JDR DC Motor Controller

Knihovna pro řízení stejnosměrných (DC) komutátorových motorů.

Základní vlastnosti:

Knihovna umožňuje obousměrné řízení rychlosti otáčení stejnosměrného komutátorového motoru, plynulé zvyšování a snižování otáček (ramping) a umožňuje nastavit pásmo necitlivosti k řídícímu signálu v okolí nulové rychlosti.

Pro připojení motoru k Arduinu je kromě motoru nezbytný také odpovídající driver; nikdy nepřipojujte motor přímo k Arduinu!

Konstruktor

Třída MotorController podporuje dva různé způsoby řízení motoru. Jejich volba se řídí použitím patřičného konstruktoru se dvěma nebo třemi parametry.
Konstruktor se automaticky zavolá při deklaraci proměnné; vytvoří jednu instanci třídy MotorController, aktivuje piny pro připojení ovladače motoru a nastaví způsob jeho řízení.

MotorController() [1/2]

Řízení s odděleným vstupem PWM.
*Tento způsob se používá v případě, že směr otáčení motoru je ovládán logickou úrovní na pinech **forwardsPin** a **backwardsPin**[1] a rychlost otáčení je dána činitelem plnění PWM na pinu **pwmPin**. Typickým příkladem tohoto způsobu řízení je dvojitý H-můstek TB6612, ale je ho možno použít i u můstků L293, L298, SN754410 a dalších.*

Syntaxe:

```
MotorController motor1(forwardsPin, backwardsPin, pwmPin);
```

[1] *0, 1 pro jeden směr a 1, 0 pro opačný směr*

Parametry:

forwardsPin (*uint8_t*) – číslo digitálního pinu
backwardsPin ((*uint8_t*) – číslo digitálního pinu
pwmPin ((*uint8_t*) – číslo digitálního pinu s funkcí analogWrite()

Návratová hodnota:

Funkce nic nevrací.

Poznámka:

Motor je po inicializaci aktivován, ale zastaven (nastavena rychlost 0).

MotorController() [2/2]

Řízení směr + rychlost.
*Tento způsob se používá v případě, že směr i rychlost otáčení motoru jsou ovládány jen piny **forwardsPin** a **backwardsPin**. Pro jeden směr otáčení je na na pinu **pin1** generována PWM modulace, která určuje rychlost otáčení motoru a pin **pin2** je nastaven na LOW (log. 0), pro opačný směr otáčení je PWM modulace generována na pinu **pin2** a pin **pin1** je nastaven na LOW (log. 0)[2]. Tento způsob řízení je nutno použít u H-můstku L9110, ale je možno ho použít i u můstků L293, L298, SN754410 a dalších.*

Syntaxe:

```
MotorController motor1(pin1, pin2);
```

Parametry:

pin1 ((*uint8_t*) – číslo digitálního pinu s funkcí analogWrite()
pin2 ((*uint8_t*) – číslo digitálního pinu s funkcí analogWrite()

Návratová hodnota:

Funkce nic nevrací.

Poznámka:

Motor je po inicializaci aktivován, ale zastaven (nastavena rychlost 0).

[2] *Lze to říci i takto: pro jeden směr se tiká na pinu 1 a pin2 je v úrovni L, pro druhý směr je pin1 v úrovni L a tiká se pinem 2.*

Členské funkce (metody)

Výkonné metody

set()

Nastaví rychlost a směr otáčení motoru.

Syntaxe:

```
set(powerPercent);
```

Parametry:

powerPercent (*int*) – číslo v rozsahu -100 až 100, přičemž 100 znamená otáčení motoru plnou rychlostí jedním směrem, 0 zastavení motoru a -100 otáčení motoru plnou rychlostí opačným směrem.

Návratová hodnota:

Aktuální rychlost motoru v procentech (-100 až 100).

