7 Image enhancement using SSIMS

This final section will demonstrate the use of the free and open-source tool **SSIMS: Preprocessing tool for UAV image velocimetry** for image enhancement. The tool is available through its GitHub repository, and it is recommended to download its latest release (v0.3.2.0 as of writing). A detailed explanation on how to download, install, and setup the tool is available on the GitHub homepage.

7.1 Video unpacking

The main form of the tool (Figure 1 left) contains several workflows for different steps of the UAV image preprocessing. The focus of this section will be on the unpacking the video and image enhancement, which uses the top-left section of the main form (dashed line in Figure 1 left).

Video unpacking can be performed by clicking on the top-left button in the main form labeled **Unpack video into frames**, which opens a new form (Figure 1 right). Here, the user can select a video file, choose camera parameters or calibrate the camera, define image extraction format and quality, cut a section of the video, etc. More detailed explanation for using this form can be found in the Readme section on the tool's GitHub homepage.

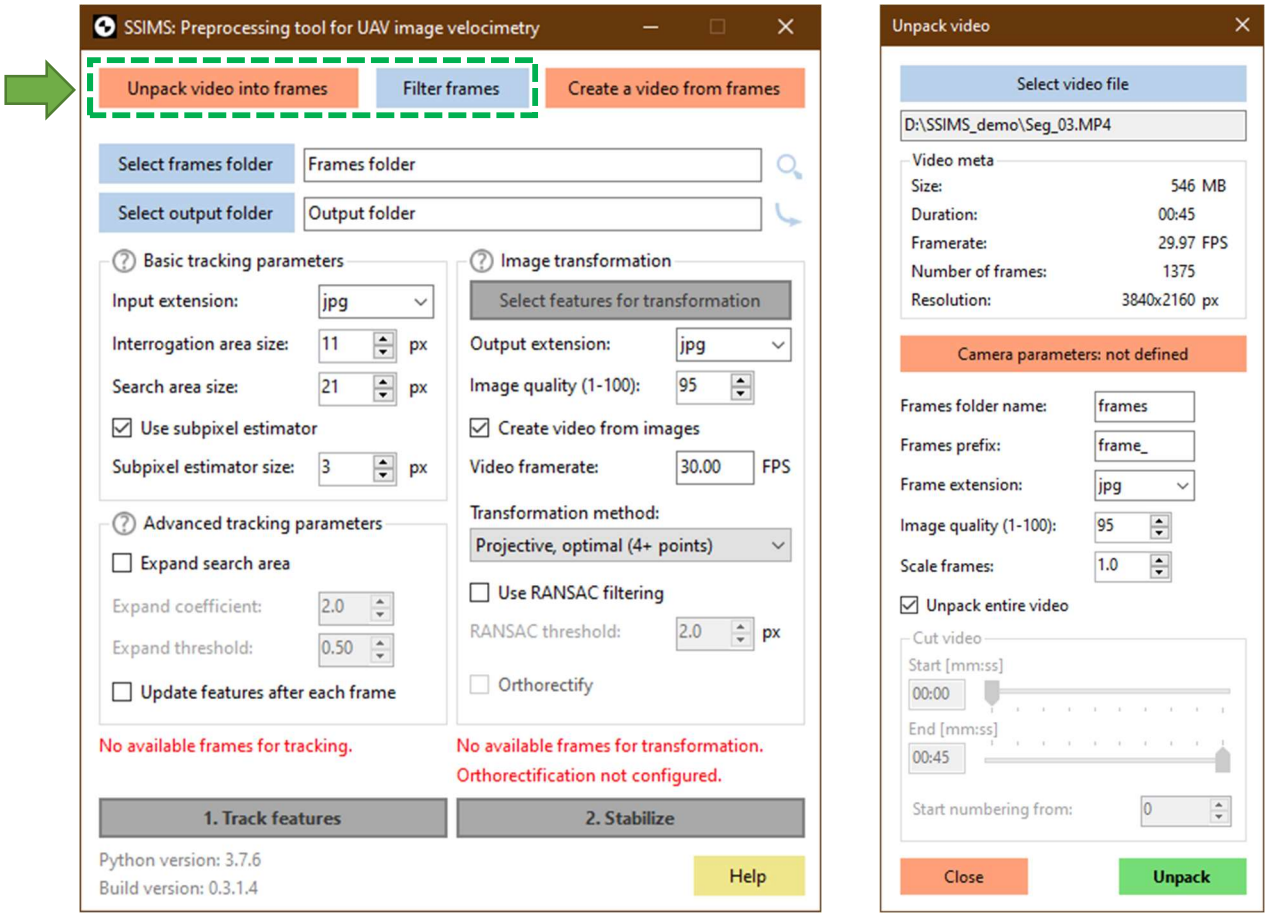

***Figure 1**. Left: SSIMS main form. Right: Video unpacking form.*

7.2 Filter frames form

Once the video has been unpacked into individual frames, we can start the image enhancement by clicking on the **Filter images** button in the top-center of the main form (Figure 1 left). This will open the "Filter frames" form (Figure 2) in which we can define the entire image enhancement procedure.

The "Filter frames" form allows us to select the folder which contains unpacked video frames, and we need to specify their file extension as well. The tool will automatically count and display the number of available frames with the given extension in the selected folder.

Tip: SSIMS will eventually create a file in the frames' folder describing the selected filters and their parameters. If the frames' folder already contains said file, we will be greeted with a prompt asking is we wish to automatically load them to the stack. This can be especially useful when revisiting old projects.

The lower section of the form is divided into two sections:

1. **Left section** contains all the available filters and image enhancement procedures, while the
2. **Right section** is the filtering stack which will eventually contain all the filters which we will select (Figure 2 right).

