v1a
December 16, 2016

2D translation stage quick-guide

Evan Angelico, Andrey Elagin, Henry Frisch, Eric Spieglan
The University of Chicago

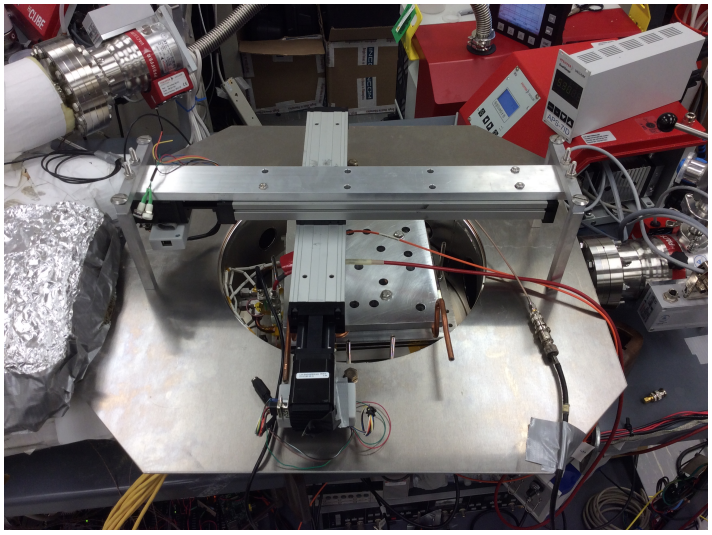

Figure 1: The two translation stages are suspended overhead an LAPPD in production.

1 Connections to the motors

- Log in to margherita@psec2.uchicago.edu. This is the control user for the 2d motor stage.
- Locate yourself into the directory,
  /local/data2/margherita/margherita_run_scripts/run_motors
- Plug USB cables attached to psec2 into the USB485 boxes that are attached to the motors (little white boxes with a USB mini port)
- Two new usb-serial ports will become active on psec2; both should now be mounted as **/dev/ttyUSB[1-2]**, or some other number. If they are not, contact Mary Heintz (maryh@edg.uchicago.edu)
- Make sure that these /dev/ttyUSB ports have permissions that allow Margherita to read and write. In chmod code, this is 666, or crw-rw-rw- .
  - What likely happened is that the permissions are set to crw- - - - - - -. If so, change them using **chmod 666 /dev/ttyUSB1** [1]
- Connect two DC power supplies, one for each motor. The motors take 10-40V supplies, and have a max current of 2 amps (nominal/default running current is 0.6 amps)

[1] You need root access for this. Ask Mary for the password or look on the first page of the Margherita 3 log book

2 Code-base and testing

2.1 Initial test

- Import the test program [2] into python:
  `python -i load2Motors.py`
- Connect to the two motors by loading them into an object variable:
  `> motors = loadMotors()`
- If your serial communications are working properly, this will have printed:
  `Motor communication port has been opened`
  `Motor communication port has been opened`
  `Motors at position:  (<x>, <y>)`
- If instead, you do not get any messages, or the messages hang and do not print the motor position or do not return you to a python command prompt, you may have the following issues:
  - **Motor is not being powered**: the motor must have its power supply plugged in, in order to receive and send serial communications. It is not supplied power by USB. So, there must be both a USB going to the grey USB485 box, and a 10-40V power supply going to the motor.
  - **USB serial port has incorrect permissions**: Usually this results in a permissions error, but make sure to follow the instructions in the first section for correcting the permissions of the motor drivers.
  - Other
- If you were able to load the motors properly, some basic test commands are
  - **Print:** `> print str(motors)`
  - **Move relative (1 cm in x, 0 in y):** `> motors.mr(1, 0)`

2.2 Program descriptions

In the directory **/local/data2/margherita/margherita_run_scripts/run_motors**, you will find a number of python programs that have a few inter dependencies. I have encapsulated the gritty details of the motor driver into this python code and (hopefully) created something usable to the point of you not having to ever look at most of the python code. Their descriptions are below:

- **Motor.py**: (dependent on none) This is a class file that defines the class, Motor. A "Motor" is an object that controls one motor and its basic functions. This class has the lower level serial communications built in to its functions.

[2] see next subsection of program descriptions

- **TwoMotors.py**: (dependent on Motor.py) This is a class file that defines the class, TwoMotors. "TwoMotors" is a way of combining two motor objects into one object that knows about its position in 2d space. The TwoMotors class also implements regulations on the spatial boundaries for both motors.

- **load2Motors.py**: (dependent on TwoMotors.py and Motor.py) This is a program that is written for testing purposes. Usage:
  ```
  python -i load2Motors.py
  > motors = loadMotors()
  > motors.mr(2,3)
  ```
  The first line imports the program functions, the second line loads the two motors into a "TwoMotor" object called "motors", and the third line does a basic function of the two-motors, "move-relative" in the x,y plane.

- **2dScan.py**: (dependent on TwoMotors.py) This program is what you will run in order to do continuous position scanning and position data logging. [3] This is where you will store "configurations", which are sets of (x,y) coordinates for which you want to scan over. The program also allows you to write your own "scanning procedure"; for example, sit→ wait → log → move or log → move → log → move, etc.

3 Miscelaneous notes:

- The motors **reset their absolute position to 0 when they are power cycled**. This is built in to their firmware. So if you have calibrated with absolute positions, manually move to motors to (0,0) before unplugging.

- The motors have a built in firmware stop that does not allow them to go to an absolute position that is less than 0. To bypass this, use (and read) the two functions:
  `TwoMotors.freeZero()` from "TwoMotors.py"
  `Motor.redefineAbsPosition(x)` from "Motor.py"

- If the motors go **haywire**, in other words they start running away out of control (this should not happen, but just in case...) unplug the motor power cable, not the usb connection.

- Find manuals for our model here:
  http://www.linengineering.com/products/integrated_motors/silverpak-17c-2/

[3] The data logging works as follows: there is a "motorLog.txt" data file that stores the position of the motors and the time at which the motors were at that position. During analysis, one can match motor positions/times to photocurrent/times and fold them together to make plots. This plotting code is not covered in this guide