Challenges going faster than certain rpm

daniel **Feb '22**

Hi community,
First I want to say, thank you for being so awesome. The library is a great project and it's amazing to see the involvement of everyone here on the forum.

I've been fidgeting with simplefoc, with different drivers, controllers, sensors and both BLDCs and stepper motors. And even though I get my motors to spin and get pretty good torque out of them, there are some aspects of which I'm not sure if it's going as well as it should. Some of the main things being that the max rpm I'm getting is a bit lower than expected, and that the movement of the motors often doesn't seem very smooth.

First off, my current setup:

- Raspberry Pi Pico controller
- Drivers : BTS7960B
- Angle sensor: AS5600
- Motor: cheap 14pole 360KV 5010 bldc from Ali-express
- Current sensors: 2 x ACS712 hall sensor
- powered from 2 li-ion cells (~7.4V)

I'm currently testing with voltage control mode. I've also gotten both of the current sense modes to work, but results were not necessarily better.

With this setup (and basically any other I've tried with the same motor), it seems that there is an 'invisible wall' to get the motor to spin at more than ~100 rad/s in closed loop velocity control or about 150 rad/s in closed loop torque control mode. Raising values above that just makes the motor more noisy, draw a lot more current and make rad/s more unstable or even go down.

Skip to main content

In open loop velocity mode I can get it to around 200 rad/s, by manually stepping up both the target velocity and current limit. I haven't tried to go above that as the sweet spot for the current limit becomes smaller and smaller, so it just becomes tedious.

I don't think the limitation is the speed of the controller; my main loop runs at a fixed 2.5 kHz (both move and foc) I've read at other posts here that foc should work well with at least 4 loop iterations per pole pair (or was that per pole, so 8 per pair?), meaning that purely from that aspect I think I should be able to reach ~560 rad/s (2500hz / 7pp /4 its/pp * 2 * pi), so that's quite a way to go from 100 or 150.

I think one issue I have is that the magnet is not perfectly aligned to the sensor. At low rpm there is a notable 'pattern' of different sounds and variation in velocity over the course of a motor rotation. However I don't think this is the issue as with higher rpm the motor is actually running smoother and smoother.

So, 1: is there anything obvious that I should be doing differently?
And 2: could this possibly be an effect of the delay in the read-out of sensor values? I haven't calculated or measured the actual delay, but as this is generally the limiting factor in loop iteration speed, I think it's save to assume that by far most of the iteration time is spend in waiting for I2C communication back and forth, meaning that the sensor value was actually sensed around half of the iteration time earlier, let's say 200uS in my case, by the time by the voltages are adjusted based on that reading. I would imagine that at low rpm this delay is no problem as it is only as small fraction of the time between two passing poles, but this fraction of course grows as rpm goes up and therefore time between passing poles reduces. Is there any compensation for this delay in the library?

🔗 BLDC Motor recommendations for Direct Drive R…

Valentine Feb '22

Skip to main content

> daniel:

> a bit lower than expected

> daniel:
>
> Angle sensor: AS5600

That's a bit pushing it. You may get better results if you go for SPI AS5047 sensor. AS5600 is a simple digital potentiometer (100RPM max), I'm surprised you got that far on it.

PS The AS5600 does not even list the RPM in the specs, it's that low.

> magnetized on-axis magnet. This AS5600 is designed for contactless potentiometer applications and its robust design eliminates the influence of any homogenous external stray magnetic fields.

Antun_Skuric **Feb '22**

Hey @daniel ,

Yes, your MCU is fast and you can have loop times even much higher than 2.5kHz if you use a different sensor. I2C based AS5600 is really not meant for high speeds or fast loop times. But even an ideal sensor you will still not be able to come to 500rad/s.

> daniel:
>
> In open loop velocity mode I can get it to around 200 rad/s, by manually stepping up both the target velocity and current limit. I haven't tried to go above that as the sweet spot for the current limit becomes smaller and smaller, so it just becomes tedious.

