fischertechnik

Tutorial ROBO Pro Coding

STEM Coding RoboMission

2024.12



Inhaltsverzeichnis

ROBO Pro Coding und TXT 4.0 Controller 2
Tutorials STEM Coding RoboMission 3
★ Encoder-Motor und Servo 4
★ Optische Farbsensoren 7
★ Schnittstellen-Test 8
★★ Python Module, Funktionen und Importe 9
★★ Motorsteuerung als Bibliothek [RoboMission_motor_servo] 10
★★ START-Button und TXT 4.0 GUI [RoboMission_display_start] 12
★★ Linienfolger mit optischen Farbsensoren [RoboMission_line] 15
★★ Python-Code mit Blockly 18
★★ RGB-Farbsensor als Bibliothek [RoboMission_rgbw_hsv] 19
★★ Messung der Akku-Spannung [test_battery] 21
★★ Vorlage für eigene Projekte [RoboMission_template] 22
★★★ Linienfolger mit PID-Regler [RoboMission_linepid] 23
Ausblick 25

ROBO Pro Coding und TXT 4.0 Controller

Die Software ROBO Pro Coding von fischertechnik bietet in ihrer mehrsprachigen Programmierumgebung neben der Möglichkeit zur graphischen Programmierung mit **Blockly** eine textbasierte über **Python3** beim TXT 4.0 Controller.

ROBO Pro Coding ist für die Betriebssysteme **Windows**, **Mac**, **Linux**, **Android** und **iOS** verfügbar:

https://www.fischertechnik.de/de-de/apps-und-software#apps

Die Dokumentation für **Blockly** in ROBO Pro Coding kann unter dem folgenden Link eingesehen werden:

https://doc.fischertechnik-cloud.com/de/blockly

Die Dokumentation des fischertechnik TXT 4.0 Controllers ist hier zu finden:

https://www.fischertechnik.de/txt40controller



Wenn Du Fragen zum Inhalt dieses Dokumentes oder Tutorial hast, wende Dich bitte an diese E-Mail-Adresse:

fischertechnik-technik@fischer.de

Tutorials STEM Coding RoboMission

In diesem Dokument werden Beispiele beschrieben, wie die Sensoren und Aktoren aus dem Robotik-Set **STEM Coding RoboMission** verwendet werden können. Alle Beispiele basieren auf dem Modellvorschlag des Roboters, können jedoch auch für eigene Modelle verwendet werden.

Alle in diesem Dokument relevanten Beispiele können direkt in ROBO Pro Coding aus dem **fischertechnik GitLab** importiert werden. Dazu kann bei *Menü -> Import -> fischertechnik GitLab* der Name des Programms im Suchfeld eingegeben und das entsprechende Programmbeispiel direkt geladen werden.



Beachte bitte, dass dieses Tutorial davon ausgeht, dass Du Dich bereits mit ROBO Pro Coding auskennst und die Grundlagen mit ROBO Pro Coding bekannt sind. Die folgenden für RoboMission relevanten Themen werden in den nächsten Kapiteln behandelt, wobei der Schwierigkeitsgrad wie folgt eingestuft werden kann:

★	einfach
★★	fortgeschritten
★★★	komplex

Um Dich mit den enthaltenen Bauteilen vertraut zu machen und für die ersten Schritte mit ROBO Pro Coding, empfehlen wir Dir diese Videos anzuschauen:

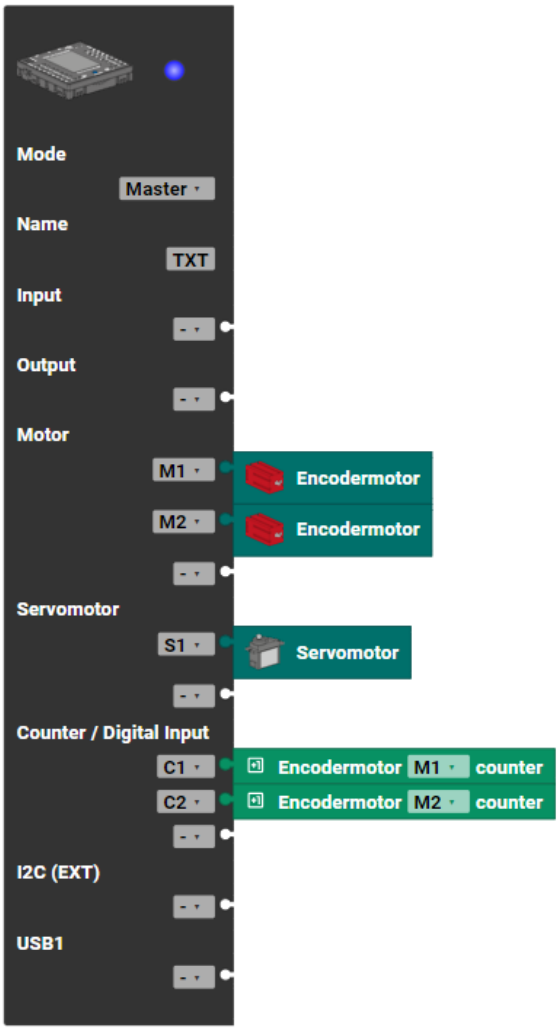
- Erste Schritte mit dem TXT 4.0 Controller
- Der fischertechnik Encoder-Motor
- Der fischertechnik Servo-Motor
- Der fischertechnik Farbsensor
- Der fischertechnik RGB-Farbsensor

★ Encoder-Motor und Servo

An einem TXT 4.0 Controller können gleichzeitig maximal 4 Encoder-Motoren (M1-M4, C1-C4) und maximal 3 Servo-Motoren (S1-S3) angeschlossen werden.





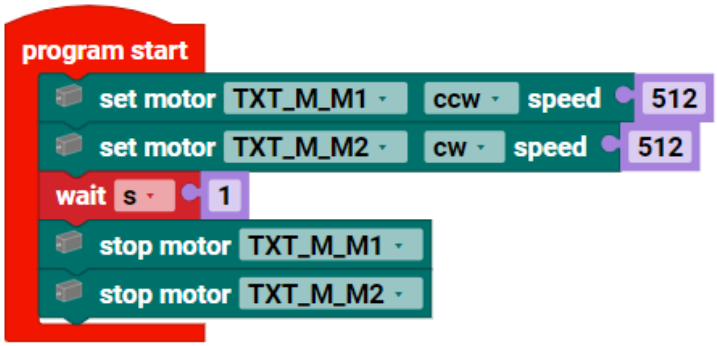


Dazu werden zuerst die entsprechenden Motoren in der Controller-Konfiguration in ROBO Pro Coding den Ausgängen zugewiesen und können dann im Hauptprogramm ausgewählt werden.

Direkte Motorsteuerung

Im einfachsten Fall können die beiden Encoder-Motoren ohne die Verwendung der Encoder gesteuert werden, indem die Drehrichtung - im Uhrzeigersinn (CW) oder gegen den Uhrzeigersinn (CCW) - und der Geschwindigkeitswert (0 ... 512) für die Pulsweitenmodulation (PWM) direkt gesetzt werden.

Das folgende Programm schaltet die beiden Motoren für 1 Sekunde ein und stoppt diese nach Ablauf der Wartezeit.

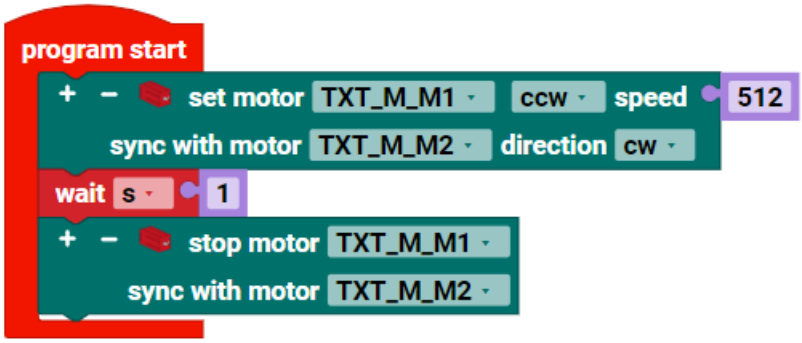


Nachteil der direkten Steuerung ist, dass die beiden Motoren miteinander nicht synchronisiert werden, d.h. ein Motor wird immer zuerst und der andere kurz danach ein- und ausgeschaltet. Außerdem beeinflussen äußere Einwirkungen auf die Motoren und die unterschiedliche Reibung bei den Rädern das Verhalten des Roboters negativ. Eine präzise Geradeausfahrt des Roboters ist damit nicht möglich.

Erweiterte Motorsteuerung mit Synchronisierung der Motoren

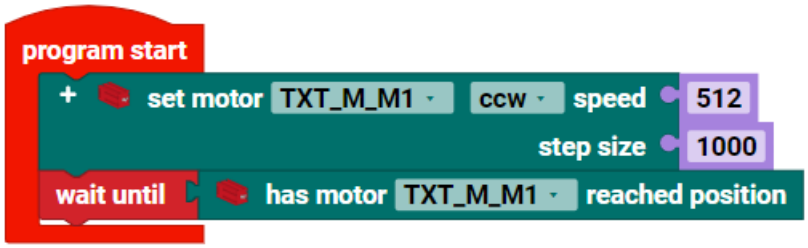
In der Firmware des TXT 4.0 Controllers ist eine erweiterte Motorsteuerung integriert, die basierend auf den Encoder-Signalen bereits eine Synchron-Steuerung durchführt.

Mit den folgenden Blöcken werden die beiden Motoren M1 und M2 miteinander synchron verfahren und der Synchronlauf wird dann nach 1 Sekunde wieder gestoppt. Wird der eine Motor in der Zeit blockiert, so folgt der andere Motor dem ersten Motor. Synchronisierung ist z.B. für die Geradeausfahrt des Roboters relevant.