Selecting filters is done by simply clicking on their corresponding orange buttons on the left side of the form, which will add them to the filtering stack on the right side. Available filters are ordered loosely by their general applicability for image enhancement for velocimetry purposes using tracer particles – filters closer to the top are usually more useful than the remainder.

Keep in mind: Selected filters will become unavailable in the left side as long as they are in the filtering stack (Figure 2 right), which means that the **same filter cannot be added twice** to the filtering stack.

***Figure 2**. Left: initial view of the image enhancement form. Right: After selecting filters.*

Filters in the stack will be executed against every frame **in the order which they hold in the stack.** We can reorder the selected filters in the stack (Figure 2 right) by dragging and dropping them at desired positions.

"Filter frames" form also contains options for performing various colorspace conversions (grayscale/RGB/HSV/L*a*b*). **By default, images are loaded in the RGB format**. SSIMS **will keep track** of the colorspace models throughout the workflow – for example, if we use (1) Convert to HSV, then (2) Convert to L*a*b*, and finally (3) Convert to RGB, the result will be identical to the original image. This is very important when using the "Single image channel" filter, which will single-out a specific image channel from a three-channel image – **channel order will always correspond to the last resulting colorspace** in the stack, or to the RGB model if no conversion was performed.

7.3 Filter parameters

Once we add the filter to the stack, we can **edit its parameters** by clicking on the corresponding green button in the stack. This action will open a window which will contain the description of the filter, and trackbar(s) + numeric box(es) which we can use to define the filter parameters (Figure 3 left). If the filter does not have any settable parameters, an appropriate message will be shown, and trackbars/numeric boxes will be omitted (Figure 3 right). Once the parameter values have been set, we can accept them by clicking on the **Apply** button.

The effects of individual filters can be examined by clicking on the **Preview this filter** button in the "Filter parameters" form. This will initiate a new window demonstrating the effects of just the selected filter. We can remove the selected filter from the stack by clicking on the **Remove** button in the Filter parameters form.

