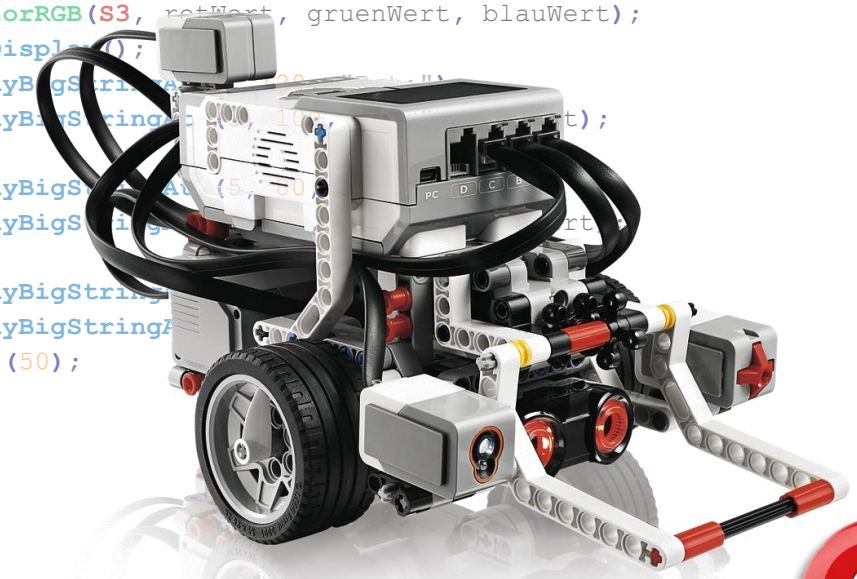
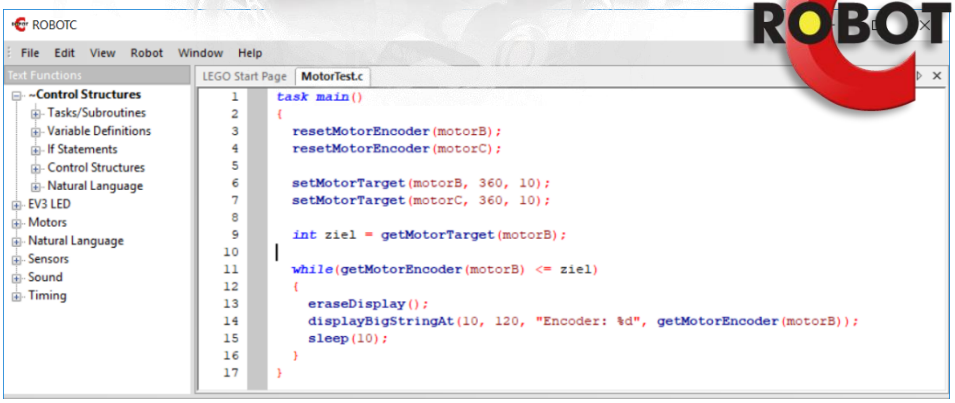
RobotC Beispielprogramme für EV3

Kepler Gymnasium Weil der Stadt | Thomas Jörg | Stand: 03. September 2018 | Version 1.5

```
task main(){
    SensorType[S3] = sensorEV3_Color;
    SensorType (S3);
    SensorType [S3] = modeEV3Color_Color;

    long rotWert;
    long gruenWert;
    long blauWert;

    while (true)
    {
        getColorRGB(S3, rotWert, gruenWert, blauWert);
        eraseDisplay();
        displayBigStringAt(10, 120, "Rot: %d", rotWert);
        displayBigStringAt(10, 100, "Gruen: %d", gruenWert);

        displayBigStringAt(5, 80, "Blau: %d", blauWert);
        displayBigStringAt(10, 60, "Rot: %d", rotWert);

        displayBigStringAt(10, 40, "Gruen: %d", gruenWert);
        displayBigStringAt(10, 20, "Blau: %d", blauWert);
        sleep (50);
    }
}
```





Textausgabe auf dem Bildschirm

Wichtiges Werkzeug für das Debugging ist die Bildschirmausgabe auf dem LCD-Display. RobotC kennt verschiedene Arten der Text- und Zahlenausgabe, die sich allesamt an die gängige C-Konvention halten:

