# Hardware

## app.py

Sets up the Flask server by defining all the endpoints. The endpoints are organized into API Endpoints, which are the endpoints intended for use, and manual endpoints, which give direct manual control over the available system actions.

On start, the application attempts a connection to the 3D printer over serial and performs its homing procedure. It then attempts a connection to the Arduino over serial/pyFirmata and homes its components. If either of these connections fail, the server will not start, unless WEBSITE\_TEST\_MODE is on.

The connections to the Ender and Arduino are made by a AISHLoader object, which contains all the relevant functions and actions for the loading/unloading procedures. A AISHExperiment object is created when the system is given an experiment and this class contains all the relevant functions and actions to execute an XRD experiment. The aish\_experiment variable is set to null if there is no currently running experiment.

## AISHExperiment.py

Contains the relevant functions and actions to execute an XRD experiment. The Diffractometer class is taken from Nathan’s AdaptiveXRD code (<https://github.com/njszym/AdaptiveXRD>) to run the Bruker. The run\_sequence function is a blocking function that runs all XRD scans at the given temperatures. Since it is blocking, the Flask server runs this function in a separate thread to prevent the server from stalling when an experiment runs.

## AISHLoader.py

Contains relevant functions and actions for the loading/unloading procedures. This is meant to be an abstraction layer to help group the low level procedures together.

## ArduinoHardware.py

Contains the control code for the Gripper and Linear rail, which are each subclasses. Each command is assigned a unique Firmata sysex command ID, which instructs the Arduino to execute its various actions. It then listens for an acknowledgement from the Arduino that the command was completed (blocking).

## Ender3.py

Contains the control code for the Ender3. This class writes gcode commands to the serial bus to control movement. Most of the movements utilize \_move\_to(x,y,z, speed) which is a function that sends the required gcode to move to the specified x,y,z position. This function is also blocking, so each move will be completed before moving to the next.

## AISH\_utils.py

Contains classes to handle communication timeouts that may occur. Since the system runs in semi open-loop, some actions are not verified completed. So these functions allow the system to ask the user to confirm that an action was completed.

## StateTracker.py

Helper class that logs the last 100 states. The other hardware classes inherit from this class, so all of them have a list of the last 100 states they were in. These states are just strings. Meant for debugging but could be helpful for logs.

# Website

## templates > index.html

Contains the html for the web UI to queue experiments. Also includes the UI for manual control of all system components

## static > js > app.js

app.js contains all the js components for website communication with the Flask server, and updating components on the webpage.

The webpage first loads utils.js, then loads app.js. app.js attaches all button callbacks when the website has loaded. A loop is started to refresh the webpage state every set interval where the webpage asks the Flask server for the current system state and displays that to the webpage.

## static > js > utils.js

utils.js includes tools for rendering the webpage like experiment time estimation, dynamic button layout (the buttons for the sample positions are generated with js).

# Arduino Code

## AISHLoader

Essentially a switch-case list of the different commands that the Arduino will perform.

## Tests > LinearRail\_Testing

Tests the Linear rail motion. Takes input from the serial monitor for how many steps to move (+ for up, - for down) and “home” to execute the homing procedure. Use this to dial in the amount of throw for the linear rail.

## Tests > Servo\_SerialTesting

Tests the gripper servo motion. Takes input from the serial monitor of the angle to move the servo to.

# Calibration

The sample positions of the Ender sometimes has to be recalibrated. The positions are found by finding the 4 corner positions, and the rest are interpolated. The calibration script moves the gripper to the desired position, moves down to grab the sample, lifts it, then puts it back. The sample should not scrape against the holder. My testing seems to show that there +/- 0.3 mm of tolerance from perfectly centered.