AN4903
Application note

Generating jerk limited move profiles with the STEVAL-IHM042V1 evaluation board

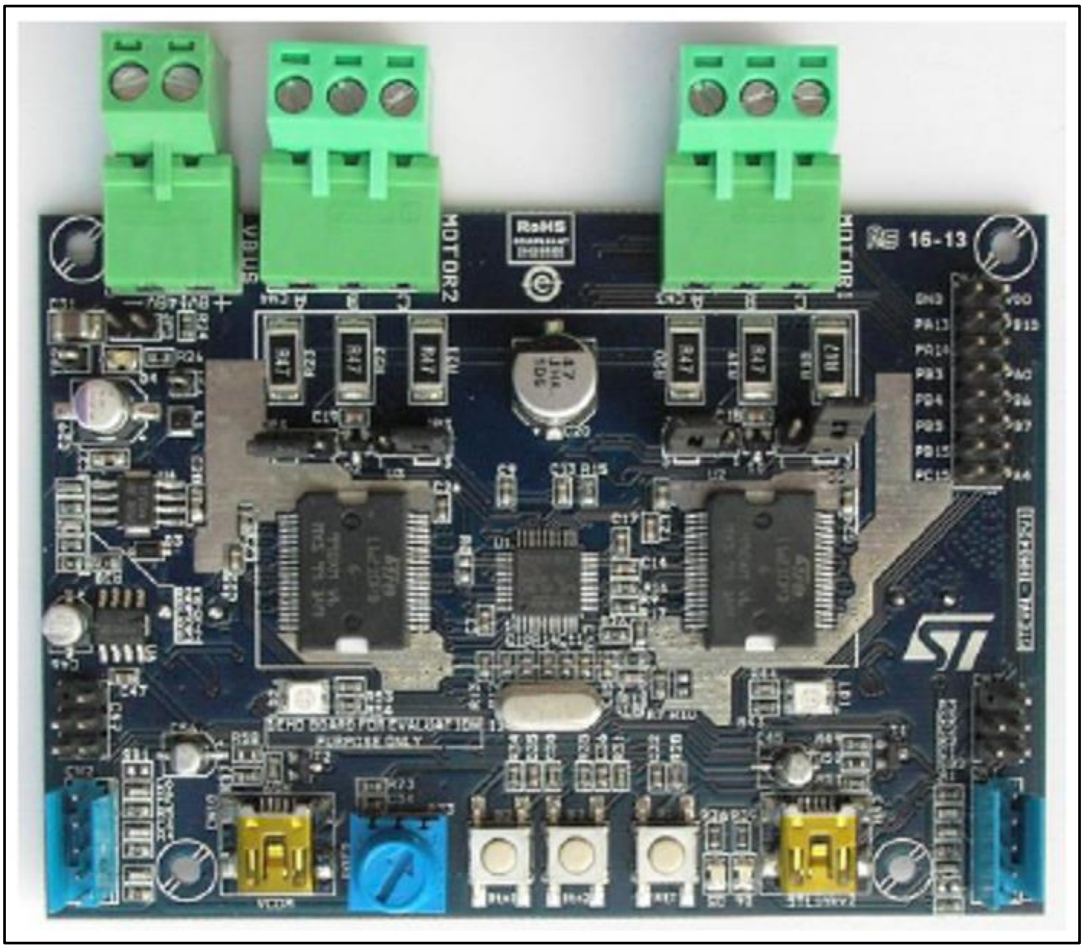
Dennis Nolan

Introduction

The STEVAL-IHM042V1 evaluation board is an ideal target for the development of two axis motion control systems such as camera gimbal drives. The board includes the STM32F303CC which controls two independent L6230 intelligent three-phase inverters.

Figure 1 shows a photo of the board while *Figures 8* and *9* provide the schematic diagrams of the STM32 and L6230 sections of the board. A full description of the board hardware and usage can be found in ST publication UM1605, which is the user manual for the board.

Figure 1: STEVAL-IHM042V1 board photo



Contents

1 Overview

The STM32 has more than sufficient computational power to control two axes of motion. Although many different techniques are in use for gimbal control, in this note we will explore the use of three phase BLDC motors employed in an open loop, sinusoidal, voltage mode. In order to achieve extra smooth movement, we will generate a jerk limited move profile. In actuality, since the bulk of the work required to generate the profile involves the real-time generation of the voltage vector angle, this method could easily be adapted to driving conventional two-phase stepper motors as well.