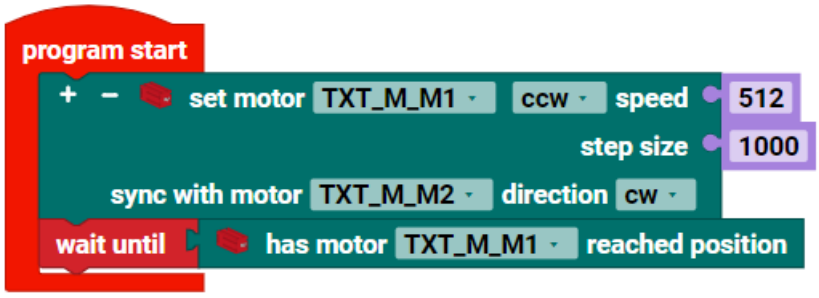


Erweiterte Motorsteuerung mit Synchronisierung und mit Vorgabe der Distanz

Zusätzlich zum Synchronlauf kann eine Distanz (Anzahl der Impulse beim Encoder) vorgegeben werden. Dies kann dann entweder nur für einen einzelnen Motor



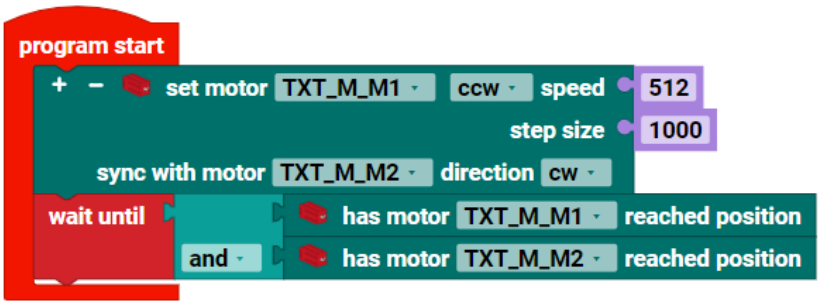
oder für mehrere Motoren synchron durchgeführt werden.



Der aktuelle Status aus der erweiterten Motorsteuerung kann mit dem Block „**has motor … reached position**“ abgefragt werden. Mit dem Status signalisiert die Motorsteuerung, dass die Position erreicht wurde, damit die Programmausführung fortgesetzt werden kann.

Nach Erreichen der Distanz, z. B. nach 1000 Impulsen wird der Motor dann automatisch gestoppt.

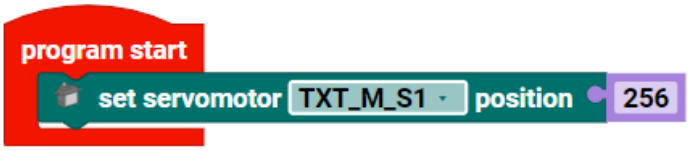
Im Normalfall reicht es den Zustand eines Motors abzufragen. Es kann jedoch auch der Zustand von allen Motoren abgefragt werden, um sicher zu gehen, dass bei Kabelbruch eines Encoders die Steuerung alle Motoren berücksichtigt.



Steuerung Servo-Motor

Im Gegensatz zum Motor, der meistens für den Antrieb des Roboters verwendet wird, kann der Servo-Motor dazu verwendet werden absolute Winkel von ca. -90 ... 90 Grad direkt anzufahren. Die Regelung und die Sensorik sind im Servo-Motor bereits enthalten. Der Servo-Motor eignet sich z. B. gut für Greifvorrichtungen des Roboters, überall dort, wo Drehbewegungen benötigt werden.

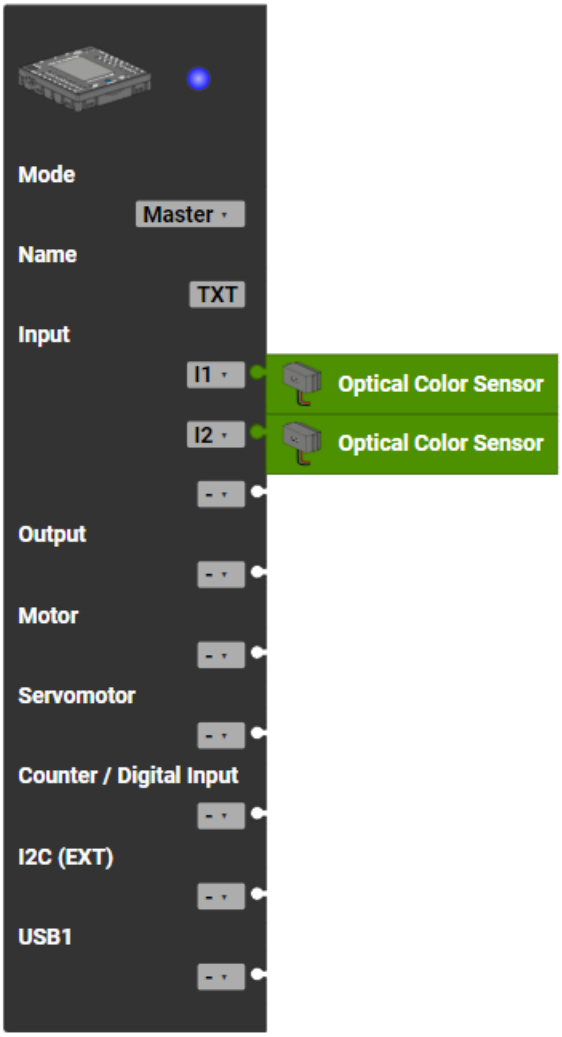
Der Servo-Motor fährt mit Vorgabe eines PWM-Wertes einen bestimmten Winkel an. Die Vorgabe ist absolut. Die Mittelstellung des Servos liegt bei ca. 256 und die beiden Grenzen bei den Werten 0 und 512. Die genannten absoluten Werte unterscheiden sich je nach Servo geringfügig. Sie sollten deswegen am besten für jeden Servo einzeln kalibriert werden. Dies kann z.B. durch Konstanten oder Variablen im Programm erfolgen.



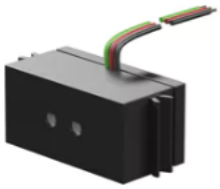
Außerdem zu beachten ist, dass der Servo keine Rückkopplung bezogen auf den Controller hat. Die vorgegebene Position über den PWM-Wert versucht der Servo zu erreichen. Wenn die Zielposition des Servos dauerhaft mechanisch blockiert wäre, kann beim Servo ein sehr hoher Strom fließen. Dieser könnte dann den Servo beschädigen. Dies sollte bei der Benutzung des Servo unbedingt verhindert werden.

★ Optische Farbsensoren

An einem TXT 4.0 Controller können digitale oder analoge Sensoren an den Universaleingängen (I1-I8) angeschlossen werden.



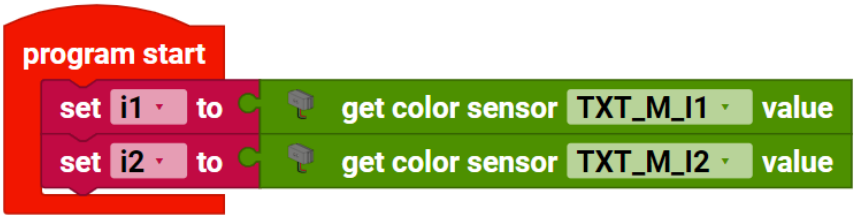
Dazu werden zuerst die entsprechenden Sensoren in der Controller-Konfiguration in ROBO Pro Coding den Eingängen zugewiesen und können dann im Hauptprogramm ausgewählt werden.



2x

Im Set STEM Coding RoboMission sind zwei **optische Farbsensoren** enthalten, die für die Erkennung von einfachen Hell-Dunkel-Übergängen auf dem Parcours z.B. schwarze Spur verwendet werden können. Mit dem optischen Sensor können auch Schwarz-Weiß-Übergänge z.B. Grauwerte erkannt werden. Dies kann dazu genutzt werden, um den Roboter präzise an der Linie auszurichten oder einen Linienfolger zu implementieren.

Mit den folgenden Blöcken kann der analoge Wert der beiden Sensoren ausgelesen werden: Der analoge Wert entspricht in mV der gemessenen Spannung an den beiden Eingängen I1 und I2.

