2. Versuchsreihe

In Versuch 2 werden verschiedene Sensoren und Aktoren von eingebetteten Systemen näher betrachtet. Hierbei lernen die Studierenden, welche Sensoren und Aktoren in aktuellen Smartphones vorkommen und wie man diese in eine App einbinden kann. Zudem werden in der Durchführungsphase bereits die ersten Sensoren aus dem Modellhaus verwendet. Hierbei lernen die Studierenden, dass für die richtige Verwendung von Sensoren und Aktoren in einem eingebetteten System verschiedene Randbedingungen beachtet werden müssen.

Kompetenzen

Die Studierenden

- können abrufen, welche Sensoren und Aktoren in aktuellen Smartphones vorkommen und verstehen grundlegend deren Funktionsweise,
- verstehen wie man eine eigene App unter Android entwickelt, welche Sensoren und Aktoren eines Smartphones verwendet.
- erkennen verschiedenen Randbedingungen, die beim Einsatz von Sensoren und Aktoren berücksichtigt werden müssen.
- können einfache elektrische Aktor- und Sensor-Schaltungen aufbauen und testen.
- können englischsprachige Datenblätter eines Sensors zu lesen und zu verstehen.

2.1. Vorbereitung

1. 🕮 Lesen Sie sich zur Vorbereitung den Anhang *Sensoren und Aktoren von einem Smartphone* durch.
2. 💻 Alle Smartphone-Besitzer finden unter Moodle oder auf der Homepage der DIE[1] die App ASV (Android Sensor Visualisation), die grob die wichtigsten Sensoren und Aktoren eine Smartphones erklärt. Installieren und testen Sie die App. Wenn Sie kein Smartphone besitzen, überspringen Sie diese Aufgabe.
3. 💻 Packen Sie Ihr Projekt ResCalc{*Gruppennummer*} aus *Versuchsreihe 1* zu einem Archiv. Hierfür können Sie den *File Browser pcmanfm* Ihrer virtuellen Maschine benutzen. Den File Browser finden Sie im Menü unter *Browser*. Schicken Sie Ihr fertiges, kommentiertes, gepacktes Projekt einmal an folgende E-Mail-Adresse: *andre.schaefer@uni-siegen.de*.
   Betreff: [Hapra] – {*Aufgabe*} + {*Gruppennummer*}

[1] http://www.die.informatik.uni-siegen.de/index.php/pgandroid.html

4. Arbeiten Sie das Tutorial 3: *Kommunikation zwischen Aktivities und mit Sensoren, DDMS* einmal durch.

5. ✎ Sie sollen für **Versuchsreihe 3** eine App programmieren, mit der Sie das Computerspiel *Pong*[2] spielen können.
   Pong war eines der ersten Computerspiele und ähnelt Tennis. Jeder Spieler steuerte mit einem Joystick einen Schläger am Bildschirmrand und musste mit diesem verhindern, dass der Ball in den Bereich hinter dem Schläger gelangte. Ziel des Spieles war, den Ball neben dem Schläger des Gegners aus dem Spielfeld zu schlagen und so einen Punkt zu erzielen und gleichzeitig Punkte des Gegners zu verhindern. In Ihrer Version von Pong sollen Sie einen Schläger mittels der Neigungssensoren steuern.

   Hinweise zur Aufgabenstellung und wie Sie Ihr Programm ohne Smartphone testen können, finden Sie im Anhang *Pong und der SensorSimulator*. Installieren und testen Sie den SensorSimlulator. Importieren Sie das Pong-Projekt. Ändern Sie den Namen Ihrer *Pong-App* gemäß der Tabelle 2.1.

Project name	Pong{*Gruppenummer*}
Build Target	Android 2.2
Application name	Pong{*Gruppenummer*}
Package name	hapra.pong{*Gruppenummer*}
Create Activity	PongActivity
Min SDK Version	8

Tabelle 2.1.: Bezeichner

   Nachdem Sie das Projekt gemäß der Beschreibung im Anhang installiert haben, finden Sie in Ihrem Projekt unter anderem die Klasse *Pong* mit folgendem Code:

```
1  //------------ Paddle 2 STeuerung ----
2  Paddle p2 = (Paddle) elements.get(1);
3  coord = p2.getCoordinates();
4  speed = p2.getSpeed();
5  speed.setX(5);
6  if (speed.getXDirection() == Speed.X_DIRECTION_RIGHT) {
7          coord.setX(coord.getX() + speed.getX());
8  } else {
9          coord.setX(coord.getX() - speed.getX());
10 }
11
12 if (coord.getX() < 0) {
13         speed.toggleXDirection();
14         coord.setX(-coord.getX());
15 } else if (coord.getX() + p2.getGraphic().getWidth() >
      getWidth()) {
16         speed.toggleXDirection();
17         coord.setX(coord.getX() + getWidth()
18         - (coord.getX() + p2.getGraphic().getWidth()));
```

[2] http://de.wikipedia.org/wiki/Pong

```
19 }
```

Listing 2.1: Steuerung des Paddle 2

Beantworten Sie folgende Fragen:

- Was ist Paddle 2?
- Wie wird Paddle 2 anhand des Listings 2.1 gesteuert?
- Wer steuert Paddle 2?

6. 🕮 Informieren Sie sich selbständig über Dioden, ihre Funktionsweise sowie Kennlinien und den P-N-Übergang einer Diode.
7. 🕮 Wiederholen Sie zur Vorbereitung im Buch *Entwurf digitaler Systeme* das erste Kapitel *Verbindungsleitungen*. Frischen Sie zudem Ihr Wissen bezüglich eines Spannungsteilers auf. Wofür wird ein Spannungsteiler verwendet?
8. ✎ Was ist ein Diode und wie verhält sich der Strom bei einer Diode?
9. ✎ Was ist der Unterschied zwischen einer:
   - Zenerdiode
   - Photodiode
   - Leuchtdiode
10. ✎ Wofür werden üblicherweise welche Dioden eingesetzt?
11. ✎ Mit einem Oszilloskop lassen sich nur Spannungen messen, es soll aber die Strom-Spannungs-Kennlinie einer LED mit dem Oszilloskop dargestellt werden. Wie erhält man diese Darstellung und mit welcher Skalierung?
12. ✎ Das Bild 2.1 zeigt den Versuchsaufbau zur Aufzeichnung der Kennlinie einer LED. Das Oszilloskop zum Aufzeichnen der Kennlinie wird zwischen Messpunkt 1 und Messpunkt 2 angelegt. Als Spannungsquelle (~U) dient eine Sinusspannung. Wozu dient der Widerstand R? Wie groß ist R zu wählen?

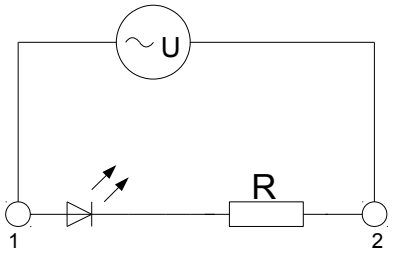


Abbildung 2.1.: Einfache LED-Schaltung zum aufzeichen der Kennlinie

13. ✎ Zum Aufzeichnen der Kennlinie wird das Oszilloskop im X-Y Betrieb genutzt. D.h., dass die X-Achse nicht mehr die Zeit, sondern einen der beiden Kanäle darstellt.
    a) Welcher Kanal wird dabei auf der X-Achse dargestellt, welcher auf der Y-Achse?
    b) Auf welcher Achse wird bei einer Strom-Spannungs-Kennlinie der Strom dar-

gestellt?

c) Was zeigt die Arbeitsgerade in einer Strom-Spannungs-Kennlinie ?

14. ✎ In unserem Modellhaus werden der Helligkeitssensor und die beiden Temperatursensoren über eine Versorgungsspannung von 5V mit Strom versorgt. In Abhängigkeit der Helligkeit, der Temperatur bzw. der Luftfeuchtigkeit ändert sich jeweils der Widerstand des betreffenden Sensors. In Tabelle 2.2 sehen Sie, wie sich der Widerstand beim Helligkeitssensor (VT93N1) in Abhängigkeit der Helligkeit verändert. Zeichnen Sie einen Spannungsteiler, bei dem Sie zwischen einem Widerstand R1 und dem Helligkeitssensor die Ausgangsspannung abnehmen. Wie hoch muss der Widerstand R1 gewählt sein, damit die Ausgansspannung in Abhängigkeit der Helligkeit möglichst gleichmäßig zwischen 0 und 5V variiert?

Helligkeit in *fc*	**Widerstand in *Ohm***
1	20000
2	12000
3	7000
4	6000
5	5000
6	4000
7	3400
8	3200
9	2900
10	2700
20	1400
30	1100
40	800
50	700
60	600
70	500
80	400

Tabelle 2.2.: Kennwerte für den Helligkeitssensor

15. ✎ Ähnlich wie in Tabelle 2.2 finden Sie auch Widerstandswerte zum Temperatursensor (KTY81-222) bzw. zum Feuchtigkeitssensor (GY-HR002) in den Datenblättern zum jeweiligen Sensor. (Die Datenblätter finden Sie unter Moodle zur Versuchsreihe 2). Bestimmen Sie auch hier, wie groß der Widerstand R2 für den Temperatursensor und der Widerstand R3 für den Feuchtigkeitssensor sein sollte. Was ist bei der Ein- und Ausgangsspannung des Feuchtigkeitssensor (GY-HR002) zu beachten?

16. Beantworten Sie die Fragen des 2. Moodle-Selbsttestes. Sollten Sie bei der Beantwortung Probleme haben, müssen Sie ggf. diesen Teil der Vorbereitung nochmal wiederholen. Bis zur Laborphase müssen Sie den Selbsttest wenigstens einmal mit dem Ergebnis ausreichend abgeschlossen haben.

2.2. Durchführung

1. Bauen Sie für den Helligkeitssensor, Feuchtigkeitssensor und Temperatursensor Ihre vorbereiteten Schaltungen aus Aufgabe 14 und 15 auf dem Steckbrett auf. Überprüfen Sie grob, ob die Sensoren im Rahmen ihrer Spezifikation funktionieren.

2. ✎ Bestimmen Sie anhand ihrer Schaltung und mit Hilfe des Datenblattes die Temperatur und Luftfeuchtigkeit im Laborraum. Halten Sie fürs Protokoll die Ergebnisse fest. Notieren Sie fürs Protokoll auch, wie Sie diese Ergebnisse bestimmt haben.

3. ✎ Schauen Sie sich das Modellhaus an. Wie groß sind die Widerstände R1, R2 und R3 hier gewählt?

4. ✎ Zur Bestimmung der Kennlinie einer LED bauen Sie die Schaltung aus Bild 2.1 auf dem Steckbrett auf. Der Funktionsgenerator liefert eine Sinusspannung zur Ansteuerung der LED. Als Funktionsgenerator dient der HAMEG HM8030. Stellen Sie die Amplitude auf den kleinsten Wert (min) und schalten Sie den Offset ab. Schließen Sie das Oszilloskop nach Bild 2.1 an die jeweiligen Kanäle an, und wählen Sie den X-Y-Modus.

   HINWEIS: Das Oszilloskop eignet sich sehr gut zur Darstellung funktionaler Abhängigkeiten, da der Elektronenstrahl sowohl in vertikaler als auch in horizontaler Richtung gelenkt werden kann und die beiden Ablenkungen voneinander unabhängig sind. In der X-Y Betriebsart werden dem Oszilloskop beide Ablenksignale von außen zugeführt, das Y-Ablenksignal über den Kanal I und das X-Ablenkungssignal über den Kanal II. Bei der Benutzung dieser Betriebsart muss darauf geachtet werden, dass zu jedem Zeitpunkt mindestens eine der beiden Ablenkspannungen genügend groß oder die Strahlintensität auf null gedreht ist, damit die Leuchtschicht der Röhre nicht lokal überlastet wird (einbrennt).

   a) Was ist auf dem Oszilloskop zu erkennen?

      Positionieren Sie das Signal mithilfe der “X-Pos” mittig auf dem Schirm. Steigern Sie nun die Amplitude soweit, bis ein Anstieg des Stroms auf dem Oszilloskop sichtbar wird.

   b) Ab welcher Spannung beginnt die LED zu leuchten?

   c) Wie kann dieser Spannungswert am Oszilloskop abgelesen werden?

   d) Wie groß ist der Strom bei einem Spannungsabfall von 1V am gewählen Widerstand?

   e) Was bedeutet dies für den Strom durch die LED?

   Halten Sie alle Antworten für das Protokoll fest und übertragen Sie die Kennlinie der Diode nach Bild 2.2.

5. ✎ Ersetzen Sie den Funktionsgenerator durch eine Gleichspannungsquelle mit $U = 5V$. Wählen Sie anhand der Kennlinie einen sinnvollen Arbeitspunkt[3] für die

[3] Der Arbeitspunkt ist ein bestimmter Punkt auf der Kennlinie eines technischen

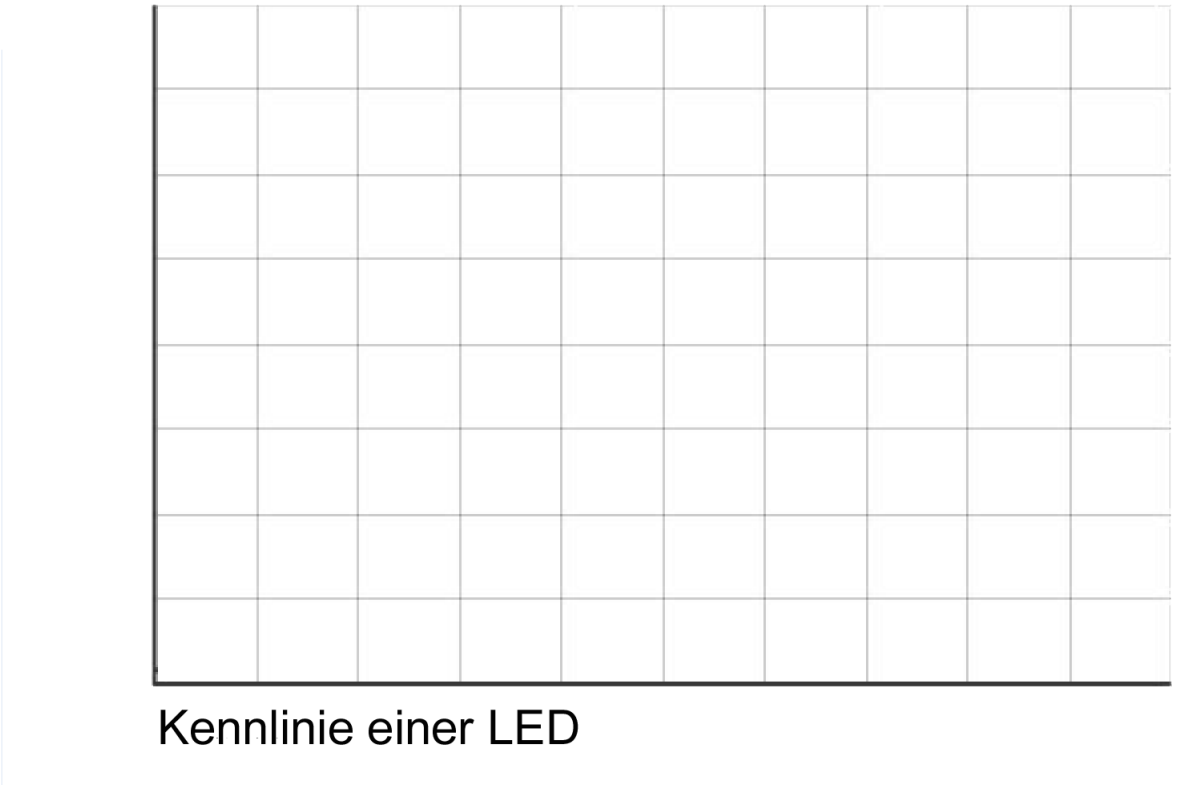


Abbildung 2.2.: Messergebnisse

LED und zeichnen Sie eine Arbeitsgerade des Vorwiderstands in Abbildung 2.2 ein. Welche beiden Punkte muss die Arbeitsgerade kreuzen? Wie kann man daraus den Vorwiderstand bestimmen?

