ArduSmartPilot: Arduino-Programmierung

Einleitung

Diese Anleitung befasst sich mit der Programmierung des Mikrocontrollers (µC) auf der Arduinoplatine des ArduSmartPilot. Sie baut auf die Anleitung „ArduSmartPilot: Android-Programmierung auf und verwendet darin entwickelte Android Apps zum Testen der Kommunikation.

Nicht Inhalt dieser Anleitung sind die Grundlagen der Programmierung eines Arduino, sowie das Anschließen der Peripherie wie Servos, Sensoren oder Spannungsquellen. Sie baut aber auf diesen Grundlagen auf und bezieht sich ausschließlich auf die Arduino-Software des ArduSmartPilot.

Zum Erlernen der Grundlagen gibt es im Internet zahlreiche Tutorials. Oder Sie nutzen die Informationen auf den offiziellen Arduino Webseiten arduino.cc. Noch besser für den Einstieg in die Arduinowelt sind Fachbücher namhafter Verlage wie z.B. O'Reilley oder Franzis geeignet, die gerne auf dem Stand von 2010 sein dürfen. Denn die hier behandelten Programme verwenden nur Programmfunktionen, die es schon 2010 beim Arduino Duemilanove und der IDE Version 0.1xx gab.

Im Literaturverzeichnis finden Sie eine Auswahl von guten Fachbüchern, aus denen Sie die Grundlagen der Arduino-Programmierung und das Anschließen von Peripherie lernen [1], [2], [3], [4].

Diese Programmierumgebung (auf Englisch „IDE“ gleich „Integrated Development Environment“) ist sehr ähnlich zu der von Processing, obwohl es zwischen Arduino und Processing wesentliche Unterschiede gibt, von denen zwei hier genannt werden sollen:

- Bei Processing muss der kompilierte Programmcode nicht an einen anderen Rechner (hier µC) übertragen werden, da das Programm auf demselben Computer ausgeführt wird wo es auch erstellt wurde.
- Processing-Programme basieren auf der Programmiersprache Java, Arduino-Programme auf C.

Trotzdem ist die Grobstruktur eines Arduino-Programms zu der eines Processing-Programms: Bei beiden Umgebungen gibt es einen `setup()`-Teil. Die `loop()`-Schleife gibt es auch bei Processing, dort heißt sie nur `draw()`.

Hinweis: Wenn du den hier aufgeführten Code verwenden willst, dann übertrage ihn nicht via copy/paste aus diesem Dokument in den Arduino-Editor. Denn dabei werden zusätzliche Zeilenumbrüche erzeugt oder Sonderzeichen wie z.B. das Hochkomma nicht korrekt übertragen. Dies führt zu Fehlern beim Compilieren des Programmcodes, die oft nur schwer zu identifizieren sind.

Die Quellcodes aller Codebeispiele liegen daher unter ardusmartpilot.de zum Download bereit.

Inhaltsverzeichnis

Kommunikation Arduino → Computer: Daten im Serial Monitor ausgeben

Problemstellung:

Beim Processingkurs hast Du gelernt, wie man mit dem Befehl `println()` Daten in Form von ASCII-Zeichenketten in einem Terminalfenster ausgeben kann. Wie macht man so etwas beim Arduino, wenn z.B. der Wert einer Laufvariablen ausgegeben werden soll?

Der Arduino hat kein eigenes Terminalfenster (er ist ja nur ein µC) und kann deshalb die Daten nur auf seiner seriellen Schnittstelle ausgeben. Um sie anzuschauen, benötigt man ein sogenanntes Terminalprogramm auf dem Computer, mit dem der Arduino verbunden ist.
Dafür kann man jedes beliebige Terminalprogramm verwenden. Jedoch ist in der Arduino IDE ein spezielles Terminalprogramm, der „Serial Monitor“ mit dabei.

Der Serial Monitor ist ein eigenständiges Computerprogramm. Es hat die Aufgabe, die Daten von der seriellen Schnittstelle im Empfang zu nehmen und als Text in seinem (Terminal-)Fenster darzustellen.

Wir werden später sehen, dass man im Serial Monitor auch einen Text eingeben kann, der dann an die serielle Schnittstelle des Arduino geschickt wird. Er wird also dazu benutzt, um manuell mit einem Arduino in beiden Richtungen zu kommunizieren. Später wird aber unser Processing-Programm diese Kommunikation ganz automatisch und selbständig übernehmen.
Bei der Kommunikation über den Serial Monitor werden Zeichen übrigens wieder im ASCII Standard übertragen und die ankommenden Daten werden automatisch als ASCII-Zeichen interpretiert.

Beschaltung des Arduino:

Für diese Aufgabe muss der Arduino nur via USB mit dem Computer verbunden zu sein. Elektronische Bauteile müssen nicht angeschlossen werden.

Programmcode:

Bei der Arduino-Software werden Daten über den Befehl `Serial.println()` auf der seriellen Schnittstelle ausgegeben.

Nachfolgend ein Beispielprogramm hierzu: Hier wird zuerst die Zeichenkette „Sensorwert=“ und

ArduinoAnleitung_SketchSerialMonitor

```
void setup() {
  Serial.begin(9600);
  pinMode(8,INPUT);
}

void loop() {
  Serial.print("Sensorwert=");
  Serial.println(digitalRead(8));
  delay(500);
}
```

dann der Wert des logischen Signals auf Pin D8 ausgegeben.

Es ist wichtig, dass der Serial Monitor auf die selbe Baudrate eingestellt ist, wie sie auch im `setup()` Teil des Arduino-Programms festgelegt. ist.

Aufgabe:

Erstelle ein Arduino-Programm, das einen Sensorwert über den Analogeingang aufnimmt und die-

sen dann zusammen mit einem Textstring einmal pro Sekunde an die serielle Schnittstelle ausgibt. Kontrolliere mit dem Serial Monitor, ob diese Daten richtig ankommen.

Hintergrundinformationen:

Nochmal zurück zu Processing:
Processing kann Texte und Werte auch auf einer Schnittstelle ausgeben: Im Programm `TextAnArduino` hattest du die Zeichenketten „LED AN“ und „LED AUS“ an die USB Schnittstelle ausgegeben und diese dadurch an die serielle Schnittstelle des Arduino gesendet. Dafür wurde z.B. der Befehl `myPort.write(„LED AN“)` verwendet.
Bei einer Verbindung über Bluetooth (BT) hattest du den Befehl `bt.broadcast(data)` verwendet und dafür ein Byte-Array vorher mit der ASCII-Zeichenkette „LED AN“ bzw „LED AUS“ gefüllt.

