# RCM

This documents the code for the reconfigurable microscope made using components from Thorlabs.

Includes the following modules:

* camera.py: methods for the RGB camera and hyperspectral camera
* light.py: methods for the DC2200 LED driver
* Spectrometer.py: methods for the USB driver to control the spectrometer
* stage.py: methods for the MCM3000 and MCM3001 3-axis controllers
* tunablefilter.py: methods for the Kurios liquid crystal tunable bandpass filters
* RCM.py: contains methods to carry out data collection using the RCM
* main.py: edit this file to run your programs

## RCM.py

Class FullControlMicroscope

* Parameters
  + wavelength\_range: range of wavelength in nm, default is [420, 730] (the spectral range of the tunable filter)
  + no\_spectra: number of wavelengths to take images at
  + exposure\_time: time of exposure for each image in seconds
  + save\_folder: path to the folder where saved images will be stored
* Methods
  + stage\_check()
    - Called during initiation of an instance of the class to ensure the stage is centered correctly and prevent overextension of motors that can cause damage
  + close()
    - Closes connection to all peripherals
  + save\_image(wavelengths, hypercube, indices)
    - For every wavelength in *wavelengths*, saves the image stored in the list *hypercube*, with identifiers in *indices* appended to the name of each image
  + acquire\_HS\_datacube()
    - Takes an image at each wavelength specified by *wavelength\_range* and *no\_spectra* and returns a list of the wavelengths and a list of their respective images. (hyperspectral imaging)
  + take\_HS\_time\_series\_images(time\_increment = 10, total\_time = 7200)
    - Performs a hyperspectral imaging every *time\_increment* seconds up until *total\_time* has passed and saves the image using save\_image()
  + mapping(sample\_dim = [], sample\_no\_per\_channel = [], RGB\_img\_too = False)
    - Divides up the stage range into a grid of dimensions *sample\_dim* [1, 1] means 1 by 1 grid) or grid with number of boxes *sample\_no\_per\_channel* ([1,1 ] means 1 box along both axes) or according to both (*sample\_dim* of [1,1] and *sample\_no\_per\_channel* of [1,1] means 1 box of 1 by 1 in both axes)
    - Does hyperspectral imaging on each box, with RGB imaging as well if *RGB\_img\_too* is true

## camera.py

Class Camera\_HS

* Note
  + Class for the hyperspectral camera
* Methods
  + single\_exposure(exposure\_time = 1)
    - Sets the exposure level to *exposure\_time* and returns a numpy array that can be saved as an image.
  + average\_exposure(exposure\_time = 1, averages = 5)
    - Sets the exposure level to *exposure\_time*, takes n = *averages* images, then returns the arithmetic mean of the images.
  + multi\_exposure(start\_exposure=1e-4, doubles=10, discard\_ratio=0.2)
    - Takes images with multiple exposure times, beginning from *start\_exposure* (s), with *doubles*doubling, then returns the arithmetic mean of the images.

Class Camera\_BA

* Note
  + Class for the RGB camera
* Contains the same function as above

Functions

* live\_time\_lapse(folderName)
  + Uses the hyperspectral camera to continuously take images with exposure time of 0.3s and storing it in *folderName*.

## light.py

Class DC2200

* Methods
  + Set\_brightness(percent = 1.1)
    - Set brightness to *percent* %
  + on()
    - Turns LED on
  + off()
    - Turns LED off
  + close()
    - Closes connection with LED driver

## stage.py

Class Controller

* Notes
  + um stands for micrometers
  + counts are the quantized unit of measurement used by the motor to determine position
  + Channel refers to a motor
  + Method names beginning with an underline are private methods used within public methods
* Parameters
  + which\_port: name of USB port connected to the 3-axis motor controller
  + name: either “MCM3000” or “MCM3001”.
  + stages: tuple of 3 motors for each axis, supports the following stages –“ZFM2020”, “ZFM2030”, and “MMP-2XY”.
  + reverse: tuple of 3 boolean values, set to True if the respective motor is reversed.
  + verbose: enables status print messages
  + very\_verbose: enables even more status print messages
* Methods
  + set\_encoder\_counts\_to\_zero(channel)
    - Sets the encoder count of the motor *channel* to 0, ensure the motors are all in the middle of movement range before zeroing
  + move\_um(channel, move\_um, relative = True, block = True)
    - Checks if the move is legal (does not exceed limits of the motor)
    - *Block* prevents motor movement
    - Either moves motor *channel* by *move\_um* relatively to its current position if *relative* is true or relative to its 0 point if false
  + close()
    - Closes port connection to motor controller
  + \_legalize\_move\_um(channel, move\_um, relative)
    - Checks if moving motor *channel* by distance *move\_um* exceeds limits of the motor
    - Raises exception if it exceeds limits]
  + \_finish\_move(channel, polling\_wait\_s = 0.1)
    - Checks the motor’s position every *polling\_wait\_s* seconds and stops movement when the count is within tolerance
  + \_encoder\_counts\_to\_um(channel, encoder\_counts)
    - Converts *encoder\_counts* of motor *channel* into micrometers
  + \_um\_to\_encoder\_counts(channel, um)
    - Converts *um* to counts of motor *channel*, rounding down
  + \_send(cmd, channel, response\_bytes = None)
    - Writes the command *cmd* serially to control motor *channel*, with an optional message *response*
  + \_get\_encoder\_counts(channel)
    - Runs during the initiation of a new instance of the class to obtain the counts and encoder of motor *channel* from the motor controller

## tunablefilter.py

Class TunableFilter

* Methods
  + open()
    - Initiates the tunable filter by calling CommonFunc() (below)
  + closeI()
    - Stops connection with the device
  + set\_bandwidth(bandwidth = 2)
    - Sets bandwidth mode to *bandwidth*.
      * 1: Black
      * 2: Wide
      * 4: Medium
      * 8: Narrow
  + set\_wavelength(wavelength = 550)
    - Sets the wavelength of the tunable filter, only works between the range of 420 to 730 nm

Functions

* Note
  + Only meant to be used internally within the file to initiate the tunable filter
* GetDeviceSNCN(IDStr)
  + Returns the optical head’s serial number and the controller’s CN of the device with id *IDStr*
* GetDeviceOpticalHeadTypeNumber(hex\_string, target\_dict, max\_length = 4)
  + Returns spectrum range and available bandwidth mode
* CommonFunc(serialNumber)
  + Starts connection with the tunable filter, returns information including id, status, temperature, specification, spectrum range, and bandwidth.