The open-loop control is really really fast. cca 10x-100x faster then the FOC and it makes a lot of difference.

> daniel:
>
> With this setup (and basically any other I've tried with the same motor), it seems that there is an 'invisible wall' to get the motor to spin at more than

Skip to main content

> ~100 rad/s in closed loop velocity control or about 150 rad/s in closed loop torque control mode. Raising values above that just makes the motor more noisy, draw a lot more current and make rad/s more unstable or even go down.

Velocity control algorithm takes longer time to execute and that is probably the reason why you see such a high difference. Try using the motion downsampling to let more time to the FOC algorithm and less for your motion control as your task for the moment is very simple. With some motion downsampling you should be able to reach at least this 150rad/s.

> daniel:
>
> I don’t think the limitation is the speed of the controller; my main loop runs at a fixed 2.5 kHz (both move and foc)

You are using a timer?

> daniel:
>
> I don’t think the limitation is the speed of the controller; my main loop runs at a fixed 2.5 kHz (both move and foc) I’ve read at other posts here that foc should work well with at least 4 loop iterations per pole pair (or was that per pole, so 8 per pair?), meaning that purely from that aspect I think I should be able to reach ~560 rad/s (2500hz / 7pp /4 its/pp * 2 * pi), so that’s quite a way to go from 100 or 150.

There is part of the equation that influences a lot and that is power supply voltage. As when going faster your back-emf rises motor’s torque fill diminish and it will not be able to reach higher speeds. So one simple way you can increase your speed range would be to use a higher power-supply voltage.

> daniel:
>
> So, 1: is there anything obvious that I should be doing differently?
> And 2: could this possibly be an effect of the delay

Skip to main content

> in the read-out of sensor values? I haven't calculated or measured the actual delay, but as this is generally the limiting factor in loop iteration speed, I think it's save to assume that by far most of the iteration time is spend in waiting for I2C communication back and forth, meaning that the sensor value was actually sensed around half of the iteration time earlier, let's say 200uS in my case, by the time by the voltages are adjusted based on that reading. I would imagine that at low rpm this delay is no problem as it is only as small fraction of the time between two passing poles, but this fraction of course grows as rpm goes up and therefore time between passing poles reduces. Is there any compensation for this delay in the library?

We do not do the compensation of the delay in the library in this way. We do have the adaptive time sampling which will help in motion control applications but not in the field vector setting. There is something that you can do if you are motivated though, we do use the `motor.zero_electric_angle` which basically determines the calibrated offset in between sensor and the motor 0 angles. For higher velocities (due to this delay you are interested in) we will have a certain offset in between where the motor is and where we this it is which will depend on velocity.
So you could add your estimation of the offset to the `motor.zero_electric_angle`:

You could do something like this in your loop

```
float normal_zero_angle = motor.zero_electric
float alpha = 0.1; // something small - this
void loop (){
   motor.zero_electric_angle=normal_zero_angl
   ...
   motor.loopFOC();
   motor.move();
}
```

daniel **Feb '22**

Skip to main content

Thank you for your responses! I was not aware that there was anything like an RPM-limitation on the AS5600. I've just received some AS5048A boards, hopefully I'll manage to print a mount for it this weekend and dome so testing with it. I'll report back on the results!

I don't necessarily need to reach 500 rad/sec, I think that for my application (biped robot legs with probably 1:9 reduction) should be okay with 200 rad/sec, maybe even less. I'm just very curious what the limiting factors are (and of course, getting higher speeds is more satisfying 😉)

I had assumed the difference between velocity control and torque control would mostly be caused by poor PID/filter tuning. I will experiment more with downsampling as well. I have been testing with higher supply voltages (adding extra li-ion cells). I mostly got higher torques but I don't think I got much higher speeds. Also with a simple ESC and two cells these motors reach much higher RPMs, so I didn't think voltage should be the limiting factor. But perhaps speed on an ESC and with FOC is completely incomparable?

