Pneumatic Actuator Test Stand

**Install, Run, & Tune the Controller**

Evan Comiskey – MIT FID Lab – 2024/08 – comiskey@mit.edu

# Description:

This How-To serves as a guide for installing the Pneumatic Actuator Test Stand’s PID control software. It will walk you through the entire process, from downloading an IDE to code, all the way to fine tuning your solenoid valve and associated controller gains. This guide assumes you have a functional hardware and electronics set-up, featuring an Arduino Nano board with PWM capabilities. Please reach out if you feel any critical information to be missing from this guide.

Table of Contents:

[Description: 1](#_Toc181552534)

[1. Installing PlatformIO 2](#_Toc181552535)

[2. Downloading the Codebase 5](#_Toc181552536)

[3. Setting up your Working Directory with PlatformIO 6](#_Toc181552537)

[4. Running the Controller 9](#_Toc181552538)

[5. Adjusting Settings & Tuning Your Solenoid 11](#_Toc181552539)

[6. Processing Test Data 14](#_Toc181552540)

# Installing PlatformIO

* Use an Interactive Development Environment (IDE) capable of running [**PlatformIO**](https://platformio.org/)projects.
  + Skip to section 2 now if your IDE already supports PlatformIO projects.
* PlatformIO Overview:
  + PlatformIO is an open-source tool for working with embedded electronics systems like Arduino microcontrollers.
  + It offers a more in-depth and organized way to work with microcontroller code than the standard ArduinoIDE.
  + Can be run using various free and popular IDEs [listed here.](https://platformio.org/install/integration)
* Setting up PlatformIO on Visual Studio Code (VS Code):
  + Download VS Code for Windows, Mac, or Linux at [this link](https://code.visualstudio.com/download) or by going to <https://code.visualstudio.com/download>.
  + Follow all default installation instructions, as PlatformIO does not require you to do anything specific during VS Code’s installation.
* Once you have successfully installed VS Code, you will see a screen as shown:
  + A screenshot of a computer

    Description automatically generatedFeel free to go through VS Code’s provided tutorials as is necessary to familiarize yourself with the application.
* Install necessary C/C++ Dependencies
  + click the **Extensions** tab shown in the menu bar near the top left of the screen, highlighted in red below at left.
  + Type “**C**” into search bar. This will cause **C/C++ by Microsoft** to appear. This is the language which PlatformIO depends upon, so we must download it beforehand.
  + A screenshot of a computer

    Description automatically generatedA screenshot of a computer

    Description automatically generatedA screenshot of a phone

    Description automatically generatedClick the **install** button for the extension.
* Similarly search for **PlatformIO** in the search bar. Download the first option that appears in the dropdown.
  + The author of this extension will also be named PlatformIO.
  + If you click on the extension, you should see a preview page which looks like the image shown at right.
  + A screenshot of a computer

    Description automatically generatedA screenshot of a computer

    Description automatically generatedIt will take a few minutes for PlatformIO to fully install. Once the application is fully installed, you will be prompted to quickly restart VS Code
* You will see a PlatformIO alien-head icon appear in the righthand menu where the extensions tab was once the application has successfully been installed.
  + Clicking on this icon will bring up the PlatformIO **project tasks** and **quick access** menus.
  + Press the PIO Home **Open** button in the quick access menus to bring up the PlatformIO homepage.
* You can start brand new projects from the PlatformIO homepage.
* A black and white logo

  Description automatically generatedA screenshot of a computer

  Description automatically generatedA screenshot of a black screen

  Description automatically generatedPlatformIO is successfully installed if you see the home screen shown below at right, and you are ready to start working with this project’s codebase.

