Goblin's Collimator Positioning Apparatus User Manual

Green's Goblins
Annalise Wolf, Ben Clark, Anna Taylor

Department of Physics
North Carolina State University
December 2022

1 Introduction

The Goblin's Collimator Positioning Apparatus (GCPA) is a device for the spatial characterization of radiation detectors. This device allows the user to scan and characterize radiation detector surfaces by moving a collimated gamma-ray source with high spatial precision. The user can choose the size of the scanning region, the resolution of the scan, and the desired exposure time, as well as other scanning properties.

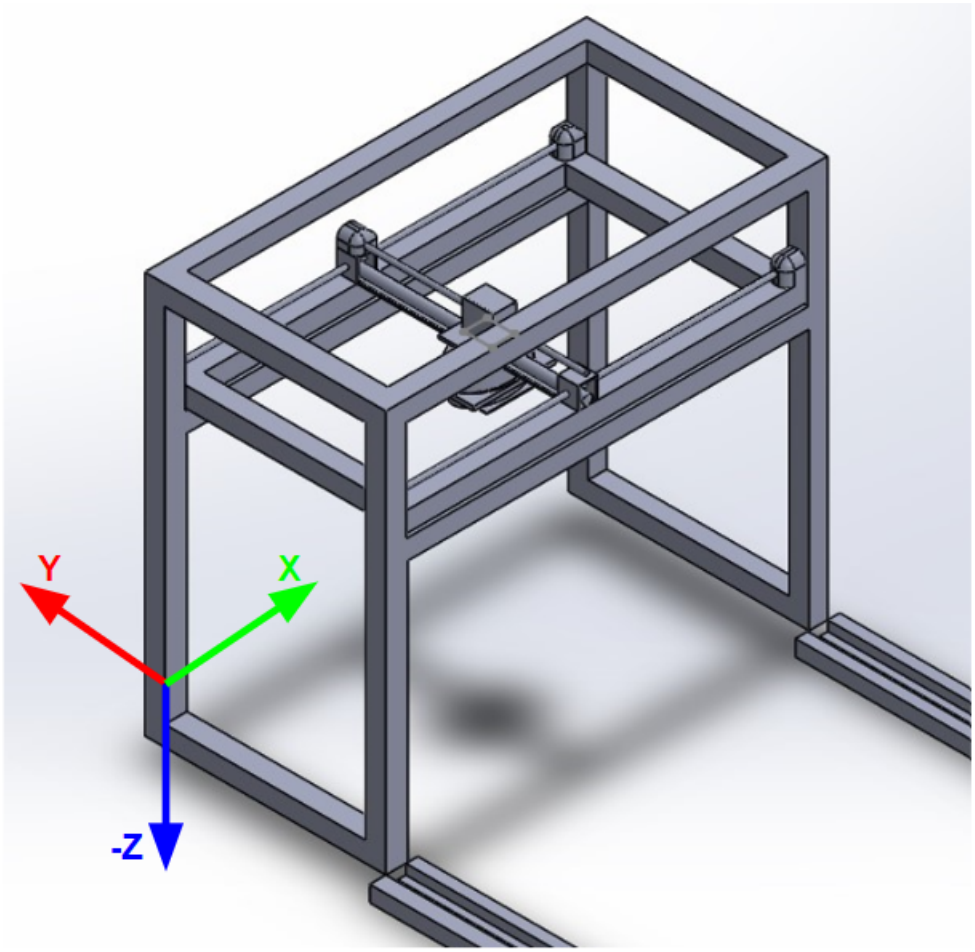

Figure 1: Visualization of the coordinate system we are adopting to describe our coordinate system

2 Parts List

Part Description	Quantity	Part Number	Function
60.5 cm T-Slot Single Frame	8	1010	Strut of the apparatus
40 cm T-Slot Single Frame	6	1010	Strut of the apparatus
T-Slot Frame Corner Bracket	28	47065T216	Connect T-Slot framing
53 cm Guide Rail	2		Guide motion
37 cm Guide Rail (Support Beam)	1		Guide motion
Support Beam Custom Barring	2		Support beam connection
Self Aligning M4 Nut	70		Fasten framing and pieces
M4 screw	70		Fasten framing and pieces
17HS19-2004S Stepper Motor	2	17HS19-2004S	Linear motion
17HS24-2104S Stepper Motor	1	17HS24-2104S	Linear motion
TB6600 Stepper Motor Driver	2	TB66000	Linear motion
DQ542MA Stepper Motor Driver	1	DQ542MA	Linear motion
50 cm Lead Screw and Fastener	2	LDSW-T8X8-500	Linear motion
8mm-8mm Coupling	3	ST-FC08	Linear motion
Mount Barring Unit	1		Upper mount
PTFE Sheet	1	9266K82	Bottom mount
5 inch B7 rods and nuts	4	98750A435	Bottom mount
5' x 5' mount plate	1		Bottom mount
Support Block	2		Bottom Mount
Ultra Flex Wire	2	1999N1	Connect counterweights
Loop and Thimble	2	1773N16	Connect counterweights
Clear PVC	2		Contain counterweights
8 lb Drop Weights	2		Counterweight
2.5 lb Plate	2		Counterweight
Double Mounted Pulley	1	3087T31	Support counterweights
3/16 Single Pulley	4	3213T52	Support counterweights
Arduino Mega 2560 Board	1	Mega 2560	Main control unit
Dupont Wires	18		Electronic connection
16 Gauge Wires	9		Electronic connection

3 Setup and Usage

Determine in which orientation scanning should be performed. We have defined a home position. In the origin, the collimator is at the far end of its support beam. This beam is in the middle along the axis with two motors. This means that y/z values range from 0 to 20, while x values range from -20 to 20. If the user would like to scan in the "horizontal" or "x-y" plane, continue with the steps below in section 3.1. If the user would like to start in the "vertical" or "x-z" plane, skip to section 3.2. If the apparatus is currently in one orientation and motion in the other is desired, refer

to section 3.3.

Important things to note:

1. The collimator and counterweights are heavy! Use extreme caution when moving them. The collimator is made of lead! Wear gloves or hold it by the handles. This system is designed to minimize the amount of collimator handling the user must do.

2. "Horizontal" (x-y) is defined by the apparatus resting in the tracks on its casters and stoppers. In this orientation, the gamma ray beam from the collimator is pointed down.

3. "Vertical" (x-z) is defined by the apparatus resting on its side, with the gamma ray beam parallel to the table.

4. It is important to note that there is an additional set of counterweights for upward motion in the vertical. If the user is going to perform scans in this direction, they must first attach the additional weight to the counterweight carabiners to ensure dependable motion.

3.1 In the Horizontal

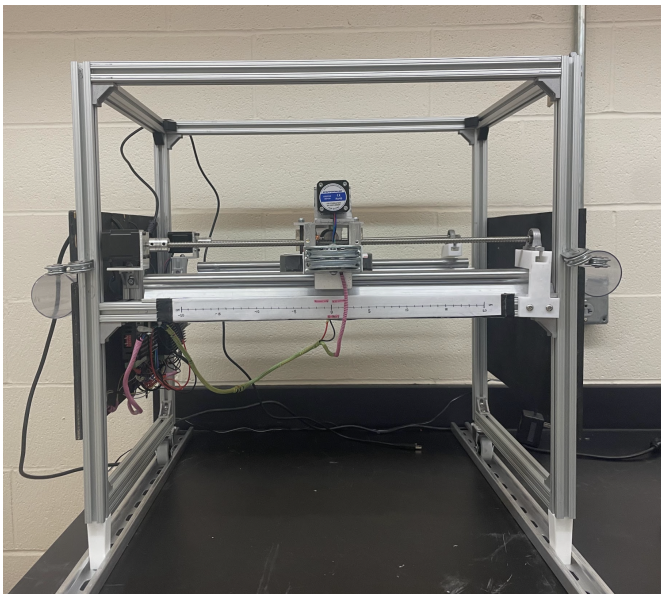

Figure 2: GCPA in the Horizontal Orientation/Mode

1. Position the device so that detector is close to the center of the scanning region. This will be more important for larger detectors. Make sure the power cable can reach the wall and the USB can reach your data acquisition computer but DO NOT plug them in yet.