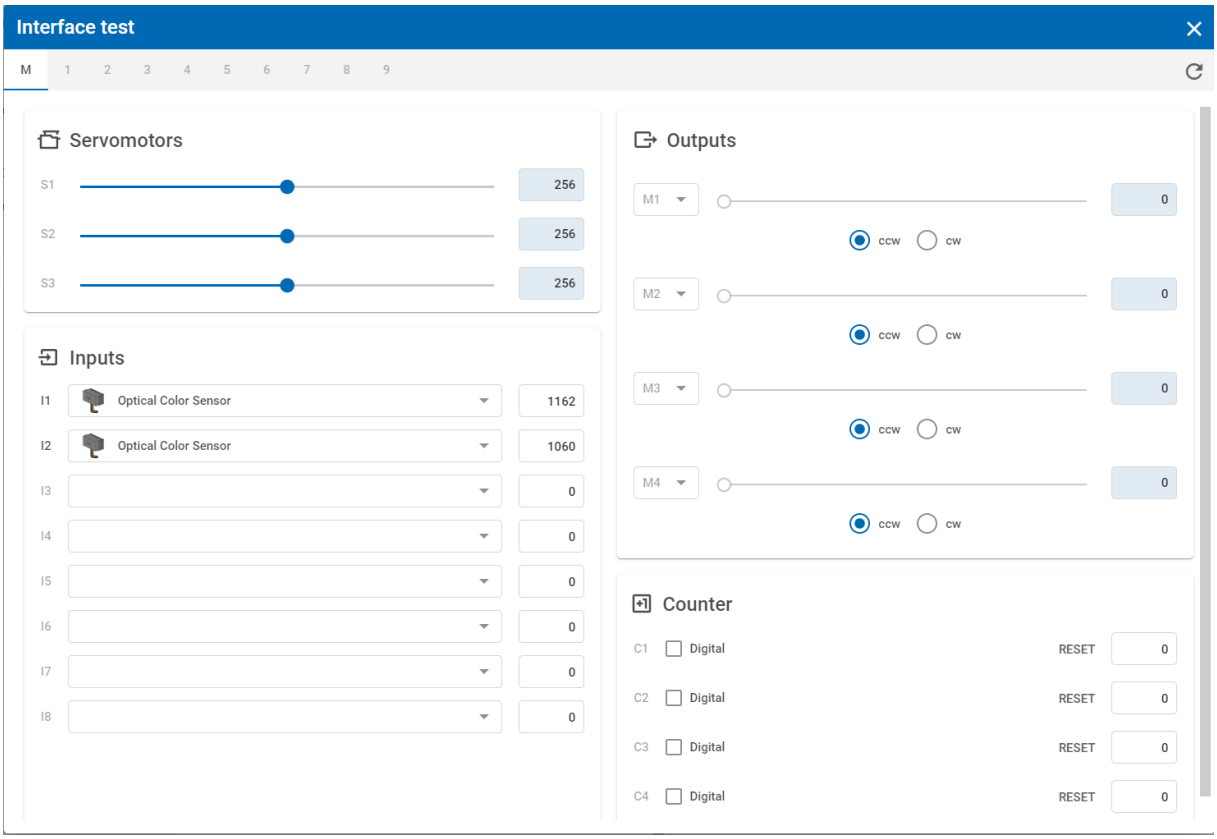


Bitte beachte, dass die RGB-Farbsensoren nicht an die Universaleingänge angeschlossen werden, sondern an den I2C-Bus (EXT1 oder EXT2 Anschluss) des TXT 4.0 Controllers. Der RGB-Farbsensor ist ebenfalls im Set enthalten und wird in den Kapiteln weiter hinten behandelt.

★ Schnittstellen-Test

Die Aktoren und Sensoren können auch ohne ein Programm schnell getestet werden. Dazu kann der Schnittstellen-Test in ROBO Pro Coding verwendet werden, wenn eine Verbindung zum TXT 4.0 Controller aufgebaut wird.





Bei den Servo-Motoren S1-S3 kann über den Slider der Wert gesetzt werden.

Bei den Motoren M1-M4 kann die Geschwindigkeit und die Drehrichtung eines Motors gesetzt werden.

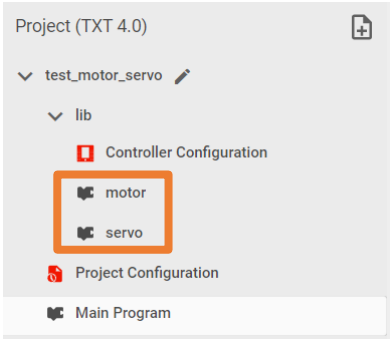
Bei den Universaleingängen I1-I8 z. B. bei den optischen Sensoren wird der Wert angezeigt. Die einzelnen Universaleingänge müssen davor über die Auswahlbox konfiguriert werden.

Der aktuelle Counter-Wert C1-C4 bei den Encoder-Eingängen wird ebenfalls als Wert angezeigt.

Bitte beachte, dass nicht gleichzeitig der Schnittstellen-Test und ein Programm gestartet werden kann.

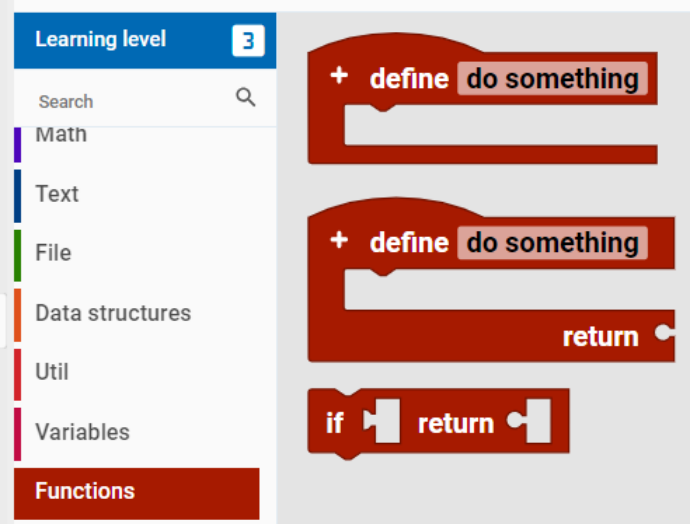
★★ Python Module, Funktionen und Importe

Im Programm-Beispiel **RoboMission_motor_servo** werden Funktionen für Motoren und Servo-Motoren in zusätzlichen Dateien definiert, die in der Struktur unter „lib“ zu finden sind. Der Vorteil von eigenen Python Modulen ist, dass dadurch der Code übersichtlicher wird und in anderen Projekten als Bibliothek wiederverwendet werden kann.

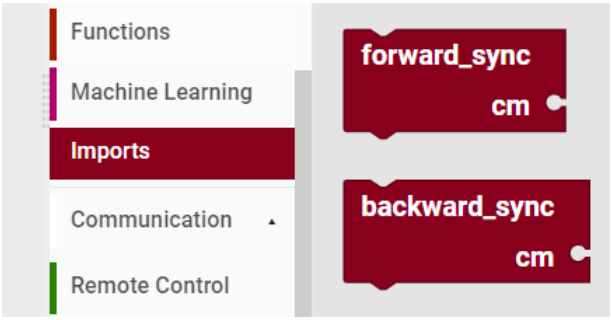


Funktionen, die lokal im Modul oder im Hauptprogramm verfügbar sind, werden mit „**Functions**“ definiert. Es können sowohl Funktionen mit Parametern als auch mit Rückgabewerten definiert werden.

Die Parameter werden mit (+) hinzugefügt und entsprechend definiert.



Wenn die Funktionen in anderen Dateien verwendet werden sollen, so können diese über „**Imports**“ hinzugefügt werden. Dabei wechselst Du in die Datei, in der die Funktion verwendet werden soll und wählst diese Funktion über „Imports“ entsprechend aus einem anderen Modul aus.

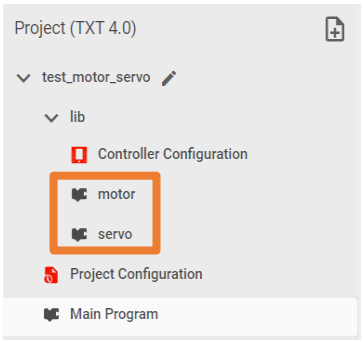


Die lokalen Funktionen „Functions“ und importierte Funktionen „Imports“ haben unterschiedliche Farben. Somit kannst Du diese voneinander im Programmablauf unterscheiden.

Bitte beachte, dass Du keine zirkulären Bezüge zwischen den Modulen definierst. Module dürfen nicht gegenseitig Funktionen importieren, weil sonst die Importe nicht richtig aufgelöst werden können.

★★ Motorsteuerung als Bibliothek [RoboMission_motor_servo]

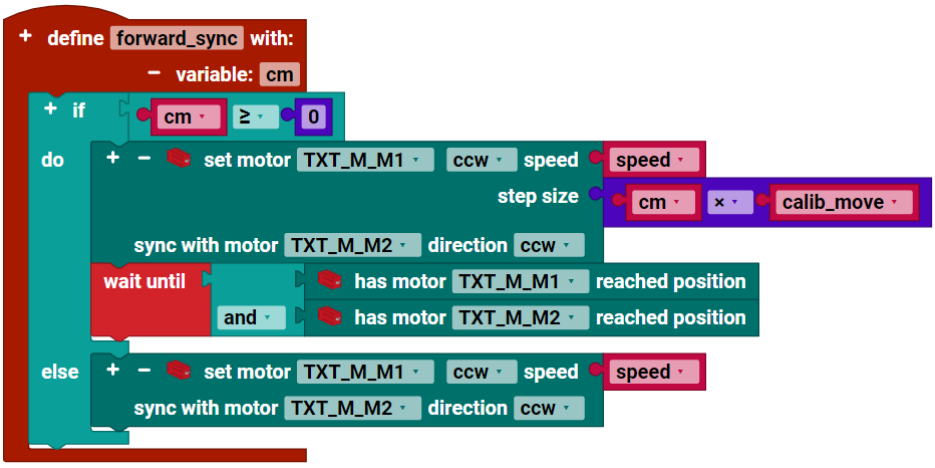
Im Beispiel **RoboMission_motor_servo** wird gezeigt, wie ein (Encoder-)Motor und ein Servo-Motor gesteuert werden können.

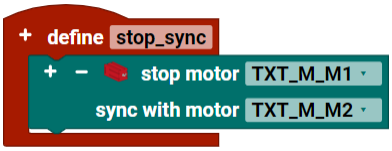


Bitte mache Dich zuerst mit der Funktionsweise von Encoder-Motoren und Servo-Motoren vertraut, bevor Du Dir das Programm anschaust. In den jeweiligen Datenblättern findest Du die technischen Daten zu den Motoren.

Im Modul „**motor**“ werden verschiedene Funktionen definiert, mit denen die Motoren asynchron (ohne die Verwendung von Encodern) oder wie diese synchron (synchronisiert mittels Encoder) gesteuert werden können.

