1. Erfassung und Verarbeitung von Sensordaten

Lernziele:

- Typische in mobilen Geräten enthaltene Sensorarten kennen,
- Daten von solchen Sensoren empfangen und verarbeiten können und
- Filter- und Schätzmethoden für Sensordaten kennen und einsetzen können.

Sensor

Ein *Sensor* ist ein technisches Bauteil, das bestimmte physikalische (oder chemische) Eigenschaften seiner Umgebung qualitativ oder als Messgröße quantitativ erfassen kann.

- Physikalische (oder chemische) Eigenschaften werden in ein elektrisches Signal umgewandelt.
- Abtastung des elektrischen Signals und
- Erzeugung eines Datenstroms der Messwerte.
- in der Regel numerische Messwerte für eine physikalischen Größe

Passive und aktive Sensoren

- Ein *passiver Sensor* benötigt keine Hilfsenergie; auf Basis des technischen Prinzips wird ein elektrisches Signal erzeugt.
  Beispiel: dynamisches Mikrofon

- Bei einem *aktiven Sensor* wird Hilfsenergie benötigt. Die Messung erfolgt, indem physikalische Eigenschaften eine Veränderung der Parameter von passiven Bauteilen bewirken.
  Beispiel: Widerstandsthermometer

Arten von Sensoren in mobilen Geräten

- verschiedene Arten von Beschleunigungssensoren
- Lagesensor
- Gyroskop
- Lichtsensor
- Entfernungsmesser
- Magnetometer

- Thermometer
- Barometer
- Hygrometer
- Lokalisierung
- Kamera und Mikrofon
- Tastatur

Anbindung weiterer Sensoren durch Ad-hoc Netzwerke (z.B. Bluetooth) möglich.

Aktoren

Gegenstück zu Sensoren, zur Erzeugung von physikalischen Effekten

In mobilen Geräten:

- Bildschirmanzeige (Views)
- Lautsprecher
- Vibrator

Sensor-Manager

Klasse `android.hardware.SensorManager`

- Schnittstelle zu den Sensoren
- Eine Referenz auf den Sensor-Manager erhält man durch:

```
SensorManager sensorManager =
    (SensorManager)getSystemService(Context.SENSOR_SERVICE);
```

- Sensor-API nicht stabil, unbedingt Online-Dokumentation heranziehen

Die Klasse `Sensor`

Klasse `android.hardware.Sensor`

- definiert Konstanten für Sensorarten
- verschiedene öffentliche Methoden für Informationen über einen Sensor: Messbereich, Auflösung, Stromverbrauch, etc.

  `public float getMaximumRange ()`
  maximum range of the sensor in the sensor's unit.

  `public int getMinDelay ()`
  the minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.

`public String getName ()`
name string of the sensor.

`public float getPower ()`
the power in mA used by this sensor while in use.

`public float getResolution ()`
resolution of the sensor in the sensor's unit.

`public int getType ()`
generic type of this sensor.

`public String getVendor ()`
vendor string of this sensor.

`public int getVersion ()`
version of the sensor's module.

Arten von Sensoren in Android

definiert in `android.hardware.Sensor`:

- `TYPE_ALL`

  alle Sensoren

- `TYPE_ACCELEROMETER`

  Beschleunigungssensor in drei Achsen, Einheit m/s^2; misst die Kräfte, die auf das Gerät wirken, daher Messwerte von 1g in Richtung Erdmittelpunkt.

- `TYPE_GRAVITY`

  Beschleunigungssensor zur Messung der Gravitationskraft in drei Achsen, Einheit m/s^2

- `TYPE_LINEAR_ACCELERATION`

  Beschleunigungssensor zur Messung der Kräfte außer der Gravitationskraft, die auf das Gerät wirken, Einheit m/s^2

  Es gilt: acceleration $=$ gravity $+$ linear-acceleration

- `TYPE_MAGNETIC_FIELD`

  Erdmagnetfeld in drei Achsen, Einheit Mikrotesla (μT)

- `TYPE_GYROSCOPE`

  Gyroskop, misst die Drehgeschwindigkeit entlang drei Achsen, Einheit rad/s.

