Stand demonstrativ pentru controlul in bucla al unui motor de curent continuu cu perii colectoare

DEAC George-Antoniu
NASTASE Robert-Paul
Facultatea de Inginerie Industrială și Robotică, Specializarea Robotică, Anul de studii 3, e-mail: george@impromedia.ro
Conducător științific: S.l. dr. Ing. Cozmin CRISTOIU

Abstract: *The paper presents the conceptual design, virtual prototype achievement and the real physical system implementation of a test stand for electric motor cascading PID control using a modified firmware version of an ODrive motor controller in order to control both, brushless and brushed DC electric motors. The initial version of the controller board is dedicated only to brushless motors, but with current firmware upgrade the cheaper brushed motors can be PID controlled proficiently. The firmware upgrade will also allow low latency force-feedback. Results from testing of the positioning closed loop control are presented. The presentation test bench consists of: an Odrive controller board (with a custom firmware), a typical brushed DC motor, a 4000cpr incremental optical encoder, individually designed 3D printed mounting brackets, indicator and angular ruler, an additional ESP32 microcontroller and an adjustable power supply.*

Key words: *Brushed motor, Brushless motor, cascading PID, motor controller, position control, force feedback.*

¶

1. INTRODUCTION

PID (proportional integrative derivative) are closed loops that are widely used in industry and not only, for systems that use electric motors. The control loop is continuously calculating an error value, e(t), as the difference between a desired setpoint, r(t), and a measured process variable, y(t), and applies a correction based on proportional, integral and derivative terms [1]. The classic PID control diagram is shown below in figure 1.

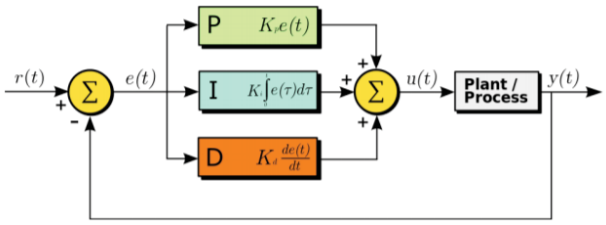

Figure 1 Classic PID control loop [2]

The general form of the control signal given by a PID controller has the following mathematical form:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt} \quad (1)$$

Where *kp, ki, kd* are a set of parameters used to tune the strength of the P, I and D parameters of the controller. Nowadays there are more architectures of PID (based on same principles) like: feedback (classic), feedforward and cascading. No matter the architecture, the key of a good control of a system consists in the fine tuning of the three parameters.
The ODrive motor controller board is a cascaded style position, velocity and current control loop, as shown in the diagram below.

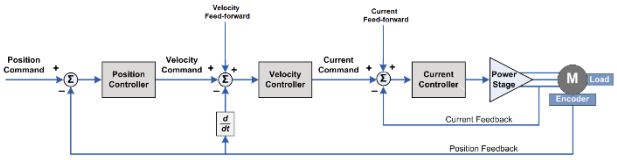

Figure 2 ODrive board motor controller loop [3]

Each stage of the control loop is a variation on a PID controller. This flexibility is essential as it allows the ODrive to be used to control all kinds of mechanical systems. The initial ODrive board is dedicated for brushless DC motor control in association with rotational encoders (optical incremental or HAL sensors). A typical setup for brushless motor control is shown in figure 3.

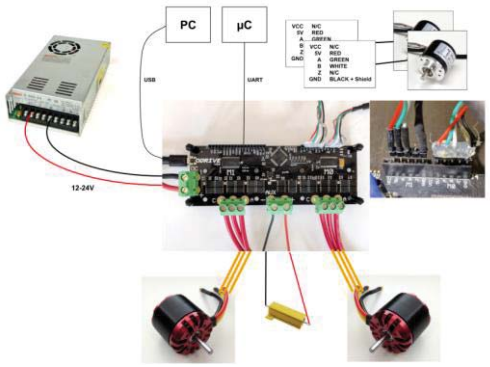

Figure 3 Brushless motor setup [3]

In this setup, the Odrive is communicating with the ESP32 interfacing microcontroller via UART using its proprietary ASCII protocol implemented on the fibre abstraction layer which also handles the communication with the PC via the virtual USB serial port. The Odrive is also connected to the optical encoder via the dedicated axis0 A, B, Z pins and the motor leads are coupled to the

last two phases of the axis. The board pinouts are configured as follows:

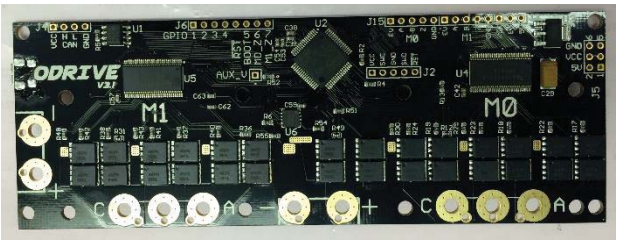

Figure 4 Odrive board layout [4]

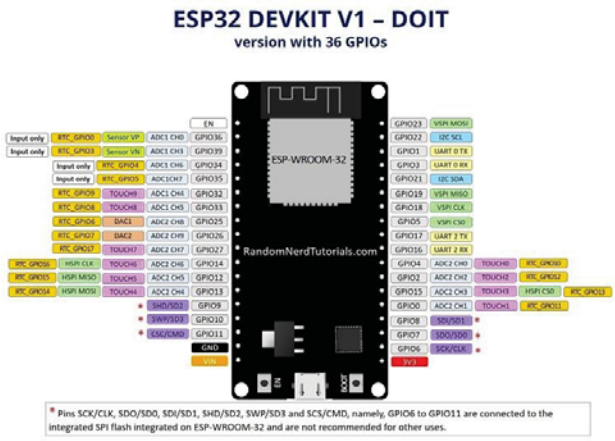

Figure 5 ESP32 Dev Kit board layout [5]

The purpose of this paper is to present the methods applied in order to upgrade the boards firmware with the goal of extending its functionality and compatibility with the cheaper brushed DC motors.

2. FIRMWARE UPGRADE

In order for the controller to work with brushed motors, the original firmware state machine needed to be modified with the specific entries in the following enumerations marked by (*):

- Motor Types:

0 High current (default)
1 Low current (not implemented yet)
2 Gimbal
3 (*)Brushed current
4 (*)Brushed voltage (this one is used currently)

- Current State:

0 Undefined state (will fall through to idle)
1 Idle state (disable PWM and do nothing)
2 Startup Sequence (the actual sequence is defined by the config.startup_ flags)
3 Full calibration sequence (run all calibration procedures, then idle)
4 Motor calibration (run motor calibration)
5 Sensorless control (run sensorless control)
6 Encoder index search (run encoder index search)
7 Encoder offset calibration (run encoder offset calibration)
8 Closed loop control (run closed loop control)
9 Axis lockspin (lockin spin)
10 Encoder direction find
11 (*)Brushed current control (not implemented)
12 (*)Brushed voltage control (run open loop brushed voltage control)

The main modifications to the principal subroutines of the firmware consist in: skipping the calibration procedure if the motor type is set accordingly for brushed motors, forcing a null encoder offset and implementing a custom voltage timings function to drive and equilibrate 2 required phases out of the 3 phases on the axis.

Moreover, the ascii protocol logic was also modified to facilitate faster response times and decreased latency for providing better force feedback. Thus, a dedicated command was integrated for the current comprised of only one character for faster serial communication and for the structure of the lookup table, hash map and ordered map were tested instead of the previously slow else/if chains.

For a typical communication scenario where the ESP32 microcontroller requires the current intensity from the Odrive, there are 3 types of latency involved: the initial packet transmission time for requesting the current, the lag caused by the replying processor (Odrive) overhead, the replied packet transmission time which holds the current value and the receiving processor overhead (ESP32). The last overhead represents the smallest one and typically can't be further improved, so is the replied packet transmission time which only comprises a value. This implementation tackles to improve the transmission time of the initial packet and the replying processor overhead. An expected time duration of a force feedback communication is composed as follows:

10bit/symbol (there is a start and a stop bit + 1byte word)
115200 baud rate UART => 115200 bit/sec => 0.0086ms/bit

Initial packet for requesting the current value = "r axis0.motor.current_control.Iq_measured\n" = 42 symbols = 420bit/115200bit/sec = 3.65ms

Replied packet containing the current value = 10 symbols = 100bit/115200bit/sec = 0.87ms

Average communication time = Packet transmission time + Reply processor overhead (Odrive) + Reply transmission time + Receive processor overhead (ESP32)
The default transmission duration is around 5ms.

