ArduSmartPilot: Android Programmierung mit Processing

Einleitung

Diese Anleitung und die Anleitung „ArduSmartPilot: Arduino-Programmierung“. ergänzen sich gegenseitig: Die Arduino-Programmierung erstellt die Software für den Mikrocontoller auf der Arduino-Platine. Die hier behandelte Android-Programmierung befasst sich mit der Software für das Smartphone, die via Bluetooth (BT) mit der Software des Arduino im fertigen ArduSmartPilot kommuniziert.

Nicht Inhalt dieser Anleitung sind die Grundlagen der Android-Programmierung.
Apps unter Android zu programmieren erfordert viel Erfahrung in objektorientierter Programmierung unter Java und in der Softwareprojektverwaltung unter einer IDE (Entwicklungsumgebung) wie z.B. Eclipse.
Diesen Anforderungen kann ein Schülerprojekt unmöglich gerecht werden.
Daher wird im ArduSmartPilot-Projekt die Entwicklungsumgebung „Processing“ verwendet, die ähnlich einfach zu bedienen ist wie die Arduino IDE.

Hier werden in Form eines Kurses nur die Processing-Kenntnisse vermittelt, die unmittelbar nötig sind für die Steuerung des ArduSmartPilot. Weitergehende oder tiefer gehende Informationen finden Sie im Internet oder in Fachbüchern (siehe [1], [2], [3], [4]).

Generelle Infos

Für wen ist der Kurs gedacht?

Dieser Kurs richtet sich an technisch interessierte Personen insbesondere Schüler und Studenten, die wenig bis keine Programmiererfahrung besitzen, die aber trotzdem in wenigen Stunden einfache Programme mit einer grafischen Bedienoberfläche für den Computer (PC oder Mac) oder für ein Androidgerät (z.B. Smartphone) schreiben möchten. Darüber hinaus sollen diese Programme nicht isoliert auf dem Computer/ Androidgerät ausgeführt werden, sondern mit der Umwelt über sogenannte „Arduinos“ (Mikrocontroller(µC)-Platinen) interagieren.

Welches Ziel hat dieser Kurs?

Die Kursteilnehmer sollen verstehen, wie ein Computerbildschirm organisiert ist und wie man dessen Inhalte über einen Programmcode steuert. Sie lernen, wie man über eine Maus oder das Berühren des Bildschirms den Programmablauf steuern kann. Ein weiteres Ziel ist die Darstellung und Animation grafischer Objekte auf einem Bildschirm.
Weiter wird gelernt, wie ein Computer über die USB oder Bluetooth (BT) Schnittstelle mit einem externen µC Informationen austauschen kann. Damit soll das Computerprogramm in seinem realen physikalischen Umfeld Aktionen auslösen (z.B. einen Motor einschalten) oder umgekehrt durch sie über Sensoren beeinflusst werden (z.B. eine Warnung ausgeben, wenn der Motor defekt ist).
Das konkrete Ziel des Kurses ist das Erstellen einer Android App, die über BT den ArduSmartPilot steuert, sowie dessen Sensorwerte darstellen kann.

Auf welche Hard- und Software ist der Kurs ausgelegt?

Dieser Kurs wurde mit der Version 2.0.1 von Processing (inkl. Ketai Bibliothek „Ketai_V8_gingerbread“[1]) an einem Windows 7 PC (32 Bit), einem Smartphone Motorola Defy+ (Android 2.3.6) sowie an einem Galaxy Tab 2 7.0 WiFi (Android 4.0.4) getestet. Die Arduino-Platine (Arduino Uno) wurde mit der Arduino IDE 1.0.5 programmiert. Als BT-Shield des Arduino wurde ein Master/Slave Shield von Itead Studio, Version 2.2 (mit Bluetoothmodul HC-05) getestet.

1 Damit die BT-Kommunikation auch mit Geräten mit Androidversion 2.3.x funktioniert, sollte nicht die neuere Version Nr. 9 verwendet werden. Die neuen Adroidversionen 4.x.x sind abwärtskompatibel. Daher sollten die hier vorgestellten Programme auf jedem Androidgerät ab 2.3.x funktionieren. Näheres hierzu findet sich in [5].

Generell sollten die hier vorgestellten Codebeispiele auch mit neueren Versionen von Processing und der Arduino IDE problemlos funktionieren. Es ist wichtig, die 32 Bit Version von Processing zu installieren[2], denn die „Serial Library“ funktioniert nicht mit der 64 Bit Version. Der Arduino Code sollte auch mit einem Arduino Duemillenova/Leonardo, sowie mit davon abgeleiteten Platinen anderer Hersteller lauffähig sein. Der Processing Code sollte mit jeder Androidversion ab 2.3.x funktionieren.

Wo finde ich weitere Informationen zu den Inhalten dieses Kurses?

Informationen aus dem Internet haben den Vorteil, dass sie oft aktueller sind als die aus Fachbüchern. Jedoch (ausgenommen processing.org und learningprocessing.com) sind diese Quellen oft von schlechter inhaltlicher Qualität, didaktisch schlecht aufbereitet und liefern selten die technischen Hintergründe.

Ein gutes Fachbuch zum Thema Arduino ist eine sehr lohnende Investition: Fachbücher namhafter Verlage wie z.B. O'Reilley haben immer eine herausragende Qualität im Vergleich zu den Internetinformationen. Seien Sie so fair und laden Sie diese Bücher nicht illegal im Internet herunter – im Vergleich zum Arbeitsaufwand erhalten die meisten Autoren ohnehin nur ein Trinkgeld vom Verlag.

Im Literaturverzeichnis finden Sie eine Auswahl guter Fachbüchern, mit denen Sie die tiefer in Processing einsteigen können. Einige dieser Bücher befassen sich auch mit der Verbindung von Processing und Arduino. Sowohl die Programmiersprache Processing als auch die Arduino-Plattform haben eigene Internetseiten, auf denen man praktisch jede gesuchte Information (meistens auf Englisch) findet[3].

Unter den Lehrbüchern für Processing sind drei besonders zu empfehlen, siehe [6], [7] und [5]. Dabei ist das Buch von C. Reas eher eine Kurzreferenz als Nachschlagewerk, das Buch von E. Bartmann behandelt u.a. die selben Inhalte, jedoch wesentlich ausführlicher, in Farbdruck und auf Deutsch. Diese beiden Bücher gehen jedoch nicht auf die Besonderheiten von Processing unter Android ein. Dafür muss auf das (englische) Buch von D. Sauter zurück gegriffen werden.

Über die Arduino-Plattform gibt es inzwischen eine Vielzahl von Büchern auch auf Deutsch. Am umfassendsten ist das „Arduino Kochbuch“ von M. Margolis [8], am interessantesten ist „Making Things Talk“ von T. Igoe [9]. Das Buch von T. Igoe geht sehr gut auf die Kombination Processing (jedoch nur im Java-Modus) und Arduino ein.

Was muss ich auf meinem Rechner installieren?

Bisher war es immer sehr aufwändig und fehleranfällig, die IDE Processing inkl. deren Android-Funktionalität auf einem PC zu installieren: Im Extremfall musste dafür zuerst Processing, dann die Android SDK und oft zusätzlich noch das JAVA Development Kit getrennt installiert werden.

Seit der Version 3 von Processing wird die zusätzlich benötigte Software glücklicherweise nun automatisch nachinstalliert. Insbesondere bei der Android SDK hat dies den Vorteil, dass wirklich nur die Softwarekomponenten installiert werden, die man tatsächlich benötigt. Weiter wird alles in das User-Verzeichnis kopiert und im eigentlichen Sinne nicht installiert - man benötigt prinzipiell keine Administratorrechte mehr (bis auf die Treiberinstallation).

Bei der automatischen Android-SDK-Installation wird jedoch nur der sogenannte API-Level 10, d.h. die Androidversion 2.3.3 installiert. Dies ist aber kein Problem, da die Androidgeräte mit neueren Androidversionen abwärtskompatibel sind.

Installationsschritte:

Unter `https://processing.org/download/?processing` Zip-Datei herunterladen (Stand 12.09.2014: Version Pre-Release 3.0a4 Windows 64-bit bzw. 32-bit) und den Ordner `processing-3.0a4` z.B. in `C:\Program Files` entpacken.

2 Dabei ist es egal, ob es sich um ein 32 oder 64 Bit Betriebssystem handelt, denn Processing läuft ja nicht direkt auf dem Betriebssystem sondern auf der virtuellen Maschine von Java.

3 Siehe arduino.cc bzw. processing.org oder learningprocessing.com.

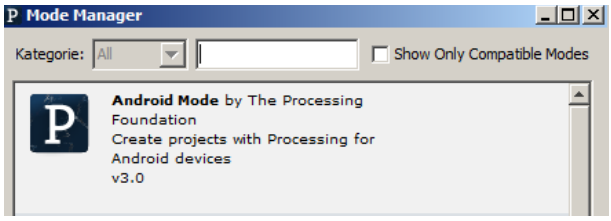
Pre-Releases

3.0a4 (12 September 2014) Win 32 / Win 64 / Linux 32 / Linux 64 / Mac OS X

Anschließend Processing starten. Da Processing nur extrahiert und nicht installiert wurde, findet sich unter Start-Programme bzw. auf dem Desktop kein entsprechendes Icon.

Um Processing zu starten, muss man im Ordner `C:\Programme\processing-3.0a4` die Anwendung `processing` ausführen. (Eine Desktop-Verknüpfung muss man sich manuell erstellen.)

Dann im Processing-Fenster rechts oben auf den Button `Java` klicken und „`Modus hinzufügen`“ auswählen:



Dann den `Android Mode` nachinstallieren und Processing anschließend mit dem selben Button in den Android Modus schalten.

Nun kann man über Processing automatisch die Android SDK installieren. Dafür bei dem Pop-Up Fenster die Option “`Download SDK automatically`“ wählen.

Dadurch werden automatisch nur die notwendigen Komponenten der SDK und nur die SDK Plattform des API Levels 10, d.h. für Android 2.3.3 installiert.

Anschließend erscheint die Warnung, dass Sie über den `SDK Manager` die Google USB Driver nachinstallieren sollten. Diese Warnung können Sie aber ignorieren, wenn Sie kein Google Nexus Androidgerät verwenden.

Zum Testen der Installation können die Beispiele unter `Datei > Beispiele > Mode Examples` verwendet werden. Wichtig ist, dass das Androidgerät sich im Entwicklermodus („Unbekannte Quellen“ und „USB Debugging“ aktivieren!) befindet und die nötigen Gerätetreiber installiert sind.

Man benötigt unterschiedliche Gerätetreiber, je nachdem ob das Smartphone nur als USB-Speicherstick erkannt werden soll, oder ob man das selbst erstellte Programm auf dem Smartphone debuggen will.

Bei Samsung-Androidgeräten gibt es dafür spezielle Installationsprogramme, die nur die Gerätetreiber installieren. Wenn Sie diese Programme auf den Samsung-Internetseiten nicht finden, dann können Sie auch die Software Kies von Samsung installieren. Dabei werden alle nötigen Gerätetreiber automatisch mit installiert.

Wenn Sie Kies (aus verständlichen Gründen) nicht auf Ihrem Rechner haben möchten, dann können Sie diese Software anschließend gleich wieder deinstallieren – die Treiber bleiben dann trotzdem auf Ihrem Rechner.

Bevor Sie das Smartphone via USB mit dem Computer verbinden muss "USB-Debugging" und "Unbekannte Quellen" aktiviert sein. Denn sonst wird das Smartphone z.B. nur als besserer USB-Stick erkannt. Wie man dies einstellt, hängt stark von der Androidversion ab. Schauen Sie am besten im Internet nach. Das Vorgehen hierfür kann teils sehr umständlich sein: Für eine bestimmtenAndroidversion muss man z.B. dafür 7 mal auf die Build-Number drücken

Falls die hier beschriebene Installation von Processing nicht funktioniert, dann sollten Sie es mit Processing Version 2 versuchen. Wie dies geht, ist z.B. [1] .ausführlich beschrieben.

Inhaltsverzeichnis

Modul 1: Der Bildschirm

Was ist eigentlich ein Pixel und wie beschreibt man dessen Farbe?

Was ist eigentlich ein Bildschirm? Gibt es eigentlich einen Unterschied zwischen dem Bildschirm von Omas alten Röhren-Farbfernseher, dem eines PCs und dem eines Android Tablets?

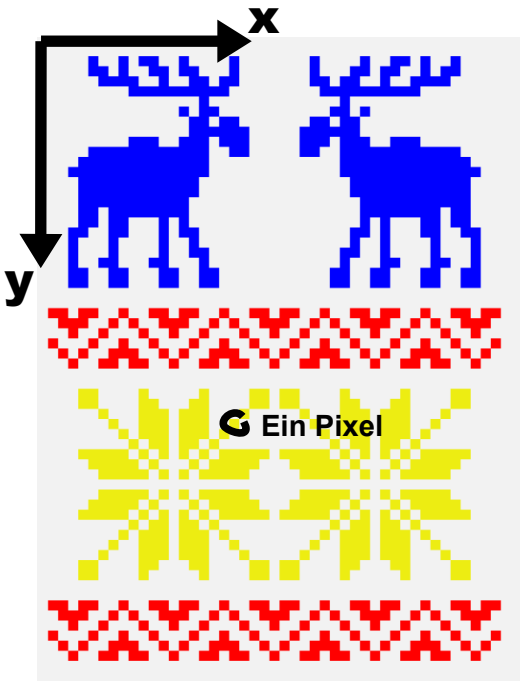


Abbildung 1: Eine Bildschirmgrafik ist wie ein Mosaik: Jeder Mosaikstein besitzt eine xy-Position und eine Farbe/Helligkeit.

Was für Informationen braucht so ein Bildschirm, damit er z.B. einen roten Kreis auf einem gelben Hintergrund darstellt?

Der Bildschirm von Omas Farbfernseher ist genau so organisiert wie der eines modernen Tablets. Wenn man genau hinschaut, dann sieht man wie die Bilder nicht aus Strichen oder Farbbereichen sondern aus vielen kleinen leuchtenden Bildpunkten bestehen. Diese Punkte heißen Pixel (von „Picture Element“) und sind regelmäßig wie die Kästchen auf einem karierten Papier angeordnet.

Eigentlich ist das Bild auf einem Computerbildschirm so etwas wie ein Mosaik, so wie in Abb. 1 das Mosaik-Muster für einen Strickpullover. Beim Strickpullover sind die einzelnen Maschen die Mosaiksteine respektive Pixel.

Dabei hat jeder Pixel seine eindeutige Position auf dem Pullover und besitzt eine definierte Farbe und Helligkeit. Die Pixelposition wird xy-Koordinaten festgelegt, ähnlich wie beim Schiffeversenken.
Die Pixelfarbe wird meistens durch drei Zahlen festgelegt: Die erste Zahl bezeichnet den Rotanteil, die zweite den Grün- und die letzte den Blauanteil.
Daher nennt man eine so festgelegte Farbe auch „RGB-Wert“ (**R**ot**G**rün**B**lau-Wert).

