HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

Entwicklung einer Überwachungsanlage mit Smartphone-Benachrichtigung über Casa Control

Bachelorarbeit

Verfasser: Eike Folkerts
Matrikel-Nummer: 1207948
E-Mail: Eike.Folkerts@stud.hs-hannover.de
Abgabedatum: 07. Juni 2016

Erstprüfer: Prof. Dr.-Ing. Martin Mutz
Zweitprüfer: Prof. Dr.-Ing. Ulrich Lindemann

Danksagung

Ich danke allen Personen, die mich bei der Ausarbeitung dieser Arbeit unterstützt haben. Einen besonderen Dank gebührt meiner Frau Imke Boerma-Folkerts, die in zweifelnden Lagen stets mit Rat zur Seite stand.
Auch Herrn Prof. Dr.-Ing. Mutz danke ich für die Betreuung und Unterstützung bei jeder Frage zu jeder Uhrzeit. Durch die Vorlesungen „Android Apps entwickeln" bei Herrn Prof. Dr.-Ing. Mutz und „Netzwerke" bei Herrn Prof. Dr.-Ing. Lindemann wurde mein Interesse im Bereich Software-Entwicklung für Mobilgeräte geweckt, woraus sich die Idee für diese Arbeit ergeben hat. Des Weiteren danke ich meinem Vater Rik Folkerts für die Korrektur und kritischen Ratschläge bezüglich der Arbeit, meinen Arbeitskollegen Michael Behrendt und Geert Kruse für Hilfestellung in Sachen Programmierung und für die Idee des Namens der App sowie meinem Arbeitgeber Karsten Einnolf für die flexible Bereitstellung von Zeit für diese Arbeit.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit
„Entwicklung einer Überwachungsanlage mit Smartphone-Benachrichtigung über Casa Control"
selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Ort, Datum	Unterschrift

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Stand der Technik

Das eigene Heim mit dem „World Wide Web" zu vernetzen ist beliebter denn je. Ständig gibt es neue Möglichkeiten, reguläre Haushaltsgeräte in multifunktionale Internetgeräte zu verwandeln, sei es auf dem kommerziellen Weg mit gekauften Produkten, oder die Eigenbau-Variante mit kleinen, günstigen elektronischen Bausteinen.

Der Oberbegriff für das Vernetzen von Heimgeräten lautet „Internet der Dinge" (englisch „Internet of Things", künftig „IoT"). IoT spornt Entwickler und Bastler z.B. dazu an, Sensorplatten in den Kühlschrank zu legen, die automatisch ein Produkt bestellen sobald dies aufgebraucht ist, einen Safe mit WLAN und Smartphone-Anbindung auszustatten und die Deckenleuchten zum Super Mario Thema tanzen zu lassen. Solche Produkte erhalten dann das Adjektiv „smart" (deutsch „schlau"), wobei diese Bezeichnung nicht immer zutrifft.

Die Motivation dieser Bachelorarbeit liegt darin, vorhandene elektronische Module durchaus „smart" zu verwenden, um Bereiche des Eigenheims zu überwachen, die nicht durchgehend unter Beobachtung stehen. Smart bedeutet in diesem Fall, dass das entwickelte System weitestgehend automatisiert und ohne viel Wartungsaufwand arbeitet. Am Beispiel einer Überwachungsanlage könnte das eine Gesichtserkennung sein, die nur bei fremden, nicht registrierten Gästen vor der Eingangstür eine Benachrichtigung an den Besitzer schickt. Auch ein Fehlalarm eines einzelnen Bewegungsmelders könnte auf smarte Art ignoriert werden, wenn ein weiterer Bewegungsmelder im selben Raum keine Bewegung feststellen konnte. Bekommt der Bewohner nichts von der Arbeit hinter seinem Smartphone mit und wird nur im Notfall benachrichtigt, arbei-

tet das System smart.

Auf dem Markt gibt es viele unterschiedliche Lösungen für Heim-Überwachung, allerdings sind diese teuer und häufig ist man auf den Hersteller und seine Server angewiesen. In vielen Fällen braucht man ein Nutzerkonto, worüber die Verwaltung aller angebundenen Geräte im Haushalt funktioniert, welches zwei große Nachteile mit sich bringt. Zum einen kann kein Hersteller für die Anonymisierung von streng vertraulichen Daten garantieren, zum anderen sind mit sehr großer Wahrscheinlichkeit auch jegliche Zugänge und Daten erloschen, sobald das Unternehmen nicht mehr existiert. Zukunftsweisend ist dann nur noch die eigene, stets erweiter- und anpassbare Variante.

Ein weiterer, wenn auch untergeordneter Aspekt ist der geringere finanzielle Aufwand. Einen Bewegungsmelder mit WLAN-Modul, der im Handel für 50 € bis 100 € zu erwerben ist, kann für 6,70 €[1] selbst gebbaut werden. Wenn dann noch ein Server als Kontrolleinheit, zwei weitere WLAN-fähige Bewegungsmelder und eine Internet-Kamera hinzukommen, liegt man mit gekauften Produkten schnell bei über 300 €, während der bastelfreudige Entwickler sich noch im zweistelligen Bereich befindet.

Der Unterschied zu Produkten, die es im Handel zu kaufen gibt, ist die Einsparung von Energie und Material. Während bei großen Herstellern Hallen mit Wärme produzierenden Hochleistungsrechnern möglichst kühl gehalten werden müssen, benötigt ein Einplatinencomputer so viel Leistung, wie ein moderner Flachbildschirm im Standby-Modus. Eine Überwachungsanlage mit möglichst wenig Ressourcen zu entwickeln, ist ein Ziel dieser Arbeit. Auch der Energie- und Materialaufwand für die Bindung von Arbeitsplätzen, die für die Wartung einer Anlage notwendig sind, kann entfallen, da die Entwicklung

[1] ESP8266 4,40 €[17, S. 52]; MB102 1,30 €; HC-SR501 1 €

der Überwachungsanlage möglichst wartungsfrei erfolgt. Die häufigste Wartungsarbeit, eine defekte mechanische Festplatte auszutauschen, entfällt, weil keine großen Mengen an Daten gespeichert werden müssen. Ein wartungsarmer Flash-Speicher in Form einer Speicherkarte reicht aus, um ein Aktivitätenprotokoll und Überwachungsbilder abzulegen.

