Monochromator Control System

Stepper-Driven Precision Wavelength Positioning

This project revives a decommissioned Princeton Instruments / Acton VM-504 vacuum monochromator that originally relied on an SD3 controller and Windows xp software and SDK (now lost). The goal is to restore full wavelength positioning using a custom serial based controller, using only two optical sensor signals as positional feedback.

The VM-504 is a Czerny-turner monochromator with a rotating diffraction grating driven by a stepper motor through a worm reduction gear, enabling controlled wavelength output at the exit slit.

How the monochromator positions wavelength (mechanical overview)

Light enters the entrance slit, reflects off internal optics, and hits a diffraction grating. The grating angle determines which wavelength exits through the exit slit.

Mechanically:

- The grating rotates via a worm gear reduction stage driven by a stepper motor.
- Two optical sensors observe “slit windows” on two rotating disks:
    - **Sensor S2 (worm gear disk)**: a slit on the large worm gear disk → provides an absolute “once-per-rev style” reference window tied to the grating/worm gear position.
    - **Sensor S1 (motor shaft disk)**: a slit on the small disk attached to the stepper shaft → provides stepper rotation / fine phase reference.

These two infrared sensors are the only feedback signals available (OPEN/BLOCKED).

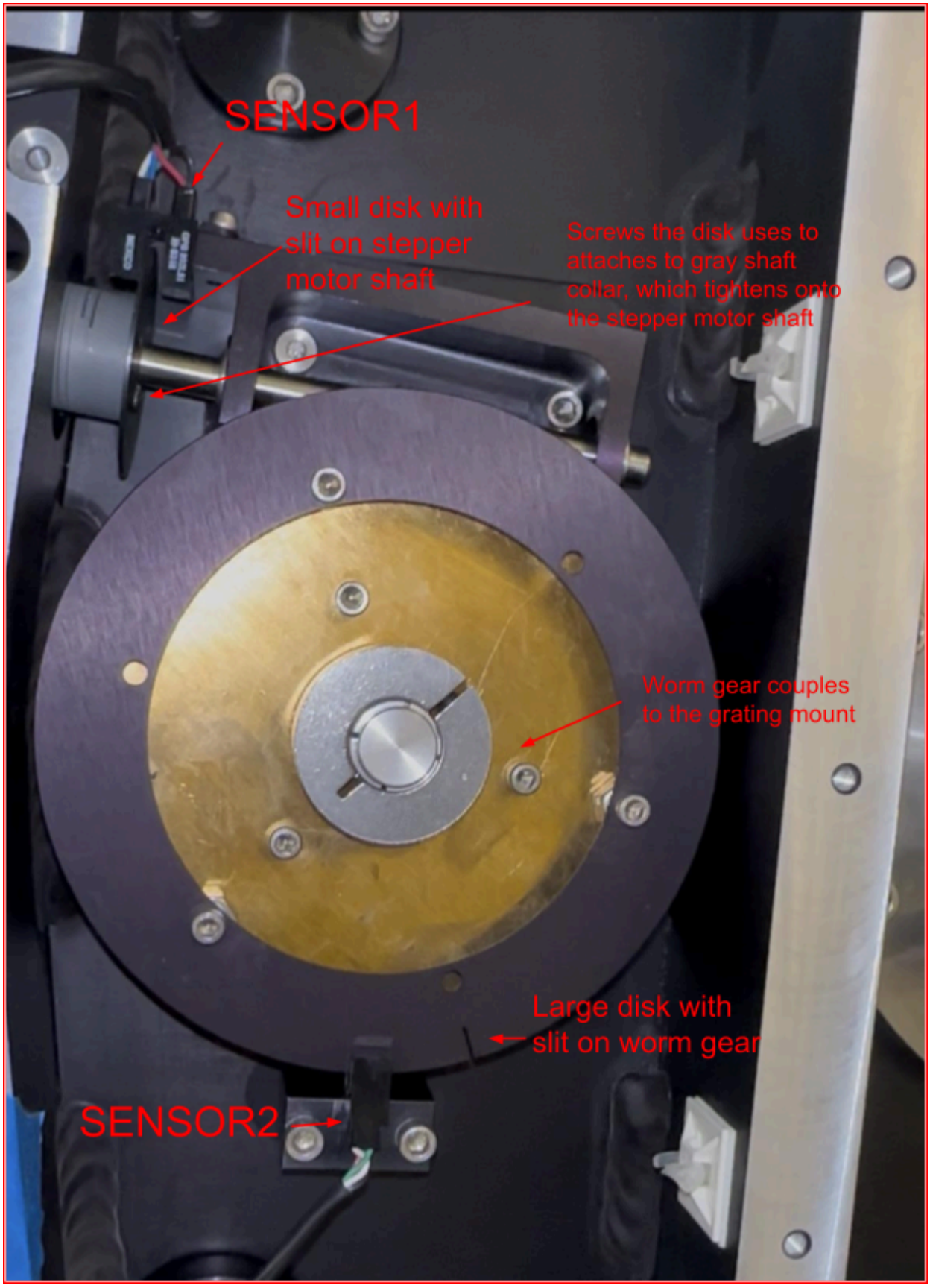
-> About every full revolution of the stepper motor is ~100 steps or about ~2° on the grating.

