Pneumatic Actuator Test Stand

**Writing Custom Trajectory Functions**

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

# Description:

This guide will walk you through the details and intricacies of the trajectory functions used by this controller, which define the pressure profile the controller attempts to follow through each cycle of the controller

Table of Contents

[Description: 1](#_Toc175398957)

[1. What is a Trajectory Function? 2](#_Toc175398958)

[2. Writing Custom Trajectory Functions 3](#_Toc175398959)

# A graph of pressure and pressure Description automatically generated What is a Trajectory Function?

* A trajectory function, as defined for this controller logic, is a pressure profile that is defined by the user, and instructs the controller what pressure the system should be targetting at any given point in time.
  + The user defines this function by filling **TIMES** and **PRESSURES**, two arrays of equal sizes, in the **adjustableSettings.cpp** file.
  + Both variables are passed to the Trajectory object in **main.cpp** to be used in tandem with the autoPID controller object.
    - The times are to be listed in milliseconds, and the pressures will be interpreted either in Kpa or PSI values, depending on whether or not the **USE\_KPA** boolean is set to true or false respectively.
* These trajectory functions can be very simple, or very complex, depending on the shape of the curve one wishes their pressure profile to follow.
  + The simpler the curve—such as the step function shown above—the less points one needs to use to define it.
* Once the last pair of a trajectory’s setpoints is reached, the controller automatically loops back to the beginning of the cycle function. This allows for long form cycle testing.
  + Long form cycle testing can be used to characterize an actuator’s behavior over its usable lifetime as its material components wear with extended use.
* The Trajectory class’s **interp()** function allows the controller to know what to set the pressure to in between defined.
  + **Interp()** takes in a variable **deltaT**, which is the time since the start of the current cycle and compares it against the values in the **TIMES** array. ‘
  + The **interp()** function determines which two coordinates **deltaT** is between, and linearly interpolates a corresponding pressure value for that time value from the **PRESSURES** array.
  + The linear interpolation equation used is as shown:
* Structuring the trajectories this way has several benefits:
  + Long hold times at a constant pressure can be written as simply two-time setpoints with equal pressure values
  + allows for quick modification, iteration, and development of different or evolving trajectory profiles as relevant to your task.

# Writing Custom Trajectory Functions

* You will more than likely want to write your own trajectory functions different than the ones included in the controller.
* Several example structures for various control signal shapes such as step, triangle, sawtooth, and sine waves are included by default in **adjustableSettings.cpp** and can be uncommented and adjusted to fit your needs.
  + Please see the **Test Stand & Controller Overview** PowerPoint for visualizations of these example signals at varying pressure magnitudes.
  + You do not have to only follow these curves or signal styles in the development of your own. They are meant to serve as handy starting points and building points.
* The possible trajectories one could make for this controller are virtually limitless, unconstructed by time.
  + The only limitations in trajectory creation are the limits of your hardware, and the memory size of the microcontroller.
  + The code has been structured such that there is ample memory space available in the program to develop highly complex setpoints beyond what is showcased by the documentation.
* The only major caveat to the creation of trajectory functions is how you begin and end them. Unless you plan to stop the test before one full cycle completes, the cyclic nature of the controller means that the moment the trajectory ends, the controller begins interpolating between the first and second setpoint of the trajectory during the next cycle immediately. As such, for smooth cycling, it is recommended that you **set the first and last setpoints of your PRESSURES array equal**.
  + It is further recommended that you set the first and last pressure values to 0. While the controller is certainly capable of handling transitioning between the end of a cycle and the start of the next with non-zero setpoints, this will likely result in some controller instability during the first few cycles, depending on the time-length of your trajectory function.