

7 API

In diesem Kapitel werden die Programmierschnittstellen der Libary erläutert. Dabei werden die verschiedenen Funktionen erklärt und die Mitgabe-Werte beschrieben.

Jedes Modul hat eine Init-Funktion, welche verwendet wird um ein Objekt zu erstellen und die Konfiguration zu erstellen. Der Rückgabewert dieser Funktion ist jeweils ein Handle für das erstellte Objekt und wird für alle anderen Funktionen des Moduls verwendet. Wenn es beim Ausführen der Funktionen einen Error gibt, wird NULL zurückgegeben. Das kann beispielsweise passieren, wenn kein Speicherplatz mehr vorhanden ist. Zudem gibt es immer eine Deinit-Funktion. Diese wird verwendet, um alle Timer zu löschen und die GPIO-Pins wieder freizugeben. Zudem wird der Speicherplatz des Objekts wieder freigegeben. Beim Aufruf dieser Funktion muss jeweils der Handle, des zu deinstallierendem Objekt mitgegeben werden. Der Rückgabewert dieser Funktion ist immer NULL.

7.1 DC-Motor

Alle Funktionen für die Ansteuerung von Gleichstrommotoren sind in der Datei dc_motor.h enthalten. Diese Funktionen werden hier beschrieben:

- `dcmotor_Handle_t dcmotor_init(uint8_t gpio_in1, uint8_t gpio_in2, uint8_t gpio_pwm, dcmotor_mode_t dc_mode)`

Diese Funktion wird verwendet, um ein DC-Motor Objekt zu erstellen. Beim Aufruf der Funktion müssen die Pins mitgegeben werden, an welche der Treiber des DC-Motors angeschlossen ist, zudem muss der Modus mitgegeben werden.

- `void dcmotor_change_speed(dcmotor_Handle_t dcmotor, uint16_t value)`

Diese Funktion setzt den Ausgabewert für den PWM-Pin und verändert dadurch die Drehgeschwindigkeit eines Motors. Beim Aufruf muss ein Handle für einen DC-Motor mitgegeben werden und den Wert auf, welcher das PWM gesetzt wird. Dabei ist ein Wert zwischen 0 und 65535 möglich. Die Drehrichtung wird beim Aufruf dieser Funktion beibehalten und es wird nur die Spannung für den Motor geändert.

- `void dcmotor_set(dcmotor_Handle_t dcmotor, dcmotor_direction_t dir, uint16_t value)`

Mit dieser Funktion kann die Drehrichtung und die Geschwindigkeit eines DC-Motors verändert werden. Neben dem Handle des DC-Motors, muss die Drehrichtung und ein Wert zwischen 0 und 65535 mitgegeben werden. Für die Drehrichtung kann ein Enum verwendet werden, welches die Richtungen Stopp, Vorwärts und Rückwärts beinhaltet.

- void dcmotor_set_value(dcmotor_Handle_t dcmotor, int32_t value)

Beim Aufruf dieser Funktion wird sowohl die Drehrichtung wie auch die Geschwindigkeit eines DC-Motors verändert. Dabei wird ein Handle eines DC-Motors und ein Wert mitgegeben. Dieser Wert darf zwischen -65535 und 65535 liegen. Hat der Wert ein positives Vorzeichen, wird der Motor in Uhrzeiger Richtung angetrieben, bei negativem Vorzeichen im die Gegen-Uhrzeiger Sinn.

- void dcmotor_stop(dcmotor_Handle_t dcmotor)

Mit dieser Funktion kann ein DC-Motor gestoppt werden. Dabei muss das Handle des jeweiligen Motors mitgegeben werden.

7.2 Schritt-Motor mit Timer

Alle Funktionen für die Ansteuerung von Schritt-Motoren mit Timer sind in der Datei stepper.h enthalten. Diese Funktionen werden hier beschrieben:

- stepper_Handle_t stepper_init(uint8_t gpio_A1, uint8_t gpio_B1, uint8_t gpio_A2, uint8_t gpio_B2, uint16_t steps_per_revolution, stepper_mode_t stepping_mode)

Diese Funktion wird verwendet, um ein Schritt Motor Objekt zu erstellen. Dabei müssen die vier Pins angegeben werden, an welche der Treiber des Schritt Motors angeschlossen ist. Zudem wird die Anzahl Schritte pro Umdrehung des Motors angegeben. Weiter muss der Modus der Ansteuerung definiert werden. Es ist möglich zwischen Single, Power und Half-Stepping zu wählen. Single ist dabei die einfachste Variante. Im Power Modus hat man mehr Drehmoment und bei Half hat man eine Doppelt so feine Auflösung.