Diese Befehle machen eigentlich genau das gleiche wie `Serial.println(„LED AN“)` bei einem Arduino-Programm. Im Arduino-Programm müssen wir lediglich keine Schnittstelle auswählen, da es nur die eine serielle Schnittstelle beim Arduino gibt.

Die Ausgabedaten des oben erstellten Programms wurden vom Serial Monitor empfangen und für dich sichtbar gemacht. Diese Daten können aber auch von einem anderen Computerprogramm entgegengenommen werden: Zum Beispiel von einem Processing-Programm!
Genau so ein Programm hast du im Processingkurs schon programmiert:
Schaue dir dazu nochmals die letzten drei Codezeilen des Programms `TextEchoArduino` an.
Hier liest der Befehl `myPort.readStringUntil(linefeed)` die serielle Schnittstelle aus und

TextEchoArduino

```
  }
  // Die verfügbaren ASCII-Zeichen aus der seriellen Schnittstelle
  // auslesen bis zum Zeichen "linefeed" (ASCII-Zeichen 10)
  echoArduino = myPort.readStringUntil(linefeed);
  // Zeichenfarbe auf Schwarz stellen
  fill(0, 0, 0);
  // Die oben rückgelesene Nachricht vom Arduino im Text an der
  // xy-Position (50,100) darstellen.
  text("Arduino: " + echoArduino, 50, 100);
}
```

der darauf folgende Befehl stellt den empfangenen Text im Grafikfenster dar.

Der Arduino beim Processingkurs war so programmiert, dass er z.B. den Empfang der Zeichenkette „LED AN“ mit dem Befehl `Serial.println(„LED ist an!“)` quittierte. Jetzt weißt du, woher diese Rückmeldung kam.

Kommunikation Computer→ Arduino: Codewörter übertragen.

Die Kommunikation des Arduino über die serielle Schnittstelle hin zum Serial Monitor haben wir nun im Griff. Im Processingkurs hast du auch gelernt, wie stattdessen ein Computerprogramm die Daten vom Arduino entgegennehmen kann.

Problemstellung:

Es bleibt noch die Kommunikation in die Gegenrichtung, also vom Computer an den Arduino. Dabei ist es letztendlich egal, ob die Daten manuell in den Serial Monitor eingetippt werden, oder ob sie aus einem Processing-Programm kommen, wie z.B. schon praktiziert im Programm `TextAnArduino` aus dem Processingkurs.

Nachfolgende wollen wir zuerst manuell mit dem Arduino kommunizieren. Wenn wir den Befehl „LED AN“ über den Serial Monitor an den Arduino übertragen, dann soll die LED 13 am Arduino aufleuchten. Senden wir den Befehl „LED AUS“, dann soll diese LED ausgehen. Das fertige Programm hat dann die gleichen Funktionen wie jenes „geheimnisvolle“ Arduino-Programm, dass da-

mals im Processingkurs zusammen mit dem Programm `TextAnArduino` verwendet wurde.

Beschaltung des Arduino:

Für diese Aufgabe muss der Arduino nur via USB mit dem Computer verbunden zu sein. Elektronische Bauteile müssen nicht angeschlossen werden.

Programmcode:

KomAnArduino

```
char einZeichen = 0; // ankommendes Byte
String zeichenInput = ""; // Puffer für ankommende Bytefolgen
String zeichenLEDan = "LED AN"; // String mit dem "Codewort";
String zeichenLEDaus = "LED AUS"; // String mit dem "Codewort";

void setup()
{
  pinMode(13, OUTPUT); // Pin 13 mit der LED als Ausgang kofigurieren.
  Serial.begin(9600); // Serielle Schnittstelle auf 9600 Baud einstellen.
}

void loop()
{
  // Solange etwas an der seriellen Schnittstelle anliegt...
  while (Serial.available() > 0)
  {
    einZeichen = Serial.read(); // ... jeweils ein einzelnes Byte lesen.
    if (einZeichen == '\n') //Wenn ein "Neue Zeile"-Zeichen kommt...
    {
      if(zeichenInput.equals(zeichenLEDan))  // Bei Übereinstimmung LED einschalten.
      {
        digitalWrite(13, HIGH);
      }
      if(zeichenInput.equals(zeichenLEDaus)) // Bei Übereinstimmung LED ausschalten.
      {
        digitalWrite(13, LOW);
      }
      zeichenInput = ""; // Puffer wieder leer machen
    }
    else
    {
      //Wenn das Endzeichen noch nicht da, neues Zeichen an die vorhandenen anhängen.
      zeichenInput += einZeichen;
    }
  }
}
```

`Serial.read()` ist sozusagen das Pendant zum Befehl `Serial.print()`, den du im vorherigen Kapitel schon verwendet hast. Es werden jetzt nur Daten gelesen statt geschrieben.
Leider gibt es aber noch einen kleinen Unterschied, der den Programmieraufwand im jetzt zu erstellenden Programm `KomAnArduino` etwas erhöht:
`Serial.read()` kann leider keine kompletten Zeichenketten, sondern immer nur einzelne Zeichen von der Schnittstelle auslesen. In diesem Fall sind es einzelne Bytes. Als Konsequenz muss das Arduino-Programm dann selbst aus den einzelnen Zeichen die empfangene Zeichenkette wieder Stück für Stück zusammensetzen. Stelle dir das so vor, als würde jemand eine gedruckte Textzeile Buchstabe für Buchstabe abschneiden und dir jeweils immer nur einen Buchstaben geben.

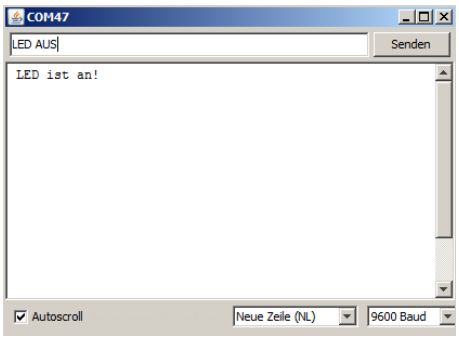
Aber woher weiß das Programm, welches Zeichen das letzte einer empfangenen Zeichenkette ist? Ganz einfach: Beide Kommunikationspartner einigen sich darauf, dass nach dem letzten Zeichen immer ein sogenannter „Zeilenvorschub“, also das Zeichen „\n“ (Hexadezimal 0A) gesendet wird. Das ist in etwa so, wie wenn du bei dem Beispiel vorhin einen Schnipsel mit einem Leerzeichen erhältst.

