

Sommario

1. Introduzione
 - Funzionalità Principali
 - Requisiti
2. Configurazione
 - Feature flags
 - Mappatura pin
 - Parametri costruttore
 - Istanze globali
3. Inizializzazione
4. API di movimento
 - Tipi di dati per i parametri
 - Movimento a velocità costante
 - Movimento con profili di accelerazione
 - Interruzione del movimento
 - Gestione del profilo S-Curve
 - Visualizzazione dell'S-Curve
5. API di configurazione e stato
 - Gestione della velocità minima
 - Metodi di supporto per il movimento
 - Gestione della posizione
 - Metodi di stato e query
6. Debug e diagnostica
 - Abilitazione del debug
 - Interpretazione dei log
7. Selezione della lingua
 - Come cambiare la lingua
 - La funzione msg()
8. Attenzione (hal_conf_extra.h)

1. Introduzione

Questo manuale d'uso descrive **StepperHAL**, libreria C++ progettata come strato di astrazione hardware (Hardware Abstraction Layer - HAL) per il controllo di motori stepper sulla famiglia di microcontrollori STM32F4x1. La libreria è stata sviluppata per semplificare la gestione del movimento, offrendo un'interfaccia intuitiva e configurabile che si integra con il framework Arduino e l'HAL di STMicroelectronics.

1.1 Funzionalità Principali

- **Controllo flessibile:** StepperHAL offre la possibilità di gestire il movimento del motore sia a velocità costante, sia con profili di accelerazione e decelerazione.
- **Profili di movimento:** La libreria supporta profili di movimento **trapezoidali** e a **S-curve** per movimenti fluidi e precisi. È possibile abilitare o disabilitare queste funzionalità modificando le impostazioni di configurazione.
- **Configurazione estesa:** La configurazione dei motori e la mappatura dei pin possono essere gestite centralmente, permettendo di definire i pin di controllo e i parametri motore come i passi per giro, i microstep e i millimetri per giro.
- **Debug e diagnostica:** È incluso un sistema di debug integrato che fornisce messaggi dettagliati sulla seriale per il monitoraggio e la risoluzione dei problemi.

1.2 Requisiti

La libreria StepperHAL richiede un ambiente di sviluppo e un hardware specifici per funzionare correttamente.

Requisiti Software

- **Framework Arduino per STM32:** La libreria si basa su questo framework per la gestione delle periferiche (Arduino_Core_STM32 (stm32duino)).
- **IDE (Ambiente di Sviluppo Integrato):** Si raccomanda Arduino IDE od altre che supportino il framework Arduino, come ad esempio Visual Studio Code con l'estensione PlatformIO.

Requisiti Hardware

- **Microcontrollore STM32F4x1:** L'architettura della libreria è ottimizzata per questa specifica famiglia di MCU, anche se potrebbe funzionare con piccole modifiche su altri microcontrollori STM32.
- **Driver per motori passo-passo:** È necessario un driver (es. A4988, DRV8825) per pilotare i motori, collegato ai pin del microcontrollore definiti in StepperHAL_Config.h.
- **Motori passo-passo:** Supporta motori NEMA17, NEMA23 o simili.
- **Scheda di sviluppo:** Una scheda basata su STM32F4x1, come la "Black Pill" o una Nucleo.

2. Configurazione

Il capitolo di configurazione di StepperHAL ti guida attraverso la personalizzazione della libreria per adattarla al tuo progetto. Tutte le opzioni di configurazione si trovano nel file **StepperHAL_Config.h**.

2.1 Feature flags

I "feature flags" ti consentono di attivare o disattivare funzionalità specifiche della libreria durante la compilazione. Questo aiuta a ottimizzare le dimensioni del codice e le prestazioni, includendo solo ciò che ti serve.

Puoi abilitare o disabilitare le funzionalità modificando le seguenti macro nel file StepperHAL_Config.h:

- **USE_S_CURVE**: Abilita i profili di movimento a S-curve, che offrono un'accelerazione e decelerazione più morbida. Per attivarla, imposta 1.
- **USE_TRAPEZOIDAL**: Abilita i profili di movimento trapezoidali, che sono più semplici e adatti a movimenti veloci. Per attivarla, imposta 1.

Nota: Se entrambi i profili di movimento sono disabilitati (0), la libreria abiliterà automaticamente il profilo trapezoidale come opzione di fallback.

2.2 Mappatura pin

Questa sezione del file StepperHAL_Config.h è fondamentale per definire le connessioni tra il microcontrollore e i driver dei motori.