2.3. Nachbereitung

1. ✎ Tragen Sie die Ergebnisse zur Kennlinie einer LED und zu den Sensorschaltungen aus der Vorbereitung und der Laborphase zusammen. Geben Sie diese Lösungen zur nächsten Laborphase ab!

2.4. Anhang

2.4.1. Sensoren und Aktoren von einem Smartphone

Abbildung 2.3 zeigt ein Smartphone (examplarisch das Modell *HTC Desire* , Sensoren und Aktoren sind bei anderen Modellen an ähnlichen Stellen zu finden), auf dem von außen sichtbare Sensoren und Aktoren markiert sind. (1) markiert die Position eines Lautsprechers, (2) und (3) einen Helligkeits- und Näherungssensor. An Position (4) befindet sich eine LED, die je nach Status des Smartphones in verschiedenen Farben leuchten kann.

Gerätes.[http://de.wikipedia.org/wiki/Arbeitspunkt]



Abbildung 2.3.: Das Smartphone *HTC Desire* [www.htc.com])

(5) ist das Display und der Touchscreen, diverse Taster befinden sich an Position (6). (7) markiert die Position eines Mikrofons und auf der Rückseite an Position (8) befindet sich eine Digitalkamera und eine LED, die als Blitzlicht oder Taschenlampe benutzt werden kann. Weitere Sensoren und Aktoren befinden sich im Inneren des Smartphones und sind ohne Demontage nicht zu sehen.

Dieses Kapitel wurde von Herrn Jan Bauerdick im Rahmen seiner Bachelorarbeit geschrieben.

2.4.1.1. Sensoren

Aktuelle Smartphones verfügen über mehrere Sensoren für die verschiedensten Zwecke. Die Android API kann unter anderem auf die folgenden Sensoren zugreifen, die in nahezu jedem Smartphone vorhanden sind: Helligkeitssensor (Lichtsensor), Neigungssensor, Näherungssensor, Beschleunigungssensor, Gyroskop und einen Magnetfeldsensor (Kompass). Allerdings verfügt nicht jedes Smartphone über alle dieser Sensoren. Zu den Sensoren gehören außerdem Touchscreen, Mikrofon, eine Digitalkamera und diverse Taster. Im Folgenden werden die Sensoren, deren Nutzen, ihre Funktionsweise und ihre Verwendung mittels der Android API beschrieben.

Touchscreen
Ein Touchscreen ist das Hauptbedienelement eines Smartphones und ersetzt größtenteils die Tastatur, die bei älteren Telefonen eingesetzt wurde. Der Benutzer tätigt Eingaben durch Berühren des Touchscreens an der Stelle, die eine Aktion auslöst. Das kann zum Beispiel ein Programmstartersymbol, ein Button eines GUI oder die Taste einer Tastatur sein. Touchscreens in Smartphones werden in zwei verschiedenen Techniken eingesetzt.

Resistiver Touchscreen

Ein resistiver Touchscreen (von lat. resistencia: Widerstand) arbeitet mit zwei übereinanderliegenden Schichten. Die einander zugewandten Seiten sind mit Indiumzinnoxid, einem lichtdurchlässigem Halbleiter beschichtet und werden von kleinen Abstandhaltern voneinander getrennt.

Die untere Schicht besteht meist aus Glas oder Kunststoff und wird auf eine stabile, feste Grundfläche aufgetragen, die obere Schicht besteht aus dehnbarem Polyester. Wird der Touchscreen mit dem Finger berührt, verformt sich die Polyesterschicht durch den Druck und die beschichteten Innenseiten kommen zum Kontakt.

Um die Druckstelle zu ermitteln, wird an beiden Schichten abwechselnd eine Gleichspannung angelegt. Die Spannungen sind orthogonal zueinander und fallen gleichmäßig vom einen zum anderen Rand ab. Berühren sich die Schichten fließt zwischen ihnen ein Strom, durch den sich die Spannungen verändern. Aus diesen Spannungsänderungen kann die Druckstelle eindeutig bestimmt werden. Ein resistiver Touchscreen benötigt immer zwei Schichten, von denen jeweils an einer eine Gleichspannung angelegt ist und die andere die Position in eine Richtung ermittelt (siehe Abbildung 2.4).

Resistive Touchscreens sind die Vorreiter der Touch-Technik, lassen sich aber im Allgemeinen nicht mit mehreren Fingern gleichzeitig bedienen (multitouch). Liegen mehrere Finger auf dem Touchscreen, so vergrößert sich die Kontaktfläche und die Finger können nicht einzeln erfasst werden. Ein weiterer groß er Nachteil ist, dass die Polyesterschicht durch das ständige Biegen porös wird und einreißen kann. Dies ändert die elektrischen Eigenschaften des Materials und die Positionsbestimmung wird ungenauer. Vorteile dieser Technik sind die geringen Herstellungskosten und dass sie mit jedem Gegenstand (z.B. einem Stift oder mit Handschuhen) bedient werden kann.[4]

Kapazitiver Touchscreen
Im Gegensatz zur resistiven Technik benötigen kapazitive Touchscreens keinen Druck und keinen Kontakt zweier Schichten. Sie bestehen aus einem zweischichtigen Netz aus Elektroden, die sich als Zeilen und Spalten anordnen. Zwischen den Elektrodenschichten befindet sich ein Dielektrikum. An der Unterseite befindet sich Schaltungen, die jederzeit die Kapazität an den Kreuzungspunkten messen (siehe Abbildung 2.5). Die Oberseite besteht aus einem stabilen Material, meistens Glas, welches die Elektroden schützt.
Da ein Finger elektrisch leitend ist, können die Ladungen abfließen, sobald er das Glas berührt. Dadurch ändert sich das elektrische Feld zwischen den Elektroden und eine änderung der Kapazität wird messbar.
Da die Messung ständig auf der gesamten Fläche ausgeführt wird, können Berührungen einzelner Finger registriert werden. Dadurch sind kapazitive Touchscreens multitouchfähig. Ein weiterer Vorteil gegenüber der resistiven Technik ist der nicht auftretende Verschleiß. Ein großer Nachteil allerdings ist, dass nur Berührungen mit leitenden Gegenständen erfasst werden, eine Bedienung mit einem (nichtleitendem) Stift oder mit Handschuhen ist nicht möglich.footnotehttp://www.weltderphysik.de/thema/hinter-den-dingen/elektronische-geraete/touchpad-und-screen/

Android wertet Berührungen des Touchscreens über ein Koordinatensystem aus, dessen

[4]http://www.weltderphysik.de/thema/hinter-den-dingen/elektronische-geraete/touchpad-und-screen/

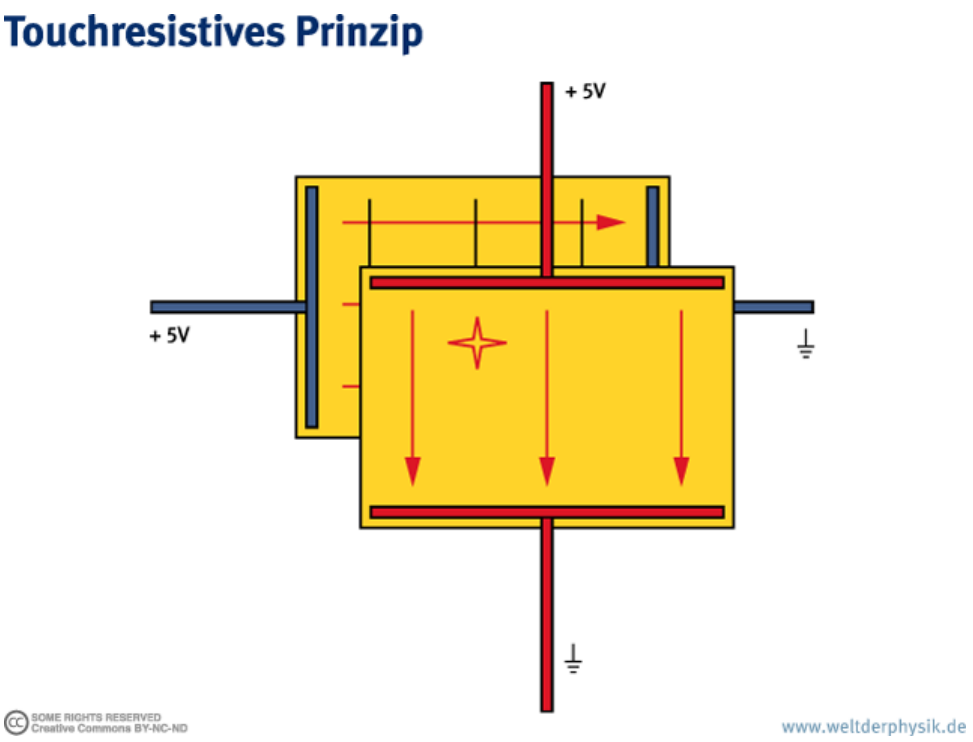


Abbildung 2.4.: Funktionsweise eines resistiven Touchscreens. Die Pfeile zeigen die Richtung der angelegten Spannung. [http://www.weltderphysik.de]

Nullpunkt in der linken oberen Ecke liegt. Die x-Achse verläuft in horizontaler Richtung nach rechts, die y-Achse in vertikaler Richtung nach unten. Die Methode `onTouchEvent` der Klasse `View` ermittelt die Koordinaten der Berührstelle über ein `MotionEvent`. Sollte sich an der Stelle auf dem Display ein GUI-Element befinden, zum Beispiel ein Button oder ein Texteingabefeld, wird automatisch die diesem Element zugeordnete Aktion ausgelöst.

Helligkeitssensor

Helligkeitssensoren messen die Lichtstärke, die auf das Smartphone trifft. Sie werden benutzt, um die Hintergrundbeleuchtung des Displays zu steuern. Bei starkem Lichteinfall wird die Intensität der Beleuchtung erhöht, damit auch bei hoher Helligkeit der Umgebung keine Probleme beim Ablesen des Displays entstehen. Nimmt die Umgebungshelligkeit ab und ist keine starke Hintergrundbeleuchtung mehr nötig, so wird die Beleuchtung gedimmt um Energie zu sparen.
Es gibt zwei Möglichkeiten, einen Helligkeitssensor zu realisieren.

Photowiderstand

Ein Photowiderstand ist ein lichtabhängiger Widerstand (LDR, von engl. Light Dependent Resistor), die Lichtmessung erfolgt über einen Spannungsteiler (siehe Abbildung 2.6). Ein Spannungsteiler besteht aus zwei in Reihe geschalteten Widerständen. Je nach Wert des Widerstandes fällt ein Teil der angelegten Spannung über dem ersten Widerstand ab, der Rest am zweiten. Die Lichtabhängigkeit des LDR macht mit Hilfe dieses Effekts einen Helligkeitsunterschied messbar. Vorteile von LDR sind Einfachheit, Robustheit und der große Dynamikbereich, Nachteile sind die sehr geringe Reaktionsgeschwindigkeit und dass viele beliebte LDR-Typen auf Cadmium, einem giftigen Schwermetall basieren. Der Einsatz

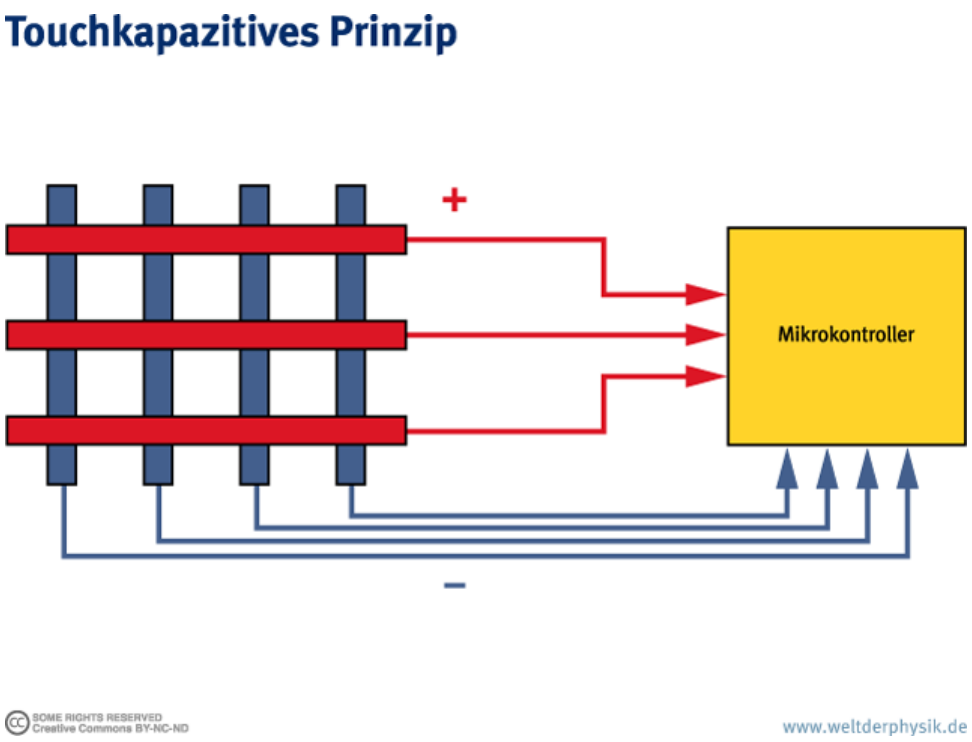


Abbildung 2.5.: Funktionsweise eines kapazitiven Touchscreens. Die roten und blauen Verbindungen bilden das Elektrodennetz, Kreuzungspunkte die Messpunkte der Kapazität. [http://www.weltderphysik.de]

von Cadmium ist nach Elektronikvorschriften verboten.[5]

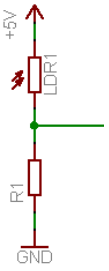


Abbildung 2.6.: Schaltbild Fotowiderstand (oben) als Bestandteil eines Spannungsteilers [www.mikrocontroller.net]

Photohalbleiter

In Photohalbleitern werden innerhalb einer PN-Sperrschicht durch Lichteinfall freie Ladungsträger erzeugt. Man unterscheidet zwischen **Photodioden** und **Phototransistoren** (siehe Abbildung 2.7).
Photodioden haben Schaltzeiten im Nanosekundenbereich, sind also sehr schnell. Allerdings liefern sie nur sehr kleine Photoströme, diese liegen im nA- bis μ A-Bereich.