(255,0,0) bedeutet z.B. Rot, (0,0,0) bedeutet Schwarz und (255,255,255) steht für Weiß.
(255,255,0) bedeutet Gelb, denn rotes und grünes Licht (additiv) gemischt ergibt Gelb.

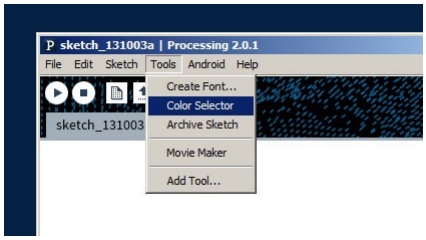


Abbildung 2: Processing Color Selector, mit dem man zu jeder gewünschten Farbe den RGB-Wert erhält.

Jede der drei Zahlen ist eine ganze Zahl zwischen 0 und 255. In Processing kann man unter dem Menüpunkt „*Tools → Color Selector*“) selbst ausprobieren (siehe Abb. 2), welcher RGB-Wert zu welcher Farbe gehört.

Auch das Icon einer Android App besteht nur aus Pixeln. Genauer gesagt besteht es aus einer Matrix von 96x96 Pixeln – also wie ein quadratisches Mosaik mit 96 x 96 Mosaiksteinen.

Ein selbst gemachtes Icon für eine Android App

Als erste Aufgabe soll ein handgemachtes Icon für deine spätere App entstehen:
Dazu wollen wir zuerst einmal ein Design für unser Icon auf dem Papier entwerfen. Zeichne hierzu ein Quadrat mit der Kantenlänge von 20 Kästchen auf einem karierten Papier. Dann fülle die Kästchen so mit Farbpunkten aus, dann von weitem betrachtet dein gewünschtes Icon entsteht. Dabei ist es am einfachsten, wenn du den Hintergrund weiß lässt.

Dein Bild sieht am Ende bestimmt viel schöner aus als das Beispielbild in Abb. 3.

Nun übertrage dein auf Papier gemaltes Icon in den Computer. Dazu öffne das Programm „MS Paint“ und erstelle eine 96 x 96 Pixel große leere Malfläche (Im Hauptmenü „Neu“ auswählen und dann unter „Start“ „Größe ändern“ wählen und dann für Horizontal und Vertikal 96 Pixel angeben. Frage den Betreuer wenn du Hilfe brauchst).

Übertrage dann dass Papierbild über Malen mit der Computermaus auf diese Malfläche und speichere es unter dem Namen `icon-96.png` ab (im Hauptmenü „Speichern unter“ wählen und „PNG-Bild“ auswählen).

Da unterschiedliche Androidgeräte unterschiedlich große Iconbilder brauchen, musst du dein Icon noch in drei weitere png-Dateien konvertieren (72, 48 und 36 Pixel im Quadrat[4]). Gebe diesen Dateien den Namen `icon-72.png`, `icon-48.png` usw. Am Ende sollten 4 verschieden große PNG-

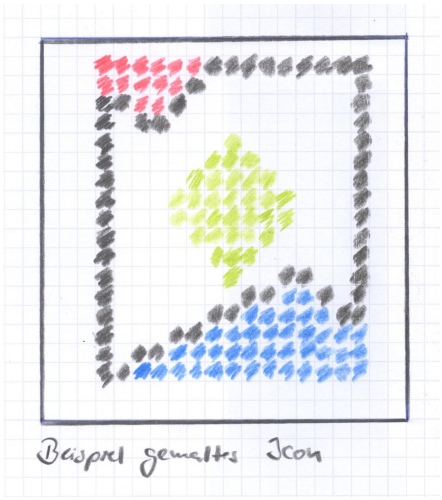


Abbildung 3: Auf Papier gemaltes Icon.

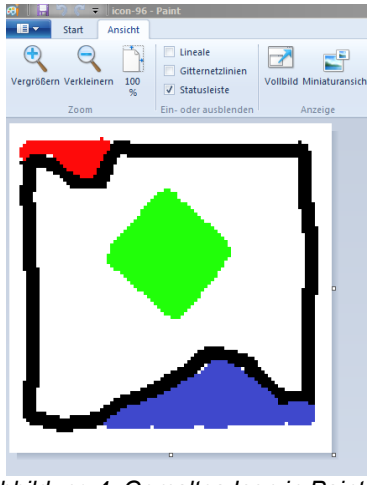


Abbildung 4: Gemaltes Icon in Paint nachgezeichnet.

Dateien mit deinem Icon vorhanden sein (siehe Abb. 6).

In Abb. 4 ist gezeigt, wie das Beispielbild aus Abb. 3 danach aussieht – irgendwie ist es dadurch auch nicht schöner geworden...

Die erste App mit Processing programmieren[5]

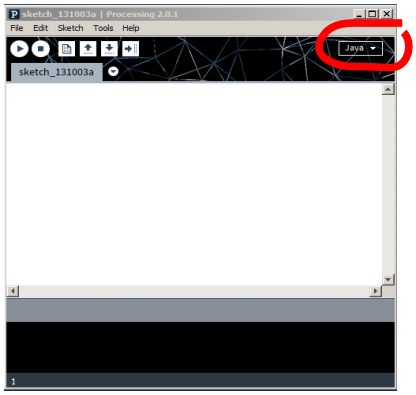


Abbildung 5: Processing im Java-Modus.

Jetzt geht es los mit dem Programmieren und du wirst in wenigen Minuten dein erstes Processing-Programm erstellt haben.

Wenn du Processing gestartet hast, ist es wichtig, dass der Java-Modus ausgewählt ist. Die Auswahl kann man rechts oben im Fenster vornehmen (siehe roter Kreis in Abb. 5).

Wenn dein Programm später auf dem Computer laufen soll, dann musst du diesen Modus auswählen, wenn es auf einem Androidgerät laufen soll, dann wähle statt „Java“ die Option „Android“ aus.
(Verwende den Emulator „Android Virtual Device“ am besten gar nicht, sondern teste die Apps immer direkt auf dem Androidgerät.)

4 Bei Paint: Im Startmenü „Größe ändern“ wählen und neue Größe in Pixeln eingeben, dann im Hauptmeü „Speichern unter“ wählen und „PNG-Bild“ auswählen. Alternativ über IrfanView „*Image → Resize/Resample*“ und dann „*File Save As ...*“ als PNG Datei abspeichern. (Siehe hierzu auch [5], Kapitel 13.2).

5 Hierfür muss nicht nur Processing installiert sein, siehe Einleitung.

Aufgabe:

Nun gebe in den Editor von Processing folgenden Code ein[6] (siehe Programmname `Bild100x100Rot`.):

Quellcode des Programms „Bild100x100Rot":

```
void setup()
{
 size(100,100);
 background(255,0,0);
}

void draw()
{
}
```

Dieses Programm macht nichts anderes als einen 100 x 100 Pixel großen Bildschirm mit hellroter Hintergrundfarbe als Fenster zu öffnen.

Was bedeuten hierbei die einzelnen Programmanweisungen?

Die geschweiften Klammern sammeln jeweils eine Abfolge von Anweisungen. Dabei stehen unter `setup()` die Anweisung, die nur einmal nach Programmstart ausgeführt werden. Unter `draw()` stehen die Anweisungen, die wiederholt bei jedem neuem Bild ausgeführt werden.

Wie ihr bestimmt wisst, besteht ein Film aus einer Abfolge von Einzelbildern. Bei einer Animation ist das ganz genau so. Normalerweise muss sich ein Bild 60 mal in der Sekunde ändern, damit der Zuschauer denkt, es sei ein bewegtes Bild. Deshalb werden die Anweisungen in der geschweiften Klammer unter `draw()` auch 60 mal pro Sekunde ausgeführt – solange bis man auf der Benutzeroberfläche von Processing auf die runde Stopptaste (zweites Symbol von links) klickt.

Das einfache Programm, welches du gerade eingetippt hast, wird übrigens mit der Starttaste auf der Benutzeroberfläche von Processing ausgeführt (erstes Symbol von links). Hier steht unter `draw()` gar keine Anweisung, weil ja auch nur das rote Quadrat und kein bewegtes Bild gezeigt werden soll.

Als nächstes erstelle einen gelben Bildschirm, der 400 x 200[7] Pixel groß ist. Das kannst du nun vielleicht schon ohne fremde Hilfe. Falls nicht, dann frage den Betreuer oder schaue im folgenden Programmcode nach (siehe Programm `Bild400x200Gelb`.):

Quellcode des Programms „Bild400x200Gelb":

```
void setup()
{
 size(400,200);
 background(255,255,0);
}

void draw()
{
}
```

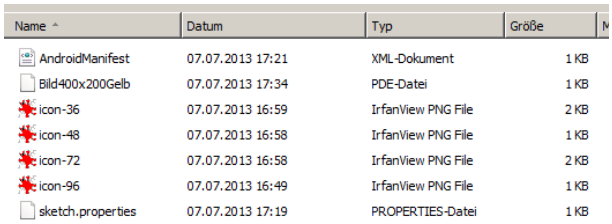
Nun kann deine Karriere als Entwickler der tollsten Killer-Apps beginnen. Beide Programme von oben kannst du ohne irgend eine Änderung im Quellcode im Android-Modus jetzt auf dem Androidgerät ausführen.

---

6 Der Quellcode kann über die Funktion „*Edit → Auto Format*“ automatisch so formatiert werden, dass wie hier die Programmstruktur einfach sichtbar wird.

7 Die erste Zahl bei der Bildschirmgröße wie auch später bei einer Position auf dem Bildschirm bezeichnet immer die horizontale, also die x-Richtung.

Dafür musst du bei Processing in den Android-Modus wechseln, indem du im Auswahlmenü (siehe Markierung in Abb. 5) auf „Android“ wechselst. Damit später auf dem Androidgerät diese App auch mit deinem eigenem Icon erscheint musst du die Pixeldateien davon in den Ordner des Processing-Programms kopieren (siehe Abb. 6). Frage dafür am besten einen Betreuer, ob er das für dich macht.
Dann verbinde das Androidgerät über USB mit dem PC[8] und starte das Processing-Programm `Bild400x200Gelb` über die Starttaste – genau so wie vorhin im Java-Modus. Nur wird nun der Start deiner App etwas länger dauern...



Name	Datum	Typ	Größe
AndroidManifest	07.07.2013 17:21	XML-Dokument	1 KB
Bild400x200Gelb	07.07.2013 17:34	PDE-Datei	1 KB
icon-36	07.07.2013 16:59	IrfanView PNG File	2 KB
icon-48	07.07.2013 16:58	IrfanView PNG File	1 KB
icon-72	07.07.2013 16:58	IrfanView PNG File	2 KB
icon-96	07.07.2013 16:49	IrfanView PNG File	1 KB
sketch.properties	07.07.2013 17:19	PROPERTIES-Datei	1 KB

Abbildung 6: Icon Dateien verschiedener Größe im Ordner des Processing Programms.

Hierbei ist neben der USB-Verbindung und der „USB-Debugging“ sowie „Unbekannte Quellen“ Einstellung am Androidgerät noch wichtig, dass der richtige Treiber für dein Smartphone auf dem Computer installiert ist.

Nun siehst du auf deinem Androidgerät ein gelbes 400 x 200 Pixel großes Rechteck. Jetzt kannst du auch statt Gelb deine Lieblingsfarbe im Programmcode eingeben und die Übertragung auf das Smartphone nochmals starten.

Wenn die App einmal von Processing aus auf das Smartphone übertragen wurde, dann erscheint sie dort unter der Rubrik „Anwendungen“ mit dem von dir erstellten Icon und mit dem Namen des Processing-Programms. Willst du diese App nochmals starten, dann musst du das nicht mehr über Startknopf von Processing machen, sondern du kannst sie wie jede andere heruntergeladene App über den Touchscreen des Androidgeräts starten.
Suche, ob du dein Icon unter der Rubrik „Anwendungen“ findest!

Fragen:

Wie viel mal wie viel Pixel ist eigentlich dein Bildschirm groß? Schätze deine Bildschirmgröße, denn die Größe des Quadrats kennst du ja.

In welcher Richtung zeigt die x-Achse und in welche die y-Achse auf deinem Bildschirm?

Was passiert, wenn du dein Androidgerät hochkant oder quer ausrichtest?

Antworten:

Der Bildschirm deines Androidgeräts ist mindestens 640 Pixel breit und 480 Pixel hoch. Oder auch ein gutes Stück größer, je nachdem was für ein Modell du hast.

Die x-Achse verläuft immer horizontal die y-Achse vertikal, egal wie das Androidgerät gedreht ist. D.h. Das Rechteck ändert sich, wenn du das Androidgerät von quer nach hochkant ausrichtest[9]. Denn ein Lagesensor (Beschleunigungssensor) im Gerät misst dessen Ausrichtung im Raum.

Genau genommen geht die x-Ache vom Benutzer aus gesehen nach Rechts und die y-Achse nach Unten. Der Ursprung des Koordinatensystems befindet sich - anders als das Koordinatensystem bei deinem Matheleher/in - in der linken oberen Ecke des Bildschirms. Also so, wie die beiden Koordinatenachsen auf dem Mosaik in Abb. 1 eingezeichnet sind.

8 Achtung: „*USB Debugging*“ und „unbekannte Quellen“ muss dafür auf dem Smartphone aktiviert sein.

9 Später wird diese automatische Ausrichtung des Bildschirms über den Befehl „`orientation(LANDSCAPE)`“ im Setup des Programms ausgeschaltet.

Modul 2: Objekte auf dem Bildschirm und deren Animation

Ein Bild mit einem Kreis drin

Das gelbe 400 x 200 Pixel große Rechteck von vorhin ist vielleicht doch nicht so ganz die Killer-App. Und bis auf das Mitdrehen beim Drehen des Androidgeräts kann man hier nun wirklich nicht von einer Animation sprechen...

Das können wir ändern: In den nächsten Schritten zeichnen wir einen Kreis auf den Bildschirm, der sich bewegt.

Erstelle im Java-Modus einen 400 x 400 Pixel großen gelben Bildschirm. Auf diesem befindet sich mittig ein roter Kreis mit 100 Pixeln Durchmesser. Am Ende soll der Bildschirm also so aussehen wie in Abb. 7.

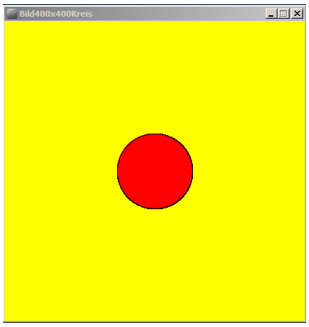


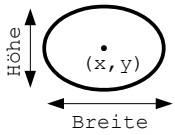
Abbildung 7: Bildschirmausgabe des Programms "Bild400x400Kreis".