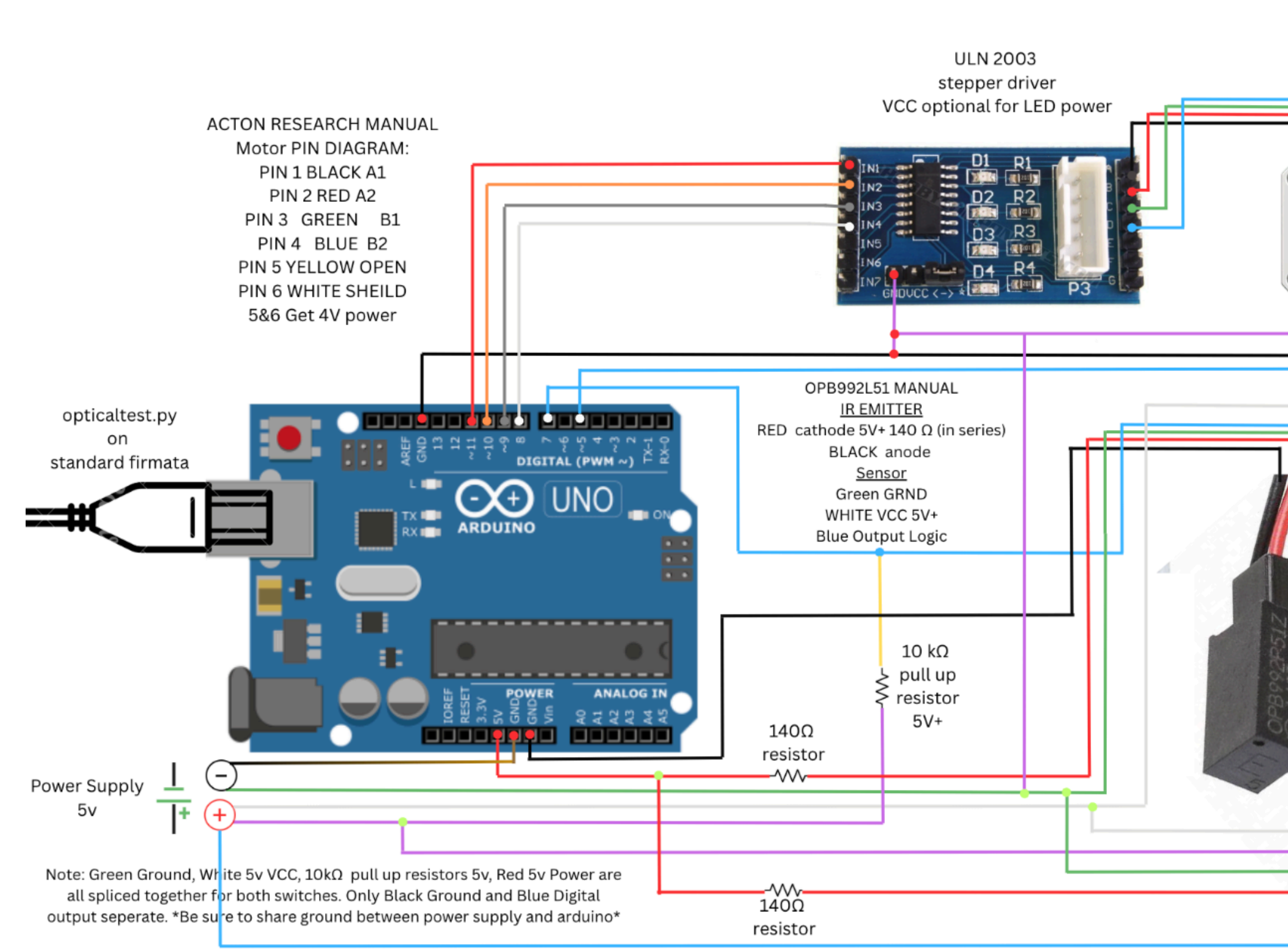
->One full revolution of the worm gear is 18000 steps, and since the grating is free turning, it can continually turn in either direction.

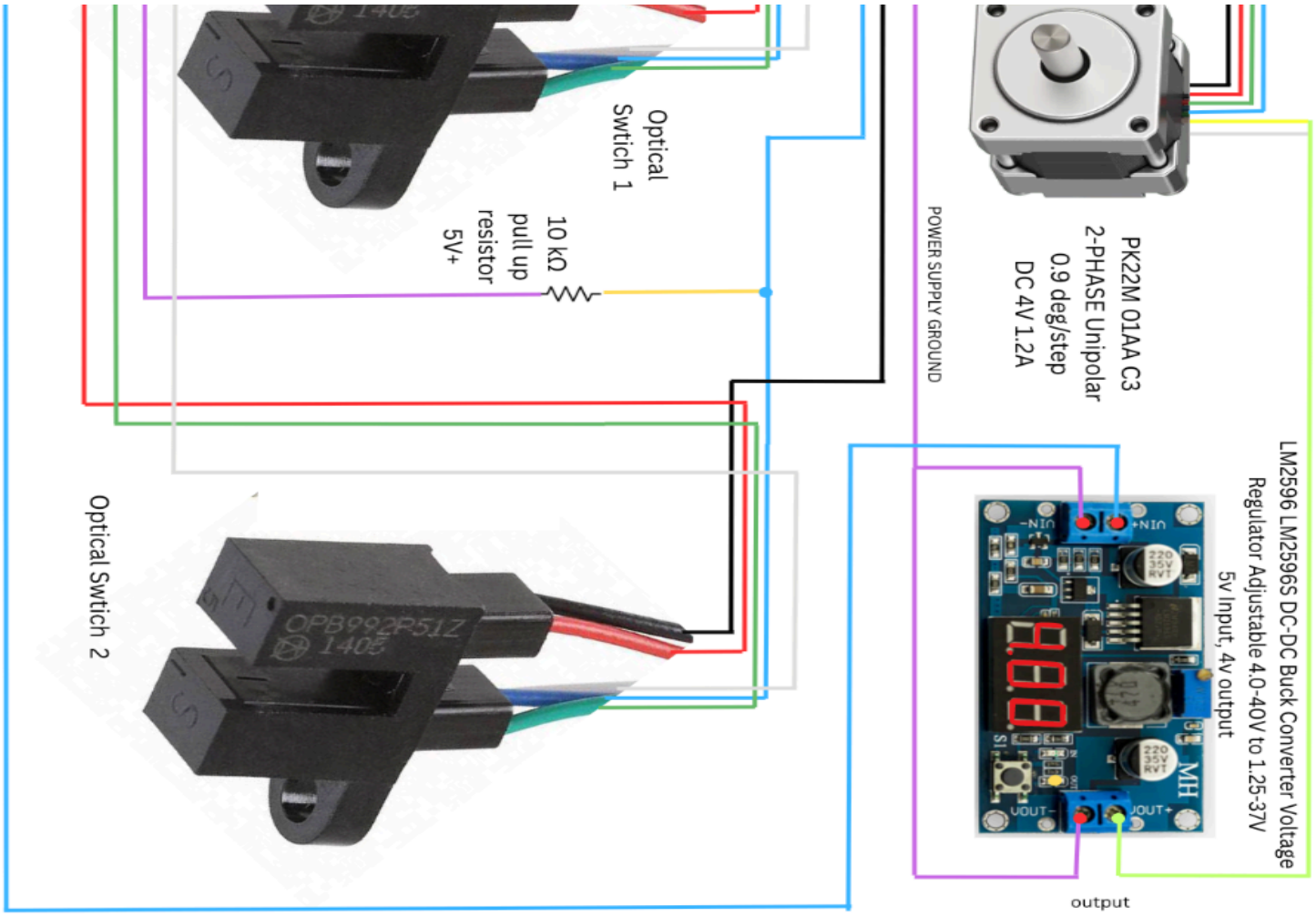
-> The grating is 1200 gr/mm

The original manual of the VM-504 can found here:

https://drive.google.com/file/d/1NKxPsK79chot_I3PJJGTL_YXj-pggTi1/view?usp=sharing


