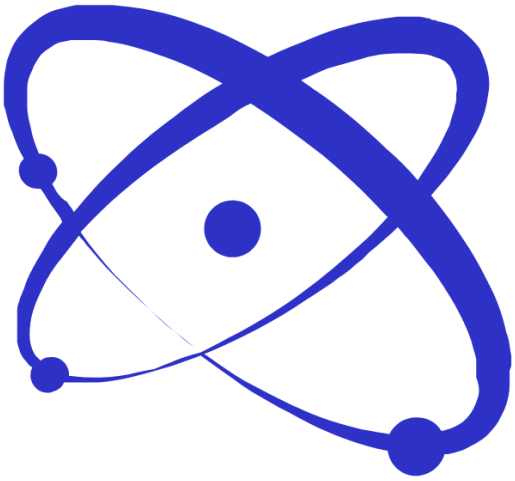
Progress Report



CCTVal

CENTRO CIENTÍFICO
TECNOLÓGICO
DE VALPARAÍSO

November 21, 2022

Revision 3.0

1 Previous state

Control of relative movements of the stepper motor is already achieved. Along with the motor rotation, the voltage on the Arduino Portenta pin `A0` is measured (a measurement for each *step* of rotation). The movement of the motor has an acceleration and deceleration period.

1.1 Wiring diagram

Fig. 1 shows the schematic for the interconnections between the Arduino Portenta board and the stepper motor. `GPIO` 1,2 and 3 are pins of the Arduino Portenta board.

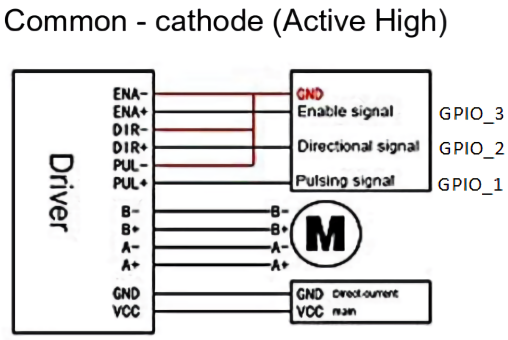
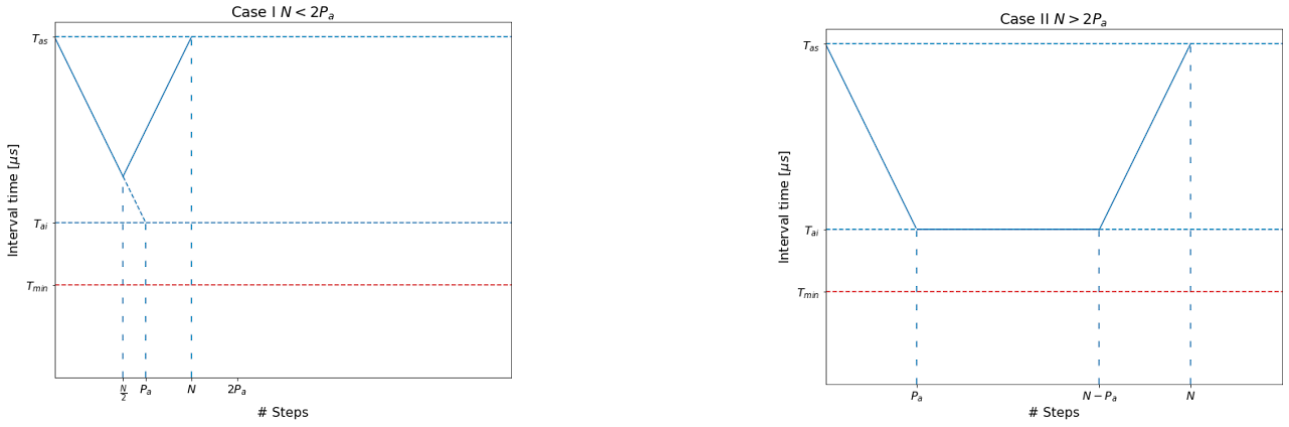


Fig. 1: Connections between stepper motor (**M**), microstep driver, and Arduino Portenta board.

