AN141

16-127174

PVT Trajectory Calculations

Introduction

Copley servo drives support a trajectory generation mode know as PVT interpolation. PVT stands for Position, Velocity, and Time. In this mode, the master controller provides the trajectory sequence to the drive as a set of points and the drive interpolates between the given points. This allows the master controller to specify an arbitrary move which the drive will smoothly follow.

Each point provided by the master includes the following information:

Position	The desired position of the motor at the start of the point. This is specified in units of encoder counts.
Velocity	The desired velocity of the motor at the start of the point. This is specified in tenths of encoder counts / second. For example, a value of 1234 would indicate a velocity of 123.4 cts/sec.
Time	The time in milliseconds until the start of the next segment. A value of zero indicates the end of the move.
Flags	Some additional information which can be used to modify how the passed data is interpreted by the drive. Currently supported flags: • Relative position flag. If set the position information supplied with the PVT point is treated as a change in position since the previous point. If not set the the position is considered absolute. • Interpolation mode. If set, the drive will ignore the velocity value passed and perform linear interpolation when approaching this segment. If clear, the drive will use cubic polynomial interpolation between the segments.

PVT points sent by the master are stored in a buffer in the drive. When the PVT move is executing, the drive will remove points from the buffer as they are needed. The master will supply additional points during the move to keep the buffer from emptying until the end of the move. The last point supplied by the master will hold a zero time value which indicates to the drive that the move is finished when that point is attained. Once the move ends the drive will stop consuming points from the buffer until the next PVT move is started.

PVT moves via binary serial

It's possible for the master to send PVT points to the drive using the drive's binary serial interface. The details of the binary serial interface are covered in another document, but the general procedure is as follows:

- The drive should be configured in programmed position mode via CME.
- The trajectory configuration (parameter 0xC8) should be set to a value of 3. This configures the drive for PVT interpolation trajectory mode.
- The master sends the PVT points to the drive by writing to serial parameter 0x115. This parameter is used to add points to the drive's internal trajectory buffer which is large enough to store a total of 32 PVT points. That buffer size may be increased in future firmware versions.
- Once the master has written the desired number of points to the buffer, the trajectory is started by sending the start move command (op-code 17). This causes the drive to start the trajectory.
- The drive will consume points from the buffer. The master must continue to write new points to the buffer during the move until the final point is written. The final point of the move is identified by the zero time value included with that point.

Parameter 0x115 is used to write points to the drive's internal trajectory buffer. The parameter is written as a 16-bit word followed by up to two 32-bit integer values.

The first 16-bit word written to parameter 0x115 holds either the PVT time value or a command used to manipulate the buffer. When used to send PVT time information, the word is formatted as follows:

Bits	Contents	Description
0-7	Time	PVT segment time in millisecond units. The maximum length of a PVT segment is 255 milliseconds long. A time value of 0 is used to denote the end of a move.
8-11	Reserved	These bits should be written as zero.
12	Relative	If this bit is clear, then the position data sent with the move is an absolute position. If this bit is set, then the position value is relative to the position in the previous PVT point. For the first PVT point in a move, if the position data is sent as a relative position then it's the distance from the commanded position at the time the move is started.
13-14	Reserved	These bits should be written as zero.
15	Zero	This bit must be zero when writing PVT time information.

For PVT buffer manipulation commands, the 16-bit word value is formatted as follows:

Bits	Contents	Description
0-7	Command data	The meaning of the value sent in these bits depends on the command code.
8	Command code	These bits identify the command being sent and the meaning of bits 0-7. Legal command codes are: 0 – Clear the buffer and abort any PVT moves in progress. 1 – Remove the N most recently added points from the buffer where N is the value passed in bits 0-7.
9-14	Reserved	These bits should be written as zero.
15	One	This bit must be set for PVT commands.

When sending buffer manipulation commands no additional data follows this 16-bit word.

When writing PVT data, either one or two 32-bit integer values must follow the initial 16-bit word holding the PVT time.

The first 32-bit integer gives the position (in encoder counts) corresponding to this PVT point. The position can either be an absolute position value, or a relative distance from the previous point (or start of move) depending on the value of bit 12 in the previous word.

The second 32-bit value gives the velocity in 0.1 encoder count/second units. This value is optional and if omitted will cause the drive to use linear interpolation in the segment approaching this PVT point. If the velocity is given, then the drive will use cubic polynomial interpolation.

Reading parameter 0x115 will return three 16-bit words of data. The first word holds the number of free positions in the trajectory buffer. The following two words are reserved and should be ignored.

PVT mode in CANopen / EtherCAT

PVT mode is used when running in interpolated position mode via either CANopen or EtherCAT. Interpolated position mode is a standard CANopen mode of operation which is more thoroughly discussed in the CANopen programmer's guide.

When running in Interpolated position mode via CANopen, there are two sets of objects which can be used to add points to the drive's internal buffers. One set is intended to conform to the DS402 CANopen specification, the other is a Copley specific set of objects which have some advantages over the standard objects.

Standard DS402 objects

These objects are defined in the DS402 CANopen specification and are summarized here.

Object 0x60C0 – Interpolation sub-mode select:
This 16-bit object selects the type of interpolation mode used by the drive. DS402 only defines one standard value for this object (0) which means linear interpolation. Values 1 through 32767 are reserved by the standard. Values -1 through -32768 are for vendor specific modes. The following values are supported by Copley drives: 0: In this mode linear interpolation will be used between points. Additionally, when this mode is selected the drive will clear the trajectory buffer when the PVT move is first started deleting any points added before that time. The move won't actually start until three new points have been added by the master. The time value used between points (specified using object 0x60C2) will be adjusted by the drive by +/- 1 servo period if necessary to try to keep the number of points in the buffer equal to 3 at all times. This mode is mostly useful if the master controller will be adding points to the drive at a constant rate, but may not be well synchronized with the drive's internal clock. The drive will make small adjustments to the PVT time value to try to prevent the buffer from overflowing or underflowing. -1: This mode also uses linear interpolation between points, but doesn't flush the trajectory buffer when the move is started, or manipulate the time value like mode 0 does. It's useful when the master is synchronized to the drive and linear interpolation is desired. The time value for every segment is constant in this mode and written to object 0x60C2. -2: This mode also uses linear interpolation, but the time value for each segment is written along with the positions to object 0x60C1. The value written to 0x60C2 is ignored in this mode. -3: This mode also uses cubic polynomial interpolation with unique time values for each segment.

