Gliederung der schriftlichen Arbeit - ALT

1. Einleitung
    1.1. Aufgabestellung und Rahmenbedingung
    1.2. Motivation
2. Grundlagen
    2.1. Stand der Technik
    2.2. Raspberry Pi
        2.2.1. Hardware
        2.2.2. Software
    2.3. Android Smartphone
        2.3.1. Android Betriebssystem
        2.3.2. Konnektivität
    2.4. Android Applikation
        2.4.1. Android Manifest
        2.4.2. Android Activity
        2.4.3. Android Ressource
        2.4.4. Android Layout
        2.4.5. Android View
    2.5. Lösungsansätze
        2.5.1. Android Applikation
        2.5.2. Übertragungsart
        2.5.3. Signalempfang und Auswertung
        2.5.4. Ausführung der Übertragene Signal
3. Vorbereitung
    3.1. Raspberry Pi
        3.1.1. Inbetriebnahme und Vorbereitung
        3.1.2. Werkzeugen
        3.1.3. Installation des Webservers
        3.1.4. Einrichtung des Webservers
    3.2. App Entwicklungsumgebung
        3.2.1. Anlegen des Projekts
        3.2.2. Einbindung Externe Library
        3.2.3. Einstellung der Version Kontrollsystem
4. Implementierung
    4.1. Raspberry Pi Software
        4.1.1. Kommunikation Komponente
        4.1.2. Interpreter Komponente
    4.2. Android App (Pi$Control)

Gliederung der schriftlichen Arbeit - NEU

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
XML	Extensible Markup Language
GUI	Graphical User Interface
GPIO	General Purpose Input Output
RasPi	Raspberry Pi
PHP	Hypertext Preprocessor
NFC	Near Fields Communication
ISR	Interrupt Service Routine
WLAN	Wireless Local Area Network
AP	Access Point
CGI	Common Gateway Interface
PWM	Pulse Width Modulation
FQN	Fully qualified Name
SDK	Software development kit

1. Einleitung

Eingebettet Systeme sind in den letzten Jahrzehnten sehr preiswert geworden. Dies beschreibt einen Computer, welcher in einem umgebenden technischen System eingebettet ist und mit diesem in Wechselwirkung steht. Der Computer übernimmt komplexe Steuerungs-, Regelungs-, Überwachungs- und Datenverarbeitungsaufgaben und verleiht damit dem umgebenden System oft einen entscheidenden Wettbewerbsvorsprung. Diese eingebetteten Systeme haben sich heutzutage in allen Bereichen der Technik etabliert wie zum Beispiel in der in der Industrie, Automobilindustrie, Medizin und in der Telekommunikation. In Moderne Personenkraftwagen der Oberklasse sind beispielsweise zwischen 70 und 80 integrierte und miteinander vernetzte Steuergeräte enthalten (1). Eingebetteten System erfordern in der meistens fällen ein Bedienungs- oder Steuerungssystem.

Heutzutage besteht ein Trend eingebettete Systeme mit Smartphones zu steuern. Dies kann besonders in der Hausautomatisierung und Robotik beobachtet werden. In der Robotik-Branche wird die Steuerung von Drohnen, Saugrobotern und Quadrocoptern mit Smartphones immer beliebter. Des Weiteren können, solange eine Internetverbindung besteht, Haushaltsgeräte mit eingebetteten Systemen unterwegs oder zuhause mithilfe eines Smartphones gesteuert werden. Bei einem Smartphone handelt es sich um ein Mobiltelefon, dass neben dem Telefonieren noch zahlreiche weitere Computertechnische Funktionen aufweist.

Die Steuerung von eingebetteten Systemen mit Hilfe des Smartphones erfolgt über vordefinierte Steuerelemente. Dies hat den Nachteil, dass bei der Entwicklung eines eingebetteten Systems, die Steuerung neu zu Implementierung ist. Um die Entwicklung des Steuerungssystems zu erleichtern, ist es sinnvoll, Ein allgemein einsetzbares Steuerungssystem bereitzustellen. Diese Bachelor Arbeit umfasst den Entwurf und die Implementierung eines Solchen Systems für den Raspberry Pi. Sie dient auch als ein Leitfaden, damit Nachbau und Weiterentwicklung ermöglicht wird.

1.1. Problemstellung/Zielsetzung

Ziel dieser Bachelorarbeit, ist die Entwicklung einer universellen Android Applikation für die Bedienung eines Raspberry Pi basierten eingebettet Systems. Die Applikation überträgt Signale zum Raspberry Pi über eine drahtlose Schnittstelle. Der Raspberry muss vorher mit einer kommunikationsschnittstelle und einer API ausgestattet werden.

Aufgabe der Kommunikationsschnittstelle ist das Empfangen und Weiterleiten von Signalen zur Benutzer Anwendung. Letzteres ist die Nutzer Baustelle, die Ihm ermöglicht durch einbinden der API auf übertragene Signale zuzugreifen und die auszuführende Aktion selber zu bestimmen. Die Schnittstelle, die API und die Benutzer Anwendung sollen möglich getrennt sein. Die gleiche Philosophie soll auch bei der Android Applikation verwendet werden.

Die Android App besteht aus einer konfigurierbaren graphische Oberfläche und einer Kommunikationsschnittstelle. Die Oberfläche ist das Benutzerfenster; sie biete unter anderen

Steuerelemente zur Verfügung. Der Nutzer soll je nach Wunsch Steuerelemente einfügen oder löschen können. Beim Einfügen bestimmt er die Eigenschaften des Elements. Aufgabe der Kommunikationsschnittstelle ist die Übertragung von Steuerelemente Signale an dem Raspberry Pi. Eine weitere Aufgabe ist das Mitteilen von Raspberry Pi Ausgaben an der Oberfläche.

Die Module, die oben erläutern wurden sollen sowohl auf dem Android Smartphone und auf dem Raspberry Pi nicht vermischt werden. Zweck ist, dem Nutzer eine Flexibilität zu bieten und Ihm den Austausch zu erleichtern.

Das Bedienungssystem (Die Applikation, die Raspberry Pi Schnittstelle und die API) soll einen idealen Einstiegspunkt für die Entwicklung eines Steuerungssystems für eingebettet System mit Raspberry Pi über Android Smartphone bieten.

Weiterhin werden alle Schritte beim Entwurf, sowie der Implementierung dokumentiert, damit zukünftige Nachbauten und Erweiterungen unkompliziert gemacht werden können.

1.2. Motivation

Die Motivation sich mit dem Thema dieser Bachelorarbeit auseinanderzusetzen, ist die Anwendung und die Erweiterung der während der Studienzeiten erworbenen Wissen in der Soft- und Hardware. Es wird Praktisch umgesetzt, wie man ein Projekt von der Grundidee bis hin zur Implementierung einschließlich Test gestaltet. Die Hauptvoraussetzung für die Implementierung der Android Applikation, sind die Kenntnisse der im Fach „Projekt Software Entwicklung“ kennengelernte Android Framework. Die Entwicklung der Raspberry Pi Module benötigt Kenntnissen vom Raspberry Pi Umgebung, sprich Linux Betriebssysteme und seine bekannten GPIO-Pins und deren Programmierung. Grundlagen aus der Elektrontechnik werden beim Test benötigt. LEDs werden an den GPIO-Pins angeschlossen, um die Funktionalität der Applikation zu testen.

Von der Grundidee bis zur Implementierung und Test wird nicht nur das im Studium erworbene Wissen benötigt, sondern auch die Ergebnisse aus eigener Recherche eingesetzt um diese Bachelorarbeit nach besten Wissen und Gewissen abzuschließen.

2. Grundlagen

Dieses Kapitel befasst sich mit den verwendeten Komponenten für das Thema dieser Bachelorarbeit. Um das Ziel zu erreichen ist es wichtig, dass das Umfeld beschrieben und die Komponenten kennengelernt werden. Die Eigenschaften des Raspberry Pis und seine Schnittstellen werden näher betrachtet. Besonderes Augenmerk wird auf den „General Purpose Input Output“ (GPIO) gelegt. Schließlich werden Funkschnittstelle betrachtet. Nach dem Raspberry Pi wird Android Smartphone auch näher betrachten. Besonders werden hier die seine Funkschnittstelle, sein Betriebssystem, der Aufbau seiner Anwendungen beschrieben.

2.1. Stand der Technik

Vor Anfang dieser Bachelorarbeit, wurde nach Projekten gesucht die das das Thema schon behandeln haben. In diesem Kapitel werden einige davon vorgestellt.

PiRelay[1]

PiReplay ist eine Android App zur Steuerung von Raspberry Pi GPIO-Ports. Die Applikation ist für das Ein- und Ausschalten von GPIO-Pins und kann bis zu fünf Raspberry Pi steuern. Zweck der Entwicklung dieser App, ist die Steuerung Haushaltsgeräten wie Lichter, Ventilatoren, Motoren, Türen und Heizung. Davor muss aber der Raspberry Pi mit einem Webserver und ein PHP Skript ausgestattet sein. Die Oberfläche ist konfigurierbar und kann bis zu 100 Steuerelemente „Relay“ (vgl. Abbildung 1).

Wenn ein „Relay“ auslöst sendet es dem Webserver eine HTTP Anfrage über WLAN. Der Server bearbeitet die Anfrage und leitet die Informationen an das PHP Skript weiter. Die Informationen enthalten die Pin-Nummer und das Signal. Mittels dieser Informationen, schaltet das Skript der Pin Ein oder Aus (2).

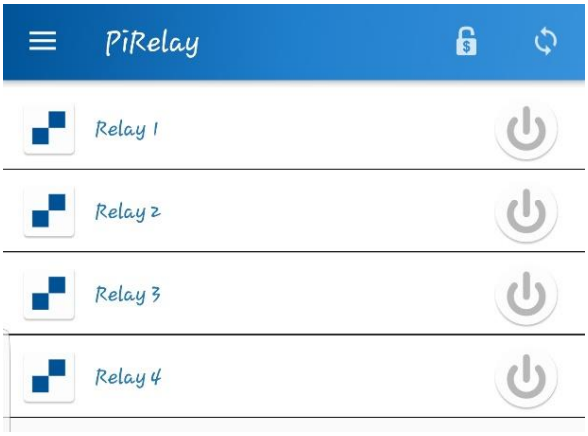

Abbildung 1: Graphische Oberfläche PiReplay (2)

[1] https://play.google.com/store/apps/details?id=com.jasonfindlay.pirelaypro [04.02.2018]

BlueTerm[2]

BlueTerm ist eine Android App zur Kommunikation mit allen Bluetooths serieller Adapter (4). Das Projekt besteht aus BlueTerm und zwei Python Skripten. Das erste Skript ist zuständig für die Bluetooth Kommunikation und das zweite für die GPIO-Pins. Im Code ist schon festgelegt welche Pins angesprochen werden sollen.

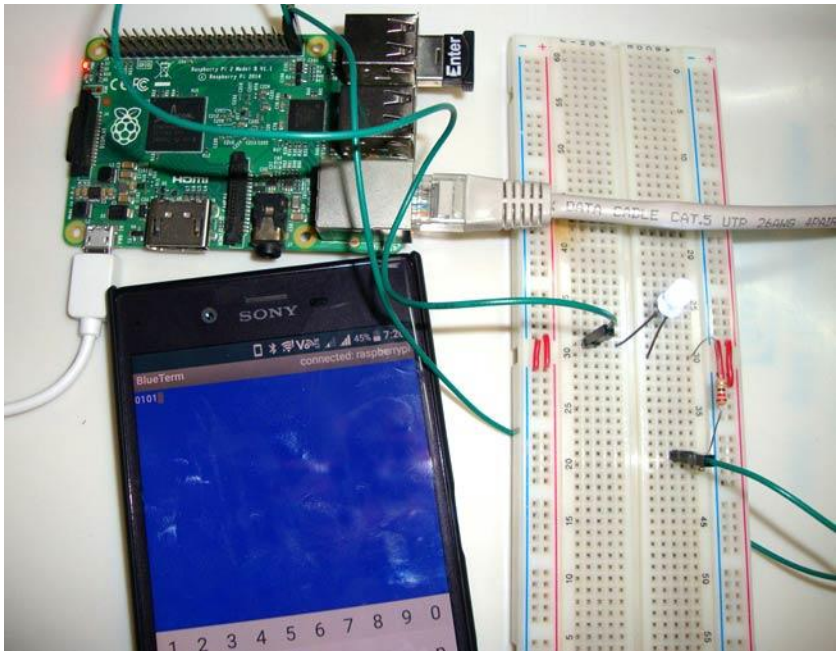

Abbildung 2: Konsolenanwendung BlueTerm

Im Gegensatz zu PiReplay besitzt BlueTerm keine Graphische Oberfläche, sondern eine Konsole für die Eingabe. Durch eingeben von Ein (1) oder Null (0) in der Konsole kann der Pin Ein- oder Ausgeschaltet werden.

Zusammenfassung

Aus der Suche ergab sich, dass unsere Thema nicht neue ist. Es existieren bereits viele Projekte die dies behandelt haben; sie erfüllen jedoch nicht komplett unsere am Anfang festgelegte Voraussetzungen. Zum einen hat der Nutzer keinen direkten Zugriff auf die übertragenen Signale und kann sie also nicht nach seinem Wunsch implementieren und zum anderen sind Projektmodule nicht austauschbar, entweder weil der Quellcode nicht Open Source ist oder keine sichtbaren Grenzen zwischen den Komponenten existiert.

2.2. Raspberry Pi

Raspberry Pi ist ein Einplatinencomputer in Kreditkarten Format. Er wurde von Videospiel-schöpfer David Braden für die Raspberry Pi Stiftung entwickelt.

[2] https://play.google.com/store/apps/details?id=es.pymasde.blueterm&hl=fr [04.02.2018]

Die Intention war es, ein günstiges und einfach zu programmierendes Produkt zu entwickeln. Der Raspberry Pi kostet ungefähr 35 Euro und bietet mit seinem günstigen Preis Leuten, die nicht über die finanziellen Mittel verfügen und sich fürs Programmieren interessieren und begeistern, die Möglichkeit einen Computer zu kaufen und sich darin einzuarbeiten (5).

Das Thema dieser Bachelorarbeit wurde mit dem Raspberry Pi 3 Modell B bearbeitet. Es ist seit Februar 2016 verfügbar. Im Lieferumfang befinden sich der Computer und der Stromadapter. Weitere Zubehör muss man sich extra kaufen.

2.2.1. Hardware

In diesem Kapitel werden physische Komponente des verwendet en Modells näher kennengelernt. Besonderes Augenmerk wird auf seine GPIO-Ports und seine Funkschnittstelle gelegt.

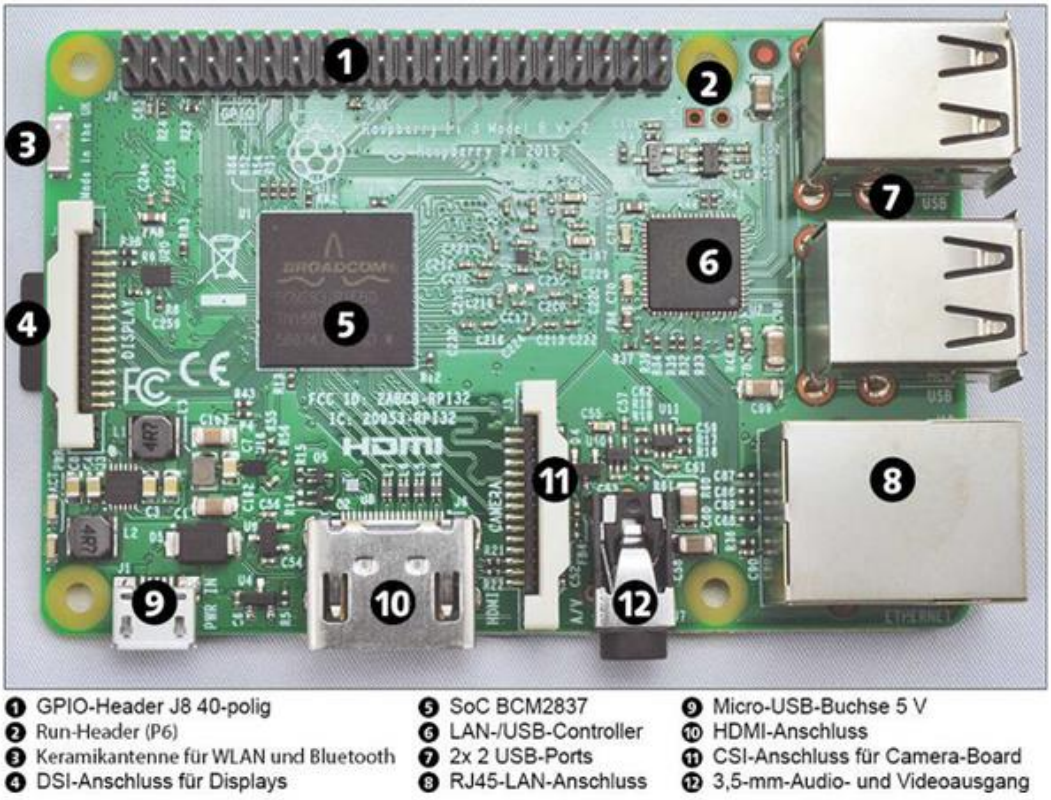

Abbildung 3: Komponentenübersicht des Raspberry-Pi 3 Modell B (5)

Maße (Länge x Breite x Höhe): 85,6 mm x 56,0 mm x 20mm

Gewicht: 40g

SoC: Broadcom-BCM2837

CPU

- Typ: ARM Cortex-A53
- Kerne: 4
- Takt: 1200 MHz
- Architektur: ARMv8-A (64 Bit)

GPU: Broadcom Dual Core Videocore IV

Arbeitsspeicher: 1024 MB

Netzwerk

- Ethernet: 10/100 Mbit/s
- WLAN: Broadcom BCM43143 2,4 GHz WLAN b/g/n
- Bluetooth: 4.1 Low Energy

Schnittstellen

- GPIO-Pins: 40
- CSI, DSI, I^2C, SPI, UART, microSD-Slot
- 4 x USB 2 Port
- DSI Display Port
- CSI Kamera Port

Videoausgabe: HDMI (Typ A), Composite Video

Audioausgabe: HDMI (digital), 3,5-mm-Klinkenstecker (analog)

Betriebsspannung: 5 Volt Micro-USB-Anschluss (Micro-USB-B)

GPIO-Pins

Sie sind auf Abbildung 3 mit der Nummer 1 repräsentiert. Sie werden verwendet, um Peripherie Geräte oder Messinstrumente anzuschließen, um Daten zu verarbeiten oder Funktionen zu steuern. Die Funktionen dahinter ermöglichen es Signale mit High- und Low-Pegel zu generieren, um einfache Bauelemente, Wie LEDs anzusteuern. Sie bieten auch Pulsweitenmodulation (PWM) an, um eine LED zu dimmen oder einen Motor mit unterschiedlich schnellen Radumdrehungen anzusteuern.

