fischertechnik

Tutorial ROBO Pro Coding

STEM Coding RoboMission

2026.01

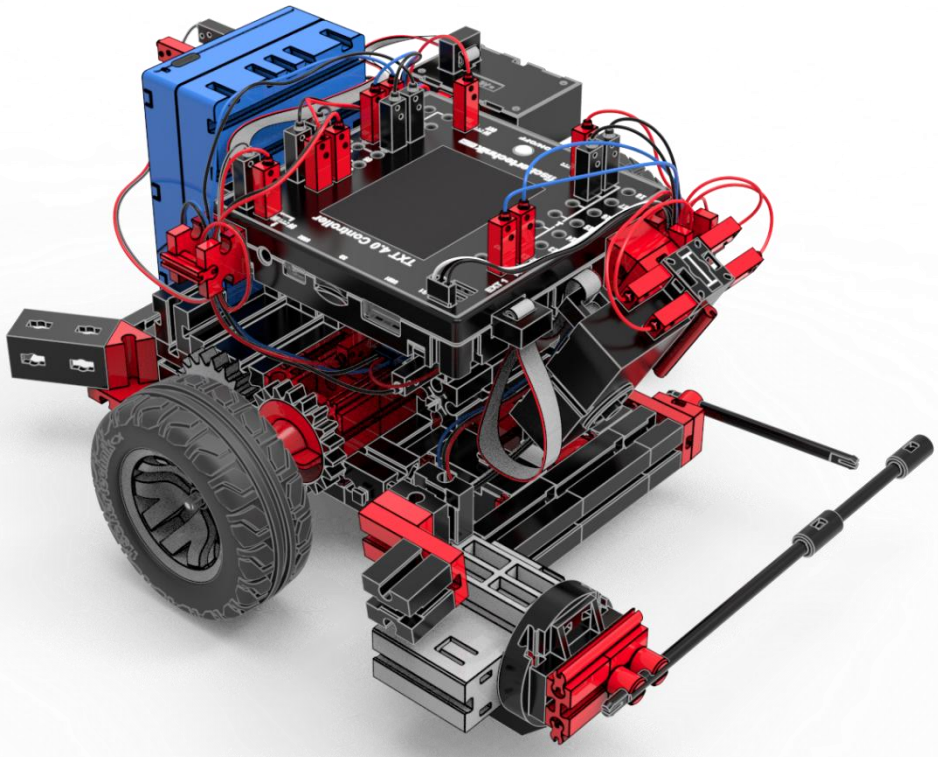


Table of Contents

Tutorials Overview 2
ROBO Pro Coding Basics 3
Introduction 3
App Configuration 3
Connection to the TXT 4.0 Controller 5
Firmware Update for TXT 4.0 Controller 6
Interface Test 7
Python Functions, Modules and Imports 8
Basic Variables 10
Python Code with Blockly 13
Importing Example Programs from GitLab 14
Debugging and Testing Programs 16
Hardware Components 18
TXT 4.0 Controller 18
Battery 19
Power Supply and Power-On 19
Encoder Motor and Servo 20
Line Sensors (Optical Color Sensors) 26
RGB Color Sensor 28
Combi Sensor 30
Differential Drive 33
Motor Control as a Library 34
Differential Drive Calculation 35
P-based Line-Following Control 43
Gyro Control – Straight Movement 45
Gyro Control – Predefined Rotation 47
TXT 4.0 Display 49
Measuring Battery Voltage 52
External I2C Interface 52
What's Next 53
Connection Plan 54

Tutorials Overview

This document provides a tutorial for the ROBO Pro Coding app and showcases example applications for the sensors and actuators included in the STEM Coding RoboMission robotics set.

All examples use the recommended robot model but can also be adapted to other models.

Please note that this tutorial assumes that you are already familiar with ROBO Pro Coding and know the basics of ROBO Pro Coding. The following chapters address topics relevant to RoboMission, with difficulty levels indicated as:

★	Basic
★★	Advanced
★★★	Complex
★★★★	Expert

For questions regarding this document or the tutorial, please contact:

fischertechnik-technik@fischer.de

ROBO Pro Coding Basics

Introduction

ROBO Pro Coding by fischertechnik offers a multilingual programming environment with both graphical programming using **Blockly** and text-based programming in **Python 3** for the TXT 4.0 controller.

ROBO Pro Coding is available for **Windows**, **Mac**, **Linux**, **Android,** and **iOS** operating systems:

https://www.fischertechnik.de/en/apps-and-software

The documentation for Blockly in ROBO Pro Coding is available at the following link:

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

App Configuration

★

When you install the **ROBO Pro Coding** App for the first time and create a new project, completing a few essential configuration steps ensures you have access to all features required for advanced programming and project development.

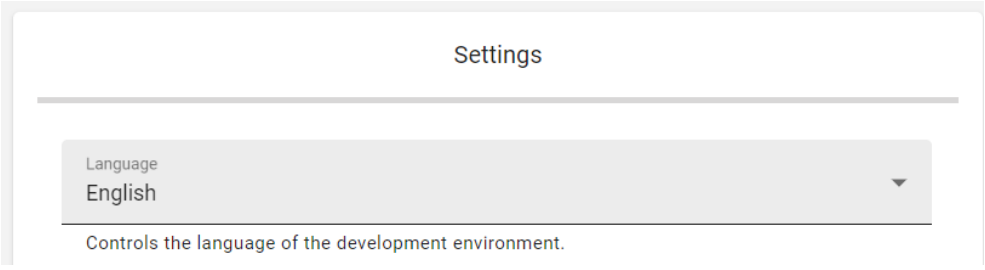
Setting Your Preferred Language

ROBO Pro Coding lets you customize the interface language for better usability and comfort.

Navigate to:

Menu → Preferences → Settings → Language

Then select the language that best fits your needs.

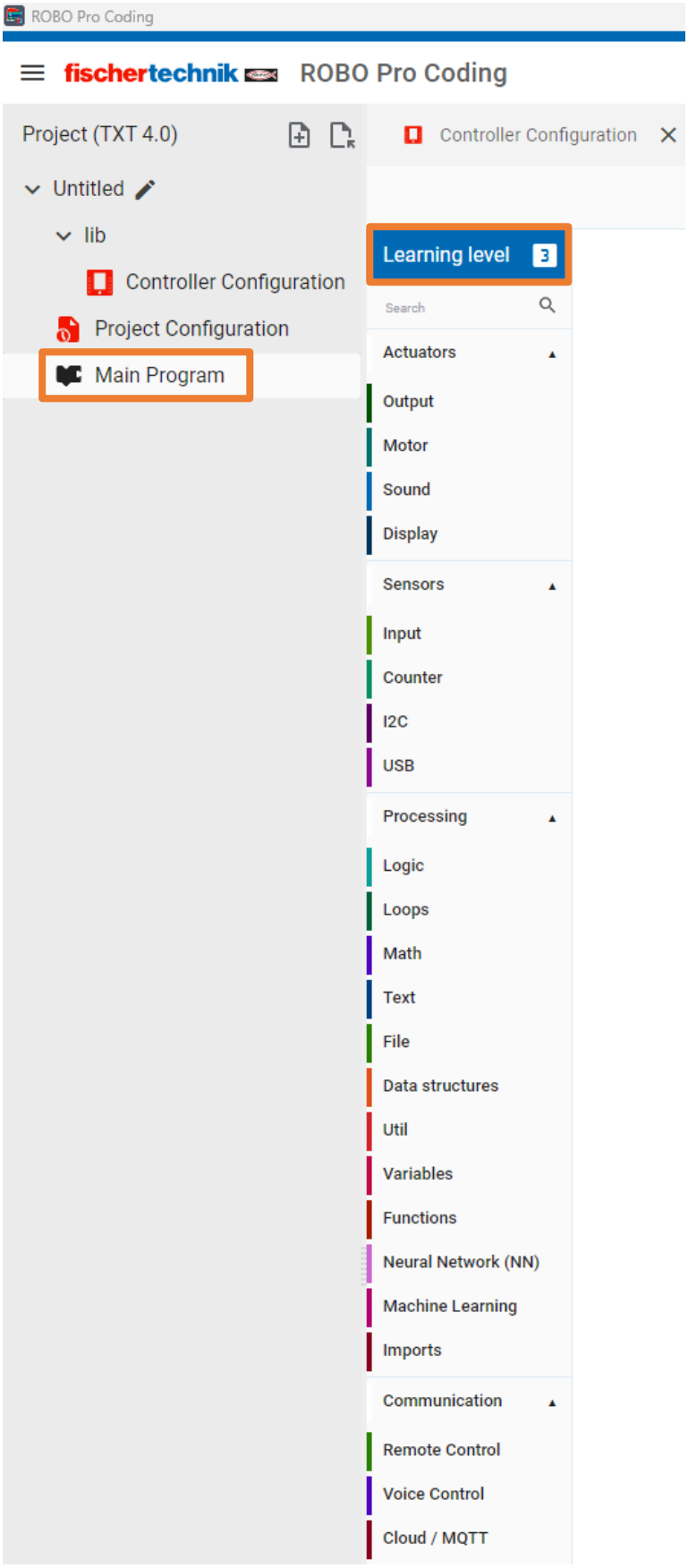


Adjusting the Learning Level

By default, the ROBO Pro Coding environment starts with Learning Level 1, which provides only basic programming blocks.

To enable advanced functions — such as extended blocks, sensor modules, and complex control structures — you must increase the learning level.

Change the value from Level 1 to **Level 3**.



Connection to the TXT 4.0 Controller

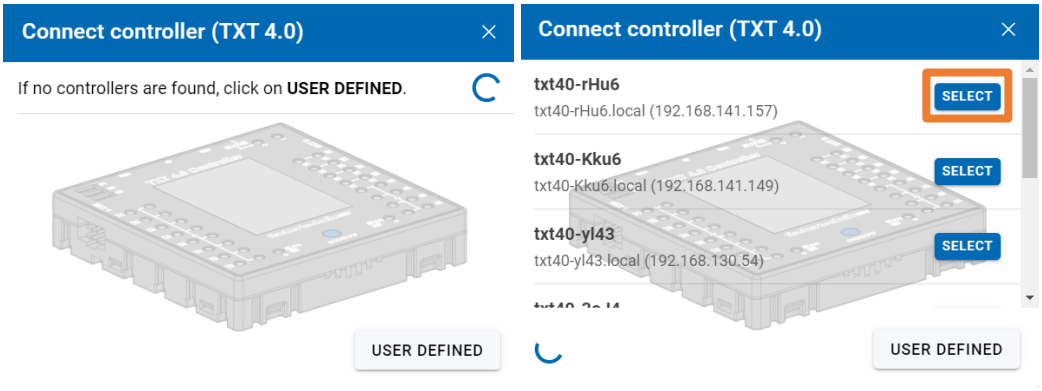
★

To establish a connection between ROBO Pro Coding and the TXT 4.0 controller, follow these steps:

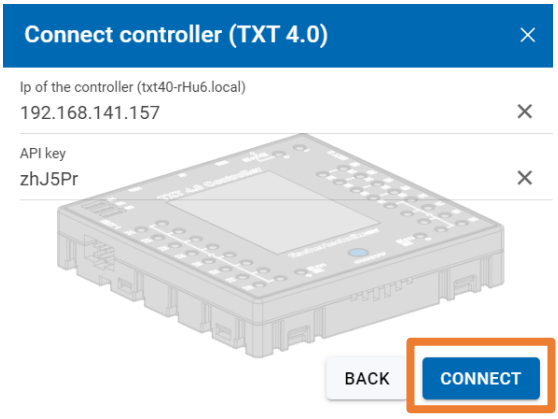
1. Power on the controller and ensure it is ready for communication.

2. In ROBO Pro Coding, click the Connect button.

3. Select the controller or click the USER DEFINED button:



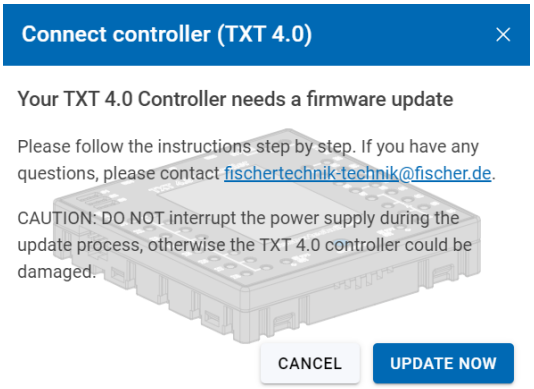
4. Click the Connect button:



5. Verify the connection status before starting programming or running interface tests.

Firmware Update for TXT 4.0 Controller

★★



Keeping your TXT 4.0 controller firmware up to date ensures optimal performance, full compatibility with ROBO Pro Coding, and access to the latest features and security improvements.

If you receive a message indicating that an update is required, please update the firmware before connecting the controller to ROBO Pro Coding.

Important: The firmware update must be initiated directly on the controller. You have two options:

1. **Update via Internet**
   Connect the controller to the internet and start the update process.
2. **Update via USB Stick or µSD card**
   Download the update file and transfer it to the controller using a USB stick or µSD card.

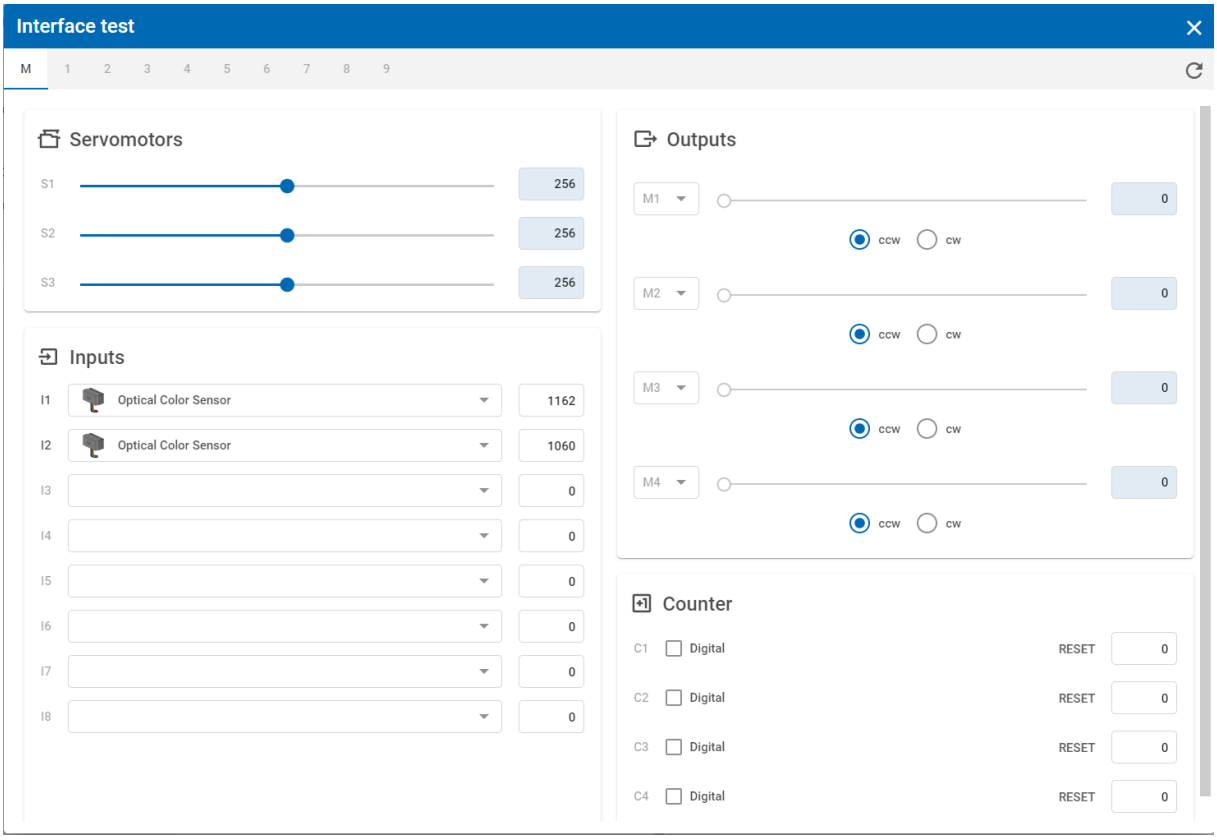
For detailed instructions and the latest firmware downloads, visit:

https://www.fischertechnik.de/en/toys/e-learning/txt-4-0-controller

Interface Test

★

Actuators and sensors can be tested quickly without writing a program by using the interface test in ROBO Pro Coding, provided a connection to the TXT 4.0 controller is established.



For servo motors S1-S3, the value can be set using the slider.

For motors M1-M4, the speed and direction of rotation of a motor can be set.

The value is displayed for the universal inputs I1-I8, e.g., for the optical sensors. The individual universal inputs must be configured beforehand via the selection box.

The current counter values C1-C4 for the encoder inputs are also displayed.

Important: Please note that the interface test and a program cannot be started at the same time.

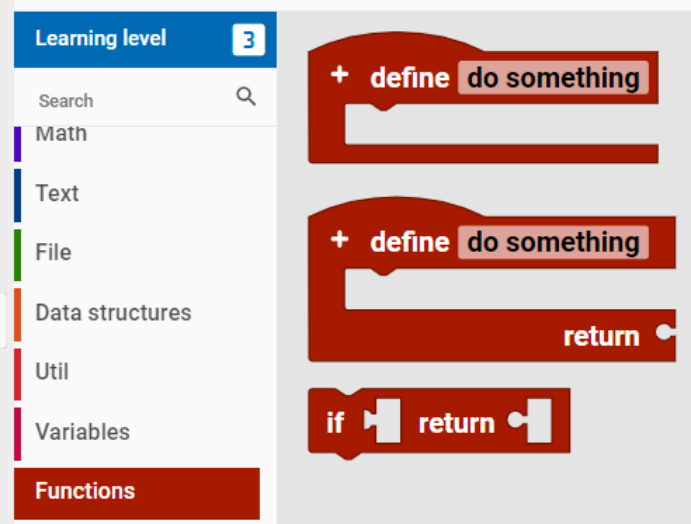
Python Functions, Modules and Imports

★★★

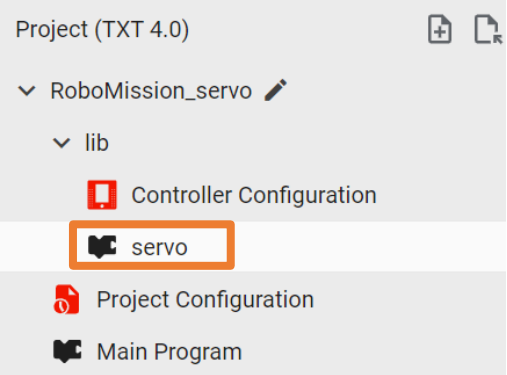
Functions that are available locally in the module or in the main program are defined with “**Functions**”.

Functions can be defined with parameters as well as with return values.

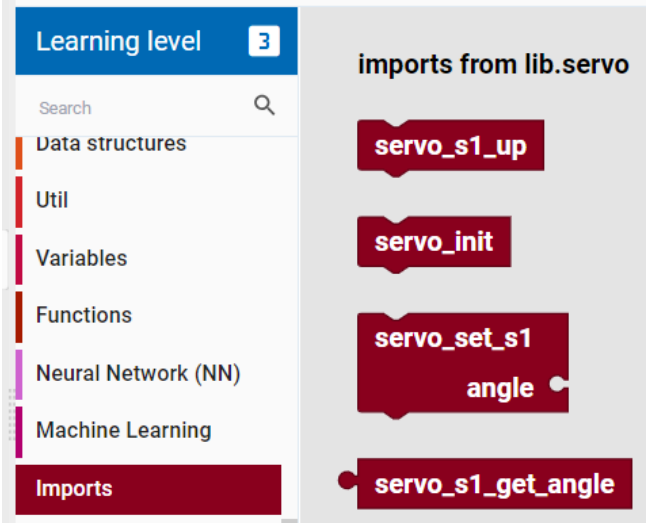
The parameters are added with (+) and defined accordingly.



In the **RoboMission_servo** program example, functions for servo motors are defined in additional files that can be found in the structure under “**lib**”. The advantage of having your own Python modules is that the code is clearer and can be reused as a library in other projects.



Functions used in other files can be added via “**Imports**”. To do this, switch to the file in which the function is to be used and select this function from another module via “Imports”.

