RocciBoard-Library

> *GitHub-Repository der RocciBoard-Library:*
> *https://github.com/Robotics-Competence-Center-Illertal-e-V/RocciBoard-Library*

Das RocciBoard Arduino Shield des Robotics Competence Center Illertal e. V. ist eine Erweiterung des Arduino Mega bzw. Arduino Giga, die grundlegende Funktionen eines Roboters bereitstellt und eine Basis für eigene Ideen und Erweiterungen bietet. Die zugehörige Bibliothek wurde ebenfalls vom Arbeitskreis Entwicklung speziell für die Verwendung zu Unterrichtszwecken entwickelt. Sie bietet ein einfaches, intuitives und erweiterbares Framework für die Verwendung von Funktionen des Boards, von Motoren und Sensoren. Die Bibliothek ist nicht allzu umfangreich gehalten, um den Kursteilnehmern zwar die Programmierung des Arduinos realitätsnah zu vermitteln, komplexere Funktionen wie die Verwendung der Sensoren aber zu vereinfachen.

Die RocciBoard-Bibliothek enthält **zahlreiche Beispielprogramme**, welche die Verwendung und Ansteuerung von Sensoren und Motoren demonstrieren. Ebenfalls verfügbar ist eine **RocciBoard-Projektvorlage** (*template.ino*), welche als vorstrukturierte, leere Vorlage verwendet werden kann. Sie beinhaltet bereits die wichtigsten Deklarationen und Anweisungen.

> *In der folgenden Dokumentation werden alle Funktionen ausgiebig erläutert und beschrieben.*

Grundlagen

> *Bibliotheken werden in der Programmiersprache C++, in welcher das Arduino (und damit auch das RocciBoard) programmiert werden, meistens in sogenannten Header-Dateien zusammengefasst. Diese erkennt man an der Dateiendung ".h". Auch die RocciBoard-Library besteht aus solchen Header-Dateien und muss dem Projekt hinzugefügt werden.*

Die Bibliothek wird durch `#include <rocciboard.h>` **in das Projekt eingebunden**. Diese sogenannte Precompiler-Anweisung beinhaltet bereits alle benötigten Bibliotheken und Abhängigkeiten.
Um das RocciBoard zu verwenden, muss in der Deklarations-Sektion (oberer Teil des Programms, außerhalb von Funktionen) des Projektes zunächst ein **RocciBoard-Objekt deklariert werden**. Dies wird durch die Anweisung `RocciBoard rb` erreicht, wobei `rb` als Bezeichner des RocciBoard-Objektes frei wählbar ist. Dieser festgelegte Name muss allerdings bei allen weiteren Aufrufen verwendet werden. In dieser Dokumentation wird der Bezeichner `rb` beibehalten.
Das RocciBoard muss vor der **erstmaligen Verwendung initialisiert** werden, dazu wird die Funktion `rb.init()` aufgerufen.

```
#include <rocciboard.h>
RocciBoard rb;
void setup() {
    rb.init();
}
```

Grundfunktionen

> *Um die Stromversorgung des Roboters zu überwachen und den Ladezustand der Batterie abzufragen, besitzt das RocciBoard eine interne Spannungsmessung. Die aktuelle Versorgungsspannung wird über einen Analog-Pin des Arduinos in einem Analog-Digital-Umwandler abgefragt. Auch das Debugging des Boards ist wichtig. Die rote Debug-LED ist gut sichtbar auf dem RocciBoard platziert und bietet beispielsweise die Möglichkeit, Störungen und Fehler auszugeben.*

Die Grundfunktionen des RocciBoard **befinden sich im RocciBoard-Object** und können über dieses aufgerufen werden.

- `[float] getBatteryVoltage()` - Gibt die aktuelle Versorgungsspannung des RocciBoards (also der Batterie bzw. des angeschlossenen Netzteils) zurück.

  ```
  float battery_voltage = rb.getBatteryVoltage();
  ```

- `[uint8_t] getBatteryCharge()` - Gibt den aktuell errechneten Ladezustand der Batterie von 0 bis 100 Prozent zurück. (Falls das RocciBoard an eine externe Stromversorgung angeschlossen ist, ergibt dieser Wert möglicherweise keinen Sinn)

  ```
  int battery_charge = rb.getBatteryCharge();
  ```

- `[void] blinkDebugLED()` - Lässt die interne Debug-LED kurz aufblinken. Diese Funktion kann beispielsweise zum Signalisieren eines Fehlers verwendet werden.

  ```
  if(has_error) rb.blinkDebugLED();
  ```

