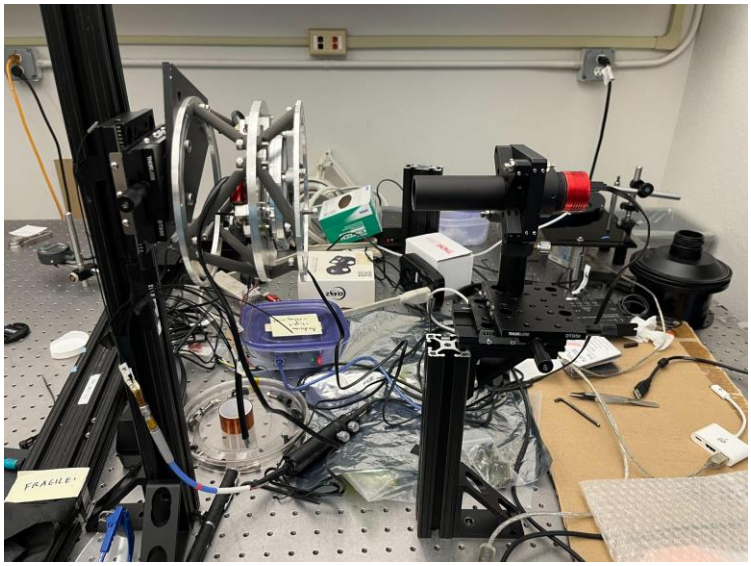
CHaS Flexure Compensation System

we may come up with a fancy name at some point

August 2024



Preliminary explanation: hardware and software

Our method of handling flexure requires multiple components in order to work. We've outlined the installation requirements and are working on a concise explanation of the scripts.

The laptop will have everything installed, so you should be able to run the relevant files right away. We can also run tests on our own end by connecting to the observatory laptop through Parsec.

How Does Everything Work?

The Flexure compensation system is made up of three components:

1) An Arduino Board
2) and 8742 controller board connected with(up to) 4 piezoelectric tiny picomotors
3) guide camera(s)

Flexure is the process that causes consistent and (at least from our graphs) slow building offsets as the telescope slews during an exposure. To counter these slow building offsets, we use these picomotors that make generally slow and tiny corrections to keep the optics generally in the same positions as they were at the beginning of an exposure, meaning we can get longer exposure times.

Part of the issue is knowing how much to move the picomotors by. The best-case scenario is the motors move exactly counter to the offset of flexure: finding this amount of offset is essentially the job of the guide camera.

A telescope guide camera, with some guiding software, finds offset off of some selected guide star. We can deceive the guiding software (metaguide) and make it believe that a small light source is a star and it should track the light source as it moves. The code creates some initial point at the start of an exposure, then tracks delta offsets in pixels for the duration of that exposure. We have pixels offsets in the x and y direction, and so we can ask the motors to move counter to offsets and to move until they have minimized those offsets within in some margin of error (currently around 8 microns or 2 pixels)

Using some optics math and the camera spec sheet, we convert those pixels to some useful distances, in our case: microns. The issue is that the company that makes these picomotors (Newport) are money machines and absolutely genius, and thus the picomotors don't understand distances at all. We do some conversion, slap on some variable gain factor, and then spit out a number of steps for the motors to take. Steps(-/+) are the only command these motors understand.

Picomotors without feedback are inconsistent and have no useful precision. So the camera isn't just for measuring the offsets, but also provides a closed feedback loop to the picomotors so they know where and how they are moving.

So what is the arduino for? Well we can't just have a light source inside CHas during an exposure, the arduino allows us control over the light source and turn it on and off with code, and the plan is to only have the light on intermittently during an exposure (i.e 1 second every 15 seconds). Secondly the arduino controls a relay switch that gives us control over the controller board that powers the motors, so we can remotely cut power to and restart the picomotors and controller board as we want (in the case of a connection issue).

TLDR: Arduino Controlled Light + Guide Camera + Metaguide software + custom sdk code = Tell picomotors which direction/axis and how many steps to go brrrrrrrr.

<u>What on Earth is this code?</u>

The code is centered around Metaguide. A free open-source guiding software (that unfortunately is windows only) and is much much faster than phd2.

The software is set up with the camera settings inside what is known as the scope setup .mg files. The main python scripts (mgpico.py) automatically opens up and boots up a .mg file that detects the light source as a star and starts outputting pixel coordinates in its GUI, but also onto a broadcast IP and a port.