Das Arduino-Programm arbeitet nur mit den beiden Codes „LED AN“ und „LED AUS“: Wenn eine Zeichenkette komplett empfangen wurde, dann wird diese mit den beiden Codes verglichen. Stimmt sie überein, dann wird die LED entsprechend geschaltet.

Hier sind die folgende Befehle von Bedeutung:

Befehl	Beschreibung
`Serial.available();`	Funktion, die die Anzahl der Zeichen zurückgibt, die an der seriellen Schnittstelle warten um vom Arduino ausgelesen zu werden.
`Serial.read();`	Lesen eines einzelnen Zeichens und zwar das, das an der seriellen Schnittstelle als nächstes anliegt.
`zeichenInput += einByte;`	Die Variable `zeichenInput` ist eine Stringvariable, also eine Zeichenkette. Hiermit wird bei dieser Zeichenkette das neu gelesene Zeichen `einByte` ans Ende angehängt.

Aufgaben:



1. Erstelle ein neues Arduino-Programm und nenne es `KomAnArduino`. Gebe dann obigen Quellcode ein. (Wie beim Processingkurs ist der graue Text nur der Kommentar. Diesen Text brauchst du natürlich nicht mit abzutippen.)
   Dann starte den Serial Monitor[1], sende die Codes an den Arduino und beobachte, ob und wie die LED reagiert. Beim Serial Monitor ist wichtig, dass neben dem richtigen Baudwert von 9600 noch „Neue Zeile (NL)“ gewählt ist. Letztere Option bewirkt, dass nach senden des Codes automatisch ein Zeilenvorschub gesendet wird, so dass der Arduino das Ende der Zeichenkette erkennt.
2. Speichere das Programm unter dem Namen `ArduinoKom` ab und erweitere es wie folgt: Wenn die LED eingeschaltet wurde, soll der Arduino die Zeichenkette „LED ist an!“ und wenn sie ausgeschaltet wurde „LED ist aus!“ an die serielle Schnittstelle zurückmelden. Verwende dafür den Befehl `Serial.println()` wie vorhin bei der Ausgabe der Sensorwerte.
   Verwende wieder den Serial Monitor, um Codes an den Arduino zu senden. Achte dieses Mal darauf, was der Arduino antwortet.
3. Erfinde einen dritten Code, der z.B. „ArduinoKomBlink“ heißen kann. Ändere das Programm `ArduinoKom` entsprechend ab, so dass die LED bei dem neuen Codewort zu blinken anfängt.

Hintergrundinformationen:

Übrigens: Jetzt hast du genau das Arduino-Programm fertig erstellt, das wir im Processingkurs als „Black Box“ verwendet hatten.

1 Der Serial Monitor ist nichts anderes als ein Terminalprogramm. Daher kannst du auch ein beliebiges anderes Terminalprogramm für die Kommunikation wie z.B. „CoolTerm“ verwenden. Dort musst du dann neben der richtigen Baudrate „ Data Bits: 8“, „Parity: none“ und „Stop Bits: 1“ einstellen. Außerdem muss nach dem Senden eines Strings das „\n“ als Abschlusszeichen angehängt werden („Termination String (Hex): 0A“).

Dieses Programm funktioniert übrigens unverändert für die BT-Kommunikation! Denn wie du weißt, kennt der Arduino nur seine Steckdose „serielle Schnittstelle“. Er merkt nichts davon, wenn die Daten danach durch das BT-Shield in ein BT-Signal umgewandelt werden.
Nachfolgend sind die Codezeilen von `ArduinoKom` wiedergegeben, in denen der Arduino sich über die serielle Schnittstelle zurückmeldet:

```
if(zeichenInput.equals(zeichenLEDan))  // Bei Übereinstimmung LED einschalten.
{
  digitalWrite(13, HIGH);
  Serial.println(" LED ist an!"); // LED Status über Schnittstelle zurück melden.
}
if(zeichenInput.equals(zeichenLEDaus)) // Bei Übereinstimmung LED ausschalten.
{
  digitalWrite(13, LOW);
  Serial.println(" LED ist aus!"); // LED Status über Schnittstelle zurück melden.
}
```

Kommunikation Computer→ Arduino: Zahlenwerte übertragen.

Problemstellung:

Wie man mit Codewörtern einen Arduino steuert, haben wir im letzten Kapitel gelernt: Ein Code wird als Zeichenkette über die serielle Schnittstelle an den Arduino geschickt. Dieser liest ein Zeichen nach dem anderen aus und vergleicht die so entstandene Zeichenkette mit den bekannten Codes. Dann schaltet er die LED entweder ein oder aus. Oder er macht gar nichts, wenn er einen unbekannten Code erhält.

Später soll der Arduino den ArduSmartPilot steuern. Dafür benötigt er Informationen, die sich nicht mehr einfach durch wenige Codes ausdrücken lassen: Das Androidgerät soll ihm später nicht nur die Anweisung „SCHNELL“ oder „LANGSAM“ senden, sondern es soll z.B. die Motorleistung von 0 bis 100 % der Maximalleistung in 5 Stufen übertragen werden. Gleiches gilt für die Position von Seiten- und Höhenruder: Auch hier reicht ein einfaches „RECHTS“ oder „HOCH“ als Code nicht aus.

Aus diesem Grund müssen wir unser Arduino-Programm `ArduinoKom` erweitern, so dass der Arduino nicht nur vorher festgelegte Codes sondern Zahlenwerte verstehen kann.

Dabei reicht das Senden einer Zahl nicht aus. Denn das Flugzeug benötigt ständig Informationen über die Stellung des Höhenruders (Zahl 1) und die Stellung des Seitenruders (Zahl 2) und über die Motorleistung (Zahl 1),.
Diese drei Zahlen müssen wir über ein sogenanntes „Datenprotokoll“ an die serielle Schnittstelle senden. Und der Arduino muss aus diesem Protokoll die drei getrennten Werte wieder extrahieren. Die Trennung wird über ein Kommazeichen zwischen den Werten möglich. Den letzten Wert erkennt man daran, dass ein Zeilenvorschub, also das Zeichen „\n“ folgt.
Sowohl die Kommazeichen als auch der Zeilenvorschub sind also Teil des Datenprotokolls.