SENSOR1
Small disk with slit on stepper motor shaft
Screws the disk uses to attaches to gray shaft collar, which tightens onto the stepper motor shaft
Worm gear couples to the grating mount
Large disk with slit on worm gear
SENSOR2


ULN 2003
stepper driver
VCC optional for LED power
ACTON RESEARCH MANUAL
Motor PIN DIAGRAM:
PIN 1 BLACK A1
PIN 2 RED A2
PIN 3 GREEN B1
PIN 4 BLUE B2
PIN 5 YELLOW OPEN
PIN 6 WHITE SHEILD
5&6 Get 4V power
IN1
IN2
IN3
IN4
IN5
IN6
IN7
D1
D2
D3
D4
R1
R2
R3
R4
P3
opticaltest.py
on
standard firmata
OPB992L51 MANUAL
IR EMITTER
RED cathode 5V+ 140 Ω (in series)
BLACK anode
Sensor
Green GRND
WHITE VCC 5V+
Blue Output Logic
10 kΩ
pull up
resistor
5V+
140Ω
resistor
Power Supply
5v
140Ω
resistor
Note: Green Ground, White 5v VCC, 10kΩ pull up resistors 5v, Red 5v Power are all spliced together for both switches. Only Black Ground and Blue Digital output seperate. *Be sure to share ground between power supply and arduino*



WIRING DIAGRAM AND COMPONENTS

The wiring diagram can be hard to follow, but can be best understood when broken down evaluated on the power and data connection requirements of each component.

1) Stepper Motor PK22M 01AA C3 2-Phase unipolar 0.9 deg/step DV 4V 1.2A.
    a) Internally has four Coils, each coil corresponds to a wire.
    b) Each coil wire from the motor is wired to the ULN2003 stepper driver
        i) BLACK A1 coil, A on stepper driver output
        ii) RED A2 coil, B on stepper driver output
        iii) GREEN B1 coil, C on stepper driver output
        iv) BLUE B2 coil, D on stepper driver output
    c) Power
        i) Yellow and White are coupled together to the output (Vout+) terminal of buck converter set to 4V

2) Stepper Driver UNL2003
    a) A, B, C, D on the output correspond to the black, red, green, blue motor wires.
    b) IN1, IN2, IN3, IN4 are input from the arduino digital pins 11, 10, 9, 8.
    c) The GND pin is connected to a ground wire that goes to the **GND pin on the arduino.**
        i) That wire is also connected to the **(-) terminal** of the 5V power supply.
        ii) Ground between the power supply and the GND of the arduino must be connected.
3) LM2596 DC-DC Buck converter

    Since the stepper motor operates at 4V and the sensors are 5V, we use a buck converter to step down from 5V to 4V for the motor's power only. There is a small flathead screw on top that can be turned slowly to set the output voltage.

    a) The Vin- and Vout -, both get connected to the 5V power supply's (-) terminal, this wire is the same wire the stepper driver's GND and also connects to the arduino GND.
        i) It is important that the converter is connected to both Arduino GND AND the 5V power supply's barrel jack's negative terminal.
    b) **VIN+** comes directly from the 5V power supply's barrel jack's (+) terminal. DO NOT use the arduino's 5V out for the VIN+.
    c) Vout+ gets wired to the yellow and white stepper motor wires together.
    d)
4) OPB992L51 IR Emitter Sensors (2)

    These are the optical sensors on which detect the disk slit position as either blocked or open. They shoot an infrared beam that when broken reads blocked, and when it goes through to the other side of the sensor, it is read as open, i.e when the slit of the respective disk is in the location of the sensor.
    Each sensor has five wires:

    a) RED cathode 5V+ 140 Ω (in series). This provides power to emit the ir beam, each sensor's wire has a 140 ohm resistor in series before both sensors' red wire connects in parallel to the Arduino 5V power pin.
    b) BLACK anode. Each sensor's black anode wire must be connected to the Arduino's GND, you can couple these together or simply also plug both separately in whichever Arduino ground pin.

    The sensor portion of the OPB992L51 has three wires;

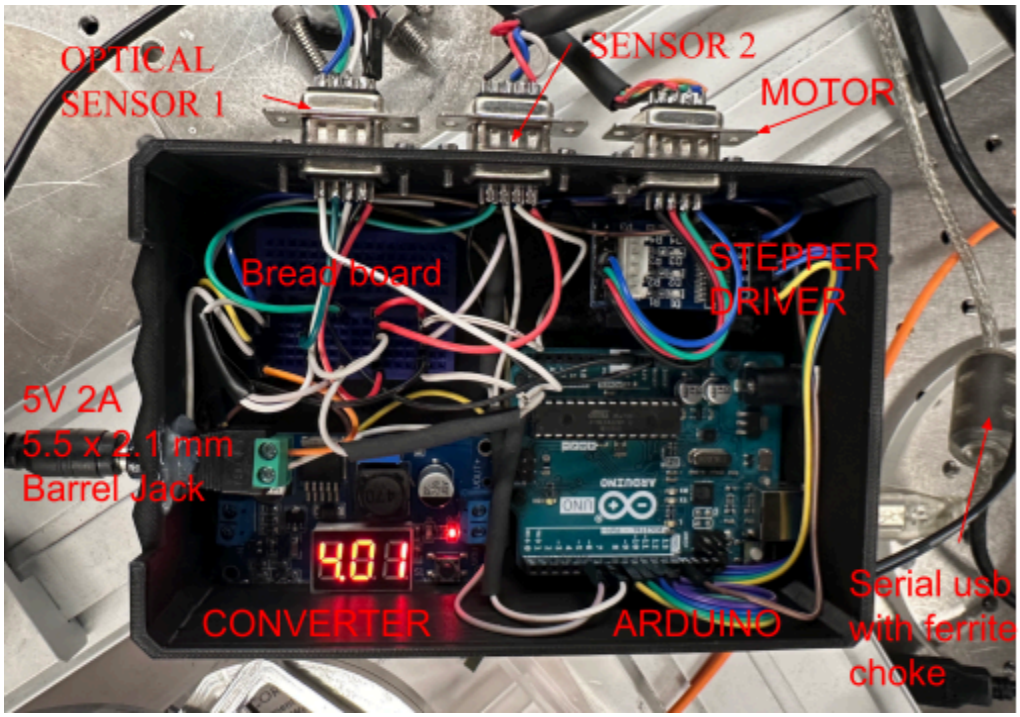
    c) Green GRND, both sensors' green ground wires can be combined in parallel and should be connected to the (-) negative terminal of the buck 5V power.
    d) WHITE VCC 5V+ both sensors' white wires can be combined in parallel and should be connected to the (+) positive terminal of the buck 5V power.