Using some code found in MGListener.py we can locally listen to this IP and port (assuming the correct port, test loopback adapter, and the right broadcast ip settings were set during setup) and this lets us access these coordinates within python.

The **mgpico.py** script not only \starts the connection with picomotor controller board and then automatically starts metaguide, but also a simulated ascom telescope (metaguide behaves differently when not connected to a telescope, hence we connect it to a simulated one). Mgpico then opens these broadcast ports and **starts MgListener** to retrieve and pass metaguide tracking coordinates to the bulk of where the actual picomotor logic and instructions are, inside **PicomotorsStandAlone.py**, which works in this order:

1) looks at the first pair of coordinates and saves them to calculate deltas, and sets a reference position for the motors known as position “0”.
2) looks at deltas that are significant to act upon (for now outside of our m.o.e of 2 pixels)
3) puts them through some code that cleans against false deltas (camera artifact or metaguide glitch can cause drastic offsets to be measured)
4)Corrects for some physical angle difference between the camera’s sensor and the axis of the picomotors
5)Decides which velocity to move with (smaller deltas need slower speed to avoid overshooting)
6) Converts them into motor steps with the proper counter direction/axis + some gain factor
7) Move one motor and waits for it to finish, and the moves another motor (Newport Controllers are only capable of moving one motor at a time)
8) Checks if there are any motors stalling (attempts to reset to reference position if there is)
9) Repeat steps 2-8 until a delta is minimized into the margin of error.
10) Correction loops indefinitely until the program is killed at the end of an exposure.

Then MgPico.py at the end:
11) Return the motors to a default reference position “1” which is set during calibration.
12) Closes out metaguide, ascom, cleans the ports, shuts down the sockets/ threads. (threads lets us do multiple scripts at the same time, sockets allow data streams across and between multiple .py files)

The remaining code in the repo is for newport/ picomotor/ ascom/ zwo camera libraries, other dependencies, code for controller board connections, or control of arduino for light/ relay switch. Alot of the original code and libraries had to be modded to work in python. There is also code that handles ports/ sockets/threads/ active connections and closes them out properly. The only other significant code that the user will interact with is the calibration.py script that is run when the camera/ optics and flexure system is installed or physically adjusted.

Setup

Below is a Google Drive link with all the files for the repository, as well as a github link for the repo to pull from:

GDrive Link:
https://drive.google.com/drive/folders/1SbAhAOaKcVH_yqxCvdHsWdjvnDl4dgIo?usp=sharing

Git Link:
https://github.com/afham-b/Picomotor-Controls-.git

We understand that these steps are tedious, we will eliminate a significant portion of them in future versions as we hope to eventually make the gui version of the flexure compensation program. Thank you for your patience and diligence.

Required software and drivers (including dependencies)

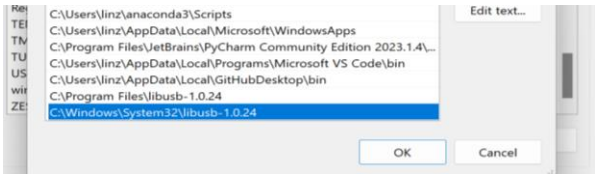
1. ASCOM Drivers (May require .NET 3.5 SP1 if not already installed)
    a. Link: https://github.com/ASCOMInitiative/ASCOMPlatform/releases/download/v6.6SP2Release/AscomPlatform662.4294.NewCertificate.exe
2. ZWO drivers
    a. Camera Driver: https://dl.zwoastro.com/software?app=AsiCameraDriver&platform=windows86&region=Overseas
    b. ZWO ASCOM Driver: https://dl.zwoastro.com/software?app=ASCOMDrive&platform=windows64&region=Overseas
    c. ZWO Direct show Driver: https://dl.zwoastro.com/software?app=DirectShowWDMDrive&platform=windows86&region=Overseas
3. Sharp Cap (recommended for camera setup, hardware installation, and light alignment process)
    a. https://www.sharpcap.co.uk/sharpcap/downloads
4. Python 3.11 (corresponding compiler if needed)
5. Python Dependencies
    a. Requirements.txt in the repository should list all the dependencies needed for this project. Make sure you have this text file downloaded from git or gdrive
    b. `pip install -r requirements.txt`
    c. Upon running mg_pico,py, you may still need to install other dependencies that were not in the requirements.txt. We will update requirements.txt
        i. Potentially: psutil, pyuac, matplotlib, pyusb
    d. MAKE SURE TO UNINSTALL PYWIN32 (and pywinauto)

