2_cap2 = intensity_capping(gray, n_std=n_cap, mode='LoD')

ax[1][1].imshow(img2_cap2[y1: y2, x1:x2], vmin=0, vmax=255)
ax[1][1].set_title('Grayscale + intensity capping (n_cap={:.1f}), SNR={:.2f}'
                   .format(n_cap, snr(img2_cap2[y1: y2, x1:x2])))
```

```
[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

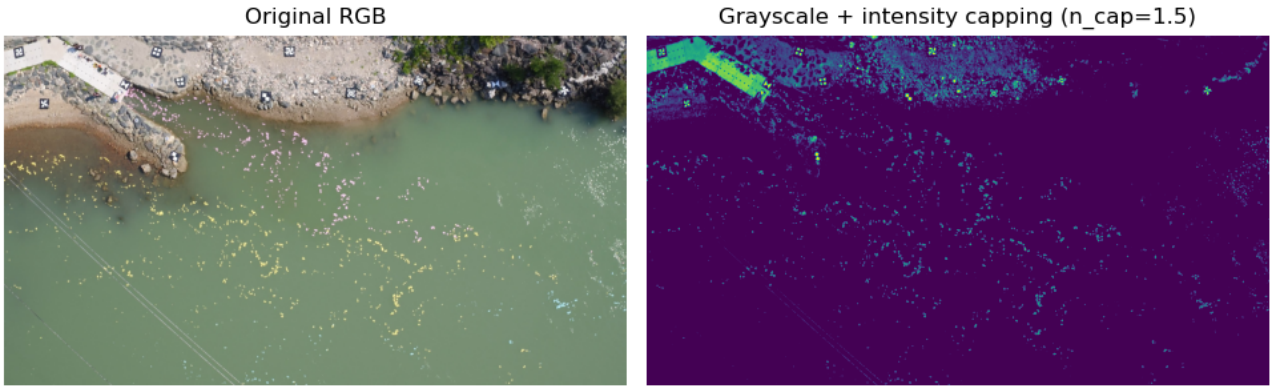
Figure



Finally, we can inspect the entire image to make sure that the tracer particles are presented correctly:

In [8]:
```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(9.8, 3.2))

ax[0].imshow(img_rgb)
ax[0].set_title('Original RGB')

ax[1].imshow(img2_cap2)
ax[1].set_title('Grayscale + intensity capping (n_cap={:.1f})'.format(n_cap))

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure



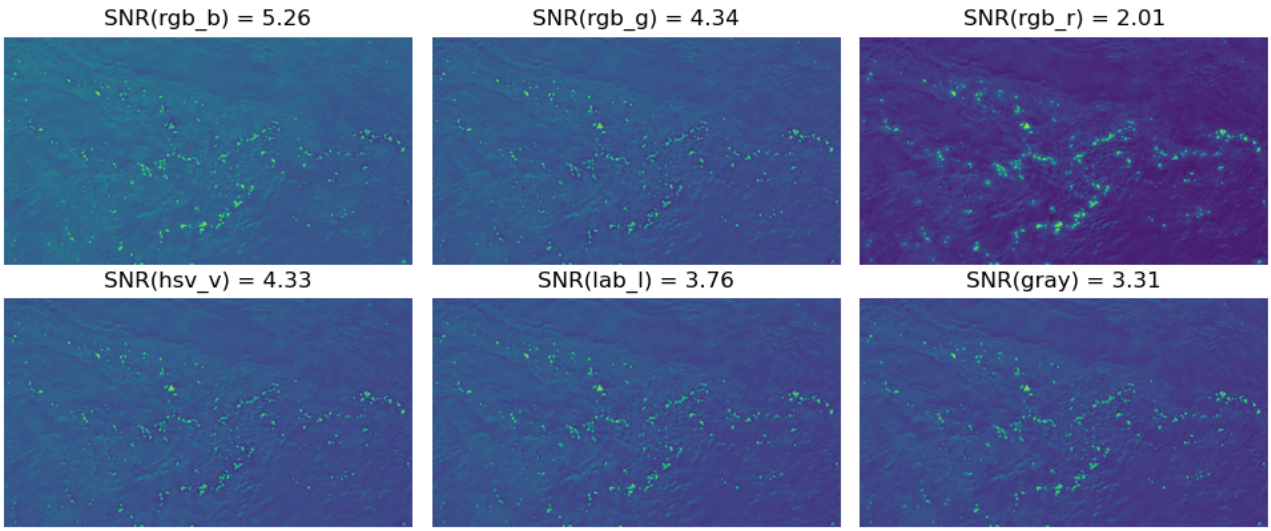
6.4 Image 3

Image 3 represents the same area as in **Image 2**, but the background contains significantly more surface wave features and light reflections. Perhaps the most intuitive reaction to the appearance of additional visual disturbances would be to increase ("strengthen") the intensity capping parameter, but it would be the wrong one. Addition of background visual disturbances actually increases the standard deviation of pixel intensities, so the correct response is more likely to reduce the capping parameter.

In [9]:
```
img_path = './1080p/3.jpg'

# Image subregion selection
(x1, x2), (y1, y2) = (1130, 1560), (670, 910)

# Conversions
img_bgr = cv2.imread(img_path)
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
img_hsv = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2HSV)
img_lab = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2LAB)

# Split into channels
rgb_b, rgb_g, rgb_r = cv2.split(img_bgr)
hsv_h, hsv_s, hsv_v = cv2.split(img_hsv)
lab_l, lab_a, lab_b = cv2.split(img_lab)
gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)

# Viable image channels
channels = [
    rgb_b, rgb_g, rgb_r,
    hsv_v, lab_l, gray
]

ncols = 3
nrows = np.ceil(len(channels) / ncols)
fig, ax = plt.subplots(nrows=int(nrows), ncols=ncols, figsize=(9.8, 4.2))

for i, c in enumerate(channels):
    c_name = [k for k, v in locals().items() if v is c][0]
    subimg = channels[i][y1: y2, x1:x2]
    ax[i//ncols, i % ncols].imshow(subimg)
    ax[i//ncols, i % ncols].set_title('SNR({}) = {:.2f}'
```

```
                                    .format(c_name, snr(subimg)))