# Downloading the Codebase

* Download the codebase by accessing the GitHub repository associated with this project by [clicking here](https://github.com/Comiskey/soft-actuator-controller), or by going to <https://github.com/Comiskey/soft-actuator-controller>.
* **A screenshot of a computer

  Description automatically generated**A screenshot of a computer

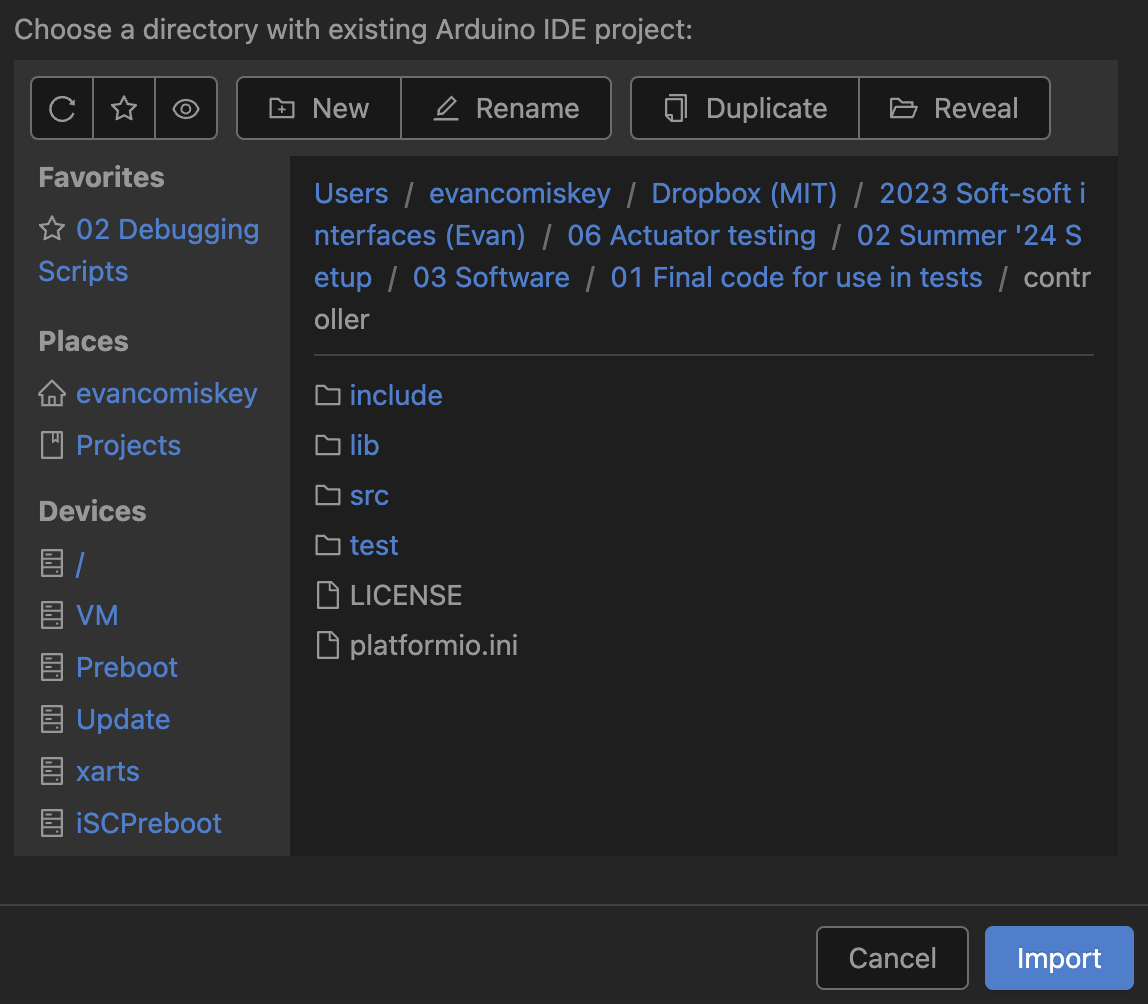
  Description automatically generatedOnce here, click the green **Code** dropdown button, then click **Download ZIP**
* Unpack the contents of this zip file in an easily accessible file location.
  + Note that at this point, you should not interact with the code directly. The first time you open this folder in a code editor, you should do so through PlatformIO, following the steps explained below.

