Bibliotheken-Befehlsreferenz

i2cMaster / anadigMaster

Es muss jeweils ein Objekt der entsprechenden Klasse aus der Bibliothek *i2cMaster* bzw. *anadigMaster* erzeugt werden:

klassenname objektname;

Display (Library i2cMaster)

Funktion	*Parameter*	*Anmerkung*
start()	keine	sollte in setup() stehen
clear()	keine	Löschen des Display-Inhalts
print(inhalt)	Zahl oder „Text“	Display-Ausgabe des Inhalts
print(zahl, n)	float zahl, byte n	Ausgabe der Zahl mit n Nachkommastellen
println(inhalt)	Zahl oder „Text“	Wie print, mit abschließendem Zeilenumbruch
setRow(reihe)	byte reihe	Festlegung der nächsten Zeile (1 … 4)

In zeitkritischen Programmteilen für Sensoren oder Motoren sollten möglichst keine Display-Funktionen verwendet werden, da der I2C-Bus zusätzlich belastet wird.

ColorSensorA (Library i2cMaster)
ColorSensorB (Library i2cMaster)

Funktion	*Parameter*	*Rückgabewert (int16_t)*	*Anmerkung*
start()	keine	kein	sollte in setup() stehen
start(d)	DigitalPin (byte)	kein	nur ColorSensorA Version 2 Standard: d = 13
calibrate()	keine	Attribute r0, g0, b0	Dunkel-Kalibierung
reset()	keine	kein	r0, g0, b0 auf 0
getRGB()	keine	Attribute r, g, b	Durchführung der Farbmessung
getRGB(r, g, b)	Variablennamen r, g, b (uint16_t)	Variablen r, g, b	Farbmessung und Speicherung in den Variablen
flashRGB()	keine	Attribute r, g, b	Differenzmessung der Farbanteile, Messdauer 5 ms, nur ColorSensorA Version 2
flashRGB(r, g, b)	Variablennamen r, g, b (uint16_t)	Variablen r, g, b	Differenzmessung und Speicherung in den Variablen, Messdauer 5 ms, nur ColorSensorA Version 2
hue()	keine	Hue-Wert des HSL-Modells	Berechnet aus letzten Messwerten (-179 … 180)
hue(r, g, b)	Farbanteile (uint16_t)	Hue-Wert des HSL-Modells	Berechnet aus den Parametern

saturation()	keine	Sättigung des HSL-Modells	Berechnet aus letzten Messwerten (0 … 100)
saturation(r, g, b)	Farbanteile (uint16_t)	Sättigung des HSL-Modells	Berechnet aus den Parametern
color()	keine	Farbcode*	Berechnet aus letzten Messwerten (0 ...5)
color(r, g, b)	Farbanteile (uint16_t)	Farbcode*	Berechnet aus den Parametern
intens()	keine	Intensität	Berechnet aus letzten Messwerten (0 … ca.2500)
intens(r, g, b)	Farbanteile (uint16_t)	Intensität	Berechnet aus den Parametern
ledOn()	keine	kein	LEDs einschalten, nur ColorSensorA Version 2
ledOff()	keine	kein	LEDs ausschalten, nur ColorSensorA Version 2
blackLimit	(Variable uint16_t)	Intensitäts-Grenz-wert für Schwarz	Einfluss auf Ergebnis von color() (default = 40)
r	(Variable uint16_t)	Rotanteil	letzter Messwert
g	(Variable uint16_t)	Grünanteil	letzter Messwert
b	(Variable uint16_t)	Blauanteil	letzter Messwert

* Farbcodes:

Code #	0	1	2	3	4	5
Farbe	BLACK	RED	YELLOW	GREEN	BLUE	WHITE
alternativ	SCHWARZ	ROT	GELB	GRUEN	BLAU	WEISS

Battery (Library anadigMaster)

Funktion	*Parameter*	*Rückgabewert*	*Anmerkung*
getVoltage()	keine	float Spannung	in Volt
percent(voltage)	float Spannung	uint16_t Prozent Ladezustand (LiPo 3 Zellen)	0% = 10.5 V ; 100% = 12.4 V

Button (Library anadigMaster)

Funktion	*Parameter*	*Rückgabewert*	*Anmerkung*
pressed()	keine	bool 0 = offen ; 1 = gedrückt	
wait(dly) wait()	uint32_t Verzögerung ms default = 0	kein	Warten bis Taster gedrückt + dly ms
count(timeout) count()	uint8_t Timeout sec default = 2	uint16_t Tasten-Klicks	Beenden nach timeout Inaktivität

Led (Library anadigMaster)