1.2 Acceleration and deceleration

In order to have a smooth transition, an asynchronous method is implemented for accelerating and decelerating the stepper motor. Fig. 2 shows two cases considered for this purpose. When the number of steps that the motor has to rotate N is less than two times the period of acceleration (deceleration) P_a (in steps), the stepper motor doesn't reach its maximum rotation speed, given by the time interval between consecutive steps T_{ai} (see Fig. 2a). If $N > 2P_a$, the motor reaches a constant rotation speed (see Fig. 2b). Time interval T_{as} is related to the initial rotation speed, and $T_{min} = 500\mu$s denotes an empirical minimum for T_{ai} (maximum velocity) for the motor to have a stable behaviour (for 1.8deg/step). However, for higher resolutions (such as 0.05625 deg/step), T_{min} can be as low as 0μs, so it doesn't add any extra delays to the movement of each step. See Table 1 for measurements for each resolution.



(a) Case I: $N < 2P_a$

(b) Case II: $N > 2P_a$

Fig. 2: Cases implemented in Arduino Portenta board to control acceleration and deceleration of stepper motor.

1.3 A/D pin reading

The voltage acquired at analog pin `A0` is converted to $2^b = 2^{10} = 1024$ discrete integer values. Eq. 1 is used to retrieve the analog voltage values V_{in}.

$$V_{in} = \frac{3.3\text{V}}{2^{10} - 1} = \frac{3.3\text{V}}{1023} \tag{1}$$

2 Current state

The code is adapted so that the Arduino Portenta board can receive instructions and send measurements to a computer via the serial port. Fig. 3 shows a flowchart of the setup, where the computer sends an instruction to the Arduino to move a certain 'X' ammount in a certain direction 'r' (or 'l'), which is passed on to the motor driver. In the meantime, the 'A0' analog pin value is read and this collected data is sent back to the computer. More details of each stage of this program are shown in the next section.

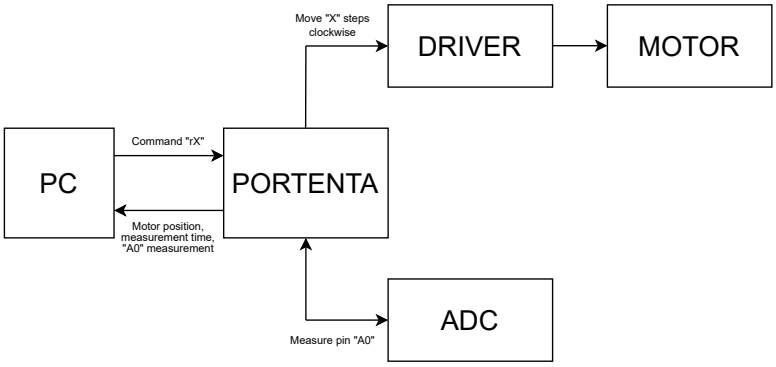


Fig. 3: Program flowchart.

2.1 Python code

The stepper motor driver allows resolutions from 1.8 deg/step (200 step/rev) to 0.05625 deg/step (6400 step/rev). Currently, the lowest resolution of 200 steps per revolution is used as a proof of concept. It is possible to increase the resolution by adjusting the motor driver connections.

The code written in Python allows instructions to be sent to the Arduino Portenta board using a serial port. The instructions sent have the format *rX*, where *X* represents a number and *r* a letter. The first character considers three cases: *a* refers to an absolute movement to a specified position *X* (in steps) (Figs. 5, 6 and 7). *r* indicates a relative movement of steps to the right, negative direction, or clockwise (Fig. 4). *l* denotes a relative movement of steps to the left, positive direction, or counterclockwise.

Each time an instruction is sent to move the stepper motor, the voltage at pin `A0` is measured at the Arduino Portenta board and displayed on screen. This data is stored in a CSV file preceded by additional information about the measurement, such as the measurement log number, date, average reading/writing time (voltage reading on pin `A0` and sending data through the serial port to the computer), average step time (reading/writing time plus movement time) and position (measured as steps, from 0 to N_{max}) at which the stepper motor is at the end of the movement. N_{max} is the number of steps per revolution. For example, a recorded CSV file looks as follows:

```
1,27 October 2022,10:55:33,227us,333us,angle 0,0.85,0.93,1.01,1.08,1.17,1.24,1.31,...
2,27 October 2022,10:57:13,221us,428us,angle 0,0.00,0.02,0.04,0.06,0.11,0.15,0.21,...
```

The voltage reading on pin `A0` shows a sinusoidal voltage from 0V to 3.1V generated by a RIGOL DG4102 signal generator.

