# Control of stepper motors for vPiP drawing machine

This document sets out the approach for driving the stepper motors of the vPiP drawing machine within the architecture set out in the wiki.

The primary goal of the stepper driver to to produce a smooth motion to enable the fastest possible motion without making the print head start swinging or creating excessive movement within the cables driving the print head (which has been observed in existing implementations with v1 software).

In addition to the fast, smooth motion the algorithm should:

- keep the print head moving as fast as possible, reducing the loss of speed when transitioning between lines
- provide progress feedback to controlling process(es)
- allow interrupt commands to be sent to pause a drawing (say to allow a pen change), which will bring the drawing to a controlled stop at the end of a line, with the pen being raised from the paper (and optionally the drawing head homed). The drawing must be able to be resumed from exactly where it was paused.

## Kinematics - Algorithm for achieving smooth, fast motion

In version 1 of vPiP, the algorithm used constant acceleration to change velocity. This produces very high jerk values (rate of change of acceleration) at the transitions, which is a cause for stress on the system, which could be responsible for generation vibrations and additional movement in the control cables. To reduce the stress in the system and provide smoother motion of the print head then a constant jerk algorithm should be used.

Using Calculus to understand the equations of motion:
Looking at displacement (s) (distance travelled from start position $s_0$):

- Velocity (v) is the first derivative of displacement with respect to time
- Acceleration (a) is the first derivative of velocity and the second derivative of displacement with respect to time (rate of change of velocity)
- Jerk (j) is the first derivative of acceleration and the third derivative of displacement with respect to time (rate of chance of acceleration)

Which gives us the kinematic equations:

$$s = s_0 + v_0 t + \frac{1}{2} a_0 t^2 + \frac{1}{6} j t^3$$

$$v = v_0 + a_0 t + \frac{1}{2} j t^2$$

$$a = a_0 + j t$$

Once the speed profile for the line has been calculated, then this needs to be translated into timer information to step each motor to create the line - this will be discussed later in the document.

Making each line a separate operation bringing the print head to a stop after each line is not ideal, so the algorithm also has to calculate the optimum speed to transition from 1 line to another. The angle between the lines will govern the amount of velocity that can be maintained during the transition and the appropriate amount of deceleration and acceleration should be calculated.

With short lines the acceleration/deceleration may take place over a number of line segments (such as curves implemented as a number of short line segments).

# Example data

With between motor driving wheels : 344mm
Distance to top left corner of board : x = 22mm; y = 36mm
Distance to paper location from top of board : x = 10mm; y = 40mm
therefore distance from (0,0) measuring point = 22+10 ; 36 + 40 = (32mm, 76mm)
Paper Size = A4 portrait : x = 210mm; y = 297mm,
therefore paper bottom right corner at 32 + 210; 76+297 = (242mm, 374mm)
Paper margin = 10mm, so drawing area is (42mm, 86mm) to (232mm, 364mm)
Drawing grid is 10,000 along x and $10000 * \frac{277}{190} = 14578.947$ along y
therefore1 point on the coordinate grid is $\frac{190}{10000} = 0.019mm$
Maximum Velocity = 30 mm/ds (millimeters per decisecond : decisecond = 0.1 seconds)
Maximum Acceleration = 10 mm/ds$^2$
Maximum Jerk = 5 mm/ds$^3$

Line in coordinates (100,9000) -> (9000,100)
Line in mm = (42+100*0.019, 86+9000*0.019) -> (42+9000*0.019, 86+100*0.019)
= (43.9,257) -> (213,87.9)

$$\sqrt{\Delta x^2 + \Delta y^2}$$

Line length = $\sqrt{(213 - 43.9)^2 + (257 - 87.9)^2}$

$$\sqrt{169.1^2 + 169.1^2}$$

$$\sqrt{57189.62} = 239.1435mm$$

# Line Geometry

To calculate the formula for the line from ($x_1$, $y_1$)->($x_2$, $y_2$):

$$y = mx + c$$

where m is the gradient of the line and c is where the line crosses the x=0 axis (unless the line is vertical),

so if x1 ≠ x2:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$y = mx + c$$

$$y - mx = c$$

$$c = y - mx$$

Plugging in one of the coordinate (100, 9000):