Das Model 3 B besitzt 40 Pins. Sie sind in zwei Reihen mit jeweils 20 Pins mit einem Abstand von 2,54 mm angeordnet; 28 davon können als GPIO verwendet werden. Die restlichen Pins sind entweder für einen Erdanschluss oder für die Stromversorgung. Zu den Stromversorgunganschlüssen zählen die 5 V und die 3.3V (7).

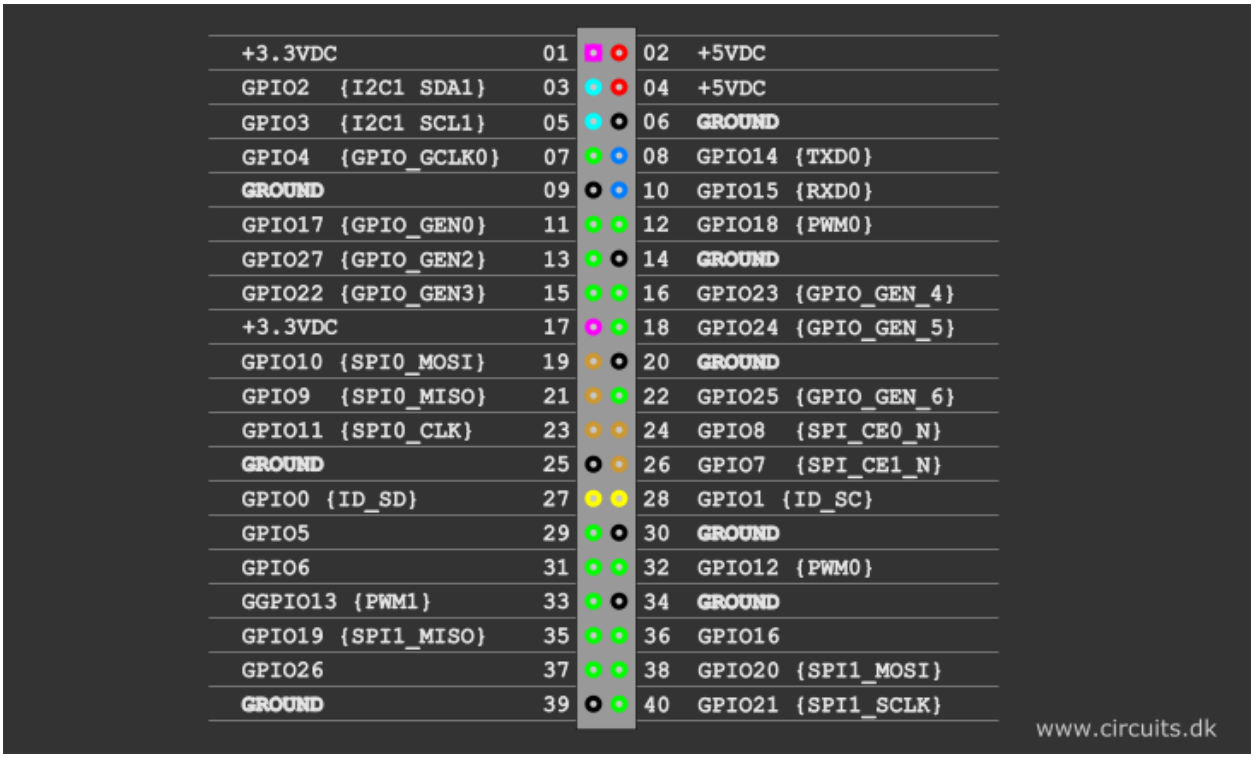

Abbildung 4: Raspberry Pi GPIO-Pins (7)

Es gibt zwei Nummerierungssysteme zur Bezeichnung der Pins. Das BOARD System bezieht sich auf die Physischen Position Pins auf dem Board. Die Nummern gehen von 1 bis 40 und der Pin mit Nummer 1 steht direkt neben die Bezeichnung J8 auf der Platine. Das BCM System bezieht sich auf die offizielle Dokumentation des auf dem Raspberry Pi verbauten BCM837-Chips. Es ist auf Abbildung 4 mit der Bezeichnung „GPIOxx“ repräsentiert. wobei „xx“ die Nummer bezeichnet.

Ferner haben die Entwickler eine weitere Bezeichnung eingeführt, welche in der Abbildung 4 nicht weiter deklariert ist, aber zum Teil den Pin in seiner Funktion ausweist. Zur Illustrierung sind die Pins farblich in Gruppen markiert, damit Zusammenhängende Funktionen der Pins deutlicher ausgemacht werden können.

Der maximale Strom ist 50 mA bei den 3.3V Pins und 1 A bei den 5.5V Pins. Die 5V Pins sollten sorgfältig behandelt werden, da sie das Board beschädigen können, wenn Sie mit andere Pins direkt verbinden sind. Sie sollten am besten vor jeder Manipulation isoliert werden.

Die Software Umsetzung der GPIO Pins ist in verschiedenen Programmiersprachen möglich. Python und C zählen zu den bekanntesten. Es existiert bereit Bibliotheken in diesen Sprachen, die die Implementierung erleichtern; beispielweise RPi.GPIO, Piggpio und WiringPi.

Es ist wichtig zu wissen, dass die Raspberry Pi GPIO für Echtzeit- oder Zeitnahreagierende Systeme nicht geeignet sind, da das Betriebssystem jederzeit einem anderen Prozess priorisieren kann.

Funkschnittstellen

Das Model 3 B beinhaltet eingebautes Bluetooth und WLAN.

Der Bluetooth Chip arbeitet mit BTLE, auch bezeichnet als Bluetooth 4.1 Low Energy. Damit bringt der Raspberry Pi 3 Model B.

Das WLAN Modul unterstützt die Standards 802.11b, g und n und arbeitet im 2.4 GHz Band (BCM43143). Das Modul hat eine maximale übertragungsrate von 150 Mbit/s (8).

2.2.2. Software

Betriebssystem

Das Betriebssystem ist eine Zusammenstellung von Computerprogrammen, die Systemressourcen eines Computers wie Arbeitsspeicher, Festplatten, Ein- und Ausgabegeräte verwaltet (8). Es gibt zwei Hauptkategorien vom Betriebssystem für den Raspberry Pi: Die eingebetten und die Mehrzwecksysteme.

Die Eingebetten sind für bestimmte Zwecke entwickelt. Sie besitzen meistens keine graphische Oberfläche. Durch Ihre geringe Anforderungen in Bezug auf Energiebedarf sowie Arbeits- und Massenspeicher eignen sich eingebettete Betriebssysteme für Zeitnah reagierende System wie Industrieanlagen, Drohnen, Antiblockiersystem und Roboter. Ein Beispiel ist Minoca OS (10).

Die Mehrzwecksysteme dagegen brauchen viel Speicher und Energie und haben längere Reaktionszeit. Sie haben aber den Vorteil, dass sie einfacher zu bedienen sind und meisten eine graphische Oberfläche besitzen. Im Internet sind mehrere Mehrzwecksysteme zu finden. Die Üblichen sind Ubuntu, Noobs und Raspbian.

Das beliebteste Betriebssystem ist der kostenlose, auf Debian Linux basierte und für Raspberry Pi Hardware optimierte Raspbian. Die Beliebtheit entsteht durch die kostenlose Lizenz, die schnelle Leistung, und die vorinstallierte Software und Tools.

Raspbian wird offiziell unterstützt von der Raspberry Stiftung und beinhaltet mehr als 35.000 Software Pakete. Dazu zählen unter anderen Textverarbeitungsprogramme wie LibreOffice, Browsers wie Chromium und Firefox, Entwicklungsprogramme wie Geany und BlueJ.

Anwendung

Hier werden die Programmiersprachen der Raspberry Pi Anwendungen erläutert.

Der Name des Raspberry Pi knüpft an der Tradition an, Computer nach Früchten zu benennen. Bekannte Vertreter sind Apple und Blackberry. Der Zusatz „Pi“, ausgesprochen wie das englische Wort „Pie“, übersetzt für Tortenstück, wird oft fälschlicherweise als Solches interpretiert, steht jedoch für Python Interpreter, eine Andeutung für die Hauptprogrammiersprache des Raspberry Pis.

Bei Python handelt es sich um eine Höhere Programmiersprache, die sich durch ihre Komplexität von der Maschinensprache entfernt. Erst der Einsatz von Interpreter oder Compiler übersetzt Befehle des Programmiercodes in Maschinensprache, die der Mikroprozessor bzw. Mikrocontroller versteht.

Der Raspberry Pi kann aber nicht nur mit Python programmiert werden. Er unterstützt auch Scratch, C, C++, Java, Perl, HTML5 und Skriptsprachen wie, PHP, JavaScript. Da der Raspberry Pi auf einem Linux-Kernel operiert, können über die angebotenen Paketquellen nach Belieben weitere Programmiersprachen hinzugefügt werden.

2.3. Android Smartphone

Wie eingangs erwähnt, ist ein Smartphone ein Mobiltelefon, das umfangreiche Computer-Funktionalitäten und Konnektivitäten zur Verfügung stellt. Bei Android Smartphones handelt es sich um Smartphones, die mit dem Android Betriebssystem ausgestattet sind.

2.3.1. Hardware

Wegen der Vielfalt von Android Smartphones, können ihre physische Komponenten nicht genau beschrieben. Laut Statista[3] befindet sich weltweit bereit 2,58 Mrd. Android Smartphone (Stand 2017) im Gebrauch. Sie sind je nach Hersteller und Modell unterschiedlich gebaut. Meistens sind sie unter anderen mit Prozessor, Speicher, Touchscreen, Batterie, Kamera, WLAN und Bluetooth ausgestattet.

Moderne Smartphone besitzen neue Technologie wie NFC, A-GPS, Glonass, und Galileo. Sie sind auch mit vielen Sensoren ausgerüstet wie Gyroskope, Barometer und Magnetometer (8).

2.3.2. Software

Betriebssystem

Android ist das Open-Source-Betriebssystem von Google, das die meistens Smartphones und Tablets auf dem Markt antreibt. Die erste Android-Version kam im September 2008 auf dem Markt (10). Es besitzt heutzutage weltweit den Größten Marktanteil von Smartphone Betriebssystemen.

[3] https://de.statista.com/themen/581/smartphones/ [06.02.2018]

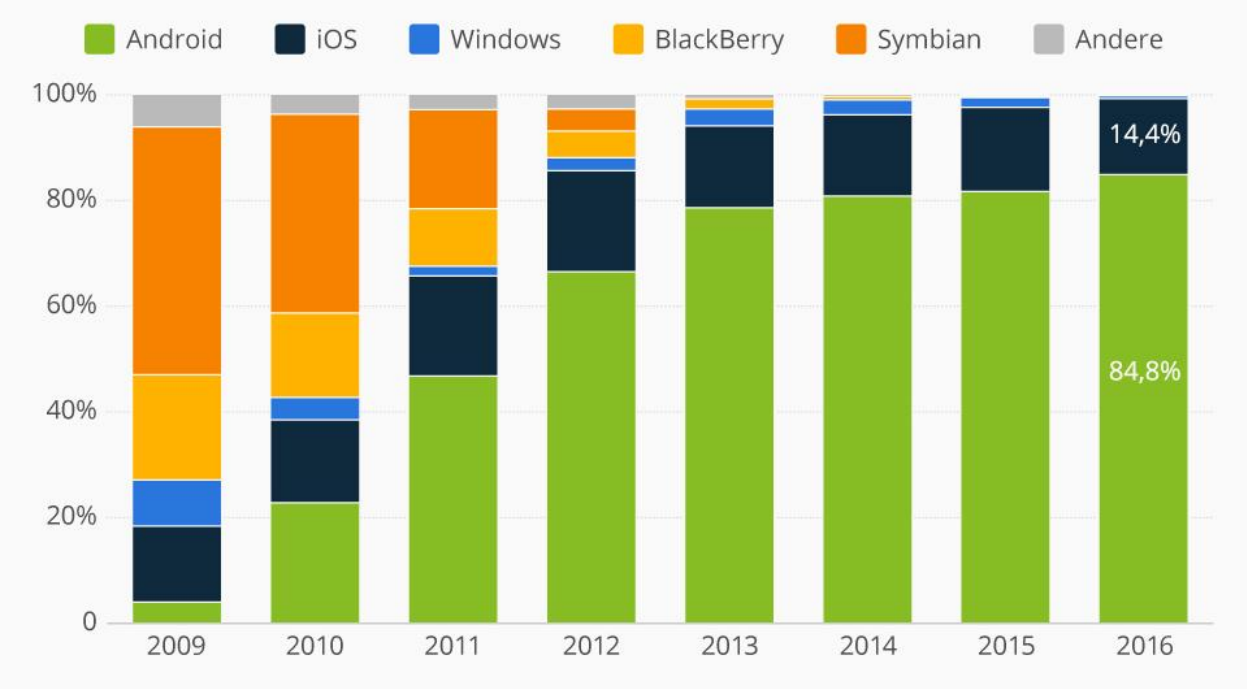

Abbildung 5: Weltweiter Marktanteil der Smartphone-Betriebssysteme (13)

Das Betriebssystem beinhaltet vier Schichten und Jede abstrahiert die darunterliegende. Es besteht aus einem angepassten Linux-Kernel. Das Kernel bildet die Schicht zwischen der Hardware und der restlichen Architektur. Er enthält unter anderem Hardwaretreiber, Prozess- Speicherverwaltung und Sicherheitsverwaltung (siehe Abbildung 6).

