Goniometer stepper driver manual

(last update: July 16, 2021)

Author:
Sébastien Manigand (sebastien.manigand83@gmail.com)

Centre for Terrestrial and Planetary exploration (C-TAPE), University of Winnipeg
515, Portage Avenue, Winnipeg R3B 2W9 MB, Canada

Contents

1 Introduction

This document describes the stepper driver system designed for the development of a goniometer instrument. The system control three steppers for the incidence and emergence arms and the sample plate positioning.

The control software is divided into two parts. The first one, called the **client**, is installed on a Raspberry Pi which manages the tensions to apply to the steppers in order to move them of the proper number of steps and in the designated direction. The second part, called the **server**, is installed on the main computer unit, which should host all the other components of the goniometer such as the spectrometer controller and the data managing system, what ever it is (database or file folders).

The communication between the server and the client is made through USB serial. In practice the command are sent by the server to the client in the form a chain of characters. The client interprets the command and execute the corresponding movement of steppers.

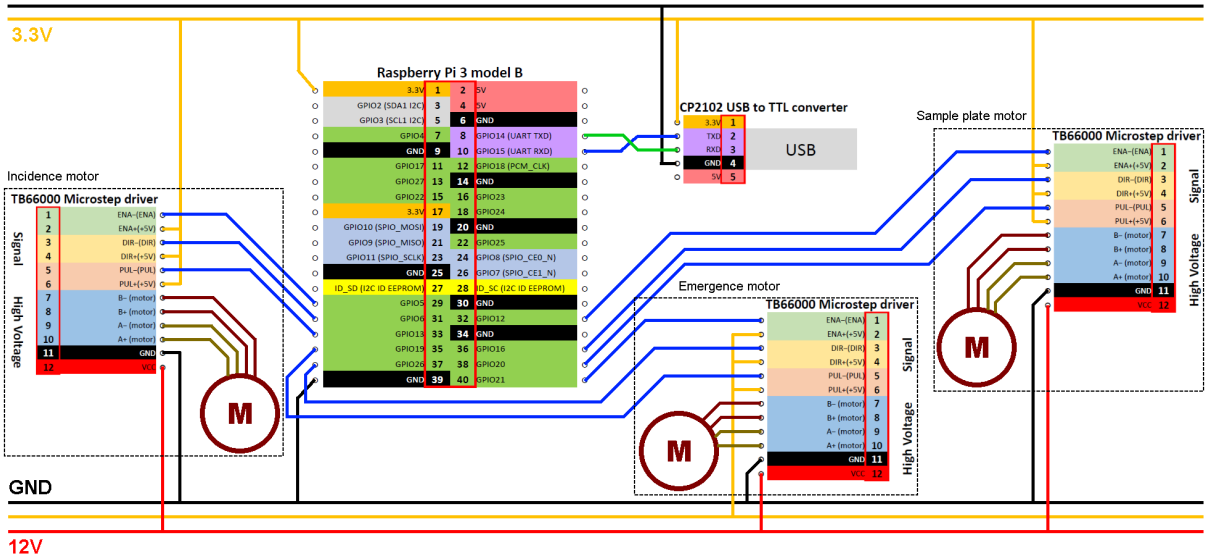

Figure 1: Wiring diagram of the stepper driver system. The colors of the wires are only used to distinguish the different kind of connections (command, stepper, power, ground, and so on). The circled M symbols correspond to the stepper motors.

2 Wiring

The hardware consists of three stepper motors type NEMA17, their respective microstep driver TB66000 which is used as power feeding, and the main controller board Raspberry Pi 3 model B. The USB serial communication is managed by the CP2102 USB to TTL converter ship, linked to the TTL ports of the Raspberry.

There are two power sources for the entire system, plus the Raspberry power. A 12V power sources is used for the three microstep drivers, which in turn power the steppers. This source can be provided by an AC adapter, as those used for phones or laptop while it delivers 12V with the proper amount of current. The other power source of 3.3V is provided by the Raspberry. This source is only used for low current needs, such as powering the ships and setting commands entries to an UP state. Do not plug something that requires a lot of current or it could damage the Raspberry.