1.2 Struktur der Arbeit

In diesem Kapitel wird der allgemeine Aufbau dieser Arbeit und die groben Inhalte der Kapitel erläutert.

In Abschnitt 2 werden dem Leser die Grundlagen näher gebracht, die zum Verständnis dieser Arbeit nötig sind. Unterteilt in Hardware und Software werden die elektronischen Bausteine und ihre Funktionen sowie die zum Programmieren der Geräte benötigte Software vorgestellt. Wie eine Kommunikation zwischen Server und Client mittels Sockets oder das Senden einer Push-Benachrichtigung an ein Android-Smartphone abläuft, wird im Unterabschnitt 2.2 „Software" erörtert. Welches Gerät im Rahmen dieser Arbeit überhaupt als Server und Client dient und welche Voraussetzungen es erfüllen muss, damit ein zuverlässiges Gesamtsystem entstehen kann, wird in Unterabschnitt 2.1 „Hardware" beantwortet.

Abschnitt 3 stellt das Konzept vor, das zur Umsetzung der in Unterabschnitt 1.1 dargelegten Grundidee führen soll. Anhand einer Grafik werden die einzelnen Bestandteile einer Überwachungsanlage und deren Funktion erläutert, sodass der Leser ein besseres Verständnis für die Umsetzung erhält.

In Abschnitt 4 wird die Realisierung des in Abschnitt 3 vorgestellten Konzeptes aufgeführt. Es sind beispielhafte Ausschnitte der Quellcodes gelistet,

die zum Beispiel bei der Übertragung von Nachrichten von Server zu Client und umgekehrt verwendet werden. Auch der Aufbau einer Android-App sowie die grafische Oberfläche von Server- und Client-Anwendungen ist dokumentiert. Wie ein Raspberry Pi als Server dient, auf welche Art und Weise ein Bewegungsmelder mit WLAN Nachrichten verschicken kann und wie eine Benachrichtigung an ein Android-Smartphone gelangt sind einige der Fragen, die in Abschnitt 4 beantwortet werden.

Zum Abschluss findet in Abschnitt 5 eine Zusammenfassung der Realisierung und ein Blick auf weitere Funktionen und Fähigkeiten, die in Verbindung mit dem umgesetzten System möglich sind, statt.

2 Grundlagen

2.1 Hardware

2.1.1 Raspberry Pi Modell B+

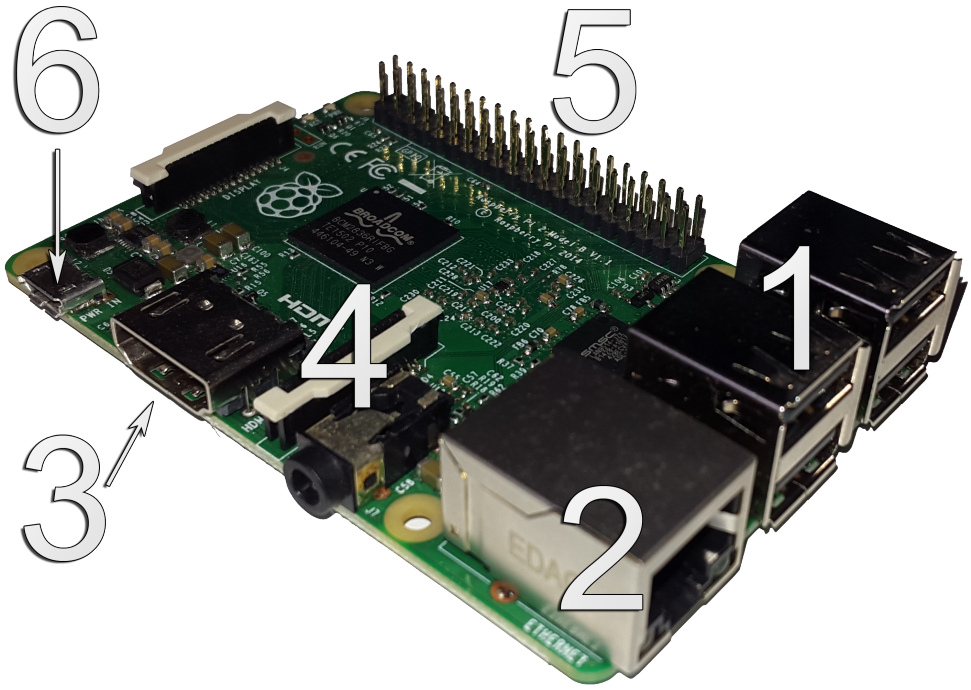


Abbildung 1: Raspberry Pi Modell B+

Bei dem Raspberry Pi (künftig „RasPi") handelt es sich um einen Einplatinencomputer mit Linux-Betriebssystem. Der RasPi (Abbildung 1) ist dank der großen Anzahl von Anschlüssen und den kleinen Abmessungen vielseitig für elektronische Projekte einsetzbar. Neben vier USB-Anschlüssen (Abbildung 1 Punkt 1), Netzwerkanschluss (Abbildung 1 Punkt 2) und HDMI-Ausgang

(Abbildung 1 Punkt 3), sind auf der Hauptplatine für den Anschluss einer Kamera ein „Camera Serial Interface" (kurz „CSI", deutsch: serielle Kamera-Schnittstelle)[8, S. 22] (siehe Abbildung 1 Punkt 4), sowie eine 40-polige Steckerleiste (Abbildung 1 Punkt 5), welche digitale Ein- und Ausgänge, sogenannte „GPIOs" („General Purpose Input and Output"[4, S. 65]), die sich von selbst geschriebenen Programmen ansprechen lassen, angebracht. Als Beispiel lässt sich über die GPIO-Steckerleiste ein Funksender anschließen, dessen Funktion näher im Unterunterabschnitt 2.1.6 erläutert wird. Die Stromversorgung des RasPi erfolgt über einen Micro-USB-Anschluss (Abbildung 1 Punkt 6).

