**Computing Junction Deviation for Marlin Firmware**

[**October 11, 2018**](http://blog.kyneticcnc.com/2018/10/computing-junction-deviation-for-marlin.html)

  As part of developing my own 3D printer firmware, I also keep an eye on what is happening in other firmware.  One feature that is causing confusion in the Marlin community is the junction deviation setting.  Up until recently, Marlin used the jerk method (hence forth referred to as "archaic jerk") it inherited from Grbl for computing corning speed (junction velocity).  With the option now in Marlin to use junction deviation instead of jerk, there are many people who want to know what are good settings for junction deviation to insure they get reasonable movement while printing.  In this post I will give an equation for converting the jerk values into junction deviation and my derivation of this equation.  
  
I will not rehash what junction deviation is in this post, so anyone interested in learning more about it should refer to these two links:  
  
<https://onehossshay.wordpress.com/2011/09/24/improving_grbl_cornering_algorithm/>  
  
<https://reprap.org/forum/read.php?1,739819>  
  
For those who are only here for the end results, here is the equation to compute junction deviation:

[](https://www.codecogs.com/eqnedit.php?latex=d=0.4\frac%7bJerk%5e%7b2%7d%7d%7bAccel_%7bprinting%7d%7d)

A typical Cartesian machine might have a jerk setting of 9mm/s and a printing acceleration of 1500mm/s^2.  Plugging these numbers into the above equation goes like this:

[](https://www.codecogs.com/eqnedit.php?latex=0.4*9*9/1500=0.022)

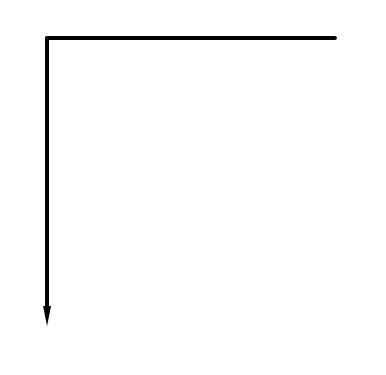
So for this example the junction deviation would be set to 0.022 and the configuration\_adv.h file would look like this:

//  
// Use Junction Deviation instead of traditional Jerk Limiting  
//  
#define JUNCTION\_DEVIATION  
#if ENABLED(JUNCTION\_DEVIATION)   
  #define JUNCTION\_DEVIATION\_MM 0.022 // (mm) distance from real junction edge  
#endif

The smaller the junction deviation number, the more the machine will slow down when encountering corners.  Making this number too small may slow down the print speed and cause over extrusion in the corners.  Making the number larger will increase the speed at which the machine traverses corners.  Setting this number too high may cause vibrations in the mechanism that produce ringing in the print surface or in extreme cases a stepper motor to loose position.

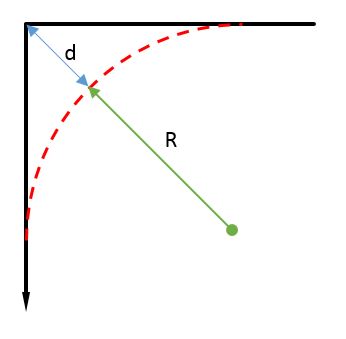
If you only came here for the basic equation, you do not need to read any further.  For those who would like to learn where the above equation came from, keep on reading.

To setup this problem, we are going to imagine a situation where the printer traverses two moves that are at right angles to each other:

[](https://1.bp.blogspot.com/-_1P1ovm-Ei4/W7-Zsy93o-I/AAAAAAAA2kA/OFPoHSa_Hv8SDWJPObKF4tkgIDgH_R_5ACPcBGAYYCw/s1600/Corner%2BMove.JPG)

This is a convenient situation as it is the case where one axis suddenly stops moving, and the other axis suddenly starts moving.  When using the archaic jerk functionality the first axis will decelerate down to the the jerk velocity in the corner and then instantly stop, at the same moment the second axis will suddenly jump up to the jerk velocity in its direction and accelerate back up to printing speed.  To simplify this, we can say that with a 90 degree corner, the junction velocity is the same as the jerk setting. (note: this is not the case for any junction that is not 90 deg).

Next we can take that same corner move and add the virtual arc that junction deviation uses to compute the junction velocity:

[](https://3.bp.blogspot.com/-n0XseZZP9MA/W7-gJw1nhLI/AAAAAAAA2kM/n8AXWhq6USYZ6sTbfbPw5gIiuac5L-csgCLcBGAs/s1600/Corner%2BMove%2Bwith%2BArc.JPG)

In the above picture **d** represents the junction deviation and **R** is the radius of the virtual arc.