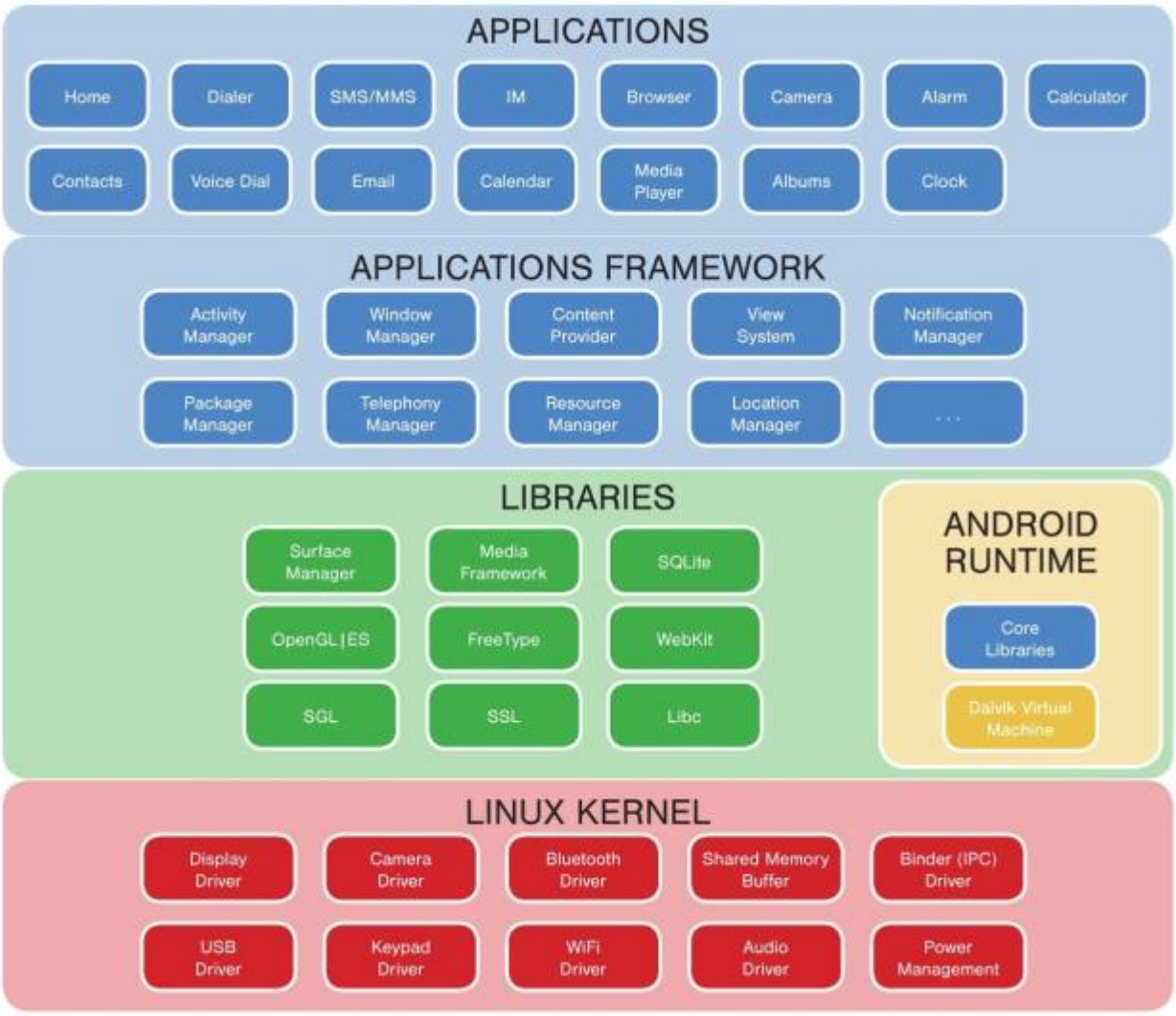

Abbildung 6: Architektur Android (11)

Da Android ein Multiprozess Betriebssystem ist, läuft jede Anwendung in ihrer eigenen Instanz der virtuellen Maschine. Dieser Umstand ist vor allem für die Sicherheit und die Anwendungsentwicklung von Bedeutung. Die Virtuelle Maschine wurde speziell dafür entwickelt mehrere virtuelle Maschinen effizient parallel ausführen zu können. Die ausgeführten Dateien sind in einem für minimalen Speicherverbrauch optimierten dex-Format. Dieses Format wird, nachdem es von Java kompiliert wurde, mit dem dx-Werkzeug in das nötige Format umgewandelt. Die Virtuelle Maschine benötigt weniger Zwischenschritte um den Bytecode auf dem Prozessor auszuführen, weil das Register nicht stapelbasiert ist. Die Maschine nutzt die Low-Level-Speicherverwaltung und das Threading des optimierten Kernels aus (12).

Das Anwendungsframework ist die Schicht, die für die Entwicklung von Android Applikationen von Bedeutung ist. Sie stellt den Rahmen für eine einheitliche Anwendungsarchitektur bereit. Ziel ist es, Anwendungen nach einheitlichen Richtlinien zu entwickeln und somit die Integration und Wiederverwendung von Anwendungen unter Android zu vereinfachen (12). Es bietet Entwicklern

Elementen und Methoden zur Erstellung von GUIs oder Nutzung von Ressourcen an. Hintergrunddienste, Nachrichtenaustausch mit anderen Applikationen und das Nutzen von Hardware Ressourcen wird ermöglich. Der Zugrifferlaubnis auf Ressourcen muss ebenfalls bei Installation der Applikation vom Nutzer akzeptiert werden. So werden unerlaubte Zugriffe vermieden.

Das Android-Betriebssystem beinhaltet grundlegende Apps wie E-Mail, SMS-, Kalender-, Karten-, Kontakte-, und Browseranwendungen. Diese Anwendungen wurden mit der Programmiersprache Java implementiert.

2.3.3. Android-Anwendung

In diesem Kapitel werden die wichtigsten Komponenten eine Android-Anwendung und deren Zusammenhang kennengelernt. Die App besteht grundsätzlich aus einem Manifest, den Java Code und Ressourcen. Der Java Code beinhaltet Activities. Ressourcen bestehen aus Layouts, und notwendige Bestandteile des Programms wie Zeichenfolgen, und Binärdateien. Der Java Code und die Ressourcen sind im Projekt unter den Ordner java beziehungsweise res zu finden. Der Java Code beinhaltet die Programmlogik, die Ressourcen dagegen die graphische Darstellung.

Abbildung 7: Aufbau Android-Anwendung

Android Manifest

Es beinhaltet grundlegende Informationen, die das System braucht, um die Applikation zu starten. Drinnen befinden sich Angaben über Komponenten und Zugriffsrechte der Applikation (siehe Abbildung 8). Zu den Komponenten zählen Activities und Hintergrund-Services. Die Manifest-Datei ist XML-formatiert.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="testapp.android.test">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.CAMERA" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Test"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Abbildung 8: Beispiel Android Manifest

Activity

Eine Android-App besteht wesentlich aus Activities (3). Sie ist ein Fenster mit dem der Nutzer mittels Tasten- oder Touchscreen Berührungen interagiert. Das Aussehen der Benutzerfenster wird mit Layouts definiert. Es kann immer nur eine Activity gleichzeitig aktiv sein, bei einem Wechsel wird die zuvor laufende pausiert. Das Aufrufen erfolgt über Intents[4]. Intents sind Objekte, die Informationen über eine Operation beinhalten und das Bindeglied zwischen mehreren Komponenten von Android bilden.

Android greift im Vergleich zu anderen Betriebssystemen stärker in den Lebenslauf einer Anwendung ein. Aus dem Benutzerverhalten und dem Activity-Management von Android resultieren für eine Aktivität verschiedene Zustände, wobei die Übergänge durch den Aufruf bestimmter Methoden gekennzeichnet sind (16).

[4] https://developer.android.com/reference/android/content/Intent.html [06.02.2018]

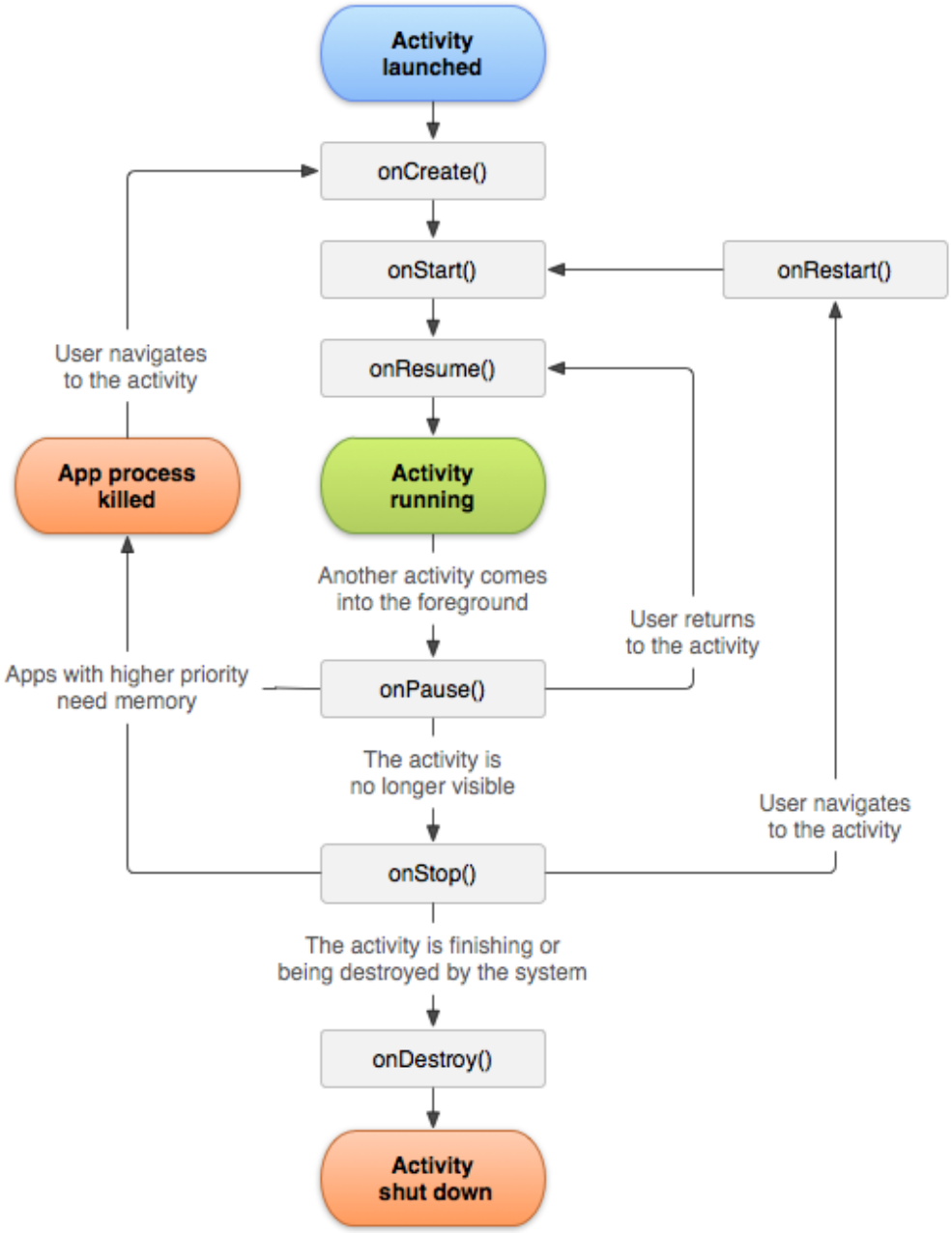

Abbildung 9: Zustandsdiagramm einer Android Activity (16)

Das aufrufen dieser Methoden wird bis auf die *onCreate*-Methode vom System übernommen. Beim Starten einer Anwendung und somit der ersten Activity, werden gleich drei Methoden Aufgerufen: *onCreate, onStart,* und *onResume* (siehe Abbildung 9). Die wesentliche Logik der Activity ist in der *onCreate*-Methode implementiert. Da wird auch mithilfe von der *setContentView*-Methode bekannt gegeben, welches Layout (GUI) zu der Activity gehört (siehe Abbildung 10). Dieser Methode muss ein Layout übergeben werden.

```java
package testapp.android.test;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Hier kann der Nutzer die Aktivitätslogik implementieren.
    }
}
```

Abbildung 10: Deklaration von Activity in Java

Layout

Ein Layout definiert die Struktur einer GUI. Es gibt verschiedene Typen von Layouts: LinearLayout, RelativeLayout, TableLayout, GridLayout. Der wesentliche Unterschied ist die Anordnung der GUI Elementen (Views). Ein LinearLayout positioniert beispielweise Elemente in einer einzigen vertikal oder horizontal Reihe. Zu jedem Layout-Typ gehört eine Java-Klasse.

Layouts können sowohl in XML als auch in Java deklariert werden. Android bietet ein einfaches XML Vokabular zur Erstellung von Layouts. Die Deklaration in Java erfordert die Vererbung der Klasse vom Layout-Typ. Die Positionierung des GUI Elements und die Große des Layouts werden mithilfe der LayoutParams bestimmt (17). Layouts können auch verschachtelt sein.

```java
import android.content.Context;
import android.util.AttributeSet;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.RelativeLayout;

public class MyRelativeLayout extends RelativeLayout {

    public MyRelativeLayout(Context context) {
        super(context);
        init();
    }

    public MyRelativeLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
        init();
    }

    private void init() {
        setLayoutParams(layoutParams());
    }

    private LayoutParams layoutParams () {
        LayoutParams params = new LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT,
                              ViewGroup.LayoutParams.WRAP_CONTENT);
        return params;
    }
}
```

Abbildung 11: Deklaration vom RelativeLayout in Java

View

Views sind Steuerelemente der Android GUI. Sie reagieren auf Touch- oder Tatstatur-Ereignissen und ermöglichen somit dem Nutzer mit der Anwendung zu kommunizieren. Android bietet verschiedene Typen von Views, die sich von ihrem Aussehen und ihrer Bedienungsform unterscheiden. Zu jedem View-Typ gehört eine Java-Klasse.

View-Klasse	Beschreibung
TextView	Einfache Textausgabe
Button, ImageButton	Schaltfläche
EditText	Texteingabe
CheckBox	Ankreuzfeld
RadioGroup, RadioButton	Auswahlschalter; einer aus RadioGroup
Spinner	Auswahlliste
DatePicker, TimePicker	Auswahl von Datum und Zeit
AnalogClock, DigitalClock	Anzeige Uhrzeit (analog, digital)

Abbildung 12: Views Klassen in Android (18)

Views können in XML oder in Java (siehe Abbildung 13) deklariert werden. Die Gestaltung von Ereignissen erfolgt über Interfaces und Methoden von View-Klassen.

```java
package testapp.android.test;

import android.content.Context;
import android.util.AttributeSet;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;

public class MyView extends EditText {

    public MyView(Context context) {
        super(context);
    }

    public MyView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }
}
```

Abbildung 13: Deklaration vom Android View in Java

3. Lösungsansatz

Dieses Kapitel umfasst die Entscheidungen über Aufbau und Entwurf der Android Applikation und der RasPi Anwendung. Die Entscheidungen beziehen sich auf die Grundlagen und das damit einhergehende, erlangte Wissen. Der Komplette Aufbau wird mithilfe von Bildern und Graphen skizziert.

3.1. Konzept

Die wesentliche Herausforderung dieser Bachelorarbeit ist die Entwicklung eines modular aufgebauten und universellen (wiederverwendbaren) Steuerungssystem. Beim Entwurf wurde auf diesen Kriterien besonders geachtet und ein Konzept entwickelt, das diese erfüllt.

Modularität

Die Modularität (auch Baustein- oder Baukastenprinzip) ist die Aufteilung eines Ganzen in Teile, die als Module, Komponenten, Bauelemente oder Bausteine bezeichnet werden und zusammen interagieren können (19). Um die Modularität des Systems zu gewährleisten, wurden Vollständig implementierten Module definiert. Bei Module handelt es sich, um Elemente, die als Bestandteil eines Systems einen bestimmten Zweck erfüllen.

Die Android Applikation besteht aus zwei Modulen: die Benutzeroberfläche (GUI) und die Kommunikationsschnittstelle. Die Benutzeroberfläche beinhaltet Steuerelemente. Ein Steuerelement beschreibt eine View, die durch Touch-Ereignisse Signale and den RasPi sendet. Die Kommunikationsschnittstelle sendet Signale über die Funkschnittstelle an den RasPi und leitet seine Nachtrichten an der Benutzeroberfläche weiter.

Die RasPi Anwendung besteht aus der Kommunikationsschnittstelle und der API. Die Kommunikationsschnittstelle steuert den Empfang und die Weiterleitung von Signalen an die Nutzer Anwendung. Nachrichten an die Android Applikation werden ebenfalls über die Kommunikationsschnittstelle übertragen. Die API ist die Schnittstelle zwischen den Signalen und der Nutzer Anwendung. Sie bietet dem Nutzer Methoden zur Verwaltung von Signalen und zur Erstellung von Nachrichten, die an die Android Applikation versendet werden. Die API ist in Nutzer Anwendung einzubinden.

Universalität

Im Kontext dieser Bachelorarbeit besteht die Universalität darin, die App für unterschiedliche Anwendungsszenarien einzusetzen. Die universelle Einsetzbarkeit des Bedienungssystems ist

durch die API gewährleistet, die dem Nutzer die Möglichkeit bietet die Signale individuell zu implementieren.