Wie man den Bildschirm zeichnet, weiß Du ja schon. Den Kreis zeichnen wir jetzt nicht im `setup()` sondern im `draw()` Teil unseres Programms mit folgenden Befehlen:

Quellcode des Programms: "Bild400x400Kreis"

```
void setup()
{
 size(400,400);
 background(255,255,0);
}

void draw()
{
 fill(255,0,0);
 ellipse(200,200,100,100);
}
```



`ellipse(x, y, Breite, Höhe)`

Abbildung 8: Processingbefehl "ellipse()" zum Zeichnen eines Kreises oder einer Ellipse.

Neu sind hier folgende Befehle:

`fill(255,0,0);`, damit der Kreis eine rote Farbe erhält.

`ellipse(200,200,100,100);`, damit der Kreis an die xy-Position (200,200) mit einem Durchmesser von 100 Pixel gezeichnet wird (siehe auch Abb. 8).

Speichere dieses Processing Programm unter dem Namen `Bild400x400Kreis` ab und führe es auf dem PC aus.

Aufgaben:

Ändere dieses Programm so ab, dass der Kreis nicht mehr in der Mitte gezeichnet wird.

Ändere dieses Programm so ab, dass der Kreis kleiner oder größer ist.

Ändere dieses Programm so ab, dass der Kreis eine andere Farbe erhält.

Bei deinen Programmen bis hierhin werden die Befehle innerhalb der geschweiften Klammern unter `draw()` zwar 60 mal in der Sekunde immer wieder und wieder ausgeführt. Jedoch sind es immer die selben Befehle, weshalb sich das Bild nicht bewegt.
Das werden wir nun über eine sogenannte Schleifen- oder Laufvariable ändern.

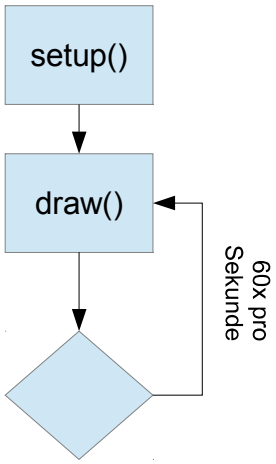


Abbildung 9: Programmablauf als Endlosschleife.

Der `draw()` Teil des Programms ist eine sogenannte Endlosschleife. Im Gegensatz dazu ist der `setup()` Teil keine Schleife. Er wird nur einmal ausgeführt[10].

Der Programmablaufplan eines leeren Processing-Programms sieht also wie in Abb. 9 aus.

Eine Schleife in der App animiert den Kreis

Speichere eine Kopie des vorherigen Programms unter dem neuen Namen `KreisAnimation` ab und ändere es wie folgt:

In den `draw()` Teil wird eine Laufvariable x eingefügt, die bei jedem Schleifendurchlauf um 1 verringert wird.

Die Laufvariable x soll die x-Position des Kreises sein. Diese Position soll in der Mitte des Bildschirms, also bei 200 anfangen und irgendwo im Nirwana enden. Damit alles nicht so schnell geht, ändern wir die Bildrate von 60 auf 2 Bilder pro Sekunde mit dem Befehl `frameRate(2);` ab. Die Befehle unter `draw()` werden jetzt also nur noch 2 mal pro Sekunde ausgeführt[11].

Quellcode des Programms „KreisAnimation":

```
int x = 200;

void setup()
{
 size(400, 400);
 frameRate(2);
}

void draw()
{
 background(255, 255, 0);
 fill(255, 0, 0);
 ellipse(x, 200, 100, 100);
x=x-1;
}
```

Starte das Programm im Java-Modus und schaue, wie sich der rote Kreis seitlich aus dem Bild heraus bewegt.
Du kannst nun an der Bewegung des roten Kreises sehen, wie eine Schleife funktioniert: Eine Reihe von Befehlen (hier die Anweisungen des `draw()` Teils) werden immer und immer wieder ausgeführt. In einer normalen for- oder while-Schleife gibt es zusätzlich noch eine Abbruchbedingungung, damit die Schleife endet. In diesem Programm würde man z.B. die Schleife dann abbrechen, wenn der Kreis aus dem Bildfeld verschwunden ist.

Jetzt, wo wir die Laufvariable x haben, können wir auch den Radius mit ihr ändern. Erstelle eine neue Kopie des aktuellen Programms unter dem neuen Namen `KreisAnimatioRadius`. Dann füge im `ellipse()` Befehl jeweils (200-x) für die Breite und Höhe der Ellipse ein. Jetzt kann auch die Bildrate auf 20 Bilder pro Sekunde gesetzt werden, damit die Animation flüssiger wirkt.

10 Jetzt muss die Farbe des Hintergrunds in den `draw()` Teil eingefügt werden, damit der sich bewegende Kreis keine schwarze Spur hinter sich her zieht. Denn im `setup()` Teil wurde der Hintergrund nur einmal ganz am Anfang gezeichnet.

11 Näheres siehe [6],Kapitel 7-2

Quellcode des Programms:"KreisAnimatioRadius"

```
int x = 200;

void setup()
{
 size(400, 400);
 frameRate(20);
}

void draw()
{
 background(255, 255, 0);
 fill(255, 0, 0);
 ellipse(x, 200, 200-x, 200-x);
 x=x-1;
}
```

Und jetzt machen wir mit der Laufvariablen noch einen dritten Effekt: Wir ändern die Farbe des Kreises. Füge dazu (200-x) für den Grünanteil im Befehl `fill()` ein und speichere dieses neue Programmversion unter dem Namen `KreisAnimationFarbe` ab.

Quellcode des Programms KreisAnimationFarbe:

```
int x = 200;

void setup()
{
 size(400, 400);
 frameRate(20);
}

void draw()
{
 background(255, 255, 0);
 fill(255, 200-x, 0);
 ellipse(x, 200, 200-x, 200-x);
 x=x-1;
}
```

Aufgaben:

Hast du noch weitere Ideen, was man alles mit der Laufvariablen x während der Animation ändern kann? Vielleicht aus dem Kreis wirklich mal eine Ellipse machen oder die Hintergrundfarbe ändern? Probiere einfach zwei Ideen als Processing-Programm aus.

If-Bedingung - damit der Kreis nicht wegläuft

Wenn du genau hinschaust, dann ahnst du, dass das Programm immer weiter läuft, auch wenn der Kreis vom Bildschirm verschwunden ist. Denn bei jedem Durchlauf von `draw()` wird die Laufvariable x um Eins kleiner, weshalb der Kreis immer weiter nach Links im Bildschirmkoordinatensystem läuft.
Wenn x kleiner als -55 ist, dann macht der Befehl `fill()` auf einmal keinen Sinn mehr, denn es gibt keinen Grünwert größer 255!
Die Programmiersprache Processing ist glücklicherweise so tolerant, dass dieser Unsinn erkannt wird. Bei anderen Programmiersprachen, würde das Programm deshalb vielleicht abstürzen.

Frage:

Wie muss das Programm erweitert werden, damit der Kreis wieder zu seiner Ausgangsposition,--radius und -farbe zurückkehrt, wenn er am Bildschirmrand angekommen ist.

Antwort:

Man muss bei jeden Durchlauf von `draw()` kontrollieren, ob x gleich Null ist, denn dann befindet sich der Kreismittelpunkt gerade am linken Bildschirmrand. Ist dies der Fall, dann wird x einfach wieder auf den Wert 200 gesetzt, und das ganze Spiel geht von Vorne los.

In der Programmiersprache macht man dies mit einer if-Bedingung. Das Flussdiagramm hierzu ist in Abb. 10 dargestellt.

Erstelle aus dem Programm `KreisAnimationFarbe` ein neues Programm mit den Namen `KreisAnimationIf`. Wie man genau das Zurückspringen des Kreises mit einer if-Bedingung programmiert, das schaust du dir am besten in dem folgenden Programmcode an:

Quellcode des Programms „KreisAnimationIf":

```
int x = 200;

void setup()
{
 size(400, 400);
 frameRate(20);
}

void draw()
{
 background(255, 255, 0);
 fill(255, 200-x, 0);
 ellipse(x, 200, 200-x, 200-x);
 x=x-1;
 if(x==0){
    x=200;
    }
}
```

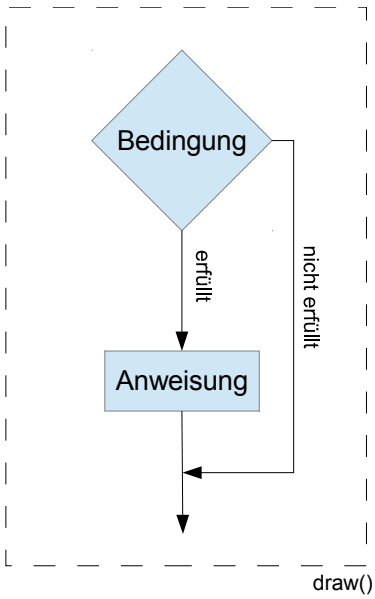


Abbildung 10: Flussdiagramm bei einer if-Bedingung.

Die letzten Programme haben wir uns ja immer nur im Java-Modus angeschaut.

Ändere Processing nun in den Android-Modus und schaue dir deine Animationen nun auf dem Androidgerät an.

So eine Animation an sich ist ja eigentlich ganz nett, aber wenn man nur zuschauen und nicht in den Programmablauf eingreifen darf, wird die ganze Sache doch recht schnell langweilig.

Deshalb lernen wir jetzt, wie man mit Mausklicks (auf einem Computer) bzw. mit Tippen auf dem Bildschirm (beim Androidgerät) ein Programm steuern kann.

Modul 3: Den Kreis klicken oder antippen: Benutzerinteraktion.

Wir verwenden das schon erstellte Programm `Bild400x400Kreis` als Ausgangspunkt für ein neues interaktives Programm.

Aufgabe:

Speichere also das Programm `Bild400x400Kreis` als `KreisKlick` ab und gebe den unten dargestellten Quellcode ein.

Bis jetzt zeichnet das Programm lediglich einen roten Kreis auf einem gelben Bildschirm. Aus dem Kreis soll jetzt aber eine Taste werden, die man im Java-Modus anklicken bzw. im Android-Modus antippen kann. Wenn der Kreis geklickt/getippt wurde, dann soll seine Farbe auf Grün wechseln.

Dazu wird nun im Processing-Programm die Mausposition bei jedem Schleifendurchlauf abgefragt . Danach wird verglichen, ob sie sich innerhalb des Kreises befindet oder nicht[12].

Im Zusammenhang mit der Maus sind bei Processing drei Parameterwerte wichtig:
Der Wahrheitswert „`mousePressed`" und die beiden Zahlenwerte „`mouseX`" und „`mouseY`".
„`mousePressed`" ist nur dann wahr, wenn die Maus gedrückt wird bzw. der Bildschirm berührt wird.
„`mouseX`" und „`mouseY`" geben einfach die Position (also die Pixelkoordinate) der Maus bzw. der Bildschirmberührung an,.

Zudem wird bei `KreisKlick` noch die Funktion `dist()` verwendet. Sie berechnet den Abstand der Mausposition vom Punkt (200,200), dem Kreismittelpunkt. Eigentlich berechnet man so etwas selbst über den Satz von Pythagoras, aber dies ist ja ein Programmierkurs und keine Mathe-Unterricht ;-)

Die if-Bedingung sieht hier etwas anders als die aus dem Programm `KreisAnimationIf`. Wie in Abb. 11 gezeigt, gibt es jetzt auch Befehle für den Fall, dass die Bedingung nicht erfüllt wird. Diese Befehle stehen innerhalb der geschweiften Klammern nach dem Wort `else`.

Quellcode des Programms „KreisKlick":

```
float abstand;

void setup()
{
 size(400, 400);
 background(255, 255, 0);
}

void draw()
{
 if (mousePressed) {
  abstand=dist(mouseX, mouseY, 200, 200);
   if (abstand <50) {
    fill(255, 0, 0);
    ellipse(200, 200, 100, 100);
   }
   else {
    fill(0, 255, 0);
    ellipse(200, 200, 100, 100);
   }
 }
}
```

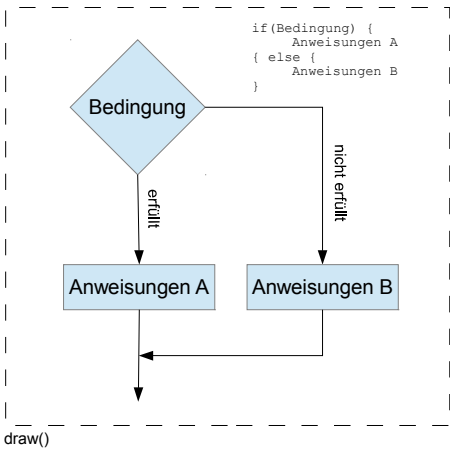


Abbildung 11: Flussdiagramm bei einer if-else-Bedingung.

12 Nähere Informationen siehe z.B. [6] Beispiel 5-16.

Frage:

Führe das Programm zuerst in Java-Modus auf dem Computer und dann im Android-Modus auf dem Androidgerät aus. Vergleiche dabei die „Maus-Klick-Aktion“ am Computer mit der „Bildschirm-Antipp-Aktion“ am Androidgerät. Was ist anders?

Antwort:

Wenn man genau hinschaut, dann bemerkt man, dass im Android-Modus die Analogie zu einem Mausklick nicht das Berühren, sondern ein Tippen des Bildschirms ist. Lässt man den Finger auf dem Bildschirm, dann ist das so, als würde man die Maus dauernd gedrückt halten.

Aufgabe:

Jetzt weißt du, wie man den Programmablauf durch Klicken bzw. Tippen manipulieren kann. Denke dir ein kleines Spiel aus, bei dem du durch Klicken/Berühren von Kreisen etwas anstellen musst (z.B. einen sich bewegenden Kreis festnageln).
Programmiere dann dieses Spiel und teste es zuerst im Java- und dann im Android-Modus.

Jetzt haben wir es geschafft, eine Animation zu programmieren und diese sogar interaktiv zu gestalten.
Der anklickbare Kreis könnte später der Ein-/Ausschalter des Flugzeugmotors sein. Und die Größe des Kreises könnte z.B. mit dem Ladezustand unseres Akkus zusammen hängen.

Wir müssen aber noch eine letzte entscheidende Sache lernen:
Wie schaffen wir es, dass unser Flugzeug erfährt, was auf dem Androidgerät so getippt und gewischt wird. Und umgekehrt – wie soll unser Androidgerät z.B. davon erfahren, wenn der Akku des Flugzeugs leer wird?

Die große Antwort auf diese Frage heißt **Kommunikation**.
Im Flugzeug befindet sich ein µC, der genau wie das Androidgerät im weitesten Sinne ein Computer ist. D.h. wir müssen lernen, wie zwei verschiedene Computer miteinander kommunizieren können.
Dies ist der Inhalt des nächsten Moduls.

