FLIR Tau 2 Emulator

Version 0.1

Version 0.1 provides the skeleton of the Tau 2 emulator. Currently, a user can run `stub.py` with Python 3 and enter packets as they would send them to the Tau 2. The emulator will accept packets and respond accordingly. However if the user sends a packet that asks for a non-implemented command, the program will respond that the requested command has not been implemented.

Using the Emulator

Users must have Python 3.6 installed, as some features of this program are not supported by earlier versions. Navigate to the `cam-stub` directory in your terminal, and enter

```
python3 stub.py
```

If all goes according to plan, you will be greeted with the console based emulator. The emulator will ask you to input a packet. A packet is a string of numbers that you will construct based on the Tau 2's packet protocol (refer to page 12 of the Tau 2 Software IDD).

In the current version, you should start each packet with `0x6E`. If you neglect to include the `0x`, the program will take care of it. The rest of

the packet should adhere to the aforementioned protocol.

CRC codes have no effect in this version, but should still be included in the packet. If the CRC codes, or at least `CRC1`, the Argument byte will not be read into the program correctly.

Adhere to these guidelines, and your experience should be fluidly limited (at least in the current version)

Developing for the Emulator

If you, as a developer, wish to add support for a command, you can do so very easily. Follow this process:

1. Navigate to the `commands` directory
2. Find the file with the same name as the command you will be implementing
   – For example, if you want to implement the SERIAL_NUMBER command, you should locate `serial_number.py`
3. Edit the logic of the function to implement the command
   – Each function takes two arguments, `settings` and `reply`. When each command is implemented, the settings and reply dictionaries should be edited appropriately.
   – If you're implementing a command that edits all settings (`SET_DEFAULTS` for example), the current settings and default settings are saved as json files in the `storage` directory.
4. Make sure the function has this as a return statement: `return settings, reply`

5. In `camera.py` add an import statement for your command
   – If you added the SERIAL_NUMBER command, you would add:

```
from commands import serial_number as SERIAL_NUMBER
```

6. In `camera.py` uncomment the corresponding command in the `self.funciton_codes` dictionary
   –If you added the SERIAL_NUMBER you should uncomment this line:

```
#   '04': SERIAL_NUMBER
```

7. Everything should be in place. Give it a shot!

Tools

The `tools` directory is a place to store scripts that relates to the development of this emulator, but **should not be used in the program**. Currently, the only file is `make_settings_json.py` It's purpose is to maintain the settings json files. As we learn more about the camera, we will edit the contents of this file's dictionary, run this file, and update the settings json files. This script has support to work on Windows, Linux, and Mac OSX.

Upcoming Features

- Commands will be implemented frequently
- A unit testing procedure should be put in place so implementation is easier

- A script should be built to automatically update `camera.py` for implementing new commands
- A help option should be added so a user can get more information about the packet protocol and what the reply means