- void stepper_set_speed_rpm(stepper_Handle_t stepper, float rpm)

Diese Funktion setzt die Geschwindigkeit für den Schritt Motor in rpm. Mitgegeben wird das Handle und den Geschwindigkeit Wert in rpm. Der mitgegebene Wert für die Geschwindigkeit muss dabei positiv sein. Durch diese Funktion wird der Motor aber nicht in Bewegung gesetzt, sie dient nur zur Festlegung der Geschwindigkeit für die kommenden Schritte. Es ist auch möglich diese Funktion bei drehendem Motor aufzurufen und die Geschwindigkeit laufend zu ändern.

- void stepper_accelerate(stepper_Handle_t stepper, float endspeed, int time_ms)

Diese Funktion wird verwendet, um ein Motor zu beschleunigen oder abzubremsen. Beim Aufruf wir der Handle und die Endgeschwindigkeit, sowie die Beschleunigungszeit angegeben. Danach wird die Geschwindigkeit des Stepper Motors in dieser Zeit bis zu Endgeschwindigkeit beschleunigt oder abgebremst. Auch diese Funktion bringt den Motor nicht zum Drehen, sondern definiert nur die Geschwindigkeit, in welcher die Schritte ausgeführt werden.

- `void stepper_step_once(stepper_Handle_t stepper)`

Diese Funktion wird verwendet um den Motor um einen Schritt zu drehen. Dabei muss nur der Handle des Schritt Motors angegeben werden.

- `void stepper_release(stepper_Handle_t stepper)`

Mit dieser Funktion kann der Schritt Motor stromfrei geschaltet werden. Mitgegeben wird beim Aufruf der Handle des Motors. Es ist wichtig die Funktion nach Gebrauch des Motors aufzurufen um Energie zu sparen und eine Überhitzung des Motors zu verhindern.

- `void stepper_rotate_steps(stepper_Handle_t stepper, int32_t steps)`

Nach dem Aufruf dieser Funktion dreht sich der Schritt Motor um die angegebene Anzahl Schritte. Mitgegeben wird der Handle und die Anzahl Schritte. Ist dieser Wert positiv dreht sich der Motor im Uhrzeigersinn. Ist der Wert negativ, dreht er sich im Gegenuhrzeigersinn. Die Schritte werden dabei mit der festgelegten Geschwindigkeit ausgeführt.

- `void stepper_rotate_degrees(stepper_Handle_t stepper, float degrees)`

Mit dieser Funktion kann der Schrittmotor um einen Winkel in Grad gedreht werden. Dabei wird der Handle und die Grad Anzahl angegeben. Auch hier ist ein negativer Wert ein Winkel im Gegenuhrzeigersinn. Es ist dabei möglich auch einen Winkel über 360 Grad anzugeben, und den Motor mehr als einmal ganz herum drehen zu lassen. Zu beachten ist dabei die Auflösung des Schritt Motors. Wenn dieser zum Beispiel 80 Schritte pro Umdrehung hat, ist mit einer Abweichung von bis zu 4.5 Grad beim Normalen Stepping Modus zu rechnen.

- `void stepper_reset_position(stepper_Handle_t stepper)`

Diese Funktion setzt die aktuelle Position auf null. Dabei wird nur der Handle des Schrittmotors, dessen Position geresetzt werden soll, mitgegeben. Die Position-Funktionen eines Schrittmotors werden verwendet, um die Position zu überwachen. Achtung, bei einem Schritt Verlust wird die Position auch falsch angezeigt.

- `int32_t stepper_get_position(stepper_Handle_t stepper)`

Mit dieser Funktion kann die Position eines Schritt Motors abgefragt werden. Dabei muss der Handle mitgegeben werden. Die Position wird in Schritte zum Nullpunkt zurückgegeben. Dabei werden die Schritte im Uhrzeigersinn hochgezählt. Wenn die Anzahl Schritte für eine Umdrehung erreicht wird, wird der Position Schritt Zähler wieder auf Null gesetzt.