Beschaltung des Arduino:

Für diese Aufgabe muss der Arduino via USB mit dem Computer verbunden sein. Zusätzlich muss eine (mit Vorwiderstand versehene) LED zwischen Digitalpin 12 und GND geschaltet werden.

Programmcode:

1. Das erste Programm (`ArduSteuerEinfach`) erwartet einen Integerwert über die serielle Schnittstelle.
   Wurde (z.B. über den seriellen Monitor) eine Zahl gesendet, so blinkt die LED 13 den Wert der Zahl aus. Wenn andere Zeichen als eine Zahl gesendet wurden, dann soll der Arduino sich mit „Verstehe ich nicht…“ über die serielle Schnittstelle zurückmelden.

ArduSteuerEinfach

```
int wert; //Variable für Integerwert

void setup() {
  Serial.begin(9600);
  pinMode(13, OUTPUT);
  Serial.println("Einen Integerwert eingeben:");
  delay(1000);
}

void loop() {
  if( Serial.available())
  {
    // ASCII Zeichen einzeln von der Schnittstelle lesen.
    char ch = Serial.read();
    // Falls die ASCII Zeichen Ziffern sind
    if(ch >= '0' && ch <= '9')
    {
      //ASCII Wert in Ziffer konvertieren und aus Ziffern Dezimalzahl machen.
      wert = (wert * 10) + (ch - '0');
    }

    else if (ch =='\n') //Wenn Ende empfangene Zeichenkette
    {
      for(int i=1; i <= wert; i++)
      {
        digitalWrite(13,HIGH);
        delay(200);
        digitalWrite(13,LOW);
        delay(200);
      }
      wert = 0; //Wert zurücksetzen
    }
    else
    {
      Serial.println("Verstehe ich nicht ...");
      wert = 0; //Wert zurücksetzen
    }
  }
}
```

2. Beim zweiten Programm (`ArduSteuerZwei`) erwartet der Arduino zwei Integerwerte mit einem Komma getrennt. Hat er ein solches Wertepaar empfangen, dann gibt er den ersten Wert auf der der an Digitalpin 12 angeschlossenen LED und den zweiten Wert auf der LED 13 aus.
   Auch hier soll der Arduino wieder ein „Verstehe ich nicht…“ zurückgeben, wenn er etwas anderes als Zahlen empfängt.
   Eine Integervariable reicht hier nicht mehr aus, da ja jetzt mehrere Werte im Datenprotokoll enthalten sind. Daher ist hier die Variable `werte` ein Integer-Array. Mit jedem Arrayelement passiert eigentlich genau das gleiche wie mit der entsprechenden Integervariablen im Programm `ArduSteuerEinfach`.
   Neu ist hier auch noch die Abfrage, ob ein Komma empfangen wurde. Ist das der Fall, dann wird einfach zum nächsten Wert im Datenprotokoll bzw. zum nächsten Arrayelement übergegangen. Denn das Komma wird im Datenprotokoll dazu verwendet um die zwei Werte zu trennen.
   Vielleicht fragst du dich, wieso man hier überhaupt mit Arrays arbeitet, zwei Integervariablen hätten es doch auch getan! Das stimmt. Aber dieser Quellcode kann mit wenigen Änderungen auf ein Protokoll mit drei, vier oder sogar hundert Einzelwerten erweitert werden. (Programmierer hassen am meisten Chefs, die „schnell mal“ von ihnen Änderungen am

Programmcode verlangen – deshalb sorgen sie vor…)

ArduSteuerZwei

```
int werte[2]; //Array für die beiden Integerwerte
int arrayIndex = 0; //Arrayindex initialisieren.

void setup() {
  Serial.begin(9600);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  Serial.println("Zwei Integerwerte getrennt mit Komma eingeben:");
  delay(1000);
}
void loop() {
  if( Serial.available())
  {
    char ch = Serial.read();
    if(ch >= '0' && ch <= '9')
    {
      werte[arrayIndex] = (werte[arrayIndex] * 10) + (ch - '0');
    }
    else if (ch == ',') // Wenn Komma, dann zum nächsten Arrayelement wechseln.
    {
      if(arrayIndex == 0)
        arrayIndex++;
    }
    else if (ch =='\n' && arrayIndex == 1) // Wenn letztes Zeichen Protokoll empfangen und zwei Werte da.
    {
      for(int i=1; i <= werte[0]; i++)
      {
        digitalWrite(12,HIGH);
        delay(200);
        digitalWrite(12,LOW);
        delay(200);
      }
      for(int i=1; i <= werte[1]; i++)
      {
        digitalWrite(13,HIGH);
        delay(200);
        digitalWrite(13,LOW);
        delay(200);
      }
      for(int i=0; i <= 1; i++) werte[i] = 0; // Wertearray zurücksetzen.
      arrayIndex = 0;
    }
    else
    {
      Serial.println("Verstehe ich nicht ...");
      for(int i=0; i <= 1; i++) werte[i] = 0; // Wertearray zurücksetzen.
      arrayIndex = 0;
    }
  }
}
```

Aufgabe:

1. Erstelle ein neues Programm und nenne es `ArduSteuerEinfach`. Gebe den entsprechenden Quellcode ein. Übertrage das Programm, sende dem Arduino über den Serial Monitor eine Zahl und beobachte die LED. Was passiert, wenn du ein Wort statt einer Zahl sendest?
   Wieso meldet sich dann der Arduino für jeden einzelnen Buchstaben mit „Verstehe ich nicht…“ zurück?
2. Erstelle ein neues Programm und nenne es `ArduSteuerZwei`. Gebe den oben dargestellten Quellcode ein. Übertrage das Programm, sende dem Arduino über den Serial Monitor eine Zahl und beobachte nun beide LEDs.
3. Später wird im ArduSmartPilot der dritte Eingabewert die Motorleistung sein. Was muss das Programm `ArduSteuerEinfach` noch nachprüfen, damit nur sinnvolle Werte (0 bis 100 %) an die Motorsteuerung weitergegeben werden? Erweitere das Programm dahingehend.