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$m = \frac{100 - 9000}{9000 - 100} = \frac{-8900}{8900} = -1$$

$$c = y - mx$$

$$c = 100 - -1 * 9000 = 9100$$

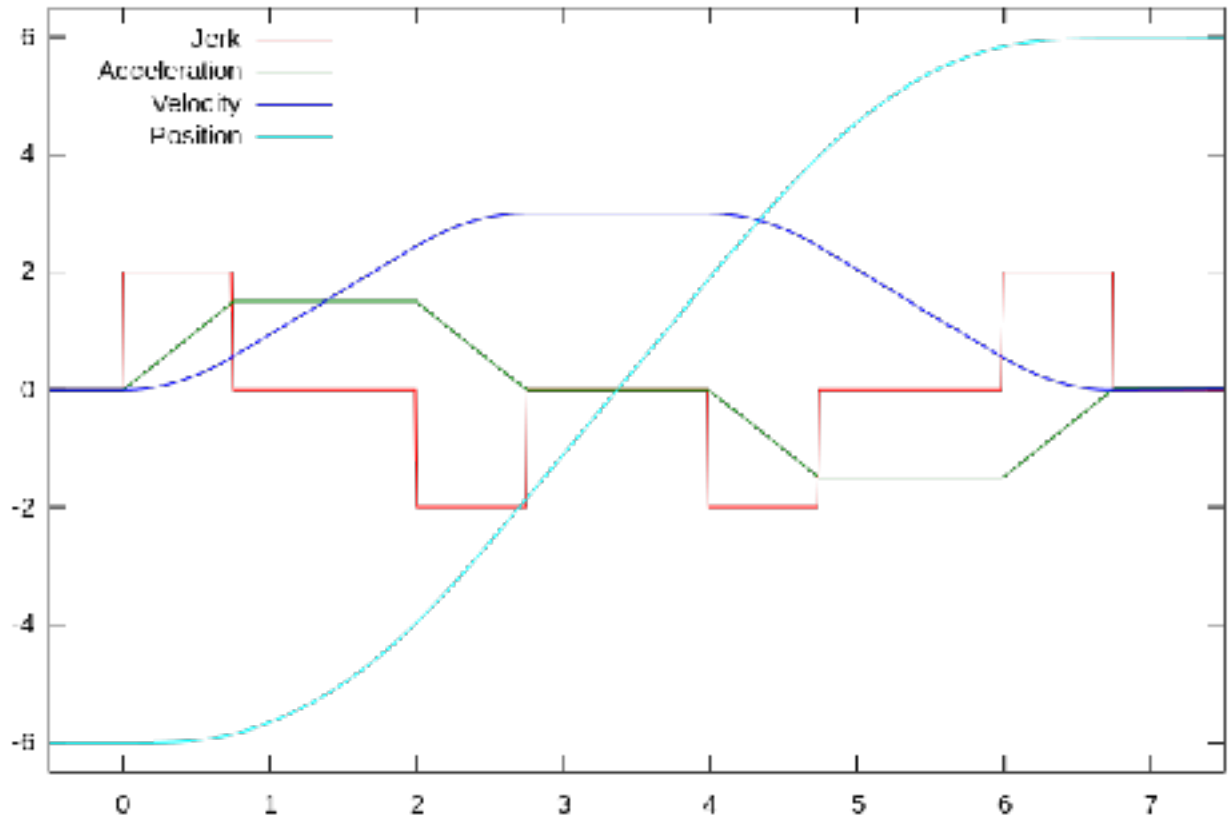
To calculate a point (x, y) a specified distance (d) from the first point ($x_1$, $y_1$) along a line ($x_1$, $y_1$)→($x_2$, $y_2$):

$$x = x_1 - \frac{d * (x_1 - x_2)}{line_{length}}$$

$$y = mx + c$$

# 7-phases for constant acceleration

Using constant jerk to control the rate of change of acceleration produces 7 stages for a line when the line needs to accelerate and decelerate and there is sufficient distance, velocity change and time to allow all 7 stages:




| | $\Delta t_1 = t_0 \rightarrow t_1$ | $\Delta t_2 = t_1 \rightarrow t_2$ | $\Delta t_3 = t_2 \rightarrow t_3$ | $\Delta t_4 = t_3 \rightarrow t_4$ | $\Delta t_5 = t_4 \rightarrow t_5$ | $\Delta t_6 = t_5 \rightarrow t_6$ | $\Delta t_7 = t_6 \rightarrow t_7$ |
|---|---|---|---|---|---|---|---|
| **Jerk - j** | $j_{max}$ | 0 | $-j_{max}$ | 0 | $-j_{max}$ | 0 | $j_{max}$ |
| **Acceleration - a** | increasing | $a_{max}$ | decreasing | 0 | increasing deceleration | $-a_{max}$ | decreasing deceleration |
| **Velocity - v** | increasing | increasing | increasing | $v_{max}$ | decreasing | decreasing | decreasing |
| **Displacement - s** | increasing | increasing | increasing | increasing | increasing | increasing | increasing |

Acceleration time $= \Delta t_1 + \Delta t_2 + \Delta t_3$
Constant velocity time $= \Delta t_4$
Deceleration time $= \Delta t_5 + \Delta t_6 + \Delta t_7$

---

## Phase 1 - Increasing Acceleration

At $t_0$:

$j_0 = j_{max} = 5$
$a_0 = a_0 = 0$
$v_0 = v_0 = 0$
$s_0 = 0$

$$\Delta t_1 = \Delta t_3 = t_1 = \frac{a_{max}}{j_{max}} = \frac{10}{5} = 2$$

$$t_0 < t \leq t_1$$

$$a_t = j_{max}t$$

$$v_t = v_0 + a_0 t + \frac{1}{2} j_{max} t^2$$

$$s_t = v_0 t + \frac{1}{2} a_0 t^2 + \frac{1}{6} j_{max} t^3$$

Note: at t = $t_1$:

$$v_1 = v_0 + \frac{a_s^2}{(2 j_{max})}$$

plugging in the sample numbers gives:

$$t_0 < t \leq t_1$$

$$v_t = 0 + 0t + \frac{1}{2} 5t^2 = \frac{5t^2}{2}$$

$$s_t = 0t + \frac{1}{2} 0t^2 + \frac{1}{6} 5t^3 = \frac{5t^3}{6}$$

at end of phase ($t_1$) we have:

$$a_{t_1} = a_{max} = 10 mm/ds^2$$

$$v_{t_1} = \frac{5t^2}{2} = \frac{5 * 2^2}{2} = \frac{20}{2} = 10 mm/ds$$

$$or$$

$$v_{t_1} = 0 + \frac{10^2}{2 * 5} = \frac{100}{10} = 10$$

$$s_{t_1} = \frac{5t^3}{6} = \frac{5 * 2^3}{6} = \frac{40}{6} = 6.667 mm$$

---

## Phase 2 - Constant Acceleration

At $t_1$

$j_1 = 0$
$a_1 = a_{max}$
$v_1 = v_{t1} = 10$ $v_2 = v_{max} - v_1 = 30 - 10 = 20$
$s_1 = s_{t1} = 6.667$

$$v_2 = v_1 + a_{max} \Delta t_2$$

$$v_2 - v_1 = a_{max} \Delta t_2$$

$$\frac{v_2 - v_1}{a_{max}} = \Delta t_2$$

$$\Delta t_2 = \frac{v_{max} - 2v_1}{a_{max}}$$

$$\Delta t_2 = \frac{30 - 2 * 10}{10} = \frac{30 - 20}{10} = 1$$

$$t_1 < t \leq t_2$$

$$v_t = v_1 + a_{max} t = 10 + 10t$$

$$s_t = v_1 t + \frac{1}{2} a_{max} t^2 = 10t + \frac{10}{2} t^2 = 10t + 5t^2$$

at end of phase ($t_2$) we have:

$$v_{t_2} = 10 + 10 * 1 = 20mm/ds$$

$$\Delta s_{t_2} = 10 * 1 + 5 * 1^2 = 10 + 5 = 15mm$$

---

## Phase 3 = Decreasing Acceleration

At $t_2$:

$j_2 = -j_{max}$
$a_2 = 10mm/ds^2$
$v_2 = 20mm/ds$
$s_2 = s_1 + \Delta s_2 = 6.667 + 14 = 20.667mm$

$t_2 < t \leq t_3$

$$a_t = j_{max}t$$

$$v_t = v_2 + a_2 t + \frac{1}{2} j_{max} t^2$$

$$s_t = v_2 t + \frac{1}{2} a_2 t^2 + \frac{1}{6} j_{max} t^3$$

plugging in the example numbers gives:
$t_0 < t \leq t_1$

$$v_t = 20 + 10t + \frac{1}{2} - 5t^2 = 20 + 10t - \frac{5t^2}{2}$$

$$s_t = 20t + \frac{1}{2} 10t^2 + \frac{1}{6} 5t^3 = 20t + \frac{10}{2} t^2 - \frac{5t^3}{6}$$

at the end of phase 3 we have :

$$a_{t_3} = 0$$

$$v_{t_3} = 20 + 10 * 2 - \frac{5 * 2^2}{2} = 20 + 20 - \frac{20}{2} = 30mm/ds$$

$$\Delta s_{t_3} = 20 * 2 + \frac{10 * 2^2}{2} - \frac{5 * 2^3}{6} = 40 + 20 - \frac{40}{6} = 40 - 6.667 = 33.33mm$$

---

## Phase 4 - Constant velocity

At $t_3$:

$j_3 = 0$
$a_3 = 0$
$v_3 = v_{max} = 30mm/ds^2$
$s_3 = s_1 + \Delta s_2 + \Delta s_3 = 6.667+15+33.333 = 55mm$ = Distance to accelerate to max velocity
$t_3 = \Delta t_1 + \Delta t_2 + \Delta t_3 = 2+1+2 = 5ds$ = Time to accelerate to max velocity

With symmetric acceleration and deceleration, distance and time to decelerate will equal time to accelerate, so for phase 4, the constant velocity phase:

Distance to travel in phase 4 = Total Distance - $2*s_3$ = 239.1435-2*55 = 239.1435-110 = 129.1435

$t_3 < t \leq t_4$

$$s_t = v_{max} t$$

so plugging in the test data we have

$$129.1435 = 30\Delta t_4$$

$$\Delta t_4 = \frac{129.1435}{30} = 4.3048ds$$

---

## Phase 5 - Increasing Deceleration

At $t_0$:

$j_4 = -j_{max}$
$a_4 = a_4$
$v_4 = v_4$
$s_4 = s_4$

$$\Delta t_5 = \Delta t_7 = t_5 = \frac{a_{max}}{j_{max}}$$

$$t_4 < t \leq t_5$$

$$v_t = v_4 + a_4 t - \frac{1}{2} j_{max} t^2$$

$$s_t = v_4 t - \frac{1}{2} a_4 t^2 - \frac{1}{6} j_{max} t^3$$

---

## Phase 6 - Constant Deceleration

At $t_1$

$j_1 = 0$
$a_1 = a_{max}$
$v_1 = v_1$ $v_2 = v_{max} - v_1$
$s_1 = s_1$

$$t_5 < t \leq t_6$$

$$v_2 = v_1 + a_{max}\Delta t_2$$

$$v_2 - v_1 = a_{max}\Delta t_2$$

$$\frac{v_2 - v_1}{a_{max}} = \Delta t_2$$

$$\Delta t_2 = \frac{v_{max} - 2v_1}{a_{max}}$$

$$v_t = v_1 + a_{max} t$$

$$s_t = v_1 t + \frac{1}{2} a_{max} t^2$$

---

## Phase 7 = Decreasing Deceleration

At $t_2$:

$j_2 = j_{max}$
$a_2 = a_{max}$
$v_2 = v_2$
$s_2 = s_2$

$t_6 < t \leq t_7$

$$v_t = v_2 + a_2 t - \frac{1}{2} j_{max} t^2$$

$$s_t = v_2 t + \frac{1}{2} a_2 t^2 - \frac{1}{6} j_{max} t^3$$

| | $t_0$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ |
|---|---|---|---|---|---|---|---|---|
| **t (ds)** | 0 | 2 | 3 | 5 | 9.3048 | 11.3048 | 12.3048 | 14.3048 |
| **a (mm/ds²)** | 0 | 10 | 10 | 0 | 0 | -10 | -10 | 0 |
| **v (mm/ds)** | 0 | 10 | 20 | 30 | 30 | 20 | 10 | 0 |
| **d (mm)** | 0 | 6.667 | 21.667 | 55 | 184.1435 | 217.4768 | 232.4768 | 239.1435 |

---

## How to deal with short lines

The 7 phases defined above requires a line to be of a certain length to allow all 7 phases to run. As line become shorter then phase 4 will reduce until it no longer contributes to the movement, so the line becomes phases 1, 2, 3, 5, 6, 7. This is the shortest line that will reach maximum velocity from a standing start and coming to a stop at the end of the line.

If the line is shorter then phases 2 and 6 will reduce symmetrically, but this will reduce the velocity reached whilst drawing the line (maximum rate of acceleration will still be reached). As the line shortens then eventually phases 2 and 6 will contribute nothing to the line, so the line becomes phases 1, 3, 5, 7. This is the shortest line that will reach maximum acceleration from a standing start and coming to a stop at the end of the line.

If the line is still shorter then phases 1, 3, 5 and 7 will reduce symmetrically, but this will prevent maximum acceleration from being achieved whilst drawing the line.

# Combining multiple lines

It is not always desirable to start a line from stop and coming to a stop at the end of every line. It is often possible to combine lines to maintain some velocity during the transition from one line to the next. The angle between the lines determines how much velocity can be carried during the transition from one line to the next.

[See https://onehossshay.wordpress.com/2011/09/24/improving_grbl_cornering_algorithm/]

For a given angle $\theta^{rad}$ the maximum velocity is given by:

$$v_{junction} = \sqrt{a_{max} R}$$

$$R = \sqrt{\frac{1 - cos(\theta^{rad})}{2}}$$

Given the 3 points that define 2 lines $(x_1, y_1)$->$(x_2, y_2)$->$(x_3, y_3)$ that join at $(x_2, y_2)$, you can use vectors to find the angle between the two lines.

$$\vec{u} = (a_1, b_1), \vec{v} = (a_2, b_2)$$

$$a_1 = x_2 - x_1, b_1 = y_2 - y_1, a_2 = x_3 - x_2, b_2 = y_3 - y_2$$

$$cos(\pi - \theta^{rad}) = \frac{(\vec{u} \cdot \vec{v})}{(\|\vec{u}\| \cdot \|\vec{v}\|)}$$

$$cos(\theta^{rad}) = -1 * \frac{(\vec{u} \cdot \vec{v})}{(\|\vec{u}\| \cdot \|\vec{v}\|)}$$

$$cos(\theta^{rad}) = -1 * \frac{(a_1 a_2 + b_1 b_2)}{(\sqrt{a_1{}^2 + b_1{}^2} * \sqrt{a_2{}^2 + b_2{}^2})}$$

*Note:* $\| \vec{u} \|$ *is the length of the vector* $\vec{u}$*, which is the length of the line* $(x_1,y_1)$->$(x_2,y_2)$ *and* $\| \vec{v} \|$ *is the length of the vector* $\vec{v}$*, which is the length of the line* $(x_2,y_2)$->$(x_3,y_3)$.

Combining the 2 formula removes the cos function:

$$v_{junction} = \sqrt{a_{max} * \sqrt{\frac{1 + (\frac{(a_1 a_2 + b_1 b_2)}{(\sqrt{a_1{}^2 + b_1{}^2} * \sqrt{a_2{}^2 + b_2{}^2})})}{2}}}$$

Using this formula a number of line segments can be combined to form a continuous movement. The motion needs to come to a complete stop when the pen needs to be raised or when the angle between lines is too small. Whilst drawing the composite line any of the 7 phases may cross multiple line segments and there may need to be additional acceleration or deceleration phases within the composite line depending on line segment joint angles.