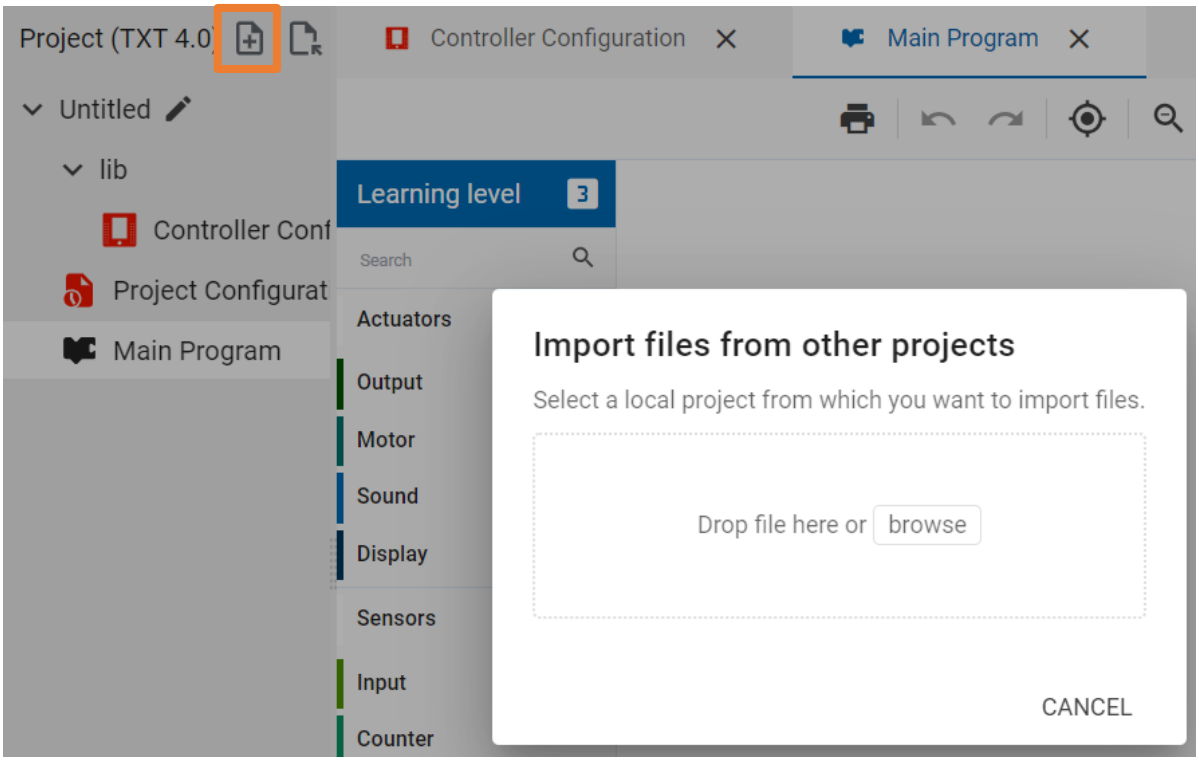


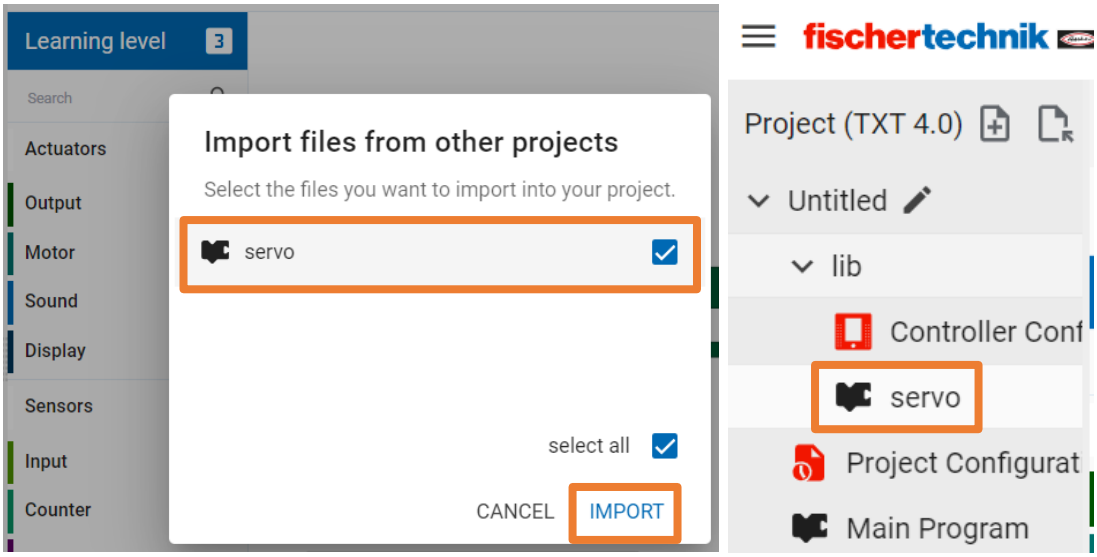
The local functions “Functions” and imported functions “Imports” have different colors. This allows you to differentiate between them in the program flow.

Please ensure that you do not create circular references between modules. If two modules import functions from each other, the imports cannot be resolved correctly.

To integrate the “servo” library or any other library into your project, follow these steps:

1. Ensure the project containing the library to be imported is saved locally on your device.
2. Open an existing project or create a new one in ROBO Pro Coding.
3. Click the Import icon in the application window and navigate to the project that contains the desired libraries.
4. From the available options in the project structure, select the appropriate library (e.g., “servo”) and click IMPORT.



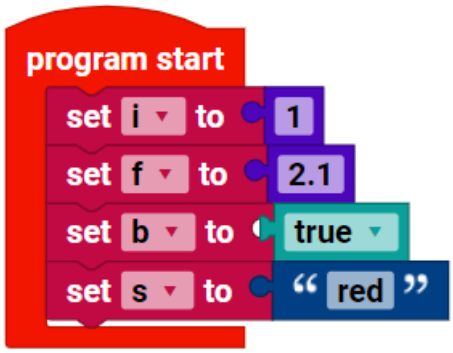


Basic Variables

★★

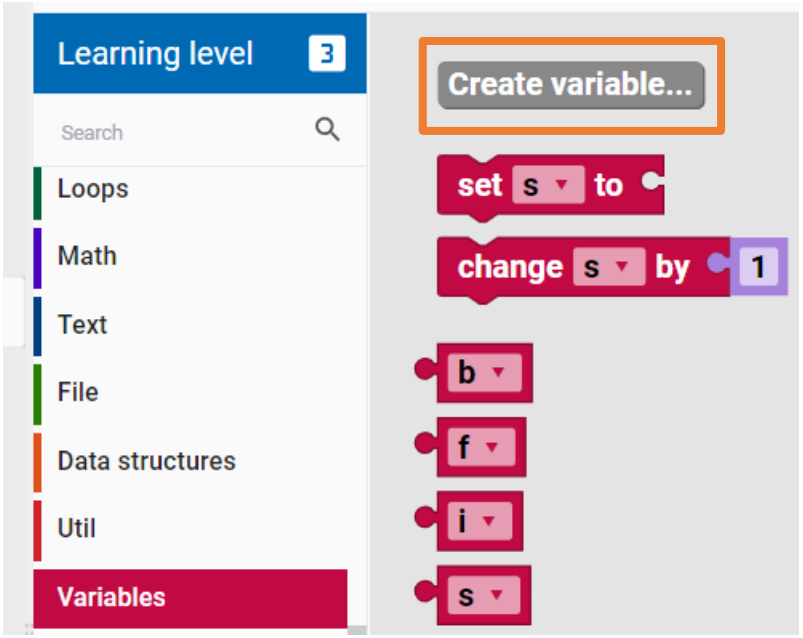
Variables are used to store values that can change during program execution. In ROBO Pro Coding, variables allow you to work with dynamic data such as sensor readings or calculated results. You can create variables of different types, including:

- **Integer** *(see "Math"):* Whole numbers, e.g., counter values or motor speeds.
- **Float** *(see "Math"):* Decimal numbers, useful for precise measurements.
- **Boolean** *(see "Logic"):* True/False values, often used for conditions.
- **String** *(see "Text"):* Text values, for labeling or displaying messages.

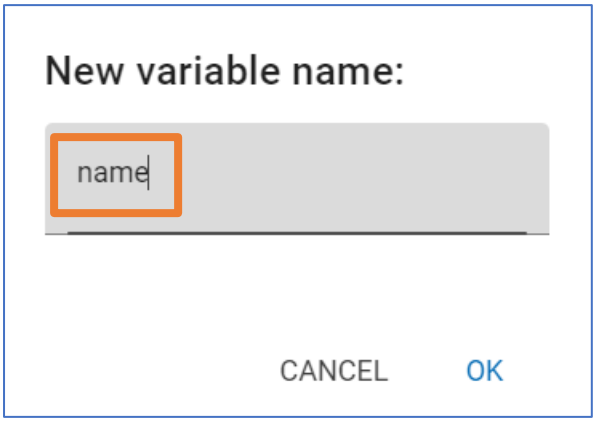


How to Create a Variable

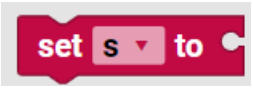
1. Open the Variables section in ROBO Pro Coding and click on "Create variable".



2. Assign a name.



3. Optionally, set an initial value.



Variables can be modified during program execution using operations such as addition, subtraction, or assignment. They are essential for implementing logic, loops, and conditions in your projects.

Scope of Variables

Variables can be defined globally or locally within a program:

- **Global** variables are accessible throughout the entire module.
- **Local** variables are limited to specific functions or blocks.

Important: Special Case in Blockly Python Generator:

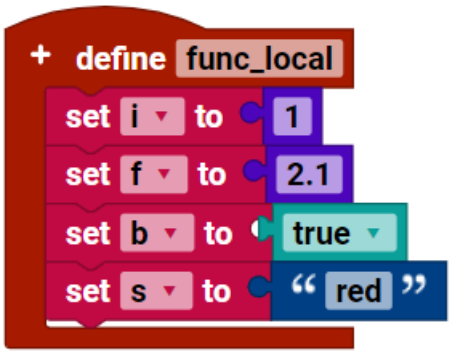
- All variables are initialized with the value **None**.
- All local variables in Blockly are treated as **global** in the generated Python file.

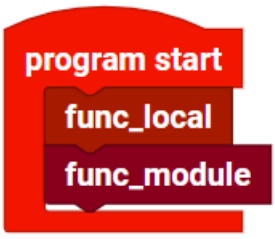
This means:

- Local variables are available throughout the generated file.
- They are not accessible in other modules defined in lib.

Examples with auto-generated Python code

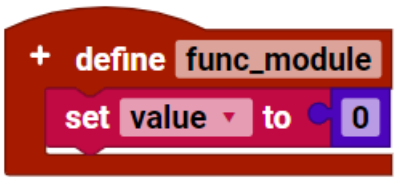
- **Main Program**






```
from lib.controller import *
from lib.module import *

i = None
f = None
b = None
s = None


def func_local():
    global i, f, b, s
    i = 1
    f = 2.1
    b = True
    s = 'red'


func_local()
func_module()
```

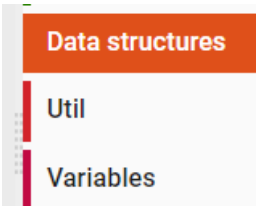
- **lib -> module**




```

value = None


def func_module():
    global value
    value = 0

```

Hint:

Structured variables such as **lists** and **dictionaries (maps)** are also supported. For more details, refer to the official ROBO Pro Coding documentation, as this tutorial focuses on basic variables.

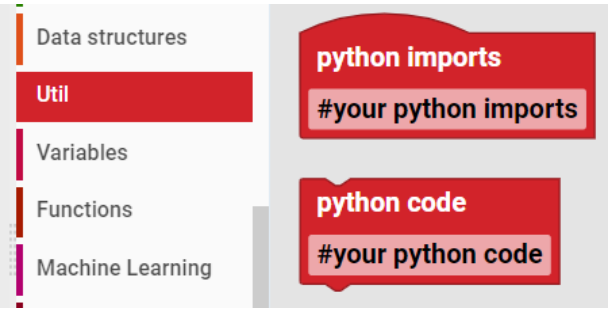


Python Code with Blockly

★★★

Most of the functions in ROBO Pro Coding can be implemented with Blockly. There are many blocks for different use cases. It is very convenient to use the blocks because the required instructions in Python are generated automatically with ROBO Pro Coding.

However, sometimes Python commands need to be integrated into Blockly programs because the corresponding block does not yet exist or because the sequence of Python commands needs to be defined differently. This can be done very easily in a Blockly program by creating the required Python commands with the following Python blocks:



The "**python imports**" block can be used to import specific Python modules. The blockly generator places the content at the beginning of the file.

The "**python code**" block can be used to define Python code anywhere in the blockly program.

A Python editor opens when the block is clicked. Python code can then also be defined over several lines.

Particular attention must be paid to the indentation of the source code, as in Python the indentation via spaces or tabs has a significant influence on the structure of the program.

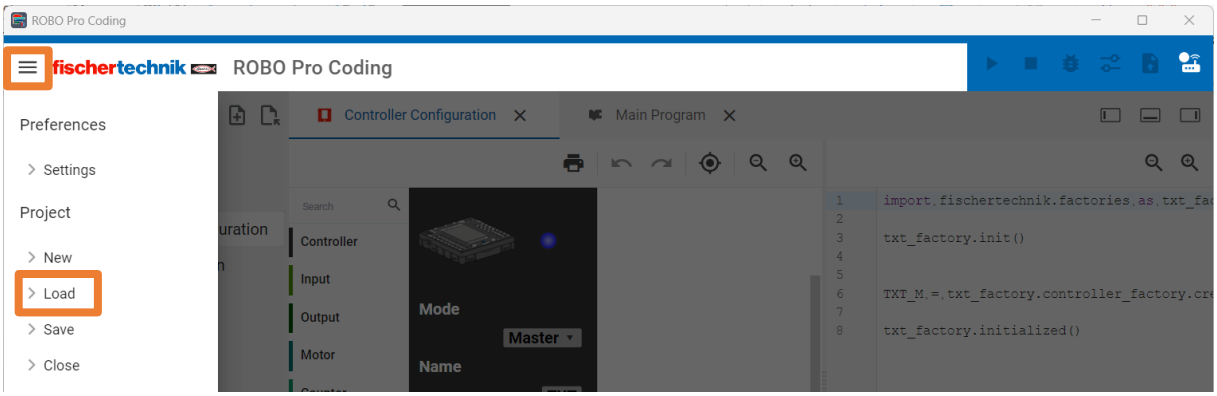
Importing Example Programs from GitLab

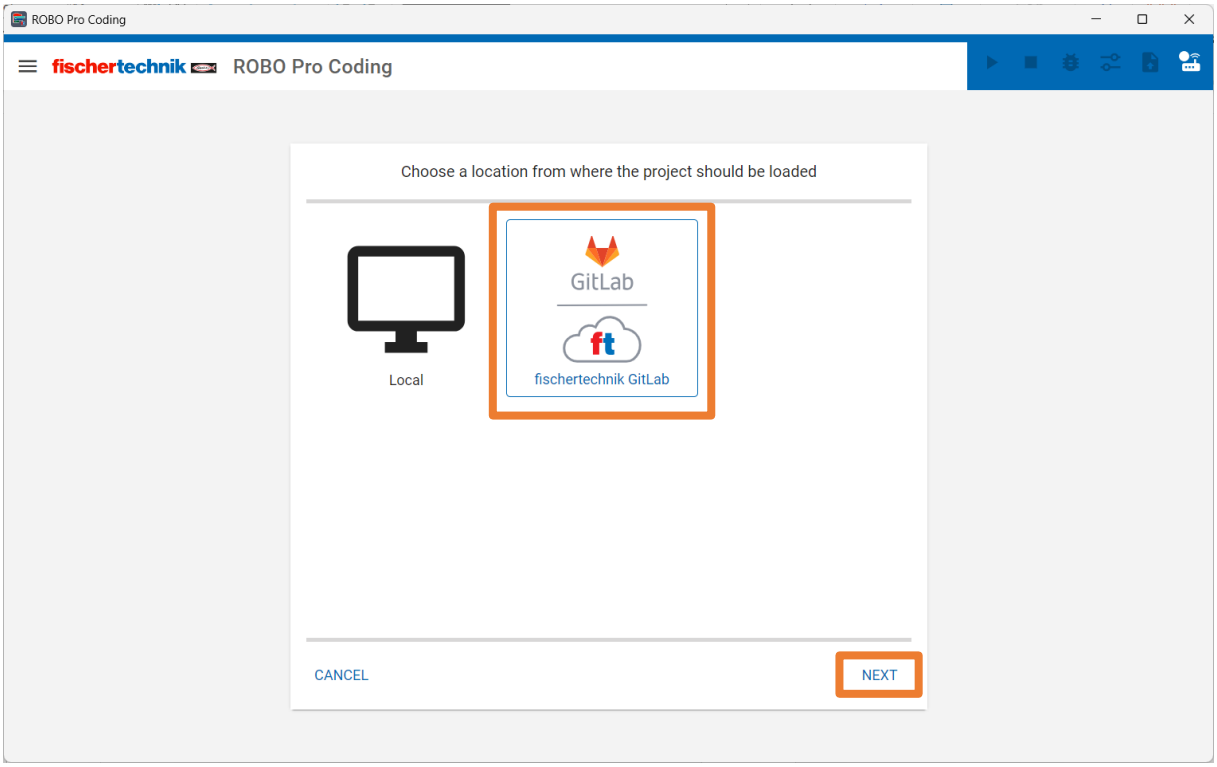
★

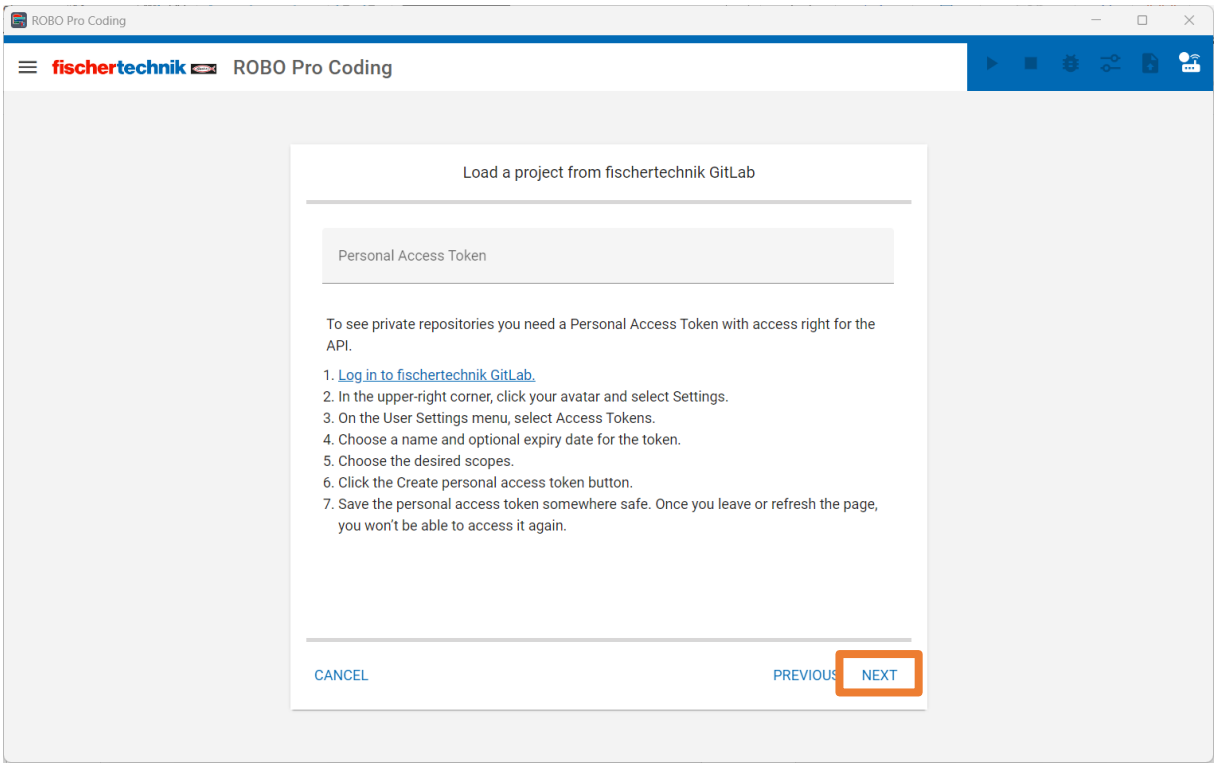
All relevant examples in this document can be imported directly into **ROBO Pro Coding** from **fischertechnik GitLab**. For this purpose, the name of the program can be entered in the search field under

