New Mirror Reflectivity Test Station

Tyler Lemon
2022-03

New Mirror Reflectivity Test Station

I developed a new LabVIEW program (Fig. 1) for acquiring data from two Thorlabs compact CCD spectrometers (CCSs) for a new reflectivity test station.

This code uses Thorlabs' drivers for communicating to the CCSs to set the CCSs' data acquisition properties and also to read the number of CCD counts detected at each wavelength. The LabVIEW program utilizes a state machine layout to iterate through all device set up and data acquisition modes.

The program first performs its own initialization, resetting its front panel indicators. Next, the program prompts the user via a pop-up window with a text input box to enter the name of the mirror to be tested. The program then enters the state machine, starting with the "initialize" state.

At this point, the program also starts monitoring for any errors in CCS communication. If at any point, an error is received, or the CCS response is not as expected, the program will notify the user which CCS (the reference CCS, measurement CCS, or both) is the source of the error. After the user acknowledges the error, the program enters the "close out" state. The "close out" state gracefully stops communication to the CCSs, writes a status message that indicates reason of program stop to a front panel indicator for user reference, and then moves the program to the "summarize" state.

In the "initialize" state, the program opens communication to each of the CSSs that are connected to the PC running the LabVIEW program using USB cables. From the response of each of the devices, the program checks whether it is indeed a CCS connected at that USB port and also verifies that the serial number of the CCS matches the serial number of the CCS designated for measuring the reference light source or measuring the reflected light off of the mirror. If there are no errors after establishing communication to the CSSs, the program enters the "set up" state.

In the "set up" state, the program uses a LabVIEW control (that can be changed by the user), to set the integration time of the two CCSs. The integration time is the time duration that the CCSs record CCD counts over. The program then reads the integration time setting back and if the integration time is not within 5% of the user input, the program gives an error, triggering it to begin its error indication routine noted above. If there are no errors and integration time is able to be set, the program continues on to the "live feed" state.

- **Developed a new program using new equipment for the mirror reflectivity test station**
- **A LabVIEW program that uses a state machine was developed that displays a live feed of data for test station alignment, records data for a user-settable number of samples, and saves that data to a text file**

New Mirror Reflectivity Test Station

In the “live feed” state, the program displays measurement counts from both of the CCSs in one graph. Additionally, the program calculates the reflectivity of the tested mirror and displays that result on the same graph but on a separate y-axis. When the program is in this state, the there are two user inputs that prompt the program to move to a different state. By changing the CCSs’ integration time, the program returns to the “set up” state. If the user clicks the “Capture Data” control, the program enters the “record” state.

In the “record” state, the program acquires a user-settable number of samples from each CCS and of the reflectivity calculation result. After successful data acquisition, the program then enters the “save” state.

In the “save” state, programmatically checks whether there is an existing results directory in the directory of the program, creating one if there is not. The program then creates in the results area based on the mirror name the user entered at program start up (if one does not already exist) and then creates a new directory to store the data for the spot measured in the “record” state. In that spot directory, it then creates yet another directory to store the raw data from the “record” state and saves the raw data in that directory in a text file. The program then averages all samples of the raw data for the reference CCS, measurement CCS, and reflectivity measurement and writes those averages and standard deviations to the spot directory. At this point, the program returns to the “live feed” state so the user can realign test station to a new spot on the mirror.

In the “summarize” state, the program determines whether any data were acquired for a spot, and if so, it calculates the overall reflectivity average of each spot, and generates a summary text file (an example is in Fig. 2). Finally, the LabVIEW program stops.

During development, one of the difficulties faced were compensating for the attenuation of light through the longer length of fiber optic cable that the measurement light must pass through before being measured by the measurement CCS. This obstacle was overcome by including an attenuation calculation based on the specifications (provided by Thorlabs) of the fiber optic cable in the measurements from the measurement CCS. Another obstacle faced was the noisy reflectivity measurements at lower wavelengths due to the overall smaller intensity of the source light at those wavelengths. This was overcome by adding the “record” state that acquires multiple samples that are averaged over, effectively eliminating the noise from the averaged result.

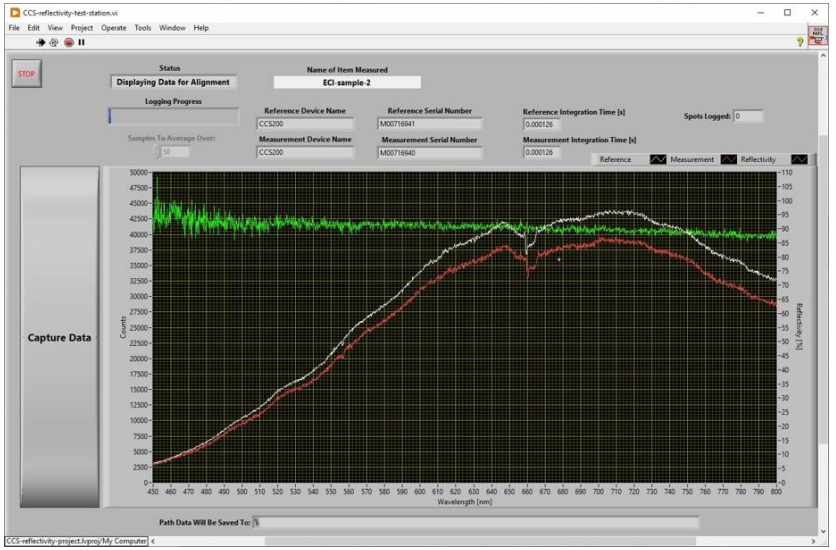

FIG. 1: Screenshot of program’s front panel. Green trace is reflectivity calculation (uses right y-axis). White trace is reference CCS data (uses left y-axis). Red trace is measurement CCS data (uses left y-axis).

```
2022-04-21 12:02:14

RICH-1-leftover-23

Reference CCD
Name: CCS200
Serial Number: M00716941
Integration Time [s]: 0.000100

Measurement CCD
Name: CCS200
Serial Number: M00716940
Integration Time [s]: 0.000100

Number of spots measured: 3

Samples Averaged Over: 1000

Overall reflectivity averages:
Spot #  Average Reflectivity [%]
Spot 1  0.94
Spot 2  0.94
Spot 3  0.94
```

Fig. 2: Example summary file output by program’s “summarize” state