

Blockreferenz ArduBlock letsgoING-Edition



Lizenz Blockreferenz

Dieses Werk von letsgoING ist lizenziert unter einer Creative Commons Namensnennung - Nicht kommerziell - Keine Bearbeitungen 4.0 International Lizenz.



Namensnennung

Sie müssen die Urheberschaft ausreichend deutlich benennen, einen Link zur Lizenz beifügen und angeben, ob Änderungen vorgenommen wurden. Diese Angaben dürfen in jeder angemessenen Art und Weise gemacht werden, allerdings nicht so, dass der Eindruck entsteht, der Lizenzgeber unterstütze gerade Sie oder Ihre Nutzung des Werks besonders.



Nicht kommerziell

Sie dürfen das Material nicht für kommerzielle Zwecke nutzen.



Keine Bearbeitungen

Wenn Sie das Material remixen, verändern oder darauf anderweitig direkt aufbauen dürfen Sie die bearbeitete Fassung der Materials nicht verbreiten.

Kurzbeschreibung: <http://creativecommons.org/licenses/by-nc-nd/4.0/deed.de>

Ausführliche Lizenz: <http://creativecommons.org/licenses/by-nc-nd/4.0/legalcode>



Lizenz ArduBlock

ArduBlock is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

ArduBlock is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with ArduBlock. If not, see <http://www.gnu.org/licenses/>.



Inhaltsverzeichnis

1	Einleitung	1
2	Steuerung	3
2.1	Programm	3
2.2	wiederhole fortlaufend	4
2.3	falls	4
2.4	falls / sonst	5
2.5	solange	5
2.6	wiederhole x-mal	5
2.7	wiederhole variable x-mal	6
2.8	warte Millisekunden	6
2.9	warte Mikrosekunden	6
2.10	warte Millis	7
2.11	Unterprogramm	7
2.12	Unterprogramm	8
2.13	interrupt	9
3	Output	10
3.1	digitalWrite	10
3.2	analogWrite	10
3.3	LED Bar	11
3.4	RGB pin 4/5	12
3.5	Servo	13
3.6	Servo trennen	13
3.7	Servo verbinden	14
3.8	Setup Neopixel	14
3.9	Neopixel Farbe bestimmen	15
3.10	An einen Neopixel Daten senden	15
3.11	Neopixel dimmen	16
4	Input	17
4.1	digitalRead	17

4.2	analogRead	17
4.3	input pullup	18
4.4	Zustand LGI-Button	18
4.5	Zustand LGI-Button entprellt	18
4.6	Zustand LGI-Schalter	19
4.7	LGI-Start-Button	19
4.8	Zustand Q-touch Button	20
4.9	Q-touch Slider	21
4.10	Linienfolger-Sensor	21
4.11	Distanzsensor	22
4.12	PIR	22
4.13	Schiebepotentionmeter	22
4.14	Touchsensor	23
4.15	Taster / Schalter	23
4.16	Lautstärkesensor	23
4.17	LDR	24
4.18	Ultraschallsensor	24
5	Logische Vergleichsoperatoren	25
5.1	analoge Vergleichsoperatoren	25
5.2	digitale Vergleichsoperatoren	25
5.3	Vergleichsoperatoren für Zeichen	25
5.4	logisches und	26
5.5	logisches oder	26
5.6	Verneinung	26
6	Mathematische Vergleichsoperatoren	27
6.1	Arithmetik	27
6.2	Betrag	27
6.3	Potenz	28
6.4	Wurzel	28
6.5	Zufallszahl	28
6.6	Minimum Maximum	29
6.7	zuordnen 8bit	29
6.8	zuordnen allgemein	29
6.9	einschränken	30

7	Variablen/Konstanten	31
7.1	Setze Digitale Variable	31
7.2	Setze Digitale Variable invertiert	32
7.3	Setze Analoge Variable	32
7.4	Setze Analoge Variable Zähler	32
7.5	Setze Analoge Variable Sensor	33
7.6	Setze Zeichen Variable	33
7.7	Setze Zeichen Variable	33
7.8	Name der analogen Variablen	34
7.9	Name der digitalen Variablen	34
7.10	Name der Zeichen-Variablen	34
8	Kommunikation	35
8.1	serial println	35
8.2	serial print	35
8.3	verbinde	36
8.4	parseInt	36
8.5	Daten verfügbar	37
8.6	Serielles Lesen	37
8.7	Wenn Daten an der seriellen Schnittstelle vorliegen: Programm unterbrechen: serialEvent	37
9	Bluetooth	38
9.1	Setup Bluetooth Hardware Serial	38
9.2	Setup Bluetooth Software Serial	38
9.3	Grove-Modul: Starten und Warten bis verbunden über Hardware Serial	39
9.4	Grove-Modul: Starten und Warten bis verbunden über SoftSerial	39
9.5	Werte und Zustände für die Fernbedienung auslesen (Fahrzeug oder LEDLampe): HWSerial	40
9.6	Werte und Zustände für die Fernbedienung auslesen (Fahrzeug oder LEDLampe): SoftSerial	40
9.7	Serial Monitor auslesen: Hardware Serial	41
9.8	Serial Monitor auslesen: Software Serial	41
9.9	Texte und Werte auf dem Smartphone ausgeben: Hardware Serial	42
9.10	Texte und Werte auf dem Smartphone ausgeben: Software Serial	42
9.11	Bluetooth RGB Farben	43
9.12	BT-LED-Schalter	43

9.13 Bluetooth Richtung und Geschwindigkeit	43
9.14 Bluetooth Button	44
9.15 Zustand des Smartphones auslesen	44
10 Fahre	45
10.1 Fahre	45
10.2 fahre Distanz	46
10.3 drehe	46
10.4 Drehe Winkel	46
10.5 fahre Kurve	47
10.6 fahre Kurve Distanz	48
10.7 Motoren stopp	48
10.8 Motoren bremsen	48
10.9 Messung starten	49
10.10Distanz ermitteln	49
11 Beispiele	50
11.1 Falls LDR-Wert größer als, dann...., sonst	50
11.2 blaue RGB dimmen: Werte nur alle 200 ms anzeigen lassen	51
11.3 Taster entprellen	53
11.4 Servo, Servo Trennen	54
11.5 LED-Balken hochzählen	55
11.6 Quadrat fahren - gefahrene Strecke messen	56
11.7 Kurven fahren - fahre Kurve Distanz	58
11.8 Neopixel QTouch	60
11.9 Bluetooth Kommunikation	62
12 Arbeit mit dem Programmablaufplan	63
12.1 Verzweigung	64
12.2 Eingabe	64
12.3 Ausgabe	65
12.4 Vorgang	65

1 Einleitung

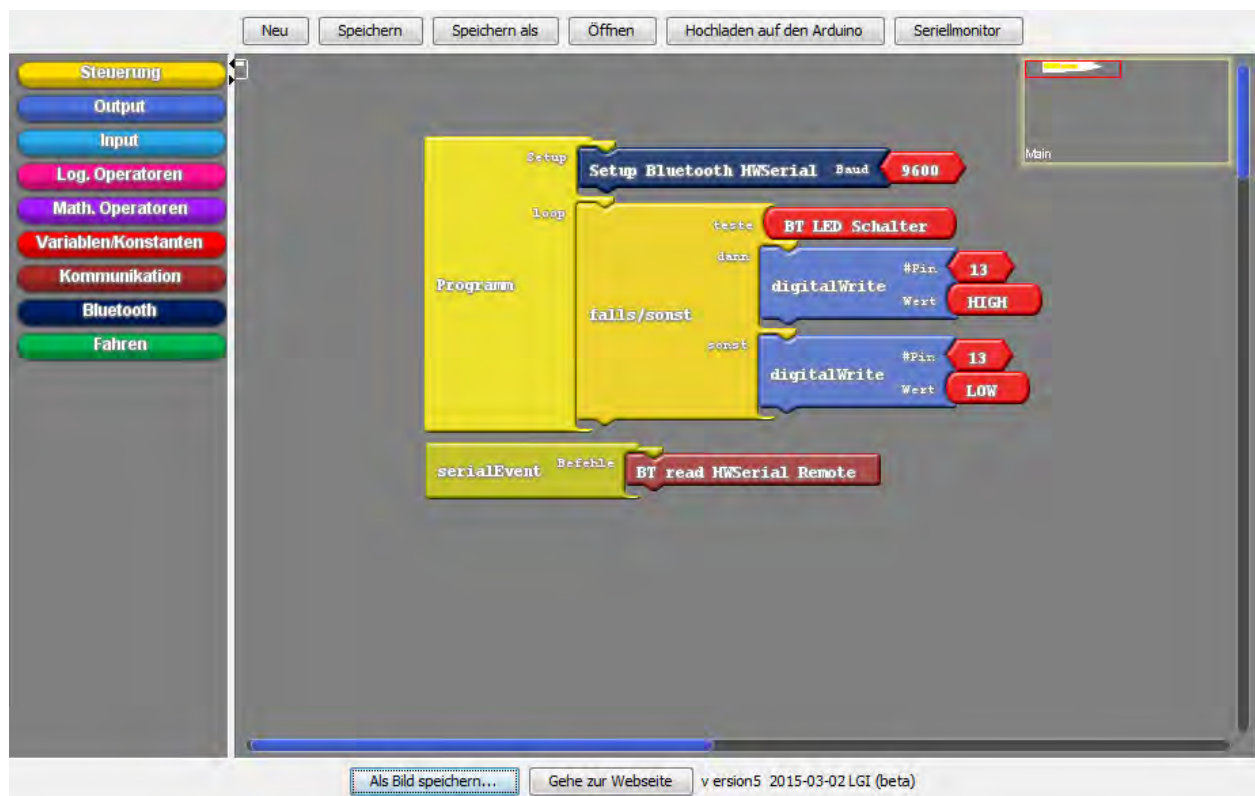
ArduBlock ist eine grafische Entwicklungsumgebung. Sie kann direkt aus der Arduino-Oberfläche aufgerufen werden. Die Programmbefehle liegen als grafische Blöcke vor.

Verschiedene Farben kennzeichnen, um welche Art von Befehlen es sich handelt: So sind z.B. alle Befehle, die mit der Steuerung des Programms zu tun haben, gelb hinterlegt. Befehle, die einen Output erzeugen sind dunkelblau. Befehle, die Werte einlesen (Input) sind hellblau. In weiteren Menüs findest du mathematische oder logische Vergleichsoperatoren. Blöcke für Variablen oder Kommunikation sind ebenfalls in eigenen Menüs zusammengefasst.

Nicht alle Befehle passen aneinander. ArduBlock sorgt dafür, dass nur Programmcode erzeugt wird, der syntaktisch "richtig" ist. Dadurch werden viele Anfängerfehler automatisch vermieden. ArduBlock kümmert sich auch um das sogenannte "Setup". Das heißt, du musst dir erst mal keine Gedanken um die Variablendeklaration oder die Festlegung der einzelnen Pins als Input- oder Outputpins machen. Wenn du ein Skript (ein Programm) aus verschiedenen Blöcken zusammengebaut hast, kannst du es mit dem Button "Hochladen auf den Arduino" übertragen. Aus deinen Blöcken wird Arduino-Code erzeugt, den du im Arduinofenster betrachten und verändern** kannst. Gleichzeitig wird das Programm, in für Menschen völlig unleserliche, Maschinensprache übersetzt. Das nennt man "kompilieren". Beim Kompilieren wird überprüft, ob in deinem Programm grammatikalische Fehler sind. Da sich ArduBlock um die ganze Zeichensetzung kümmert, sollte der Code in der Regel ohne Fehler auf den Arduino übertragen werden.

Diese Blockreferenz dient als Nachschlagewerk und enthält die Beschreibung aller Blöcke die in der letsGoING - Edition des ArduBlock - Programms verwendet werden können.

** Anmerkung: Wenn du den Code im Arduinofenster veränderst, dann kannst du ihn über den Button "Upload" auf den Arduino übertragen - die Veränderungen werden allerdings nicht in dein ArduBlock-Programm übernommen!



- "Neu" erzeugt eine neue ArduBlock-Datei
- "Speichern" speichert die aktuelle ArduBlock-Datei
- "Speichern als" speichert die aktuelle ArduBlock-Datei unter einem anderen Namen
- "Öffnen" öffnet eine neue ArduBlock-Datei
- "Hochladen auf den Arduino" übersetzt den ArduBlock-Code in Arduino-Code und überträgt ihn auf den Arduino (in der Arduino-IDE muss der richtige Serielle Port und das richtige Board ausgewählt sein)
- "Seriellmonitor" öffnet den seriellen Monitor
- "Als Bild speichern" speichert ein Bild der Arbeitsfläche. Da das Bild die gesamte Arbeitsfläche speichert, muss man es noch mit einem Bildbearbeitungsprogramm bearbeiten.
- "Gehe zur Webseite" öffnet die Wiki-Seite von letgoING
- mit rechter Maustaste auf Block klicken: klonen
- mit rechter Maustaste auf Arbeitsfläche klicken: Blöcke neu anordnen
- Blöcke entfernen: Blöcke mit gedrückter Maustaste ins Menü ziehen

2 Steuerung

Im folgenden Teil sind alle Blöcke beschrieben, welche man unter dem ArduBlock-Menüpunkt Steuerung findet.

2.1 Programm

ArduBlock:



Arduino:

```
void setup()  
{  
  //Anweisungen  
}  
void loop()  
{  
  //Anweisungen  
}
```

void setup() void loop()

Jedes Arduino Programm besteht aus einem Setup-Programmteil und einem Loop-Teil. Der Loop-Teil wird wiederholt, sobald das Ende erreicht wurde. Die Setup-Funktion wird nur einmal nach dem Start ausgeführt. ArduBlock nimmt einem viel Arbeit ab, indem es die Grundeinstellungen, Variablendeklarationen und die Konfiguration der seriellen Schnittstelle eigenständig erledigt. Meist reicht der "Wiederholte Fortlaufend Block". Wenn man selbst Eintragungen im Setup vornehmen möchte, kann dies mit dem "Programm"-Block erreicht werden.

2.2 wiederhole fortlaufend

ArduBlock:



Arduino:

```
void loop()
{
  //Anweisungen
}
```

void loop()

Jedes Programm braucht einen loop-Teil. Der Block "wiederhole fortlaufend" stellt den Grundbestandteil jedes Arduino-Programms dar und ermöglicht das endlose Abarbeiten und Wiederholen der Anweisungen innerhalb des Befehls.