- `TYPE_ROTATING_VECTOR`

  Repräsentiert die Ausrichtung des mobilen Geräts durch Angabe einer Drehachse und eines Drehwinkels

- `TYPE_ORIENTATION`

  Orientierung des Gerätes in Grad entlang drei Achsen; deprecated

- `TYPE_LIGHT`

  Lichtsensor, Umgebungslicht, Einheit Lux (lx)

- `TYPE_RELATIVE_HUMIDUTY`

  relative Luftfeuchtigkeit in Prozent

- `TYPE_AMBIENT_TEMPERATURE`

  Temperaturmessung, Einheit Celsius

- `TYPE_PROXIMITY`

  Entfernungsmessung, Einheit Zentimeter (cm).

Sensoren ermitteln

In der Klasse SensorManager folgende Methoden verwenden:

- Ermittlung aller Sensoren eines Typs:
  `List<Sensor> getSensorList(int type)`
- alle Sensoren:
  `List<Sensor> getSensorList(TYPE_ALL)`
- Standardsensor eines Typs ermitteln:
  `Sensor getDefaultSensor(int type)`

Sensoreigenschaften

- Genauigkeit

  Wie hoch ist der mittlere Messfehler? `SENSOR_STATUS...`

- Änderungsrate

  Wie häufig können Messwerte aktualisiert werden? `SENSOR_DELAY...`

Vordefinierte Konstanten als Werte für beide Eigenschaften in der Klasse `SensorManager`.

Genauigkeit

Konstanten in der Klasse `SensorManager`:

- `SENSOR_STATUS_ACCURACY_HIGH`

  Sensor arbeitet mit höchster Genauigkeit.

- `SENSOR_STATUS_ACCURACY_MEDIUM`

  Sensor arbeitet mit mittlerer Genauigkeit, eine Kalibrierung kann die Messgenauigkeit verbessern.

- `SENSOR_STATUS_ACCURACY_LOW`

  Sensor arbeitet mit geringer Genauigkeit, eine Kalibrierung erscheint notwendig.

- `SENSOR_STATUS_UNRELIABLE`

  Sensor liefert unzuverlässige Werte, eine Kalibrierung ist notwendig oder es können keine Sensordaten gelesen werden.

Änderungsrate

Konstanten in der Klasse `SensorManager`:

- `SENSOR_DELAY_FASTEST`

  höchstmögliche Änderungsrate

- `SENSOR_DELAY_GAME`

  Änderungsrate geeignet für Spiele

- `SENSOR_DELAY_NORMAL`

  Standard-Änderungsrate

- `SENSOR_DELAY_UI`

  Änderungsrate geeignet für Benutzerschnittstelle

Sensordaten empfangen (1)

- via Schnittstelle `android.hardware.SensorEventListener`
- Achtung: `SensorListener` ist deprecated!
- `void onSensorChanged (SensorEvent event)`

  Neue Messdaten liegen für einen Sensor vor. Sensor und Werte werden durch ein Objekt der Klasse `android.hardware.SensorEvent` beschrieben.

- `void onAccuracyChanged (Sensor sensor, int accuracy)`

  Die Genauigkeit für den Sensor `sensor` hat sich gändert. `accuracy` ist der neue Genauigkeitswert.

Sensordaten empfangen (2)

Registrierungsmethoden für den `SensorEventListener` in der Klasse `SensorManager`:

- `public boolean registerListener(SensorEventListener listener, Sensor sensor, int rate)`

  `listener` erhält von `sensor` Daten mit Änderungsrate `rate`

  Liefert `true`, wenn der Sensor unterstützt und verfügbar ist.

- `public boolean registerListener (SensorEventListener listener, Sensor sensor, int rate, Handler handler)`

  wie oben, aber die `SensorEvents` werden über `handler` verteilt

- `public void unregisterListener (SensorEventListener listener)`

  `listener` von allen Sensoren abmelden

- `public void unregisterListener (SensorListener listener, Sensor sensor)`

  `listener` von `sensor` abmelden

Sensordaten

Klasse android.hardware.SensorEvent mit `public` Instanzvariablen statt Getter-Methoden:

- `public Sensor sensor`

  Der Sensor, der das Ereigenis ausgelöst hat.

- `public long timestamp`

  Zeit in Nanosekunden, zu der das Ereignis eintrat.

- `public final float[] values`

  Werte für die Sensoren, hängen vom Sensortyp ab.