2. Adjust the top of the frame which supports the electronics, guide rails, lead screws, and etc. so that it is at the desired height. To adjust the height, loosen the four corner screws, move the top of the frame to the desired height, then re-tighten the screws. Make sure not to fully

remove the four corner screws as it is difficult to get them back in if you do. Also note that the closer the collimator is to the detector the better your results so the user should try to lower the system as close to the detector surface as possible.

3. After ensuring that all four corner screws are tightened and at the same height with a level, place the radioactive source inside the collimator and place the cap on top to secure it within the collimator. WARNING, THE COLLIMATOR IS HEAVY!

4. Place the four bolts through the holes on the top of the mount on the "trolley piece" and slide the PTFE sheet up along the bolts, through the holes. Make sure the side labeled "COLLIMATOR" is facing the collimator to ensure the correct side will be clamped against the support beam. Next place the collimator on top of the metal support plate. Using one hand,or an assistant, lift up the metal plate and thread the bolts through the holes of the metal plate, with the major axis of the collimator perpendicular to the support beam. Tighten the clamping nuts in place. Wedge the support blocks between the bolts, above the PTFE sheet and below the "trolley piece" to secure the collimator in place.

5. Now that the collimator is secured, plug in the DC power supply and USB-B cord to the Arduino, and the AC power supply to the correct port to power the drivers.

6. The Arduino will come installed with the necessary software. The data acquisition computer requires the Python-based control software to be downloaded. The Python program, along with the Arduino program, can be found in https://github.com/GreensGoblins/GCPA2022. For more information on the software see Section 4. Run GCPA_PythonSoftware.py, Follow the onscreen prompts to choose operation orientation and location to send the collimator. Wait time and granularity can be easily changed in the code. For a more detailed description of these parameters see Section 4.

3.2 In the Vertical

1. If scanning in the "vertical" or "x-z" plane, the user must first reorient the apparatus before securing the collimator. In order to do so, grab the top T-slotted rails with both hands, and start to pull down to slide the casters inside the tracks. Pull the apparatus away from the detector space, and lay it down on its side.

2. First place the radioactive source inside the collimator and place the cap on top to secure it within the collimator. WARNING, THE COLLIMATOR IS HEAVY!

3. Place the four bolts through the holes on the top of the mount on the "trolley piece" and slide the PTFE sheet up along the bolts, through the holes. Make sure you place the side labeled "COLLIMATOR" down to ensure the correct side will be clamped against the support beam.

4. Attach the clear PVC pipes to the side of the apparatus, using the screws on the black panel.

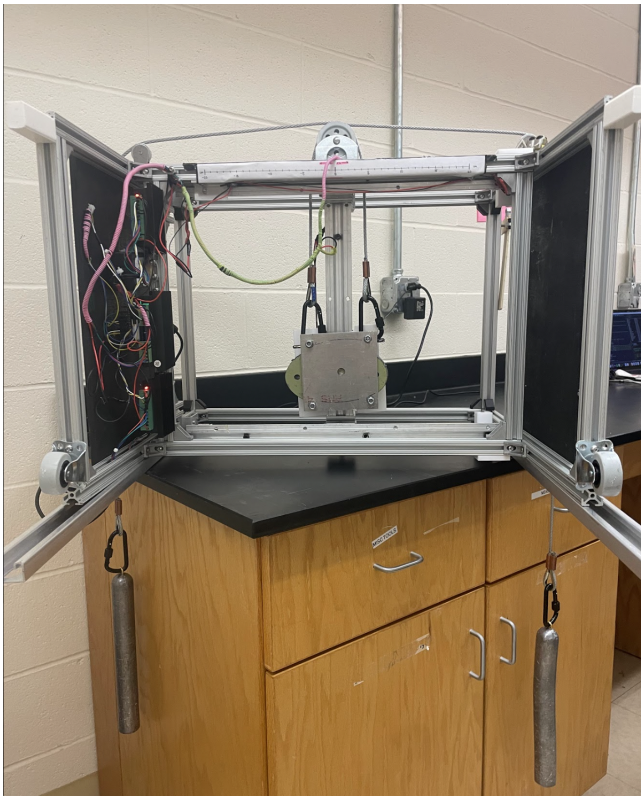

Figure 3: GCPA in the Vertical Orientation/Mode

5. Next slide the thimbles of the steel rope onto the top bolts. Thread the ropes up along the pulleys and through the clear PVC pipes. Then attach the counterweights to the other ends with the carabiners. Next, pick up the collimator by its handles.

6. Placing the cap-side towards the PTFE sheet, set the collimator onto the wooden stand, between the bottom bolts. Slide the metal plate onto the bolts through its holes and secure with the clamping nuts. Wedge the support blocks between the bolts, the PTFE sheet, and the "trolley piece" to secure the collimator in place and slide the thin wood support between the top of the rounded edge of the collimator and the top bolts.

7. Now that the collimator is secured, plug in the DC power supply and USB-B cord to the Arduino, and the AC power supply to the correct port to power the drivers.

8. The Arduino will come installed with the necessary software. The data acquisition computer requires the Python based control software to be downloaded. The Python program, along with the Arduino program, can be found in https://github.com/GreensGoblins/GCPA2022. For more information on the software see Section 4. Run GCPA_PythonSoftware.py. Follow the onscreen prompts to choose operation orientation and location to send the collimator. Wait time and granularity can be easily changed in the code. For a more detailed description of these parameters see Section 4.

9. If moving up in the vertical direction, attach the additional set of counterweights to the carabiners.

3.3 Transitioning

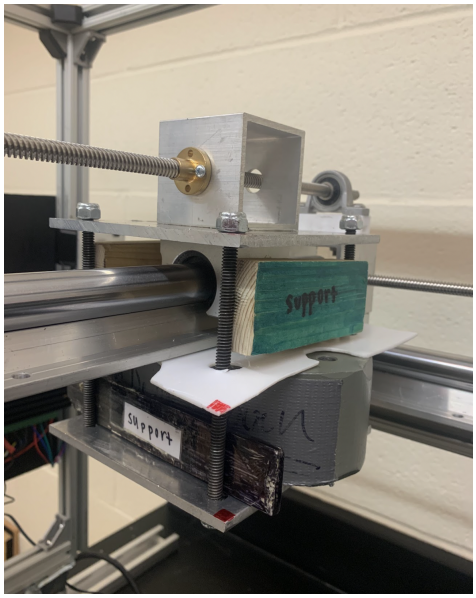

Figure 4: GCPA: The mount for the collimator

3.3.1 From Horizontal to Vertical

1. After scanning in the horizontal is complete, follow the prompts on the data acquisition computer to calibrate the system and move it into the "Transition Position". This will send the collimator into the optimal place for rotation.

2. In order to rotate the system, grab the T-slotted rail labeled "HOLD HERE" with both hands, and start to pull down to slide the casters inside the tracks. Pull the apparatus away from the detector space, and lay it down on its side.

3. Attach the clear PVC pipes to the side of the apparatus, using the screws on the black panel.

4. Remove the support blocks. Unscrew and slide out the top bolts, and thread the end of the counterweight rope through the bolt. These should be between the trolley piece and the PTFE sheet. Run the ropes along the pulley system and attach the counterweights with carabiners.

5. Replace the support blocks. The system is now ready to scan in the vertical directions. Follow the prompts on the data acquisition computer to move the collimator.

6. It is important to note that there is an additional set of counterweights for upward motion in the vertical. If the user is going to perform scans in this direction, they must first attach the additional weight to the counterweight carabiners to ensure dependable motion.

3.3.2 From Vertical to Horizontal

1. After scanning in the vertical is complete, follow the prompts on the data acquisition computer to calibrate the system and move it into the "Transition Position". This will send the collimator into the optimal place for rotation.

2. The user must remove the counterweights from the system before rotating. Unscrew the top bolts and slide them out from the mount. One at a time, unthread the rope from around the pulleys and set aside.