2.3 falls

ArduBlock:



Arduino:

```
if (a<10) { //Anweisungen }
```

if()

Mit dem falls-Block kann man eine Bedingung formulieren, die man eindeutig mit ja oder nein beantworten kann. Ist die Falls Bedingung erfüllt, dann mache das, was im falls-Block steht. Wenn Bedingung nicht erfüllt, dann überspringe diesen Teil. Mit den Operatoren kann man festlegen, wie die Bedingung ausformuliert wird: Ist etwas gleich, kleiner, größer etc. als etwas anderes.

Bei digitalen Variablen kann man die digitale Variable auch direkt an den falls-Block anhängen: "falls Linienfolger" bedeutet also: falls der Wert des digitalen Linienfolgersensors HIGH ist, dann mache folgendes, falls der Wert jedoch LOW ist, dann überspringe den Block.

2.4 falls / sonst

ArduBlock:



Arduino:

```
if (a<10) { //Anweisungen }
else { //Anweisungen }
```

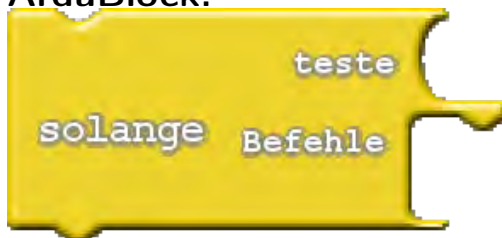
if() else

Beim Befehl „falls – sonst“, wird der „Sonst“-Strang ausgeführt, wenn die Bedingung nicht erfüllt ist (entweder – oder – Entscheidung).

Man kann diesen Befehl auch schachteln, also mehrmals ineinander hängen.

2.5 solange

ArduBlock:



Arduino:

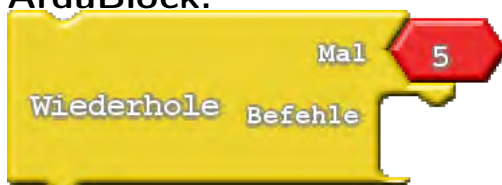
```
while(wert<200)
{ //Anweisungen }
```

while()

Die While-Schleife wird solange durchlaufen, bis der Soll-Wert erreicht ist. Ist das Programm in der Schleife, so werden nur die Befehle in dieser ausgeführt. Wenn Werte eingelesen werden sollen, muss dies in der While-Schleife passieren. (einzige Ausnahme sind Interrupts!)

2.6 wiederhole x-mal

ArduBlock:



Arduino:

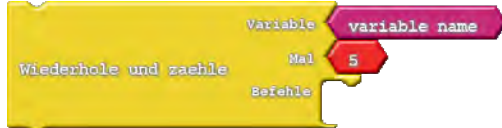
```
for (int i=0; i<10;i++)
{ //Anweisungen }
```

for()

Mit dem wiederhole x-mal Befehl kann man festlegen, wie oft etwas wiederholt werden soll. Ist das Programm in der Schleife, so werden nur die Befehle in dieser ausgeführt.

2.7 wiederhole variable x-mal

ArduBlock:



Arduino:

```
for (int i=0; i<10;i++)
{ //Anweisungen }
```

for()

Mit dem wiederhole x-mal Befehl kann man festlegen, wie oft etwas wiederholt werden soll. Im Unterschied zum obigen Block, kann man hier den Namen der Variablen frei bestimmen.

2.8 warte Millisekunden

ArduBlock:



Arduino:

```
delay(1000);
```

delay()

Mit dem Block "Warte Millisekunden" kann man festlegen, wie viele Millisekunden das Programm pausieren soll. Während dieser Zeit können keine anderen Befehle ausgeführt werden (Ausnahme: Interrupts)

2.9 warte Mikrosekunden

ArduBlock:



Arduino:

```
delayMicroseconds(1000);
```

delayMicroseconds()

Mit dem Block "Warte Mikrosekunden" kann man festlegen, wie viele Mikrosekunden das Programm pausieren soll. Während dieser Zeit können keine anderen Befehle ausgeführt werden (Ausnahme: Interrupts)

2.10 warte Millis

ArduBlock:



Arduino:

Variablendeklaration:

```
unsigned long current;
int time;
```

Aufruf im loop:

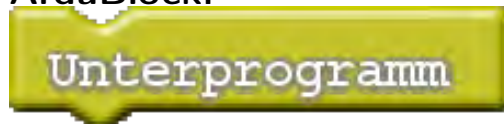
```
current=millis();
while(current+1000>millis());
{Pausenprogramm}
```

warte Millis

Die Funktion `millis()` ermittelt die Anzahl der Millisekunden, die seit dem Start des Arduinos verstrichen ist. Der Block "warte Millis" sollte verwendet werden, wenn parallel zur Wartezeit andere Aufgaben erledigt werden sollen. Beispiel: Eine LED blinkt im Sekundentakt. Währenddessen soll ein Sensor abgefragt werden. Wenn die normale Warte-Funktion verwendet würde, würde der Sensor nur einmal pro Sekunde abgefragt. Mit "Zeit" wird die Wartezeit des Hauptprogramms festgelegt. In der Millis-Schleife (mache) schreibt man das Pausenprogramm (z.B. die Abfrage des Sensors und was passieren soll, wenn der Sensor einen Sollwert überschreitet.)

2.11 Unterprogramm

ArduBlock:



Arduino:

Aufruf des Unterprogramms im loop:

```
Unterprogramm();
```

Unterprogramm

```
void Unterprogramm(){Anweisungen}
```

void subroutine()

Mit ArduBlock kann man Unterprogramme aufrufen. Wichtig ist, dass das Unterprogramm den gleichen Namen hat, wie sein Aufruf (auch Groß- und Kleinschreibung sind zu beachten). Das Unterprogramm ist ein eigenständiges Programm und wird als eigener Block dargestellt.

2.12 Unterprogramm

ArduBlock:



Arduino:

Aufruf des Unterprogramms im loop:

Variable = UnterprogrammReturn();

Unterprogramm

```
int UnterprogrammReturn()
{
  Anweisungen
  return lokale Variable;
}
```

void subroutine()

Mit dem Block UnterprogrammReturn kann man ein Unterprogramm ausführen, das einen Wert an das Hauptprogramm zurückgibt: Das Unterprogramm kann z.B. etwas berechnen und das Ergebnis dann in eine lokale Variable speichern. Das Hauptprogramm "sieht" diese lokale Variable nicht. Damit das Hauptprogramm den Wert dieser Variablen in eine eigene Variable schreiben kann, muss sie über den Konnektor "Rueckgabewert" an das Hauptprogramm zurückgegeben werden.

Da der Wert im Hauptprogramm von einer Variablen übernommen werden muss, kann man die Unterfunktion jetzt nicht mehr einfach aufrufen, sondern man muss den Aufruf über eine Variablenzuweisung tätigen. Der Rueckgabewert (also das, was in der Unterfunktion berechnet und in eine lokale Variable abgelegt wurde) wird dann in die aufrufende Variable geschrieben.

2.13 interrupt

ArduBlock:



Arduino:

im setup:

```
attachInterrupt(0,interruptPin2,CHANGE);
```

kein Aufruf in der loop:

Ereignis wird nicht von loop ausgelöst,
sondern durch Änderung an Pin2

Unterprogramm (Interrupt Service Routine ISR:)

```
void interruptPin2()
```

```
{z.B. Hochzählen einer Variablen}
```

attachInterruptPin2();

Ein Interrupt bewirkt eine Unterbrechung des laufenden Programms. Überprüft wird, ob sich an Pin2 oder Pin3 der Wert ändert. Bei jeder Änderung (CHANGE) wird das Hauptprogramm unterbrochen und das Unterprogramm aufgerufen und ausgeführt. Erst nach dem Abarbeiten des Unterprogramms geht es im Hauptprogramm weiter. Beim Uno und Leonardo gibt es nur zwei Interrupts. Sie sind fest mit den Pins D2 und D3 gekoppelt. Wenn ein Inkrementalgeber an Pin2 angeschlossen ist, führt jede Änderung (also sehe Lücke UND sehe keine Lücke) zu einem Aufruf des Unterprogramms InterruptPin2. Man muss den Block "Unterprogramm" auf die Arbeitsfläche ziehen und in InterruptPin2 umbenennen. In diesem Unterprogramm kann man z.B. die Interrupts hochzählen.

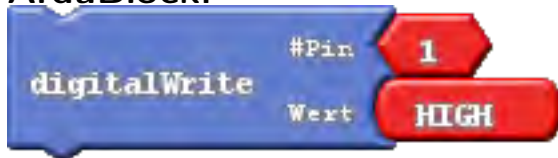
Hinweis: Ardublock reagiert auf alle Änderungen. Im Arduinoprogramm kann man statt CHANGE auch LOW, RISING oder FALLING als Änderungsparameter wählen.

3 Output

Im folgenden Teil sind alle Blöcke beschrieben, welche man unter dem ArduBlock-Menüpunkt "Output" findet.

3.1 digitalWrite

ArduBlock:



Arduino:

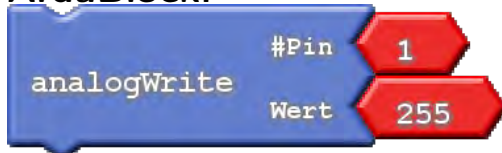
```
digitalWrite(1,HIGH);
```

digitalWrite()

Mit dem Befehl digitalWrite wird ein Ausgang gesetzt. Man muss die Nummer des Ports festlegen. Die Konfiguration des Ports als Ausgang übernimmt ArduBlock. HIGH bedeutet 5V Spannung – LOW bedeutet 0V Spannung. Bei der Motorsteuerung wird mit digitalWrite die Richtung festgelegt.

3.2 analogWrite

ArduBlock:



Arduino:

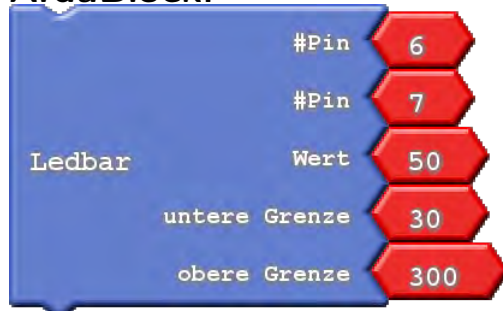
```
analogWrite(1,200);
```

analogWrite()

Der Arduino kann keine echte analoge Spannung ausgeben. Er taktet, das heißt er schaltet den Ausgang ständig ein- und aus. Man nennt dies Pulsweitenmodulation (PWM). Der Wert liegt im Bereich zwischen 0 und 255. Der Wert Null entspricht 0 Volt. 255 entspricht 5 Volt. Nur die Pins 3, 5, 6, 9, 10 und 11 können als analoge Ausgänge geschaltet werden. Bei den Motoren wird mit analogWrite die Geschwindigkeit eingestellt.

3.3 LED Bar

ArduBlock:



Arduino:

Einbinden der Bibliothek:

```
#include <lets go ING_LEDBar.h>
```

Funktionsprototyp:

```
LEDBar LEDBar(6, 7);
```

Aufruf der Funktionen im loop:

```
LEDBar.setMapLow(30);
```

```
LEDBar.setMapHigh(300);
```

```
LEDBar.analogToStack(50, 0);
```

ledBar

Für die LED bar gibt es eine eigene Bibliothek, die beim Verwenden dieses Blocks automatisch eingebunden wird. Die Datenleitungen der LED Bar werden in Pin6 und Pin7 eingesteckt: Das weiße Kabel an Pin6 und das gelbe Kabel an Pin 7. In die mittlere Buchse des Blocks wird der einzulesende Wert angedockt, z.B. `analogRead1`.

Mit "untere Grenze" kann der Wert definiert werden, bei dem der erste Balken leuchten soll. Mit "obere Grenze" wird der Wert festgelegt bei dem der rote Balken angezeigt wird.

3.4 RGB pin 4/5

ArduBlock:



Arduino:

Einbinden der Bibliothek:

```
#include <RGBdriver.h>
```

Definitionsanweisung:

```
#define CLK 4
```

```
#define DIO 5
```

Funktionsprototyp:

```
RGBdriver Driver(CLK,DIO);
```

Aufruf der Funktionen im loop:

```
Driver.begin();
```

```
Driver.SetColor(0,0,0);
```

```
Driver.end();
```

RGB pin4/5

Der Block dient ausschließlich zur Ansteuerung des Seeed Studio Grove - LED Strip Drivers. Die LED-Kette wird durch eine eigene Bibliothek eingebunden. Die Pins können nicht frei gewählt werden. Man muss die LED-Kette mit den Pins 4/5 verbinden. Für rot, grün und blau können Werte von 0 bis 255 gewählt werden.

3.5 Servo

ArduBlock:



Arduino:

Einbinden der Bibliothek:

```
#include <Servo.h>
```

Funktionsprototyp:

```
Servo servo_pin_5;
```

Aufruf im Setup Block:

```
servo_pin_5.attach(5);
```

Aufruf der Funktion im loop:

```
servo_pin_5.write( 0 );
```

attach(pin) write(winkel)

Bei Verwendung des Servo-Blocks wird die Servo-Bibliothek eingebunden. Man kann den Pin wählen und die Gradzahl auf die der Servo drehen soll (0 bis 180 Grad). Bei Verwendung dieses Blocks wird im Setup-Code mit dem Befehl `servo.attach(pin)` ein Servo-Objekt erzeugt. Im Loop-Teil wird der Winkel eingestellt. Achtung: Bei Verwendung der Servo-Bibliothek wird unabhängig davon welcher Pin gewählt wurde, die `analogWrite` (PWM)-Funktionalität von Pin9 und Pin10 deaktiviert.

3.6 Servo trennen

ArduBlock:



Arduino:

Einbinden der Bibliothek:

```
#include <Servo.h>
```

Funktionsprototyp:

```
Servo servo_pin_5;
```

Aufruf der Funktion im loop:

```
servo_pin_5.detach();
```

servo.detach()

Mit diesem Block wird der Servo vom gewählten Pin getrennt und wird nicht mehr auf Position gehalten. Der Servo braucht jetzt keinen Strom mehr. Nach dem Trennen kann die `analogWrite` (PWM)-Funktionalität von Pin9 und Pin10 wieder genutzt werden.

