XY Gantry System: Connection, Python Code Guide, and Resources- Shouvik Mondal (University of Utah)

This document provides guidance on connecting your XY gantry system, which uses a Galil DMC-21x2 series motion controller (communicating via a USB-to-Serial adapter), to various operating systems. It also offers a detailed explanation of the provided Python script (gantry_simple_loop_script) used for controlling the system and lists useful resources.

Part 1: Connecting the XY System (Galil Controller via USB-to-Serial)

Galil motion controllers like the DMC-21x2 often use an RS-232 serial port for communication. To connect this to modern computers, a USB-to-Serial adapter is typically used. The computer will see this adapter as a virtual COM port (Windows) or a TTY device (macOS/Linux).

General Principles:

- **USB-to-Serial Adapter Drivers:** The adapter requires a driver to function. Common chipsets for these adapters include FTDI, Prolific, CH340/CH341, or SiLabs. The operating system needs the correct driver for the specific chipset in your adapter.
- **Galil Software (GDK):** It's highly recommended to first establish communication using Galil's official software, like Galil Design Kit (GDK), available from Galil or your system supplier (Newmark Systems). This helps verify the physical connection, port settings, and basic controller functionality before scripting. The "NSC-G Getting Started.pdf" mentions downloading GDK.
- **Baud Rate:** The Galil DMC-21x2 typically defaults to a baud rate of **19200**. Both the controller and the communication software on your computer must be set to this same baud rate.

Connecting on Windows

1. **Install USB-to-Serial Adapter Driver:**
    - When you plug in the USB-to-Serial adapter, Windows might try to install a driver automatically.
    - If not, or if it fails, you'll need to identify the chipset of your adapter (often written on the adapter or found in its documentation) and download the driver from the chipset manufacturer's website (FTDI, Prolific, WCH for CH340, etc.) or from Galil/Newmark if they provided the adapter.

    - Install the driver following the manufacturer's instructions.
2. **Identify the COM Port:**
    - Once the driver is installed and the adapter is plugged in, open **Device Manager** (you can search for it in the Start Menu).
    - Look under the "Ports (COM & LPT)" section. You should see your USB-to-Serial adapter listed with an assigned COM port number (e.g., COM3, COM4, etc.). Note this COM port number.
    - If you see a yellow exclamation mark next to the device, there's a driver issue.
3. **Test with GDK (or other terminal software):**
    - Open Galil Design Kit (GDK).
    - In the GDK Manager, find your controller (it should appear under the identified COM port).
    - Connect to it, ensuring the baud rate is set to 19200.
    - Open the GDK Terminal. You should be able to send a simple command like MG _GN (which returns the Galil version string) or TC1 (Tell Code, should return "0 No error" if all is well). If you get a response (often starting with a colon :), the connection is working.
4. **Python Script:**
    - In your Python script, set SERIAL_PORT to the COM port number you found (e.g., SERIAL_PORT = 'COM3').

Connecting on macOS

1. **Install USB-to-Serial Adapter Driver (if needed):**
    - macOS has built-in drivers for many common FTDI and Prolific chipsets.
    - For some chipsets (especially CH340/CH341), you may need to download and install a driver from the chipset manufacturer. Ensure the driver is compatible with your macOS version and architecture (Intel/Apple Silicon). A system restart might be required.
2. **Identify the TTY Device:**
    - Open **Terminal** (Applications > Utilities).
    - With the USB-to-Serial adapter plugged in, type the command: ls /dev/tty.*
    - Look for entries like /dev/tty.usbserial-XXXX (for FTDI-like devices) or /dev/tty.wchusbserialXXXX (for CH340 devices) or similar. Note this full path.
3. **Test with GDK (or other terminal software):**
    - Galil provides GDK for macOS.
    - Alternatively, you can use terminal programs like screen or minicom (installable via Homebrew), or GUI-based serial terminals like "Serial Tools".
    - When using these, you'll need to specify the device path (e.g., /dev/tty.usbserial-XXXX) and the baud rate (19200). Example with screen:

screen /dev/tty.usbserial-XXXX 19200
    - Send MG _GN or TC1 to test.
4. **Python Script:**
    - In your Python script, set SERIAL_PORT to the full device path (e.g., SERIAL_PORT = '/dev/tty.usbserial-XXXX').