- `int16_t stepper_get_position_degrees(stepper_Handle_t stepper)`

Diese Funktion liest die Position eines Schrittmotors aus und gibt sie als Winkel in Grad zurück. Dabei wird der Winkel zwischen 0 und 360 Grad ausgegeben. Null Grad ist dabei die Initialisierung Position oder die Position, an welcher ein reset ausgeführt wurde.

- `bool stepper_is_moving(stepper_Handle_t stepper)`

Mit dieser Funktion kann geprüft werden, ob sich der Stepper Motor aktuell dreht. Beim Aufruf muss der Handle mitgegeben werden und ein Bool wird zurückgegeben. Wenn sich der Motor momentan dreht, wird ein True zurückgegeben, ansonsten ein False.

7.3 Schritt-Motor mit PIO

Alle Funktionen für die Ansteuerung von Schritt-Motoren mit Timer sind in der Datei stepper.h enthalten. Diese Funktionen werden hier beschrieben:

- `pio_stepper_Handle_t pio_stepper_init(uint8_t gpio_A1, uint8_t gpio_B1, uint8_t gpio_A2, uint8_t gpio_B2, uint16_t steps_per_revolution, pio_stepper_mode_t stepping_mode)`

Mit dieser Funktion wird ein stepper-pio objekt erstellt und konfiguriert. Dieses wird verwendet, um einen Schrittmotor mit 3 PIO-Zustandsmaschinen anzusteuernd. Dabei müssen die vier GPIO-Nummern der Pins mitgegeben werden, an welcher der Motorentreiber angeschlossen ist. Dabei sind die beiden A-Pins und die beiden B-Pins jeweils ein Spulenpaar. Weiter muss die Anzahl Schritte angegeben werden, welche der Motor ausführen muss, um eine ganze Umdrehung zu erreichen. Zudem muss der Stepping-Modus angegeben werden. Dabei kann zwischen Single, Power und Half-Stepping gewählt werden. Diese Funktion kann nur einmal im Programm aufgerufen werden und es kann nur ein Schritt Motor mit PIO angesteuert werden. Da diese Methode drei Zustandsmaschinen verwendet und die anderen noch für Encoder verwendet werden. Als Rückgabewert erhält man ein Handle, welcher für die weiteren Funktionen verwendet wird und deklariert welches Objekt angesteuert werden soll.

- `void pio_stepper_set_speed_rpm(pio_stepper_Handle_t stepper, float rpm)`

Diese Funktion setzt die Geschwindigkeit für den Schritt Motor mit PIO-Steuerung. Mitgegeben wird das Handle und der Geschwindigkeit Wert in rpm. Der mitgegebene Wert für die Geschwindigkeit muss dabei positiv sein. Durch diese Funktion wird der Motor aber nicht in Bewegung gesetzt, sie dient nur zur Festlegung der Geschwindigkeit für die kommenden Schritte. Es ist auch möglich diese Funktion bei drehendem Motor aufzurufen und die Geschwindigkeit laufend zu ändern.

- `void pio stepper set mode(pio stepper Handle_t stepper,
pio stepper mode_t stepping mode)`

Mit dieser Funktion kann der Stepping Modus des PIO-gesteuerten Schrittmotor geändert werden. Dabei wird der Handle und der Modus mitgegeben. Es kann zwischen Single, Power und Half-Stepping gewählt werden. Diese Funktion kann auch bei drehendem Motor aufgerufen werden. Dabei ist aber darauf zu achten, dass beim Wechsel in oder vom Half-Stepping Modus die Auflösung ändert. Dadurch ist die zuvor angestrebte Drehung um einen gewissen Winkel nicht mehr korrekt. Denn es müssen beispielsweise doppelt so viele Halbschritte ausgeführt werden um den gleichen Winkel wie im Power-Stepping Modus mit normalen Schritten zu erreichen

- `void pio stepper set direction(pio stepper Handle_t stepper,
pio stepper direction_t direction)`

Diese Funktion ändert die Drehrichtung des Schritt Motors. Mitgegeben wird der Handle und die gewünschte Drehrichtung. Die Drehrichtung kann mit einem Enum Element angegeben werden und kann stop, clockwise oder counterclockwise sein. Mit dieser Funktion kann die Drehrichtung auch bei drehendem Motor gewechselt werden. Zudem ist es möglich den Motor mit der Richtung stop zu stoppen.

