Fixed Time Motion Control of 3D Printers using Marlin

Ulendo

1 Motivation

Marlin firmware is widely used software that can be configured to control 3D printers and other machines. Its method of motion [stepper motor] control updates physical outputs only as needed. In other words, it generates control signals between which the timing is dynamic / irregular. This makes for difficult or even impossible implementation of control algorithms which rely on manipulating control signals on a real-time basis. Such algorithms include input-shapers, which require modulation of the control signal at times proportional to frequencies they wish to suppress, and Ulendo Filtered-B-Splines (FBS), which requires a time-based view of the motion trajectory.

2 Feature Submission

The changes made in this submission implement a fixed-time control of motion on Marlin firmware. This facilitates the use of input-shapers, Ulendo FBS, and other algorithms with little additional changes or configuration. The submitted changes include implementations of popular input-shaper algorithms: ZV, ZVD, EI, 2-Hump EI, 3-Hump EI, and MZV.

3 Description of Changes

3.1 Overview

Architecturally, all changes lie between the planner and the stepper pulse software. As such, fixed-time control is implemented with the least impact to the rest of the software. The flow of a G-code instruction is unaltered up to being loaded into the planner. Beyond this the flow is roughly outlined as:

- A block is unloaded from the planner in a manner similar to the stock software.
- All the block's data describing the motion trapezoid is extracted.
- This data is used to generate data points of [position] coordinates of each axis spaced apart by a fixed time interval - the *motion control* time resolution.
- The coordinates are interpolated to a *motor control* time resolution and converted to stepper motor direction and step bits, then loaded into a buffer.

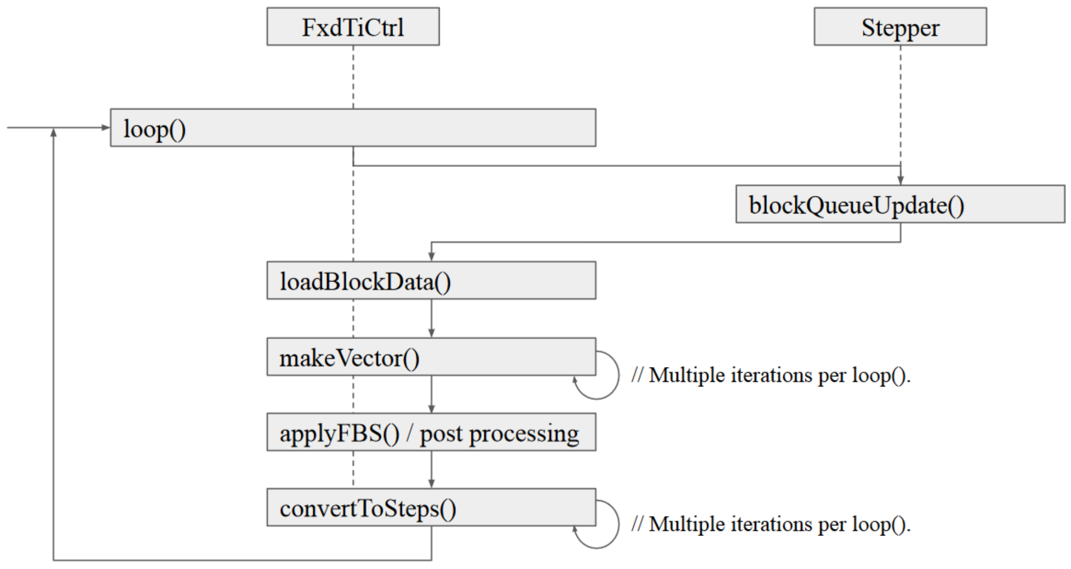
FxdTiCtrl
Stepper
loop()
blockQueueUpdate()
loadBlockData()
makeVector()
// Multiple iterations per loop().
applyFBS() / post processing
convertToSteps()
// Multiple iterations per loop().

Figure 1: Main loop() sequence.

Concurrently, the stepper pulse phase interrupts as needed to retrieve and output commands ready in the buffer.

The majority of changes are to the stepper module (Marlin\src\module\stepper.cpp/.h) and a new module added to contain the coordinate generation logic (Marlin\src\module\fixed_time_ctrl.cpp/.h).

3.2 Tasks

Tasks are modified as follows:

- The interval for Stepper::isr() is set according to the step timing generated by convertToSteps(). The upper limit of the interrupt rate is the *motor control* time resolution. (e.g.: 50 µsec or 20000 Hz). With fixed-time-control enabled, the stock pulse_phase_isr() and block_phase_isr() are no longer called. Stepper pulses are output using Stepper::fxdTiCtrl_stepper().
- Within the Stepper::isr(), a slower task (e.g.: 2.5 ms or 400 Hz) is setup to call the auxiliary functions: endstops.update(), babystepping_isr(), and fxdTiCtrl_refreshAxisDidMove(), added to keep the axis_did_move variable updated.
- FxdTiCtrl::loop() is added as the main loop controlling the fixed time conversion and stepper command generation. It is called as part of the idle() task.