Variante 1: Einfache Textausgabe

```
task main(){                          // Start Hauptprogramm
    eraseDisplay();                   // Gesamtes Display loeschen
    displayText(1,"Hallo Welt");      // Textausgabe in Zeile 1
    sleep(2000);                      // 2000 Millisekunden warten
    displayText(2,"Text Zeile 2");    // Textausgabe in Zeile 2
    waitForButtonPress();             // Auf Knopfdruck warten
}                                     // Programm Ende
```

Variante 2: Textgröße und Platzierung am LCD definieren

```
task main(){                          // Start Hauptprogramm
    eraseDisplay();                   // gesamtes Display loeschen
    displayBigStringAt(10,120, "GANZ OBEN");// Textausgabe x=10, y=120
    displayStringAt(10,100, "Eins drunter");// Textausgabe x=10,y=100
    waitForButtonPress();             // Auf Knopfdruck warten
}                                     // Programm Ende
```

Variante 3: Variablenausgabe einer Integer-Variablen

```
task main(){
    int Variable_1 = 5;               // Variable 1 den Wert 5 zuweisen
    int Variable_2 = 3;               // Variable 2 den Wert 3 zuweisen
    eraseDisplay();                   //gesamtes Display loeschen

    // Um Zahlen darzustellen, werden Platzhalter als %d geschrieben.
    displayBigStringAt(0, 120, "Werte: %d und %d", Variable_1, Variable_2);
    waitForButtonPress();
}
```

Variante 4: Batteriespannung als Float (Fließkomma darstellen)

```
task main(){
    // Messen der Batteriespannung
    float batterieSpannung = getBatteryVoltage();
    eraseDisplay();
    // mit %f wird eine Float-Variable am Bildschirm ausgegeben
    displayBigStringAt(0, 120, "Volt: %f", batterieSpannung);
    waitForButtonPress();
}
```

Motorsteuerung

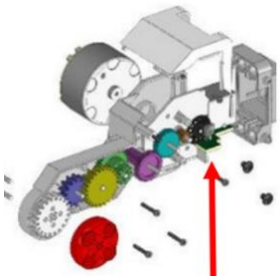
5. Vor- und Zurückfahren mit Zeitsteuerung

Dieses Programm lässt den Roboter für 2 Sekunden mit 20% vorwärts und dann rückwärts fahren:

```
task main(){

    setMotor(motorC, 20); // Motor in Port C mit 20% Geschwindigkeit
    setMotor(motorB, 20); // Motor in Port B mit 20% Geschwindigkeit
    sleep(2000);       // 2000 Millisekunden warten

    stopAllMotors();       // Alle Motoren stopp!

    setMotor(motorC, -20);// Motor C rückwärts mit 20%
    setMotor(motorB, -20);// Motor B rückwärts mit 20%
    sleep(2000);
}
```

---

6. Vor-, zurückfahren und drehen mit Encodersteuerung



Ein Encoder ist ein Sensor, der direkt am Rad angebracht ist. Dieser misst die bereits durchgeführte Drehung des Motors in Grad. Damit kann man einen Roboter millimeter-genau steuern. Zu Beginn des Programms setzt man die gemessene Gradzahl des Encoders bei allen Motoren auf den Wert „0“ zurück, dann beginnt man zu messen.

```
task main(){
    // Vorwaertsfahren, jedes Rad dreht sich um 360 Grad bei 25% Speed
    resetMotorEncoder(motorB);
    resetMotorEncoder(motorC);

    setMotorTarget(motorB, 360, 25);
    setMotorTarget(motorC, 360, 25);

    waitUntilMotorStop(motorB);
    waitUntilMotorStop(motorC);

    // Rueckwaertsfahren, jedes Rad dreht sich um 360 Grad rueckwaerts bei 25% Speed
    resetMotorEncoder(motorB);
    resetMotorEncoder(motorC);

    setMotorTarget(motorB, -360, 25);
    setMotorTarget(motorC, -360, 25);

    waitUntilMotorStop(motorB);
    waitUntilMotorStop(motorC);

    // Drehung, Rad B dreht sich um 360 Grad rueckwaerts, Rad C vorwaerts bei 25% Speed
    resetMotorEncoder(motorB);
    resetMotorEncoder(motorC);

    setMotorTarget(motorB, -360, 25);
    setMotorTarget(motorC, 360, 25);

    waitUntilMotorStop(motorB);
    waitUntilMotorStop(motorC);
}
```