# Setting up your Working Directory with PlatformIO

* A screenshot of a computer

  Description automatically generatedNavigate to the PlatformIO home page and click the **Import Arduino Project** button
* An import menu will appear.
  + This menu has two parts: **board selection** and **folder directory selection**.
  + Click **board selection**, and type Nano into the search bar. From here,
    - select **Arduino Nano ATmega328 (New Bootloader)**.
  + A screenshot of a computer program

    Description automatically generatedA screenshot of a computer program

    Description automatically generatedClick the **Use Libraries installed by Arduino IDE** checkbox to include necessary Arduino dependencies for the codebase.
  + In **folder directory selection**, navigate to where you had previously unpacked the downloaded GitHub repository, then open the **controller** sub-folder. You should see the folder structure highlighted in blue in the directory selection sub-screen below. Once you do, click the **Import** button.
* You will have to redo this process if you decide to move the repository within your computer’s folder structure later, as PlatformIO will have to reinstall its dependencies in that new location.
* PlatformIO will begin automatically formatting the content of the directory to match the selected board once the project has been imported.
  + This may take a few minutes.
  + You can also change this later in the *platform.ini* file.
* A screenshot of a computer program

  Description automatically generatedSet-up will take a few seconds, but once completed, will bring you to the following screen:
* PlatformIO has opened the controller code’s directory in VS Code’s **Explorer** tab (highlighted in red above).
  + From now on, whenever you open the folder in VS Code, all necessary PlatformIO dependencies will open as well.
* A screenshot of a computer

  Description automatically generatedYou will know that PlatformIO is properly configured and running by the series of button options which will appear at the bottom of the screen when the directory is opened:
* Of these buttons, the most important are the following:
  + **Home:** Brings up the PlatformIO home page shown previously at the top of section 3
  + **Build:** Compiles all code in the directory. This button allows you to check the code for syntax errors without having to be directly connected to a microcontroller
  + **Upload:** Compiles, and sends the code to the microcontroller
  + **Serial Monitor:** Opens the serial monitor. Useful for set-up and general debugging
* Press the **Build** button to ensure that the directory has been properly configured as a PlatformIO project.
  + A screenshot of a computer

    Description automatically generatedIf it has been, the terminal will automatically open and present a success message after a few seconds as shown:
* Move on to running the controller once you see the success message appear in the terminal.

# A screenshot of a computer Description automatically generatedRunning the Controller

* This section will cover how to run the controller during normal operation/cycle testing.
  + This section assumes that your solenoid valve has been properly tuned prior to your use of the stand. If this is not the case, please read through this section for context, before moving on to section 5 to walk through controller tuning procedures.
  + If someone in your group has successfully used the stand before, you do not need to read section 5.
* All settings relevant to the operation of the controller are found in a single file: **adjustableSettings.cpp**, which lives in the **src** folder.
  + **adjustableSettings.cpp** only contains variable definitions and is only relevant to quick set-up, adjustment, and use of the controller.
  + For a detailed explanation of what each variable does, please see the **Test Stand & Controller Overview** Word document.
* Ensure that the first 10 variables—pin definitions—match your electronics’ layout.
* Ensure that both **TUNE\_PRESSURE** and **TUNE\_VENT** are set to false.
  + Otherwise, the controller will operate in tuning mode, rather than as the normal controller.
* **fileName** is a string which sets the naming scheme for the tests you wish to run.
  + Files will automatically be generated on the system SD card in the form **fileName**\_#.csv, where # represents the lowest unused integer value.
* **USE\_SD\_CARD** when true, will send all test data to the onboard SD card. The controller *will not function properly* without an SD card inserted in this state
  + When this Boolean is set to false, all data will be exported to the serial port, for use with MATLAB plotting scripts for faster gain-tuning workflows. All relevant data is still preserved in csv files automatically.
* The next relevant setting will be **USE\_KPA**.
  + If true, it will record data in kilopascals.
  + If false, it will record in PSI.
  + Trajectory setpoints, as well as the overpressure limit, will automatically be interpreted in the units associated with this variable. Gains will be affected.
  + Whether **USE\_KPA** is set to true or false is automatically included in the output file of a test.
* Next, select your delays for pressure reading/writing, interpolation, and controller action using **PRESSURE\_READ\_DELAY**, **INTERP\_CALC\_DELAY**, and **CONTROLLER\_DELAY** respectively.
  + **PRESSURE\_READ\_DELAY** is the delay in milliseconds between each reading and logging of pressure data from the pressure sensor
    - The shorter the delay, the faster data files generated during tests will grow within a given time frame.
  + **INTERP\_CALC\_DELAY** is the delay in milliseconds between each interpolation between setpoints defined by the trajectory function
  + **CONTROLLER\_DELAY** is the delay in milliseconds between controller actions, based on the most recent trajectory interpolations and pressure readings.
    - It is not recommended to set this value below the mechanical actuation time of your solenoid valve (25ms for the Clippard EFB-1DV-24-L used in the thesis discussing the controller)
* Then, set your PID gain values with **KP**, **KI**, and **KD**.
* Set the error **THRESHOLD** for the trajectory.
  + The smaller this value, the less error between the trajectory setpoint and the measured setpoint it will take to trigger the “failure to follow trajectory” test stop condition.
  + Units for **THRESHOLD** are automatically assigned based on the status of the **USE\_KPA** Boolean.
* Once settings are adjusted to your liking:
  1. plug your computer into the USB port of the Arduino Nano on the electronics stand
  2. press the **upload** button highlighted in green in section 3, then wait for the terminal to display the success message
  3. unplug your computer from the microcontroller
  4. plug your electronics board into a 24V DC power supply
  5. connect your pressure source to the inlet pressure line.
  6. Plug your microSD card into the appropriate slot on the board.
  7. connect your volume line to your actuator, then open the line.
  8. turn the power to the board on.
* Once these steps are completed, the system will open the venting valve and close the inlet pressure to ensure the actuator is not unintentionally overinflated.
* Next, it will initialize the SD card.
  + During this initialization, a csv file is generated with the format of cycle\_test\_#.csv, where # is the first available lowest number not yet stored on the SD card. This file will contain all the data during that test.
    - If tests 1-9 and 11-12 are on the card, 10 will be written
  + If the Arduino is reset, or power to the board is turned off/lost at any point, the controller will consider this a new test, and begin writing to a new file upon start-up/reset.
  + Your specified settings remain stored on the microcontroller until the next upload, so you will only have to reupload code to the computer in the event you want to make new adjust any settings.
* Once the SD card has been initialized, the LCD on the electronics board will prompt you to press the *start button* to begin the test and will wait indefinitely for you to do so.
* After the start button is pressed, the test will begin.
* A blue screen with white text

