CellAntenna

System Description

The purpose of this project is to connect the user, via a graphical user interface (GUI), to an API on a micro-controller. The name of the application is **MGTron GUI**. The application will be functional on Windows 10, Windows 11, Debian, Red Hat, and Arch based operating systems. The application will be a single executable that is expected to work across the previously listed operating systems. The deployment method allows for versatile deployment. The MGTron GUI front-end expectation will be provided in an image later within this document. The purpose of the image is to provide the expectation of what the GUI should look like to a user, to include colors.

The GUI should have, but not be limited to, the following features and characteristics:

- Configurable configurations for up to eight devices.
- Save a configuration file for the given input of frequency, power, and

bandwidth.

- Easily installable to most operating systems.
- Choose a specific device based on its serial number.
- Easy one-click to turn off a generating channel.
- Send to all channels at once or individually.
- Set the frequency, power, and bandwidth of each of the eight channels.
- Configurable mission buttons.
- Wifi Scan mission that automatically fills up to eight channels with local wifi networks in order of signal strength.
- Establish a serial connection to a **Teensy 4.1** microcontroller
- Stateful
- Update user on current state of selected devices
- Display, in the GUI window, *CellAntenna* in the upper right left hand corner. (At least on Linux)
- The version, starting at 3.0.0, will be displayed in the bottom right hand corner
- If the wifi scanner does not fill all of the fields then set the power level to zero on non-results

Detailed front-end description

The GUI will be 1250 x 735, initially. The window will be resizable and all elements within will scale accordingly. There will be a log file that overwrites itself everytime the GUI is launched. The code will be commented heavily, but not as often as every line. The Python source code will be provided. The image of the GUI shows plus and minus buttons, remove all of the plus and minus buttons. The primary method of input will be a human interface device (HID) such as a keyboard or a numpad. The GUI will be deployed on a computer connected, via HDMI, to a touch screen display. Again, the primary input will be an HID. Other than the plus and minus buttons next to the float and integer inputs, the GUI is expected to resemble the provided image of the present implementation of the GUI.

- **Configurable configurations for up to eight devices**

  - The GUI will allow a user to save a default configuration linked to a serial number that loads when the requisite device is selected. The default values to be saved are frequency (3 significant figures), power (int), and bandwidth (int).

- **Save a configuration file for the given input of frequency, power, and bandwidth**
  - Use the *"SAVE CONFIG"* button to pop-up an input that will allow a user to type the name of the save file. The save config file will consist of the values that are presently in the input fields of the frequency, power, and bandwidth of each of the eight channels. The values in the input fields are to remain untouched during and after the save process.

- **Easily installable to most operating systems**
  - The final product will be a single executable. The only external dependency will be the database. The code will internally handle platform checking and switch from *COM* ports to file paths according to the operating system.

- **Choose a specific device based on its serial number**
  - The upper drop down labled *Device Config* display the connected devices' serial number. The serial device must be checked to ensure it is a **Teensy 4.1** before the serial number can be displayed. The serial numbers should be kept in a locally referential data structure to be used in another feature.

- **Easy one-click to turn off a generating channel**
  - One button will set the power level to zero for every channel of the selected serial device.

- **Send to all channels at once or individually**
  - Give the user the ability to send the values presently in the input fields individually or send every value in every input field for every channel at once. This applies to the presently selected card only. It is accepted that there exist the state in which a serial device has been set then de-selected; the settings are still set but no state of the card is tracked.

- **Set the frequency, power, and bandwidth of each of the eight channels**
  - Set the frequency between 50 and 6400, power 0 to 63, and bandwidth 0 to 100. The user input will be an HID. The frequency inputs can be floating point values, power inputs are integers, and bandwidth inputs are integers. Input fields will be provided for each of the eight channels and each frequency, power, and bandwidth.

- **Configurable mission buttons**
  - Make the *MISSION* buttons execute a saved file if the saved file has the identical name as the button. If more than one save file has the save name then the particular button in question will, essentially, load the latest save name file by date and time of save.

- **Wifi Scan**
  - Use the host computers wireless card to scan for wifi SSIDs and the SSID's RSSI. The units of the RSSI will be in dBm and be, most likely, negative. The more negative the RSSI the weaker the signal strength the respective SSID. The fastest method is to use an API to leverage the host systems wifi card.

- **Establish a serial connection**
  - The serial device is a Teensy 4.1. The serial device is expected to have a proprietary firnware. The firmware has an API with which makes keeping state possible. An example of the type of response is included later in this document. The serial device controls, based on user input, a signal generator. The signal generator is capable of producing frequencies from 50 MHz to 6400 MHz. The API documentation will be included.

Back-end

The back-end will, in part, save user configurations, store the saved configurations per serial number, and allow a user to name a saved configuration at the time of saving via an input popup. The back-end is also responsible for processing the user-input to be in a format suitable for consumption by the firmware API. The back-end is not limited to the preceding listed items.

The back-end is expected to process the user inputs via the press of the buttons presented on the GUI. The back-end is also expected to be state-aware. The API of the serial device allows for feedback. This feedback is to be reflected, in real-time, via the user-facing interface after a refresh button is pressed. The present implementation employs a JSON database for all storage needs. This may be a limiting factor and is to be replaced with a suitable solution. The wifi scanner leverages the a Linux command **nmcli** to scan the presently transmitted wifi frequencies. The results are processed in the back-end and transformed into a format digestable for the front-end. A Windows solution has not been researched. The scanner will list the scan results in frequency strength order starting with the strongest result in channel one.

