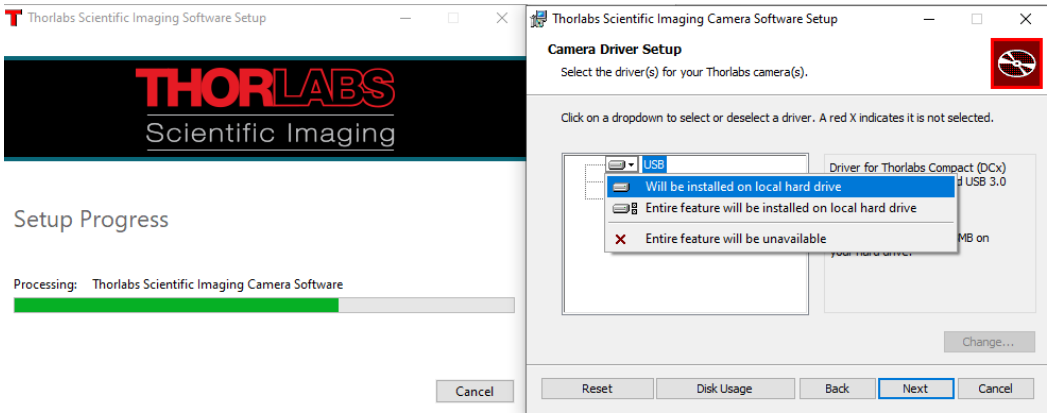
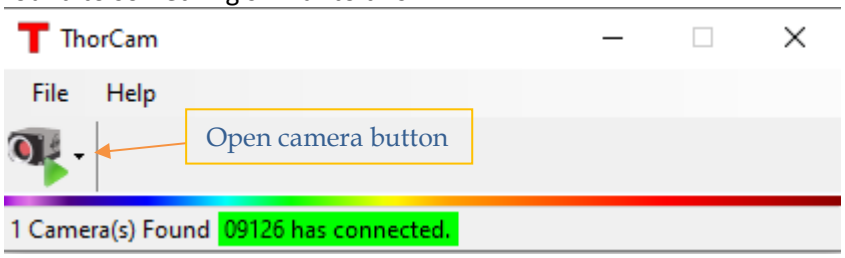
This is a quick start manual to get you started using a Thorlabs camera.

Using Thorcam (windows)

a) Go to the Thorlabs website and download the Thorcam Software package.
b) Install the Thorcam software with the USB driver selected:



c) Open Thorcam and connect the Thorlabs camera by inserting the USB into a USB3.0 compatible port on the computer. The Thorcam software should change from 0 Camera(s) found to something similar to this:



d) Click the open camera button and connect to the selected camera.

Now you can view, capture and save images or video from the camera. Also basic functionality such as changing contrast and drawing lines is included.

(optional) Using your favorite programming language using Thorlabs API's

a) Locate the Thorcam installation folder. This is probably C:\Program Files\Thorlabs\
b) Open \Scientific Imaging\Scientific Camera Support\Scientific_Camera_Interfaces.zip\Scientific Camera Interfaces\
c) Information about using the API can be found here. The key to use the selected API is to copy the folder of the selected API to some working directory, and after that, copy the DotNet or the Native .dll files into the appropriate folder, as described in the readme files provided with each API.

(optional) Use python with LPM library to get a Thorlabs camera working with openCV

LPM has provided a library and some examples to get you started with using a Thorlabs camera (and also many other cameras) and do Realtime image processing. It is here possible to use the OpenCV functionality. OpenCV is a very powerful library with many tools and functions to filter or process images in almost any way.
First we need to Install python and numpy, matplotlib, and opencv packages. The LPM library is tested with the Spyder IDE with Anaconda. Other environments may work, but this is not tested.

Follow the steps described below to install Anaconda with the appropriate packages:

a) Download and install the latest version of Anaconda with the standard settings.
b) Open the application `Anaconda prompt(Anaconda 3)` and run the following commands:
   - `conda update numpy`
   - `conda update matplotlib`

- `pip install opencv-contrib-python`

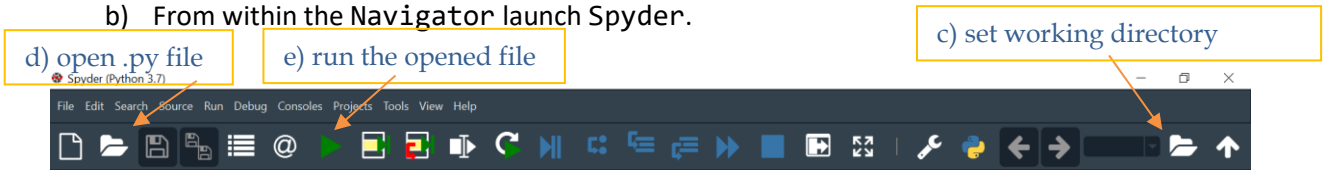
Navigate to the LPM_microscope folder using `cd <folder name>` or `cd ..` to go up.

- `pip install ".\Python Compact Scientific Camera Toolkit\thorlabs_tsi_camera_python_sdk_package.zip"`

c) Close `Anaconda prompt`

Now everything is ready. To run examples follow the steps below:

a) Open the application `Anaconda Navigator (Anaconda 3)`.
b) From within the `Navigator` launch `Spyder`.



c) Set working directory the `LPM_Microscope` folder
d) Open `show_cameras.py`
e) Run `show_cameras.py`. This should print something like the following output:

```
OpenCV camera ID number(s) detected under standard driver:
[0]

No Thorlabs cameras detected!
```

It should print an ID for the Thorlabs camera if you connected the Thorlabs camera already.

f) Open and run `LPM_simple_example.py`. It will try open and show the first camera found under the directshow driver (this is the standard windows driver. If you're not on windows: remove `cv2.CAP_DSHOW` flag in `\LPM_image_capture_library\opencv_camera.py`).
If a Thorlabs camera is connected to the computer and an ID was shown in when show_cameras.py was run, you can change:

Now it will show the output of the Thorlabs camera.

The while loop in LPM_simple_example.py handles two processes: retrieving the frames from the camera with camera.get_frame, and showing the retrieved frame with show.frame. Inbetween these processes we could do image processing on the frames. For example, if you add `import cv2` at the top, then you could put `img = cv2.normalize(img, None, 0, 255, norm_type = cv2.NORM_MINMAX)` in between the processes to maximize the intensity range.
However, when running this capturing, image processing, and showing simultaneously the throughput of the frames is limited (depending on the size of the images and the type of filtering). Therefore multiprocessing is a real must if realtime viewing and processing is required. The main challenge of multiprocessing is communication between the processes and the correct routing of the data. LPM has provided a solution with its library.

g) Open the LPM_multiprocessing_example.py. For routing the data it is needed to know the size of the frames in advance, so check if img_shape represent the correct frame size and correct if needed.
h) The procs dictionary datastructure is at the heart of interconnecting the various processes. Fill in the image processing function name at the (2) core_function entry, and the needed arguments at (4) as shown by the prepared example.
i) Run the example and play around.

Possible functionality that would be useful to implement in the future versions for multiprocessing:

- Video saving functionality using cv2.VideoWriter
- Importing images functionality using cv2.imread as alternative to cv2.VideoCapture
- Dynamically parse arguments to image processes using multiprocessing.Manager()
- Add frame rate monitor
- Auto detect image size