Definisci i pin per ogni motore utilizzando le macro dedicate. Ad esempio:

```
#define PIN_DIR_MOTOR1 PA_9  
#define PIN_EN_MOTOR1 PA_8
```

- **PIN_DIR_MOTORx**: Il pin collegato all'ingresso DIR (direzione) del driver.
- **PIN_EN_MOTORx**: Il pin collegato all'ingresso EN (enable) del driver. Se non si usa un pin di abilitazione (il driver è sempre acceso), impostalo a -1. Può essere condiviso con altri pin EN.

È importante notare che il pin di STEP (`_stepPin`) non viene definito in questa sezione, ma è automaticamente determinato dalla libreria in base al timer e al canale scelti per ogni motore. Questo perché la generazione dell'onda quadra per il controllo del passo è gestita via hardware dal timer, che è mappato a un pin specifico del microcontrollore.

Timer	Channel	STEP PIN
TIM2	CH1	PA_15
TIM3	CH2	PB_5
TIM4	CH1	PB_6
TIM5	CH3	PA_2

Default timer, channel e pinSTEP; possono essere modificati ma non è consigliato.

2.3 Parametri costruttore

I parametri del costruttore, definiti anch'essi in `StepperHAL_Config.h`, consentono di configurare le proprietà fisiche e meccaniche di ciascun motore.

Il formato per ogni motore è il seguente:

```
#define MOTORx_PARAMS (timer, channel, dirPin, enPin, stepsPerRev, microstep, mmPerRev)
```

- **timer:** Il timer hardware STM32 da utilizzare (es. TIM2, TIM3).
- **channel:** Il canale del timer.
- **dirPin:** Il pin di direzione precedentemente definito.
- **enPin:** Il pin di abilitazione precedentemente definito.
- **stepsPerRev:** Il numero di passi fisici del motore per ogni giro completo (es. 200).
- **microstep:** Il valore di microstepping del driver (es. 1 per passo intero, 256 per 1/256 di passo).
- **mmPerRev:** I millimetri per rivoluzione; passo vite, circonferenza tamburo o puleggia. Se non si usa un sistema lineare, imposta questo valore su 1.0f.

Oppure, puoi usare il seguente formato alternativo:

```
#define MOTORx_PARAMS (timer, channel, dirPin, enPin, effectiveStepsMotorRev, mmPerRev)
```

- **effectiveStepsMotorRev:** Il valore di passi totali per rivoluzione assegnati dal driver (es. 1000).

2.4 Istanze globali

Per semplificare l'utilizzo e garantire una singola fonte di verità, la libreria definisce istanze globali degli oggetti StepperHAL per ogni motore. Non è necessario creare manualmente nuovi oggetti StepperHAL nel tuo codice principale (.ino).

Queste istanze sono dichiarate nel file `StepperHAL_Instances.h` e definite nel file `StepperHAL_Config.cpp`, utilizzando i parametri che hai impostato nel capitolo precedente (non vanno modificate).

- **Esempio di dichiarazione (`StepperHAL_Instances.h`):**

```
extern StepperHAL motor1;
extern StepperHAL motor2;
```

```
extern StepperHAL motor3;  
extern StepperHAL motor4;
```

- **Esempio di definizione (StepperHAL_Config.cpp):**

```
#include "StepperHAL_Config.h"  
#include "StepperHAL_Instances.h"  
#include "StepperHAL_STM32F4x1.h"
```

```
StepperHAL motor1(MOTOR1_PARAMS);  
StepperHAL motor2(MOTOR2_PARAMS);  
StepperHAL motor3(MOTOR3_PARAMS);  
StepperHAL motor4(MOTOR4_PARAMS);
```

Come usarle:

Nel tuo sketch Arduino, puoi semplicemente includere il file StepperHAL_Instances.h per accedere a queste istanze e chiamare i metodi della libreria. Non è richiesto alcun oggetto aggiuntivo.

Per esempio, per inizializzare il motore 1, il tuo codice sarà:

```
#include "StepperHAL_Instances.h"  
  
void setup() {  
  motor1.begin();  
  // ...  
}
```

Questo approccio centralizzato rende il codice più pulito e gestibile, evitando potenziali conflitti o duplicazioni di risorse.

3. Inizializzazione

Prima di poter usare i motori, è necessario inizializzare la libreria e configurare le periferiche hardware. Questo si fa con la funzione `begin()`.