Object 0x60C1 – Interpolation data record

This object is used to write PVT data to the drive's trajectory buffer. It has several sub-indices which are used to write the different data values.

Sub-index 1: This 32-bit sub-index is used to write the position data in all sub-modes. For sub-modes 0 and -1 which use linear interpolation with fixed times, writing a value to this sub-index will immediately cause the value to be added to the drive's trajectory buffer. Positions are written in units of encoder counts.

Sub-index 2: This 8-bit sub-index is used to write the time value in millisecond units for sub-modes -2 or -3. In sub-mode -2 writing a time to this sub-index will cause the position (from sub-index 1) and time to immediately be written to the drive's trajectory buffer. This sub-index should not be written when running in modes 0 or -1.

Sub-index 3: This 32-bit value is used to write the velocity values for use in sub-mode -3. Writing this sub-index will cause the PVT data to be added to the buffer. This sub-index shouldn't be written in any mode other then -3. Velocities are written in units of 0.1 counts/second.

When running in sub-mode 0 or -1 the master simply writes each position to sub-index 1 of this object. Each time a new position is written, it will be stored in the drives trajectory buffer.

When running in sub-mode -2, the master should write positions to sub-index 1 first, then write the time that goes along with that position to sub-index 2. In this mode the data is copied to the drive's buffer when sub-index 2 is written.

When running in sub-mode -3, the master should write the position and time to the first two sub-indexes first, then write the velocity last. In this mode the copy to the drive's trajectory buffer is triggered by the write to sub-index 3.

Object 0x60C2 – Interpolation time period

This object stores the fixed time period for use in sub-modes 0 and -1. The times are given in scientific notation using two sub-indexes.

Sub-index 1: This 8-bit unsigned value gives the number of time units.

Sub-index 2: This 8-bit signed value gives the time unit value. It's specified as 10^x seconds where x is the value written. Legal values range from -3 for milliseconds to -6 for microseconds.

Normally sub-index 2 is left at it's default value of -3 and sub-index 1 is used to write the PVT period in millisecond units. For example, setting sub-index 1 to 15 and sub-index 2 to -3 would mean a time period of $15x10^{-3}$ seconds, or 15ms.

Copley specific objects

An alternative to using the standard objects described above is to use the following Copley specific objects to add points to the buffer.

The main advantages of the Copley objects is that a PVT point can be added using a single write to object 0x2010. This object is designed to fit in a CANopen PDO message which allows points to be added very quickly and efficiently over CANopen.

Object 0x2010 – PVT data object
This object is 8 bytes long which the maximum length of a CANopen datagram. The first byte of data written to this object holds a command or formatting information. It defines how the remainder of the command is interpreted. The remaining seven bytes hold the position, velocity, and time information for PVT segments. Details of this object can be found in the CANopen programmer's manual, available for download on Copley's web site.

Object 0x2011 – Trajectory buffer free count
This 16-bit object gives the number of free spaces available in the trajectory buffer.

Object 0x2012 – Trajectory buffer status
This 32-bit object gives detailed information about the trajectory buffer status including the number of free positions and any latched errors (underflow, overflow, etc). See the CANopen programmer's guide for details.

Linear interpolation

When points are specified without velocity information the drive will use simple linear interpolation to interpolate between the points. This is often referred to as PT mode as opposed to PVT mode since no velocities are needed. With linear interpolation the drive will calculate a new velocity for each set of points and will use this velocity value to update the trajectory position every servo cycle.

Linear interpolation has the advantage that it's very easy for the master to calculate the data to send to the drive because there are no velocities to calculate. It does not produce as smooth motion as is possible with cubic polynomial interpolation however.

Cubic Polynomial Interpolation

The normal type of interpolation used by the drive during PVT trajectory processing is called cubic polynomial interpolation. When using this interpolation mode the drive calculates the coefficients of a cubic polynomial which it then uses to smoothly interpolate between the position of the starting point to the position of the ending point. The velocity at the start and end of the segment will also smoothly transition between the values provided by the master.

The formula used to calculate the polynomial coefficients is very simple. The inputs to the calculation are the position and velocity at the start (P0,V0) and end (PT,VT) of the segment, and the time (T) between the two segments. Given those inputs, the drive calculates the four coefficients of a cubic polynomial that will transition between the specified (p,v) states in the time required:

$$p(t) = b0 + b1*t + b2*t^2 + b3*t^3$$

where 0 <= t <= T, and b(0..3) are the coefficients of the polynomial.

To calculate the coefficients, we first will find the first derivative of the polynomial:

$$p'(t) = v(t) = b1 + 2*b2*t + 3*b3*t^2$$

Now, we already know p(0) and v(0) which were given, so solving the two equations with t = 0 gives us:

$$p(0) = b0 = P0$$
$$v(0) = b1 = V0$$

We can solve for b2 and b3 by substitution at time T:

Rewrite the p(t) formula solving for b2:

$$b2 = (PT - P0 - V0*T - b3*T^3) / T^2$$

substitute that into the p'(t) formula:

$$VT = V0 + 2*T*[(PT-P0-V0*T-b3*T^3) / T^2] + 3*b3*T^2$$

Now, solve for b3:

$$VT - V0 = 2*(PT-P0)/T - 2V0 - 2*b3*T^2 + 3*b3*T^2\ VT$$
$$+ V0 = 2*(PT-P0)/T + b3*T^2$$
$$b3 = (VT+V0)/T^2 + 2*(P0-PT)/T^3$$

Now, substitute back into the p(t) formula to find b2

$$PT = P0 + V0*T + b2*T^2 + T^3*[\ (VT+V0)/T^2 + 2*(P0-PT)/T^3\ ]\ PT$$
$$- P0 = V0*T + b2*T^2 + T*(VT+V0) + 2*(P0-PT)$$
$$b2*T^2 = (PT-P0) - V0*T - (VT+V0)*T + 2*(PT-P0)$$
$$b2 = (3*(PT-P0) - V0*T - VT^2 - V0*T)\ /\ T^2$$
$$b2 = 3*(PT-P0)/T^2 - (2*V0+VT)/T$$

