Space Rover mit dem MIT App Inventor und Arduino
Dokumentation

Informatik
Oberstufe

Katharina Zorko, Eric Brinkmann
kontakt@ericbrinkmann.de

15.12.2020

Vorwort

Dieses Projekt ist in zwei Teilen aufgebaut, die Programmierung des Arduinos und die Programmierung des MIT App Inventors. Je nach Ziel der Unterrichtseinheit und des Vorwissens der SuS kann natürlich alles von den SuS selbst programmiert und zusammengebaut werden, das kostet allerdings auch enorm viel Zeit. Die denkbar günstigste Lösung wäre eine Kooperation mit dem Fach NWT, wo dann der technisch anspruchsvolle Teil stattfinden kann.

Das hier dargestellte Autos ist nur eine von vielen Möglichkeiten, der grundlegende Code sollte aber im Großen und Ganzen gleich sein. Lediglich bei der Pinbelegung müssen bei anderer Hardware ggf. Änderungen vorgenommen werden.

Die hier verwendeten Sensoren bieten eine große Vielfalt und müssen je nach Zeit und Verfügbarkeit nicht alle genutzt werden. Bei der Wahl der Sensoren sind aber theoretisch keine Grenzen gesetzt.

1 Arduino

Für dieses Projekt werden folgende Bestandteile verwendet:

- Arduino Uno (oder Arduino Mega, siehe Kapitel 1.3)
- L293d Motor Driver Shield oder L298n Dual H-Bridge
- HC-06 Bluetoothmodul zur Kommunikation
- Powerbank o.Ä. (siehe Abschnitt 1.2 auf der nächsten Seite)
- DHT11: Temperatur- und Feuchtigkeitsmodul (nicht notwendig, wenn BME280 genutzt wird)
- BME280 Barometrischer Sensor für Luftfeuchtigkeit, Luftdruck und Temperatur
- Fotowiderstand
- HC-04 Ultraschallsensor zur Hinderniserkennung
- GY-271 (QMC5883L) Kompassmodul zur Messung des Magnetfeldes
- MQ-2 Gassensor (optional)