Using the Pythagorean theorem and a little algebra and we get this relationship between **R** and **d**:

[](https://www.codecogs.com/eqnedit.php?latex=\left&space;(&space;R+d&space;\right&space;)%5e%7b2%7d=R%5e%7b2%7d+R%5e%7b2%7d)

[](https://www.codecogs.com/eqnedit.php?latex=d=\sqrt%7b2*R%5e%7b2%7d%7d-R)

[](https://www.codecogs.com/eqnedit.php?latex=d=\left&space;(&space;\sqrt%7b2%7d-1&space;\right&space;)*R)

[](https://www.codecogs.com/eqnedit.php?latex=d\approx&space;0.4*R)           (1)

Next we take the equation for radial acceleration and solve for the radius:

[](https://www.codecogs.com/eqnedit.php?latex=A_%7bradial%7d=\frac%7bV%5e%7b2%7d%7d%7bR%7d)

[](https://www.codecogs.com/eqnedit.php?latex=R=\frac%7bV%5e%7b2%7d%7d%7bA_%7bradial%7d%7d)        (2)

If we combine equations 1 and 2 above we find the following:

[](https://www.codecogs.com/eqnedit.php?latex=d\approx&space;0.4\frac%7bV%5e%7b2%7d%7d%7bA_%7bradial%7d%7d)

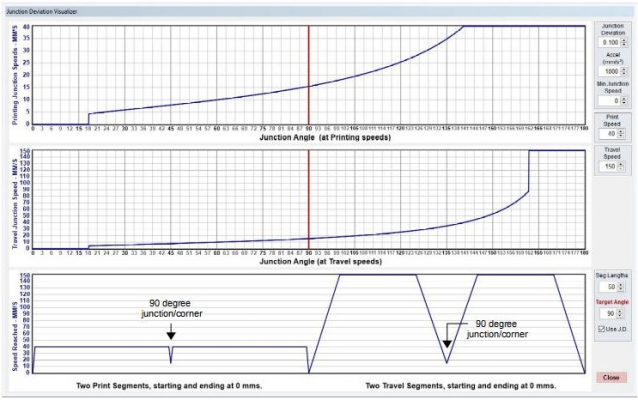
In this situation, the radial acceleration is the same as the printing acceleration and the junction velocity is the same as the archaic jerk setting.  Substituting these in, we get the final equation:

[](https://www.codecogs.com/eqnedit.php?latex=d=0.4\frac%7bJerk%5e%7b2%7d%7d%7bAccel_%7bprinting%7d%7d)

Using this equation will allow someone who is migrating from the archaic jerk to junction deviation to start with settings that are very close to each other without excessive experimentation or testing.  
  
Please let me know if you have any questions.  
  
Phillip

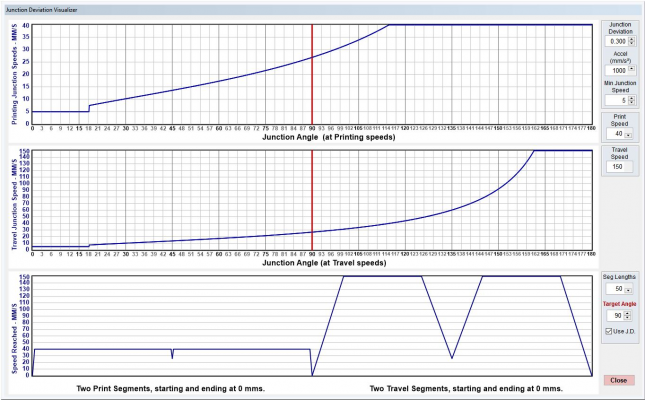
# Junction Deviation Explained and Visualized

The following will illustrate how **Junction Deviation** and **Acceleration** work together for smooth cornering.  
  
**Intro:**  
Printer controllers use a motion control program to plan the timing of every step. Motion controllers based on [GRBL](http://dank.bengler.no/-/page/show/5470_grbl) use a cornering algorithm called Cornering **Junction Deviation**, which I'll focus on here. (Other firmware use a different algorithm called Jerk which may be implemented differently - I am not discussing Jerk here.)  
I've been researching Junction Deviation as part of my research into pressure compensation.  
  
**In layman's terms:**  
- Junction Deviation determines how tight the controller's motion planner sees an upcoming corner to be.  
- Acceleration controls how sharply the breaks and gas pedal are applied.  
- The smaller the Junction Deviation, the slower the cornering speed will be.  
- Don't worry, the term "Junction Deviation" doesn't mean the printer will deviate from it's course!  
  
I wanted to be able to visualize the effect that Junction Deviation and Acceleration have on cornering speed and segment time. So I extracted and "functionated" the Junction Deviation calculation from Smoothieware and created a visualizer for my use, and these screenshots are the result.  
  