Cosa fa la funzione `begin()`:

- **Configurazione dei pin:** Imposta i pin di direzione e di abilitazione come output.
- **Configurazione del timer:** Configura il timer hardware STM32 scelto in `StepperHAL_Config.h` per la generazione del segnale PWM necessario a pilotare il motore.
- **Abilitazione dei driver:** Abilita il driver del motore tramite il pin di abilitazione (se specificato).
- **Impostazione iniziale della direzione:** Imposta la direzione del motore su un valore predefinito.
- **Abilitazione degli interrupt:** Abilita l'interrupt del timer. Questa operazione è fondamentale, poiché l'avanzamento dei passi viene gestito all'interno dell'interrupt service routine (ISR) per garantire precisione e tempismo.

Se viene utilizzato il pin EN (enable), la funzione `motorX.setEnable()` deve essere chiamata nel `setup()` e impostata su `true`.

Come chiamare `begin()`:

La funzione `begin()` deve essere chiamata nel `setup()` unitamente a `setEnable()` del tuo sketch Arduino per ogni motore che vuoi utilizzare (se il driver motore usa un pin per EN).

```
#include "StepperHAL_Instances.h"
```

```
void setup() {  
    // Inizializza la comunicazione seriale per il debug  
    Serial.begin(115200);  
  
    // Inizializza tutti i motori che utilizzi nel tuo progetto  
    motor1.begin();  
    motor1.setEnable(true);  
    // ...  
}  
  
void loop() {  
    // Il tuo codice per controllare i motori va qui  
}
```

4. API di movimento

La libreria StepperHAL fornisce un insieme completo di funzioni per controllare i motori, accettando diverse unità di misura e gestendo sia movimenti a velocità costante che con profili di accelerazione.

4.1 Tipi di dati per i parametri

I metodi di movimento utilizzano tipi di dati dedicati per specificare l'unità di misura:

- `step`: Il numero di passi o microstep.
- `mm`: La distanza in millimetri.
- `Speed`: La velocità in giri al minuto (RPM).
- `Feed`: La velocità in mm/min.
- `mms2`: L'accelerazione o la decelerazione in mm/s².

Questi tipi sono definiti come struct per fornire un'unità di misura esplicita, rendendo il codice più leggibile. Ad esempio, per creare un valore di tipo `mm`, si usa `mm(10.0f)`.

4.2 Movimento a velocità costante

Questi metodi muovono il motore a una velocità fissa, senza utilizzare profili di accelerazione. Sono utili per movimenti semplici o per la calibrazione, dove non è richiesta un'accelerazione controllata.

Movimento assoluto (vai a una posizione)

- `void moveToPosition(mm distance, Speed rpm)`: Sposta il motore a una posizione assoluta in mm a una velocità in giri/min.
- `void moveToPosition(mm distance, Feed feedRate)`: Sposta il motore a una posizione assoluta in mm a una velocità di avanzamento in mm/min.
- `void moveToPosition(step steps, Speed rpm)`: Sposta il motore a una posizione assoluta in passi a una velocità in giri/min.
- `void moveToPosition(step steps, Feed feedRate)`: Sposta il motore a una posizione assoluta in passi a una velocità di avanzamento in mm/min.

Movimento relativo (muoviti da qui)

- `void moveRelative(mm delta, Speed rpm)`: Sposta il motore di una distanza relativa in mm a una velocità in giri/min.
- `void moveRelative(mm delta, Feed mmPerMin)`: Sposta il motore di una distanza relativa in mm a una velocità di avanzamento in mm/min.
- `void moveRelative(step delta, Speed rpm)`: Sposta il motore di un numero di passi relativi a una velocità in giri/min.
- `void moveRelative(step delta, Feed mmPerMin)`: Sposta il motore di un numero di passi relativi a una velocità di avanzamento in mm/min.

4.3 Movimento con profili di accelerazione

Questi metodi, disponibili se USE_TRAPEZOIDAL o USE_S_CURVE è abilitato, utilizzano profili di accelerazione e decelerazione per garantire movimenti fluidi, riducendo le vibrazioni e la perdita di passi.

Movimento assoluto (vai a una posizione)

- void moveToPositionWithAccel(mm distance, Speed rpm, mms2 accel): Sposta il motore a una posizione assoluta in mm con un'accelerazione costante fino a una velocità di crociera in giri/min.
- void moveToPositionWithAccel(mm distance, Feed feedRate, mms2 accel): Simile al precedente, ma usa la velocità di avanzamento in mm/min.
- void moveToPositionWithAccel(step steps, Speed rpm, mms2 accel): Sposta il motore a una posizione assoluta in passi con accelerazione e velocità in giri/min.
- void moveToPositionWithAccel(step steps, Feed feedRate, mms2 accel): Sposta il motore a una posizione assoluta in passi con accelerazione e velocità di avanzamento.