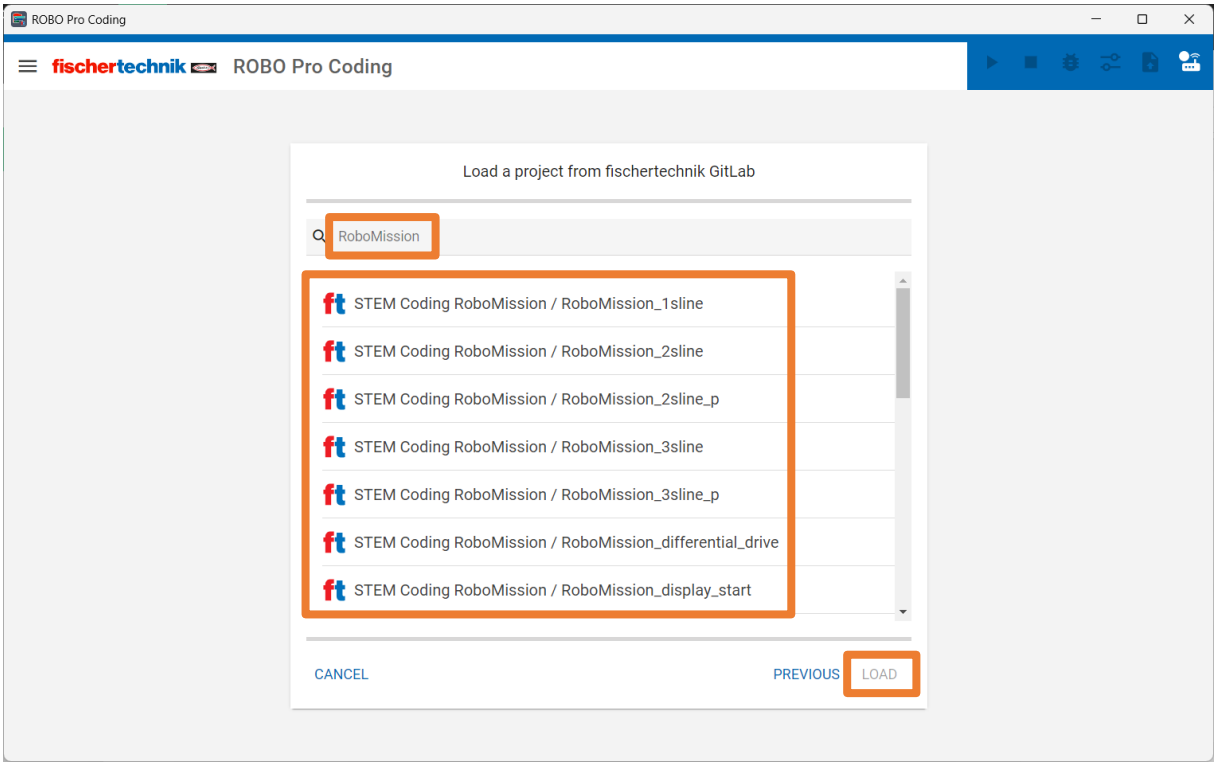
Menu -> Load -> fischertechnik GitLab

and the corresponding program example can be loaded directly.






ROBO Pro Coding
fischertechnik ROBO Pro Coding
Load a project from fischertechnik GitLab
Personal Access Token
To see private repositories you need a Personal Access Token with access right for the API.
1. Log in to fischertechnik GitLab.
2. In the upper-right corner, click your avatar and select Settings.
3. On the User Settings menu, select Access Tokens.
4. Choose a name and optional expiry date for the token.
5. Choose the desired scopes.
6. Click the Create personal access token button.
7. Save the personal access token somewhere safe. Once you leave or refresh the page, you won't be able to access it again.
CANCEL
PREVIOUS
NEXT


ROBO Pro Coding
fischertechnik ROBO Pro Coding
Load a project from fischertechnik GitLab
RoboMission
STEM Coding RoboMission / RoboMission_1sline
STEM Coding RoboMission / RoboMission_2sline
STEM Coding RoboMission / RoboMission_2sline_p
STEM Coding RoboMission / RoboMission_3sline
STEM Coding RoboMission / RoboMission_3sline_p
STEM Coding RoboMission / RoboMission_differential_drive
STEM Coding RoboMission / RoboMission_display_start
CANCEL
PREVIOUS
LOAD

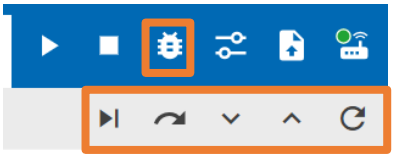
Debugging and Testing Programs

★★

Effective debugging is essential when developing and testing programs in ROBO Pro Coding. It helps you identify, analyze, and resolve errors or unexpected behavior in your code.

Debugging Methods

- **Debugging Window in ROBO Pro Coding** (Single-Thread Only)

  ROBO Pro Coding provides a debugging window that displays all print outputs and error messages during program execution. The window supports the standard functionality of the Python pdb debugger, including breakpoints, expression evaluation, and callback controls.

  Core Workflow pdb debugger:

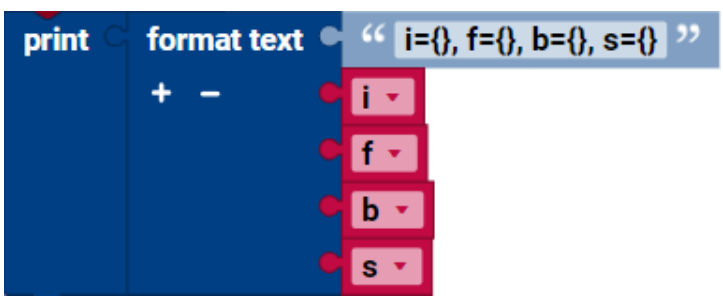
  - **Pause** at a breakpoint or when an exception occurs.
  - **Inspect** variables, the current stack frame, and program state.
  - **Step** through code to analyze control flow.
  - **Set breakpoint** to isolate the relevant code paths.
  - **Continue** execution until the next breakpoint or program termination.

  Important Limitation: The debugging window only supports simple, single-threaded programs. Debugging multithreaded programs is not currently supported.

- **Using print() Statements**

  Insert print() statements to display:

  - Variable values
  - Sensor readings
  - Program states

  This helps you trace program flow and pinpoint where problems occur.

- **Using Python's logging Module**

  For more advanced debugging, you can use the built-in Python logging module, see https://docs.python.org/3.5/howto/logging.html

  Advantages of logging:

  - Flexible log levels: DEBUG, INFO, WARNING, ERROR, etc.
  - Structured output: timestamps, severity levels, module names
  - Easy filtering: quickly focus on relevant messages

Best Practices

- Place print or logging statements at **key points**:
  - Before and after critical operations
  - When entering or exiting functions
- Avoid frequent output inside **time-critical loops**, because it may slow down your system.
- Use **clear and descriptive messages** to make troubleshooting easier.

Additional Tips

- Carefully read **error messages** in the log window for clues.
- Test your program in **small sections** and make incremental changes to improve stability.

By following these debugging techniques, you can efficiently identify and resolve issues in your ROBO Pro Coding projects, making development faster and more reliable.

Hardware Components

TXT 4.0 Controller

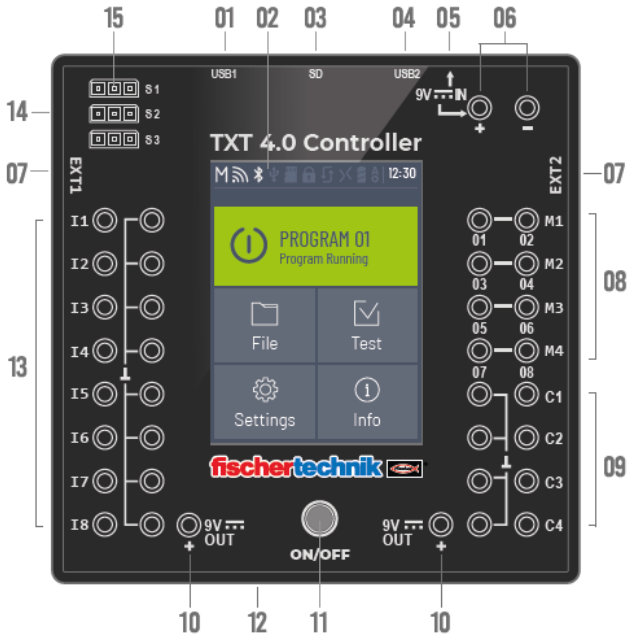
The robot's central component is the fischertechnik TXT 4.0 Controller (Art.-No. 173067). The official documentation and First Steps video for the controller are available here:

- https://www.fischertechnik.de/en/toys/e-learning/txt-4-0-controller
- First steps with the TXT 4.0 controller

Below is a summary of its key specifications:

- 8 Universal inputs
- 4 Counting inputs
- 4 Motor outputs, alternatively 8 individual outputs
- 3 Servo Outputs
- Touch Display
- 2 6-pin sockets for connecting additional 3.3V I2C sensors

Programming can be done both graphically and with Python using Robo Pro Coding software.



01 USB-A port (USB1)
02 Touch display
03 Micro SD card slot
04 Mini USB port (USB2)
05 9V IN, DC port
06 9V IN, Battery Pack connection
07 EXT connections for expansions
08 Outputs M1–M4 or O1–O8
09 Inputs C1–C4
10 9V Out
11 On / off switch
12 Speaker
13 Universal inputs I1– I8
14 Outputs S1-S3 for servo motors

Battery

The Getting Started guide for the fischertechnik battery (Art.-No. 219823) can be viewed at the following link: 576108 fischertechnik Battery

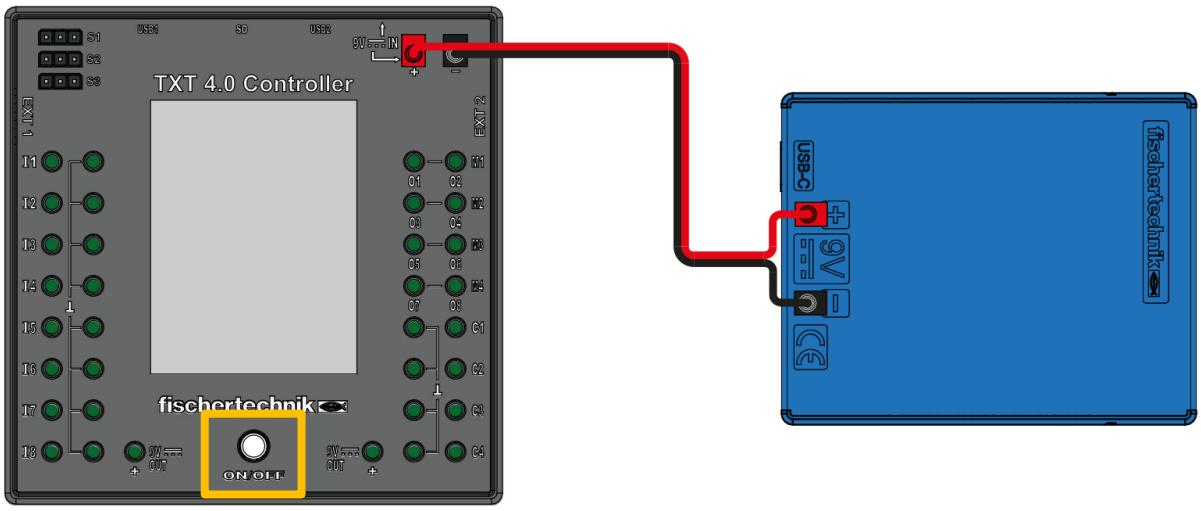
Note:
In the event of a short circuit, the battery activates an internal protection mechanism and switches itself off for safety reasons. To reactivate it, simply connect a USB-C charger to the battery. The protection circuit resets automatically, and the battery can then be used as normal.

Power Supply and Power-On

★

To power up the TXT 4.0 controller, please proceed as follows:

1. **Connect the battery** securely to the controller's power connector.
2. Ensure the battery is **fully charged.**
3. **Turn on the TXT 4.0 Controller** by pressing and holding the **ON/OFF button for more than 3 seconds**.
   The status LEDs will illuminate, indicating that the device is starting up.
4. Wait until the controller has fully booted before proceeding with further steps.



Encoder Motor and Servo

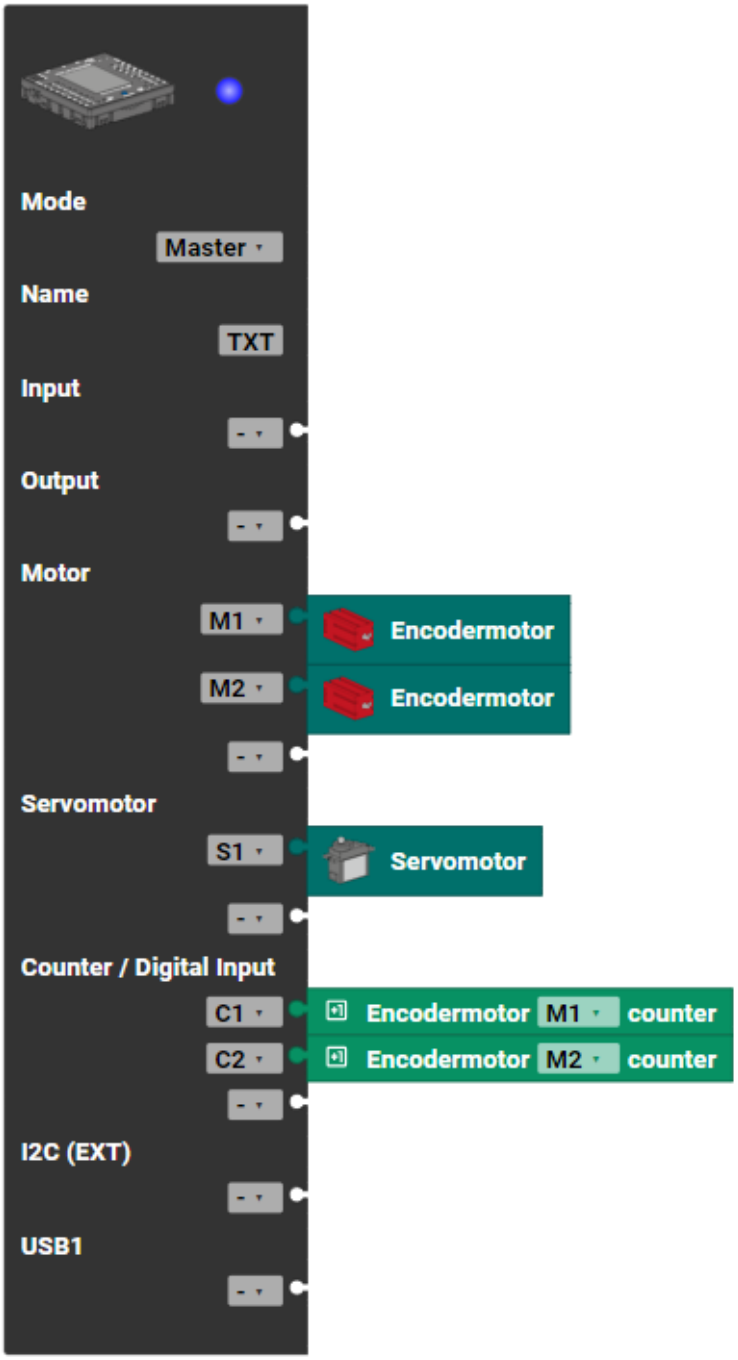
★

A maximum of four encoder motors (Art.-No. 186175 or 153422) and up to three servo motors (Art.-No. 203659 or 132292) can be connected to a TXT 4.0 controller simultaneously.

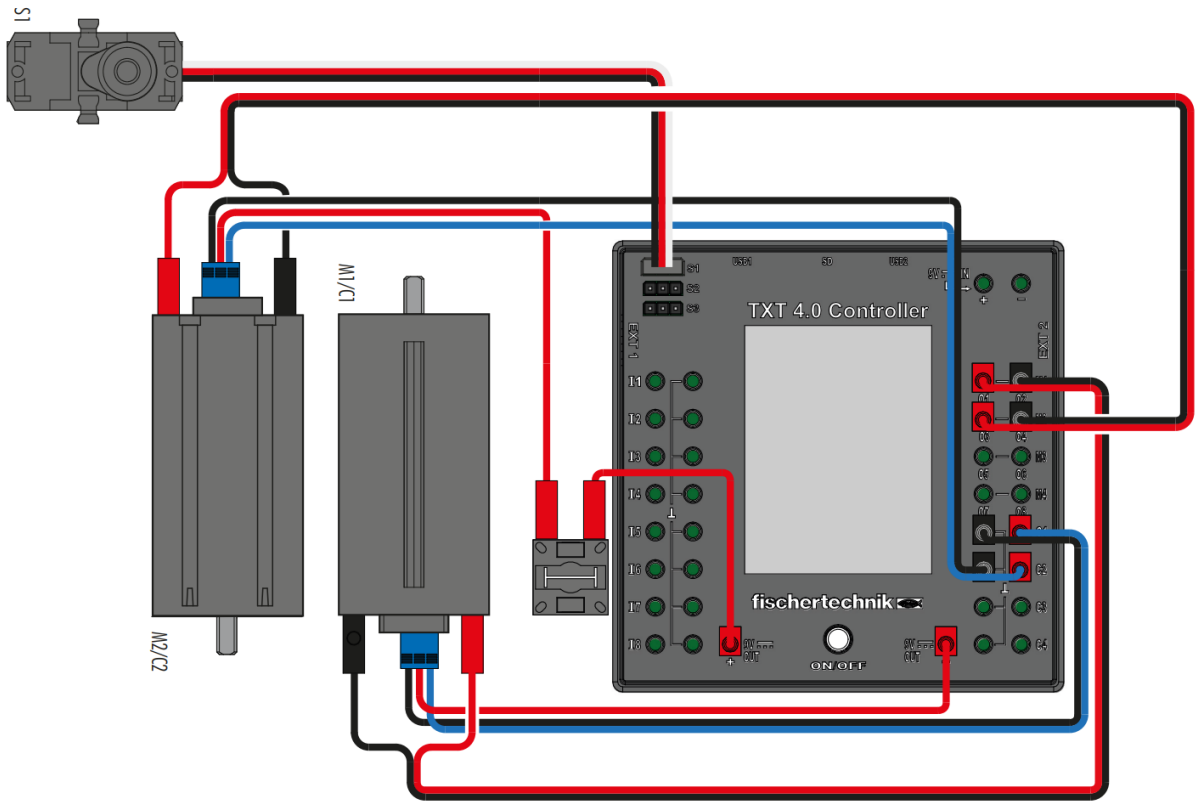
First Steps videos:

- The fischertechnik encoder motor
- The fischertechnik servo motor

To operate the actuators, assign the motors (M1-M4, C1-C4) and servos (S1-S3) to the appropriate outputs in the controller configuration within ROBO Pro Coding. They can then be selected in the main program.



The actuators for the STEM Coding RoboMission kit are connected as follows.

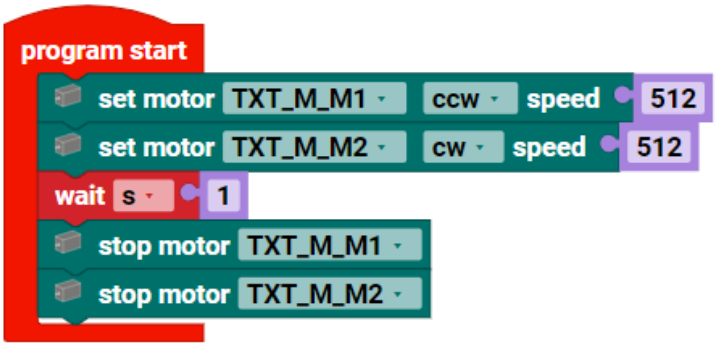


Direct Motor Control

In the simplest scenario, two encoder motors can be controlled without encoder feedback by directly setting the rotation direction — clockwise (CW) or counterclockwise (CCW) — and the speed value (0–512) for pulse-width modulation (PWM).