2 Methodology

Most stepper motor move profiles in use today are of the “trapezoidal” variety, where the operational parameters are specified as acceleration limit, deceleration limit, and maximum speed. While this type of move profile is satisfactory for most motion control applications, it may be unacceptable for some of the more demanding applications such as pan and tilt motions in camera gimbals (which generally require very smooth motion) or applications where liquids are being moved and we need to minimize the “slosh” factor. While the traditional trapezoidal profile controls acceleration, the derivative of acceleration, which is usually called jerk, is not controlled and will actually be allowed to make step changes. It is primarily jerk that we feel when we are riding in an elevator that feels rough. If acceleration is constant then Newton’s law (F = ma) says that we will feel a constant force and this is just as we expect. If acceleration changes abruptly, however, then we feel a variation in force and this is perceived as rough operation.

Many of the newer gimbal designs are using specialized three phase BLDC motors in an open loop, voltage mode configuration. This type of design is directly analogous to conventional two-phase steppers utilizing micro-stepping drive techniques except that the voltage mode drive utilizes a six transistor, three phase, inverter drive instead of two H-bridges. The applied voltages are a traditional three-phase, 120 degree shifted, set. While current mode operation is more common with micro-stepping, bemf compensated voltage mode operation is becoming more popular in order to improve smoothness and reduce hardware requirements. In the case of a gimbal drive, since operating speeds are often very low, bemf and inductance effects are negligible so that voltage mode yields very similar results to current mode.

Important variables in the program for the pan axis are:

- Pantickcounter: unsigned 32 bit. This is the basic time variable. Units are executions of the fixed time interrupt hosting the profile calculations. We will refer to these units as ticks.
- Panmovestate: unsigned 8 bit. This determines the current state of the main state machine (switch statement).
- Panjerk: signed 16 bit. This is generally a fixed magnitude which “drives” the motion. In the course of a symmetrical move profile, the polarity will be reversed in order to drive velocity back to zero at the “landing” position.
- Panacceleration: signed 32 bit. This is the axis velocity and it is the pure time integral of panjerk.
- Panvelocity: signed 64 bit. This is the axis velocity and it is the pure time integral of panacceleration.
- Panposition: signed 64 bit. This is the axis position and it is the pure time integral of panvelocity.

- Panangle: unsigned 16 bit. This is the electrical angle of the sinusoidal output voltage and is scaled such that there are 1024 counts in one electrical cycle. Panangle is derived from and is directly proportional to panposition.

The general formulas of motion are:

P = Po + Vo * t + ½ * Ao * t^2 + 1/6 * J * t^3

V = Vo + Ao * t + ½ * J * t^2

A = Ao + J * t

Where: P is position, V is velocity, A is acceleration, Po is initial position, Vo is initial velocity, Ao is initial acceleration, and t is time. If the initial values are all zero (starting from rest) the we have just P = 1/6 * J * t^3 because of the triple-integration, numerical values have a tendency to increase in magnitude very quickly. In the example program on which this note is based, the fixed time interrupt executes at 20 kHz so a time tick is 50 microseconds. If we assume a jerk value of 1 then, after just one second:

- panacceleration = 50,000
- panposition = 1/2(50,000)^3 = 2.083333333 X 10^13
- panposition = 1/6 (50,000)^3 = 2.083333333 X 10^13

Clearly, these numbers are getting very big very quickly. A signed 32 bit variable can contain a magnitude up to (2.147 * 10^9) so 32 bit is clearly insufficient to hold panposition. 32 bits is also a bit snug for panvelocity as well. Also keep in mind that we do not want to be using nominal values of panjerk of one count since this gives us virtually no resolution for settability. Thus, panvelocity and panposition are best selected as 64 bit variables.

In this example program, as has been mentioned, a natural selection for scaling of the panangle is 1024 counts for one full cycle or 360 electrical degrees. This stems from a natural selection of 256 entries in a sine wave lookup table which covers one quadrant or 90 degrees of the full cycle. Since the sine wave is symmetrical and repeats each quadrant, this is a natural choice. If we use an unsigned 16 bit variable for panangle, then the lower 8 bits represent the angular position within a given quadrant while the next two bits represent which of the four quadrants. In all then, we have 10 bits to represent the full cycle, taking on values of 0 to 1023 for 0 to 360 electrical degrees.

Consider a three-phase gimbal drive BLDC motor with 7 pole pairs, which is a common value for these types of motors. One rotation would thus be represented as a change of 1024 * 7 or 7168 counts of panangle. If we were to use a scaling of one to one between panposition and panangle, then our earlier example (with panjerk=1) would result in a motion of:

2.083333333 X 10^13 / 7168 = 2.906436012 * 10^9 rotations. And that would be in one second!

Clearly, we need a scale factor between our purely mathematical panposition and our real-world panangle. For ease of calculation, consider a scale factor of 2^-32 (shift right by 32 bits). In the case of a jerk setting of 1, this would reduce the total rotation in one second to 0.6767 rotations. This is a far more reasonable number!