For now I've simply limited the loop iteration speed with a simple timing check (and tested that the slowest loop iterations finish well in time of course). I might play around with some interrupt-based options later.

I realize that indeed much higher speeds can be achieved with open-loop control, given it's simplicity and non-dependence on slow sensors and noisy values etc. But with zero ability to deal with disturbances of course. Is it expected though that I need to manually adjust the current limit corresponding to the speed? IE, at low speeds it doesn't work with a high current limit so I need to start out with low current limits, then as speed is going up it doesn't work anymore with the low current limits so I need to crank up the limits.

I was already sniffing around in the library code to find a good spot to experiment with some sort of timing compensation based on current velocity, but your suggestion to adjust the zero_electric_angle is much simpler! I will definitely play around a bit with that 🙂

Skip to main content

Thanks again for your thoughts!

---

daniel **Feb '22**

I tried with the AS5048 and it was a difference like day and night! The motor is running super smooth at both low as well as higher rpm and effortlessly reaches velocities well above 200 rad/s! I sort of feel like I've just been wasting a lot of time with the 5600 😛 although I do wonder if the difference is fully caused by the sensor itself, or if perhaps the alligment also plays a role, since the 5600 are not even centered on their boards properly (its the cheap white boards from Ali Express) and the cogging and alignment tests reported error distributions in the range of 30 degrees with the as5600 (after I got it to work, see **https://github.com/simplefoc/Arduino-FOC/pull/157**).

I did notice something weird with the 5048 though. In the sensor test example sketch it just worked flawlessly, but with I my own code it didn't work. Somehow it gets a very low clock rate (~2kbps), at which it seems the chip just refuses to respond. After a lot of trial and error I noticed one difference between the senor test example and my code : the sensor test example uses the as5147 default config and my code used the as5048 default config. This turned out to be the key: when I changed my code to use the 5147 default config it immediately worked. I'm super puzzled over this though, because as far as I understand from checking out MagneticSensorSPI.cpp, the as5048 default config is literally a copy of the as5147 default config. They do clearly behave differently though, so there must be something going on.

---

runger **Feb '22**

> daniel:
>
> Ithough I do wonder if the difference is fully caused by the sensor itself

It's really mainly the I2C interface vs. SPI. That makes a huge speed difference, which is what is allowing you to

Skip to main content

reach higher RPMs. But it is certainly also true that the 10 vs 14 bits and the quality of the sensor PCB help get smoother, better results.

Thanks a lot for reporting the problems with the MagneticSensorSPI class. I've made an issue and we'll investigate it.
In the meantime you could also try the MagneticSensorAS5048A class from our drivers repository. This is specific to the AS5048A, and also lets you read out the magnitude and diagnostic registers which you can't get with the generic MagneticSensorSPI class.

RaphZufferey **Sep '23**

Hello !!

I've been able to successfully run full FOC on the G431B-ESC board with an SPI magnetic sensor (AS5047) a 1 MHz thanks to all the great tips I could find on this forum (**B-G431B-ESC1: Beginner guide + I2C guide - #91 by Grizzly**). Motor is 100W AT2303 1500KV.

While operation at low RPMs runs smoothly after tuning, I can't achieve speeds beyond 750 rad/s. As I increase velocity, current draw also increases as expected (all below 1A). Going beyond 600-800 rad/s, the current suddenly jumps to the power source limit (3A), the motor becomes very noisy and the reported speed actually decreases (and doesn't reach the target).

For my application, I would need to go to 1500 rad/s. Is there any tips you would have to reach that? I am open to any changes, going open loop at high speed or even switching to 6 step control etc…

Speed limited to ~400 rad/s with low resistance, h…

Motor not reaching rated speed

Candas1 **Sep '23**

Skip to main content

Have you tried voltage mode with lag compensation ?
What is your power supply voltage ?
At what voltage target are you having problems ?

---

runger **Sep '23**

Hey, 1500rad/s is pretty fast for FOC control. You'll have to push the MCU to its limits 🙂

How many pole pairs is this motor? Probably 7?