Poznámka:

Pokud je rampování zapnuté, je třeba po nastavení požadované cílové rychlosti touto funkcí opakovaně volat funkci `run()`*, která postupně rychlost motoru upravuje v souladu s nastaveným zrychlením a zpomalením (viz funkce* `setRampRate()`*).*

Pokud je rampování vypnuté, tato funkce nastaví rychlost motoru okamžitě na požadovanou hodnotu a funkci `run()` *není potřeba následně vůbec volat.*

Příklad:

```
motor1.set(50);
```

run()

Výkonná funkce, která upraví řízení motoru podle aktuální potřeby s ohledem na nastavené zrychlení resp. zpomalení. V případě zapnutého rampování je třeba funkci volat dostatečně často, aby motor měnil rychlost plynule[3]. Je-li rampování vypnuté nebo pokud motor již dosáhl požadované cílové rychlosti, není třeba tuto funkci dále volat.

Syntaxe:

```
run();
```

Parametry:

Funkce nemá žádné parametry.

Návratová hodnota:

Aktuální rychlost motoru v procentech (-100 až 100).

Poznámka:

Pokud je zapnuté rampování, funkce vrací hodnotu tak, že při zavolání okamžitě po funkci `set()` *bude hodnota blízká rychlosti před zavoláním funkce* `set()` *(tj. rychlost, ze které se začíná). Při dalších voláních funkce run bude vracet postupně se měnící rychlost směrem k požadované cílové rychlosti.*

Příklad:

```
int aktualniRychlost;
motor1.disableRamping();
motor1.set(100);
delay(2000);
motor1.setRampRate(25);
motor1.enableRamping();
motor1.set(0);
do
{
  aktualniRychlost = motor1.run();
}
while(aktualniRychlost != 0);
Serial.println("Stojíme.");
```

[3] *Co je „dostatečně často“ záleží na aplikaci; obecně se dá říci, že pokud má být rampování rychlé, je třeba tuto funkci volat velmi často a pokud je rampování pomalé, stačí méně často. Každopádně pokud motor při rampování viditelně „skáče“ (tj. mění rychlost skokově místo plynule), pak je funkce run() volána málo často.*

V tomto příkladu nejprve vypneme rampování a nastavíme rychlost motoru na 100%. Po časové prodlevě 2 sekund na roztočení motoru[4] nastavíme rampování na 25%, rampování povolíme a nastavíme rychlost na 0, aby motor během 4 sekund zastavil. K tomu musíme opakovaně volat funkci `run()`, která rychlost vždy upraví podle potřeby. Tím motor postupně zastavíme a teprve potom budeme pokračovat dál. To zajistíme tak, že budeme kontrolovat, jakou aktuální rychlost funkce `run()` právě vrací – jako podmínku pro běh cyklu uvedeme, že cyklus má běžet, dokud funkce `run()` vrací číslo různé od nuly. Jakmile se motor zastaví, tj. rychlost klesne na 0, funkce `run()` vrátí 0, takže podmínka již nebude splněna a program bude pokračovat dále (v tomto případě kontrolním výpisem pro uživatele).

Nastavení parametrů

setRampRate()

Nastaví strmost rozběhové / doběhové rampy.
Rampování je ve výchozím stavu vypnuto.

Syntaxe:

```
setRampRate(rampRate);
```

Parametry:

rampRate (*uint8_t*) – kladné celé číslo v rozsahu 1 až 100, představující procento (%) přírůstku rychlosti za jednu sekundu (například hodnota 100 znamená dosažení nastavené rychlosti za jednu sekundu, hodnota 50 dosažení nastavené rychlosti za dvě sekundy atd.).

Návratová hodnota:

Funkce nic nevrací.

Příklad:

```
motor1.setRampRate(5); // motor se z 0% na 100% bude rozbíhat 20 s.
```

[4] *Pokud je motorek malý a není moc zatížený, dvě sekundy by na plné roztočení měly stačit.*

enableRamping()

Povoluje (zapíná) rampování.

Syntaxe:

```
enableRamping();
```

Parametry:

Funkce nemá žádné parametry.

Návratová hodnota:

Funkce nic nevrací.

Příklad:

```
motor1.enableRamping();
```

disableRamping()

Zakazuje (vypíná) rampování.

Syntaxe:

```
disableRamping();
```

Parametry:

Funkce nemá žádné parametry.

Návratová hodnota:

Funkce nic nevrací.

Příklad:

```
motor1.disableRamping();
```

setDeadZone()

Nastavuje velikost zóny nečinnosti kolem nulové rychlosti.