In any real world system, we will have to trade off the ability to make a movement quickly against the acceptable level of jerk and the available torque of the system. In working on a small gimbal system recently, I arrived at a value of 20 as a good overall choice for the jerk level. Since position (for a given move time) is directly proportional to jerk, this would result in a movement of 20 * 0.6767 = 13.534 rotations in one second. Now, the pan motion of any gimbal will usually be limited to less than one full rotation, but the desire is also to make the moves in well under one second. Since position is proportional to time cubed, we can precisely calculate the time required to make a move of any desired distance.

In order to get a nice and smooth motion, with our system landing at the required position with zero acceleration and zero velocity, we would like to generate a smooth and symmetrical velocity curve as shown in figure 4. If we differentiate this velocity curve we arrive at the required acceleration profile, which is shown in figure 3. Further differentiating figure 3 yields the required jerk, as shown in figure 2. We can pick and choose the magnitude of the jerk profile to trade off smoothness of motion and magnitude of peak accelerations against the need to make a move in a given time. Clearly, as we allow a higher magnitude of jerk, a given move will be accomplished in less time. In a given system, however, we are likely to select a magnitude of jerk that we will stick with, at least within a given set of operating circumstances.

In our example, we have chosen a magnitude of 20 as our jerk limit. Operating with a fixed magnitude of jerk, and within this profile, total movement will be determined by the time we spend moving. Since the applied jerk profile must be symmetrical, we can define the profile by the amount of time that we initially apply a positive value of jerk, knowing that we will always then apply a negative jerk for twice that interval and then finishing with positive jerk for the same interval. We will call this interval the panbase and in this example panbase is 50 ticks. We can read from figure 5 that the total move distance is 5,305,804 units.

We can see in figure 7 that, when the panbase is increased from 50 to 100, the move distance increases to 42,448,300 which illustrates that, allowing for some error because of the flat-topped integration of my spreadsheet simulation, the move distance will be proportional to the cube of the panbase. Twice the panbase yields eight times the move. To complete the analysis/characterization of this profile, figure 6 shows the position vs. time with the panbase back to 50 but the jerk changed from 20 to 30 and from this we can read the final position at 7,958,940. This indicates, as expected, that the move distance is directly proportional to jerk. With this information, we can characterize the expected move distance as follows:

Move distance = 5,305,804 * (jerk/20) * (panbase/50) ^ 3

Going back to our original example of a system with seven pole pairs, a scale factor of 2^32 between panposition and panangle, and an intrinsic scaling of 1024 counts of panangle for one electrical, let’s calculate the move parameters for a move of 45 mechanical degrees. 45 mechanical degrees will be 45 * 7 = 315 electrical degrees. 315 electrical degrees translates into (315/360) * 1024 = 895 panangle counts. 895 panangle counts translates into a panposition value of 895 * 2^32 = 3.84399573 X 10^12. If we stick with a jerk setting of 20, we have 3.84399573 X 10^12 = 5,305,804 * (panbase/50)^3 and this implies a value for panbase of 4490.7 and a full move time of 4 * 4490.7 = 17962.8 ticks. Since we assumed 20,000 ticks per second, this is a total move time of 17962.8/20,000 = 0.89814 seconds.

So, if we program our move profile generator with a jerk setting of 20, and a panbase of 4491, we should hit our move distance target of 45 mechanical degrees with a smooth, jerk limited profile, in a move time of 0.89814 seconds. If this move time is too long, we can increase the jerk setting and re-calculate the required panbase time.

3 Pratical implementation

The following C language function is a practical implementation of this motion profile generator. Inputs are the panjerkset and panbasetime global command variables and the continuous output is panposition. It should be executed in a fixed time routine. Another routine will scale panposition into panangle and using a sine wave lookup table, generate duty cycle values to write to the microcontroller timer that generates gating signals out to the three-phase bridge.

```
void doprofile(void)
{
pantickcounter++; // measures relative time
switch(panmovestate)
{
case 0:  // initialization state
pantickcounter=0;
panjerk= panjerkset;
panvelocity=0;
panacceleration=0;
panmovestate=10;
break;
case 10: // first profile segment, positive jerk
panacceleration = panacceleration + panjerk; //integrate jerk into acceleration
panvelocity = panvelocity + panacceleration; //integrate acceleration into velocity
panposition=panposition+panvelocity;;           // integrate velocity into position
if(pantickcounter==panbasetime)  { panjerk= -panjerkset; panmovestate=20;
pantickcounter=0; }
break;
case 20: // second profile segment, negative jerk
        panacceleration = panacceleration + panjerk;
        panvelocity = panvelocity + panacceleration;
        panposition =panposition+panvelocity;
        if(pantickcounter==(panbasetime*2)){ panjerk = panjerkset; panmovestate=30;
pantickcounter =0;}
        break;
case 30:  //
        third profile segment, back to positive jerk
        panacceleration = panacceleration + panjerk;
        panvelocity = panvelocity + panacceleration;
       panposition=panposition+panvelocity;

if(pantickcounter== panbasetime ) {panvelocity=0;panmovestate=40;}
break;
case 40:  /move completed.
          panposition=pannewpos;  // just for insurance in case of small profile
error.Should already be there.
          panmovestate=50;
          break;
case 50:  // just holding
break;
} // end of pan move state
} // end of doprofile function
```

