hochschule mannheim

Fakultät Informationstechnik
Studiengang Technische Informatik

Bachelorarbeit

Entwicklung eines Frameworks zur Darstellung von Smartphone-Sensordaten für die didaktische Unterstützung von Programmiervorlesungen

Marius Cerwenetz

Abschlussarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

Vorgelegt von	Marius Cerwenetz
am	XX. Juli 2022
Referent	Prof. Dr. Peter Barth
Korreferent	Prof. Dr. Jens-Matthias Bohli

Schriftliche Versicherung laut Studien- und Prüfungsordnung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mannheim, XX. Juli 2022

Marius Cerwenetz

Zusammenfassung

Programmierenlernen fällt besonders am Anfang schwer. Embeddedprojekte erlauben mit vergleichsweise wenig Aufwand einen gelungen Einstieg mit effektiver Lernerfahrung. Solche Projekte benötigen allerdings viel Peripherie und Hardware. Diese benötigt wiederrum eine nicht niedrigschwellige Erfahrung zum Beispiel im Umgang mit Microcontrollern. Smartphones haben diese Nachteile nicht bieten allerdings trotzdem einen hohen Umfang an Sensoren.
In dieser Arbeit wird ein Framework erstellt, dass das Smartphone nutzt um mit dem Microcontroller kleine Softwareprojekte umzusetzen. Kleine Aufgabenstellungen mit Musterlösungen werden ausgearbeitet und mitgereicht.

Inhaltsverzeichnis

Kapitel 1

Einführung

MINT-Berufe leiden hierzulande unter einem akuten Fachkräftemangel. Das Institut der deutschen Wirtschaft ermittelte für April 2021 ein Unterangebot von 145.100 Personen [**?**] in 36 MINT-Berufskategorien. Digitalisierungs-Projekte geraten dadurch ins Stocken.
Nicht zuletzt er auch ein Kräftemangel in der Softwareentwicklung. Es fehlen Programmiererinnen und Programmierer. Softwareentwicklung ist gerade in der Lernphase nicht trivial und abstrakt. Unlebendige Übungsaufgaben die beispielsweise Konsolenein- und ausgaben realisieren schrecken zukünftige Programmierinnen und Programmierer eher ab als sie zu ermutigen.
Microcontroller sind bereits eine große Hilfe, da hier spielerisch kleine Projekte realisiert werden können. So können schon früh in Schulen Kinder an die Programmierung herangeführt werden. Sie lernen spielerisch kleine Programme zu entwickleln und verstehen die ihnen beigebrachten Abläufe durch schnelle Anwendung. Microcontroller sind jedoch auch mit Anschaffungskosten verbunden und für kleine Anwendungen, welche Sensoren verwenden, wird viel zusätzliches Material wie zum Beispiel Breadboards, Verbindungskabel und Erweiterungsboards benötigt. Moderne Smartphones bieten hier Abhilfe da Sie meistens mit verschiedenen Sensoren bespickt sind, wie zum Beispiel: Kompass, GPS, Microphon oder Kamera. Viele Kinder besitzen bereits mit 10 Jahren [**?**] ein Smartphone.
Im Rahmen dieser Arbeit soll ein Framework entwickelt werden, dass das Smartphone von Anwendern einbindet um Sie beim Programmierenlernen zu unterstützen. Dieses orientiert sich ganz an bereits bestehenden System wie Arduino und Microbit. Kleine Interaktionsmöglichkeiten werden geschaffen.

Benötigt werden dafür eine Library zum Einbinden, eine Python-Anwendung und eine mobile Anwendung für Android Smartphones.

Android-App

Die Anwendung besteht aus einem Textfeld, einer Ausgabe-LED, einer Signal LED für Nutzereingaben und zwei Buttons. Auf dem Textfeld können verschiedene Ausgaben präsentiert werden. Die Ausgbale LED kann vom Computer aus an und ausgeschaltet werden. Die Signal-LED dient dazu dem Nutzer zu signalisieren, ob sich das Smartphone gerade in einem Aufnahmeprozess befindet. Die Buttons werden verwendet um auf dem PC Steuerbefehle auszuführen.
Neben den Ein- und Ausgabemöglichkeiten im UI-Screen kann zur Eingabe auch noch das Microphon und das accelerometer genutzt werden, sofern vorhanden. Als Ausgabemöglichkeiten kommen der Vibrationsmotor sowie der Lautsprecher des Smartphones zum Einsatz.