- `void pio stepper accelerate(pio stepper Handle_t stepper, float endspeed, int time_ms)`

Diese Funktion wird verwendet, um ein Schritt Motor mit PIO-Steuerung zu beschleunigen oder abzubremsen. Beim Aufruf wird der Handle und die Endgeschwindigkeit, sowie die Beschleunigungszeit angegeben. Danach wird die Geschwindigkeit des Stepper Motors in dieser Zeit bis zu Endgeschwindigkeit beschleunigt oder abgebremst. Auch diese Funktion bringt den Motor nicht zum Drehen, sondern definiert nur die Geschwindigkeit, in welcher die Schritte ausgeführt werden.

- `bool pio stepper step once(pio stepper Handle_t stepper)`

Diese Funktion führt einen Schritt in die zuvor definierte Richtung aus. Mitgegeben wird der Handle des PIO gesteuerten Schritt Motors. Wenn der Schritt ausgeführt wurde, wird True zurückgegeben. Wenn der Motor sich beim Aufruf jedoch schon dreht, kann der Schritt nicht einzeln getätigter werden und False wird zurückgegeben.

- `void pio stepper release(pio stepper Handle_t stepper)`

Mit dieser Funktion kann der Schritt Motor stromfrei geschaltet werden. Mitgegeben wird beim Aufruf der Handle des Motors. Es ist wichtig die Funktion nach Gebrauch des Motors aufzurufen um Energie zu sparen und eine Überhitzung des Motors zu verhindern.

- `bool pio stepper rotate steps(pio stepper Handle_t stepper, int32_t steps)`

Nach dem Aufruf dieser Funktion dreht sich der Schritt Motor mit der PIO-Steuerung um die angegebene Anzahl Schritte. Mitgegeben wird der Handle und die Anzahl Schritte. Ist dieser Wert positiv dreht sich der Motor im Uhrzeigersinn. Ist der Wert negativ, dreht er sich im Gegenuhrzeigersinn. Die Schritte werden dabei mit der festgelegten Geschwindigkeit ausgeführt.

- `void pio stepper rotate degrees(pio stepper Handle_t stepper, float degrees)`

Mit dieser Funktion kann der Schrittmotor mit PIO-Steuerung um einen Winkel in Grad gedreht werden. Dabei wird der Handle und die Anzahl Grad angegeben. Auch hier ist ein negativer Wert ein Winkel im Gegenuhrzeigersinn. Es ist dabei möglich auch einen Winkel über 360 Grad anzugeben, und den Motor mehr als einmal ganz herum drehen zu lassen. Zu beachten ist dabei die Auflösung des Schritt Motors. Wenn dieser zum Beispiel 80 Schritte pro Umdrehung hat, ist mit einer Abweichung von bis zu 4.5 Grad beim normalen Stepping Modus zu rechnen.

- `bool pio stepper is moving(pio stepper Handle_t stepper)`

Wenn beim Aufruf dieser Funktion der Schrittmotor in Bewegung ist, gibt diese Funktion ein True zurück. Wenn das nicht der Fall ist, wird ein False zurückgegeben. Beim Aufruf muss nur der Handle des Motors mitgegeben werden.

7.4 Quadratur Encoder

In diesem Abschnitt werden die verschiedenen Funktionen für einen Quadratur Encoder aufgelistet und erläutert. Diese können verwendet werden, um ein Encoder zu konfigurieren und auszulesen, um beispielsweise ein DC-Motor zu überwachen. All diese Funktionen sind in quadratur_encoder.h Datei definiert.

- `quadrature encoder Handle_t Quadratur_Encoder_Init(uint8_t gpio_pin1, int max_step_rate, uint16_t puls_per_rotation)`

Diese Funktion wird verwendet, um ein Quadratur Encoder zu initialisieren. Beim Aufruf der Funktion muss die GPIO-Nummer des ersten Pins des Encoders mitgegeben werden. Der zweite Pin sollte der anschließende Pin sein. Zudem wird `max_step_rate` mitgegeben. Diese wird verwendet, um die Abtastrate zu verkleinern. Denn oft wird für eine Anwendung nicht die maximale Rate von 12.5M Sample pro Sekunde verwendet. Dadurch kann Energie gespart werden. Wenn der Wert Null mitgegeben wird, wird die maximale Abtastrate verwendet.

- `int32_t Quadratur_Encoder_get_count(quadrature_encoder Handle_t encoder)`

Mit dieser Funktion kann die aktuelle Puls Anzahl ausgelesen werden. Beim Aufruf wird der Handle mitgegeben und die Anzahl Pulse wird zurückschreiben.