3.7 Servo verbinden

ArduBlock:



Arduino:

Einbinden der Bibliothek:

```
#include <Servo.h>
```

Funktionsprototyp:

```
Servo servo_pin_5;
```

Aufruf der Funktion im loop:

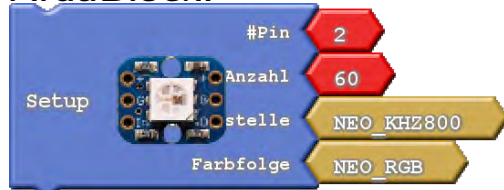
```
servo_pin_5.attach(5);
```

servo.attach(pin)

Wenn man mit dem detach-Befehl den Servo getrennt hat, muss man ihn mit dem attach(pin)-Befehl neu verbinden.

3.8 Setup Neopixel

ArduBlock:



Arduino:

Einbinden der Bibliothek:

```
#include <Adafruit_NeoPixel.h>
```

Funktionsprototyp: Adafruit_NeoPixel

```
strip_pin2 = Adafruit_NeoPixel(60,2,  
NEO_RGB + NEO_KHZ800);
```

Setup

```
void setup()  
{strip_pin2.begin();  
strip_pin2.show();}
```

Setup Neopixel

NeoPixel von Adafruit ist eine kleine RGB-LED mit eigenem Prozessorchip. Man kann mehrere in Reihe schalten und braucht nur einen Datenpin. Mit dem Setup-Block wird die Neopixel-Library eingebunden.

"#Pin" bezeichnet den Pin an welchem die Kette hängt

"Anzahl" gibt an, wie viele Pixel in der Kette enthalten sind

"Schnittstelle" gibt an, in welchem Takt der Chip arbeitet (i.d.R. 800kHz)

"Farbfolge" legt fest, ob rot-grün-blau oder grün-rot-blau (hängt von der Pixelsorte ab: fast alle Neopixel brauchen NEO_GRB, nur Einzel-RGB klassisch 5mm oder 8mm brauchen die Einstellung NEO_RGB)

3.9 Neopixel Farbe bestimmen

ArduBlock:



Arduino:

Aufruf der Funktion im loop:

```
void loop()
{
  strip_pin2.setPixelColor(20,255,255,255
);
}
```

Farbe fuer Pixel [0...255]

Mit dem Block "Farbe fuer pixel RGB [0...255]" kannst du gezielt einen Pixel aus der Kette ansprechen:

"#Pin" bezeichnet den Pin an welchem die Kette hängt

"Nummer Pixel": Welcher Pixel soll programmiert werden? Achtung: die Zählung beginnt mit Null!

"rot-grün-blau": Welchen PWM-Wert erhalten die Farben: Bereich von [0 bis 255]

Wenn du statt der Werte Variablennamen nimmst, kannst du mit wenigen Blöcken die verschiedenen Pixel ansprechen und mit Farbwerten versehen!

3.10 An einen Neopixel Daten senden

ArduBlock:



Arduino:

Aufruf der Funktion im loop:

```
void loop()
{
  strip_pin2.show();
}
```

Daten an Pixel senden

Erst dieser Block sendet die Festlegungen, die du mit dem "Farbe fuer Pixel"-Block getätigt hast, an die Pixel. Ohne ihn bleibt der "Farbe fuer Pixel"-Block wirkungslos. Mit diesem Block kann man festlegen, wann die Daten an die Pixel gesendet werden. Wenn du z.B. 3 Neopixel in einer Reihe hast, kannst du mit drei "Farbe fuer Pixel"-Blöcken unterschiedliche Farben festlegen und dann die Daten an die Pixel senden oder nach jedem "Farbe fuer Pixel"-Block direkt die Daten an die Pixel senden.

3.11 Neopixel dimmen

ArduBlock:



Arduino:

Aufruf der Funktion im loop:

```
void loop()
{
  strip_pin2.setBrightness(255);
}
```

Neopixel Helligkeit

Mit dem Block "Helligkeit [0...255]" werden alle Farben im Verhältnis skaliert und damit gedimmt.

4 Input

Im folgenden Teil sind alle Blöcke beschrieben, welche man unter dem ArduBlock-Menüpunkt Input findet.

4.1 digitalRead

ArduBlock:



Arduino:

```
digitalRead(1);
```

digitalRead()

Mit dem Befehl `digitalRead` wird der Wert eines Ports eingelesen. Man muss die Nummer des Ports festlegen. Die Konfiguration des Ports als Eingang übernimmt ArduBlock. Mit dem Block "falls" kann man überprüfen, ob an diesem Pin HIGH oder LOW anliegt. Wenn man den Wert wiederverwenden will, sollte man ihn an eine digitale Variable übergeben.

4.2 analogRead

ArduBlock:



Arduino:

```
analogRead(1)
```

analogRead()

Analoge Eingänge werden vom Arduino mit 10 bit aufgelöst, das entspricht einem Bereich von 0 bis 1023. Das Eingangssignal liegt zwischen 0 Volt (das entspricht dem Wert 0) und 5 Volt (das entspricht dem Wert 1023). Man muss angeben, an welchem Port die Spannung eingelesen werden soll. Wenn ich mit dem "falls-Block" den Wahrheitsgehalt abfragen möchte, muss ich "analogRead" in einen "logischen Operator Block" einbauen und mit einem anderen Wert vergleichen. Wenn man den Wert wiederverwenden will, sollte man ihn an eine analoge Variable übergeben. Beim Uno werden die Ports 0 bis 5 als analoge Eingänge verwendet. (sie werden als A0 bis A5 bezeichnet)

4.3 input pullup

ArduBlock:



Arduino:

```
pinMode( 1 , INPUT);
digitalWrite(1, HIGH);
```

input pullup

Wenn man einen Pin als Eingang festlegt, dann sollte man ihn mit einem hochohmigen Widerstand entweder an Ground oder an 5V binden. Damit hat der Eingang einen definierten Zustand und flattert nicht herum. Wenn du einen Taster ohne einen externen Widerstand einsetzen möchtest, kannst du mit dem Block "input pullup" den Pin über einen internen Widerstand an den HIGH-Pegel binden.

4.4 Zustand LGI-Button

ArduBlock:



Arduino:

```
Einbindung der library und Aufruf der
AnalogButton-Instanz
#include <LGI_AnalogButton.h>
AnalogButton Button1(1);
variable = Button1.getButton() ;
```

Zustand LGI-Button

Auf dem LGI-Shield befinden sich neben dem Reset-Button zwei Buttons, die beide über Spannungsteiler mit AnalogPin3 verbunden sind. Liegt der eingelesene Wert bei ca. 511, dann ist Button1 gedrückt. Liegt der Wert bei ca. 341 dann ist Button2 gedrückt. Bei einem Wert um 614 sind beide Buttons gleichzeitig gedrückt. Der Block rechnet den Analogwert in einen Digitalwert um.

4.5 Zustand LGI-Button entprellt

ArduBlock:



Arduino:

```
Einbindung der library und Aufruf der
AnalogButton-Instanz
#include <LGI_AnalogButton.h>
AnalogButton Button1(1);
var = Button1.getButtonDebounce() ;
```

Zustand LGI-Button entprellt

Vgl. Erläuterungen bei "Wert von LGI-Button". Bei diesem Block sind die Taster entprellt (debounced). Mechanische Kontakte neigen dazu, sich nach dem Schließen noch mehrmals zu öffnen und wieder zu schließen. Sie federn. Man weiß dann nach einem Tastendruck nicht, ob die Taste 1 Mal, 2 Mal oder x-Mal gedrückt wurde. Um diese Störungen zu beseitigen, wird der Taster entprellt. Das bedeutet, dass nach dem Drücken das Programm kurz wartet und erst nach dieser Pause erneut testet, ob der Taster gedrückt wurde.

4.6 Zustand LGI-Schalter

ArduBlock:



Zustand LGI-Schalter1

Arduino:

Einbindung der library und Aufruf der AnalogButton-Instanz

```
#include <LGI_AnalogButton.h>
```

```
AnalogButton Button1(1);
```


```
var = Button1.getSwitchDebounce();
```

Zustand LGI-Schalter

Ein Taster kann auch als Schalter verwendet werden. Mit Hilfe der LGI_AnalogButton-library wird der Zustand des Schalters bei jedem Tastendruck geändert: Einmal gedrückt bedeutet "eingeschaltet", beim zweiten Druck wechselt der Zustand in "ausgeschaltet".

4.7 LGI-Start-Button

ArduBlock:



LGI-Start-Button1

Arduino:

Einbindung der library und Aufruf der AnalogButton-Instanz

```
#include <LGI_AnalogButton.h>
```

```
AnalogButton Button1(1);
```

```
void setup()
```

```
{Button1.startButton();}
```

Start Button

Der LGI-Startbutton ist nur im Zusammenhang mit dem ArduRover zu verwenden. Er hält das Programm solange auf, bis man den linken Button gedrückt und wieder losgelassen hat. Der Start-Button sorgt z.B. dafür, dass der ArduRover nicht gleich losfährt. Am Besten packt man den Start-Button ins Setup-Programm.

4.8 Zustand Q-touch Button

ArduBlock:



Arduino:

```
#include <LGI_QTouch.h>
QTouchButton Button1(1,2);
void setup()
{
  Button1.init();
}
void loop()
{
  offsetButton1 = Button1.getOffset() ;
  var = Button1.isTouched() ;
}
```

Q-touch Button

Mit dem LGI-Touchsensor können auf einfache und sehr günstige Art bis zu 6 Touchsensoren über die 6 analogen Pins gewonnen werden (mit Ardublock nur 3). Die Touchsensoren sind kapazitive Sensoren. Durch Berührung verändert sich die Kapazität des Sensors. Die Touch-Fläche kannst du als eine Platte eines Plattenkondensators ansehen. Die Library arbeitet mit Lade- bzw. Entladespannung dieses Kondensators.

Wenn du mit dem Cursor über die Blöcke fährst, verwandelt sich der Pfeil in eine Hand. In der Mitte des Blocks erscheint ein kleiner "Drop-Down-Pfeil". Mit dem kannst du aussuchen, ob du Button1, Button2 oder Button3 programmierst.

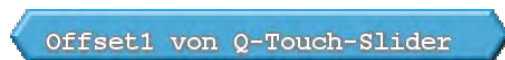
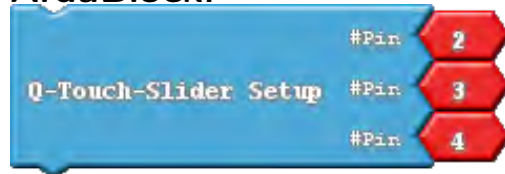
Mit dem Block "Setup für Q-Touch-Button" werden zwei analoge Pins bestimmt. Bevor du die Q-Toch-Buttons verwenden kannst, musst du im Setup festlegen, an welchen Pin dein Q-Touch Button hängt. Du brauchst einen weiteren Pin für die Entladung des eingebauten Kondensators. Du kannst mehrere Q-Touch-Sensoren jedoch miteinander koppeln: Q-Touchsensor1 hängt an A0 und verwendet A1 als Partner-Pin. Q-Touchsensor2 hängt an A1 und nimmt A2 als Partner-Pin und Q-Touchsensor3 nimmt A2 und verwendet A0 als Partner-Pin. So kann ich mit nur 3 Pins 3 Q-Touch-Sensoren einsetzen.

Mit dem Block "Offset von Q-Touch-Button" kannst du dir den Offset eines Buttons anzeigen lassen.

Mit dem Block "Q-Touch-Button1 beruehrt?" kannst du dir anzeigen lassen, ob der Button gedrückt wurde oder nicht: 1 bedeutet berührt, 0 bedeutet nicht berührt.

4.9 Q-touch Slider

ArduBlock:



Arduino:

```
#include <LGI_QTouch.h>
QTouchSlider Slider(2,3,4);
void setup()
{
  Slider.init();
}
void loop()
{
  varOffset = Slider.getOffset(1);
  var = Slider.getTouchPosition();
}
```

Q-touch Slider

Der Q-Touch-Slider liefert Ergebnisse vergleichbar eines Schiebepotentiometers. Der Q-Touch-Slider arbeitet aber nicht wie ein Widerstand, sondern ebenfalls kapazitiv.

Er arbeitet nach dem gleichen Prinzip wie der Q-Touch-Sensor. Der Slider gibt jedoch keine boolschen Werte [0,1] zurück, sondern Werte zwischen 0 und 100. Für den Q-Touch-Slider brauchst du 3 Pins, die du über den Q-Touch-Slider Setup Block bekannt machen musst. Die drei Offset-Werte kannst du dir über die Blöcke "Offset1 (bis3) von Q-Touch-Slider" ausgeben lassen. Damit der Slider den gesamten Bereich zwischen Null und 100 zurückgeben kann, musst du ihn zu Beginn "eichen", das bedeutet nach dem Start einmal über den gesamten Sensor streichen.

4.10 Linienfolger-Sensor

ArduBlock:



Arduino:

```
digitalRead(1)
```

Linienfolge-Sensor

Der Linienfolge-Sensor ist ein digitaler Sensor: das bedeutet er zeigt entweder HIGH oder LOW an. HIGH bedeutet "schwarz". Man kann stattdessen einen digitalRead Block verwenden.

4.11 Distanzsensor

ArduBlock:



Arduino:

```
analogRead(1)
```

Distanzsensor

Der Distanzsensor ist ein analoger Sensor: das bedeutet er zeigt Werte von Null bis 1023. Er arbeitet nach dem Triangulationsprinzip. Die eingelesenen Werte sind nicht linear. Im Nahbereich unter 5 cm ist er blind, d.h. dort werden keine korrekten Werte angezeigt. Man kann zum Einlesen alternativ auch den Block analogRead verwenden.

4.12 PIR

ArduBlock:



Arduino:

```
digitalRead(1)
```

Digit. Infrarot Bewegungsmelder

Ein Bewegungsmelder ist ein elektronischer Sensor, der Bewegungen in seiner näheren Umgebung erkennt und dadurch als elektrischer Schalter arbeiten kann. Der PIR-Sensor (englisch passive infrared) ist der am häufigsten eingesetzte Typ von Bewegungsmeldern. Der Sensor erkennt dabei die abgestrahlte Wärme der vorbeigehenden Menschen. Bewegt sich die Wärmequelle, so löst der Sensor aus. Der Sensor hat zwei Stellschrauben: Mit der linken wird die Haltezeit (max 25s, min 1s) eingestellt. Mit der rechten der Abstand auf den der Sensor reagiert: (max 6m, min 5cm)

4.13 Schiebepotentionmeter

ArduBlock:



Arduino:

```
var=analogRead(1);
```

Schiebepotentionmeter

Ein Potentiometer (kurz Poti, nach neuer deutscher Rechtschreibung auch Potenziometer) ist ein elektrischer Widerstand, dessen Widerstandswerte mechanisch (durch Drehen oder Verschieben) veränderbar sind. Es hat mindestens zwei Anschlüsse und wird hauptsächlich zum veränderlichen Teilen von Spannungen eingesetzt.