Kapitel 2

Experimente/Aufgaben

Aufgaben

Dieses Kapitel enthält verschiedene Beispielaufgaben die mit dem Framework gelöst werden sollen. Die Benutzung der API dazu wird später geschildert.

Disco
Die LED muss ganz schnell blinken.

Diebstahl-Alarm
Wenn nach dem Telefon gegriffen wird soll die LED blinken. Wenn man sich davon entfernt soll sie wieder aus sein.

Würfeln
Das Smartphone wird geschüttelt. Ein random Zahlenwert wird zurückgegeben. Je nachdem welches Ergebnis zurückkommt soll ein Wert auf dem Display angezeigt werden.

Klatsch-Zähler
Der Anwender möchte wissen wie oft in einem bestimmten Zeitraum geklatscht wurde. Ein Methodenaufruf startet in der Androidanwendung eine Activity, die innerhalb des im Argument genannten Zeitraums die Anzahl der maximalen Lautstärkeamplituden des Mikrophons misst und die Anzahl zurückgibt. Diese soll im Textfeld angezeigt werden.

Dreh-Zähler
Ein Nutzer möchte in einem bestimmten Zeitraum zählen wie oft das Smartphone

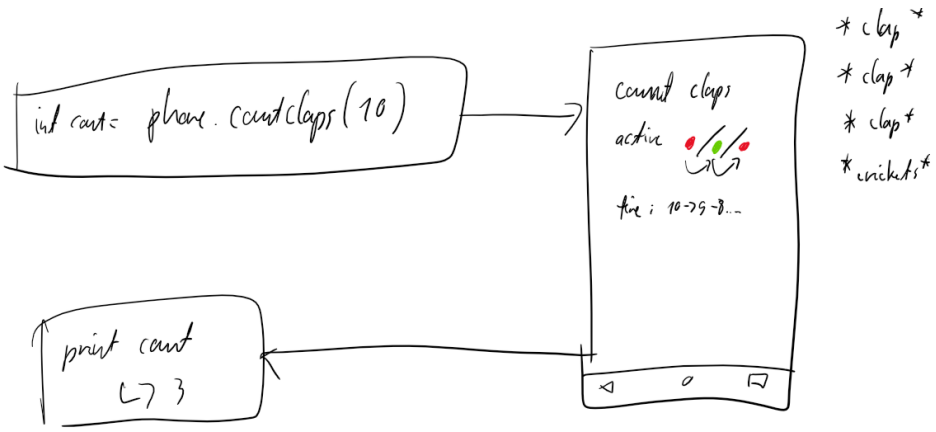

Abbildung 2.1: Klatsch-Zähler

gedreht wurde.

API

Gelöst werden sollen die Aufgaben durch das Aufrufen der API. Diese bietet die benötigten Funktionen an. Die Aufrufe sind frei miteinander kombinierbar, so dass Aufgaben erweitert werden können.

Eingaben

Auslesen der accelerometer-Daten
Ein User will den Wert der X, Y und Z Koordinaten des Smartphones wissen. So kann er bspw. feststellen, ob das Smartphone gerade nach unten, oben oder horizontal bewegt wurde. Kippbewegungen werden nicht detektiert.

```
1 Phone p;
2 float x_val = p.get_x_accell();
```

Lautstärkepegelmessung
Ein User möchte den aktuellen Lautstärkepegel messen.

```
1 Phone p;
2 float volume = p.getVolume();
```

Annäherungssensor-Messung
Ein User möchte wissen, ob ein Objekt unmittelbar vor den Annäherungssensor steht. Der Aufruf erfolgt folgendermaßen.

```
Phone p;
bool triggered = p.primity_triggerd();
```

Amplituden-Spike-Messung mit Zeitraum
Ein User möchte wissen, wie oft die Lautstärke innerhalb eines angegebenen Zeitraums t ein gewisses Limit überstiegen hat. Der Aufruf könnte dabei folgendermaßen laufen.

```
Phone p;
int timeframe = 1000; //1000 ms = 1s
int num = p.getNumOfSpikes(timeframe);
```

Umdrehungsmessung mit Zeitraum
Ein User möchte wissen, wie oft das Smartphone innerhalb eines angegebenen Zeitraums t gedreht wurde. Der Aufruf sieht folgendermaßen aus.

```
Phone p;
int timeframe = 1000; //1000 ms = 1s
int num = p.getNumOfSpikes(timeframe);
```