Modul 4: Kommunikation Processing-Programm mit Arduino über USB-Schnittstelle bzw. über Bluetooth (BT)[13]

An der übergroßen Fußnote unten (Fußnoten sind übrigens eher für den Betreuer als für dich gedacht) siehst du, dass hier einiges schief gehen kann.
Daher kriege bitte keine Panik, wenn nicht alles auf Anhieb funktioniert ;-)

Einige Bemerkungen vorab zur Kommunikation bei Computern und µC

Die Kommunikation zwischen zwei Computern kannst du dir so vorstellen wie ein Festnetztelefonat: Bei dir zuhause gibt es eine Telefonsteckdose und bei deinem Kommunikationspartner ebenfalls. Wenn du deinem/r Freund/in am anderen Ende der Leitung etwas sagst, dann geht die Sprachinformation aus deinem Telefon in deine Telefonsteckdose hinein und kommt bei deinem/r Freund/in aus dessen/deren Telefonsteckdose wieder heraus.

Abbildung 12: Die beiden Gesprächspartner machen sich bestimmt keine Gedanken darüber, was zwischen deren Telefonsteckdosen so alles passiert...

Was zwischen diesen beiden Steckdosen technisch alles passiert, kann euch beiden egal sein.

Außerdem ist es gar nicht so einfach, dies herauszukriegen. (Je nachdem wie weit ihr voneinander entfernt seid, wird eurer Telefonat über eine Kupferleitung, über eine Glasfaserleitung, über Satellitenfunk oder über Richtfunk bzw. über eine Kombination dieser Technologien übertragen.)

Die Kommunikation zwischen einem Arduino µC und einem Computer/Androidgerät ist gar nicht so verschieden von dem Festnetztelefonat zwischen dir und deinem/r Freundin:
Das Arduino-Programm hat eine Steckdose (die heißt „serielle Schnittstelle“) und eine entsprechende Steckdose hat auch der Computer bzw. das Androidgerät. Was zwischen diesen beiden Steckdosen passiert, kann den beiden Programmen genau so egal sein wie dir und deinem/r Freund/in beim Telefonieren.

---

13 Wichtige generelle Infos:
Der Arduino soll hier vorerst als „Black Box“ aufgefasst werden. Auf den Arduino muss das Programm `Arduino-Kom` übertragen werden. Dabei ist es egal, ob später die Kommunikation via USB mit dem Computer (Processing im Java- Modus) oder via BT mit dem Androidgerät (Processing im Android-Modus) abläuft.
Aus Sicht des Arduinos macht es also keinen Unterschied, ob er via USB oder BT kommuniziert.
Denn im ersten Fall werden die seriellen Signale in USB-Signale und dann im Computer wieder zurück in serielle Signale gewandelt. Im zweiten Fall werden die seriellen Signale durch das BT-Shield in BT-Signale und dann im Androidgerät wieder zurück in serielle Signale gewandelt.

Es wird der Einfachheit halber eine Baudrate von 9600 verwendet. Beim Java-Modus muss das BT-Shield entfernt werden, da sonst die serielle Schnittstelle des Arduino vom Shield blockiert und nicht auf die USB-Schnittstelle umgesetzt wird.
Bevor das Processing Programm im Android-Modus ausgeführt wird, muss das jeweilige BT-Shield mit dem jeweiligen Smartphone gepaart werden und die Adresse des BT-Shields muss im Processing Programm eingetragen werden. (Da im Kurs viele BT-Shields parallel betrieben werden, macht diese eindeutige Zuordnung die Kommunikation einfacher.)
Falls die BT-Kommunikation nicht funktioniert, dann liegt das meistens an der falsch eingestellten Baudrate oder falsch gesetzten Jumpern am BT-Shield.
Weiter muss bei Processing die richtige Ketai-Bibliothek installiert werden. Da es sich um eine ältere Version handelt, muss diese manuell und darf nicht innerhalb vom Processing über „`Sketch → Import Library → Add Library`“ eingefügt werden (siehe Kapitel 2.5 in [5]).

Insbesondere ob zwischen diesen Steckdosen eine BT-Funkverbindung oder eine USB-Kabel seinen Dienst verrichtet, ist dem Arduino-Programm wir auch dem Processing-Programm herzlich egal. Beide kommunizieren nur mit ihrer jeweiligen Steckdose, die „serielle Schnittstelle“ heißt.[14]

Ähnlich wie in den Kapiteln zuvor werden wir hier auch wieder zuerst Processing im Java-Modus und dann erst im Android-Modus verwenden. Dieses Mal kann leider nicht mehr genau der selbe Quellcode für beide Modi verwendet werden.
Das liegt daran, dass im Java-Modus die Kommunikation über USB und im Android-Modus über BT läuft.

Jetzt denkst du bestimmt: *„Häää? Eben war doch noch die Rede davon, dass es dem Arduino und dem Processing Programm egal ist, ob nach der Steckdose eine USB oder eine BT Verbindung kommt!“*
Dem Arduino ist es tatsächlich egal.
Dem Processing Programm aber nicht ganz: Will Processing über USB kommunizieren, dann benötigt das Programm die Information, welcher USB-Anschluss denn gemeint ist. Normalerweise hat man neben dem Arduino ja noch andere Geräte via USB angeschlossen (Maus, Tastatur, ...).

Will Processing über BT kommunizieren, dann macht das mit dem Anschluss wie vorhin keinen Sinn, denn BT-Module sind nun mal nicht über einen Stecker angeschlossen (sonst wären es ja auch keine BT-Module). Um mit einem BT-Modul Kontakt aufzunehmen, braucht das Processing Programm deren Nummer. Diese Nummer ist wie eine Ausweisnummer für das BT-Modul, es gibt sie nur ein einziges Mal auf der Welt und sieht z.B. so aus: 00:11:12:11:04:73.

Übertragen auf dein Telefonat ist der Unterschied also nur folgender:
Bei einer USB-Kabelverbindung musst du zum Telefonieren vorher die richtige Steckdosen auswählen, die zu dem/r Freund/in gehört.
Bei einer BT-Verbindung musst du die komplette Telefonnummer wählen.

ASCII Zeichen – die Muttersprache der Computer und µC

Computer und µC können miteinander kommunizieren. Welche Sprache verwenden sie da eigentlich?

Generell gibt es hierfür mehrere Sprachen, die einen sind auf eine möglichst schnelle Kommunikation, die anderen auf eine übertragungssichere Kommunikation und wieder andere z.B. auf eine geheime Kommunikation ausgelegt. Leider beherrscht nicht jeder Computer jede Sprache, weshalb es bei der Kommunikation zwischen Computern oft zur kompletten Sprachlosigkeit kommt. So etwas hast du bestimmt schon einmal erlebt, wenn z.B. ein angeschlossener Drucker einfach nicht drucken will.

Es gibt aber eine „Sprache“, mit der sich fast alle Computer und µC miteinander verständigen können:
Dies sind die sogenannten **ASCII Zeichen** (siehe z.B. auch [9], Seite 54). ASCII ist eine Abkürzung für *„American Standard Code for Information Interchange“* was soviel heißt wie „Amerikanische Standardsprache zwischen Computern“.

Diese Sprache ist übrigens schon sehr alt, vermutlich älter als viele eurer Lehrer.
Zu den ASCII-Zeichen gehören alle Zeichen, die man auf einer Computertastatur findet, also Buchstaben (groß und klein), Ziffern, Satzzeichen oder auch das berühmte „@“.

14 Vom BT-Shield weiß der Arduino also gar nichts. Er kommuniziert aus seiner Sicht nur mit seiner seriellen Schnittstelle. Die Leitungen vom µC dorthin werden bei aufgestecktem BT-Shield einfach auf das BT-Modem umgeleitet.

```
                    ___
                /======/
          ____ //      \___          ,/
           | \\  //           :,    ./
     |________|__|_//           ;:;  /
    _L_____________\o           ;;;/
 ____(CCCCCCCCCCCCCC)___________-/____________________
```

Zeichnung 1: ASCII Bild. Bildquelle: asciikunst.com.

Zur Zeit, als ASCII noch eine ganz neue Sprache war, konnten auf einem Computerbildschirm nur Textzeichen (also nur die ASCII-Zeichen) dargestellt werden. Es gab keine Computergrafiken, nicht einmal solche wie der rote Kreis aus Modul 1 dieses Processingkurses.

Wenn man Bilder darstellen wollte, dann musste man mit den ASCII-Zeichen einen „Text“ tippen, der dann nur von weitem so aussah wie das eigentliche Bild. Das war eigentlich schon eine richtige Kunstform, von der ein tolles Beispiel in Zeichnung 1 dargestellt ist.

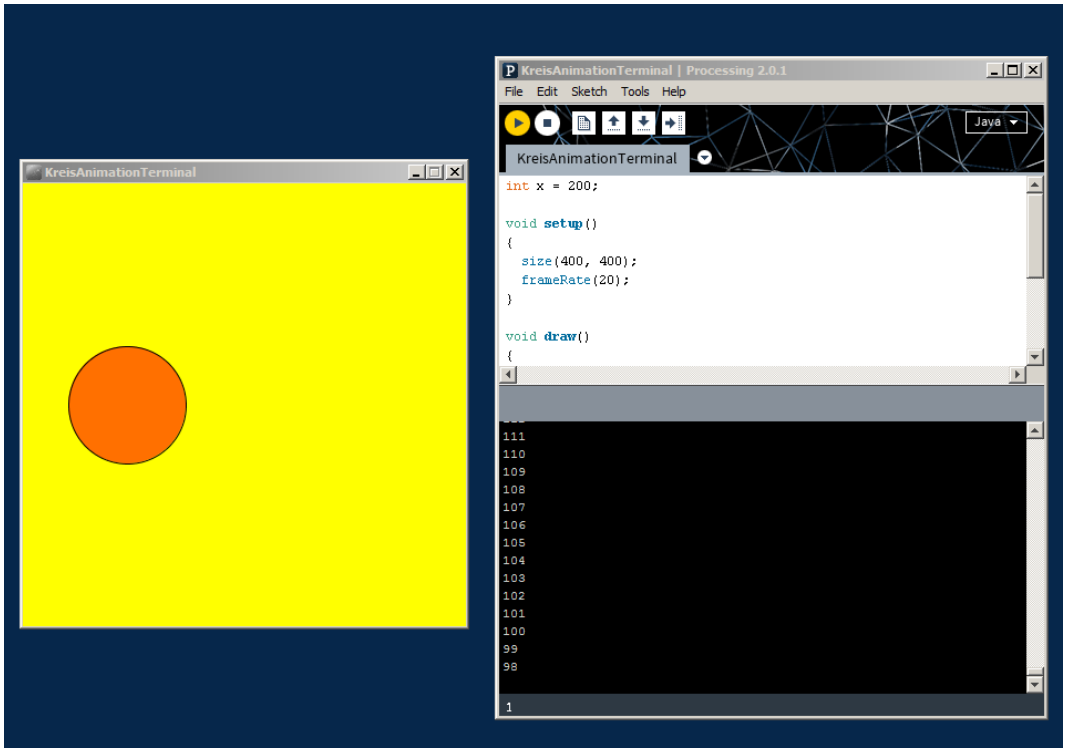


Abbildung 13: Die drei Fenster von Processing: Links das Grafikfenster, rechts oben das Editorfenster und rechts unten das Terminalfenster, in dem gerade der Wert der Variablen x ausgegeben wird.

Ausgabe von Variablenwerten im Terminalfenster

Diesen Computerbildschirm, der nur ASCII-Zeichen ausgeben kann, nannte man früher **Terminal** oder Konsole. Bei Processing existiert auch ein „Terminalfenster“ direkt unter dem Editorfenster, in dem man Quellcode eintippt (siehe Abb. 13).

Wozu so ein Terminal bei Processing? Wie du bis hierhin vielleicht schon gemerkt hast, funktionieren selbstgeschriebene Programme selten auf Anhieb so, wie man sich das vorstellt. Meistens muss man sie noch „debuggen“, was so viel heißt, wie die Fehler im Programmablauf finden. Und dafür braucht man auch bei heutigen Computern noch ein Terminalfenster.

Wenn man z.B. wissen will, welchen Wert gerade eine bestimmte Variable hat (z.B. die Laufvaria-

ble aus dem Programm `KreisAnimationIf`), dann kann man deren Wert bei jedem Schleifendurchlauf einfach im Terminalfenster ausgeben (als ASCII-Zeichen versteht sich...).

Aufgabe:

Genau das machen wir jetzt: Speichere das Programm `KreisAnimationIf` unter `KreisAnimationTerminal` ab und gebe über den Befehl `println(x)` bei jedem Schleifendurchlauf den Wert der Variablen x aus. Starte das Programm dann im Java-Modus[15].

Quellcode des Programms „KreisKreisAnimationTerminal":

```
int x = 200;

void setup()
{
 size(400, 400);
 frameRate(20);
}

void draw()
{
 background(255, 255, 0);
 fill(255, 200-x, 0);
 ellipse(x, 200, 200-x, 200-x);
 x=x-1;
 if (x==0) x=200;
 println(x);
}
```

ASCII-Zeichen werden also mit dem `println()` im Terminalfenster ausgegeben. Dabei wandelt dieser Befehl z.B. eine Ganzzahl wie die Laufvariable x automatisch in eine Kette aus Ziffern um: Statt 157 z.B. werden nacheinander die Ziffern '1', '5' und '7' ausgegeben.

Ausgabe von ASCII-Zeichen im Terminalfenster

Es gibt aber auch einen speziellen Variablentyp, der extra dafür gemacht ist, ein einzelnes ASCII-Zeichen zu speichern. Dies ist der Typ `Char`. Im folgenden Processing-Programm `TextTerminal` nutzen wir diesen Variablentyp, um vier ASCII-Zeichen auf drei verschiedene Arten auf dem Terminal auszugeben. Die `println()` Befehle ohne Argument sind nur dafür da, um Leerzeilen zu erzeugen.

Aufgabe:

Erstelle ein neues Processing-Programm mit dem Namen `TextTerminal` und gebe den nachfolgenden Quellcode ein.

15 Eigentlich sollte die Konsole im Android-Modus genau so funktionieren. Dies ist prinzipiell in einem speziellen Debug-Modus auch so vorgesehen, funktioniert aber noch nicht einwandfrei.

Quellcode des Programms TextTerminal:

```
char buchstabeA = 'A';
char buchstabeB = 'B';
char zahlEins  = '1';
char atZeichen  = '@';

void setup()
{
 size(400, 400);
 background(255, 255, 0);
 println(buchstabeA);
 println(buchstabeB);
 println(zahlEins);
 println(atZeichen);
 println();
 print(buchstabeA);
 print(buchstabeB);
 print(zahlEins);
 print(atZeichen);
 println();
 println();
 println("AB1@");
 println();
}

void draw()
{
}
```

Da in diesem Programm die Befehle im `setup()` Teil stehen, werden sie nur einmal ausgeführt. Schaue dir die drei verschiedenen Ausgaben im Terminalfenster an:

Zuerst wird durch `println()` jedes Zeichen in einer eigenen Zeile ausgegeben.
Dann wird durch `print()` jedes Zeichen in der selben Zeile nacheinander ausgegeben.
Das gleiche Ergebnis auf dem Bildschirm zeigt sich auch wenn man alle vier Zeichen in Gänsefüßchen über `println()` ausgibt. Diese vier zusammenhängenden Zeichen nennt man „`String`“. Wenn man eine Zeichenkette übertragen will, dann kann man sich also die Tipparbeit für das Ausgeben einzelner Zeichen sparen.