2.2 Time measurements

To obtain the execution time of the program, two time measurements are made: reading/writing average time and the average step time. The former only measures the time taken to read the analog pin and send the data to the computer through the serial port, while the latter also measures the time taken to move the motor a step. As explained in section 1.2 of this document, a time delay between successive motor movements (parameter T_{ai}) is used to avoid malfunctioning, but it also increases the total movement time. A series of experiments were conducted

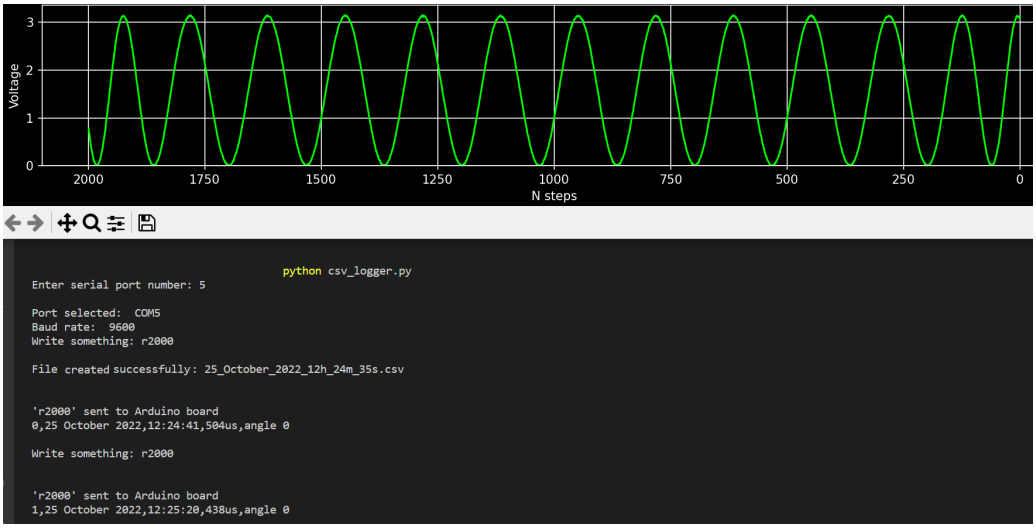


Fig. 4: Relative rotation of 2000 steps, counterclockwise: *'r2000'*

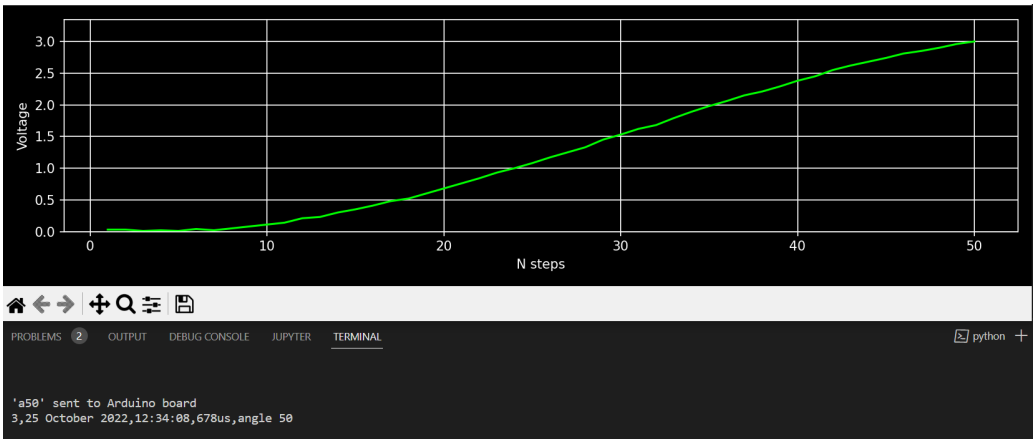


Fig. 5: Absolute rotation to step 50: *'a50'*.

