Proceedings in Manufacturing Systems, Volume 14, Issue 3, 2019, 85-90 ISSN 2067-9238

# PID CONTROL WITH CUSTOM CONTROLLER FIRMWARE FOR BRUSHED DC MOTORS

**Cozmin CRISTOIU[1,*], Mario IVAN[2], George DEAC[3], Robert NASTASE[4],**

[1)] Assist. Prof., PhD Student, Robotics and Production Systems Department, University "Politehnica" of Bucharest, Romania
[2)] Lecturer, PhD., Robots and Manufacturing Systems Department, University "Politehnica" of Bucharest, Romania
[3)] Student, Robots and Manufacturing Systems Department, University "Politehnica" of Bucharest, Romania
[4)] Student, Robots and Manufacturing Systems Department, University "Politehnica" of Bucharest, Romania

***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 goal of this firmware update and of the tests that are presented in the paper is to show that this new firmware upgrade allows brushed DC motors to be PID controlled by the ODrive controller board and not only the brushless motors for which the board was initially developed. The project was developed in four stages. During the first stage the core system components – the brushed DC motor, the controller and the encoder – were integrated and the issue of correct impulse reading was addressed. During the second stage the experimental stand was built. In the third stage, the PID control algorithm was implemented. The fourth stage consisted of measurements regarding the angular accuracy of the motor spindle positioning using the previously developed algorithm. 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 that will allow future improvements and control optimization. Results from testing of the positioning closed loop control are presented.*

***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 in Fig. 1 [2].

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

$$u(t) = Kp \cdot ep(t) + Ki \int_0^t ei(\tau) d\tau + Kd \frac{de_d(t)}{dt}. \quad (1)$$

Where $K_p$, $K_i$, $K_d$ 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.

Regardless of 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 Fig. 2 [3]. This flexibility is essential as it allows the ODrive to be used to control all kinds of mechanical systems. The advantages of the ODrive board are linked to the fact that it is open source (both hardware and software), affordable and offers high performance control for robotics and brushless motors. 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 Fig. 3 [3]. In this setup, the Odrive (shown in Fig. 4 [4]) is communicating with the ESP32 interfacing microcontroller (shown in Fig. 5 [5]) via UART using its proprietary ASCII protocol implemented on the fiber 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 axis. A, B, Z pins and the motor leads are coupled to the last two phases of the axis. The board pinouts are configured as shown in Fig. 4 [4].

* Corresponding author: Splaiul Independentei 313, Bucharest, Romania
Tel.: 0742133392;
E-mail addresses: *cozmin.cristoiu@gmail.com* (C. Cristoiu), *andrei.mario@yahoo.com* (M. Ivan), *george.deac@impromedia.ro* (G. Deac),