- `RB_DEBUG_LED` - Diese Konstante beinhaltet den Pin der eingebauten Debug-LED. Sie kann verwendet werden, um eigene Funktionen zur Ansteuerung der LED zu schreiben.

  ```
  digitalWrite(RB_DEBUG_LED, HIGH);
  delay(1000);
  digitalWrite(RB_DEBUG_LED, LOW);
  ```

Motoren

> *Motoren spielen in der Robotik eine wichtige Rolle, da sie die Fortbewegung des Roboters und die Interaktion mit der Umwelt und die Manipulation dieser ermöglichen. Das RocciBoard besitzt vier Motortreiber, mit welchen 12V-Bürsten-Gleichstrommotoren angetrieben und gesteuert werden können.*

Die **Verwendung von Motoren** ist in jedem RocciBoard-Projekt **standardmäßig eingebunden**, bedarf also keiner weiteren Initialisierung. Funktionen zur Ansteuerung von Motoren **finden sich in den RBMotor-Objekten**, die unter `rb.motor[i]` als Array im RocciBoard-Objekt zu finden sind. Der Index `i` gibt den anzusteuernden Motor an, also die Nummer des Motors von 0 bis 3. Eine Funktion der Motoren wird mit `rb.motor[i].eineFunktion()` aufgerufen.

- `[void] rotate(int16_t speed)` - Der Motor beginnt mit der Geschwindigkeit `speed` zu rotieren. Die Geschwindigkeit wird von -255 bis +255 angegeben, negative Werte lassen den Motor rückwärts rotieren.

```
rb.motor[0].rotate(100);
rb.motor[3].rotate(-200);
rb.motor[EINE_INT_KONSTANTE].rotate(eine_int_variable);
```

- `[void] stop(bool brake)` - Der Motor wird gestoppt. Der Parameter `brake` gibt an, ob der Motor elektromechanisch gebremst wird oder ob er bis zum Stillstand auslaufen soll. Er ist optional und wird bei weglassen durch `true` ersetzt.

```
rb.motor[1].stop();
rb.motor[2].stop(false);
rb.motor[eine_int_variable].stop(EINE_BOOL_KONSTANTE);
```

- `[int16_t] getSpeed()` - Gibt die aktuell eingestellte Geschwindigkeit (von -255 bis +255) des Motors zurück.

```
int speed = rb.motor[1].getSpeed();
eine_int_variable = rb.motor[EINE_INT_KONSTANTE].getSpeed();
ein_int_array[1] = rb.motor[eine_int_variable].getSpeed();
```

Standardisierte Sensoren

> *Sensoren sind von größter Bedeutung, da der Roboter für die selbstständige Fortbewegung und die Ausführung seiner Aufgaben auf Informationen über die Außenwelt angewiesen ist. Das RocciBoard verwendet, wie viele andere Systeme in der Robotik, zur Kommunikation mit Sensoren den I²C-Bus. Dieser ist standardmäßig jedoch häufig von Adresskonflikten und Verbindungsproblemen betroffen. Um diese Fehler möglichst komplett zu beheben läuft die Kommunikation über einen Multiplexer, welcher immer nur den Kommunikationskanal zu jeweils einem Sensor öffnet und so Konflikte behebt. Um die etwas komplexere Ansteuerung der Sensoren zu vereinfachen, wurden einige häufig verwendete Sensoren bzw. deren Bibliotheken für das RocciBoard standardisiert und angepasst.*

Das **RBSensor-Interface** ermöglicht eine einfache Deklaration und Initialisierung der Sensoren und eine reibungsfreie Verwendung.
Zunächst muss der jeweilige Sensor deklariert werden. Dies geschieht durch Erzeugen eines Sensorobjektes mit `RBSensor sensor(1)` unter Angabe des zu verwendenden Ports. `RBSensor` wird hier durch die gewünschte Sensorklasse ersetzt, `sensor` durch den zugewiesenen Namen des Sensors und `1` durch den Port des Sensors am RocciBoard (von 0 bis 7).
In der Setup-Funktion muss der Sensor dann initialisiert werden. Diese Aufgabe wird durch das RocciBoard erledigt, welches dem Sensor den Multiplexer zuweist und die Initialisierung des Sensors anstößt. Die Funktion lautet `rb.initRBSensor(sensor)`. Der zu initialisierende Sensor `sensor` wird hier mit dem als Objekt (Referenz) übergeben.

> *Diese Art der Deklaration und Initialisierung kann auf alle unten aufgelisteten, standardisierten Sensoren angewandt werden.*

```
RBCompass compass(0);
void setup() {
    rb.initRBSensor(compass);
}
```

Kompass-Sensor (BNO055)

> *Der BNO055 Kompass-Sensor liefert Daten über seine Ausrichtung, seine Bewegung und die Umgebungstemperatur.*

Der Kompass-Sensor wird als Objekt mit `RBCompass compass(1)` eingebunden, wobei der Name des Sensors `compass` frei wählbar ist. Der übergebene Wert `1` gibt den Sensor-Port des Sensors am RocciBoard an.