  Description automatically generatedDuring the test, the total number of completed cycles of your specified trajectory will be displayed on the LCD, along with the file on the SD card that data for this test is being written to. Example shown at right.
* **Write down the name of this test file somewhere you will remember**
* **A blue screen with white text

  Description automatically generated**When the test ends (stop pressed, failure to follow trajectory, or overpressure) the reason for test stoppage will replace the test file name on screen, as shown at right.
* Once a test has ended, power the board off with the switch.
* In all test-ending scenarios, the inlet valve is closed and then the venting valve is opened inside the solenoid.
  + The inside of the actuator volume is always reset to atmospheric pressure and is safe to remove from the pneumatic system.
* Disconnect/close your pressure source.
* Remove the SD card from its port to extract the data stored in the csv files onto your personal computer to process as you see fit.
  + So long as memory is available on the card, the controller program will never overwrite existing data on an SD card. Feel free to perform multiple tests before ever extracting data from the SD card.

# Adjusting Settings & Tuning Your Solenoid

* This section will explain how to tune your controller to the specific solenoid valve used in your set-up.
* Even if you are using the same model of Clippard electronic valve listed in the BOM, **you must still tune the controller to that specific 2/2 solenoid valve**, as all solenoid valves physically vary to a slight degree.
  + If someone else in your group has used this system for tests before, you do not need to go through this tuning procedure, and just use the controller normally.
* To get started, set **TUNE\_PRESSURE** to true as shown, and upload the code to the controller.
  + Ensure the end of your line to test is capped, such as with a closed valve.
* Power the system on before opening the inlet airline to the system
* Open the serial monitor on your computer by clicking the button specified in section 3.
  + The open serial monitor will look something like what is shown below.
* Set the **Baud rate** option to 115200. Connect to the appropriate port on your computer using the **port dropdown menu**.
  + If you are familiar with ArduinoIDE, the PlatformIO’s serial monitor behaves in the exact same way for projects build on the Arduino libraries.
* A screenshot of a computer

  Description automatically generatedWait for the controller to go through its start-up routine; its steps displayed on the LCD on the electronics stand.
* A close up of a device