Falls die beiden Motoren **synchron** verfahren werden, so wartet die Steuerung bis beide Motoren eine bestimmte Anzahl an Impulsen erreicht haben. Falls der eine Motor währenddessen blockiert wird, so bleibt der zweite Encoder-Motor so lange stehen, bis der erste Motor die Position auch erreicht hat. Für präzise Bewegungen des Roboters auf dem Parcours müssen die beiden Motoren am besten immer synchronisiert werden. Es kann außerdem auch eine Distanz angegeben werden. Wenn die Distanz endlos sein soll, kann bei der Funktion **forward_sync** ein negativer Parameter, z. B. -1 übergeben werden. Die beiden Motoren verfahren dann endlos synchron bis wieder beide Motoren mit **stop_sync** gestoppt wurden.

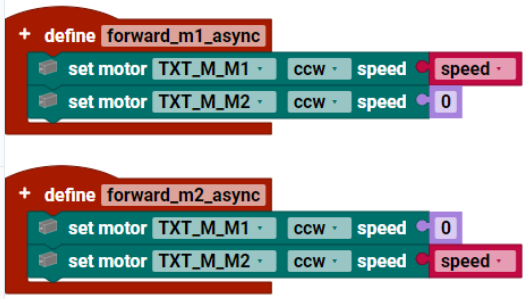




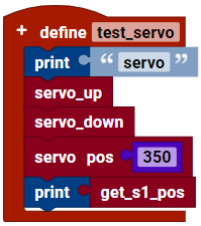
Damit die Bewegung des Roboters bei den Motor-Funktionen z. B. in der Einheit **cm** oder in **Grad** angegeben werden kann, müssen die Impuls-Counter der Encoder mit den entsprechenden Faktoren im Programm umgerechnet werden. Diese Faktoren sind in der Kalibrier-Funktion enthalten. Falls das Modell des Roboters insbesondere der Radabstand und

Raddurchmesser aus STEM Coding RoboMission geändert wird, müssen die Faktoren **calib_move** und **calib_rotation** dann entsprechend an das neue Modell angepasst werden.

Bei einem Linienfolger oder einer komplexen Regelung ist es am besten, wenn die beiden Motoren nicht miteinander synchronisiert werden. Für diese Anwendungsfälle ist das direkte Setzen der Geschwindigkeit über Pulsweitenmodulation (PWM) vorteilhaft, damit die beiden Motoren schnell unabhängig voneinander gesetzt werden können.

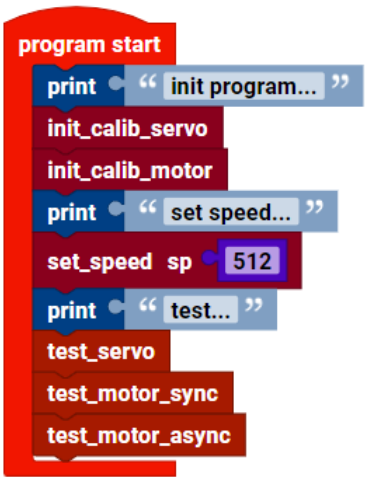


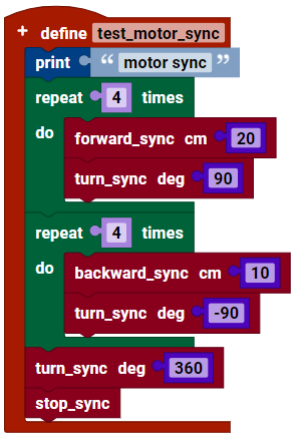
Im Modul „**servo**“ sind Funktionen für den Servo-Motor enthalten. Der Servo-Motor enthält eine eigene Regelung und bekommt eine absolute Position über (PWM: 0...512) vorgegeben, die dieser möglichst schnell zu regeln versucht. Es können teilweise sehr große Ströme kurzzeitig fließen und je nach aktueller Stellung können die Bewegungen dann sehr ruckartig sein. Damit beim Servo-Motor ruckartige Bewegungen vermieden werden, müssen Zwischenpositionen in bestimmten Zeitabständen durch das Programm vorgegeben werden. Dann sind auch langsame Bewegungen beim Servo-Motor möglich.

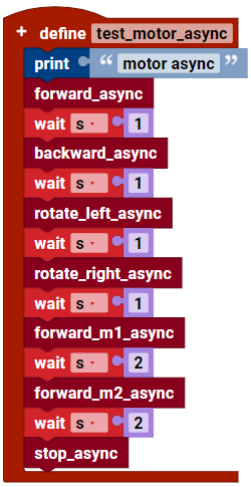


Auch beim Servo-Motor können die Werte für die einzelnen Positionen kalibriert werden. Dies ist erforderlich, weil jeder Servo durch die Toleranzen leicht unterschiedliche Absolutpositionen aufweist.

Das Hauptprogramm in **RoboMission_motor_servo** sieht dann folgendermaßen aus:

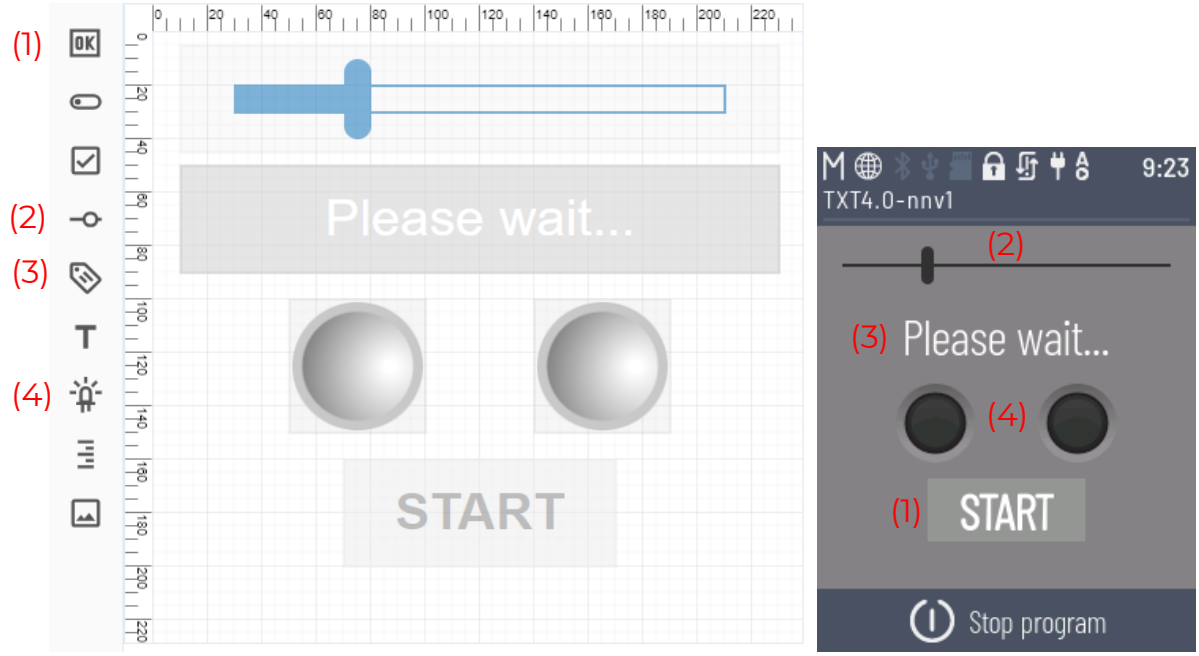






★★ START-Button und TXT 4.0 GUI [RoboMission_display_start]

Im Beispiel **RoboMission_display_start** wird gezeigt, wie eine grafische Oberfläche (GUI) auf dem Display des TXT 4.0 Controllers in ROBO Pro Coding erstellt werden kann. Auf der linken Seitenleiste im GUI-Editor stehen dazu verschiedene Elemente in ROBO Pro Coding zur Verfügung.



In diesem Beispiel werden die folgenden Elemente verwendet:

Button (1):	Starten des Programmablaufs
Slider (2):	Setzen von Variablenwerten
Label (3):	Ausgabe von Text
Statusindikator (4):	Ausgabe von binären Zuständen

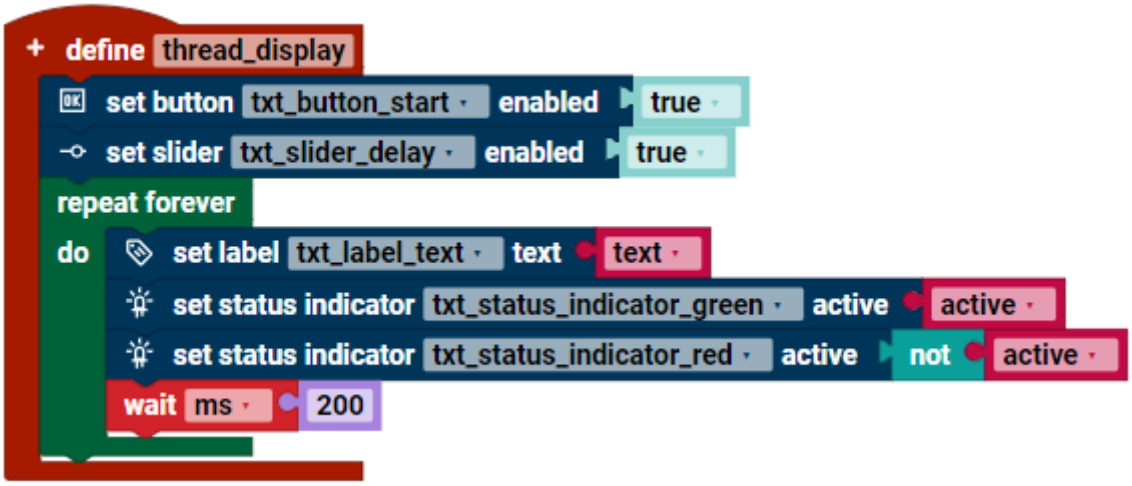
Links ist die Definition im GUI-Editor, rechts ist die Darstellung auf dem TXT 4.0 Display abgebildet.