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure

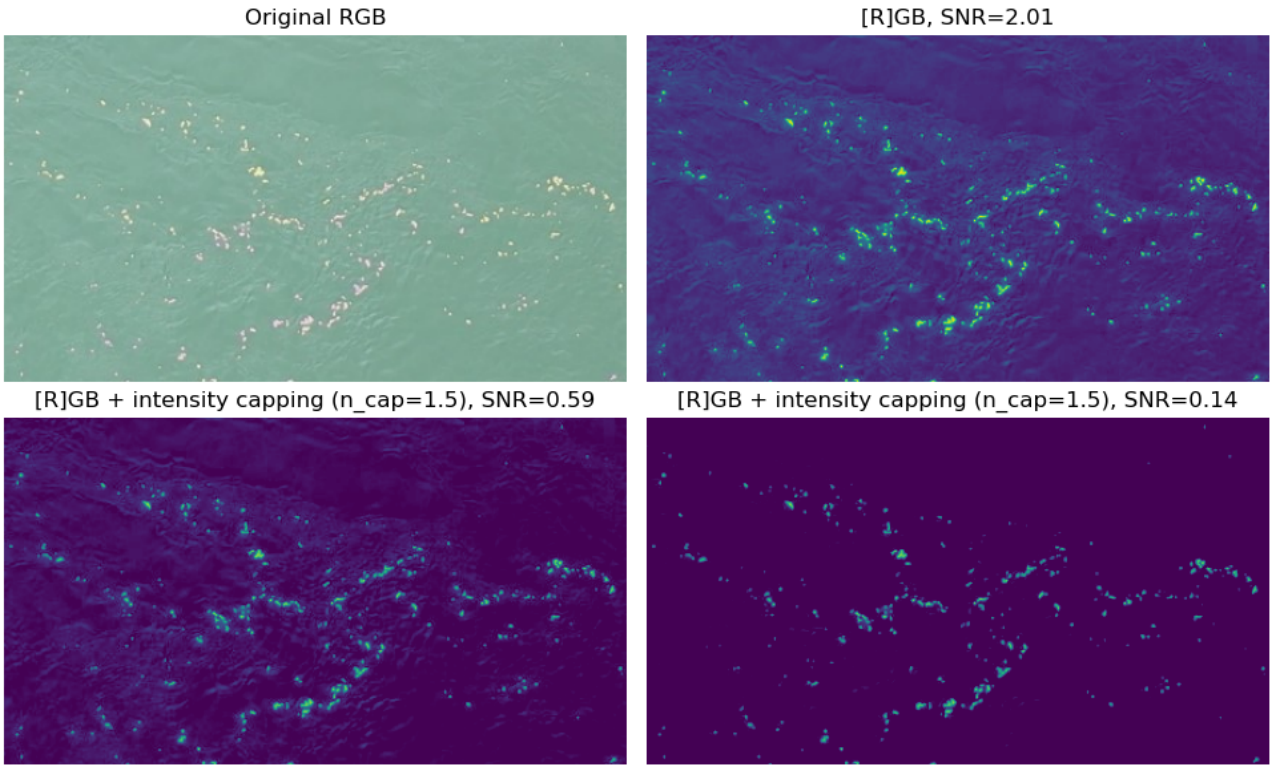


The [R]GB channel has considerably more contrast than the remaining options and will be used further on. This will, however, limit the applicability of the enhanced images for velocimetry using only yellow- and red-colored tracer particles. We can capture cyan-colored tracer particles using the same procedure in RG[B] channel. To capture all of the tracer particles within a single image, we can superimpose the resulting [R]GB and RG[B] components:

In [10]:
```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(9.8, 6.0))

ax[0][0].imshow(img_rgb[y1: y2, x1:x2])
ax[0][0].set_title('Original RGB')

ax[0][1].imshow(rgb_r[y1: y2, x1:x2])
ax[0][1].set_title('[R]GB, SNR={:.2f}'.format(snr(rgb_r[y1: y2, x1:x2])))

# Intensity capping in [R]GB
n_cap_r = 0.0
img3_cap1 = intensity_capping(rgb_r, n_std=n_cap_r, mode='LoD')

ax[1][0].imshow(img3_cap1[y1: y2, x1:x2], vmin=0, vmax=255)
ax[1][0].set_title('[R]GB + intensity capping (n_cap={:.1f}), SNR={:.2f}'
                   .format(n_cap, snr(img3_cap1[y1: y2, x1:x2])))

# Intensity capping in [R]GB
n_cap_r = 0.5
img3_cap2 = intensity_capping(rgb_r, n_std=n_cap_r, mode='LoD')

ax[1][1].imshow(img3_cap2[y1: y2, x1:x2], vmin=0, vmax=255)
ax[1][1].set_title('[R]GB + intensity capping (n_cap={:.1f}), SNR={:.2f}'
                   .format(n_cap, snr(img3_cap2[y1: y2, x1:x2])))

# Intensity capping RG[B]
```

```
n_cap_b = 1.0
img3_cap3 = intensity_capping(rgb_b, n_std=n_cap_b, mode='LoD')

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

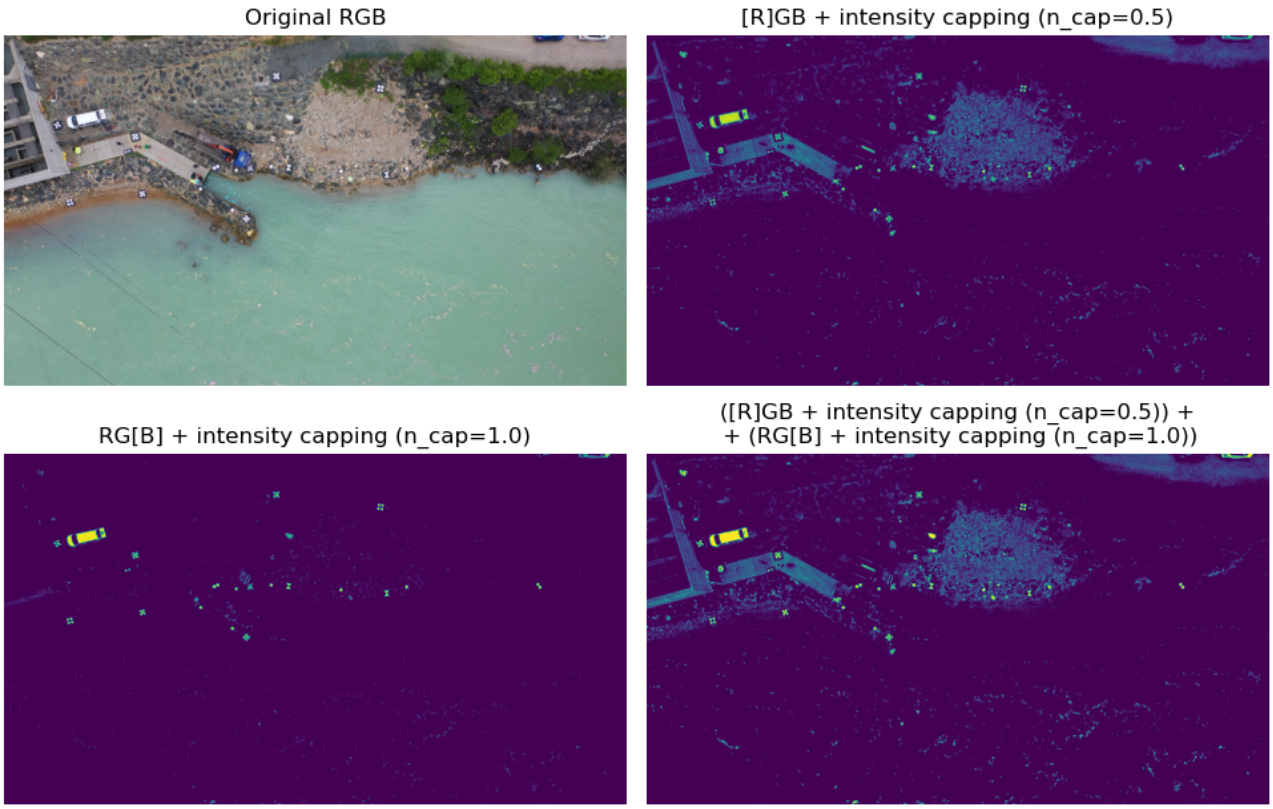
Figure



Inspect the whole image, and pay close attention to the region around [730, 490]:

In [11]:
```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(9.8, 6.4))