To summarize, our 4 coefficients are:

$$b0 = P0$$
$$b1 = V0$$

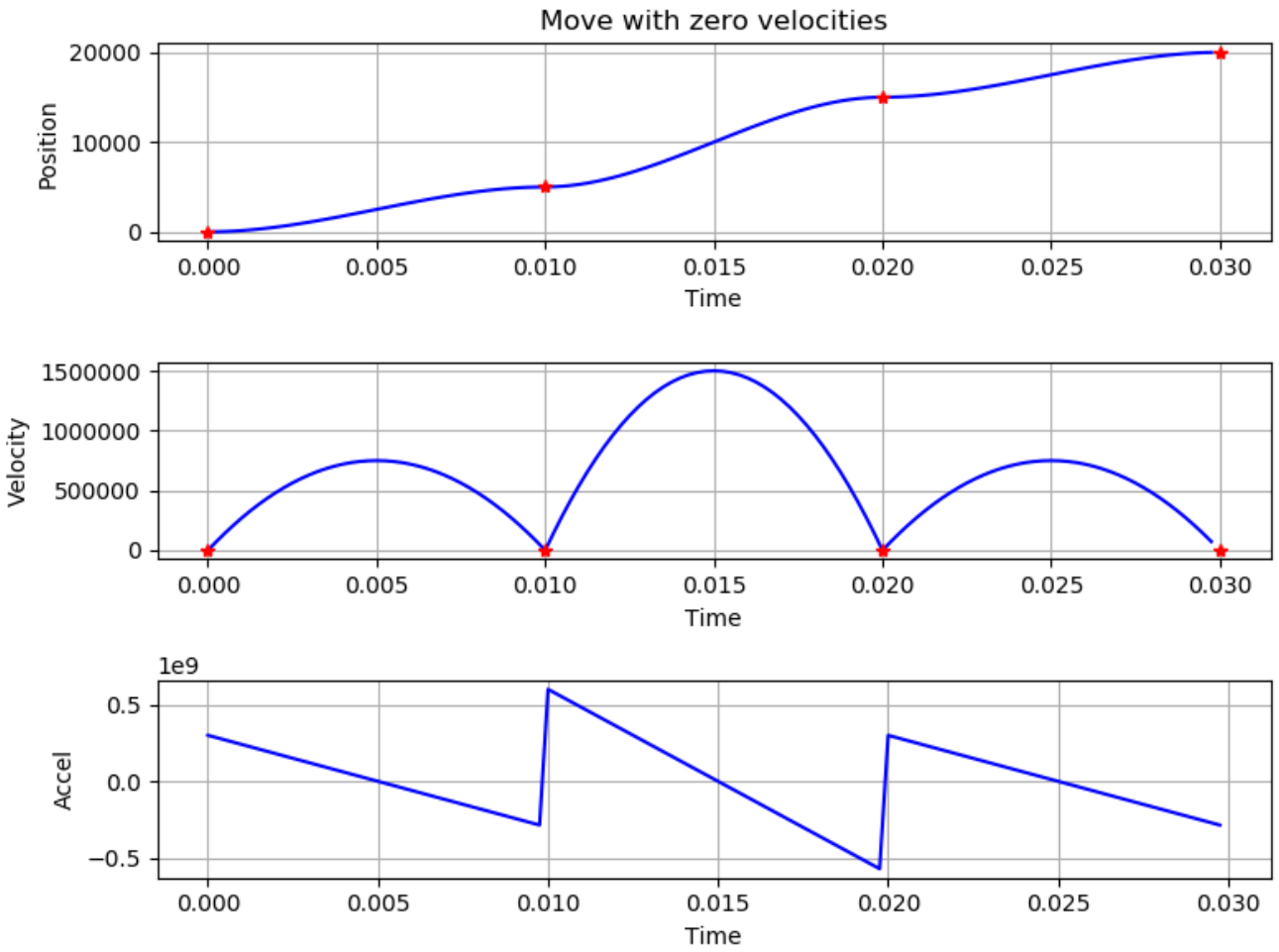
$$b2 = 3*(PT-P0)/T^2 - (2*V0+VT)/T$$
$$b3 = (VT+V0)/T^2 + 2*(P0-PT)/T^3$$

So, between any two points in a PVT trajectory the drive calculates these four coefficients (b0, b1, b2, b3), and uses them to determine the commanded position and velocity during that segment. By the time the second point is reached the position and velocity will exactly equal the values given for that point. The drive then starts over by calculating new coefficients using the second and third points in the PVT, etc.

By definition, the position and velocity during the move will be continuous. Acceleration will change linearly during each segment, but may jump to a new value at the start of the next segment. Jerk (the rate of change of acceleration) will be a constant value for each segment in the PVT profile.