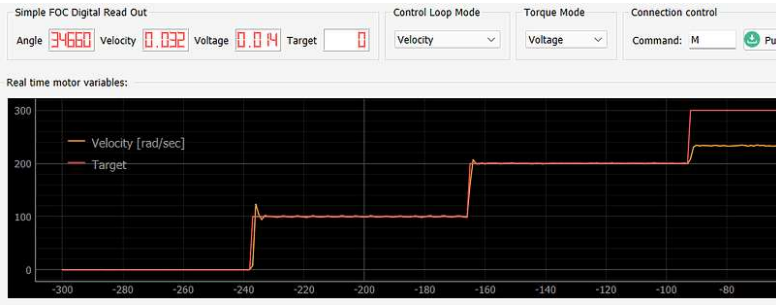
Could you add some timing to estimate your main loop speed (like, count the number of iterations in one second).

One thing you could try is to activate the CORDIC for the trig calculations, this should give a small speed increase.

RaphZufferey **Sep '23**

Thanks for the super fast reply!

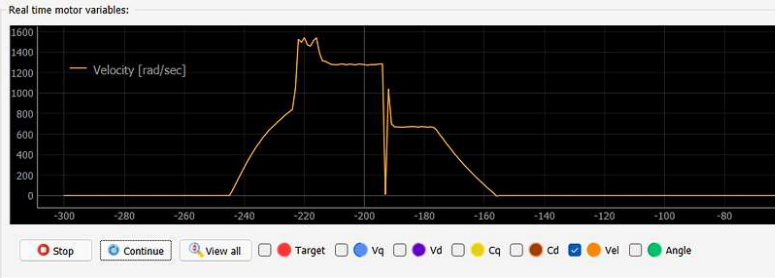
Trying Torquemode:voltage does not let me go beyond ~230rad/s. I will try the lag compensation now.



Power supply: 12V
I've been doing all my tests in velocity feedback, as such I don't have voltage targets. I'll update soon on that.

Here you can see the jump around 750 rad/s when I control in current (FOC current mode):

Skip to main content



@runger , that's what I figured looking through the comments, but it still seemed possible? I will try your suggestions, thank you so much! Yes, the motor has 7PP. Any chance that switching to 6 step control at higher speed could be a solution?
EDIT: the main loop runs at **7 kHz** quite consistently across speeds. In voltage mode I get to 7.8 kHz.

Skip to main content

Skip to main content

Skip to main content

Skip to main content

Skip to main content

Skip to main content

Skip to main content

VIPQualityPost

Sep '23

> runger:
>
> changing to a Pico would not be faster, in my opinion.

Oh, maybe I misunderstood. If the G431 is being used, just keep that! I saw in the top post that the Pico was the board being used here.
I definitely think the STM32HWEncoder is the way to go here.

---

runger **Sep '23**

> RaphZufferey:
>
> Would it be imaginable to stop using the sensor completely at higher speeds and instead rely on BEMF measurement or is this non-sense?

It's not nonsense, and the B-G431-ESC1 is set up for voltage measurements in terms of the hardware, IIRC. But it's not supported at all in our library at the moment… so in those terms it's not a quick solution for you.

I believe the combination of ABZ Sensor used via HWEncoder and the CORDIC optimization may get you to your goal.

---

dekutree64 **Sep '23**

> Anthony_Douglas:
>
> This delay between the info of the sensor coming in and then the electrical angle being set is a significant thing for sure, and not just due to I2C communication times. I used PWM and analog and it was still an issue just due to code execution time, because it affects motor timing.

In the PM conversation @Candas1 mentioned in post #17, we were thinking of leveraging SmoothingSensor for this. Add a new variable to apply a fixed offset to the

Skip to main content

timestamp before calculating the predicted angle. To get good results, you'd have to measure the average time between the sensor update and PWM duty update on your specific hardware.