Hintergrundinformationen:

Mit folgender Codezeile wird im Programm `ArduSteuerEinfach` überprüft, ob die vom Arduino gelesenen Zeichen auch wirklich Ziffern sind:

```
if(ch >= '0' && ch <= '9')
```

Falls die Variable `ch` eine Ziffer als Zeichen beinhaltet, dann wird die Ziffer mit folgender Codezeile in eine Integerzahl umgewandelt:

```
(ch - '0')
```

Die beiden Operationen sind möglich, da es sich um ASCII Zeichen handelt. Jedes ASCII Zeichen hat einen Dezimalwert (siehe z.B. Wikipedia): Die Ziffer „0“ hat den Dezimalwert 48 und die Ziffer „9“ den Dezimalwert 57. Die Ziffern „2“ bis „8“ haben entsprechend die Dezimalwerte dazwischen. Die Operationen in den beiden Codezeilen oben beziehen sich auf diese Dezimalwerte.

Ansteuerung von Servos

Problemstellung:

Modellbauservos werden über sogenannte Pulsweitensignale (PWM) angesteuert. Mit Hilfe der Arduino-Bibliothek „Servo“ kann man einem angeschlossenen Servo direkt die gewünschten Winkelstellung in Grad übermitteln.
Näheres hierzu findest Du in den genannten Fachbüchern bzw. Internetseiten.

Aufgaben:

1. Teste den Servo mit einem Programm, das ihn nacheinander auf die verschiedene Winkel 0°, 10°,... 180° einstellt.
2. Nutze das Programm `ArduSteuerEinfach` als Grundlage und benenne es in `ArduSteuerServo` um.
   Das neue Programm soll vom Serial Monitor eine Winkelstellung in Grad erhalten und diese am Servo einstellen. Ändere das Programm also so ab, dass nicht mehr die LED den Zahlenwert ausblinkt, sondern dieser an den Servo steuert.

Hierbei musst du entweder einen kleinen Servo verwenden, oder den Arduino (bzw. den Servo) mit einem zusätzlichen Netzteil versorgen. Denn Servos benötigen beim Stellen kurzzeitig recht hohe Ströme. Dadurch kann die Spannung auf dem Arduino kurz zusammenbrechen wodurch meist die serielle Kommunikation zusammenbricht. Wenn kein Netzteil zur Hand ist, dann sollte man den Servo zumindest an V_{in} statt an 5 anschließen: Dann erhält er seinen Strom direkt vom USB-Anschluss des Computers. Dieser Anschluss kann höhere Ströme liefern als der Arduino.

Teste dann das Programm über den Serial Monitor indem du dort verschiedene Stellwinkel eingibst.

Programmcode:

ArduSteuerServo

```
void loop() {
  if( Serial.available())
  {
    // ASCII Zeichen einzeln von der Schnittstelle lesen.
    char ch = Serial.read();
    // Falls die ASCII Zeichen Ziffern sind
    if(ch >= '0' && ch <= '9')
    {
      //ASCII Wert in Ziffer konvertieren und aus Ziffern Dezimalzahl machen.
      wert = (wert * 10) + (ch - '0');
    }

    else if (ch =='\n') //Wenn Ende empfangene Zeichenkette
    {
      mServo.write(wert);
      wert = 0; //Wert zurücksetzen
    }
    else
    {
      Serial.println("Verstehe ich nicht ...");
      wert = 0; //Wert zurücksetzen
    }
  }
}
```

Hintergrundinformationen:

Bei dem ArduSmartPilot werden später nicht nur die beiden Servos vom Höhen- und Seitenruder über ein PWM-Signal gestellt sondern auch die Motorleistung. Dafür gibt es spezielle Motorregler, die „ESC“ (Electronic Speed Controller) genannt werden.

Rückmeldung, Auslesen und Übertragen der Sensorwerte über USB

Problemstellung:

Eigentlich können wir ja jetzt loslegen mit dem Fliegen: Du weißt, wie man Zahlenwerte in Form eines Datenprotokolls an den Arduino überträgt. Das Programm `ArduSteuerServo` muss nur noch mit Hilfe des Programms `ArduSteuerZwei` auf drei Werte (Höhenruder, Seitenruder und Motorleistung) erweitert werden, und schon können wir starten…

Ganz so einfach ist es leider nicht, denn wir haben (mindestens) noch ein wichtigen Aspekte ganz vergessen:

Ähnlich wie wenn deine Mutter dich zum Abendessen ruft, ist bei der Kommunikation mit dem Arduino die Kontrolle wichtig, ob er überhaupt die gesendeten Daten richtig und vollständig empfangen hat. Beim Programm `ArduinoKom`, bei dem nur Codes an den Arduino gesendet werden, hast du diese Rückmeldung sogar schon programmiert: Der Arduino meldet bei jedem Kommunikationsvorgang zurück, ob die LED an oder aus ist.
Da der Arduino sowieso während des Flugs Sensorwerte übermitteln soll, nehmen wir am besten diese Kommunikation als Rückmeldung. D.h. jedes Mal, wenn er neue Werte für Motor und Ruderstellungen erhält, dann sendet der Arduino ein eigenes Datenprotokoll zurück. Darin sind die aktuellen Sensorwerte enthalten und eine Rückmeldung, ob die Steuerwerte richtig empfangen wurden.

Beschaltung des Arduino:

An den Analogeingang A0 wird ein Helligkeitssensor oder irgend ein anderer analoger Sensor angeschlossen. Der Servo erhält sein PWM-Signal über den Digialpin D3. Der Arduino kommuniziert über USB mit dem Computer.

Aufgaben:

Verwende das Programm `ArduSteuerZwei` als Ausgangspunkt, speichere es unter dem Namen `ArduSteuerEcho` ab und erweitere es auf ein Protokoll mit drei Integerwerten. Diese Werte werden jetzt nicht mehr dazu verwendet, um LEDs blinken zu lassen. Das Programm soll der Einfachheit halber nur den ersten Wert verwenden, um einen Servo am Digitalpin 3 anzusteuern. Die anderen Werte werden einfach ignoriert.

Das Datenprotokoll des Arduino soll wie folgt aussehen:
Es besteht aus zwei Werten, die durch ein Komma getrennt sind.
Der erste Wert ist ein Statuswert der Kommunikation. Er soll angeben, wie viele Kommata pro Nachricht empfangen wurden. Falls einer der Werte keine Zahl war, wird der Statuswert auf 99 gesetzt.
Ein Statuswert von 2 bedeutet also fehlerfreie Kommunikation. Eine andere Zahl verrät, dass eine falsche Zahl von Werten oder unsinniger Inhalt empfangen wurde.
Der zweite Wert gibt den Digitalwert des Helligkeitssensors wieder.

