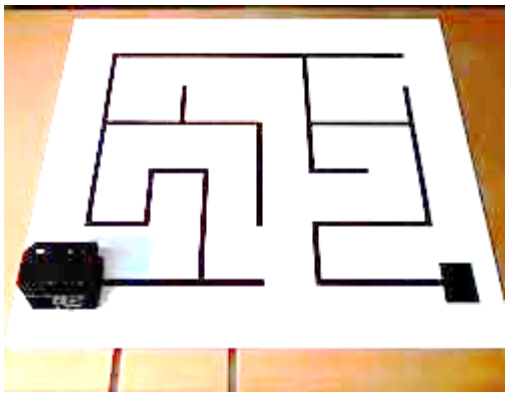
Ein Labyrinth besteht aus vielen Wegen, die verschiedene Richtungsänderungen haben:

es gibt Kurven nach rechts, nach links, Kreuzungen und Sackgassen – und hoffentlich 1 oder mehrere Ziele!



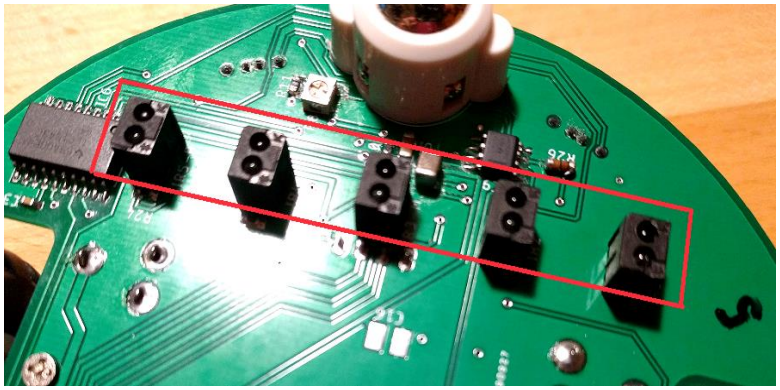
Für unser Labyrinth verwenden wir schwarze Linien auf weißem Untergrund. Mit passenden Abstandssensoren könnte unser Programm den Roboter auch ein Labyrinth mit Wänden lösen lassen (diese Sensoren sind aber komplizierter als unsere Linien-Sensoren).

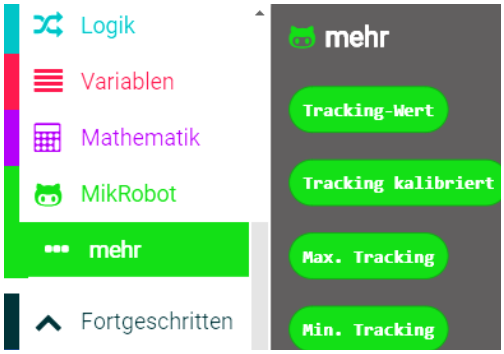
<u>Mit welchem Trick (oder besser gesagt „Regel“) kann ein Roboter den richtigen Weg finden?</u>

Wir verwenden die „Linke-Hand“-Regel: in einem Labyrinth, welches keine Schleifen enthält, findet man das Ziel, indem man das Labyrinth so entlang geht, dass die linke Hand immer an der Wand entlang streift.

<u>Was müssen wir dazu programmieren?</u>

Für das Erkennen des Weges verwenden wir die 5 „Tracking“-Sensoren unseres mik:robot. Diese befinden sich auf der Unterseite und messen, wieviel Licht vom Boden reflektiert (zurückgestrahlt) wird.





Der weiße Untergrund gibt eine große Zahl, die schwarze Linie eine kleine Zahl.

Damit die 5 Sensoren nicht unterschiedliche Zahlen bei gleichem Untergrund messen, müssen wir „Kalibrieren“.

Dazu muss unser Roboter „Mik“ beim Einschalten auf dem schwarzen Streifen stehen: er dreht sich dann hin und her, macht viele Messungen, merkt sich für jeden Sensor die kleinste und größte Zahl und rechnet dann ein „Tracking kalibriert“ für jeden Sensor aus (siehe Kurs 4 „Licht“, wo wir den Mittelwert aus den Zahlen „oben“ und „unten“ ausgerechnet haben).