ax[0][0].imshow(img_rgb)
ax[0][0].set_title('Original RGB')

ax[0][1].imshow(img3_cap2)
ax[0][1].set_title('[R]GB + intensity capping (n_cap={:.1f})'.format(n_cap_r))

ax[1][0].imshow(img3_cap3)
ax[1][0].set_title('RG[B] + intensity capping (n_cap={:.1f})'.format(n_cap_b))

ax[1][1].imshow(cv2.add(img3_cap2, img3_cap3))
ax[1][1].set_title('([R]GB + intensity capping (n_cap={:.1f})) +'\
                   '\n + (RG[B] + intensity capping (n_cap={:.1f}))'
                   .format(n_cap_r, n_cap_b))

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure



The idea of superimposing data from different image channels is quite powerful when tracers of different color are used for seeding. An alternative approach could be to perform image velocimetry using several sets of images (for each of the colors) and superimpose the velocity data, but such approach will consume more time and storage space.

6.5 Image 4

The following example is likely the most complex one in this report. **Image 4** contains magenta- and cyan-colored particles in various seeding densities across the water surface, visible riverbed of various shades of green color, surface waves and light reflections, etc. Unlike previous examples, this image cannot be tackled by applying a single filter. Therefore, we can try the following:

1. **Select the appropriate image channel** based on visual inspection and SNR metric,
2. Remove as many background features such as riverbed using a **highpass filter**,
3. Apply **intensity capping** to remove the remaining background features.

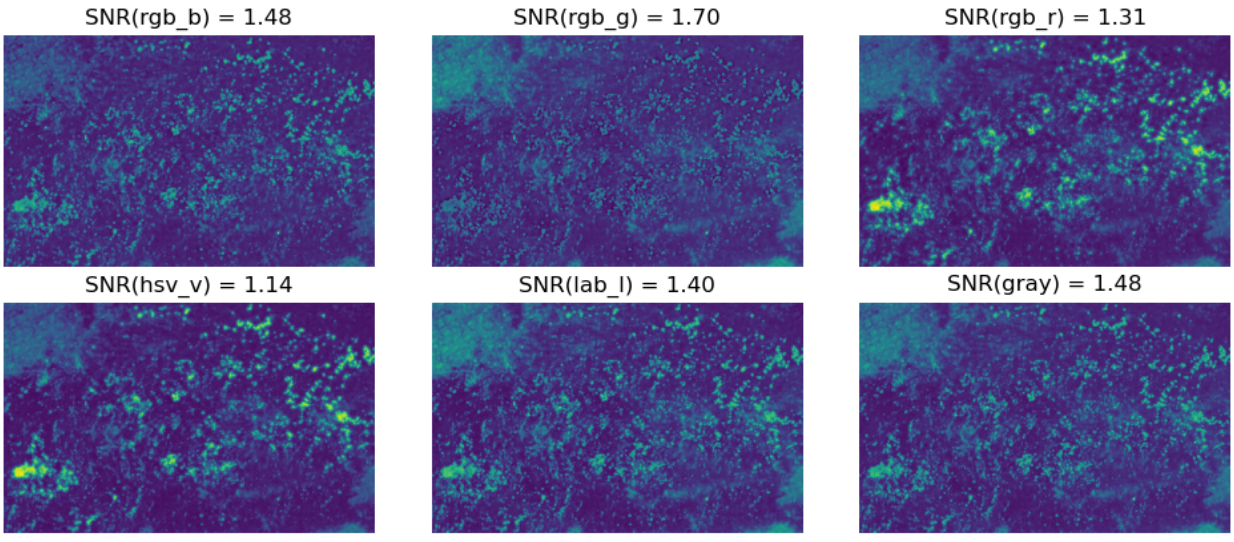
This procedure is adequate, but not the only possible, as we'll show later on.

First we should analyze the viable colorspace channels:

In [12]:
```
img_path = './1080p/4.jpg'

# Image subregion selection
(x1, x2), (y1, y2) = (560, 930), (150, 380)

# Conversions
```

```
img_bgr = cv2.imread(img_path)
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
img_hsv = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2HSV)
img_lab = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2LAB)

# Split into channels
rgb_b, rgb_g, rgb_r = cv2.split(img_bgr)
hsv_h, hsv_s, hsv_v = cv2.split(img_hsv)
lab_l, lab_a, lab_b = cv2.split(img_lab)
gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)

# Viable image channels
channels = [
    rgb_b, rgb_g, rgb_r,
    hsv_v, lab_l, gray
]

ncols = 3
nrows = np.ceil(len(channels) / ncols)
fig, ax = plt.subplots(nrows=int(nrows), ncols=ncols, figsize=(9.8, 4.2))

for i, c in enumerate(channels):
    c_name = [k for k, v in locals().items() if v is c][0]
    subimg = channels[i][y1: y2, x1:x2]
    ax[i//ncols, i % ncols].imshow(subimg)
    ax[i//ncols, i % ncols].set_title('SNR({}) = {:.2f}'
                                      .format(c_name, snr(subimg)))

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure

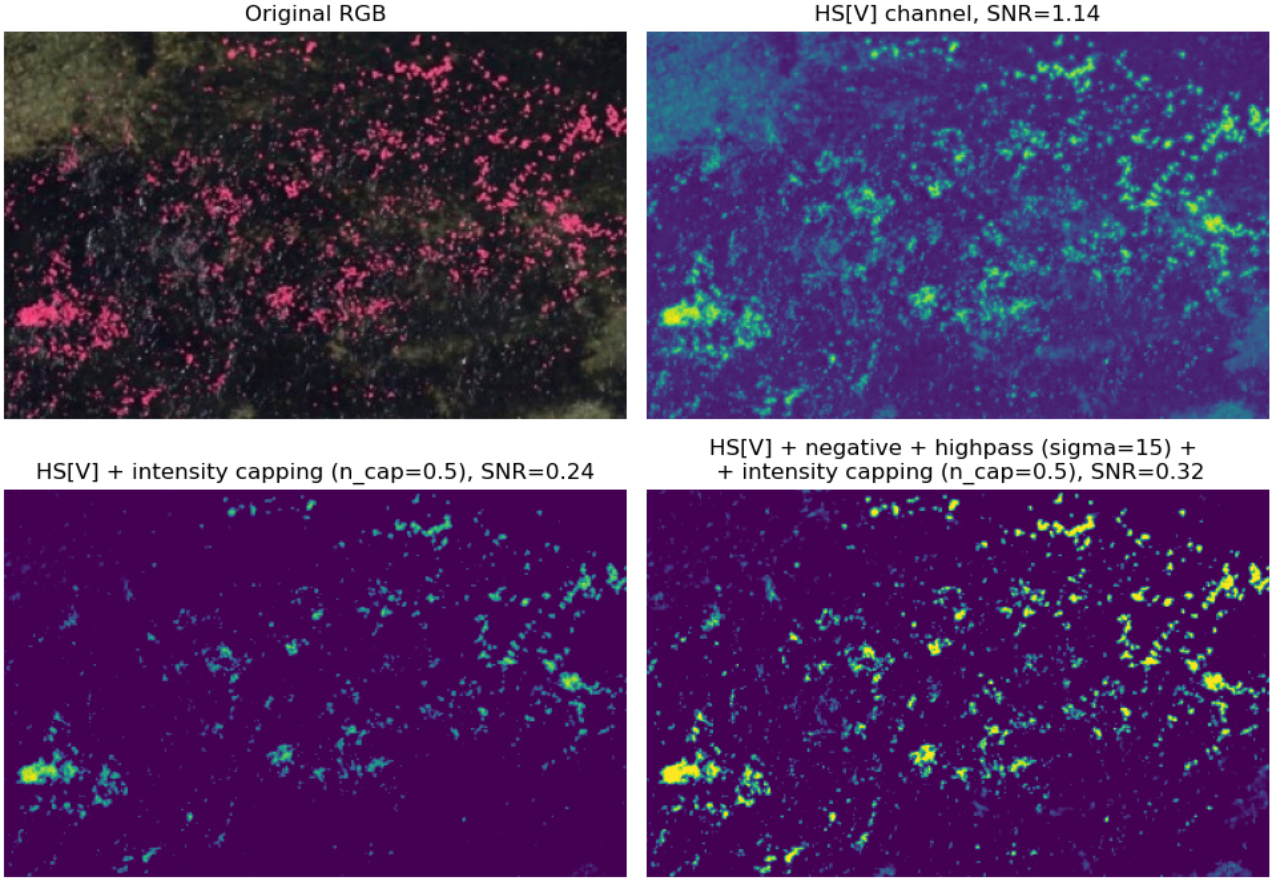


Now we can try **highpass filter** and **intensity capping** to remove as much background features as possible without losing too much information about tracer particles:

In [13]:
```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(9.8, 7.0))

ax[0][0].imshow(img_rgb)
```

```
ax[0][0].set_title('Original RGB')

ax[0][1].imshow(hsv_v)
ax[0][1].set_title('HS[V] channel, SNR={:.2f}'.format(snr(hsv_v[y1: y2, x1:x2])))

# Intensity capping
n_cap = 0.5
img4_v1 = intensity_capping(hsv_v, n_std=n_cap, mode='LoD')

ax[1][0].imshow(img4_v1, vmin=0, vmax=255)
ax[1][0].set_title('HS[V] + intensity capping (n_cap={:.1f}), SNR={:.2f}'
                   .format(n_cap, snr(img4_v1[y1: y2, x1:x2])))

# Intensity capping
sigma = 15
n_cap = 0.5
img4_v2 = lowpass_highpass(hsv_v, sigma=sigma)[1]
img4_v2 = intensity_capping(img4_v2, n_std=n_cap, mode='LoD')

ax[1][1].imshow(img4_v2, vmin=0, vmax=255)
ax[1][1].set_title('HS[V] + negative + highpass (sigma={}) + \n + intensity capping (n_cap={:
                   .format(sigma, n_cap, snr(img4_v2[y1: y2, x1:x2])))

[a.axis('off') for a in ax.reshape(-1)]
[a.set_xlim([x1, x2]) for a in ax.reshape(-1)]
[a.set_ylim([y2, y1]) for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure



As evident from the figure above, when riverbed is quite visible with pixel intensities similar to those of the tracer particles, intensity capping on its own is not adequate for particle accentuation - there is a significant and unacceptable tradeoff between riverbed filtering and particle accentuation. A workaround is to first remove low frequency content from the image, which riverbed features generally belong to. Afterwards, intensity capping will likely provide better results. However, while it would appear that the highpass filter slightly increases the SNR, and thus worsens the particle accentuation, visual inspection of images (especially comparison with the original RGB) indicates otherwise. This also leads to the conclusion about SNR metric:

> **Warning:** Signal-to-noise metric is not a perfect indicator of the tracer particle accentuation and should only be used as a loose guide during the enhancement process. **VISUAL INSPECTION should always dictate of the image enhancement workflow**.

There is one more, albeit exotic, approach available for images such as these. In the notebook on **Image colorspaces** we've discussed the L*a*b* model and how it manipulates colors in ranges between red and green (L*[a*]b*), and blue and yellow (L*a*[b*]). However, L*[a*]b* and L*a*[b*] suffer from one significant limitation - they will always appear relatively blurry compared to the [L*]a*b* and the original image. There is an interesting but obscure manipulation available that can help properly accentuate particles of red, green, blue, and yellow color - we can rearrange the channels of the L*a*b* model as follows:

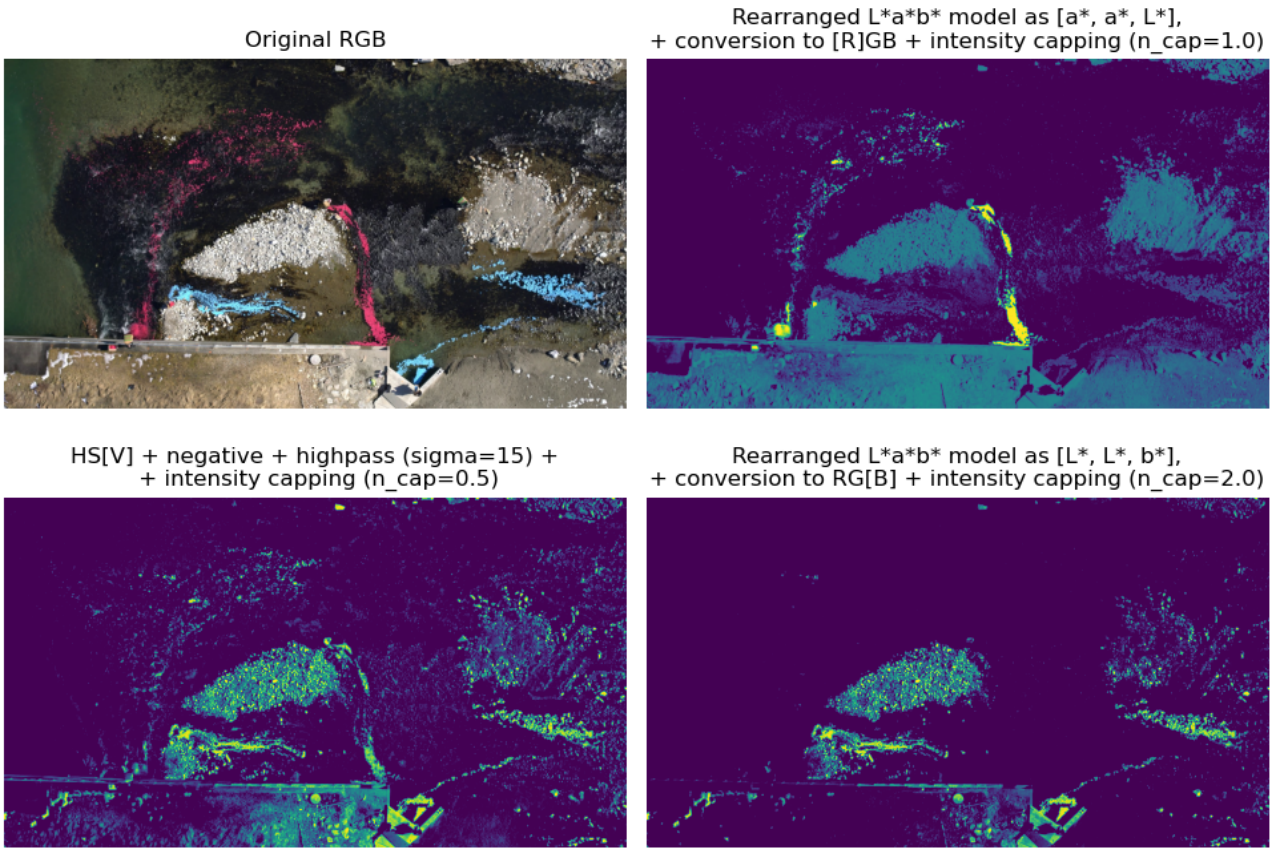
1. For the accentuation of red particles = arrangement (a*, a*, L*),
2. For the accentuation of green particles = arrangement (~a*, ~a*, L*), where ~ operator denotes image negative,
3. For the accentuation of cyan particles = arrangement (L*, L*, b*),
4. ...

The idea is usually (but not always) to put the [L*]a*b* in the place of a channel that is not representing the targeted color, and the channel with the targeted color in place of the [L*]a*b*. The only thing left then is to convert from L*a*b* to RGB/BGR and find the adequate subchannel which best represents the targeted features (trial and error). You can see it in action wit the code below, where we will also apply only the intensity capping (make sure to read the code comments for additional info):

In [14]:
```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(9.8, 7.0))

ax[0][0].imshow(img_rgb)
ax[0][0].set_title('Original RGB')

ax[1][0].imshow(img4_v2, vmin=0, vmax=255)
ax[1][0].set_title('HS[V] + negative + highpass (sigma={}) +' \
                   '\n + intensity capping (n_cap={:.1f})'
                   .format(sigma, n_cap))

# Targeting MAGENTA tracer particles ========================================

# Change the order and structure of L*a*b* model
lab_rearranged = cv2.merge([lab_a, lab_a, lab_l])

# Convert to RGB and take the first (red) channel
```

```
red_rearranged = cv2.cvtColor(lab_rearranged, cv2.COLOR_LAB2RGB)[:, :, 0]

# Apply some intensity capping
n_cap = 1.0
red_rearranged_cap = intensity_capping(red_rearranged, n_std=n_cap, mode='LoD')

ax[0][1].imshow(red_rearranged_cap, vmin=0, vmax=255)
ax[0][1].set_title('Rearranged L*a*b* model as [a*, a*, L*],'\
                   '\n+ conversion to [R]GB + intensity capping (n_cap={:.1f})'
                   .format(n_cap))

# ------------------------------------------------------------------------

# Targeting CYAN tracer particles ============================================

lab_rearranged = cv2.merge([lab_l, lab_l, lab_b])

# Convert to RGB and take the third (blue) channel
blue_rearranged = cv2.cvtColor(lab_rearranged, cv2.COLOR_LAB2RGB)[:, :, 2]

n_cap = 2.0
blue_rearranged_cap = intensity_capping(blue_rearranged, n_std=n_cap, mode='LoD')

ax[1][1].imshow(blue_rearranged_cap, vmin=0, vmax=255)
ax[1][1].set_title('Rearranged L*a*b* model as [L*, L*, b*],'\
                   '\n+ conversion to RG[B] + intensity capping (n_cap={:.1f})'
                   .format(n_cap))

# ------------------------------------------------------------------------

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure



As evident from the figure above, compared to simply taking the HS[V] channel, we can target particular tracer particle colors using L*a*b* model and obtain results that are far "cleaner" - fewer "parasitic" features and visual disturbances, fewer waves and light reflections, etc. This method allows for more controlled targeting of color key and chroma, but requires a bit more experimentation, especially when rearanging the channels of the L*a*b* model.

Tip: The resulting images in the previous figure also contain visible static features such as islands and some light reflection of surface waves. While these are usually undesirable, sometimes they cannot be removed by an automated procedure. However, if the video is stabilized such features will appear motionless and can easily be filtered by velocity magnitude after the image velocimetry step. Alternatively, images filtered in this way can also be subjected to background removal.

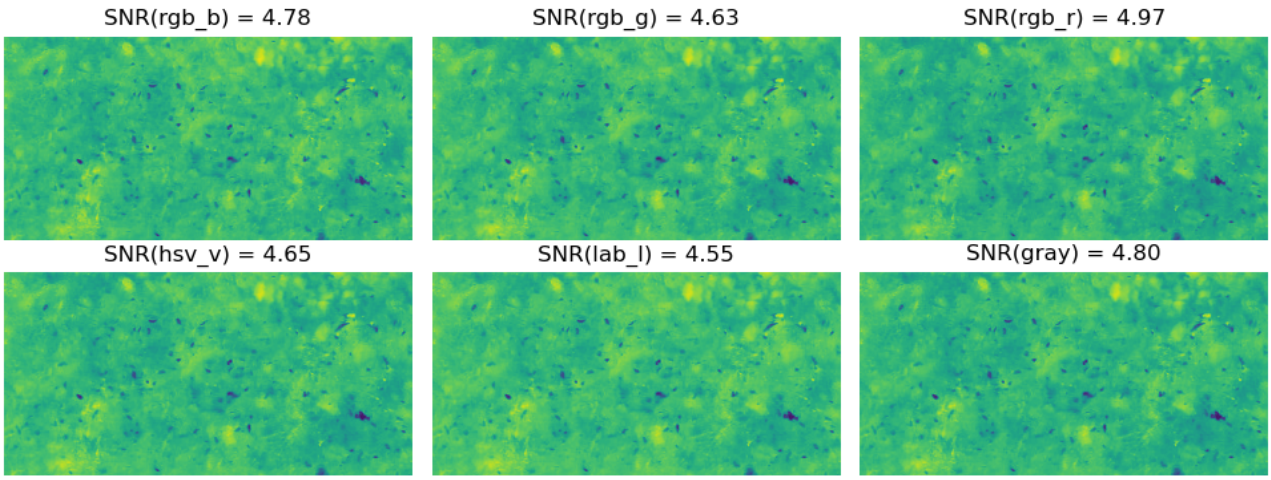
6.6 Image 5

The final example in this report is also somewhat unusual as the tracer particles are darker than the water surface, and the flow is supercritical. Same as with all the previous examples, the three RGB channels, HS[V] and [L*]a*b* are viable options for enhancement:

In [15]:
```
img_path = './1080p/5.jpg'