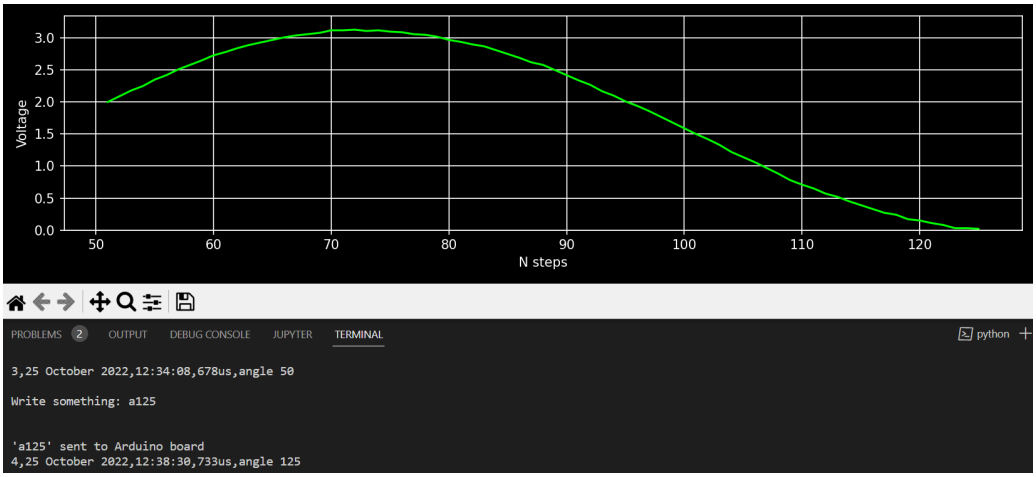


Fig. 6: Absolute rotation to step 125: *'a125'*.

to study the time behaviour while changing the step resolution of the motor. For every resolution the motor is instructed to move 10 revolutions, while for some cases the acceleration parameters are also changed. The results of this study are shown on Table 1, where the obtained average times are displayed as a function of the used resolution and parameters.

Note that for resolutions higher than 800 step/rev no acceleration change is used ($P_a = T_{ai} = T_{as} = 0$), since

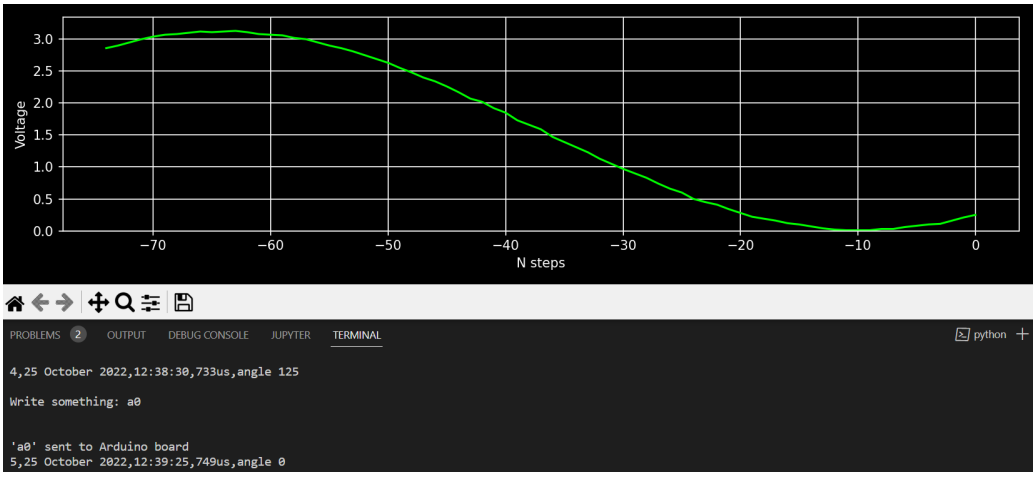


Fig. 7: Absolute rotation to step 0: *'a0'*.

Step resolution (step/rev)	Ang. resolution (deg/step)	P_a	T_{ai}	T_{as}	Av. r/w time (μs)	Av. step time (μs)	Rev. time (s)	Movement
6400	0.05625	0	0	0	243	250	1.6	✓
3200	0.1125	0	0	0	242	249	0.7968	✓
1600	0.225	0	0	0	242	249	0.3984	✓
800	0.45	0	0	0	242	249	0.1992	✓
800	0.45	400	0	400	241	259	0.2072	✓ (1)
400	0.9	0	0	0	-	-	-	✗ (2)
400	0.9	100	100	200	-	-	-	✗
400	0.9	100	200	400	221	429	0.1716	✓ (3)
400	0.9	200	200	300	221	428	0.1712	✓
200	1.8	100	300	400	-	-	-	✗ (4)
200	1.8	100	450	600	224	905	0.181	✗ (5)
200	1.8	200	500	700	239	1015	0.203	✓ (6)
200	1.8	100	600	800	224	1206	0.2412	✓ (7)