Ein Mittelwert von 300 sollte für die meisten Mik passen; man kann aber auch mit einem Kabel die Zahlen zum PC schicken (siehe Kurs 4 „Licht“, Seiten 4 und 5) und diese auf weiß und schwarz ausprobieren.

Zuerst programmieren wir die Grundaufgabe: Mik muss der Linie nachfahren können.

Die Werte der 5 Sensoren sind in einem „Array“ (eine Liste mit 5 Elementen) gespeichert. In Computer-Programmen werden oft Arrays verwendet, weil man damit viele Variablen gleichzeitig bearbeiten kann.

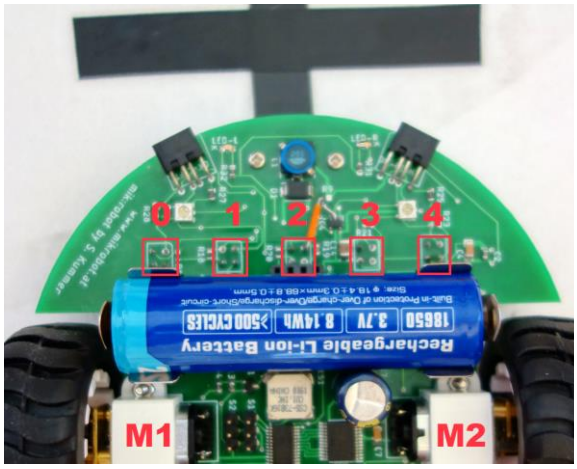
Aus der Bibliothek „Arrays“ (dazu muss man in Schritt 1 auf den schwarzen Pfeil neben „Fortgeschritten“ drücken) benötigen wir den Block „Liste rufe Wert ab bei…“

Wir müssen in der „dauerhaft“-Schleife jedes Mal die „Liste“ auf die aktuellen Messwerte des „Tracking“-Sensors setzen!

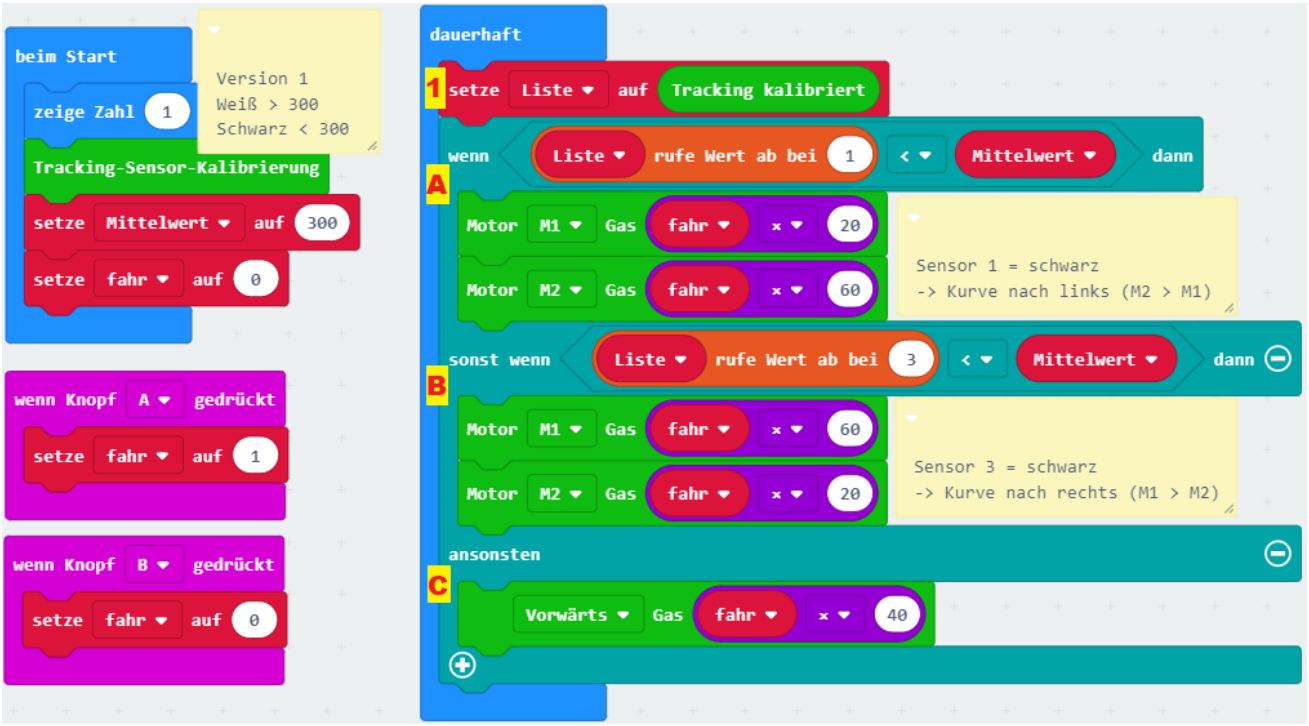
Im Normalfall ist die schwarze Linie in der Mitte des Roboters – das ist Sensor 2 (siehe Bild unten).