The most critical inputs to the calculation are the **Acceleration**, **Junction Angle**, and the **Junction Deviation**, where Junction Deviation is a number between 0 and about 1 (numbers over 0.5 are ridiculous tho). I won't go into the [details of the equation](https://onehossshay.wordpress.com/2011/09/24/improving_grbl_cornering_algorithm/) here, I'm focusing on the effects.  
  
The top plot shows the cornering speed at different junction angles for a maximum printing speed of 40 mm/s, given the inputs on the right. A 90 degree junction angle is simply a 90 degree corner between two segments. The red line shows where the target angle intersects the travel speed - giving the cornering speed for the given angle.  
  
The middle plot shows the cornering speeds for up to a 150 mm/s travel speed.  
If the acceleration is the same, then the plots for printing and travel are identical (up to the maximum print speed).  
  
When printing a 90 degree corner: the effector will slow to about 15mm/s given the junction deviation of 0.100 and 1000 mm/sec Acceleration. Increasing either the Junction Deviation or the Acceleration will shift that line up, and the machine will corner faster.

[](https://dl.dropboxusercontent.com/s/n37olj3pogxe744/Junction_Deviation_Visualized_.10.jpg?dl=0)

[Junction\_Deviation\_Visualized\_.10.jpg](https://dl.dropboxusercontent.com/s/n37olj3pogxe744/Junction_Deviation_Visualized_.10.jpg?dl=0)

The bottom plot shows a simulation of:  
- two printing segments with a 90 degree corner between them,  
- and then two travel segments with a 90 degree corner between them.  
  
Each of these pairs of segments shows a dip in the center down to 15 mm/s - this is the cornering speed at the target 90 degree angle.  
  
The motion for each pair of segments in detail is: Accelerate from 0 to the requested speed, speed is flat to before the corner/junction point, decelerate to junction speed between segments, accelerate back to the requested speed in the next segment, speed is flat to before end of segment, decelerate to 0.  
  
If Junction Deviation was not used: the printer would just try to slam around the corner at full speed, and that would cause many problems (lost steps, extreme vibration, etc). If I could turn Junction Deviation off, then the plot of each pair of segments would not dip in the center.  
  
There is one other parameter used with Junction Deviation: "**Minimum Planner Speed**" in Smoothieware.  
  
For my screen I labeled it "Minimum Junction Speed" as I think that better describes it.  
Notice at the beginning of the first two plots there is a knee, it starts out with a speed of 0, and then jumps up suddenly at about 18 degrees. This is a hardcoded "feature" of the J.D. calculation as it was implemented - any angle below about 18 degrees is considered essentially 0 degrees, and the "Minimum Planner Speed" is used.  
If you want to make very sharp corners print faster then you can increase the "Minimum Planner Speed" from the default 0 mm/s.  
  
Notice also that at about 162 degrees there is a sharp rise to full speed. This is also how J.D. was implemented - nearly strait angles more than (180 minus 18 degrees) are considered strait.  
  
For comparison the following shows what happens if junction deviation is increased to 0.300. The cornering speed at 90 degrees has increased from about 15 mm/s to about 27 mm/s. That could well be enough to cause ringing - depending on the printer.

[](https://dl.dropboxusercontent.com/s/olnj2l68t9o7u8u/Junction_Deviation_Visualized.30.JPG?dl=0)

[Junction\_Deviation\_Visualized.30.JPG](https://dl.dropboxusercontent.com/s/olnj2l68t9o7u8u/Junction_Deviation_Visualized.30.JPG?dl=0)

Note that this would be equal to the speed reached if you just changed the acceleration from 1000 mm/sec/sec to 3000 mm/sec/sec (although the acceleration slope would be much sharper!).  
  
I also increased the "Minimum Planner Speed" to 5 mm/s - notice how the beginning of the first two plots does not start at 0. This amount only will affect angles below 18 degrees, however if you increased Minimum Planner Speed to 10 it would speed up corners with 30 degree angles to 10 mm/s. I'm not necessarily recommending it - only testing will tell how this would affect your printer!  
  
You can change the Junction Deviation and Minimum Planner Speed (Minimum Junction Speed) in the Smoothie Config file, or using the [M205](https://reprap.org/wiki/G-code#M205:_Advanced_settings) command. Other firmware will use different settings - you will need to do your own research on that.  
  
You can test Junction Deviation and acceleration using a model like [this one](https://www.youmagine.com/designs/multi-angle-test-tower-for-calibrating-jerk).  
  
If I've made any errors, or if you have something to add, please let me know!