Die beiden Werte sollen vom Arduino mit einem Komma getrennt über die serielle Schnittstelle gesendet werden. Danach soll ein Zeilenvorschub als Abschlusszeichen folgen.

Nachfolgend ist ein Codeausschnitt dargestellt, in dem diese beiden Werte mit einem Komma getrennt an die serielle Schnittstelle ausgegeben werden. Der letzte Befehl muss ein `Serial.println()` sein, damit der Zeilenvorschub als Abschlusszeichen angehängt wird.

```
Serial.print(kommaZahl);
Serial.print(",");
Serial.println(sensorWert);
```

Nachfolgend ist ein Codeausschnitt dargestellt, in dem empfangenen Kommata gezählt werden.

```
else if (ch == ',') // Wenn Komma, dann zum nächsten Arrayelement wechseln.
{
  arrayIndex = min(arrayIndex++,2); // Den Index auf maximal 2 begrenzen
  kommaZahl++;
}
```

Wenn du nicht zurechtkommst, dann schaue in dem nachfolgendem Codebeispiel (Programm `ArduSteuerEcho`) nach.

Teste das neue Programm über den Serial Monitor indem du jeweils drei mit Kommas getrennte Zahlenwerte (also sogenannte „Zahlentripel“) überträgst.

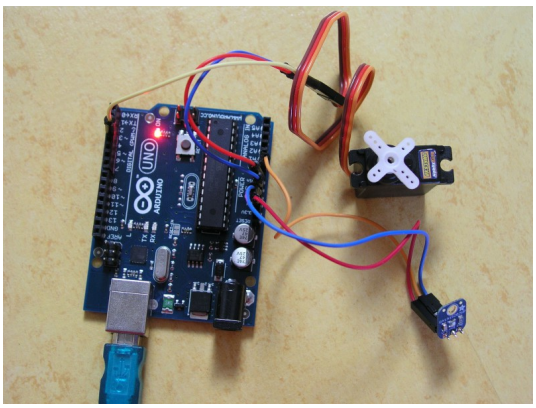


Abbildung 1: Steuerung über USB-Schnittstelle und Serial Monitor.

ArduSteuerEcho

```
#include <Servo.h>

Servo mServo;
int werte[3]; //Array für die beiden Integerwerte
int arrayIndex = 0; //Arrayindex initialisieren.
int kommaZahl = 0; //Anzal der empfangenen Werte
int sensorWert = 0; //Digitalwert des Helligkeitssensors

void setup() {
  Serial.begin(9600);
  mServo.attach(3);
  Serial.println("Drei Integerwerte getrennt mit Komma eingeben:");
  delay(1000);
}

void loop() {
  sensorWert = analogRead(A0);
  if( Serial.available())
  {
    char ch = Serial.read();
    // Falls die ASCII Zeichen Ziffern sind.
    if(ch >= '0' && ch <= '9')
    {
      //ASCII Wert in Ziffer konvertieren und aus Ziffern Dezimalzahl machen.
      werte[arrayIndex] = (werte[arrayIndex] * 10) + (ch - '0');
    }
    else if (ch == ',') // Wenn Komma, dann zum nächsten Arrayelement wechseln.
    {
      arrayIndex = min(arrayIndex++,2); // Den Index auf maximal 2 begrenzen
      kommaZahl++;
    }
    else if (ch =='\n') // Wenn letztes Zeichen Protokoll empfangen und zwei Werte da.
    {
      mServo.write(werte[0]);
      Serial.print(kommaZahl);
      Serial.print(",");
      Serial.println(sensorWert);
      for(int i=0; i <= 2; i++) werte[i] = 0; // Wertearray zurücksetzen.
      arrayIndex = 0;
      kommaZahl = 0;
    }
    else
    {
      kommaZahl = 99;
    }
  }
}
```

Hintergrundinformationen:

In der oben dargestellten Version des Programms `ArduSteuerEcho` sendet der Arduino bei jedem empfangenem Datensatz den Sensorwert zurück.

Für die Steuerungs-App hat sich gezeigt, dass 20 Steuerbefehle pro Sekunde für ein flüssiges Steuern des ArduSmartPilot ausreichend sind. Dies wird im Processing-Programm im `setup()`

Teil über den Befehl `frameRate(20)` eingestellt. Genau so oft sendet der Arduino den Sensorwert zurück. Dies führt dazu, dass später das BT-Modul 40 mal pro Sekunde zwischen Senden und Empfangen umschalten muss.

Es gibt BT-Module die damit überfordert sind. Außerdem reicht es, die Sensordaten nur zweimal pro Sekunde zu aktualisieren. Denn der Luftdruck oder die Lage des Flugzeugs ändert sich nicht signifikant innerhalb dieser Zeitspanne. Daher wird in der letzten Version des Arduino-Programms dieser Anleitung der Sensorwert nur dann zurück gesendet, nachdem 10 Steuerbefehle empfangen wurden – also nur alle 500 ms.

Um einen ausreichend hohen Datenfluss zu gewährleisten wird in der finalen Programmversion die Baudrate auf 57600 Baud erhöht[2].

Rückmeldung, Auslesen und Übertragen der Sensorwerte über BT

Problemstellung:

Den Serial Monitor hatten wir nur zum manuellen Testen unseres Arduino-Programms verwendet. Jetzt wo, das Programm `ArduSteuerEcho` funktioniert, musst du es auch mit einem Androidgerät testen.
Hierbei musst du keine Zahlentripel mehr eintippen. Vielmehr ist jeder Taste ein Zahlentripel zugeordnet, welches beim Drücken der Taste an den Arduino gesendet wird.

Aufgaben:

Verwende das fertige Processing-Programm `ArduSmartPilot`. Öffne das Programm über die Processing IDE und trage dort die richtige Adresse deines BT-Shields ein.

Dann setze das BT-Shield auf den Arduino und schließe oben auf dem Shield den Helligkeitssensor an (der Pin A0 sowie 5 V und GND findest du auch auf dem BT-Shield, denn die entsprechenden Leitungen des Arduino darunter werden über die Steckkontakte einfach nach oben verlängert). Entsprechend kannst du auch den Servo am Shield (Pin D3) anschließen.