Table 1: Time measurements comparisons. (Ang. stands for angular, Av. for average, r/t for read/write and Rev. for revolution).

the movement is rather slow (revolution time: $\sim$ 0.4s to 1.2s). However, an acceleration change for a resolution of 800 step/rev is tested to measure how this affects the total time, resulting in a rather small growth compared to the same resolution but without acceleration change (1). An interesting result is that for a resolution of 400 step/rev and $T_{ai} = 0$ or $T_{ai} = 100$ the motor doesn't move (2,3). This proves the need for the time delay induced to avoid malfunctioning, but is only needed on lower resolutions (400 and 200 step/rev). Note that in (3), for a resolution of 400 step/rev, the motor does move with $T_{ai} = 200$, so it can be concluded that each resolution must have a minimum delay value in order to function correctly (however, as mentioned before, high resolution values don't need this time delay because their movement is slow). For a 200 step/rev resolution the minimum T_{ai} value is 500, as seen in (4,5,6). In (7) a bigger T_{ai} value is tested, only increasing the revolution time by a slight amount.

3 Graphical User Interface

A Graphical User Interface is designed to run the program previously described. A current mock-up for this GUI is shown in Fig. 8, whose caption is clickable and links to a video of the GUI. This GUI consists of four basic *widgets*, which are framed in Fig. 9 and are described below.

3.1 Connection widget

This widget is dedicated to establish the connection with the Portenta board via a serial port, and consists of one combo box, two push buttons, and a label. This widget is framed in red in Fig. 9.

3.1.1 Available ports combo box

The leftmost combo box displays a list of the serial ports available.

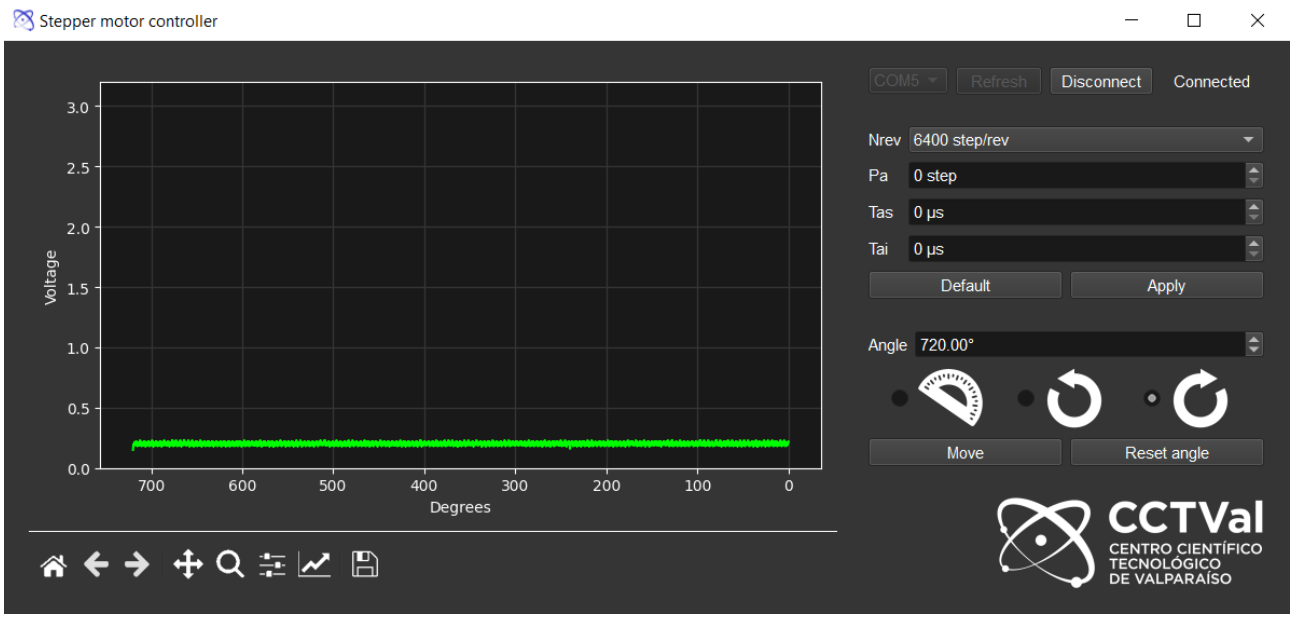


Fig. 8: Graphical User Interface mock-up.