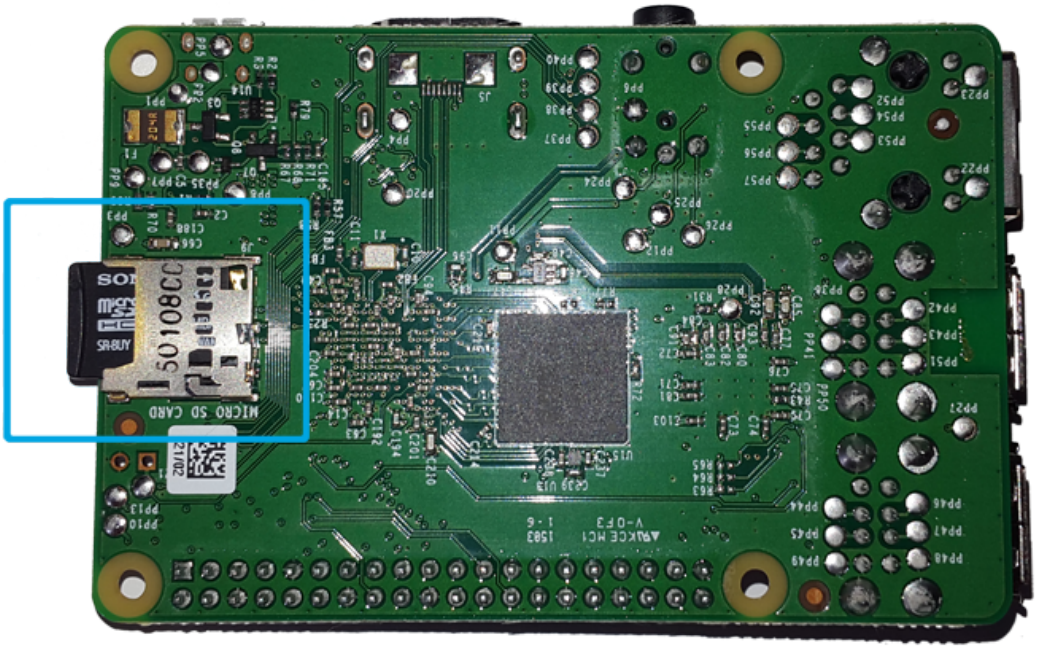


Abbildung 2: Raspberry Pi - Unterseite mit Kartenleser für MicroSD

Der Speicher des RasPi ist eine microSD-Karte, die auch das Betriebssystem

bereitstellt. Der Steckplatz für die MicroSD-Karte befindet sich auf der Unterseite des RasPi (Abbildung 2 Markierung). Es existiert eine große Auswahl an Betriebssystemen, das bekannteste unter ihnen ist „Raspbian"[2], eine vielseitige Linux-Distribution, die eine grafische Oberfläche und eine Grundausstattung an Software, wie zum Beispiel einen Internetbrowser, Dateimanager und eine Java-Laufzeitumgebung (englisch Java Runtime Environment, kurz „JRE"[24]), zur Verfügung stellt. Unter Raspbian lassen sich Anwendungen starten, die in Python, C, C++, Java oder Perl geschrieben sind. Mit Zusatzpaketen lassen sich auch in C# oder PHP geschriebene Programme starten[16]. Der Name „Raspberry Pi" ergibt sich aus der Kombination zwischen dem Trend, Produkte oder Firmen nach Früchten zu benennen und „Pi" als Kurzform für Python, der Programmiersprache, die ursprünglich die einzig unterstützte sein sollte[25].

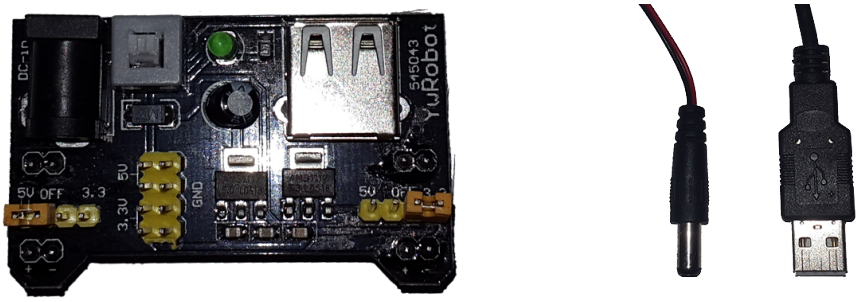
2.1.2 Raspberry Pi-Kamera

Abbildung 3: Raspberry Pi Kamera ohne Infrarotfilter

[2]Link: `https://www.raspbian.org`

Wie in Unterunterabschnitt 2.1.1 erwähnt, stellt der RasPi eine spezielle 15-polige Schnittstelle (CSI, Abbildung 1 Punkt 4) für den Anschluss einer Kamera bereit. In Abbildung 3 ist die Version ohne Infrarotfilter zu sehen. Ein Infrarotfilter dient in der Regel dazu, Farben bei Tageslicht realistischer darstellen zu können, allerdings ist eine Sicht in der Nacht dadurch nicht möglich. Da bei einer Überwachungsanlage das Bild in der Nacht jedoch unabdingbar ist, sollte die sogenannte „NoIR"-Version verwendet werden.
Die Kamera hat eine Auflösung von 5 Megapixel und kann Videos in 1080p (1980x1080 Pixel) aufnehmen. Durch die kleinen Abmessungen von 20 x 25 mm lässt sich die Kamera gut in kleine Gehäuse verstecken, sodass sie unauffällig als Überwachungskamera genutzt werden kann[11, S. 514].

2.1.3 Energieversorgung MB102



(a) MB102 (b) 5,5 x 2,1 mm & USB Type-A

Abbildung 4: Energieversorgungsmodul MB-102

Die im aktuellen Kapitel folgenden zwei Komponenten ESP8266 (Unterunterabschnitt 2.1.4) und HC-SR501 (Unterunterabschnitt 2.1.5) benötigen für

einen reibungslosen Betrieb zwei unterschiedliche Spannungen. Das ESP8266 WLAN-Modul wird mit 3,3 V betrieben, auch die zum ESP8266 führenden Signale, wie zum Beispiel ein vom Bewegungsmelder HC-SR501 erhaltendes Signal für erkannte Bewegung darf 3,3 V nicht überschreiten, um sowohl kurzzeitige Störungen, als auch langfristige Überspannungsschäden zu vermeiden. Der Bewegungsmelder HC-SR501 selbst kann mit Spannungen im Bereich zwischen 4,5 V und 20 V betrieben werden [11, S. 490].
Um 3,3 V und 5 V über eine Quelle zur Verfügung zu stellen, eignet sich das MB-102 (Abbildung 4a). Es kann im Bereich von 6,5V bis 12V sowohl über ein USB-Kabel vom Typ-A, als auch über einen 5,5 mm x 2,1 mm Versorgungsstecker, der beispielsweise an eine 9V Block-Batterie angeschlossen ist (Abbildung 4b), betrieben werden[12]. Das MB-102 hat zwei Spannungsausgänge, die unabhängig voneinander über einen Jumper wahlweise 3,3 V, 5 V oder keine Spannung bereitstellen.