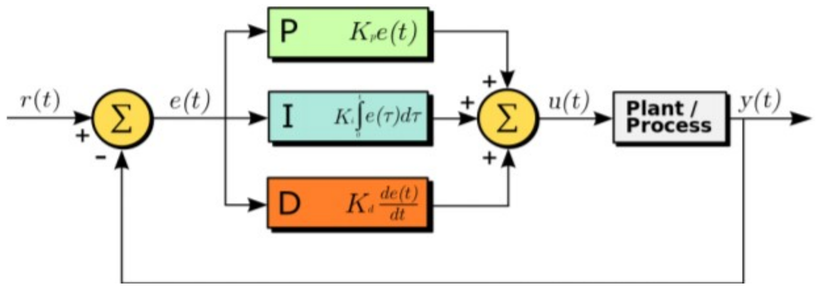

**Fig. 1.** Classic PID control loop [2].

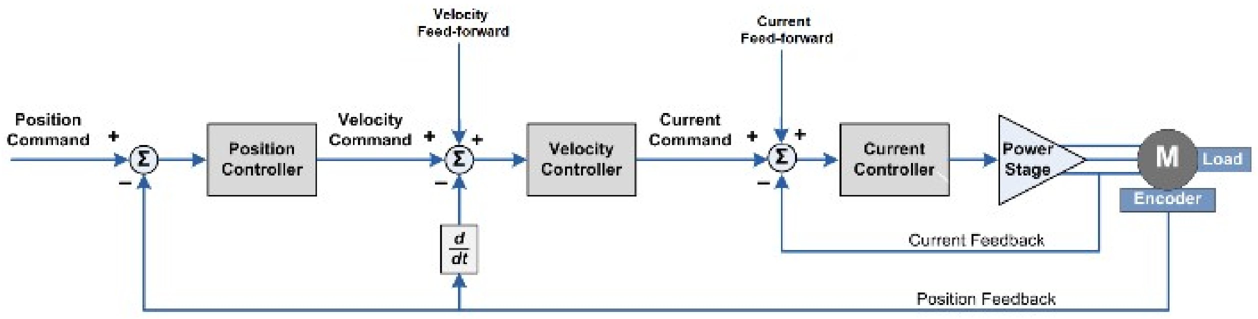

**Fig. 2.** ODrive board motor controller loop [3].

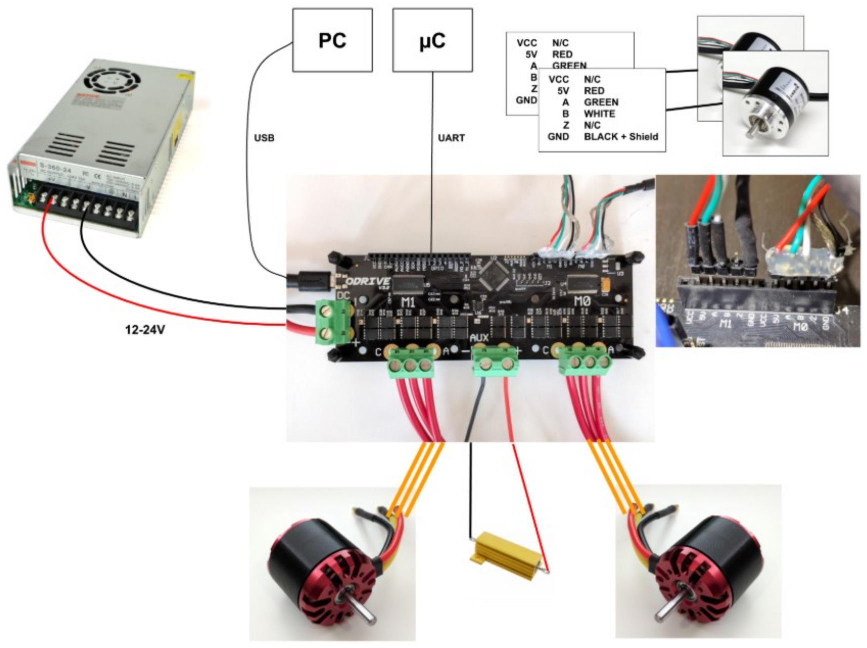

**Fig. 3.** Brushless motor setup [3].

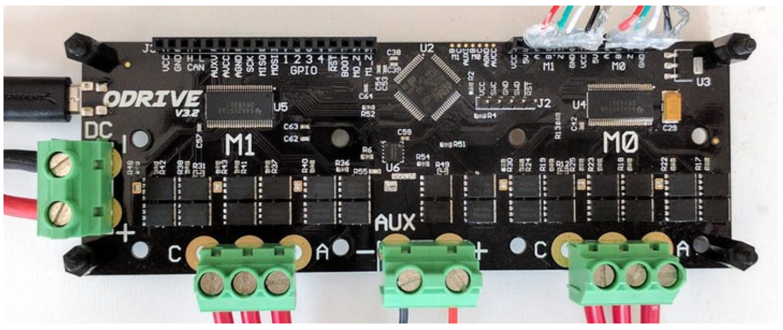

**Fig. 4.** Odrive board layout [4].

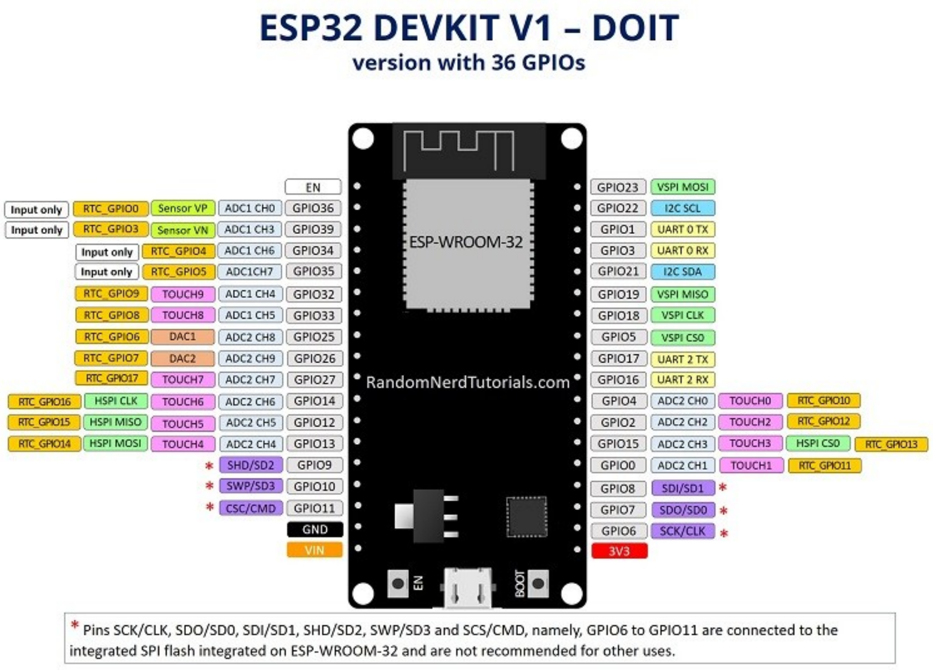
ESP32 DEVKIT V1 – DOIT
version with 36 GPIOs
ESP-WROOM-32
RandomNerdTutorials.com
EN
Input only | RTC_GPIO0 | Sensor VP | ADC1 CH0 | GPIO36
Input only | RTC_GPIO3 | Sensor VN | ADC1 CH3 | GPIO39
Input only | RTC_GPIO4 | ADC1 CH6 | GPIO34
Input only | RTC_GPIO5 | ADC1CH7 | GPIO35
RTC_GPIO9 | TOUCH9 | ADC1 CH4 | GPIO32
RTC_GPIO8 | TOUCH8 | ADC1 CH5 | GPIO33
RTC_GPIO6 | DAC1 | ADC2 CH8 | GPIO25
RTC_GPIO7 | DAC2 | ADC2 CH9 | GPIO26
RTC_GPIO17 | TOUCH7 | ADC2 CH7 | GPIO27
RTC_GPIO16 | HSPI CLK | TOUCH6 | ADC2 CH6 | GPIO14
RTC_GPIO15 | HSPI MISO | TOUCH5 | ADC2 CH5 | GPIO12
RTC_GPIO14 | HSPI MOSI | TOUCH4 | ADC2 CH4 | GPIO13
* SHD/SD2 | GPIO9
* SWP/SD3 | GPIO10
* CSC/CMD | GPIO11
GND
VIN