3. Replace the bolts and screw the clamping nuts back in. Slide the support blocks back between the bolts.

4. In order to rotate the system, grab the T-slotted rail labeled "HOLD HERE" with both hands, and start to pull up to slide the casters inside the tracks. Push the apparatus over the detector space, and ensure that the stoppers and casters rest inside the tracks for stability.

5. The system is now ready to scan in the horizontal direction. Follow the prompts on the data acquisition to scan in this plane.

4 Software

To access the software needed to use the GCPA, go to the GitHub repository

`https://github.com/GreensGoblins/GCPA2022`

The repository includes the file that comes uploaded to the Arduino "GCPA_ArduinoSoftware.ino" and the Python file "GCPA_PythonSoftware.py" needed to run the program. Additionally, the Arduino and Python software is written out in the Appendix of this manual.

4.1 Arduino Software

The Arduino program defines the pins used for the stepper motor driver and switches. There are four functions defined as subroutines for the Arduino to execute.The four different functions are: movex, movextoEND, movey, and moveytoEND. movex and movextoEND move the single x motor and movey and moveytoEND move the two y motors. In these functions, there are conditions that will stop the motor if a switch is hit. movex and movey will move the motors in a specified direction and specified number of steps. movextoEND and moveytoEND will continue to move the motors until a switch is hit. It also counts the number of steps it takes until it hits a switch and returns that count. In the loop portion of the Arduino code, a serial port is open and it waits to receive inputs from the port. From the Python script, a string is sent to the Arduino. In the

loop, the string is parsed into variables motor, steps, and direc. The motor variable denotes which function to use: if motor = 'x' then movex is used, if motor = 'y' then movey is used, if motor = 'xEND' then movextoEND is used, and finally if motor = 'yEND' then moveytoEND is used. The steps variable tells the stepper motors how many steps to take in the movex or movey functions. Lastly, the direc variable tells the motors which way to turn in the movex, movey, xEND and yEND functions. A '1' means to turn counterclockwise and '0' means clockwise.

4.2 Arduino Software Variables

The Arduino software waits for an input string which the Python software sends. This string will take the form of:

```
motor,steps,direction
```

The first word, "motor," will determine which subroutine the Arduino will have the stepper motors execute. "Steps" determines the number of steps which the stepper motor will attempt to execute. And "direction" determines the direction of rotation of the stepper motors in their respective routines.

An example of the four subroutines which the Arduino can execute are:

Example String	Subroutine	Description
x,50,1	Move in the "x" direction	Rotates the single stepper motor a number of steps in a specified direction
y,100,0	Move in the "y" direction	Rotates the duel stepper motors a number of steps in a specified direction
xEND,,0	Move to a "x" limit switch	Rotates the single stepper motor one direction until a limit switch is completed
yEND,,1	Move to a "y" limit switch	Rotates the duel stepper motors one direction until a limit switch is completed

4.3 Python Software

In order to run the Python code, the user might first need to install some packages into their Python environment. For the serial connection, you must install the serial package by running the command

```
pip install serial
```

in a terminal. In order for the Python software to output time, it uses the "datetime" package. The user can install this package with a similar terminal command given by:

```
pip install datetime
```

The first cell (In[1]:) in the Python software simply imports the packages to be used in the software including serial, numpy, tkinter, etc., and defines constants associated with the stepper motors for the code to use.

Figure 5: The user input prompt for the USB port.

The next cell (In[2]:) in the Python software prompts the user to input the string containing the path or name of the USB port to which the Arduino is connected to (Figure 5).

Depending on what operating system the data acquisition computer has, this string will look different and be found in different places. The instructions for Mac and Windows OS are laid out below.

- For a Mac OS:
    1. Open a terminal window.
    2. Type "cd /dev", hit enter.
    3. Type "ls", hit enter. A list of device names will be shown.
    4. Look for any device name that looks like "usbmodem1200", or any name that starts with "usbmodem" and ends in a number.
    5. Once you find this device name, type it, and the path, into the user USB modem user prompt given by the Python software which will look something like "/dev/usbmodem...", for example "/dev/usbmodem1200".
- For a Windows OS:
    1. Open settings.
    2. Go to the "Bluetooth and Devices" page. A list of device names will be shown.
    3. Find the device plugged into a USB port. It will have a name like "COM3", or a name that starts with "COM" and ends in a number.
    4. Once you find this device name, type it into the user USB modem user prompt given by the Python software which will look something like "COM...", for example, "COM3" (no path is needed in this case).

In the third and final cell (In[3]:), The user is first prompted to input which orientation the GCPA is in, horizontal ("H") or vertical ("V"), or if the user wants to change from one orientation to another, transition ("T") (Figure 6).

4.3.1 Horizontal Mode

If the user chooses the horizontal orientation, the collimator will move to the home position (0,0), and the user will first receive information about the coordinate system and a plot showing the

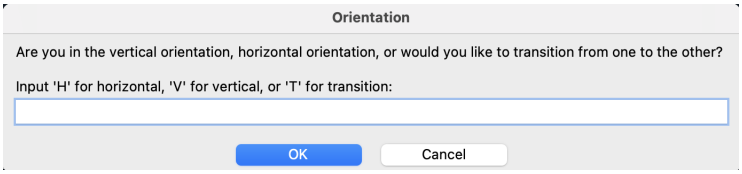

Figure 6: The user input prompt for the orientation or transition.

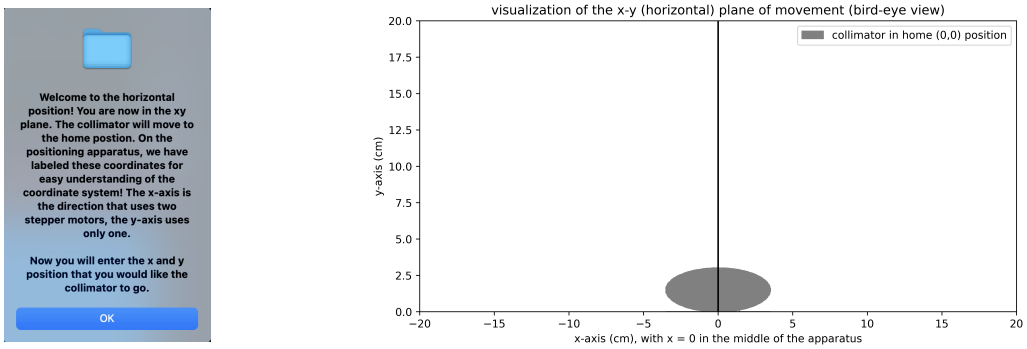

Figure 7: Information for the user about the coordinate system in the horizontal orientation.

coordinate system (Figure 7). Next, a text file is opened up where the positions and timestamps will be stored throughout the positioning process.

Next, the user is prompted to input the x and y coordinates of the position they would like the collimator to go to from the home (0,0) position (Figures 12 and 9).

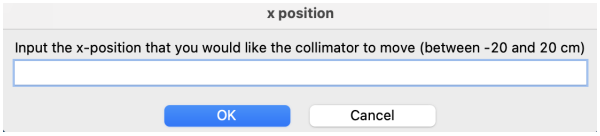

Figure 8: The user input prompt for the x coordinate in the horizontal mode.

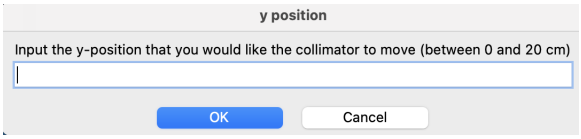

Figure 9: The user input prompt for the y coordinate in the horizontal mode.

If the user puts in any coordinates outside of the system, they will be prompted to put a correct value in. Then, the stepper motors will move the collimator to the given position in default steps of 1mm with wait times of 2 seconds (can be changed in the code). If the user inputs a coordinate that is not a multiple of 1mm, for instance, 0.24 cm, then the code will take 1mm steps until it needs to take a smaller step in order to achieve this step. The user should refrain from inputting steps with more than two decimal places. This motion is documented in the output file. Finally, the user asked if they would like to move to another position in the horizontal mode (Figure 10).