# Image subregion selection
(x1, x2), (y1, y2) = (820, 1220), (340, 540)
```

```
# Conversions
img_bgr = cv2.imread(img_path)
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
img_hsv = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2HSV)
img_lab = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2LAB)

# Split into channels
rgb_b, rgb_g, rgb_r = cv2.split(img_bgr)
hsv_h, hsv_s, hsv_v = cv2.split(img_hsv)
lab_l, lab_a, lab_b = cv2.split(img_lab)
gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)

# Viable image channels
channels = [
    rgb_b, rgb_g, rgb_r,
    hsv_v, lab_l, gray
]

ncols = 3
nrows = np.ceil(len(channels) / ncols)
fig, ax = plt.subplots(nrows=int(nrows), ncols=ncols, figsize=(9.8, 3.8))

for i, c in enumerate(channels):
    c_name = [k for k, v in locals().items() if v is c][0]
    subimg = channels[i][y1: y2, x1:x2]
    ax[i//ncols, i % ncols].imshow(subimg)
    ax[i//ncols, i % ncols].set_title('SNR({}) = {:.2f}'
                                      .format(c_name, snr(~subimg)))

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure



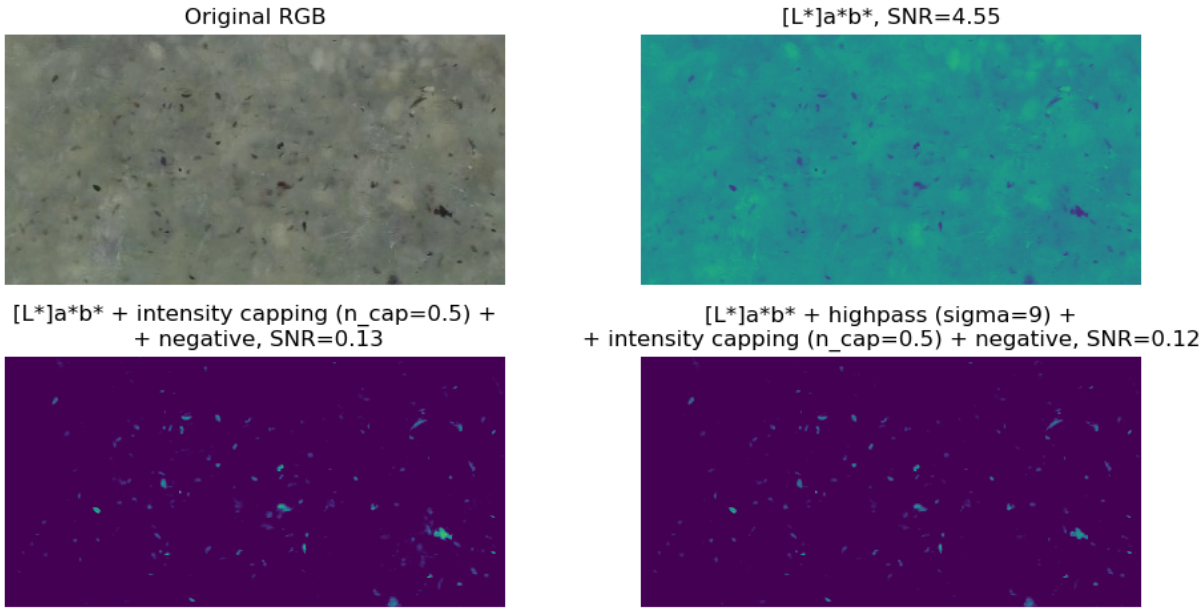
There are very few differences between the image channels in the figure above, although the [L*]a*b*, R[G]B, and HS[V] have a somewhat higher SNR than the remaining options. The primary task here should be the removal of visible riverbed features, which is usually done with image capping.

Keep in mind: When images contain dark tracer particles on lighter background, one should apply

negative filter (~ operator) before calculating SNR metric to obtain proper results where lower SNR are better.