Funktion	*Parameter*	*Anmerkung*
on()	keine	Schaltet LED ein
off()	keine	Schaltet LED aus
blink(count, period)	uint8_t Anzahl Blinker, uint16_t Periode in ms	LED blinkt

LineSensor (Library anadigMaster)

Funktion	*Parameter*	*Rückgabewert*	*Werte*
getAmbient(a1, a2)	Variablennamen a1, a2 (int16_t)	Variablen a1, a2	1 … 1023
getReflections(a1, a2)	Variablennamen a1, a2 (int16_t)	Variablen a1, a2	1 … 1023
getDiff()	keine	int16_t Differenz der Reflexionen	a1 - a2
getSum()	keine	int16_t Summe der Reflexionen	a1 + a2
getAmbientDiff()	keine	int16_t Differenz der Lichter	a1 - a2
getAmbientSum()	keine	int16_t Summe der Lichter	a1 + a2
ledOn()	keine	kein	
ledOff()	keine	kein	
calibrate(wL, wR, sL, sR)	Sensorwerte L/R für weiß/schwarz (nur für getOffset)	kein	1 … 1023
getOffset()	keine	int16_t Relative Abweichung von der Linienmitte (links <0, rechts >0)	-1000 … +1000

GetOffset liefert umgekehrtes Vorzeichen wie getDiff !

UltrasonicSensor (Library anadigMaster)

Funktion	*Parameter*	*Rückgabewert*	*Werte*
getDistance() getDistance1()	keine	Gemessener Abstand in mm des ersten Sensors	1 … 1000 0 = kein Messwert
getDistance2()	keine	Gemessener Abstand in mm des zweiten Sensors	1 … 1000 0 = kein Messwert
soundSpeed	(Variable uint16_t)	Schallgeschwindigkeit m/s	default = 343

ServoMotor (Library anadigMaster)

Bei der Objektdefinition muss der Typ angegeben werden (port = 8 oder 9 je nach Controllerbuchse):
ServoMotor servo(GEEK, port); oder ServoMotor servo(MINI, port);

Funktion	*Parameter*	*Anmerkung*
turnTo(winkel)	int16_t Zielwinkel	Absoluter Winkel gegen UZS GEEK: 0 … 360 MINI: 0 … 180
slowTo(winkel, speed)	int16_t Zielwinkel, int16_t Speed	Drehung mit speed Grad/Sek.
coast()	keine	Servomotor stromlos geschaltet

Drivetrain = Fahrmotoren-Steuerung (Library i2cMaster)

Steuerung der Schrittmotoren durch I2C-Kommandos:
Die Bibliotheks-Funktionen werden über den I2C-Bus übertragen.
Es muss ein Objekt der Klasse *Drivetrain* aus der Bibliothek *i2cMaster* erzeugt werden (in ino-Datei):

```
#include "i2cMaster.h"
Drivetrain robotDrive(4);   // I2C address
```

Die Motor-Kommandos werden per I2C-Bus übertragen mit Hilfe des sendCommand-Befehls.

command #	Command *(unit8_t)*	Bedeutung	Einheit	Wert/Bereich *(int16_t)*
0	NONE_S	keine	-----	0
1	GO	Starte Antrieb	-----	0
2	STOP	Stoppe Bewegung	-----	0
3	SPEED	Setze Geschwindigkeit	Steps/s	-2000 … +2000 *
4	STEERING	Setze Lenkwert	-----	-100 … +100
5	ACCEL	Setze Beschleunigung	Steps/s²	100 … 5000 *
6	DECEL	Setze Bremsverzögerung	Steps/s²	100 … 5000 *
7	TARGET	Setze Ziel	Steps	1 … 32767
8	COAST	Setze Freilaufmodus	-----	0
9	BRAKE	Setze Blockiermodus	-----	0

* praktische Grenze

Bibliotheks-Funktionen:

Funktion	*Parameter (int16_t)*	*Bedeutung*	*Einheit*	*Bereich (int16_t)*	*Default wert*
sendCommand	command, wert	Motor-Kommando über I2C (s. oben)	s. oben	s. Tabelle oben	-----
setTargetSteps	steps	Schritte zum Ziel	steps	-32768 – +32767	-----
setSpeed	speed	Geschwindigkeit	20 steps/s cm/s #	-100 – +100 *	-----
setSteering	steering	Lenkwert	-----	−100 – +100	-----

