Pneumatic Actuator Test Stand

**Controller Settings Overview**

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

# Description:

This guide walks you through the functions of all variables relevant to the operation and adjustment of the test stand’s PID control software, in the order in which they appear in the main settings file. Any implied or uncontextualized purposes of variables discussed in this overview can likely be explained by reviewing the **Test Stand & Controller Overview** PowerPoint document

Table of Contents

[Description: 1](#_Toc184497260)

[1. Where Relevant Settings Live 2](#_Toc184497261)

[2. Pin Definitions 2](#_Toc184497262)

[3. Pneumatic Valve Settings 2](#_Toc184497263)

[4. Filename Setting 3](#_Toc184497264)

[5. Tuning Settings 3](#_Toc184497265)

[6. Pressure Sensor Settings 3](#_Toc184497266)

[7. Frequency Settings 4](#_Toc184497267)

[8. PID Controller Settings 5](#_Toc184497268)

[9. Trajectory Settings 5](#_Toc184497269)

# A screenshot of a computer Description automatically generatedWhere Relevant Settings Live

* You only need to interact with **adjustableSettings.cpp** to use the controller as originally designed in August 2024.
  + It is located within the **src** sub folder of the downloadable codebase shown at right.
  + For information on how to set up the codebase, see the **Install, Tune, & Run the Controller** Word document.

# Pin Definitions

* A screen shot of a computer program

  Description automatically generatedThe first five variables are static pin definitions for all the valves, sensors, and buttons involved in the controller logic. They should never be changed once initially set, unless the arrangement of your physical electronics has changes too.
  + Change the default definitions listed to those appropriate to your set-up if they differ.
  + **PRESSURE\_PIN** and **VENT\_PIN** must be PWM-capable pins on your microcontroller; if they aren’t the controller won’t be able to open/close the solenoid valves.

# Pneumatic Valve Settings

* A black background with blue text

  Description automatically generatedThese four variables dictate the minimum and maximum analog signals that can be sent to the solenoid valves.
  + The defaults listed in the code are specific to the set-up originally created for the MIT Fabrication-Integrated Design Lab in August 2024. Anytime you create a new version of this set-up with a different solenoid valve, **even with the same exact model as listed in the original BOM**, you must tune the controller program for that specific 2/2 solenoid valve.
  + Due to mechanical wear in the valves, you may need to retune your solenoid valves if you notice consistent, unexpected poor controller behavior.
* Tune the solenoid valve by following the tuning procedure described in the **Install, Tune, & Run the Controller** file.

# Filename Setting

* The **fileName** setting allows the use of custom naming schemes for the test files generated by the controller. There is no restriciton on what can be included in the name, so long as it totals less than **50** characters.
* Test files are generated in the form fileName\_#.csv, where # represents the lowest unused integer whereever the file is being saved.
  + At the start of a test, If cycle\_test\_1, and cycle\_test\_2 already exist, the controller will automatically create a new cycle\_test\_3 file to store the new test’s data

# A black background with blue text Description automatically generatedTuning Settings

* **TUNE\_PRESSURE** and **TUNE\_VENT** should always be set to false, unless the system’s pressure and vent valves are being tuned as outlined in section 5 of the **Install, Tune, & Run the Controller** guide.