Varianti con accelerazione e decelerazione separate

Questi metodi offrono un controllo più preciso, permettendo di impostare valori diversi per l'accelerazione e la decelerazione. Sono ideali per applicazioni che richiedono profili di movimento asimmetrici, come quando la coppia del motore cambia in base al carico o alla direzione.

- void moveToPositionWithAccel(mm distance, Speed rpm, mms2 accel, mms2 decel): Sposta il motore a una posizione assoluta in mm con un profilo di movimento che usa accelerazione (accel) e decelerazione (decel) separate, fino a una velocità di crociera in giri/min.
- void moveToPositionWithAccel(mm distance, Feed feedRate, mms2 accel, mms2 decel): Simile al precedente, ma usa una velocità di avanzamento in mm/min per la fase di crociera.
- void moveToPositionWithAccel(step steps, Speed rpm, mms2 accel, mms2 decel): Sposta il motore a una posizione assoluta in passi con profili di accelerazione e decelerazione separati, fino a una velocità in giri/min.
- void moveToPositionWithAccel(step steps, Feed feedRate, mms2 accel, mms2 decel): Sposta il motore a una posizione assoluta in passi con profili di accelerazione e decelerazione separati, fino a una velocità di avanzamento.

Movimento relativo (muoviti da qui) con accelerazione

Questi metodi spostano il motore di un numero di passi o di una distanza specificata rispetto alla posizione corrente, utilizzando i profili di accelerazione e decelerazione. Sono ideali per compiere spostamenti precisi e fluidi da un punto all'altro.

Varianti con accelerazione singola

Questi metodi utilizzano lo stesso valore di accelerazione per la fase di rampa in salita e per la decelerazione.

- `void moveRelativeWithAccel(mm delta, Speed rpm, mms2 accel)`: Sposta il motore di una distanza relativa in mm a una velocità in giri/min, usando il profilo di accelerazione specificato.
- `void moveRelativeWithAccel(mm delta, Feed feedRate, mms2 accel)`: Sposta il motore di una distanza relativa in mm a una velocità di avanzamento in mm/min.
- `void moveRelativeWithAccel(step delta, Speed rpm, mms2 accel)`: Sposta il motore di un numero di passi relativi a una velocità in giri/min.
- `void moveRelativeWithAccel(step delta, Feed feedRate, mms2 accel)`: Sposta il motore di un numero di passi relativi a una velocità di avanzamento in mm/min.

Varianti con accelerazione e decelerazione separate

Questi metodi permettono di impostare valori diversi per l'accelerazione e la decelerazione.

- `void moveRelativeWithAccel(mm delta, Speed rpm, mms2 accel, mms2 decel)`: Sposta il motore di una distanza relativa in mm con profili di accelerazione e decelerazione separati, fino a una velocità in giri/min.
- `void moveRelativeWithAccel(mm delta, Feed feedRate, mms2 accel, mms2 decel)`: Sposta il motore di una distanza relativa in mm con profili di accelerazione e decelerazione separati, fino a una velocità di avanzamento in mm/min.
- `void moveRelativeWithAccel(step delta, Speed rpm, mms2 accel, mms2 decel)`: Sposta il motore di un numero di passi relativi con profili di accelerazione e decelerazione separati.
- `void moveRelativeWithAccel(step delta, Feed feedRate, mms2 accel, mms2 decel)`: Sposta il motore di un numero di passi relativi con profili di accelerazione e decelerazione separati, fino a una velocità di avanzamento.

4.4 Interruzione del movimento

- `void stop()`: Questo comando arresta immediatamente il motore, interrompendo qualsiasi movimento in corso. A differenza di un'operazione che attende il raggiungimento di una posizione, `stop()` forza un'interruzione istantanea del movimento, il che può essere utile.

4.5 Gestione del profilo S-Curve

Questi metodi sono disponibili solo se il profilo `USE_S_CURVE` è abilitato. Permettono di personalizzare la curva di accelerazione "a S" che controlla la variazione di accelerazione e decelerazione per ottenere movimenti più fluidi, riducendo vibrazioni e shock meccanici.

Cos'è il Jerk?