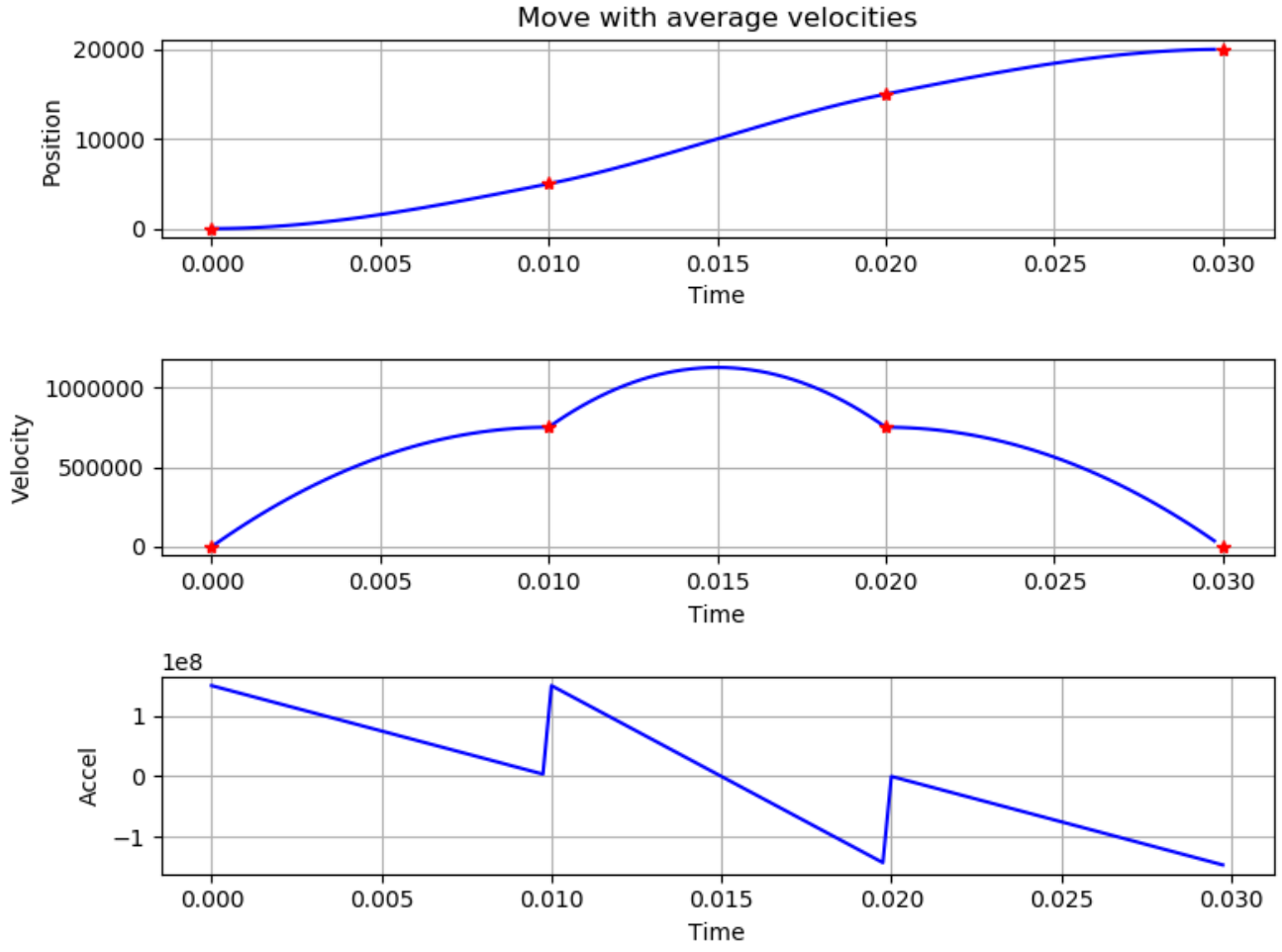
The quality of the resulting move will be entirely determined by the PVT data input by the master. For example, the following graphs each show the interpolated results of a short PVT move consisting of three segments between the positions [0, 5000, 15000, 20000]. The time between the given points is 10ms for each graph. The only difference is the velocity values provided with those positions.

First, an example of what the drive would interpolate if all the velocity values were zero. The drive will happily interpolate between these points, but the resulting trajectory will be quite rough. This graph shows the position, velocity and acceleration values for the three segments between the four points:



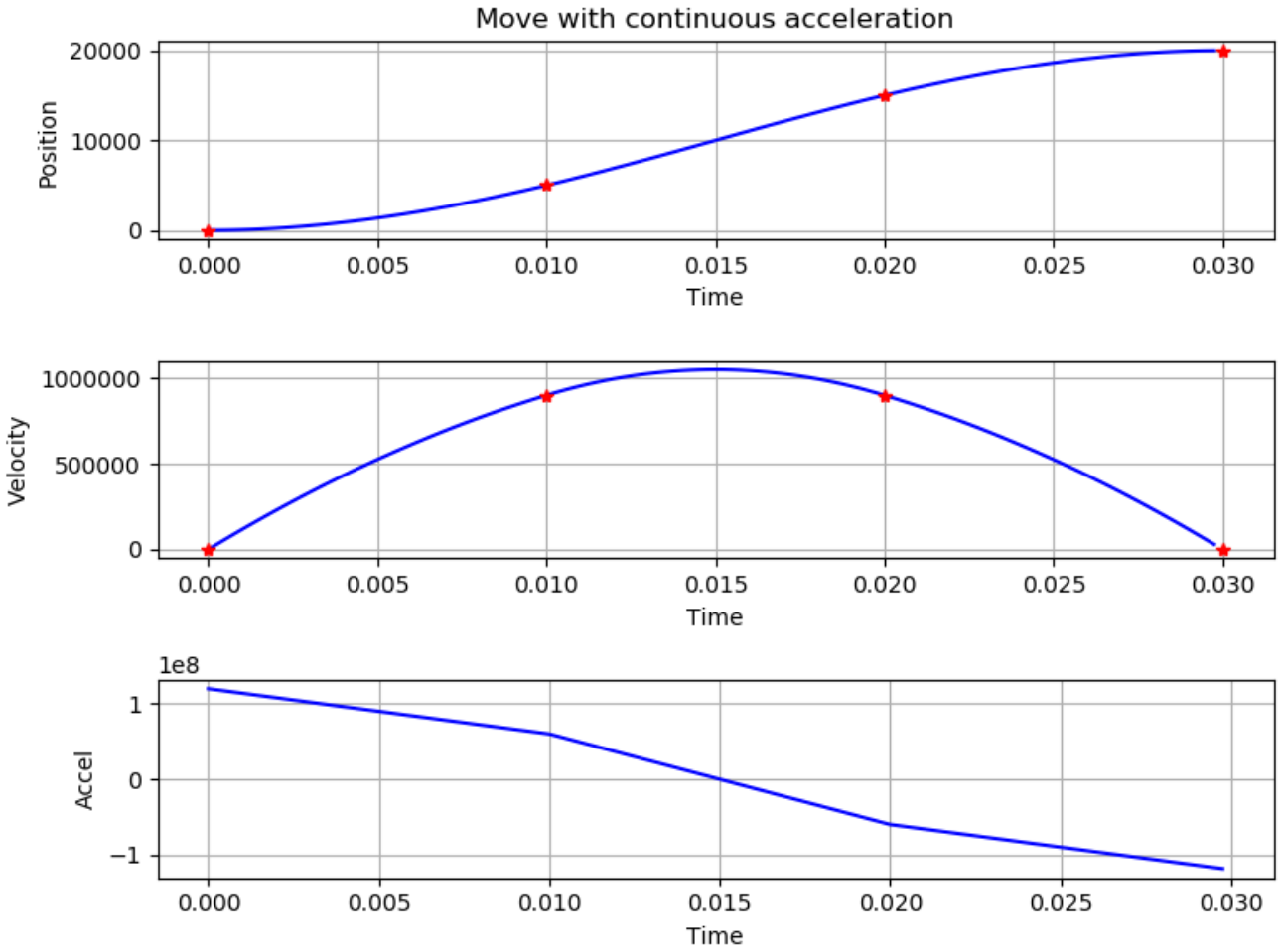
In the position and velocity graphs the points given by the master are identified with a red asterisk. The blue line shows the interpolated data between the points. Notice that the drive is correctly moving between the given data points and that position and velocity are continuous. The interpolated acceleration value moves linearly between points, but jumps to a new value at the start of each segment.