When using variables for speed, invert the direction by adding a minus sign before the value (-512 … 0 … 512).

The following example switches both motors on for one second and then stops them.



The main advantage of direct control is that the program provides full control over the motor at all times.

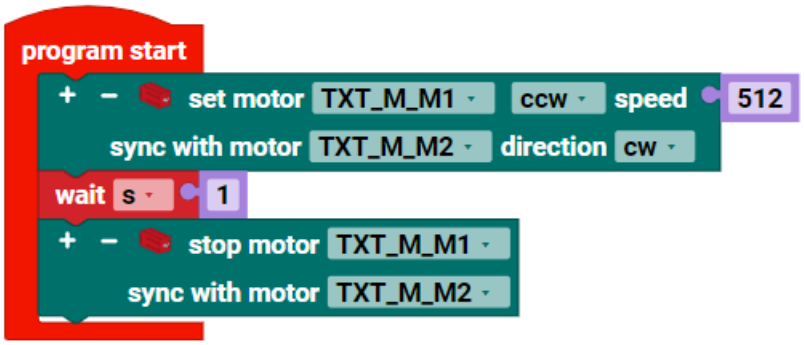
The drawback is that the motors are not synchronized; one motor always starts and stops slightly before the other. External factors such as wheel friction also negatively affect the robot's movement, making precise straight-line travel impossible.

For implementing custom controllers, direct motor control is recommended.

Advanced Motor Control with Synchronization

The TXT 4.0 firmware includes an advanced motor control feature that synchronizes motors using encoder signals.

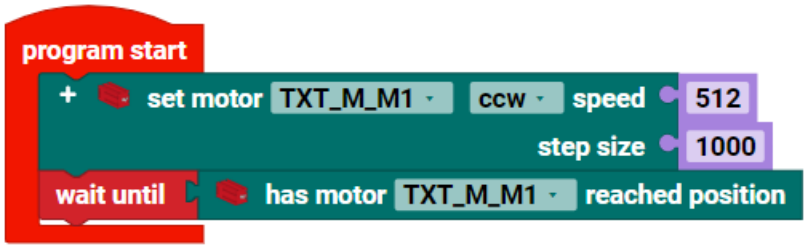
Using the following blocks, motors M1 and M2 run synchronously and stop after one second. If one motor is blocked, the other follows its movement. Synchronization is essential for straight-line motion.



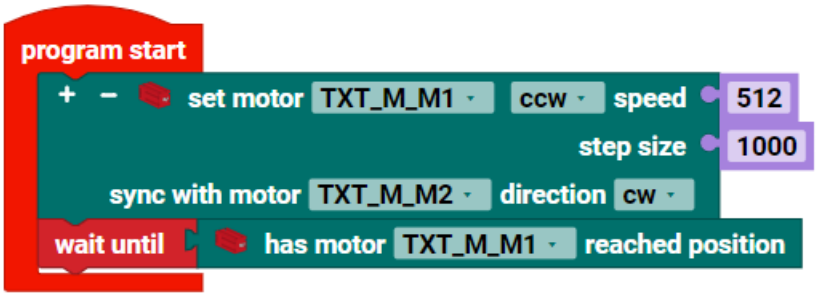
Keep in mind: when synchronization is enabled, use a different stop block to halt advanced motor control.

Synchronization with Distance Specification

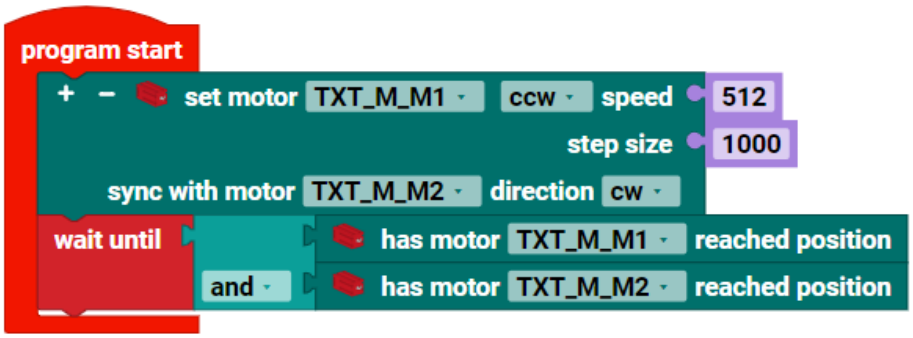
In addition to synchronous operation, you can specify a distance (encoder pulse count) for one or multiple motors.



The “has motor … reached position” block checks whether the target position is reached, allowing program execution to continue. Once the set distance (e.g., 1000 pulses) is reached, the motor stops automatically.



Usually, checking one motor’s status is sufficient, but you can query all motors to ensure proper handling in case of an encoder cable break.

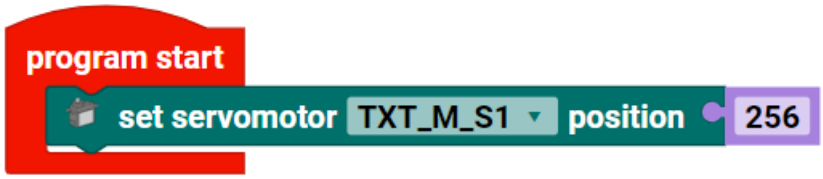


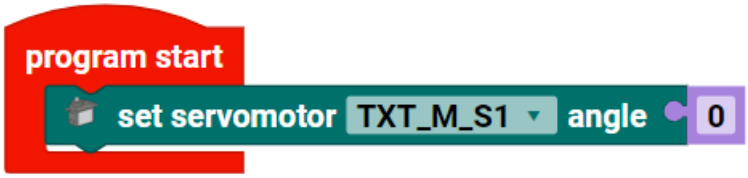
Servo Motor Control

Unlike drive motors, servo motors can directly move to absolute angles of approx. -90° to +90°. Control and sensors are integrated into the servo, making it ideal for grippers or other rotary movements.

The servo moves to a specific angle by setting an absolute PWM value. The center position is approx. 256, with limits at 0 and 512. These values vary slightly by servo and should be calibrated individually using constants or variables.

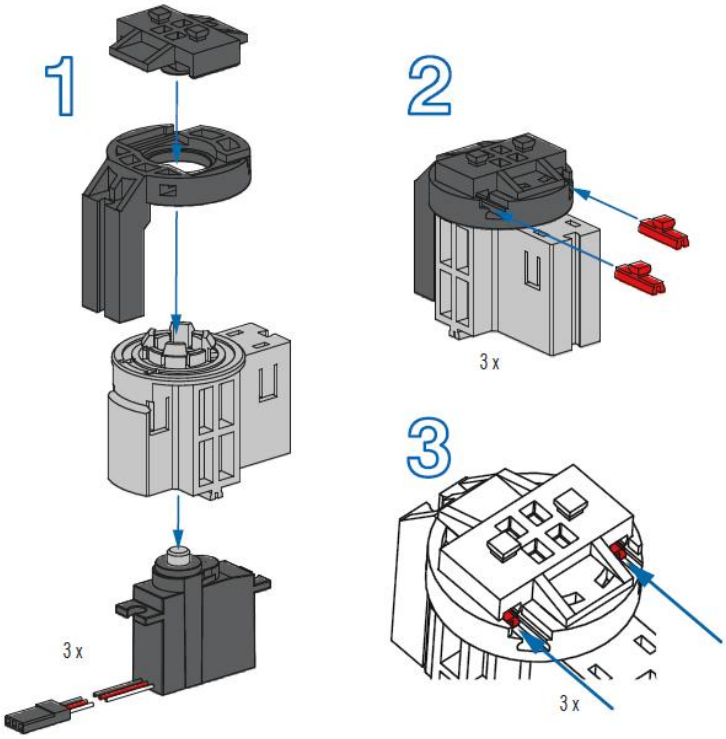
There are two blocks available for servo control: one for specifying the raw PWM value and another for defining the angle directly.





Important:

- **The servo does not provide feedback to the controller. If the target position is mechanically blocked, the servo may draw excessive current, causing damage. Avoid this at all costs.**
- The **zero position** must be **mechanically calibrated** during installation. Ensure the servo is mounted as shown in the reference image.



In the servo menu, the following elements are available:

The TXT 4.0 controller allows you to calibrate the minimum and maximum pulse width for each connected servo S1–S3. By adjusting these limits, you can **fine-tune** the effective travel and thus the center position of the servo. This compensates for minor mechanical tolerances or small differences between individual servos. In case of larger deviations or mechanical issues, the servo or mechanics should be checked instead of trying to correct everything in software.

To open the calibration menu:

Menu → Settings → Servo → Servo 1...3

In the servo menu, the following options are available:

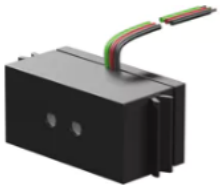
- **(-) / (+) at “min”**
  Decrease or increase the minimum pulse width in microseconds (µs). This changes the lower end position of the servo.
- **(-) / (+) at “max”**
  Decrease or increase the maximum pulse width in microseconds (µs). This changes the upper end position of the servo.
- **R (Reset)**
  Restores the factory default value for the respective limit (min or max). Use this if, after multiple changes, the servo no longer reaches the correct range.
- **Test buttons MIN / MID / MAX**
  - **MIN**: Moves the servo to the currently set minimum position.
  - **MID**: Moves the servo to the center position, calculated automatically from the min and max values. Use this to align the mechanical zero position.
  - **MAX**: Moves the servo to the currently set maximum position.

By alternately adjusting the values with (-)/(+) and checking the positions with the test buttons, you can precisely set the motion range of the servo. Once the end positions and the center position match the mechanics of your model, exit the menu. The values are then stored in the controller.

Line Sensors (Optical Color Sensors)

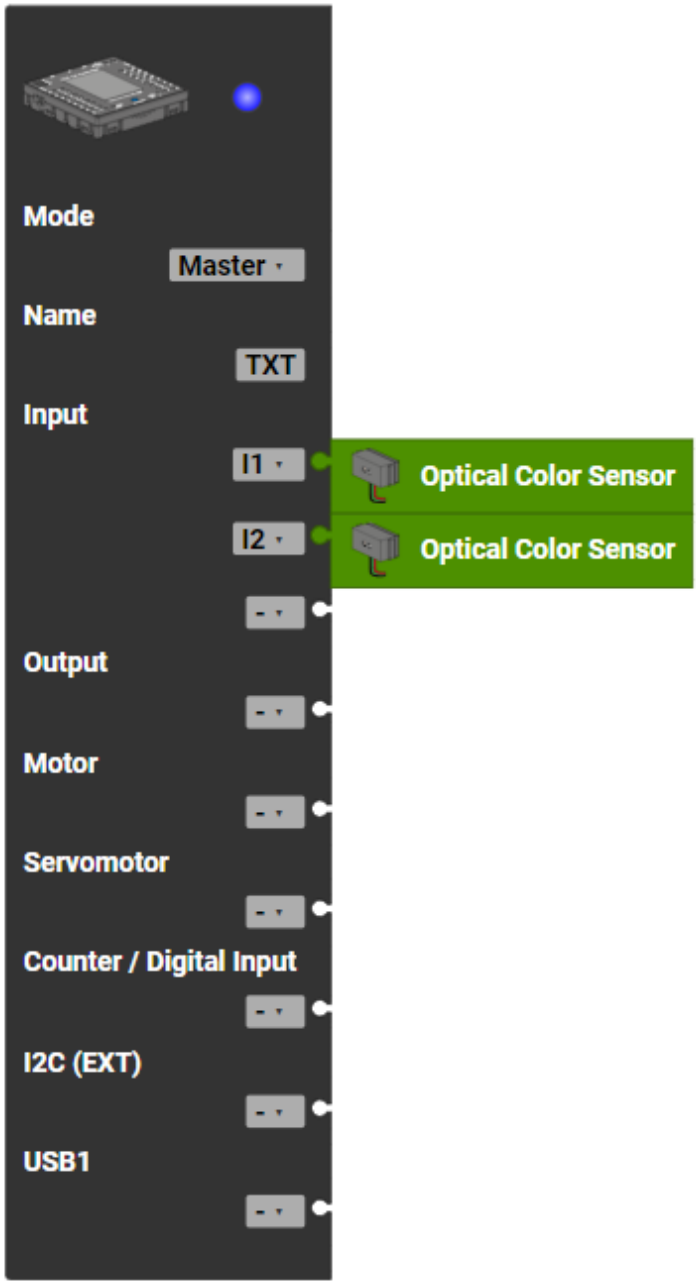
★

Digital or analog sensors can be connected to the universal inputs (I1–I8) of the TXT 4.0 controller. To use them, assign the corresponding sensors to the inputs in the controller configuration within ROBO Pro Coding. They can then be selected in the main program.



2x

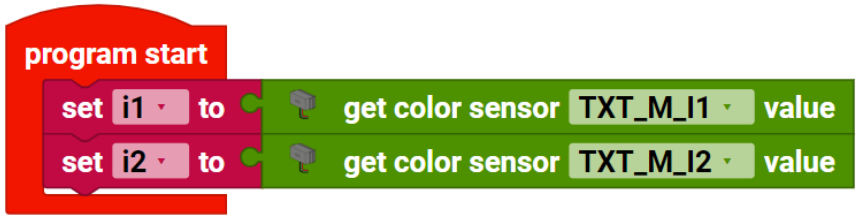
The STEM Coding RoboMission set includes two optical color sensors (Art.-No. 128599, 213966 or 222966) that detect simple light-dark transitions on the course (e.g., a black track). These sensors can also recognize black-white transitions and gray values, enabling precise line alignment or implementation of a line-following algorithm.



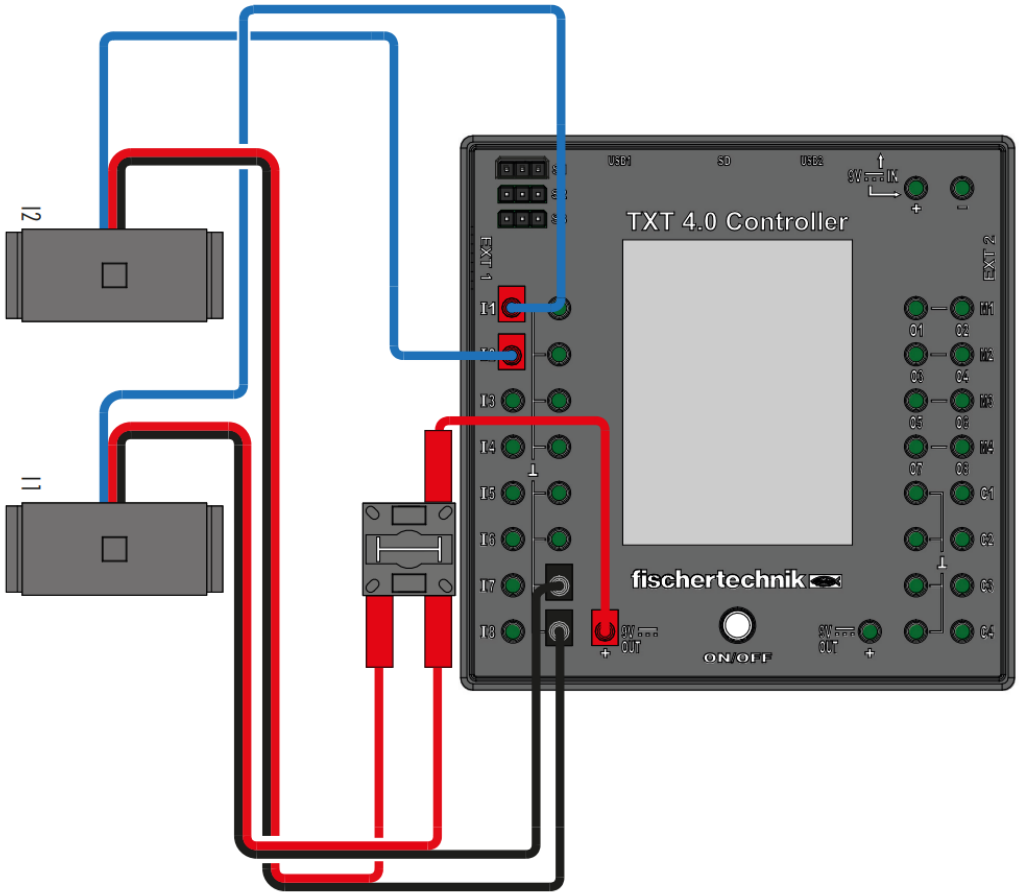
First Step videos:

- The fischertechnik color sensor

The analog values of both sensors can be read using the provided blocks. These values correspond to the voltage in millivolts (mV) measured at inputs I1 and I2.



The line sensors for the STEM Coding RoboMission kit are connected as follows.



Important: There are significant differences between the line sensor (optical color sensor) and the RGB color sensor. The optical color sensor is used as a line sensor, whereas the RGB sensor is designed to detect colored objects.

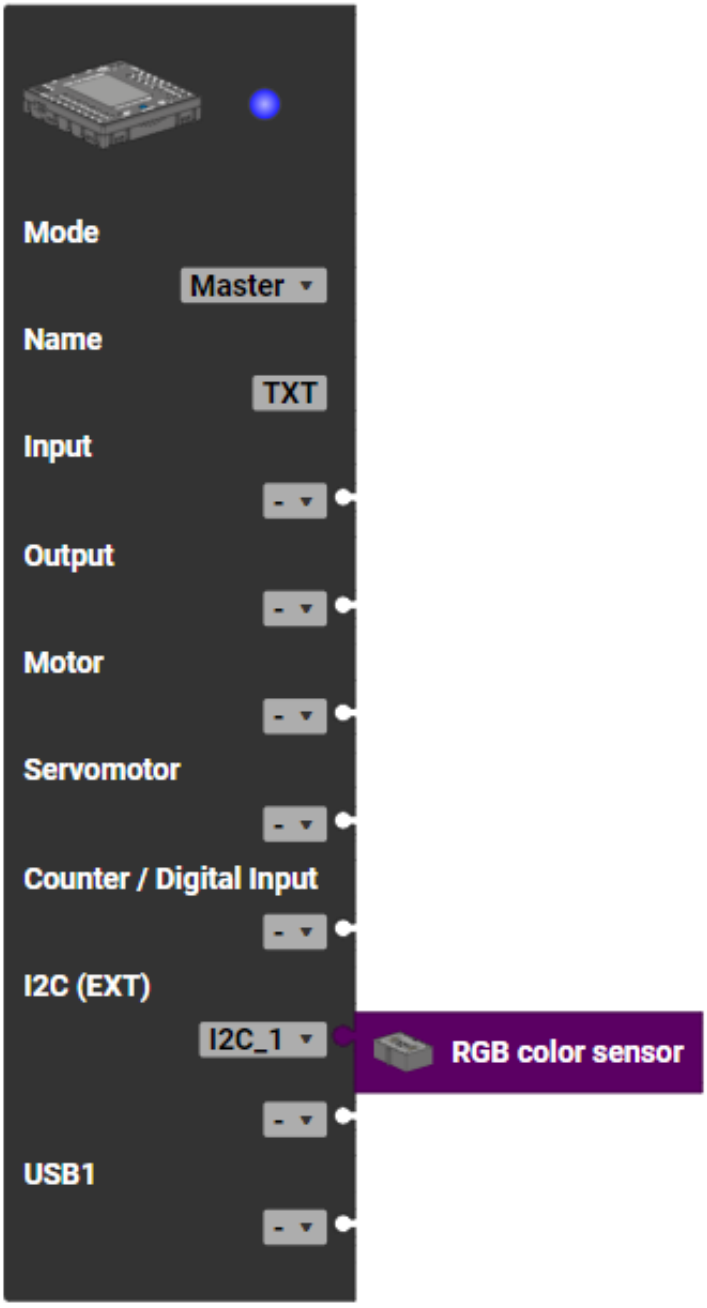
RGB Color Sensor

★★

The RGB color sensor (Art.-No. 213965) is connected via the I2C bus of the TXT 4.0 controller. To establish the connection, plug the sensor into the EXT1 or EXT2 port using a ribbon cable.