But if using hardware current sense, you'd ideally want to make two separate angle predictions, or perhaps use the previous frame's angle for the current transformation. Otherwise you'll be transforming the start-of-the-frame current readings by the end-of-the-frame angle, and cancel out your improvement. But I'm not sure how to cleanly integrate that into the library code.

How fast can a BLDC rotate?

---

o_lampe **Sep '23**

> RaphZufferey:
>
> Motion Downsample = 3

I tried downsample rates of 22 on my ST32F030, but I didn't notice any difference.
I also mentioned before, there is a typo in the default.h file (@runger)

```
// default monitor downsample
#define DEF_MON_DOWNSMAPLE 100 //!< default m
#define DEF_MOTION_DOWNSMAPLE 0 //!< default
```

I've changed the names, but maybe that's the reason why it doesn't work in my case?
Maybe the typo goes on in other files?

RaphZufferey **Sep '23**

Hi all,

I've tried to play with the **motor.zero_electric_angle** value to see whether this would help, as suggested **above**. And the difference is massive.
Not only do I reach speeds of **1700 rad/s** (and possibly

Topics

Skip to main content

Categories

announcement

applications

developement

giveaway

hardware support

All categories

Tags

stm32

b_g431_esc

as5600

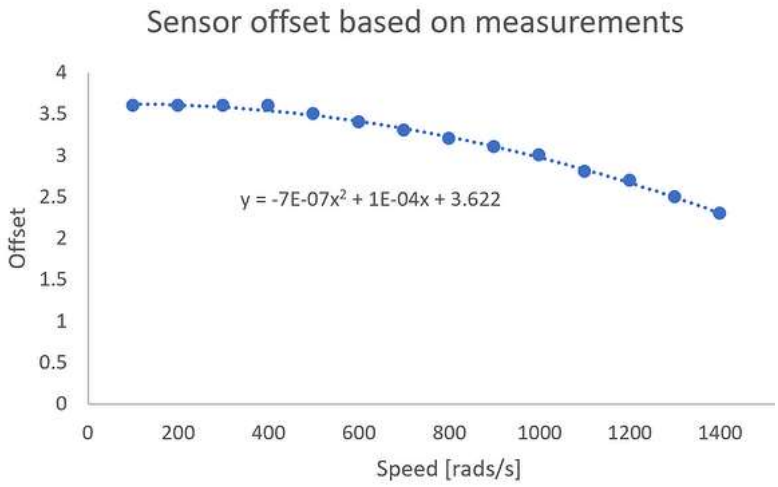
esp32

arduino_uno

All tags

beyond), the currents are much lower at high speeds. Main loop is still around 13 kHz with the changes I reported here.

I tweaked the sensor offset manually at every speed so at to get the lowest multimeter-based current draw and then fitted the data as follow:



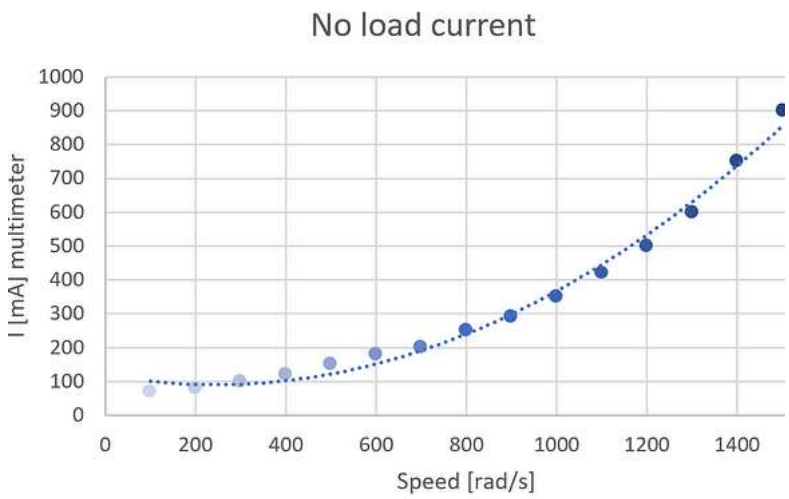

```
float normal_zero_angle = 0;
void setup(){
    ...
    normal_zero_angle = motor.zero_electric_a
}
void loop(){
    static int i = 0;
    i ++;

    motor.loopFOC();
    motor.move();
    motor.monitor();
    command.run();

    if(i%10 == 0){
        motor.zero_electric_angle=-0.0000007*
    }
```

Since the new sensor value calculation was slowing down the main loop by 1 kHz, I chose to only update the value every 10 loops.