2.1.4 WLAN Modul ESP8266

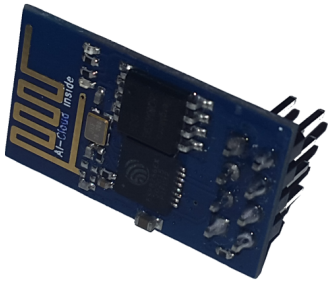


Abbildung 5: WiFi-Modul ESP8266

Der ESP8266 aus Abbildung 5 ist ein vom Hersteller Espressif hergestelltes WLAN-SoC („System-on-a-Chip", deutsch „Ein-Chip-System"[8, S. 20]) mit den Schnittstellen UART und SPI für serielle Kommunikation[13]. Die Abkürzung UART steht für „Universal Asynchronous Receiver Transmitter"[15] und bezeichnet eine weit verbreitete serielle Schnittstelle, die Daten in Wörtern zwischen 5-9 Bit übertragen kann. „Serial Peripheral Interface", kurz SPI[14], ist ein Bussystem mit drei Leitungen für eine synchrone Datenübertragung.
Das Besondere an dem ESP8266 ist, dass Mikrocontroller und WLAN-Empfänger in einem Gerät vereint sind. Als Mikrocontroller besitzt er, wie auch der Raspberry Pi einen Anschluss-PIN für Ein- und Ausgangssignale. Über diesen GPIO kann ein Signal verarbeitet oder ausgegeben werden. So kann zum Beispiel das Signal eines Bewegungsmelders (Unterunterabschnitt 2.1.5) erkannt, und im Programmcode des ESP8266 weiter verarbeitet werden.
Die auf dem ESP8266 vorinstallierte Firmware, also die Software, die in einem Speicherchip eingebunden und verantwortlich für die Grundfunktionalität des Gerätes ist [7], erlaubt es, AT-Befehle auszuführen. Der AT-Befehlssatz, entwickelt von der Firma „Hayes Communications", ermöglicht allgemein die Parametrierung und Konfiguration von Modems [21]. „AT" steht für „attention" (deutsch „Achtung") und ist für das Modem als Vorbereitung auf ankommende Befehle zu verstehen. Über den AT-Befehlssatz lässt sich das WLAN-Modul zum Beispiel mit einem vorhandenen WLAN verbinden, als Client eine Verbindung zu einem TCP-Server („Transmission Control Protocol"[28]) aufbauen oder als TCP Server einsetzen.
Da AT-Befehle „per Hand" über die RX-Leitung des ESP8266 empfangen werden, und sich nicht als Programmcode auf das WLAN-Modul übertragen lassen, eignet sich die Firmware nicht für ein eigenständiges System. Eine Firm-

ware, die Programme abrufen und bei angeschlossener Stromversorgung Programme automatisch starten kann ist notwendig, damit das ESP8266 WLAN-Modul als Verbindung zwischen Bewegungsmelder und Server fungieren kann. Die benötigte Firmware, sowie die Funktionsweise und Programmierung jener, wird in Unterabschnitt 4.2 verdeutlicht.

2.1.5 Bewegungsmelder HC-SR501

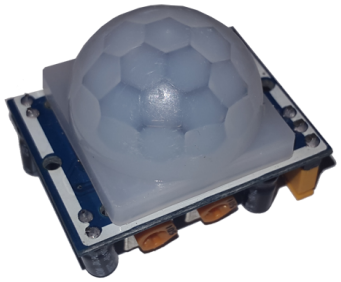
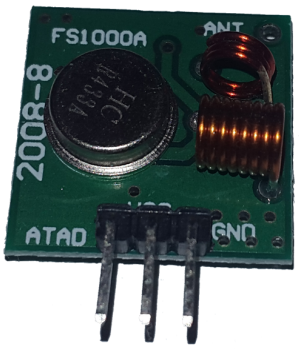


Abbildung 6: Bewegungsmelder HC-SR501

Bei dem HC-SR501 (Abbildung 6) handelt es sich um einen digitalen Bewegungssensor auf der bei Bewegungsmeldern am häufigsten verwendeten Infrarot-Basis. Der Bewegungsmelder, künftig „PIR-Sensor" („passive infrared", deutsch „passives Infrarot") reagiert auf Temperaturänderung im 180°-Umkreis von 3 bis 7 Metern[8, S. 134], wobei sich die Temperaturunterschiede nur auf sich ändernde Signale beziehen, beispielsweise wenn ein Mensch den Bereich der Infraroterkennung betritt [22].
Der HC-SR501 hat drei Anschluss-Pins: „VCC", „GND" und „DATA". VCC

und GND sind mit dem Plus- und Minuspol des Energieversorgungsmoduls MB-102 (Unterunterabschnitt 2.1.3) zu verbinden. Relevant für den Einsatz in Verbindung mit dem ESP8266 WLAN-Modul ist, dass der digitale Ausgangs-Pin DATA ein Spannungsignal von 3.3 V zurück gibt, sobald eine Bewegungserkennung seitens des PIR-Sensors vorliegt [10, S. 123]. Wie im Unterunterabschnitt 2.1.4 bereits erwähnt, müsste ein 5 V-Signal erst auf 3,3 V reduziert werden, um Schäden am ESP8266 zu vermeiden[10, S. 19].

2.1.6 Funksender MX-FS-03V



(a) 433 Mhz Funksender

(b) 433 Mhz Funksteckdose

Abbildung 7: Funksender- und Empfänger

Das MX-FS-03V-Modul (Abbildung 7a), fungiert als Funksender mit einer Frequenz von 433 Mhz. Der Großteil aller im Handel käuflichen Funksteckdosen (Abbildung 7b) empfangen Frequenzen um die 433 Mhz. Am Raspberry Pi angeschlossen, eignet sich das Funksender-Modul MX-FS-03V als Ein- oder

Ausschalter von solchen Steckdosen, wenn diese sich in einem Umkreis von wenigen Metern befinden. Weil Funkwellen elektromagnetische Wellen sind, die durch unterschiedliche Materialien dringen können, eignet sich ein Funksender in Kombination mit einer Funksteckdose auch raumübergreifend. Wie Funksignale gesendet werden können, ist in Unterunterabschnitt 4.1.1 erläutert.