Übertragung von ASCII-Zeichen an den Arduino über die serielle Schnittstelle

Jetzt sind wir soweit, dass unser Processing-Programm mit einem Arduino kommunizieren kann. Den Arduino betrachten wir dabei als „Black Box“, mit der man nur über zwei Befehle kommunizieren kann: Er versteht nur Zeichenkette „`LED AN`“ und die Zeichenkette „`LED AUS`“. Alle anderen Nachrichten ignoriert er einfach. Bei der ersten Nachricht schaltet der Arduino seine rote LED (die mit Buchstaben „L“) ein und bei der zweiten Nachricht aus.

Wie man einen Arduino so programmiert, ist ganz einfach. Den Quellcode hierzu findest du im Anhang dieses Kurses[16]. Die bereitgestellten Arduinos sind schon fertig programmiert, du kannst also direkt loslegen. Das Programmieren eines Arduinos lernst du in einem anderen Kurs.

Eigentlich könnten wir jetzt wie im Programm `TextTerminal` statt `println("AB1@");` einfach `println("LED AN");` bzw. `println("LED AUS");` als Befehle einfügen.
Wir müssen dem Programm nur noch mitteilen, dass es diese Zeichenketten nicht auf dem Terminalfenster sondern auf die Kommunikationssteckdose „serielle Schnittstelle“ ausgeben soll. Das macht man dadurch, dass der Begriff `Serial` dem `println()` Befehl vorangestellt wird, also `Serial.println("LED AN");`.

16 Das Arduino-Programm hat den Namen „ArduinoKom“ ist im Anhang aufgelistet. Es wird sowohl für die USB als auch für die BT-Kommunikation verwendet.

Soweit würde das Programm nun einmal die LED einschalten und direkt danach wieder ausschalten, was wirklich nicht besonders spektakulär ist. Aber zum Glück kannst Du ja schon ein interaktives grafisches Programm erstellen:

Aufgabe:

Nimm das Programm `KreisKlick` als Ausgangspunkt und speichere es unter dem neuen Namen `TextAnArduino` ab. Beim Antippen des Kreises soll dieser jetzt nicht nur seine Farbe wechseln, sondern zusätzlich auch noch bei Wechsel auf Rot die Zeichenkette „`LED AN`“ und bei Wechsel auf Grün die Zeichenkette „`LED AUS`“ an die serielle Schnittstelle ausgeben.
Gebe nun folgenden Programmcode (ohne die mit „`//`“ beginnenden Kommentare) ein:

Quellcode des Programms TextAnArduino:

```
//Bibliothek für die serielle Kommunikation einbinden.
import processing.serial.*;
//SerielleSchnittstelle(USB-Verbindung zum Arduino)
SerialmyPort;
float abstand;

void setup()
{
 size(400, 400);
 background(255, 255, 0);
 // Alle verfügbaren Schnittstellen auflisten.
 println(Serial.list());

 // Hier die richtige COM Port Nummer angeben
 // Dafür in der Liste im Terminal nachschauen.
 myPort=new Serial(this, Serial.list()[0], 9600);
}

void draw()
{
 ellipse(200, 200, 100, 100);
 if (mousePressed) {
  abstand=dist(mouseX, mouseY, 200, 200);
  if (abstand <50) {
   fill(255, 0, 0);
   myPort.write("LED AN");
   myPort.write('\n');
 }
 else {
  fill(0, 255, 0);
  myPort.write("LED AUS");
  myPort.write('\n');
  }
 }
}
```

Die neuen Befehle sind im Quellcode über Kommentare erklärt. Der Betreuer wird sie dir näher erklären.

Schließe den (fertig programmierten) Arduino via USB an den Computer an. Starte das Processing-Programm im Java-Modus.
Das Processing-Programm muss wissen, an welchen USB-Anschluss sich der Arduino befindet. Dazu schaue in das Terminalfenster von Processing. Du wirst dort eine Nachricht ähnlich wie in Abb. 14 sehen.

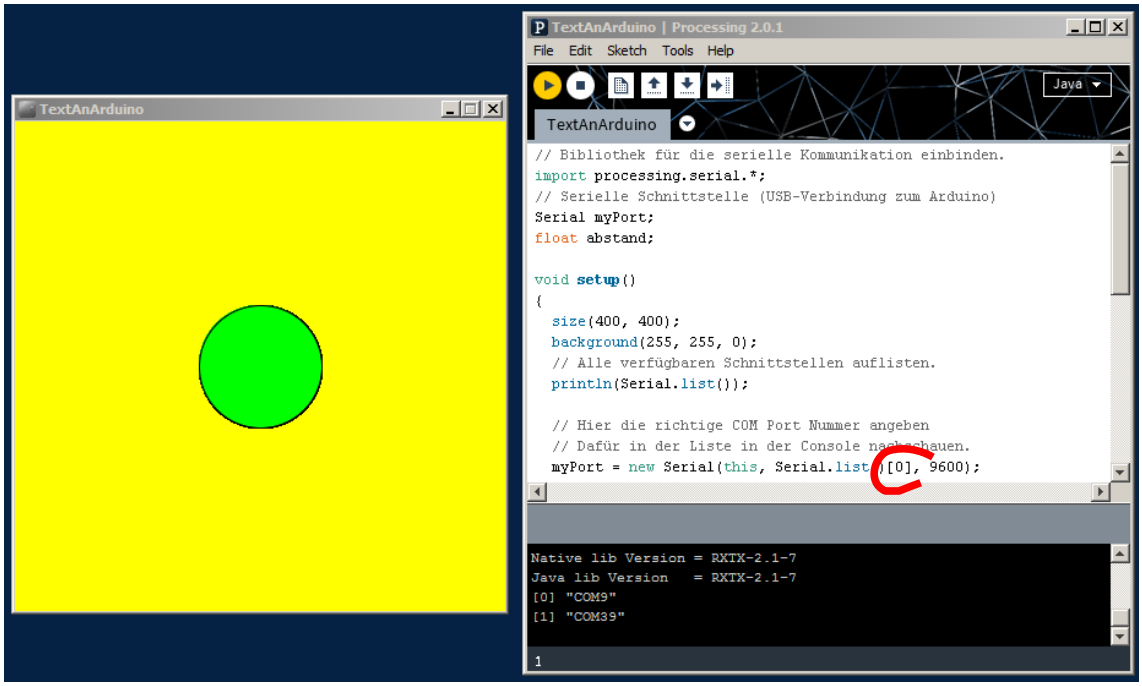


Abbildung 14: Ausgeführtes Programm `TextAnArduino`: Im Terminalfenster ist hier zu sehen, dass der USB-Anschluss "COM9", an dem der Arduino angeschlossen ist, die Nummer Null hat. Daher muss in der Quellcodezeile darüber diese Anschlussnummer eingetragen werden.

In dem Terminalfenster sind alle seriell angeschlossenen Geräte aufgelistet.
In dem Beispiel von Abb. 14 sind zwei Geräte an den seriellen Schnittstellen des Computers angeschlossen. Einer davon ist unter COM9 und der andere unter COM39 angeschlossen, wobei „COM“ für serielle Schnittstelle steht. Der Arduino ist unter COM9 angeschlossen, was man entweder durch Ausprobieren oder über den Gerätemanager von Windows herausfinden kann. Da COM9 die Nummer Null hat muss in der untersten Codezeile im Editorfenster im Befehl

```
myPort=new Serial(this, Serial.list()[0], 9600);
```

in den eckigen Klammern ebenfalls die Null eingetragen werden.

Wenn nötig ändere diese Codezeile und starte das Programm erneut. Wenn Du jetzt auf den Kreis klickst, sollte die rote LED schalten[17].

Frage:

Klicke nochmals auf den Kreis und schaue dir die beiden grünen LEDs mit der Bezeichnung „RX“ und „TX“ auf dem Arduino an.

Was beobachtest du?

Antwort:

Jedes Mal, wenn du klickst, leuchten diese LEDs. Es sind sozusagen die Betriebsleuchten der seriellen Schnittstelle am Arduino – du kannst also erkennen, wenn der Arduino kommuniziert.

17 Die Funktion des Arduino-Programms kann man auch über den Serial Monitor getrennt überprüfen. Außerdem sind im Arduinoquellcode noch Befehle auskommentiert, die man für das Debuggen verwenden kann.

Kommunikation zwischen Processing und Arduino in beiden Richtungen via USB

Frage:

Zum Glück geht Kommunikation nicht immer nur in eine Richtung. So soll es auch beim Arduino sein.

Wieso ist auch bei Computern wichtig, dass der Empfänger einer Nachricht zurückmeldet, dass er auch zugehört hat?

Antwort:

Es kann ja mal passieren, dass die Verbindung abbricht, weil z.B. jemand das USB-Kabel gezogen hat. Falls der Arduino nicht jedes Mal eine Rückmeldung gibt, wenn er eine Nachricht empfängt, dann wird das Processing-Programm diesen Verbindungsabbruch nicht erkennen können.
Später, wenn du mit dem Processing-Programm auf einem Androidgerät ein Flugzeug steuern willst, ist es noch viel wichtiger, dass du (und das Flugzeug) merkst, wenn die BT-Verbindung auf einmal schlechter wird. Sonst würde das Flugzeug bei einem kompletten Verbindungsabbruch einfach davon fliegen...

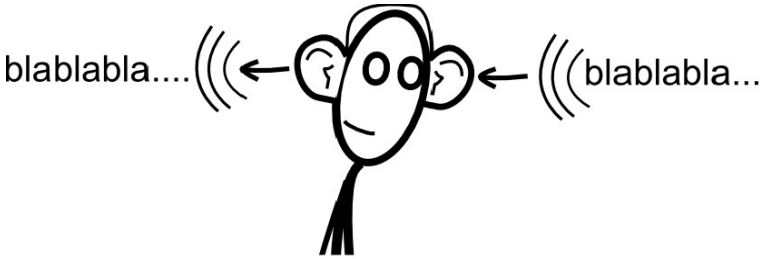


Abbildung 15: Beim Arduino kann so etwas auch passieren. Deshalb sollte er immer an Processing zurückmelden, dass er eine Nachricht empfangen hat.

In Wirklichkeit ist der Arduino so programmiert, dass er bei jedem vollständig empfangenen LED-Befehl eine Rückmeldung über den Schaltzustand der LED gibt:
Hat er sie eingeschaltet, dann meldet er die Zeichenkette „`LED ist an!`“ über die serielle Schnittstelle zurück. Hat er sie ausgeschaltet, dann meldet er „`LED ist aus!`“ zurück.

Unser aktuelles Processing-Programm `TextAnArduino` verhält sich aber noch so wie so mancher Student in der Vorlesung, siehe Abb.15!

Aufgabe:

Das muss sich ändern: Speichere das Programm unter dem neuen Namen `TextEchoArduino` ab und füge folgende neue Codezeilen (wie immer ohne Kommentare) ein:

1) Der Befehl zum Auslesen der seriellen Schnittstelle bis zum Zeichen „linefeed“.

```
echoArduino=myPort.readStringUntil(linefeed);
```

2) Die Zeichenfarbe auf Schwarz einstellen und dann die vom Arduino erhaltene Nachricht im Bild an der xy-Position (50,100) darstellen.

```
fill(0, 0, 0);
text("Arduino: " + echoArduino, 50, 100);
```


Quellcode des Programms TextEchoArduino:

```
//Bibliothek für die serielle Kommunikation einbinden.
import processing.serial.*;
//Serielle Schnittstelle(USB-Verbindung zum Arduino)
SerialmyPort;
float abstand;
//Abschlusszeichen, wenn Arduino etwas über die Schnittstelle überträgt.
int linefeed = 10;
String echoArduino = "";

void setup()
{
 size(400, 400);
 background(255, 255, 0);
 // Alle verfügbaren Schnittstellen auflisten.
 println(Serial.list());

 // Hier die richtige COM Port Nummer angeben
 // Dafür in der Liste im Terminalfenster nachschauen.
 myPort=new Serial(this, Serial.list()[0], 9600);

 textSize(24);
}

void draw()
{
 if (mousePressed) {
  background(255, 255, 0);
  abstand=dist(mouseX, mouseY, 200, 200);
   if (abstand <50) {
    fill(255, 0, 0);
    ellipse(200, 200, 100, 100);
    myPort.write("LED AN");
    myPort.write('\n');
   }
   else {
    fill(0, 255, 0);
    ellipse(200, 200, 100, 100);
    myPort.write("LED AUS");
    myPort.write('\n');
   }
  echoArduino=myPort.readStringUntil(linefeed);
  fill(0, 0, 0);
  text("Arduino: " + echoArduino, 50, 100);
 }
}
```

Abbildung 16: Bildschirmanzeige des Programms "TextEchoArduino" nachdem der Kreis angeklickt wurde.

Führe das Programm im Java-Modus aus und teste, ob die Rückantwort des Arduino wirklich auf dem Bildschirm erscheint.

Kommunikation zwischen Processing und Arduino in beiden Richtungen via Bluetooth

Leider kann man ein Flugzeug nicht über eine USB-Verbindung fernsteuern. Dafür braucht man eine drahtlose Übertragungstechnik zwischen den beiden Steckdosen. Deshalb wechseln wir jetzt in den Android-Modus von Processing.

Aufgabe:

Speichere das Programm `TextEchoArduino` unter den neuen Namen `BtTextEchoArduino` ab, denn wir müssen noch kleinere Änderungen vornehmen, damit dieser Quellcode auch auch deinem Androidgerät läuft.

Im Wesentlichen bleibt der Quellcode aber unverändert, denn es ändern sich ja nicht die Steckdosen (die bleiben für beide Seiten serielle Schnittstellen) sondern nur die Verbindung zwischen ihnen.

Am Ende soll das Antippen des Kreises beim Androidgerät genau so die rote LED ein- und ausschalten wie bisher dein Klicken am Computerbildschirm.

Ändere und ergänze den Quellcode von `BtTextEchoArduino` so wie unten dargestellt.

Wenn Du den Processing-Code nicht selbst entwickeln willst, dann kannst Du auch das Processing-Programm aus der Dokumentation verwenden. Das richtige Programm heißt dann `ManifestBtTextEchoArduino` siehe Fußnote 19.