6. Parsec for remote access
    a. To Allow for remote access over internet to any computer running the flexure compensation system, while windows remote works for local networks, it is far too significant of a security risk (malware/ransomware) to use the default windows OS remote software for remote internet access, and parsec is a free 3rd party software that securely accomplishes this.
    b. Download and Install: https://parsec.app/downloads
        i. It will require a parsec account to use
        ii. A default account has been created on the asus zenbook laptop using the following credentials:
7. Metaguide (just released ver 6.1.7)
    a. Requires Visual C++ **x86** Distribution install:
        i. https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170
    b. Meta Guide: https://smallstarspot.com/metaguide/ (scroll down for download link)
        i. Windows will flag it as an unknown publisher, run anyway.
        ii. We're using Metaguide because it's free and offers a very high frame rate
    c. Frank also runs a help forum: https://www.smallstarspot.com/index.php
8. Newport Picomotor Drivers + App
    a. https://www.newport.com/p/8742 (Under Resources & Downloads)
        i. Get: Picomotor App + Driver and TIVA Firmware
            1. Install the Picomotor App first (may give multiple installation prompts)
9. Arduino IDE + File load
    a. https://www.arduino.cc/en/software
    b. Pip install pyFirmata in your Chas Flexure Directory
        i. 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.
10. Custom DLL file copied into System32
    a. Pip install pylablib
    b. Go libusb website and download windows binaries zip folder:
        i. https://github.com/libusb/libusb/releases/download/v1.0.27/libusb-1.0.27.7z
        ii. Extract and go into MingGW32 and into dll folder, copy this libusb dll file from here and paste it into the Windows-> System32 Directory folder.

iii. If this fails (connection issues with picomotor controller persist), pip install libusb, pip show libusb, go to location, copy both x84&x64 dll from there.

11. Windows Test LoopBack Adapter Install and setup (Avoids the need to connect via a router)
    a. Open Device Manager
    b. Click on Network Adapters → Action menu → Add legacy hardware → Next
        i. Choose Install Hardware Manually (advanced) → Next → Choose Network Adapters → Manufacturer is Microsoft → scroll down to bottom of model section and select Microsoft KM-Test Loopback Adapter → Next → Finish
12. Prep Metaguide software and scope files
    a. Move scope setup files into the PC
        i. We're using fibertest1.mg for the microlens/lensets (previously used test1.mg for pre-commissioning run prep)
        ii. In the CHaS flexure directory you pulled from Git or downloaded from Gdrive, find the scope setup file, then copy all files inside the scope setup directory
            1. Paste in Users\Documents\MetaGuide
        iii. In the mg_pico python file in the chas flexure directory, edit the relay_scope_setup_path variable (around line 203) to the relaytest1.mg file from your Document\Metaguide directory address, and edit the address of lenslet_scope_setup_path as well to reflect your device's directories
            1. I.e the chas laptop's address is: C:\Users\schim\Documents\Metaguide\fibertest1.mg
    b. Setup the appropriate broadcast mask
        i. Open up cmd on the side and in terminal type ipconfig /all
        ii. Scroll to the network device that say TEST LoopBack Adapter
        iii. Copy KM-TEST loopback adapter's ipv4 address and its subnet Mask
        iv. Open up broadcast_mask_calculator.py in the chas flexure directory and past your copied ipv4 and subnet mask values into the variables
        v. Run python broadcast_mask_calculator.py and copy the calculated broadcast mask it outputs
        vi. From that Documents\MetaGuide directory, open up fibertest1.mg by double clicking, should launch metaguide automatically
        vii. Go to Setup Menu (bottom middle red button) -> in new window Click on the Extra tab, and paste the calculated broadcast mask into the broadcast mask field, click ok, then in the main MetaGuide window -> File-> Save
    c. Change Admin Settings
        i. if you're getting popups about permissions, open directory where MG is installed, right click on the .exe file and select compatibility tab, and make sure run as administrator is unchecked (or it was previously unchecked and you had errors, attempt checking it)

