How to Combine S-curve Moves with CML

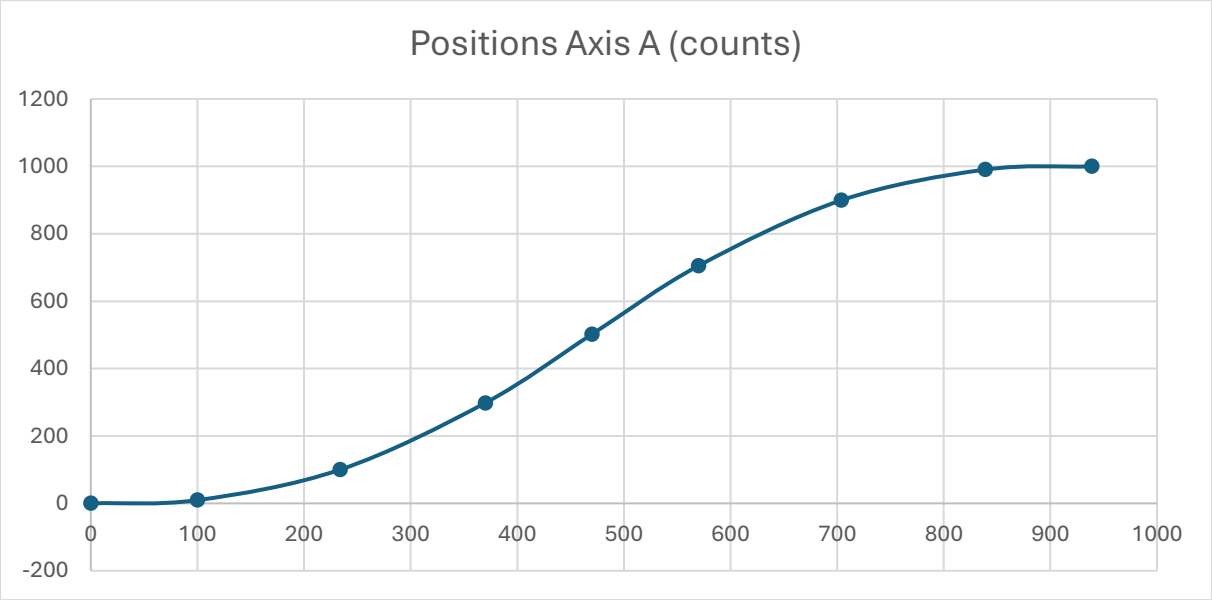
First, review the C++ example code on the Copley Controls GitHub page.

```
double velocity = 10000; // firmware units = 0.1 counts/sec
double accel = 10000;    //                = 10 counts/sec^2
double decel = 10000;    //                = 10 counts/sec^2
double jerk = 100000;    //                = 100 counts/sec^3
```

PVT table for first move on Axis A:

Time Absolute (ms)	Positions Axis A (counts)
0	0
100	9.6225
234	100.139
370	298.009
470	501.765
570	705.043
704	899.495
839	990.566
939	1000

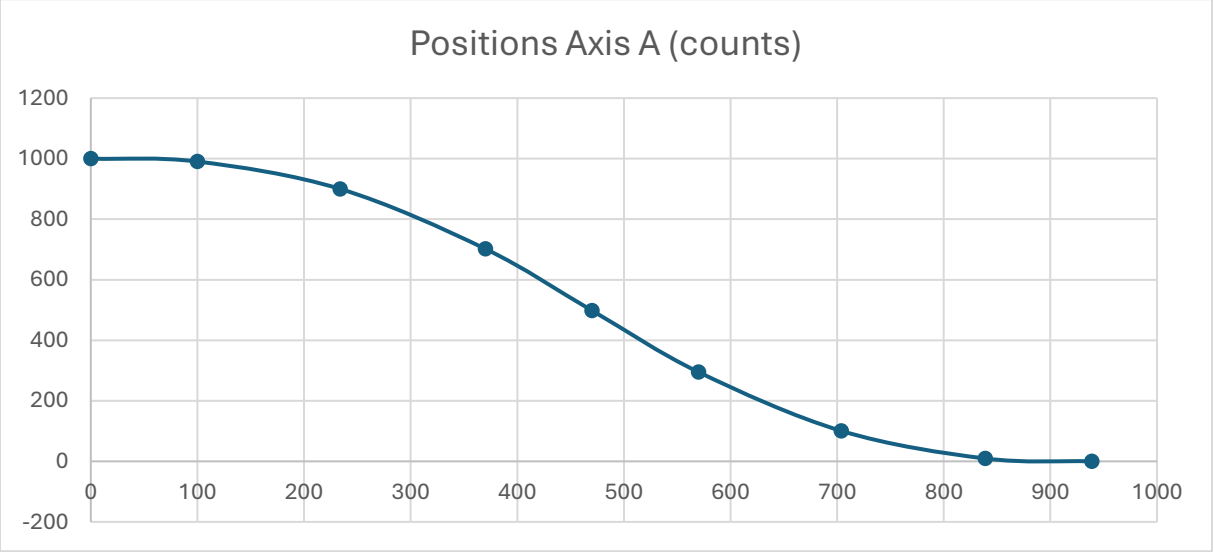
First move on Axis A:



PVT table for second move on Axis A:

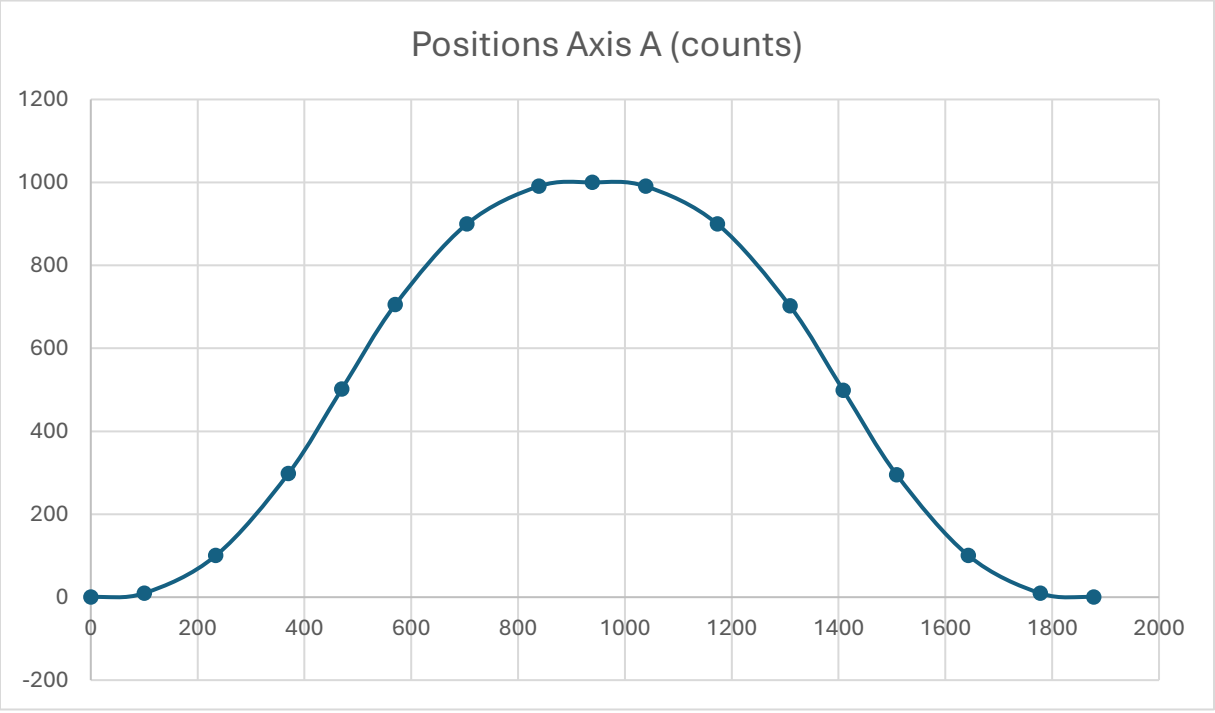
Time Absolute (ms)	Positions Axis A (counts)
0	1000
100	990.377
234	899.861
370	701.991
470	498.235
570	294.957
704	100.505
839	9.43432
939	0

Second move on Axis A:

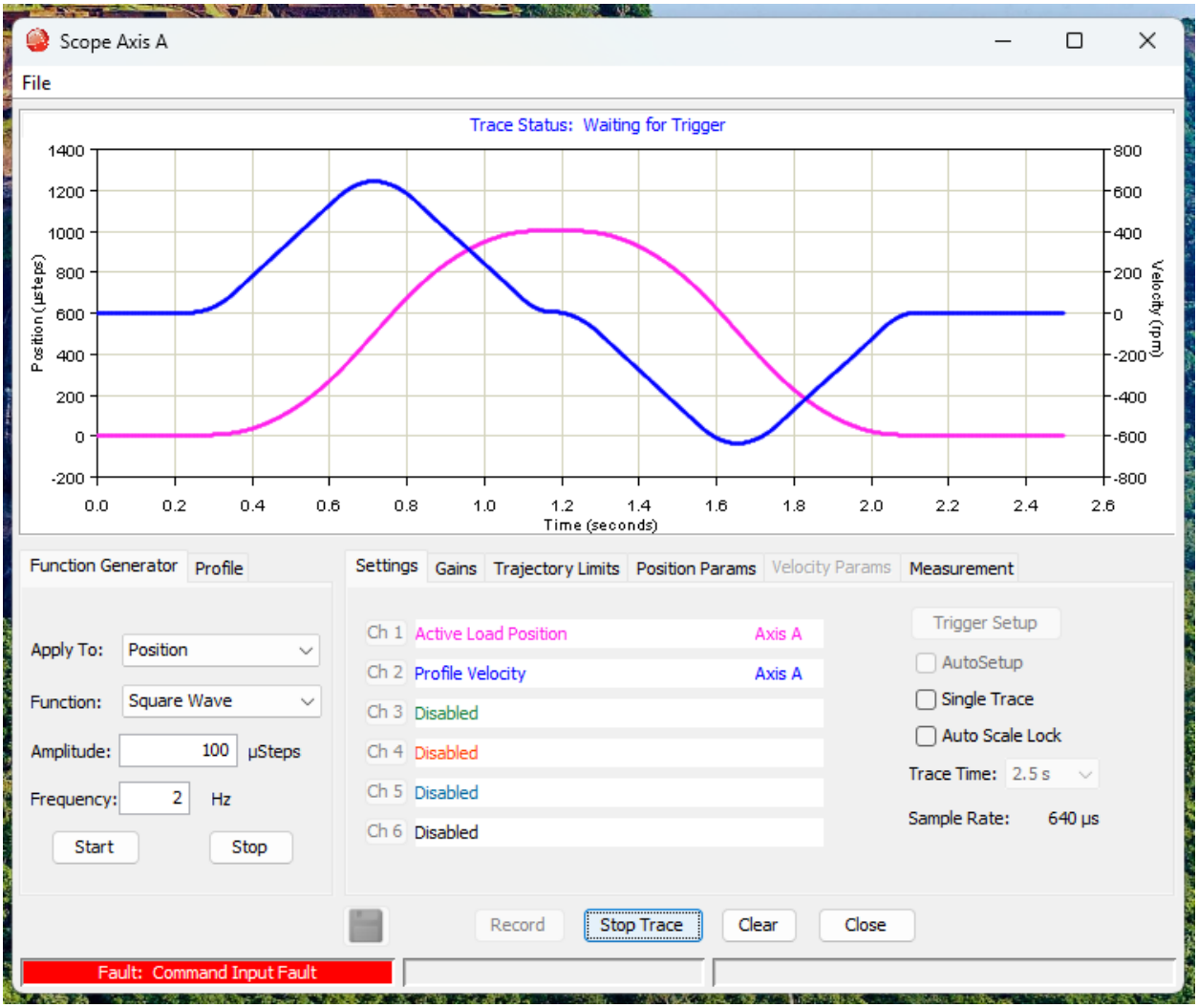


When we combine these two moves, we can erase the last point of the first move because it is the same as the starting point for the second move.