Quellcode des Programms BTTextEchoArduino:

```
import android.os.Bundle; // Braucht man wegen der onCreate Callbackfunktion.

//KetaiBibliothekfürdieBT-Kommunikation.
//QuasiderErsatzfürdieSerialBibliothek
import ketai.net.bluetooth.*;

boolean bReleased = true; // Damit nur gesendet wird, wenn der Finger den
Bildschirm wieder verlässt nach einer Berührung.
float abstand; // Abstand des "Klicks" zum Kreismittelpunkt
String echoArduino; // Pufferstring, in dem die via BT empfangen Zeichen hinein
kommen.
KetaiBluetoothbt;
boolean dataEnd = false; // Flag, ob String komplett angekommen, da String von
Arduino manchmal häppchenweise vom BT-Modul übertragen wird

//*****************************************************************************
//DieserQuelltextwirdfürdieOrganisationderBT-Kommunikationgebraucht
//*****************************************************************************
void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   bt=new KetaiBluetooth(this);  // Konstruktor
}

//*****************************************************************************
//JetztkommtdereigentlicheQuelltext!
//*****************************************************************************
void setup()
{
   // Zeile unten kann auskommentiert werden, wenn im Manifest im <activity>
   Element
   // android:screenOrientation="portrait" eingefügt wird. Dann funktioniert
   // BT Kommunikation auch bei Start im Portrait-Ausrichtung des
   Androidgeräts.

   orientation(PORTRAIT);

   bt.start(); // BT Service starten.
   //*************************************************************
   // Hier muss die individuelle Adresse deines BT Moduls eingegeben werden.
   //bt.connectDevice("00:11:12:11:04:73");
   bt.connectDevice("00:12:08:30:01:02");
   //*************************************************************
   size(400, 400);
   background(255, 255, 0);
   textSize(24);
   frameRate(10);
   echoArduino=new String();
}
```

```
void draw()
{
  if ((mousePressed) && (bReleased == true)) //Wenn auf Bildschirm getippt
  wird
  {
    background(255, 255, 0);
    abstand=dist(mouseX, mouseY, 200, 200);
    if (abstand <50) { // Antippen innerhalb des Kreises.
      fill(255, 0, 0);
      ellipse(200, 200, 100, 100);
      byte[] data = {
      'L', 'E', 'D', ' ', 'A', 'N', '\n'
      };//Codewort für Arduino
      bt.broadcast(data);//Daten über BT senden
      bReleased=false; // Damit nicht gleich wieder Daten gesendet werden,
      wenn Finger noch auf Display
    }else { // Antippen außerhalb des Kreises.
      fill(0, 255, 0);
      ellipse(200, 200, 100, 100);
      byte[] data = {
      'L', 'E', 'D', ' ', 'A', 'U', 'S', '\n'
      };//Codewort für Arduino
      bt.broadcast(data);//Daten über BT senden
      bReleased=false; // Damit nicht gleich wieder Daten gesendet werden,
      wenn Finger noch auf Display
    }
  }
  if (mousePressed == false) //Wenn Finger vom Display weg ist
  {
  bReleased=true; // Beim nächsten Berühren werden dann wieder Daten gesendet.
  }
}

void onBluetoothDataEvent(String who, byte[] data) //Callback Methode: Es liegen
Daten vom Arduino an der BT-Schnittstelle an.
{
  //Falls noch kein \n empfangen wurde, String vom Arduino also nicht nicht
  komplett empfangen.
  if (!dataEnd) {
    //Byte Array elementweise in Char umwandeln und dem String hinzufügen.
    for (int i=0; i<data.length; i++ ) {
      echoArduino+=String.valueOf((char)data[i]);
    }
  }

  //Wenn das letzte Zeichen ein \n war, ist String vom Arduino komplett
  empfangen worden.
  if ((char)data[data.length-1]=='\n') dataEnd = true;

  //Falls String vom Arduino komplett empfangen worden.
  if (dataEnd) {
    fill(0, 0, 0);
    text(echoArduino, 50, 100); // Empfangene Daten in schwarzer Schrift
    darstellen.
    echoArduino=""; // Stingpufer zurück setzen.
    DataEnd=false; // Empfang des nächsten Strings ermöglichen.
  }
}
```

Ganz wichtig ist, dass du noch die richtige Adresse deines BT-Shields (siehe Aufkleber[18]) im Quellcode angibst – sonst schaltest du am Ende vielleicht die LED deines Nebensitzers ein und aus ;-)

Wie du bestimmt bemerkt hast, haben sich die Kommunikationsbefehle im Vergleich zur USB-Version leicht geändert.
Wenn der Finger länger auf dem Display des Androidgeräts bleibt, ist es für das Processing-Programm das Gleiche, wie ein dauerndes Mausdrücken. Normalerweise würde das Processing-Programm dann wieder und immer wieder dieselbe Nachricht an den Arduino schicken. Damit das nicht passiert und nur so viel gequasselt wird wie nötig, wird dies über die Variable `bReleased` abgefangen.

Anders als bei der Kommunikation via USB fragt das Programm nicht andauernd mit der der Anweisung *`myPort.readStringUntil(linefeed)`* nach, ob Daten an der seriellen Schnittstelle anliegen. Hier bei der BT-Kommunikation wird eine sogenannte „Callback-Funktion“ verwendet: Das Android-Betriebssystem ist nun so nett und gibt unserem Programm Bescheid, wenn Daten an der BT-Schnittstelle anliegen. Dann wird nämlich automatisch die Callback-Funktion *`onBluetoothDataEvent()`* ausgeführt. D.h. in diese Funktion muss man die Anweisungen schreiben, die im Fall einer empfangenen BT-Nachricht ausgeführt werden sollen. In dem Byte-Array *`data`* wird automatisch der Inhalt der BT-Nachricht abgespeichert.

Später werden wir einer weiteren Callback-Funktion begegnen: Die Werte des Beschleunigungssensors werden damit empfangen. Auch dieses Bauteil des Androidgeräts mag nicht dauernd gefragt werden, was es Neues gibt. Der Sensor meldet sich über die Callback-Funktion, wenn der Sensorwert sich geändert hat.

Bevor du das Programm startest musst du noch bei Processing unter dem Menüpunkt „*`Android → Sketch Permissions`*“ die Kästchen unter „`BLUETOOTH`“ und „`BLUETOOTH_ADMIN`“ aktivieren und beim Smartphone die Bluetoothfunktion einschalten.

Weiter muss noch das BT-Shield auf den Arduino aufgesetzt und mit dem Smartphone gekoppelt werden. Auf dem Arduino ist immer noch das Programm *`ArduinoKom`*. Die darin eingetragene Baudrate muss mit der des BT-Shields unbeding übereinstimmen, sonst kommt keine Kommunikation zustande. Beachte bitte, dass für diesen Kurs eine Baudrate von 9600 und für den späteren ArduSmartPilot eine Baudrate von 57600 verwendet wird.

Da die rote LED auf der Arduino-Platine durch das BT-Shield verdeckt wird, stecke zwischen Port 13 und GND auf dem BT-Shield eine LED inkl. Vorwiderstand, die du vom Betreuer erhältst. Diese LED kannst du nun mit deinem Androidgerät ein- und ausschalten.

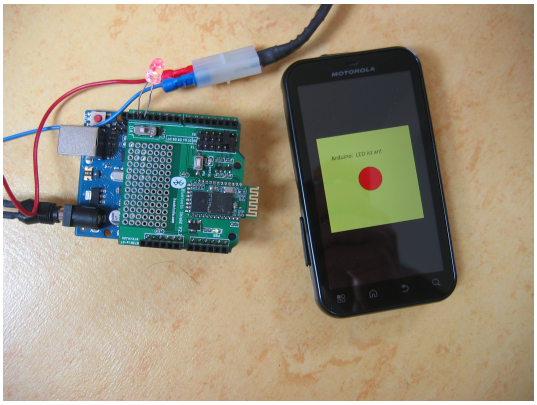


Abbildung 17: LED ist eingeschaltet.

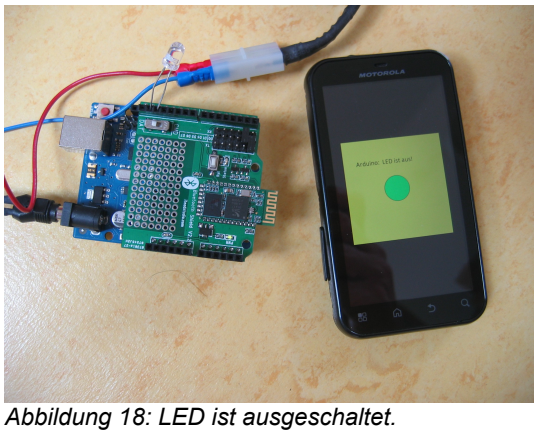


Abbildung 18: LED ist ausgeschaltet.

18 Wenn die Adresse des BT-Shields oder BT-Moduls nicht bekannt ist, dann kann man diese mit einer Android App herausfinden. Dafür eignet sich z.B. die App „Bluetooth Scanner & Widget“ von Google Play. Bei eingeschaltetem Bluetooth am Smartphone kann man das (ebenfalls eingeschaltete) BT-Shield suchen und dann dessen Namen und Adresse ablesen. Um später mit dem Arduino zu kommunizieren muss man das BT-Shield aber noch paaren. Das geht zuverlässig über die Bluetooth-Einstellungen am Smartphone, manchmal aber auch über eine Bluetooth-Scanner-App. Die hierfür nötige Pin ist meistens „1234“.

Kontrolliere ob das BT-Shield richtig aufgesetzt ist und diessen grüne LED schnell und regelmäßig blinkt. **Halte dein Androidgerät hochkant ausgerichtet**[19] und starte jetzt das Processing-Programm. Wenn eine Verbindung zwischen dem Androidgerät und dem BT-Shield hergestellt ist, dann blinkt die grüne LED immer zwei Mal kurz hintereinander auf.

Jetzt kannst du wie in Abb. 17 und 18 gezeigt, die LED mit deinem Androidgerät ein- und ausschalten.

Aufgabe:

Probiere aus, wie hoch die Reichweite hierbei ist: Wie weit kannst du dich mit deinem Androidgerät entfernen und deine Fernsteuerung funktioniert immer noch?
Für dieses Experiment erhältst du vom Betreuer einen Akkupack, mit dem der Arduino inkl. BT-Shield mit Spannung versorgt wird. Damit brauchst du das USB-Kabel nicht mehr und kannst die Reichweite z.B. auf dem Flur oder besser sogar im Freien testen.

Fragen:

Was bemerkst du an der Rückmeldung des Arduino wenn du langsam an die Grenze der BT-Reichweite kommst?

Messen deine Mitschüler mit anderen Androidgeräten die gleichen Reichweiten wie du?

Antworten:

Gerätst du an die Grenze der BT-Reichweite, dann dauert es merklich länger bis der die Rückantwort des Arduino auf dem Bildschirm erscheint.

Eigentlich haben alle Androidgeräte ein ähnlich starkes Funkmodul. Daher sollten die gemessenen Reichweiten deiner Mitschüler nicht groß von deiner abweichen.
Große Unterschiede können aber entstehen, wenn viele andere BT-Sender oder Gebäudewände in der Nähe sind. Daher testet man die Reichweite am besten alleine im Freien ein paar Hundert Meter weg von irgendwelchen Gebäuden.

19 Im Quelltext des Programms ist mit dem Befehl `orientation(PORTRAIT)` die Hochkant Ausrichtung festgelegt. Es ist ein bekannter Bug von Processing, dass die `setup()` und `onCreate()` Methoden zweimal ausgeführt werden, wenn das Androidgerät beim Start anders ausgerichtet ist. Das führt dazu, dass die BT-Verbindung zwar aufgebaut wird, aber keine BT-Kommunikation möglich ist.
Eine Lösung dieses Problems ist es, den `orientation()` Befehl ganz wegzulassen und die feste Ausrichtung im Android Manifest über `android:screenOrientation="portrait"` im `<activity>` Element zu bestimmen. Dann kann das Programm in jeder Ausrichtung des Androidgeräts gestartet werden. Die Programmversion hierzu heißt `ManifestBtTextEchoArduino`.

Modul 5: Lagesensoren des Androidgeräts im Processing-Programm auslesen

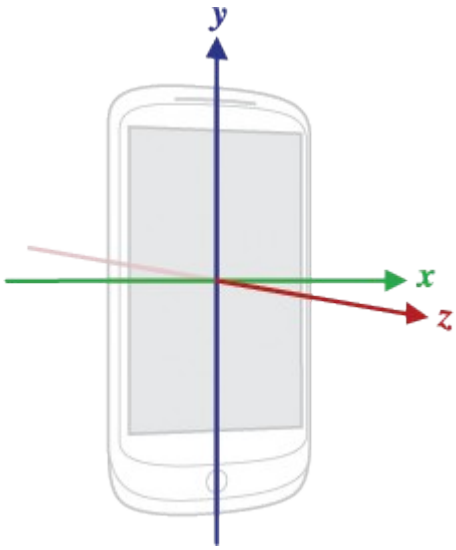


Abbildung 19: Festlegung der x,y und z-Achse für den Lagesensor im Androidgerät. Bildquelle: http://developer.android.-com/reference/android/hardware/SensorEvent.html.

Bei deiner ersten App war dir sicher aufgefallen, wie sich das Rechteck mitdrehte, wenn das Androidgerät gedreht wurde. Du kennst sicher auch Apps, bei denen man das Androidgerät als Lenkrad für ein Autorennen benutzt.

Diese Funktionen sind nur möglich, weil sich unter anderem ein Lagesensor im Androidgerät befindet, der die Kippung aus der Horizontalen heraus messen kann. Als Lagesensor wird der Beschleunigungssensor des Androidgeräts verwendet.

Mit einem Beschleunigungssensor kann man sehr viel mehr als nur eine Kippung messen: Die Beschleunigung eines Autos etwa, oder das Rütteln einer Maschine. Man kann mit ihm aber genau so gut auch nur die Erdbeschleunigung messen. Wir degradieren den stolzen Beschleunigungssensor also jetzt zu einer einfachen Wasserwaage ;-)

Die Erdbeschleunigung ist im strengen Sinn gar keine Beschleunigung, denn solange du dein Androidgerät gut festhältst, wird es ja nicht Richtung Boden beschleunigt.

Mit Beschleunigung ist hier aber die Anziehungskraft der Erde gemeint. Und nach Newton gilt $\vec{F} = m \cdot \vec{a}$, weshalb Anziehungskraft und Erdbeschleunigung praktisch das gleiche sind.

Wie du beim Thema Pixel gelernt hast, verlaufen die x- und y- Achsen in der Ebene des Bildschirm (siehe Abb. 1). Die z-Achse verläuft senkrecht dazu und schaut aus dem Bildschirm heraus.

Dummerweise verläuft für den Lagesensor die y-Achse genau anders herum als für den Bildschirm. Vergleiche dazu Abb. 1 und Abb. 19.

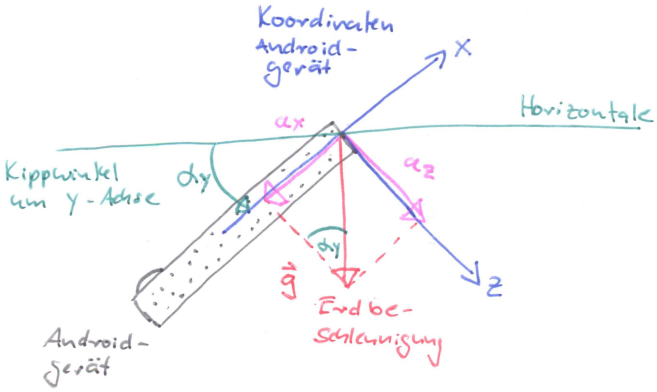


Abbildung 20: Für das Androidgerät wird dessen Kippwinkel um die y-Achse über die gemessene Beschleunigung (=Anziehungskraft) in x- und z-Richtung berechnet.