e) Blue Output Logic. Each Sensor gets its own blue output logic wire that plugs into separate Arduino digital pins.
   i) SENSOR1 at Digital Pin 7, SENSOR2 at Digital Pin 5. (you can swap these for different pins on the arduino, as long as the correct pin configuration is detailed in the script, controller.py
   ii) Each Blue logic wire of both respective sensors is connected in parallel with a wire that has a **10KΩ** resistor in series.
      (1) Both sensors' wires that have the 10Kohm resistor are combined in parallel and connect to the (+) positive terminal of the buck 5V power.
      (2) Each of the respective blue wires go to their respective digital pins on the arduino. (sensor1 pin7, sensor2 pin5).

An enlarged and not sideways wiring diagram can viewed here:
https://drive.google.com/file/d/1kn51ruv-ujxeURqIXM2b6oq9Y5f3awLR/view?usp=sharing

The controller unit itself is a 3d printed case that uses serial port wiring for the optical sensors and the stepper motor wiring. (It may have been switched to GX12 aviator pins for more secure wiring by the time you are reading this).

>try to use a serial usb2.0 that has a ferrite choke wire to connect to the arduino.
>use a 5V 2A barrel jack 5.5x2.1 mm to power the motor and sensors.
>if power and ground is properly provided, you will see the buck converter display 4.00 as the set output voltage on a small display. If power is plugged in and ground all properly connected, this converter should have its red display lit up.



Code Overview:

controller.py is the script that runs the monochromator controller. Ultimately what the script does is keep persistent track of the steps, gets information from both optical sensors and relates that into positional information and allows the user to rotate the grating via the stepper motor using various different methods. The most important feature is that the user can input a wavelength and order, and the stepper motor will rotate the grating so that wavelength is what comes out of the exit slit.

The code can be found at:
https://github.com/afham-b/monochromator-control.git

Using the Code:

1. Install atleast python 3.10
2. Python Dependencies
   ```
   > pip install pyserial pyfirmata numpy matplotlib scipy
   ```
3. Arduino IDE + File load
   a. https://www.arduino.cc/en/software
   b. Pip install pyFirmata in your working Directory
      i. Once package is installed, in terminal: pip show pyFirmata
      ii. Go to location of package → open pyfirmata.py
         1. ~line 185: Replace inspect.getargspec with inspect.getfullargspec and SAVE
   c. Setup: Plug board into computer
      i. First time plug in may prompt installation requests and admin requests, approve all
      ii. Then in IDE go to File->Examples->Firmata->StandardFirmata
         1. This will open new arduino ide windows, once code loads, you want to click on arrow (upload) top right

      Note: Arduino Steps shouldn't have to be repeated unless the board is reset.
   d. In the Arduino IDE note down which port the board is detected at using the select port drop down menu at the top left of the IDE
      i. If you get connection errors, or Arduino initialization errors, then you can try in terminal, `setx ARDUINO_PORT your_port_here` , then you try rerunning controller.py
4. Running the script **python controller.py** will start up the script to the main menu.

What the code actually does:

- It controls the steppermotor which turns the worm gear and therefore the grating.
- It collects open or closed information from the s1 s2 optical sensors
- Its has a Disk home reference (position where both slits are in the open state and therefore the wavelength is at zero-order position, i.e. the input slit is the same as the exit slit

- Has an optical home that a user defineable (where the user visually verifies and defines what position that that zero-order is actually achieved)
- It has persistence state storage, it keeps track of all steps and the last position of the motor when the script was last exited.
- It allows for backlash compensation and directional bias if any are found during calibration.
- It allows for a wavelength calibration model to be fitted based upon user inputted reference points, using a reference input light and its spectra (for example using a helium gas lamp, and using its known emission spectra to create reference points)
- Using the created model, it allows for users to search for a wavelength and order.
- It can also explicitly be commanded to go to specific angles instead of just wavelengths and orders.
- It can also be explicitly to go to a specific step number using a GOTO menu.
- It can also be jogged in both directions.
- Rehomed back to either disk home (both slits open) or to user defined optical home

Reference Definitions

2.1 Disk Homes

Mechanical home defined by:

- S1 = OPEN
- S2 = OPEN

At disk home:

```
step_count = 0
```

This is purely mechanical indexing. In theory zero order should be here as well, but due to mechanical slack and changes while taking things off and putting them back on, usually zero order (what is referred to as optical home) is typically -/+ 10 steps around from the disk home.

>>It is important to note that disk home can physically be changed:

1. Go to the current disk home, ensuring at least S2 says open
2. Loosen grey shaft collar is loosen from the stepper motor's shaft using an allen key.
3. Run the controller.py script , slowly jog the motor until zero order comes in sight from the exit slit while the collar is loosened and not rotating with the shaft (and therefore the s1 disk is decoupled from the shaft rotation). You can physically hold the collar in place with your fingers to keep it from rotating with the shaft.
4. Now that zero order is achieved, slowly and gently rotate the loose grey collar until the disk on the back of it goes into s1's open position.
    a. Monitor the status of s1 and s2 and try to get both to say OPEN while moving around the shaft collar, WITHOUT getting the shaft to rotate or move (this would

start moving the worm gear and will cause zero order to go out of the exit slit's view).

b. Once you have gotten both sensors to say open and zero order is still visually in the view of the exit slit, you can loosen the allen screw and tighten the shaft collar in its current position.

2.2 Optical Home

Optical zero defined by:

> Zero-order emission centered in the exit slit.