Photodioden nutzen den photoelektrischen Effekt, der Strahlung, hier vor allem sichtbares

[5] http://www.mikrocontroller.net/articles/Lichtsensor_/_Helligkeitssensor

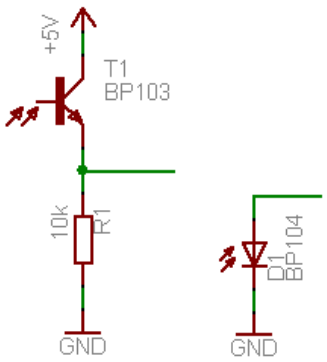


Abbildung 2.7.: Schaltbilder Photohalbleiter (links Phototransistor (oben) als Bestandteil eines Spannungsteilers, rechts Photodiode) [www.mikrocontroller.net]

Licht[6] in einen elektrischen Strom umwandelt.
Phototransistoren haben den Vorteil, dass sie große Photoströme liefern, bzw. schalten können. Die Schaltzeit liegt im Mikrosekundenbereich, also sind sie deutlich langsamer als Photodioden. Phototransistoren werden in Bipolartechnik hergestellt, wobei die Basis-Kollektor-Sperrschicht der Lichtquelle ausgesetzt ist. Sie ähneln einer Photodiode mit Verstärker.[7]

Dieser Sensor gibt in Android den Helligkeitswert in der Einheit lux zurück. Ansprechbar ist er über den `SensorManager` über die Konstante `Sensor.TYPE_LIGHT`.[8]

Neigungssensor
Neigungssensoren messen die Abweichungen der Lage eines Smartphones von einem, beim Kalibrieren des Sensors, festgelegten Koordinatensystems. Mittels der Werte dieses Sensors wird zum Beispiel der Moment ermittelt, in dem das Display vom Hochformat zum Querformat umschalten soll. Mit einer zusätzlichen App kann die Funktionalität des Telefons um eine Wasserwaage erweitert werden. Außerdem existieren weitere Apps, vorrangig Spiele, die diesen Sensor nutzen.
Die Abweichung vom Koordinatensystem kann über ein Feder-Masse-System, welches dem Prinzip des Lots oder Senkbleis ähnelt, bestimmt werden. Neigungssensoren in Smartphones arbeiten mit Flüssigkeiten und nutzen die Reflexion, die Brechung der Flüssigkeit, oder berechnen über Widerstand und Kapazität die Lage.
Der Sensor besteht aus einem Hohlkörper, in dem sich Elektroden befinden, die teilweise von Elektrolyt, einer leitfähigen Flüssigkeit, umspült werden (siehe Abbildung 2.8). Anregungssignale mit verschiedenen Frequenzen und Phasenlagen an einzelnen Elektroden machen die Messung der Neigungsrichtung möglich.
Der Rückgabewert dieses Sensors in Android ist ein Array aus drei Werten, die den Winkel der Neigung in x-, y- und z-Richtung in Grad angeben. Auch dieser Sensor wird über den `SensorManager` mit der Konstante `Sensor.TYPE_ORIENTATION` angesprochen. [9]

Näherungssensor

[6] ca. 400nm - 750nm Wellenlänge
[7] http://www.mikrocontroller.net/articles/Lichtsensor_/_Helligkeitssensor
[8] http://developer.android.com/reference/android/hardware/SensorEvent.html
[9] http://developer.android.com/reference/android/hardware/Camera.html

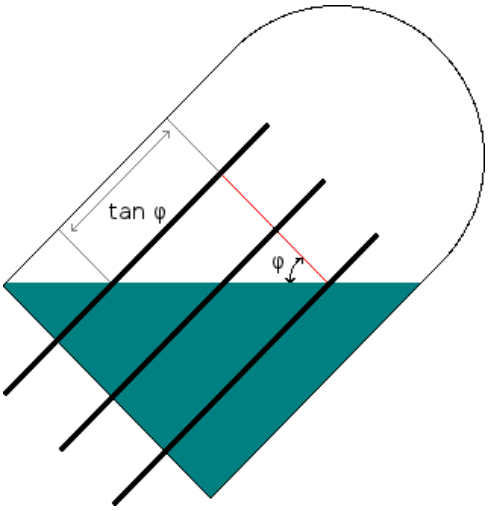


Abbildung 2.8.: Schema eines Neigungssensors

Ein Näherungssensor misst den Abstand eines Objektes vom Smartphone. Bei einem Telefonat wird er benutzt um das Display und den Touchscreen abzuschalten, damit nicht versehentlich Eingaben mit dem Ohr getätigt werden. Er arbeitet mit infrarotem Licht, welches vom sich annähernden Objekt reflektiert wird. Das reflektierte Licht wird von einem Lichtsensor (zum Beispiel dem eingebauten Helligkeitssensor) gemessen und aus diesem die Entfernung berechnet. Aufgrund der unterschiedlichen Reflexionseigenschaften von verschiedenen Materialien ist das Schaltverhalten vom Objekt abhängig.
Da der Sensor mit Licht arbeitet gibt es keine Störungen durch magnetisches Material oder elektromagnetische Felder. Er reagiert allerdings auf Nebel oder Rauch. Verschmutzungen haben oft eine Fehlfunktion oder Ausfall des Sensors zur Folge.[10]
Android liefert bei Abfrage dieses Sensors den Abstand eines Objekts in cm an. Einige Sensoren liefern nur zwei verschiedene Werte, der angibt ob ein Objekt weit entfernt oder nahe ist. Angesprochen wird er über den `SensorManager` über die Konstante `Sensor.TYPE_PROXIMITY`.[11]

Beschleunigungssensor
Beschleunigungssensoren sind in der Lage, beschleunigende Kräfte, die von außen auf das Smartphone wirken, zu messen und deren Richtung zu bestimmen. Mit ihnen ist es möglich, auf Bewegungen des Telefons zu reagieren, was in einigen Apps, vorrangig Spielen, ausgenutzt wird. Zum Beispiel kann in einem Würfelspiel das Schütteln des Würfelbechers imitiert werden.
Außerdem kann auch mit einem Beschleunigungssensor die Lage des Telefon aufgrund der Gravitationskräfte entlang eines im Vorfeld kalibrierten Koordinatensystems bestimmt werden.
Eine Möglichkeit, einen Beschleunigungssensor zu bauen, besteht aus einem Feder-Masse-System, bei dem eine bekannte Masse an einer Feder befestigt ist. Bewegt man dieses System, so wirkt eine Trägheitskraft auf diese Masse, welche die Feder zum Schwingen bringt. Anhang der Schwingungen und der Masse kann man die Beschleunigung und die Richtung des Systems ermitteln.
Der in Smartphones verbaute Sensor arbeitet nach diesem Prinzip. Die Rolle der Feder

[10] http://de.wikipedia.org/wiki/Optischer_N%E4herungsschalter
[11] http://developer.android.com/reference/android/hardware/Camera.html

übernimmt ein sehr feiner Siliziumstab, auch die Masse besteht aus Silizium. Bewegungen der Masse werden von einer festen Elektrode registriert, die aus der Bewegung resultierende Kapazitätsänderungen misst, aus denen wiederum die Beschleunigung und Richtung ermittelt wird.
Da diese Sensoren sehr robust sind und sehr schnell reagieren, werden sie u.a. von der Automobilindustrie in Airbags eingesetzt.
Android hat mehrere Möglichkeiten, die Werte dieses Sensors zu interpretieren. Er kann die Beschleunigung des Smartphones angeben oder die Gravitation, die auf das Smartphone wirkt. In beiden Fällen besteht der Rückgabewert aus einem Array aus drei Werten, die die Beschleunigung oder Gravitation auf die x-, y- und z-Achse des Smartphones angeben, oder die Addition dieser Werte. Seine Konstanten im `SensorManager` sind je nach gewünschten Daten `Sensor.TYPE_ACCELEROMETER`, `Sensor.TYPE_GRAVITY` oder `Sensor.TYPE_LINEAR_ACCELERATION`. [12]

Gyroskop
Ein Gyroskop (von gr. gyros: Drehung und gr. skopein: sehen) ist ein schnell rotierender, symmetrischer Kreisel, der sich in einem beweglichen Lager dreht. Gyroskope werden zur Positions- und Lagebestimmung verwendet.
Ein Gyroskop ist ein geschlossenes System, dessen Drehimpuls konstant bleibt. Wird die Drehachse des Kreisels durch eine äußere Kraft gekippt resultiert ein Drehmoment und die Drehachse kippt senkrecht zu dieser Kraft.
Die folgenden Messprinzipien sind möglich:

1. Die Stabilität der Kreiselachse: Ein frei laufender, symmetrischer Kreisel hat das Bestreben, die Richtung seiner Drehachse im Inertialraum beizubehalten. – Ein Bezug der Lage ist gegeben (Folge der Massenträgheit)

2. Die Präzession: Versucht eine äußere Kraft, die Achsenrichtung eines laufenden Kreisels zu ändern, so folgt die Kreiselachse nicht der Angriffsrichtung dieser Kraft, sondern weicht rechtwinklig zu ihr im Sinne der Kreiseldrehung aus. – äußere Kraft und Präzession stehen in direktem Zusammenhang, eine Lageänderung wird messbar (Folge des Drehimpuls)

Zukünftige Gyroskope in Smartphones könnten mit Laserstrahlen (optisches Gyroskop) arbeiten. Die Rolle des Kreisel übernimmt ein Halbleiterlaser, dessen Licht sich bei Drehung in Wellenlänge und Intensität verändert und durch diese Änderungen eine Orientierung möglich macht. Diese Technik ist 2010 von Prof. J. Scheuer der Tel Aviv Universität in Zusammenarbeit mit dem israelischem Verteidigungsministerium entwickelt worden. Optische Gyroskope sind in viel geringeren Größen herstellbar und passen auf einen Computerchip. Im Gegensatz zu GPS machen Gyroskope die Positionsbestimmung ohne Satellitenkontakt möglich (zum Beispiel in Tunneln), Smartphones brauchen daher keine dauerhafte GPS-Verbindung, um die genaue Position zu bestimmen. Allerdings wird zur Bestimmung der Startposition weiterhin GPS benötigt.[13] [14]
Das Gyroskop liefert in Android ein Array aus drei Werten, die die Winkelgeschwindigkeit um die x-, y- und z-Achsen angeben. Es wird über die Konstante `Sensor.TYPE_GYROSCOPE` angesprochen. [15]

[12] http://developer.android.com/reference/android/hardware/Camera.html
[13] http://www.netzwelt.de/news/84294-smartphones-winziges-gyroskop-koennte-gps-sensoren-ueberfluessig.html
[14] http://de.wikipedia.org/wiki/Kreiselinstrument
[15] http://developer.android.com/reference/android/hardware/SensorEvent.html

Kompass
Ein Kompass misst das Magnetfeld der Erde und zeigt die Ausrichtung bezüglich des magnetischen Nordpols. Er wird in Navigationssystemen verwendet. Es existieren weitere Apps, die diesen Sensor nutzen, zum Beispiel ein trivialer Kompass oder eine Sternenkarte, die mittels Kompassdaten und Daten über die Lage des Smartphones Sterne und Planeten in der aktuellen Blickrichtung anzeigt. In Smartphones kommen zur Messung des Magnetfelds Hall-Sonden zum Einsatz, die das Magnetfeld mittels des Hall-Effekts bestimmen.
Fließt Strom durch ein Halbleiterplättchen, z.B. Indiumarsenid oder Indiumantimonid, das sich in einem Magnetfeld befindet, werden die Elektronen durch die Lorentzkraft zum seitlichen Rand dieses Plättchens abgelenkt (siehe Abbildung 2.9). Ein elektrisches Feld entsteht und damit eine Spannung zwischen den Rändern. Dieses elektrische Feld wirkt der Lorentzkraft entgegen und sorgt dafür, dass ab einer gewissen Spannung keine Elektronen mehr abgelenkt werden und die Spannung konstant bleibt. Diese Spannung wird als *Hall-Spannung* U_H bezeichnet.
Die Hall-Spannung ist abhängig von der Hall-Konstanten R_H, die vom Material abhängig ist, von der Dicke d des Halbleiterplättchens, vom Strom I, der durch das Plättchen fließt und der magnetischen Flussdichte B. Daraus ergibt die Formel $U_H = \frac{R_H B I}{d}$.
Um mit einer Hall-Sonde das Magnetfeld zu messen, wird der Strom konstant gehalten, wodurch die Hall-Spannung nur noch von der magnetischen Flussdichte B abhängig ist. Die Änderung der Hall-Spannung ist proportional zur magnetischen Flussdichte.
Rückgabewert dieses Sensors in Android ist die Stärke des Magnetfelds entlang der x-, y- und z-Achsen in Mikrotesla, seine Konstante ist `Sensor.TYPE_MAGNETIC_FIELD`.[16]

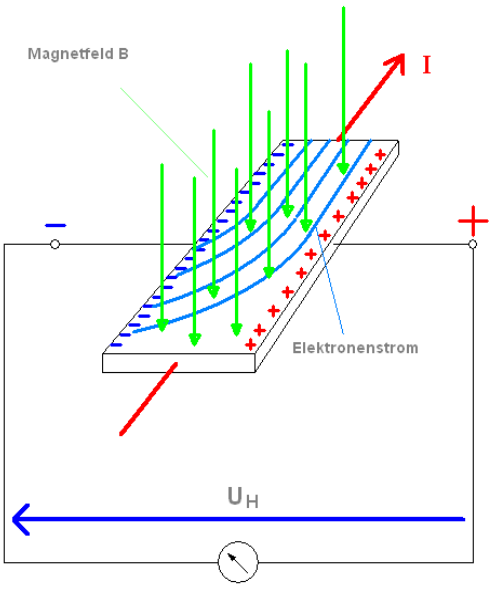


Abbildung 2.9.: Der Hall-Effekt [http://www.elektro-wissen.de]

Taster
Jedes Smartphone verfügt über mindestens einen Taster um das Smartphone einzuschalten, da der Touchscreen im ausgeschalteten Zustand nicht funktioniert. Dieser Taster wird außerdem oft dazu benutzt, das Display bei Nichtbenutzung abzuschalten und wieder einzuschalten. Viele Smartphones haben weitere Taster, die bei Betätigung die Rückkehr zum

[16] http://developer.android.com/reference/android/hardware/SensorEvent.html

Homescreen, das Aufrufen eines Optionsmenüs, die Rückkehr zur zuletzt benutzten App oder das Öffnen eines Suchfeldes bewirken. Weitere Taster ermöglichen die Erhöhung oder Verringerung der Lautstärke der Lautsprecher. Im Gegensatz zu einem Schalter kann man an der Position nicht erkennen, ob ein Taster betätigt worden ist.
Taster sind meist mechanisch arbeitende Elemente, bei dessen Betätigung Kontakte bewegt werden. Berühren sich die Kontakte bei Betätigung nennt man den Taster *Schließer*, trennt er die Verbindung zweier Kontakte nennt man ihn *öffner*.
In Smartphones kommen *Folientaster* zum Einsatz, die statt massiver Metallstücke Druck auf zwei übereinanderliegende Folien ausüben. In diesen Folien sind Leiterbahnen eingearbeitet, die durch den Druck gebogen werden und sich berühren (siehe Abbildung 2.10).

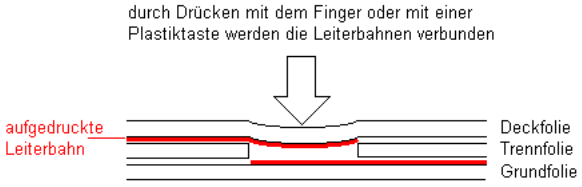


Abbildung 2.10.: Folientaster. Durch Druck auf die Deckfolie werden die beiden Leiterbahnen (rot) verbunden. [Wikipedia]

Mikrofon

Jedes Smartphone verfügt über mindestens ein Mikrofon. Ein Mikrofon wandelt Sprache (Schallwellen) in elektrische Signale, was vor allem während eines Telefonats geschieht. Auch beim Aufzeichnen eines Videos oder einer Sounddatei wird das Mikrofon benutzt. Es kann außerdem benutzt werden, um den Geräuschpegel der Umgebung zu messen. Manche Smartphones verfügen über ein zweites Mikrofon, meist auf der Rückseite, welches Nebengeräusche identifizieren und mit seinen Daten das Filtern dieser Geräusche erleichtert. In Smartphones kommen *Elektretmikrofone* zum Einsatz.
Elektret ist ein dauerelastisches Material, welches durch Elektronenbeschuss während der Herstellung dauerhaft elektrisch geladen wird. Dieses Material wird auf die Sprechmembran aufgebracht und übernimmt die Rolle des Permanentmagneten. Nahe am Elektret befindet sich eine feste Elektrode, die die Rolle der Spule übernimmt. Wenn nun die Membran und damit das Elektret durch Schall in Schwingungen versetzt wird, bewirkt dieses aufgrund der Ladung eine kapazitive Änderung an der festen Elektrode, die von Android weiterverarbeitet wird. [17]
Android ermöglicht es, über die Klasse `MediaRecorder` die Audiosignale des Mikrofons aufzuzeichnen. [18]

Digitalkamera

Mit einer Digitalkamera hat ein Smartphone die Möglichkeit Fotos und Videos aufzunehmen. Das Smartphone kann so eine herkömmliche Digitalkamera in begrenztem Umfang ersetzen. Einige Apps bieten die Möglichkeit Fotos zu analysieren und über eine Verbindung zum Internet weitere Informationen zu erkannten Sehenswürdigkeiten oder Firmenlogos abzurufen. Wenn die Übertragungsgeschwindigkeit ins Internet ausreichend schnell ist, ist mit einer Digitalkamera auch Videotelefonie möglich.