Die Oberfläche auf dem Display kann beliebig konfiguriert werden. Es lassen sich auch Schriftgröße und Schriftstil bei den Eigenschaften der Elemente einstellen.

Damit über die Oberfläche Aktionen ausgelöst oder Zustände des Programms am Display angezeigt werden können, muss noch der Programmablauf definiert werden. Es wird außerdem empfohlen für die Aktualisierung des Displays einen eigenen Thread (Eine eigene Endlosschleife) vorzusehen. Da das Auge träge ist, sollte das Display maximal alle 200ms ... 1s aktualisiert werden. Außerdem wird durch einen

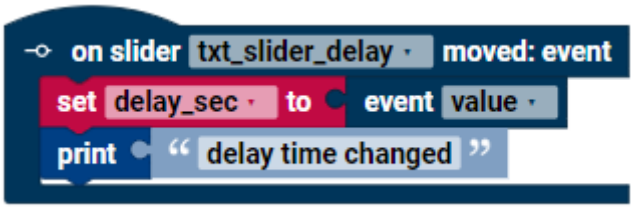
eigenen Thread für das Display die Hauptschleife entlastet, in der häufig der Roboter gesteuert wird und die Reaktionszeit sehr kurz sein muss.

Das Laden der Oberfläche dauert im Regelfall einige Sekunden. Damit bei der Initialisierung keine Elemente betätigt werden können, können alle Elemente, die eine Aktion auslösen, am Anfang in den Eigenschaften deaktiviert werden. Bei der Initialisierung des Display-Threads können dann die entsprechenden Elemente gezielt mit **set … enabled** aktiviert werden.



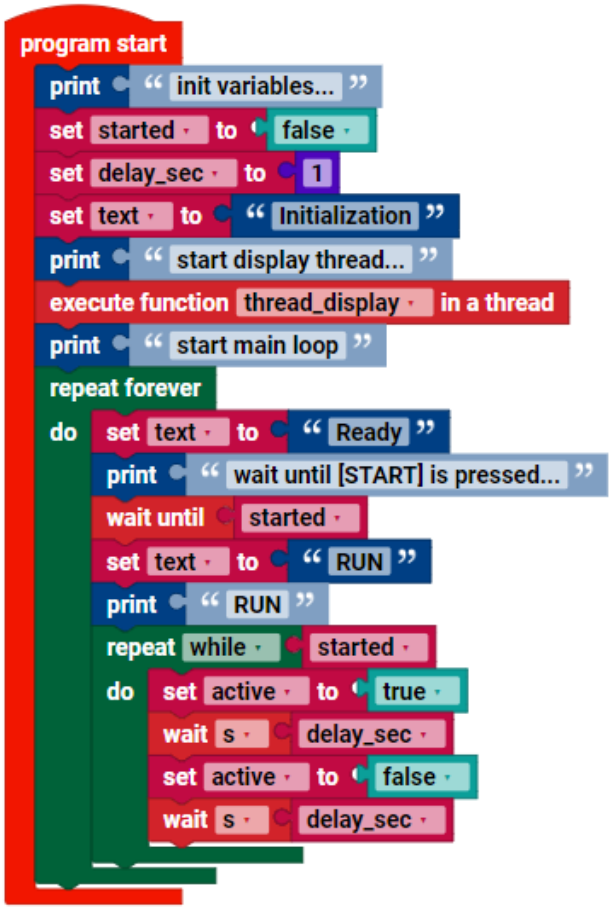
Für alle Aktionen (z. B. „**on button … clicked**“ und „**on slider … moved**“) müssen dann noch Callback-Funktionen definiert werden, die bei den entsprechenden Ereignissen Variablen setzen. Diese können dann im Hauptprogramm ausgewertet werden.

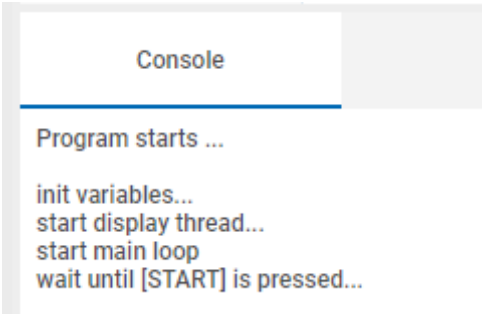




Es wird in den Callback-Funktionen empfohlen, nur kurze und keine blockierenden Funktionen aufzurufen, weil sonst sich die Callback-Funktionen gegenseitig beeinflussen können.

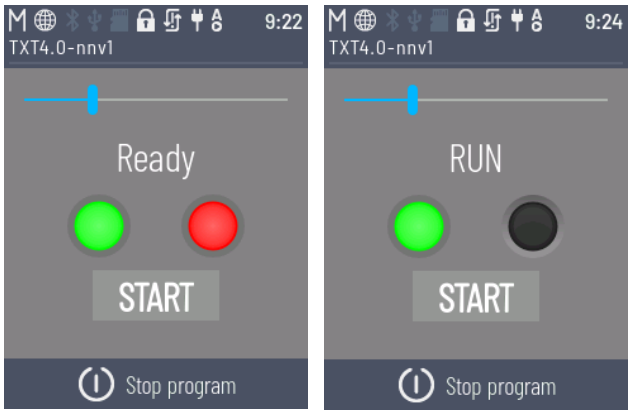
Der Programmablauf in der Hauptschleife kann dann wie folgt aussehen:





Um das Programm besser nachvollziehen zu können, ist es oft ratsam zusätzlich zu den Ausgaben auf dem Display auch Ausgaben mit **print(...)** im Programmablauf vorzusehen. Wenn das Programm in ROBO Pro Coding dann gestartet wird, so können in der Konsole alle Ausgaben mit **print()** bei der Analyse des Programms helfen, wann welche Anweisung im Programm ausgeführt wurde. Bei Regelschleifen mit kurzen Reaktionszeiten von z. B. 10ms sollte jedoch die **print()** Ausgabe eher gemieden werden.

Im Folgenden sind noch die Ausgaben am TXT 4.0 Controller Display gezeigt, wenn das Programm ausgeführt wird. Der Zustand wechselt zuerst von „Please wait ...“ auf „Ready“ und schließlich auf „RUN“.



Mit dem START-Button ist es nun z. B. möglich, dass der Roboter erst beim Drücken auf START den Programmablauf startet.

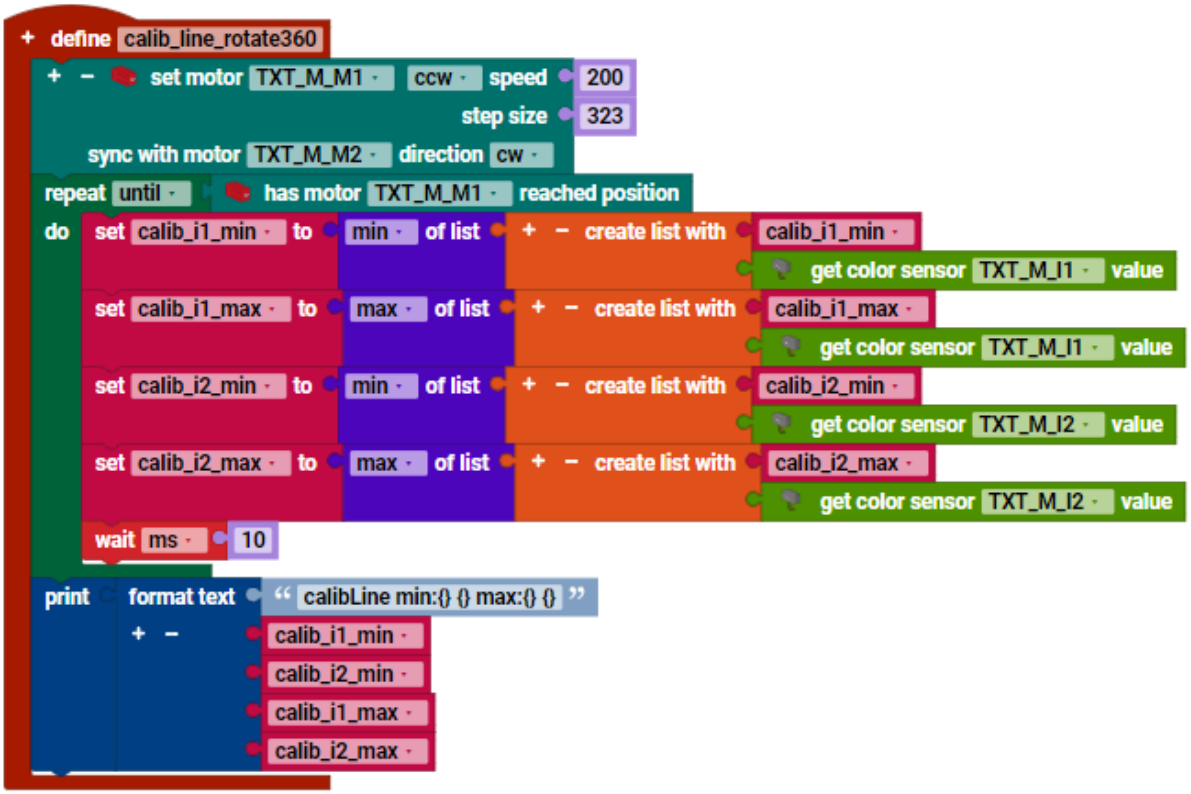
★★ Linienfolger mit optischen Farbsensoren [RoboMission_line]

In diesem Beispiel **RoboMission_line** wird gezeigt, wie ein einfacher Linienfolger mit optischen Farbsensoren an den Universaleingängen realisiert werden kann.