GPIO23 | VSPI MOSI
GPIO22 | I2C SCL
GPIO1 | UART 0 TX
GPIO3 | UART 0 RX
GPIO21 | I2C SDA
GPIO19 | VSPI MISO
GPIO18 | VSPI CLK
GPIO5 | VSPI CS0
GPIO17 | UART 2 TX
GPIO16 | UART 2 RX
GPIO4 | ADC2 CH0 | TOUCH0 | RTC_GPIO10
GPIO2 | ADC2 CH2 | TOUCH2 | RTC_GPIO12
GPIO15 | ADC2 CH3 | TOUCH3 | HSPI CS0 | RTC_GPIO13
GPIO0 | ADC2 CH1 | TOUCH1 | RTC_GPIO11
GPIO8 | SDI/SD1 *
GPIO7 | SDO/SD0 *
GPIO6 | SCK/CLK *
3V3
EN
BOOT
* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

**Fig. 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

The core subject of the paper consists in the new firmware upgrade. Unlike the original version of the controller board that is designed only for control of brushless motors, this new upgrade allows also for more cheaper brushed DC motors to be proficiently controlled through PID. 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);
- 11 5200 baud rate UART => 115200 bit/sec => 0.0086 ms/bit;
- Initial packet for requesting the current value = "r axis0.motor.current_control.Iq_measured\n" = 42 symbols = 420 bit / 11 5200 bit/sec = 3.65 ms;