Time Absolute (ms)	Positions Axis A (counts)
0	0
100	9.6225
234	100.139
370	298.009
470	501.765
570	705.043
704	899.495
839	990.566
939	1000
1039	990.377
1173	899.861
1309	701.991
1409	498.235
1509	294.957
1643	100.505
1778	9.43432
1878	0



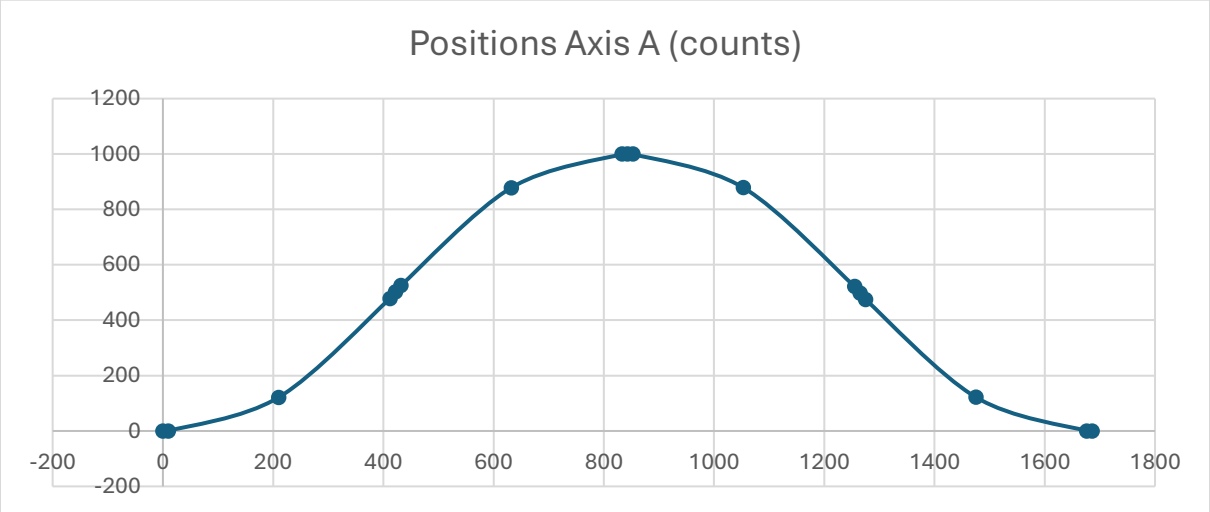
In a real system, here is the performance:



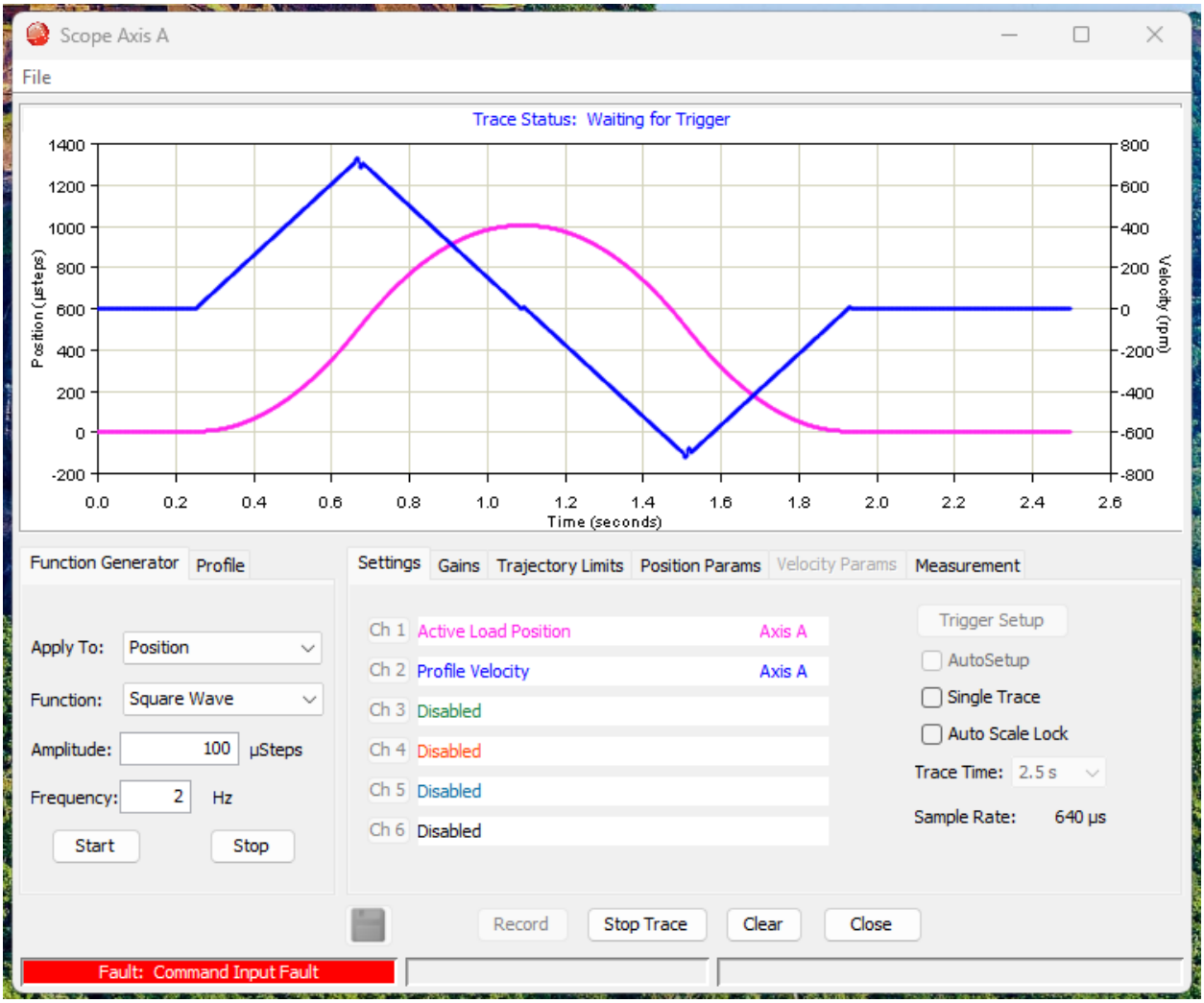
Too much settling time between the two moves? Increase your jerk value in CML.

```
double velocity = 10000; // firmware units = 0.1 counts/sec
double accel = 10000;    //                = 10 counts/sec^2
double decel = 10000;    //                = 10 counts/sec^2
double jerk = 1000000;   //                = 100 counts/sec^3
```