4.14 Touchsensor

ArduBlock:



Arduino:

```
digitalRead(1)
```

Touchsensor

Der Touch-Sensor reagiert auf die Änderung seiner Umgebung (Kapazität der Umgebung). Er erkennt ob sich z.B. ein Finger oder nur Luft um ihn herum befindet. Berührt man die Touch-Fläche so löst der Sensor aus.

4.15 Taster / Schalter

ArduBlock:



Arduino:

```
digitalRead(1)
```

Taster / Schalter

Ein Taster ist ein digitaler Sensor. Beim Einlesen eines Tasters werden häufig mehrere Signale mit HIGH und LOW eingelesen. Dieses Verhalten wird auch als Prellen oder Bouncing bezeichnet. Um für jeden Tastendruck genau ein Signal zu erhalten, muss man den Taster entweder hardwareseitig oder durch ein kleines Zusatzprogramm entprellen.

4.16 Lautstärkesensor

ArduBlock:



Arduino:

```
analogRead(1)
```

Lautstärkesensor

Der Grove Loudness-Sensor misst die Lautstärke der Umgebungsgeräusche. Der Sensor besteht aus einem eingebauten Mikrophon und einem Verstärker. Um unnötige Signalstörungen zu vermeiden, wird das Eingangssignal gefiltert. Durch Drehen der kleinen Potentiometer-Schraube kann man die Ausgangsverstärkung einstellen.

4.17 LDR

ArduBlock:



Arduino:

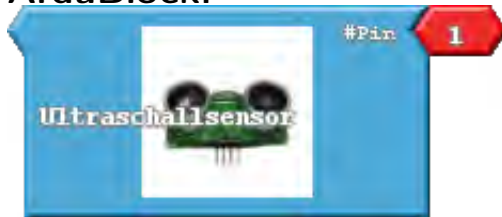
`analogRead(1)`

LDR

Ein Fotowiderstand (englisch Light Dependent Resistor, LDR) ist ein lichtabhängiger Widerstand aus einer amorphen Halbleiter-Schicht. Je höher der Lichteinfall, desto kleiner wird - aufgrund des inneren fotoelektrischen Effekts - sein elektrischer Widerstand. Fotowiderstände eignen sich, aufgrund ihrer hohen Empfindlichkeit, sehr gut um die Helligkeit zu messen. Sie reagieren jedoch sehr langsam (Bereich 1 ms bis einige Sekunden).

4.18 Ultraschallsensor

ArduBlock:



Arduino:

Unterprogr. Ultraschall_Sensor:

Puls aussenden

Puls empfangen

Abstand berechnen

cm-Wert zurückgeben

im loop-Teil:

Abstandswert in cm

Ultraschallsensor

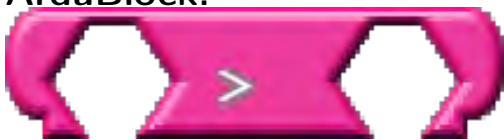
Weil der Ultraschallsensor ein getaktetes Signal ausgibt, muss er an einen digitalen Pin eingesteckt werden. Der Ultraschallsensor pulst Signale und misst die Zeit, bis die ausgesendeten Ultraschallsignale zurückkommen. Aus der gemessenen Zeit wird der Abstand berechnet. Das Pulsen, Empfangen und Berechnen wird in die Unterfunktion "Ultraschall_Sensor" ausgelagert. Mit einem normalen `analogRead`-Befehl kann der Ultraschall-Sensor nicht ausgelesen werden.

5 Logische Vergleichsoperatoren

Im folgenden Teil sind alle Blöcke beschrieben, welche man unter dem ArduBlock-Menüpunkt logische Operatoren findet.

5.1 analoge Vergleichsoperatoren

ArduBlock:



Arduino:

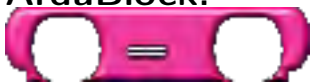
if (var > 10)

analoge Vergleichsoperatoren

Es gibt 6 Vergleichsoperatoren für analoge Werte. Erkennbar sind sie an den sechseckigen Platzhaltern. Man kann mit ihnen Variable oder Konstante miteinander vergleichen. Z.B. ist eine Variable größer als, kleiner als, größer oder gleich, kleiner oder gleich, gleich oder nicht gleich einem anderen Wert.

5.2 digitale Vergleichsoperatoren

ArduBlock:



Arduino:

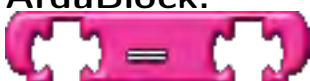
if (var == HIGH)

digitale Vergleichsoperatoren

Mit den digitalen Vergleichsoperatoren kann man überprüfen, ob ein Wert TRUE oder FALSE ist. Erkennbar sind die digitalen Vergleichsoperatoren an den runden Platzhaltern.

5.3 Vergleichsoperatoren für Zeichen

ArduBlock:



Arduino:

if (var == 'A')

Vergleichsoperatoren für Zeichen

Mit den beiden Vergleichsoperatoren für Zeichen kann man überprüfen, ob zwei Zeichen identisch oder nicht identisch sind. Man erkennt die Vergleichsoperatoren für Zeichen an den Platzhaltern "Rechteck mit Flügeln".

5.4 logisches und

ArduBlock:



Arduino:

`if (var == HIGH && i > 10)`

logisches und

Mit dem "logischen und" kann man überprüfen, ob zwei Vergleiche gleichzeitig wahr sind. Das Resultat ist TRUE, wenn beide Werte TRUE sind.

5.5 logisches oder

ArduBlock:



Arduino:

`if (var == HIGH || i > 10)`

logisches oder

Mit dem "logischen oder" kann man überprüfen, ob mindestens einer von zwei durchgeführten Vergleichen wahr ist. Das Resultat ist TRUE, wenn mindestens ein Wert TRUE ist.

5.6 Verneinung

ArduBlock:



Arduino:

`if (!var == HIGH)`

logisches nicht

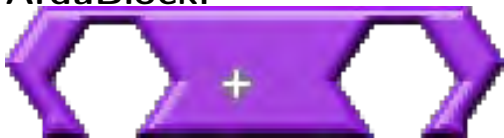
Mit dem "logischen nicht" wird überprüft, ob ein Vergleich falsch ergibt. Das Resultat ist TRUE, wenn der Vergleich FALSE ergibt.

6 Mathematische Vergleichsoperatoren

Im folgenden Teil sind alle Blöcke beschrieben, welche man unter dem ArduBlock-Menüpunkt mathematische Operatoren findet.

6.1 Arithmetik

ArduBlock:



Arduino:

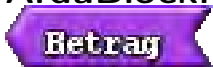
```
y = y + 5;  
a = 6 % 4; //Modulo mit Ergebnis 2
```

Berechnungen

Mit den arithmetischen Blöcken kann man Werte addieren, subtrahieren, multiplizieren, dividieren und den Rest einer Division ermitteln (Modulo)

6.2 Betrag

ArduBlock:



Arduino:

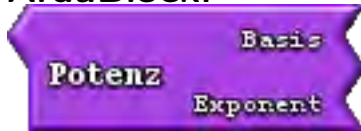
```
int a = -50;  
int b = abs(a); //Ergebnis b=50
```

abs()

Mit diesem Block kann man den Absolutwert (Betrag) ermitteln.

6.3 Potenz

ArduBlock:



Arduino:

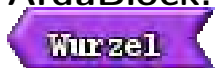
```
int y = pow(x,3); //y = x hoch 3
```

pow(Zahl, Exponent)

Mit diesem Block kann man potenzieren.

6.4 Wurzel

ArduBlock:



Arduino:

```
int y = sqrt(x); // y = Wurzel von x
```

sqrt(x) sin(rad) cos(rad) tan(rad)

Mit diesem Blöcken kann man die Wurzel / Sinus / Cosinus oder Tangens einer übergebenen Zahl berechnen.

6.5 Zufallszahl

ArduBlock:



Arduino:

```
var = random( 1024 ) ;  
// Zufallszahl zwischen 0 und 1023
```

random(x)

Mit diesem Block kann man eine Zufallszahl zwischen Null und der angegebenen Zahl generieren.

6.6 Minimum Maximum

ArduBlock:



Arduino:

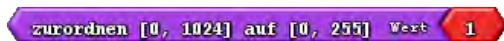
wert = min(x,y); // Ermittelt Minimum der beiden Werte

min(x,y)

Ermittelt welcher der beiden Werte kleiner ist. max(x,y); Ermittelt welcher der beiden Werte größer ist.

6.7 zuordnen 8bit

ArduBlock:



Arduino:

map(sensorwert,0,1023,0,255); // Konvertiert den Sensorwert

map(fromLow, fromHigh, toLow, toHiGH)

Mit diesem Block kann man den 10-bit-Wertebereich (von 0 bis 1023) auf den 8-bit-Wertebereich (von Null bis 255) konvertieren. Z.B. der vom Sensor eingelesene 1023-Wert soll die Max-Geschwindigkeit ergeben, also 255.

6.8 zuordnen allgemein

ArduBlock:



Arduino:

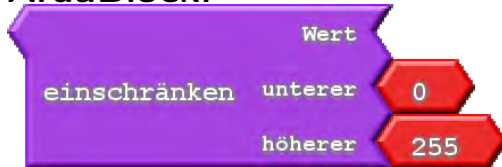
map(sensorwert,0,1023,0,255); // Konvertiert den Sensorwert

map(fromLow, fromHigh, toLow, toHiGH)

Mit diesem Block kann man einen Wertebereich in einen anderen Wertebereich konvertieren. Im Gegensatz zum obigen Block, kann man hier alle Werte frei wählen. Man kann damit auch einen Bereich invertieren: Also den Bereich von 0 bis 1023 zum Bereich 1023 bis 0.

6.9 einschränken

ArduBlock:



Arduino:

```
constrain(Zahl,Minimum,Maximum);
```

Schränkt den Bereich ein

constrain(x,a,b)

Mit diesem Block kann ein Wert so festgelegt werden, dass er in einem Sollbereich liegt. Diesen Block braucht man z.B. in Kombination mit dem map-Befehl: wenn z.B. ein oberer Wert von 700 dem Output-Maximalwert von 255 zugeordnet würde, dann aber der Sensor einen Wert von größer 700 (z.B. 720) einliest, dann käme es zu einem Überlauf. Mit dem constrain-Befehl kann ich das abfangen.

7 Variablen/Konstanten

Im folgenden Teil sind alle Blöcke beschrieben, welche man unter dem ArduBlock-Menüpunkt "Variablen/Konstanten" findet.

Werte, die im Programm mehrfach gebraucht werden, kann man in eine Variable speichern. Der Variablenname sollte möglichst sprechend sein. Der Wert einer Variablen kann sich im Programm laufend ändern. Eine Variable hat immer auch einen Datentyp, der den Wertebereich festlegt. Bei ArduBlock wird der Datentyp einer Variablen durch die Wahl der Blöcke festgelegt (Sechseck = integer-Werte, Kreis=boolesche Werte und Rechteck mit Flügeln=character Werte). Variablen, die sich im Programm nicht ändern, nennt man Konstanten. ArduBlock hat keinen eigenen Block für die Definition von Konstanten. Man verwendet statt dessen die Blöcke "Setze Variable". In Kombination mit den Blöcken aus dem Menü "Math. Operatoren" können Zuweisungen getätigt werden. (Im Arduino-Code erfolgt dies mit dem einfachen Gleichheitszeichen)

7.1 Setze Digitale Variable

ArduBlock:



Arduino:

im Deklarationsblock
`bool Linienfolgesensor=false ;`
in der loop
`Linienfolgesensor = HIGH;`

bool var=HIGH

Mit dem Block "Setze Digitale Variable" kann man einen Variablennamen wählen und den Variablenwert festlegen (HIGH oder LOW). Die Variablendeklaration tätigt ArduBlock selbstständig. Man kann den Wert direkt übergeben, ihn berechnen lassen oder über einen digitalen Sensor beziehen.

7.2 Setze Digitale Variable invertiert

ArduBlock:



Arduino:

im Deklarationsblock:

```
bool DigitalWert= false ;
```

in der loop

```
DigitalWert = !( DigitalWert ) ;
```

bool var=!var

Hier wird der Wert der Variablen "DigitalWert" invertiert.

7.3 Setze Analoge Variable

ArduBlock:



Arduino:

im Setup:

```
int Distanzsensor = 0;
```

in der loop

```
Distanzsensor = 100;
```

var=10

Mit dem Block "Setze Analoge Variable" kann man einen Variablennamen wählen und den Variablenwert festlegen (eine Zahl). Die Variablendeklaration tätigt ArduBlock selbstständig. Man kann den Wert direkt übergeben, ihn berechnen lassen oder über einen analogen Sensor beziehen.

7.4 Setze Analoge Variable Zähler

ArduBlock:



Arduino:

im Deklarationsblock:

```
int i = 0;
```

in der loop

```
i = i+1;
```

var=var+1

Mit diesen Blöcken wird die Variable "Zähler" um 1 hoch gezählt. Über diesen Block werden Änderungen einer Variablen getätigt.

7.5 Setze Analoge Variable Sensor

ArduBlock:



Arduino:

im Deklarationsblock:

```
int Distanzsensor = 0;
```

in der loop

```
Distanzsensor = analogRead(0);
```

var=analogRead()

Mit dem Blöcken wird der Wert des Sensors, der am Pin A0 liegt, in die Variable Entfernung geschrieben.

7.6 Setze Zeichen Variable

ArduBlock:



Arduino:

im Deklarationsblock:

```
char ZeichenVariablen = ' ';
```

in der loop

```
ZeichenVariable = 'A';
```

Zeichen='A'

Mit dem Block "Setze Zeichen Variable" kann man einen Variablennamen wählen und den Variablenwert festlegen (ein Zeichen). Die Variablendeklaration tätigt ArduBlock selbstständig.

7.7 Setze Zeichen Variable

ArduBlock:



Arduino:

im Deklarationsblock:

```
char Zeichen = ' ';
```

in der loop

```
Zeichen = Serial.read();
```

Zeichen=Serial.read()

Mit dem Block "Setze Zeichen Variable" kann man einen Variablennamen wählen und das über Serial.read eingelesene Zeichen in diese Variable schreiben. Die Variablendeklaration tätigt ArduBlock selbstständig.