# A screen shot of a computer Description automatically generatedPressure Sensor Settings

* **USE\_SD\_CARD** dictates where the data the controller generates during a test is sent.
  + If set to true, the controller writes all test data to a new csv file on the SD card.
  + If false, writes all test data to the serial port, where it can be intercepted and plotted in real-time by the **LiveSerialPlotting.m** script.
    - This will save all data to excel files of the form fileName\_MATLAB\_#.csv
    - Allows for quicker gain-tuning workflows
* The second of the five pressure-related settings is the **USE\_KPA** Boolean.
  + If set to true, will interpret all pressure related values and variables in Kilopascals.
  + All pressure data coming from the pressure sensor will automatically be interpreted in units of Kpa and will be written to the microSD card that way as well.
  + If **USE\_KPA** is set to false, the system behavior will remain the same, but use units of **PSI**.
* **FILTER\_ALPHA** is passed to the pressure sensor collection logic and implemented as the coefficient in a simple low pass filter used to smooth the pressure data which is eventually fed to both the PID controller and recorded to the SD Card. \
  + During preliminary testing, it was found that even a filter alpha as low as 0.01 would make the controller noticeably more unstable for negligible smoothing benefit.
  + It is typically **best to leave the controller off** by setting the alpha equal to 0.
* **OVERPRESSURE\_LIMIT** serves as a safety measure to prevent accidental overinflation of actuators.
  + Set the variables just under the safe operating limit of your actuator.
    - If your actuator bursts at 30 PSI, you will likely want to set this value somewhere around 25 PSI.
  + This setting directly interferes with burst testing trajectories and can lead to premature test endings, so be sure to alter along with your trajectory as appropriate.
  + **It is not recommended to manually set a pressure max by adjusting the regulator down**, as a lower inlet pressure differential will lead to a worsened ability for the controller to follow its programmed trajectory.
* **SENSOR\_OFFSET** exists to allow calibration of the pressure sensor used in the system. Once set for a specific pressure sensor
  + It will not need to be adjusted again.
  + The offset will be different for every sensor
  + Recommend finding the offset for a specific pressure sensor using simple Arduino code that prints pressure values to Serial while the sensor sits at normal atmospheric conditions.

# Frequency Settings

* These three settings determine the rate at which the microcontroller performs the following tasks:
  + Reads and writes the pressure data.
  + Writes other controller and cycle-related variables to the microSD card.
  + Performs trajectory interpolation (refer to the trajectory section for more information).
  + Updates the output of the PID controller.
* These values are frequency settings, but they must be set in units of milliseconds.
* It is recommended to keep these variables at their default values based on initial performance testing.
* If you do change these values, it is recommended to:
  + Keep **INTERP\_CALC\_DELAY** and **CONTROLLER\_DELAY** at the same value. Performance seems to improve when both actions occur in sync.

# A black screen with white text Description automatically generatedPID Controller Settings

* These 6 variables relate to the overall behavior of the proportional-integral-derivative (PID) controller and its outputs.
* **THRESHOLD** represents the range outside of which a test will fail if the controller fails to follow a non-zero setpoint.
  + If you set your trajectory function as a 15 PSI step function and set a 2 PSI threshold, then the test stopping safety logic will activate if the controller fails to stay within a range of 13-17 PSI **for more than 2 seconds***.*
  + The added time delay prevents disturbances from abruptly ending tests early. It is recommended to set this threshold to 50% of your operating pressure for long form cycle or burst testing, to be able to characterize actuator behavior after material failure.
* **OUTPUT\_MIN** and **OUTPUT\_MAX** are helper variables that are used to map the outputs of the PID controller to the PWM signals defined in section 3 of this guide.
  + **Do not change these variables** from their defaults if you are not doing major restructuring to the code’s logic.
* **KP**, **KI**, and **KD** represent the gains for the proportional, integral, and derivative controller actions respectively.
  + Preliminary testing suggests starting with a **KP** value on the order of 10-1, and a KI value on the order of 10-3
  + Do not involve derivative control (i.e., keep KD at 0) given its tendency to introduce high frequency instability into PneuNet systems.

# Trajectory Settings

* The last two variables listed in the file concern defining the trajectory settings of your controller.
  + For more definition on how to develop a trajectory function for the actuator to follow, see the **Write Custom Trajectory Functions** word document.
* **TIMES** is an array of time setpoints expressed in milliseconds, and **PRESSURES** contains the associated pressure setpoints for each time instance.
  + **Ensure that these arrays are of the same length**, else the program will fail to run properly.
* For convenience, example Step, Sawtooth, Reverse Sawtooth, Triangle, Sine, Ramp, and Burst trajectories have been included in the file as starting points for the development of your own custom trajectories.
  + Because these variables are defined as constant (const) in the program, only **one** set of definitions for **TIMES and PRESSURES** can be uncommented at once, else the code will fail to compile.