3.1.2 Refresh push button

This button is used to refresh the list of available serial ports (e.g. when a new USB connector is plugged in).

3.1.3 Connect/Disconnect push button

The *Connect* button triggers the dispatch of a message with initialisation parameters to the Portenta board. If the board responds with an acknowledgement message, the connection is then established successfully, the name of the button is changed to *Disconnect*, and the other widgets are enabled. If the board does not respond, an error window is displayed (see Fig. 10a). The **Disconnect** button sets the variable related to the port to `None`, and the interaction with the other widgets is disabled.

3.2 Parameters widget

This widget is dedicated to set the resolution and acceleration parameters described in Section 1.2, and consists of one combo box, three spin boxes and two push buttons. This widget is framed in green in Fig. 9.

3.2.1 Resolution combo box

The resolution parameter (Nrev) sets the number of steps per revolution of the stepper motor. This parameter must match the resolution set at the stepper motor driver for the conversion to sexagesimal degrees to be accurate.

3.2.2 Acceleration parameters spin boxes

These spin boxes set the period of acceleration (deceleration) P_a, and time intervals T_{as} and T_{ai} described in Section 1.2.

3.2.3 Default push button

Pressing the *Default* button sets the acceleration parameters to the optimum for maximum speed, as calculated empirically in Table 1.

3.2.4 Apply push button

Pressing the *Apply* button sends the chosen parameters to the Portenta board via the serial port selected in the Connection widget. If the board responds with an acknowledgement message, an information window pops up (see Fig. 10b).

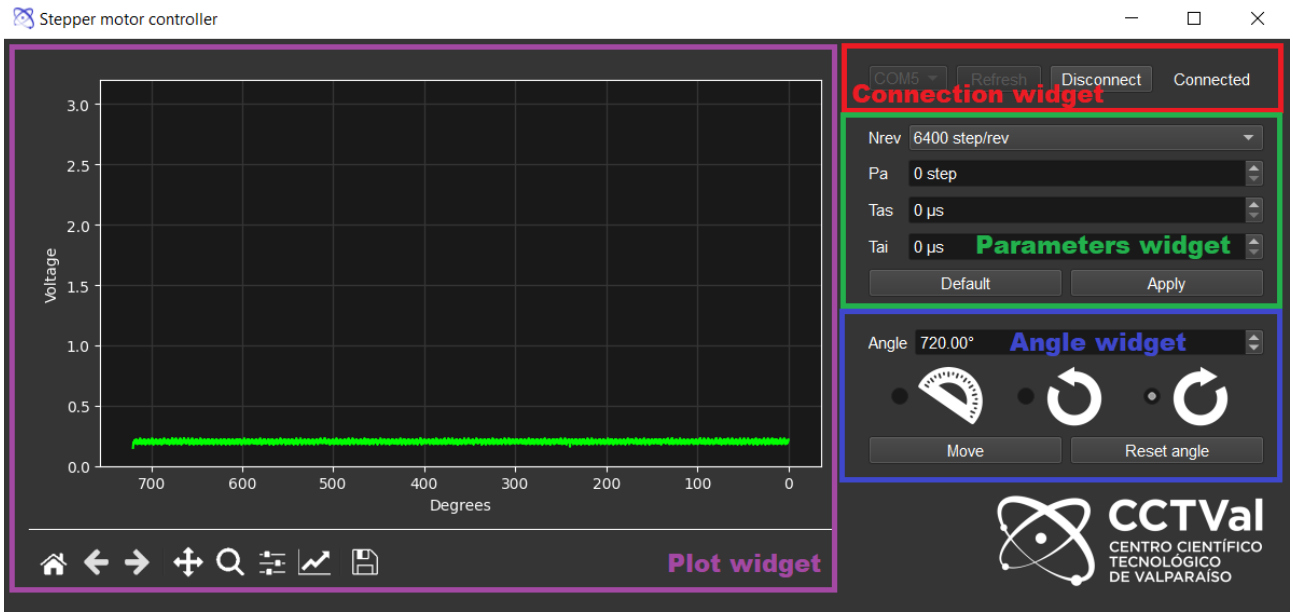
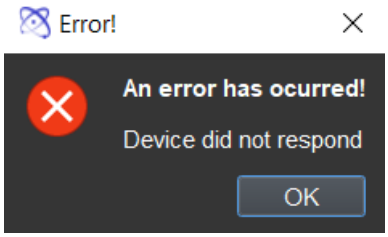
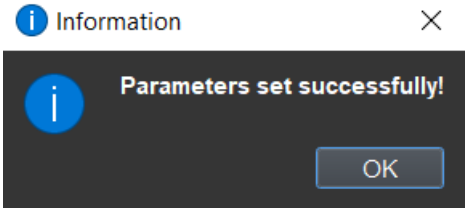


Fig. 9: Basic GUI widgets.