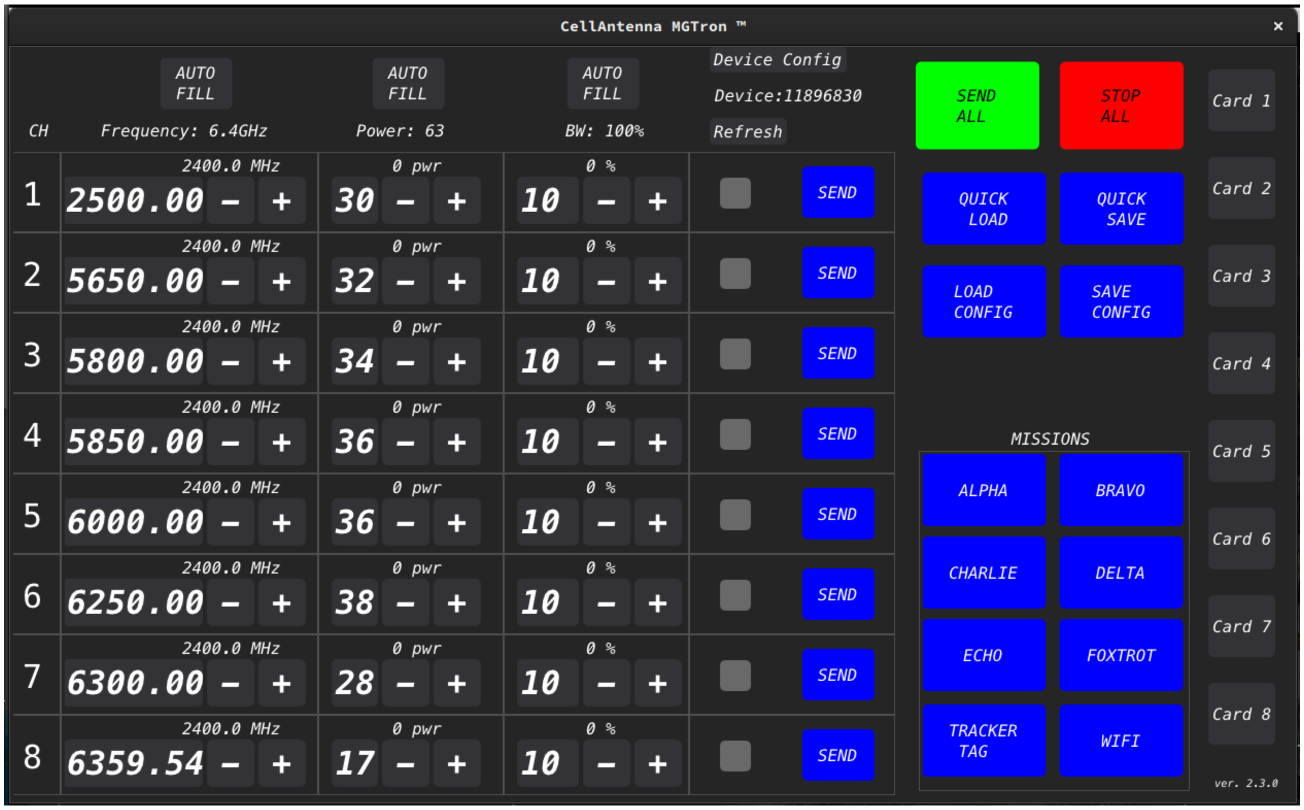
Serial Device

The serial device is a **Teensy 4.1**. There is a custom API on the serial device which allows communication between any application and the firmware on the serial device. The only feedback the serial device returns is in the following format:

```
Serial Command Received
Command: 's'

-Channel 1 Status-
 Frequency: 2400.00MHz
 Power: 0
 Bandwidth: 0.00%

 -Channel 2 Status-
 Frequency: 2400.00MHz
 Power: 0
 Bandwidth: 0.00%

 -Channel 3 Status-
 Frequency: 2400.00MHz
 Power: 0
 Bandwidth: 0.00%

 -Channel 4 Status-
 Frequency: 2400.00MHz
 Power: 0
 Bandwidth: 0.00%

 -Channel 5 Status-
 Frequency: 2400.00MHz
 Power: 0
 Bandwidth: 0.00%

 -Channel 6 Status-
 Frequency: 2400.00MHz
 Power: 0
 Bandwidth: 0.00%
```

```
-Channel 7 Status-
Frequency: 2400.00MHz
Power: 0
Bandwidth: 0.00%

-Channel 8 Status-
Frequency: 2400.00MHz
Power: 0
Bandwidth: 0.00%

New command: $
```

Front-end



Device:11896830

CH	Frequency: 6.4GHz	Power: 63	BW: 100%	
1	2400.0 MHz 2500.00	0 pwr 30	0 % 10	SEND
2	2400.0 MHz 5650.00	0 pwr 32	0 % 10	SEND
3	2400.0 MHz 5800.00	0 pwr 34	0 % 10	SEND
4	2400.0 MHz 5850.00	0 pwr 36	0 % 10	SEND
5	2400.0 MHz 6000.00	0 pwr 36	0 % 10	SEND
6	2400.0 MHz 6250.00	0 pwr 38	0 % 10	SEND
7	2400.0 MHz 6300.00	0 pwr 28	0 % 10	SEND
8	2400.0 MHz 6359.54	0 pwr 17	0 % 10	SEND

Auto-fill

The **AUTO-FILL** button has different functionality for *FREQUENCY* as opposed to the *POWER* and *BANDWIDTH* fields. Under *FREQUENCY*, the **AUTO-FILL** button takes the difference of the second and first inputs and increments the remaining inputs the value of the resulting difference.