Optische Farbsensoren liefern eine analoge Spannung als Wert in mV je nachdem wie viel Licht vom Parcours zurück reflektiert wird. Weiße Flächen haben einen niedrigen Wert, schwarze Flächen einen hohen Wert. Befindet sich der Sensor an der Kante der Linie, so bewegt sich der ausgelesene Wert des Sensors zwischen den beiden min und max Werten.

Weil jeder optischer Farbsensor etwas unterschiedliche min und max Werte aufweist, ist es vorteilhaft jeden Sensor einzeln zu kalibrieren. Dazu kann z.B. eine 360 Grad Drehung ausgeführt werden und von allen gemessenen Werten der min und max Wert pro Sensor bestimmt und gespeichert werden. Die nun kalibrierten Werte können dann bei der Berechnung weiterverwendet werden.

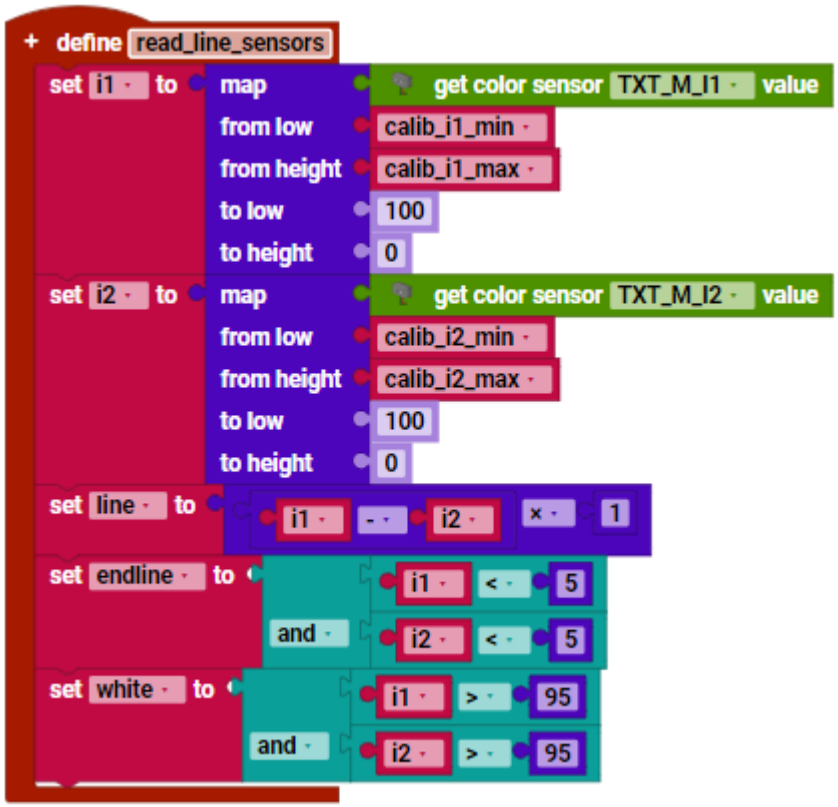
Wenn der Roboter auf einer Linie platziert wird und dann die Funktion **calib_line_rotate360** ausgeführt wird, so werden die min und max Werte pro Sensor kalibriert und in den entsprechenden Variablen gespeichert:



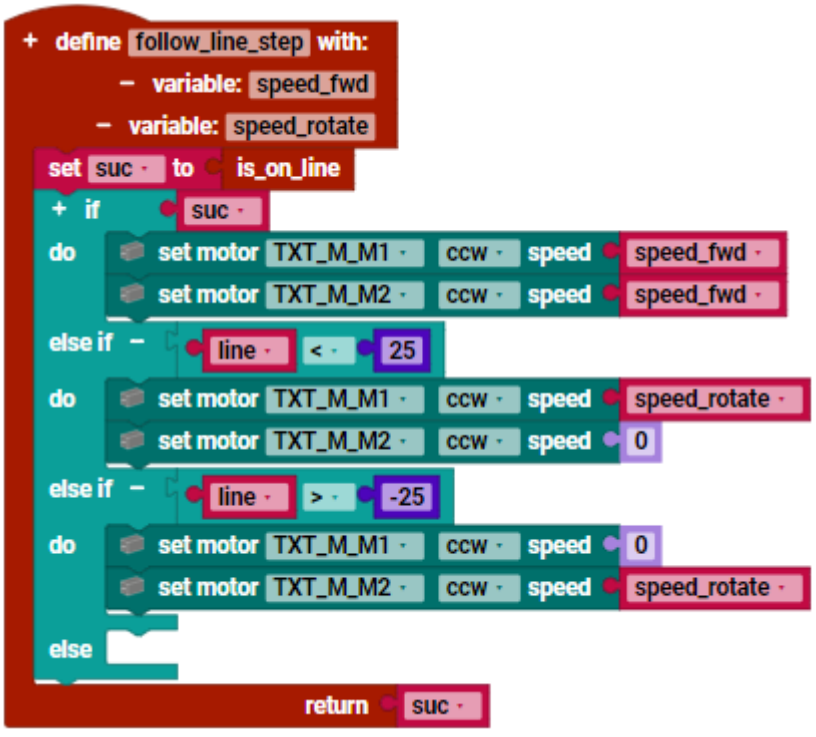
Die mit dieser Funktion kalibrierten Werte können z.B. wie folgt aussehen:

calibLine min:173 171 max:1559 1485

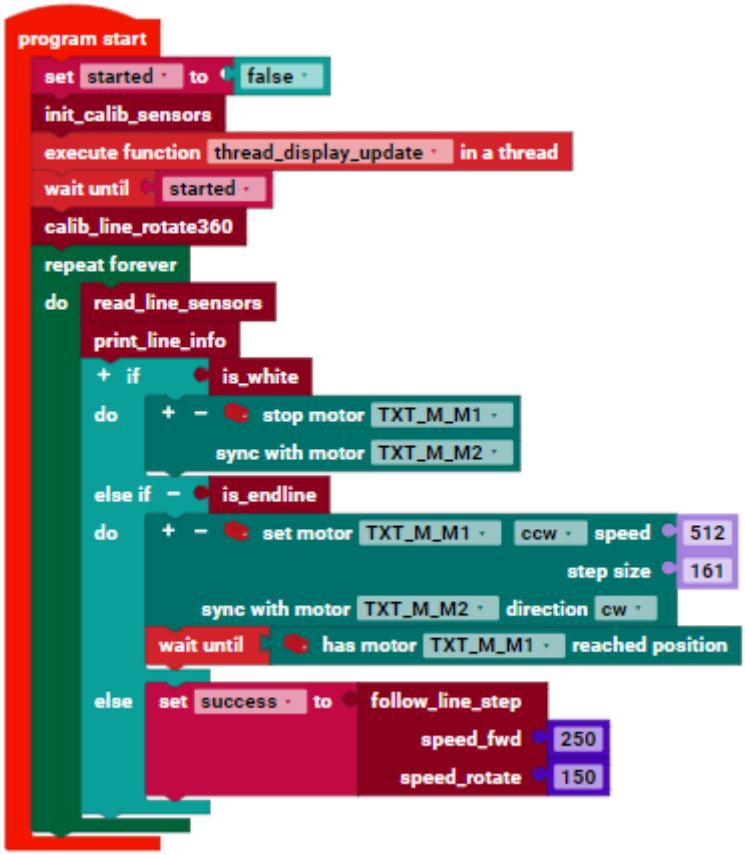
In der Funktion **read_line_sensors** zum Auslesen der Sensoren werden dann die Messwerte normiert, so dass die Variable **line** Werte zwischen -100 ... 0 ... 100 aufweist. Zusätzlich werden mehrere Boolean-Variablen gesetzt, ob ein Linienende erkannt oder eine weiße Fläche detektiert wurde. Diese Zustände können dann dazu verwendet werden, um am Linienende eine 180-Grad-Drehung auszuführen oder den Linienfolger zu stoppen, wenn der Roboter die weiße Fläche passiert.



In einer Endlosschleife wird dann die Funktion **follow_line_step** aufgerufen, die dann die Motoren abhängig von der berechneten Variable line ansteuert. Der Roboter fährt einen Schritt geradeaus, nach links oder nach rechts. Dies passiert in sehr kurzen Zeitabständen.



Die Hauptschleife im Programm kann dann z. B. wie folgt aussehen:



Wird die Geschwindigkeit der Motoren zu hoch eingestellt, so verliert der Roboter die Linie sehr schnell. Wird die Geschwindigkeit der Motoren zu niedrig eingestellt, so reicht die Kraft der Motoren bei niedriger Akku-Ladung nicht aus oder der Roboter bewegt sich sehr langsam auf der Linie.

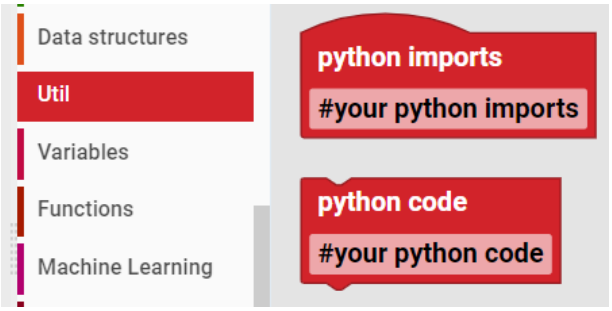
Bei einem Linienfolger sind viele unterschiedliche Verfahren denkbar. So kann z.B. statt der Linie auch einer Linienkante gefolgt werden. Dann muss die Funktion **follow_line_step** entsprechend angepasst werden.

Der Linienfolger kann mit einem PID-Regler noch weiter verbessert werden, damit der Roboter auch bei einer sehr hohen Geschwindigkeit die Linie nicht verliert. Die Implementierung eines solchen PID-Reglers erfordert Kenntnisse über Regelungstechnik, die an dieser Stelle im Tutorial nicht weiter betrachtet werden sollen.