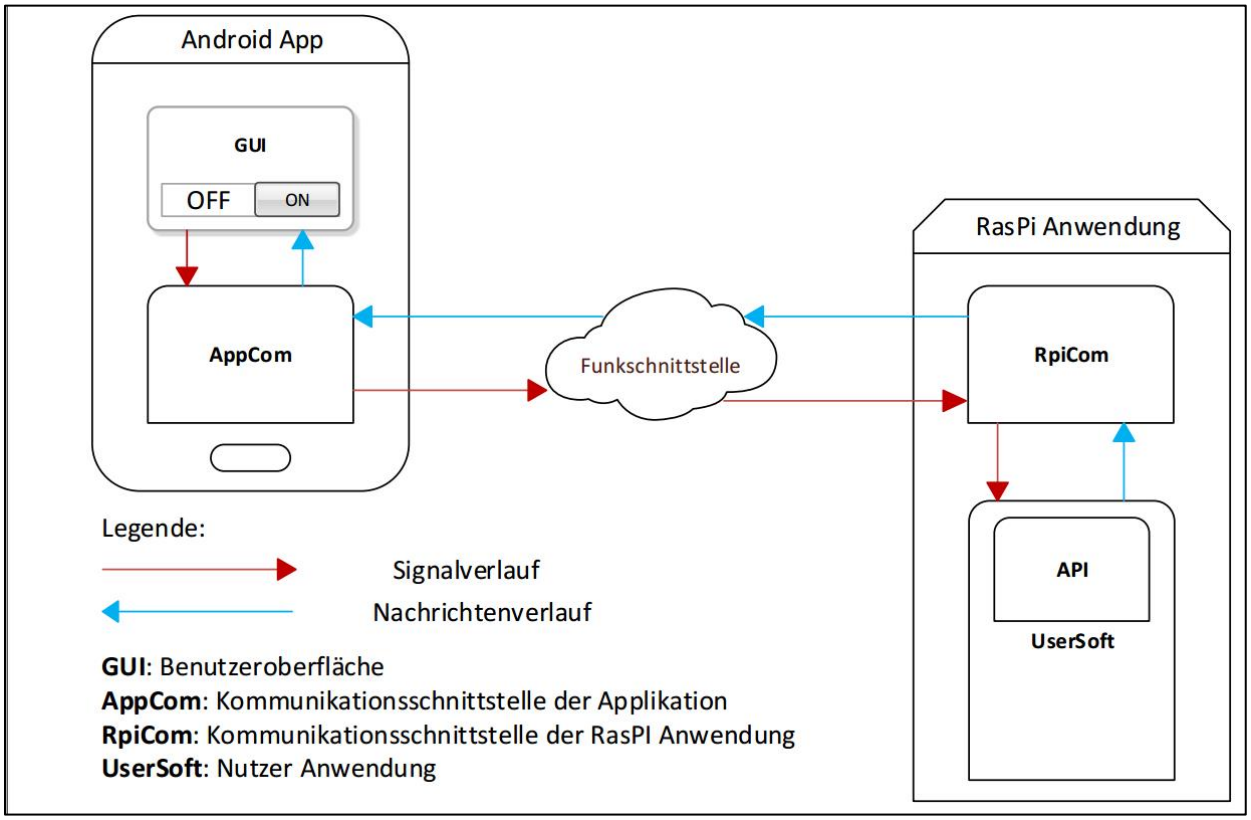

Abbildung 14: Aufbau des Steuerungssystems

3.2. Entscheidungen über Implementierung

Wir haben sowohl bei dem Android-Smartphone, als auch bei dem Raspberry Pi erfahren welche Funkschnittstelle sie anbieten, welche Programmiersprachen sie unterstützen. In diesem Kapitel werden wir begründete Entscheidungen für den Entwurf treffen.

3.2.1. Übertragung

Bei der Signalübertragung geht aus um zwei Aspekte: Die Schnittstelle und der Übertragungsmodus.

Schnittstelle

Das Android-Smartphone und der RasPi besitzen Bluetooth und WLAN. Die Übertragung kann über eine der beiden Schnittstellen erfolgen. Beide Schnittstellen besitzen Vor- und Nachteile.

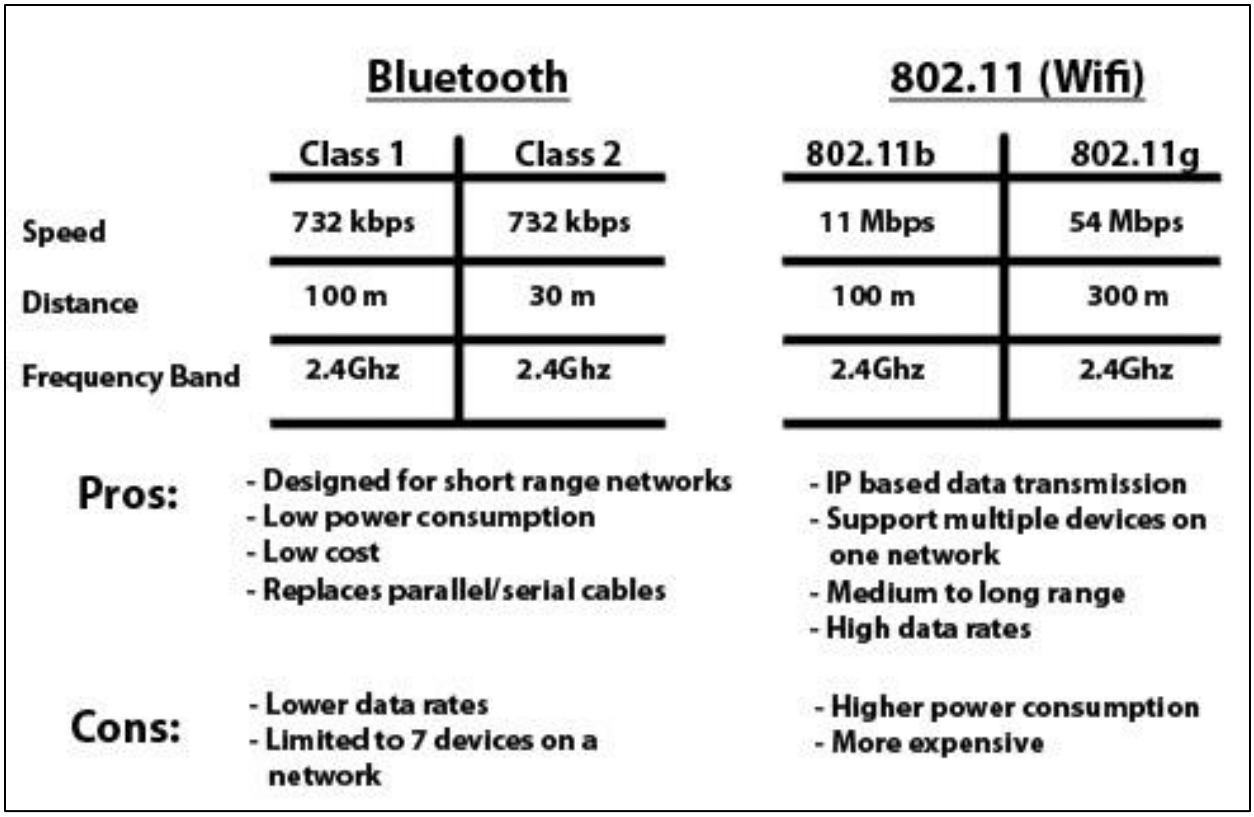

Abbildung 15: Vergleich Bluetooth und WLAN (20)

Die Entscheidung hängt von den Anforderungen an das System und den Zielen des Nutzers ab.

Für diese Arbeit wurde die WLAN-Schnittstelle verwendet. Die Gründe dafür sind die höhere Reichweite und die Unterstützung mehrerer Geräte. Die in Abbildung 15 erwähnte Nachteile der WLAN-Schnittstelle treffen in unserem Kontext nicht zu, da sowohl das Smartphone, wie auch der RasPi integriertes WLAN besitzen.

WLAN-Übertragungsmodus

Die WLAN-Schnittstelle bietet wiederum zwei übertragungsarten: Wi-Fi Direct WIFI Access Point.

Wi-Fi Direct ist ein Standard zur Datenübermittlung zwischen zwei WLAN-Endgeräten ohne zentralen Wireless Access Point (21). Jedoch unterstützen nicht alle Android-Smartphones diese Technologie und laut (22) unterstützt der im Raspberry Pi Modell 3 B integrierte WLAN-Modul diese Technologie nicht. Ein separates WLAN-Modul lässt sich allerding nachträglich hinzufügen.

Ein Access Point ist ein Gerät, das ein drahtloses lokales Netzwerk bereitstellt, mit dem andere Geräte (Clients) verbinden können. Das Hauptziel besteht häufig darin dem Client der Internetzugang zu ermöglichen. Clients können über das Netzwerk aber auch Nachrichten austauschen. In unseren Fall fungiert der RasPi als der Access Point und das Smartphone als Client. Das Integriertes WLAN-Modul des RasPi kann als Access Point konfiguriert werden und jedes Smartphone unterstützt den Client-Modus. Nachteil ist, dass der Client keinen Internetzugang erhält, wenn der Access Point keiner Internetverbindung aufbauen kann.

Bei dieser Bachelorarbeit wurde die Access Point-Variante verwendet. Diese Variante hat den Vorteil, dass sie mit vielen Geräten kompatibel ist. Ein weiterer Vorteil ist, dass mehrere Geräte sich gleichzeitig verbinden können. Dies ist beispielsweise für Anwendungen des Steuerungssystems im Bereich Hausautomatisierung von Vorteil.

3.2.2. Raspberry Pi-Module

Dieser Abschnitt behandelt die Entscheidung über Komponenten, die den Raspberry Pi umfassen. Es wird erläutert wie Signale empfangen werden sollen. Des Weiteren werden Entscheidungen über die Programmiersprachen der Module getroffen.

Webserver

Wie oben erläutert, besteht das Hauptziel eines Access Point darin, ein lokales Netzwerke zu erstellen und den Internetzugang bereitzustellen. Andere Anwendungsszenarien sind ebenfalls möglich und haben entsprechende Softwareanforderungen. Damit der Access Point Anfragen von Client bearbeiten kann, muss er mit einem Webserver ausgestattet sein. Für diese Arbeit wurde Apache Web Server[5] verwendet. Die Signale werden dann von der Android Applikation als HTTP-Anfrage gesendet. HTTP-Anfragen ermöglichen generell dem Client Informationen zum Webserver zu senden, um Formulare abzusenden, Dateien hochzuladen oder Dateien abzurufen (23). In unseren Fall rufen wir die Kommunikationsschnittstelle des RasPi mit Parametern auf.

```
http://www.webserver-adress.com/file_to_call.cgi?param1=val1+param2=val2+param3=val3
```

Abbildung 16: Beispiel HTTP Anfrage mit Parametern

Module

„File_to_call.cgi“ in Abbildung 16 ist die Schnittstelle auf dem Server, die die Anfrage empfängt. Es handelt sich um ein CGI-Skript. Das Common Gateway Interface (CGI) ist ein Standard zum Schreiben von Programmen, die über einen Webserver mit einem Client interagieren können, auf dem ein Webbrowser ausgeführt wird. Grundsätzlich interpretiert, verarbeitet er die an ihn gesendeten Informationen und generiert die Antwort, die an den Client zurückgesendet wird. Meistens werden diese Informationen über Umgebungsvariablen[6] in das CGI-Skript eingegeben (25).

Wie in den Grundlagen erläutert, lassen sich als Programmiersprachen Python, C/C++ und Java im RasPi verwenden. Außer C sind die anderen Sprachen sogenannte interpretierte Sprachen. Sie haben den Nachteil, dass sie langsamer sind als Maschinensprachen.

[5] https://www.raspberrypi.org/documentation/remote-access/web-server/apache.md [10.02.2018]
[6] https://de.wikipedia.org/wiki/Umgebungsvariable [13.02.2018]

Auf (26) wurde dieser Unterschied bewiesen. Dafür wurde ein Programm in Python und C geschrieben, dass dauerhaft die LED ein- beziehungsweise ausschaltet und auf dem Raspberry Pi ausgeführt. Die Ergebnisse befinden sich in der unteren Abbildung.

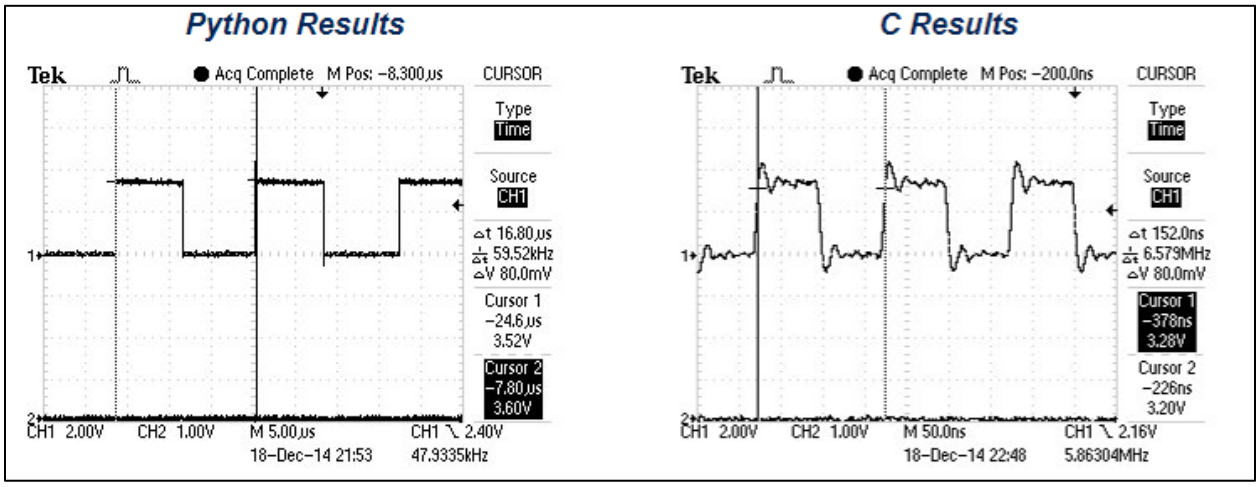

Abbildung 17: Leistung Python und C (26)

Die Zeitachse ist in Mikrosekunde und Nanosekunde bei Python beziehungsweise C. Letzteres ist ca. 100-mal schneller als Python. Jedoch besteht der Nachteil in der Komplexität.

Die Geschwindigkeit einer höheren Programmiersprache reicht für unseren Anwendungsfall allerdings vollkommen aus. Deswegen wurden die Kommunikationsschnittstelle und die API in Java beziehungsweise Python implementiert. Ein Grund dafür beide Module in verschiedenen Sprachen zu implementieren, besteht im modularen Aufbau des Systems.

3.2.3. Android Applikation

Bei Android-Anwendungen ist die Auswahl an Programmiersprachen begrenzt. Diese können nur in Java programmiert werden. Die Kommunikationsschnittstelle wird dann in einem separaten Paket abgelegt, um einen späteren Austausch zu erleichtern.

4. Entwurf

Dieses Kapitel umfasst den praktischen Teil der Bachelorarbeit. Das erste Unterkapitel „Pflichtenheft“ gliedert sich in drei Kategorien „Muss“, „Wunsch“ und „Abgrenzung“ auf. In Stichpunkten wird aufgezählt, welche notwendigen Schritte in den Entwurf notwendig sind, welche Wunschkriterien gegebenenfalls realisiert werden, aber nicht zwingend erforderlich sind und explizite Abgrenzungen, die nicht umgesetzt werden.

Im Anschluss daran folgt der Entwurf der in Abbildung 14 definierten Module. Schrittweise wird der Entwurf der RasPi Anwendung mithilfe von Bilder und Graphen beschrieben. Danach folgt der Entwurf der Android Anwendung. Für ein besseres Verständnis wird der Entwurf mithilfe von Klassendiagrammen erklärt. Die wichtigsten Klassen, Interfaces und Komponenten werden dargestellt und deren Zusammenhang verdeutlicht.

4.1. Pflichtenheft

- Musskriterien
  - Implementierung der Android App mit sinnvollen Steuerelemente
  - Implementierung der RasPi Schnittstelle und der API
  - Entwurf eines Modular und Universelle Bedienungssystem.
- Wunschkriterien
  - Implementierung einer 8Bit-Wert PWM Steuerelement
  - Einfügen von Steuerelement auf Android App implementieren
  - Löschen von Steuerelement auf Android App implementieren
  - Anzeige der RasPi Ausgabe auf Android App
- Abgrenzungskriterien
  - Test auf einem Praktischen Projekt

4.2. Raspberry Pi Anwendung

4.2.1. API

Wie eingangs erwähnt ist die API in der Programmiersprache Python geschrieben und ist in dem Nutzer Anwendung einzubinden. Ihre wichtigsten Methoden werden in der folgenden Tabelle aufgeführt:

Methode Name	Beschreibung
get	Gibt das Argument Wert zurück
log	Formatiert eine Meldung
getName	Gibt den Name des Steuerelements zurück
handleSwitchControl	Gibt das Signal eines *SwitchControl* aus
handleButtonControl	Gibt das Signal eines *ButtonControl* aus
handleBlinkControl	Gibt das Signal eines *BlinkControl* aus
handlePwmControl	Gibt das Signal eines *PwmControl* aus

Tabelle 1: Wichtigsten Methoden der API

Die Argumente der Nutzer Anwendung beinhalten das Signal und werden als Name-Wert-Paaren übergeben (siehe Abbildung 18). Der *get*-Methode muss der Argument Name übergeben werden.

```
UserSoftware.py “pin_number=5” “state=1” “mode=11” signal_type=0” “number_of_cycles=5”
```

Abbildung 18: Aufruf der Nutzer Anwendung mit Argumenten

Die *log*-Methode Formatiert die Meldung in ein HTML Paragraph Element. Die Formatierte Ausgabe beinhaltet auch weitere Informationen wie das Datum, die Uhrzeit und die Kategorie. Letzteres wird mithilfe der HTML Klassen-Attribut angegeben (siehe Abbildung 19).

```
<p class="interpreter debug">2018-02-13 11:59:55 __main__</p>
<p class="interpreter debug">2018-02-13 11:59:55 get(name)</p>
<p class="interpreter debug">2018-02-13 11:59:55 key: 'name' - value: 'SwitchControl'</p>
<p class="interpreter debug">2018-02-13 11:59:55 get(mode)</p>
<p class="interpreter debug">2018-02-13 11:59:55 key: 'mode' - value: '11'</p>
<p class="interpreter debug">2018-02-13 11:59:55 get(pin_number)</p>
<p class="interpreter debug">2018-02-13 11:59:55 key: 'pin_number' - value: '5'</p>
<p class="interpreter debug">2018-02-13 11:59:55 get(state)</p>
<p class="interpreter debug">2018-02-13 11:59:55 key: 'state' - value: '0'</p>
<p class="interpreter debug">2018-02-13 11:59:55 Port 5 cleaned</p>
```

Abbildung 19: Formatierte Ausgabe mit *log*-Methode

SwitchControl, *ButtonControl*, *BlinkControl* und *PwmControl* sind Steuerelemente der Android Applikation.