7.8 Name der analogen Variablen

ArduBlock:



var

Diesen Block braucht man, wenn man den Wert einer schon gesetzten analogen Variablen abfragen möchte.

Wenn man mit einer Steuerfunktion (if, if else, while, for) überprüfen möchte, ob eine Bedingung zutrifft. Hier: solange die Distanz größer als 1 ist.

7.9 Name der digitalen Variablen

ArduBlock:



var

Diesen Block braucht man, wenn man den Wert einer schon gesetzten digitalen Variablen abfragen möchte.

Wenn man mit einer Steuerfunktion (if, if else, while, for) überprüfen möchte, ob eine Bedingung zutrifft. Hier: falls der Linienfolgesensor (LFS) schwarz sieht.

7.10 Name der Zeichen-Variablen

ArduBlock:



var

Diesen Block braucht man, wenn man den Wert einer schon gesetzten Zeichen-Variablen abfragen möchte.

Wenn man mit einer Steuerfunktion (if, if else, while, for)überprüfen möchte, ob eine Bedingung zutrifft. Hier: falls das Zeichen gleich "a" ist.

8 Kommunikation

Im folgenden Teil sind alle Blöcke beschrieben, welche man unter dem ArduBlock-Menüpunkt "Kommunikation" findet.

8.1 serial println

ArduBlock:



Arduino:

```
Serial.print( "Hallo");  
Serial.println();
```

serial println

Dieser Block sendet Daten an die serielle Schnittstelle mit anschließendem Zeilenumbruch (In steht für linefeed). "message" kann mit einem eigenen Texteintrag überschrieben werden. In der Regel wird der Befehl zusammen mit den Befehlen "verbinde" + der passenden Variable eingesetzt.

8.2 serial print

ArduBlock:



Arduino:

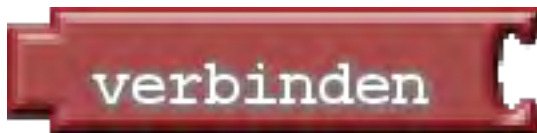
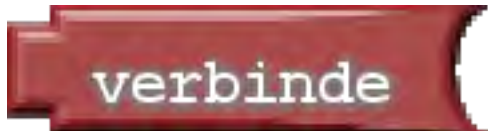
```
Serial.print( "Hallo");  
Serial.println();
```

serial print

Dieser Block sendet Daten an die serielle Schnittstelle. Man kann über die Eingabe "true" oder "false" festlegen, ob ein Zeilenumbruch folgen soll. Der Block "message" kann mit einem eigenen Texteintrag überschrieben werden. In der Regel wird der Befehl zusammen mit den Befehlen "verbinde" + der passenden Variable eingesetzt.

8.3 verbinde

ArduBlock:

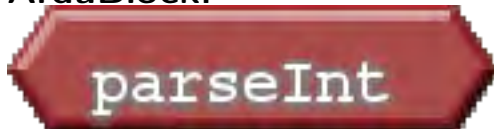


verbinde

Mit diesem Block kann man die jeweils passenden Variablen mit dem print-Befehl verknüpfen

8.4 parseInt

ArduBlock:



Arduino:

```
rot = Serial.parseInt() ;
gruen = Serial.parseInt() ;
blau = Serial.parseInt() ;
```

Serial.parseInt

Mit dem Befehl "Serial.parseInt" wird die erste Integer Zahl aus dem seriellen Pufferspeicher ausgegeben. Zeichen, die keine Ganzzahlen sind, werden ebenso wie ein Minuszeichen ignoriert bzw. weggeschnitten. Das erste Zeichen, das keine Ziffer ist, beendet den Befehl.

8.5 Daten verfügbar

ArduBlock:



serial data available

Arduino:

```
while (Serial.available() > 0)
```

Serial.available

Dieser Block prüft, ob im Eingangspuffer der seriellen Schnittstelle Daten vorhanden sind. Wenn ja, wird die Anzahl der vorhandenen Bytes zurückgegeben, die dann mit Serial.read ausgelesen werden können.

8.6 Serielles Lesen

ArduBlock:



Serielles lesen

Arduino:

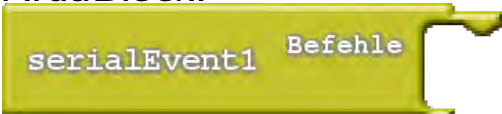
```
if (Serial.read() == '\n')
//falls Zeilenumbruch
(gelesen wird nur der backslash)
```

Serial.read

Dieser Block liest das erste Zeichen aus dem Eingangspuffer der seriellen Schnittstelle. Falls keine Daten vorhanden sind, wird der Wert -1 von der Funktion zurückgegeben.

8.7 Wenn Daten an der seriellen Schnittstelle vorliegen: Programm unterbrechen: serialEvent

ArduBlock:



serialEvent1 Befehle

Arduino:

```
void serialEvent1();
void setup(){}
void loop(){}
void serialEvent1(){}

```

serialEvent

Entweder prüft man in jeder loop, ob neue Daten vorliegen oder man verwendet den Block serialEvent. Dieser Block sorgt dafür, dass sobald Daten an der Hardware-Seriellen-Schnittstelle anliegen, das Programm unterbrochen wird. Zuerst werden nun die Befehle, die im serialEvent stehen ausgeführt. Erst dann wird das Programm an der alten Stelle fortgesetzt.

9 Bluetooth

Im folgenden Teil sind alle Blöcke beschrieben, welche man unter dem ArduBlock-Menüpunkt "Bluetooth" findet. Die braunen Blöcke sind eigentlich Blöcke für die serielle Kommunikation. Statt dem Serial Monitor wird hier das Smartphone oder das Tablet als Monitor eingesetzt. Die roten Blöcke sind Werte oder Zustände, die über das Tablet an den Arduino via Bluetooth gesendet werden.

9.1 Setup Bluetooth Hardware Serial

ArduBlock:



Arduino:

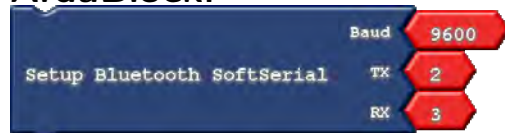
```
#include <letsGoING_Arduroid.h>
Arduroid Remote;
void setup()
{Remote.BTserial_begin(9600);}
```

Setup Bluetooth HWSerial

Jedes Bluetooth-Modul braucht einen Setup-Block. Mikrocontroller-Boards wie der Arduino-UNO haben eine eingebaute serielle Hardware-Schnittstelle am Pin0 und Pin1. Wenn du dein Bluetooth-Modul über diese Schnittstelle betreibst, musst du diesen Block ins Setup einbauen. Beim ArduRover und beim Cherokey wird standardmäßig diese Schnittstelle verwendet.

9.2 Setup Bluetooth Software Serial

ArduBlock:



Arduino:

```
#include <letsGoING_Arduroid.h>
Arduroid Remote(2,3);
void setup()
{Remote.BTsoftSerial_begin(9600);}
```

Setup Bluetooth SoftSerial

Wenn du dein Bluetooth-Modul nicht über die Hardware-Serial-Schnittstelle anschließt, dann musst du diesen Setup-Block im Programm-Setup verwenden. Der Attiny hat keine Hardware-Serial-Schnittstelle. Beim Einsatz vom Attiny brauchst du immer diesen Setup-Block.

9.3 Grove-Modul: Starten und Warten bis verbunden über Hardware Serial

ArduBlock:

BT Grove Starten und WartenBisVerbunden HWSerial

Arduino:

```
#include <letsgoING_Arduroid.h>
void setup()
{
  delay(2000);
  Remote.printMonitor("\r\n+INQ=1\r\n");
  //Grove BTmodul gibt Verbindungszu-
  stand zurück (0-4)
  while(Remote.readConState() != 4);
}
```

BT Grove Starten und WartenBis-Verbunden HWSerial

Die Grove-Module brauchen zwingend einen zweiten Setup-Block. Wenn das BT-Grove-Modul an Pin0 und Pin1 angeschlossen ist, dann braucht es diesen Block. Der Vorteil beim Grove-Modul ist, dass hier kontrolliert wird, ob die Verbindung zustande gekommen ist. Das Modul gibt den Wert 4 zurück, wenn die Verbindung steht.

9.4 Grove-Modul: Starten und Warten bis verbunden über SoftSerial

ArduBlock:

BT Grove Starten und WartenBisVerbunden SoftSerial

Arduino:


```
#include <letsgoING_Arduroid.h>
void setup()
{
  delay(2000);
  Remote.printSoftMonitor("\r\n+INQ=1\r\n");
  //Grove BTmodul gibt Verbindungszu-
  stand zurück (0-4)
  while(Remote.readSoftConState() != 4);
}
```

BT Grove Starten und WartenBis-Verbunden SoftSerial

Die Grove-Module brauchen zwingend einen zweiten Setup-Block. Wenn das BT-Grove-Modul eine SoftSerial-Verbindung nutzt, dann braucht es diesen Block. Der Attiny nutzt immer eine SoftSerial-Schnittstelle! Der Vorteil beim Grove-Modul ist, dass hier kontrolliert wird, ob die Verbindung zustande gekommen ist. Das Modul gibt den Wert 4 zurück, wenn die Verbindung steht.

9.5 Werte und Zustände für die Fernbedienung auslesen (Fahrzeug oder LEDLampe): HWSerial

ArduBlock:



BT read HWSerial Remote

Arduino:


```
#include <lets go ING_Arduroid.h>
Arduroid Remote;
void setup()
{Remote.BTserial _begin(9600);}
void loop()
{Remote.readRemote();}
```

Remote.readRemote();

Diesen Block brauchst du, wenn du für die Fernbedienung deines Fahrzeugs oder für die Fernbedienung deiner Lampe die Zustände und Werte von der App über Pin0 und Pin1 auslesen möchtest. Der Block braucht im Setup den Setup-Bluetooth-HWSerial-Block. Der Block muss in jeder Loop ausgelesen werden oder kann auch im SerialEvent-Block ausgelesen werden.

9.6 Werte und Zustände für die Fernbedienung auslesen (Fahrzeug oder LEDLampe): SoftSerial

ArduBlock:



BT read SoftSerial Remote

Arduino:

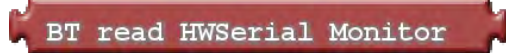
```
#include <lets go ING_Arduroid.h>
Arduroid Remote(2,3);
void setup()
{Remote.BTsoftserial _begin(9600);}
void loop()
{Remote.readSoftRemote();}
```

Remote.readRemote();

Diesen Block brauchst du, wenn du für die Fernbedienung deines Fahrzeugs oder für die Fernbedienung deiner Lampe die Zustände und Werte von der App über SoftSerial auslesen möchtest. Der Block braucht im Setup den Setup-Bluetooth-SoftSerial-Block.

9.7 Serial Monitor auslesen: Hardware Serial

ArduBlock:



Arduino:

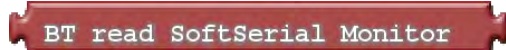
```
#include <letsgoING_Ardudroid.h>
Arduroid Remote();
char Name_der_ZeichenVariablen = ' ';
void setup()
{Remote.BTserial_begin(9600);}
void loop()
{Name_der_ZeichenVariablen =
Remote.readMonitor();}
```

BT read HWSerial Monitor

Wenn du in die Kommandozeile deiner Monitor-App einen Befehl schreibst, wird der via Bluetooth an den Arduino übertragen. Mit diesem Block wird ein ASCII-Zeichen ausgelesen und dann der Speicher wieder geleert. Wir verwenden den Block gemeinsam mit dem "Setze Zeichen-Variable Block". Das BT-Modul ist an Pin0 und Pin1 vom Arduino angeschlossen.

9.8 Serial Monitor auslesen: Software Serial

ArduBlock:



Arduino:

```
#include <letsgoING_Ardudroid.h>
Arduroid Remote(2,3);
char Name_der_ZeichenVariablen = ' ';
void setup()
{Remote.BTsoftSerial_begin(9600);}
void loop()
{Name_der_ZeichenVariablen = Remote.readSoftMonitor();}
```

BT read SoftSerial Monitor

Wenn du in die Kommandozeile deiner Monitor-App einen Befehl schreibst, wird der via Bluetooth an den Arduino übertragen. Mit diesem Block wird ein ASCII-Zeichen ausgelesen und dann der Speicher wieder geleert. Wir verwenden den Block gemeinsam mit dem Setze Zeichen-Variable-Block. Das BT-Modul ist über SoftSerial mit dem Arduino verbunden.

9.9 Texte und Werte auf dem Smartphone ausgeben: Hardware Serial

ArduBlock:



Arduino:


```
#include <lets goING_Arduroid.h>
Arduroid Remote();
void setup()
{Remote.BTserial_begin(9600);}
void loop()
{Remote.printMonitor("message";
Remote.printMonitor("\n");}
```

BT print HWSerial Monitor

Du nimmst diesen Block, wenn du in der Monitor-App Werte oder Texte ausgeben möchtest. Du überschreibst einfach "message" mit einem Text. Mit Hilfe eines passenden Verbinde-Blocks aus dem Menü "Kommunikation" kannst du dann noch die passenden Variable anhängen. Dieser Block wird über die Hardware-Schnittstelle (Pin0 und Pin1) verwendet.

9.10 Texte und Werte auf dem Smartphone ausgeben: Software Serial

ArduBlock:



Arduino:

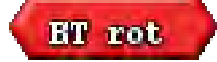
```
#include <lets goING_Arduroid.h>
Arduroid Remote(2,3);
void setup()
{Remote.BTsoftSerial_begin(9600);}
void loop()
{Remote.printSoftMonitor("message";
Remote.printSoftMonitor("\n");}
```

BT print SoftSerial Monitor

Du nimmst diesen Block, wenn du in der Monitor-App Werte oder Texte ausgeben möchtest. Du überschreibst einfach "message" mit einem Text. Mit Hilfe eines passenden Verbinde-Blocks aus dem Menü "Kommunikation" kannst du dann noch die passenden Variable anhängen. Dieser Block wird über die Software-Schnittstelle verwendet.

9.11 Bluetooth RGB Farben

ArduBlock:



Arduino:

```
Remote.getRed()  
Remote.getGreen()  
Remote.getBlue()
```

BT rot

An den "RGB Pin 4/5-Block" oder an den "Neopixel-Block" können die sechseckigen Blöcke "BT rot", "BT gruen" und "BT blau" angehängt werden. Damit werden für die LED-Kette oder die Neopixel die auf dem Android-Gerät eingestellten Werte für die Farben übernommen.