2.2 Software

2.2.1 IntelliJ IDEA

Mit über 20% ist Java die populärste Programmiersprache laut TIOBE-Index[20]. Entsprechend häufig kommt es vor, dass die Software eines elektronischen Gerätes, wie zum Beispiel eines Autoradios oder eines Satelliten-Empfängers, in Java geschrieben ist. Auch viele Betriebssysteme unterstützen das Ausführen von Java-Anwendungen. Drei solcher Betriebssysteme sind Windows auf einem Rechner, Linux auf dem Raspberry Pi und Android auf einem Smartphone. Da alle drei Systeme eine gute Basis einer Überwachungsanlage ergeben, ist es ratsam die benötigte Software mit Java zu entwickeln.
Eine vereinfachte Bedienung der Software einer Überwachungsanlage kann mit Hilfe einer grafischen Benutzeroberfläche realisiert werden. Für Java existiert dafür das Programmiergerüst (engl. „Framework") „JavaFX"[2, S. 203-212], das erstmals 2007 angekündigt wurde. Heutzutage ist JavaFX in der Version 8 Bestandteil des JDK („Java Development Kit"), welches zum Entwickeln und Ausführen von Java-Anwendungen benötigt wird.

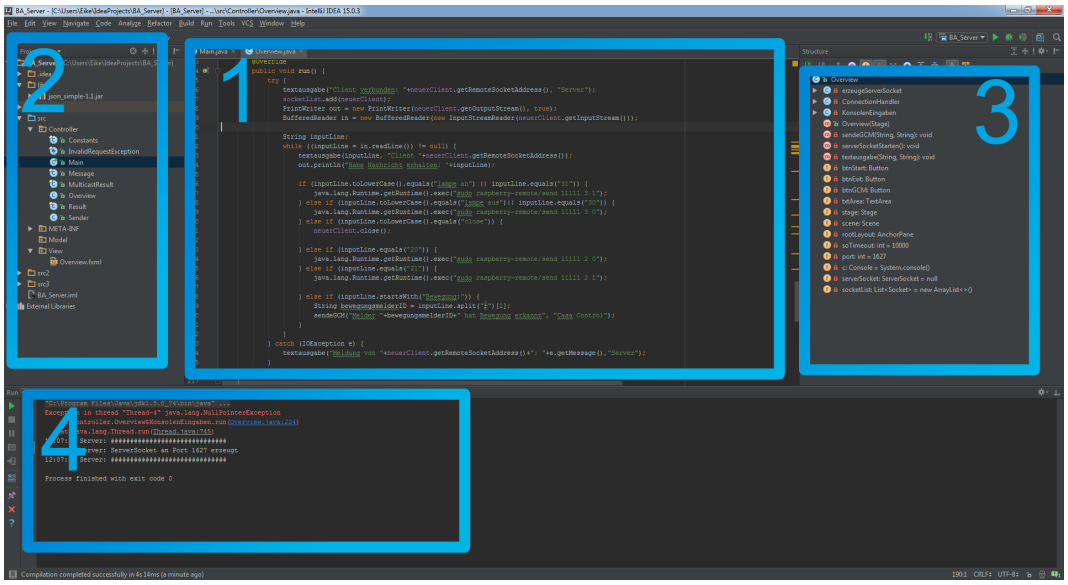


Abbildung 8: Entwicklungsumgebung JetBrains IntelliJ IDEA

Als integrierte Entwicklungsumgebung („IDE" für engl. „integrated development environment") für Java-Software dient „IntelliJ IDEA", entwickelt vom Software-Unternehmen JetBrains[3]. In IntelliJ wird der Java-Quellcode geschrieben (Abbildung 8 Punkt 1) und anschließend als .jar-Datei herausgegeben, die beispielsweise auf dem Raspberry Pi ausgeführt werden kann. Die Klassen- und Dateistruktur ist im „Project-Explorer" (Abbildung 8 Punkt 2) übersichtlich dargestellt, die Struktur der Methoden und Funktionen innerhalb einer Java-Klasse in Abbildung 8 Punkt 3. In der Konsole (Abbildung 8 Punkt 4) werden Textausgaben, die von einem ausgeführten Java Programm kommen, sowie Fehlermeldungen ausgegeben.
IntelliJ IDEA ist die Basis für „Android Studio" (Unterunterabschnitt 2.2.2), der IDE zur Entwicklung von Android-Applikationen (künftig „App"). Die Ver-

[3]Link: `https://www.jetbrains.com/idea/`

wendung beider IDEs ermöglicht eine systemübergreifende Programmierung, so kann eine Java-Anwendung, die mit IntelliJ IDEA für ein Windows-System geschrieben wurde, leicht zu einer App für das Android-Betriebssystem mit Android Studio umgeschrieben werden. Lediglich die grafischen Bedienelemente müssen angepasst werden.