[17] http://www.itwissen.info/
[18] http://developer.android.com/reference/android/media/MediaRecorder.html

Im Gegensatz zu normalen Digitalkameras, die mit CCD-Sensoren arbeiten, werden bei Smartphones wegen des geringeren Stromverbrauchs und höheren Geschwindigkeit CMOS-Bildsensoren eingesetzt. CMOS-Sensoren sind außerdem weniger empfindlich für Bildstörungen. CCD-Sensoren arbeiten mit Schieberegistern, die das Bild zeilenweise abarbeiten (siehe Abbildung 2.11). Damit während des Schiebens keine Bildfehler durch eintreffendes Licht

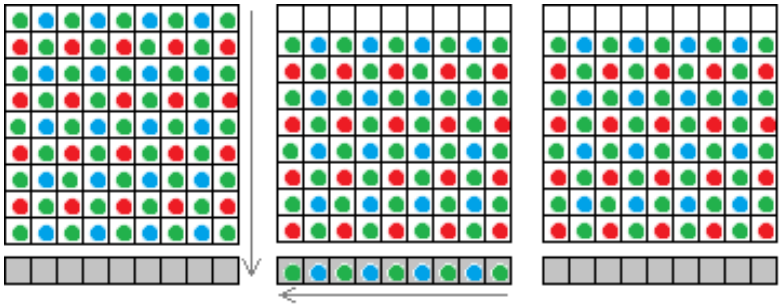


Abbildung 2.11.: Schema eines CCD-Sensors mit Schieberegister, Ablauf von links nach rechts

entstehen muss die Linse für kurze Zeit mechanisch geschlossen werden. Dieser Verschluss ist bei Videoaufnahmen nicht möglich, was in einigen Fällen zu einer hellen, vertikalen Linie im Bild (*Smear*) führt. CMOS-Sensoren verzichten auf die Schieberegister, indem zu jedem Sensor ein Kondensator parallel geschaltet ist, aus dem das Signal direkt verarbeitet werden kann. Bei Videoaufnahmen wird so das Smear verhindert. Aufgrund der Kondensatoren benötigen CMOS-Sensoren jedoch mehr Platz als CCD-Sensoren.
Ein Bildsensor besteht aus einer aus einem Halbleiter hergestellten Matrix von Pixeln, die zur Messung den photoelektrischen Effekt nutzen. Trifft ein Photon auf den Halbleiter, werden durch dieses Elektronen gelöst und bewirkt so einen Strom. Mit diesem Verfahren können nur Helligkeitsunterschiede, aber keine Farbinformationen gespeichert werden. Um Farbbilder zu erhalten, werden vor jedem Pixel rote, grüne und blaue Farbfilter eingesetzt, die nur das Licht in entsprechender Farbe passieren lassen. Jeder Pixel hat nun Informationen über die Helligkeit der Farbe seines Filters. Um daraus ein gewöhnliches Farbbild zu erstellen, muss aus der Information der benachbarten Pixel die ursprüngliche Farbe berechnet (interpoliert) werden. Da das menschliche Auge im grünen Bereich am empfindlichsten ist, werden diese Filter am häufigsten eingesetzt. Eine Zeile besteht aus Filtern für Grün und einer weiteren Farbe, die darunter liegende Zeile aus Filtern für Grün und der anderen Farbe. Die grünen Filter sind dabei jeweils versetzt (siehe Abbildung 2.12). Da dies bei scharfen Farbkanten und feinen Strukturen zu Fehlern führen kann, wird dem Bild durch einen Antialiasingfilter Unschärfe hinzugefügt, welche per Software wieder entfernt wird. [19]

Android unterscheidet zwischen der Foto- und Videoaufnahme. Für Fotografien steht die Klasse `Camera` zur Verfügung, die Aufzeichnung von Videos geschieht über die Klasse `MediaRecorder`. [20] [21]

[19] http://www.convida.at/knowhow/digital/bildsensoren.html
[20] http://developer.android.com/reference/android/hardware/Camera.html
[21] http://developer.android.com/reference/android/media/MediaRecorder.html

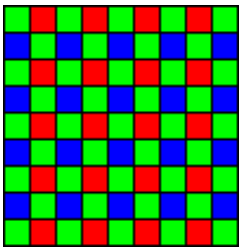


Abbildung 2.12.: Anordnung der Pixel mit Farbfilter. Jeder Kreuzungspunkt entspricht einem Pixel, dessen Farbe aus den vier anliegenden Sensoren interpoliert wird

2.4.1.2. Aktoren

Aktoren geben dem Benutzer Rückmeldung über verschiedene Ereignisse. Einer der wichtigsten Aktoren eines Smartphones ist das Display, welches dem Benutzer momentan benötigte Informationen anzeigt. Ein weiterer wichtiger Aktor ist ein Lautsprecher, der vor allem für die Telefonie unverzichtbar ist. Smartphones verfügen außerdem über eine Vibrationsfunktion und diverse LED, die zur allgemeinen Information, zur Beleuchtung von Tastern oder als Blitzlicht für die Digitalkamera verwendet werden. Im Folgenden werden diese Aktoren, ihre Funktionsweise, Nutzen und ihr Verwendungszweck beschrieben.

LED
LEDs (engl. light-emitting diode, dt. lichtemittierende Diode / Leuchtdiode) werden eingesetzt, um dem Benutzer eines Smartphone eingetretene Ereignisse zu signalisieren. So wird zum Beispiel ein Blinken einer LED genutzt, falls eine neue SMS/MMS oder E-Mail eingetroffen ist. Diese LED kann außerdem geringe Restkapazität des Akkus, einen Ladevorgang und den Abschluss des Ladevorgangs signalisieren. Oft wird auf der Rückseite in der Nähe der Digitalkamera eine weitere LED eingesetzt, die die Funktion eines Blitzlichtes übernimmt. LEDs können von jeder App gesteuert werden. So ist es zum Beispiel möglich, durch Einschalten einer LED das Telefon als Taschenlampe zu nutzen.
Eine Diode besteht aus einem p-dotierten und einem n-dotierten Halbleiter. An der Schnittstelle zwischen p- und n-Dotierung (p-n-Übergang) bewegen sich Elektronen der n-Dotierung in die Löcher der p-Dotierung, wodurch beide Halbleiter nicht mehr elektrisch neutral sind. Ein elektrisches Feld (Raumladungszone, RLZ) entsteht (siehe Abbildung 2.13). Dieses Feld bewirkt eine Spannung am p-n-übergang, die verhindert, dass sich weitere Elektronen zur anderen Seite bewegen können. Es entsteht eine Sperrschicht.
Die Sperrschicht vergrößert sich, wenn eine Spannung von außen mit Pluspol am n-Leiter und Minuspol am p-Leiter angelegt wird (Sperrrichtung). Diese Richtung wird Sperrrichtung genannt. Der Widerstand der Diode wird größer und es fließt nur ein geringer Sperrstrom. Legt man eine Spannung von außen mit Pluspol an den p-Leiter und Minuspol an den n-Leiter (Durchlassrichtung) nimmt die Dicke der Sperrschicht ab. Der Widerstand verkleinert sich und ein Durchlassstrom fließt.
Bei LEDs ist die p-Schicht im Vergleich zur n-Schicht sehr klein. Legt man von außen eine Spannung in Durchlassrichtung an und fließt der Durchlassstrom, rekombinieren die Elektronen in den Löchern, wobei sie Photonen emittieren. Diese können die dünne p-Schicht

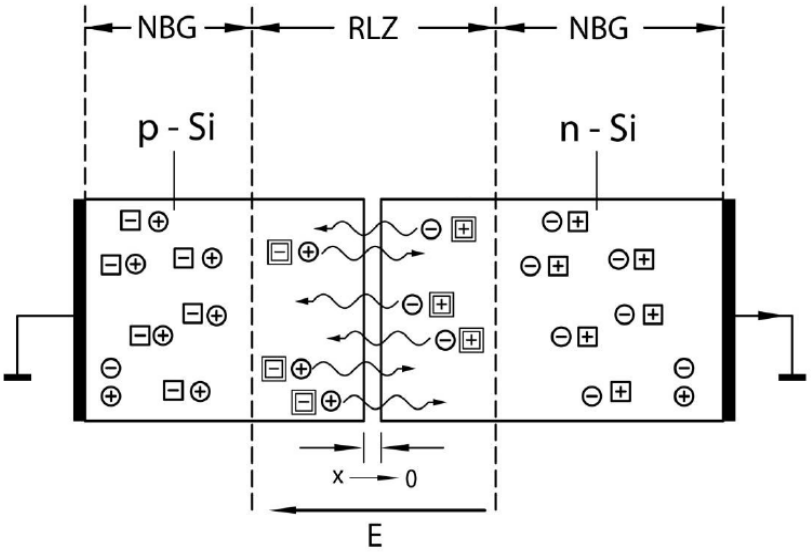


Abbildung 2.13.: Der p-n-Übergang [Skript: Grundlagen der Halbleiterphysik, Böhm]

durchdringen und sind als sichtbares Licht[22] wahrnehmbar. Dieses Licht kann durch Linsen gebündelt werden und so mit geringem Strom im mA-Bereich große Leuchtkräfte erzielt werden. Die Farbe der LED (die Wellenlänge des emittierten Photons) ist abhängig vom gewählten Halbleiter. [23] [24]
Die Steuerung der LEDs ermöglicht die Klasse `NotificationManager`. [25]

Display
Das Display ist einer der wichtigsten Aktoren eines Smartphones. Auf ihm werden alle Informationen angezeigt, die der Benutzer zur Interaktion braucht. Das Display arbeitet eng mit dem Touchscreen zusammen.
Displays in den meisten aktuellen Smartphones nutzen die AMOLED-Technologie (Active Matrix Organic Light Emitting Diode). Hier werden alle Pixel über eine aktive Matrix gesteuert. Dabei hat jeder einzelne Pixel seine eigene Stromversorgung, die ihn zum Leuchten bringt. Im Gegensatz zur älteren LCD-Technik wird dabei keine Hintergrundbeleuchtung benötigt und der Kontrast verbessert sich erheblich. Durch die fehlende Beleuchtung wird der Energieverbrauch verringert.
Bei Aktiv-Matrix Displays sind die Pixel in einer Matrix angeordnet, wobei jeder Pixel über einen aktiven Verstärker und eine eigene Stromversorgung verfügt (siehe Abbildung 2.14). Der Verstärker erfüllt zwei wichtige Aufgaben:

1. Mit Zunahme der Zeilen und Spalten einer Pixelmatrix sinkt die Steuerspannung für jeden Pixel. Passive Displays sind daher in der Größe begrenzt. Ein Verstärker kann die Steuerspannung für jeden Pixel auf einen ausreichend hohen Wert erhöhen.

2. Die Zunahme der Zeilen und Spalten und gleichzeitige Abnahme der Pixelgröße bewirkt, dass sich die parasitären Kapazitäten eines Pixels stetig erhöhen. Diese Ka-

[22] ca. 400nm - 750nm Wellenlänge
[23] Prof. Dr.-Ing. M. Böhm. Grundlagen der Halbleiterphysik. Universität Siegen, 2011.
[24] http://www.led-info.de/
[25] http://developer.android.com/reference/android/app/NotificationManager.html

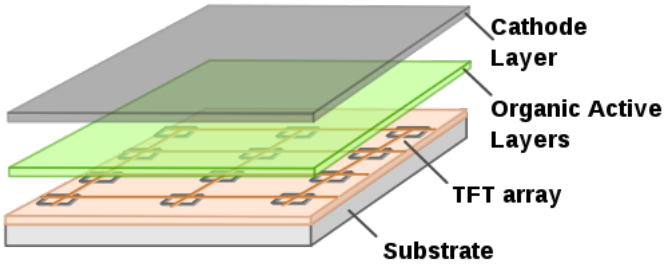


Abbildung 2.14.: Schema eines AMOLED-Displays mit aktiver Matrix (TFT-Array) und LEDs, bestehend aus Organic Active Layers und Cathode Layer [Wikipedia]

pazität erschwert das schnelle Umschalten, was für eine hohe Bildfrequenz nötig ist. Ein Verstärker garantiert mit kurzen, starken Impulsen die schnelle Umschaltung.

Durch die Lösung dieser beiden Aufgaben sind Aktiv-Matrix-Displays die einzigen, die die Wünsche nach immer größeren Displays bei hoher Auflösung erfüllen können. [26]

Vibrationsalarm

Der Vibrationsalarm dient dazu, dem Nutzer lautlos ein eingetretenes (meist konfigurierbares) Ereignis zu signalisieren. Statt eines Ruftons oder in Ergänzung zu diesem löst das Smartphone eine Vibration aus, wenn in der Umgebung ein Rufton nicht gewünscht ist oder überhört werden könnte. Da der Vibrationsalarm von jeder App steuerbar ist, benutzen viele weitere Apps diese Funktion.
Erzeugt wird die Vibration meistens durch einen kleinen Motor, der eine Unwucht trägt (siehe Abbildung 2.15). Durch die Massenträgheit beginnt das Gehäuse zu vibrieren. Eine alternative Möglichkeit stellt der Lautsprecher dar, der, wenn er mit niedriger Frequenz betrieben wird, ebenfalls das Gehäuse vibrieren lässt. [27]
Die Klasse `Vibrator` stellt Methoden zur Verfügung um den Vibrationsmotor zu steuern. [28]



Abbildung 2.15.: Motor mit Unwucht von 2005 [Wikipedia]

Lautsprecher

Ein Lautsprecher wandelt elektrische Signale in Schallwellen und ist damit das Gegenstück zum Mikrofon. Er wird vor allem während eines Telefonats genutzt. Der Lautsprecher wird außerdem eingesetzt um Ereignisse, zum Beispiel ein eingehender Anruf oder den Empfang einer Nachricht (E-Mail, SMS/MMS) zu signalisieren. Ein Abspielprogramm kann mit seiner Hilfe Ton aus Sounddateien oder Videos wiedergeben. Viele weitere Apps nutzen

[26] http://en.wikipedia.org/wiki/Active-matrix_OLED
[27] http://de.wikipedia.org/wiki/Taste
[28] http://developer.android.com/reference/android/os/Vibrator.html

den Lautsprecher für unterschiedliche Zwecke.
Grundprinzip eines Lautsprechers ist, dass eine Membran mittels Permanentmagnet und Spule in Schwingungen versetzt wird.
Es existieren zwei verschiedene Formen der Membran.

1. Konus: Hier ist die Membran wie ein Kegel geformt, diese Form wird für den niedrigen Frequenzbereich eingesetzt. Der Konus hat dabei eine große Fläche und schwingt stark. Im der Mitte des Konus ist meistens ein halbkugelförmiger Staubschutz (Staubschutzkalotte) angebracht (siehe Abbildung 2.16). Die Spule ist mit dem spitzen Ende des Konus verbunden.



Abbildung 2.16.: Konuslautsprecher [www.lautsprechershop.de]

2. Kalotte: Bei dieser Form wird auf den Konus verzichtet, der Lautsprecher besteht nur aus der Kalotte, an welcher die Spule direkt angebracht ist. Durch Verzicht auf den Konus ist der Platzbedarf kleiner (siehe Abbildung 2.17). Diese Form wird für die Wiedergabe mittlerer bis hoher Frequenzbereiche eingesetzt.