To trigger a move, we can use the panmove function. The function argument, degrees, is intended to be mechanical degrees assuming a motor with 7 pole pairs. This must be re-scaled within the function for a different number of pole pairs.

```
void panmove( signed short degrees){unsigned long long llong0;
          pannewpos = degrees;
          pannewpos = pannewpos * 85517571050;
         pannewpos = pannewpos + panposition;
         if(degrees>=0) panjerkset=20;
         panmovestate=0;
         llong0 = 2137939276;
         llong0 = llong0 * degrees;
```

```
         llong0 = cuberoot(llong0);
          panbasetime = llong0;}
```

Just to be complete, here is a cube root function that runs well on the STM32. It utilizes the STM32 hardware multiply capability with an overall successive approximation approach.

```
unsigned short cuberoot( unsigned long long x)
        {unsigned long long test;
         unsigned short word0;
         unsigned short word1;
         word0 = 0x8000; // sliding 1's
         word1 = 0;
         while(1)
         {word1 = word1 | word0;
          test=word1;
test = test * test *test; //cubed
if(test>x) word1 = word1 ^ word0;
word0 = word0>>1;
if(word0==0) break;
} // end of while loop
   return word1;
} // end of while loop
return word1;
} // end of cube root function
```

4 Waveforms

Figure 2: Applied jerk vs. time with jerk = 20, panbase = 50

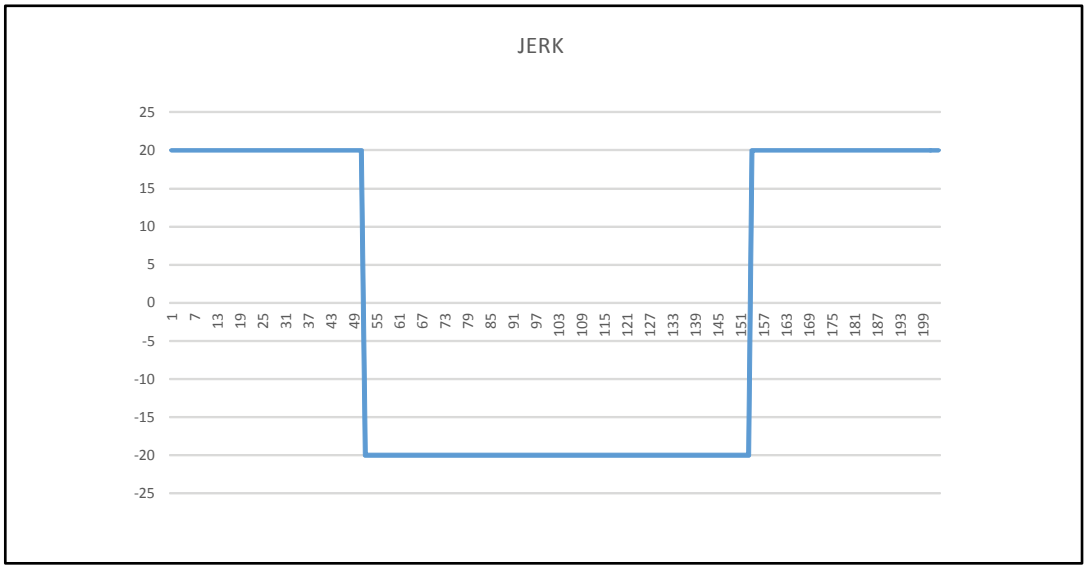


Figure 3: Acceleration vs. time with jerk = 20, panbase = 50

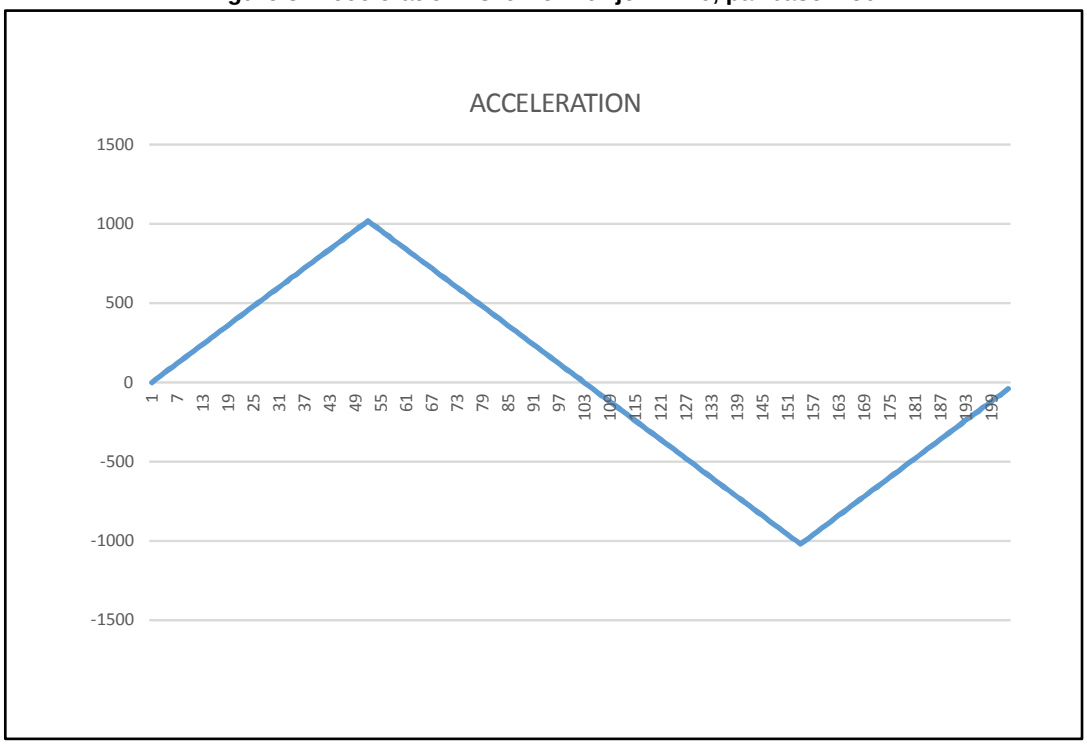


Figure 4: Velocity vs. time with jerk = 20, panbase = 50

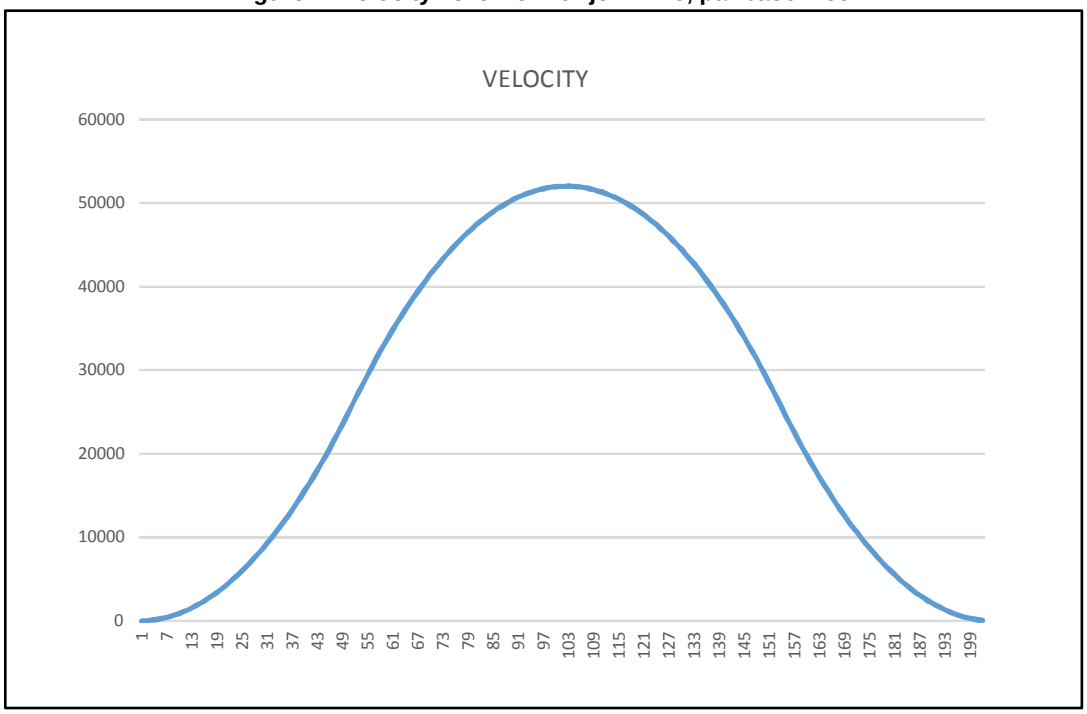


Figure 5: Position vs. time with jerk = 20, panbase = 50

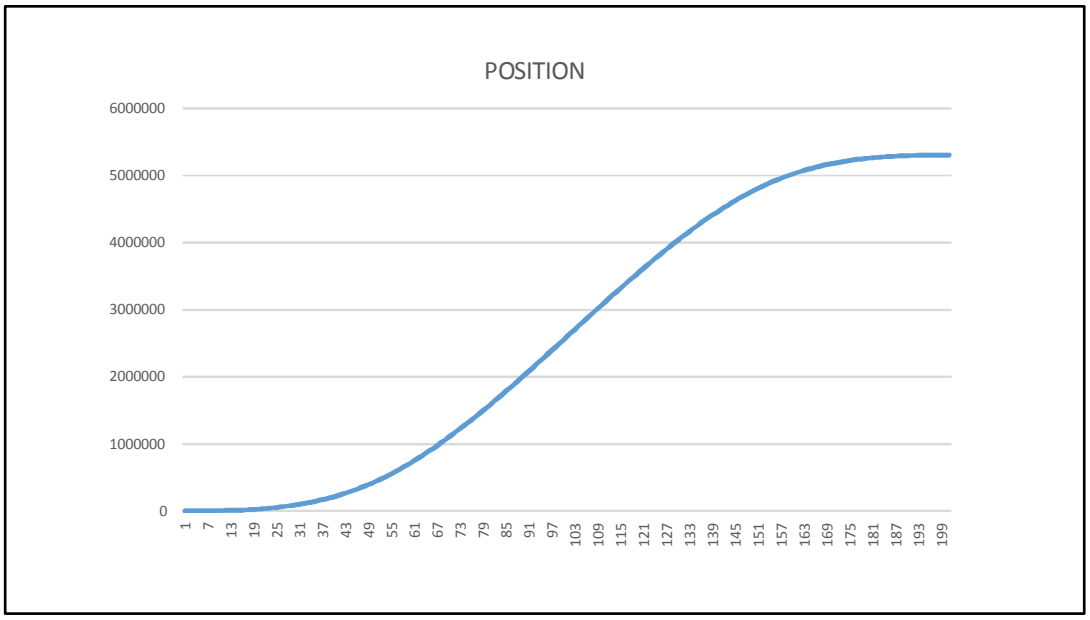


Figure 6: Position vs. time with jerk = 20, panbase = 100