In conjunction with the communication latency, the actual delay also encompasses the access time of the else/if chains present in the command interpreter which represents the most overhead of the replying processor, in O(N) in time complexity, thus the access time grows linearly to the number of entries in the protocol. The

introduction of a hash map which is O(1) time complexity for access or an ordered map, of O(logN) complexity, will further improve the speed by reducing the reply processor overhead.

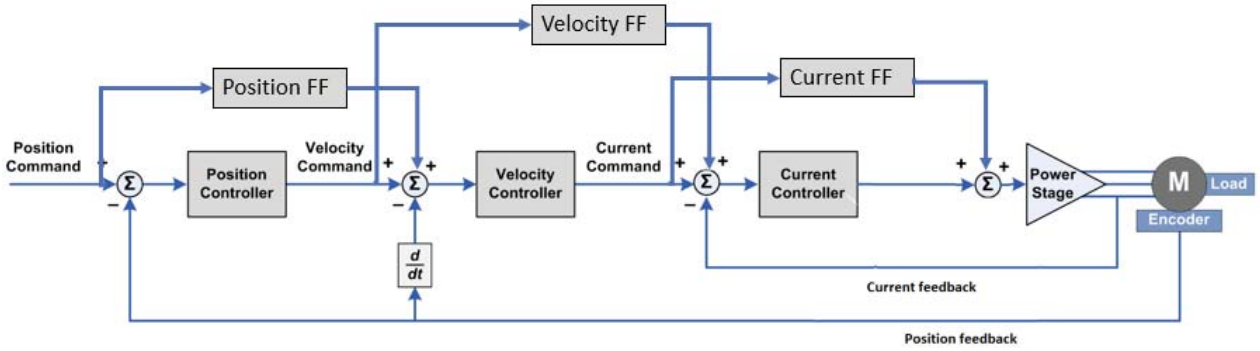

Figure 6 ODrive board motor controller cascading PID

The mathematical relations for the position loop, velocity loop and current loop are written as follows:

a) Positioning loop

pos_error = pos_setpoint - pos_feedback
*vel_cmd = pos_error * pos_gain + vel_feedforward*

b) Velocity loop

vel_error = vel_cmd - vel_feedback
*current_integral += vel_error * vel_integrator_gain*
*current_cmd = vel_error * vel_gain + current_integral + current_feedforward*

c) Current loop

current_error = current_cmd - current_fb
voltage_integral += current_error
current_integrator_gain
*voltage_cmd = current_error * current_gain +*
voltage_integral (+ voltage_feedforward when we have motor model)

Tuning the motor controller is an essential step to unlock the full potential of the ODrive. Tuning allows for the controller to quickly respond to disturbances or changes in the system (such as an external force being applied or a change in the setpoint) without becoming unstable.

Correctly setting the three tuning parameters (called gains) ensures that ODrive can control your motors in the most effective way possible. For now, gain values were determined empirically [4]. The gain values determined were set up via controller interface using the following command lines:

<axis>.controller.config.pos_gain = 100
<axis>.controller.config.vel_gain = 0.0005
<axis>.controller.config.vel_integrator_gain = 0.00005

The startup procedure usually requires running the motor calibration sequence. In this case, since we use a brushed motor and the commutation is done mechanically an automatic calibration of the motor is not required and therefore it's skipped.

3. EXPERIMENTAL STAND

The test stand is shown in the figure 4 and it includes: the Odrive controller board (with the upgraded firmware), a brushed DC motor R406-011E Sanio Denki, an incremental optical encoder 1000PKVF3 P1215 with 4000cpr, the mounting board and 3D printing brackets with an indicator and an angular ruler, an additional *ESP32* microcontroller and an adjustable power supply.

Key electrical, mechanical and electromagnetic specifications of the used motor and the controller board are presented in tables 1 and 2:

Table 1: Motor specifications

Nominal Power	60W
Rated Torque	0.19 Nm
Rated Current	1.4 A
Rated Speed	3000 rpm
Max Speed	5000 rpm
Max angular acceleration	111x103 rad/s^2
Rotor inertia	0.0108x10^{-3} kg·m2
Armature inductance	4.4 mH

Table 2: ODrive board specifications

Control	2 motors
Voltage	24 V
Peak current	>100A per motor
Braking modes	Brake resistor and regenerative braking
Interfaces	USB, Step/direction, UART, Servo PWM,PPM, CAN, digital and analog pins.