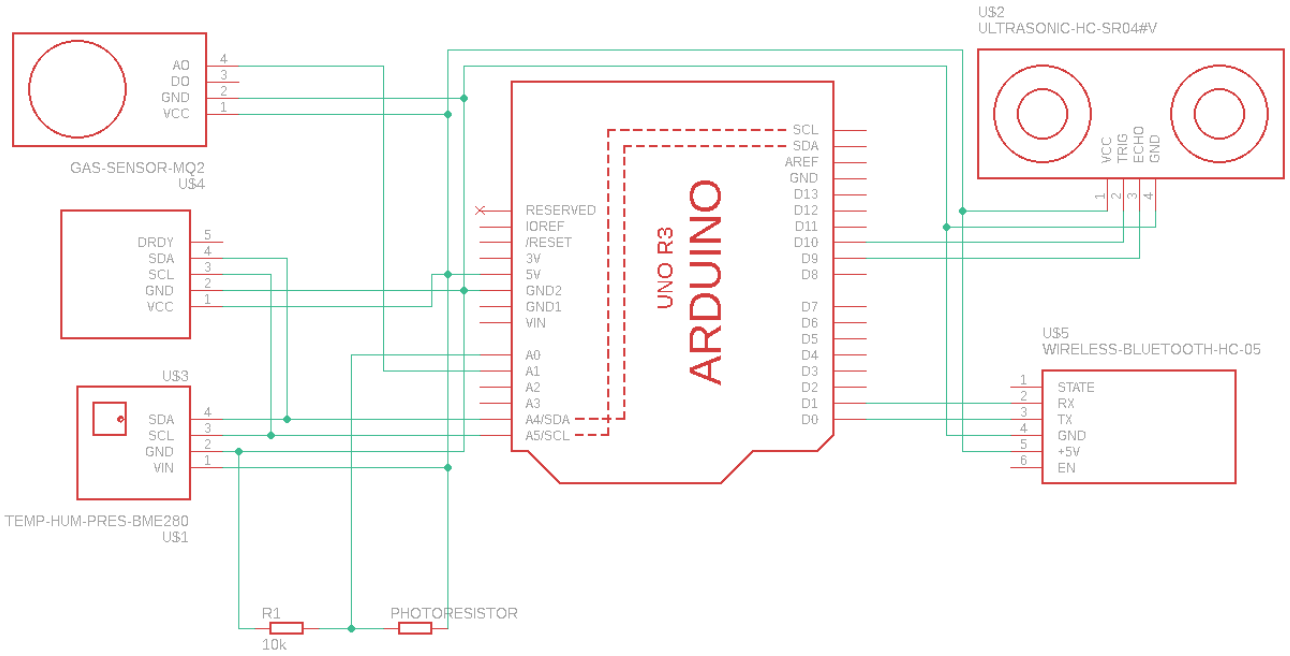

Abbildung 1: Übersicht aller Sensoren

Generell gilt für die folgenden Kapitel, dass diese nur als Schnellübersicht gedacht sind, für die meisten Sensoren ist noch eine weiterführende Anleitung auf Englisch verlinkt.

1.1 Das Auto

Als Basis des Autos dient ein Set bestehend aus einer vorgefertigten Karosserie mit vier Gleichstrommotoren (3-6V, max 200mA) plus Reifen. Alternativ zur vorgefertigten Karosserie kann natürlich auch ein selbstgebautes oder gedrucktes Auto verwendet werden. Die Anforderungen an die Karosserie sind nicht besonders hoch, es sollte lediglich in der Lage sein, sich fortzubewegen. Bei einem Auto mit zwei Schichten ist es sinnvoll, obere Schicht erst ganz zum Schluss zu fixieren, um die Kabel der Sensoren einfacher verlegen zu können.

Wo genau die Sensoren platziert werden, weitestgehend irrelevant, es sollte nur darauf geachtet werden, dass sie ihre Werte nicht gegenseitig verfälschen. So sollte z.B. der MQ2, der sehr warm werden kann, nicht direkt neben dem Thermometer platziert werden und der Fotowiderstand sollte nicht verdeckt sein.

Zur Ansteuerung der Motoren gibt es verschiedene Möglichkeiten, die in den folgenden Kapiteln erläutert werden.

1.2 Stromversorgung

Zur Stromversorgung des Arduinos dient in diesem Fall eine Powerbank mit zwei Ausgängen mit jeweils 5V/2.4A und 10.000 mAh Kapazität. Das Problem bei den meisten Powerbanks ist, dass sie sich automatisch ausschalten, wenn nicht ausreichend Strom (meist mindestens 80-100mA) an einem Ausgang ausgegeben wird. Das Bluetoothmodul scheint allerdings recht viel Strom zu verbrauchen, sodass zumindest das hier verwendete Modell einer Powerbank[1] dauerhaft angeschaltet bleibt. Oft wird auch gerne eine 9V Batterie verwendet. Da diese aber Probleme mit dem Bluetooth-Modul gemacht hat und recht schnell geleert ist, würde ich davon für einen dauerhaften Einsatz absehen.

Die Stromversorgung für die Motoren hängt maßgeblich davon ab, welche Art von Motoren und welche Kontrolleinheit für die Motoren verwendet werden. Die Ausführungen hier beziehen sich auf typische kleine gelbe Gleichstrommotoren mit 3-6V und max 200mA. Werden stärkere Motoren verwendet, wird dementsprechende eine stärke Stromquelle benötigt. Der Unterschied von zwei möglichen Kontrolleinheiten wird in den folgenden Kapiteln beleuchtet.

[1] Auf eine explizite Namensnennung wird hier verzichtet

Im Optimalfall könnte man die gesamte Stromversorgung auch mit Solarzellen und einem Akku umsetzen. Das wird allerdings sehr schnell sehr teuer und komplizierter. Eine mögliche Umsetzung folgt aber bei Gelegenheit.

1.3 L293d Motor Driver Shield

Das L293d Motor Driver Shield hat den Vorteil, dass es einfach angesteckt werden kann, und alle vier Räder unabhängig voneinander angesteuert werden können. Nachteilig ist jedoch, dass alle Sensoren angelötet werden müssen, da das Shield die Anschlüsse der Pins verdeckt. Außerdem werden bereits viele Pins vom Shield selbst benutzt, siehe hier. Das vollständige Projekt nutzt so alle digitalen Pins aus. Nutzt man statt des Arduino Unos die etwas teurere und größere Mega-Variante, so kann man auch dieses Problem umgehen, da noch zusätzliche Pins zur Verfügung stehen, die nicht vom Shield verdeckt sind. In dem Fall muss die Pinbelegung im Code angepasst werden.