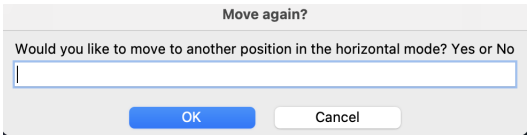

Figure 10: Prompting the user if they would like to move again in the horizontal mode.

If the user inputs "yes," the user will be prompted to input the x and y coordinates of the position they would like the collimator to go to from the current position (Figures 12 and 9). The user will be continually asked if they would like the collimator to move to a new position in the horizontal mode until they input "no." When the user types no, the cell will exit and if the user wants to continue to move in the horizontal position, or transition from the horizontal mode to the vertical mode, they must re-run cell 3 (In[3]:).

4.3.2 Vertical Mode

If the user chooses the vertical mode ("V"), the collimator will move to the home position (0,0), and the user will first receive information about the coordinate system and a plot showing the coordinate system (Figure 11). Next, a text file is opened up where the positions and timestamps will be stored throughout the positioning process.

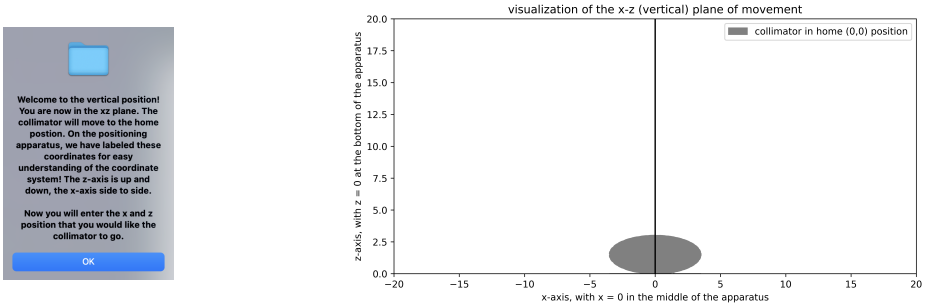

Figure 11: Information for the user about the coordinate system in the vertical orientation.

Next, the user is prompted to input the x and z coordinates of the position they would like the collimator to go to from the home (0,0) position (Figures 12 and 13).

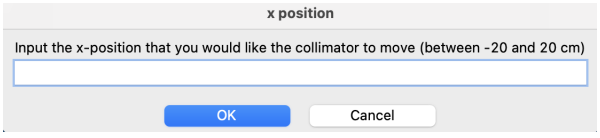

Figure 12: The user input prompt for the x coordinate in the vertical mode.

If the user puts in any coordinates outside of the system, they will be prompted to put a correct value in. If the user puts anything but zero into the z position input, the software will warn the user to ensure that the extra weight is attached to the pullies, in order to ensure the collimator will move accurately (Figure 14).

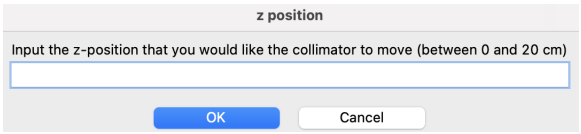

Figure 13: The user input prompt for the z coordinate in the vertical mode.

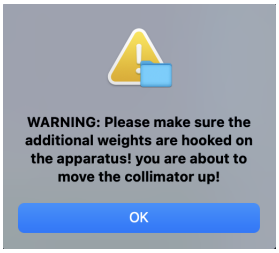

Figure 14: The warning to ensure the correct weight is on the pulley.

Then, the stepper motors will move the collimator to the given position in default steps of 1mm and wait times between steps of 2 seconds (can be changed in the code). If the user inputs a coordinate that is not a multiple of 1mm, for instance, 0.24 cm, then the code will take 1mm steps until it needs to take a smaller step in order to achieve this step. The user should refrain from inputting steps with more than two decimal places. This motion is documented in the output file. Finally, the user asked if they would like to move to another position in the vertical mode (Figure 15). If the user inputs "yes," the user will be prompted to input the x and z coordinates

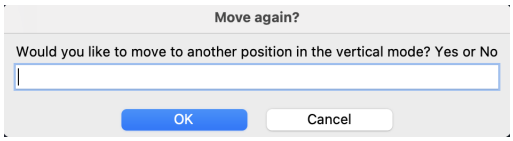

Figure 15: Prompting the user if they would like to move again in the vertical mode.

of the position they would like the collimator to go to from the current position (Figures 12 and 13). The user will be continually asked if they would like the collimator to move to a new position in the vertical mode until they input "no." If the collimator is going to be moved up based on the user input, the warning in Figure 14 will pop up. If the collimator is to move down based on the user input, a warning similar to Figure 14 will pop up warning the user to remove the extra set of weights to ensure the collimator moves accurately. When the user types no, the cell will exit, and if the user wants to continue to move in the vertical mode or transition from the vertical mode to the horizontal mode, they must re-run cell 3 (In[3]:).

4.3.3 Transition Mode

If the user chooses the transition mode ("T"), the collimator will move the home position no matter which mode (vertical or horizontal) it is currently in and print out brief instructions on changing modes, either from vertical to horizontal or vice versa (Figure 16).

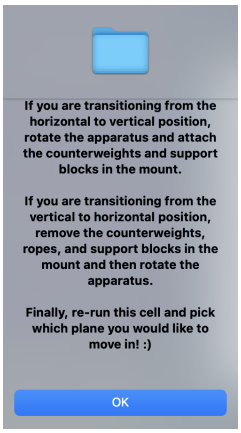

Figure 16: Information for the user about the transition mode.

4.4 Data Output

While running, this device generates a text file called "sourcepositions.txt" to record the current position of the gamma radiation beam and the timestamp at 2-second intervals. The left column is the timestamp, the middle column is the x position, and the right column is the y or z position (depending on orientation). These positions are relative to the home position in the bottom left corner.

5 Electronics

The main electronic components consist of:

- 1 Arduino Mega 2560 Board
- 2 TB6600 Stepper Motor Drivers
- 1 DQ542MA Stepper Motor Driver
- 2 17HS19-2004S1 Stepper Motors
- 1 17HS24-2104S Stepper Motor

The 2 17HS19-2004S1 stepper motors are connected to the 2 TB6600 stepper motor driver and the 17HS24-2104S stepper motor is connected to the DQ542MA stepper motor driver. The second set of connections link the enable, direction and pulse pins from the stepper motor drivers to the Arduino board. The connections and correct pin numbers are shown in Figure 17. The correct pin configuration for each of the stepper motor drivers is also shown in Figure 17.

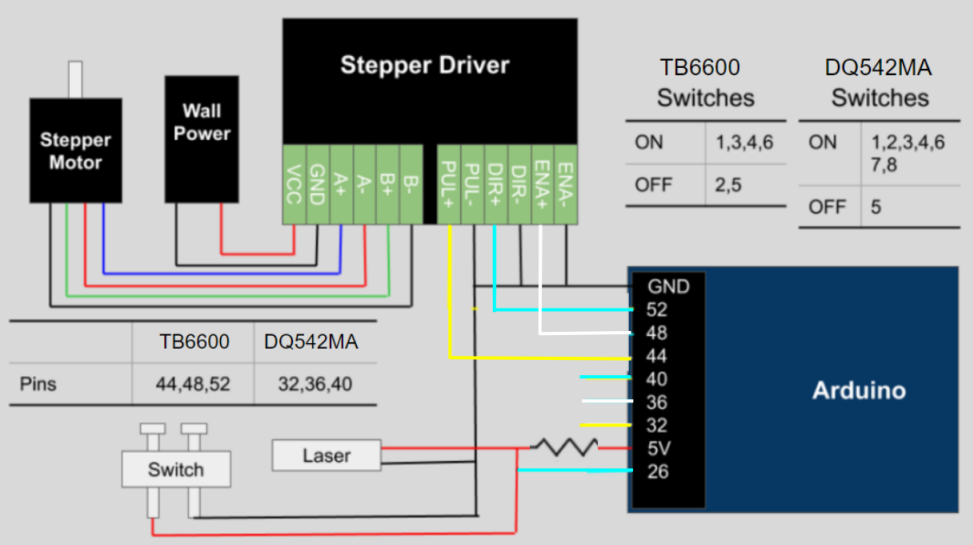

Figure 17: The electronic circuit diagram. The order of the pin numbers for each stepper driver goes as: PUL+,ENA+,DIR+. The resistor is a 10Kohm resistor.