2.2.2 Android Studio

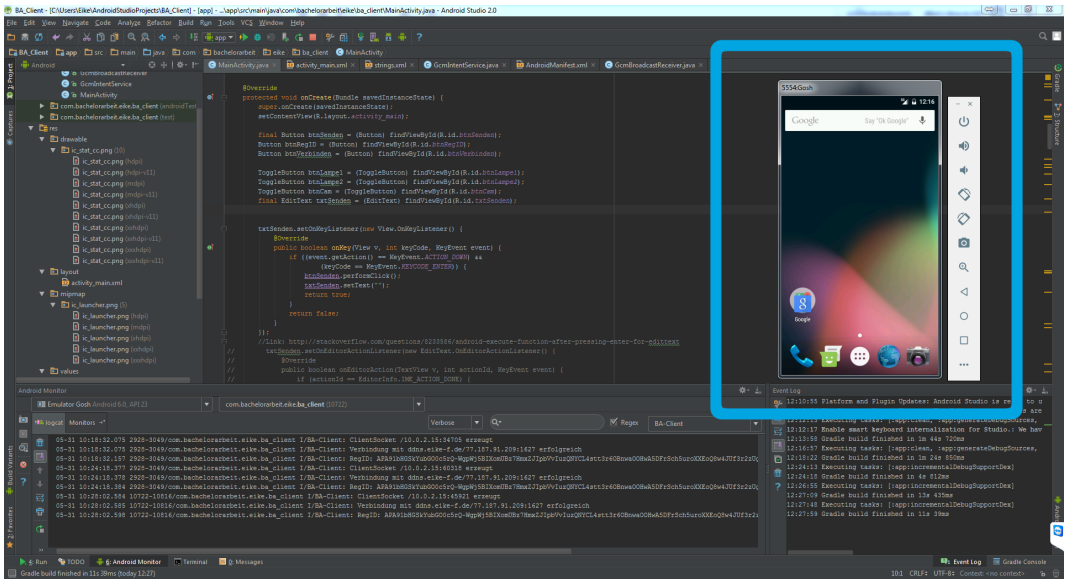


Abbildung 9: Entwicklungsumgebung JetBrains Android Studio

Wie in Unterunterabschnitt 2.2.1 beschrieben, ist die IDE Android Studio der IDE IntelliJ IDEA ähnlich. Der Unterschied bei Android Studio ist, dass alle Werkzeuge, die zur Entwicklung für eine Android-App benötigt werden, bereits integriert sind. So bringt Android Studio eine Emulator-Funktion mit sich, die das Emulieren eines Smartphones auf einem Windows-PC ermöglicht

(Abbildung 9 Markierung). Ein Emulator hat den Vorteil, dass kein Android-Smartphone zum Testen von Apps vorhanden sein muss, zusätzlich kann jede Version von Android heruntergeladen und emuliert werden, wodurch Programmierfehler verhindert werden können, die versionsabhängig sind.

2.2.3 Google Cloud Messaging

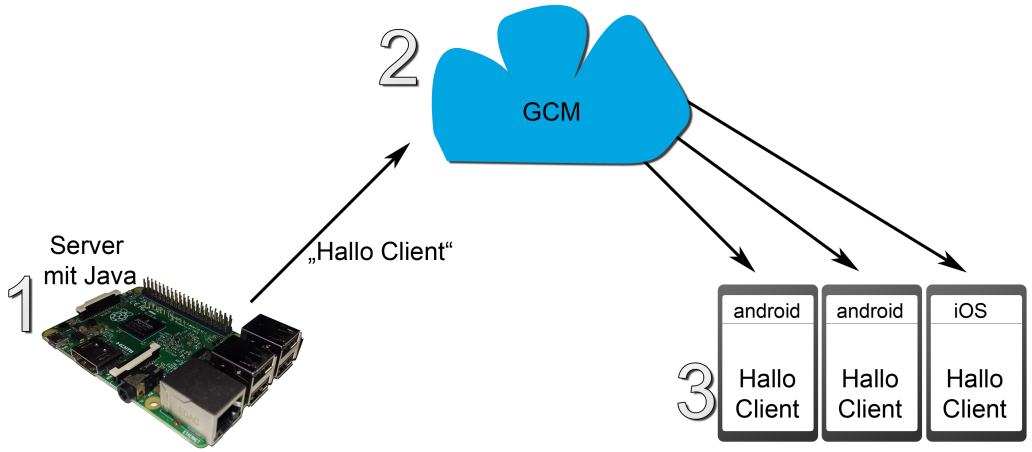


Abbildung 10: Google Cloud Messaging - Nachrichtentransport

Google Cloud Messaging (künftig „GCM”) ist ein Dienst von Google, der eine Kommunikation auf Nachrichtenbasis zwischen einem Server und einem oder mehreren Empfängern (Clients) ermöglicht. GCM (Abbildung 10 Punkt 2) dient dabei als Vermittler von Nachrichten zwischen Server (Abbildung 10 Punkt 1) und Client (Abbildung 10 Punkt 3). Für die Verwendung von GCM anstelle einer einfachen Frage-Antwort-Kommunikation zwischen Client und Server spricht die Einsparung von Datenverkehr und Akkuleistung eines Smartphones. Ein Beispiel ohne GCM ist eine Mail-App auf dem Smartphone, die

alle 5 Minuten den E-Mail Server fragt, ob neue E-Mails zum Abruf bereitstehen. Alle 5 Minuten entsteht somit ein Verbrauch von Datenpaketen und Akkuleistung, auch wenn keine Nachrichten vorhanden sind.
Mit GCM agiert die Mail-App nicht mehr aktiv alle 5 Minuten, sondern verhält sich durchgehend passiv. Es ist nun die Aufgabe des Servers, eine Benachrichtigung zum GCM-Dienst zu senden, sobald eine neue E-Mail eingetroffen ist. Der GCM-Dienst leitet die Benachrichtigung über eine neue Mail weiter an den Client (Mail-App), für den die Benachrichtigung bestimmt ist, woraufhin die Mail-App eine direkte Verbindung zum Mail-Server aufbauen kann, um die E-Mail samt Anhang herunterzuladen. Zeitgleich erscheint auf dem Smartphone, auf dem die Mail-App installiert ist, eine sogenannte Push-Benachrichtigung (eine Benachrichtigung über das Android-Betriebssystem).
Das passive Verhalten einer App mit GCM verbraucht keine Datenpakete, wenn keine Nachrichten seitens GCM-Dienst für das Smartphone vorliegen. Ein weiterer Vorteil ist die Verwendung des GCM-Dienstes mit Apples Betriebssystem iOS. Eine vom Server verschickte GCM-Benachrichtigung muss demnach nicht an das Ziel-Betriebssystem angepasst sein.
Für GCM werden folgende Daten benötigt:

- **Registrierungs-ID**

  Die Registrierungs-ID ist eine eindeutige Identifikation des Android-Smartphones, welches Nachrichten für eine bestimmte Android App erhalten soll. Die Registration-ID erhält das Android-Smartphone über den GCM-Dienst, wenn die bestimmte Android App zum ersten mal gestartet wird.

- **Projektnummer**

  Die Projektnummer (auch Sender-ID) ist eine Nummer für eine App, die von Google angefordert werden kann. Der Entwickler kann über die

Entwickler-Konsole von Google[4] ein Projekt für seine App anlegen, das automatisch mit einer Projektnummer, der Sender-ID versehen wird.

- **API-Schlüssel**

  Wird in Googles Api-Manager[5] („application programming interface", deutsch Programmierschnittstelle) der GCM-Dienst aktiviert, erhält das für die Sender-ID angelegte Projekt einen API-Schlüssel, der für eine eindeutige Identifikation des Sende-berechtigten sorgt. Ein berechtigter Sender von Nachrichten kann ein Server sein, der allen Nutzern (Clients) eine Push-Benachrichtigung schickt.