Because disk home ≠ optical zero in practice, we define:

```
OPTICAL_HOME_OFFSET_STEPS
```

So:

```
Optical Home Step = Disk Home Step + OPTICAL_HOME_OFFSET_STEPS
```

All wavelength math is referenced to **optical home**, not disk home, since optical home is truly where zero-order is.

You can set optical home, from the main menu > Home menu > 3. Set optical home

> This will make the system go to disk home (s1 and s2 open), from there it will start jog mode, allowing you to slowly jog in either CW or CCW until you see zero-order in the exit slit, and once you have achieved that, you can hit s to save this optical home.

---

3. Coordinate System

Steps are +N Forward CW, -N Reverse CCW

Quantity	Reference
step_count	Disk Home
θ (angle)	Optical Home
λ (wavelength)	Optical Model

---

4. Motion Architecture

Absolute Step Targeting

Final motor target is always computed as:

```
target_steps =
    OPTICAL_HOME_OFFSET_STEPS
    + angle_to_steps(theta)
```

This ensures:

- Zero order lands at optical home, All wavelengths are offset-corrected

---

Final Approach Strategy

To improve repeatability and reduce backlash effects:

```
FINAL_APPROACH_DIR = +1   # CW
FINAL_APPROACH_BACKOFF = 80 steps
```

Motion algorithm:

1. Move near target
2. Back off
3. Re-approach from fixed direction

This ensures consistent gear loading.

Homing Approach Strategy

If you have used the monochromator you'll have seen the window that the s2 sensor is open is more than a few steps, and s1 as well can sometimes be 1-2 steps of being open. Because the disk provides positional information in these ranges, we have to be more specific to get home.

>Going home works by first seeing what step we are at and start jogging in the opposite magnitude of the step count until S2 sensor detects it first open

>Then from first opening the motor keeps going until it detects the other edge of the slit where the sensor will go from open-> close.

>It constructs a window between the first close-> open and the last open->close step count, and then goes to the midway point of that window by first backing off ClockWise and then approaching that midway point.

---

5. Wavelength Model

5.1 Physical Equation

General grating equation:

`mλ = d (sin α ± sin β)`

Simplified symmetric case:

`mλ = 2d sin θ`

5.2 Fitted Model

We fit:

`λ/m ≈ a + b · (2d sin θ)`

Where:

- `a` → systematic offset (zero-order shift)
- `b` → effective groove spacing scaling

This is the model used to find specific angles based on wavelengths and order the user inputs. It calculates the angle using m for order and `λ` wavelength and a and b are coefficients calculated based on fitting from calibration. d is the grating constant, which for our grating is 1200 gr/mm. How to fit the model based on reference points is discussed in the calibration section.

5.3 Wavelength Model is Stored in:

`state/wavelength_cal.json`

Structure:

```
{
  "refs": [
    {"theta_deg": ..., "lambda_nm": ..., "order_m": ...}
```

```
  ],
  "model": {
    "a": ...,
    "b": ...
  }
}
```

It loads up on start-up everytime controller.py runs and loads the json data to be referenced. See the persistence section to familiarize yourself with the functions that store, and retrieve these values,

---

Section 6: Calibration Procedures and Operational Guidelines

6.1 Calibration Overview

Calibration of this system consists of five independent layers. Each layer corrects a different physical or mechanical uncertainty in the instrument:

- 1. S1 Motor Integrity Calibration (Stepper validation)
  -> tells how many steps/revolution of the small disk.
- 2. S2 Worm Gear Angular Calibration (Angle-to-step mapping)
  -> from here we get steps/degree
- 3. Optical Home Calibration (True zero-order alignment)
  -> know where true zero order is relative to disk home
- 4. Backlash and Directional Bias Compensation
  -> tells us if we have to compensate for extra or less steps depending on direction change
- 5. Wavelength Model Fitting (Spectral calibration)

  -> uses reference points to solve for the coefficients in our littrow model.

Calibration MUST be performed in the above order. Performing wavelength fitting before mechanical calibration will result in persistent systematic errors.

6.2 S1 Motor Integrity Calibration

Purpose: Verify the number of motor steps per revolution and confirm no steps are being lost.

Procedure:

1. Start the controller and enter the calibration menu.
2. Select S1 calibration (1)
3. Allow the motor to rotate slowly until two consecutive S1 OPEN detections occur.
4. The system records total step count between events.
5. Value is saved to state/calibration.json.

Run this calibration after any motor replacement, wiring modification, or if step drift is suspected.

6.3 S2 Worm Gear Angular Calibration

Purpose: Determine true motor steps per worm gear revolution and compute STEPS_PER_DEG.

Procedure:

1. Select S2 calibration from the menu. (2)
2. System detects first S2 OPEN edge.
3. The motor rotates full revolution until the next OPEN detection.
4. Total steps are recorded as S2_STEPS_PER_REV.
5. STEPS_PER_DEG is computed and saved.

This calibration anchors the angular coordinate system. Re-run after mechanical disassembly or if wavelength errors grow linearly. Note that this calibration takes a long time to complete, and expects a value of around 18000 steps revolution of the large worm gear.

6.4 Optical Home Calibration

Purpose: Align mechanical disk home to true optical zero-order emission.

Procedure:

1. Navigate to Home Menu → Set Optical Home.
2. System moves to Disk Home (S1 OPEN, S2 OPEN).
3. Enter jog mode.
4. Slowly jog until zero-order light is centered in the exit slit.
5. Press 's' to save offset.

This writes OPTICAL_HOME_OFFSET_STEPS to state/optical_home_offset.txt. All wavelength calculations reference this offset.

6.5 Backlash and Directional Compensation

Purpose: Compensate for mechanical slack in the worm gear.

The system enforces a fixed final approach direction (CW) with a programmed backoff distance to ensure consistent gear loading.

1. Optional Measurement Procedure, in the calibration menu, pick 3
2. Approach a known spectral line from CW direction and record step position.
3. Approach the same line from CCW direction and record step position.
4. Measure difference and store it as BACKLASH_STEPS.

Always approach spectral reference points from the same direction during wavelength calibration.