Il Jerk (scatto) è la variazione dell'accelerazione nel tempo. In un profilo di movimento standard (trapezoidale), l'accelerazione cambia istantaneamente da zero a un valore costante, creando

una "scossa" o uno "strappo" all'inizio e alla fine del movimento. Il profilo a S (S-Curve) elimina questo brusco cambiamento, gestendo l'accelerazione in modo graduale. Un jerk elevato corrisponde a un cambio di accelerazione più repentino, mentre un jerk basso garantisce un movimento più morbido e progressivo.

- `void useSCurve(bool enable)`: Abilita o disabilita l'uso del profilo S-curve.
- `void useDefaultJerk(bool enable)`: Utilizza i modelli di calcolo predefiniti per automatizzare la gestione del jerk, creando un profilo S-curve bilanciato, abilitato per default.
- `void setJerkRatio(float ratio)`: Imposta un rapporto che modifica il bilanciamento tra l'accelerazione e la decelerazione quando si utilizzano i modelli di calcolo predefiniti. Ha effetti solo quando si utilizza il metodo `useDefaultJerk(true)`.
- `void setManualJerk(...)`: Questo metodo consente una personalizzazione avanzata del jerk. Offre un controllo preciso su ciascuna delle sei fasi del profilo S-curve, permettendo di regolare l'andamento dell'accelerazione e della decelerazione. I sei parametri corrispondono alle seguenti fasi del movimento:
 - `jerkAccelUp`: L'inizio della rampa di accelerazione, quando il motore comincia a salire in velocità.
 - `jerkAccelFlat`: La parte centrale della fase di accelerazione, dove l'accelerazione è costante.
 - `jerkAccelDown`: La fine della rampa di accelerazione, quando il motore si avvicina alla velocità di crociera.
 - `jerkDecelUp`: L'inizio della rampa di decelerazione, quando la velocità inizia a diminuire.
 - `jerkDecelFlat`: La parte centrale della fase di decelerazione.
 - `jerkDecelDown`: La fine della rampa di decelerazione, quando il motore si ferma completamente.

Si consiglia di partire da una configurazione automatica (`useDefaultJerk(true)`) per comprendere il comportamento del motore. Successivamente, è possibile abilitare il debug (`StepperHAL_DEBUG_ENABLE` a `true`) per visualizzare i valori di jerk nel log, per poi variare gradualmente i singoli parametri tramite `setManualJerk` ed osservare come le modifiche influenzano il movimento.



4.6 Visualizzazione dell'S-Curve

Il profilo di movimento a S-Curve si distingue per la sua capacità di rendere il movimento più fluido grazie alla gestione graduale dell'accelerazione e della decelerazione. Questo profilo è diviso in sette fasi, di cui sei sono le fasi di **jerk**.

Come si può notare dal grafico:

Jerk Accelerazione (fase 1-3): La velocità aumenta in modo controllato. Il "jerk" (la variazione dell'accelerazione) è graduale:

- *jerkAccelUp*: L'accelerazione aumenta da zero al suo valore massimo.
- *jerkAccelFlat*: L'accelerazione rimane costante.
- *jerkAccelDown*: L'accelerazione diminuisce fino a zero.

Velocità di Crociera (fase 4): Il motore si muove a velocità costante senza accelerazione o decelerazione.

Jerk Decelerazione (fase 5-7): La velocità diminuisce in modo controllato. Anche qui, il jerk è graduale:

- *jerkDecelUp*: La decelerazione aumenta da zero al suo valore massimo.
- *jerkDecelFlat*: La decelerazione rimane costante.
- *jerkDecelDown*: La decelerazione diminuisce fino a zero, portando il motore a fermarsi dolcemente.

5. API di configurazione e stato

Questi metodi permettono di configurare il motore e di verificarne lo stato in tempo reale.

5.1 Gestione della velocità minima

Questi metodi permettono di definire e recuperare la velocità minima alla quale il motore può muoversi. Impostare una **velocità minima** è cruciale per evitare che il motore si blocchi (stalli) a frequenze troppo basse.

- void setMinFreqHz(float freqHz): Imposta la frequenza minima di step direttamente in Hertz. È utilizzato da setMinRPM() per convertire il valore in RPM e assicurarsi che la frequenza non scenda mai al di sotto di un valore minimo, prevenendo stalli a velocità vicine allo zero. I valori di default sono in StepperHAL_Config.h.
- void setMinRPM(float rpm): Questo metodo imposta la velocità minima del motore in **giri al minuto (RPM)**. La libreria calcola internamente la frequenza equivalente in Hertz (Hz) e la memorizza. Durante qualsiasi movimento, la velocità del motore non scenderà mai al di sotto di questo valore, anche se la velocità di crociera impostata è inferiore.
- float getMinFreqHz() const: Restituisce la **frequenza minima** di step in Hertz (passi al secondo) che è stata calcolata e impostata internamente tramite setMinRPM(). È un metodo utile per verificare il valore della frequenza minima.

