HILO App

The HILO app is designed as a user-friendly software to control a HILO spinning machine. The functionality is limited (for now) to controlling the thickness and amount of twist of spun yarn. Users can select a HILO device connected to their computer (via a USB serial port) or use a simulator. The app is written in Processing and can be launched from the Processing IDE, or exported as a stand-alone app for MacOS, Windows or Linux.

Keyboard Controls

The app is made to be used with the mouse or touchpad. A few keyboard shortcuts were left in, which can be helpful when using or debugging.

* **c**​ - connect to HILO
* **C**​ (uppercase, Shift + c) - disconnect from HILO
* **Spacebar**​ - start or stop spinning
* **l**​ (lowercase L) - print the list of available serial ports to the console (for debug)
* **I**​ (uppercase i, Shift + i) - load a new image (only in pattern mode)

Settings Files

The app is provided with a set of configuration files, within the ​**data/settings/**​ folder. The files are written in plain text using a JSON format. They can be opened and edited using a simple text editor (e.g. TextEdit, Notepad) and are easy enough to understand and modify. The original names and locations should be kept, as the software is looking for these exact files at the exact location. Descriptions of each file and contents follow.

App Settings

Filename: ​**HILO\_AppSettings.json**

This file contains basic settings for the app’s behavior and appearance. This includes the size of the viewport window and a reference for the file from which the app’s text is loaded.

* **prefPort**​:thenameofthepreferredserialport,withindoublequotes.Forinstance **"COM4"**​ on Windows or ​**"/dev/tty.usbserial001"**​ on Mac OS. You can leave it blank, as just a pair of double quotes ​**""**​. Whenever you successfully connect to a HILO machine via serial port, the software sets that port as the preferred port and updated the settings file.
* **fullscreen**​:ifsetto​**true**​(noquotes)theappwillstartinfullscreenmodeandignore the ​**width**​ and ​**height**​ settings. If set to ​**false**​ (no quotes) the app’s window size is determined by ​**width**​ and ​**height**​.
* **langFile**​:thenameofthefile(inquotes)containingtheapp’stext,bydefault **"HILO\_AppText\_EN.json"**​. It would be possible to use a different file, so the app can be translated to other languages.
* **width**​and​**height**​:positiveintegernumbersdeterminingthesizeoftheapp’swindow, in pixels. If the ​**fullscreen**​ option is set to ​**true**​ then ​**width**​ and ​**height**​ are ignored.

Machine Settings

Filename: ​**HILO\_MachineSettings.json**

This file contains settings for the HILO machine. Every time a connection is established, these values are set in the machine. The values can’t be changed in the UI. They are pre-determined through tests using the in-house test app.

* **deliverySpeedSteps**​: the speed in steps per second (positive integer value, like​**300**​) for the delivery speed. Essentially, the speed at which the machine is running.
* **draftingSpeedPercMax**​:thehighestvalueforthespeedofthedraftingroll,as a percentage of the delivery speed. A positive integer between 0 and 100, like ​**70**​. This corresponds to the drafting speed for thick yarn.
* **draftingSpeedPercMin**​:thelowestvalueforthespeedofthedraftingroll,asa percentage of the delivery speed. A positive integer between 0 and 100, like ​**20**​. This corresponds to the drafting speed for thin yarn.
* **spindleSpeedStepsMax**​:thespeedinstepspersecond(positiveintegervalue,like​**700**​) for the spindle, corresponding to the maximum amount of twist in spin mode.
* **spindleSpeedStepsMin**​:thespeedinstepspersecond(positiveintegervalue,like​**200**​) for the spindle, corresponding to the minimum amount of twist in spin mode.

HILO Devices

A HILO device is represented as an abstract in the app, as a ​**HILOInterface**​ (found in the tab **HILO\_BaseClasses**​). This defines a common set of operations for any HILO device, and allows us to use an actual device (communicating through a serial port); or a “simulator” so that we can test and demonstrate the app without needing a machine or other hardware. In the future we may wish to define other devices such as remote/networked HILO machines and clusters, or different versions of the HILO machine.

The simulator acts for most (but not all) purposes like an actual machine. As mentioned above, it is meant to replace an actual machine during app testing and quick demos. The simulator class ​**HILOSimulator**​ is defined in the tab ​**HILO\_Simulator**​, whereas the actual HILO (which connects to a serial port and communicates with a machine using the HILO protocol) is implemented as class ​**HILODevice**​ in the tab ​**HILO\_Device**​.

A device or simulator should be assigned a callback handler (implementing the **HILOCallbackHandler**​ interface, see the ​**HILO\_BaseClasses**​ tab) which gets notified of changes in the state of HILO and acts accordingly. The actual handler used by the app for HILO events is an ​**AppHILOCallbackHandler**​ (implementing the ​**HILOCallbackHandler**​ interface) and can be found in the tab ​**App\_Callbacks\_HILO**​.

User Interface

The layout of the UI is based on a mixture of fixed values and reactive behaviour - that is, the position and size of some elements is calculated depending on the size of the app’s window.

Some of the UI elements are based on visual assets (e.g. icons) which can be found in the folder ​**data/icons/**​ folder. There are different asset folders (​**small**​, ​**medium**​ and ​**large**​) corresponding to different window sizes. The fonts used by the app can be found within the **data/fonts/**​ folder. Fonts are in a vector-format which (contrary to the icons) can be scaled for different sizes without a loss in quality.