A smoother move between the same set of positions [0, 5000, 15000, 20000] can be made by calculating better velocity values to go with each position. In the next graph we use zero velocities for the start and end of the move, but use a simple average for the internal velocities. For example, the velocity passed with the second point (P1 with position 5000) is P2-P0 / (2T), i.e. (15000-0)/0.020 or 750000.



This is a much smoother looking move then the previous one. As before, the position and velocity points are hit exactly by the drive. The acceleration still changes abruptly at the start of each point, but the change is less severe then in the previous example.

If we were to very carefully calculate our velocity values then we could produce an even smoother curve between the given points. In the next graph the velocities were chosen such that the acceleration values calculated by the drive would be continuous from the end of one segment to the start of the next segment:



As always, the positions and velocities (marked in red) are hit exactly by the interpolation calculations. The accelerations happen to be continuous between the points due to the way the velocities were calculated for this move.

Calculating velocity for continuous accelerations

The velocity values used in the last graph above were calculated in such a way that the resulting accelerations would be continuous between points in the move. This is just one possible way to calculate PVT points which would result in a relatively smooth move between a set of positions.

As described previously, the coefficients of the polynomial used when interpolating between sets of points are calculated for each set of points (P(n),V(n)) to (P(n+1),V(n+1)) as:

$$b0 = P(n)$$
$$b1 = V(n)$$

$$b2 = 3*(P(n+1)-P(n))/T^2 - (2*V(n)+V(n+1))/T$$
$$b3 = (V(n+1)+V(n))/T^2 + 2*(P(n)-P(n+1))/T^3$$

Where P(n), V(n) are one supplied PVT point and P(n+1), V(n+1) are the next given point. T is the time between those two points.

Since the position over that segment is a cubic polynomial using those four coefficients, we can calculate the formula for the velocity, acceleration, and jerk over that segment by differentiation:

$$V(t) = b1 + 2*b2*t + 3*b3*t^2$$
$$A(t) = 2*b2 + 6*b3*t$$
$$J(t) = 6*b3$$

Here we can see that Jerk is constant over each segment and acceleration changes linearly during the segment.

The coefficients for each segment are calculated to ensure that the position and velocity at the end of one segment are equal to the given position and velocity at the start of the next. Position and velocity will therefore always be continuous during a PVT move.

The acceleration resulting from the PVT calculations will change linearly during a segment, but may jump to a new value at the start of the next segment. These discontinuities in acceleration will result in rough motion and are generally undesirable.

If we have a set of points that we want to move through and need to calculate velocities for them, then one possible method of calculating those velocities would be to calculate them in such a way that the accelerations remain continuous over the course of the move.

Lets say we have just three points (P0,V0), (P1,V1), and (P2,V2) and a constant time T between each pair of points. Lets further say that we know V0 and V2 and want to find V1 such that the acceleration remains continuous over those points.

We know from above that the acceleration over a segment will be

$$A(t) = 2*b2 + 6*b3*t$$

where t ranges from 0 to T.

If we set the ending acceleration of one segment equal to the starting acceleration of the next segment, then the resulting accelerations will be continuous over those two segments.

$2*(3*(P1-P0)/T^2 - (2*V0+V1)/T) + 6*T*((V1+V0)/T^2 + 2*(P0-P1)/T^3) = 2*(3*(P2-P1)/T^2 - (2*V1+V2)/$
$T)\ 3*(P1-P0)/T^2 - (2*V0+V1)/T + 3*T*((V1+V0)/T^2 + 2*(P0-P1)/T^3) = 3*(P2-P1)/T^2 - (2*V1+V2)/T$
$3*(P1-P0)/T^2 - (2*V0+V1)/T + 3*(V1+V0)/T + 6*(P0-P1)/T^2 = 3*(P2-P1)/T^2 - (2*V1+V2)/T$
$3*(P1-P0)/T - (2*V0+V1) + 3*(V1+V0) + 6*(P0-P1)/T = 3*(P2-P1)/T - (2*V1+V2)$
$3*(P1-P0)/T + 6*(P0-P1)/T -3*(P2-P1)/T = (2*V0+V1) - (2*V1+V2) - 3*(V1+V0)$
$3/T*(P1 -P0 +2*P0- 2*P1 -P2 +P1) = 2*V0 +V1 - 2*V1 -V2 - 3*V1 -3*V0$
$3/T*(P0-P2) = -V0 -4*V1 -V2$
$V1 = (-3/T*(P0-P2) -V0 -V2)/4$ call this formula *F1*

So, if we knew the surrounding velocities and all the positions we could find the center velocity (V1) such that the acceleration remains continuous.

We can extend this to four points, (P0,V0) … (P3,V3). Lets say we know the V0 and V3 and want to find V1 and V2 for continuous accelerations.

We know from above that we could find V1 or V2 if we knew the surrounding velocities:

```
V1 = (-3/T*(P0-P2) - V0 - V2) / 4
V2 = (-3/T*(P1-P3) - V1 - V3) / 4
```

Substituting the V2 formula into V1 gives us:

```
V1 = (-3/T*(P0-P2) - V0 - ((-3/T*(P1-P3) - V1 - V3) / 4)) / 4
V1 = -3/T*(P0-P2)/4 - V0/4 + (3/T*(P1-P3) + V1 + V3) / 16
V1 = 3(P2-P0)/(4T) - V0/4 + 3(P1-P3)/(16T) + V1/16 +
V3/16 V1 * 15/16  = 3(P2-P0)/(4T) - V0/4 + 3(P1-P3)/(16T)
+ V3/16 V1 =  (12(P2-P0)/T - 4*V0 + 3(P1-P3)/T + V3) / 15
```

call this formula *F2*

This gives us V1 based just on the positions and V0 and V3. We can then find V2 using formula *F1* above that uses the positions and V1 and V3.