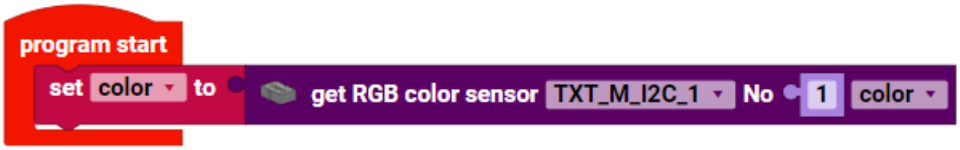
First Steps video:

- The fischertechnik RGB color sensor

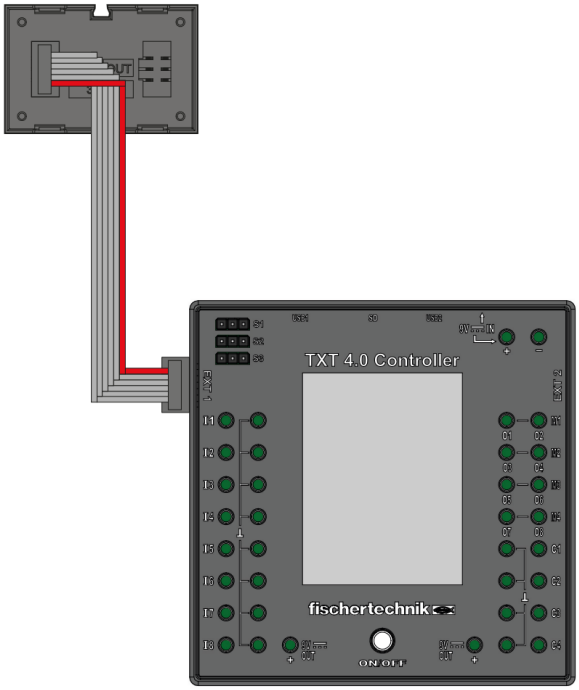


To use it, assign the corresponding sensor to the inputs in the controller configuration within ROBO Pro Coding. They can then be selected in the main program.

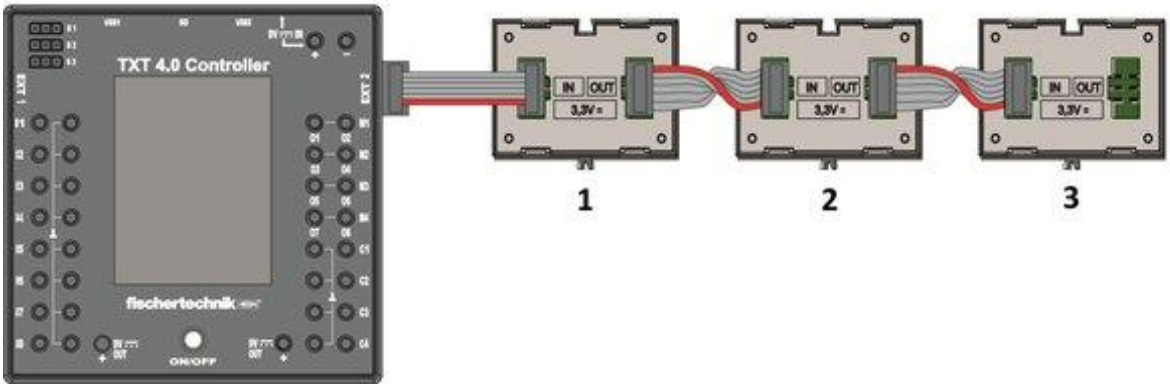
Unlike optical color sensors (which connect to the universal inputs), the RGB color sensor is designed for reliable detection of colored obstacles at short distances of approximately 1-5 cm.



The RGB color sensor for the STEM Coding RoboMission kit is connected as follows.



Multiple RGB sensors can be used simultaneously by connecting them in a daisy-chain configuration to the main sensor.



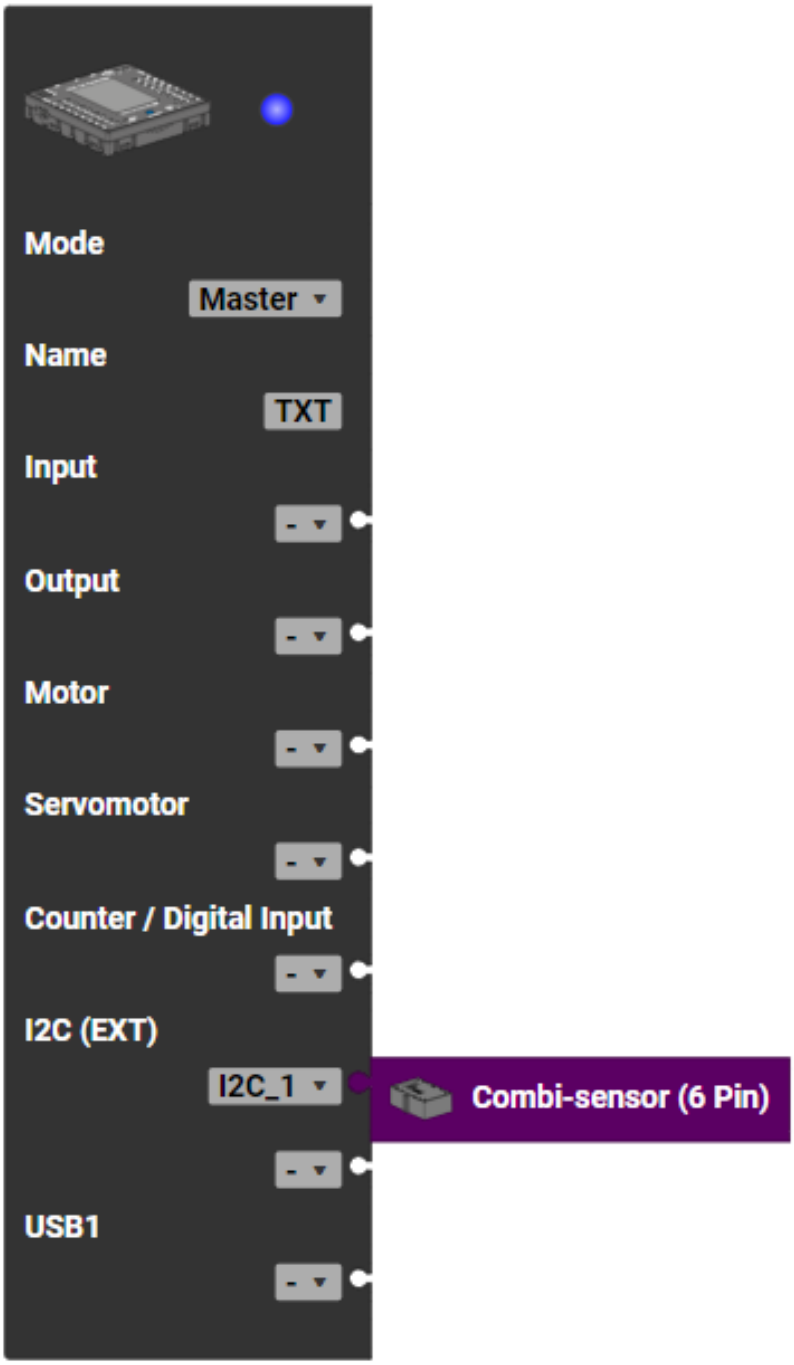
The example program **test_rgbcolorsensor_simple** shows how the sensor values can be read out.

Note: Ambient light or strong external light sources can interfere with the measurement accuracy of the RGB color sensor. For best results, avoid exposing the sensor to direct or fluctuating light during operation.

Combi Sensor

★★★

The combination sensor (Art.-No. 201257) is connected via the I2C bus of the TXT 4.0 controller. To establish the connection, plug the sensor into the EXT1 or EXT2 port using a ribbon cable.



The module contains **three sensor types**, each measuring motion or magnetic fields along **three axes**:

- **Accelerometer** (ax, ay, az)
- **Gyroscope** (rx, ry, rz)
- **Magnetometer** (mx, my, mz)

In ROBO Pro Coding, assign the sensor to the corresponding controller input. Afterwards, the values can be accessed directly in your program.

Sensor Values and Their Use in Robotics

3-Axis Accelerometer – ax, ay, az

Measures linear acceleration along the X, Y, and Z axes (m/s² or g).
Typical uses in robotics:

- Detecting **tilt or inclination**
- Monitoring **vibrations** from motors or gearboxes
- Recognizing **impacts or collisions**
- Rough estimation of **movement direction**

3-Axis Gyroscope – rx, ry, rz

Measures angular velocity (°/s or rad/s).
Typical uses in robotics:

- Measuring **rotational speed** of a robot or joint
- **Stabilizing** mobile robots
- Enabling **smooth and controlled turns**

3-Axis Magnetometer – mx, my, mz

Measures magnetic field strength (µT).
Typical uses in robotics:

- Determining the robot's **heading** relative to Earth's magnetic field (e.g., "facing north")
- **Correcting gyroscope drift** during long movements

Combined as a 9-DoF IMU (Inertial Measurement Unit)

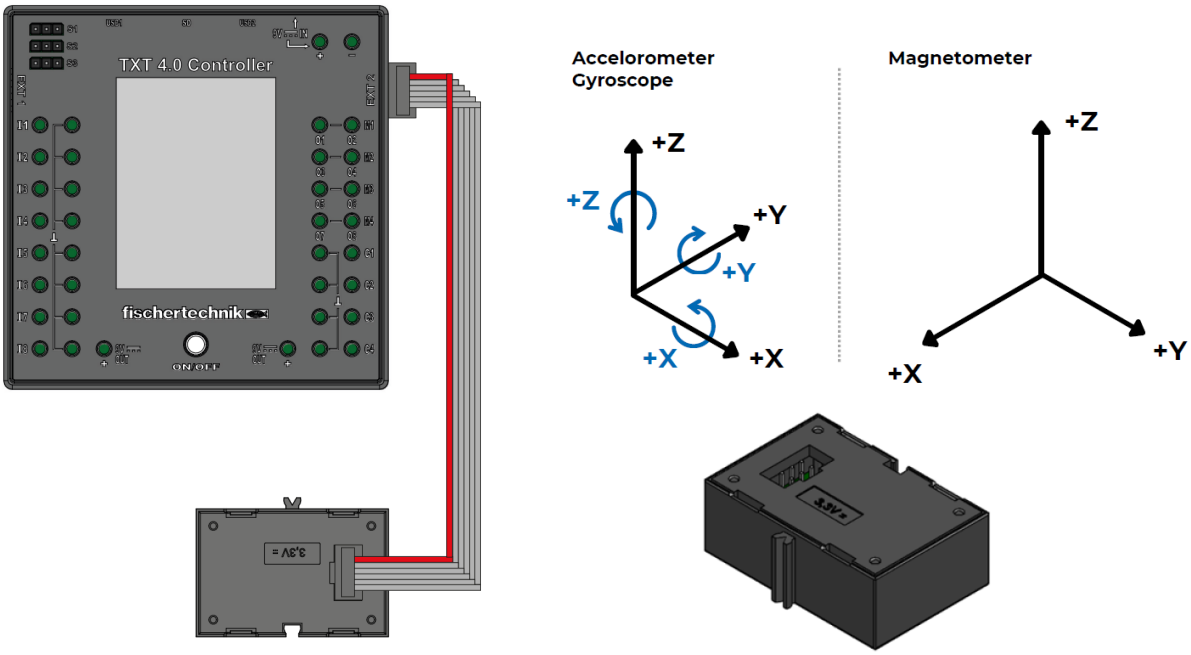
All nine values together:
ax, ay, az, rx, ry, rz, mx, my, mz

These can be combined (sensor fusion) to compute:

- **Orientation in 3D** (roll, pitch, yaw)
- **Stable navigation and balancing**
- Components of **localization** / SLAM (Simultaneous Localization and Mapping)

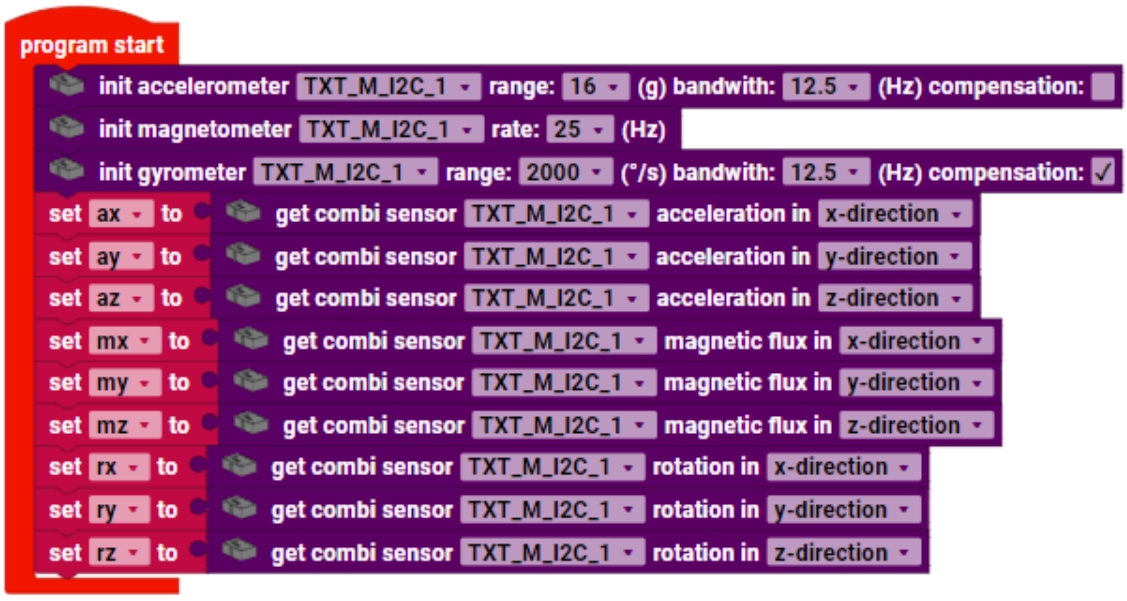
Note: Sensor fusion and orientation algorithms are complex and not part of this tutorial.

The combi sensor for the STEM Coding RoboMission kit is connected as shown.



The example program **test_combi_sensor_6pin** shows how the sensor values can be read out.

Note: The “compensation” parameters are not used with the 6-pin combi sensor; it applies only to the 10-pin combi sensor.



Important: The combination sensor only provides **raw values**. However, it can still be used effectively for:

- **Straight-line** driving
- **Precise turns** and rotation control
- **Tilt measurement** relative to gravity

Example Programs

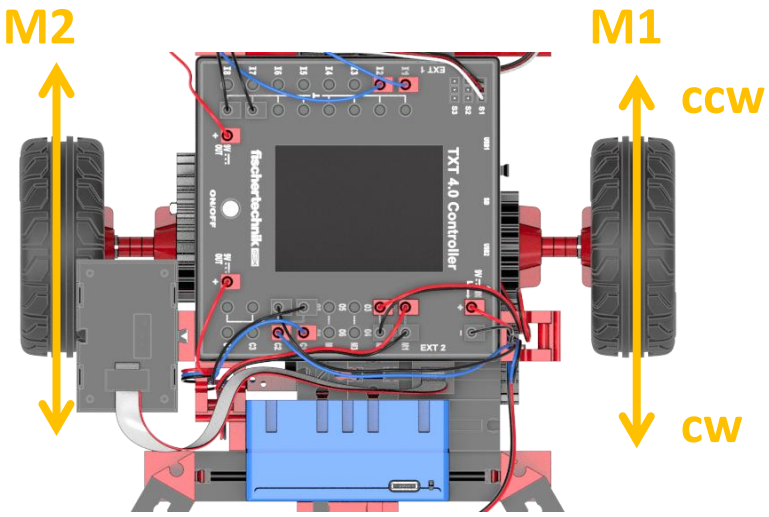
The following chapters present example programs that illustrate the practical application of key concepts. Each example includes a concise explanation, relevant code snippets, and guidance on how to adapt the program for various use cases. These examples are designed to deepen understanding and encourage hands-on experimentation.

Additionally, you can import the examples directly into **ROBO Pro Coding** from GitLab, making it easy to explore, modify, and integrate them into your own projects.

Differential Drive

★

Differential drive is a widely used locomotion method for mobile robots and vehicles with two independently driven wheels mounted on the same axis. By varying the speed and direction of each wheel, the robot can move forward, backward, and rotate.



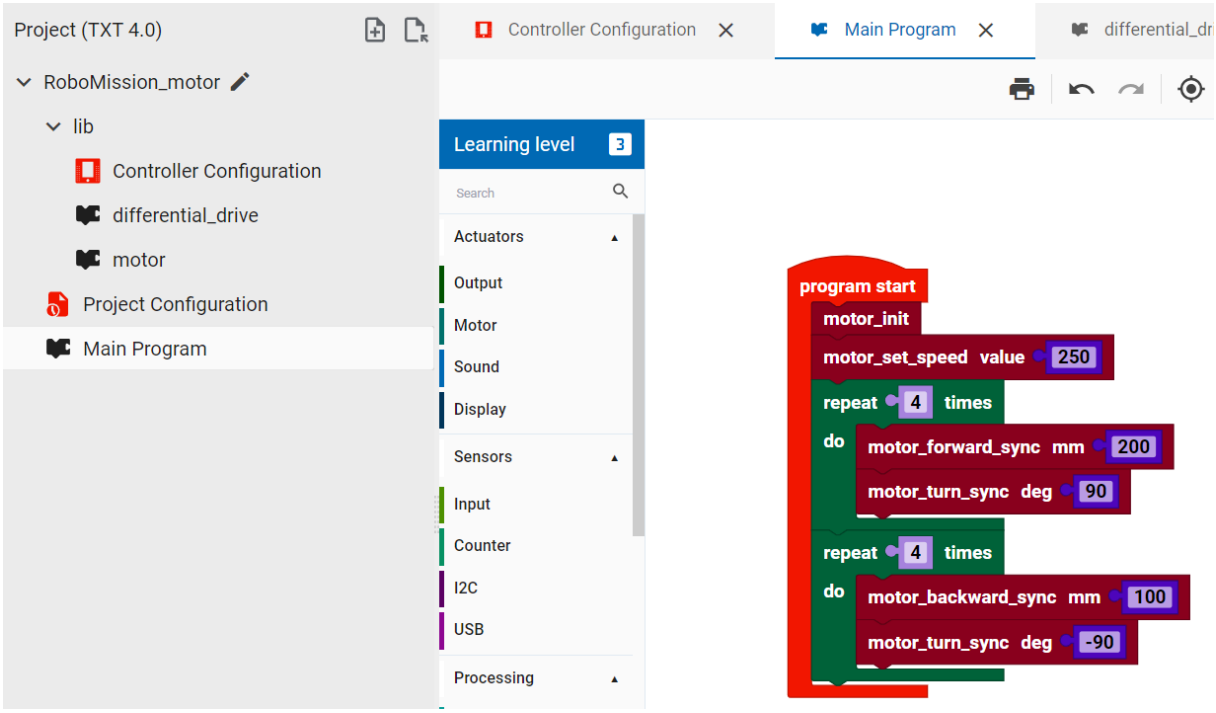
Basic Movement Principles:

- Both wheels move at the same speed and in the same direction
  → The robot moves straight.
- Wheels move at equal speed but in opposite directions
  → The robot rotates in place.
- One wheel moves faster than the other
  → The robot follows a curved path.

Motor Control as a Library

★★

The **RoboMission_motor** example demonstrates how to control the robot's encoder motors using synchronized motor operations.



Before exploring the program, familiarize yourself with how encoder motors work. Technical specifications can be found in the respective data sheets.

In the **motor** module, various functions are defined to enable synchronous motor control. This is essential for ensuring the robot moves precisely straight or rotates by a defined angle.

The **differential_drive** module contains the calculations required for differential drive control and will be explained in the next chapter.

Program Behavior

- The robot first moves forward in a square pattern (4 times: forward 200 mm, left turn 90°).
- Then, it moves backward in a smaller square pattern (4 times: backward 100 mm, right turn 90°).

Note: The actual movement of the robot may deviate from the programmed path due to mechanical tolerances, friction, or sensor inaccuracies. The calibration values in the initialization function can be adjusted to improve accuracy.