Abbildung 2.17.: Kalottenlautsprecher [www.itwissen.info]

Die am Konus bzw. an der Kalotte befestigte Spule wird um einen Permanentmagneten gewickelt. Fließt das elektrische Signal durch die Spule, bewirkt diese ein Magnetfeld und wird im Feld des Permanentmagneten vor und zurück bewegt (siehe Abbildung 2.18). Die Schwingung des Konus/der Kalotte wird an Luftmoleküle übertragen und eine Schallwelle entsteht.

2.4.1.3. Netze

Damit Smartphones mit der Außenwelt kommunizieren können, ist eine Verbindung in verschiedene Kommunikationsnetze erforderlich. Bekannte Netze sind die Mobilfunknetze GSM und UMTS, die drahtlose Netzwerkverbindung Wireless LAN sowie das Satellitennavigationssystem GPS. Diese Netze und ihr Aufbau werden im Folgenden beschrieben.

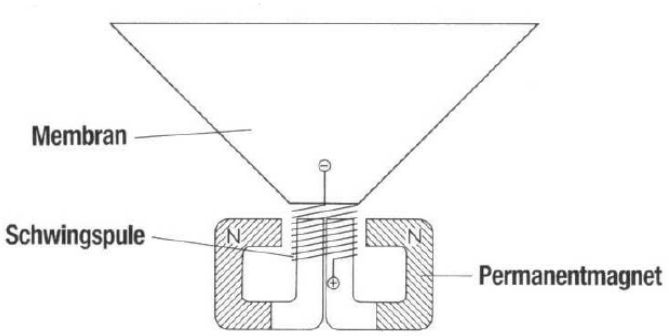


Abbildung 2.18.: Schema eines Konuslautsprechers

GSM/UTMS
GSM (Global System for Mobile Communications) und UMTS (Universal Mobile Telecommunication System) sind Mobilfunknetze, die drahtlose Telefonie und drahtlose Datenübertragung bieten. Smartphones sind dauerhaft mit diesen Netzen verbunden und nutzen diese für ihre wichtigsten Funktionen: Telefonie, den Versand von Kurznachrichten (SMS/MMS) und Zugang zum Internet.

GSM
GSM ist die zweite Generation des Mobilfunknetzes. Es ist aufgeteilt in den D-Netz-Bereich (900MHz) und den E-Netz-Bereich (1800MHz).
Viele Basisstationen (Base Transciever Station, BTS) sorgen für nahezu flächendeckenden Mobilfunkempfang. Eine BTS kann ein Gebiet von 35km Radius versorgen, hat aber nur begrenze Kapazität, weshalb sie in den meisten Fällen dichter beieinander stehen. Jede BTS spannt mit ihren Antennen eine (oder mehrere) Funkzelle(n) auf. Die BTS sind verbunden mit dem Base Station Controller (BSC). Der BSC mit den angeschlossenen BTS bilden ein Base Station Subsystem (BSS) welches mehrere, in den meisten Fällen zwischen 10 und 100 BTS umfasst. Das BSS ist wiederum verbunden mit einer Vermittlungsstelle (Mobile Switching Centre, MSC), welches die Verbindung zum kabelgebundenen Telefonnetz und anderen BSS herstellt, und der Packet Control Unit (PCU), welche die Verbindung zum GPRS Core Network (General Packet Radio Service) bereitstellt. Das GPRS Core Network stellt die Verbindung zum Internet her. Weitere Bestandteile des MSC sind das Home Location Register (HLR), das Authentication Register (AUC) und das Equipment Identity Register (EIR). Die Struktur des Aufbaus zeigt Abbildung 2.19.
Nach dem Einschalten versucht das Smartphone eine Verbindung zu einer BTS herzustellen. über das BSC wird eine Verbindung zum MSC aufgebaut, welches in HLR, AUC und EIR die Daten des Handys prüft, zum Beispiel die SIM-Kartennummer und die IMEI und eine Antwort an die BTS zurückschickt. Bei positiver Antwort baut die BTS eine Verbindung zum Smartphone auf.
Sollte sich ein Smartphone zur Grenze zwischen zwei Funkzellen bewegen, stellt es auf einer anderen Frequenz eine Verbindung zur neuen Funkzelle her und trennt die alte Verbindung (Hard-Handover).
Die Datenübertragungsrate von GSM beträgt 9,6kBit/s, kann aber für die Datenübertragung mit Erweiterungen wie Enhanced Data Rates for GSM Evolution (EDGE) auf 220kBit/s gesteigert werden. [29]

[29] http://www.stefansteiner.net/index.php?page=358

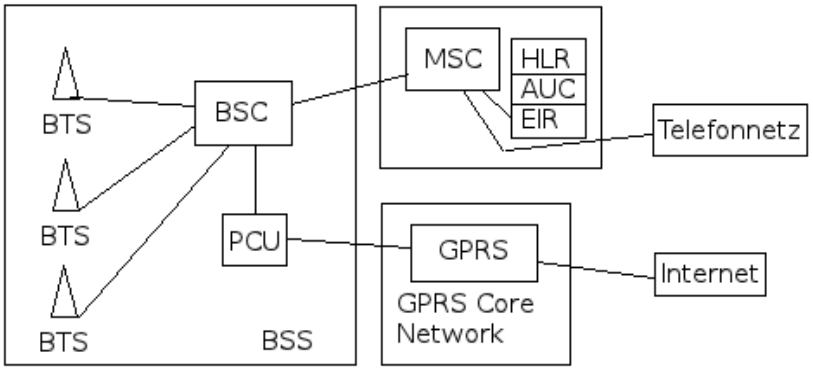


Abbildung 2.19.: Struktur des GSM Mobilfunknetzes

UMTS
UMTS ist die dritte Generation des Mobilfunknetzes und nutzt den Frequenzbereich um 1900MHz. Das Netz ist wie GSM in Funkzellen aufgebaut. Die Basisstationen werden als NodeB bezeichnet, die mit dem Radio Network Controller (RNC) verbunden sind. Das RNC bildet mit den angeschlossenen NodeB ein Radio Network Subsystem (RNS). Das RNS ist mit der 3G-MSC verbunden, welche die Verbindung ins Telefonnetz sowie ins Internet zur Verfügung stellt (siehe Abbildung 2.20). Das GPRS Core Network entfällt bei

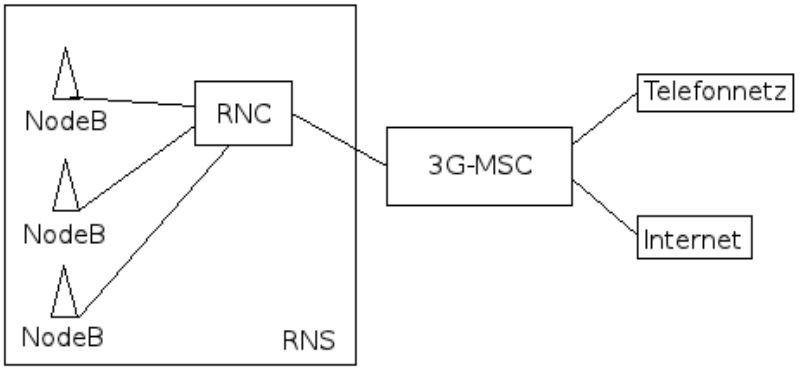


Abbildung 2.20.: Struktur des UMTS Mobilfunknetzes

UMTS.
Der Verbindungsaufbau ist dem von GSM sehr ähnlich, beim Übergang zwischen zwei Funkzellen ist die Verbindung zu mehreren Basisstationen gleichzeitig möglich (Soft-Handover). Im Gegensatz zu GSM konnte die maximale Datenübertragungsrate auf 384kBit/s gesteigert werden, mit Erweiterungen wie High Speed Downlink Packet Access (HSDPA) sogar auf 14,4MBit/s. Durch die höhere Datenübertragungsrate und höhere Frequenz ist die Anzahl der benötigten Basisstationen stark gestiegen. [30]

Wireless LAN
Wireless LAN (W-LAN) ermöglicht die drahtlose übertragung von Daten und/oder die

[30] http://www.stefansteiner.net/index.php?page=365

Teilnahme an einem Netzwerk (Local Area Network, LAN). W-LAN ist von der IEEE in IEEE-802.11 standardisiert. Aktuell eingesetzte Varianten dieses Standards sind 802.11b und 802.11g, die sich in der Datenübertragungsrate und Funkfrequenz unterscheiden. 802.11b nutzt die lizenzfrei nutzbare Frequenz 2.4GHz bei einer Bruttodatenrate von 11MBit/s, 802.11g kann zusätzlich noch das 5GHz Band benutzen und erreicht unabhängig von der Frequenz eine Bruttodatenübertragungsrate von 54MBit/s.
W-LAN unterstützt zwei Betriebsmodi. Im *Infrastruktur*-Modus existiert ein Access Point, zu dem die Endgeräte eine Verbindung aufbauen und darüber meist eine Verbindung zum Internet erhalten. Der Access Point sendet in regelmäßigen Abständen (üblich sind alle 100ms) Beacons, die den Namen des Netzwerks (Service Set Identifier, SSID), die unterstützten Übertragungsraten und die Art der Verschlüsselung, falls erforderlich, enthalten. Durch diese Beacons kennen alle Endgeräte die Eigenschaften des Netzes, die zur Verbindung erforderlich sind. Der Infrastruktur-Modus ist der am Häufigsten genutzte. Im *Ad-Hoc*-Modus wird auf den Access Point verzichtet, alle Endgeräte bauen gegenseitig Verbindungen auf. Voraussetzung ist dabei, dass alle Endgeräte wegen der begrenzten Reichweite nahe beieinander sind, damit eine Kommunikation stattfinden kann. W-LAN wird nur selten für diese Art der Vernetzung benutzt, meistens werden stattdessen andere Techniken, wie Bluetooth oder Infrarot genutzt.
Die Reichweite von W-LAN auf freier Fläche schwankt je nach Gerät zwischen 30m und 100m, in Gebäuden liegt die Reichweite aufgrund der Dämpfung durch Wände meist darunter. Da das Signal Wände durchdringen kann und so von jedermann in der Nähe empfangbar ist, muss zur Wahrung der Vertraulichkeit und Integrität der Daten das Netz verschlüsselt werden. Die ersten Geräte nutzten *Wired Equivalent Privacy* (WEP), was aber aufgrund mehrerer gravierender Schwächen und dadurch entstehender Sicherheitsprobleme nicht mehr eingesetzt werden sollte. Besser sind die Weiterentwicklungen *Wi-Fi Protected Access* (WPA) und dessen Weiterentwicklung *WPA2*, welche diese Schwächen nicht aufweisen und mit WPA2 sogar den starken Verschlüsselungsalgorithmus AES bietet. Mit Übertragungsraten von 11MBit/s bzw. 54MBit/s ermöglicht es W-LAN in vielen Fällen eine deutlich schnellere Datenübertragung als die Mobilfunknetze, ist aber nicht flächendeckend nutzbar.[31] [32]

GPS

Das Global Positioning System (GPS), offiziell NAVSTAR GPS, ist ein globales satellitengestütztes Navigationssystem. Es wurde am 17.07.1995 offiziell in Betrieb genommen. Bis zum 02.05.2000 wurde das GPS durch das Hinzufügen von künstlichem Rauschen (Selective Availability) für alle Benutzer außer dem US-Militär gestört, die Genauigkeit lag zu dem Zeitpunkt im Bereich von 100m oder schlechter. Seit Abschaltung dieser Maßnahme ist eine auf mindestens 15m genaue Ortung für jedermann möglich.
GPS basiert auf mindestens 24 Satelliten, die per Radiosignal ständig ihre aktuelle Position und die genaue Uhrzeit ausstrahlen. GPS-Empfänger berechnen aus den Signallaufzeiten ihre genaue Position. Dazu ist das Signal von mindestens drei Satelliten nötig, da aber die meisten GPS-Empfänger nicht über Uhren verfügen, die genau genug sind, wird meistens das Signal eines vierten Satelliten benutzt, um diesen Nachteil auszugleichen.
Um jederzeit den Kontakt zu mindestens vier Satelliten sicherzustellen, befinden sich mindestens 24 Satelliten im Erdorbit. Jeder Satellit umkreist auf einer mittleren Bahnhöhe von 20.200km an einem Sterntag[33] zwei Mal die Erde.

[31] Prof. Dr. R. Wismüller. Rechnernetze II. Universität Siegen, 2011
[32] http://de.wikipedia.org/wiki/Wireless_Local_Area_Network
[33] 23 Stunden, 56 Minuten, 4,091 Sekunden

Das Radiosignal wird auf zwei Frequenzen, 1575,42 MHz (L1 genannt) und 1227,60 MHz (L2) ausgestrahlt und breitet sich ähnlich wie sichtbares Licht beinahe geradlinig aus. Bewölkung und Niederschlag beeinflusst das Signal kaum, aber durch die schwache Sendeleistung von 50 Watt ist die Ortung in Gebäuden oder im Wald nur sehr ungenau oder gar nicht möglich. Auch zwischen hohen Gebäuden kann es durch mehrfach reflektierte Signale zu Ungenauigkeiten kommen.
Eine Konkurrenz zu GPS ist das Galileo Satellitennavigationssystem der EU und der Europäischen Weltraumorganisation ESA. Im Gegensatz zu GPS ist Galileo nur für zivile Zwecke konzipiert und unterliegt keiner nationalen militärischen Kontrolle. Die Fertigstellung ist für 2013 geplant.[34]

[34]http://www.sat-nav.de/

2.4.2. Android

Diese Tutorial erklärt den Zugriff auf Sensoren und das Lesen ihrer Werte am Beispiel des Beschleunigungssensors (siehe Abbildung 2.21) und der Ortungsdaten des GPS-Systems (siehe Abbildung 2.22). Im Verlauf des Tutorials wird aufgrund unzureichender Berech-

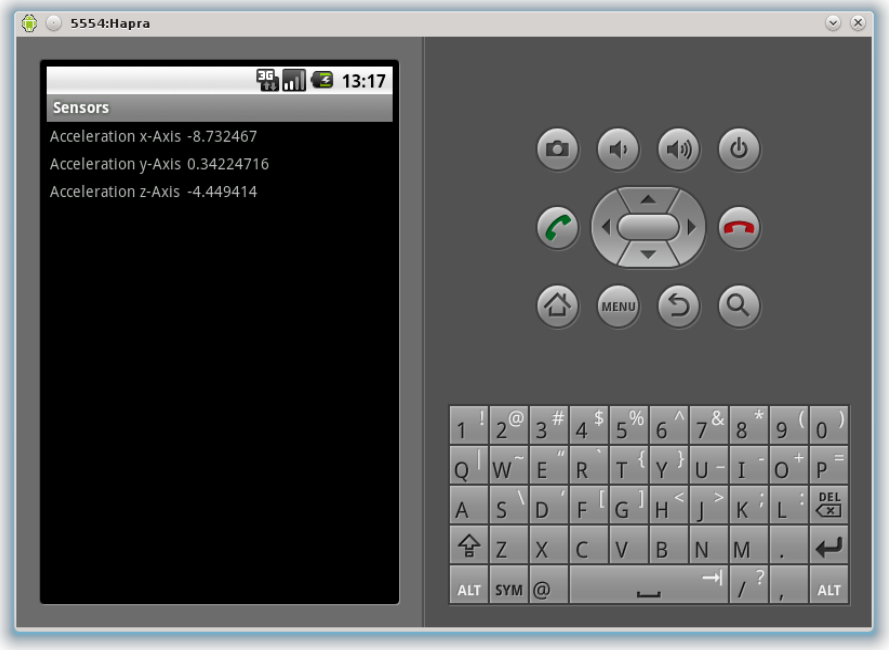


Abbildung 2.21.: Die Sensor-App, Beschleunigungssensor

tigungen der App ein Fehler auftreten, der mittels des Dalvik Debug Monitor Servers (DDMS) debuggt wird. Auch diese App wird mit, im Text erklärten und daher unkommentierten, Codelistings Schritt für Schritt erweitert. Dies Tutorial wurde von Herrn Jan Bauerdick im Rahmen seiner Bachelorarbeit entwickelt.