That's great if we only have four points and know the starting and ending velocities, say zero since we probably want to start and end the move with zero velocity. If we had five points we could extend this formula again by substituting and simplifying, but what if we have 10,000 points? It's not really practical to extend this formula that far.

What we can do is make a rough guess at reasonable velocities for a set of positions using a very simple algorithm like averaging the velocity change across two segments, then use the above formulas to refine those velocities to smooth out the move.

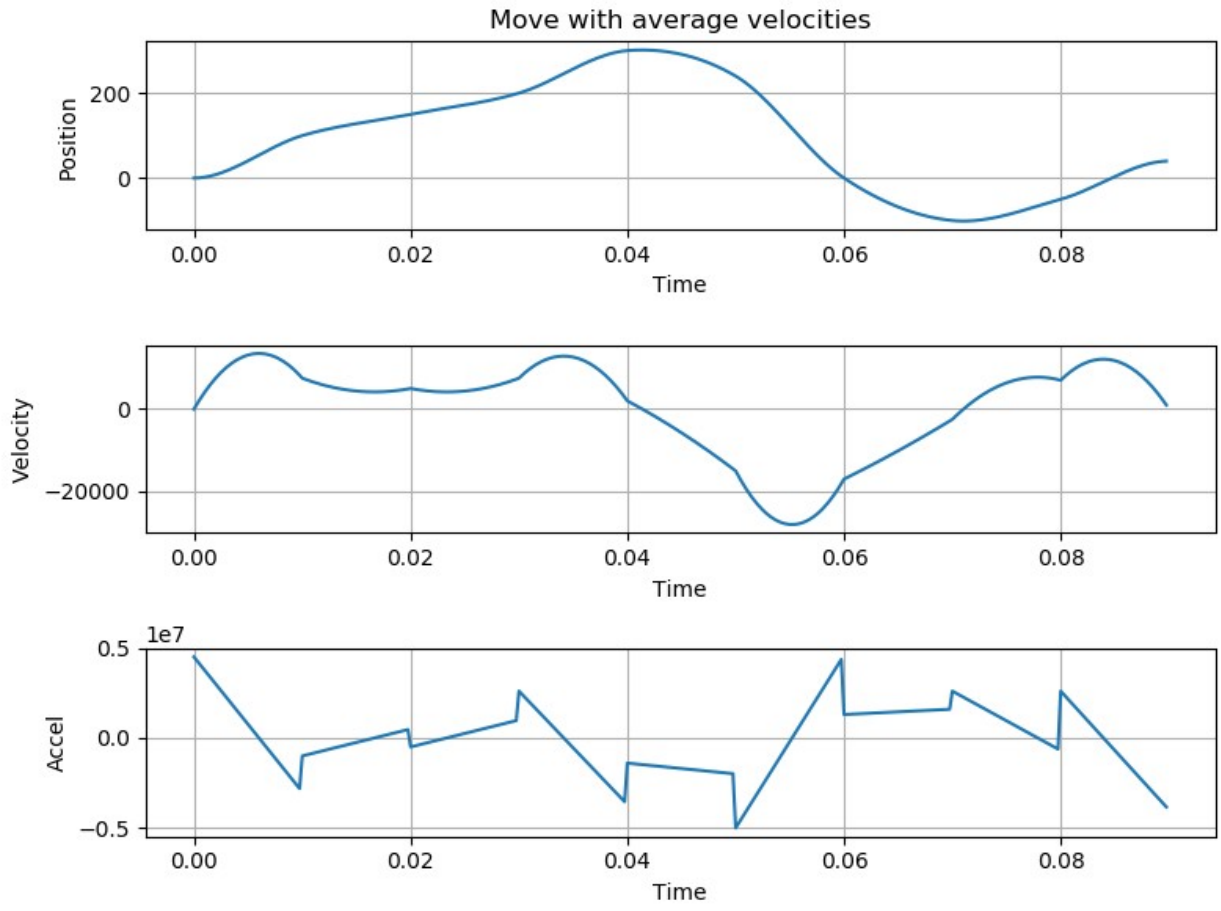
Take the following 10 positions chosen pretty randomly:

```
pos = [0, 100, 150, 200, 300, 240, 0, -100, -50, 40]
```

First, we'll take a rough guess at the velocities for each point by averaging the velocity over each pair of segments, and using zero as the starting and ending velocity. For example, the velocity at the start of the second segment would be (pos[2]-pos[0])/(2*T) where T is the time between segments, say 10ms. That gives us the following first guess at velocities:

vel = [0, 7500.0, 5000.0, 7500.0, 2000.0, -15000.0, -17000.0, -2500.0, 7000.0, 0]