Differential Drive Calculation

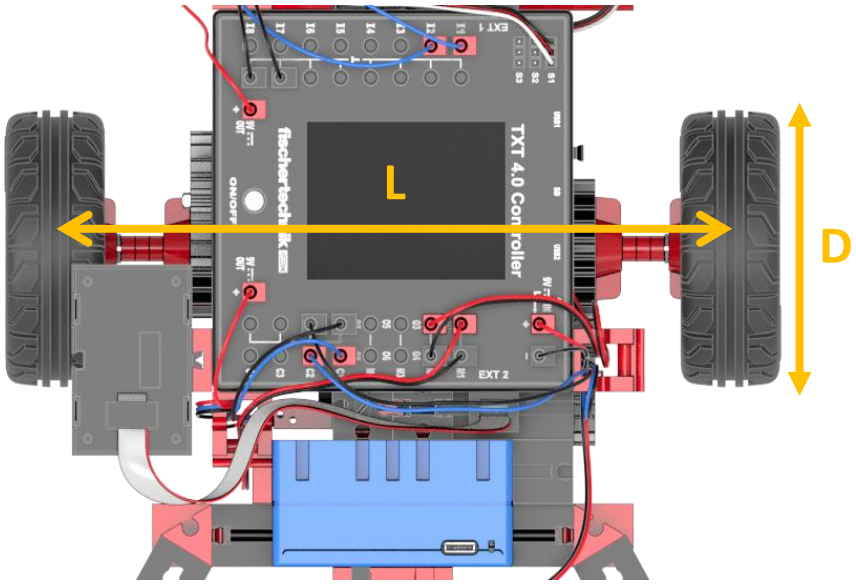
★★★

In a differential drive system, the robot moves by controlling the speed and rotation of two independent wheels. To achieve precise navigation, we need to calculate the **linear distance** traveled and the **rotation angle** based on encoder ticks. These calculations are essential for tasks such as path following, turning, and positioning in robotics.

The following example uses the specifications of the fischertechnik WRO RoboMission Model (2026):

- **Wheelbase L** = 169 mm = 0.169 m
- **Wheel diameter D** = 65 mm → wheel radius R = 32.5 mm = 0.0325 m
- **Encoder ticks per motor revolution T** = 63.9
- **Gear ratio G** = 1 (1:1)

Top:



Bottom:

Example Calculation

Input values:

- Encoder ticks left ΔL=100,
- Encoder ticks right ΔR=105

Distance per wheel:

$$s = \frac{\Delta}{T \cdot G} \cdot 2\pi R$$

For the left wheel:

$$s_L = \frac{100}{63.9 \cdot 1} \cdot 2\pi \cdot 0.0325 \approx 0.0314\,\text{m} = 3.14\,\text{cm}$$

For the right wheel:

$$s_R = \frac{105}{63.9 \cdot 1} \cdot 2\pi \cdot 0.0325 \approx 0.0330\,\text{m} = 3.30\,\text{cm}$$

Translation:

$$s = \frac{s_L + s_R}{2} = \frac{3.14 + 3.30}{2} = 3.22\,\text{cm}$$

Rotation:

$$\theta = \frac{s_R - s_L}{L} = \frac{0.0330 - 0.0314}{0.169} \approx 0.0095\,\text{rad} \approx 0.54^\circ$$

Effective Wheelbase

For robots with thick tires, the effective wheelbase differs from the rim-to-rim distance.

Let:

- d = geometric wheelbase
- w = tire thickness
- d_{eff} = effective wheelbase

A simple approximation is:

$$d_{\text{eff}} = d + \frac{w}{2}$$

Example:

$$d_{\text{eff}} = 169\,\text{mm} + \frac{42.5\,\text{mm}}{2} = 169\,\text{mm} + 21.25\,\text{mm} = \boxed{190.25\,\text{mm}}$$

Note: This is a simplified approximation. For precise navigation or mapping, experimental calibration is recommended (e.g., rotate on the spot and measure wheel paths).

Example Program

Run the ROBO Pro Coding program: **RoboMission_differential_drive**

Experiment with different parameter values and observe how the robot moves.

Important: The model parameters

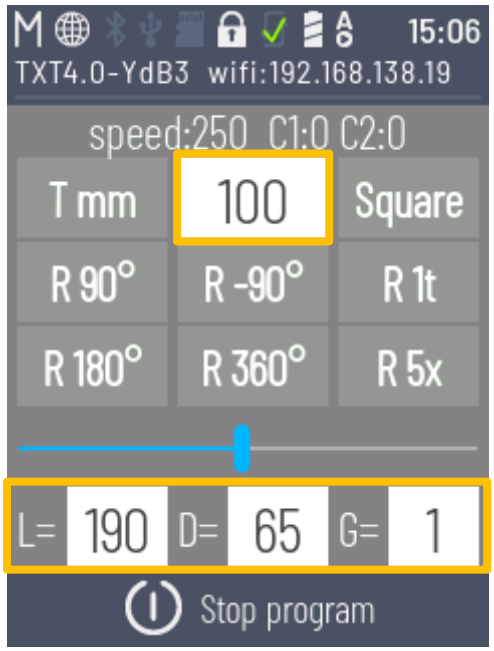
- wheelbase (L),
- wheel diameter (D), and
- gear ratio (G)

differ between the fischertechnik WRO RoboMission Models 2024 and 2026. Ensure you use the correct specifications for your model when performing

calculations or programming.

In the interface:

- The distance (in millimeters) is entered at the top (see screenshot).
- Model parameters are set at the bottom and typically remain unchanged.



Legend:

- **T**: Translation
- **R**: Rotation
- **Square**: Combined movement
- **R 1t**: Rotate by one encoder tick
- **R 5x**: Rotate five times
- **C1, C2**: Counter values
- **speed**: Motor speed (adjustable via slider)

Line Following Navigation

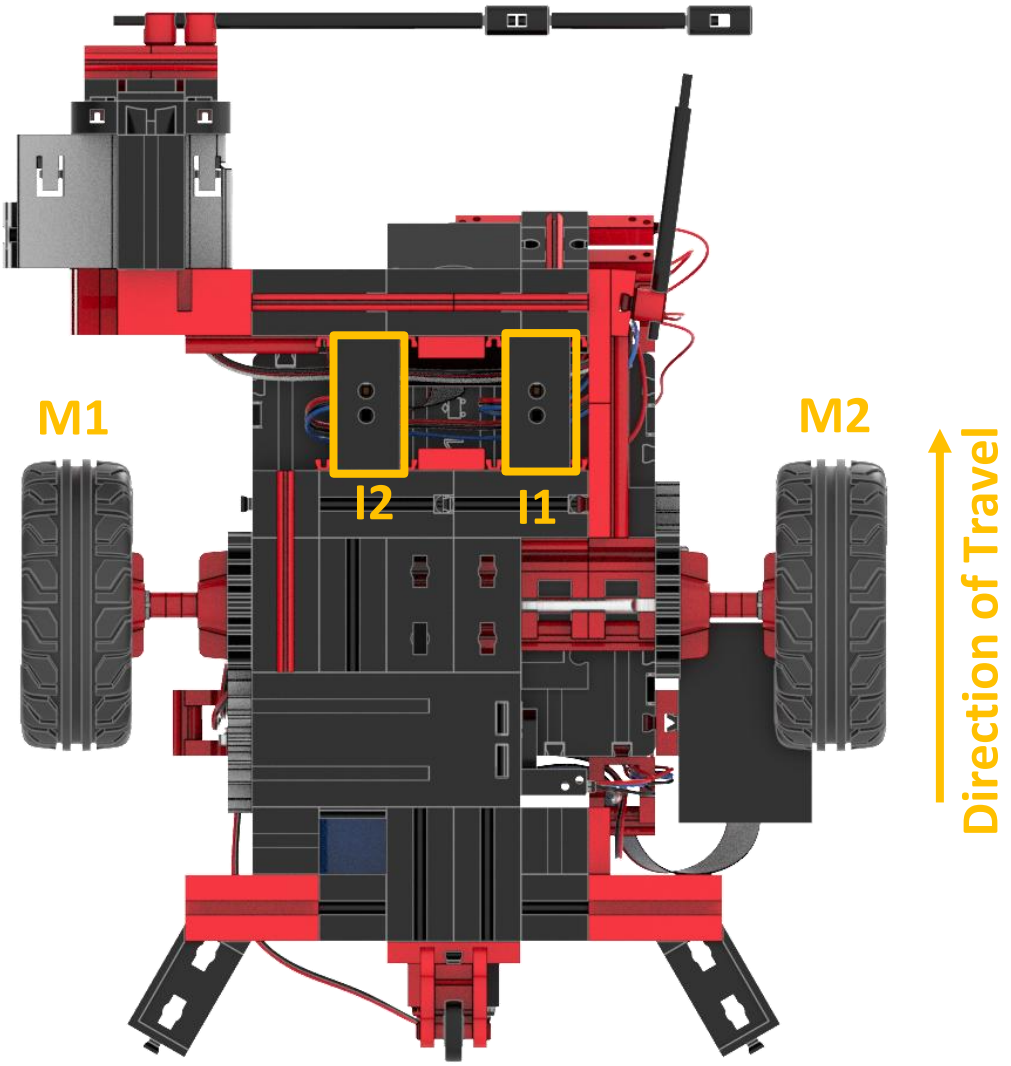
★★

For navigation tasks, it is often necessary for the robot to follow a predefined line on the ground. This functionality can be implemented using line sensors.

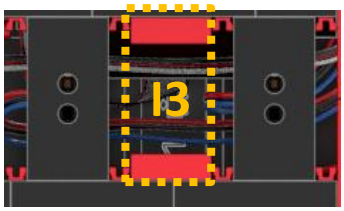
Depending on the complexity and accuracy required, different numbers of sensors can be used:

- **1 Sensor** – Typically positioned to detect the right or left edge of the line.
- **2 Sensors** – Allows detection of both edges of the line for better control.
- **3 Sensors** – Provides more precise line tracking and error correction. Note: the third sensor (Art.-No. 128599, 213966 or 222966) is not included in the standard set.

The following illustration shows the possible sensor positions:



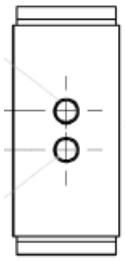
Note: Installation of a third optional sensor (not included in the kit) in the middle to improve detection:

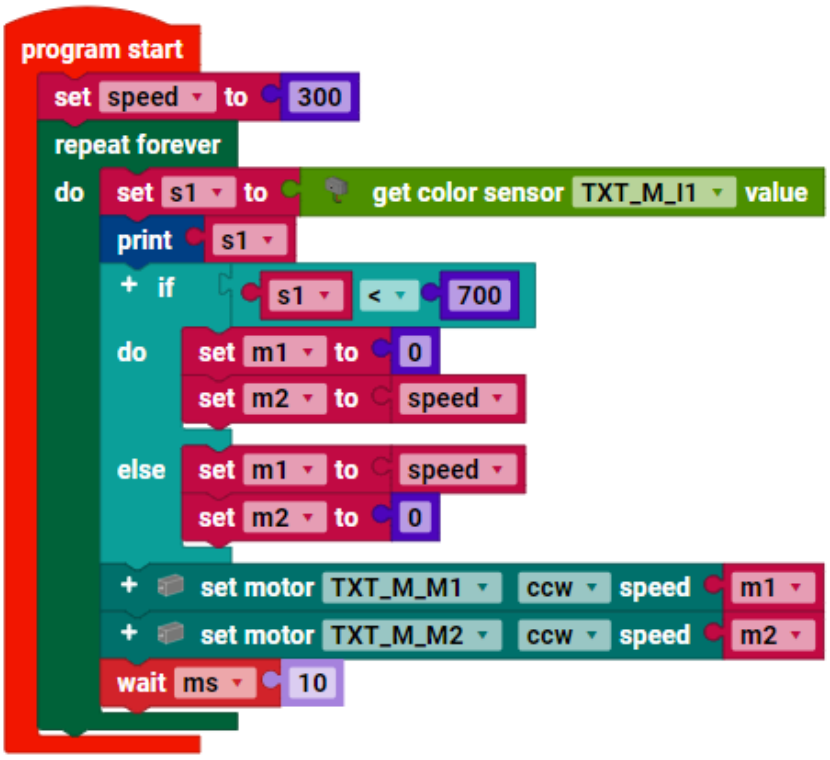


Example Programs

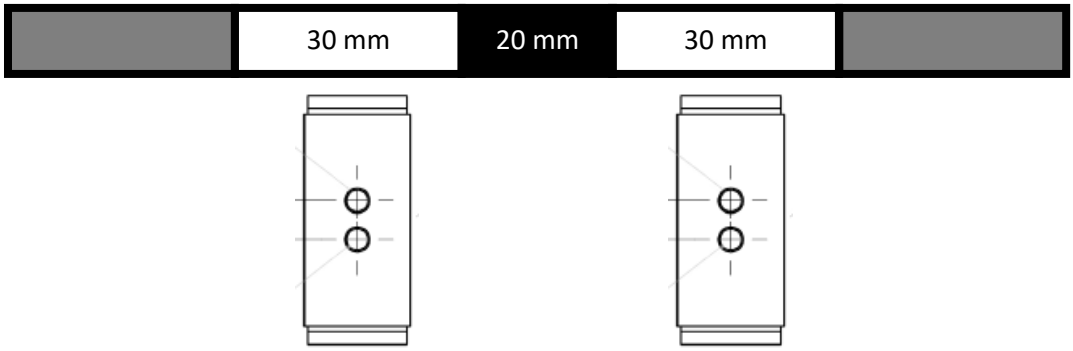
The next examples demonstrate how a simple line follower can be implemented using optical color sensors connected to the universal inputs I1-I3.

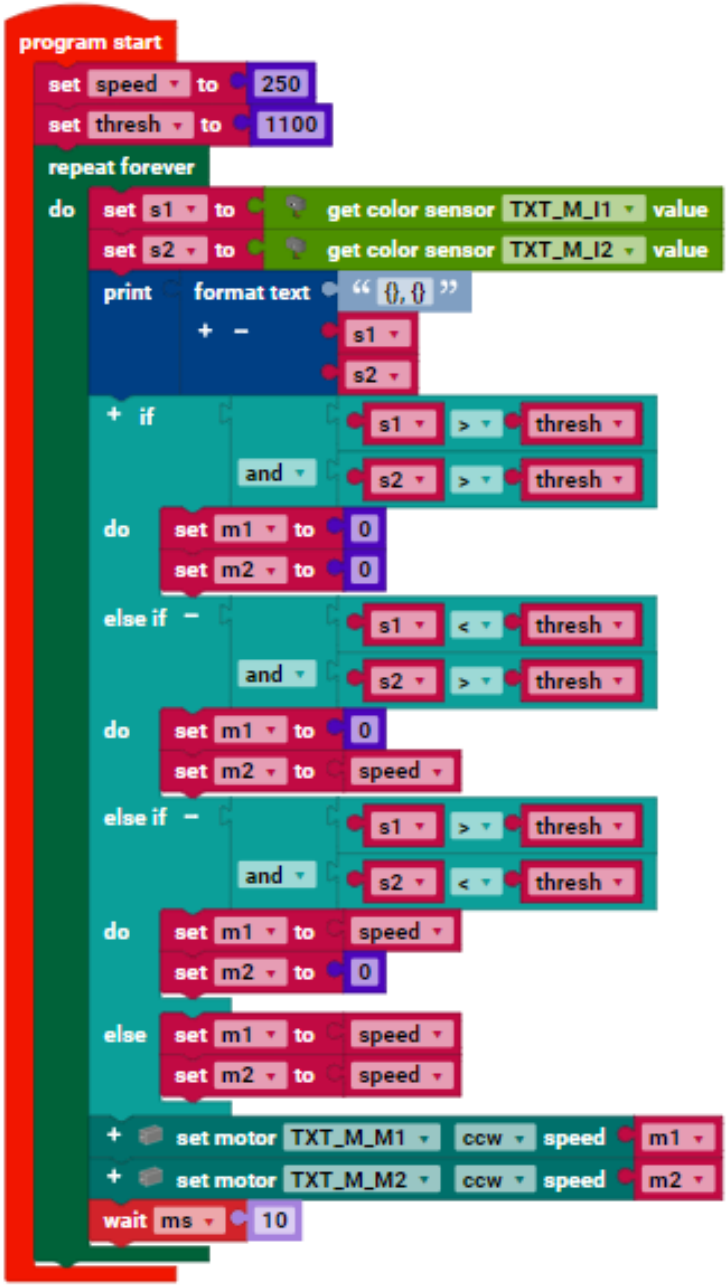
1 Sensor (RoboMission_1sline)





2 Sensors (RoboMission_2sline)

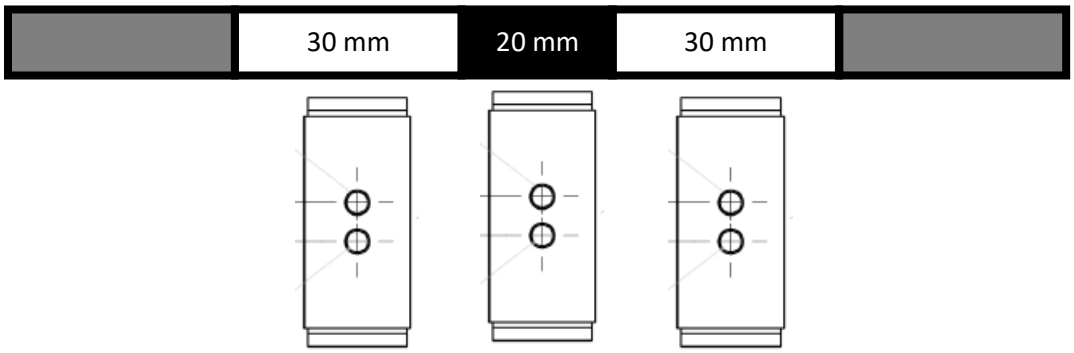


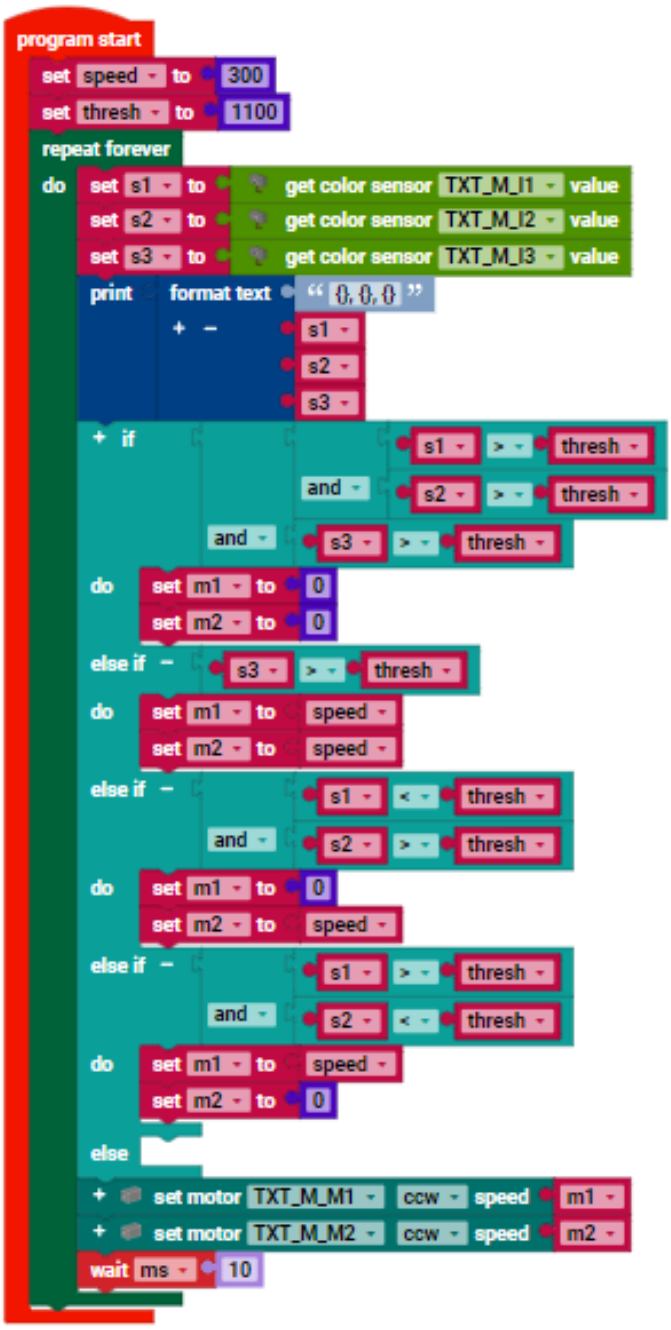


Legend:

- **thresh:** limit value for black/white detection

3 Sensors (RoboMission_3sline)





Legend:

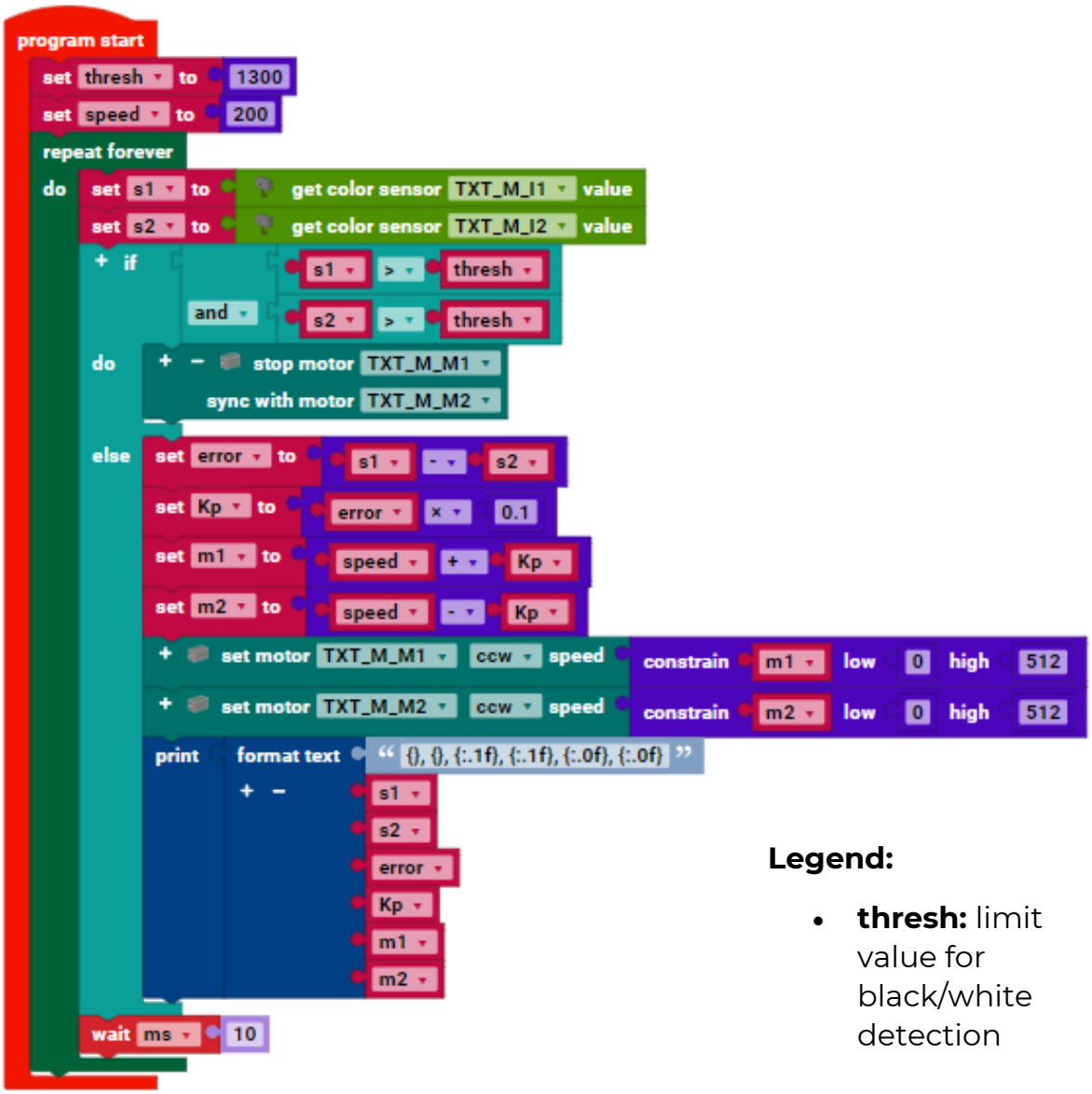
- **thresh:** limit value for black/white detection

P-based Line-Following Control

★★★

The line follower can be further improved by using a **P** (Proportional) or **PID** (Proportional–Integral–Derivative) controller, which enables the robot to stay on the line even at high speeds. Implementing such a controller requires an understanding of control theory, which will not be covered in detail in this tutorial.

The examples **RoboMission_2sline_p** and **RoboMission_3sline_p** demonstrate how a **P controller** can be used.



Legend:

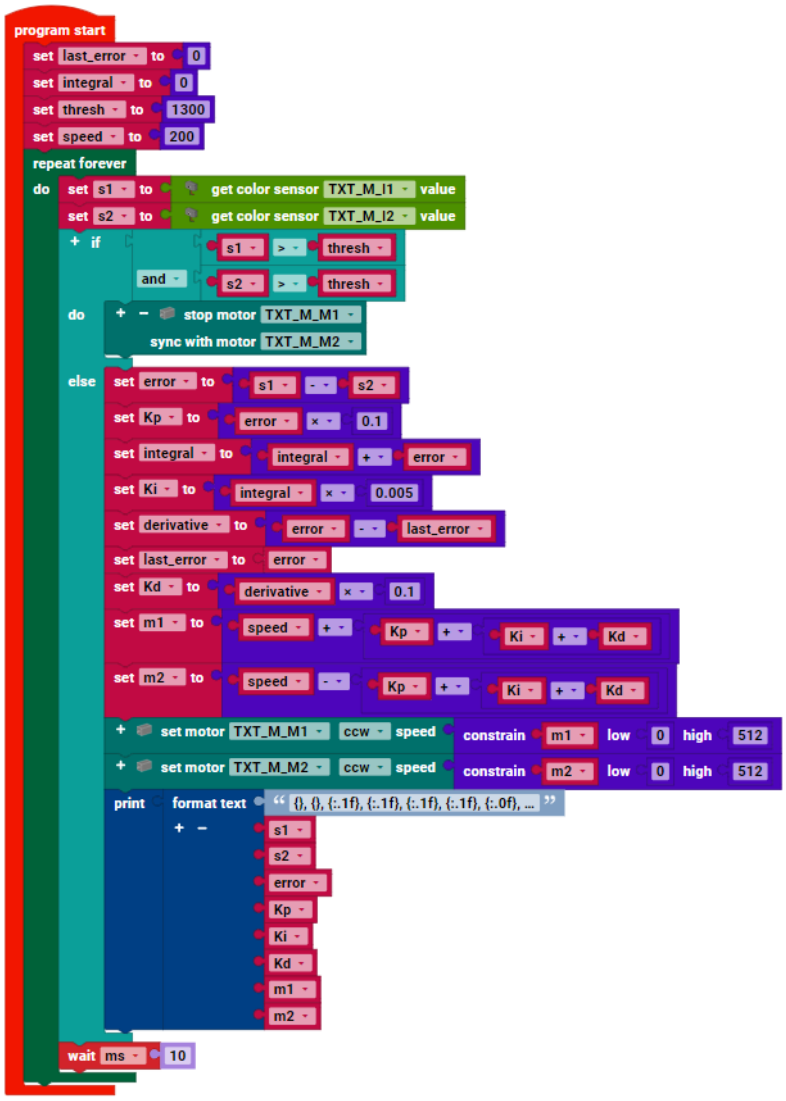
- **thresh:** limit value for black/white detection

Please adjust the **Kp** value accordingly:

- For a straight line: **Kp ≈ 0.1**
- For a curved line: **Kp ≈ 2**

★★★★

The additional example **RoboMission_2sline_pid** shows how a **PID controller** can be implemented. This will not be explained in this tutorial. Please refer to additional tutorials for more details.



Legend:

- **thresh:** limit value for black/white detection

The parameters **Kp**, **Ki**, and **Kd** must be set. Determining suitable values for these constants typically requires experimentation and can be time-consuming. For an introduction to PID control and guidance on choosing parameters, please refer to the relevant external literature.

Important:

- If the motor speed is too high, the robot will lose the line quickly.
- If the motor speed is too low, the motors may lack sufficient power when the battery charge is low, or the robot will move very slowly along the line.

Gyro Control – Straight Movement

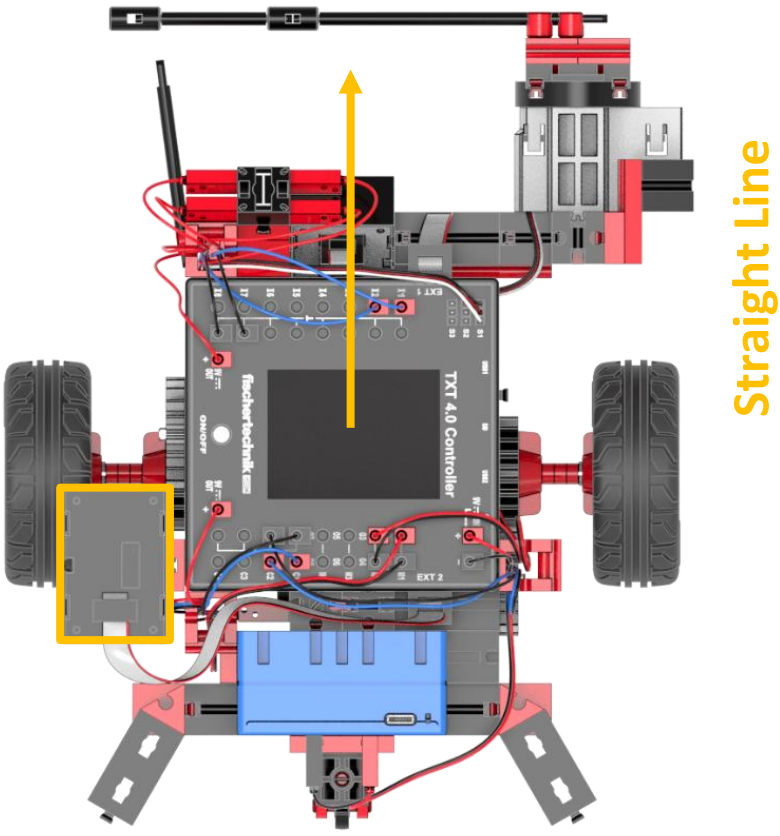
★★★

For navigation tasks, it is often essential that the robot moves in a perfectly straight line, even when external disturbances such as uneven surfaces, wheel slip, or minor mechanical inaccuracies occur.

A **gyroscope sensor** can be used to measure the robot's **angular velocity** and calculate its **orientation**. By constantly checking this heading, the robot can notice when it starts to drift and adjust itself to keep moving in a straight line.

How It Works

- The gyroscope detects rotational movement around the vertical axis.
- The controller integrates angular velocity over time to determine the current heading.
- If the heading deviates from the target (e.g., 0°), motor speeds are adjusted to compensate.



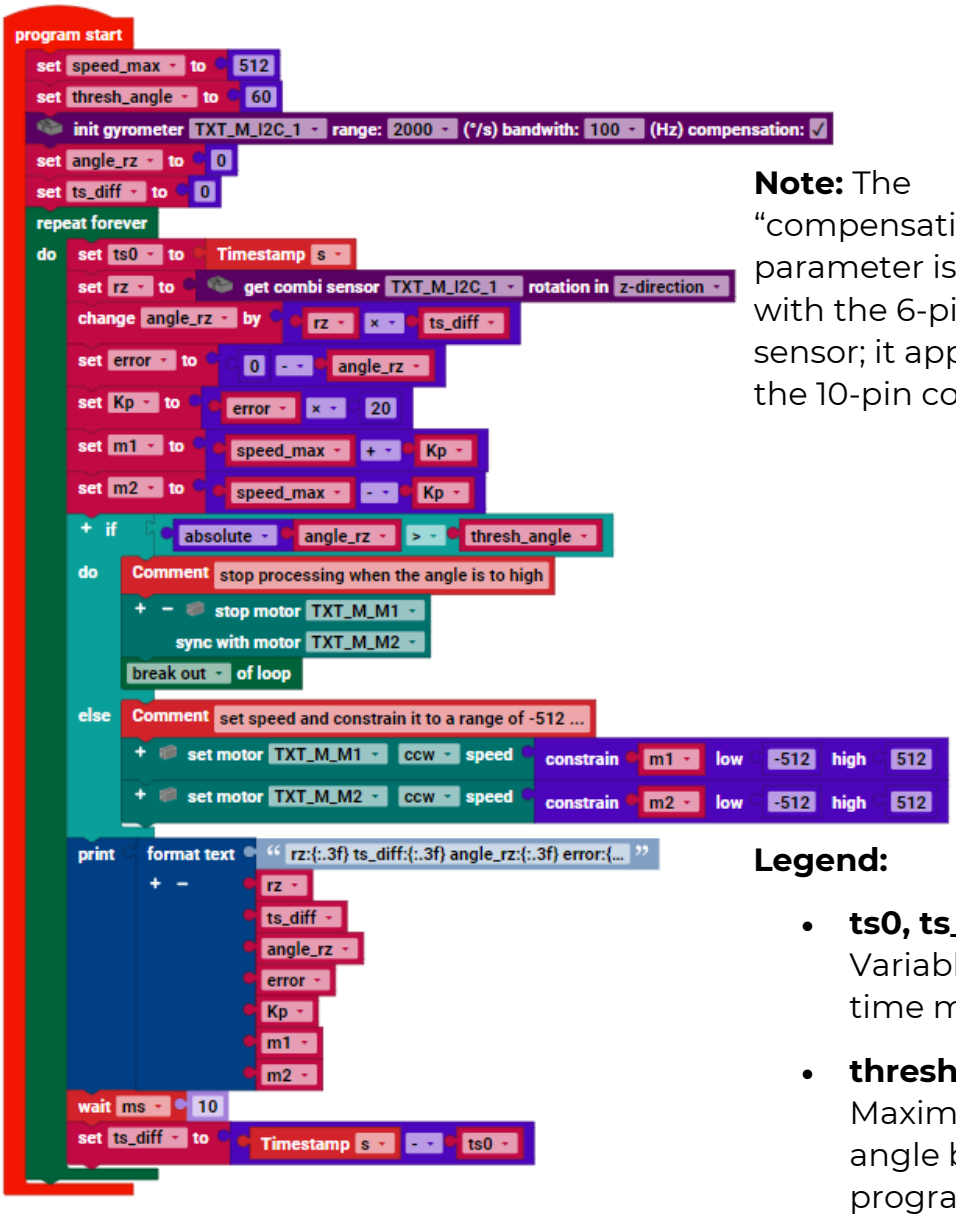
Example: P-Control for Straight Movement (*RoboMission_gyro_p*)

The following program demonstrates how to use gyro data to maintain a straight trajectory.

This approach significantly improves straight-line accuracy, especially on uneven surfaces or when wheel slip occurs.

Implementation Steps

1. Initialize the gyro sensor
2. Read angular velocity (**rz**) at regular intervals (10 ms).
3. Calculate heading (**angle_rz**) by integrating angular velocity.
4. Apply correction using a control algorithm (**P-Control**).



Note: The "compensation" parameter is not used with the 6-pin combi sensor; it applies only to the 10-pin combi sensor.

Legend:

- **ts0, ts_diff:** Variables used for time measurement
- **thresh_angle:** Maximum allowed angle before the program stops

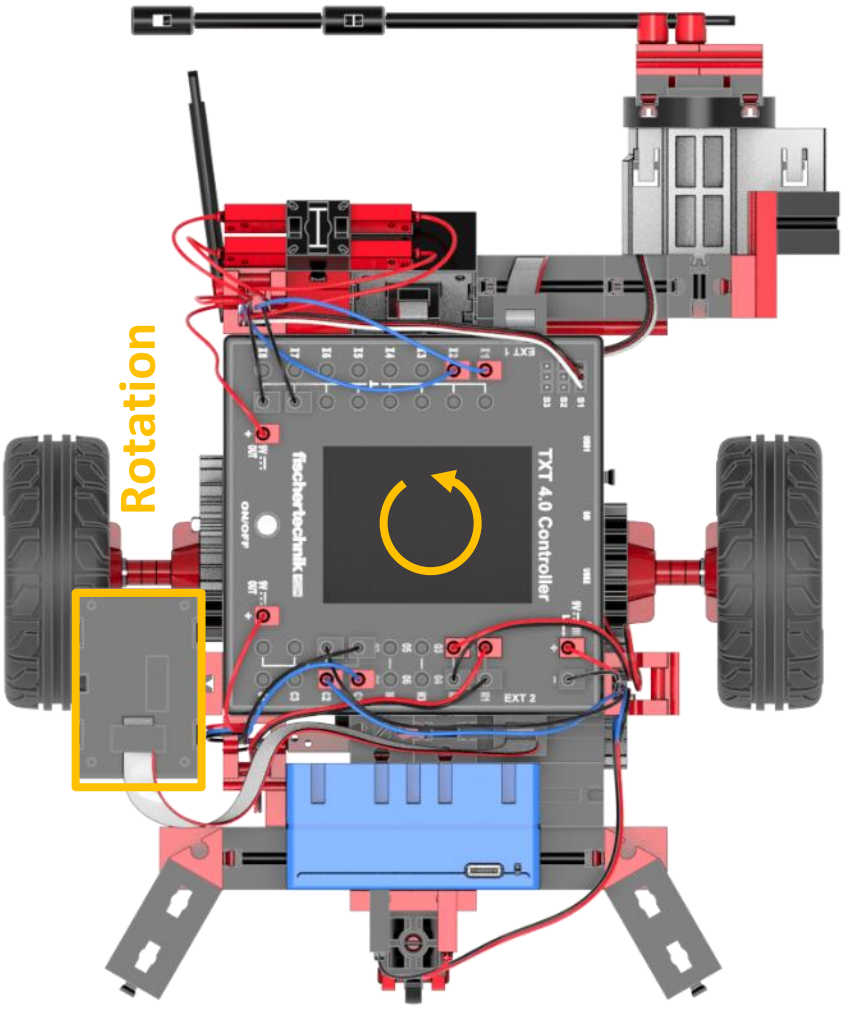
Gyro Control – Predefined Rotation

★★★

For navigation tasks, it is often essential that the robot performs precise rotations, such as turning exactly 90° for accurate path planning.