The 2 17HS19-2004S1 stepper motors fixed parallel to each other along the frame of GCPA while the 17HS24-2104S stepper motor is fixed on the support beam opposing the home position as shown in Figure 18.

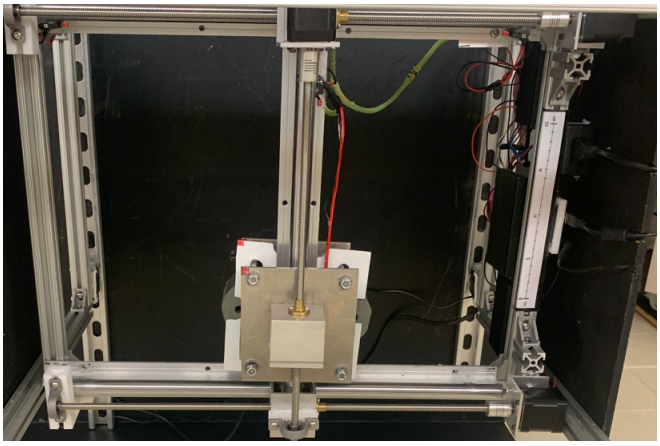

Figure 18: The singular stepper motor located along the moving support beam and the duel stepper motors located along the frame.

Appendix

Arduino Software

Here is a copy of the Arduino code:

```
#include <SPI.h>
//Defined varibles
String motor;
String steps;
String dir;
String direc;
String xEND;
String yEND;
int steps_x;
int steps_y;

//Pin Varibles and their description of output
int PUL_T_x = 32; //define Pulse pin for motor 1, the single motor on the support beam
int DIR_T_x = 40; //define Direction pin for motor 1, the single motor on the support beam
int ENA_T_x = 36; //define Enable Pin for motor, the single motor on the support beam

int PUL_T_y = 44; //define Pulse pin for motors 2,3, the duel motors which run in parrellel
int DIR_T_y = 52; //define Direction pin for motors 2,3, the duel motors which run in parrellel
int ENA_T_y = 48; //define Enable Pin for motors 2,3, the duel motors which run in parrellel


int home_switch1 = 26; //define home switches pin
//Next to be defined are the sub routines which the arduino can run
//The first is the movex function which either rotates the single stepper motor clockwise or counterclockwise
//This function requires an input number of steps to take and a set direction
void movex(int steps, int dir) {
    for (int i = 0; i < steps; i++) {
        digitalWrite(ENA_T_x,HIGH);
        digitalWrite(PUL_T_x,HIGH);
        digitalWrite(DIR_T_x,dir);
        digitalWrite(PUL_T_y,LOW);
        digitalWrite(ENA_T_y,LOW);
        delayMicroseconds(1000);
        digitalWrite(PUL_T_x,LOW);
        digitalWrite(ENA_T_x,LOW);
        digitalWrite(DIR_T_x,dir);
        digitalWrite(PUL_T_y,LOW);
        digitalWrite(ENA_T_y,LOW);
        delayMicroseconds(1000);
    }
    if (!(digitalRead(home_switch1)==1 || digitalRead(home_switch1)==1 || digitalRead(home_switch1)==1)){
        for (int i = 0; i < 200; i++) {
          digitalWrite(ENA_T_x,HIGH);
          digitalWrite(PUL_T_x,HIGH);
          digitalWrite(DIR_T_x,abs(1-dir));
          digitalWrite(PUL_T_y,LOW);
          digitalWrite(ENA_T_y,LOW);
          delayMicroseconds(1000);
          digitalWrite(PUL_T_x,LOW);
          digitalWrite(ENA_T_x,LOW);
          digitalWrite(DIR_T_x,abs(1-dir));
          digitalWrite(PUL_T_y,LOW);
          digitalWrite(ENA_T_y,LOW);
```

```
          delayMicroseconds(1000);
      }
    }

}

//The second xmove subroutine which we will define is the movex to the end. This subroutine will have the stepper motor
//continuously step until one of the two limit switches are completed. The input of a step direction decides which direction
//the stepper motor will step and by extension which limit switch will be activated.
int movex_toEND(int dir) {
    //int StepsToEnd = steps_x - steps;
    int stepCount = 0;  // number of steps the motor has taken

    while (digitalRead(home_switch1)==1 || digitalRead(home_switch1)==1 || digitalRead(home_switch1)==1) {
      digitalWrite(DIR_T_x, dir);      // (HIGH = anti-clockwise / LOW = clockwise)
      digitalWrite(PUL_T_x, HIGH);
      delay(2);                        // Delay to slow down speed of Stepper
      digitalWrite(PUL_T_x, LOW);
      delay(2);
      stepCount++;
      }

    for (int i = 0; i < 200; i++) { //Back off switch
            digitalWrite(ENA_T_x,HIGH);
            digitalWrite(PUL_T_x,HIGH);
            digitalWrite(DIR_T_x,abs(1-dir));
            digitalWrite(PUL_T_y,LOW);
            digitalWrite(ENA_T_y,LOW);
            delay(1);
            digitalWrite(PUL_T_x,LOW);
            digitalWrite(ENA_T_x,LOW);
            digitalWrite(DIR_T_x,abs(1-dir));
            digitalWrite(PUL_T_y,LOW);
            digitalWrite(ENA_T_y,LOW);
            delay(1);
          }


  return stepCount;
}

//Simular to the movex subroutine, the movey subroutine is simularly formatted to have the stepper motor rotate either
//clockwise or counterclockwise only this time controls the duel stepper motors which run in parrellel with one another.
void movey(int steps, int dir) {
  for (int i = 0; i < steps; i++) {
      digitalWrite(ENA_T_y,HIGH);
      digitalWrite(PUL_T_y,HIGH);
      digitalWrite(DIR_T_y,dir);
      digitalWrite(PUL_T_x,LOW);
      digitalWrite(ENA_T_x,LOW);
      delayMicroseconds(1200);
      digitalWrite(PUL_T_y,LOW);
      digitalWrite(ENA_T_y,LOW);
      digitalWrite(DIR_T_y,dir);
      digitalWrite(PUL_T_x,LOW);
      digitalWrite(ENA_T_x,LOW);
      delayMicroseconds(1200);
    }
```

```
    if (!(digitalRead(home_switch1)==1 || digitalRead(home_switch1)==1 || digitalRead(home_switch1)==1)) {
      for (int i = 0; i < 200; i++) {
            digitalWrite(ENA_T_y,HIGH);
            digitalWrite(PUL_T_y,HIGH);
            digitalWrite(DIR_T_y,abs(1-dir));
            digitalWrite(PUL_T_x,LOW);
            digitalWrite(ENA_T_x,LOW);
            delayMicroseconds(1000);
            digitalWrite(PUL_T_y,LOW);
            digitalWrite(ENA_T_y,LOW);
            digitalWrite(DIR_T_y,abs(1-dir));
            digitalWrite(PUL_T_x,LOW);
            digitalWrite(ENA_T_x,LOW);
            delayMicroseconds(1000);
          }

    }
}

//And again we'll have a second subroutine for the y limit switches as well so that we can also
//decide which direction we would
//like the stepper motors to turn and which limit switch we will be waiting for.
int movey_toEND(int dir) {
    //int StepsToEnd = steps_y - steps;
    int stepCount = 0;  // number of steps the motor has taken
    while (digitalRead(home_switch1)==1 || digitalRead(home_switch1)==1 || digitalRead(home_switch1)==1) {
      digitalWrite(DIR_T_y, dir);
      digitalWrite(PUL_T_y, HIGH);
      delayMicroseconds(800);                      // Delay to slow down speed of Stepper
      digitalWrite(PUL_T_y, LOW);
      delayMicroseconds(800);
      stepCount++;
      }
    for (int i = 0; i < 200; i++) { //Back off switch
          digitalWrite(ENA_T_y,HIGH);
          digitalWrite(PUL_T_y,HIGH);
          digitalWrite(DIR_T_y,abs(1-dir));
          digitalWrite(PUL_T_x,LOW);
          digitalWrite(ENA_T_x,LOW);
          delayMicroseconds(800);
          digitalWrite(PUL_T_y,LOW);
          digitalWrite(ENA_T_y,LOW);
          digitalWrite(DIR_T_y,abs(1-dir));
          digitalWrite(PUL_T_x,LOW);
          digitalWrite(ENA_T_x,LOW);
          delayMicroseconds(800);
        }

  return stepCount;
}


void loop() {
  // The ardino will contimue to cycle through the input string which is sent from the python code to the arduino
  // if the string contains the correct labels and inputs of steps or direction, depending upon the command, the Arduino
  // will then execute the correct subroutine.
    if(Serial.available()>0) {
    motor = Serial.readStringUntil(',');
    steps = Serial.readStringUntil(',');
```

```
    direc = Serial.readStringUntil('\n');


    if(motor == String('x')){
    movex(steps.toInt(), direc.toInt());
    }

    else if(motor == String('y')){
    movey(steps.toInt(), direc.toInt());
    }

    else if(motor == "xEND"){
    int step_end = movex_toEND(direc.toInt());
    Serial.println(step_end);
    }

    else if(motor == "yEND"){
    int step_end = movey_toEND(direc.toInt());
    Serial.println(step_end);
    }


    else if(motor == "Calibration"){
    Calibration();
    Serial.println(steps_x);
    Serial.println(steps_y);
    }
  }

}
```