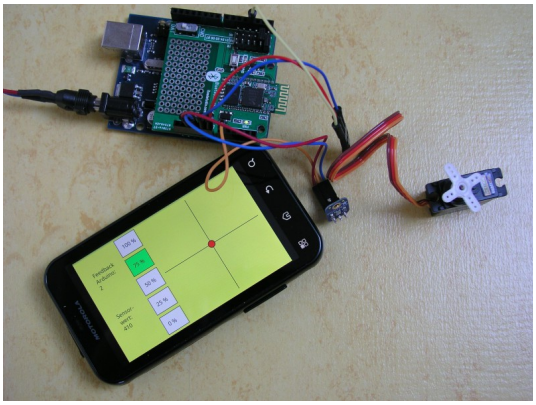


Abbildung 2: Steuerung über BT-Shield und Androidgerät.

Starte das Processing-Programm in Android-Modus und teste ob es funktioniert. (Es hat sicher noch einige Bugs, die du zu gegebener Zeit noch beheben musst: Es kann sein, dass dieses App nur funktioniert, wenn dein Androidgerät beim Start quer ausgerichtet ist. Eventuell sind beim Fadenkreuz auf der rechten Seite auch x- und y-Achse vertauscht oder stehen auf dem Kopf.)
Der erste Wert im Datenprotokoll des Processingprogramms ist die Motorleistung. Diese stellst du über die fünf Tasten am linken Rand auf die angezeigten Werte. Die Servostellung (in Grad) soll also die eingestellte Motorleistung anzeigen. Der rote Punkt vor dem Fadenkreuz soll die Neigung des Androidgeräts anzeigen. Später werden darüber das Höhen- und Seitenruder gesteuert.

2 Bei dieser Änderung bitte unbedingt darauf achten, dass das BT-Modul ebenfalls auf diese neue Baudrate konfiguriert wird. Achtung: Die nächsthöhere Baudrate von 115200 Baud läuft beim Arduino Pro Mini 3,3 V 8 MHz nicht fehlerfrei!

Kontrolle der BT-Funkverbindungsqualität

Problemstellung:

Wenn der Arduino als Statuswert für die Kommunikation keine 2 zurückgibt, ist es für das Flugzeug schon zu spät ;-)

Es ist besser frühzeitig zu erkennen, wenn die Kommunikation schlechter wird.
Das ist genauso wie wenn dein Vater dich zum Essen ruft und du bist weit weg oder hörst laute Musik: Er muss halt öfters rufen, bist du verstanden hast, was Sache ist. Dein „Statuswert für die Kommunikation“ in Form eines „Ja ich komme!“ kommt also verzögert.

Genauso kann mein bei der Kommunikation zum Arduino messen, wie lange es dauert, bis der Arduino sich rückgemeldet hat.
Schickt man ihm die drei Steuerdaten über den Serial Monitor, so ist diese Messung schlecht mit einer Stoppuhr machbar, da es sich ja um Millisekunden handelt. Kommuniziert aber ein Processing-Programm mit dem Arduino, dann kann dieses Programm die Zeitmessung auf einfache Weise vornehmen. Denn die Prozessoren im Androidgerät haben Taktzyklen im Bereich von Nanosekunden.

Auf diese Weise könnte also das Processing-Programm den Kommunikationsabbruch frühzeitig erkennen. Aber was nützt dir das, wenn Du weißt, dass dein Flugzeug gleich keinen Funkkontakt mehr hat? Schaffst du es nicht mehr, das Flugzeug zurück zu steuern, dann fliegt es trotzdem weg.

Viel besser ist doch folgendes Vorgehen:
Wenn der Arduino eine schlechter werdende Funkverbindung erkennt und dann soll er vorsorglich den Motor ausschalten und eine leichte Kurve fliegen (also Seitenruder leicht einschlagen, Höhenruder in neutraler Stellung). Dadurch kann das Flugzeug zumindest nicht wegfliegen. Wenn die Funkverbindung wieder stabil ist, dann wird der Motor wieder eingeschaltet und die Ruder können wieder normal gesteuert werden.

Aufgaben:

Das Processing-Programm sendet dem Arduino 20x pro Sekunde aktuelle Steuerdaten, egal ob der Arduino diese empfängt oder nicht. Der Arduino misst die Zeit zwischen 11 empfangenen Steuerdaten. Bei einer ungestörten Kommunikation liegen zwischen dem 1. und 11. Steuersignal genau 500 ms. Das Arduino-Programm muss also die Länge dieses Zeitraums kontrollieren: Ist er länger als 2000 ms, dann soll der Motor abgeschaltet werden.

Wie in den Hintergrundinformationen von Kapitel „Rückmeldung, Auslesen und Übertragen der Sensorwerte über USB“ erwähnt, soll der Sensorwert ebenfalls nur dann zurück gesendet werden, wenn 10 Steuerbefehle empfangen wurden.

Führe in das Programm `ArduSteuerEcho` einen Zähler ein, mit dem jeder 10. Steuerbefehl erkannt wird. Nutze die Funktion `millis()` um die Zeiten zu messen. Verwende eine If-Bedingung und eine boolsche Variable um den Motor auszuschalten, wenn der Zeitraum größer als 2000 ms wird. Sende statt des Sensorwertes die gemessene Zeit zurück.
Speichere das Programm unter dem Namen `Motorflug` ab.

In diesem Programmcode werden nun auch beide Servos und der Motorregler angesteuert. Über den Befehl `map()` werden die Stellwerte des Processing-Programms in sinnvolle Servostellwinkel oder Motorreglerwerte umgerechnet. Denn je nach Flugzeugmodell kann der Bereich von 0 bis 180° vielleicht gar nicht vom Servo gestellt werden.

Den Motorregler muss man üblicherweise einlernen. Für den obigen Programmcode wurde der Regler so parametriert, dass 20° „Motor aus“ und 160° „volle Motorleistung“ bedeutet, wobei das Processing-Programm die Motorleistung in Prozent überträgt.

Die boolsche Variable `signalFehlt` nimmt den Wert Wahr an, wenn für den Empfang von 10 Steuerbefehlen mehr als 2000 ms gemessen wurden.