Wir fragen die beiden Sensoren rechts und links neben Sensor 2 ab:

- A: Wenn Sensor 1 eine Zahl < Mittel-wert (kleiner als der Mittelwert, also schwarz) sieht, dann ist Mik zu weit rechts und wir müssen eine Kurve nach links machen.
- Für die Linkskurve muss der rechte Motor (M2) schneller drehen als M1.
- B: Sieht Sensor 3 die schwarze Linie, dann müssen wir eine Kurve nach rechts machen: M1 > M2
- C: Wenn weder Sensor 1 noch Sensor 3 die Linie sehen, dann fahren wir gerade („wenn A … sonst wenn B … ansonsten C …)

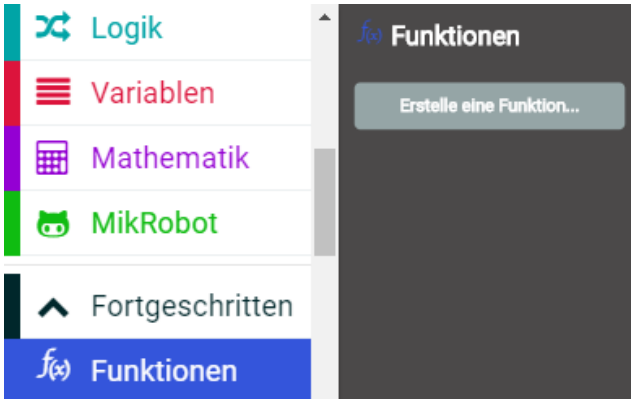


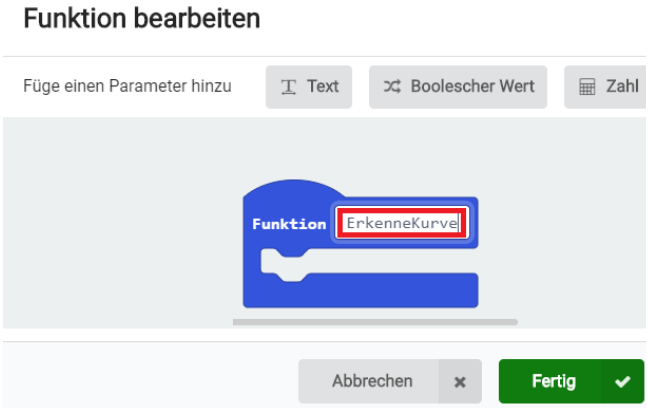
Damit der Roboter nicht immer fährt, haben wir eine Variable „fahr“. Diese kann 0 oder 1 sein. Weil wir als „Gas“ eine Zahl mit der Variablen „fahr“ multiplizieren und z.B. 0 mal 20 = 0 („null mal 20 ist null) ist, bleibt der Roboter bei „fahr = 0“ stehen. Mit „fahr = 1“ fährt er sein Programm.