5.2 Metodi di supporto per il movimento

Questi metodi non avviano un movimento, ma permettono di gestirne il comportamento, la compensazione o l'interruzione.

- void invertDIR(bool state); Inverte la direzione del motore.
- void setTiming(uint32_t stepPulseUs, uint32_t timerBaseFreq); Inizializza la struttura interna _timing con la durata del segnale STEP e la frequenza del clock timer, quindi calcola i parametri reali del timer (prescaler, auto-reload, contatori) necessari per generare impulsi di passo precisi.
 - stepPulseUs (uint32_t) • Durata minima di ciascun impulso STEP in microsecondi (μs).
 - timerBaseFreq (uint32_t) • Frequenza in Hertz (Hz) del clock del timer periferico usato per generare STEP.

Chiamare subito dopo begin(). Deve essere eseguito prima di qualsiasi comando di movimento (moveTo...) per garantire che i valori di temporizzazione del timer siano corretti.

- void backlashCompensation(step value): Imposta un valore di gioco (backlash) in passi.
- void backlashCompensation(mm value): Imposta un valore di gioco (backlash) in millimetri.

Il **backlash** è il gioco meccanico che si verifica durante l'inversione della direzione del motore. Questo valore viene utilizzato per compensare automaticamente il movimento per garantire un

posizionamento preciso. Per gestire il backlash, la libreria **StepperHAL** mantiene due posizioni interne: una **posizione logica** e una **posizione reale**.

- La **posizione logica** è la posizione ideale che il motore dovrebbe avere.
- La **posizione reale** è la posizione fisica del motore, che può differire dalla posizione logica.

Se non viene impostata alcuna compensazione (`backlashCompensation(0)`), le due posizioni coincidono. Quando invece imposti un valore di backlash (in mm o passi), la libreria tiene conto automaticamente di questo gioco. La posizione reale differirà dalla posizione logica per i passi necessari alla compensazione. Ogni volta che il motore inverte la sua direzione, StepperHAL applicherà la compensazione. Questo avviene in modo automatico, senza che tu debba intervenire ulteriormente nel codice. Lo scostamento tra posizione reale e logica avviene sempre e solamente dopo un movimento nella direzione opposta alla prima direzione utilizzata (es. se il primo movimento è CW gli scostamenti saranno dopo un movimento CCW e viceversa).

5.3 Gestione della posizione

Questi metodi permettono di definire o reimpostare la posizione del motore, un'operazione fondamentale per la calibrazione e il corretto funzionamento. La loro differenza principale risiede nel modo in cui influenzano lo stato interno della compensazione del backlash.

- `void resetPosition()`: Reimposta la posizione corrente del motore a zero. È utile per stabilire un nuovo punto di riferimento (home) senza dover muovere fisicamente il motore. Questo metodo **resetta anche lo stato interno del backlash**. Ciò significa che, dopo la chiamata a questa funzione, la posizione logica e la posizione reale del motore torneranno a coincidere, ed il meccanismo di compensazione del backlash ripartirà da zero.
- `void setPositionSteps(int32_t steps)`: Imposta la posizione corrente del motore a un valore specifico in passi.
- `void setPositionMM(float mm)`: Imposta la posizione corrente del motore a un valore specifico in millimetri.

Questi metodi sono utili per le calibrazioni, ad esempio per settare la posizione iniziale a un valore diverso da zero. A differenza di `resetPosition()`, questi metodi **non alterano lo stato del backlash**. Se è già stata applicata una compensazione, questa rimarrà attiva, assicurando che la posizione reale continui a tenere conto del gioco meccanico e dell'ultima direzione di movimento.

5.4 Metodi di stato e query

Questa sezione descrive i metodi che ti consentono di interrogare lo stato attuale del motore e recuperare informazioni per il controllo del movimento.