The microsteps are linked to the main board through GPIO ports and are controlled in BOARD mode, in the software. The port of the microstep linked to the Raspberry are the one at 0V (the + are connected to the 3.3V power line) to avoid any back current on the board that could damage it with time. The state of the control port is not inverted though, i.e. 0V (LOW) and 3.3V (HIGH) correspond to DISABLED and ENABLED states, respectively.

The Raspberry Pi port for the serial communication are annotated UART RXD ("receiver") and TXD ("transmitter") and are linked to the corresponding port of the CP2102 ship, the transmitter of the one to the receiver of the other and the other way around. Notice that the communication can be from the main computer to the Raspberry and the other way around, the last communication mode could be used to send the state of the motors or an acknowledgement signals to the main computer. For now it is not used like that.

Last but not least, the ground (GND) of the different power lines and one of the Raspberry GND ports are linked together to unify the ground level. Figure 1 shows the wiring diagram of the entire system and the board port that will be used in the client software.

3 Server part

The server software **test_send-cmd.py** is not the definitive controller part of the instrument. It consists of a simple sandbox to test if the command are correctly sent to the client and well executed by the stepper motors.

The program simply set the serial communication and wait for the user to write in the shell the command to send to the motors. These commands are chains of characters corresponds to different action to execute by the client software and the motors.

4 Client part

The client software consists of two python scripts: **stepper.py** and **stepper-benchmark.py**. The first script manages all that concerns the GPIO port handling and the Raspberry. The second one sets the USB communication with the main computer and interprets the command to execute using the first script.

To test if the stepper are well wires and if the they are well functioning, you can execute the first script stepper.py. It will execute a few movement of each steppers (move +30 degrees, wait 1 second, move back -30 degrees).

To turn on the client, you have to execute the second script stepper-benchmark.py. From that point, the script is waiting for commands from the USB communication.

5 serial communication

The serial communication is set on both server and client softwares, with the same com parameters:

- baud rate: 9600,
- byte size: 8 bits,
- parity: None,
- stop bits: 1.

The timeout, time after what the communication is considered to have failed, is arbitrary set to 0.25 second.

The default name of the serial port on the Raspberry is '/dev/ttyS0'. The serial port to set-up on the computer should be named 'COM22', which is the common name for this kind of serial communication. In fact, this is the set-up found in the majority of the tutorials around.

6 command syntax

As described before, the command are (for now) written by the user on server software on the main computer. Then they are interpreted and executed by the client software on the Raspberry. To send a command, the user have to write it in the server software and press enter. If the syntax is correct, a message of success is displayed: "cmd: '<command>' successfully sent!". An error message is displayed if the communication failed to send the message. If the syntax is incorrect, the error message "syntax error..." is displayed on the client software.

The syntax of the command is described in the following table:

Table 1: List of the command

Command	**Description**
`get_inc`	Returns the incidence angle of the corresponding motor, as it is at the time of the command. The angle returned is the angle of the incidence arm of the goniometer.
`get_eme`	Returns the emergence angle of the corresponding motor, as it is at the time of the command. The angle returned is the angle of the emergence arm of the goniometer.
`get_sam`	Returns the sample plate angle of the corresponding motor, as it is at the time of the command. The angle returned is the angle of the sample plate of the goniometer.
`goto_inc(<FLOAT>)`	Move the incidence arm to the angle position indicated by the <FLOAT> number. The number can be an integer, or a float value.
`goto_eme(<FLOAT>)`	Move the emergence arm to the angle position indicated by the <FLOAT> number. The number can be an integer, or a float value.
`goto_sam(<FLOAT>)`	Move the sample plate to the angle position indicated by the <FLOAT> number. The number can be an integer, or a float value.
`q` or `quit` or `exit`	Close the server software, terminate the communication and close properly the Raspberry controller.