Die *handle*-Methoden verwenden Methoden der Python Bibliothek RPi.GPIO um die Signale Auszugeben.

Der Raspberry Pi unterstützt keine Hardware-PWM, daher wird dies mit einer Software-Schleife emuliert. In einem getrennten Thread wird das Signal dauerhaft gesetzt und zurückgesetzt. Beendet sich das Programm, wird der Thread auch beendet und somit das PWM Signal ausgeschaltet (26).

Eine Schleife wäre aber in unserem Kontext ungünstig, weil die Anwendung kein Signal mehr empfängt bis die Schleife beendet wird. Damit die PWM eingeschaltet bleibt, wurde ein Daemon[7] in *PwmDaemon.py*-Datei implementiert, der die PWM in einem separaten Thread behandelt. Das Skript muss beim Aufruf in folgender Reihenfolge die Port-Nummer, die Frequenz und die Duty Cycle der PWM übergeben werden.

4.2.2. Kommunikation Schnittstelle

Die Schnittstelle leitet Signale an die Nutzer Anwendung weiter und stellt seine Ausgabe zur Verfügung (siehe Abbildung 19). Es ist in der Programmiersprache Java geschrieben. Das Modul generiert eine HTML Seite, die unter anderen die Ausgabe der Nutzer Anwendung enthält.

[7] https://pypi.python.org/pypi/python-daemon/ [13.02.18]

Here are the CGI environment variables/value pairs passed in from the CGI script:

CONTENT_TYPE
::
CONTENT_LENGTH
::
REQUEST_METHOD
:*GET*:
QUERY_STRING
:*state=0&pin_number=5&direction=0&signal_type=0&mode=11&name=SwitchControl*:
SERVER_NAME
:*192.168.42.1*:
SERVER_PORT
:*80*:
SCRIPT_NAME
:*/cgi-bin/RpiComIface.cgi*:
PATH_INFO
::

Here is the command used to run user software:

python /usr/lib/cgi-bin/RpiInterpreter.py state=0 pin_number=5 direction=0 signal_type=0 mode=11 name=SwitchControl

Here is the user software output:

2018-02-13 11:59:55 __main__
2018-02-13 11:59:55 get(name)
2018-02-13 11:59:55 key: 'name' - value: 'SwitchControl'
2018-02-13 11:59:55 get(mode)
2018-02-13 11:59:55 key: 'mode' - value: '11'
2018-02-13 11:59:55 get(pin_number)
2018-02-13 11:59:55 key: 'pin_number' - value: '5'
2018-02-13 11:59:55 get(state)
2018-02-13 11:59:55 key: 'state' - value: '0'
2018-02-13 11:59:55 Port 5 cleaned

Abbildung 20: Ausgabe der Raspberry Pi Kommunikationsschnittstelle

In der folgenden Tabelle werden die Wichtigsten Methoden dieser Schnittstelle aufgeführt.

Methode Name	Beschreibung
generateCmd	Generiert das Kommando zum Ausführen der Nutzer Anwendung
runUserSoftware	Führt Nutzer Anwendung mittels generierten Kommando aus
getProcessOutput	Stellt Ausgabe der Nutzer Anwendung auf HTML Seite dar

Tabelle 2: Wichtigsten Methoden der Raspberry Pi Kommunikationsschnittstelle

Die Schnittstelle erhält Signale von CGI-Skript namens *RpiComIface.cgi*. Letzteres übergibt das Signal über Umgebungsvariablen der Schnittstelle und ruft sie auf. Um die Manipulation von Signalen zu erleichtern, wurde die Java Bibliothek namens *CgiLib.java* bereitgestellt. Die Bibliothek wurde von hier[8] entnommen.

[8] https://www.javaworld.com/article/2076863/java-web-development/write-cgi-programs-in-java.html [13.02.18]

4.3. Android Applikation

Die Applikation besteht aus einem Activity. Sein Layout ist ein vertikales ListView. Er beinhaltet die Steuerelemente und ihre Beschreibung. Unter Beschreibung versteht man, welcher Port wird mit dem Element gesteuert und welches Signal wird ausgegeben, sowie zu welchem Zweck wurde es angelegt.

Wird der Bildschirm, vom linken Rand nach rechts betätigt, taucht das Menu auf. Unter „Configure Manager“ können bestimmte Eigenschaften der Applikation geändert werden. Das Einfügen und Löschen von Steuerelementen geschieht über „Add Control“ beziehungsweise „Remove Control“ (vgl. Abbildung 21).

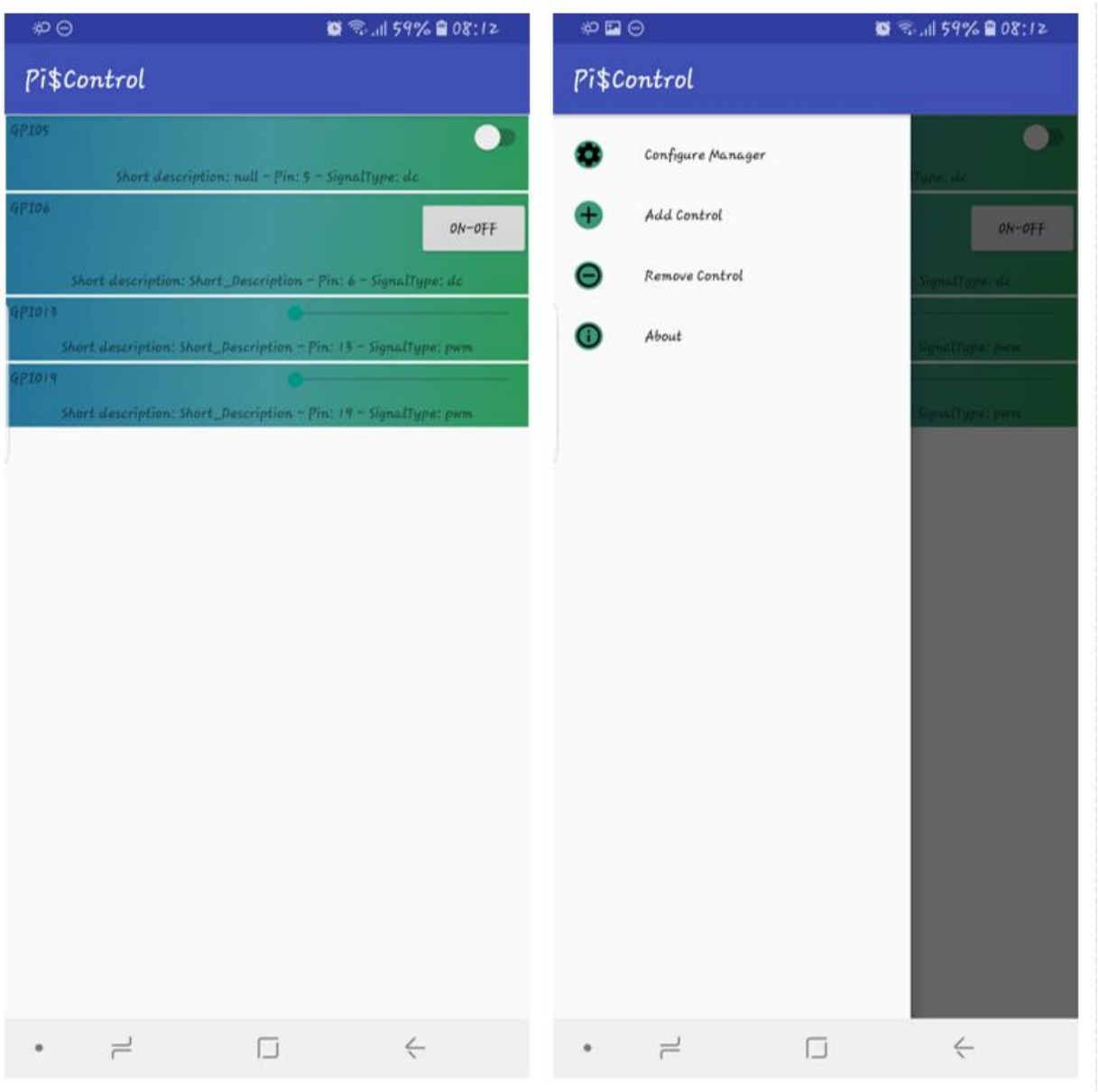

Abbildung 21: Android Applikation GUI

4.3.1. Steuerelement

Ein Steuerelement (*Control*) ist einer View, der bei einem Touch-Ereignis eine oder mehrere Signale an den RasPi sendet. Es ist eigentlich ein java-Klasse, die folgende Kriterien erfüllt (siehe Abbildung 22).

- Implementierung eines *Control*-Unterinterface
- Vererbung einer View-Unterklasse.
- Registrierung eines Touch-Events auf der geerbten View-Unterklasse.

In der folgenden Abbildung ist das Klassendiagramm eines *SwitchControl* dargestellt. Die vererbte Unterinterface von *Control* ist *OutputControl*. Die Klasse erbt auch vom View-Unterklasse Switch. Die Vererbung von Interface OnCheckedChangeListner hilft dabei, den Touch-Event auf der geerbten View-Unterklasse zu registrieren. Die einzelnen Kriterien werden in folgenden Abschnitten erklärt.

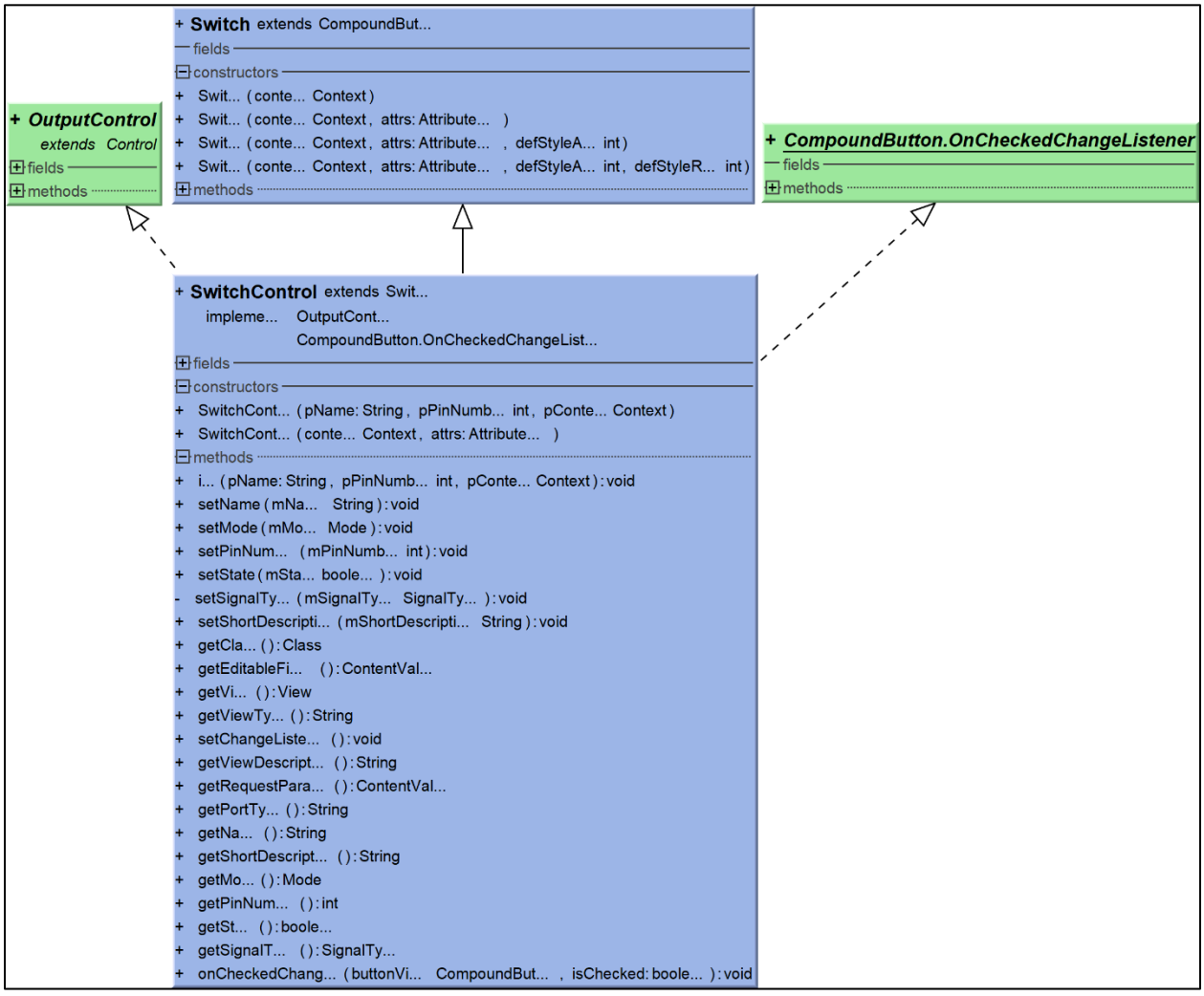

Abbildung 22: *SwitchControl* Klassendiagramm

Implementierung von *Control*-Unterinterface

Control-Unterinterfaces beinhalten, notwendige Methoden für die Anzeige eines Steuerelements und der dazugehörigen Übertragung von Signalen.

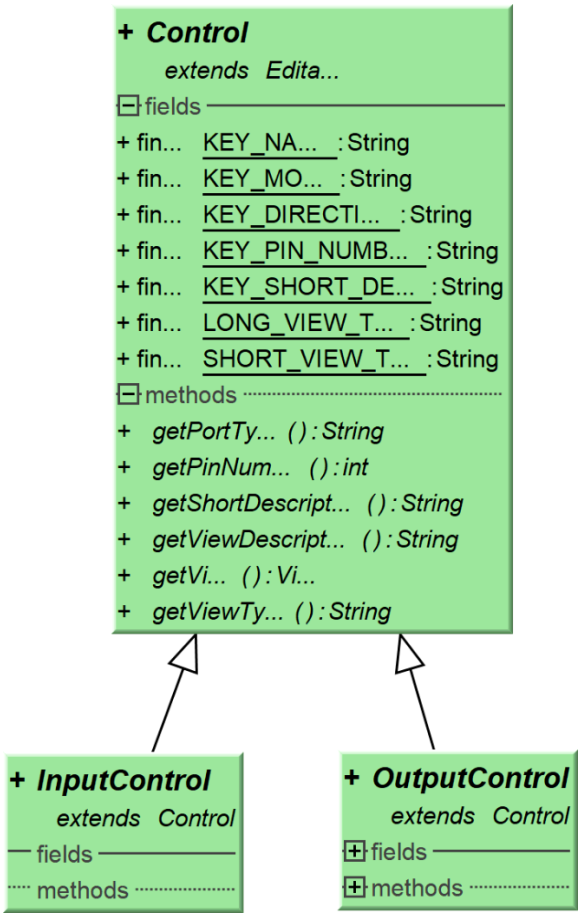

Abbildung 23: Klassendiagramm von *Control*

Die Wichtigsten Methoden diesen Unterinterfaces werden in der folgenden Tabelle aufgeführt.

Methode Name	Beschreibung
getView	Gibt View zurück, der zum Steuerelement gehört
getViewDescription	Gibt die Beschreibung des Steuerelements zurück
getRequestParams	Gibt Parameter für die Sendung des Signals zurück
setChangeListener	Hier wird festgelegt, auf welchem Ereignis der View reagieren soll

Tabelle 3: Wichtigsten Methoden von Control Interface

Vererbung einer View-Unterklasse

Durch Vererbung einer View-Unterklasse wird festgelegt, mit welchem View der Nutzer interagieren wird. Die zu vererbte Unterklasse hängt vom Signal typ ab. Für ein DC Signal wird beispielweise vom Switch-Klasse (Schalter) geerbt. Der View wird vom *getView*-Methode zurückgegeben.

