Chapter 4

Zeiss Axio Lab.A1

The following is a brief introduction to the Zeiss Axio Lab.A1 microscope that we will use for the course. The microscope is set up for image acquisition on bioimager, the iMac sitting next to it.

4.1 Cleaning Up

This is important enough to mention first. When you are done using the microscope for the day, and no one else is waiting to use it, do the following:

1. Switch the objective turret back to the 5× lens.
2. Remove the specimen from the stage and place it in the slide box.
3. Turn off the microscope lamp.
4. Disconnect the back plug from the Firewire hub.
5. Cover the microscope with the blue dust cover, making sure the cover comes around the stage, down to the table.

4.2 Alignment

You should align the microscope for proper Köhler illumination before each use. You can do this either while looking through the ocular lenses or using a digital video stream.

1. Switch the objective turret to the 10× lens.
2. Place a high contrast specimen on the stage. The leaf litter or octopus muscle tissue are good examples.
3. Turn on the lamp (green button on left). Once the lamp is on, keep it on until you are done using the microscope. Turning this on and off could damage the lamp!

4. Adjust the lamp intensity to a comfortable viewing brightness. The blue tape on the lamp adjustment should be oriented on top of the knob.

5. Focus on the specimen

6. Close the field diaphragm until you can see the blades in the field of view.

7. Adjust the condenser focus knob to focus to sharpen the image of the field diaphragm.

8. Open the field diaphragm until the blades disappear from view.

9. Remove the ocular and look down the tube of the microscope.

10. Close the aperture diaphragm so it fills about 75% of the field. For the 5× lens, this is just to the right of the right piece of blue tape on the condenser. For the 10× lens, this is at the right piece of blue tape. For the 40× lens, this is at the left piece of blue tape.

Note that you should not need to adjust the set screws on the condenser.

This process is simple to perform: the minor adjustment is simply to focus on the specimen, close the field diaphragm, adjust the condenser so the field diaphragm is in focus, and open the field diaphragm. You should really do this frequently: when you start using the scope, when you switch lenses, and when you switch specimens.

There is a manual for the microscope in the lab as well as a textbook on microscopy [Murphy, 2001].

4.3 Cameras

The cameras attached to the microscope are PointGrey Research Flea2 IEEE 1394 (Firewire) color cameras. To use them, plug the Firewire connection from the computer into the Firewire hub. Be sure to unplug the hub when you are done.

The pair of cameras can just be left mounted on the microscope, on their mounting brackets. You should not need to remove them if you use the tape guides on the condenser for adjusting the aperture diaphragm during Köhler alignment.

The `FirewireVideo.py` file provided on the website is a Python interface to a wrapper for libdc1394, a C library that provides video capture from Firewire cameras. Running FirewireVideo.py from the command line will capture and display a video stream from the camera.

It is normal to see error messages every line; we may get a workaround for this at some point.

There are manuals for the cameras in the lab.
4.4 Setup

To get the FirewireVideo program running from your own account, you need to first compile the associated C library. The Python program expects to find the C library in the same directory. To compile and test:

```bash
$ cmake .
$ make
$ ./fwtest 5
$ display image-000.ppm
```

You should see 5 .ppm files created. The display program views one of them. Now FirewireVideo.py should also work.

4.5 FirewireVideo

The FirewireVideo.py file provides access to the Firewire camera, and enables nearly complete control of the camera hardware. There are a few important points about connecting to an using the cameras to acquire images. Listing 4.1 provides an example.

Listing 4.1: FirewireVideo.py test code

```python
def testStart():
    # connect to firewire camera, select frame format
    fwcam = FirewireVideo(0, DC1394_ISO_SPEED_800)
    fwcam.setVideoMode(DC1394_VIDEO_MODE_640x480_RGB8,
                        DC1394_FRAMERATE_30)

    # set up camera parameters
    fwcam.setExposureAbsolute(brightness=0, gamma=1.0)
    fwcam.setAutoExposure(True)
    fwcam.setColorAbsolute(whiteBlue=1023, whiteRed=276)

    # start grabbing video frames
    fwcam.startTransmission()

    # display frames forever
    index = 0
    poll = 100
    t0 = time.time()

    key = None
    while key != 27:
        # grab a frame
        frame = fwcam.acquireFrame()
        key = imgutil.imageShow(frame, "fwvideo", False, 10)

        index += 1
        t1 = time.time()
```

35
Camera control often involves providing special flags to functions, for example, to choose a video frame size and color coding. The file `dc1394.py` contains all these codes; this is imported at the top of `FirewireVideo.py` using `from dc1394 import *` because the flag variables are quite long, e.g. `DC1394_VIDEO_MODE_640x480_RGB8`.

The FirewireVideo constructor connects to a camera by selecting a device ID (0 or 1, if two cameras are connected) and selecting a transmission speed for the IEEE 1394 bus. The cameras can run at either 400 or 800 rates, which corresponds to how often data packets are sent over the bus. To run at the faster rate (800), use a IEEE 1394b cable.

Before you can acquire images from the camera, you need to tell the camera what type of images you want. The `setVideoMode` lets you select a frame format and frame rate. Available video modes are printed to the console when you first connect to the camera. If you specify an illegal video mode and frame rate combination for the camera, the program will crash or you will not get any frames. You can safely ignore the `FORMAT` video mode, which is what the optional parameters for this function deal with.

After setting the video mode, you need to start transmitting data from the camera using `startTransmission()`. Eventually you will need to use `stopTransmission()` to stop the camera.

Many camera features can be controlled. See the output from the function `printFeatures()` to see what you can control. The PGR Flea cameras can have features set using either register values or absolute values. Absolute values are easier to set, because the values you provide correspond to real values instead of some unsigned integer code.

Use the `fwcam.acquireFrame()` to grab frames. This will always return an RGB frame as a NumPy array, regardless of the color format selected using `setVideoMode`. Note that the order of color channels is RGB, not BGR.

The camera will acquire frames at its configured frame rate. You set the maximum frame rate when you configure the video mode, but you can also adjust the frame rate using `setFrameAbsolute()`. If the software does not grab frames from the camera fast enough, the circular ring buffer (in the C library) will fill up, and you will start dropping frames. When this happens, you will
see the following message as the C library detects this situation and flushes the ring buffer.

1 FV_acquireFrame: lagging 8 frames
2 FV_FlushRingBuffer
3 ...7 ...6 ...5 ...4 ...3 ...2 ...1 ...0...done

This is not necessarily a problem, but when this happens you will probably notice lag in the video display. If this happens frequently, you may want to drop your frame rate.

FirewireVideo is pretty well commented. Read the comments to find out more. If you are interested, you can also consult the documentation for the libdc1394 C library on which this code depends. This may include more information than you want about controlling Firewire cameras.