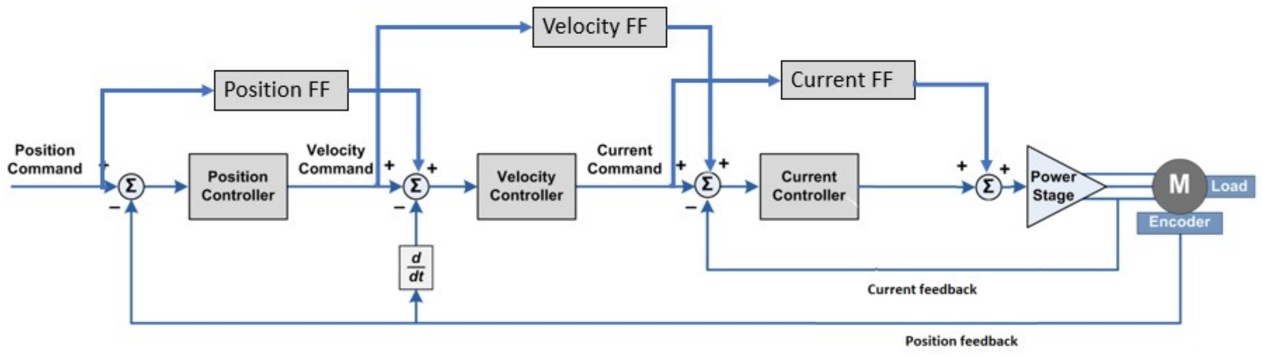

**Fig. 6.** ODrive board motor controller cascading PID.

- Replied packet containing the current value = 10 symbols = 100 bit / 115200 bit/sec = 0.87 ms;
- Average communication time = Packet transmission time + Reply processor overhead (Odrive) + Reply transmission time + Receive processor overhead (ESP32).

The default transmission duration is around 5 ms.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.

The mathematical relations for the position loop, velocity loop and current loop are transposed in firmware code as follows:

- Positioning loop:

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

- 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.*

- 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, as shown in Fig. 6. For now, gain values were determined empirically [6], only by manual trials for gain factors values until visible improvements could be observed. 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 Fig. 7 and it includes: Odrive controller board (with the upgraded firmware), brushed DC motor R406-011E Sanio Denki, incremental optical encoder 1000PKVF3 P1215 with a resolution of 4000 pulses per revolution (also known as counts per revolutions or "cpr"), mounting board and 3D printing brackets with indicator and protractor, additional *ESP32* microcontroller, and adjustable power supply.

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

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.

*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 |

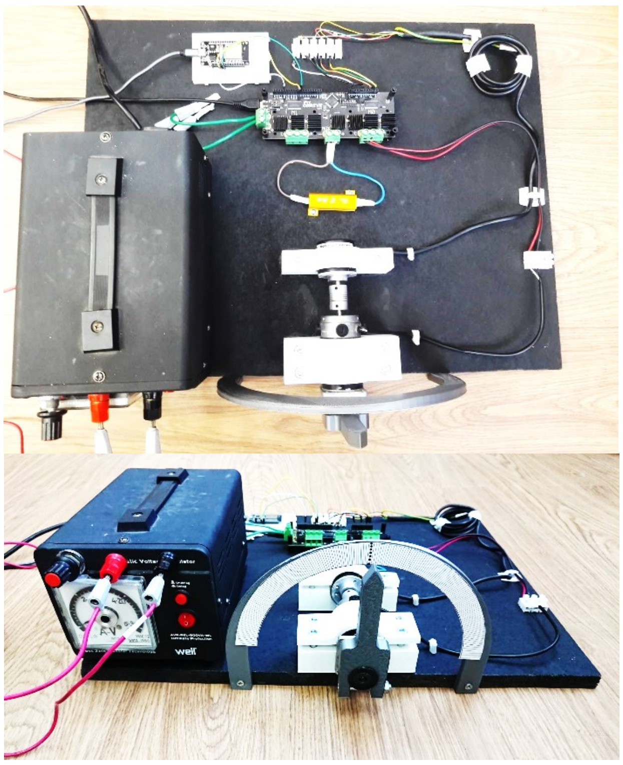
**Fig. 7.** Test stand.

## 4. EXPERIMENTAL PROCEDURES

For each test, a number of full revolutions was configured through the controller. The number of revolutions was set sequentially at 10, 100, 1000 and 1000, and for each number of revolutions the measurements were conducted using different speeds (20%, 40% and 60% of the motor rated speed). The motion control test was set to measure angular deviation from the programmed position (Δα) at the end of each series of revolutions (after the motor spindle stopped spinning). The positioning repeatability was measured in each case. The unit of measurement was converted from encoder pulses (technically known as "counts") to degrees for expressing the deviation in an absolute way for any system.

$$\theta[°] = \frac{360°}{4000 \cdot \frac{counts}{rev}} = 0.09°/count \cdot \qquad (2)$$

The tests were repeated twice, first time without trapezoidal trajectory (with speed variation slopes in our case set up in order to reach the maximum velocity in 0,5 seconds and a braking time from maximum velocity to full stop also in 0.5 seconds) and second time with trapezoidal trajectory enabled. The angular acceleration and deceleration ($\varepsilon_{acc}$, $\varepsilon_{dec}$ ) were set double compared to the angular velocity in order to constrain the acceleration ($t_{acc}$) time to 0.5 sec. Experimental values are presented in the following chapter.