setAccelerations	accel, decel	Beschleunigungen	20 steps/s² cm/s² #	10 – 500 *	250
go()	-----	Start der Roboterfahrt	-----	-----	-----
stop()	-----	Sofortiger Stopp	-----	-----	-----
brake()	-----	Motoren im Bremsmodus	-----	-----	-----
coast()	-----	Motoren im Freilaufmodus	-----	-----	-----
getStatus()	-----	Rückgabe (int16_t) des Fortschritts	steps	0 – 32767 -1 = Motoren aus -9 = Fehler	-----
isRunning()	-----	Rückgabe (bool) true, wenn Motoren laufen	-----	true/false	-----
wait()	-----	Wartet, solange Motoren laufen	-----	-----	-----
estimateTime	distance, speed, accel, decel	Rückgabe (int16_t) der Fahrzeit	ms	wie Einzelparameter	-----

mit 200 mm Radumfang und 400 steps/Umdrehung * praktische Grenze

Nach Start der Fahrbewegung mittels go() lassen sich die Zielschritte (setTargetSteps), Geschwindigkeit (setSpeed) und Beschleunigungen (setAccelerations) nicht mehr verändern. Lediglich der Lenkwert (setSteering) kann während des Laufs variieren (z.B. für eine Linienverfolgung).

getStatus() ist immer abrufbar und die Motoren lassen sich mittels stop() jederzeit stoppen.

MotorsX = Hilfsmotoren-Steuerung (Library i2cMaster)

Die Funktionen basieren auf Schrittmotoren mit 400 Halbschritten/Umdrehung (0.9 °/step).

Steuerung der Schrittmotoren durch I2C-Kommandos:
Die Bibliotheks-Funktionen werden über den I2C-Bus übertragen.
Es muss ein Objekt der Klasse *MotorsX* aus der Bibliothek *i2cMaster* erzeugt werden (in ino-Datei):

```
#include "i2cMaster.h"
MotorsX motorAB(5);  // I2C address
```

Die Motor-Kommandos werden übertragen mit Hilfe des Befehls:

```
motorAB.sendCommand(command, wert);
```

command #	command *(unit8_t)*	Bedeutung	Einheit	Wert/Bereich *(int16_t)*
0	NONE_X	keine	-----	0
1	GO_A	Starte Antrieb	-----	0
2	STOP_A	Stoppe Bewegung	-----	0
3	SPEED_A	Setze Geschwindigkeit	steps/s	-1200 … +1200 *
4	ACCEL_A	Setze Beschleunigung	steps/s²	100 … 10000 *

5	DECEL_A	Setze Bremsverzögerung	steps/s²	100 … 10000 *
6	TARGET_A	Setze Ziel	steps	1 … 32767
7	COAST_A	Setze Freilaufmodus	-----	0
8	BRAKE_A	Setze Blockiermodus	-----	0
9	GO_B	Starte Antrieb	-----	0
10	STOP_B	Stoppe Bewegung	-----	0
11	SPEED_B	Setze Geschwindigkeit	steps/s	-1200 … +1200 *
12	ACCEL_B	Setze Beschleunigung	steps/s²	100 … 10000 *
13	DECEL_B	Setze Bremsverzögerung	steps/s²	100 … 10000 *
14	TARGET_B	Setze Ziel	steps	1 … 32767
15	COAST_B	Setze Freilaufmodus	-----	0
16	BRAKE_B	Setze Blockiermodus	-----	0

_A, _B entsprechend Motoranschluss * praktische Grenze

Bibliotheks-Funktionen:

Funktion	*Parameter (int16_t)*	*Bedeutung*	*Einheit*	*Bereich (int16_t)*	*Default wert*
setTargetSteps_A setTargetSteps_B	steps	Schritte zum Ziel	Steps (0.9°/step)	-32768 – +32767	-----
setSpeed_A setSpeed_B	speed	Geschwindigkeit	°/s	-1080 – +1080 *	-----
setAccelerations()	-----	Beschleunigungen	°/s²	-----	5000
setAccelerations_A setAccelerations_B	accel, decel	Beschleunigungen	°/s²	100 – 10000 *	-----
go_A() go_B()	-----	Start der Drehbewegung	-----	-----	-----
stop_A() stop_B()	-----	Sofortiger Stopp	-----	-----	-----
brake_A() brake_B()	-----	Beide Motoren im Bremsmodus	-----	-----	-----
coast_A() coast_B()	-----	Beide Motoren im Freilaufmodus	-----	-----	-----
getStatus()	-----	Rückgabe (int16_t) des Fortschritts	-----	0 = beide ruhig 1 = MotorA bew. 2 = MotorB bew. 3 = beide bewegt -9 = Fehler	-----
isRunning_A() isRunning_B()	-----	Rückgabe (bool) true, wenn Motor läuft	-----	true/false	-----
wait_A() wait_B()	-----	Wartet, solange Motor läuft	-----	-----	-----

* praktische Grenze