Tutorial 3: Kommunikation zwischen Aktivities und mit Sensoren, DDM
In diesem Tutorial[35] lernen Sie die Kommunikation zwischen Activities, das Lesen von Sensoren und die Arbeit mit dem Dalvik Debug Monitor Server (DDMS) kennen. Ziel des Tutorials ist eine App, die aus drei Activities besteht, wovon eine die Startactivity ist, eine die Daten des Beschleunigungssensors ausgibt und die letzte den aktuellen Längen- und Breitengrad mit Hilfe von GPS.
Erstellen Sie wie gewohnt ein neues Android-Projekt. Geben Sie dem Projekt den Namen *Sensors* und benennen das Package mit *hapra.sensor*. Wählen Sie dieses mal das Target für die Android Platform 2.3.3 mit dem API Level 10.
Erstellen Sie wie im vorhergehenden Tutorial den Ordner *menu* im Ordner *res* und legen Sie dann im Ordner *res/menu* die Datei mit dem Namen *menu.xml* an, welches die Elemente mit der Beschriftung *Accelerometer*, *GPS* und *Exit* enthält (Listing 2.2).

```
1 <?xml version="1.0" encoding="utf-8"?>
```

[35] Auch in diesem Tutorial werden Sie für einige Codebestandteile zusätzliche Import hinzufügen müssen. Sollten Fehler auftreten achten Sie zuerst darauf, dass alle benötigten Klassen importiert sind.



Abbildung 2.22.: Die Sensor-App, GPS-Daten

```
<menu xmlns:android="http://schemas.android.com/apk/
   res/android">
    <item android:id="@+id/accel"
       android:title="@string/accelItem"
       android:enabled="true" android:visible="true"></item>
    <item android:id="@+id/gps"
       android:title="@string/gpsItem" android:enabled="true"
       android:visible="true"></item>
    <item android:id="@+id/exit"
       android:title="@string/exitItem"
       android:enabled="true" android:visible="true"></item>
</menu>
```

Listing 2.2: Datei res/menu.xml im Projekt Sensors

Strings sollten Sie wieder in die Datei *res/values/strings.xml* auslagern. Bisher ist folgendes benötigt:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World,
       SensorsActivity!</string>
    <string name="app_name">Sensors</string>
    <string name="accelItem">Accelerometer</string>
    <string name="gpsItem">GPS</string>
    <string name="exitItem">Exit</string>
    <string name="exitConfirmTitle">Really exit?</string>
    <string name="exitConfirmText">End application?</string>
```

```
10 </resources>
```

Listing 2.3: Datei res/values/strings.xml im Projekt Sensors

Erstellen Sie im Ordner *res/layout* zwei weitere Layouts für die zusätzlichen Activities mit den Namen *accel.xml* und *gps.xml*. Der Beschleunigungssensor liefert drei Werte zurück, die für die Koordinatenachsen eines dreidimensonalen Koordinatensystem stehen (siehe Abbildung 2.23). Für die Darstellung dieser Werte sind also mindestens drei TextViews erforderlich. GPS liefert den Längen- und Breitengrad zurück, diese Aktivity benötigt also

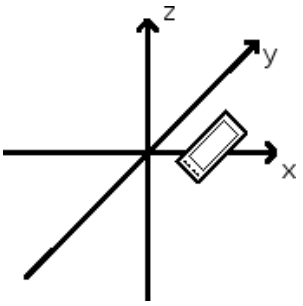


Abbildung 2.23.: Koordinatenachse mit Smartphone

zwei TextViews.
Listing 2.4 zeigt, wie die Datei *res/layout/accel.xml* aussehen könnte. Dieses Layout hat einen tabellarischen Aufbau (TableLayout) mit zwei Spalten und drei Zeilen. Die linke Spalte enthält TextViews zur Beschreibung der Werte, die rechte Spalte enthält die Werte der Sensoren für die x-, y- und z-Achsen.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <TableLayout
3    xmlns:android="http://schemas.android.com/apk/ res/android"
4    android:layout_width="fill_parent"
5    android:layout_height="fill_parent">
6      <TableRow android:id="@+id/tableRow1"
7        android:layout_width="wrap_content"
8        android:layout_height="wrap_content">
9          <TextView android:padding="3dip"
10           android:layout_width="fill_parent"
11           android:layout_height="wrap_content"
12           android:id="@+id/tvCoordX"
13           android:text="@string/tvCoordX">
14         </TextView>
15         <TextView android:padding="3dip"
16           android:layout_width="fill_parent"
17           android:layout_height="wrap_content"
18           android:id="@+id/coordX"
19           android:text="@string/coordX">
20         </TextView>
21     </TableRow>
22     <TableRow android:id="@+id/tableRow2"
23       android:layout_width="wrap_content"
```

```
24        android:layout_height="wrap_content">
25          <TextView android:padding="3dip"
26            android:layout_width="fill_parent"
27            android:layout_height="wrap_content"
28            android:id="@+id/tvCoordY"
29            android:text="@string/tvCoordY">
30          </TextView>
31          <TextView android:padding="3dip"
32            android:layout_width="fill_parent"
33            android:layout_height="wrap_content"
34            android:id="@+id/coordY"
35            android:text="@string/coordY">
36          </TextView>
37      </TableRow>
38      <TableRow android:id="@+id/tableRow3"
39        android:layout_width="wrap_content"
40        android:layout_height="wrap_content">
41          <TextView android:padding="3dip"
42            android:layout_width="fill_parent"
43            android:layout_height="wrap_content"
44            android:id="@+id/tvCoordZ"
45            android:text="@string/tvCoordZ">
46          </TextView>
47          <TextView android:padding="3dip"
48            android:layout_width="fill_parent"
49            android:layout_height="wrap_content"
50            android:id="@+id/coordZ"
51            android:text="@string/coordZ">
52          </TextView>
53      </TableRow>
54  </TableLayout>
```

Listing 2.4: Datei res/layout/accel.xml im Projekt Sensors

Da neue Strings hinzugefügt wurden, muss die Datei *res/values/strings.xml* wie in Listing 2.5 beschrieben erweitert werden.

```
1     <string name="tvCoordX">Acceleration x-Axis</string>
2     <string name="tvCoordY">Acceleration y-Axis</string>
3     <string name="tvCoordZ">Acceleration z-Axis</string>
4     <string name="coordX">0.0</string>
5     <string name="coordY">0.0</string>
6     <string name="coordZ">0.0</string>
7     <string name="accelError">No accelerometer found</string>
```

Listing 2.5: Erweiterung der Datei res/values/strings.xml

Jetzt sollten keine Fehler mehr angezeigt werden!

Öffnen Sie anschließend im Ordner *res/layout* die Datei *gps.xml*, die das Layout für die GPS-Activity enthält und legen darauf vier TextViews an, zum Beispiel wie in Listing 2.6 beschrieben.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/ res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
    <TextView android:layout_height="wrap_content"
      android:layout_width="wrap_content"
         android:id="@+id/tvLon"
      android:text="@string/tvLon">
    </TextView>
    <TextView android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:textAppearance="?android:attr/
         textAppearanceMedium"
      android:id="@+id/lon"
      android:text="@string/lon">
    </TextView>
    <TextView android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:id="@+id/tvLat" android:text="@string/tvLat">
    </TextView>
    <TextView android:layout_height="wrap_content"
      android:layout_width="wrap_content"
        android:textAppearance="?android:attr/
           textAppearanceMedium"
        android:id="@+id/lat" android:text="@string/lat">
    </TextView>
</LinearLayout>
```

Listing 2.6: Datei res/layout/gps.xml im Projekt Sensors

Wieder muss die Datei res/values/strings.xml erweitert werden:

```
    <string name="tvLon">Longitude</string>
    <string name="lon">0.0</string>
    <string name="tvLat">Latitude</string>
    <string name="lat">0.0</string>
    <string name="gpsDisabled">GPS disabled</string>
    <string name="gpsEnabled">GPS enabled</string>
```

Listing 2.7: Zweite Erweiterung der Datei res/values/strings.xml

Über das Optionsmenü soll es später möglich sein die Sensoren auszulesen sowie die App zu beenden. Erweitern Sie die Klasse `SensorsActivity` um den zur Darstellung und Funktionalität benötigten Code (Listing 2.8).

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        boolean result = super.onCreateOptionsMenu(menu);
        MenuInflater mi = getMenuInflater();
        mi.inflate(R.menu.menu, menu);
```

```
6         return result;
7 }
8
9 @Override
10 public boolean onOptionsItemSelected(MenuItem item) {
11    switch (item.getItemId()) {
12    case R.id.accel:
13       Intent accel = new Intent(this,
             AccelerometerActivity.class);
14       startActivity(accel);
15       return true;
16    case R.id.gps:
17       Intent gps = new Intent(this, GPSActivity.class);
18       startActivity(gps);
19       return true;
20    case R.id.exit:
21    new AlertDialog.Builder(this)
22              .setTitle(R.string.exitConfirmTitle)
23              .setMessage(R.string.exitConfirmText)
24              .setPositiveButton("OK",
25                   new DialogInterface.OnClickListener() {
26                      public void onClick(DialogInterface
                            dialog,
27                            int which) {
28                               finish();
29                            }
30                   })
31              .setNegativeButton("Cancel",
32                   new DialogInterface.OnClickListener() {
33                      public void onClick(DialogInterface
                            dialog,
34                            int which) {
35                            }
36                   }).show();
37       return true;
38    }
39    return true;
40 }
```

Listing 2.8: Methoden onCreateOptionsMenu und onOptionsItemSelected der Klasse SensorsActivity

In diesem Code kommt zum ersten Mal ein `Intent` vor. Ein Intent erklärt die Absicht, eine neue Aktivity zu starten. Der Code in Listing 2.9 startet die Activity, die in der Klasse `Activity.class` beschrieben ist ist aus dem aktuellen Context (this) heraus. Die Activities für die Darstellung der Daten der Beschleunigungssensoren und des GPS werden hier gestartet.

```
1 Intent i = new Intent(this, Activity.class);
2 startActivity(i);
```

Listing 2.9: Intent-Code

Der Code zum Beenden der App ist identisch mit dem aus dem vorhergehenden Tutorial. Die Vorbereitung eine andere Activity zu starten ist damit abgeschlossen.

HINWEIS: Zur Fehlerbehebung müssen Sie wie im letzten Tutorial wieder die *Import*-Anweisungen, wie von Ellipse vorgeschlagen um (*view.Menu*, *view.MenuInflater*, *view.MenuItem*, *content.Intent*, *app.AlertDialog*, *content.DialogInterface*) erweitern. Da die Klassen *AccelerometerActivity* und *AccelerometerActivity* noch nicht existieren, werden zum Schluss immer noch zwei Fehler angezeigt. Diese Klassen werden später noch angelegt.

Der Beschleunigungsensor
Für den Zugriff auf den Beschleunigungssensor erstellen Sie eine Klasse mit dem Namen `Accelerometer`. Innerhalb dieser Klasse legen Sie ein Interface mit dem Namen `Callback` an, welches die Methode `accelerationChanged(float x, float y, float z)` enthält. Dieses Interface wird in der fertiggestellten App mit einer Activity kommunizieren, damit diese auf Veränderungen der Sensorwerte reagieren kann, die Verschachtelung vereinfacht diese Kommunikation aus dieser Klasse heraus. Bisher sieht die Klasse wie folgt aus:

```
1 public class Accelerometer {
2
3         public interface Callback {
4                 public void accelerationChanged(float x,
                     float y, float z);
5         }
6
7 }
```

Listing 2.10: Die Klasse Accelerometer mit Interface Callback

Der Zugriff auf die Sensoren wird durch die Klasse `SensorManager` gesteuert. Legen Sie in der Klasse *Accelerometer* dafür ein Attribut an. Ein weiteres Attribut vom Typ `Accelerometer.Callback` wird für die Kommunikation mit einer Activity benötigt. Legen Sie außerdem einen Konstruktor an, in dem diese beiden Attribute initialisiert werden (siehe Listing 2.11).

```
1 private SensorManager sm;
2 private Accelerometer.Callback cb;
3
4 public Accelerometer(Context c, Accelerometer.Callback cb) {
5         this.cb = cb;
6         sm = (SensorManager) c.getSystemService(Context.
7                 SENSOR_SERVICE);
8         //hier wird in ein paar Schritten ein Listener
             registriert
9 }
```

Listing 2.11: Attribute für den Zugriff auf den SensorManager und Callback und Initialisierung über einen Konstruktor

HINWEIS: Erweiterten Sie hier und im Folgenden wieder selbständig die *Import*-Anweisungen.

Der Konstruktor erwartet zwei Argumente. Das erste ist der `Context` einer Activity, über den mit der Methode `getSystemService` der SensorManager initialisiert wird. Das zweite Argument ist der Callback der Activity, mit dem die Kommunikation stattfinden wird. Der letzte Schritt ist die Registrierung eines Listeners an dem gewünschten Sensor. Bevor dies gemacht werden kann muss der Listener vom Typ `SensorEventListener` in der Klasse *Accelerometer* erstellt werden. Den dazu benötigten Code finden Sie in Listing 2.12.

```
1  private SensorEventListener listener = new
       SensorEventListener() {
2
3          @Override
4          public void onAccuracyChanged(Sensor sensor, int
               accuracy) {
5          }
6
7          @Override
8          public void onSensorChanged(SensorEvent event) {
9                  cb.accelerationChanged(event.values[0],
                       event.values[1], event.values[2]);
10         }
11 };
```

Listing 2.12: Der Listener für den Beschleunigungssensor

Ein `SensorEventListener` verlangt das Überschreiben von zwei Methoden. Die Methode `onAccuracyChanged` reagiert auf Änderungen, die die Genauigkeit des Sensors betreffen und wird in diesem Beispiel nicht benötigt. Die Methode `onSensorChanged` reagiert auf veränderte Messwerte des Sensors und ist damit die Methode, die in den meisten Apps benötigt wird. Übergeben wird ihr ein `SensorEvent`. Dieses enthält den Sensor, dessen Daten gemessen wurden im Attribut `sensor` und die gemessenen Werte im float-Array `values`. In der Methode werden diese Werte an die Methode `accelerationChanged` des Callback übergeben.
Nachdem der Listener erstellt ist, kann er im Konstruktor am Beschleunigungssensor registriert werden. Fügen Sie dem Konstruktor (von Listing 2.11 folgende Zeile hinzu:

```
1  sm.registerListener(listener,
       sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
       SensorManager.SENSOR_DELAY_NORMAL);
```

Listing 2.13: Registrierung des Listeners

Diese Zeile registriert den Listener am Standardbeschleunigungssensor mit normaler Abfragerate.
Diese Klasse *Accelerometer* benötigt eine weitere Methode, die den Listener deregistriert sobald die App beendet wird (siehe Listing 2.14).

```
1  public void close() {
2          sm.unregisterListener(listener);
3  }
```

Listing 2.14: Deregistrierung des Listeners

Mit dieser Methode ist diese Klasse vollständig.
Erstellen Sie nun die Activity, die die Daten des Beschleunigungssensors anzeigt. Erstellen Sie die Klasse mit dem Namen `AccelerometerActivity` an. Wichtig ist, dass die Klasse von der Oberklasse `Activity` erbt und das eben erstellte Interface `Accelerometer.Callback` implementiert (*public class AccelerometerActivity extends Activity implements Accelerometer.Callback*).