7. Motor: laufend den aktuellen Encoderstand überprüfen

Mit dem Befehl `getMotorEncoder()` liest man den aktuellen Stand des Motorencoders aus.
Mit dem Befehl `getMotorTarget()` liest man den vorher mit `setMotorTarget()` gesetzten Zielwert aus.
In der `while` -Schleife wird der aktuelle Encoderwert ausgelesen und auf dem Display dargestellt.

```
task main(){
    resetMotorEncoder(motorB);
    resetMotorEncoder(motorC);

    setMotorTarget(motorB, 360, 10);
    setMotorTarget(motorC, 360, 10);

    int ziel = getMotorTarget(motorB);

    while(getMotorEncoder(motorB) <= ziel)
    {
        eraseDisplay();
        displayBigStringAt(10, 120, "Encoder: %d", getMotorEncoder(motorB));
        sleep(10);
    }
}
```

Einfache Sensorik

Hier ein Überblick über die verschiedenen Sensortypen und Ports und wie sie in RobotC bezeichnet werden:

sensorEV3_Touch	*EV3 Berührungssensor*
sensorEV3_Color	*EV3 Color Sensor*
sensorEV3_Ultrasonic	*EV3 Ultraschall Sensor*
sensorEV3_Gyro	*EV3 Gyro Sensor*
sensorEV3_IRSensor	*EV3 Infrarot Sensor*
sensorEV3_GenericI2C	*Generisches I2C Gerät*

S1 S2 S3 S4	*Portbezeichungen am EV3*

8. TIMER Zeitsensoren: vier verschiedene Stoppuhren

In RobotC sind 6 verschiedene Uhren des EV3 programmierbar! Wir benötigen nur die vier einfachsten, nämlich Timer1, Timer2, Timer3 und Timer4. Im Beispiel werden Timer T1 und T2 benutzt. Während der T1 nur zu Beginn zurückgesetzt wird und bis Programmschluss läuft, wird T2 nach jedem Schleifendurchlauf wieder auf 0 gesetzt.

```
task main(){
    clearTimer(T1);
    while(getTimer(T1, milliseconds) < 10000)
    {
        eraseDisplay();
        displayBigStringAt(10, 120, "Gesamt: %d", getTimer(T1, milliseconds));
        displayBigStringAt(10,  90, "Schleife: %d", getTimer(T2, milliseconds));
        clearTimer(T2);
        sleep(10);
    }
}
```

9. Anwendung des Timers: Wie schnell ist der EV3?

Timer `T1`wird zurückgesetzt. Die Integer-Variable `durchlaeufe` wird mit Wert 0 initialisiert. Nun beginnt eine **`while`** -Schleife; diese wird solange immer wieder durchlaufen, bis `T1`eine Sekunde gezählt hat. Bei jedem Schleifen-durchlauf wird die Variable durchlaeufe eins hochgezählt. Zuletzt wird deren Wert ausgegeben. Er liegt bei ca. 32.000, das heißt: die drei Befehle `getTimer` aufrufen, **`while`** -Bedingung berechnen und `durchlaeufe` hochzählen benötigen ca. 31,25 Millionstel Sekunden. Ein Befehl benötigt daher ca. 10 Millionstel Sekunden!

```
task main(){
    clearTimer(T1);
    int durchlaeufe = 0;
    while(getTimer(T1, milliseconds) < 1000){
        durchlaeufe = durchlaeufe + 1;
    }
    eraseDisplay();
    displayBigStringAt(10, 120, "Anzahl: %d", durchlaeufe);
    waitForButtonPress();
}
```