Die Erdbeschleunigung g zeigt immer vertikal in Richtung Erdmittelpunkt.
Ist das Androidgerät nun horizontal ausgerichtet so misst es in <u>seiner eigenen</u> x- und y-Richtung keine Beschleunigung a_x bzw. a_y.
In <u>seiner eigenen</u> z-Richtung misst es die komplette Erdbeschleunigung von a_z = 9.81 m/s², denn seine z-Achse zeigt ja genau in Richtung der Erdbeschleunigung g.

Was passiert, wenn jetzt das Androidgerät um 45° um seine x-Achse gedreht wird? (Siehe Abb.20)

Dann ist die x-Beschleunigung immer noch gleich Null. Die z-Beschleunigung hat auf $\frac{1}{\sqrt{2}} \cdot 9{,}81 \frac{m}{s^2}$ abgenommen und die y-Beschleunigung auf den gleichen

Wert zugenommen. Die y- und z-Beschleunigungen sind jetzt gleich groß, da die Erdbeschleunigung g mit beiden Achsen den gleichen Winkel einschließt.

Bei Beschleunigungen ist es so wie bei einem Kräfteparallelogramm: Die Vektoraddition aller drei Beschleunigungskomponenten ergibt die Erdbeschleunigung. In diesem Fall ergibt die Vektoraddition der y- und z-Beschleunigung wieder die Erdbeschleunigung g.

Aus dem dem Beschleunigungsdiagramm in Abb. 20 lässt sich ablesen, dass die Kippung α_x um die x-Achse über den Arcustangens berechnet werden kann:

$$\alpha_x = \arctan\left(\frac{a_y}{a_z}\right)$$

Genau so kann man entsprechende Formel kann man auch für die Kippung α_y um die y-Achse herleiten. Hier gilt:

$$\alpha_y = \arctan\left(\frac{a_x}{a_z}\right)$$

Wie im Modul 4 benutzen wir jetzt wieder die Ketai Bibliothek.
Dieses Mal nutzen wir den Teil der der Bibliothek, der sich um das Auslesen der Sensoren im Androidgerät kümmert.

Ausgabe der Werte des Lagesensors auf dem Bildschirm

Daher steht hier nun in der ersten Quellcodezeile `import ketai.sensors.*` statt `import ketai.net.bluetooth.*`. Nähere Informationen zur Ketai-Bibliothek und generell zu BT und zu der Nutzung der internen Sensoren eines Androidgeräts findest du in [5].

Aufgabe:

Erstelle ein neues Processing-Programm mit dem Namen `SensorTest` und gebe den nachfolgenden Programmcode[20] ein (wie immer ohne die Kommentare).

Quellcode des Programms SensorTest:

```
//Verweis auf die Ketai-Bibliothek mittels der auf die Sensorwerte zugegriffen wird.
import ketai.sensors.*;

//Analog wie beider BT-Bibliothek ketai.net.bluetooth.* bedarf es eine Variable für das
Sensorobjekt.
KetaiSensor sensor;
//In diesen Variablen werden später die x-,y-und z-Beschleunigungswerte eingetragen.
float beschleunigungX, beschleunigungY, beschleunigungZ;

void setup()
{
 size(400, 400);
 // Analog wie bei der BT-Bibliothek ketai.net.bluetooth.* muss ein Sensorobjekt erzeugt
werden.
 sensor=new KetaiSensor(this);
 // Hiermit wird ein Dienst gestartet, der sich über die Callbackmethode zurück meldet,
 // falls sich ein Sensorwert geändert hat.
 sensor.start();
 // Bildschirm des Androidgeräts bleibt quer ausgerichtet, egal wie es gedreht wird.
 orientation(LANDSCAPE);
 // Der Text auf dem Bildschirm soll in der Mitte dargestellt werden in der Schriftgröße
36.
 textAlign(CENTER, CENTER);
 textSize(36);
```

20 Der Quellecode stammt teilweise aus dem Buch von D. Sauter [5], siehe auch http://danielsauter.com.

```
}

void draw()
 {
 //Gelber Hintergrundbildschirm
 background(255, 255, 0);
 //Ausgeben der Beschleunigungswerte auf dem Bildschirm in Schwarz.
 fill(0, 0, 0);
 text("B-Sensorwerte (m/s²):\n" + //(3)
  "ax: " + nfp(beschleunigungX, 2, 3) + "\n" +
  "ay: " + nfp(beschleunigungY, 2, 3) + "\n" +
  "az: " + nfp(beschleunigungZ, 2, 3), width/2, height/2);
}

//Diese Methode wird jedes Mal automatisch aufgerufen, wenn sich ein
//Sensorwert geändert hat.
void onAccelerometerEvent(float x, float y, float z)
{
 beschleunigungX=-x;
 beschleunigungY=-y;
 beschleunigungZ=-z;
}
```

Abbildung 21: Programm SensorTest auf dem Androidgerät.

Die Bedeutung der einzelnen Befehle ist in den Kommentaren erklärt.
Ähnlich wie beim Programm `BtTextEchoArduino` gibt es hier auch wieder eine Callback-Funktion. Nur hier wird diese Funktion nicht ausgeführt, wenn Daten an der BT-Schnittstelle anliegen, sondern, wenn der Beschleunigungssensor einen neuen Messwert liefert. Das Programm `SensorTest` wartet bis ein neuer Sensorwert vorliegt und stellt dann die x-, y- und z-Beschleunigungswerte auf dem Bildschirm dar.

Aufgabe:

Starte das Programm `SensorTest` im Android-Modus und teste ob die angezeigten Messwerte des Beschleunigungssensors bei Kippung in verschiedene Richtungen Sinn machen[21]. Beachte dabei das Koordinatensystem in Abb. 19 und denke daran, dass die Erdbeschleunigung 9,81 m/s² beträgt. Falls es hierbei Abweichungen gibt, korrigiere über den Quellcode deines Programms, so dass dies auf deinem Androidgerät korrekt funktioniert.

Frage:

Welchen maximalen und minimalen Beschleunigungswert solltest du für jede Koordiantenrichtung messen?

Antwort:

Mehr als die Erdbeschleunigung von 9,81 m/s² kann man nicht messen. Dieser Maximalwert stellt sich dann ein, wenn die entsprechende Achse in Richtung Erdmittelpunkt ausgerichtet ist. Ist die Achse genau entgegengesetzt ausgerichtet, dann wird der minimale Wert von -9,81 m/s² gemessen.
Tatsächlich kann es sein, dass dein Androidgerät Beschleunigungen größer 10 m/s² bzw. kleiner

21 Je nach Androidgerät kann die Ausrichtung der x- und y-Achse von Bild 19 abweichen. Ob das ein Bug von Processing ist, oder am Gerät liegt ist unklar.

-10 m/s² ausgibt. Das liegt daran, dass der Beschleunigungssensor wie man bei den Ingenieuren sagt „nicht kalibriert“ ist, wie ein Lineal, was zu heiß geworden ist und dadurch geschrumpft ist. Für den Lagesensor ist dieser Messfehler aber unerheblich, da hier nur das Verhältnis der Beschleunigungen zweier Achsen ausgewertet wird.

Steuerung einer Animation über den Lagesensor

Prinzipiell gibt es zwei Möglichkeiten, den ArduSmartPilot über das Androidgerät zu steuern:

1) Man kann die Position des Fingers auf dem Bildschirm auslesen und über dessen x-Position das Seitenruder und über dessen y-Position das Höhenruder steuern.
Wie man die Position des Fingers ausliest haben wir in Modul 3 gelernt.

2) Man kann das Seitenruder aber auch über die Kippung des Androidgeräts um die y-Achse und das Höhenruder über die Kippung um die x-Achse steuern. Dafür müssen wir das Programm `SensorTest` jetzt noch erweitern.

Aufgabe:

Speichere das Programm `SensorTest` unter dem neuen Namen `SensorKipp` ab.
Im Quellcode sollen jetzt aus den Sensorwerten die beiden Kippwinkel berechnet werden. Zusätzlich kommt jetzt wieder der rote Kreis ins Spiel: Er soll seine xy-Position in Abhängigkeit von den beiden Kippwinkeln annehmen. Falls du im Abschnitt zuvor die Ausrichtung der xy-Achsen korrigieren musstest, dann denke jetzt auch daran.

Gebe den nachfolgenden Quellcode ein.

Quellcode des Programms SensorKipp:

```
import ketai.sensors.*;

KetaiSensor sensor;
//In diesen Variablen werden später die x-,y- und z-Beschleunigungswerte eingetragen.
//Initialisierung mit Wert 0, damit auch ohne Sensor Event Kreisgezeichnet wird.
float beschleunigungX=0, beschleunigungY=0, beschleunigungZ=0;

void setup()
{
 size(400, 400);
 sensor=new KetaiSensor(this);
 sensor.start();
 orientation(LANDSCAPE);
 textAlign(CENTER, CENTER);
 textSize(24);
}

void draw()
{
 //Variablen für x- bzw. y-Position des Kreises
 int x=0, y=0;
 //Variablen für Kippwinkel um y- bzw. x-Achse
 int kippWinkelY=0, kippWinkelX=0;
 background(255, 255, 0);
 fill(255, 0, 0);
 //Trigonometrische Berechnung der Kippwinkel in Grad, mit (int) Umwandlung von float →
int.
 kippWinkelX=(int)(atan(beschleunigungY/beschleunigungZ)*180/PI);
 kippWinkelY=(int)(atan(beschleunigungX/beschleunigungZ)*180/PI);
 //Umrechnen der Kippwinkel in xy-Koordinatenpositionen
 x=(int)map(kippWinkelX, -90, 90, 0, 400);
 y=(int)map(kippWinkelY, -90, 90, 0, 400);
 //Kreis an der Position (x,y) zeichnen
 ellipse(x, y, 50, 50);
 fill(0, 0, 0);
```

```
 //Kippwinkel als Text ausgeben
 text("Kippwinkel (°):\n" + //(3)
  "alpha x: " + kippWinkelX + "\n" +
  "alpha y: " + kippWinkelY + "\n", width/2, height/2);
}

//Diese Methode wird jedes Mal automatisch aufgerufen, wenn sich ein
//Sensorwert geändert hat.
void onAccelerometerEvent(float x, float y, float z)
{
 beschleunigungX=-x;
 beschleunigungY=-y;
 beschleunigungZ=-z;
}
```

Die Bedeutung der neuen Befehle ist wieder in den Kommentaren erklärt.

Starte das Programm in Android-Modus und kontrolliere, ob sich der rote Kreis durch Kippen des Androidgeräts so bewegt wie er es soll (also wie ein Kugel, die auf dem Bildschirm rollt).

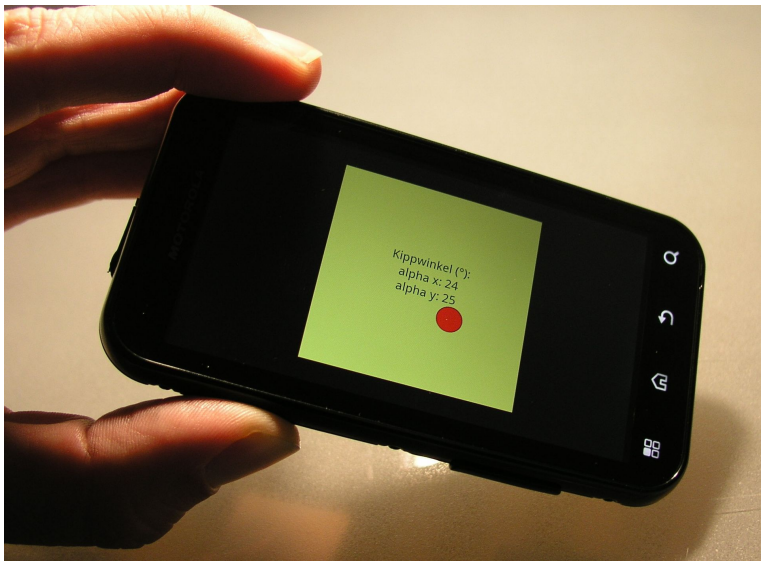


Abbildung 22: Programm SensorKipp auf dem Androidgerät.

Aufgabe:

Erstelle ein Processing-Programm, in dem weitere Eigenschaften des Kreises (Farbe, Durchmesser, Form der Ellipse usw.) sich mit dem Kippwinkel ändern.

Was hast du gelernt?

Jetzt haben wir es geschafft. Wir haben alle alle Teile einer Android-Fernsteuerung für unseren ArduSmartPilot programmiert und getestet:

- Du kannst über Antippen von Bildschirm-Objekten etwas ein- und ausschalten (z.B. später den Motor deines Flugzeugs).
- Du kannst über Antippen des Bildschirms eine xy-Position erzeugen (um damit später in eine Position von Höhen- und Seitenruder zu steuern).
- Du kannst die oben erzeugten Befehle drahtlos über BT an das Flugzeug senden.
- Du kannst Daten vom Flugzeug empfangen und auf dem Bildschirm darstellen (z.B. ein Sensorwert oder eine Warnung, wenn der Akku leer wird).
- Du kannst über die Sensoren die Kipplage des Androidgeräts auslesen (um damit später in eine Position von Höhen- und Seitenruder zu steuern).
- Du kannst Animationen auf dem Bildschirm ablaufen lassen (um z.B. den aktuellen Kippwinkel graphisch anzuzeigen).

Fertige Beispielapp zur Steuerung des ArduSmartPilot

Im Anhang ist der Code dieses funktionsfähigen Processing-Programms wiedergegeben.

Abbildung 23: Bildschirmlayout der Beispiel-App ArduSmartPilot.

Wie in Abb. 23 zu sehen ist der Bildschirm des Androidgeräts in drei vertikale Bereiche eingeteilt:
Ganz links werden die beiden vom Arduino zurückgemeldeten Werte angezeigt. Rechts daneben kann man über fünf Tasten die Motorleistung auswählen. Auf der rechten Seite wird über einen roten Punkt wie im Abschnitt „Steuerung einer Animation über den Lagesensor„ die Kippwinkel um die x- bzw. y-Achse angezeigt. Das Fadenkreuz zeigt den Kippwinkel 0° für x und y an.

Dieses Processing-Programm sendet alle 50 ms einen Zahlentripel an den Arduino: Die ersten zwei Zahlen sind die Kippwinkel um die x- bzw. y-Achse (jeweils -90...+90), die dritte Zahl ist die Motorleistung (0, 25, 50, 75 oder 100).

Diese App funktioniert nur zusammen mit dem Arduino-Programm Motorflug, welches im Arduinokurs entwickelt und erklärt wird.
Falls diese App nur richtig funktioniert, wenn sich das Androidgerät in Querausrichtung befindet, dann muss manuell im Manifest eine Codezeile eingefügt werden. Siehe dazu die Fußnote auf Seite 27.