Protocol	Goto (positioning control with trajectory planning), Position command, Velocity command, Torque command

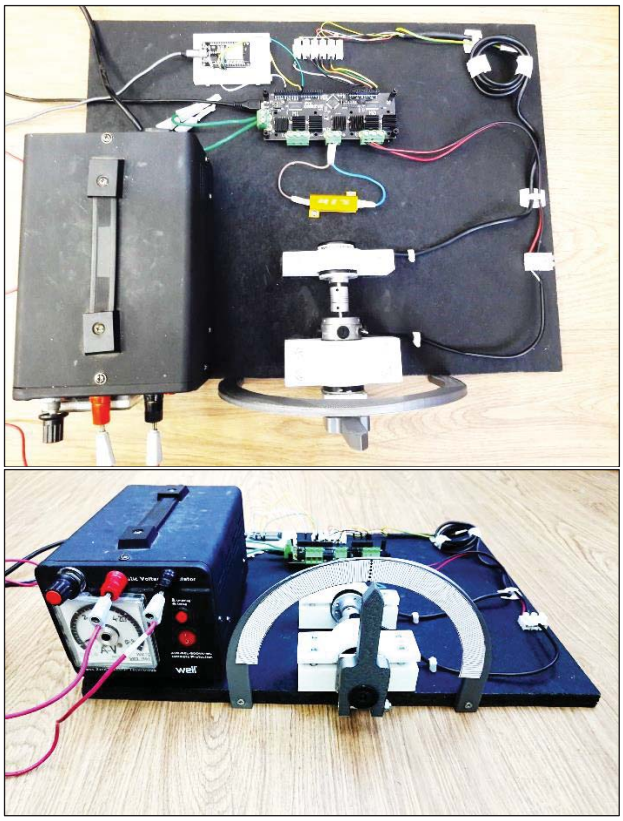

Figure 7 Test stand

The encoder is assembled on the motor back shaft and both are wired into the corresponding pins of the ODrive board. The *ESP32* microcontroller is communicating via UART serial with the ODrive board and is responsible with the displaying and interfacing of the system. During testing commands are sent to the microcontroller and drive board from a PC via the virtual serial on the fibre abstraction layer.

4. EXPERIMENTAL PROCEDURES

The motion control test was set to measure angular deviation from the programmed position for different speeds and different number of revolutions: 10, 100, 1000 revolutions were programmed for different working speeds (20%, 40% and 60% of the motor rated speed) and the positioning repeatability and precision was measured in each case. The unit of measurement was converted from encoder pulses to degrees for expressing the deviation in an absolute way for any system:

For angular positioning:

$$\theta[º] = \frac{360º}{4000*\frac{counts}{\text{rev}}} = \frac{0{,}09º}{count}$$

$$\Rightarrow \theta[º] = 0{,}09*\theta[count] \quad (2)$$

For angular speed:

$$\omega = \frac{rev}{min} = \frac{4000counts}{min*\frac{counts}{rev}} = \frac{4000counts}{sec*60*\frac{counts}{rev}} = \frac{360º}{sec}$$

$$\Rightarrow \omega[\frac{º}{sec}] = \frac{4000}{60*360}*\omega[\frac{\text{counts}}{sec}] \quad (3)$$

The tests were repeated twice, first time without trapezoidal trajectory and second time with trapezoidal trajectory enabled. The angular acceleration and deceleration (ε_{acc}, ε_{dec}) were set to be double in norm compared to the angular velocity in order to constrain the acceleration (t_{acc}) time to 0,5sec. Experimental values are presented in the following chapter.

5. RESULTS

Subsequently, the test results seem to closely follow an ordered logarithmic pattern following:

Table 3

Repeatability Without Trapezoidal Trajectory			
Rev	**Speed 20%**	**Speed 40%**	**Speed 60%**
	Error [deg]	**Error [deg]**	**Error [deg]**
10	(-)0.99	(-)0.81	(-)0.81
100	3.51	3.78	3.87
1000	6.3	5.13	16.2
10000	7.38	11.88	18.36

Ensuing, it is presented the corresponding line graph:

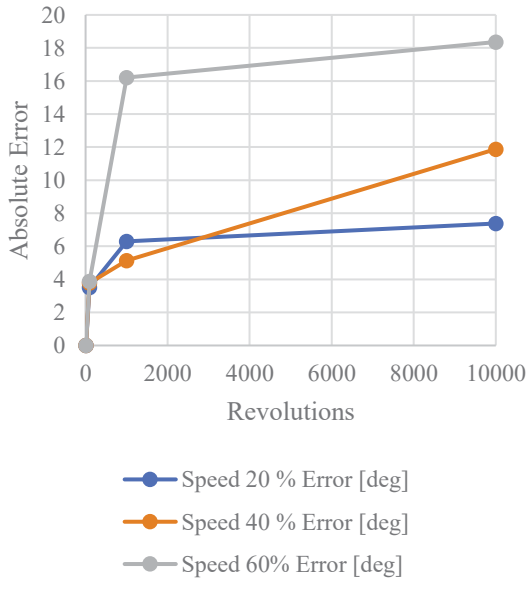

Figure 8

Table 4

Repeatability With Trapezoidal Trajectory			
Rev	**Speed 20%**	**Speed 40%**	**Speed 60%**
	Error [deg]	**Error [deg]**	**Error [deg]**
10	(-)1.17	(-)0.09	(-)0.45
100	3.69	3.96	3.42
1000	6.75	5.94	17.46
10000	7.83	11.43	18.9

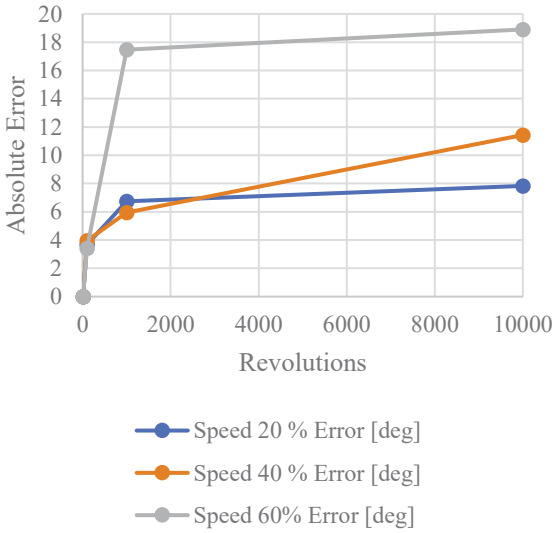

Figure 9

From both of the previously provided tables it can immediately be observed that the compensation trend in the first row with 10rev increments tends to undershoot, whilst in all the other cases there is an logarithmically increasing overshoot error trend which starts to plateau faster in the outer speed regions (20%, 60%). Whereas, the middle speed region (40%) shows a wider error variation tolerance.

Moreover, by comparing the similarity in the error curves, it is shown that the main causative factor of the errors is the improper empirical PID tuning, since the results are similar, independently of the chosen trajectory generation scheme.

To conclude, this test is essential in showing potential deviations that are caused outside the PID positioning control loops, such as improper field-oriented control (FOC) commutation [7], potential causes influenced by variation in the inertial loading profile of each trajectory type (for instance a rectangular trajectory has by far the highest inertia peaks, while in the case of a trapezoidal or S-shaped trajectory the inertial loading is evenly distributed in time) such as mechanical problems.

6. FUTURE IMPROVEMENTS

Lastly, the force feedback requires additional experimentation using an oscilloscope to precisely determine the communication time improvements by analyzing each transmitted packet time domain and the corresponding delay between them.

7. CONCLUSIONS

REFERENCES

[1] M. Alboelhassan, “A Proportional Integral Derivative (PID) Feedback Control without a Subsidiary Speed Loop”, Acta Polytechnica Vol. 48 No. 3/2008, Czech Technical University in Prague

[2] Arturo Urquizo. PID Controller—Wikipedia, the Free Encyclopedia. 2011. Available online: https://en.wikipedia.org/wiki/PID_controller (accessed on 30 March 2018).

[3] *** https://docs.odriverobotics.com/control.html

[4] *** https://github.com/madcowswe/ODriveHardware

[5] *** https://randomnerdtutorials.com/wp-content/uploads/2018/08/ESP32-DOIT-DEVKIT-V1-Board-Pinout-36-GPIOs-Copy-768x554.jpg

[6] J.G. Ziegler, N.B. Nichols “Optimum Settings for Automatic Controllers”, TRANSACTIONS OF THE A.S.M.E. NOVEMBER 1942

[7] *** https://en.wikipedia.org/wiki/Vector_control_(motor)