- `public int accuracy`

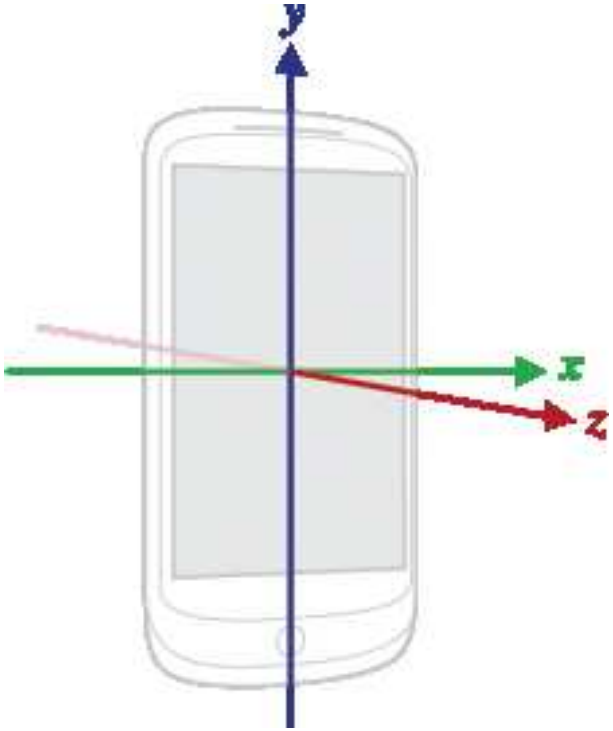
  Genauigkeit

Beschleunigungssensor

- Beschleunigung entspricht der Änderung der Geschwindigkeit pro Zeiteinheit. Erste Ableitung der Funktion $v(t)$.
- Beschleunigungsmesser können nicht zwischen Beschleunigung durch Gravitation oder durch Bewegung unterscheiden. Konsequenz: Beschleunigung von $-9.8 m/s^2$ im Ruhezustand in Richtung der y-Achse der Zeichnung (vertikal).
- Konstante für Erdbeschleunigung:
  `SensorManager.STANDARD_GRAVITY`
- Messung der Beschleunigung erfolgt für drei Achsen.

Wenn das Gerät aufrecht steht:

- lateral: links (+) / rechts (–)
  x-Achse
- vertikal: hoch (+) / runter (–)
  y-Achse
- longitudinal: näher (+) / weiter (–)
  z-Achse


y
x
z

Zugriff auf die Werte im Feld `values` nach Konvention:

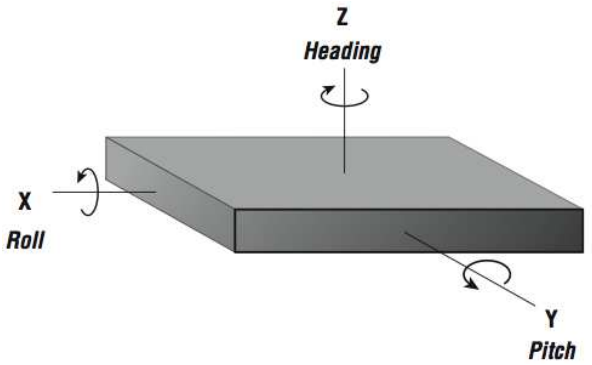
- `values[0]: Beschleinigung in x-Richtung minus Gravitation entlang der x-Achse`
- `values[1]: dto. für y-Achse`
- `values[2]: dto. für z-Achse`

Man beachte:

- In Ruhe und obiger Position gilt $\texttt{values}[1] = g$.

Lagesensor (Orientation)

- Roll, beschreibt ein Kippen des Gerätes nach links (+) oder rechts (–)
- Pitch, beschreibt ein Kippen des Gerätes nach vorne (+) oder hinten (–)
- Heading, Bearing, Rotation um die z-Achse der Zeichnung,
  Heading alleine stellt einen Kompass dar.



Kombination aus Kompass (Heading) und Beschleunigungssensor um Pitch und Roll zu ermitteln.

Achsen sind in Geräten wiederum anders als in der Zeichnung:

- `values[0]: heading`
- `values[1]: pitch`
- `values[2]: roll`

Virtueller Sensor

- Der Orientierungssensor ist in den meisten Geräten ein sogenannter *virtueller Sensor*.
- Die physikalischen Werte (hier Winkel) werden nicht direkt gemessen, sondern aus anderen Sensorwerten durch ein Modell abgeleitet.
- Die Lage wird z.B. aus den Messwerten des Beschleunigungssensors und des Sensors für das Erdmagnetfeld ermittelt.
- Android stellt die Methoden zur Berechnung der Lage öffentlich zur Verfügung.