(a) Error window.



(b) Information window.

Fig. 10: Error and Information windows.

3.3 Angle widget

This widget is assigned to set the angle of movement for the stepper motor, and consists of one spin box, three radio buttons and two push buttons. This widget is framed in blue in Fig. 9.

3.3.1 Angle spin box and radio buttons

From left to right, the radio buttons determine whether the angle selected at the *Angle* spin box dictates an **absolute** movement, a relative rotation **counterclockwise** or a relative rotation **clockwise**.

3.3.2 Move push button

Once the *Move* button is clicked, a command is sent to the Portenta board in order to move the stepper motor.

3.3.3 Reset angle push button

Since the stepper motor has no way of knowing an absolute zero angle, the zero angle is set arbitrarily at the current position of the motor. Clicking the *Reset angle* button then sets the current position as the initial angle.

3.4 Plot widget

Once a movement is commanded by pressing the *Move* button, measurement data is received from the Portenta board and plotted in this widget. This is implemented with the Python library Matplotlib. This widget is framed in purple in Fig. 9.

4 Graphical User Interface V2

The Graphical User Interface is updated to comply to new requirements such as to display the angular speed, have user guidance help prompts, save measured data and start a routine that incorporates elevation as well as azimuth movements.

A scroll area is added to the right widget since the new functions added do not fit in a minimised window (see Fig. 11). When the window is maximised (see Fig. 12) all widgets are shown without the need for a scroll bar.

Fig. 11: Minimised GUI.

Fig. 12: Maximised GUI.

4.1 Azimuth settings

This section of the rightmost widget is used to modify settings for the stepper motor used to rotate the antenna in azimuth. This provides the option to rotate the azimuth motor to an arbitrary angle to set the absolute angle zero where desired. Note that a stepper motor does not have an absolute register of its position as compared to, for instance, a servo motor, so the tracking of position is made by software with a global variable at the microcontroller.

4.1.1 Angular resolution

The stepper motor used for development has a *native* angular resolution of 1.8 deg/step. A micro-step driver is used to modulate the current at the stepper motor to make the motor move in smaller steps. To change the resolution of the micro-step driver, switches must be moved manually. Since it is inconvenient to have to set the switches manually for choosing each micro-stepping resolution, the resolution is always set at 32 subdivisions per step (i.e. 6400 step/rev or 0.05625 deg/step) and the resolution set at the GUI is used to determine the number of steps to skip to achieve each resolution. For example, for a resolution of 3200 step/rev or 0.1125 deg/step, a measurement at the analogue pin is taken every two steps only.

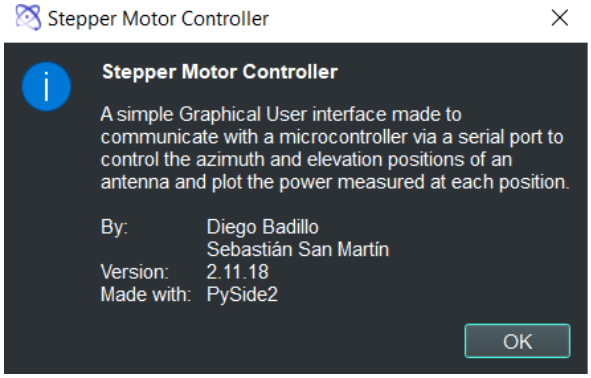
Table 2 includes a summary of the average angular velocities achieved for each resolution. The measurements are made both with 20 and 2 revolutions to show the effect of the acceleration periods. Note that for an acceleration period of zero, the average RPMs are virtually the same for 20 and 2 rotations, whereas when an acceleration period is added, the average RPMs are less for 2 rotations than for 20 rotations.