In [16]:
```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(9.8, 4.8))

ax[0][0].imshow(img_rgb)
ax[0][0].set_title('Original RGB')

ax[0][1].imshow(lab_l)
ax[0][1].set_title('[L*]a*b*, SNR={:.2f}'
                   .format(snr(~lab_l[y1: y2, x1:x2])))

# Intensity capping, set dark-on-light (DoL) mode
n_cap = 0.5
img5_v1 = intensity_capping(lab_l, n_std=n_cap, mode='DoL')

ax[1][0].imshow(~img5_v1, vmin=0, vmax=255)
ax[1][0].set_title('[L*]a*b* + intensity capping (n_cap={:.1f}) +' \
                   '\n + negative, SNR={:.2f}'
                   .format(n_cap, snr(~img5_v1[y1: y2, x1:x2])))

# Intensity capping
sigma = 9
n_cap = 0.5
img5_v2 = lowpass_highpass(lab_l, sigma=sigma)[1]
img5_v2 = intensity_capping(img5_v2, n_std=n_cap, mode='DoL')

ax[1][1].imshow(~img5_v2, vmin=0, vmax=255)
ax[1][1].set_title('[L*]a*b* + highpass (sigma={}) +' \
                   '\n+ intensity capping (n_cap={:.1f}) + negative, SNR={:.2f}'
                   .format(sigma, n_cap, snr(~img5_v2[y1: y2, x1:x2])))

[a.axis('off') for a in ax.reshape(-1)]
[a.set_xlim([x1, x2]) for a in ax.reshape(-1)]
[a.set_ylim([y2, y1]) for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure



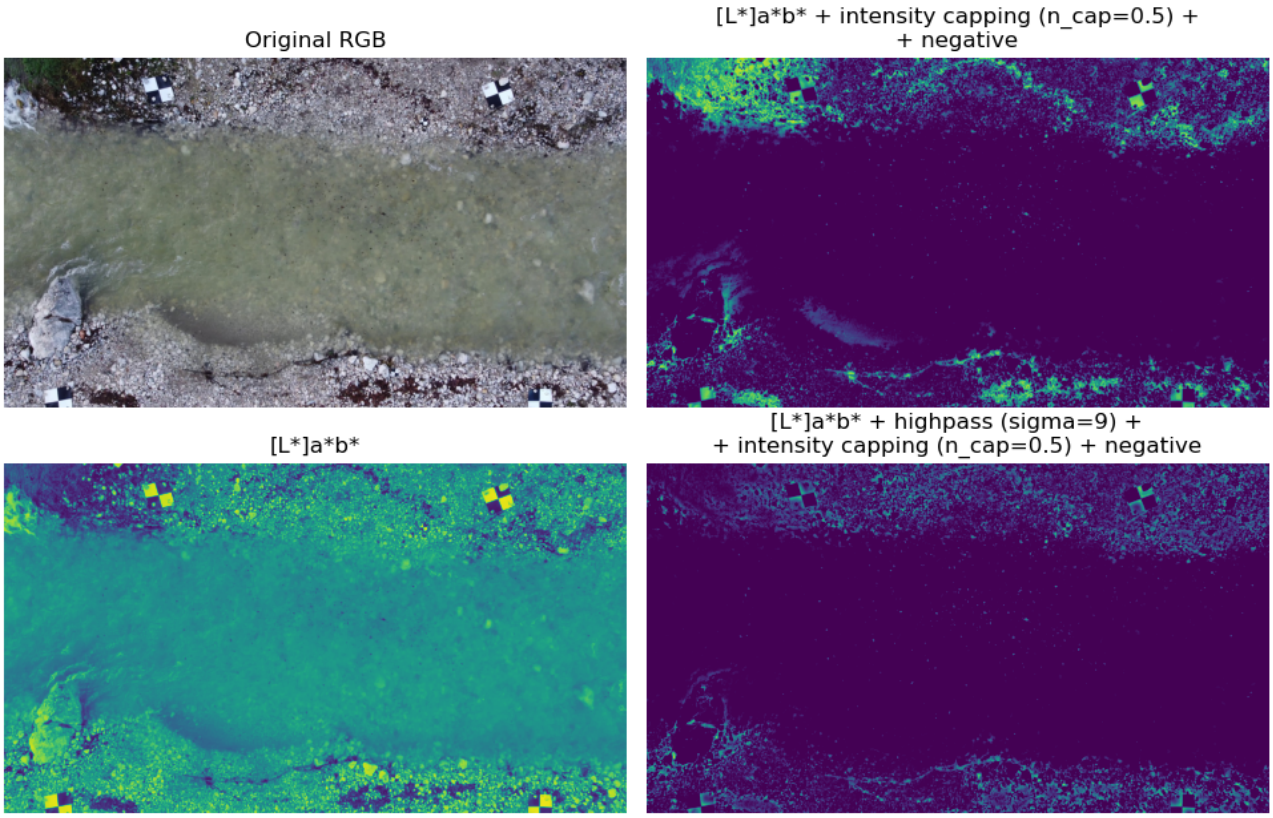
At the first glance, the differences between the two approaches are quite small - highpass filter manages to remove some additional riverbed noise (consider area around [1145, 495]), but the remainder of the subimage is practically identical. However, larger differences are more obvious when the entire image is considered. Highpass filter does not help accentuate the tracer particles, but manages to remove surface wave features and riverbed features (consider, for example, area around [600, 840]). The conjunction of highpass and intensity capping should therefore ALWAYS start with highpass filter followed by intensity capping, as the latter indiscriminately removes a significant amount of information.

> **Tip:** Image enhancement workflow which involves several filter is usually performed in order of the least invasive filter towards the most invasive. For example, highpass filter generally precedes intensity capping because it removes less information than capping. Thresholding algorithms such as intensity capping are often performed last.

In [17]:
```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(9.8, 6.4))

ax[0][0].imshow(img_rgb)
ax[0][0].set_title('Original RGB')

ax[1][0].imshow(lab_l)
ax[1][0].set_title('[L*]a*b*')

ax[0][1].imshow(~img5_v1)
ax[0][1].set_title('[L*]a*b* + intensity capping (n_cap={:.1f}) +' \
                   '\n+ negative'.format(n_cap))

ax[1][1].imshow(~img5_v2)
ax[1][1].set_title('[L*]a*b* + highpass (sigma={}) +' \
                   '\n+ intensity capping (n_cap={:.1f}) + negative'
                   .format(sigma, n_cap))