Ausgaben

Text-Ausgabe
Um auf dem Smartphone einen beliebigen Text anzuzeigen. Dies kann er zum Beispiel wie im folgenden Beispielcode machen.

```
Phone p;
p.displayText("Hallo Welt");
```

LED leuchten lassen
Ein User möchte eine LEDäuf dem Display ansteuern. Dies kann er so machen.

```
Phone p;
p.led_on();
p.led_off();
```

LED toggeln
Ein User möchte den Zustand einer LED auf dem Display auf äusßtellen wenn Sie an war und auf än"wenn Sie aus war. Dies kann er so machen.

```
Phone p;
p.led_on();
p.led_off();
```

Vibrationsauslöser
Ein Nutzer möchte das Smartphone für eine gewisse Zeitspanne vibrieren lassen.

```
1 Phone p;
2 p.vibrate(500);
```

Android Anwendung

Für Ausgaben existiert eine Android Anwendung die verschiedene Interaktionsmöglichkeiten bereitstellt. Das Userinterface wird in Grafik **??** gezeigt.
Bereit stehen eine LED bzw. ein Button, eine Textausgabezeile und eine Checkbox. Der Button ist nicht drückbar. Eine Eingabe ist nicht vorgesehen. Er fungiert als Signal-LED die getoggelt werden kann.
In der Textausgabe können beliebige Texte ausgegeben werden.
Bei der checkbox kann der Haken gesetzt und wieder entfernt werden. Nachdem der Zustand geändert wurde wird eine Antwort über den aktuellen Zustand der Checkbox zurückgegeben.