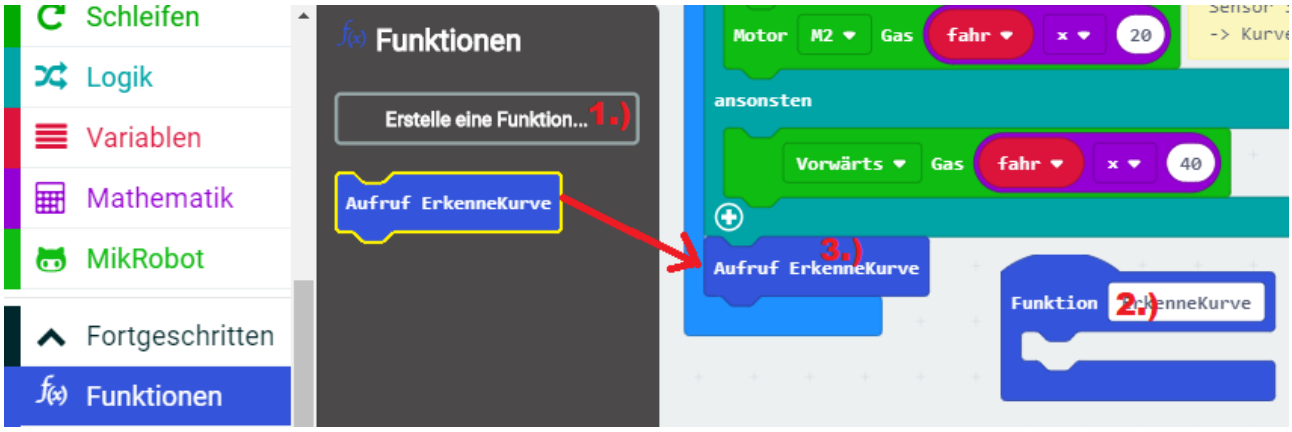
Im nächsten Schritt müssen wir die verschiedenen Kurven und Kreuzungen mit den Sensoren erkennen.

Damit unser Programm nicht zu groß und unübersichtlich wird, werden wir „Funktionen“ verwenden. Eine Funktion kann viele Befehle und Blöcke enthalten, die wir mit einem „Aufruf“ starten können. Dabei können wir auch „Parameter“ mitgeben: Zahlen oder einen Text, die in der Funktion wie eine Variable verwendet werden können.

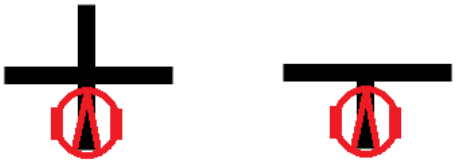




Jetzt werden wir den neuen Funktionen-Block in unser Programm einfügen:

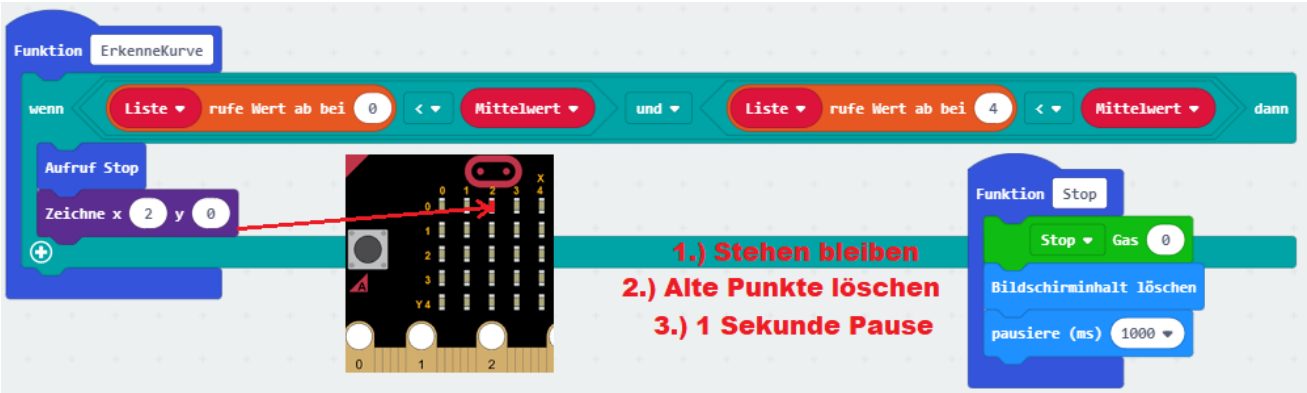


In die Funktion „ErkenneKurve“ werden wir nun die Befehle zum Kurven-Erkennen einfügen.