Registrierung eines Touch-Events auf der geerbten View-Unterklasse

Wie schon oben erläutert, soll es über *setChangeListener*-Methode geschehen. Das Registrierte Event hängt vom geerbten View ab. Auf Switch zum Beispiel wurde OnCheckedStateChanged-Listener registriert werden (siehe Abbildung 22).

4.3.2. Anzeige von Steuerelement

Ein Steuerelement wird immer mit seiner Beschreibung in einem RelativeLayout angezeigt. Diese Anzeige übernimmt die *ControlView*-Klasse. Der Konstruktor dieser Klasse muss ein *Control* übergeben werden.

Die Klasse verwendet *Control*-Methoden, um das Steuerelement und seine Beschreibung abzurufen (siehe Tabelle 3). Die Position der Elemente in dem RelativeLayout bestimmen die LayoutParams[9].

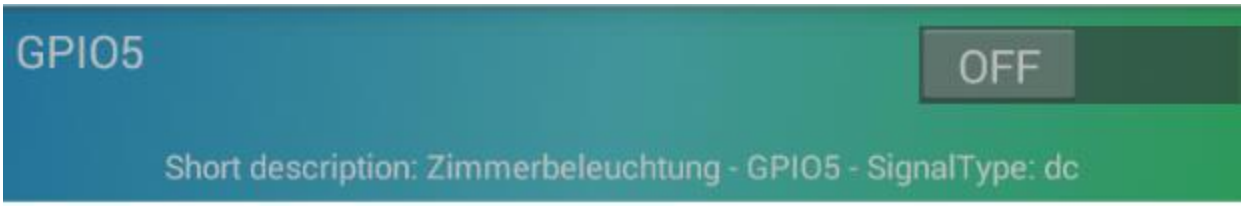

Abbildung 24: Anzeige eines Steuerelements (*SwitchControl*)

4.3.3. Signalübertragung

Klassen die für die Kommunikation zuständig sind, wurden im Paket *communication* zusammengestellt.

Wie bereits erwähnt, werden die Signale als HTTP Anfragen über WLAN an den RasPi übertragen. Es gibt verschiedene Anfragemethoden: GET, POST, PUT, DELETE. Für jede Methode gibt es eine Klasse, die für die Übertragung zuständig ist (siehe Abbildung 25). Damit die Oberfläche während der Übertragung nicht blockiert wird, werden Anfragen in einem separaten Thread[10] ausgeführt.

[9] https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html [23.02.2018]
[10] https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html [16.02.18]

Die Klasse *AsyncTask*[11] vom Android Framework erleichtert die Gestaltung von separaten Thread.

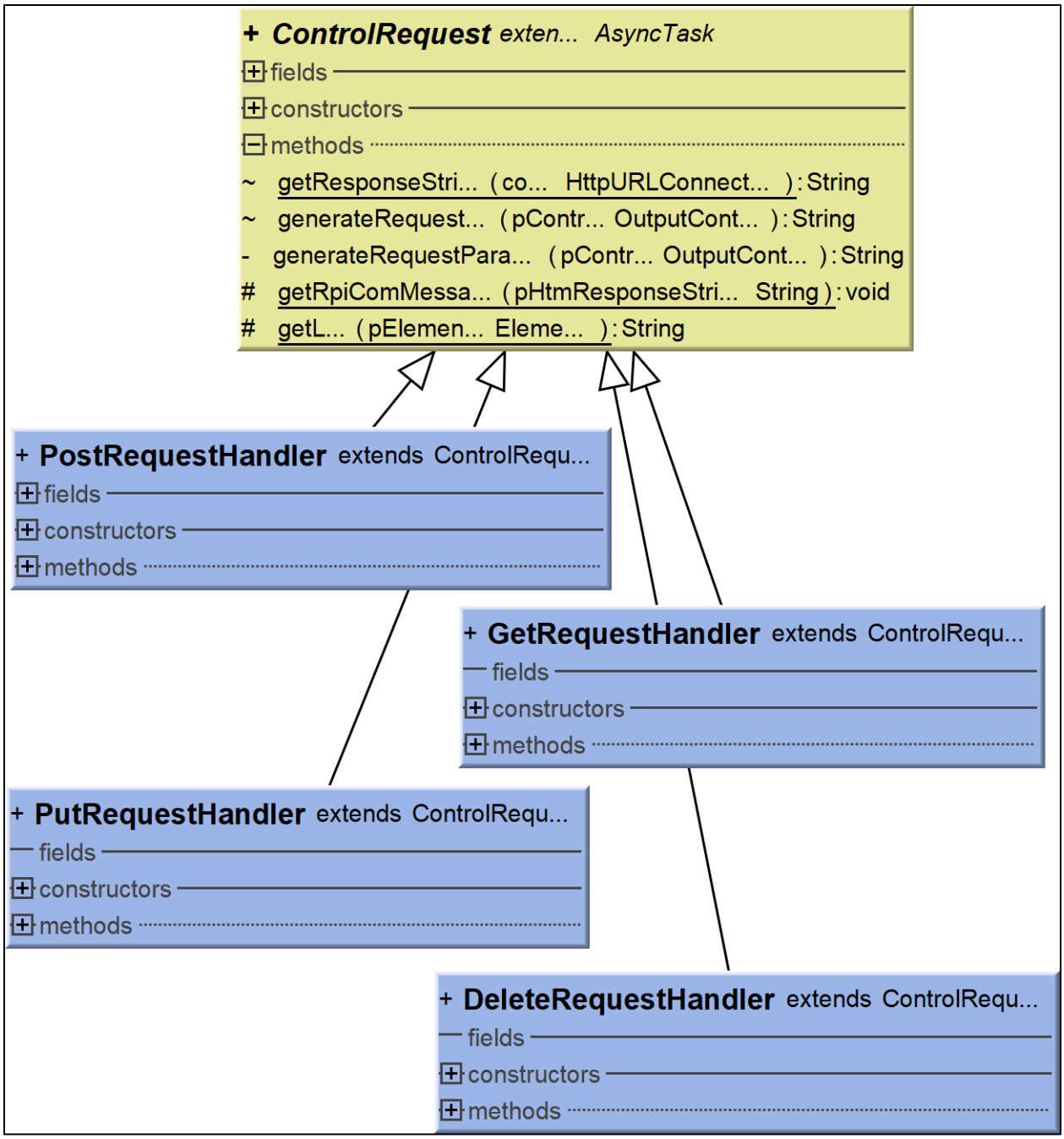

Abbildung 25: Klassendiagramm *communication*-Paket

Die *ControlRequest*-Klasse umfasst, Konstanten und grundlegende Methoden, die für die Übertragung der Signale und die Bearbeitung der RasPi Rückmeldung notwendig sind. Letzteres beinhaltet unter anderen Die Ausgabe der Nutzer Anwendung (siehe Abbildung 20).

[11] https://developer.android.com/reference/android/os/AsyncTask.html [16.02.2018]

```
// For the connection
public static final String URL_SEPARATOR = "/";
public static final String URL_PARAM_SEPARATOR = "?";
public static final String FORM_PARAM_SEPARATO = "&";
public static final String KEY_VALUE_SEPARATOR = "=";
public static final String PROTOCOL = "http://";
public static final String RPI_COM_IFACE_NAME = "RpiComIface.cgi";
public static final String CGI_BIN_PATH = "/cgi-bin/";
// For Logging
public static final String COM_MSG_ID = "comMessage";
public static final String INTERPRETER_CLASS_NAME = "interpreter";
public static final String INTERPRETER_DEBUG_CLASS_NAME = INTERPRETER_CLASS_NAME + " debug";
public static final String INTERPRETER_INFO_CLASS_NAME = INTERPRETER_CLASS_NAME + " info";
public static final String INTERPRETER_WARN_CLASS_NAME = INTERPRETER_CLASS_NAME + " warn";
public static final String INTERPRETER_TODO_CLASS_NAME = INTERPRETER_CLASS_NAME + " todo";
public static final String INTERPRETER_ERROR_CLASS_NAME = INTERPRETER_CLASS_NAME + " error";
public static final String INTERPRETER_NONAME_CLASS_NAME = INTERPRETER_CLASS_NAME + " noname";
// HTTP Methoden
public static final String METHOD_POST = "POST";
public static final String METHOD_GET = "GET";
public static final String METHOD_PUT = "PUT";
public static final String METHOD_DELETE = "DELETE";
```

Abbildung 26: Konstanten der *ControlRequest*-Klasse

Die Konstanten-Namen, die in der Abbildung 26 zu sehen sind mit dem Suffix „CLASS_NAME“, bezeichnen die HTML-Klassen, die in der API *log*-Methode vergeben sind. Durch sie kann die Applikation in der RasPi Rückmeldung die Meldungen der Nutzer Anwendung erkennen und Klassifizieren.

Die Übertragung ist in *ControlRequest*-Unterklassen implementiert. Dem Konstruktor diesen Klassen muss ein *Control*-Unterinterface übergeben werden. In der *doInBackground*-Methode wird in einem separaten Thread, die URL generiert, das Signal Übertragen und die Ausgabe der Nutzer Anwendung empfangt.

```java
@Override
protected String doInBackground(String... params) {
    try {
        // Generate URL
        URL postUrl = new URL(generateRequestUrl(mControl));
        mConnection = (HttpURLConnection) postUrl.openConnection();
        // Setup Connection
        mConnection.setRequestMethod(METHOD_POST);
        mConnection.setReadTimeout(10000);
        mConnection.setConnectTimeout(5000);
        // Send Request
        mConnection.connect();
        // Get Response
        mResponseString = getResponseString(mConnection);
    } catch (MalformedURLException e) {
        Toast.makeText(mContext, e.getMessage(), Toast.LENGTH_LONG).show();
    } catch (IOException e) {
        Toast.makeText(mContext, e.getMessage(), Toast.LENGTH_LONG).show();
    }
    return null;
}
```

Abbildung 27: Überschreibung der *doInBackground*-Methode in *PostRequestHandler*-Klasse

4.3.4. Einfügen von Steuerelement (*Control*)

Die Implementierung dieser Funktionalität erhält zwei zwischenschritte. Zuerst wurden GPIO-Port der RasPi auf Android Applikation abgebildet und danach Ihre Verwaltung implementiert. Schließlich konnte das Einfügen von Steuerelement implementiert.

Abbildung GPIO-Ports auf Android Applikation

Zweck dieses zwischenschrittes ist die Abbildung der Port Eigenschaften; darunter versteht man unter anderen:

- Welche Signale Sie ausgeben können
- Wie viele *Control* einen Port gleichzeitig steuern dürfen
- Welche Pin Nummer sie besitzen je nach Nummerierungssystem
- Um Welche Port Type handelt es sich (GPIO, POWER oder GROUND)

Diese Eigenschaften wurden in *Port-Klasse* und ihre Unterklassen nachgebildet.

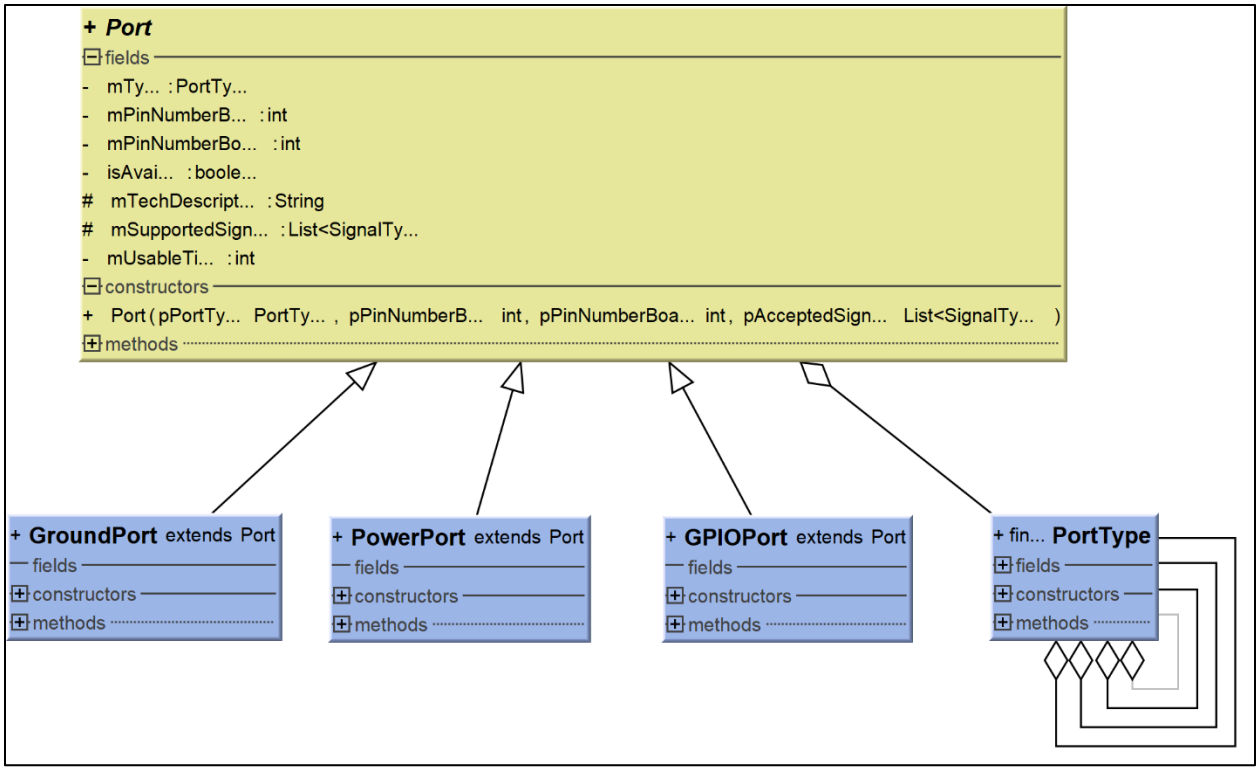

Abbildung 28: Klassendiagramm von *Ports*

Port ist eine Abstrakte Klasse, die gemeinsame Port Eigenschaften und Methoden zusammenfasst. *PortType* ist ein Enum[12]; er bestimmt der Port Type (GPIO, GROUND, POWER).

GPIO-Ports Verwaltung

Diese Aufgabe übernimmt die *ControlManager*-Klasse. Sie umfasst GPIO-Ports Informationen. Darunter versteht man wie viele Ports sind vorhanden, welche davon sind verfügbar, welches Nummerierungssystem ist angewendet. Sie beinhaltet auch die URL des RasPi. Die Klasse wurde nach dem Singleton[13] Entwurfsmuster implementiert, damit es nur eine einzige Instanz existiert.

[12] https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html [17.02.18]
[13] https://de.wikipedia.org/wiki/Singleton_(Entwurfsmuster) [17.02.18]

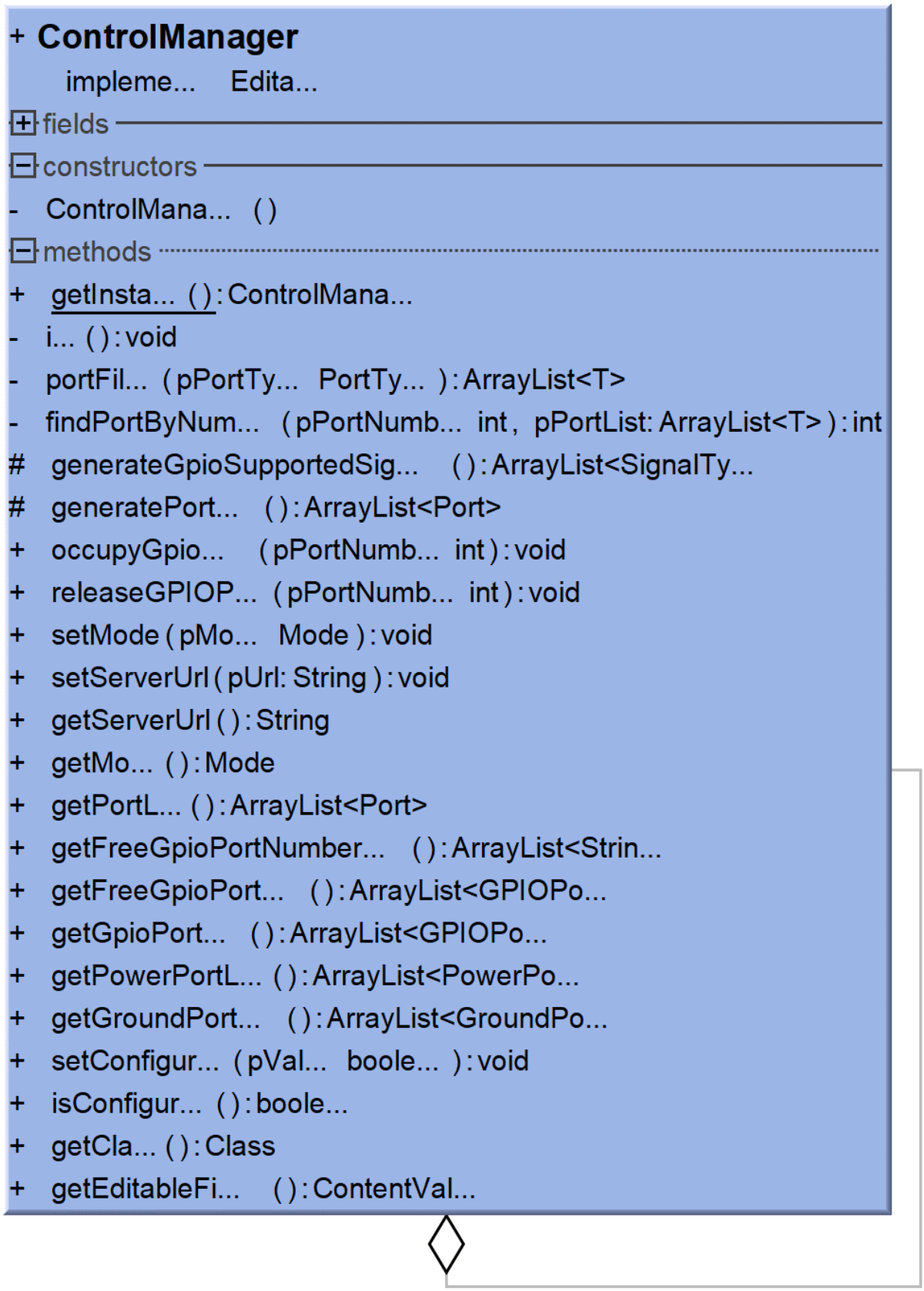

Abbildung 29: Klassendiagramm *ControlManager*

Die Wichtigsten Methoden dieser Klasse werden in der folgenden Tabelle aufgeführt:

Methode Name	Beschreibung
generatePortList	Generiert Port-Liste gemäß Abbildung 4
PortFilter	Filtert die Port-Liste nach dem Port Typ
getFreeGpioPortNumberList	Gibt die Liste der Verfügbare Port-Nummer zurück
getServerUrl	Gibt den URL des Servers zurück

Tabelle 4: Wichtigsten von *ControlManager*-Klasse

Einfügen-Prozess

Der Einstiegspunkt dieses Prozesses ist das Interface *Editable*. Es liefert Informationen, die bei der *Control* Instanziierung gebraucht werden und besitzt *getClazz* und *getEditableAttributes*-Methoden:

Die Methode *getClazz*-Methode gibt das Klassen-Objekt des *Control* zurück.

getEditableAttributes: Diese Methode liefert *Control* Attribute, die für Erstellung des Steuerelements notwendig sind. Sie werden als Name-Wert-Paare zurückgegeben. Als Wert wird den Datentyp des Attributes als FQN[14] zurückgegeben (vgl. Abbildung 30).

```
@Override
public Class getClazz() { return getClass(); }

@Override
public ContentValues getEditableAttributes() {
    ContentValues result = new ContentValues();
    result.put(KEY_NAME, String.class.getName());
    result.put(KEY_PIN_NUMBER, Integer.class.getName());
    result.put(KEY_FREQUENCE, Integer.class.getName());
    result.put(KEY_DUTY_CYCLE, Integer.class.getName());
    result.put(KEY_PWM_OUTPUT_TYP, PwmOutputType.class.getName());
    result.put(KEY_SHORT_DESC, String.class.getName());
    return result;
}
```

Abbildung 30: Implementierung von *Editable*-Methoden in *PwmControl*-klasse

Control-Interface ist ein *Editable* (Vererbung) und somit besitzen alle *Controls* beide Methoden.

Mit den Attributen in der Abbildung 30 wird folgende Dialog[15] erstellt.

[14] https://en.wikipedia.org/wiki/Fully_qualified_name#Examples [17.02.18]
[15] https://developer.android.com/guide/topics/ui/dialogs.html [17.02.2018]

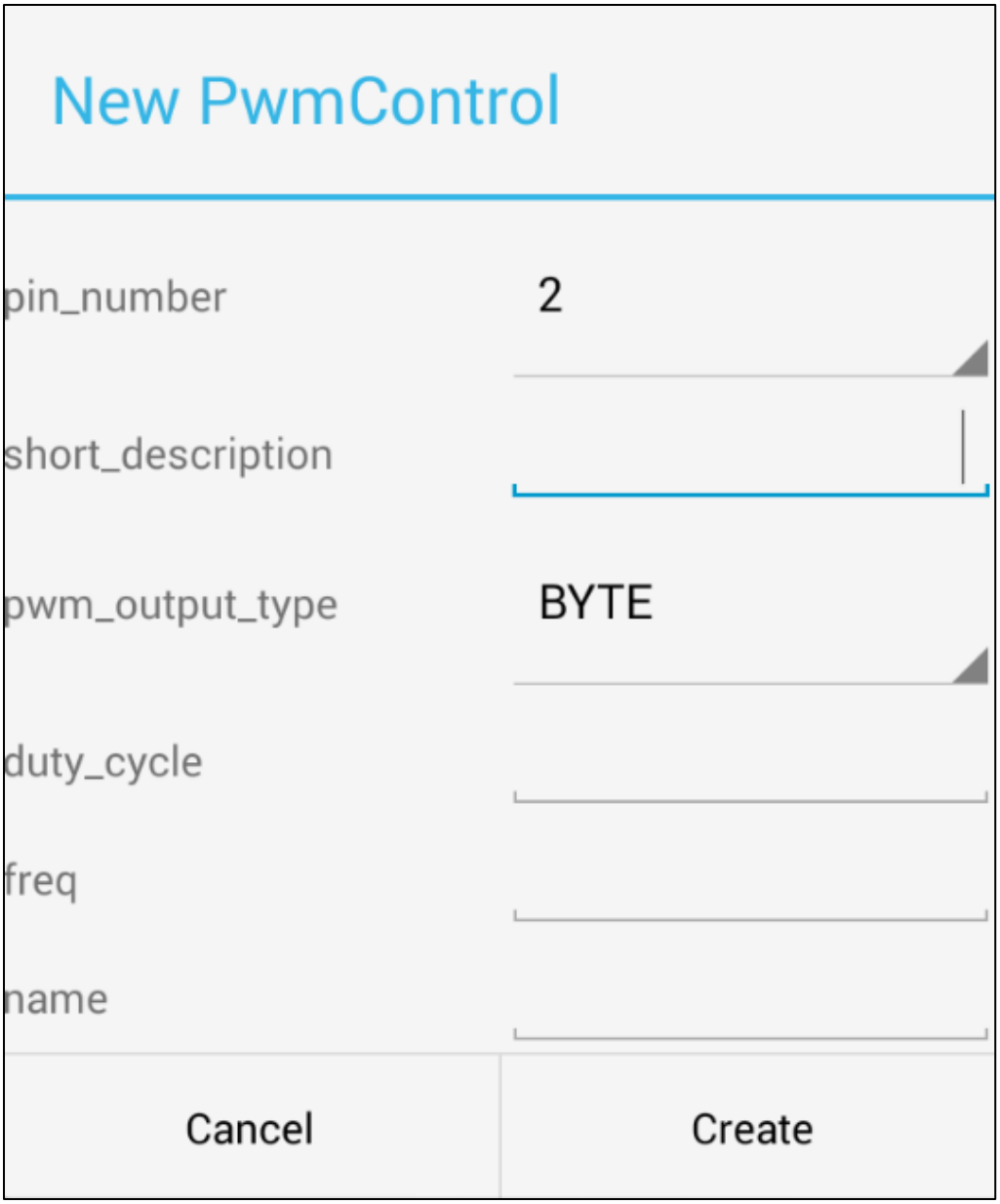

Abbildung 31: Dialog zur Erstellung eines *PwmControl*

Die Erstellung von Dialogs Übernimmt die *EditableDialog*-Klasse. Sein Konstruktor wird dem *Editable* übergebe. Die *getContentView*-Methode von *EditableDialog*-Klasse macht eine Schleife über die übergebenden Attribute und erzeugt für jeden Eintrag ein Horizontales LinearLayout, der aus zwei Views besteht: Das erste beinhaltet den Eintragsnamen und das Zweite sein Eingabefeld. Das Horizontale LinearLayout wird vom *EditableAttributEntryView* generiert.

Abbildung 32: Generiertes Layout fürs Attribut Pin Nummer

Das Dialog-Layout besteht aus ein vertikales LinearLayout, das Layouts der Abbildung 32 umfasst.

Beachtenswert in Abbildung 32 ist, dass das Eingabefeld ein Spinner[16] ist. Hier werden nur Nummern von verfügbaren GPIO-Ports aufgelistet. Die Liste befindet sich im *ControlManager* (vgl. Abbildung 33). Die Behandlung erfolgt über die *setupValueView*-Methode.

```
private void setupValueView() {
    String val = mPair.getValue().toString();
    // generate Spinner for pin selection; the spinner show only available pins;
    // the available pins depend on the pinUsableTime in Port class
    if(mPair.getKey().equals(Control.KEY_PIN_NUMBER)) {
        mValue = new Spinner(mContext);
        ArrayAdapter<String> adapter = new ArrayAdapter<>(mContext,
                android.R.layout.simple_spinner_dropdown_item,
                ControlManager.getInstace().getFreeGpioPortNumberList());
        Spinner v = (Spinner) mValue;
        v.setAdapter(adapter);
    }
```

Abbildung 33: Behandlung des Eingabefeldes fürs Pin Nummer

Bei Enum Daten-Typ wird auch ein Spinner generiert.

```
// generate spinner when value equals FQN of Enum
else if (val.equals(Enum.class.getName())) {
    mValue = new Spinner(mContext);
    ArrayAdapter<String> adapter = new ArrayAdapter<>(mContext,
            android.R.layout.simple_spinner_dropdown_item, getEnumValues(val));
    Spinner v = (Spinner) mValue;
    v.setAdapter(adapter);
```

Abbildung 34: Behandlung von Enum Eingabefeldern

Bestätigt man seine Eingabe mit „Create“ (Abbildung 31), wird der *Control* erstellt und angezeigt.

Die Erstellung des Controls übernimmt die *ControlFactory*-Klasse. Seine *getControl*-Methode generiert das *Control*. Die Klasse wurde nach dem Factory[17] Entwurfsmuster implementiert.

[16] https://developer.android.com/guide/topics/ui/controls/spinner.html [17.02.2018]
[17] https://www.tutorialspoint.com/design_pattern/factory_pattern.htm [17.02.2018]

5. Implementierung

Dieser Kapitel umfasst Software, Entwicklungsumgebungen und Bibliotheken, die diese Bachelorarbeit verwendet wurden. Verfolgte Anleitungen werden auch referenziert und die Abweichungen auch erläutern. Der Kapitel unterteil sich in Raspberry Pi Unterkapitel und Android Unterkapitel.

5.1. Raspberry Pi

5.1.1. Inbetriebnahme

Für die Inbetriebnahme des RasPi ist ein Betriebssystem erforderlich. Es muss vorerst auf einer microSD-Karte installiert werden.

Das Betriebssystem dieser Bachelorarbeit (2017-07-05-raspbian-jessie) wurde auf Herstellerseite[18] heruntergeladen, aus dem Archiv entpackt und mit einem zusätzlichen Programm (Etcher[19]) auf die microSD-Karte kopiert (installiert).

Eine SSH-Verbindung mit dem Laptop wurde dann eingerichtet, sodass der Raspberry Pi selbst auf Tastatur und Monitor verzichten kann. Die Anleitung befindet sich auf (28).

Nach der Startvorgang muss man sich anmelden. Der Default Benutzername ist „pi“ und das Passwort ist „raspberry“. Die Auflösung wurde für eine bessere Darstellung erhöht

Nach dem Startvorgang erflogt die Anmeldung unter dem Benutzernamen „pi“ und das Passwort „raspberry“. Für eine bessere Darstellung wurde in Grundeinstellung unter **Advanced Option > Resolution** die Auflösung auf 1920*1080 geändert. Die Grundeinstellungen erhält man durch Eingabe des Befehls **sudo raspi-config** in der Konsole.

5.1.2. WLAN Access Point: Einrichtung

Die Installation und Einrichtung des Access Point erfolgte nach der Anleitung auf (29). Es handelt sich um ein PDF-Datei; Eine bearbeitete Version befindet sich im Anhang unter **RasPi/Anleitungen/wlanAP.pdf**; die Datei beinhaltet Kommentare, die Besonderheiten oder Unterschiede zu dieser Bachelorarbeit kennzeichnen.

5.1.3. Webserver: Apache2 – Installation und Einrichtung

Apache Server (Version 2.4.29) wurde durch Ausführen folgender Kommando in Terminal installiert: *sudo apt-get install apache-y* (30). Die Installation muss man noch bestätigen. Dies erfolgt durch Eingabe von „y“ und drücken auf Enter-Taste.

[18] https://downloads.raspberrypi.org/raspbian/images/raspbian-2017-07-05/ [18.02.18]
[19] https://etcher.io/?ref=etcher_update [18.02.18]

Der Default Pfad für die Server Skripten ist */usr/lib/cgi-bin/*. Da wurden die RasPi Module abgelegt. Damit GPIO-Port vom Smartphone ansprechbar sind, musst der Client Nutzergruppe (*www-data*) zur GPIO (*gpio*) eingefügt werden. Zu diesem Zweck führt man folgende Kommando in der Konsole aus: *sudo adduser www-data gpio*.

5.1.4. Module

Die Entwicklungsumgebung der Kommunikation Schnittstelle und der API ist ein Texteditor namens Geany[20]. Sie bietet viele Vorteile wie Syntaxhervorhebung, Autovervollständigung und Code-Folding.

Die API wird mit der Programmiersprache Python in der Version 2.7.9 umgesetzt. Sie erfordert folgende Bibliotheken:

- **RPI.GPIO** (Version 0.6.3): vereinfacht den Zugriff auf GPIO-Pins; kann mit dem Befehl *sudo apt-get install python dev python-rpi.gpio* installiert werden.
- **sys**: ermöglicht den Zugriff auf Skript Argument
- **datetime**: für Datum und Zeit Ausgabe
- **time**: zum Anhalten der Anwendung beim Blinken
- **runner**: für PWM Behandlung (siehe **Fehler! Verweisquelle konnte nicht gefunden werden.**); Sie befindet sich in **daemon**[21] Paket.

Die Kommunikation Schnittstelle wurde in der Programmiersprache Java in der Version ... umgesetzt. Sie benötigt folgende Dateien:

- CgiLib.java: vereinfacht der Zugriff auf HTTP Anfrage Parametern; Sie befindet sich im Anhänge unter dem Verzeichnis **/RasPi/Software**. Die Originale Bibliothek befindet sich hier (24).
- RpiComIface.cgi: Empfängt die Anfrage und leitet sie an die Kommunikation Schnittstelle weiter; sie ist im Verzeichnis */RasPi/Software*

Die Module befinden sich im Anhang unter */RasPi/Anwendung/.*

5.2. Android Anwendung

Die Anwendung wurde mit der Entwicklungsumgebung Android Studio (Version 2.3.3) entworfen. Die Umgebung wurde von Google und für Android veröffentlich und integriert schon alle Komponenten, die Für die Entwicklung notwendig sind. Zusätzliche Bibliotheken kann man immer im *build.gradle* Datei einbinden (31). Für diese Bachelorarbeit wurden folgende Bibliotheken eingebunden:

- **Jsoup**[22] (Version 1.11.2): vereinfach die Bearbeitung der RasPi Rückmeldung.

[20] https://www.geany.org/Main/HomePage [18.02.2018]
[21] https://pypi.python.org/pypi/daemon-runner [18.02.2018]
[22] https://github.com/jhy/jsoup [18.02.2018]

- **Design Support Library**[23] (Version 25.3.1): Für App Menü

Die Anwendung wurde mit Android SDK Version 25 kompiliert kann unter Android Version 4.4 oder früher laufen.

Die Anwendung wurde zuerst zeitnah mit einem Android Emulator namens Genymotion[24] entwickelt und getestet. Die Verwendung dieser Emulator im Android Studio erfordert der Genymotion Plugin. Der Plugin kann man sich auf Android Repository (**File > Setting > Plugins**) holen und installieren. Nach der Installation wird das Genymotion Symbol in Symbolleiste angezeigt. Beim erst Start ist der Emulator einzurichten.