In dieser Klasse werden in jedem Fall die Methoden `onCreate` aufgrund der Oberklasse `Activity` sowie die Methode `accelerationChanged` des Interfaces benötigt. Außerdem wird ein Attribut benötigt, in dem eine Instanz der Klasse `Accelerometer` gespeichert wird. Legen Sie diese an (siehe Listing 2.15).

```
1  private Accelerometer a;
2
3  /** Called when the activity is first created. */
4  @Override
5  public void onCreate(Bundle savedInstanceState) {
6          super.onCreate(savedInstanceState);
7          setContentView(R.layout.accel);
8          a = new Accelerometer(this, this);
9  }
10
11 @Override
12 public void accelerationChanged(float x, float y, float z) {
13         TextView tx = (TextView) findViewById(R.id.coordX);
14         tx.setText(x + "");
15         TextView ty = (TextView) findViewById(R.id.coordY);
16         ty.setText(y + "");
17         TextView tz = (TextView) findViewById(R.id.coordZ);
18         tz.setText(z + "");
19 }
```

Listing 2.15: Das Attribut und die Methoden onCreate und accelerationChanged der Activity

In der Methode `onCreate` wird die Activity mit dem in der Datei *res/layout/accel.xml* definierten Layout geladen und eine Instanz von `Accelerometer` erstellt. Die Methode `accelerationChanged` empfängt die vom Beschleunigungssensor gemessenen Werte und zeigt diese in den im Layout definierten TextViews an.
Zum Schluss müssen Sie noch die Methode `onDestroy` erstellen, die aufgerufen wird, wenn die Activity beendet wird. Diese Methode ruft lediglich `onDestroy` der Oberklasse auf und beendet den `Accelerometer` (siehe Listing 2.16).

```
1  @Override
2  public void onDestroy() {
3          super.onDestroy();
4          a.close();
5  }
```

Listing 2.16: Die Methode onDestroy

Lesen der GPS-Daten

Das Lesen der GPS-Daten ist weniger aufwändig als beim Beschleunigungssensor. Es wird nur eine Klasse benötigt, die gleichzeitig die Activity ist. Erstellen Sie eine Klasse mit dem Namen `GPSActivity`, welche von der Oberklasse `Activity` erbt und das Interface `LocationListener` implementiert. Erstellen Sie in dieser Klasse die Methode `onCreate`, wie in Listing 2.17 zu sehen.

```
1 @Override
2 public void onCreate(Bundle savedInstanceState) {
3    super.onCreate(savedInstanceState);
4    setContentView(R.layout.gps);
5    LocationManager lm = (LocationManager)
        getSystemService(Context.LOCATION_SERVICE);
6    LocationListener ll = this;
7    lm.requestLocationUpdates(LocationManager. GPS_PROVIDER,
        0, 0, ll);
8 }
```

Listing 2.17: Die Methode onCreate der Klasse GPSActivity

Diese Methode lädt das Layout, welches in *res/layout/gps.xml* erstellt wurde und zusatzlich einen LocationManager sowie einen LocationListener, welche für den Zugriff auf GPS-Daten benötigt werden.
Durch implementieren des Interfaces `LocationListener` müssen die vier Methoden `onLocationChanged`, `onProviderDisabled`, `onProviderEnabled` und `onStatusChanged` überschrieben werden. Fehlen die Methoden in Ihrer Klasse, meldet Eclipse dies als Fehler und kann diese mit einem Klick auf die Fehlermarkierung am Zeilenanfang hinzufügen. Ändern Sie die Methode `onLocationChanged` wie in Listing 2.18 angegeben.

```
1 @Override
2 public void onLocationChanged(Location l) {
3    double lon = l.getLongitude();
4    double lat = l.getLatitude();
5    TextView tvLon = (TextView)findViewById(R.id.lon);
6    TextView tvLat = (TextView)findViewById(R.id.lat);
7    tvLon.setText(lon + "");
8    tvLat.setText(lat + "");
9 }
```

Listing 2.18: Die Methode onLocationChanged der Klasse GPSActivity

Diese Methode liest Längen- und Breitengrad, sucht die gewünschten TextViews zur Darstellung und schreibt die aktuellen Werte hinein. Die Methoden `onProviderDisabled` und `onProviderEnabled` werden genutzt, um Meldungen über den Status des GPS-Moduls auszugeben (siehe Listing 2.19).

```
1 @Override
2 public void onProviderDisabled(String provider) {
3    Toast.makeText(this, R.string.gpsDisabled,
        Toast.LENGTH_LONG).show();
4 }
5
6 @Override
```

```
7 public void onProviderEnabled(String provider) {
8    Toast.makeText(this, R.string.gpsEnabled,
        Toast.LENGTH_LONG).show();
9 }
```

Listing 2.19: Die Methoden onProviderDisabled und onProviderEnabled der Klasse GPSActivity

Die Methode `onStatusChanged` reagiert auf Statusänderungen des Providers und wird für diese App nicht benötigt.

Die Klassen `GPSActivity` und *AccelerometerActivity* müssen in *AndroidManifest.xml* bekannt gemacht werden. Dies erledigt der Code aus Listing 2.20, den Sie in die Datei *AndroidManifest.xml* einfügen.

```
1 <activity android:label="@string/app_name"
     android:name=".AccelerometerActivity">
2 </activity>
3 <activity android:label="@string/app_name"
     android:name=".GPSActivity">
4 </activity>
```

Listing 2.20: GPSActivity in AndroidManifest bekannt machen

HINWEIS:Die *Activities* müssen innerhalb der *Application* stehen.

Fehlersuche mit dem Dalvik Debug Monitor Server
Starten Sie die App im Emulator. Wenn Sie zur Activity des Beschleunigungssensors wechseln, zeigt Ihnen der Sensor des Emulators Zufallswerte an. Wechseln Sie zur GPS-Activity wird Ihre App fehlerhaft beendet. Um den Fehler zu finden, wechseln Sie in Eclipse in die DDMS-Perspektive, indem Sie in der Menüleiste *Window*, *Open Perspective*, *other* und aus der nun folgenden Liste den Eintrag *DDMS* auswählen.

Der Dalvik Debug Monitor Server (DDMS) ist eine Hilfe beim Debuggen von Android-Apps. über eine Perspektive in Eclipse lässt sich DDMS bedienen. Die Perspektive ist aufgebaut wie in Abbildung 2.24 zu sehen. Im linken Bereich mit dem Namen *Devices* sehen Sie eine Auflistung aller Geräte und Emulatoren und eine Übersicht über die Prozesse, die diese ausführen. Darunter im Bereich *Emulator Control* haben Sie die Möglichkeit den Emulator zu kontrollieren. Zum Beispiel dem Emulator einen Anruf schicken oder einen Netzausfall zu simulieren, um Ihre App unter diesen Bedingungen zu testen. Im rechten Bereich finden Sie weitere detaillierte Informationen, die beim Debuggen nötig sind. Der wichtigste Teil, die Ausgabe des Tools *logcat*, finden Sie im unteren Bereich. Hier landen alle Ausgaben aller Apps und die Informationen zu Exceptions. Damit keine Meldung übersehen wird, ist es möglich nach Loglevel von *verbose* (alles anzeigen) bis *error* (nur kritische Fehler anzeigen) zu filtern. Diese Filter können Sie im oberen rechten Teil von logcat verwalten.
Die App wurde sofort wegen eines kritischen Fehlers beendet, als der GPS-Empfänger angesprochen wurde. Filtern Sie also die Meldungen nach kritischen Fehlern und scrollen Sie nach unten. Sie finden dort eine RuntimeException, die von hapra.sensors.GPSActivity ausgelöst wurde. Im Fehlertext steht der genaue Grund: **java.lang.SecurityException: Requires ACCESS_FINE_LOCATION** Wegen des hohen Missbrauchpotentials ist für

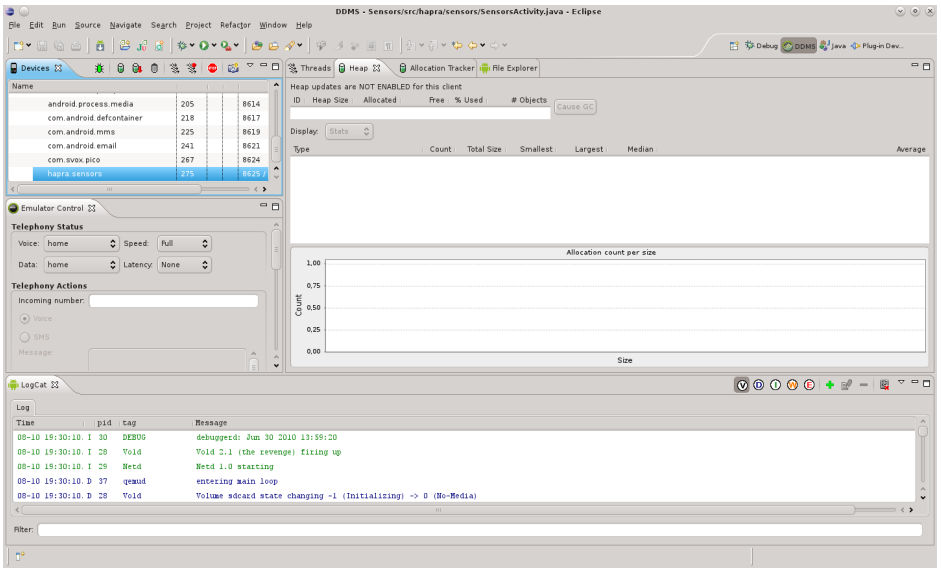


Abbildung 2.24.: Die Perspektive DDMS

das Lesen der GPS-Daten eine zusätzliche Berechtigung nötig. Diese werden in der Datei *AndroidManifest.xml* angegeben. Wechseln Sie wieder zur Java-Perspektive und öffnen Sie diese Datei. Am unteren Rand des Editors finden Sie den Abschnitt *Permission*, klicken Sie darauf. In der neuen Ansicht klicken Sie auf *Add* und wählen *Uses Permission*. Nun können Sie im rechten Bereich im Drop-Down-Menü die Berechtigung auswählen. Der Name der Berechtigung wurde in der Exception genannt, es ist *ACCESS_FINE_LOCATION*. Wählen Sie diese und speichern Sie ab.
Starten Sie die App erneut. Nun ist der Abruf der GPS-Daten erfolgreich, der Emulator liefert für Längen- und Breitengrad jeweils den Wert 0,0. Mit DDMS ist es aber möglich, diese Daten zu ändern und die Funktion zu testen.
Wechseln Sie wieder zur DDMS-Perspektive und markieren oben links Ihren Emulator. Darunter im Bereich *Emulator Control* finden Sie unter *Location Controls* zwei Eingabefelder, wo Sie die neuen Koordinaten eingeben können. Ein Klick auf *Send* schickt diese an den Emulator, wo die App die neuen Werte anzeigen sollte. Alternativ können Sie wie aus den anderen Tutorials bekannt die App auf Ihrem Smartphone ausführen und mit echten Sensoren testen.

2.4.3. Pong und der SensorSimulator

Wie Sie im Tutorial 2 gelernt haben, geben die Sensoren eines AVD nur feste Werte zurück. Zur Steuerung des Spiels ist das nicht geeignet. Der SensorSimulator der Firma OpenIntents UG löst dieses Problem. Mittels minimaler Anpassungen der App ermöglicht es die Simulation der Sensoren und macht die Übermittlung der Werte über eine GUI möglich.
Der SensorSimulator besteht aus mehreren Teilen. Die Datei *SensorSimulator.jar* ist die GUI, die die Steuerung der Sensoren und Übermittlung der Werte übernimmt. Die Datei *SensorSimulatorSettings.apk* ist eine App, die die Verbindung zum SensorSimulator herstellt. Die Datei *sensorsimulator-lib-1.1.1.jar* stellt die Bibliotheken zur Verfügung, die Ihre App zur Kommunikation mit den simulierten Sensoren benötigt.
Um den Umgang mit dem SensorSimulator zu lernen, werden Sie jetzt die in Tutorial 2 erstellte App *Sensors* dahingehend anpassen, dass diese die simulierten Sensorwerte anzeigt.
Starten Sie dazu Eclipse und öffnen das Projekt *Sensors*. Zunächst erstellen Sie einen neuen Ordner, in welchem zusätzliche Bibliotheken bereitgestellt werden. Klicken Sie mit der rechten Maustaste auf den Projektordner und wählen Sie *New*, *Folder*. Geben Sie dem Ordner den Namen *lib* und kopieren Sie die Datei *sensorsimulator-lib-1.1.1.jar* in diesen Ordner. Die Datei liegt unter Moodle in Versuch 2. Öffnen Sie den Browser auf Ihrem virtuellen PC und laden Sie diese Datei runter. Damit diese Bibliotheken im Projekt genutzt werden können, müssen Sie diese dem Build Path des Projekts hinzufügen. Klicken Sie dazu erneut mit der rechten Maustaste auf den Projektordner und wählen im Popup-Menü den Punkt *Properties*. Ein neues Fenster öffnet sich (siehe Abbildung 2.25), in welchem

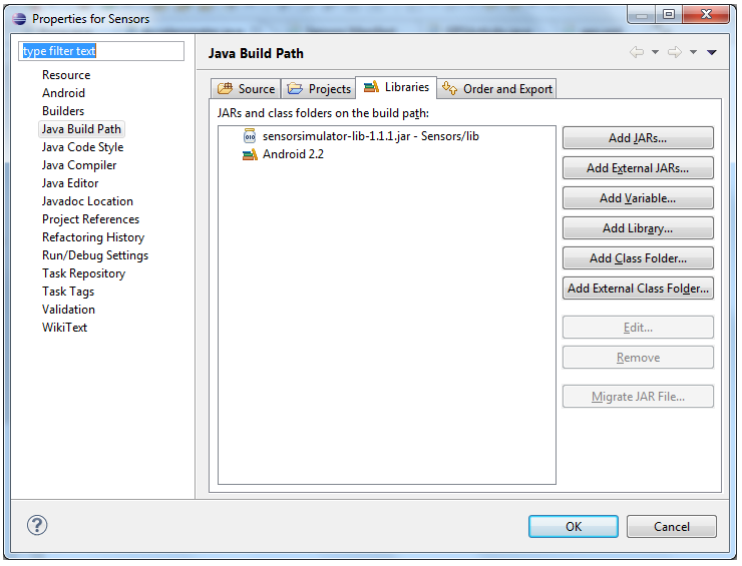


Abbildung 2.25.: Eigenschaften des Projekts zum Ändern des Build Path.

Sie viele projektspezifische Einstellungen vornehmen können. Klicken Sie im linken Bereich auf den Eintrag *Java Build Path* und wählen anschließend im rechten Bereich den Reiter *Libraries*. Klicken Sie auf den Button *Add JARs...* und wählen im folgenden Auswahlmenü die Datei *sensorsimulator-lib-1.1.1.jar* aus dem Ordner *lib* Ihres Projekts. Bestätigen Sie zwei Mal mit *OK*. Ihr Projekt kann jetzt die Klassen dieser Bibliothek nutzen.
Die Kommunikation der App mit dem Simulator geschieht über eine TCP/IP-Verbindung. Damit die App eine solche Verbindung herstellen kann, benötigt Sie eine zusätzliche Be-

rechtigung. Öffnen Sie die Datei *AndroidManifest.xml* aus dem Projektordner und wechseln Sie zum Reiter *Permissions*. Fügen Sie mittels Klick auf *Add* die *Uses Permission* `android.permission.INTERNET` hinzu und speichern Sie die Datei.
Das Lesen der Werte des Beschleunigungssensors wurde in der Klasse `Accelerometer` im Package `hapra.sensors` implementiert. Öffnen Sie diese Klasse. Löschen Sie aus dieser Klasse zunächst die Imports `android.hardware.Sensor`, `android.hardware.SensorEvent` und `android.hardware.SensorEventListener` oder kommentieren Sie diese aus. Die nun auftretenden Fehler ignorieren Sie vorerst. Löschen Sie im Konstruktor der Klasse die Zeile `sm = (SensorManager)...` oder kommentieren Sie diese aus. Anschließend fügen Sie dort den Code für den Zugriff auf den SensorSimulator hinzu, wie in Listing 2.21 zu sehen.

```
1 this.cb = cb;
2 //sm = (SensorManager)
     c.getSystemService(Context.SENSOR_SERVICE);
3 sm = SensorManagerSimulator.getSystemService(c,
     Context.SENSOR_SERVICE);
4 sm.connectSimulator();
5 sm.registerListener(listener,
     sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
     SensorManager.SENSOR_DELAY_NORMAL);
```

Listing 2.21: Änderung des Konstruktors zum Zugriff auf den SensorSimulator

Das Attribut `sm` dieser Klasse müssen Sie zum Typ `SensorManagerSimulator` ändern. Wichtig ist der Aufruf der Methode `connectSimulator`, fehlt diese ist keine Verbindung zum SensorSimulator möglich. Der Listener sowie der Rest der App muss nicht verändert werden.
Da Sie die Imports gelöscht bzw. auskommentiert haben, meldet Eclipse zahlreiche Fehler. Um die Fehler zu beheben fügen Sie folgende Imports hinzu:

```
import org.openintents.sensorsimulator.hardware.Sensor;
import org.openintents.sensorsimulator.hardware.SensorEvent;
import org.openintents.sensorsimulator.hardware.SensorEventListener;
import org.openintents.sensorsimulator.hardware.SensorManagerSimulator;
```

Diese Imports enthalten Nachbildungen der Sensoren aus der dem Build Path hinzugefügten Bibliotheken. Ihre App kann nun mit dem SensorSimulator kommunizieren.