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]
```

```
plt.tight_layout()
plt.show()
```

Figure



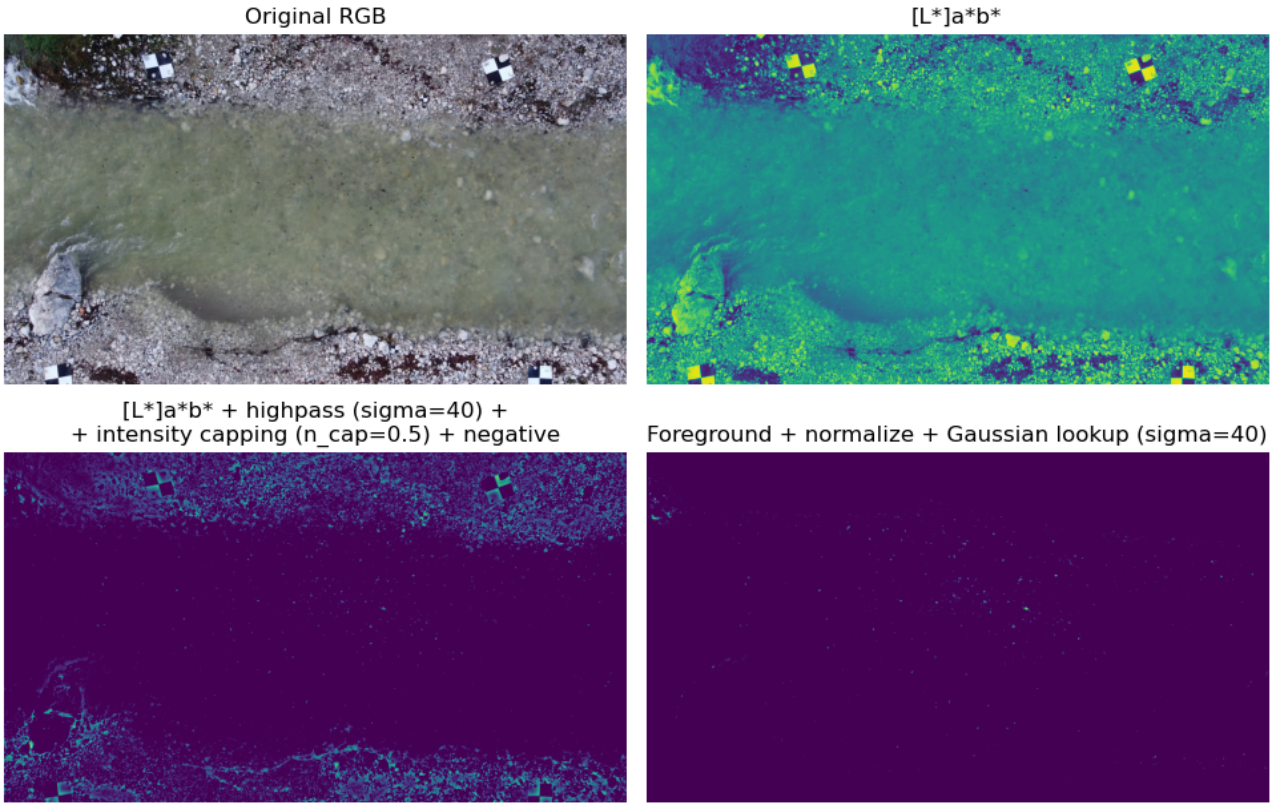
In conclusion of the final example, the combination of highpass filter to remove riverbed and wave features and intensity capping to flatten the background produces a well-defined foreground with tracer particles across the entire water surface.

An alternative approach to this example could be to use **background removal**, followed by **image normalization** and **Gaussian lookup**:

In [18]:
```
# Load background from file and get the [L*]a*b* channel
background_bgr = cv2.imread('./background.jpg')
background_l = cv2.cvtColor(background_bgr, cv2.COLOR_BGR2LAB)[:, :, 0]

# Calculate the foreground
foreground = cv2.subtract(background_l, lab_l)

# Normalize foreground
foreground = normalize_image(foreground)

# Gaussian lookup
sigma = 40
_, _, foreground = gaussian_lookup(foreground, sigma=sigma)

fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(9.8, 6.4))

ax[0][0].imshow(img_rgb)
ax[0][0].set_title('Original RGB')

ax[0][1].imshow(lab_l)
ax[0][1].set_title('[L*]a*b*')
```

```
ax[1][0].imshow(~img5_v2)
ax[1][0].set_title('[L*]a*b* + highpass (sigma={}) +' \
                   '\n+ intensity capping (n_cap={:.1f}) + negative'
                   .format(sigma, n_cap))

ax[1][1].imshow(foreground)
ax[1][1].set_title('Foreground + normalize + Gaussian lookup (sigma={})'.format(sigma))

[a.axis('off') for a in ax.reshape(-1)]
[a.callbacks.connect('ylim_changed', on_lims_change) for a in ax.reshape(-1)]

plt.tight_layout()
plt.show()
```

Figure



Background removal procedure appears to perform quite well, with the tracer particles being far more accentuated when compared to the highpass + intensity capping approach.

6.7 Conclusions on filter stacking

The presented examples underline the conclusion from notebook on Image filtering that not all of the described procedures are equally important and adequate for achieving optimal enhancement for image velocimetry. The most pervasive problems which require image enhancement are:

1. Low contrast between the tracer particles and the water surface,
2. Surface waves (when such are not used as features for tracking),
3. Light reflections (usually against surface waves), and
4. Visible riverbed.

The presented enhancement procedures have all begun with colorspace model exploration, and their analysis in terms of signal-to-noise (SNR) metric to quantify the local accentuation of tracer particles against the water surface. This approach is highly recommended before the actual application of filtering methods, especially in highly complex ground/water surface conditions. This is most evident in Section 6.5 where specific colorspace manipulations have been applied to obtain images with highly accentuated tracer particles. This underlines another important recommendation:

> Use of colored tracer particles - especially those in base (red, green, blue, yellow) colors - is highly recommended in complex ground/water surface conditions, as they enable various colorspace manipulations, which can significantly facilitate the image enhancement workflow.

Of all the presented filtering methods, pixel value **intensity capping** and **highpass filter** are demonstrated to hold the most practical potential. For cases 1..3 described above, **intensity capping** is usually capable enough to accentuate the tracer particles against the background, especially when applied to an adequate colorspace model. Image normalization should generally follow the intensity capping procedure. The capping method is usually monoparametric, with the capping parameter performing well when in range between 0.0 and 2.0 (higher values are more "aggressive", removing more background).

Highpass filter, which targets and removes low-frequency content, is an adequate method for the removal of larger background features such as riverbed structures and larger patches of light reflections or surface waves. However, intensity capping is still a recommended follow-up step. Highpass filter method is also monoparametric, with the parameter `sigma` defining the size of the targeted low-frequency content. While this parameter's value is usually in the double-digits, it is best explored through trial and error for each different case.

The two methods mentioned above also underline an important filter stacking principle - experience indicates that it is more feasible to apply less aggressive methods first, which target and remove less information, and usee the more aggressive methods in the end. This approach of gradual increase of the filtering intensity seems to retain the most ammount of relevant information.

The conclusions about the remaining filtering methods can be summarized as follows:

1. Adjustments of brightness, contrast, and gamma are usefull only if the images are over- or underexposed,
2. Both global histogram equalization and CLAHE are generaly unfavorable as they almost inevitably lead to overexposed water surface,
3. Denoising algorithms should be applied only when there is visible salt-and-pepper noise in the image,
4. Background removal is useful, especially when followed by other filtering methods such as image normalization, Gaussian lookup, etc.

Continue to next chapter: Image enhancement using SSIMS >>

or

<< Back to MAIN notebook