- `bool Quadratur_Encoder_start(quadrature_encoder_Handle_t encoder)`

Beim Aufruf dieser Funktion wird ein Timer gestartet, welcher die Puls Anzahl regelmässig abfragt und abspeichert. Dadurch ist es möglich die Geschwindigkeit danach abzufragen und sofort ein Ergebnis zu erhalten. Mitgegeben wird beim Aufruf der Handle des entsprechenden Encoders. Der Rückgabewert zeigt an, ob das Erstellen des Timers funktioniert hat. Bei False hat es einen Fehler gegeben.

- `bool void Quadratur_Encoder_stop(quadrature_encoder_Handle_t encoder)`

Mit dieser Funktion wird der Timer des Encoders gelöscht. Mitgegeben wird nur der Handle. Danach dauert die Abfrage der Geschwindigkeit wieder eine `Quadrature_Encoder_Periode` und kann nicht sofort ausgerechnet werden. Wenn dies aber nicht nötig ist, macht es Sinn die Timer Software nicht unnötig zu belasten.

- `float Quadratur_Encoder_get_speed(quadrature_encoder_Handle_t encoder)`

Zum Auslesen der Geschwindigkeit eines Quadratur Encoder kann diese Funktion verwendet werden. Dabei wird der Handle des Encoders mitgegeben und die Geschwindigkeit als float zurückgegeben. Ist beim Aufruf der Timer des Encoders eingeschaltet, wird die Geschwindigkeit sofort ausgerechnet und zurückgegeben. Ansonsten wird die Puls Anzahl ausgelesen, eine `Quadratur_Encoder_Periode` gewartet und danach die Puls Anzahl nochmals ausgelesen. Erst danach kann die Geschwindigkeit zurückgegeben werden. Die Periodendauer kann in der config-Datei definiert werden.

7.5 Taster/ Schalter

Die in diesem Abschnitt erläuterten Funktionen sind für die Auswertung von Digitalen Eingangs Pins. An diese können zum Beispiel Taster, Schalter, Lichtschranken oder Endschalter angeschlossen werden. Da die Funktionen und Anforderungen je nach Anwendung sehr unterschiedlich sind, werden in der `button.h`-Datei nur grundlegende Funktionen behandelt. Die in der C-Datei programmierte Implementation soll als Vorlage dienen, welche schnell mit den gewünschten Funktionen erweitert und angepasst werden kann.

- `button_Handle_t button_init(uint8_t gpio, uint32_t event, pull_t pull, bool debounce)`

Diese Funktion wird verwendet um ein `button_t` Objekt zu erstellen. Dabei wird die GPIO-Nummer des Pins mitgegeben, an welcher der Taster angeschlossen ist. Weiter ist es möglich die Events zu konfigurieren, bei welchen das Interrupt ausgelöst wird. Dabei ist es möglich auch mehrere Events zu wählen. Es gibt die vier Events Level Hoch, Level Tief ,und fallende und steigende Flanke. Die Events können mit dem Enum `gpio_irq_level` ausgewählt werden. Zudem kann gewählt werden, ob ein Pull up oder down Widerstand bei diesem GPIO initialisiert werden

soll. Dies kann mit dem enum pull_t gemacht werden, welches none, pull_up und pull_down enthält. Weiter kann mit einer Bool Variabel die Entprellung des Signals eingeschaltet werden. Wenn man True mitgibt, wird nach einem Interrupt von diesem Objekt eine Verzögerung aktiviert. Während dieser Zeit kann kein weiteres Interrupt von diesem Pin ausgelöst werden. Dadurch werden die Schwingungen beim Prellen ignoriert. Als Rückgabewert gibt diese Funktion ein Handle für einen Taster zurück, welcher für den Aufruf der anderen Funktionen verwendet wird.

- `bool button_get(button_Handle_t button)`

Mit dieser Funktion kann das aktuelle Level eines Tasters ausgelesen werden. Mitgegeben wird der Handle und zurückgegeben wird ein Bool. Dabei steht True für das Level hoch und False für ein tiefes Level.

7.6 ADC

Die in diesem Abschnitt erläuterten Funktionen werden verwendet, um einen analogen Eingang mittels ADC abzutasten.