9.12 BT-LED-Schalter

ArduBlock:



Arduino:

```
if (Remote.getLedSwitch())
```

BT LED Schalter

Der Block "BT LED Schalter" kann in einem falls/-sonst Block überprüfen, ob am Smartphone oder Tablet der Schalter "LED An" angeklickt wurde. Man kann damit z.B. die LED-Kette ausschalten.

9.13 Bluetooth Richtung und Geschwindigkeit

ArduBlock:



Arduino:

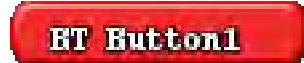
```
Remote.getDirection()  
Remote.getSpeed()
```

BT Geschwindigkeit

Über die Blöcke "BT Richtung" und "BT Geschwindigkeit" kann man im "fahre Kurve Block" festlegen, dass die vom Android-Gerät gesendeten Werte für Richtung (Rotation) und Geschwindigkeit (Translation) aufs Fahrzeug übertragen werden.

9.14 Bluetooth Button

ArduBlock:



Arduino:

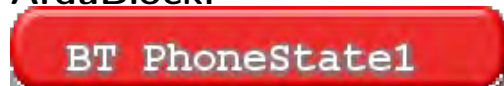
```
if (Remote.getButton(1))
```

BT Button

Wenn man den "BT Button" Block auf die ArduBlock-Fläche zieht, kann man über das kleine Pfeilchen auf dem Block auch den "BT Button2" sowie die Blöcke "BT Switch1" und "BT Switch2" auswählen. Über falls/sonst kann man überprüfen, ob diese Schalter / Tasten auf dem Smartphone gedrückt wurden und Anweisungen für die Fälle programmieren.

9.15 Zustand des Smartphones auslesen

ArduBlock:



Arduino:

```
if (Remote.getState(1))
```

BT PhoneState

Wenn man den "BT PhoneState1" Block auf die ArduBlock-Fläche zieht, kann man über das kleine Pfeilchen auf dem Block auch den "BT PhoneState2" bis "BT PhoneState6" auswählen. Über falls/sonst kann man abhängig von den Zuständen deines Smartphones Anweisungen programmieren. Die Android-Anwendung zur Nutzung der PhoneStates gibt es im Augenblick noch nicht.

10 Fahre

Im folgenden Teil sind alle Blöcke beschrieben, welche man unter dem ArduBlock-Menüpunkt Fahre findet. Wenn man einen Fahrbefehl-Block in sein Programm einbaut, wird automatisch die Bibliothek "letsgoING_Drive.h" eingebunden. Dort sind Werte für die Fahrzeuggeometrie und den Radencoder hinterlegt.

Voreingestellt sind folgende Pins:

Geschwindigkeit rechts (PWM1): Pin 10

Geschwindigkeit links (PWM2): Pin 11

Richtung rechts (dir1): Pin 9

Richtung links (dir2): Pin 12

Wenn man andere Pins verwenden möchte, dann muss man über den Befehl „Drive Rover“ mit folgenden Parametern versehen: Drive Rover (PWM1, PWM2, dir1, dir2) Drive Rover(5,6,4,7); 5 für Geschwindigkeit rechts, 6 für Geschwindigkeit links, 4 für Richtung rechts und 7 für Richtung links. Bei der Übersetzung in die Arduinosprache erkennt man die Befehle aus der Fahren – Library an der Vorsilbe Rover.

10.1 Fahre

ArduBlock:



Arduino:

Einbindung der library und Aufruf der Rover-Instanz

```
#include <letsgoING_Drive.h>
```

```
Drive Rover;
```

Befehl in der loop

```
Rover.drive(50,HIGH);
```

Fahre

Mit dem Block Fahre kann man festlegen, mit welcher Geschwindigkeit (0 bis 100) und in welche Richtung (HIGH oder LOW) das Fahrzeug fahren soll.

10.2 fahre Distanz

ArduBlock:



Arduino:

Einbindung der library und Aufruf der Rover-Instanz

```
#include <lets go ING_Drive.h>
```

```
Drive Rover;
```

Befehl in der loop

```
Rover.driveDistance(50,50,HIGH);
```

fahre Distanz

Mit dem Block Fahre Distanz kann man festlegen, mit welcher Geschwindigkeit (0 bis 100), in welche Richtung, (HIGH) oder (LOW) und wie weit (in cm) das Fahrzeug fahren soll. Die Distanz wird aus den von den Raddecodern erzeugten Interrupts berechnet. Beim Decoder kommt es häufig zu prellenden Werten. Deshalb wird nur der niedrigere Wert ausgewählt.

10.3 drehe

ArduBlock:



Arduino:

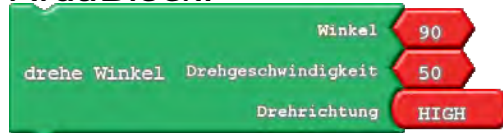
```
Rover.turn(50,HIGH);
```

drehe

Mit dem Block Drehe kann man festlegen mit welcher Geschwindigkeit (0 bis 100) und in welche Richtung (HIGH) oder (LOW) das Fahrzeug um die eigene Achse drehen soll.

10.4 Drehe Winkel

ArduBlock:



Arduino:

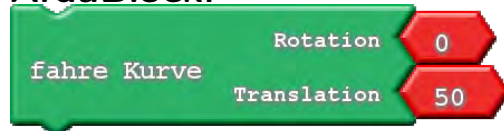
```
Rover.turnAngle(90,50,HIGH);
```

drehe Winkel

Mit dem Block Drehe Winkel kann man festlegen mit welcher Geschwindigkeit (0 bis 100), in welche Richtung (HIGH) oder (LOW) und um welchen Winkel (0 bis 360 grad) das Fahrzeug um die eigene Achse drehen soll.

10.5 fahre Kurve

ArduBlock:



Arduino:

```
Rover.driveCurve(0,50);
```

fahre Kurve Mit dem Block “Fahre Kurve” kann man wie bei einer Panzersteuerung die Drehbewegung (Rotation) $[-100,100]$ und die Vorwärtsbewegung (Translation) $[-100,100]$ festlegen.

In der LGL_Drive-Library werden aus diesen beiden Bewegungen die Geschwindigkeiten für die beiden Motoren berechnet.

Mit diesem Block können vom Gerauslauf über weite und enge Kurven bis hin zur Drehung um die eigene Achse alle Bewegungen ausgeführt werden.

Bei einer reinen Drehbewegung bewegt sich ein Motor mit der eingegebenen Geschwindigkeit vorwärts, der andere rückwärts. Die resultierende Bewegung ergibt sich aus der Addition der Vorwärtsbewegung und Drehbewegung.

Beispiele findest du in Kapitel 11.7

10.6 fahre Kurve Distanz

ArduBlock:



Arduino:

```
Rover.driveCurveDistance(50,0,50);
```

fahre Kurve Distanz

Mit dem Block Fahre Kurve Distanz kann man über die Panzersteuerung festlegen wie weit (in cm) das Fahrzeug in die eingestellte Richtung fahren soll. Rotation = Drehbewegung [-100,100] und Translation = Vorwärtsbewegung [-100,100].

10.7 Motoren stopp

ArduBlock:



Arduino:

```
Rover.stopp();
```

Motoren stopp

Stoppt beide Motoren. Das Fahrzeug wird allerdings nicht gebremst und fährt somit noch ein Stück weiter.

10.8 Motoren bremsen

ArduBlock:



Arduino:

```
Rover.brake();
```

bremsen

Das Fahrzeug wird aktiv gebremst. Es hält ruckartig an.

10.9 Messung starten

ArduBlock:



Arduino:

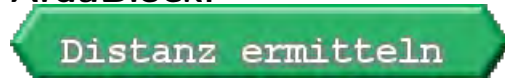
```
Rover.startMeasure();
```

Messung starten

Mit dem Block "Messung starten" kann der Punkt festgelegt werden, ab welchem die gefahrene Distanz ermittelt werden soll.

10.10 Distanz ermitteln

ArduBlock:



Arduino:

```
if (Rover.getDistance() > 50)
```

Distanz ermitteln

Mit dem Block "Distanz ermitteln" wird die gefahrene Distanz zurückgegeben.

11 Beispiele

In diesem Kapitel findest du einige Programm-Beispiele. Du kannst sie als "Steinbruch" oder als Nachschlagewerk verwenden. Die meisten Blöcke, die in der Blockreferenz vorgestellt wurden, sind hier in einem Programm integriert. Im Text wird kurz erläutert, welche Funktion ein Block im Programm hat.

11.1 Falls LDR-Wert größer als, dann..., sonst

setze analoge Variable auf - serial println - falls / sonst - warte ms - serial println

ArduBlock:



Erläuterung:

Wiederhole fortlaufend: loop-Block: Alles was in der Klammer unter "mache" steht wird nacheinander abgearbeitet. Sobald das Programm unten angekommen ist, fängt es oben wieder an.

Der am Analog-Pin 1 anliegende Wert wird in die Variable ldr geschrieben.

Falls die Variable "ldr" größer als 500 ist, dann setze digital Pin 13 auf HIGH, sonst auf LOW

Warte 200 Millisekunden

Gib auf dem Serial Monitor die Zeichenkette "Wert ldr:" gefolgt vom Wert der Variablen "ldr" aus.

11.2 blaue RGB dimmen: Werte nur alle 200 ms anzeigen lassen

warte Millis - Unterprogramm

Aufgabe: Der PWM-Wert für die blaue LED soll über die Variable "farbwert" alle 20 ms um 1 erhöht werden. Der momentan erreichte Wert dieser Variable soll aber, der besseren Lesbarkeit willen, nur alle 200 ms auf dem Serial Monitor angezeigt werden.

Problem: Wenn du in das Programm eine Wartezeit (delay) von 200 ms einbaust, dann stockt das Programm an dieser Stelle für 200 Millisekunden. Die Variable "farbwert" wird während dieser Pause nicht weiter hochgezählt. Was tun?

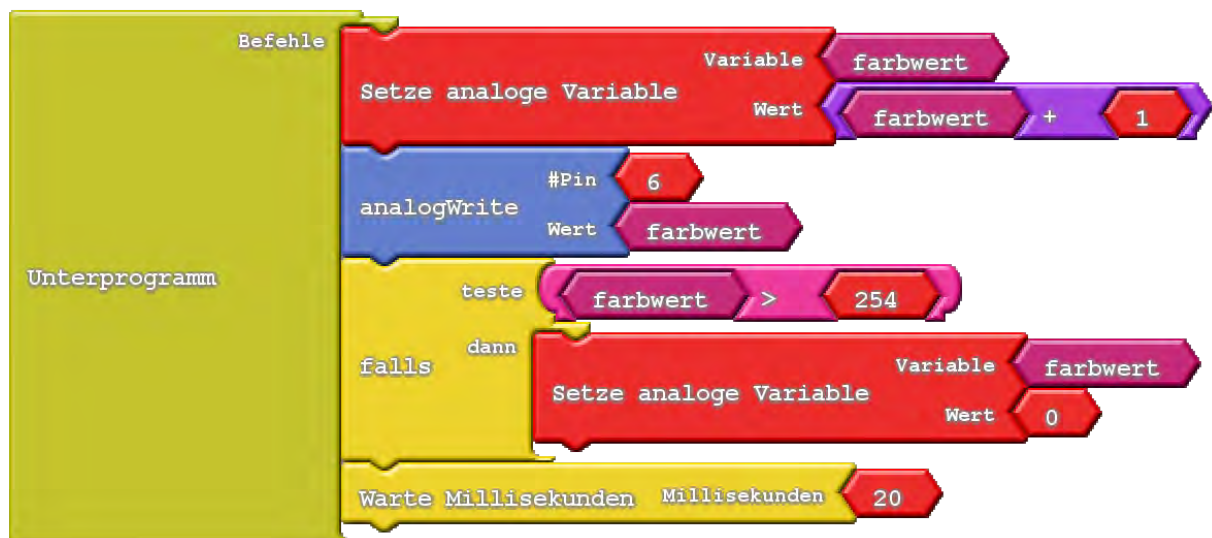
warte Millis: Mit dem Block "Warte Millis" kann man diese Problem lösen. Alles was in diesem Block unter "make" steht, ist das Pausenprogramm, das permanent abgearbeitet wird. Unter "Zeit" schreibt, man, alle wie viele Millisekunden das Rahmenprogramm ausgeführt werden soll. Das Rahmenprogramm steht unterhalb vom "warte Millis Block".

Unterprogramm: Damit die Sache etwas übersichtlicher ist, wollen wir das Pausenprogramm in ein Unterprogramm auslagern. Wir ziehen den Block "Unterprogramm" in die "make-Schleife". Dem Gegenstück "Unterprogramm" müssen wir exakt den gleichen Namen geben. Dieser Block hat keine Konnektoren.

Nun zum Unterprogramm:

1. der Farbwert wird um 1 hochgezählt
2. die RGB wird mit dem neuen Wert versehen und somit etwas dunkler leuchten (common Anode)
3. Falls der "farbwert" einen Wert größer als 254 annehmen sollte, wird er auf den Wert Null zurückgesetzt.
4. Das Programm wartet 20 Millisekunden bevor eine neue Schleife durchlaufen wird.