Pasting here the current measurements with this correction. Note that beforehand, with a static

Skip to main content

zero_electric_angle, at above 1000 rad/s I was maxing out my power supply at 5A.



Candas1 **Sep '23**

> Candas1:
>
> One workaround in voltage mode could be to use the lag compensation and tweak the phase_inductance to also compensate for the delay.

As I said you can overcompensate with this

[EDIT] I think as the speed increases the interrupts slow down loopfoc more and more

---

VIPQualityPost **Sep '23**

wow, really impressive result!
how did you plot the offset graph? manually adjusting it or did you setup some kind of automated measurement?

RaphZufferey **Sep '23**

All manual, tuning the offset value at every speed point. Since the current measure (which I was trying to minimize) was taken from a multimeter, I couldn't really automate anything.

Skip to main content

Candas1 **Sep '23**

@runger Do you guys really need the **adc interrupts** ? This is probably also slowing down foc_current.

In my gd32 implementation I read the injected adc registers only in **getPhaseCurrents** function.

[EDIT] **Example** with STM32F1

Before:
RAM: [=] 9.1% (used 4456 bytes from 49152 bytes)
Flash: [===] 26.8% (used 70256 bytes from 262144 bytes)
loopfoc=296us

After:
RAM: [=] 9.0% (used 4404 bytes from 49152 bytes)
Flash: [===] 26.7% (used 69880 bytes from 262144 bytes)
loopfoc=266us

Same would be possible with the G4 implementation as it's also using injected adc

[EDIT2] And you will see a difference even in voltage mode, because if current sense is initialized, those interrupts are triggered even if you are not in foc_current mode.

---

runger **Sep '23**

This seems like a good idea to me. 🙂

I don't have an F1 set up to test it at the moment…

Candas1 **Sep '23**

Ran it yesterday on a STM32F103RCT6(hoverboard controller), it was working.
Just want to make sure I am not missing something else.

I made it optional so the current setup with interrupts can stay, in case someone want's to run loopfoc from there in the future.

Skip to main content

I have a few G431B-ESC's also, I really need to set it up so I can also test this on G4.
There will be a lot more people that can test this I imagine.

Candas1 **Sep '23**

Just one doubt @runger , there are 2 ways to use current sense with G431.
The G431B-ESC specific one with DMA's and no interrupt, and the generic G4 implementation that uses interrupts and injected adc?

---

o_lampe **Sep '23**

> Candas1:
>
> Ran it yesterday on a STM32F103RCT6(hoverboard controller), it was working.
>
> There will be a lot more people that can test this I imagine.

I can test it on a GD32F103RCT if you like.
I had no current sensing so far and was pretty happy with plain voltage controlled velocity.
Just need to know what and where I have to download.
But let's discuss this in the hoverboard thread…

New & Unread Topics

Topic	Replies	Views	Activity
Arduino Uno | magnetic_sensor_spi_example | Arduino IDE working & PlatformIO not working	**8**	213	**May '23**

Skip to main content

Topic	Replies	Views	Activity
Problem with B-G431B-ESC1 Current Sensing b_g431_esc current_sensing	**30**	803	**Jun '23**
A thread for finally figuring out sensorless drive strategies	**49**	1.7k	**Oct '23**
RC ESC based Brushed DC motor closed loop control brushed continuous-servo dc-servo rc dc-motor	**16**	615	**Aug '23**
Motor Coil Current	**1**	101	**Nov '23**

Want to read more? Browse all categories or view latest topics.