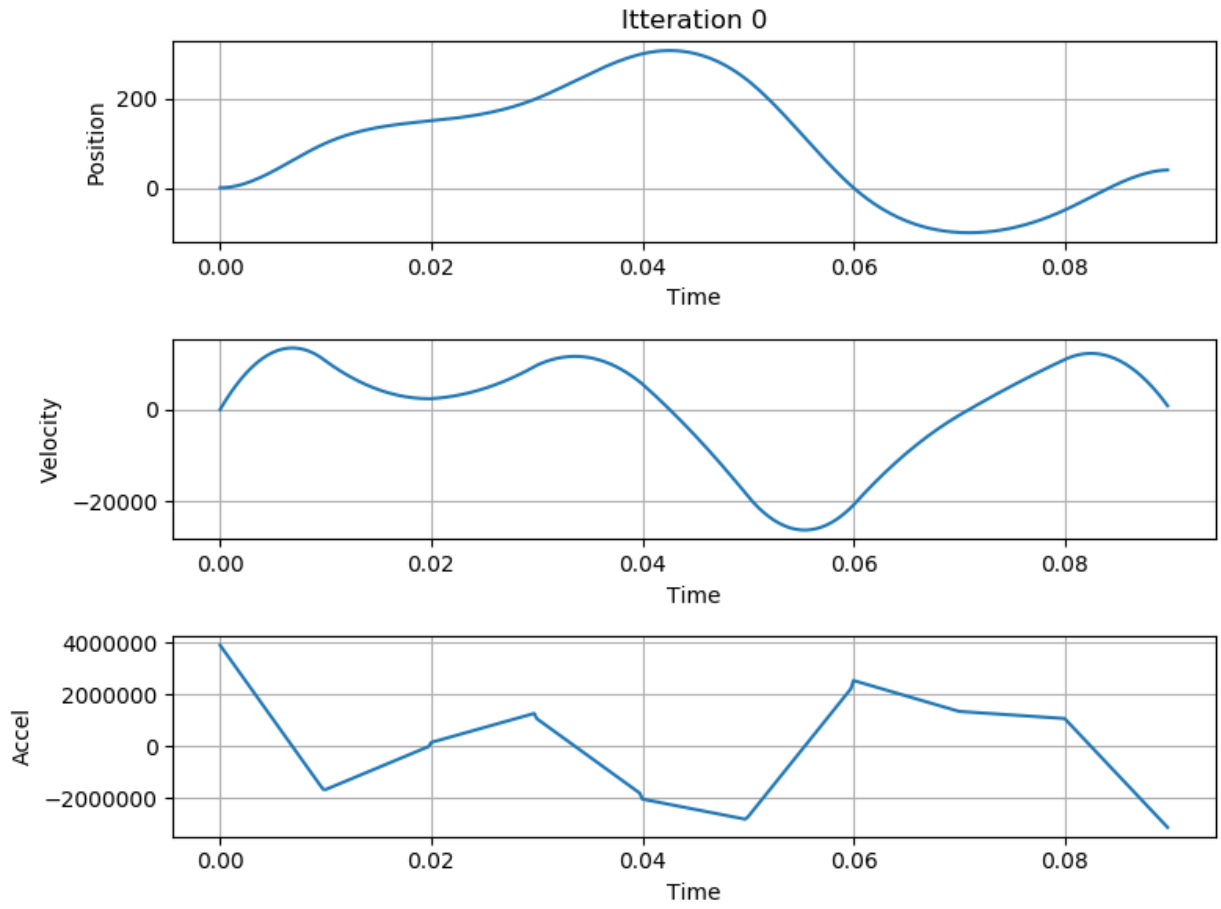
Since we know the formula the drive uses to interpolate points, we can calculate and graph the resulting trajectory that the drive would create with this input. Here's a graph of the position, velocity and acceleration of the resulting move using this first guess:



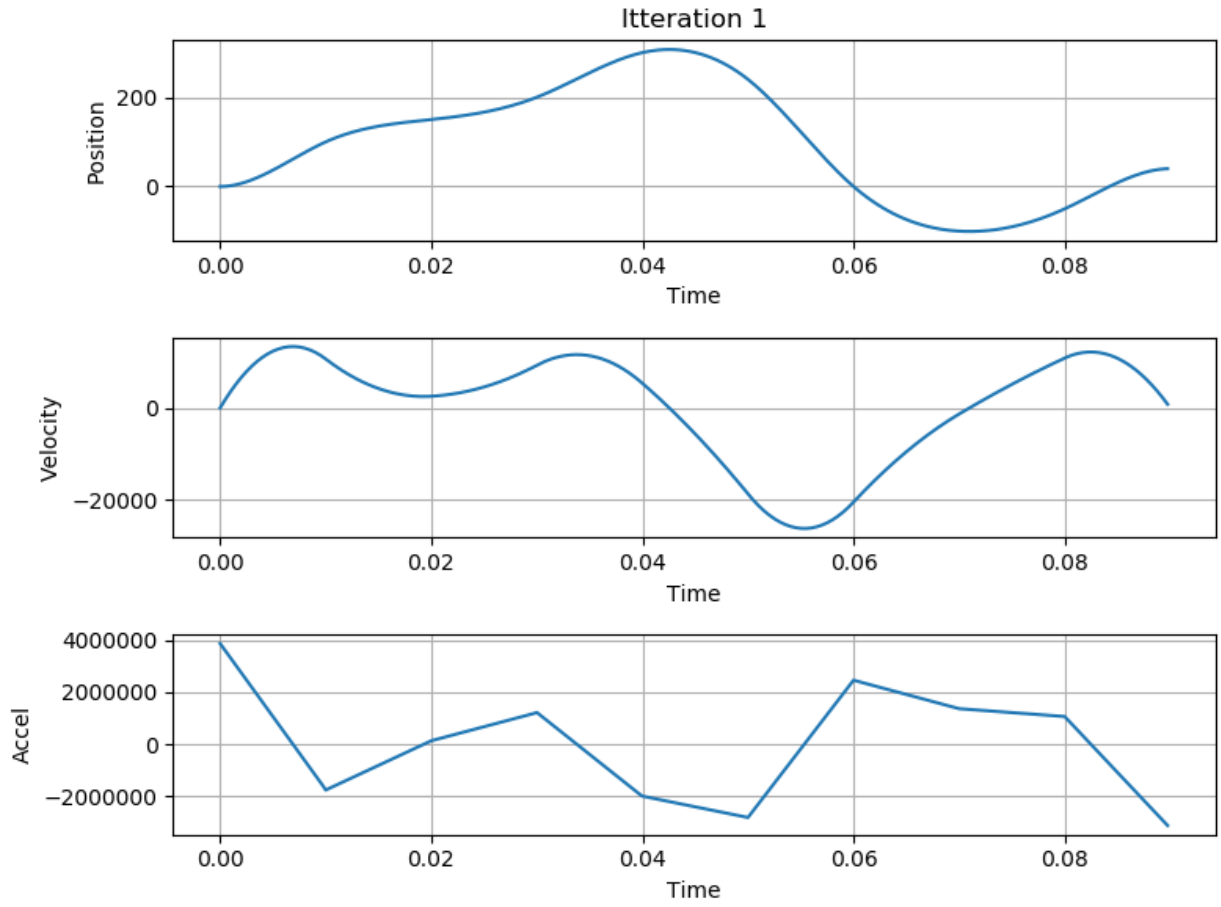
You can see that the positions are reasonably smooth and pass through all our points. The velocities are a little rough due to the discontinuities in the acceleration values. Not great, but a reasonable first guess.