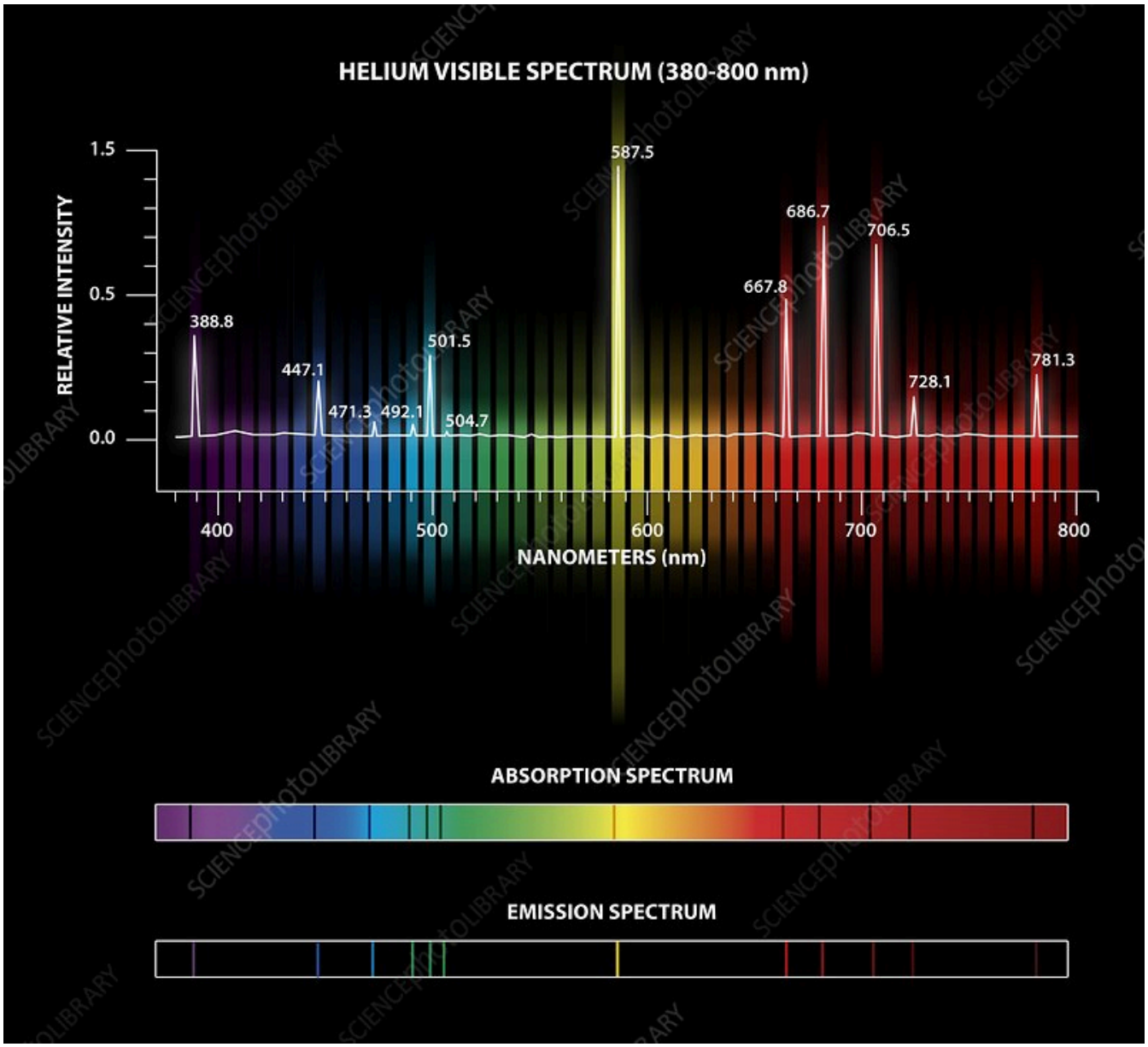
6.6 Wavelength Model Calibration

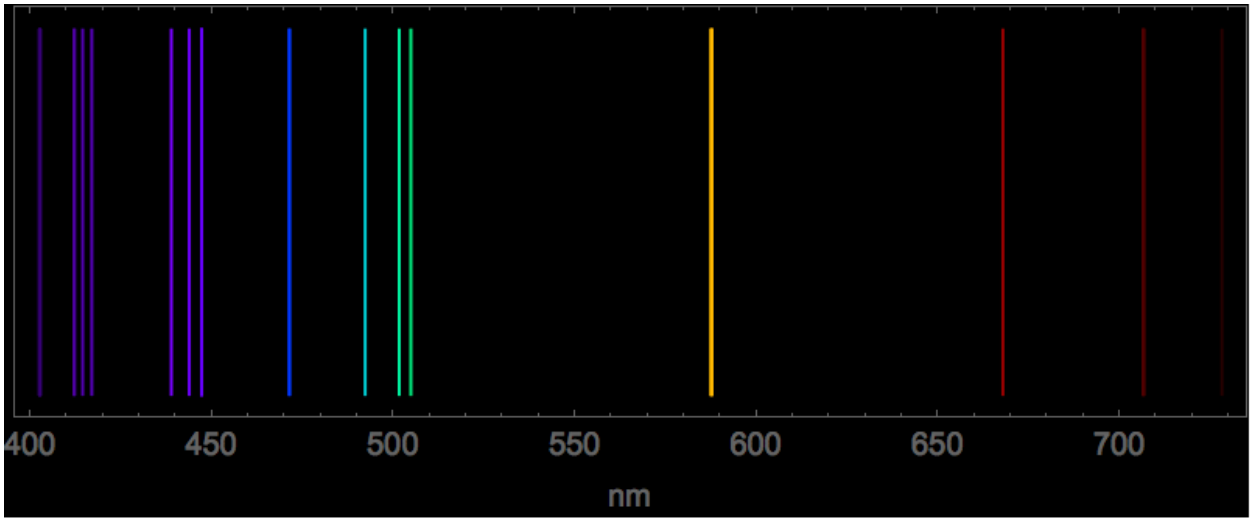
Purpose: Fit mathematical relationship between grating angle and wavelength.