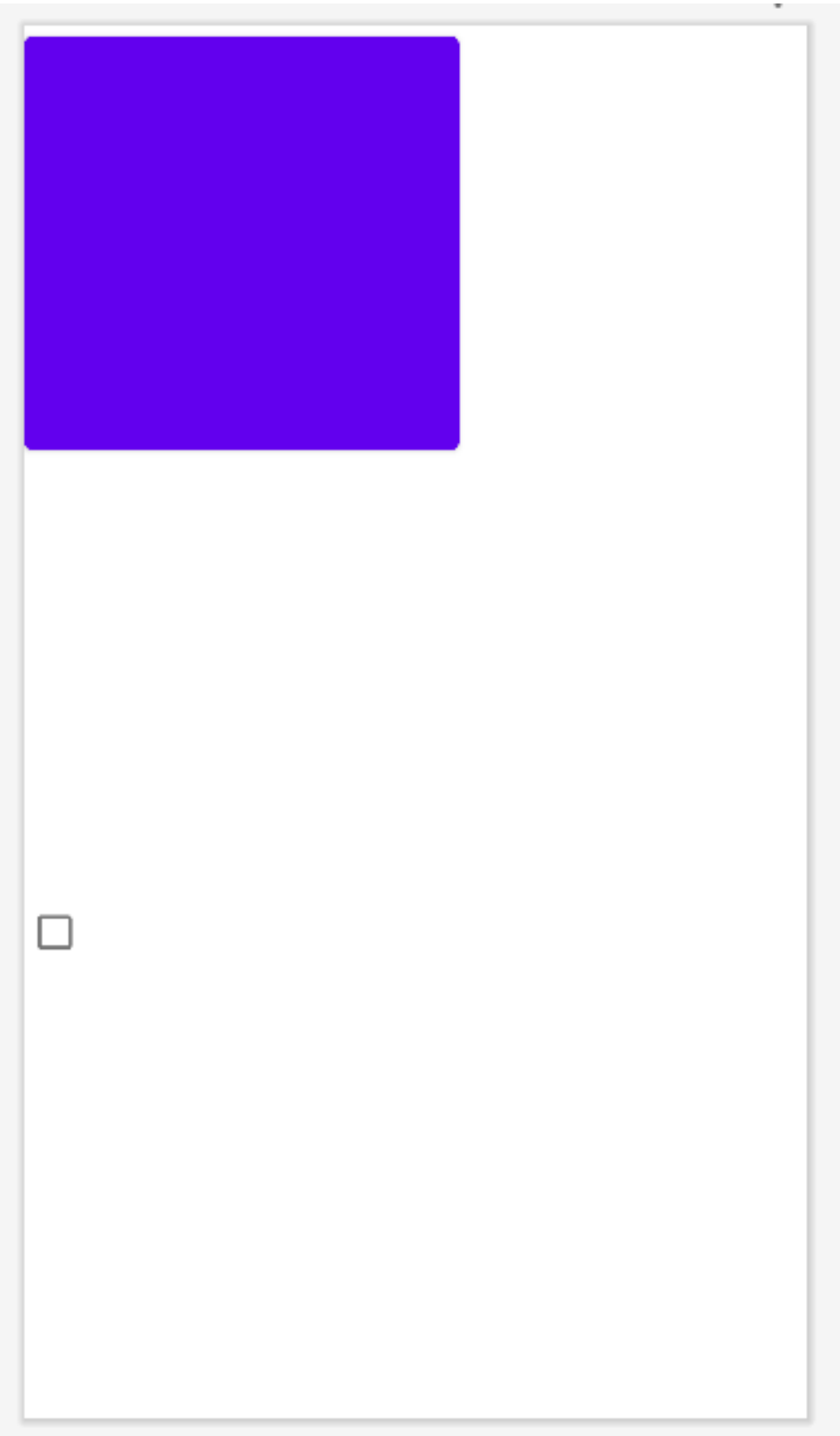

Abbildung 2.2: Android App UI

Kapitel 3

Architektur

Der Aufbau des Frameworks besteht aus einer mobilen Anwendung für Android-Smartphones, Bibliothek in der Programmiersprache C und einer Middleware die den Austausch Koordiniert.
Clientseitig erfolgen sämtliche Aufrufe die Sensordaten abgreifen oder Steueranfragen senden immer erst per UDP über die Middleware.
Die Middleware sammelt, sobald sich eine Android-Anwendung bei ihr anmeldet, gewisse Sensordaten und speichert Sie intern zwischen, damit Sie schnell vorrätig vorliegen. Die Speicherung erfolgt in einem seperaten Thread, der die letzten Zustände der Sensordaten hält. Werden Sensordaten abgerufen werden Sie aus der Liste entfernt. Sind noch keine Daten vorhanden, oder wird ein Sensor angefragt der im Smartphone nicht existent ist, wird ein Fehlercode zwischengespeichert. Die Verfügbarkeit der jeweiligen Sensoren wird beim Start der Anwendung ermittelt.
Steueraufrufe werden beim Aufruf über ein seperates Topic versandt. Der Austausch erfolgt auf mehreren Topics, da manche Nachrichten, wie Steuerbefehle wie in **??** erwähnt, mit einem höheren QOS-Level versendet werden müssen als im Moment existente Sensordaten die nur eine kurzzeitige Relevanz besitzen und deren Zustellung nicht obligatorisch ist. Auf diesem Topic werden Nachrichten mit der QOS von 2 versandt auf dem für reguläre Sensordaten mit einer QOS von 0.

Android Anwendung

Die Android-Anwendung läuft auf dem Smartphone des Anwenders. Sie besteht aus einer Haupt-Activity deren außerlicher Aufbau in **??** beschrieben ist.
In der Haupt-Activity wird ein Service gestartet und gebunden. Der Service baut eine Verbindung zu einem MQTT Server auf. Dort verbindet er sich den beiden Topics. Die Activity bindet beim Start den MQTT-Service ein. Stellt ein Client über die Bibliothek