Setup (step/rev)	Ang. resolution (deg/step)	Accel. period (rev)	Inferior accel. time (μs)	Superior accel. time (μs)	RPM (20 revs)	RPM (2 revs)
6400	0.05625	0	0	0	37.4	37.2
3200	0.1125	0	0	0	74.4	74.4
1600	0.225	0.75	0	12	146.5	133.9
800	0.45	0.75	4	22	234.4	176.9
400	0.9	0.75	6	22	323.3	199.5
200	1.8	0.75	10	22	-	-

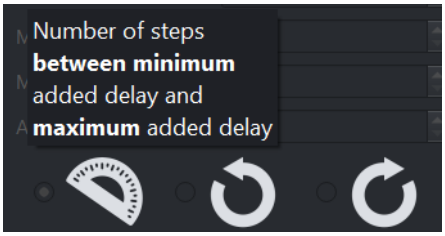
Table 2: Angular velocities comparison.

4.2 Elevation settings

This section is used to modify settings for the stepper motor used to rotate the antenna in elevation. This works the same way as for azimuth rotation, and since in the development end only one stepper motor is available, these options control the same motor provisionally. In case a second motor is available, the respective pins must be set at the Arduino code to control it independently of the azimuth motor.



(a) About window.



(b) Example of a tool tip.

Fig. 13: Error and Information windows.

4.3 Data statistics labels

Below the plot area, four labels are added which display characteristics of the measurements as well as the motor angular speed. From left to right, these are **Peak Power**, **Peak Angle**, **Mean Power** and **Revolutions Per Minute** (see Fig. 12 or Fig. 11). Since the Portenta Breakout board has got an AD converter, the vertical axis of the plot shows voltage, so the peak and mean *power* are shown in voltage for now. A power meter is supposed to be

connected in the future to the analogue pin, in which case the power measured at the antenna will be mapped to a voltage.

The angular speed displayed in RPM is calculated by measuring the average time that the stepper motor takes to move each step, and considering that it will rotate 6400 micro-steps each turn. Note that this conversion is hard coded for a stepper motor with an angular resolution of 1.8 deg/step and a micro-step driver with a maximum subdivision of 32 per step, as this is the setup available at the development end.

4.4 Guidance help prompts

A menu bar is added in this new version of the GUI. By clicking Help > Help Content, this progress report is opened in the browser as a PDF file stored at GitHub. By clicking Help > About, a new window is opened, where basic information of the software are given, such as the authors, a brief description of what the GUI is designed to do, and version (see Fig 13a)

Tool tips are added to each new widget. These are revealed by hovering the cursor at a widget. See Fig. 13b for an example.

4.5 Save as CSV option

By clicking File > New Save File, a file path is established to start saving following measurement data. Each log contains the same information used for plotting the data, such as direction of movement (clockwise or counterclockwise), final angular position, and angular resolution used for the movement.

When a path is chosen, all movements made at the stepper motor are saved with their respective log time and date until the program is closed.

4.6 Routine settings

The implemented routine corresponds to a sequence of movements of the azimuth and elevation motor. The range of these movements is determined by the values entered by the user in the widgets "Initial angle" and "Final angle", both for the azimuth and elevation movement (see Fig. 14). This sequence can be described by the following steps:

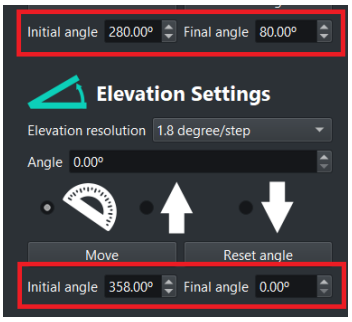


Fig. 14: Initial and final angle widgets.

1. Both motors move to their respective starting position determined by the value of the "Initial angle" widget.
2. Then, the azimuth motor moves from its initial angle to its final angle, recording the measured data.
3. The elevation motor moves one step towards its final angle. This step is calculated based on the elevation resolution entered by the user.
4. The azimuth motor moves once again from its initial to its final angle.
5. The elevation motor moves another step. These last two steps are repeated until the final elevation angle is reached.

To start the routine, the user first needs to set up the initial and final angle values for both motors (default cases are shown on Fig. 14) and then press the *Start* button, which will initialise the routine. Note that the motors will use the currently set parameters (acceleration and resolution) for this routine, so if changes are required, the apply button must be pressed before starting the sequence.

Currently, this routine has been implemented and tested using only one motor, so both azimuth and elevation movements are applied on the same axis, which can lead to unexpected errors. For this reason, a second motor needs to be acquired in order to correctly debug and test the current implementation.