Wichtiger Hinweis: Die Kommunikation zwischen dem SensorSimulator und Ihrer App erfolgt über IP. Leider hat sich gezeigt das SensorSimulator mit der IP-Adresse der virtuellen Maschine nicht klar kommt. Es gibt jedoch zwei Möglichkeiten das Problem zu umgehen. Erst mal ist es wichtig, dass Sie die IP-Adresse ihres Host-Rechners wissen. Im Labor finden Sie die IP-Adresse auf dem Gehäuse des PCs. Entweder starten Sie im Folgenden den SensorSimulator (siehe Schritt 1) auf ihrem Host oder Sie definieren bevor Sie den virtuellen Rechner starten eine Regel zur Portweiterleitung siehe Abbildung 2.26.

Die Host-IP müssen Sie natürlich jeweils anpassen. Als Port verwenden Sie sowohl für den Host wie auch für Gast *8010*. Die IP-Ihres virtuellen Rechners ist üblicherweise *10.0.2.15*. Überprüfen Sie dies, nachdem Sie Ihren virtuelle Maschine gestartet haben. Damit die

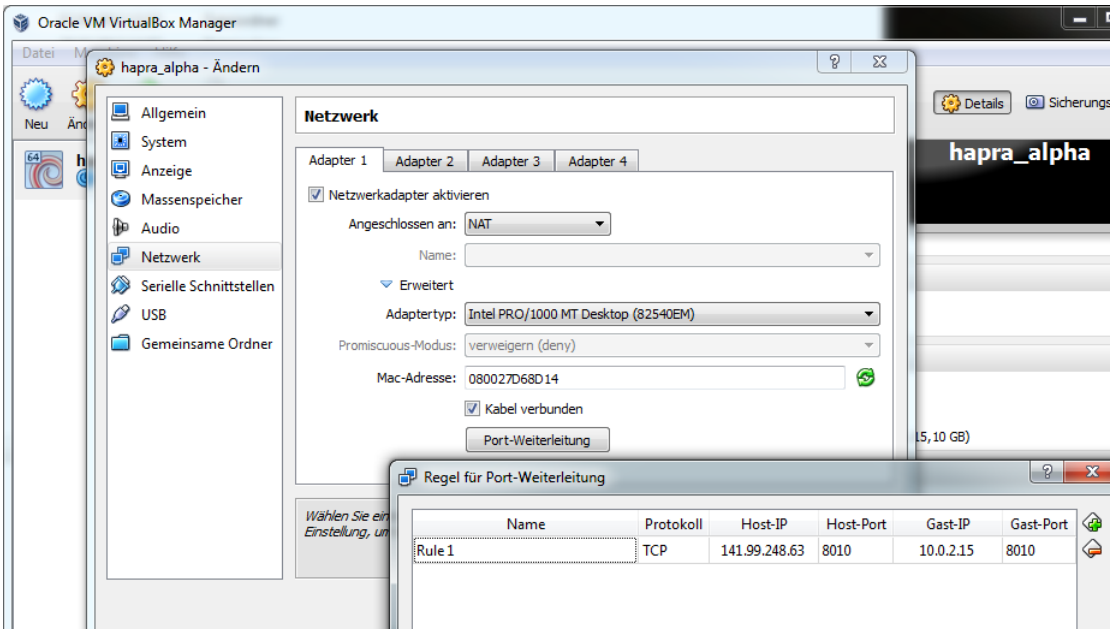


Abbildung 2.26.: Regel zur Portweiterleitung

Kommunikation mit dem SensorSimulator korrekt funktioniert, ist es wichtig, dass die folgenden Schritte in der richtigen Reihenfolge ausgeführt werden!

1. Starten Sie zuerst die Datei *SensorSimulator.jar*[36]. Dieses Programm ermöglicht es, die Sensorwerte zu ändern und schickt diese Werte an Ihre App. Zu diesem Zweck startet es einen Server, über den die nachgebildeten Sensoren Ihre Werte beziehen. Starten Sie das Programm auf Ihrem Host oder bei Verwendung der Portweiterleitung auf Ihrem virtuellen Rechnern. Wählen Sie im Eingabefeld *Socket* (nicht *Telnet*!) einen Port, an den der Server gebunden werden soll und starten Sie ihn mittels Klick auf *Set*.

2. Starten Sie nun das AVD. Bisher wurde das AVD gestartet, wenn Sie aus Eclipse eine App gestartet haben. Diesmal muss das AVD alleine gestartet werden. Wählen Sie dazu in Eclipse aus der Menüzeile den Wert *Window* und klicken Sie auf den Eintrag *AVD Manager*. Das Android SDK, welches Sie aus Tutorial 0 vom Installieren der Android-Platform und Erstellen des AVD kennen, startet. Wählen Sie im linken Bereich den Eintrag *Virtual Devices*, markieren Sie Ihr erstelltes AVD und klicken auf *Start*. Achten Sie darauf, dass Sie einen *Virtual Devices*, für die Android Platform 2.3.3 starten. Im folgenden Fenster belassen Sie die Standardeinstellungen und klicken auf *Launch*. Warten Sie, bis das AVD gestartet ist.

3. **Dieser Schritt muss nur einmal durchgeführt werden. Auf ihrem USB wurde das Programm bereits installiert!**
   Wenn das AVD gestartet ist, starten Sie eine Eingabeaufforderung oder Terminal und installieren die App *SensorSimulatorSettings.apk* mittels des Befehls

   ```
   ./adb install /path/to/SensorSimulatorSettings.apk
   ```

[36] Das Programm finden Sie unter Moodle. Auf ihrem virtuellen PC ist das Programm bereits installiert und kann direkt vom Desktop aus gestartet werden.

wobei Sie /path/to durch den Pfad zu der Datei austauschen. Sollten der Befehl *adb* nicht gefunden werden, wechseln Sie in das Verzeichnis *android-sdk/android-sdk-linux/plattform-tools* und führen obigen Befehl erneut aus.

4. Starten Sie die App *SensorSimulatorSettings.apk* aus dem App-Drawer (diesen erreichen Sie im AVD durch das mittlere Symbol am unteren Rand, siehe Abbildung 2.27), falls diese noch nicht läuft. Wechseln Sie zur GUI des SensorSimulator. Aktivieren Sie in den Bereichen *Supported Sensors* und *Enabled Sensors* die Checkbox *Accelerometer* und stellen Sie sicher, dass die anderen Checkboxen deaktiviert sind. Geben Sie die IP Ihres Host-Rechners und den Port *8010* im Eingabefeld der App vom *Virtual Devices* ein und klicken Sie dann auf den Button *Testing*. Hier klicken Sie auf *Connect*. Wenn die Verbindung erfolgreich hergestellt wurde zeigt die App nun den Beschleunigungssensor an. Aktivieren Sie ihn durch Aktivieren der Checkbox. **Beenden Sie die App nicht!**.

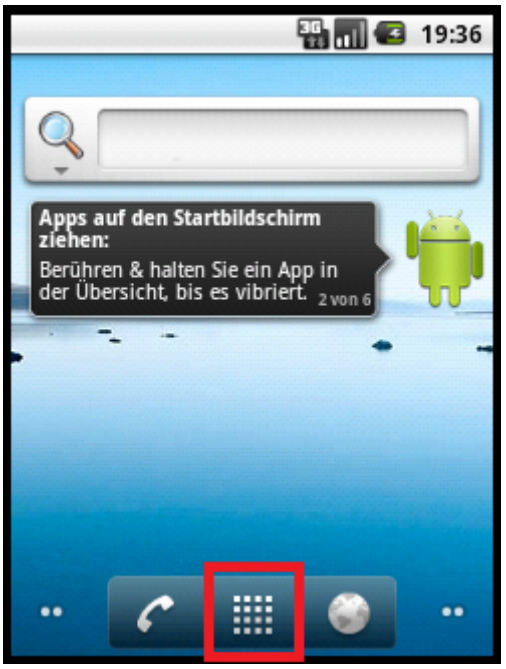


Abbildung 2.27.: Der App-Drawer von Android

5. Wechseln Sie zu Eclipse und starten die App *Sensors*, die auf dem AVD automatisch gestartet wird. Wechseln Sie zum Beschleunigungssensor. In der GUI des SensorSimulator können Sie jetzt das Smartphone *bewegen* durch Benutzen der Radiobuttons und Verschieben des Smartphones (siehe grün umrandeter Bereich in Abbildung 2.28 oder durch ändern der Werte der Slider (blau umrandeter Bereich in Abbildung 2.28).

2.4.3.1. Pong

Laden Sie das Archiv *Pong_aufgabe.zip* herunter und importieren Sie es in Eclipse in Ihren Workspace. In diesem Archiv ist eine Version des Spiels Pong enthalten, die Sie erweitern sollen. Zu erweiternde Stellen erkennen Sie im Code am Kommentar `//TODO`.

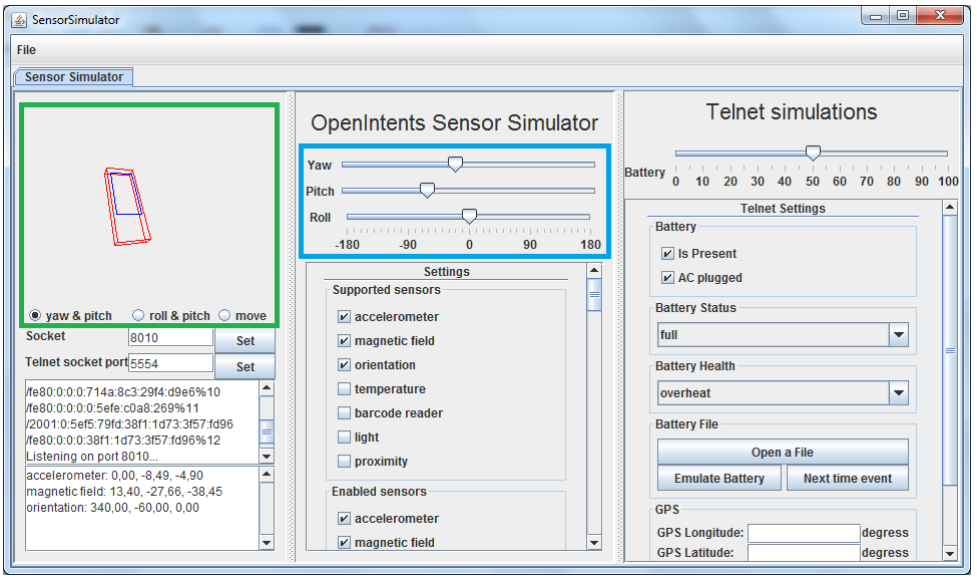


Abbildung 2.28.: Der SensorSimulator

Startactivity des Projekts ist die Klasse `PongActivity`. In ihr wird der Portait-Modus des Displays (quer) aktiviert und die Klasse `Pong` gestartet. Die Activity ist fertiggestellt und bedarf keiner Änderung.
Die Klasse `Pong` stellt die Spiellogik dar und zeichnet den aktuellen Spielstand auf dem Display. Diese Klasse erweitert `SurfaceView`, eine Klasse, die Bewegungen auf dem Display anzeigen kann und in der Elemente frei platziert werden können. Sobald der `SurfaceView` aufgebaut wurde, wird die Methode `onSurfaceCreated` aufgerufen. Diese Methode initialisiert das Spiel über `initGame` und startet einen Thread, der das Spiel ausführt. Zu diesem Zeitpunkt befinden sich zwei unbewegliche Schläger auf dem Display. Die Methode `onSurfaceChanged` reagiert auf Veränderungen des SurfaceView. Da bei Pong in diesem Fall keine Aktion erforderlich ist, ist der Methodenrumpf leer. Die Methode `onSurfaceDestroyed` wird aufgerufen, wenn das Spiel beendet wird. Sie stoppt den Thread. Für den Start eines neuen Spiels ist die Methode `onTouchEvent` verantwortlich. Sie reagiert auf Berührung des Touchscreens. Bei Berührung wird dem Spiel ein Ball hinzugefügt, dessen Richtung und Geschwindigkeit zufällig festgelegt ist, bevor das Spiel dann freigegeben wird. In der Methode `updatePhysics` befindet sich die Spiellogik. In dieser Methode werden die neuen Positionen der Spielelemente berechnet, auf Kollisionen des Balls mit den Seiten reagiert, sowie auf Kollisionen mit den Schlägern und die Prüfung, ob der Ball am Schläger vorbei ins Aus gegangen ist.
In der Klasse `Pong` muss von Ihnen das Lesen des Neigungssensors (über den Namen *orientation* ansprechbar) sowie die Reaktion auf diese Werte implementiert werden. Außerdem soll bei jeder Kollision des Balles mit einem Schläger oder dem Bildschirmrand eine kurze (50ms) Vibration ausgelöst werden. Sollte ein Spieler es schaffen einen Punkt zu erzielen, soll das Smartphone 250ms vibrieren.
Die Klasse `GameThread` definiert den Thread, der das Spiel ausführt. Solange das Spiel läuft, werden in diesem Thread die neuen Positionen der Spielelemente berechnet und gezeichnet.
Die Klasse `GameElement` beschreibt ein Element des Spiels, d.h. `Ball` und `Paddle`. Jedes Spielelement verfügt über eine Position, die in `Coordinates` gespeichert wird sowie die Bewegung in x- und y-Richtung, die in `Speed` gespeichert wird.

Die Klasse `Orientation` implementiert den Zugriff auf den (simulierten) Neigungssensor. Diese Klasse sollen vollständig von Ihnen implementiert werden.

- Implementieren Sie die Klasse `Orientation` für den Zugriff auf den (simulierten) Neigungssensor
- Lesen Sie die Werte des Neigungssensors in der Klasse `Pong`
- Bewegen Sie den vom Spieler gesteuerten Schläger entsprechend den Werten des Sensors
- Implementieren Sie eine kurze Vibration bei Kollision des Balles mit der Wand oder einem Schläger, sowie eine längere Vibration bei einem Punkt
- (**Freiwillige Zusatzaufgabe:** Der von der CPU gesteuerte Schläger ist nicht sehr intelligent programmiert. Wenn Sie wollen können Sie die KI des Spiels optimieren.
- Testen Sie Ihre App mit dem SensorSimulator. (Obwohl das Pong-Projekt mit der Android Version 2.2. angelegt wurde sollten Sie für den Test ein *Virtual Device* mit der Android Plattform 2.3.3 zusammen mit dem SensorSimulator verwenden.)

Sie sollten bereits in der Vorbereitungsphase von Versuch 2 mit der Programmierung des Pong-Spielen beginnen und diese dann in der Vorbereitungsphase von Versuch 3 fertigstellen!