How It Works

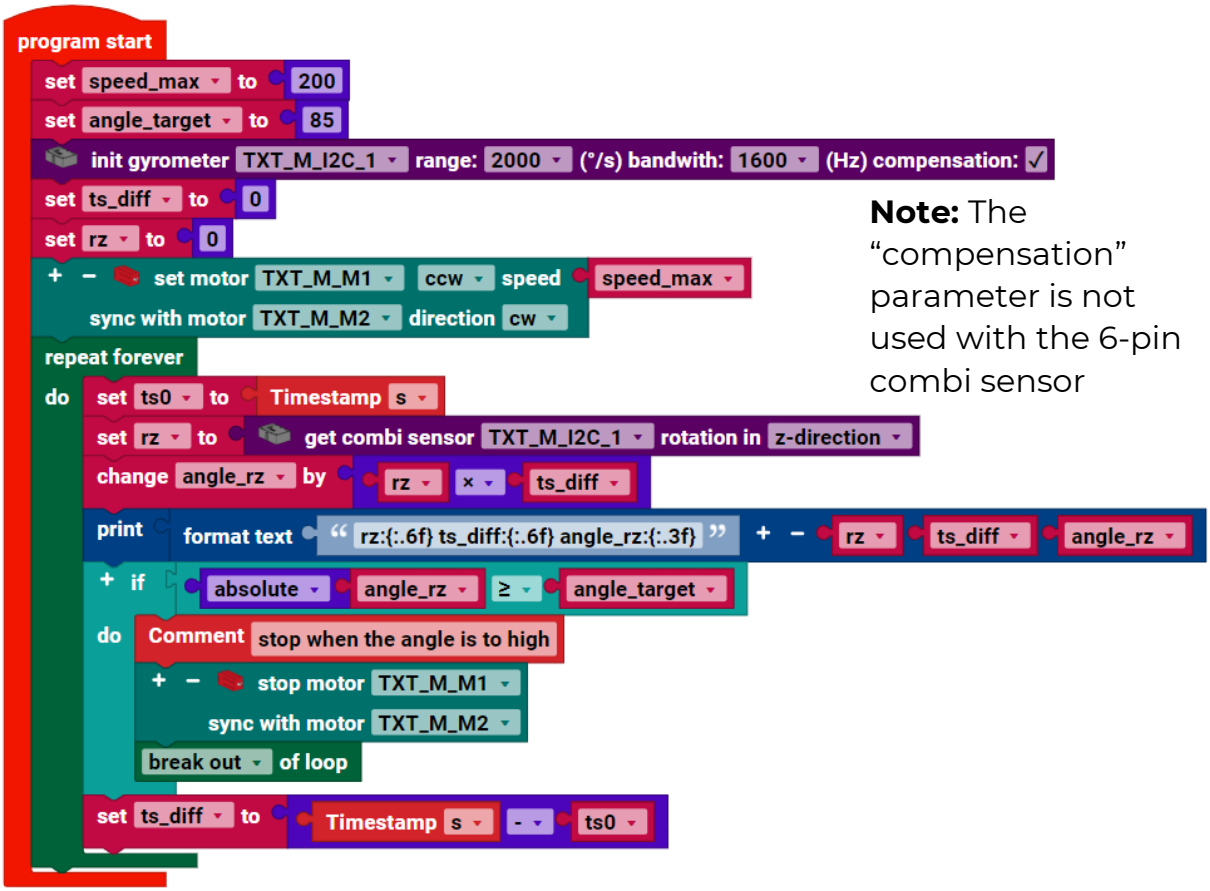
- The gyroscope measures angular velocity.
- The controller integrates this data to track the rotation angle.
- The robot stops turning when the target angle (e.g., 90°) is reached.



Example: Precise 90° Turn (*RoboMission_gyro_rot90*)

This example shows how the robot performs a controlled rotation using the gyroscope:

- Set the target angle (e.g., 90° for a right turn).
- Start rotating the robot by running the motors in opposite directions.
- Continuously monitor the integrated gyro angle.
- Stop the motors once the target angle is reached.



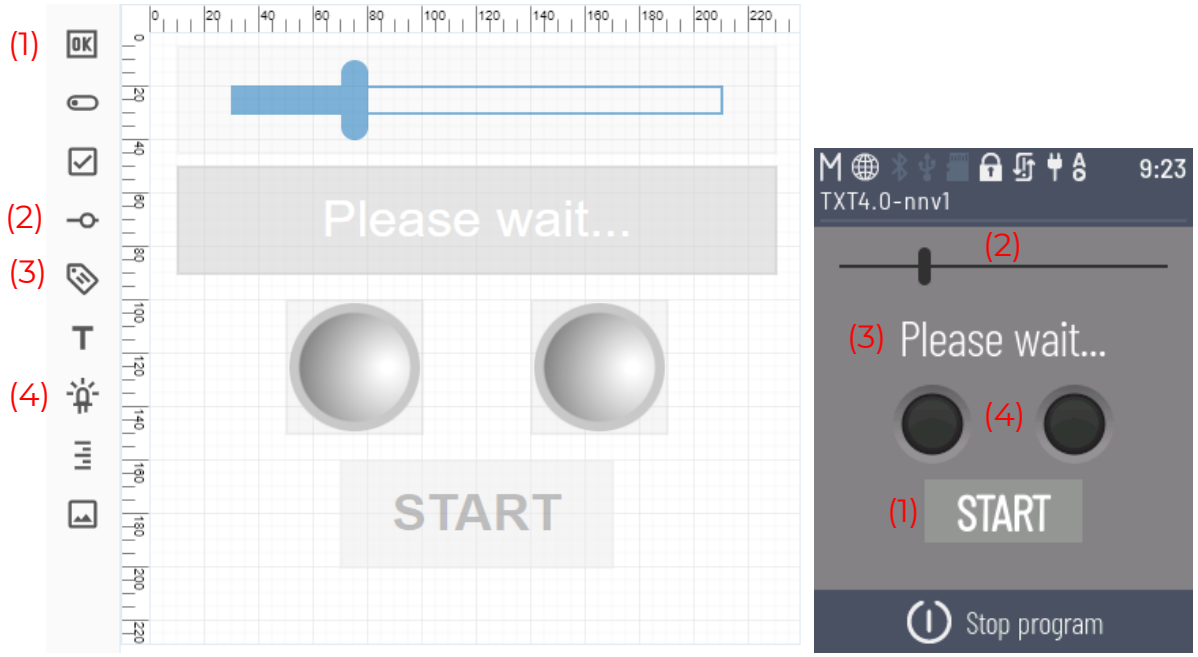
Note: The "compensation" parameter is not used with the 6-pin combi sensor

Note: To achieve the exact angle, you may need to slightly reduce the target value — for example, use 85 instead of 90.

TXT 4.0 Display

★★

The example **RoboMission_display_start** demonstrates how to create a graphical user interface (GUI) on the TXT 4.0 controller's display using ROBO Pro Coding. The GUI editor in ROBO Pro Coding provides various interface elements, accessible from the left sidebar.



The following elements are used in this example:

Button (1): Starting the program sequence

Slider (2): Setting variable values

Label (3): Output of text

Status indicator (4): Output of binary states

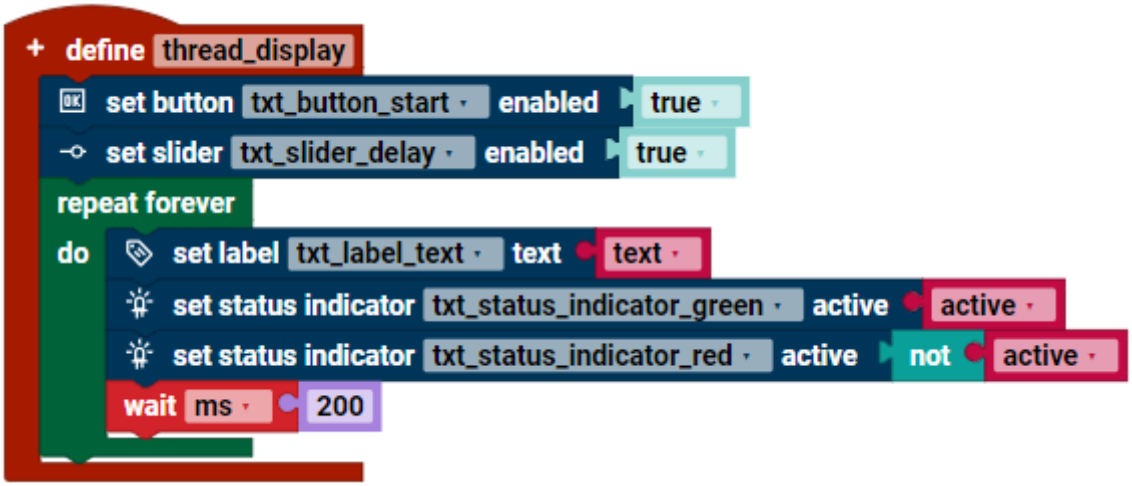
On the left, you see the definition in the GUI editor; on the right, the corresponding representation on the TXT 4.0 display.

The interface on the display can be fully customized. Font size and style can be adjusted in the element properties.

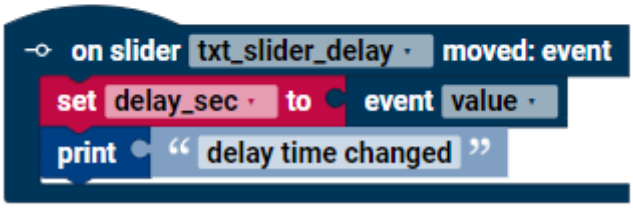
To enable interaction and display program states, the program sequence must be defined accordingly. It is recommended to use a separate thread (endless loop) for updating the display. Since human vision is relatively slow, the display should be refreshed every 200 ms to 1 s at most. A dedicated

display thread also reduces the load on the main loop, which often controls the robot and requires very short response times.

Loading the interface usually takes a few seconds. To prevent elements from being activated during initialization, disable all action-triggering elements in their properties at the start. Once the display thread is initialized, these elements can be re-enabled using **set...enabled**.



Callback functions must be defined for all actions (e.g., **“on button...clicked”**, **“on slider...moved”**). These callbacks should set variables for the corresponding events, which can then be evaluated in the main program.

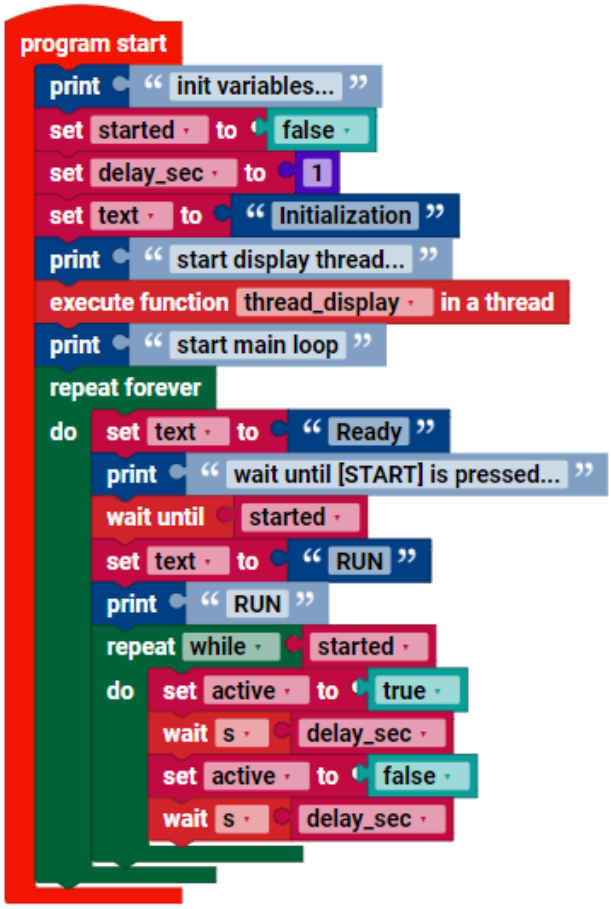


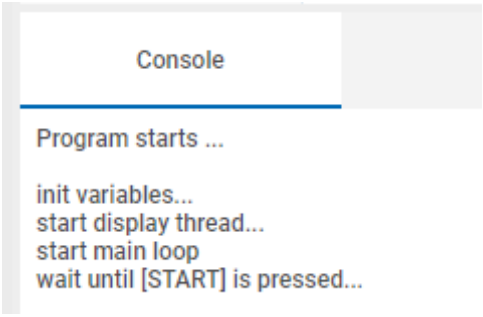
Important:

- In callback functions, only call short, non-blocking functions. Blocking calls can cause callbacks to interfere with each other.

Program Flow

The main loop might look like this:





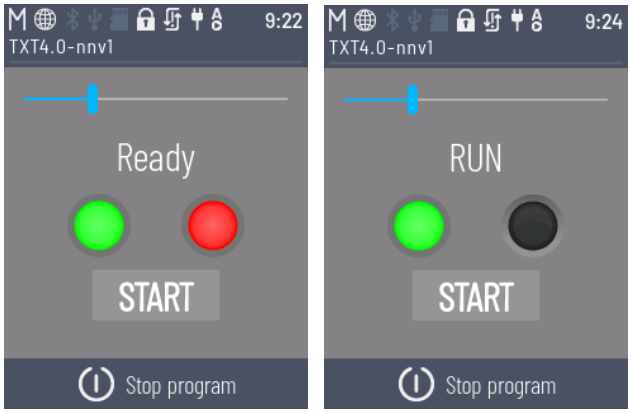
To better understand the program, it is often helpful to include **print**(...) statements in addition to display output. When the program runs in ROBO Pro Coding, console outputs from print() can help analyze when specific instructions were executed. However, avoid print() in control loops with short response times (e.g., 10 ms), as it can slow down execution.

Example Output

When the program runs, the TXT 4.0 display shows status changes such as:

- **“Please wait...”** → **“Ready”** → **“RUN”**

For example, the **START** button can be used so that the robot begins its program sequence only after START is pressed.



Measuring Battery Voltage

★★★

The charge level of the battery connected to the TXT 4.0 controller can significantly affect the current supplied to the motors and, consequently, the robot's performance.

To monitor low voltage conditions, the controller provides an internal voltage input that enables reading the battery voltage in volts.

In the **test_battery** example, both the voltage and the estimated state of charge are retrieved using a Python function.

External I2C Interface

★★★★

The TXT 4.0 controller features two external I2C connectors, EXT1 and EXT2, both located on the external I2C bus. Refer to the operating manual for detailed information.

The I2C power supply is 3.3 V. All connected sensors must be compatible with 3.3 V or require a level shifter.

Further information and programming examples are available on GitHub:

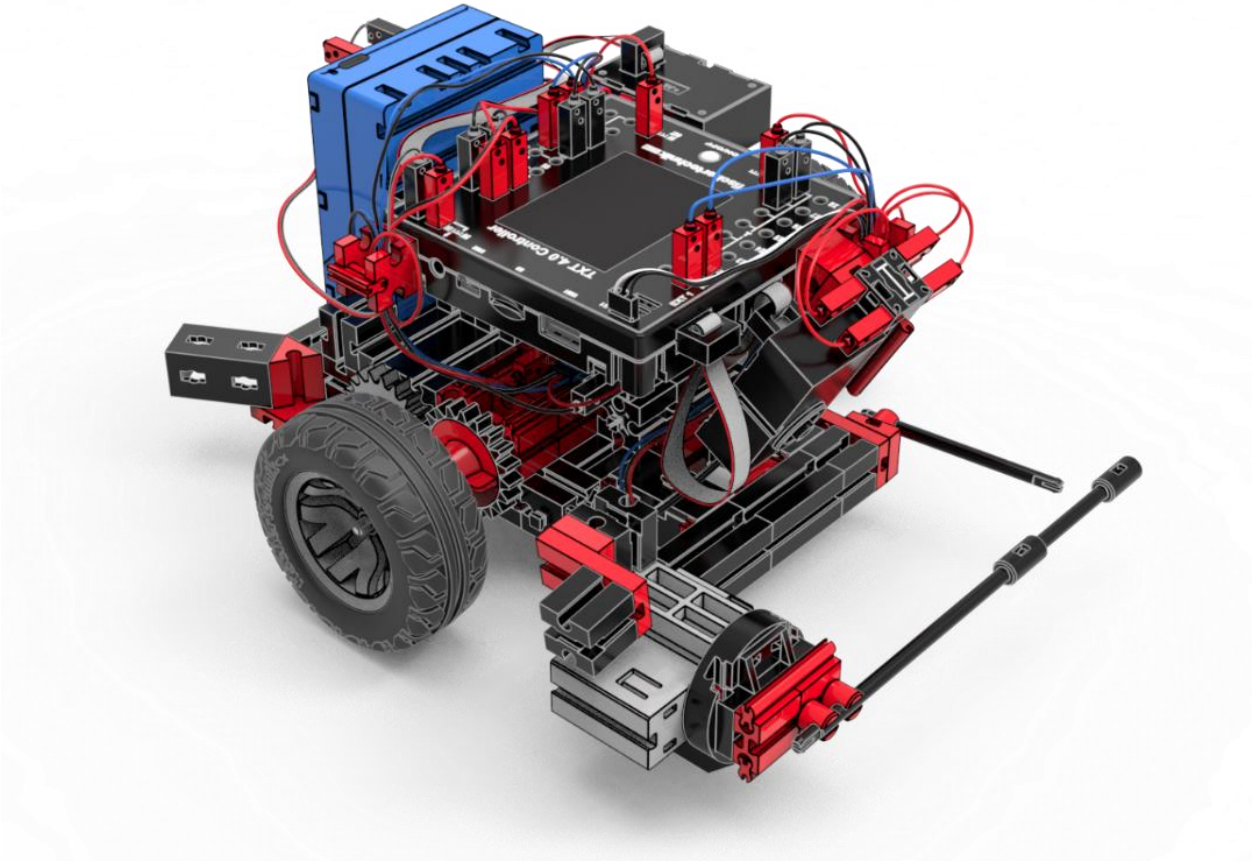
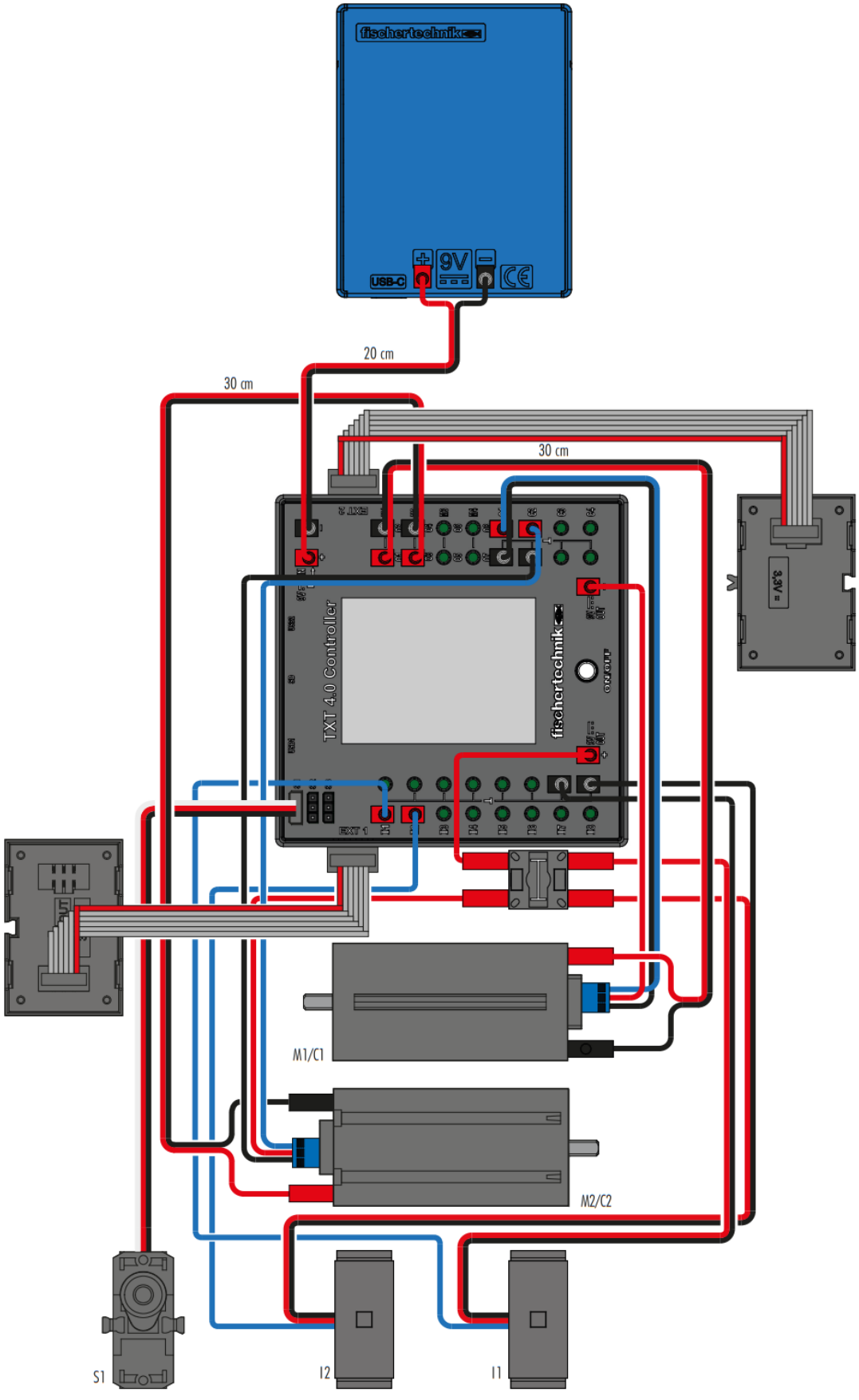
https://github.com/fischertechnik/TXT40-Controller-I2C

What's Next

Beyond the STEM Coding RoboMission set, fischertechnik provides a wide range of additional components and individual parts for extended projects.

Explore more on the official fischertechnik website:

http://www.fischertechnik.de/en.



Connection Plan