Default Model: $\lambda/m = a + b \cdot (2d \sin \theta)$

Procedure:

1. Use known emission sources (e.g., Helium lamp) aligned at the entrance slit.
2. Enter the position menu from the main menu
3. Select Wavelength calibration (refs & fit) by selecting 5
4. Enter Jog Mode
5. Move to different visible spectral emission lines using jog.
6. Record lambda_nm, and order_m by identifying the emission line from a reference chart, for example, Helium gas lamp was used to fit this model, and hence these charts were references to match a wavelength nm to the spectral line being observed from the exit slit(i.e i see a intense orangish emission so i mark it down as 587.5)





7. Exit jog without moving anything and you should be back at the wavelength calibration menu
8. Use (1) to add the current point you are at (you should still be at the emission line you wanted to mark) as a reference point and input the matched wavelength and order. (for the model I fitted using order m=1 reference points).
9. Collect 4–6 reference points across a range by repeating steps 4-8.
    a. **Do not add all known emission lines** as reference points, as we will use some known points as test points to see the accuracy of our model after fitting.
10. Once you have good points, Fit model and save to state/wavelength_cal.json using Fit and save model (a,b) (3) in the wavelength calibration menu.

>You can view the current model and saved refs using **4) Show current references and model i**n the wavelength calibration menu. This can be useful if checking for bad reference points being put in, or seeing if the saved model was actually correctly loaded. These are persisted by being saved to a json file that is loaded everytime the script is restarted.

>If NO MODEL is detected the code defaults to a=0, and b=1

>When adding reference points note that you are adding in addition to the previously saved reference points. Excessively adding references could result in overfitting the model to reference light sources. You always clear the model and start adding reference points from scratch.

>IT IS BEST PRACTICE TO GO TO REFERENCE POINTS in one direction, do not excessive switch directions scanning for emission lines as this allows backlash to bleed into the model. It is recommended going in one direction continuously until you get to an emission line, add it as a reference point, and then either go forward or return to home.

6.7 Calibration Verification Test

After completing calibration, perform the following repeatability test

1) Go to wavelength λ1.
2) Go to wavelength λ2.
3) Return to wavelength λ1.
4) Verify deviation is ≤ 5 steps.

If deviation exceeds tolerance, inspect backlash settings, sensor stability, or potential step loss.

If the deviation is sufficient

1. input a known emission line that you didn't have as a reference and can see if the fitted model accurately goes to said point.

>for extra accuracy you can test the model with another emission source with known spectra to see if the fitted model was not overfitted to your reference spectra.

Further Discussion of Calibration and Model Method

The accuracy of the monochromator is only as good as the model it has fitted. Since the model is made using user input and response, any inconsistency will extraneously affect the model. Factors such as lining up the emission line in the middle of the exit slit, any physical shifts or changes in angles of the light source at the entrance slit, changes in the position of the user observing at the exit slit can introduce imprecision into the model. Other factors such as miscalibration of other components, slack in the gears, backlash and lost steps can also cause an incorrect step count to be attributed to be an inputted reference wavelength, which will lead to a deviation in the model.

One possible method to increase precision while fitting the model is to:

1) use a fiber optic at the entrance slit from the light source so as to keep the position and angle of the incoming light source consistent at the entrance slit
2) Use a mounted camera with a lens that focuses on the emission spectra at the exit slit, this way the position and method the emission lines are recorded is more consistent and can lead to a better model

> This implementation is in progress and the documentation will be updated.

Another possibility is changing the model and seeing which would more accurate given the fitted reference points,

Instead of: $\lambda/m = a + b\cdot(2d \sin \theta)$

Fit full: $m\lambda = d\ (\sin \alpha + \sin \beta)$

> this also being tested and will the documentation will be updated with the results

7. Persistence System

If in the case the model is not being loaded correctly, or the saved calibrated settings are unretrievable, they can be directly accessed and modified in these respective files:

7.1 Step Count

Stored in:

`state/step_count.txt`

Updated every move.

7.2 Optical Offset

Stored in:

`state/optical_home_offset.txt`

7.3 Calibration Store

Handled via:

`cal_store.py`

Functions:

- `load_cal()`
- `save_cal()`
- `apply_cal_to_globals()`
- `load_wl_cal()`
- `save_wl_cal()`

---

8. Modes and Menu functions

Upon startup, the following occurs automatically:

1. Arduino is detected and initialized
2. Optical sensor states are read
3. Persistent step count is restored (if present)
4. Optical home offset is restored
5. Calibration parameters are loaded
6. Logging is initialized

Main Menu
After initialization, the following main menu options are presented:

- G — GoTo Step
- J — Jog Mode
- V — Speed Settings
- H — Home Menu
- C — Calibration Menu
- P — Position Menu (Angle / Wavelength)
- S — Scan Menu
- Q — Quit Controller

GoTo Step (G)
Moves the motor to an explicit absolute step count referenced to Disk Home. Positive steps move clockwise (CW), negative steps move counter-clockwise (CCW).

Jog Mode (J)
Jog Mode allows manual incremental motion of the grating using keyboard input. It provides real-time feedback of sensor states and updates the internal step count.

Speed Settings (V)
Allows adjustment of the stepper motor. Slower speeds improve repeatability during calibration, while faster speeds reduce repositioning time. Recommended to leave these settings as is to avoid setting a speed that would increase rate of step loss or worsen backlash. If a speed change introduces inaccuracy into the positioning system, it is recommended to recalibrate at the new speed.

Home Menu (H)
The Home Menu controls mechanical and optical reference positioning.

- Go to Disk Home — Re-indexes using both optical sensors.
- Go to Optical Home — Applies the stored optical offset.

- Set Optical Home — Allows user-defined zero-order alignment.

Calibration Menu (C)
Provides access to all calibration routines including motor integrity, worm gear calibration, backlash compensation, and wavelength model fitting. See calibration section above for recommended procedure.

Position Menu (P)
Primary scientific operation menu for positioning the monochromator.

- Move to Wavelength — Computes grating angle from wavelength and order.
- Move to Angle — Direct angle-based positioning.
- Motion Mode Selection — From Home or Shortest Path, or Backstep

>Note that from Home rehome the monochomator first before going to the inputted position. Shortest path directly goes to the position from whichever position the user is currently at. Backstep (still being implemented) steps back from the current point ~80 steps and then goes to the inputted position in order to approach it from a consistent direction (i.e always going to the final position CW)

Scan Mode (S)

Will allow the user to scan between two inputted positions.

