Wind-Wave Documentation  
Jeff Colvin, July 2020

# Python Code

The Python scripts were written using Anaconda3 (with Python 3.7) and OpenCV 4.3. To install OpenCV, use the command: ‘pip install opencv-python’ from the Anaconda Shell.

**camCalibrate.py**: This script will save the camera calibration parameters as a numpy (.npz) file. The calibration parameters are determined from a short video recorded with the same camera settings as will be used in the field. For this research, a GoPro HERO 4 Silver edition camera was used in 1080p (HD) mode with a frame rate of 60 fps. To make the calibration video, mount a checkerboard pattern of known size to a flat surface and start recording a short video while moving the camera around at different angles and distances with respect to the pattern.

The data directory has a sample calibration video (GoProCalibVideo.mp4) from the GoPro used in this research. Also included in the data directory is the checkerboard pattern (checkerboard.svg) in vector-graphics format (to ensure the correct size when printed) that can be used to create the calibration video for any desired video camera.

The script will prompt the user for the name of the video file to use for calibration. It can be run using the sample calibration video, GoProCalibVideo.mp4. However, the user will want to print out the checkerboard pattern and create a calibration video using his/her own camera. The script will then ask for the number of calibration images to capture. For this research, 15 calibration images were used. The video will start playing and the user needs to hit the ‘space’ key to capture each calibration image. Be sure to capture all of them before the video ends. Also, try to capture images when the checkerboard pattern is at different angles and distances from the camera. Once all calibration images are captured, the script will locate the corners of the squares that make up the checkerboard pattern on each image and save the calibration parameters. The script will then undistort all the calibration images. Samples of the images generated – the calibration images, the corner location images, and the undistorted images – are included in the data directory (calimage.png, calimage\_corners.png, calimage\_undist.png) and shown in Figure 1.

  

Figure 1. Example images output from the camera calibration process, created by camCalibrate.py

**createTimestack.py**: This script will use a previously created camera calibration file (from camCalibrate.py) and use it to undistort each frame of a user-selected video file. From each video frame the script will strip a single column of pixels, specified by the user, and stack the columns together in order to form a timestack image. The timestack image has units of time on the x-axis and distance and/or height on the y-axis. The script will open a blank timestack image and the columns will fill in left to right as the script progresses. When complete, the image will close and be saved in the current directory.

The script will prompt the user for the following inputs:

1. **Name of source video**: Type the name of the video from which to generate the timestack.
2. **Column of video frame to be extracted**: The script will extract this column from each frame of the video and stack them together to form the timestack image.
3. **Framestep of timestack**: The framestep tells the script to use every frame (=1), every other frame (=2), every 3rd frame (=3), and so on. This determines the frequency of the timestack. For example, with a 60fps video, the frequency would be determined from the timestep as follows: 1 = 60fps, 2 = 30fps, 3 = 20fps, 4 = 15fps, and so on.
4. **Duration of the timestack**: Enter the duration of the timestack (in seconds). This is limited by the length of the video and the chosen starting point (step 5) within the video.
5. **Number of seconds into video for timestack creation to begin**: The script will start at this location within the video and proceed for the length of the duration given in step 4.
6. **Name of camera calibration file**: This is the name of the calibration file created from the camCalib.py script.

The script can be run with the example camera calibration file (camcalib.npz) and GoPro video (GoProWaveVideo.mp4) included in the data folder. However, because of file size limitations on GitHub, the sample video is only 10 seconds long, so be careful of the settings entered in steps 4 and 5 above.

An example timestack image (timestack\_60fps\_64s.png) is also included in the data folder and shown in Figure 2.

A picture containing outdoor, water, grass, herd

Description automatically generated

Figure 2. An example timestack image output from createTimestack.py. This image was generated from 1080p (HD) video recorded at 60 fps with a timestack duration of 64 s.

**vidUndistort.py**: This script will use a previously created camera calibration file (from camCalibrate.py), undistort each frame of a user-selected video file, and save the undistorted version of the video in the same directory. It was written as a precursor to createTimestack.py and is included here in case the user simply wants to undistort a video. Note that the output video is saved in .avi format (a limitation of OpenCV).

The GoProCalibVideo.mp4 was undistorted using this script. The output video is GoProCalibVideo\_undistort.avi and is available in the data folder.

# R Code

The R scripts were updated to R version 4.0.2 in July 2020. However, these scripts are still very much “science code”. The scripts are commented throughout to indicate lines that a user will likely want to edit.

The scripts make use of the EBImage library (<https://github.com/aoles/EBImage>) which is installed by starting R and entering:

Install.packages(“BiocManager”)  
BiocManager::install(“EBImage”)

**SpectralTS.R**: This script will read in a timestack image generated by createTimestack.py and a row number of the image (see script comments for more information). The script will convert the timestack image to grayscale and perform a Gaussian blur (low pass filter) before extracting the specified row of the image and creating a timeseries of the normalized pixel intensities. A Fast Fourier Transform (FFT) is then performed and a spectral plot is generated, from which the user can obtain the peak wave frequency.

The script is coded to use row number 655 of the included timestack (hm1timestack\_60fps\_64s.png) and should produce the plots shown in Figure 3.

A picture containing bird

Description automatically generated  
A picture containing bird

Description automatically generated

Figure 3. The normalized pixel intensity time series plot (top) and the corresponding FFT frequency plot.

**WaveHeightTS.R**: This script will read in a data file containing pixel rows indicating a waves crest and trough position in the video frame. This data was determined by manually analyzing the timestack image. A sample file (wavedata.txt) from a timestack during Hurricane Matthew is included in the data folder. In addition to providing the crest and trough pixel location data, the user must also specify the height of the camera above still water level and the position of the optical axis, which comes from the camera calibration file generated by camCalibrate.py.

The script will then calculate the wave height of each crest-trough pair found in wavedata.txt as well as the average wave height, which was found to correlate with the significant wave height as determined by a wave gauge (See article for details).

**WaveAnalysisWG.R**: This script reads in a data file from the wave gauge and calculates various wave parameters using both a zero down-crossing method and a variance-based method. A sample data file (hm1\_WG.txt) is included in the data folder.

**WaveAnalysisTS.R**: This script reads in pixel intensity time series data generated from a timestack and calculates various wave parameters using both a zero down-crossing method and a variance-based method. A sample data file (hm1\_TS.dat) is included in the data folder.

This script also includes a scale factor for the pixel intensity data. So far, this factor was determined experimentally by trial and error. Methods to determine this scale factor directly from a timestack are being explored.

Current work continues on both determining the scale factor, as well as experimenting with various image processing methods to automate the process of locating the wave crests and troughs from a timestack. The success of either one would automate the entire process of extracting both wave period and significant wave height from a timestack image.