---

10. Berührungssensor abfragen

Abgefragt wird der Berührungs-Sensor in Port S1. Um sicherzustellen, dass der Sensor keine Werte von vergangenen Betätigungen mehr in seinem Puffer gespeichert hat, führt man einen `sensorReset()` an seinem Port aus. Der ausgelesene Sensorwert beträgt entweder ‚0' für ‚nicht gedrückt' oder ‚1' für ‚gedrückt':

```
task main(){                                // Hauptprogramm Start
    SensorType[S1] = sensorEV3_Touch;       // In Port S1 wird Touchsensor angemeldet
    sensorReset(S1);
    while ( true )                          // Endlosschleife Start
    {
        int wert = SensorValue[S1];         // Sensor wird ausgelesen und Wert zugewiesen
        displayBigStringAt(10, 120, "Sensorwert: %d", wert); // Bildschirmausgabe
        sleep ( 100 );                      // 0,1 Sekunde warten
        eraseDisplay();
    }                                       // Endlosschleife Ende
}                                           // Hauptprogramm Ende
```

---

11. Ultraschallsensor abfragen

Dieser Ultraschallsensor steckt in S4 (er könnte in jedem anderen Port stecken). Um sicherzustellen, dass sich der Sensor in „cm" und nicht in „zoll" misst, führt man einen sensorReset auf Port S4 aus. Der Abstand wird als Fließkommazahl (float) ermittelt.

```
task main(){

    SensorType[S4] = sensorEV3_Ultrasonic;
    sensorReset(S4);
    while ( true )
    {
        float wert = getUSDistance(S4);
        displayBigStringAt(5, 120, "Abstand in cm:");
        displayBigStringAt(5, 100, "%f", wert);
        sleep ( 100 );
        eraseDisplay();
    }
}
```

12. Gyrosensor abfragen

An Port S2 wird ein Gyrosensor angemeldet. Dieser muss zwingend (!) zurückgesetzt werden mit sensorReset. Eine Fließkommavariable ‚drehwinkel' wird mit ‚0' initialisiert; diese soll die aktuellen Messwerte des Gyrosensors abspeichern. Als nächstes werden die beiden großen Motoren so angesteuert, dass sich der Roboter langsam dreht.

Die darauffolgende while-Schleife wird solange immer wieder durchlaufen, bis der Drehwinkel bei mindestens 90 Grad liegt. Ist er kleiner als 90 Grad, wird der aktuelle Messwert auf dem Display ausgegeben und die Schleife wird erneut durchlaufen. Ist die gewünschte Drehung erreicht, werden die Motoren gestoppt.

```
task main(){
    SensorType[S2] = sensorEV3_Gyro;
    sensorReset(S2);
    float drehwinkel = 0;

    setMotor(motorC, -10);
    setMotor(motorB, 10);

    while (drehwinkel<90.0)
    {
        drehwinkel = getGyroDegrees(S2);

        displayBigStringAt(5, 120, "Drehwinkel");
        displayBigStringAt(5, 100, "%f", drehwinkel);
        sleep (10);
        eraseDisplay();
    }
    stopAllMotors();
}
```

---

13. Infrarotsensor abfragen

Im Beispiel steckt der Infrarotsensor in S4. Um sicherzustellen, dass sich der Sensor in „cm" und nicht in „zoll" misst, führt man einen sensorReset auf Port S4 aus. Der Abstand wird als Fließkommazahl (float) ermittelt. Der Sensorwert kann nicht als Abstand in cm interpretiert werden, er misst nur einen ‚relativen Abstand'. Dabei bedeutet ein Prozentwert von 100, dass der Sensor nichts sieht. Je kleiner der Prozentwert, desto näher das Objekt.

```
task main(){
    SensorType[S4] = sensorEV3_IRSensor;
    sensorReset(S4);
    while ( true )
    {
        int prozentwert = getIRDistance(S4);
        displayBigStringAt(5, 120, "Relativer Abstand:");
        displayBigStringAt(5, 100, "%d", prozentwert);
        sleep ( 100 );
        eraseDisplay();
    }
}
```

Der Farbsensor

ist auf den ersten Blick recht komplex, weil er 6 verschiedene Betriebsmodi besitzt. Es sind aber nur 3 Modi wichtig.