- `[int16_t] getHeading()` - Gibt die aktuell gemessene Drehung um die Gierachse (Ausrichtung bzw. Himmelsrichtung) zurück.
- `[int16_t] getPitch()` - Gibt die aktuell gemessene Drehung um die Nickachse (Kippung nach hinten/vorne) zurück.
- `[int16_t] getRoll()` - Gibt die aktuell gemessene Drehung um die Rollachse (Kippung nach links/rechts) zurück.
- `[int8_t] getTemperatureCelsius()` - Gibt die aktuell gemessene Temperatur des Sensors in Grad Celsius zurück.
- `[RBVector] getVecOrientation()` - Gibt die aktuell gemessene Orientierung des Sensors als RBVector zurück.
- `[RBVector] getVecAccelerometer()` - Gibt die aktuell gemessene Gesamtbeschleunigung des Sensors als RBVector zurück.
- `[RBVector] getVecLinearAcceleration()` - Gibt die aktuell gemessene Beschleunigung des Sensors ohne die Gravitationsbeschleunigung als RBVector zurück.
- `[RBVector] getVecGyroscope()` - Gibt die aktuell gemessene Drehgeschwindigkeit des Sensors als RBVector zurück.
- `[RBVector] getVecMagneticField()` - Gibt die Stärke des aktuell gemessenen Magnetfeldes als RBVector zurück.
- `[RBVector] getVecGravity()` - Gibt die aktuell gemessene Gravitationsbeschleunigung des Sensors als RBVector zurück.

```
RBCompass compass(2);
void setup() {
    rb.initRBSensor(compass);
}
void loop() {
    int pitch = compass.getPitch();
    RBVector magnetic_field_vec = compass.getVecMagneticField();
}
```

Ein RBVector ist ein Struct-Datentyp, welcher drei Werte (x,y,z) speichern kann. Die Einzelwerte können mit `my_vector.x` ausgerufen werden.

```
RBVector my_vector = {1.0f, 2.0f, 3.0f};
float y_value = my_vector.y;
my_vector.x = y_value;
my_vector.z = 4.0f;
```

Laser-Sensor (VL53L0X bzw. VL53L1X)

> *Die Sensoren VL53L0X und VL53L1X sind Laser-Distanz-Sensoren, welche mithilfe des Time-of-Flight-Verfahrens den Abstand von Objekten messen können. Der VL53L0X ist für eine kurze Distanz bis ca. 50 Zentimeter ausgelegt, bei größeren Distanzen*

> *wird die Messung ungenau. Der VL53L1X ist für größere Distanzen bis ca. 2 Meter ausgelegt.*

Der Laser-Sensor wird als Objekt mit `RBLaser laser(1, TYPE_LONGRANGE)` eingebunden, wobei der Name des Sensors `laser` frei wählbar ist. Der Typ `TYPE_LONGRANGE` gibt an, dass es sich bei dem Sensor um einen VL53L1X-Sensor mit großer Reichweite handelt. Es stehen die Typen `TYPE_LONGRANGE` bzw. `TYPE_VL53L1X` und `TYPE_SHORTRANGE` bzw. `TYPE_VL53L0X` zur Verfügung.

- `[uint16_t] getDistanceMillimeters()` - Gibt die aktuell gemessene Entfernung in Millimetern zurück.
- `[float] getDistanceCentimeters()` - Gibt die aktuell gemessene Entfernung in Zentimetern zurück. Das Ergebnis wird aus Millimetern errechnet und ist eine Fließkommazahl (float).
- `[float] getDistanceMeters()` - Gibt die aktuell gemessene Entfernung in Metern zurück. Das Ergebnis wird aus Millimetern errechnet und ist eine Fließkommazahl (float).
- `[float] getDistanceInches()` - Gibt die aktuell gemessene Entfernung in Zoll (Inches) zurück. Das Ergebnis wird aus Millimetern errechnet und ist eine Fließkommazahl (float).
- `[float] getDistanceFeet()` - Gibt die aktuell gemessene Entfernung in Fuß (Feet) zurück. Das Ergebnis wird aus Millimetern errechnet und ist eine Fließkommazahl (float).

```
RBLaser laser(5, TYPE_SHORTRANGE);
void setup() {
    rb.initRBSensor(laser);
}
void loop() {
    int distance = laser.getDistanceMillimeters();
}
```

Farb-Reflexions-Sensor (TCS34725)

> *Der TCS34725 Farb-Reflexions-Sensor kann die Farben seiner Umgebung messen und auswerten. Dafür emittiert er Licht über seine interne LED, welches von der zu messenden Oberfläche reflektiert wird. Die farbliche Zusammensetzung dieses Lichts wird anschließend durch mehrere Sensoreinheiten mit Farbfiltern bestimmt.*

Der Farb-Reflexions-Sensor wird als Objekt mit `RBColor color(1)` eingebunden, wobei der Name des Sensors `color` frei wählbar ist. Der übergebene Wert `1` gibt den Sensor-Port des Sensors am RocciBoard an.

