Dokumentation: `Robot`

Jarne Götz

2026

Git-Repository: https://git.zerozipp.dev/zerozipp/robot

1 Überblick

Dieses Dokument beschreibt die Klassen `Motor` und `Robot`. `Motor` implementiert eine einfache, nicht-blockierende Ansteuerung eines 4-poligen Schrittmotors über GPIO. `Robot` koordiniert drei Motoren (Basisrotation `yaw` sowie zwei Armachsen `arm1`, `arm2`) und bietet sowohl eine Winkelsteuerung als auch eine Punktsteuerung in der Ebene.

2 Klasse `Motor`

2.1 Zweck

Die Klasse `Motor` kapselt:

- die Ausgabe einer festen 4-stufigen Schrittsequenz auf vier GPIO-Pins,
- eine zeitgesteuerte Schrittweiterleitung über `millis()` (nicht-blockierend),
- die Winkelabschätzung über einen internen Schrittzähler.

2.2 Konstruktion und Parameter

```
Motor(GPIO_TypeDef* gpio, int pinOffset, int maxSteps)
```

- `gpio`: GPIO-Port (z. B. `GPIOA`).
- `pinOffset`: Bit-Offset des ersten verwendeten Pins; es werden vier zusammenhängende Pins verwendet.
- `maxSteps`: Anzahl der Schritte für eine volle Umdrehung (360°); dient als Kalibrierparameter.

2.3 Initialisierung

```
void begin();
```

`begin()` aktiviert den GPIO-Takt (portspezifisch), konfiguriert die Ausgabe und setzt Zeit- sowie Zählvariablen zurück.

2.4 Bewegung und Winkel

```
void update(int target, int delay);
```

Bewegt den Motor in Richtung `target` (Grad). Pro Aufruf wird höchstens ein Schritt ausgeführt, sofern seit dem letzten Schritt mindestens `delay` Millisekunden vergangen sind.

```
int angle();
```

Berechnet den Winkel aus dem Schrittzähler:

$$\texttt{angle} = \frac{\texttt{stepCount}}{\texttt{maxSteps}} \cdot 360$$

2.5 Technisches Prinzip (kurz)

Die Rotation entsteht durch zyklisches Umschalten einer 4-stufigen Bitsequenz auf den vier Motorpins. Die Schrittfrequenz wird über `millis()` begrenzt, wodurch die Ansteuerung ohne blockierende Delays erfolgen kann.

3 Klasse `Robot`

3.1 Zweck

`Robot` fasst drei `Motor`-Instanzen zu einem einfachen Roboterarm zusammen:

- `yaw`: Rotation der Basis zur Ausrichtung in der Ebene,
- `arm1`, `arm2`: zwei Gelenkwinkel.

3.2 Initialisierung und Grundfunktionen

```
void begin();
bool reset();
```

`begin()` initialisiert alle Motoren. `reset()` fährt alle Achsen auf 0 Grad zurück (`pose(0,0,0)`).

3.3 Winkelsteuerung: `pose`

```
bool pose(int tYaw, int tArm1, int tArm2);
```

`pose` berechnet die Abweichung der aktuellen Winkel zu den Zielwinkeln und steuert die Motoren so, dass Achsen mit größerem Weg schneller getaktet werden. Die Methode liefert `true`, sobald alle Zielwinkel erreicht sind.

3.4 Punktsteuerung: `point`

```
bool point(float x, float y);
```

`point` berechnet den Abstand zum Zielpunkt

$$dist = \sqrt{x^2 + y^2}$$

und bestimmt `yaw` aus dem Richtungswinkel:

$$tYaw = \text{degrees}(\text{atan2}(x, y))$$

Die Armwinkel werden über eine vereinfachte Dreiecksgeometrie mit `size` als Skalierungsmaß berechnet und anschließend an `pose` übergeben.

4 Minimalbeispiel

```
Motor m1(GPIOC, 0, 2048);
Motor m2(GPIOC, 4, 2048);
Motor m3(GPIOC, 8, 2048);
Robot bot(7.0f, m1, m2, m3);

void setup() { bot.begin(); }
void loop() { bot.point(5, 8); }
```

5 Lizenz