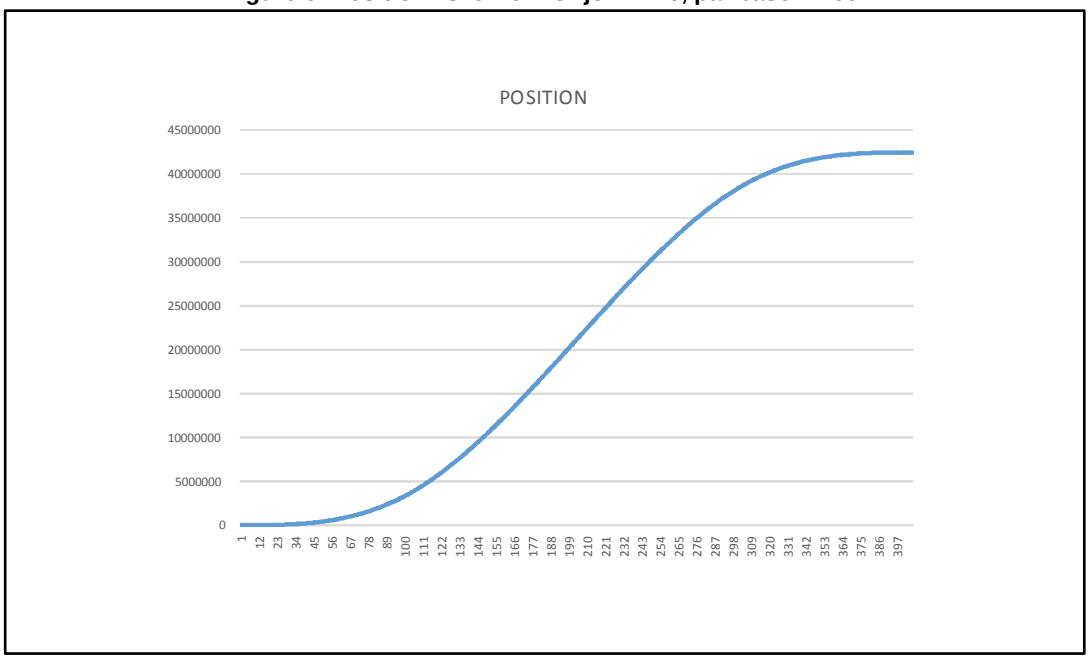


Figure 7: Position vs. time with jerk = 30, panbase = 50

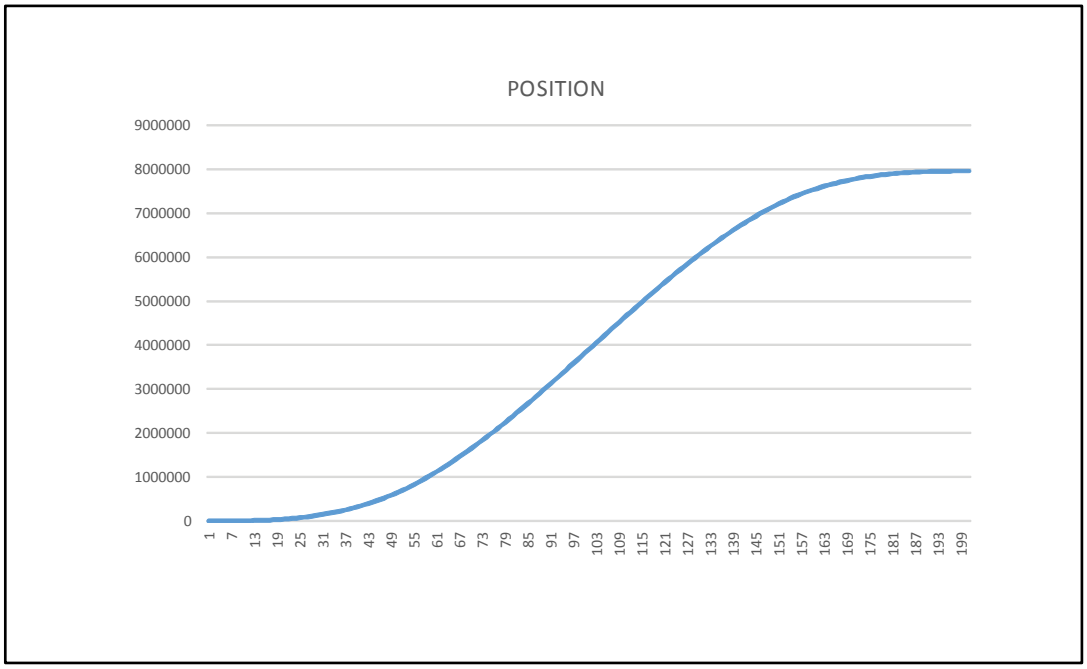


Figure 8: STM32 schematic diagram

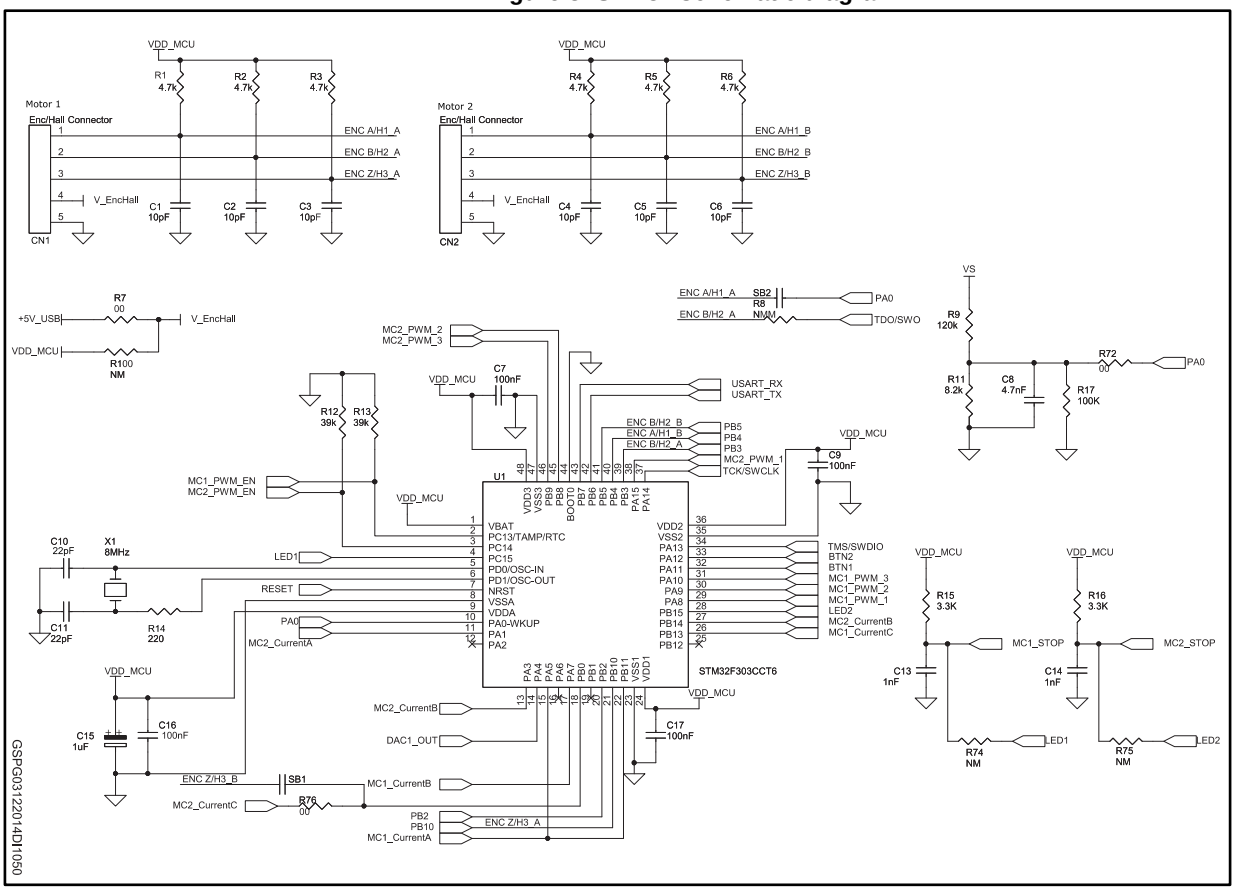
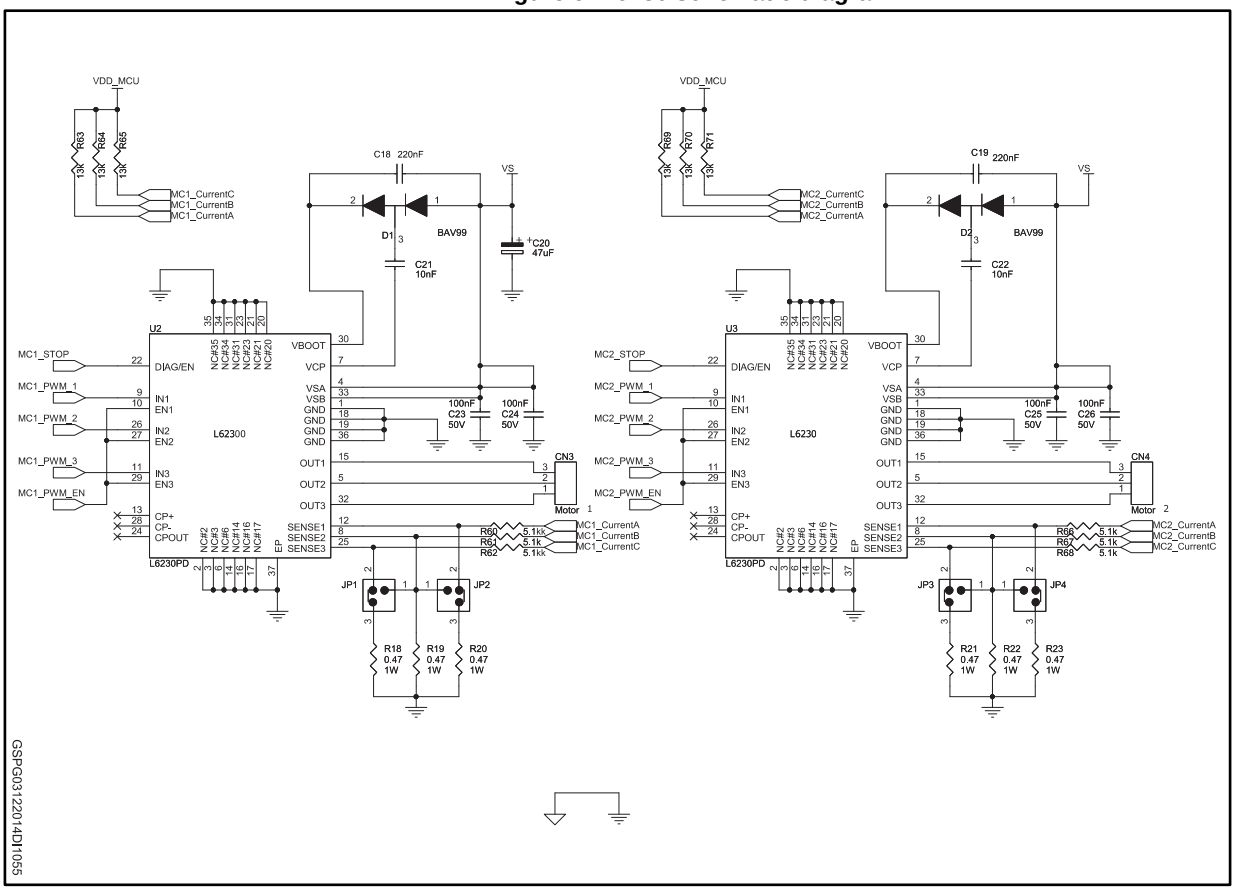


Figure 9: L6230 schematic diagram



5 Revision history

Table 1: Document revision history

Date	Revision	Changes
16-Feb-2017	1	Initial release.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.