Tip: If the resulting image is grayscale, i.e., all three image channels are identical, the preview window will apply a colormap to provide more viewing contrast (Figure 4). However, once saved to disk, the images will be grayscale.

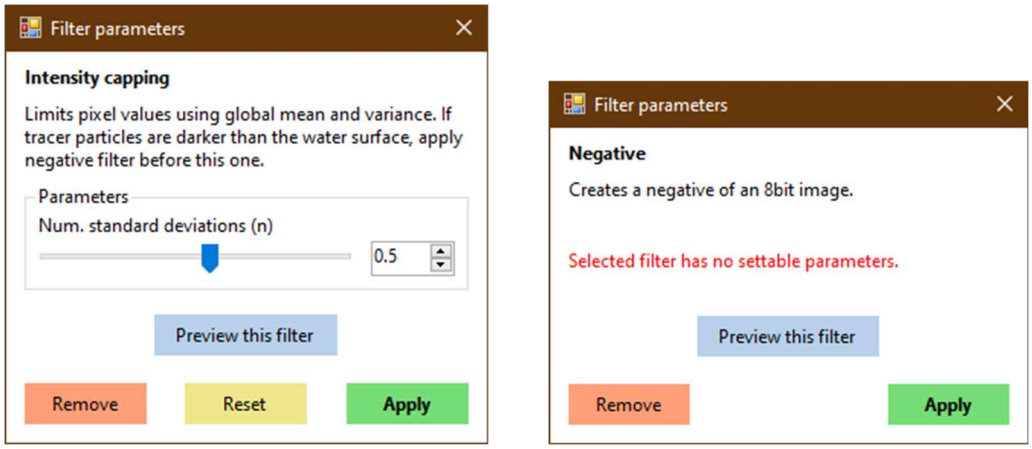

Figure 3. Examples of Filter parameters form.