★★ Python-Code mit Blockly

Die meisten Funktionen in ROBO Pro Coding können mit Blockly implementiert werden. Es existieren sehr viele Blöcke für unterschiedliche Anwendungsfälle. Es ist sehr komfortabel die Blöcke zu verwenden, weil die benötigten Anweisungen in Python mit ROBO Pro Coding automatisch generiert werden.

Manchmal müssen jedoch Python-Befehle in Blockly-Programme integriert werden, weil es den entsprechenden Block noch nicht gibt oder weil die Abfolge von Python-Befehlen anders definiert werden soll. Dies ist sehr leicht in einem Blockly-Programm möglich, indem die geforderten Python-Befehle mit den folgenden Python-Blöcken erstellt werden:

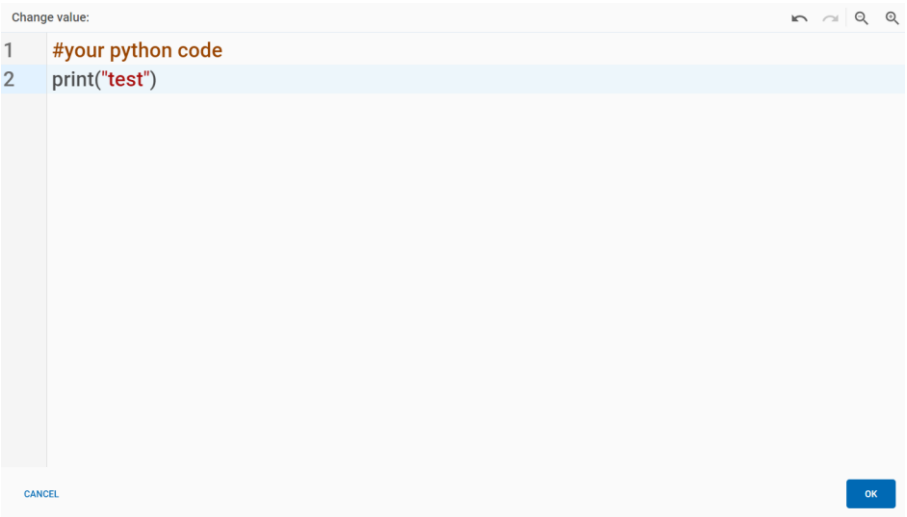


Mit dem Block „**python imports**“ können gezielt Python-Module importiert werden. Der Blockly-Generator setzt den Inhalt an den Anfang der Datei.

Mit dem Block „**python code**“ kann Python-Code an einer beliebigen Stelle im Blockly-Programm definiert werden.

Es öffnet sich jeweils ein Python-Editor, wenn auf den Block geklickt wird. Python-Code kann dann auch über mehrere Zeilen definiert werden.

Besonders geachtet werden muss auf die Einrückung des Quelltextes, da in Python die Einrückung über Leerzeichen oder Tabs die Struktur des Programms wesentlich beeinflusst.

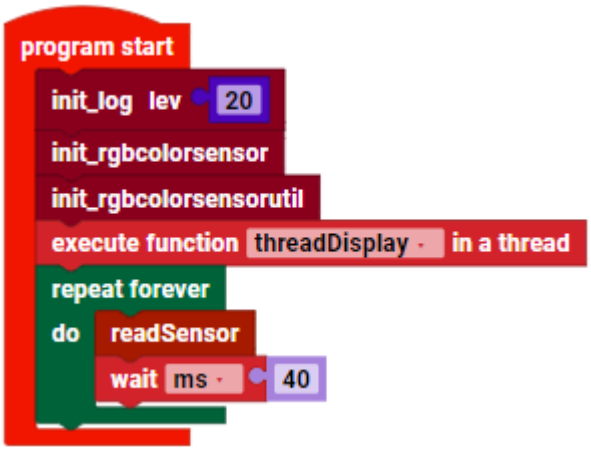


★★ RGB-Farbsensor als Bibliothek [RoboMission_rgbw_hsv]

RGB-Farbsensor wird an den I2C-Bus des TXT 4.0 Controllers angeschlossen. Dazu muss ein RGB-Farbsensor mit dem Anschluss EXT1 oder EXT2 des TXT 4.0 Controllers mittels eines Flachbandkabels verbunden werden.

Der RGB-Farbsensor ist im Gegensatz zu optischen Farbsensoren, die an den Universaleingängen des TXT 4.0 Controllers angeschlossen werden, für die Erkennung von verschieden farbigen Hindernissen gut geeignet, wenn es darum geht, dass diese in einem größeren Abstand erkannt werden sollen. Es können beim RGB-Farbsensor auch mehrere Sensoren verwendet werden, die dann in einer Kette an den Hauptsensor angeschlossen werden.

Im Beispielprogramm **RoboMission_rgbw_hsv** wird gezeigt, wie die Werte des Sensors ausgelesen werden können.



Das Modul **log** enthält Funktionen zum Loggen, wobei das Logging-Level geändert werden kann.

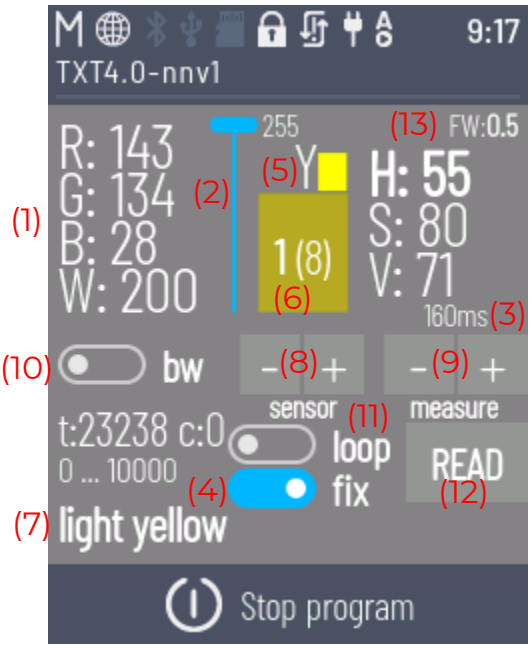
Das Modul **rgbcolorsensor** enthält alle grundlegenden Funktionen zum Auslesen der Register des RGB-Farbsensors. Hier sind alle Funktionen implementiert, die für die Kommunikation zum RGB-Farbsensor erforderlich sind.

Das Modul **rgbcolorsensorutil** enthält weitere Funktionen, um z.B. Farbwerte aus dem RGBW-Modell (Red, Green, Blue, White) in das HSV-Modell (Hue, Saturation, Value) oder als HEX-Werte umzurechnen. Es sind außerdem einige Fallunterscheidungen für die Standard-Farben bereits implementiert.

Die Werte des Sensors mit dem Index 0 (Hauptsensor) können dann z. B. wie folgt ausgelesen werden:



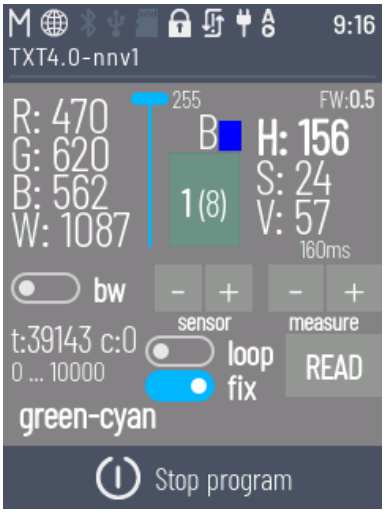
In einem weiteren Beispielprogramm **test_rgbcolorsensor_smbus** wird gezeigt, wie die komplette Schnittstelle des RGB-Farbsensors verwendet werden kann. Im Screenshot sind diverse zusätzliche Schnittstellen zu finden, siehe Screenshot.



(1) Rohwerte des Sensors RGBW: Red/Green/Blue/White
(2) Helligkeit LEDs einstellbar 0...255
(3) Berechnete HSV-Werte: Hue/Saturation/Value
(4) fix: Referenzwert für HSV-Berechnung entweder fest (10000) oder adaptiv mit gemessenen min ... max Werten aus der Historie
(5) Erkannte Block-Farbe R/G/B/Y
(6) Erkannte Farbe als HEX-Wert
(7) Erkannte Farbe als Text allgemein
(8) Sensor 1 (von 8 erkannten Sensoren), wird ausgewählt über (-) und (+) Buttons
(9) Eingestellte Zeit „160ms“ für Messung, wird ausgewählt über (-) und (+) Buttons
(10) bw: Schalter für Schwarz/Weiß-Modus
(11) loop: Endlosmessung, wenn aktiviert
(12) Einzelmessung starten mit „READ“, falls loop deaktiviert
(13) Firmware Version FW: 0.5 des angeschlossenen Sensors







Im Beispiel **test_rgbcolorsensors_smbus** wird gezeigt, wie mehrere RGB-Farbsensoren (Index 0-7) ausgelesen werden und die Werte aller Sensoren gleichzeitig am Display angezeigt werden können.



★★ Messung der Akku-Spannung [test_battery]

Der Ladezustand des Akkus, der am TXT 4.0 Controller angeschlossen ist, kann einen Einfluss darauf haben, mit welchem Strom die Motoren beaufschlagt werden und somit das Verhalten des Roboters beeinflussen. Um bei niedriger Spannung darauf reagieren zu können, kann die Spannung über einen internen Spannungseingang des TXT 4.0 Controllers in Volt ausgelesen werden.

Im Beispiel **test_battery** kann die Spannung und somit der ungefähre Ladezustand mittels einer Python-Funktion ausgelesen werden.