Connecting on Linux (Ubuntu - as per your setup)

1. **Driver Installation (Usually Automatic):**
    - Most common USB-to-Serial adapter chipsets (FTDI, CH340/CH341, Prolific PL2303) are supported by drivers built into the Linux kernel. Usually, no separate driver installation is needed.
2. **Identify the TTY Device:**
    - Plug in your USB-to-Serial adapter.
    - Open a **Terminal**.
    - Type dmesg | tail. This will show the latest kernel messages, and you should see your adapter being recognized and assigned a device name like /dev/ttyUSB0, /dev/ttyUSB1, etc. (Your logs show /dev/ttyUSB3).
    - Alternatively, you can list devices with ls /dev/ttyUSB*.
3. **Permissions:**
    - By default, your user might not have permission to access serial ports directly. You usually need to add your user to the dialout group (or sometimes tty group).
    - In the terminal, run: sudo usermod -a -G dialout $USER
        - (Replace $USER with your actual username if needed, though $USER usually works).
    - You will need to **log out and log back in** for this group change to take effect. Some systems might require a full reboot.
4. **Test with GDK (or other terminal software):**
    - Galil provides GDK for Linux.
    - Alternatively, use terminal programs like minicom (install via sudo apt install minicom), picocom, or screen.
    - Example with minicom:
        - sudo minicom -s to configure it first (set serial device to /dev/ttyUSB3, baud rate to 19200, turn off hardware/software flow control). Save the configuration as default (dfl).
        - Then run minicom to connect.
    - Send MG _GN or TC1 to test.
5. **Python Script:**
    - In your Python script, set SERIAL_PORT to the device path (e.g., SERIAL_PORT

= '/dev/ttyUSB3').

Part 2: Python Code Explanation (gantry_simple_loop_script)

This section explains the Python script currently in the immersive artifact gantry_simple_loop_script, which is designed to home your XY gantry to its reverse limits and then move through a sequence of predefined X and Y positions.

```
import gclib
import time
```

- import gclib: Imports the Galil gclib library, enabling Python to communicate with the Galil motion controller.
- import time: Imports Python's standard time library, used for creating pauses (time.sleep()) in the script.

Configuration Variables (# --- Configuration ---)

```
SERIAL_PORT = '/dev/ttyUSB3'
BAUD_RATE = 19200

SPEED_XY_COUNTS_PER_SEC = 20000
ACCEL_XY_COUNTS_PER_SEC_SQ = 10000
DECEL_XY_COUNTS_PER_SEC_SQ = 10000
HOMING_SPEED_COUNTS_PER_SEC = 5000
MOVE_INTERVAL_SECONDS = 1.0
RECOVERY_DISTANCE_FORWARD = 1000
RECOVERY_DISTANCE_REVERSE = -1000
```

- SERIAL_PORT, BAUD_RATE: Define the connection parameters for the serial port.
- SPEED_XY_COUNTS_PER_SEC: Target speed for normal X and Y moves (Galil SP command).
- ACCEL_XY_COUNTS_PER_SEC_SQ: Acceleration for moves (Galil AC command).
- DECEL_XY_COUNTS_PER_SEC_SQ: Deceleration for moves (Galil DC command).
- HOMING_SPEED_COUNTS_PER_SEC: A typically slower speed used for the homing routine (Galil JG command).
- MOVE_INTERVAL_SECONDS: The pause duration after each X and Y move.
- RECOVERY_DISTANCE_FORWARD, RECOVERY_DISTANCE_REVERSE: Distances (in counts) used if the script needs to attempt to move an axis off an already active

limit switch during the homing pre-check.

```
X_AXIS = 'A'
Y_AXIS = 'B'
```

- Maps the script's concept of 'X' and 'Y' to the Galil controller's axis designations (A and B).

```
X_POSITIONS = [1000, 2000, ..., 10000] # Example values
Y_POSITIONS = [1000, 2000, ..., 5000]  # Example values
```