13. Ports
    a. Make sure you have the right COM for your device
        i. Set Arduino COM
            1. Open Device Manager and check (should be under Ports (COM & LPT)) → check the COM number, then within lighton.py AND lighttest.py AND mgpico.py,
                a. For lighton.py and lighttest.py, in main(), find arduino = ArduinoController('COM#', 8, 2)
                b. Change to specific COM (ex. CHaS laptop is COM3)
    b. Also in Device Manager, go to Universal serial Bus controllers → Generic USB Hub → Properties → Power Management → *uncheck "Allow the computer to turn this device off"* to ensure maximum (non interrupted) data stream
14. Libusb & new Path variables (unresolved libusb dll files and unresolved directories will results in "back-end" type errors or no connection to the 8742 controller board)
    a. DO NOT FORGET
    b. 

    c. Libusb is needed for newport communication, the libusb dll files are located inside of the git repo, and are linked below, you will need both the x84 and x64 dll files, as well as the entire libusb folder also copied into program files, before making individual path folders for those directories (make sure it's in Program Files AND Windows (System 32))
    d. https://stackoverflow.com/questions/13773132/pyusb-on-windows-no-backend-available
    e. Make copies of all the DLL files from libusb folder into to System 32 folder, and then copy the entire libusb folder into program files
    f. Add to environment variables: C:\Program Files\libusb-1.0._ver_
        i. I have 1.0.24, Asus as of 8/31/2024 has 1.0.27
        ii. Here are the laptop's environment variables:

C:\Users\schim\Desktop\Chas-Flexure-Compensation-master
C:\Users\schim\AppData\Local\Programs\Python\Python311\Lib\site-packages\libusb_platform_windows\x64
C:\Users\schim\AppData\Local\Programs\Python\Python311\Lib\site-packages\libusb_platform_windows\x86
C:\Windows\System32\libusb-1.0.dll
C:\Program Files\libusb-1.0.27

15. Eat cake

Hardware installation

When installing the picomotors, there are a few hardware steps to complete, the process is similar regardless of if it's the relay or the microlens array.

1) After installation of the parts into chas, you can plug in the camera to a computer running sharpcap and use it to **bring the light source into the center of the view** and in focus by adjusting the camera.
2) Using Arduino light (optional)
    a) If the lightsource isn't from a fiber connected to a lamp from the lab, it can be a fiber paired with the led light source connected to the arduino.
        i) If the arduino is plugged into the usb hub and connected to the the computer (and the correct com port is set, the ASUS's com # for the arduino is 3, but has already been set) then run **python lighton.py** in terminal and ctrl+c to turn light off.
3) Once that light source is aligned around the center of the camera's fov, switch from sharpcap and open metaguide manually by going to documents/metaguid and opening the .mg scope setup file that corresponds to what you are installing **(fiber1test.mg is currently for the microlens, and relaytest1.mg is for the relay**)
    a) If the sharpcap alignment was successfully you should see the light source in this metaguide windows as well, you can click on enhance, and then open VidProps (at the bottom in pink), **leave usb rate at 100%, try keeping exposure to around 1ms, attempt to keep gain as low as possible and same for gamma**, use these 3 setting to sharpen the image, increase the brightness of the lightsource in view, but avoid introducing noise via high gain. You want to **keep the size of the light small**
    b) If the lightsource size is too large in view, then you can lower the Star thres parameter in the Metaguide main Gui (near the bottom middle), the default is 50, keeping this number high is vital to avoiding detecting noise/camera artifacts as false light sources.
    c) If your settings are good, you will see a crosshair automatically pop up on your lightsource indicating that it is detecting and able to track a "star".
    d) Hit ok, then the top left under file, click save to save your settings.
4) **Connect the picomotor wires installed into the 8742 controller board**. Y axis should go into motor 1, X axis into motor 2. Look at the physical setup to determine the axis.
    a) Still Don't know which axis is which? Take a guess and plug in and then open up Metaguide on the connected computer with the lightsource on (using the lamp or python lighton.py command if using arduino Led) and then open New Focus Picomotor Newport Application on the connected computer.
    b) Click on motor 1, and switch to Relative mode, type in Relative: 5000 steps and hit the right arrow.
    c) Quickly switch to metaguide and see which or the coordinates are changing the most at the top right where the x and y field are.

i) If the y axis is plugged into motor 1 you'll see changes in the Y and possible minor shifts in X, and if the x axis is in motor 1 spot on the controller board, you'll see shifts in X dramatically more than in Y.
    (1) In the latter of these two cases, just switch the wires plugged into the controller board motor 1 port and motor 2 port.