Die wohl geschickteste Variante wäre das Adafruit Motor Shield V2, eine verbesserte Version dieses Shields, welches die Pins freigibt. Da es allerdings auch viel teurer ist, hätte es den preislichen Rahmen dieses Projekts gesprengt.

Zwar ist es grundsätzlich möglich, Arduino und L293d Motor Driver Shield mit nur einer Stromquelle zu versorgen, aber da es dadurch aber häufig zu Fehler kommen kann (entweder durch *noise* im Stromkreislauf oder zu wenig Stromstärke), empfiehlt es sich, beide separat zu betreiben.

Mit dem Shield reichen die 5V und ca. 1.2A des zweiten Ausgangs der Powerbank auf jeden Fall aus, wenngleich die Motoren so nicht die maximale Drehzahl erreichen. Auch ist die Geschwindigkeitssteuerung eher schwierig, denn mit viel weniger als der maximalen Drehzahl bleibt das Auto auch gerne mal stehen, da die Motoren dann nicht stark genug sind.

1.4 L298n Dual H-Bridge

Wenn die Steuerung des Autos sich wie die eines Panzer verhalten soll (es wird nur zwischen linker und rechter Motor unterschieden), reicht theoretisch eine L298n Dual H-Brücke, die in der Lage ist, zwei Gleichstrommotoren unterschiedlich zu steuern, auch aus. Die L298n Dual H-Bridge kann, wie der Name schon vermuten lässt, zwei Gleichstrommotoren gleichzeitig steuern. Bei ausreichender Stromversorgung kann aber jeder Anschluss doppelt belegt werden (unterteilt in rechte und linke Motoren), sodass auch ein Auto mit vier (oder sogar mehr) Rädern problemlos gesteuert werden kann.

Der Vorteil gegenüber des L293d Shields ist, dass es erstens günstiger ist und zweitens die meisten Pins nach wie vor frei sind.

Das einzig Problematische ist der Spannungsverlust von 2V der Eingangslast. Schließt man jetzt z.B. die 5V Powerbank an, kommen noch knappe 3V bei den Motoren an, wodurch diese sehr langsam sind, und was eine Geschwindigkeitsregulierung unmöglich macht. Es ist also durchaus sinnvoll eine stärkere Spannung anzulegen.

Hinweis: Es wurde keine gesonderte Version für diese Art der Ansteuerung hochgeladen, um bei etwaigen kleinen Änderungen leichter den Überblick zu behalten. Um das Hauptprogramm anzupassen muss lediglich der Code für das L293d Motor Driver Shield durch den oben verlinkten (und auch unter *Sensoren* im Ordner zu finden) ersetzt werden.

1.5 HC-04 Ultraschallsensor

1.6 DHT11

Der DHT11 wird nur benötigt, wenn der BME280 nicht benutzt wird. Andernfalls wird das Programm unnötig verlangsamt.

Zur Messung von Temperatur eignen sich der DHT11 und der fast baugleiche DHT22. Ersterer ist etwas günstiger, aktualisiert die Werte jede Sekunde und ist schon in vielen Sets enthalten, ist aber nur auf +-2° genau. Der DHT22 ist zwar genauer, kostet dafür aber auch mehr und ist deshalb für den Schuleinsatz nur bedingt zu empfehlen. Den DHT11 gibt es mit drei Pins und bereits eingebautem Widerstand sowie als Version mit drei Pins, bei dem ein Widerstand von 4,7kOhm hinzugefügt werden muss.

Der Einfachheit halber wurde hier die Version mit drei Pins gewählt, die einfach nur (von vorne links anfangend) an einem digitalen Pin, 5V und GND angeschlossen werden, wie Abbildung 2 zu entnehmen.