The variables, objects and UI setup functions can be found in the ​**UI\_All\_Layout**​ tab; with special attention to the ​**uiSetup()**​ function, where UI elements are created and laid-out.  
The app’s UI is built from a series of UI elements. These are objects descending from class **UIBasicElement**​ or otherwise conforming to the ​**UIElement**​ interface (see the tab **UI\_BaseClasses**​). Each UI element can be assigned a callback handler (​**UICallbackHandler** object or a descendant, see the same tab) which gets notified of changes in the element and acts accordingly. For instance: when a slider knob is moved, the slider element notifies its handler via the ​**callbackUISliderDragged()**​ callback, and the handler object changes the appearance of another UI element, or triggers an update on the HILO machine.

Classes defining UI elements can be found in tabs with the prefix ​**UI\_**​ and a tab may contain more than one class - for instance ​**UI\_BaseClasses**​ or ​**UI\_Text**​. The actual handler used by the app for UI events is an ​**AppUICallbackHandler**​ (extending the original **UICallbackHandler**​) and can be found in the tab ​**App\_Callbacks\_UI**​.

Serial Communication Protocol

The HILO machine’s “brain” is an Arduino-based board. The original prototype used an Arduino Uno model with a custom-made shield, whereas the new prototype uses an Arduino Mega 2560 with a RAMPS v1.4 shield. The machine can be controlled by sending simple text-based messages to it via the Arduino’s serial port (USB) running at a speed of 115200 Baud. This is commonly done by an app running a graphic user interface (HILO App) with buttons and sliders, but you can also control the machine “directly” by opening a serial port terminal/monitor (such as the ​*Serial Monitor*​ tool in the Arduino IDE) and typing in the messages yourself between [square brackets](https://www.youtube.com/watch?v=pxh87KO5VPU)

The protocol defining the messages and behavior is rather simple. The user or app sends a command to the machine; and the machine replies with a conformation or error. For instance, the app sends a command to start spinning with the drafting speed at 40% of the delivery speed, and the machine confirms that it started spinning:

● app sends message ​**[R,40]**​ which means “run at 40% drafting speed”

● machine replies with ​**[R]**​ which means “I’m running”

Each message (command or reply) is contained in square brackets and consists of an upper-case character and one or two optional parameters (integer numbers).  
Once a command is sent, the machine will typically reply with the same command (upper-case character), without parameters, as above. If an error occurs, it replies with an error message, which contains an error code number - for instance ​**[E,6]**​ when receiving an unknown command. Please refer to the firmware for each code’s description.  
A machine programmed with a debug-enabled firmware will also send debug messages starting with the character ​**X**​ such as ​**[X,handleCommandRun(): HILO is already spinning]**​. These are just meant to provide feedback when testing, and are otherwise inconsequential. The HILO app (or rather, the ​**HILODevice**​ class used in the app) prints these messages to the console.

>> Please check our video How to use arduino serial port

<https://vimeo.com/639190238>

|  |  |  |
| --- | --- | --- |
| **Command** | **Meaning and reply** | **Example**  **(command/reply)** |
| **[R,n]** | Run the machine, with a drafting speed as a percentage of the delivery speed, as an integer number between 0 and 100. Can also be called while the machine is running, to change the drafting speed.  **Reply:**​ ​**[P]**​ in case of success, or ​**[E,n]**​ in case of error (number or value of parameters, or the elevator is resetting) | **[R,70]**  **[R]** |
| **[S]** | Stop the machine. This can be used while the machine is spinning or resetting the elevator.  **Reply:** ​**[S]**​. | **[S]**  **[S]** |
| **[D,n]** | Set the speed of the delivery motor(s) in steps/second.  **Reply:**​ ​**[D]**​ in case of success, or ​**[E,n]**​ in case of error (number of parameters or value). | **[D,300]**  **[D]** |
| **[P,n]** | Set the speed of the spindle motors in steps/second.  **Reply:**​ ​**[P]**​ in case of success, or ​**[E,n]**​ in case of error | **[P,550]**  **[P]** |
| **[A]** | Ping the machine, which can be used to check if the a HILO machine is connected to a serial port and responding properly.  **Reply:**​ ​**[A]**​. | **[A]**  **[A]** |
| **[T]** | Request the machine’s current state. **[T]**  **Reply:**​ either ​**[S]**​, ​**[R]**​ or ​**[V]** when stopped, running or resetting. | **[S]**  ​**[R]**​  ​**[V]** |
| **[I]** | Request information about the machine. Mostly useful for debug purposes, or to check which version of the firmware is running in a machine.  **Reply:**​ a debug message like  [X,HILO Firmware 0.1.0. State S. Delivery speed -240. Spindle speed 700. Elevator height 0] |  |
|  |  |  |

Firmware

The firmware for the HILO machine - that is, the software which runs on the machine itself - is developed using the Arduino IDE and meant to be used in Arduino-compatible boards. The firmware uses the ​AccelStepper library​ by Mike McCauley, which can be installed via the ​*Library Manager*​ tool in the Arduino IDE.

>> Check the video How To install HILO Firmware in Arduino