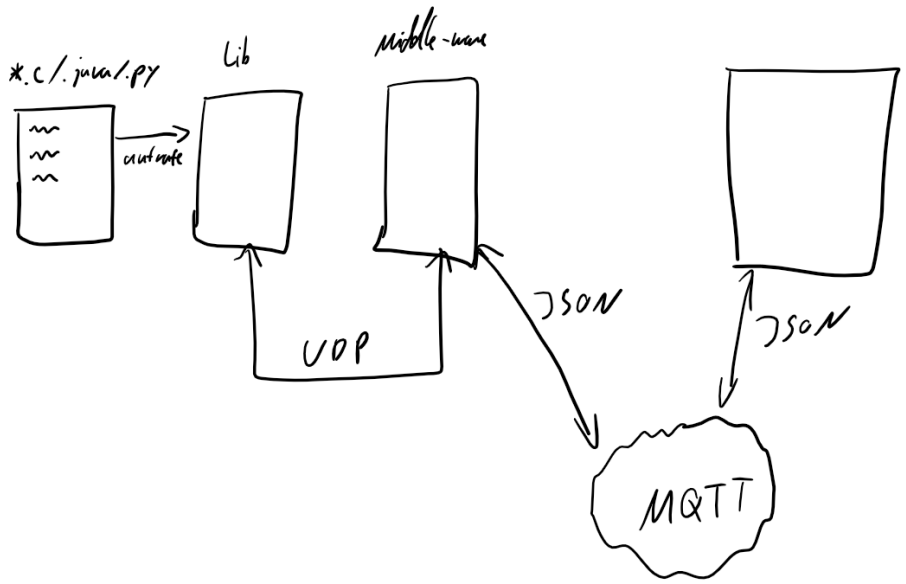

Abbildung 3.1: System-Aufbau

auf einem Computer eine Anfrage wird dieser Aufruf, wie z.B. eine Ausgabe auf einem Textfeld an das Smartphones, zuerst zur Middleware geleitet. Diese leitet die Anfrage weiter an das Smartphone, dass dann den jeweiligen Befehl ausführt. Dies beeinhaltet vor allem Ausgaben, sowie Eingaben die eine Nutzerinteraktion mit reduziertem Ergebnis. Zum Beispiel alle Eingaben die etwas in einem angegebenen Zeitraum messen. Falls ein Sensor angefragt wird der auf dem Smartphone nicht existiert wird eine Nachricht mit einer Fehlermeldung zurückgegeben.

Kommandos und Ausgaben

Für die Ausgabe auf dem Smartphone sind verschiedene Kommandos definiert. Diese sind der Tabelle **??** zu entnehmen. `CMD-Kürzel` beschreibt die Notation des Kürzels mit dem eine Aktion ausgeführt werden kann. Diese wird unter `Beschreibung` kurz zusammengefasst. `return` gibt an, ob der Aufruf des Requests eine Antwort rücksendet und somit auch, ob ein Aufruf der Funktion in der Library blockiert oder nur sendet.

Sensoren

Smartphones beeinhalten verschiedene Sensoren die Daten über die Umgebung erfassen können. In der Android Anwendung werden folgende Sensortypen verwendet:

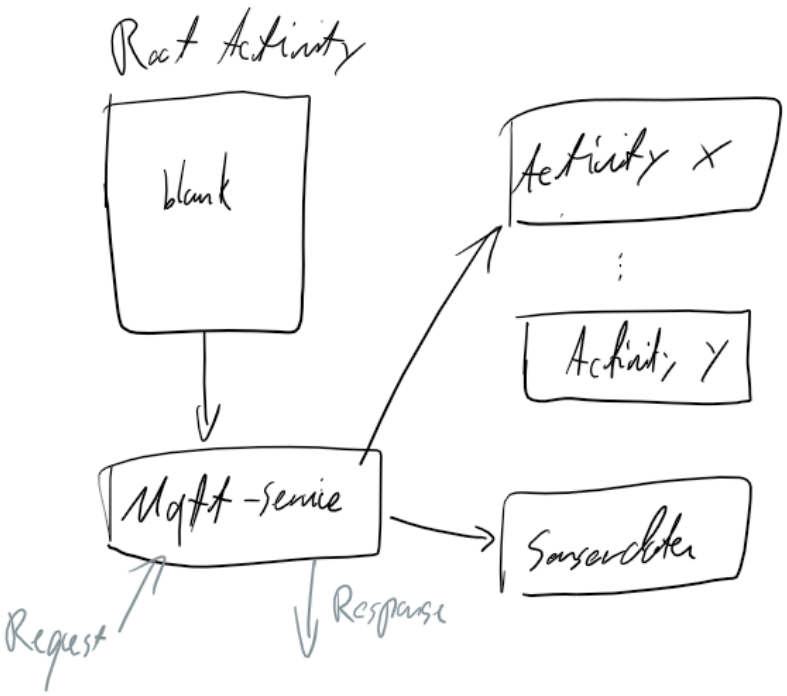

Abbildung 3.2: Android-App-Aufbau

CMD-Kürzel	Beschreibung
textview	Setzt `value` als Text im Textvie
button • false : setzt die Farbe auf Rot • toggle : Kehrt die Farbe um	Setzt Button auf value. Dies kann sein • true : setzt die Farbe auf Gr False

Tabelle 3.1: Kommando-Kürzel mit Beschreibung

- Lineare Beschleunigungssensoren
- Microfon
- Annäherungssensor
- Gyroskop

Lineare Beschleunigungssensoren

Beschleunigungssensoren messen die Beschleunigung in m/s^2 für die drei Bewegungsrichtungen: X-, Y- und Z-Achse in einem festgelegten Zeitraum. Eine Übersicht über die Anordungen der drei Axen ist in Grafik **??** zu sehen. Die X-Achse verläuft horizontal durch das Display des Smartphones hindruch, die Y-Achse vertikal und die Z-Achse durchschneidet das Smartphone in die Tiefe.