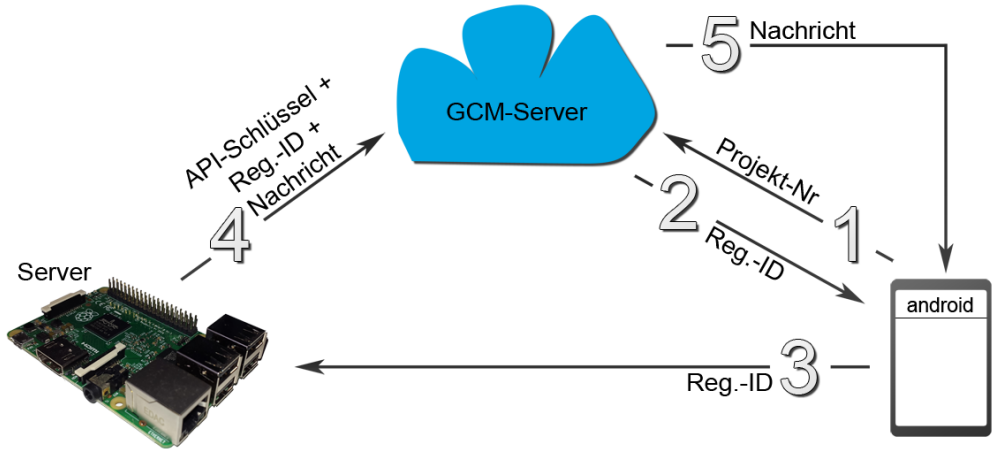


Abbildung 11: Google Cloud Messaging - Registrierung eines Clients

Der Ablauf einer Registrierung bei GCM (Abbildung 11) ist in folgende Schritte unterteilt:

[4]Link: `https://console.developers.google.com/`
[5]Link: `https://console.developers.google.com/apis/`

- **1: Projekt-Nr an GCM**

  Die Android App fordert nach der Installation beim ersten Start eine Registrierungs-ID bei Googles GCM-Server mit einer festen, in der Android App hinterlegten Projektnummer an.

- **2: Reg.-ID an App**

  Der GCM-Server kann die Projektnummer zuordnen und erkennt, dass für dieses Android-Smartphone noch keine Registrierung hinterlegt ist. Der GCM-Server bestätigt dem Android-Smartphone die Registrierung und antwortet auf die Anfrage mit der eindeutigen Registrierungs-ID.

- **3: Reg.-ID an Server**

  Die Aufgabe der Android App ist es nun, den Server über die neue Registrierungs-ID zu informieren, damit dieser sie in die Liste der registrierten Android-Smartphones übernehmen kann.

- **4: API-Schlüssel, Reg.-ID & Nachricht an GCM**

  Wenn der Server eine Push-Benachrichtigung an das bestimmte Android-Smartphone schicken möchte, berechtigt ihn der im Server hinterlegte API-Schlüssel, eine Sender-Anfrage an den GCM-Server zu schicken. Eine bestimmte Nachricht kann dann über die Registrierungs-ID einen bestimmten Empfänger erreichen.

- **5: Nachricht an App**

  Die vom Server erhaltene Kombination aus API-Schlüssel, Registrierungs-ID und Nachricht ermöglichen dem GCM-Server eine eindeutige Zuordnung des Empfängers. Die Android-App bestätigt den Empfang einer GCM-Nachricht mit einer Push-Benachrichtigung[18].

2.2.4 ESPlorer

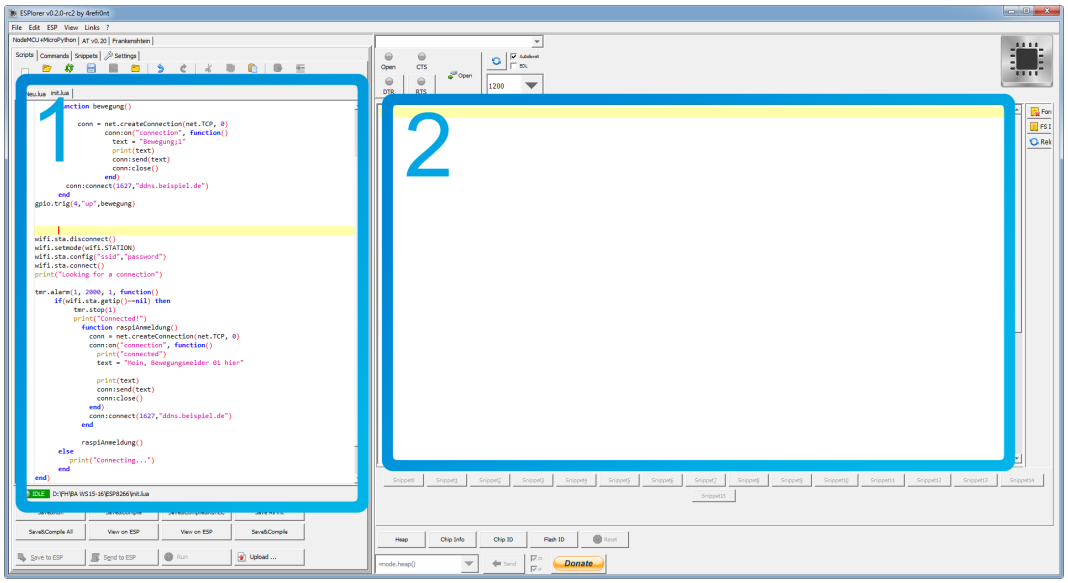


Abbildung 12: Entwicklungsumgebung ESPlorer

ESPlorer[6] ist eine in Java geschriebene IDE, die Quellcode der Programmiersprache „Lua" übersetzen und auf den Speicher eines Mikrocontrollers, zum Beispiel den des ESP8266 übertragen kann[10, S. 50]. Lua ist eine schnelle, kompakte und mächtige Skriptsprache, die unter anderem Verwendung in der Spieleentwicklung (Angry Birds) und in industriellen Anwendungen wie Adobe Lightroom findet [10, S. 32].