- `adc_sensor_Handle_t my_adc_init(uint gpio)`

Mit dieser kann ein ADC-Objekt erzeugt werden. Dabei wird die GPIO-Nummer des Pins mitgegeben, an welchen das analoge Signal angeschlossen ist. Ein solches Signal kann beispielsweise von einem Druck-, Distanz- oder Temperatur-Sensor kommen. Dabei ist wichtig zu beachten, dass beim Raspberry Pi Pico nur die Pins 26 bis 28 an den ADC angeschlossen sind und nur diese Pins hier verwendet werden können. Diese Funktion schreibt ein Handle für das erstellte Objekt zurück. Gibt es bei der Ausführung einen Fehler, wird NULL zurückgegeben.

- `void my_adc_sampel(adc_sensor_Handle_t sensor)`

Beim Aufruf dieser Funktion wird die Abtastung des Eingangs Pins gestartet. Dabei muss der Handle zu einem ADC-Objekt mitgegeben werden. Je nach Konfiguration in der Config-Datei wird das Signal einmal oder gleich mehrmals abgetastet. Die erhaltenen Werte werden im Objekt abgespeichert und können nach der Abtastung mit der Funktion `my_adc_get` abgefragt werden.

- `uint16_t my_adc_sampelandread(adc_sensor_Handle_t sensor)`

Mit dieser Funktion kann ein analoges Signal abgetastet und ausgelesen werden. Beim Pico dauert eine solche Abtastung 2 Mikro Sekunden. Mitgegeben wird der Handle zu einem ADC-Objekt und der abgetastete Wert wird zurückgegeben.

- `uint16_t my_adc_get(adc_sensor_Handle_t sensor)`

Mit dieser Funktion kann der Wert der letzten Abtastung abgefragt werden. Dabei wird der Handle eines ADC-Objektes angegeben. Je nach Konfiguration in der Config-Datei wird dabei der Wert der letzten Abtastung oder der Mittelwert der erfolgten Abtastungen zurückgegeben.

7.7 PID-Geschwindigkeitsregler eines DC-Motors

In diesem Abschnitt werden die Funktionen für einen PID-Geschwindigkeitsregler für einen DC-Motor beschrieben. Dabei muss die Drehung des DC-Motors mit einem Quadratur Encoder überwacht werden und beide Objekte müssen initialisiert sein. Diese Funktionen sind in der PID_drive.h Datei enthalten.

- `pid_drive_Handle_t PID_drive_Init(dcmotor_Handle_t dcmotor,
quadrature_encoder_Handle_t encoder)`

Mit dieser Funktion wird ein PID-Regler für einen DC-Motor mit Encoder initialisiert. Beim Aufruf müssen die beiden initialisierten Handle des DC-Motors und des Quadratur Encoders mitgegeben werden. Der Rückgabewert ist ein Handle für den PID-Regler. Wenn einer der Handle nicht initialisiert ist oder zu wenig Speicherplatz vorhanden ist wird NULL zurückgegeben.

- `void PID_drive_SetParameters(pid_drive_Handle_t pid_drive,float Kp,
float Ki, float Kd, float N)`

Mit dieser Funktion können die Parameter des PID-Reglers gesetzt werden. Neben dem Handle müssen die einzelnen Parameter Kp, Ki, Kd und N mitgegeben werden. Die Übertragungsfunktion des Reglers bei der Verwendung dieser Funktion ist in der Formel 5 ersichtlich.

$$C(s) = Kp + \frac{Ki}{s} + \frac{Kd * s}{\frac{Kd}{N * Kp} * s + 1} \quad (5)$$

- `void PID_drive_SetParameters_ideal(pid_drive_Handle_t pid_drive,float Kp,
float Ti, float Td, float N)`

Mit dieser Funktion können die Parameter des PID-Reglers ebenfalls gesetzt werden. Neben dem Handle müssen die einzelnen Parameter Kp, Ti, Td und N mitgegeben werden. Die Übertragungsfunktion des Reglers bei der Verwendung dieser Funktion ist in der Formel 6 ersichtlich.

$$C(s) = Kp * (1 + \frac{1}{Ti * s} + \frac{Td * s}{\frac{Td}{N} * s + 1}) \quad (6)$$

- `void PID_drive_SetSpeed(pid_drive_Handle_t pid_drive,float speed)`

Mit dieser Funktion kann die Referenz Geschwindigkeit für den Regler vorgegeben werden. Beim Aufruf muss der Handle des Reglers und die gewünschte Geschwindigkeit in rpm mitgegeben werden. Nach dem Aufruf wird die Geschwindigkeit des DC-Motors auf diesen Wert geregelt.