Lageermittlung in Android

- Die Repräsentation der Lage eines mobilen Gerätes erfolgt mit Hilfe einer sogenannten *Drehmatrix* bzw. *Rotationsmarix*.
- Solch eine Matrix beschreibt eine Drehung in einem euklidischen Raum, hier im $\mathbb{R}^3$.
- Android stellt in der Klasse SensorManager eine Methode bereit, um eine Rotationsmatrix zu berechnen, die die aktuelle Lage des Gerätes beschreibt.
- Eingabe: Messwerte des Beschleunigungs- und des Erdmagnetfeldsensors

- Mit Hilfe dieser Rotationsmatrix kann über eine weitere Methode die Lage des Gerätes, beschrieben durch Winkel (Heading, Pitch, Roll), ermittelt werden.

Drehung/Rotation

Eine *Drehung* ist eine lineare Abbildung, die längentreu, winkeltreu und orientierungserhaltend ist.

- Kongruenzabbildung: Form und Größe werden nicht verändert
- Affine Abbildung: Kollinearität, Parallelität und Längenverhältnisse bleiben erhalten

Drehmatrix

Im $\mathbb{R}^2$ wird eine Drehung um den Ursprung mit Winkel α durch eine Drehmatrix beschrieben:

$$\mathrm{R}(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

Drehmatrizen haben eine Reihe von interessanten Eigenschaften (allgemein, nicht nur im $\mathbb{R}^2$)

Eigenschaften von Drehmatrixzen

- Die Spalten einer Drehmatrix bilden ein Orthogonalsystem, d.h., sie stehen paarweise senkrecht aufeinander.
- Zwei Vektoren x und y sind orthogonal, wenn

$$< x, y >= \sum_{i=1}^{n} x_i \cdot y_i = 0$$

  gilt.
- Die Länge der Spaltenvektoren ist gleich 1, dies entspricht der Längentreue.

- Insgesamt bilden die Spalten damit ein Orthonormalsystem.
- $\det \mathrm{R}(\alpha) = 1$, dies entspricht der Orientierungserhaltung.
- Für die Inverse einer Drehmatrix gilt:

$$\mathrm{R}^{-1}(\alpha) = (\mathrm{R}(\alpha))^{\mathsf{T}}$$

Damit im $\mathbb{R}^2$:

$$\mathrm{R}^{-1}(\alpha) = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix}$$

Rotationsmatrizen in 3D

- Während in 2D eine Drehung durch einen Winkel beschrieben wird, benötigen wir für 3D drei Winkel, da wir insgesamt drei Paare von Achsen haben.
- Diese Winkel werden auch als Eulersche Winkel bezeichnet.
- Wir können uns die Lage eines Körpers vorstellen, als drei nacheinander ausgeführte Drehungen um verschiedene Achsen.
- Jede Achsendrehung hat ihre eigene Rotationsmatrix.

- Drehung um die x-Achse um den Winkel γ:

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix}$$

- Drehung um die y-Achse um den Winkel β:

$$R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}$$

- Drehung um die z-Achse um den Winkel α:

$$\mathrm{R}_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Durch Multiplikation der Matrizen entsteht die Gesamtrotationsmatrix R, z.B.

$$\mathrm{R} = \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix}$$

Ermittlung der Drehwinkel

- Hat man eine Rotationsmatrix R, kann man aus den r_{ij} die entsprechenden Winkel ermitteln.
- Die Auflösung kann auf verschiedene Weise erfolgen, man sollte auf numerische Stabilität achten.
- Beispiel:

$$\begin{aligned} \beta &= \arcsin(-\mathrm{r}_{31}) \\ \alpha &= \arcsin(\frac{\mathrm{r}_{21}}{\cos\beta}) \\ \gamma &= \arcsin(\frac{\mathrm{r}_{32}}{\cos\beta}) \end{aligned}$$

- Es sind aber auch andere Formeln möglich, siehe Literatur.

Ermittlung der Rotationsmatrix

- Lage des Koordinatensystems: x-Achse nach Osten, y-Achse nach Norden, z-Achse zum Himmel
- Rotationsmatrix R ergibt sich aus:

$$R \cdot \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix}$$

und

$$I \cdot R \cdot \begin{pmatrix} m_x \\ m_y \\ m_z \end{pmatrix} = \begin{pmatrix} 0 \\ m \\ 0 \end{pmatrix}$$

- Hierbei sind a_x, a_y, a_z die Beschleunigungsmesswerte in Richtung der drei Achsen und m_x, m_y, m_z die Messwerte für das Erdmagnetfeld.