5) Figure out **which axes may be inverted**
    a) Since the motors technically could be installed in any orientation, during initial installation it needs to be determined whether axises may be inverted.
        i) Open metaguide, light on (lamp or lighton.py), open new focus picomotor application.
        ii) Switch to motor 1, which should be y axis
            (1) Input a 5000 relative movement with the right arrow. This should move the light source in the positive y direction in the metaguide view, verify if it does, make a note if it doesn't
        iii) Switch to motor 2, which should be x axis
            (1) Input a 5000 relative movement with the right arrow. This should move the light source in the positive x direction in the metaguide view, verify if it does, make note if it doesn't
    b) For the axis where the direction wasn't a positive movement in their respective direction x/y, then you must comment out some code within in picomotorstandalon.py script:
        i) Go to around line 231/232 and comment out steps_x = steps_x * invert or steps_y = steps_y * invert depending on which motor DID NOT MOVE in the positive direction when given a positive input.
6) Determine Magnification
    a) If there is a lens on the camera, then some parameters will have to be modded inside the picomotorsstandalone.py to reflect the current effective pixel size.
        i) Two measurements are required for a lens, distance between lens and the sensor of the camera, and then distance between lens and light source. These measurements can be used to approximate the magnification.
    b) **If there is no lens on the camera, then in Picomotorstandalone.py comment out** :
        i) self.magnification = self.distance_lens_ccd /self.distance_lens_lightsource
        ii) Comment back in: #self.magnification = 1

7) Center the motors + arms
    a) Open New Focus Picomotor application, and using large relative steps (i.e 10000 or 5000, move the motors negative or positive, until the motors are centered.
    b) Check via visual inspection that the arms the motors move have good contact. Use allen keys to adjust motor positions on the relay or on the microlens to get a good contact point with the tip of the picomotor. DO NOT OVERTIGHTEN

A Centered motor will have the two screw regions noted above in equal length, use the new focus application to apply steps until both sections are roughly equal.

c) Repeat for all motors.

8) Calibration steps: **DO with TELESCOPE AT ZENITH!**
   a) If the above steps have been completed successfully, you can proceed with the last installation step, which is calibration.
   b) Run python calibration.py <enter_system_name here>
      i) If it's for the **relay**, input **python calibration.py nosecone**
         (1) We will fix the names later
      ii) For the **microlens**, input **python calibration.py microlens**
   c) Wait for calibration to complete, and see what the theta is in console output.
      i) Note: this is when the code sets up reference point “1”, at zenith
   d) Looking at the theta, physically rotate the guide camera counterclockwise or clockwise depending on the theta (counter for negative) a magnitude corresponding to the theta value it printed.
   e) Rerun part B, and then part D as needed until the theta is low as possible
      i) In the past it usually took around 3-4 tries to get near 0.15 theta, which is small enough.

Running correction

If all the steps above have been completed above and calibration is complete.

Then one last step remains:

1) Setting Correct Filenames
    a) Depending on which flexure you will be correcting (relay or microlens), comment out and comment in the filename lines
        i) In **Picomotorstandalone.py file at lines 18/19, comment in the filename** for the relay if that is what you are testing on, and comment out the other filename line

Now that is done you are ready for correction. All you have to do is close any open instances of metaguide, ascom, or new focus picomotor application, and then run this command in terminal:

python mg_pico.py

Everything from here will be automated. It takes a few seconds (~30-45) to acquire connections, open ports, get all programs live, initialize coordinates and set reference points), then you will see the deltas printed out in the terminal as correction runs. (in addition to hearing it).

End flexure correction

Use keyboard interrupt: Ctrl + c

This will turn off the Arduino, close the log txt file, close MetaGuide and ASCOM, clean the ports being used, and end the active threads (Listener and Monitor). This will also initiate final motor movements: each axis will take turns homing to their position at zenith. You should be able to hear when one motor stops moving and another starts, this is part of the rehoming process and should take around 2 minutes in total.

Common software and hardware issues

Connection problems can occur at any time. We believe they're mainly related to the USB hub. If connection problems (including those that say 'invalid literal for int()') occur at initialization, Ctrl+c and restart the script.