- `bool isActive()`: Restituisce `true` se il motore è in movimento, `false` altrimenti. Questo metodo è utile per controllare lo stato del motore, ad esempio prima di avviare un nuovo movimento o per attendere che il movimento corrente sia completato.
- `bool hasBacklash()`: Restituisce `true` se è stato impostato un valore di gioco (backlash) tramite i metodi `backlashCompensation()`. È un semplice metodo di verifica per sapere se la compensazione del gioco è attiva o meno.
- `int32_t getPositionSteps()`: Restituisce la posizione corrente in passi. Questo valore corrisponde alla **posizione logica**, che è la posizione ideale del motore, non ancora corretta per il gioco meccanico.
- `float getPositionMM()`: Restituisce la posizione corrente in millimetri. Anche questo valore corrisponde alla **posizione logica**, convertita in millimetri. È un modo per leggere la posizione del motore in un'unità di misura più intuitiva.
- `uint32_t getStepsPerRev()`: Restituisce il numero di passi fisici per ogni rivoluzione del motore. Questo valore è un parametro di base che viene impostato nel costruttore e non cambia durante il funzionamento.
- `bool targetReached() const`: Questo metodo verifica se il motore ha raggiunto il punto finale del movimento desiderato. Restituisce `true` se la posizione corrente coincide con la posizione di destinazione effettiva e `false` in caso contrario. È un metodo fondamentale da utilizzare in un ciclo di attesa (loop) per sapere quando un movimento è terminato. La posizione di destinazione effettiva tiene già conto delle compensazioni interne, come quella del backlash, garantendo una verifica precisa.

6. Debug e diagnostica

La libreria StepperHAL include un sistema di debug integrato che fornisce informazioni dettagliate in tempo reale sul comportamento del motore. È uno strumento essenziale per la calibrazione, l'ottimizzazione e la risoluzione dei problemi, come lo stallo o i movimenti imprecisi.

6.1 Abilitazione del debug

Il sistema di debug è controllato dalla direttiva `#define StepperHAL_DEBUG_ENABLE` che si trova nel file `StepperHAL_Debug.h`.

- Per **abilitare il debug**, assicurati che la riga sia impostata su `true`:
`#define StepperHAL_DEBUG_ENABLE true`
- Per **disabilitare il debug** e ottimizzare le prestazioni, imposta il valore su `false`. Quando il debug è disabilitato, il codice non genera alcun output e non ci sono overhead inutili.

6.2 Interpretazione dei log

Dopo aver abilitato il debug, le informazioni vengono stampate sul terminale seriale. A seconda del profilo di movimento utilizzato (S-curve o trapezoidale), il log fornirà dettagli specifici.

Esempio di log (profilo S-Curve)

Questo log mostra il dettaglio di un movimento con accelerazione a S.

[StepperHAL] (motor2) Jerk mode: default

[StepperHAL] (motor2) Movimento iniziato da posizione logica=0 : reale=0 / target=6000 / step necessari=6000 / Dir=CW

[StepperHAL] (motor2) Jerk default — aUp=23.2 aFlat=0.0 aDown=23.2 dUp=23.2 dFlat=0.0 dDown=23.2 JerkRatio=1.00

[StepperHAL] (motor2) S-curve fasi accelerazione (Up/Flat/Down)= 592/594/592

[StepperHAL] (motor2) S-curve fasi decelerazione (Up/Flat/Down)= 592/594/592

[StepperHAL] (motor2) Profilo DMA separato -> 6000 steps, accel = 100.00 mm/s², decel = 100.00 mm/s²

[StepperHAL] (motor2) -> Accelerazione: 1778

[StepperHAL] (motor2) -> Regime: 2444

[StepperHAL] (motor2) -> Decelerazione: 1778

[StepperHAL] (motor2) -> Profilo: S-curve

[StepperHAL] (motor2) DMA separato → Target richiesto=6000 Vmax 800.00 RPM@ 13333.33 Hz

[StepperHAL] (motor2) Movimento completato: fermo in posizione logica=6000: reale=6000 / target=6000 / Dir=CW

- **Jerk mode: default:** La libreria sta usando i valori di jerk predefiniti.
- **Jerk default — ...:** Vengono visualizzati i valori di jerk calcolati per le sei fasi del profilo a S.
- **S-curve fasi accelerazione/decelerazione:** Indicano il numero di passi in ciascuna delle tre sotto-fasi (Up, Flat, Down) della rampa di accelerazione e decelerazione.
- **Accelerazione / Regime / Decelerazione:** Indicano il numero totale di passi dedicati a ciascuna fase del movimento (rampa di accelerazione, velocità di crociera costante, rampa di decelerazione). La somma di questi valori dà il numero totale di passi.
- **Vmax 800.00 RPM@ 13333.33 Hz:** Riassume la velocità massima raggiunta durante il movimento, sia in giri al minuto che in frequenza di step.

Esempio di log (profilo trapezoidale)

Questo log mostra un movimento con un profilo trapezoidale, che è più semplice e non include le fasi di jerk.