- Lists of target absolute positions (in motor counts) for the X and Y axes. These positions are relative to the '0' point established during the homing routine.

```
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# CRITICAL SETTING: Adjust CN_SETTING_IS_ACTIVE_HIGH based on your hardware
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
CN_SETTING_IS_ACTIVE_HIGH = True # YOU MUST VERIFY AND SET THIS!
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

- CN_SETTING_IS_ACTIVE_HIGH: A crucial boolean.
  - If True, the script configures the controller (CN=1) to consider limit switches **active when their input signal is HIGH**. This is common for Normally Closed (NC) switches that open upon contact (and have a pull-up resistor).
  - If False, the script configures (CN=-1) for **active-LOW** limits. This is common for Normally Open (NO) switches that close to ground upon contact, or NC switches wired to provide a LOW signal when *not* pressed.
  - **Incorrect setting here will lead to misinterpretation of limit switch states and likely motion failures.**

Helper Function: get_limit_status_for_homing()

```
def get_limit_status_for_homing(g, axis, cn_is_active_high):
    # ... (implementation details as in the script) ...
    return fwd_active, rev_active, ts_val
```

- **Purpose:** Queries the controller for the status of an axis's limit switches and interprets the result based on the cn_is_active_high setting.
- Uses g.GCommand(f"TS{axis}") to get the raw status value.

- Performs bitwise operations (>>, &) to isolate the Forward Limit Switch (FLS - bit 3) and Reverse Limit Switch (RLS - bit 2) states.
- **Returns:** A tuple (is_fwd_limit_active, is_rev_limit_active, raw_ts_value).

Homing Function: home_axis_to_reverse_limit()