ArduBlock:



11.3 Taster entprellen

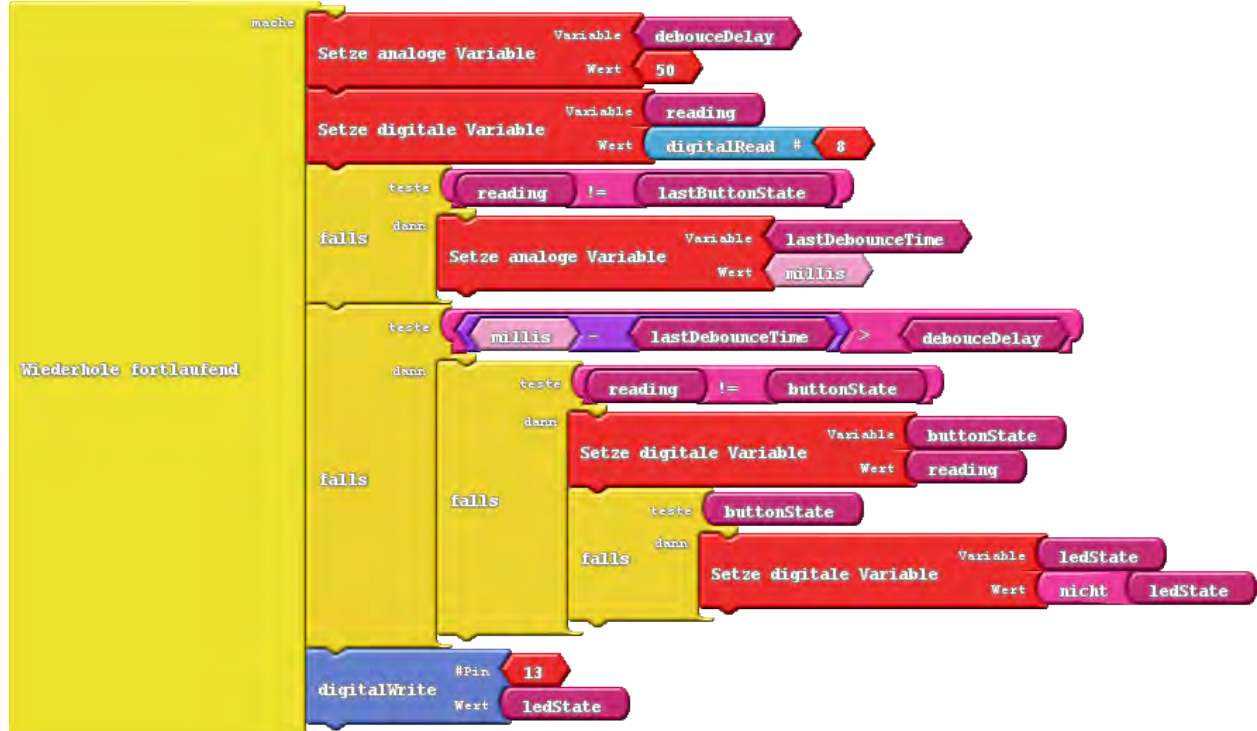
setze digitale Variable - logischer Operator nicht - invertieren

Aufgabe: Softwarelösung für das Entprellen ein Tasters / Schalters

Problem: Wenn man auf den Taster drückt, federt er. Das bedeutet der Kontakt wird nicht direkt geschlossen, sondern in Bruchteilen von Sekunden mehrfach auf und zu gemacht, bis er endlich ganz geschlossen ist. Dadurch kommt es zu unerwünschten Effekten: z.B. obwohl man nur einmal auf den Taster gedrückt hat, wird die LED erst ein und dann gleich wieder ausgeschaltet. Wenn man diesen Effekt vermeiden will, muss man nach dem Schließen des Kontakts ca. 100ms warten, bis das Federn aufgehört hat. Der englische Begriff für das Entprellen lautet "debounce". Eine einfache Variante besteht darin eine Pause (delay) von 100 ms einzubauen.

millis: Da wir das Programm nicht wirklich anhalten wollen, arbeiten wir mit der Funktion "millis". Mit dem Befehl "millis" wird nach dem Programmstart eine Stoppuhr gestartet, die die Millisekunden zählt. Im folgenden Programm wird unser Schalterprogramm nur dann gestartet, wenn die verstrichene Zeit größer als die festgelegt Pause (debouceDelay) ist.

ArduBlock:



11.4 Servo, Servo Trennen

Servo Winkel einstellen, Servo verbinden, Servo trennen

ArduBlock:



Erläuterung:

Der Servo hat drei Kabel: Das rote kommt an 5 Volt, das schwarze an Gnd, das orange bzw. weiße an einen Pin (hier Pin8).

Manche Servos brauchen mehr als 500 mA Strom. Der Arduino liefert über seinen Spannungswandler jedoch nur maximal 500 mA. Auf der Platine des lets go ING-Fahrzeugs kann man daher die Servos direkt über die Batteriespannung betreiben. Dazu muss die Pin-Brücke an Pin 8-2 auf BATT gesetzt werden. Nun kannst du den Servo an D8-2 anschließen. Signal kommt auf 1.

Der Servo dreht auf 0 Grad. Nach einer Wartezeit von 1000 ms dreht er auf 180 Grad. Er hält aktiv diese Position bis wir ihn mit dem Block "Servo Trennen" freischalten. Nun lässt er sich für eine Sekunde mit der Hand freidrehen, bevor das Programm wieder von vorne anfängt.

Den Block "Servo verbinden" brauchen wir nur, wenn im Programm der "Servo trennen"-Block vorkommt. Standardmäßig schreibt der Block "Servo drehe auf Grad" den "Servo verbinden Befehl" ins Setup des Programms. Da dies jedoch nur einmal ausgeführt wird, würde in unserem Fall beim 2. Durchlauf nichts mehr passieren.

11.5 LED-Balken hochzählen

for-Schleife - led bar - analoge Variable - Serial print

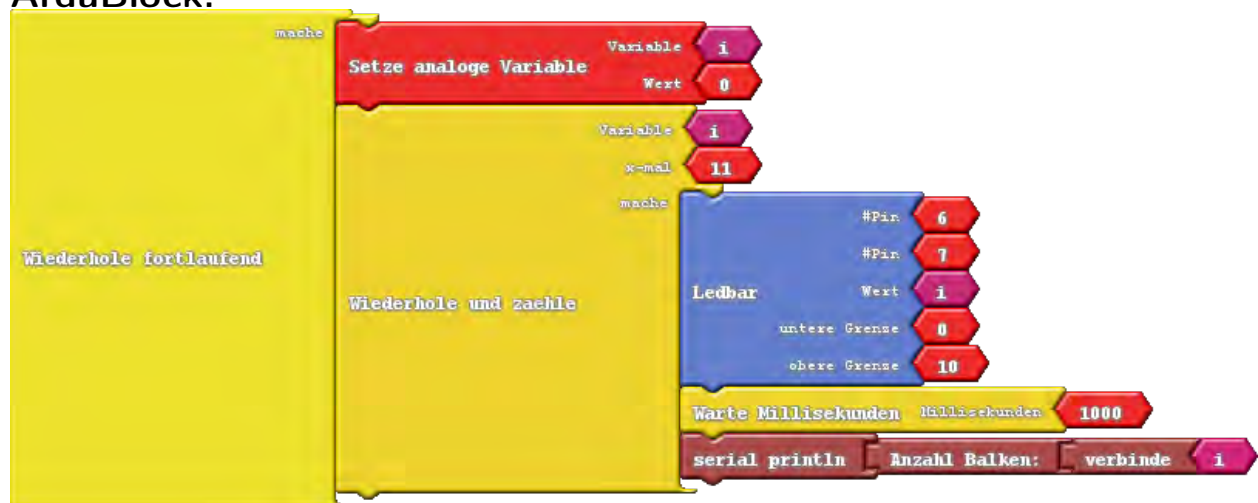
Aufgabe: Zu Beginn soll kein Balken leuchten. Im Sekundentakt soll ein Balken hinzukommen. Wenn alle leuchten, soll das Programm von vorne beginnen.

led bar: Die led bar wird mit Pin 6 (gelbes Kabel) und Pin 7 (weißes Kabel) verbunden. Rotes Kabel an 5V, Schwarzes Kabel an 0V.

Untergrenze und Obergrenze festlegen

Wenn man als Untergrenze 0 und als Obergrenze 10 festlegt, dann werden die Balken gleichmäßig auf diese Werte verteilt: Null ist kein Balken – 10 sind 10 Balken.

ArduBlock:



11.6 Quadrat fahren - gefahrene Strecke messen

Fahre Distanz, drehe Winkel, Messung starten, Distanz ermitteln, bremsen

Aufgabe Teil A: Das Fahrzeug soll ein Quadrat mit Seitenlänge 50 cm fahren.

Lösung:

Im "Wiederhole 4 mal - Block" stehen die Anweisungen

"fahre 50cm mit Geschwindigkeit 50 vorwärts (HIGH)"

"drehe um 90 Grad mit Drehgeschwindigkeit 50 im Uhrzeigersinn (HIGH)".

Aufgabe Teil B: Das Fahrzeug soll während der Fahrt über die LED-Bar die gefahrene Strecke anzeigen. Ein Balken soll 10 cm gefahrene Strecke markieren.

Lösung:

Messung starten - Fahrzeug starten: Mit dem Block "Messung starten" wird der Startwert jetzt auf Null gesetzt. Anschließend wird über den Block " Fahre mit Geschwindigkeit 50 und Richtung HIGH" das Fahrzeug in Bewegung gesetzt.

solange-Schleife: Das Fahrzeug soll nun solange fahren, wie die Variable "Distanz gefahren" kleiner als 100 cm ist. Die tatsächlich gefahrene Strecke muss in der "solange -Schleife" ständig neu in die Variable "Distanz gefahren" eingelesen werden. Dies erreicht man, indem man die analoge Variable "Distanz gefahren" auf den Block "Distanz ermitteln" setzt.

led bar: Als untere Grenze wählt man bei der led bar den Wert 0. Als obere Grenze den Wert 100. Als Wert wird die Variable "Distanz gefahren" eingetragen.

Schleife verlassen: Wenn die gefahrene Distanz nicht mehr kleiner als 100 cm ist, kann das Programm die "solange-Schleife" verlassen. Das Fahrzeug wird mit dem Block "bremsen" aktiv gestoppt. Es rollt nicht aus!. Nach 5 Sekunden Pause beginnt das Programm von vorne.

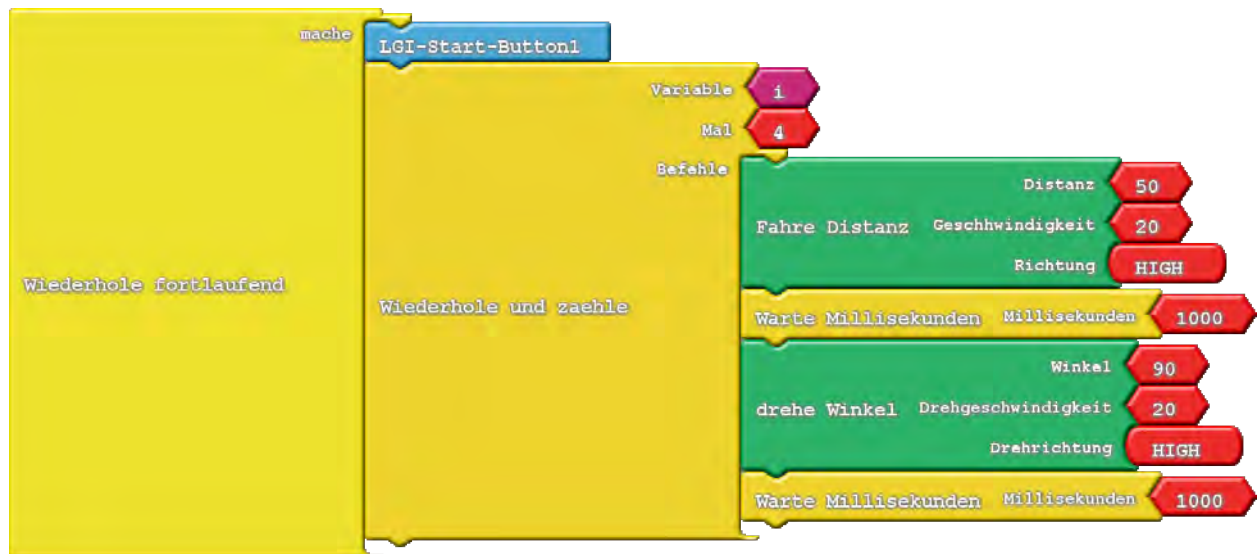


Abbildung 11.1: Quadrat fahren

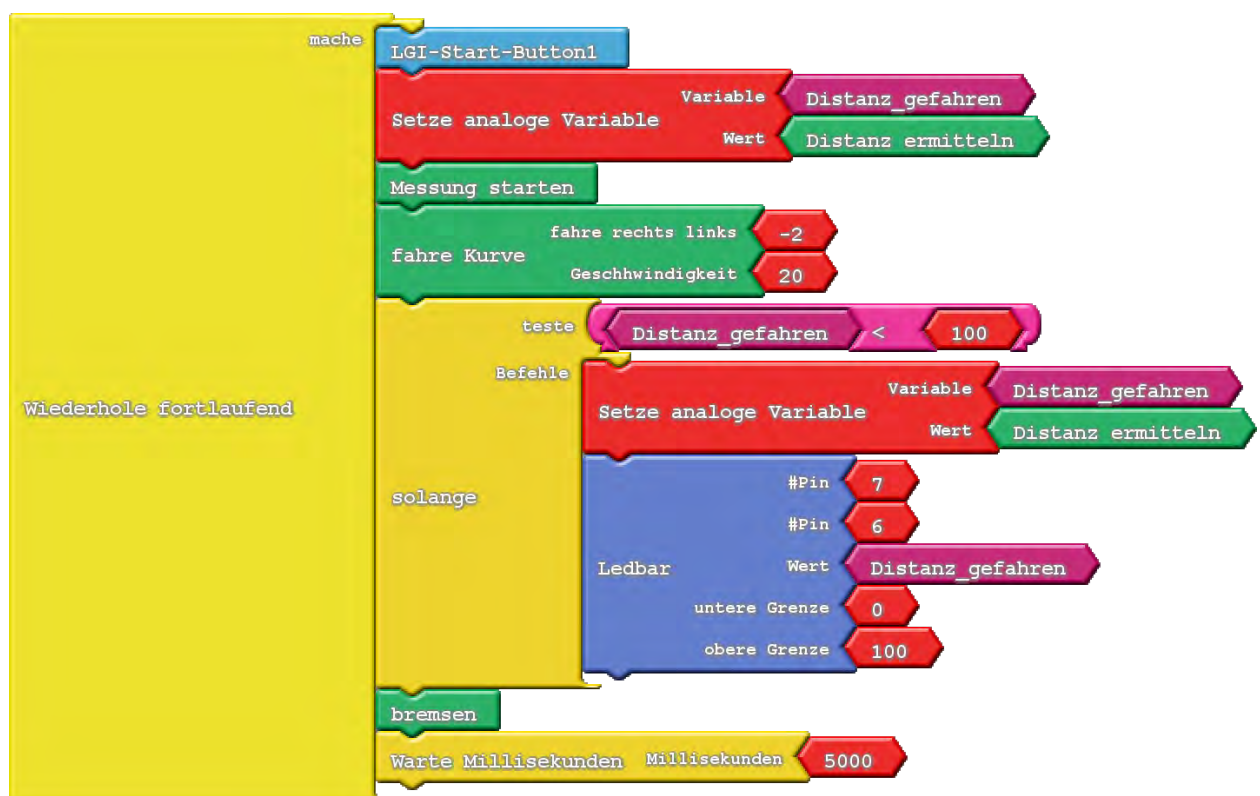


Abbildung 11.2: Strecke messen

11.7 Kurven fahren - fahre Kurve Distanz

Aufgabe: Das Fahrzeug soll mit dem "Fahre Kurve Distanz Block" bestimmte Strecken fahren

Mit dem Fahre-Kurve-Block legt man sowohl die Vorwärtsbewegung (Translation) als auch die Drehbewegung (Rotation) fest. In der LGL_Drive-Library werden aus diesen beiden Bewegungen die Geschwindigkeiten für die beiden Motoren berechnet.