[StepperHAL] (motor2) Movimento iniziato da posizione logica=10000 : reale=10000 / target=0 / step necessari=10000 / Dir=CCW

[StepperHAL] (motor2) Profilo DMA separato → 10000 steps, accel = 100.00 mm/s², decel = 100.00 mm/s²

[StepperHAL] (motor2) -> Accelerazione: 1000 steps

[StepperHAL] (motor2) -> Regime: 8000 steps

[StepperHAL] (motor2) -> Decelerazione: 1000 steps

[StepperHAL] (motor2) -> Profilo: trapezoidale

[StepperHAL] (motor2) DMA separato → Target richiesto=0 Vmax 600.00 RPM@ 10000.00 Hz

[StepperHAL] (motor2) Movimento completato: fermo in posizione logica=0: reale=0 / target=0 / Dir=CCW

- **Accelerazione / Regime / Decelerazione:** Anche qui, vengono indicati il numero di passi per ciascuna fase del movimento. La somma di questi valori ti permette di verificare la correttezza del calcolo del profilo.
- **Vmax 600.00 RPM@ 10000.00 Hz:** Riassume la velocità massima raggiunta.

Il sistema di debug è uno strumento potente per la calibrazione, specialmente quando si ottimizzano i profili di movimento per ridurre al minimo le vibrazioni e le perdite di passi.

7. Selezione della lingua

La libreria StepperHAL offre la possibilità di visualizzare i messaggi di debug in diverse lingue, un'opzione utile per una migliore leggibilità in base alle preferenze dell'utente. Questa funzionalità è gestita da una semplice direttiva nel file **StepperHAL_Lang.h**.

7.1 Come cambiare la lingua

Per cambiare la lingua dei messaggi di debug, basta modificare la seguente riga nel file StepperHAL_Lang.h:

```
#define LANGUAGE_IT // Commenta per passare all'inglese / Comment for English
```

- Per abilitare la lingua **italiana**, assicurati che la riga `#define LANGUAGE_IT` non sia commentata.
- Per passare alla lingua **inglese**, commenta la riga aggiungendo `//` all'inizio:
`// #define LANGUAGE_IT // Commenta per passare all'inglese / Comment for English`

Questo approccio a tempo di compilazione assicura che il codice sia leggero e che l'impatto sulle performance sia minimo, poiché i messaggi vengono inclusi solo nella lingua selezionata.

7.2 La funzione msg()

La logica di localizzazione è gestita internamente dalla funzione `msg()`, che prende due stringhe come parametri: una per la versione in italiano e una per la versione in inglese.

```
inline String msg(const String& it, const String& en) {  
#ifdef LANGUAGE_IT  
    (void)en; // evita warning unused parameter quando LANGUAGE_IT è definito  
    return it;  
#else  
    (void)it; // evita warning unused parameter quando LANGUAGE_IT non è definito  
    return en;  
#endif  
}
```

Questa funzione è utilizzata ovunque nella libreria per stampare i messaggi di debug. Ad esempio, nel file `StepperHAL_STM32F4x1.cpp` troverai chiamate simili a questa:

```
SerialDB.print(msg("Movimento completato: fermo...", "Movement completed: stopped..."));
```

In base al valore di LANGUAGE_IT, la funzione msg() restituirà la stringa in italiano o in inglese. Questo garantisce che tutti i messaggi di output siano coerenti con la lingua scelta, senza che tu debba modificare il codice della libreria.

8. Attenzione (hal_conf_extra.h)

Il file `hal_conf_extra.h` è necessario per abilitare o disabilitare moduli HAL in modo esplicito, senza modificare i file interni del core STM32 Arduino. Nel caso di StepperHAL, è fondamentale per disabilitare il modulo `HAL_TIM` del core, altrimenti si verifica un conflitto con le ISR (`TIMx_IRQHandler`) già definite da StepperHAL.

Infatti, il core STM32 Arduino include un file chiamato `HardwareTimer.cpp` che definisce le stesse ISR per i timer hardware (`TIM2`, `TIM3`, `TIM4`, ecc.). Se entrambe le definizioni sono presenti, il linker genera un errore di "multiple definition".

Per evitare questo conflitto, nella directory dove installata la libreria c'è `hal_conf_extra.h` che contiene:

```
cpp
#define                                     HAL_TIM_MODULE_ONLY
```

questa macro disattiva il modulo `HAL_TIM` del core, lasciando attive solo le definizioni di StepperHAL.