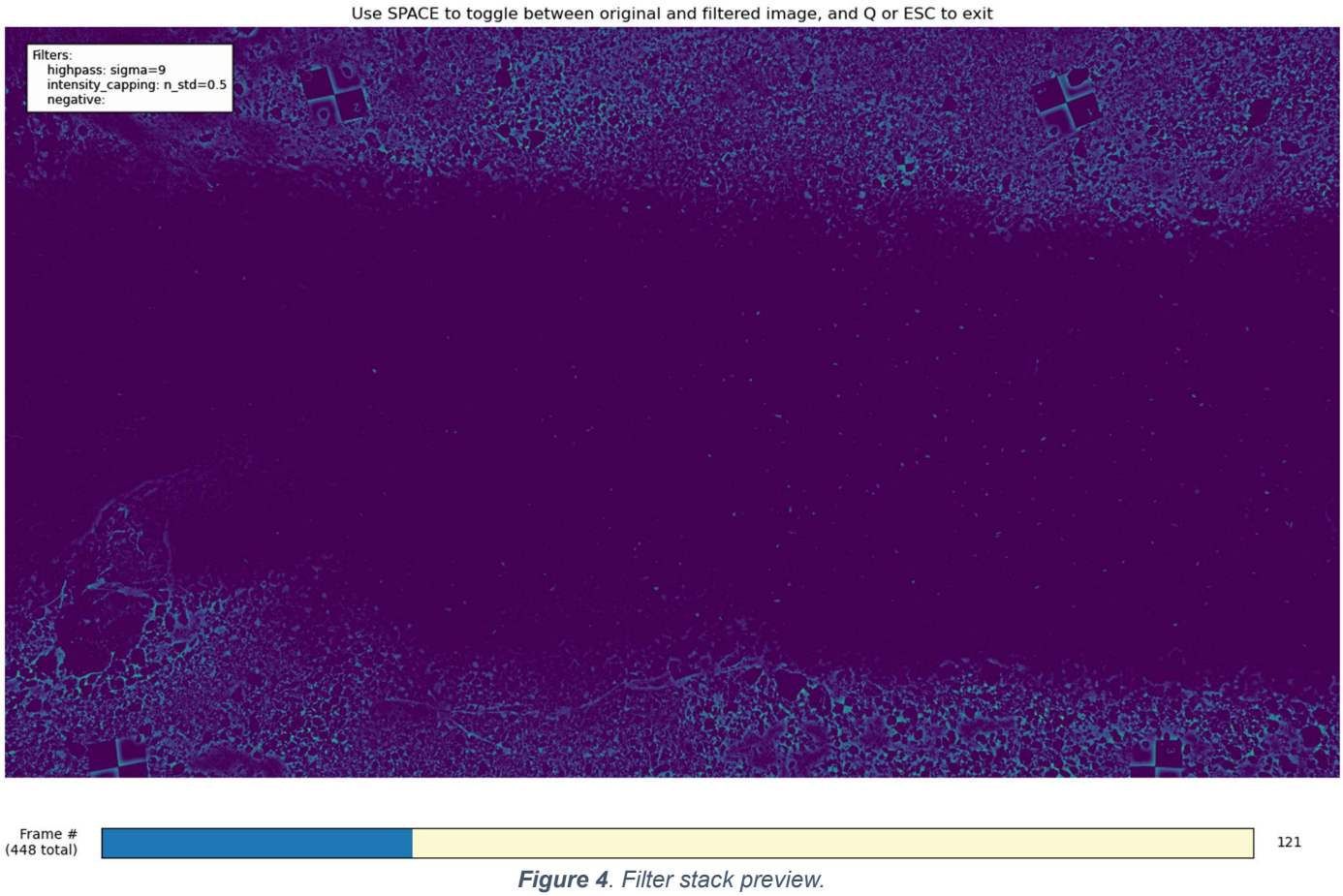

Figure 4. Filter stack preview.

7.4 Final results

When desired filters have been selected and their parameters are set, we can **preview their combined effect** by clicking the **Preview filter stack** button in the bottom center of the "Filter frames" form (Figure 2). This will be shown in a new window, as shown in Figure 4. List of applied filters and their parameters will be shown in the top left corner of the window. To fully examine the final results, a slider shown at the bottom of the window can be used to switch between the individual frames from the selected folder. Cross-examination between the original image and the filtered one can be performed by toggling the **SPACE** key on the keyboard.

Once we are happy with the results of our filter stack, we can start filtering all images from the selected folder by clicking on the **Apply filters** button in the bottom right of the "Filter frames" form. This will create a new folder next to the frames' folder with added suffix `_filtered`, where the filtered frames will be saved.

Appendix: Creating and modifying filters

Even though SSIMS comes with a number of available filters, user can implement and add own filters with relative ease. The procedure consists of two steps and requires basic knowledge of the Python programming language and familiarity with the OpenCV library.

Tip: The author of the tool welcomes **comments and suggestions** about new functionalities, **ideas** on what to include in the future releases, as well as constructive **criticisms** about the SSIMS tool, and encourages sending them directly at rljubicic@grf.bg.ac.rs or ljubicicrobert@gmail.com, or through the official GitHub repository.

Step 1: Creating filter definition

We need to edit the `filters.xml` file, located in the `%SSIMS_path%/scripts` folder. This file contains the definitions for individual filters, with the structure as shown in Figure 5. Definition requires:

`<Filter>…</Filter>`	= Enclosing tag of the single filter definition,
`<Name>…</Name>`	= Short name of the filter, to display in "Filter frames" form,
`<Func>…</Func>`	= Name of the Python function,
`<Description>…</Description>`	= Short description of what the filter does,
`<Parameters>…</Parameters>`	= Collection of filter parameters.

```
<Filter>
    <Name>CLAHE</Name>
    <Func>clahe</Func>
    <Description>Adaptive version of the histogram equalization with histogram clipping.</Description>

    <Parameters>
        <Parameter>
            <PName>Clip limit</PName>
            <PType>float</PType>
            <PUpper>10.0</PUpper>
            <PStart>2.0</PStart>
            <PLower>0.1</PLower>
            <PStep>0.1</PStep>
        </Parameter>
        <Parameter>
            <PName>Tile size</PName>
            <PType>int</PType>
            <PUpper>64</PUpper>
            <PStart>8</PStart>
            <PLower>4</PLower>
            <PStep>4</PStep>
        </Parameter>
    </Parameters>
</Filter>
```