Python Software

```
# -*- coding: utf-8 -*-
"""
Green's Goblins Senior Design Code

Goblin's Collimator Positioning Apparatus

Fall 2022

@author: Anna Taylor, Ben Clark, Annalise Wolfe
"""

# In[1]:
import numpy as np
import matplotlib.pyplot as py
import time
import serial
import datetime
import tkinter as tk
from tkinter import simpledialog
import tkinter.messagebox as tkmb

#Constants related to the Arduino, stepper motors,and the default granularity and wait time
pitch = .8 # distance between threads in metric units (cm)
step_ang = 1.8 # degrees
steps_per_rev_x = 400 # number of steps to travel the distance equivalent to pitch
steps_per_rev_y = 400 # number of steps to travel the distance equivalent to pitch
steps_per_rev_z = 400 # number of steps to travel the distance equivalent to pitch
gran = 0.1 # the default step size taken to get to a coordinate (in cm)
wait_time = 2 # the default wait times taken in between steos to get to a coordinate (in cm)
ROOT = tk.Tk()
ROOT.geometry('1000x1000')
ROOT.withdraw()
date = datetime.datetime.now().timestamp()
# In[2]:
#Prompt for the user to put in the name and/or path to the USB connection
usb = simpledialog.askstring(title="USB Port Name", prompt= "Welcome to the Lead Collimator
Positioning System! \n\nPlease insert the name of or path to the USB port you are using. This can be
found in the devices settings on a Windows OS (example 'COM3') \nor in the /dev directory on a Mac
(example 'usbmodem1200') \n\nInput USB Port:")

# Establish a serial connection with Arduino
try:
    ser = serial.Serial(usb, 9600)
except:
    tkmb.showerror("Error!", "Try unplugging and replugging the Arduino cable! Then re-run this cell.
    \n\nIf that still results in an error, re-check your usb port name. \n\n If you need more help
    please refer to the user manual.")
# In[3]:

# Choosing which mode the user wants to be in
ot = simpledialog.askstring(title="Orientation", prompt= "Are you in the vertical orientation,
horizontal mode, or would you like to transition from one to the other? \n\nInput 'H' for horizontal,
'V' for vertical, or 'T' for transition:")
if ot == 'H' or ot =='h':

    #Moving to home (0,0) position
    ser.write(str.encode("xEND,,1"))
    time.sleep(20)
```

```
ser.write(str.encode("yEND,,1"))
time.sleep(30)
start = (20/pitch) * steps_per_rev_x
movehome = 'y,' + str(start) + ',0'
ser.write(str.encode(movehome))
time.sleep(20)
file = open(r'sourcepositions.txt', 'w+')
tkmb.showinfo("Horizontal Position", "Welcome to the horizontal position! You are now in the xy
plane. The collimator will move to the home postion. On the positioning apparatus, we have
labeled these coordinates for easy understanding of the coordinate system! The x-axis is the
direction that uses two stepper motors, the y-axis uses only one. \n \nNow you will enter the x
and y position that you would like the collimator to go.")

# Visualization of Coordinate System
py.figure(figsize = (10, 5))
u=0.     #x-position of the center
v=1.5    #y-position of the center
a=3.5     #radius on the x-axis
b=1.5    #radius on the y-axis
x = np.linspace(-20, 20, 100)
py.plot( u+a*np.cos(x) , v+b*np.sin(x) , color = 'gray')
py.axvline(x = 0, color = 'black')
py.fill_between(u+a*np.cos(x) , v+b*np.sin(x) , alpha=1, color ='gray', label = "collimator in
home (0,0) position")
py.ylim(0,20)
py.xlim(-20,20)
py.legend()
py.xlabel("x-axis (cm), with x = 0 in the middle of the apparatus")
py.ylabel("y-axis (cm)")
py.title("visualization of the x-y (horizontal) plane of movement (bird-eye view)")
py.show()

#Asking which position on the coordinate system the user wants the collimator to move to
xloc = float(simpledialog.askstring(title="x position", prompt= "Input the x-position that you
would like the collimator to move (between -20 and 20 cm)"))
while (xloc)>20 or (xloc)<-20:
     xloc = float(simpledialog.askstring(title="x position", prompt= "Oops! Please enter a valid
     x position. \nInput the x-position that you would like the collimator to move (between -20
     and 20 cm)"))

yloc = float(simpledialog.askstring(title="y position", prompt= "Input the y-position that you
would like the collimator to move (between 0 and 20 cm)"))
while (yloc)>20 or (yloc)<0:
     yloc = float(simpledialog.askstring(title="y position", prompt= "Oops! Please enter a valid
     y position. \nInput the y-position that you would like the collimator to move (between 0 and
     20 cm)"))

x_pos = 0
y_pos = 0

if (xloc > 0):
    start_stepsx = (xloc/pitch) * steps_per_rev_x
    move_x = (gran/pitch) * steps_per_rev_x
    date = datetime.datetime.now().timestamp()
    file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
    while(start_stepsx >= move_x):
        inputmovex = 'y,' + str(move_x) + ',1'
        ser.write(str.encode(inputmovex))
        start_stepsx -= move_x
```

```
        x_pos += gran
        date = datetime.datetime.now().timestamp()
        file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
        time.sleep(wait_time)
    if start_stepsx != 0:
        temp = xloc % gran
        move_extra = start_stepsx
        inputmoveextra = 'y,' + str(move_extra) + ',1'
        ser.write(str.encode(inputmoveextra))
        x_pos += temp
        date = datetime.datetime.now().timestamp()
        file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
        time.sleep(wait_time)

if (xloc < 0):
    xloc = -xloc
    start_stepsx = (xloc/pitch) * steps_per_rev_x
    date = datetime.datetime.now().timestamp()
    file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
    move_x = (gran/pitch) * steps_per_rev_x
    while(start_stepsx >= move_x):
        inputmovex = 'y,' + str(move_x) + ',0'
        ser.write(str.encode(inputmovex))
        start_stepsx -= move_x
        x_pos -= gran
        date = datetime.datetime.now().timestamp()
        file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
        time.sleep(wait_time)
    if start_stepsx != 0:
        temp = xloc % gran
        move_extra = start_stepsx
        inputmoveextra = 'y,' + str(move_extra) + ',0'
        ser.write(str.encode(inputmoveextra))
        x_pos -= temp
        date = datetime.datetime.now().timestamp()
        file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
        time.sleep(wait_time)

start_stepsy = (yloc/pitch) * steps_per_rev_y
date = datetime.datetime.now().timestamp()
file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
move_y = (gran/pitch) * steps_per_rev_y
while(start_stepsy >= move_y):
    inputmovey = 'x,' + str(move_y) + ',0'
    ser.write(str.encode(inputmovey))
    start_stepsy -= move_y
    y_pos += gran
    date = datetime.datetime.now().timestamp()
    file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
    time.sleep(wait_time)
if start_stepsy != 0:
    temp = yloc % gran
    move_extra = start_stepsy
    inputmoveextra = 'x,' + str(move_extra) + ',0'
    ser.write(str.encode(inputmoveextra))
    y_pos += temp
    date = datetime.datetime.now().timestamp()
    file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
    time.sleep(wait_time)
```

```
# Asking if the user wants the collimator to move to a new position
move = str(simpledialog.askstring(title="Move again?", prompt= "Would you like to move to another
position in the horizontal mode? Yes or No"))
while (move == "yes" or move == "Yes" or move == "y"):

        xloc = float(simpledialog.askstring(title="x position", prompt= "Input the x-position that
        you would like the collimator to move (between -20 and 20 cm)"))
        while (xloc)>20 or (xloc)<-20:
             xloc = float(simpledialog.askstring(title="x position", prompt= "Oops! Please enter a
             valid x position. \nInput the x-position that you would like the collimator to move
             (between -20 and 20 cm)"))

        yloc = float(simpledialog.askstring(title="y position", prompt= "Input the y-position that
        you would like the collimator to move (between 0 and 20 cm)"))