Always ensure that when you're running the script, only one instance of Metaguide is open. Multiple open Metaguide windows will stop the program from running correctly. If you need to force the program to end (which means you don't go through the ending sequence that turns

everything off), before rerunning, you'll need to close the Metaguide window, which will also close ASCOM.

List of issue and possible solutions:

- Control+C isn't ending the program or your are getting errors when you close the program: you can always just close the terminal, and the motors should automatically stop running as well, in the case you can still hear/see them moving, you can quickly attempt to restart the program and reclose as this will cycle the power to the controller, and reattempt the shutdown procedure.

- Port error: getting errors that say cant find port, or errors related to com. These may occur if the arduino isn't plugged in correctly, or if you switched to a different usb hub. Use device manager to ensure that you are able to see the arduino, the zwo camera, and the picomotor controller. Try replugging in the usb hub. If errors priests refer to step 13 of the setup above and reattempt.

- Backend error or no backend error in terminal: There is an issue with dll and libusb installed, refer to step 14.

- New Focus Picomotor Application won't find and connect to a 8742 controller board. Asure that you are not running mg_pico at the same time. Only one instance can communicate with the controller at once and others must be closed out. I.e close out the terminal that was running mg_pico or the compiler that has files from that directory open.

- Arduino Errors: Not found or port error, or light on the switch relay fails. Refer to step 9 of the software steps. In the case the arduino issues, assure that the light wires are plugged into ground and pin #8 into the board. The relay should be wired to 5V, Ground (on the side opposite of the the ground the light is plugged in), and into pin #8.

- Motors running, but no correct being made: if you can hear the motors running and no difference being made to deltas being printed out (or you can see 1 motor correcting, but the other is just making noise and not actually reducing the delta) for a prolonged time period (i.e 1 min). Three possibilities to check for:
    1) Software glitch: there could be a port error in metaguide that halted the data stream and now there is effectively no feedback. Restart the program. Make sure the computer you are using is not in battery saver or eco mode (as this may put programs/ threads/ processes into the background). If a motor is still on and doesn't correct deltas at all, proceed with the next two steps.
    2) Asure there is no mechanical jam on the correction stages, give light nudges to the relay translation stage, or the microlens arms to get some movement out of them.

3) The motor has lost contact, visually inspect inside chas and see that the motor screw tip is still in direct contact with the arm (for the microlens) or that the motor screw head is in contact with the relay stage screws. In case it has slipped passed, readjust the arms, and recentering the motors may be required. This is the most extreme case and is unlikely.
4) The motor has jammed. If the screw used to hold the picometers in place are over-tightened, the friction can cause them to jam in some sections of their movement. Removed the jammed motor, and reinstall at a slightly new rotated angle, without overtightening.

- No Connect to the picomotors: when you run mg_pcio you keep finding no newport controller or no pico motors. Attempt the mg_pico at least 4-5 times before attempting this troubleshooting. Also asure no New Focus Picomotor Applications are open when you are attempting running mg_pico. Check device manager to see that a picomotor controller is listed as connected to the usb. Reopen new focus picomotor applications and see if the controller pops up (you will have to close the compiler to open new focus sometimes), and that you can successfully give it commands and movements. If that succeeds, close the new focus picomotor application, and run python lighton.py (this will indirectly cycle the arduino's power, therefore also as a side effect restart the relay switch that controls the picomotor controller board and resets the board).

- The picomotors seem to make the correction in the wrong direction and make deltas grow worse instead of better: You axises may be inverted, repeat step 5 of the hardware setup.

- Mg_pico gets stuck at initialization: If the terminal won't move past initialization, it means there is an issue with the data stream from metaguide, or the more likely issue: the light isn't being picked up by metaguide. If the light source is coupled with the arduino, assure the arduino is wired correctly, and visually inspect the led connected to the arduino turns on when the computer runs the python lighton.py command. If it does not, check for loose wire, and refer to the port error issue above for more troubleshooting steps. If the light is fully on, assure that you can see it in the metaguide. Open metaguide manually via the search bar and use vidprops (pink button at bottom of screen) to adjust exp, gain, and gamma to get the light source in view. See setup step 3 for more details.

Contact info

We can use Parsec to remote into the system from our computers

Lindsay	lyk2116@columbia.edu
Afham	ab5509@columbia.edu