Time Absolute (ms)	Positions Axis A (counts)
0	0
10	0.096225
210	121.34
412	478.211
422	501.877
432	525.498
632	878.165
833	999.92
843	1000
853	999.904
1053	878.66
1255	521.789
1265	498.123
1275	474.502
1475	121.835
1676	0.0803922
1686	0



Here is what this profile looks like in a real system.



The s-curve algorithm in CML is trying to maintain the s-curve profile despite being acceleration limited. That is why there are 2 extra points adjacent to the apex (highlighted in yellow). Let's manually remove them to refine the transition between the two moves.

New Table:

Time Absolute (ms)	Positions Axis A (counts)
0	0
10	0.096225
210	121.34
412	478.211
422	501.877
432	525.498
632	878.165
843	1000
1053	878.66
1255	521.789
1265	498.123
1275	474.502
1475	121.835
1676	0.0803922
1686	0

Now we can manually define these points in a vector in C++ and add them to the PvtObj in CML.

```
    vector<vector<double>> manuallyEditedPositions =
    {
        {0.0, 0.0, 0.0},
        {0.096225, 0.096225, 0.096225},
        {121.34, 121.34, 121.34},
        {478.211, 478.211, 478.211},
        {501.877, 501.877, 501.877},
        {525.498, 525.498, 525.498},
        {878.165, 878.165, 878.165},
        {1000.0, 1000.0, 1000.0},
        {878.66, 878.66, 878.66},
        {521.789, 521.789, 521.789},
        {498.123, 498.123, 498.123},
        {474.502, 474.502, 474.502},
        {121.835, 121.835, 121.835},
        {0.0803922, 0.0803922, 0.0803922},
        {0.0, 0.0, 0.0}
    };

    vector<uint8> manuallyEditedTimes = { 10, 200, 202, 10, 10, 200, 211, 210, 202,
10, 10, 200, 201, 10, 0 };

    LoadPointsIntoPvtObj(manuallyEditedPositions, manuallyEditedTimes, pvtObj);

    err = link.SendTrajectory(pvtObj);
    showerr(err, "starting the move");

    err = link.WaitMoveDone(-1);
    showerr(err, "waiting for move to finish");
```

This is an example of manual trajectory editing. The resulting profile velocity looks like more of a straight line during the transition between the two moves.