[23] https://developer.android.com/topic/libraries/support-library/packages.html#design [18.02.2018]
[24] https://www.genymotion.com/ [18.02.2018]

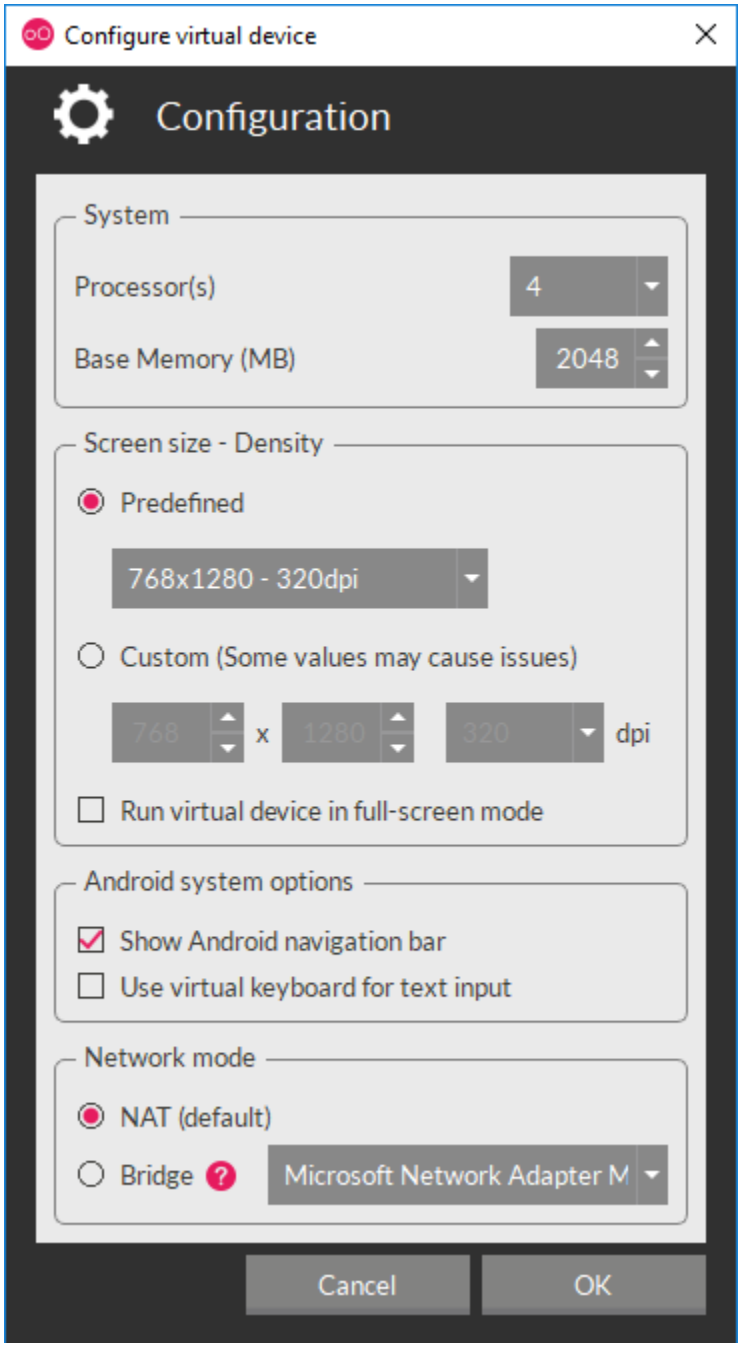

Abbildung 35: Emulator Konfiguration

Auf dem Emulator wurde Android 4.4.2 installiert

Anschließend wurde die Anwendung auf ein Samsung Galaxy S8 Plus (Android Version 7 und 8) getestet.

Der Quellcode befindet sich im Anhang unter **/Smartphone /Pi$Control/**

6. Test

Die vollständig implementierten Funktionalitäten der Android App und seine API werden in diesem Kapitel getestet. Hier wird geprüft, inwieweit die Anwendung unseren am Anfangt definierten Voraussetzungen und Bedienungen erfüllt. Dieser Kapitel beinhaltet zwei Schritten: der Aufbau und die Bewertung der Ergebnisse.

6.1. Test Aufbau

Sowohl die Android Anwendung als auch der Raspberry Pi werden eingerichtet.

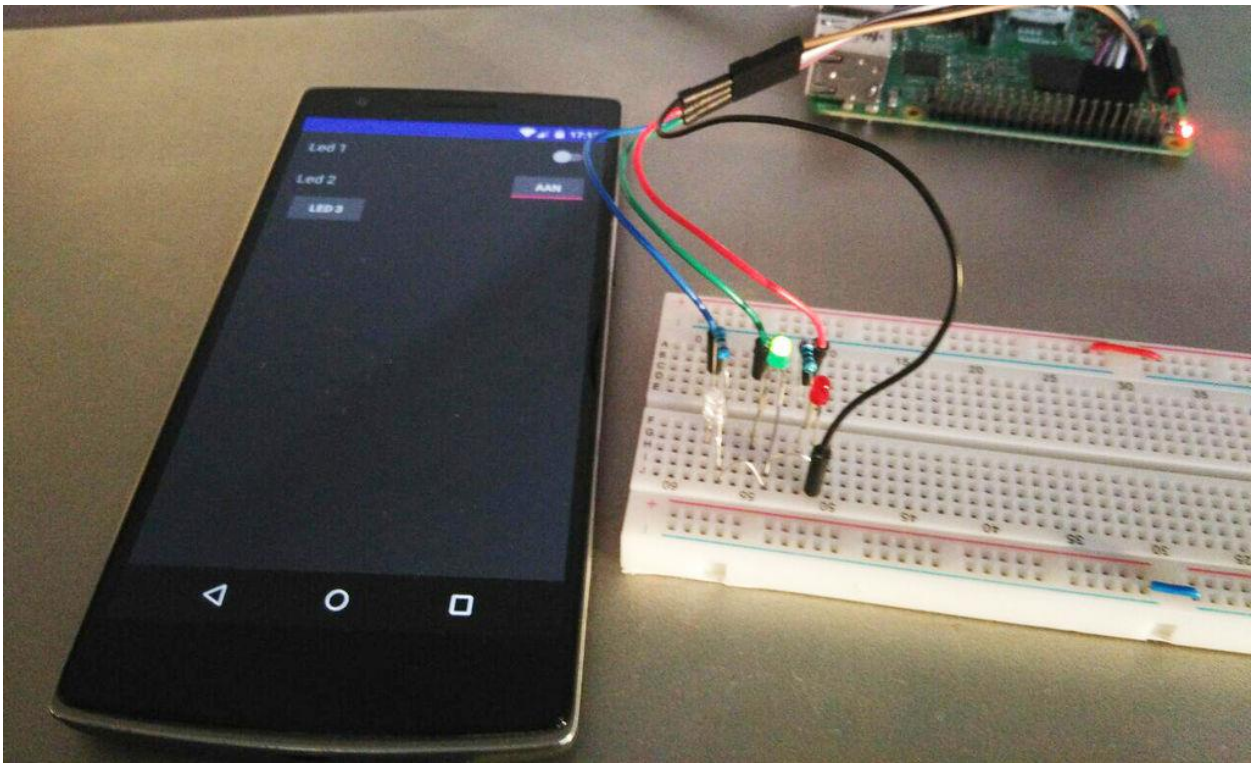

Abbildung 36: Mein Bild Kommt noch HAHAH

Android Smartphone

Die Anwendung wurde auf ein Samsung Galaxy S8 Plus installiert und beinhaltet vier *Control* für den Test: einen Schalter, einen Knopf, und zwei Schieber (vgl. Abbildung 37). Weiterhin wird auch das Einfügen von *Control* getestet.

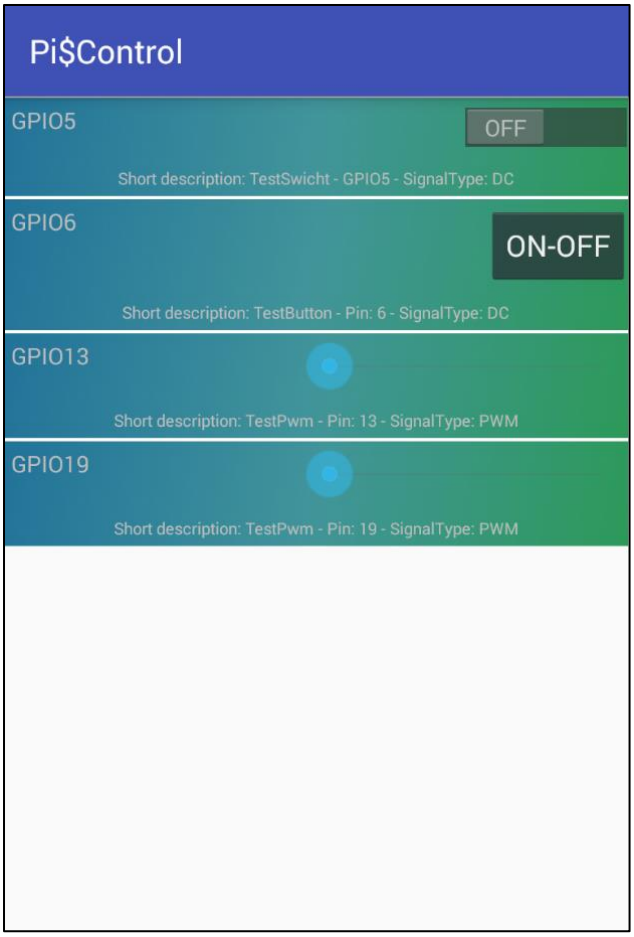

Abbildung 37: Android App für den Test

Der Letzte *Control* (GPIO9) liefert ein 8 Bit PWM Signal (Skala von 0 bis 255) aus und wird nicht komplett getestet. Es wird nur geprüft, ob der 8 Bit wert übertragen wird. Hier wurde das BCM Nummerierungssystem verwendet.

Raspberry Pi

Hier wird mit LEDs getestet, ob Signale richtig an den GPIO-Ports ankommen. Zu diesem Zweck werden LEDs am GPIO-Ports angeschlossen, die bei der Android App Angesprochen sind (siehe Abbildung 37). Die Betroffene Ports sind: GPIO5, GPIO6, GPIO13. Jeder LED wird wie auf Abbildung 38 verdrahtet.

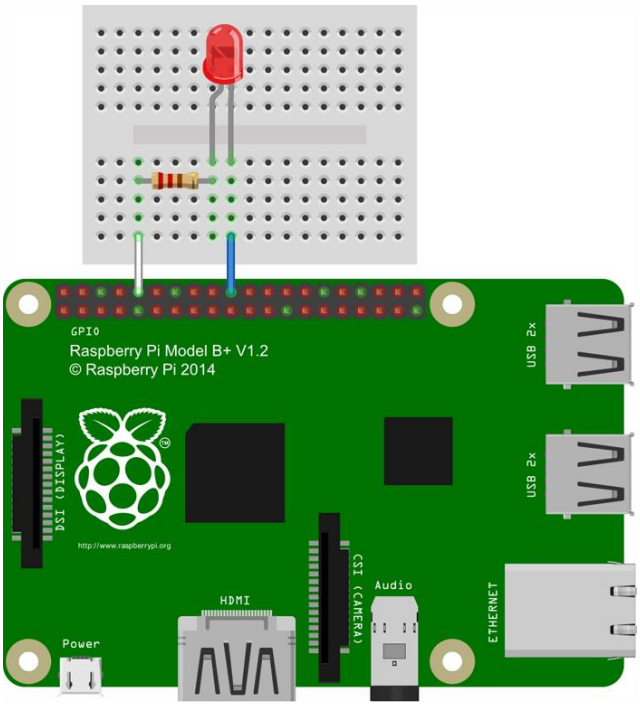

Abbildung 38: LED Verdrahtung auf GPIO-Port

Schließlich wird auf RasPi eine kleine Anwendung zum Test der API Funktionalitäten geschrieben. Die Anwendung nutzt API Funktionen um die Signale zu bewerten und darauf zu reagieren.

```
# start of programm
if __name__ == "__main__":
        AppLib.log("d", "__main__")
        #handleParam()
        if(AppLib.getName() == AppLib.SWITCH_CONTROL_NAME):
                AppLib.handleSwitchControl()
        elif(AppLib.getName() == AppLib.BUTTON_CONTROL_NAME):
                AppLib.handleButtonControl()
        elif(AppLib.getName() == AppLib.BLINK_CONTROL_NAME):
                AppLib.handleBlinkControl()
        else:
                AppLib.log("t", "Control Handler not implemented yet...")
```

Abbildung 39: Raspberry Pi Test Anwendung

6.2. Bewertung der Ergebnisse

Die LED von GPIO5 geht an und aus, wenn man den Schalter Ein-/Ausschaltet.

Die LED von GPIO6 geht an, wenn der Knopf gedrückt ist. Beim Loslassen geht sie wieder aus

Die LED von GPIO13 wird auch richtig gedimmt; je Großer der Schieberwert ist, desto heller wird die LED und umgekehrt. Jedoch geht die LED erst aus und dann an. Die Erklärung liegt bei der *handlePwmControl*-Methode. Wie im 4.2.1 erläutern, werden PWM mit Daemon gehandelt. Daemon werden aber nach Ihrem Start im Prozess umgewandelt und man kann also auf seine

Variablen nicht mehr zugreifen. Beim Änderung der Schiebewert wird zuerst der laufende Daemon (Prozess) gestoppt und ein Neuen mit dem Aktuellen Schiebewert gestartet.

Der 8 Bit PWM Wert wird auch richtig übertragen.

Die Steuerelemente verhalten sie also wie erwartet; das Einfügen eines Steuerelements durch App Menü wurde auch erfolgreich durchgeführt.

7. Erreichter Stand & Ausblick

7.1. Erreichter Stand

Anhand der Kriterien, die im Pflichtheft in Kapitel 4.1 aufgeführt wurden, wird der erreichte Stand aufgezählt.

Die Android Anwendung wurde mit Sinnvollem Steuerelemente implementiert. Zur sinnvollen Steuerelemente zählen ein der Schalter, die Taste und der Schieber. Die Sinnvolle Steuerelemente sind reine Aus

Die RasPi Schnittstelle und die API wurden implementiert und deren Funktionalitäten im Kapitel 6 getestet.

Die Musskriterien des Pflichtenhefts konnten während der Bearbeitungszeit umgesetzt werden.

Einige Wunschkriterien konnten auch erfüllt werden.

Der 8Bit-Wert PWM Steuerelement könnte auch in der Bearbeitungszeit implementiert werden. Es wurde auch geprüft, ob der Wert Korrekt umgewandelt wird.

Das Einfügen von Steuerelemente konnte bereit implementiert und Funktionsfähig. Der Nutzer bestimmt beim Einfügen die Eigenschaften des Steuerelements. Jedoch konnte aus Zeitlichen Gründen das Löschen von Steuerelemente nicht implementiert werden.

Aufgrund verbliebene Zeitdauer konnten die restlichen Wunschkriterien nicht umgesetzt werden. Dies betrifft die Anzeige der RasPi Ausgabe auf Android App und das Löschen von Steuerelemente.

Aus den Wunschkriterien konnten zwei von vier umgesetzt werden.

Die Abgrenzungskriterien wurden als nichterreichbar deklariert. Am Anfang wurde kein verfügbares Projekt gefunden, um die Android Anwendung zu testen.

7.2. Fazit & Ausblick

Das Bachelorthema zu wählen, um ein modular und wiederverwendbar Android Anwendung für die Steuerung auf Raspberry Pi basierten eingebettet System zu entwerfen, war eine sinnvolle Entscheidung. Während der Entwurf wurden neuen Wissen erworben, die in der Zukunft sicherlich von Vorteil sein werden.

Überlegungen mussten angestellt werden, damit am Ende ein sinnvolles Projekt entstehen entsteht. Vieles aus dem Studium erworbenen Kenntnissen wurden angewendet und halfen bei der Umsetzung.

Der Grund sich mit dem Thema dieser Bachelorarbeit auseinanderzusetzen war einen sinnvollen Einstiegspunkt für die Entwicklung Bedienungssystem von Raspberry Pi basierten eingebettet System anzubieten. Durch das Modul Konzept ist der Nutzer möglich, Teile des Bedienungssystems auszutauschen. Die Konfigurierbare Oberfläche bietet den Nutzer die Möglichkeit, seine eigene Steuerelemente zu erstellen und die Übertragenen Signale mithilfe der API nach seinem Wunsch implementieren.

Als Übertragungsschnittstelle wurde WLAN für seine längere Reichweite gewählt. Ein weiterer Grund war, dass sowohl der Raspberry Pi 3 Modell als auch das Smartphone schon ein eingebautes WLAN Modul besitzen. Als Übertragungsmodus wurde der Access Point Variante gewählt, weil er Einfach mit vielen Android Geräten Kompatibel und vermeidet, dass der Raspberry Pi von anderen Netze abhängig ist.

Das Bedienungssystem bietet noch viele Erweiterungsmöglichkeiten.

Die restlichen Wunschkriterien können noch implementiert werden. Dazu gehören die Anzeige der Konsole Ausgabe der Raspberry Pi auf der Android Applikation und das Löschen von Steuerelementen.

Die Speicherung einer eingerichtete Bedienoberfläche wäre noch eine sinnvolle Funktionalität. Die gespeicherte Oberfläche könnte später geladen oder als Datei exportiert werden. Interessant wäre auch, das Verhalten der Applikation während Zustandsänderung der Activity zu implementieren, Damit beispielweise die Freigabe von RasPi Ressourcen erfolgt, wenn die Applikation vom System beendet wird.

Bei einem erfolglosen Signal Übertragung wird der Nutzer von der App informiert. Der Control Zustand ändert sich aber trotzdem, obwohl das Signal auf dem RasPi Port nicht ausgegeben wurde. Es wäre dann sinnvoll, dass der Control dann automatisch in den alten Zustand geht.