Wie geht es weiter?

Als erstes kannst du die Benutzeroberfläche deinem Geschmack anpassen.
Die Steuerung der Motorleistung kann auch über die Lautstärketasten anstelle der virtuellen Tasten auf dem Bildschirm erfolgen.

Vielleicht willst du auch mehr Sicherheit in das System bringen: Die App könnte z.B. den Motor ausschalten, wenn das Androidgerät heruntergefallen ist oder gar nicht mehr bewegt wird. Oder es meldet sich ein Vibrationsalarm/Audiosignal, wenn die Akkuladung zu Ende geht.

Hast du eine erste Version fertig, dann musst du diese zuerst ausgiebig am Boden testen. Denke an alles, was beim Flug schief gehen kann und bereite dein Programm darauf vor: Z.B. muss das Programm verhindern, dass man aus Versehen den Propeller einschaltet. Der Pilot sollte vom Programm gewarnt werden, wenn der Akku leer wird.
Auch musst du am Androidgerät den Display-Timeout ausschalten, damit während des Fluges die App vom Betriebssystem nicht in Wartestellung versetzt wird.

Mit Processing kann man ein Androidgerät noch weitaus mehr programmieren als hier gezeigt wurde. Man kann sämtliche andere Sensoren inkl. der Kamera des Androidgeräts nutzen, oder man kann über WLAN kommunizieren, um nur zwei Möglichkeiten zu nennen.

Die vom Androidgerät empfangenen Sensordaten des Arduino können für eine spätere Auswertung zwischengespeichert werden. Vielleicht möchtest du sie aber auch in Echtzeit auf dem Bildschirm des Androidgeräts visualisieren.

Wenn du viel tiefer in die Programmierung von Androidgeräten einsteigen willst, dann kannst du auch auf eine professionelle Entwicklungsumgebung wie z.B. Eclipse umsteigen[22]. Dafür kannst du die bisher in Processing erstellten Programme konvertieren. Der große Vorteil von Eclipse ist das komfortable Debuggen (=Fehlersuche) deines Programms. Außerdem kannst du deine fertige App damit bei Google Play einstellen.

...ja und da ist noch die Programmierung des Arduino. Zum Glück sind die Softwarewerkzeuge und die Programmierung dazu recht ähnlich zu Processing.
Viel Spaß beim nun folgenden Arduinokurs, in dem einíge der hier entwickelten Apps zum Testen der Arduino-Programme verwendet werden.

Literaturverzeichnis

[1]: Sauter, Daniel: Rapid Android Developement. O'Reilly, Sebastopol, 2013.
[2]: Reas, Casey et al.: Getting Started with Processing. O'Reilly, Sebastopol, 2010.
[3]: Bartmann, Erik: Processing. O'Reilly, Sebastopol, 2010.
[4]: Margolis, Michael: Arduino Kochbuch. O'Reilly, Sebastopol, 2012.
[5]: Igoe, Tom: Making Things Talk. O'Reilly, Sebastopol, 2011.

Prof. Dr. rer. nat. Stefan Mack

Hochschule Reutlingen, Fakultät Technik
Studienbereich Mechatronik

Alteburgstr. 150

72762 Reutlingen

Kontakt: stefan.mack@hochschule-reutlingen.de

22 Siehe dazu in [5] Kapitel 13.3.

Anhang:

Für die Kommunikation mit dem Arduino (egal ob über USB oder über BT) muss auf den Arduino das Programm `ArduinoKom` übertragen werden.

Der Quellcode ist nachfolgend wiedergegeben. Er wird im Dokument „ArduSmartPilot: Arduino-Programmierung“ schrittweise entwickelt und erklärt. Bitte beachte, dass dies ein Arduino- und kein Processing-Programm ist. Daher hat das Programm auch die Endung „`.ino`“ und nicht „`.pde`“.

Wenn du diesen Code in Processing verwenden willst, dann übertrage ihn nicht via copy/paste aus diesem Dokument in den Processing-Editor. Denn dabei werden zusätzliche Zeilenumbrüche erzeugt oder Sonderzeichen wie z.B. das Hochkomma nicht korrekt übertragen.
Die Quellcodes aller Codebeispiele liegen unter ardusmartpilot.de zum Download bereit.

Quellcode des Arduino-Programms ArduinoKom:

```
char einByte = 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)
  {
    einByte = Serial.read(); // ... jeweils ein einzelnes Byte lesen.
    if (einByte == '\n') //Wenn ein "Neue Zeile"-Zeichen kommt...
    {
      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.
      }
      zeichenInput = ""; // Puffer wieder leer machen
    }
    else
    {
      //Wenn das Endzeichen noch nicht da, neues Zeichen an die vorhandenen anhängen.
      zeichenInput += einByte;
    }
  }
}
```


Quellcode des Programms ArduSmartPilot:

```
// Braucht man wegen der onCreate Callbackfunktion.
import android.os.Bundle;

// Ketai Bibliothek für die BT-Kommunikation.
// Quasi der Ersatz für die Serial Bibliothek
import ketai.net.bluetooth.*;
// Bibliothek für das Abfragen der Sensoren des Androidgeräts
import ketai.sensors.*;

// Konstanten für die Positionen der Buttons für Motorsteuerung
final static int XPOSBUT = 200, XSIZBUT = 100, YFIRBUT = 10, YDELBUT = 95, YSIZBUT = 80;
boolean bReleased = true; // Damit nur gesendet wird, wenn der Finger den Bildschirm wieder verlässt nach einer
Berührung.
// String für Rückmeldung des Arduino
String echoArduino = "";
// Datenstring mit den Servowerten, der als Byte Array konvertiert über die Schnittstelle gesendet wird.
String datenString = "";
boolean dataEnd = false; // Flag, ob String komplett angekommen, da String von Arduino manchmal häppchenweise vom BT-
Modul übertragen wird


KetaiBluetooth bt;
KetaiSensor sensor;

// In diesen Variablen werden später die x-,y- unc z-Beschleunigungswerte eingetragen.
// Initialisierung mit Wert 0, damit auch ohne SensorEvent Kreis gezeichnet wird.
float beschleunigungX=0, beschleunigungY=0, beschleunigungZ=0;

int seitenRuder = 90, hoehenRuder = 90, motorLeistung = 0;  // Servowinkel in Grad, Motorleistung in %

String feedbackArdu = "", sensorArdu = "";

//*********************************************************************
// Dieser Quelltext wird für die Organisation der BT-Kommunikation gebraucht
//*********************************************************************
void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  bt = new KetaiBluetooth(this);  // Konstruktor
}

//*********************************************************************
// Jetzt kommt der eigentliche Quelltext!
//*********************************************************************
```

```
void setup()
{
  // Wenn Orientierung im manifest.xml festgelegt, dann diesen Befehl auskommentieren
  // orientation(LANDSCAPE);
  bt.start(); // BT Service starten.
  //************************************************************
  // Hier muss die individuelle Adresse deines BT Moduls eingegeben werden.
  bt.connectDevice("00:13:01:06:20:83"); // HC-05 BT-Modul "SIA-Flug-2"
  //bt.connectDevice("00:11:12:11:04:73"); // BT-Shield Arduino
  //************************************************************
  delay(2000);
  size(800, 480);

  sensor = new KetaiSensor(this);
  sensor.start();
  frameRate(20);
  echoArduino = new String();
}

void draw()
{
  background(255, 255, 0);

  //Variablen für x- bzw. y-Position des Kreises
  int x=0, y=0;
  //Variablen für Kippwinkel um y- bzw. x-Achse
  int kippWinkelY=0, kippWinkelX=0;

  // Fadenkreuz in die Mitte des xy-Bildschirms zeichnen
  strokeWeight(2);
  line(550, 20, 550, 460);
  line(320, 240, 780, 240);

  //Trigonometrische Berechnung der Kippwinkel in Grad, mit (int) Umwandlung von float → int.
  kippWinkelX = (int)(atan(beschleunigungY/beschleunigungZ)*180/PI); //Kippen nach Rechts -> pos. Winkel
  kippWinkelY = (int)(atan(beschleunigungX/beschleunigungZ)*180/PI); //Kippen zum Betrachter hin -> pos. Winkel
  //Umrechnen der Kippwinkel in xy-Koordinatenpositionen
  x = (int)map(kippWinkelX, -90, 90, 320, 780);
  y = (int)map(kippWinkelY, -90, 90, 20, 460);
  //Kreis an der Position (x,y) zeichnen
  fill(255, 0, 0);
  ellipse(x, y, 30, 30);
  fill(0, 0, 0);
```

```
if (mousePressed)
  //if ((mousePressed) && (bReleased == true)) //Wenn auf Bildschirm getippt wird
{
  // Wenn x-Position des Klicks in der im Bereich der Buttons
  if ((mouseX>XPOSBUT)&&(mouseX<XPOSBUT+XSIZBUT))
  {
    // Je nach y-Position des Klicks, Auswahl Motorleisung und damit Button
    if ((mouseY>YFIRBUT+(0*YDELBUT))&&(mouseY<YFIRBUT+(1*YSIZBUT)))  motorLeistung = 100;
    else if ((mouseY>YFIRBUT+(1*YDELBUT))&&(mouseY<YFIRBUT+(1*YDELBUT)+YSIZBUT))  motorLeistung = 75;
    else if ((mouseY>YFIRBUT+(2*YDELBUT))&&(mouseY<YFIRBUT+(2*YDELBUT)+YSIZBUT))  motorLeistung = 50;
    else if ((mouseY>YFIRBUT+(3*YDELBUT))&&(mouseY<YFIRBUT+(3*YDELBUT)+YSIZBUT))  motorLeistung = 25;
    else if ((mouseY>YFIRBUT+(4*YDELBUT))&&(mouseY<YFIRBUT+(4*YDELBUT)+YSIZBUT))  motorLeistung = 0;
  }
}

// Alle Buttons in Weiß darstellen
fill(255, 255, 255);
rect(XPOSBUT, YFIRBUT+ (0*YDELBUT), XSIZBUT, YSIZBUT);
rect(XPOSBUT, YFIRBUT+ (1*YDELBUT), XSIZBUT, YSIZBUT);
rect(XPOSBUT, YFIRBUT+ (2*YDELBUT), XSIZBUT, YSIZBUT);
rect(XPOSBUT, YFIRBUT+ (3*YDELBUT), XSIZBUT, YSIZBUT);
rect(XPOSBUT, YFIRBUT+ (4*YDELBUT), XSIZBUT, YSIZBUT);

// Zuletzt geklickter Button (= Wert Motorleisung) grün darstellen
fill (0, 255, 0);
switch(motorLeistung)
{
case 100:
  rect(XPOSBUT, YFIRBUT+ (0*YDELBUT), XSIZBUT, YSIZBUT);
  break;
case 75:
  rect(XPOSBUT, YFIRBUT+ (1*YDELBUT), XSIZBUT, YSIZBUT);
  break;
case 50:
  rect(XPOSBUT, YFIRBUT+ (2*YDELBUT), XSIZBUT, YSIZBUT);
  break;
case 25:
  rect(XPOSBUT, YFIRBUT+ (3*YDELBUT), XSIZBUT, YSIZBUT);
  break;
case 0:
  rect(XPOSBUT, YFIRBUT+ (4*YDELBUT), XSIZBUT, YSIZBUT);
```

```
    break;
  }

  // Buttons mit Wert Motorleistung beschriften.
  textSize(24);
  fill(0, 0, 0);
  text("100 %", XPOSBUT+20, YFIRBUT+(0*YDELBUT)+50);
  text("75 %", XPOSBUT+20, YFIRBUT+(1*YDELBUT)+50);
  text("50 %", XPOSBUT+20, YFIRBUT+(2*YDELBUT)+50);
  text("25 %", XPOSBUT+20, YFIRBUT+(3*YDELBUT)+50);
  text("0 %", XPOSBUT+20, YFIRBUT+(4*YDELBUT)+50);

  // Anzeige Rückmeldung Arduino Zahl empfangener Werte
  text("Feedback", 50, 100);
  text("Arduino:", 50, 100+30);
  text(feedbackArdu, 50, 100+60);

  // Anzeige Rückmeldung Arduino Sensorwert
  text("Sensor-", 50, 300);
  text("wert:", 50, 300+30);
  text(sensorArdu, 50, 300+60);

  // Kippwinkel auf Bereich 0...180 Grad übertragen
  // Umrechnen in nötige in Seiten-/Höhenruderwerte (=Servowinkel) im Arduino.
  seitenRuder = (int)map(kippWinkelX, -90, 90, 0, 180);
  hoehenRuder = (int)map(kippWinkelY, -90, 90, 0, 180);

  // Der Datenstring wird nur dazu benutzt um die Zeichenkette mit den Kippwinkel und der Motorleistung zusammen zu
  // bauen und dann anschließend in einen Byte Array zu konvertieren.
  datenString += seitenRuder + "," + hoehenRuder + "," + motorLeistung +"\n"; // String erstellen mit den Servowinkeln

  // Datenstring elementweise (d.h.jedes char) in ein byte umwandeln. Da für die ASCII Zeichen nur
  // das erste Byte genutzt wird, geht dabei keine Information verloren. Der resultierende Byte Array
  // hat die gleiche "Länge" wie der String.
  byte[] daten = new byte[datenString.length()];
  for (int i=0; i<datenString.length (); i++) {
    daten[i] = byte(datenString.charAt(i));
  }
  bt.broadcast(daten); //Daten über BT senden. Da diese Methode nur bytes versendet, musste man vorher den string in ein
byte-Array umwandeln.
  delay(10);
  datenString="";
}
```

```
void onBluetoothDataEvent(String who, byte[] data) //Callback Methode: Es liegen Daten vom Arduino an der BT-
Schnittstelle an.
{
  //Falls noch kein \n empfangen wurde, String vom Arduino also nicht nicht komplett empfangen.
  if (!dataEnd) {
    //Byte Array elementweise in Char umwandeln und dem String hinzufügen.
    for (int i=0; i<data.length; i++ ) {
      echoArduino +=String.valueOf((char)data[i]);
    }
  }

  //Wenn das letzte Zeichen ein \n war, ist String vom Arduino komplett empfangen worden.
  if ((char)data[data.length-1]==&apos;\n&apos;) dataEnd = true;

  //Falls String vom Arduino komplett empfangen worden.
  if (dataEnd) {
  feedbackArdu = split(echoArduino, &apos;,&apos;)[0]; // Substring vor dem ersten Komma heraustrennen
  sensorArdu = split(echoArduino, &apos;,&apos;)[1];  // Substring zwischen ersten und zweiten Komma heraustrennen
  echoArduino=""; // Stingpufer zurück setzen.
    dataEnd = false; // Empfang des nächsten Strings ermöglichen.
  }
}

//Diese Methode wird jedes Mal automatisch aufgerufen, wenn sich ein
//Sensorwert geändert hat.
void onAccelerometerEvent(float x, float y, float z)
{
  beschleunigungX = -x;
  beschleunigungY = -y;
  beschleunigungZ = -z;
}
```