This requires a planning algorithm to:
- Combine individual lines into line segments
- Determine maximum joint velocities
- Map the composite line into a series of motion phases

---

## Map lines into a series of motion phases

Turning the line segments into one of the 7 phases of motion (as described above). For any line there are a number of possible options:

1. The initial and final velocities are low enough and there is sufficient distance to travel to go through all 7 phases
2. The initial velocity is too hight, so only time for phases 1 and 3 while accelerating
3. The final velocity is too high to go through all deceleration phases, so only phases 5 and 7.
4. The line is too short to allow phase 4 (max velocity may not be possible), phases 1,2,3,5,6,7
5. The line is too short to allow constant acceleration phases, only phases 1,3,5,7
6. The line is too short to accelerate to reach the final velocity. Only phases 1,3 or 1,2 and 3. The length will determine how near the final velocity can be attained, but the line may complete with a slower than planned maximum velocity.
7. The line is too short to decelerate to meet the required final velocity. The start velocity to meet then required final velocity to be reach needs to be calculated then the previous line needs to be altered to finish at the required lower velocity so the current line can decelerate to the required final velocity.

# Turning the planned line into vPip movement

Up to now this document has focussed on calculating the motion for the lines. Once the lines have been planned and the set of motion phases has been calculated to create the lines this needs to be mapped into a set of stepper instructions for the 2 motors on a vPiP drawing machine. The planning and calculations required to create the motion phases is too complex for a small 8-bit processor, such as the Arduino that drives the steppers, so the computation is performed on the Raspberry Pi then a series of instructions is passed to the Arduino to control the stepper drivers.

There are a number of possible approaches for the communication between the Raspberry Pi and the Arduino. There may have to be a trade off between accuracy and maximum achievable data rate between the Raspberry Pi and the Arduino.

In version 2 there are options to look at using different communication mechanisms between the Raspberry Pi and the Arduino:

- Serial (UART)
- $I^2C$ - (cannot run at maximum speed, as there is a hardware bug in the Raspberry Pi, which the Arduino cannot cope with)
- SPI - (not enabled with the current version of the vPIP controller circuit board)

Other options that could be considered include modifying the module used to control the vPiP plotter (ESP32, NXP or ST Micro arm based module - these give 32-bit processing capability, so may remove the need to split the calculation and motor driving, or could leave the planning on the Pi or a laptop/workstation, but move the stepper calculations to the vPiP controller board)

These decisions need to be made up front as the outcome will define the vPiP controller board interface and API.

---

## Time Slicing

In version 1 of vPiP the Raspberry Pi to Arduino communication is based on a series of equal time slices, where the number of steps required for each motor is calculated and sent over the serial link to the Arduino. This produces an approximation of the calculated motion as for each time slice the required steps are spaced equally over the time rather than being at the exact calculated time interval.

Using time slices we have all the equations we need to determine the position the drawing head (displacement - s) should be at for each time slice interval, so for a given time t we can calculate the displacement (s) then calculate the coordinate in mm coordinate space. This can then be plugged into the equations that calculate the length of the control cables from each stepper motor moving the drawing head, and then from there the number of steps each motor should be at from the home position. The smaller the time slice the more accurate the step timing, but the higher the data rate between the Raspberry Pi and the Arduino.

---

## Fully accurate time steps

This is the most desirable outcome, as each stepper motor will be driven at exactly the correct time (allowing for CPU clock speed). For each step of each motor calculate the time the step should be taken, then combine the step times for each motor to create a stream of delays then step instructions (L step, R step or both step). This will require a high data rate (need to calculate) to pass these instructions to the stepper controller, and may need a high CPU clock speed on the controller to be able to parse the stream and implement the stepper controls.

One of the issues with this approach is the mathematics needed to calculate the time for each step, as phases 1, 3, 5 and 7 involve cubic equations, which need to be solved for t, as s, v, a and j will be known:

$$s_t = v_2 t + \frac{1}{2} a_2 t^2 + \frac{1}{6} j_{max} t^3$$

[TODO research algorithms for solving cubic functions - maybe https://stackoverflow.com/questions/1829330/solving-a-cubic-equation or https://www.codeproject.com/Articles/798474/To-Solve-a-Cubic-Equation provides a solution?]