Programmcode: motorflug

```
#include <Servo.h>

Servo seitenRuder, hoehenRuder, motorRegler;   // Variablen für die Servos (PWM-Signale).
int posSeiten = 0;                 // Initialwerte.
int posHoehen = 0;
int leistMotor = 0;
int werte[3]; //Array für die beiden Integerwerte
int arrayIndex = 0; //Arrayindex initialisieren.
int kommaZahl = 0; //Anzal der empfangenen Werte
int sensorWert = 0; //Wert Sensorsignal
unsigned long zeitSteuerAlt = 0;
unsigned long zeitSteuerNeu = 0;
int zaehlSensor = 0;
boolean signalFehlt = false; //BT-Verbindung abgebrochen?

void setup() {
  Serial.begin(57600);
  seitenRuder.attach(9);  // PWM-Signal für Servo an Ausgang D9.
  hoehenRuder.attach(10); // PWM-Signal für Servo an Ausgang D10.
  motorRegler.attach(11); // PWM-Signal für Motorregler an Ausgang D11.
  seitenRuder.write(90);  // Servo zu Beginn auf 90 Grad stellen.
  hoehenRuder.write(90);
  motorRegler.write(0); // Motor ausschalten
  delay(500);
  Serial.println("OK");  // Rückmeldung, dass Setup ausgeführt ist.
}

void loop() {
  sensorWert = analogRead(A0);
  if( Serial.available())
  {
    char ch = Serial.read();
    if(ch >= '0' && ch <= '9')
    {
      werte[arrayIndex] = (werte[arrayIndex] * 10) + (ch - '0');
    }
    else if (ch == ',') // Wenn Komma, dann zum nächsten Arrayelement wechseln.
    {
      arrayIndex = min(arrayIndex++,2); // Den Index auf maximal 2 begrenzen
      kommaZahl++;
    }
    else if (ch =='\n') // Wenn letztes Zeichen Protokoll empfangen und zwei Werte da.
    {
      // Genaues Mapping hängt von (mechan.) Stellbereich der Servos ab.
      posSeiten = (int)map(werte[0],0,180,0,140); // Hoher Winkel = Kippen nach Rechts
      posHoehen = (int)map(werte[1],0,180,125,0); // Hoher Winkel = Kippen zum Betrachter hin
      // Genaues Mapping hängt davon ab, wie Regler eingelernt ist (derzeit 20...160).
      leistMotor = (int)map(werte[2],0,100,20,160);
      // Empfangene Werte an den Pins ausgeben
      seitenRuder.write(posSeiten);
      hoehenRuder.write(posHoehen);
      motorRegler.write(leistMotor);
      // Rückmeldung nach jeder zehnten Kommunikation über die serielle Schnittstelle ausgeben
      if((zaehlSensor % 10 == 0)) //Modulo Funktion (Rest nach Division)
      {
        zeitSteuerNeu = millis();
        Serial.print(kommaZahl);
        Serial.print(",");
        Serial.println((zeitSteuerNeu - zeitSteuerAlt) % 10000);
        zeitSteuerAlt = zeitSteuerNeu;
      }
      zaehlSensor++;
      for(int i=0; i <= 2; i++) werte[i] = 0; // Wertearray zurücksetzen.
      arrayIndex = 0;
      kommaZahl = 0;
    }
    else
    {
      kommaZahl = 99;
    }
  }
  if ((millis()- zeitSteuerAlt) > 2000) signalFehlt = true; // Falls seit mehr als 2 s kein Signal.
  else signalFehlt = false;
  if (signalFehlt) motorRegler.write(20); // Dann Motor ausschalten.
}
```

Dieser Programmcode kann nun für Reichweitetests verwendet werden, indem die zurückgemeldete Zeit als Maß für die BT-Verbindungsqualität verwendet wird.
Für einen ersten Testflug eignet sich dieses Programm ebenfalls. Dazu müssen aber noch die Stellbereiche der Ruder und des Motorreglers angepasst werden. Du musst auch noch für den Fall einer schlechten Verbindung auch noch einen Befehl für das Einschlagen des Seitenruders einfügen, damit der ArduSmartPilot nicht einfach geradeaus weitersegelt.

Bitte vergesse nicht, die Baudrate auf dem BT-Modul auf 57600 zu ändern.

Je nach verwendetem ECS ist eine bestimmte Einschaltfolge notwendig, damit dieser beim Einschalten der Flugzeugelektronik nicht in einen Programmiermodus gelangt. Bei dem verwendeten ESC SS 8-10 A von Hobbyking muss zuerst die Akkuspannung anliegen und dann erst das PWM-Signal. Ist dies der Fall, dann quittiert der ESC dies mit einem langen Piepston. Beim Arduino-Programm `Motorflug` ist diese Einschaltfolge berücksichtigt.

Für das Überspielen der Software auf den Arduino Pro Mini benötigt man einen Programmieradapter, da diese Platine keinen USB-Seriell Wandler enthält. Bei der Verwendung des Foca-Adapters müssen folgende Pins miteinander verbunden werden: GND>GND, VCCIO>VCC, DTR>GRN, TX>RX und RX>TX.

Ausblick:

Beim Programm Motorflug werden anstatt der Sensorwerte die Kommunikationszeiten zusammen mit der Kommaanzahl übertragen. Wenn mehr übertragen werden soll, dann musst du das Datenprotokoll entsprechend ändern.

Nützlich ist z.B. eine Funktion im Arduino, die vor dem erstmaligen Starten des Motors über einen Summer den Piloten warnt. Oder wenn du ganz sicher gehen willst, dass der Motor nicht unbeabsichtigt gestartet wird: Der Arduino kann abfragen, ob maximal 10 Sekunden vor dem Motorstart ein Taster am Arduino betätigt wurde.

Zum Glück hast du kein fertig gekauftes Modellflugzeug sondern ein ArduSmartPilot in deinen Händen: Du hast es sprichwörtlich in der Hand, jetzt ist deine Kreativität gefragt. Wenn du eine gute Idee zu einer neuen Funktion hast, dann musst du kein neues Flugzeug kaufen, sondern kannst deinen ArduSmartPilot nach Lust und Laune erweitern.
Dies gilt natürlich nicht nur für die Arduino-Software, sondern auch für die Android-App und die Bauteile am Flugzeug.

Literaturverzeichnis

1: Bartmann, Erik: Die elektronische Welt mit Arduino entdecken. O'Reilly, Sebastopol, 2011.
2: Banzi, Massimo: Arduino für Einsteiger. O'Reilly, Sebastopol, 2012.
3: Brühlmann, Thomas: Arduino Praxiseinstieg. mitp, Bonn, 2012.
4: Margolis, Michael: Arduino Kochbuch. O'Reilly, Sebastopol, 2012.

Prof. Dr. rer. nat. Stefan Mack

Hochschule Reutlingen, Fakultät Technik
Studienbereich Mechatronik

Alteburgstr. 150

72762 Reutlingen

Kontakt: stefan.mack@hochschule-reutlingen.de