Wenn der Roboter auf diese beiden Kreuzungen (+ und T) zufährt, sehen diese für den Roboter gleich aus. Wir werden die beiden äußeren Sensoren (0 und 4) abfragen und wenn beide gleichzeitig „schwarz“ sehen, haben wir diese Kreuzung erkannt („wenn A und B“ – beide Blöcke finden wir unter „Logik“).

Um unser Programm zu testen, werden wir dann auf dem micro:bit anzeigen, dass wir ein „T“ erkannt haben und stehen bleiben. Das Stehen-Bleiben hilft uns beim Testen. Wir können unseren Roboter immer wieder neu hinstellen, neu starten und ausprobieren, ob das Programm diese Kreuzung in allen Fällen (z.B. auch wenn der Roboter schief auf die Kreuzung zufährt) erkennt. Für das Stehen-Bleiben verwenden wir wieder eine „Funktion“ (weil wir diesen Befehl noch oft brauchen werden). Für das Anzeigen werden wir nur Punkte zeichnen, weil der micro:bit beim Anzeigen von Text und Zahlen eine Pause macht – das würde unser Programm zu langsam machen. Wir verwenden dazu den Block „Zeichne x y“ auf der Bibliothek „LED“:



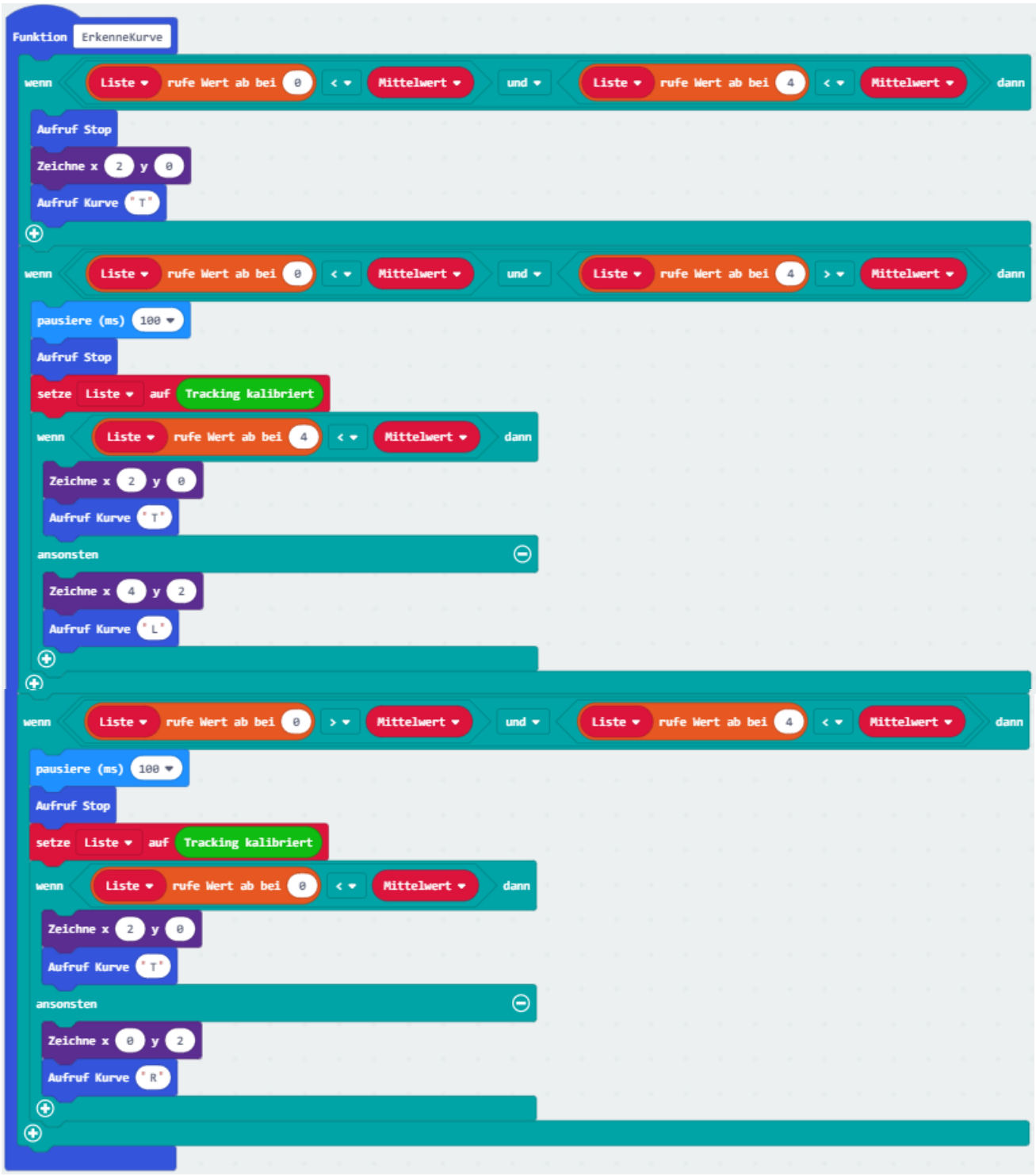
Wenn unser Roboter schief auf das T zufährt, dann wird zuerst nur einer der beiden äußeren Sensoren die schwarze Linie erkennen.