- I ist eine simple Rotationsmatrix, die die Messwerte des magnetischen Feldes in das gleiche Koordinatensystem überführt, wie das zur Beschleunigungsmessung.

Android-Methoden

- `public static boolean getRotationMatrix (float[] R, float[] I, float[] gravity, float[] geomagnetic)` Muss zuerst aufgerufen werden. R wird vorher allokiert (als Feld) und die Werte werden dort zeilenweise abgelegt.

  `gravity` und `geomagnetic` sind die Felder mit Messwerten von den Sensoren.

- `public static float[] getOrientation (float[] R, float[] values)`

  Zur Ermittlung der Winkel (im Bogenmaß), das Feld `values` wird gefüllt.

Berechnungen für Sensorwerte

- In vielen Fällen ist man nicht an Beschleinungswerten (oder Winkelgeschwindigkeiten) interessiert,
- sondern man nutzt diese Daten, um Geschwindigkeit oder Position zu bestimmen.
- Beispiel Koppelnavigation: Bestimmung des Standortes zum Zeitpunkt t aus Geschwindigkeit v_0 zum Zeitpunkt t_0 und Beschleunigungswerten zwischen t_0 und t.
- Richtungsänderungen können durch ein Gyroskop erfasst werden.
- Einsatz z.B. in Navigationssysteme, wo kein GPS-Empfang (Tunnel)

Geschwindigkeit und Strecke: Theorie

- Gegeben sei die Beschleunigung $a : [t_0, t_1] \rightarrow \mathbb{R}$, die für jeden Zeitpunkt zwischen t_0 und t_1 einen Wert für die Beschleunigung angibt.
- Wir betrachten die Beschleunigung zunächst als skalaren Wert (eindimensional).
- Dann gilt für die Geschwindigkeit $v(t)$:

$$v(t) = v_0 + \int_{t_0}^{t} a(x)\,dx$$

- v_0 ist die Geschwindigkeit zum Zeitpunkt t_0.

- Weiterhin gilt für die zurückgelegte Strecke $s(t)$:

$$s(t) = s_0 + \int_{t_0}^{t} v(x)dx$$

- s_0 ist die zum Zeitpunkt t_0 zurückgelegte Strecke.

Beispiel: Geschwindigkeit und Strecke

Beispiel 1.1. $a(t) = 5, t_0 = 0, v_0 = 0, s_0 = 0.$

Dann erhalten wir

$$v(t) = \int_0^t 5 dx = 5t$$

$$s(t) = \int_0^t 5x dx = \frac{5}{2}t^2$$

Bei einer konstanten Beschleunigung von $a(t) = 5m/s^2$ hat mal also nach $t = 3$ Sekunden eine Geschwindigkeit von $15m/s$ und bis dahin $22.5m$ zurückgelegt.

Geschwindigkeit und Strecke: Praxis

- Die Funktion $a(t)$ ist nicht ermittelbar, da Beschleunigungswerte nur zu diskreten Zeitpunkten vom Sensor gemessen werden.
- Dementsprechen können wir die obigen Formel nur stückweise anwenden und müssen ansonsten addieren.
- Im einfachsten Fall nehmen wir dazu an, dass zwischen zwei Messzeitpunkten t_i und t_{i+1} die konstante Beschleunigung $a(t_i)$ vorliegt.

- Es sei $a_i, i = 0, 1, 2, \ldots$ die Beschleunigung zum Zeitpunkt t_i und v_0, s_0 siehe oben. Dann schätzen wir für $i = 1, 2, \ldots$:

$$
\begin{aligned}
v(t) &= v_{i-1} + \int_{t_{i-1}}^{t} a(x)\,dx \qquad \text{für } t \in [t_{i-1}, t_i] \\
&= v_{i-1} + (t - t_{i-1}) \cdot a_{i-1} \\
v_i &= v_{i-1} + (t_i - t_{i-1}) \cdot a_{i-1} \\
s_i &= s_{i-1} + \int_{t_{i-1}}^{t_i} v(x)\,dx \\
&= s_{i-1} + \int_{t_{i-1}}^{t_i} v_{i-1} + (x - t_{i-1}) \cdot a_{i-1}\,dx \\
&= s_{i-1} + (t_i - t_{i-1}) \cdot v_{i-1} + \frac{a_{i-1}}{2}(t_i - t_{i-1})^2
\end{aligned}
$$