- Move from Wavelength λ_1 to λ_2— Will allow user to input starting and ending wavelength
- Move to Angle θ_1 to θ_2 — Will allow user to input starting grating angle and ending grating angle

The system allows users to set the following parameters for scanning:

1. Approach from Home Or Shortest Path?
    a. Goes to the λ_1 or θ_1 by either first going home and then to initial position or immediately to initial position from where the system currently is
    b. Enter in either H for Home or S for shortest Path
2. Repeats
    a. Allows multiple scans between inputted positions, leaving it blank by hitting enter will default to only 1 scan.
    b. Enter in a positive integer
3. PingPong
    a. If doing multiple scans, you can choose to restart the scan from the initial position or inverse the directions for the scan.
    b. Enter Yes or No using Y/N
4. Rehome per repeat
    a. Gives options to rehome before every single repeat for increased precision.
    b. Not recommended to use in addition with PingPong mode unless needed.
    c. Enter Yes or No using Y/N

5. Dwell Time at each point
    a. This lets you set time for the motor to pause at its steps for a specific amount of time if you need to collect more data at points.
    b. If you set dwell time to zero or hit enter (defaulting to zero), the scan effectively becomes a continuous scan between the initial and final position instead of pausing every every single step
    c. Enter non-negative value in seconds.
6. Wavelength or Angle
    a. Will ask you to enter in λ_1 to λ_2 in nm and enter in m, order
    b. Or will ask you to enter in θ_1 to θ_2 for degrees for grating angle and m, order
7. Step size
    a. Input positive value in nm, the recommended default is 1.0 nm
    b. For a a fine scan set a positive dwell time and use 0.1-0.25nm
    c. Coarse scan: 2-5nm
    d. For angles, enter positive value in degrees, recommended default is 0.05 °
    e. For finer scan, set positive dwell time and use 0.01-0.02°, and for a larger scan you use 0.1-0.2°
8. Set Marginal offset
    a. Allows users to input offsets to the starting and end points of the scan.
    b. For example add in a 10 nm buffer to the inputted endpoints in case you are not confident in the current model the system runs on.
    c. The default margin is zero.

Quit Controller (Q)
Safely exits the controller, saves persistent state, disables motor outputs, and closes the Arduino connection. Always exit using this option.

Recommended Use Flow

1. Power system
2. Start controller script
3. Go to Optical home
4. Move to wavelength or other functions
5. Perform experiment
6. Quit controller

7. Power off controller box.

---

9. Error Sources & Mitigation

This section lists known failure modes observed during operation of the Monochromator Controller System and provides mitigation steps.

Safety and Handling Notes

Before troubleshooting, ensure power is OFF when inspecting wiring. Do not force mechanical motion by hand. Avoid touching optical surfaces.

No Light at Exit Slit (Entrance Slit Confirmed Open)

- Symptoms:
    - No measurable light or visible emission at the exit slit.
    - The entrance slit appears open and aligned.
- Likely Causes:
    - The input shutter/lever near the entrance slit is in the BLOCK position.
    - Monochromator is positioned far from expected wavelength/zero-order.
    - Light source alignment into the entrance slit is incorrect.
- Mitigation:
    - Verify the lever near the entrance slit is flipped to UNBLOCK incoming light.
    - Run Home → Go to Optical Home, then attempt a known visible wavelength.
    - If still no light, confirm the light source is properly aligned into the entrance slit.

Positioning Error / Steps ‘Off’ From Expected Output

- Symptoms:
    - Commanded wavelength does not match observed wavelength.
    - Zero-order not found near Optical Home.
- Likely Causes:
    - Controller state step_count is incorrect due to prior improper shutdown.
    - Mechanical slip or missed steps occurred.
    - Optical home offset changed after disassembly.
- Mitigation:
    - Rehome using Home Menu (Disk Home then Optical Home).
    - Restart the script to reload state cleanly.
    - Inspect persisted step_count for illogical values and reset by re-homing if needed.
    - Re-run calibration (S1, S2, Optical Home, then wavelength model).

Optical Sensors Stuck OPEN or Stuck BLOCKED

- Symptoms:
  - S1 or S2 always reads OPEN regardless of motion.
  - S1 or S2 always reads BLOCKED regardless of motion.
- Likely Causes:
  - Loose sensor connector or broken wire (especially output logic wire).
  - Missing common ground between Arduino and sensor power supply.
  - Incorrect pull-up resistor wiring on sensor output.
- Mitigation:
  - Power OFF. Check wiring continuity and connector seating.
  - Confirm Arduino GND and external 5V supply negative are tied together and the sensors are correctly grounded.
  - Confirm pull-up resistor to +5V is present on each sensor output line.
  - If only one sensor fails, swap pins (in software) to isolate wiring vs sensor failure.

Wavelength Model ‘Far Gone’ / Large Systematic Error

- Symptoms:
  - Multiple wavelengths are incorrect even after homing.
  - The model appears to move in the wrong direction or overshoots heavily.
- Likely Causes:
  - Bad wavelength reference point entered (wrong λ, wrong order, wrong θ).
  - Reference set built from mixed approach directions (CW vs CCW).
  - Optical home offset changed but model was not refit.
- Mitigation:
  - Inspect current reference points and remove any suspicious entry.
  - If unsure, clear all reference points and rebuild from scratch.
  - Load backup/fallback mode if supported; otherwise refit with fresh points.
  - Always collect reference points using the same final approach direction.

Significant Step Loss / Revolution Mismatch (S1)

- Symptoms:
  - After ~100 steps, S1 does not transition OPEN→BLOCKED→OPEN.
  - S1 revolution counts drift over time.
- Likely Causes:
  - Mechanical slack in couplers or shaft collar.

    - Loose set screws on shaft collar or disk mount.
    - Insufficient motor torque (voltage drop, current limit).

- Mitigation:

    - Inspect and tighten all relevant screws (shaft collar, disk mounting screws).
    - Check motor supply voltage at load; verify buck converter still outputs ~4.0V.
    - Reduce speed (increase step delay) to prevent missed steps.

Jog Mode Unresponsive / No Keyboard Response

- Symptoms:

    - Jog mode enters but arrow keys do not move the motor.
    - Terminal appears stuck or ignores keypresses.

- Likely Causes:

    - Terminal input buffering / carriage return interaction.
    - Keyboard hook conflict (pynput/keyboard package differences).

- Mitigation:

    - Attempt to quit jog mode if possible and restart the controller.
    - If stuck, close the terminal window or kill the process, then restart.

Additional Failure Modes

- Arduino Port / Connection Failure

    - Symptom: Program exits at startup with Arduino initialization failure.
    - Mitigation: Set ARDUINO_PORT environment variable or confirm correct port.

Documentation will be updated as the calibration method and scan mode are implemented and errors and their solutions will be updated as observed and resolved.

Need to reach out for questions?

Contact ab5509@columbia.edu