Hier hilft ein Trick: wir werden in diesen 2 Fällen nicht sofort stehen bleiben, sondern noch ganz kurze Zeit weiterfahren, dann stehen bleiben und nochmals nachmessen (dazu die Variable „Liste“ auf „Tracking kalibriert“ setzen).

Wenn dann der Sensor, der zuerst noch „weiß“ gesehen hat, „schwarz“ sieht, dann ist es eine T-Kreuzung. Wenn der Sensor dann noch immer „weiß“ sieht, dann ist es eine Kurve nach links „L“ (oder rechts „R“).

Bei unserer Geschwindigkeit (40) passt bei den meisten Robotern eine Pause von 100. Beim Testen kann man auch andere Werte ausprobieren. Wenn der Roboter nicht schön der Linie folgt, kann man auch die Werte für das Gas der Motoren ändern (z.B. 50 oder 70 statt 60).

So sieht die fertige Funktion „ErkenneKurve“ aus (bereits mit den Aufrufen zum Kurven-Fahren; diese kann man zu Testen noch weglassen):



Die Funktion „Kurve“ hat einen Text-Parameter, diesen nennen wir „Richtung“.

Wir müssen uns aber noch überlegen, wie wir diese 2 Fälle lösen:

Geradeaus fahren

Rechts fahren

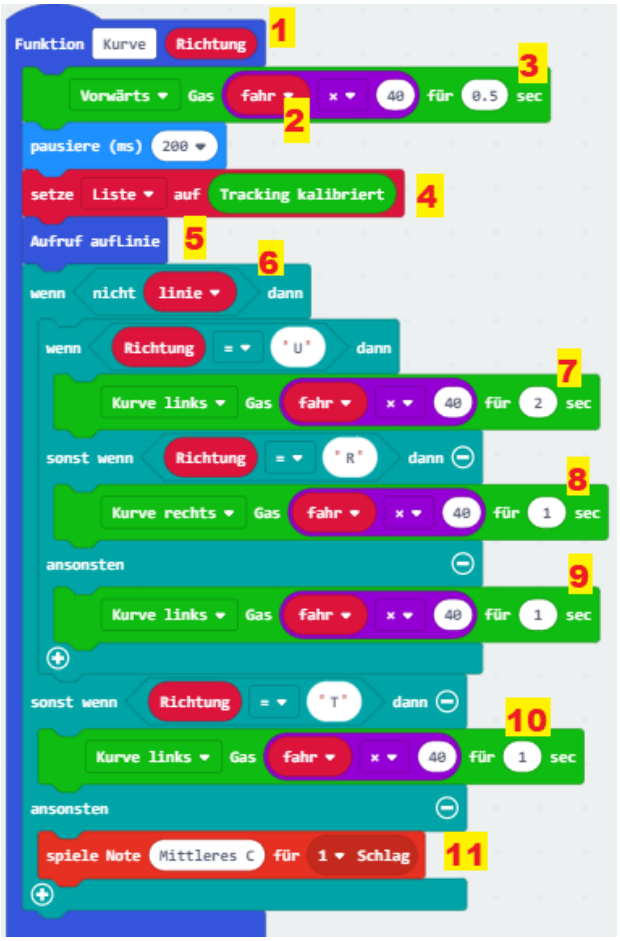
In beiden Fällen haben wir eine Kurve nach rechts („R“) erkannt. Wir müssen aber der Linie folgen, falls es einen geraden Weg gibt (wegen der „Linke-Hand“-Regel).

Deshalb fahren wir mit dem Roboter zuerst noch ein Stück gerade nach vorne und messen nochmals nach.

Auch für die Rechtskurven und Linkskurven dürfen wir uns erst drehen, wenn der Roboter mit seiner Mitte genau über der Kreuzung steht! Hier musst du ausprobieren, ob eine halbe Sekunde (0.5) mit Gas 40 passt!