        while (yloc)>20 or (yloc)<0:
             yloc = float(simpledialog.askstring(title="y position", prompt= "Oops! Please enter a
             valid y position. \nInput the y-position that you would like the collimator to move
             (between 0 and 20 cm)"))

        x_move = xloc - x_pos
        if (x_move > 0):
         date = datetime.datetime.now().timestamp()
         file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
         start_stepsx = (x_move/pitch) * steps_per_rev_x
         while(start_stepsx >= move_x):
             inputmovex = 'y,' + str(move_x) + ',1'
             ser.write(str.encode(inputmovex))
             start_stepsx -= move_x
             x_pos += gran
             date = datetime.datetime.now().timestamp()
             file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
             time.sleep(wait_time)
         if start_stepsx != 0:
             temp = xloc % gran
             move_extra = start_stepsx
             inputmoveextra = 'y,' + str(move_extra) + ',1'
             ser.write(str.encode(inputmoveextra))
             x_pos += temp
             date = datetime.datetime.now().timestamp()
             file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
             time.sleep(wait_time)
        if (x_move < 0):
         x_move = -x_move
         start_stepsx = (x_move/pitch) * steps_per_rev_x
         while(start_stepsx >= move_x):
             date = datetime.datetime.now().timestamp()
             file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
             inputmovex = 'y,' + str(move_x) + ',0'
             ser.write(str.encode(inputmovex))
             start_stepsx -= move_x
             x_pos -= gran
             time.sleep(wait_time)
         if start_stepsx != 0:
             temp = xloc % gran
             move_extra = start_stepsx
             inputmoveextra = 'y,' + str(move_extra) + ',0'
             ser.write(str.encode(inputmoveextra))
             x_pos -= temp
```

```
                date = datetime.datetime.now().timestamp()
                file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
                time.sleep(wait_time)

        y_move = yloc - y_pos

        if (y_move > 0):
         start_stepsy = (y_move/pitch) * steps_per_rev_z
         while(start_stepsy >= move_y):
                date = datetime.datetime.now().timestamp()
                file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
                inputmovey = 'x,' + str(move_y) + ',0'
                ser.write(str.encode(inputmovey))
                start_stepsy -= move_y
                y_pos += gran
                time.sleep(wait_time)
         if start_stepsy != 0:
                temp = yloc % gran
                move_extra = start_stepsy
                inputmoveextra = 'x,' + str(move_extra) + ',0'
                ser.write(str.encode(inputmoveextra))
                y_pos += temp
                date = datetime.datetime.now().timestamp()
                file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
                time.sleep(wait_time)
        if (y_move < 0):
         y_move = -y_move
         start_stepsz = (y_move/pitch) * steps_per_rev_x
         while(start_stepsy >= move_y):
                date = datetime.datetime.now().timestamp()
                file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
                inputmovey = 'x,' + str(move_y) + ',1'
                ser.write(str.encode(inputmovey))
                start_stepsy -= move_y
                y_pos += gran
                time.sleep(wait_time)
         if start_stepsy != 0:
                temp = yloc % gran
                move_extra = start_stepsy
                inputmoveextra = 'x,' + str(move_extra) + ',1'
                ser.write(str.encode(inputmoveextra))
                y_pos -= temp
                date = datetime.datetime.now().timestamp()
                file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , y_pos))
                time.sleep(wait_time)

        move = str(simpledialog.askstring(title="Move again?", prompt= "Would you like to move to
        another position in the horizontal mode? Yes or No"))

    file.close()

if ot == 'V' or ot == 'v':
    #Moving to home (0,0) position
    tkmb.showwarning("Weights", "WARNING: Please make sure the additional weights are NOT hooked on
    the apparatus! You are about to move the collimator down!")
    time.sleep(1)
    ser.write(str.encode("xEND,,1"))
    time.sleep(20)
    ser.write(str.encode("yEND,,1"))
```

```
time.sleep(30)
start = (20/pitch) * steps_per_rev_x
movehome = 'y,' + str(start) + ',0'
ser.write(str.encode(movehome))
file = open(r'sourcepositions.txt', 'w+')
tkmb.showinfo("Vertical Position", "Welcome to the vertical position! You are now in the xz
plane. The collimator will move to the home postion. On the positioning apparatus, we have
labeled these coordinates for easy understanding of the coordinate system! The z-axis is up and
down, the x-axis side to side. \n \nNow you will enter the x and z position that you would like
the collimator to go.")

# Visualization of Coordinate System
py.figure(figsize = (10, 5))
u=0.     #x-position of the center
v=1.5    #y-position of the center
a=3.5   #radius on the x-axis
b=1.5    #radius on the y-axis
x = np.linspace(-20, 20, 100)
py.plot( u+a*np.cos(x) , v+b*np.sin(x) , color = 'gray')
py.axvline(x = 0, color = 'black')
py.fill_between(u+a*np.cos(x) , v+b*np.sin(x) , alpha=1, color ='gray', label = "collimator in
home (0,0) position")
py.ylim(0,20)
py.xlim(-20,20)
py.legend()
py.xlabel("x-axis, with x = 0 in the middle of the apparatus")
py.ylabel("z-axis, with z = 0 at the bottom of the apparatus")
py.title("visualization of the x-z (vertical) plane of movement")
py.show()

#Asking which position on the coordinate system the user wants the collimator to move to
xloc = float(simpledialog.askstring(title="x position", prompt= "Input the x-position that you
would like the collimator to move (between -20 and 20 cm)"))
while (xloc)>20 or (xloc)<-20:
     xloc = float(simpledialog.askstring(title="x position", prompt= "Oops! Please enter a valid
     x position. \nInput the x-position that you would like the collimator to move (between -20
     and 20 cm)"))

zloc = float(simpledialog.askstring(title="z position", prompt= "Input the z-position that you
would like the collimator to move (between 0 and 20 cm)"))
while (zloc)>20 or (zloc)<0:
     zloc = float(simpledialog.askstring(title="z position", prompt= "Oops! Please enter a valid
     z position. \nInput the z-position that you would like the collimator to move (between 0 and
     20 cm)"))

x_pos = 0
z_pos = 0

if (xloc > 0):
    start_stepsx = (xloc/pitch) * steps_per_rev_x
    move_x = (gran/pitch) * steps_per_rev_x
    date = datetime.datetime.now().timestamp()
    file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
    while(start_stepsx >= move_x):
        inputmovex = 'y,' + str(move_x) + ',1'
        ser.write(str.encode(inputmovex))
        start_stepsx -= move_x
        x_pos += gran
        date = datetime.datetime.now().timestamp()
```

```
        file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
        time.sleep(wait_time)
    if start_stepsx != 0:
        temp = xloc % gran
        move_extra = start_stepsx
        inputmoveextra = 'y,' + str(move_extra) + ',1'
        ser.write(str.encode(inputmoveextra))
        x_pos += temp
        date = datetime.datetime.now().timestamp()
        file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
        time.sleep(wait_time)

if (xloc < 0):
    xloc = -xloc
    start_stepsx = (xloc/pitch) * steps_per_rev_x
    date = datetime.datetime.now().timestamp()
    file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
    move_x = (gran/pitch) * steps_per_rev_x
    while(start_stepsx >= move_x):
        inputmovex = 'y,' + str(move_x) + ',0'
        ser.write(str.encode(inputmovex))
        start_stepsx -= move_x
        x_pos -= gran