3.3 Fixed Time Control Functions

The following outline and Figure 1 describe the logic flow throughout the main loop(), and describes the purpose for each function called.

- Block Conversion

  - The first check is if processing has been started on a block by using the blockProcRdy variable. If a block isn't ready, stepper.fxdTiCtrl_BlockQueueUpdate() is called to check the planner for new blocks.
  - Once a block is ready, its trapezoid data is loaded using loadBlockData().
  - Once trapezoid data is loaded, a batch of coordinate points are generated using makeVector(). Input-shaping and linear advance adjustments are applied as this occurs. Data is loaded into arrays xd, yd, zd, and ed.
  - Once a batch of coordinate points is generated, its indicated ready for post processing using the batchRdy variable. Once all points of the block are generated, the block is marked complete using blockProcDn and is then discarded.

- Post Processing

  - The main loop() will apply post processing (Ulendo FBS or other) to the coordinate points once a batch is ready. Completion is indicated with batchRdyForInterp variable.
  - Post processed data is loaded into xm, ym, zm, and em arrays.
  - With no post processing this extra step / storage is not necessary. This and the coordinate point storage are the primary storage overhead / cost for this approach, and is proportional to the batch size.

- Interpolation

  - Once post processing is ready, loop() invokes convertToSteps() over each coordinate in the data arrays.
  - This function interpolates between two [position] coordinate points and discretizes the interpolated positions into motor steps.
  - As this is performed, interpolated points that result in a motor step are loaded into a buffer, stepperCmdBuff[], as a command byte consisting of a DIR output bit and STEP output bit for each axis. Two accompanying buffers contain timing of the step command and an indicator of whether the DIR output needs to be set.

4 Offline Configuration

The implementation can be configured using the definitions suffixed to Marlin\Configuration_adv.h. These allow for changing the rates / time intervals for motion planning and stepper control, and controlling the size of aforementioned buffers. Additionally, some variables are provided that can be used to control the average processing time of the loop().

Other configurations are available as definitions suffixed to Marlin\Configuration.h. These are meant to control default modes (such as enabling these changes, enabling an input-shaper, enabling linear advance etc.) and default calibrations (such as input-shaper frequencies, or the linear advance gain).

5 Live Configuration - M950 G-code

The M950 G-code allows live changes of all the calibrations as follows:

5.1 Usage

M950 [M<int>] [P<int>] [K<float>] [D<int>] [A<float>] [B<float>] [G<float>] [H<float>]

5.2 Parameters

[M<int>]	Sets the mode as:
	0: Fixed time control disabled; stock behavior.
	1: Fixed time control enabled.
	10: Fixed time control enabled w/ ZV input-shaping.
	11: Fixed time control enabled w/ ZVD input-shaping.
	12: Fixed time control enabled w/ EI input-shaping.
	13: Fixed time control enabled w/ 2 Hump E-I input-shaping.
	14: Fixed time control enabled w/ 3 Hump E-I input-shaping.
	15: Fixed time control enabled w/ MZV input-shaping.
[P<int>]	Sets the pressure control mode as:
	0: Disabled; no pressure control mode.
	1: Linear advance.
[K<float>]	Sets the linear advance gain [sec].
[D<int>]	Sets dynamic input-shaper frequency mode as:
	0: Disabled. The frequency used is constant.
	1: Z-position-based: The frequency used is calculated based on the Z axis position.
	2: Mass-based: The frequency used is calculated based on the extruder position (total mass extruded).
[A<float>]	Sets the input-shaper frequency [Hz] for the X/A axis. If dynamic frequency is enabled, this is the *base* frequency.
[B<float>]	Sets the input-shaper frequency [Hz] for the Y/B axis. If dynamic frequency is enabled, this is the *base* frequency.
[G<float>]	Sets the input-shaper frequency *scaling* [Hz/mm] for the X/A axis. If dynamic frequency is enabled, this value is multiplied by the Z axis position or E axis position, then added to the *base* frequency to get the active input-shaper frequency.
[H<float>]	Sets the input-shaper frequency *scaling* [Hz/mm] for the Y/B axis. If dynamic frequency is enabled, this value is multiplied by the Z axis position or E axis position, then added to the *base* frequency to get the active input-shaper frequency.

6 Known Issues and Miscellaneous

- No known issues with testing primarily done on the Ender 3 V2 Neo. Other testing was limited to Cartesian, single extruder platforms; not tested on IDEX platforms.
- Users should exercise caution when using this software, especially on other printers. Protections against incorrect configurations are not complete; selection of parameters especially dynamic frequencies should be carefully considered.
- Discontinuous motion might occur if the combined processing time of blocks (e.g.: reading G-code, managing the planner, and fixed-time motion control) exceeds the planned physical motion time commanded by the blocks. This is more likely to occur in sections with many short moves like curves. Increasing the BLOCK_BUFFER_SIZE size may resolve this issue.
- The operation noise generated by the machine changes under this control method.