Syntaxe:

```
setDeadZone(deadZone);
```

Parametry:

deadZone ((*uint8_t*) – velikost zóny nečinnosti v procentech rozsahu (celé číslo)

Návratová hodnota:

Funkce nic nevrací.

Pokud je při následném nastavování rychlosti funkci `set()` požadována rychlost (libovolným směrem) menší nebo rovna nastavené velikost zóny nečinnosti, bude motor zastaven, tj. jeho rychlost bude nastavena na 0.
Pro zrušení zóny nečinnosti nastavte zónu na 0.

Zóna nečinnosti se hodí například pro nastavování rychlosti motoru ručně pomocí potenciometru, kdy běžný uživatel má problém se trefit přesně na nulovou polohu. Nastavením nenulové zóny bude motor stát, i když potenciometr nebude nastaven na nulovou polohu přesně.

Příklad:

```
motor1.setDeadZone(3); // nastavíme-li rychlost motoru na hodnotu
                       // v rozsahu -3 až +3, bude stát.
```

Dotazy na nastavení

getActualValue()

Vrací okamžitou hodnotu rychlosti motoru.

Syntaxe:

```
getActualValue();
```

Parametry:

Funkce nemá žádné parametry.

Návratová hodnota:

Okamžitá hodnota rychlosti.
Funkce vrací hodnotu, která při správném řízení odpovídá aktuální rychlosti motoru. Je-li zapnuto rampování, je potřeba pro správný chod motoru opakovaně volat funkci `run()`. Funkce `getActualValue()` pak vrací hodnotu, která byla uplatněna při poslední aktualizaci rychlosti.

Příklad:

```
Serial.print("Zrovna jedeme na ");
Serial.print(motor1.getActualValue());
Seriál.println(" procent.“);
```

getTargetValue()

Vrací cílovou hodnotu rychlosti motoru, tj. hodnotu parametru powerPercent, nastavenou při posledním volaní funkce set().

Syntaxe:

```
getTargetValue();
```

Parametry:

Funkce nemá žádné parametry.

Návratová hodnota:

Cílová hodnota parametru **powerPercent** funkce `set()`.

V průběhu rampování (pokud je zapnuto) se aktuální a cílová rychlost liší (aktuální se mění z počáteční hodnoty směrem k cílové). Po ustálení rychlosti motoru již budou stejné.

Příklad:

```
Serial.print("Chceme jet na ");
Serial.print(motor1.getTargetValue());
Seriál.println(" procent.“);
```

isRamping()

Vrací TRUE nebo FALSE podle toho, zda je rampování zapnuto či nikoli.

Syntaxe:

```
isRamping();
```

Parametry:

Funkce nemá žádné parametry.

Návratová hodnota:

TRUE – pokud je rampování zapnuto (*bool*).

FALSE – v opačném případě (*bool*).

Příklad:

```
if(motor1.isRamping())
{
  Serial.println(“Rampování zapnuto.“);
}
else
{
  Serial.println(“Rampování vypnuto.“);
}
```

getRampRate()

Vrací naposledy nastavenou hodnotu strmosti rozběhové / doběhové rampy.

Syntaxe:

```
getRampRate();
```

Parametry:

Funkce nemá žádné parametry.

Návratová hodnota:

Procento přírůstku rychlosti za jednu sekundu (viz popis funkce `setRampRate()`).

Příklad:

```
rampa = motor1.getRampRate());
Serial.print("Nastavené zrychlení: ");
Serial.print(rampa);
Serial.println(" procent za sekundu.");
Serial.print("Z 0 na 100 za ");
Serial.print(100.0/rampa);
Serial.println(" sekund");
```

getDeadZone()

Vrací naposledy nastavenou hodnotu pásma nečinnosti.

Syntaxe:

```
getDeadZone();
```

Parametry:

Funkce nemá žádné parametry.

Návratová hodnota:

Aktuálně nastavená zóna nečinnosti (viz popis funkce `setDeadZone()`).

Příklad:

```
int zona = motor1.getDeadZone();
Serial.print("Zóna nečinnosti je od ");
Serial.print(-zona);
Serial.print(" do ");
Serial.print(zona);
Serial.println(“ včetně.”);
```