***Figure 5**. Example of a filter definition from `filters.xml` file.*

`<Func>…</Func>` **must be equal** to the name of the Python function which handles the filtering. This will be explained further in Step 2. `<Parameters>…</Parameters>` collection contains definitions of all the individual parameters of the filter. Each parameter definition must contain the following:

`<Parameter>…</Parameter>`	= Enclosing tag of the single parameter definition,
`<PName>…</PName>`	= Name of the parameter,
`<PType>…</PType>`	= Can be either `int` or `float`,
`<PUpper>…</PUpper>`	= Highest allowed value of the parameter,
`<PStart>…</PStart>`	= Initial (default) parameter value.
`<PLower>…</PLower>`	= Lowest allowed value of the parameter.
`<PStep>…</PStep>`	= Increment/decrement step of the parameter value.

Multiparameter filters should contain consecutive `<Parameter>…</Parameter>` definitions. Filters with no settable parameters should just contain empty `<Parameters></Parameters>` collection.

SSIMS tool will now automatically recognize the filter definition and add it to the list in the “Filter frames” form.

Step 2: Creating a Python filter function

Second and final step of the procedure is to add a function which handles the actual filtering to the `filter_frames.py` file, also located in the `%SSIMS_path%/scripts` folder. **The name of the function must correspond** to the name defined in the corresponding `<Func>…</Func>` in the `filters.xml` file (Figure 6).

```
def clahe(img, clip=2.0, tile=8):
    print('[FILTER] CLAHE: clip={:.1f}, tile={:.0f}'.format(clip, tile))

    clahe = cv2.createCLAHE(clipLimit=clip, tileGridSize=(int(tile), int(tile)))
    img_gray = convert_img(img, colorspace, 'grayscale')
    img_clahe = clahe.apply(img_gray)

    return convert_img(img_clahe, 'grayscale', colorspace)
```

***Figure 6**. Example of a Python function which handles a filter.*

Function header should be defined as:

```
def function_name(img, *params):
```

where `function_name` corresponds to the `<Func>…</Func>` from the `filters.xml`, `img` is the initial (original) image and must be the first parameter, with the remaining parameters are to be listed after. The names of the remaining parameters do not have to correspond to their names from the `filters.xml`, but their total number and order should.

Keep in mind: Parameter values are **passed to the function as floats** by default. If you require their integer values, this conversion must be explicit – see end of line 4 in Figure 6.

The return value of the function is usually defined as:

```
return convert_img(resulting_img, colorspace_filter, colorspace)
```

where **`convert_img`** is the in-built function which handles all colorspace model conversions, `resulting_img` is the filtered image, `colorspace_filter` is the colorspace used when handling image inside the filter function. Third parameter `colorspace` should not be changed, as the `colorspace` global variable keeps track of the current working colorspace model.

For example in Figure 6, image was converted to grayscale model (line 5) from the working colorspace defined by variable `colorspace`, because OpenCV’s CLAHE function only works with single-channel images. However, the image was later returned to the working colorspace model (line 8) using an inverse transformation.

Keep in mind: Each filter receives a three-channel image and **must return a three-channel image**. This allows various filters to be performed sequentially, without the need to keep track of the previous image manipulations. If the output is a single-channel (grayscale) image, the image should be copied to channels two and three:

```
return cv2.merge([img_single, img_single, img_single])
```

The Python script will now be able to properly connect the filter definition to the handling function.