Und eine Funktion „aufLinie“, die kontrolliert, ob wir noch auf der Linie stehen, hilft uns auch im Programm. Dazu erstellen wir noch eine Variable „linie“ (ist 1 wenn wir die Linie unter den Sensoren 1, 2 oder 3 – also den mittleren 3 Sensoren – sehen und ist 0, wenn wir die Linie verloren haben).

Falls ein Labyrinth eine Sackgasse hat (die schwarze Linie hört einfach auf), dann soll unser Roboter umdrehen (Kurve „U“).



Erklärungen zur Funktion „Kurve“:

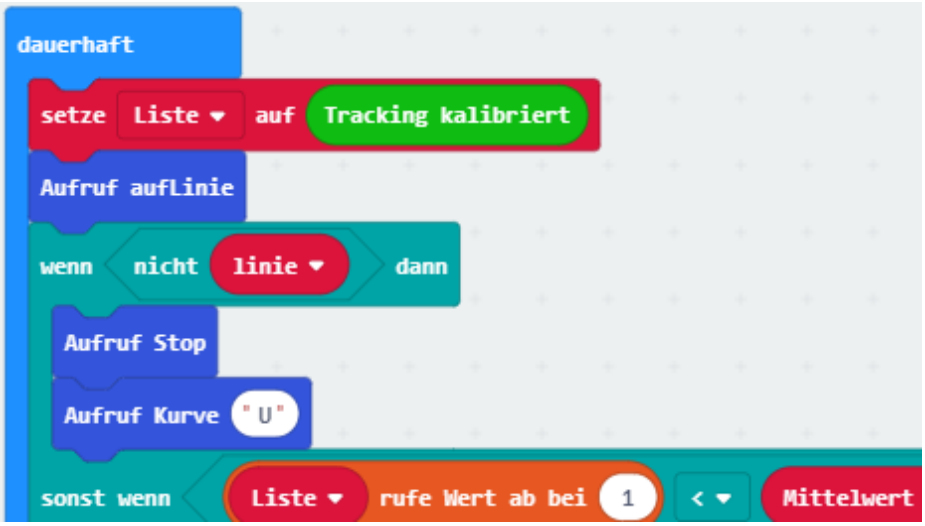
1.) Beim Aufruf von „Kurve“ wird ein Buchstabe (Text) übergeben. Zieht man mit der Maus das runde rote Feld „Richtung“ auf einen runden Platzhalter, kann man den Buchstaben abfragen und vergleichen
2.) „fahr“ ist 0, wenn der Roboter stehen soll
3.) Eine halbe Sekunde mit Gas 40 (damit wir genau über der Kurve stehen)
4.) Wir messen nochmals die Sensoren
5.) Die Funktion „aufLinie“ setzt die Variable linie auf 0, wenn die Sensoren 1, 2 der 3 kein „schwarz“ sehen
6.) Wenn wir keine Linie mehr sehen, dann müssen wir entscheiden, wie wir uns drehen
7.) Bei einer Sackgasse „U“ drehen wir 2 Sekunden nach links (wir drehen uns um/180°)
8.) Bei einer „echten“ Rechtskurve (kein |-) drehen wir nach 1sec nach rechts (ausprobieren, ob die Zeit passt!)
9.) Jetzt bleibt nur mehr das „echte“ T und das L übrig -> nach links drehen
10.) Hier stehen wir auf einer + Kreuzung und die Linie geht gerade weiter (wir haben zuerst „T“ erkannt) -> wir drehen nach links (wegen der „Linke-Hand“-Regel)
11.) Das ist der Fall mit dem |- und wir fahren geradeaus weiter. Weil das ein komischer Fall ist (oder wir eine Kurve erkannt haben, die gar keine ist), geben wir einen Ton aus 😊