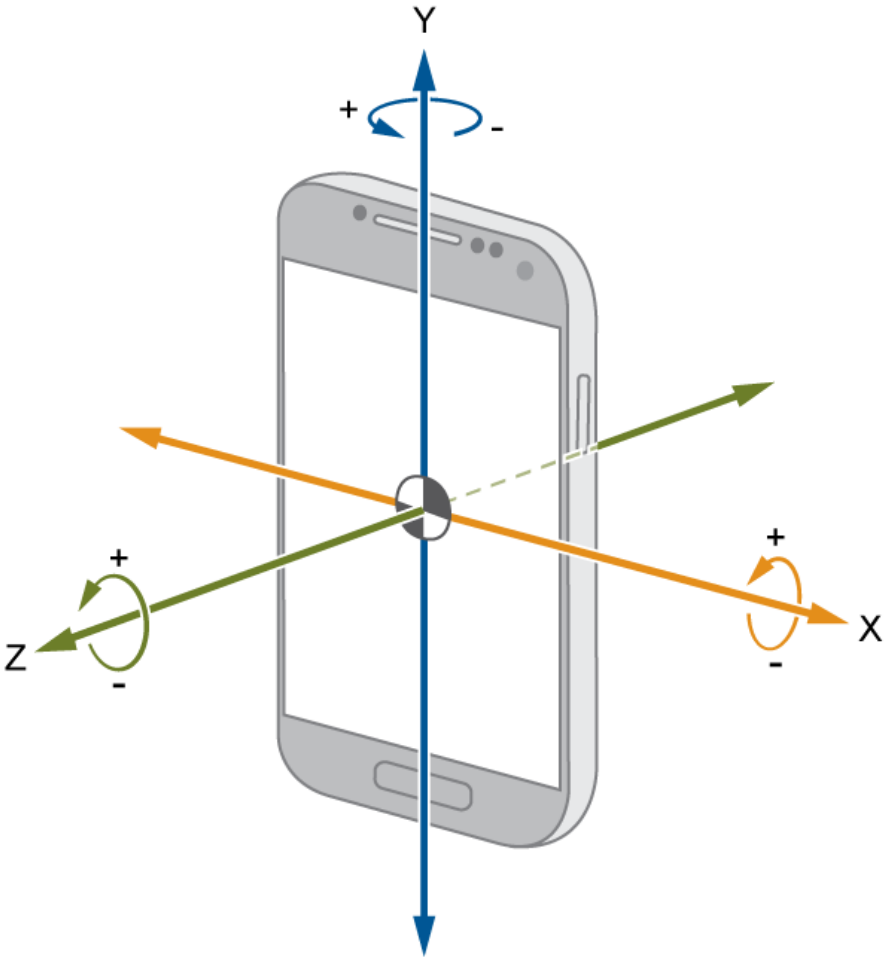

Abbildung 3.3: Android-Koordinatensystem

Die Beschleunigungsdaten werden dabei in einem festen Zeitraum gemessen. Hierfür stehen vier Schnelligkeitsstufen in absteigender Frequenz bereit:

- SENSOR_DELAY_FASTEST : Kein Verzögerung. Verwendet die Frequenz des Sensors.
- SENSOR_DELAY_GAME : Verzögerung um 1ms
- SENSOR_DELAY_UI : Verzögerung um 2ms
- SENSOR_DELAY_NORMAL : Verzögerung um 3ms

Die mit der jeweiligen Taktrate aufgenommenen Beschleunigungssensordaten umfassen jedoch auch die Ergbeschleunigung g in Höhe von $9.81 m/s^2$. Diese muss für die bereinigten, realen Werte zuerst noch von den aufgenommenen Werten subtrahiert werden.
Hierfür muss zuerst der Gleichanteil der Graviation mittels eines Tiefpassfilters von den gemessenen Werten isoliert werden.
Der Algorithmus dafür ist folgendermaßen definiert:
Für eine unendliche Folge an isolierten Gravitationswerten g und Beschleunigungssensor-Messwerten e mit index $n \in \mathbb{N}$ gilt:

$$g_n = g_{n-1} \cdot \alpha + (1 - \alpha) \cdot e_n \tag{3.1}$$

Die Konstante α wird dabei folgendermaßen berechnet:

$$\alpha = t/(t + \Delta_t) \tag{3.2}$$

t ist dabei die Zeitkonstante des Tiefpass-Filters und Δ_t die Anpassungsrate.
Die Berechnung des bereinigten Beschleunigungssensorwerts ist durch einen Hochpass definiert:

Für eine unendliche Folge an ereinigten Beschleunigungssensor-Messwerte l, isolierten Gravitationswerten g und Beschleunigungssensor-Messwerten e mit index $n \in \mathbb{N}$ gilt:

$$l_n = e_n - g_n \tag{3.3}$$

Die Graviation könnte auch unter Zuhilfenahme des Gyroskops isoliert werden. Die wirkenden Kräfte auf die drei Achsen könnten je nach Neigung einzeln bemessen werden. Hierfür müsste die Masse des Smartphones jedoch bekannt und gleich verteilt sein. Deshalb wird hierfür auf die Pragmatische Hochpass-Tiefpass-Lösung zurückgegriffen.

Angaben für die Nachrichtenformate

Alle Sensordaten besitzen ein festgelegtes Kürzel zur Standardisierung des Nachrichtenverkehrs. Sie dienen vor allem der Adressierung der jeweiligen Daten in der Middleware und in der Library.
Die Sensortyp Kürzel sind in Tabelle **??** zu finden.

TYPE-Kürzel	Beschreibung
accell_x	Beschleunigungssensor für die X-Richtung
accell_y	Beschleunigungssensor für die Y-Richtung
accell_z	Beschleunigungssensor für die Z-Richtung

Tabelle 3.2: Sensor-Kürzel mit Beschreibung

Nachrichtenformate

Die Nachrichten werden im JSON-Format übertragen. Jede Nachricht beeinhaltet mindestens die Angabe eines Nachrichtenformat-Typs. Für die Aufrufe wurden verschiedene Nachrichtenformate definiert.
Insgesamt gibt es vier Nachrichtentypen:

1. sensor_request
2. sensor_update_request
3. rpc_request
4. rpc_response

sensor_requests werden vom Endanwender-PC per UDP an die MIddleware gesendet. Sie beeinhalten neben dem Typ auch noch das Feld `sensor_type`. Dieses definiert den Sensortyp für den eine Anfrage generiert wurde. Eine Übersicht ist in Listing zu finden.
Im Listing wurde der Platzhalter `TYPE` für alle möglichen Sensortypen angegeben. Eine vollständige Aufschlüsselung ist in Tabelle **??** zu finden.

```
1 {
2    "type":"sensor_request",
3    "sensor_type": "TYPE"
4 }
```

Listing 3.1: sensor_request

sensor_update_requests werden von Smartphone über MQTT an die Middleware versandt. Sie umfassen ebenfalls wie `sensor_requests` das Feld `sensor_type`, jedoch zusätzlich auch das Feld `value` in dem der gemessene Sensorwert gespeichert wird. Dieser wird von der Middleware in eine interne Datenstruktur eingetragen.

rpc_requests werden vom Endanwender-PC per UDP an die Middleware gesendet. Sie lösen Aktionen auf dem Smartphone aus wie zum Beispiel das anschalten der LED, oder das Starten von Messungen. Sie werden von der Middleware per MQTT an das Smartphone weitergereicht. Die Übermittlung erfolgt jedoch über ein seperates Topic mit einem Quality of Service Wert von 2 um eine garantierte Übertragung zu gewährleisten. Das Smartphone hat dieses TOPIC ebenfalls mit einer QOS von 2 abboniert. eine Liste der Tabelle **??** zu entnehmen.

rpc_requests und response beeinhalten außerdem mindestens ein `command`-Feld und ein `value`-Feld.

Nachfolgendes Listing zeigt beispielsweise den Aufruf die LED einzusschalten.

```
1 {
2    "type" : "rpc_request",
3    "command" : "set_button",
4    "value" : "true"
5 }
```

Folgende Nachricht beschreibt beispielsweise den übermittelten Sensorwert des accelerometers.

```
1 {
2    "type" : "sensor_response",
3    "sensor_type" : "accell_x",
4    "value" : "-0.151342"
5 }
```

Literaturverzeichnis

Online-Quellen