        date = datetime.datetime.now().timestamp()
        file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
        time.sleep(wait_time)
    if start_stepsx != 0:
        temp = xloc % gran
        move_extra = start_stepsx
        inputmoveextra = 'y,' + str(move_extra) + ',0'
        ser.write(str.encode(inputmoveextra))
        x_pos -= temp
        date = datetime.datetime.now().timestamp()
        file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
        time.sleep(wait_time)

start_stepsz = (zloc/pitch) * steps_per_rev_y
move_z = (gran/pitch) * steps_per_rev_y

if (zloc!=0):
    tkmb.showwarning("Weights", "WARNING: Please make sure the additional weights are hooked on
    the apparatus! You are about to move the collimator up!")

date = datetime.datetime.now().timestamp()
file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
while(start_stepsz >= move_z):
    inputmovez = 'x,' + str(move_z) + ',0'
    ser.write(str.encode(inputmovez))
    start_stepsz -= move_z
    z_pos += gran
    date = datetime.datetime.now().timestamp()
    file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
    time.sleep(wait_time)
if start_stepsz != 0:
    temp = zloc % gran
    move_extra = start_stepsz
    inputmoveextra = 'x,' + str(move_extra) + ',0'
    ser.write(str.encode(inputmoveextra))
    z_pos += temp
```

```
    date = datetime.datetime.now().timestamp()
    file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
    time.sleep(wait_time)

# Asking if the user wants the collimator to move to a new position
move = str(simpledialog.askstring(title="Move again?", prompt= "Would you like to move to another
position in the vertical mode? Yes or No"))
while (move == "yes" or move == "Yes" or move == "yes"):

       xloc = float(simpledialog.askstring(title="x position", prompt= "Input the x-position that
       you would like the collimator to move (between -20 and 20 cm)"))
       while (xloc)>20 or (xloc)<-20:
            xloc = float(simpledialog.askstring(title="x position", prompt= "Oops! Please enter a
            valid x position. \nInput the x-position that you would like the collimator to move
            (between -20 and 20 cm)"))

       zloc = float(simpledialog.askstring(title="z position", prompt= "Input the z-position that
       you would like the collimator to move (between 0 and 20 cm)"))
       while (zloc)>20 or (zloc)<0:
            zloc = float(simpledialog.askstring(title="y position", prompt= "Oops! Please enter a
            valid z position. \nInput the z-position that you would like the collimator to move
            (between 0 and 20 cm)"))

       x_move = xloc - x_pos
       if (x_move > 0):
        date = datetime.datetime.now().timestamp()
        file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
        start_stepsx = (x_move/pitch) * steps_per_rev_x
        while(start_stepsx >= move_x):
            inputmovex = 'y,' + str(move_x) + ',1'
            ser.write(str.encode(inputmovex))
            start_stepsx -= move_x
            x_pos += gran
            date = datetime.datetime.now().timestamp()
            file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
            time.sleep(wait_time)
        if start_stepsx != 0:
            temp = xloc % gran
            move_extra = start_stepsx
            inputmoveextra = 'y,' + str(move_extra) + ',1'
            ser.write(str.encode(inputmoveextra))
            x_pos += temp
            date = datetime.datetime.now().timestamp()
            file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
            time.sleep(wait_time)
       if (x_move < 0):
        x_move = -x_move
        start_stepsx = (x_move/pitch) * steps_per_rev_x
        while(start_stepsx >= move_x):
            date = datetime.datetime.now().timestamp()
            file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
            inputmovex = 'y,' + str(move_x) + ',0'
            ser.write(str.encode(inputmovex))
            start_stepsx -= move_x
            x_pos -= gran
            time.sleep(wait_time)
        if start_stepsx != 0:
            temp = xloc % gran
            move_extra = start_stepsx
```

```
                inputmoveextra = 'y,' + str(move_extra) + ',0'
                ser.write(str.encode(inputmoveextra))
                x_pos -= temp
                date = datetime.datetime.now().timestamp()
                file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
                time.sleep(wait_time)

            z_move = zloc - z_pos

            if (z_move > 0):
             tkmb.showwarning("Weights", "WARNING: Please make sure the additional weights are hooked
             on the apparatus! You are about to move the collimator up!")
             start_stepsz = (z_move/pitch) * steps_per_rev_z
             while(start_stepsz >= move_z):
                 inputmovez = 'x,' + str(move_z) + ',0'
                 ser.write(str.encode(inputmovez))
                 start_stepsz -= move_z
                 z_pos += gran
                 date = datetime.datetime.now().timestamp()
                 file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
                 time.sleep(wait_time)
             if start_stepsz != 0:
                 temp = zloc % gran
                 move_extra = start_stepsz
                 inputmoveextra = 'x,' + str(move_extra) + ',0'
                 ser.write(str.encode(inputmoveextra))
                 z_pos += temp
                 date = datetime.datetime.now().timestamp()
                 file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
                 time.sleep(wait_time)
            if (z_move < 0):
             tkmb.showwarning("Weights", "WARNING: Please make sure the additional weights are NOT
             hooked on the apparatus! You are about to move the collimator down!")
             z_move = -z_move
             start_stepsz = (z_move/pitch) * steps_per_rev_x
             while(start_stepsz >= move_z):
                 inputmovez = 'x,' + str(move_z) + ',1'
                 ser.write(str.encode(inputmovez))
                 start_stepsz -= move_z
                 z_pos -= gran
                 date = datetime.datetime.now().timestamp()
                 file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
                 time.sleep(wait_time)
             if start_stepsz != 0:
                 temp = zloc % gran
                 move_extra = start_stepsz
                 inputmoveextra = 'x' + str(move_extra) + ',1'
                 ser.write(str.encode(inputmoveextra))
                 z_pos -= temp
                 date = datetime.datetime.now().timestamp()
                 file.write("%5.2f %5.2f %5.2f\n" % (date, x_pos , z_pos))
                 time.sleep(wait_time)

            move = str(simpledialog.askstring(title="Move again?", prompt= "Would you like to move to
            another position in the vertical mode? Yes or No"))

    file.close()

if ot == 'T' or ot =='t':
```

```
tkmb.showinfo("Transition Position", "Welcome to the transition position! The collimator will
move to the home (transition) position, and then you may rotate the apparatus. \n\nIf you are
transitioning from the horizontal to vertical position, rotate the apparatus and attach the
counterweights and support blocks in the mount. \n\nIf you are transitioning from the vertical to
horizontal position, remove the counterweights, ropes, and support blocks in the mount and then
rotate the apparatus.\n\nFinally, re-run this cell and pick which plane you would like to move
in! :)")
tkmb.showwarning("Weights", "WARNING: If transitioning from vertical to horizontal please make
sure the additional weights are NOT hooked on the apparatus! You are about to move the collimator
down!")
time.sleep(5)
# Moving to the home (0,0) position
ser.write(str.encode("xEND,,1"))
time.sleep(20)
ser.write(str.encode("yEND,,1"))
time.sleep(30)
start = (20/pitch) * steps_per_rev_x
movehome = 'y,' + str(start) + ',0'
ser.write(str.encode(movehome))
```