```
def home_axis_to_reverse_limit(g, axis, axis_name, cn_is_active_high):
  # ... (implementation details as in the script) ...
  return True # or False if homing fails
```

- **Purpose:** Moves the specified axis towards its REVERSE limit switch, detects activation, stops, moves slightly off, and defines this position as '0' (DP{axis}=0).
- **Logic Flow:**
  1. **Initial Check:** Checks if the RLS (or FLS) is already active.
  2. **Pre-Recovery:** If RLS is active (and FLS is not also active, which would be a critical fault), it attempts a small forward move to clear the RLS.
  3. **Jog Towards Limit:** Sets a jog speed (JG command, negative for reverse) and begins motion (BG).
  4. **Poll for Limit:** Continuously checks TS status until the RLS is detected or a timeout occurs. Also checks for accidental FLS activation.
  5. **Stop and Define Position:** Stops motion (ST, AM), moves a small distance forward (PR) to ensure it's not on the switch, then defines the current position as 0 (DP{axis}=0).
  6. **Returns True** on success, **False** on failure.

Main Function: main()

1. **Initialization & Connection:** Creates gclib object, sets up try...except...finally, connects to the controller (g.GOpen()), and gets controller info (g.GInfo()).
2. **Initial Setup:**
   - Enables motors (SH command).
   - Sets limit switch polarity (CN command) based on CN_SETTING_IS_ACTIVE_HIGH.
   - Checks for errors after CN using TC1.
   - Disables software limits (FL, BL commands).
3. **Homing Sequence:**
   - Calls home_axis_to_reverse_limit() for the X-axis, then for the Y-axis. Exits if homing fails for either.
   - Prints final positions and limit status after homing.
4. **Set Motion Profile:** Sets SP, AC, DC for the main sequence of moves.

5. **Sequential Move Loops:**
    - **Outer for loop (X-axis):** Iterates through X_POSITIONS.
        - Includes a pre-move check to see if an X-limit would prevent the move.
        - Commands X-axis to target (PA{X_AXIS}=...), begins motion (BG{X_AXIS}), waits for completion (AM{X_AXIS}).
        - Pauses for MOVE_INTERVAL_SECONDS.
    - **Inner for loop (Y-axis):** For each X position, iterates through Y_POSITIONS.
        - Includes a pre-move check for Y-limits.
        - Commands Y-axis to target (PA{Y_AXIS}=...), begins motion (BG{Y_AXIS}), waits for completion (AM{Y_AXIS}).
        - Pauses for MOVE_INTERVAL_SECONDS.
6. **Error Handling & Cleanup:** Catches errors, tries to get TC1 for details, and ensures the connection is closed (g.GClose()) in the finally block.

How it Achieves "Start from Absolute Zero":
The home_axis_to_reverse_limit() function is key. It physically moves each axis to its reverse limit switch. After moving slightly off this switch, it executes g.GCommand(f"DP{axis}=0"). This DP (Define Position) command tells the Galil controller: "The current physical location of this axis is now the logical position 0." All subsequent PA (Position Absolute) commands in X_POSITIONS and Y_POSITIONS are then relative to this (0,0) origin established at the reverse limits.

Part 3: Useful Websites and Resources

Here are some websites and resources that can be helpful when working with your Galil controller, gclib, and related technologies:

1. **Galil Motion Control (Manufacturer of your DMC-21x2):**
    - **Main Website:** https://www.galil.com
        - This is the primary source for product information, datasheets, manuals, software downloads (like GDK - Galil Design Kit), and support.
    - **Software Tools (GDK):** https://www.galil.com/downloads/software-tools
        - GDK is invaluable for initial setup, testing, and troubleshooting communication with your controller.
    - **Manuals:** https://www.galil.com/downloads/manuals
        - Look for the "DMC-21x2/21x3 User Manual" and the "Command Reference" (often a general one for the controller family, like the "Optima/Econo DMC-2xxx Series Command Reference" you have - manc2xxx.pdf). These are essential for understanding all controller commands and features.
    - **gclib Information/Download:** https://www.galil.com/sw/pub/all/doc/gclib/html/python.html (or search for

gclib python on their site). This is the official documentation and source for gclib. The PDF you have (gclib.pdf) is likely the documentation for this library.
    - **Support/Contact:** https://www.galil.com/contact
2. **Newmark Systems (Your System Supplier):**
    - **Main Website:** http://www.newmarksystems.com
    - As seen in your "NSC-G Getting Started.pdf", they provide specific setup instructions and may host relevant software or drivers for the systems they integrate. Check their support or downloads section for your specific gantry model.
3. **USB-to-Serial Adapter Chipset Manufacturers (for Drivers):**
    - **FTDI (Future Technology Devices International):** https://ftdichip.com/drivers/
        - Very common chipset. Their VCP (Virtual COM Port) drivers are widely used.
    - **Prolific Technology Inc.:** http://www.prolific.com.tw/US/ShowProduct.aspx?p_id=225&pcid=41 (for PL2303 chipset, check for your specific model)
        - Another common chipset. Be cautious with counterfeit Prolific chips; official drivers may not work with them.
    - **WCH (Jiangsu Qin Heng Co., Ltd. - for CH340/CH341):** http://www.wch-ic.com/downloads/CH341SER_ZIP.html (Windows) or search for "CH340 CH341 mac driver" or "CH340 CH341 linux driver".
        - Often found in lower-cost adapters.
    - **Silicon Labs (SiLabs):** https://www.silabs.com/interface/usb-to-uart-bridge-vcp-drivers (for CP210x chipsets)
        - Another reputable chipset manufacturer.
4. **Python Resources:**
    - **Official Python Website:** https://www.python.org
        - Downloads, documentation, tutorials.
    - **Python Package Index (PyPI):** https://pypi.org
        - Where you can find and install Python libraries (though gclib is typically provided directly by Galil).
    - **Stack Overflow:** https://stackoverflow.com
        - A vast Q&A site for programming questions, including Python and hardware interfacing.
5. **Linux Serial Port and Permissions:**
    - General Linux documentation or forums for your specific distribution (e.g., Ubuntu forums) can be helpful if you encounter issues with serial port access

or permissions beyond the dialout group.

Tips for Finding Information:

- When searching for drivers, always try to identify the exact chipset on your USB-to-Serial adapter.
- For Galil-specific questions, their official documentation and support channels are the best starting points.
- The PDF manuals you have (man21x2.pdf, manc2xxx.pdf, gclib.pdf) are your primary references for controller commands and gclib usage. Keep them handy!

This list should provide you with good starting points for further research, driver downloads, and understanding the technologies involved.

Part 4: What not to do :)

1. Don’t connect the motor controller to the wrong port(it might crash the PC!) You can connect the controller via RS-232 to the USB port, ethernet cable(but can’t access the internet in that case :(
2. If the code fails, try to re-run(i need to fix some bugs, it still works well!)
3. Don’t update the linux or download any un-necessary softwares, it already crashed 3-times :(..... if that happens again contact Benjamin from the help-desk(?) of our department.
4. Don't run the motor via GDK terminal and python script simultaneously!!
5. Check if the motor controller is on or not everytime before running any code or GDK terminal! AND Turn off the controller after use!!

If you run into a problem, don't forget to email(shouvik.mondal@utah.edu) or message me in slack(Shouvik Mondal).