Die verschiedenen Modi des Farbsensors

Von diesen 6 Möglichkeiten sind aber lediglich 3 Modi relevant, hier im Überblick:

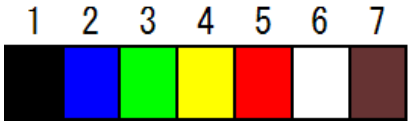
```
// die LEDs für weißes Licht am Sensor leuchten, zum Farben messen
SensorMode[S3] = modeEV3Color_Color;


// die rote LED am Sensor leuchten,              zum Helligkeit messen
SensorMode[S3] = modeEV3Color_Reflected;


// die blaue LED am Sensor leuchtet schwach,     zum Umgebungs-Helligkeit messen
SensorMode[S3] = modeEV3Color_Ambient;
```

(Weiterhin gibt es noch `modeEV3Color_ReflectedRaw`, `modeEV3Color_RGB_Raw`, eher unwichtig)

---

14. Farben als feste Werte messen

Mit der Funktion `getColorName`() wird die gemessene Farbe einem festen Farbwert zugeordnet. Diesen Farbwert kann man mit einer Mehrfachverzweigung abfragen und unterscheiden.




```
task main(){
    SensorType[S3] = sensorEV3_Color;
    sensorReset(S3);
    SensorMode[S3] = modeEV3Color_Color;

    while (true){
        switch(getColorName(S3))
        {
            case colorBlack:    displayBigStringAt(4, 80, "schwarz"); break;
            case colorBlue:     displayBigStringAt(4, 80, "blau");    break;
            case colorGreen:    displayBigStringAt(4, 80, "gruen");   break;
            case colorYellow:   displayBigStringAt(4, 80, "gelb");    break;
            case colorRed:      displayBigStringAt(4, 80, "rot");     break;
            case colorWhite:    displayBigStringAt(4, 80, "weiss");   break;
            case colorBrown:    displayBigStringAt(4, 80, "braun");   break;
            default:            displayBigStringAt(4, 80, "unbekannt");
        }
        sleep(20);
        eraseDisplay();
    }
}
```

15. Regenbogenfarben messen: der HUE-Wert

Jeder Farbe des Regenbogenspektrums wird ein bestimmter Zahlenwert zugeordnet. Diesen Zahlenwert bezeichnet man als HUE-Wert. Der Farbsensor kann diese HUE-Werte bestimmen und an das Programm zurückgeben.


```
task main(){
    SensorType[S3] = sensorEV3_Color;
    sensorReset(S3);
    SensorMode[S3] =  modeEV3Color_Color;
    while (true)
    {
        long farbwert = getColorHue(S3);
        displayBigStringAt(5, 120, "Farbwert");
        displayBigStringAt(5, 100, "%ld", farbwert);
        sleep (50);
        eraseDisplay();
    }
}
```

---

16. RGB-Farben messen: drei unabhängige Farbwerte

Zur den jeweiligen Farbbestandteilen Rot, Grün und Blau werden die Anteile gemessen. Dabei sollte man beachten, dass die RGB-Farben folgendermaßen zusammengesetzt sind:

Spektralfarbe		Rot	Grün	Blau
Rot		Ja	-	-
Gelb		Ja	Ja	-
Grün		-	Ja	-
Türkis		-	Ja	Ja
Blau		-	-	Ja
Magenta		Ja	-	Ja
Weiss		Ja	Ja	Ja

```
task main(){
    SensorType[S3] = sensorEV3_Color;
    sensorReset(S3);
    SensorMode[S3] = modeEV3Color_Color;
        long rotWert, gruenWert, blauWert;

    while (true){
        getColorRGB(S3, rotWert, gruenWert, blauWert);
        displayBigStringAt(5, 120, "Rot:");
        displayBigStringAt(5, 100, "%ld", rotWert);

        displayBigStringAt(5, 80, "Gruen:");
        displayBigStringAt(5, 60, "%ld", gruenWert);

        displayBigStringAt(5, 40, "Blau:");
        displayBigStringAt(5, 20, "%ld", blauWert);
        sleep (50);
        eraseDisplay();
    }
}
```