  Description automatically generatedPress the start button once prompted by the LCD display shown at left.
* You should now see pressure and PWM readings start to appear on the LCD.
  + If nothing is displaying, ensure you have pressed the **start monitoring button**, and are connect to the correct port on your personal device.
* With the pressure and PWM values being displayed at regular intervals in the serial monitor, now simply wait.
* At the point that the valve begins to open, you will see a change in the pressure reading on screen and hear a click or high-pitched whine coming from the solenoid.
* At this point, press the stop button on the electronics stand, but **do not close or reset the serial monitor**.
  + You will want to comb back through this pressure and PWM data to observe at what exact PWM value the valve began to open.
* Set **ANALOG\_PRESSURE\_MIN** to this value and save your change to the code.
* A black background with blue text

  Description automatically generatedRepeat this entire process again, this time setting **TUNE\_PRESSURE** false and **TUNE\_VENT** to true.
  + As part of the vent valve tuning protocol, the volume line will fill for 0.1 seconds before holding both valves shut.
* You will then want to observe the serial monitor for **decreases** in pressure values as the PWM signal increases.
* Once you have determined which PWM value corresponds to the first noticeable decrease in pressure, set that value to **ANALOG\_VENT\_MIN**.
  + For a new Clippard EFB-1DV-24-L, expect the values to be in a nominal range between 120 and 200, and the mins and maxes to be roughly 30-40 values apart from one another.
* Now define the end of the corresponding ranges, also known as **ANALOG\_PRESSURE\_MAX** and **ANALOG\_VENT\_MAX**.
  + Based on prior testing, it is recommended to initially set both these values at their respective MIN value + 30 (i.e. 180, if your min value is 150).
* Once set, start tuning the controller gains to your specific actuator of choice using the normal operating procedures detailed in section 4.
  + If during this PID gain tuning, you find that the controller produces bang-bang control-like hysteresis around your setpoints, consider reducing the PWM range that can be written to the solenoids by reducing the values of either MAX variable as appropriate.
    - bang-bang-like behavior during periods of positively sloping trajectory should be met with decreases to **ANALOG\_PRESSURE\_MAX**, and bang-bang-like behavior during periods of negatively sloping trajectory should be met with decreases to **ANALOG\_VENT\_MAX**.
  + If you instead find that the controller is incapable of producing responses within what you consider a reasonable timeframe, consider increasing the PWM range that can be written to the solenoids by increasing the MAX values when appropriate following the same positive/negative slope checks as above.

# Processing Test Data

* This section explains how to get started processing your test data, using the included MATLAB processing scripts as a jumping off point for your own analysis.
* A screenshot of a spreadsheet

  Description automatically generatedA test file from this controller automatically includes the relevant settings used in the associated test, as well time data in milliseconds, pressure, the controller error and integral values, and a cycle start marker. This is organized as shown:
* Included in the repository for this controller are several MATLAB scripts: **DebuggingDashboard.m**, **SerialPlotting.m**,and **PressureVsTimeOverlap.m**, along with a few helper functions.
* A graph of a diagram

  Description automatically generated with medium confidence**DebuggingDashboard.m** displays the behavoir of the controller over time based on the specific Kp and Ki values used for a given test test to help determine optimal controller behavoir. It produces plots like shown:
* **SerialPlotting.m** generates a similar graph in real time, using the same data used to generate the test files, transferred via serial to a computer. All relevant data is automatically saved to a csv file with the naming scheme fileName\_MATLAB\_#.csv
* A graph of a cycle test

  Description automatically generated**PressureVsTimeOverlap.m** displays all completed cycles from the test overlayed atop one another, using the cycle\_start (normally 0) variable to deliniate the start and end (demarked as 1) of each cycle.
  + This script is helpful for quick data sharing, and for checking cycle-to-cycle consistency of the controller with your acutator
* Both helper funtions additionally plot an **Ideal Trajectory** reference function, which is plotted by defining the trajectory function in the **getTracjectory.m** helper function in the helper functions folder, in the same way as defined in the controller
  + See **Write Custom Trajectory Functions** for more information
* A graph of a cycle test

  Description automatically generatedAdditionally, there is a digital low-pass filter functionality included to allow for curve smoothing for creation of prettier graphs
  + This feature is set to off/false by default