Now, let's use these velocities and positions and run them through the formulas we came up with above to calculate a new set of velocities with the goal of removing the discontinuities in acceleration that our rough guess caused. We'll leave the starting and ending velocities at zero which seems reasonable, and calculate all the intervening velocities using the formula (*F2*). The

very last point will use the simpler formula (*F1*). The resulting trajectory after this step looks better:



There are still some slight discontinuities in acceleration, but they're much smaller then in the previous graph. We could further improve the move by running these improved velocities through the same set of equations:



At this point there are no obvious discontinuities in the acceleration values and the resulting positions and velocities look quite smooth. We could run the velocities through our smoothing formulas again, but it probably wouldn't change them very much.

This approach is one possible method for calculating reasonable velocity values to generate a PVT move through a set of known points. It's certainly not the only possible method and probably not the optimal method for all applications, but it's simple and fast and gives reasonable results.

Using PVT points to generate S-curve moves

An S-curve move is a type of trajectory between two positions in which velocity and acceleration remain continuous during the move. S-curve moves can be fit very easily into a series of PVT segments resulting in a very smooth move between two points.

The simplest form of S-curve move between two points consists of just four segments of equal time. At the start of the first segment the velocity and acceleration are zero. During that first segment, the acceleration increases linearly at a constant rate (known as jerk). During the second and third segments the acceleration decreases linearly at that same rate, and increases again during the last segment again at the same rate.

This type of move can be specified using just 4 PVT segments (i.e. 5 points). The calculation of the positions and velocities for the five points is quite simple. First, assume that we're starting from position 0. If we know the time of each segment (T) and the jerk (J), then we can calculate the required positions and velocities using a bit of calculus (derivation left as an exercise to the reader):

$$pos = [0,\ J*T^3/6,\ J*T^3,\ J*T^3*(11/6),\ 2*J*T^3]$$

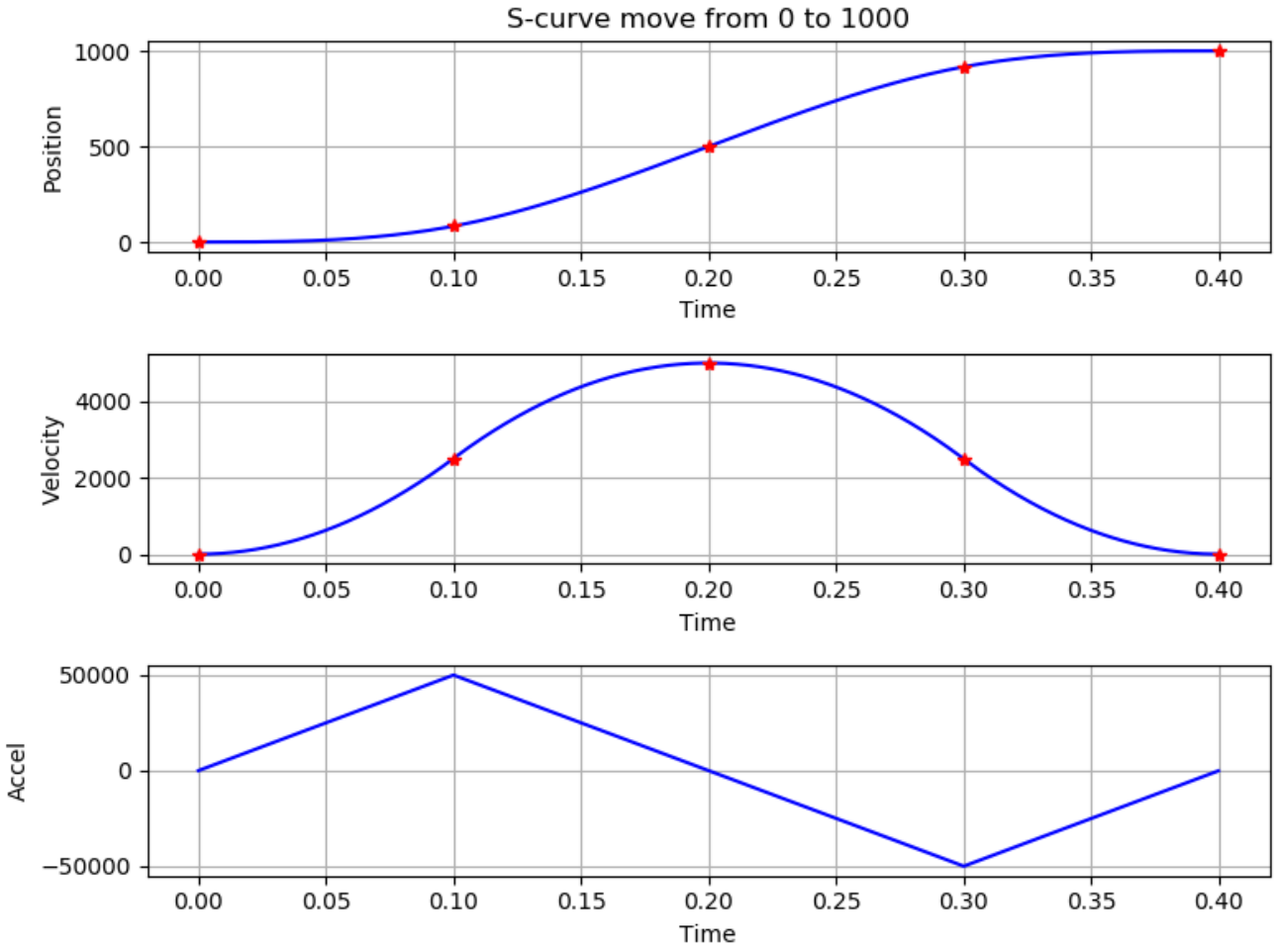
$$vel = [0,\ J*T^2/2,\ J*T^2,\ J*T^2/2,\ 0]$$

Presumably we know the distance of the desired move (D), so we can calculate jerk using that:

$$J = D/(2 * T^3)$$

Alternately, if we knew the desired jerk value, then we could use it to calculate the time of the segments by rearranging the above formula.

Here's an example of an s-curve move from 0 to 1000 counts using the above formula with a segment time of 100ms:



One could calculate slightly more complex s-curve moves which limited the maximum velocity and/or acceleration to desired values. Such moves would consist of more segments then the simple s-curve move shown above, but any such move can be easily described as a set of PVT points.

Conclusion

PVT trajectory mode is a very flexible way for the master to send complex trajectories to the drive. There are many different ways to calculate such trajectories, this applications note just gives a few possible examples.

Revision History

Date	Version	Revision
5/5/2020	Rev 00	Initial release