Jetzt müssen wir noch unsere „Hauptschleife“ (dauerhaft) erweitern und eine Erkennung der Sackgasse einbauen. Dazu rufen wir gleich nach dem Messen (setze Liste auf Tracking kalibriert) die Funktion „aufLinie“ auf.

Dann können wir mit einem „wenn … dann“ die Variable „linie“ abfragen. Wenn „linie“ = 0 ist, dann sehen die 3 Sensoren in der Mitte die Linie nicht.

Der Block 

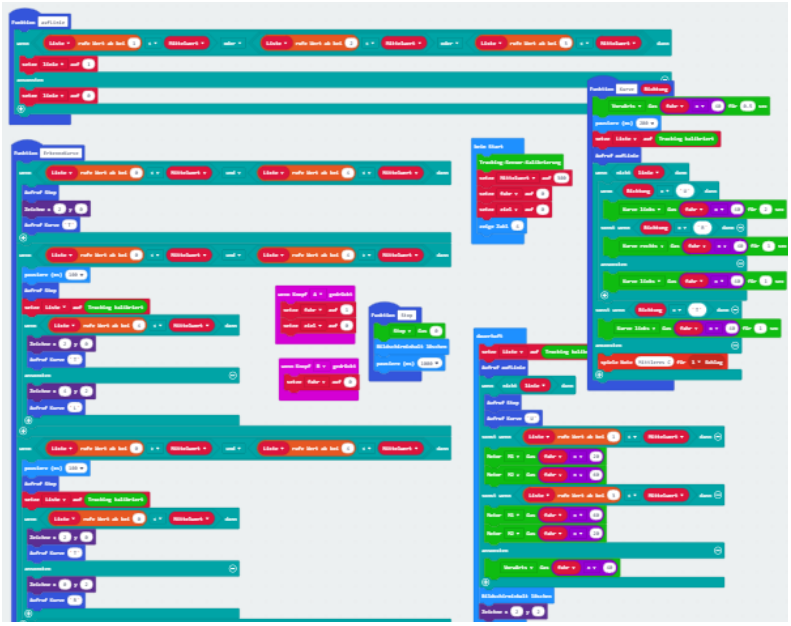
ist dann wahr, wenn „linie“ = 0 ist. Dann soll der Roboter stehen bleiben und umdrehen (Aufruf Kurve „U“).



Damit wir auch Anzeigen, wenn der Roboter der Linie nachfährt (wenn er gerade **keine** Kurve erkennt), bauen wir am Ende der Hauptschleife noch folgendes ein:

Das löscht zuerst den alten Punkt und zeigt dann einen Punkt in der Mitte an.

Unserem Programm fehlt jetzt nur noch die Erkennung des Ziels. Das könnte ein Hindernis sein, das Mik mit seinen Abstandssensoren und dem Ultraschall-Sensor erkennen kann.



Wo würdest du diese Erkennung in das Programm einbauen?