- `[uint16_t] getRed()` - Gibt den aktuell gemessenen Reflexionswert der Farbe Rot zurück.
- `[uint16_t] getGreen()` - Gibt den aktuell gemessenen Reflexionswert der Farbe Grün zurück.
- `[uint16_t] getBlue()` - Gibt den aktuell gemessenen Reflexionswert der Farbe Blau zurück.
- `[uint16_t] getClear()` - Gibt den aktuell gemessenen, farblosen (über alle Farben gemittelten) Reflexionswert zurück.
- `[uint16_t] getColorTemperature()` - Gibt die aktuell errechnete Farbtemperatur der Reflexion in Kelvin zurück.

- `[uint16_t] getLux()` - Gibt die aktuell errechnete Beleuchtungsstärke des Sensors in Lux zurück.

```
RBColor color(3);
void setup() {
    rb.initRBSensor(color);
}
void loop() {
    int red = color.getRed();
}
```

Nicht-standardisierte Sensoren

> *Selbstverständlich können mit dem RocciBoard auch weitere, nicht-standardisierte Sensoren verwendet werden. Damit diese an den Sensorports des RocciBoard verwendet werden können müssen allerdings einige Dinge beachtet werden, weil diese Ports - wie weiter oben beschrieben - nur über den internen I²C-Multiplexer erreichbar sind.*

Bevor eine Anweisung eines Sensors, der eine Kommunikation über I²C verwendet, ausgeführt werden kann, **muss der jeweilige Port des Multiplexers geöffnet werden**. Dazu wird die Funktion `rb.openSensorPort(1)` aufgerufen, welche den Sensorport `1` zur Kommunikation mit dem Sensor öffnet.
Wenn der Kanal geöffnet ist, **kann die Anweisung des Sensors ausgeführt werden**, beispielsweise eine Initialisierung oder eine Abfrage der Sensorwerte.
Nach der Benutzung des Sensor **muss der Port wieder geschlossen werden**, um die Kommunikation mit anderen Sensoren nicht zu stören. Dies wird mit der Funktion `rb.closeSensorPort(1)` erreicht.
Nun kann entweder ein weiterer Kanal geöffnet werden oder mit der Ausführung des restlichen Programmes fortgefahren werden.
Alternativ zu einem spezifischen Kanal können mit `rb.closeAllSensorPorts()` auch alle Kanäle geschlossen werden.

> *Neben Sensoren können mit dieser Vorgehensweise auch andere I²C-Geräte eingebunden werden.*

```
rb.openSensorPort(2);
int my_value = mySensor.getValue();
rb.closeSensorPort(2);
```

Erweitert: Standardisierte Sensoren ohne Multiplexer

> *Die standardisierten Sensoren können auch ohne den Multiplexer direkt an einen I²C-Bus des Arduino angeschlossen werden. Dies ist allerdings nicht empfehlenswert, da Probleme wie Adresskonflikte auftreten können. Deshalb sollte dies nur von fortgeschrittenen Benutzern durchgeführt werden.*

Die Verwendung des Kompass-Sensors wird über dieses Interface aktuell nicht unterstützt!

Die RBSensor-Sensoren werden statt mit dem Sensor-Port mit der dem Wire-Objekt des jeweiligen nativem I²C-Busses deklariert, also mit `Wire` für den Standard-I²C, `Wire1`

für I²C-1 oder `Wire2` für I²C-2. Hierdurch wird die Multiplexer-Funktionalität der Sensoren deaktiviert und der gewünschte I²C-Bus aktiviert. Die Initialisierung findet wie gewohnt über das RocciBoard mit `rb.initRBSensor(sensor)` statt. Alternativ kann der Sensor auch direkt mit `sensor.init()` initialisiert werden.

```
RBCompass compass(Wire1);
void setup() {
     rb.initRBSensor(compass);
}
```

Jonas Biener und Alexander Ulbrich (2023-09)
Robotics Competence Center Illertal e. V.
Sachsenstraße 12, 89250 Senden (Germany)