17. Helligkeitswerte messen

Bei der Helligkeitsmessung im Reflected-Modus wird nur die rote LED des Farbsensors angeschaltet. So lässt sich einfach auch optisch überprüfen, ob der Sensor korrekt programmiert wurde. Der Wertebereich der wiedergegebenen Helligkeitswerte reicht von 0 bis 100. Je heller die Messung, desto größer der Messwert.

```
task main(){
    SensorType[S3] = sensorEV3_Color;
    sensorReset(S3);
    SensorMode[S3] = modeEV3Color_Reflected;

    while (true){
        short helligkeit = getColorReflected(S3);
        displayBigStringAt(5, 120, "Helligkeit");
        displayBigStringAt(5, 100, "%d", helligkeit);
        sleep(20);
        eraseDisplay();
    }
}
```

---

18. Umgebungslicht messen

Der Ambient-Modus funktioniert wie der Reflected-Modus, allerdings mit dem Unterschied, dass **die blaue LED des Sensors schwach leuchtet**. So lässt sich einfach auch optisch überprüfen, ob der Sensor korrekt programmiert wurde. Es wird die Helligkeit des Umgebungslichtes gemessen. Der Wertebereich der Helligkeitswerte reicht von 0 bis 100. Je heller das Umgebungslicht, desto größer der Messwert.

```
task main(){
    SensorType[S3] = sensorEV3_Color;
    sensorReset(S3);
    SensorMode[S3] = modeEV3Color_Ambient;

    while (true){
        short umgebungslicht = getColorAmbient(S3);
        displayBigStringAt(5, 120, "Umgebungslicht");
          displayBigStringAt(5, 100, "%d", umgebungslicht);
        sleep(20);
        eraseDisplay();
    }
}
```

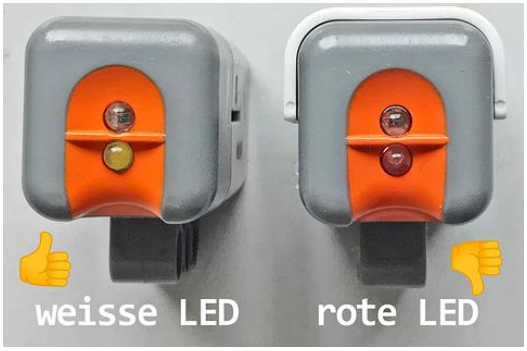
---

19. Töne erzeugen

```
task main()
{
    clearSounds();                  //löscht den Sound-Abspielpuffer
    for (int i=0; i<=10; i++){      //Zählschleife zählt von 0 bis 10
        setSoundVolume(i*10);       //Lautstärke wird gesetzt auf 0,10,20,30,40...
        playTone(440, 5);           //Ton mit 440 Hz, 50 Millisek. Länge wird gestartet
        while (bSoundActive){};     //Warten, bis der Ton fertig abgespielt ist
  }
}
```

NXT-Sensoren

20. NXT-Lichtsensor Reflectance Modus.



Es gibt von den NXT-Lichtsensoren zwei verschiedene: Die eine Variante leuchtet im aktiven Reflectance-Modus mit einer roten LED (nicht so gut), die andere Variante besitzt eine weisse LED (besser). Aufpassen muss man in jedem Fall darauf, dass die beiden Sensorvarianten unterschiedliche Helligkeitswerte erzeugen!

```
task main(){

    SensorType[S1] = sensorLightActive;
    sensorReset(S1);
    SensorMode[S1] = modePercentage;

    while (true){
        int wert = SensorValue[S1];
        displayBigStringAt(10, 120, "Helligk.: %d", wert);
        sleep ( 100 );
        eraseDisplay();
    }
}
```

---

21. NXT-Ultraschall im cm-Modus

Man sollte zwischen zwei Messungen des Ultraschallsensors etwas mehr als 100 Millisekunden verstreichen lassen, damit man nicht mit eventuellen Echos falsche Messwerte erzeugt.

```
task main(){

    SensorType[S1] = sensorSONAR;
    sensorReset(S1);

    while (true){
        int wert = SensorValue[S1];
        displayBigStringAt(10, 120, "Distanz cm: %d", wert);
        sleep ( 500 );
        eraseDisplay();
    }
}
```

---

22. NXT-Soundsensor im DB-Modus

```
task main(){

    SensorType[S1] = sensorSoundDB;
    sensorReset(S1);

    while (true){
        int wert = SensorValue[S1];
        displayBigStringAt(10, 120, "Sound DB: %d", wert);
        sleep ( 500 );
        eraseDisplay();
    }
}
```

Inhalt