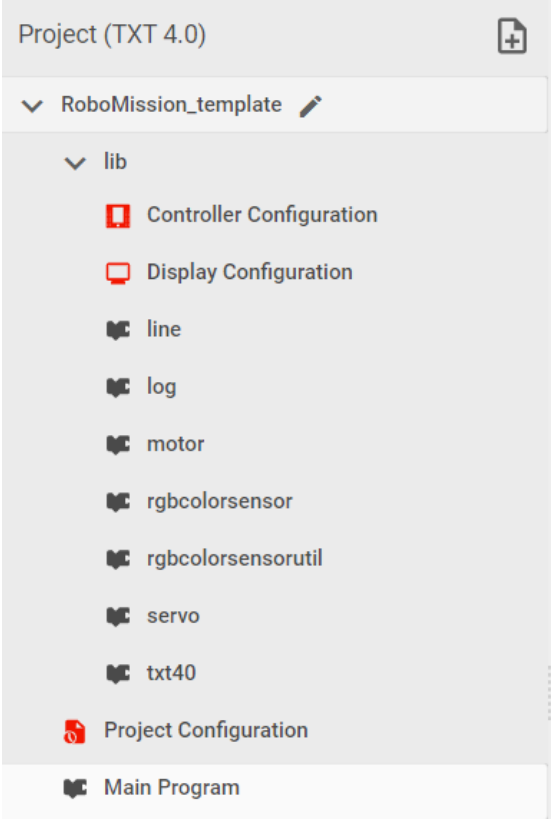
★★ Vorlage für eigene Projekte [RoboMission_template]

Das Beispiel **RoboMission_template** kann dazu verwendet werden, ein eigenes Projekt zu erstellen.

Importiere dieses Projekt und modifiziere dieses nach Deinen eigenen Bedürfnissen.

Die benutzten Funktionen aus diesem Tutorial sind darin als Module enthalten und können im Hauptprogramm oder in anderen neu erstellten Modulen verwendet werden.

Baue Dir dein eigenes Template und nutze dieses für Deine eigenen Projekte.



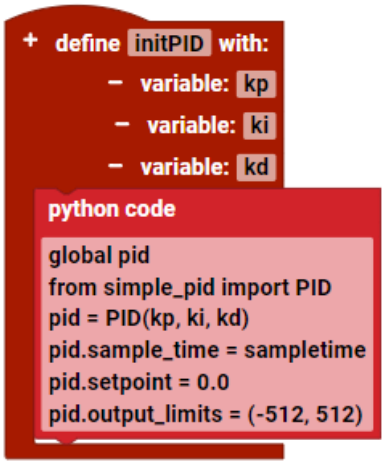
★★★ Linienfolger mit PID-Regler [RoboMission_linepid]

Im Beispiel **RoboMission_linepid** wird exemplarisch gezeigt, wie beim Linienfolger ein PID-Regler realisiert werden kann.

In Python existiert dazu z.B. die Bibliothek **simple-pid.** Diese externe Bibliothek ist in Version 1.0.0 bereits in der Firmware des TXT 4.0 Controllers 3.1.8 enthalten und kann direkt verwendet werden. Die Dokumentation zu simple-pid ist unter dem folgenden Link zu finden:

https://github.com/m-lundberg/simple-pid/tree/v1.0.0

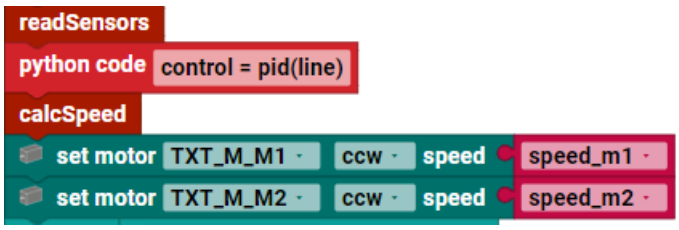
Zuerst wird der PID-Regler initialisiert, wobei die Parameter **Kp**, **Ki** und **Kd** eingestellt werden müssen. Die Einstellung der Konstanten muss experimentell ermittelt werden und kann sehr aufwändig sein. Für die Grundlagen zu PID und die richtige Auswahl der Parameter wird auf die entsprechende externe Literatur verwiesen.



Bedeutung der Parameter:

- **Proportionalverhalten** Kp
- **Integralverhalten** Ki
- **Differentialverhalten** Kd

In der Regelschleife werden in jedem Schritt die folgenden Funktionen aufgerufen:

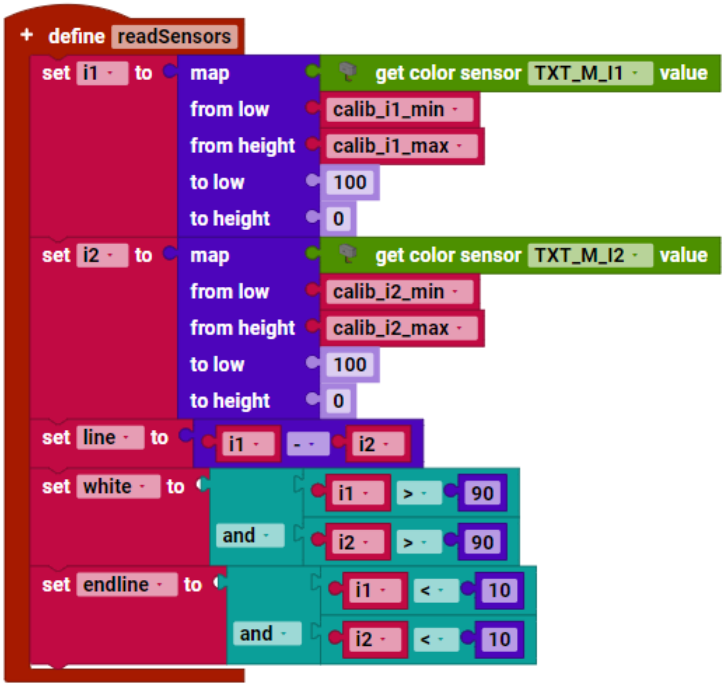


Im ersten Schritt werden die Sensoren ausgelesen.

Dann wird die Berechnung im PID-Regler durchgeführt. Die Variable **line** (-100 … 0 … 100) ist die Stellgröße. Der Sollwert ist in der Mitte der Linie beim Wert 0.

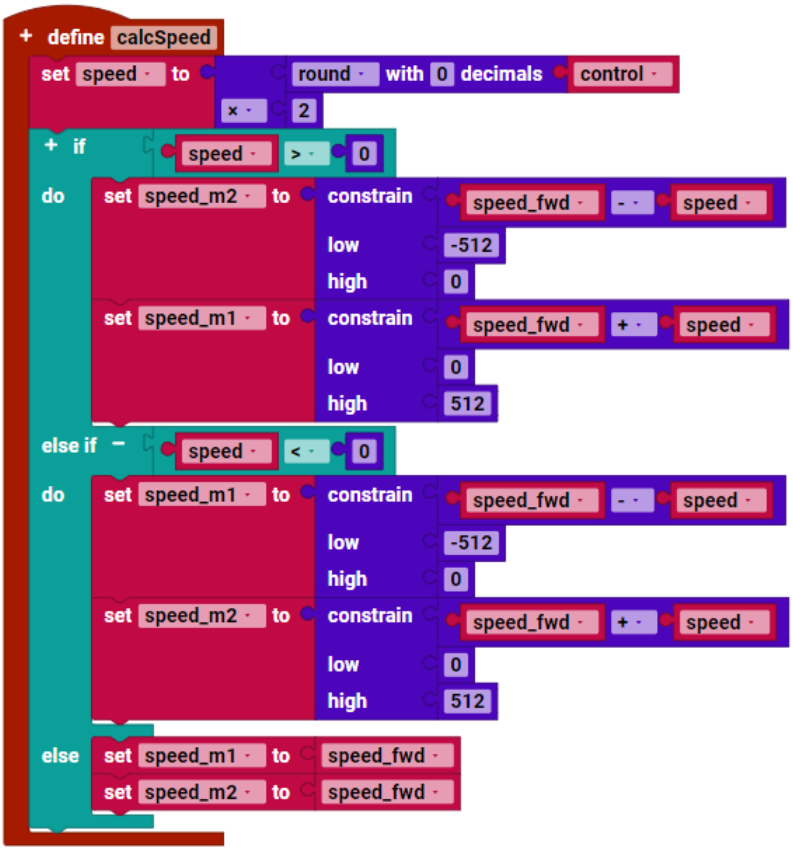
Anschließend werden die Geschwindigkeiten für die beiden Motoren berechnet und gesetzt. Zu beachten ist, dass die Geschwindigkeit der Motoren auf Werte zwischen -512 bis 512 begrenzt werden muss.

Die Sensorwerte werden in der Funktion **readSensors** ausgelesen und auf die kalibrierten Werte normiert. Dazu wird einmalig am Anfang des Programms die Kalibrierroutine aufgerufen, bei der die min und max Werte gesetzt werden. Falls sich die min und max Werte nicht ändern sollten, so können die Werte im Programm bei der Initialisierung direkt gesetzt werden. Dann muss die Kalibrierfunktion nicht aufgerufen werden.



Die Berechnung der Geschwindigkeit findet in der Funktion **calcSpeed** statt.

Experimentiere mit verschiedenen Parametern Kp, Ki und Kd. Versuche auch die Geschwindigkeit der Motoren zu verändern. Wenn die Werte richtig eingestellt sind, so sollte der Roboter dann noch präziser der Linie folgen.



Ausblick

Neben dem Set STEM Coding RoboMission gibt es von fischertechnik viele andere Zusatzkomponenten und Einzelteile, die ebenfalls verwendet werden können.

Schaue am besten auf der fischertechnik Homepage www.fischertechnik.de vorbei.