Mit diesem Block können vom Geradeauslauf über weite und enge Kurven bis hin zur Drehung um die eigene Achse alle Bewegungen ausgeführt werden.

Bei einer reinen Drehbewegung bewegt sich ein Motor mit der eingegebenen Geschwindigkeit vorwärts, der andere rückwärts. Die resultierende Bewegung ergibt sich aus der Addition der Vorwärtsbewegung und Drehbewegung.

Beispiel:

Will ich, dass mein Fahrzeug eine "weite Kurve" fährt, muss es sich leicht drehen und mit mittlerer Geschwindigkeit vorwärts fahren. Dazu gebe ich der Drehbewegung den Wert 10 [-100 bis 100] und der Vorwärtsbewegung den Wert 50 [-100 bis 100].

Die Berechnung sieht dann so aus:

$$\text{Motor 1} = \text{Vorwärtsbewegung} + \text{Drehbewegung} = 50 + 10 = 60$$

$$\text{Motor 2} = \text{Vorwärtsbewegung} - \text{Drehbewegung} = 50 - 10 = 40$$

- a) Fahre 50 cm vorwärts
- b) Fahre 50 cm rückwärts
- c) Fahre 50 cm in einem weiten Bogen vorwärts nach links
- d) Fahre 50 cm in einem engeren Bogen vorwärts nach links
- d) Fahre 50 cm in einem leichten Bogen rückwärts in die Ausgangsposition
- e) Drehe dich um die eigene Achse gegen den Uhrzeigersinn 50 cm
- f) Drehe dich um das linke Rad 50 cm
- g) Drehe dich um die eigene Achse im Uhrzeigersinn 50 cm

ArduBlock:

a) Vorwärts: geradeaus mittlere Geschwindigkeit (50cm)	
b) Rückwärts: geradeaus mittlere Geschwindigkeit (50cm)	
c) Vorwärts: Weite Linkskurve mittlere Geschwindigkeit (50cm)	
d) Vorwärts: Enge Linkskurve mittlere Geschwindigkeit (50cm)	
e) Rückwärts in Ausgangsposition Enge Kurve mittlere Geschwindigkeit (50cm)	
f) Dreht um das linke Rad mit mittlerer Geschwindigkeit (50cm)	
g) Dreht gegen den Uhrzeigersinn um eigene Achse mit mittlerer Geschwindigkeit (50cm)	

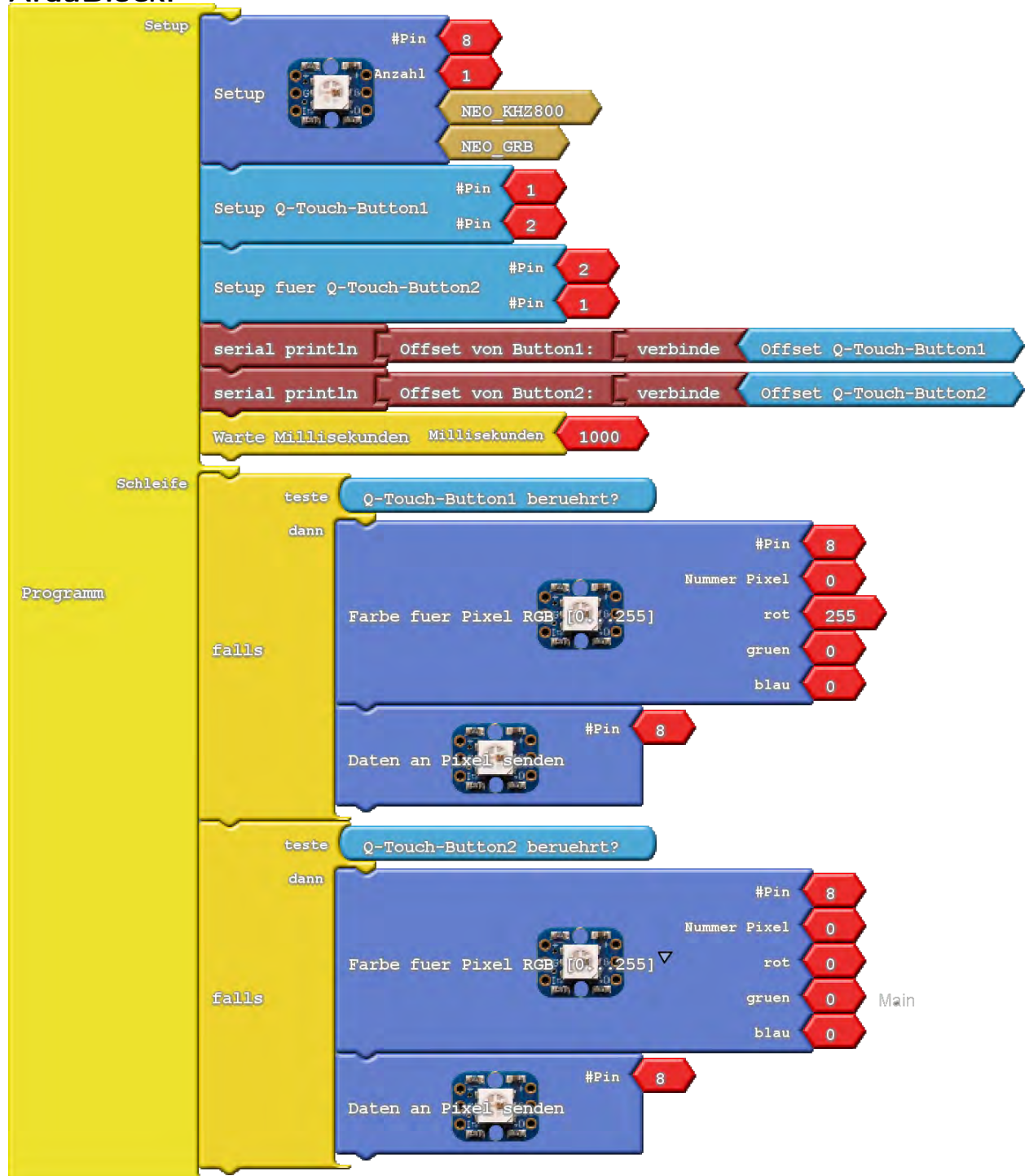
11.8 Neopixel QTouch

Aufgabe: Beim Berühren von Q-Touch1 soll die Neopixel-GRB rot leuchten, bei Berühren von Q-Touch2 soll sie aus gehen.

Mit den Blöcken werden automatisch die beiden Libraries Adafruit_NeoPixel.h und LQI_QTouch.h einbunden.

- a) Setup für Neopixel: Wir verwenden keine RGB-Neopixel, sondern eine GRB-Neopixel (green, red, blue) mit 800kHz. Im Beispiel sind die Neopixel mit Pin 8 verbunden. Es wird hier nur ein Pixel verwendet.
- b) Die Pins der beiden Q-Touch-Buttons werden im Setup festgelegt: Q-Touch-Button1 hängt an Pin1 und hat Pin2 als Partnerpin. Q-Touch-Button2 hängt an Pin2 und hat Pin1 als Partnerpin.
- c) Über den Serialprint-Befehl werden die beiden Offset-Werte einmalig angezeigt. Nach einem Reset hat man 1000 ms Zeit den SerialMonitor einzuschalten.
- d) Falls Q-Touch-Button1 berührt, dann soll die GRB-Neopixel rot leuchten.
- d) Mit dem Block "Daten an Pixel" senden werden die Informationen wirksam.
- e) Falls Q-Touch-Button2 berührt, dann soll die GRB-Neopixel ausgehen.
- f) Mit dem Block "Daten an Pixel" senden werden die Informationen wirksam.

ArduBlock:



11.9 Bluetooth Kommunikation

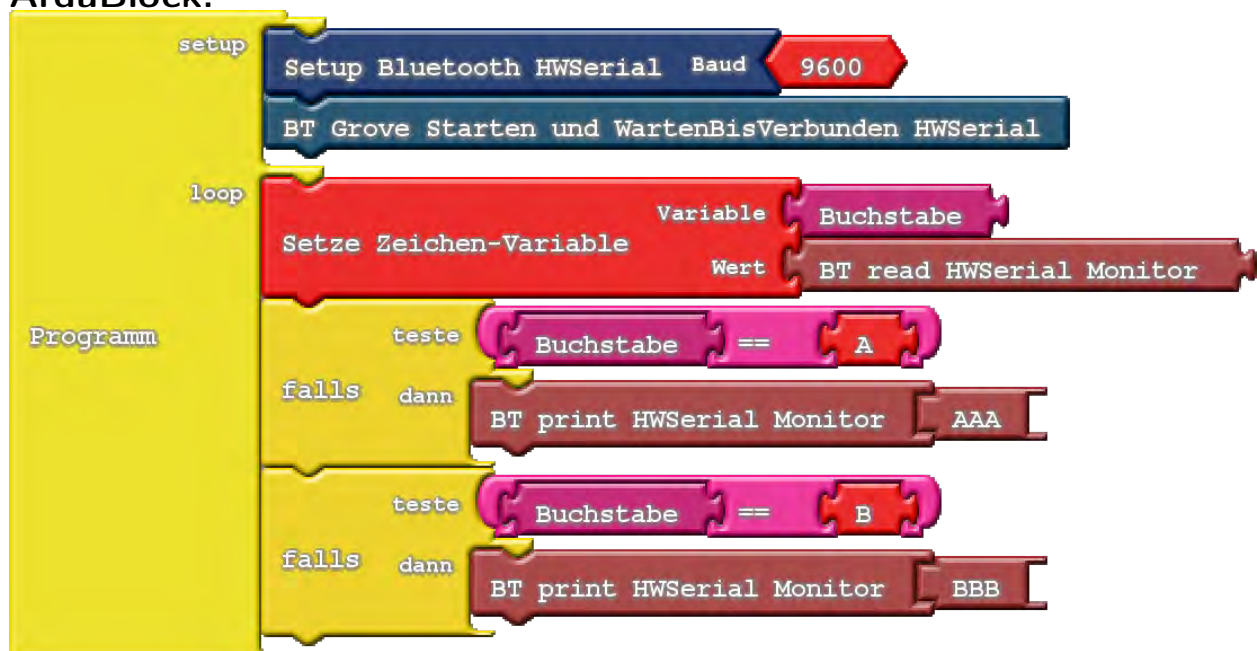
Aufgabe: Vom Monitor des Tablets sollen Daten an den Arduino gesandt werden. Wenn man A sendet, dann soll der Arduino AAA zurückgeben. Wenn man B sendet, dann soll der Arduino BBB zurücksenden. Vor dem Übertragen des Programms muss du das BT-Modul ausstecken!

- Setze in den Setup den Block "Setup Bluetooth HWSerial"
- Wenn du das Grove-Modul verwendest, dann füge den Block "BT Grove Starten und WartenBisVerbunden HW Serial" ein
- Setze Char-variable "Buchstabe" auf "BT read HWSerial Monitor"
- falls "Buchstabe" == A, dann schreibe AAA
- falls "Buchstabe" == B, dann schreibe BBB

Nach dem Übertragen des Programms muss du das BT-Modul wieder bei Pin0 (gelb) und Pin1 (weiß) einstecken!

Bevor du Daten empfängst, musst du deinen Arduino und das BT-Modul Resetten!

ArduBlock:



12 Arbeit mit dem Programmablaufplan

Hier kann kannst du den PAP-Designer umsonst herunterladen:
friedrich-folkmann.de/papdesigner/Hauptseite.html

Mit einem Programmablaufplan kann man zeigen, welche Vorgänge nacheinander ausgeführt werden. Die Aufgabenstellung liegt meist als Text vor.

1. Lies diesen Text sehr aufmerksam durch.
2. Unterstreiche alle Schlüsselworte: Schlüsselworte sind alle Begriffe, die Auskunft darüber geben, wann was unter welchen Bedingungen wie oft geschieht. Denke daran, dass unser Mikrocontroller das Programm Zeile für Zeile abarbeitet.

Der Programmablauf wird auch als Flussdiagramm bezeichnet. Man kann sich vorstellen, dass man oben in ein Boot einsteigt und mit dem Boot einen Fluss herunter treibt.

12.1 Verzweigung

PAP Symbol:



Verzweigung

In einem Programm kann es immer wieder zu einer Verzweigung kommen, an der prinzipiell zwei Möglichkeiten offen stehen: Ja oder Nein! Wie lautet die Entscheidungsfrage?

Beispiele:

Zählschleife: Ist die Zählvariable größer als der Sollwert, dann Schleife verlassen.

Variable TRUE oder FALSE: Ist eine Variable TRUE, dann den ja-Weg gehen, sonst den FALSE - Weg gehen.

Variable TRUE: Nur wenn die Variable TRUE ist, dann abbiegen.

Variable größer als: Nur wenn Variable größer als bestimmter Wert oder andere Variable, dann abbiegen

12.2 Eingabe

PAP Symbol:



Eingabe

Mit dem Eingabesymbol werden Vorgänge erfasst, die von außen kommen. z.B. wenn ein Wert über einen Sensor eingelesen wird.

12.3 Ausgabe

PAP Symbol:



Ausgabe

Mit dem Ausgabesymbol werden Vorgänge erfasst, die nach draußen gehen: z.B. wenn Werte auf dem Serial Monitor angezeigt werden sollen oder eine LED ein- oder ausgeschaltet wird. Oder ein Motor mit einer bestimmten Geschwindigkeit sich drehen soll.

12.4 Vorgang

PAP Symbol:



Vorgang

Mit dem Symbol "Vorgang" werden z.B. Berechnungen oder Operationen gekennzeichnet: z.B. setze eine Variable auf folgenden Wert oder invertiere einen Wert.

Die Symbole kannst du mit Pfeilen verbinden. Auch Schleifen werden mit Pfeilen gekennzeichnet. Bei Arduinoprogrammen gibt es in der Regel kein Ende. Der Block "Wiederhole fortlaufend" beinhaltet entweder die Anweisung "falls i kleiner Unendlich" oder die Anweisung "solange TRUE", was immer der Fall ist.