## 5. RESULTS

Following the experimental procedures described above, a set of experimental results were obtained. The test results seem to closely follow an ordered logarithmic pattern, as shown in Table 3 and Table 4.

In order to properly analyze the obtained experimental data, the results shown in the above tables were structured in diagram form, showing the evolution of the angular error (Δα) with respect to the number of motor spindle revolutions performed. The diagrams are showed in Fig. 8 – for the analysis of repeatability without trapezoidal trajectory – and Fig. 9 – for the analysis of repeatability with trapezoidal trajectory.

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

Moreover, by comparing the similarity of 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. These can be in the form of an improper

*Table 3*

**Repeatability without trapezoidal trajectory**

| **Rev** | **Error Δα [deg]** | | |
|---|---|---|---|
| | **Speed 20%** | **Speed 40%** | **Speed 60%** |
| 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 |

*Table 4*

**Repeatability with trapezoidal trajectory**

| **Rev** | **Error Δα [deg]** | | |
|---|---|---|---|
| | **Speed 20%** | **Speed 40%** | **Speed 60%** |
| 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 |

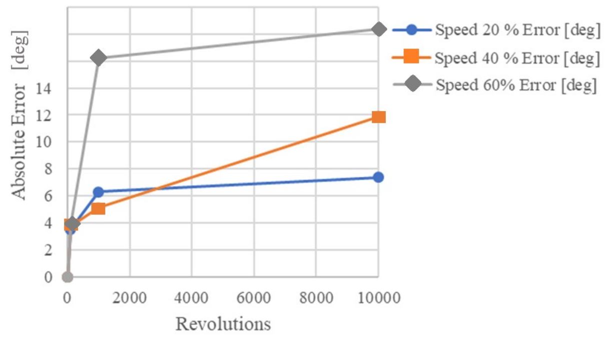

**Fig. 8.** Diagram showing the correspondence between the angular positioning error and the number of motor spindle revolutions (without trapezoidal trajectory).

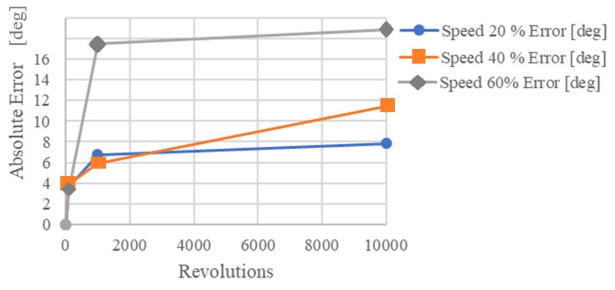

**Fig. 9.** Diagram showing the correspondence between the angular positioning error and the number of motor spindle revolutions (with trapezoidal trajectory).

field-oriented control (FOC) commutation [7] or 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. CONCLUSIONS

The first conclusion that can be drawn from the research presented in this paper is that PID control can also be achieved with brushed DC motors. This is an important aspect due to the fact that the brushed motors are affordable, can be integrated with simple and accessible controllers and have good behavior in extreme environments.

Because no motor calibration is performed, the configuration of the $K_p$, $K_i$ and $K_d$ parameters has to be done after a more thorough analysis. Also, the actual settings using empirically configured factors seem to generate good results only for fewer motor spindle rotations (around the value of 100 revolutions). It is clearly that the manual setup of the gain factors have to be more fine-tuned in order to obtain better precision values thus not affecting the promptitude of the system. measured angular errors (which can be seen in Tables 1 and 2, and also in Figs. 8 and 9) could be influenced by the fact that the configuration of the PID parameters was performed only for certain current and speed limits (40 000 counts per second).

The experimental results showed in the above tables were measured during the tests by reading the encoder values immediately after the motor spindle stopped spinning. For every measurement obtained in this way it could be observed that the PID function was still adjusting the position of the motor spindle several seconds after the rotation stopped (a slow adjustment rate), showing once again that the PID parameters should be further optimized for a quicker adjustment (the main objective being an adjustment that should be completely achieved immediately after the motor spindle stopped spinning).

The research project presented in this paper has good potential for further development. The next stage of this work will be focused on developing an algorithm that will provide the ability to automatically adjust the PID parameters for brushed DC motor control. This capability will have a significant impact in the research field, eliminating the requirement of PID parameter configuration.

A second objective for future research development will be the integration of force feedback options, which will allow the PID control to continuously adapt to work environment changes and also provide a research path toward using PID control in interactive systems. 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.

## 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] A. 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)`.