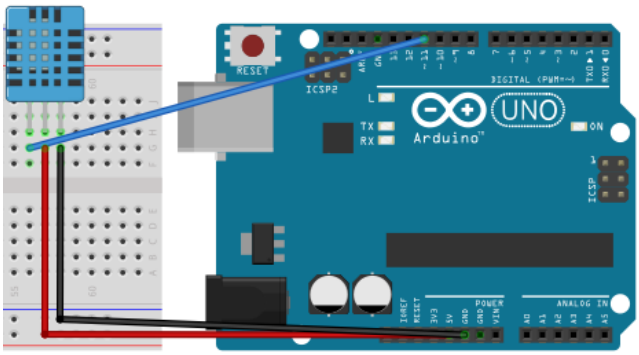

Abbildung 2: Verbindung des DHT11 mit 3 Pins

Die Nutzung des Sensors ist nicht intuitiv möglich, denn es wird noch die *dht.h*-Bibliothek benötigt, die zudem noch einige Einstellungen verlangt. Ist dies geschehen, ist das eigentliche Auslesen dann aber kein Problem:

```
#include "DHT.h"
#define DHTPIN 11
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600); //Initialisierung Serieller Monitor
  dht.begin(); //Initialisierung Temperatursensor
}

void loop() {
  Serial.println(dht.readTemperature());
  delay(1000);
}
```

1.7 BME280

Der BME280 nutzt I2C und muss dementsprechend an die I2C Pins des Arduinos angeschlossen werden. Dies sind im Falle des Arduino Unos A4 für SDA und A5 für SCL. Einige Modelle haben noch ein zusätzliches Paar I2C Pins. Da diese untereinander verbunden sind, macht es keinen Unterschied, welcher von beiden genutzt wird. Wie üblich müssen dann auch noch GND und 5V verbunden werden.

Für die Verwendung des Sensors muss noch die Adafruit BME280 Bibliothek installiert werden, sowie die Bibliotheken auf denen diese aufbaut (wird im Installationsmenu abgefragt). Ein Basis-Programm könnte wie folgt aussehen:

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME280 bme;

```

```
void setup() {
  Serial.begin(9600);
  bme.begin();
}

void loop() {
  float temperature = bme.readTemperature(); //C
  float pressure = bme.readPressure() / 100.0F; //hPa
  float altitude = bme.readAltitude(SEALEVELPRESSURE_HPA);
  float humidity = bme.readHumidity();
  delay(1000);
}
```

1.8 GY-271

Für den GY-271 gibt es zwei unterschiedliche Bauweisen, nämlich den originalen HMC5883 und eine Nachbau, den QMC5883L. Die beiden sind von ihrer Funktionsweise gleich, nutzen aber unterschiedliche Register und Adressen und benötigen deshalb unterschiedliche Bibliotheken. Bevor es weitergeht gilt es also erstmal herauszufinden, welche Bausweise vorliegt. Hier wurde der QMC5883L verwendet.

Angeschlossen wird das Modul ebenfalls über die I2C Pins und natürlich GND und 5V.

Für die Nutzung wird wieder eine zusätzliche Bibliothek benötigt, die in dem Fall manuell als .ZIP-Datei eingebunden werden muss. Heruntergeladen werden kann die Biblitothek hier. Dort gibt es auch direkt eine Erläuterung zum Code.

Mit den Standard-Funktionen der Bibliothek wird bisher nur die Stärke aller Sensoren für der jeweiligen Vektoren in µG ausgelesen. Für die Anwendung müssen die Werte dann noch in µT konvertiert und die absolute Stärke des Magnetfeldes berechnet werden. Wichtig: Die Konvertierung muss vor der Berechnung stattfinden, da die Werte in µG zu groß sind und es ansonsten zu einem Integer-Overflow kommt.

```
//read values in microGauss
int xG, yG, zG, t;
if (compass.ready()) {
  int heading = compass.readHeading();
  int r = compass.readRaw(&xG, &yG, &zG, &t);
}
double xT = xG / 100; double yT = yG / 100; double zT = zG / 100; // ↵
   convert from microGauss to microTesla
double absoluteMagneticFieldStrength = sqrt((xT * xT) + (yT * yT) + ↵
   (zT * zT));
```

1.9 Fotowiderstand

Der Fotowiderstand dient dazu, die Helligkeit zum messen, die Benutzung ist denkbar einfach. Angeschlossen wird der erste Pin an 5V und der zweite Pin an einen analogen Pin des Arduinos. Zusätzlich muss lediglich noch ein 10kOhm Widerstand zwischen dem zweiten Pin und GND eingebaut werden eingebaut werden, siehe hierzu auch Abbildung 3 auf der nächsten Seite.

Der der Widerstand des Fotowiderstandes wird nun bei geringerem Lichteinfall geringer, sodass beispielsweise eine angeschlossene LED dadurch immer heller wird. Bei einem Widerstand von 10kOhm sollten Werte zischen ca. 880 und 1024 herauskommen.

Ausgelesen wird der Fotowiderstand wie jeder andere analoge Sensor auch:

```
int pinPhotoresistor = A1;
```

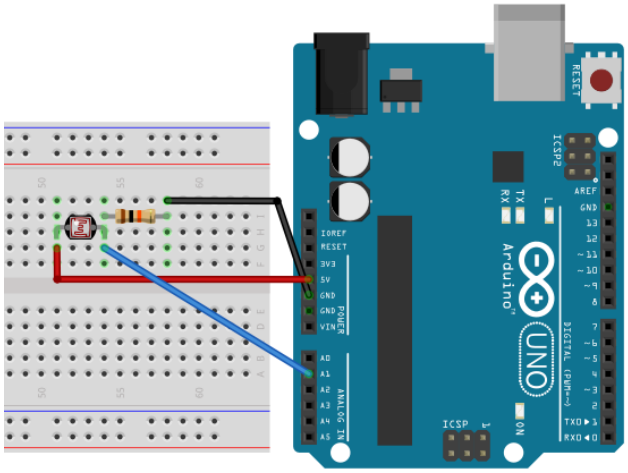

Abbildung 3: Verbindung des Fotowiderstandes

```
2
3  void setup() {
4    Serial.begin(9600);
5  }
6
7  void loop() {
8      Serial.println(analogRead(pinPhotoresistor));
9      delay(1000);
10   }
11 }
```

Listing 1: Quellcode für den Fotowiderstand

Da die ADC-Werte noch keine Aussagekraft haben, müssen diese umgerechnet werden. Die Implementierung ist von James Carlson in den Code übernommen worden und gibt nun reelle Werte in lux aus.

1.10 MQ2 Gassensor

Der MQ2 Gassensor kann LPG, i-Butan, Propan, Methan, Alkohol, Wasserstoff und Rauch in der Luft messen und kann direkt an den Arduino angeschlossen werden.

GND, 5V und der analoge Pin werden wie üblich an den Arduino angeschlossen. Ausgelesen wird der Sensor anschließend wie gewohnt mit *analogRead()*. Ausgegeben werden, wie üblich, Werte von 0-1023. Welche Werte hier für welches Gas sinnvoll sind, muss durch Ausprobieren herausgefunden werden.

Vorsicht! Der Sensor kann unter Umständen heiß werden!

2 Bluetooth

2.1 Arduino

Als Bluetooth-Modul eigenen sich entweder der HC-05 oder HC-06, wobei hier der letzterer verwendet wird. HM-10 funktioniert nicht, da dieser Sensor den BLE (Bluetooth Low Energy) Standard verwendet und sich nur indirekt mit speziellen Apps mit dem Smartphone verbinden lässt, nicht aber mit der selbsterstellten App.

Verbunden wird der Sensor mit GND, 5V, sowie der RX-Pin des Arduinos mit dem TX-Pin des HC-06 und andersherum. Da das Bluetooth Modul die serielle Schnittstelle nutzt, die auch beim Aufspielen eines Programms verwendet wird, dürfen **R**X und TX beim Aufspielen nicht verbunden sein, sonst kommt es zu einem recht unspezifischen Fehler.

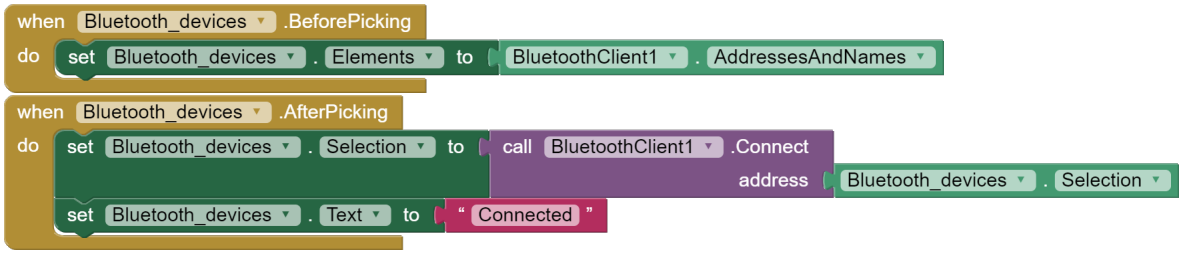

Abbildung 4: Quellcode zum Auswählen eines Bluetooth-Gerätes

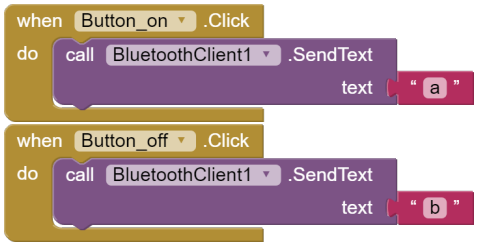

Abbildung 5: Versenden von Nachrichten per Bluetooth mit dem MIT App Inventor

Der HC-06 sollte nun als Bluetooth-Gerät auf dem Smartphone angezeigt werden. Die Standard-PIN lautet 1234. Zum Empfangen wird *Serial.read()* und zum Schreiben *Serial.print()* bzw. *Serial.println()* verwendet. Der Arduino kommuniziert mit dem Bluetooth-Modul also quasi wie mit einem normalen seriellen Monitor.

2.2 MIT App Inventor

Im MIT App Inventor muss zunächst im Designer unter *Connectivity* ein *Bluetooth-Client* eingefügt werden. Um nun ein Gerät aus der Liste der verfügbaren Geräte auswählen zu können muss jetzt nur noch der Quellcode aus Abbildung 4 eingefügt werden.

2.3 Kommunikation zwischen Arduino und der App

Mit dem Arduino als Empfänger ist der Datenaustausch, auch sehr vieler Informationen, sehr einfach. Hierzu verschickt die App einen String/Integer, für den im Quellcode des Arduinos jeweils eine Aktion hinterlegt ist.

Ein Standard-Projekt ist hier z.B. das An- und Ausschalten einer LED am Arduino. Hier werden zwei Zeichen von der App versendet (Abbildung 5), die dann wie folgt vom Arduino abgefangen werden können:

```
1  int pinLED = 13;
2
3  void setup() {
4    Serial.begin(9600);
5    pinMode(pinLED, OUTPUT);
6  }
7
8  void loop() {
9    if (Serial.available()>0)
10   {
11     char data = Serial.read();
12     if (data == 'a')
13     {
14       digitalWrite(pinLED, HIGH);
15     }
16     else if (data == 'b')
17     {
18       digitalWrite(pinLED, LOW);
```

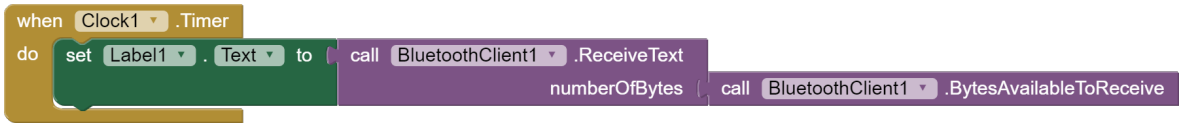

Abbildung 6: Empfang einer Information per Bluetooth mit dem MIT App Inventor

```
19          }
20      }
21  }
```

Andersherum funktioniert der Informationsaustausch für einzelne Informationen noch sehr einfach. Will man beispielsweise lediglich die Temperatur abfragen, kann mithilfe eines Timers in bestimmten Zeitabständen die gesendete Information leicht weiterverarbeitet werden (Abbildung 6).

Für mehrere Sensoren gestaltet es sich schon etwas schwieriger, die gesendeten Daten richtig zuzuordnen. Eine Möglichkeit wäre, für jeden Sensor neben den eigentlichen Daten noch Schlüsselwort mitzuschicken, wie z.B. *"tmp:20.1"*, sodass die App mit geeigneten if-Abfragen die Daten zuordnen kann.

Eine weitere, vermutlich noch geschicktere Lösung, ist das Verschicken der Daten direkt hintereinander mit *Serial.print()*. Die Werte müssen mit einem Sonderzeichen, wie einer Pipe (|), getrennt und nach dem letzten Wert ein Zeilenumbruch eingebaut werden. Einer Verkettung der Werte als String ist nicht nötig und funktioniert auch nicht besonders gut.

Die *Text*-Bausteine des App Inventors bieten nun die Möglichkeit, den empfangenen String an dem Sonderzeichen zu teilen und in die geteilten Elemente in einer Liste zu speichern zu speichern. Kennt man nun die Reihenfolge, in der die Werte geschickt wurden, kann man sehr übersichtlich damit weiterarbeiten.