Die Bedienoberfläche der IDE ESPlorer ist übersichtlich: Im linken Bereich (Abbildung 12 Punkt 1) ist der Lua Quellcode zu Bearbeiten. Analog zur Konsole bei IntelliJ IDEA (Abbildung 8 Punkt 4) befinden sich die Textausgabe und das Protokoll des Datenverkehrs auf der rechten Seite des ESPlorers

[6] Link: `http://esp8266.ru/esplorer/`

(Abbildung 12 Punkt 2). Wie in Unterunterabschnitt 2.1.4 erläutert, bedarf es einer Firmware, die ein automatisches Laden bestimmter Skripte (Programme) bei einem Neustart erlaubt. Eine geeignete Firmware ist die open-source (deutsch „quelloffen", d.h. der Quellcode ist für jeden einsehbar) Firmware „NodeMCU"[7][10, S. 31]. Über NodeMCU ist es möglich, Lua-Skripte auf einem ESP8266 auszuführen. Die Firmware lässt sich über einen seriellen USB Adapter unter Verwendung der Software „NodeMCU Flasher"[8] auf einen ESP8266 übertragen.

2.2.5 Dynamisches DNS

Damit eine Überwachungsanlage ihren Zweck, das Überwachen aus der Ferne, erfüllt, sollte diese nicht nur im lokalen Netzwerk, sondern auch von überall auf der Welt, bevorzugt von einem Smartphone aus, erreichbar sein. Jedes Gerät, das eine Einbindung in das lokale Netzwerk erhält, bekommt eine interne IP-Adresse zugewiesen, um direkt ansprechbar zu sein. Allgemein erfolgt der Anschluss eines netzwerkfähigen Gerätes und die Zuweisung einer IP-Adresse über einen internetfähigen Router, entweder über ein LAN-Kabel (LAN = „Local Area Network") oder über eine kabellose W-LAN Verbindung („Wireless Local Area Network").
Innerhalb des Netzwerkes kann nun eine Kommunikation stattfinden, da die internen IP-Adressen der Geräte sichtbar und erreichbar sind. Das Problem an einer internen IP-Adresse ist, dass diese nicht aus dem Internet erreichbar ist. Von dem Internetanbieter, der für den Internetzugang nach außen zuständig ist (zum Beispiel Kabel-Deutschland, Vodafone, Telekom) erhält der Internetnutzer eine öffentliche IP-Adresse, über die er auch erreichbar ist. Da die-

[7]Link: `http://nodemcu.com/index_en.html`
[8]Link: `https://github.com/nodemcu/nodemcu-flasher`

se öffentliche IP-Adresse sich jedoch häufig mehrmals am Tag ändert, ist ein Dienst notwendig, der die sich ändernde Adresse unter einer nicht wechselnden, öffentlichen Adresse zugänglich macht. Dieser Dienst nennt sich „Dynamisches DNS" (künftig „DDNS").

DNS ist eine englische Abkürzung und steht für „Domain Name Server". Ein DNS ist dafür zuständig, dass Internetseiten wie zum Beispiel Google stets über die Domain google.de erreichbar sind, eine Domain ist also nichts anderes, als ein Name für eine IP-Adresse. Googles DNS sorgt nun dafür, dass jeder, der www.google.de in die Adressleiste seines Internetbrowsers eingibt, automatisch zu der richtigen, öffentlichen IP-Adresse des deutschen Google-Servers umgeleitet wird. Die öffentliche IP-Adresse der Google-Server ändert sich nur selten, hauptsächlich wenn ein Standortwechsel oder ein Austausch der Server selbst stattfindet [11, Vgl. S.170f]. Mit einer statischen, öffentlichen IP-Adresse sparen Google und andere Internetpräsenzen sich den Aufwand, die Domain google.de ständig einer anderen IP-Adresse zuzuordnen.

Auch private Internetnutzer können eine statische, also feste IP-Adressen bei ihren Internetanbietern erhalten, weil eine Umstellung jedoch Aufwand und Kosten mit sich bringt ist der Einsatz eines dynamischen DNS Dienstes sinnvoll. Dynamisches DNS funktioniert wie folgt:

Zunächst ist eine Registrierung bei einem DynDNS-Anbieter wie z.B. „Dyn"[9], „Selfhost"[10] oder „Strato"[11] erforderlich. Für eine Veranschaulichung ist künftig anzunehmen, dass die Internet-Domain beispiel.de registriert, und Vollzugriff auf alle Dienste bei Strato gewährt wurde. Als nächstes ist bei Strato anzugeben, auf welche Domain der DynDNS-Dienst angewandt werden soll. Da sich

[9] Link: `www.dyn.com`
[10] Link: `www.selfhost.de/cgi-bin/selfhost`
[11] Link: `https://www.strato.de/`

bei Strato kostenlos „Subdomains", also unterteilte Domains, erstellen lassen, definiert man für die Subdomain ddns.beispiel.de einen DynDNS-Dienst. Darauf folgt die Verknüpfung des Routers im Heimnetz mit dem DynDNS-Dienst. Zu den wenigen Routern, die Dynamisches DNS unterstützen, gehören die meisten FritzBox-Router der Firma AVM[12]. In der Routerkonfiguration wird definiert, über welchen DynDNS-Dienst die FritzBox erreichbar sein soll. Sobald der Strato-Account mit Domain, Benutzername und Passwort in der Routerkonfiguration als DynDNS-Dienst eingerichtet ist, ist die Verbindung zwischen FritzBox im Heimnetz und der öffentlichen Subdomain ddns.beispiel.de hergestellt. Die FritzBox erkennt nun, wenn sich die öffentliche IP-Adresse des Routers ändert und teilt dies dem DynDNS-Dienst mit, sodass ddns.beispiel.de immer auf die aktuelle, öffentliche IP-Adresse des Routers umgeleitet wird.
Bis hier ist lediglich der Router über das Internet erreichbar. Um nun die weiteren, am Router verbundenen Geräte zu erreichen, ist ein weiterer Schritt notwendig, der im folgenden Unterunterabschnitt 2.2.6 behandelt wird.

2.2.6 Portweiterleitung

Um eine Server-Anwendung direkt über einen vordefinierten Port zu erreichen, muss dieser aus dem Internet über den Router an den RasPi weitergeleitet werden. Eine IP-Adresse in Verbindung mit dem Port ist die direkte Zuordnung, sodass die Server-Anwendung unmittelbar angesprochen werden kann.
Ein konkretes Beispiel lässt sich anhand folgender Abbildung 13 erläutern:

[12]Link: `https://avm.de/produkte/fritzbox/`

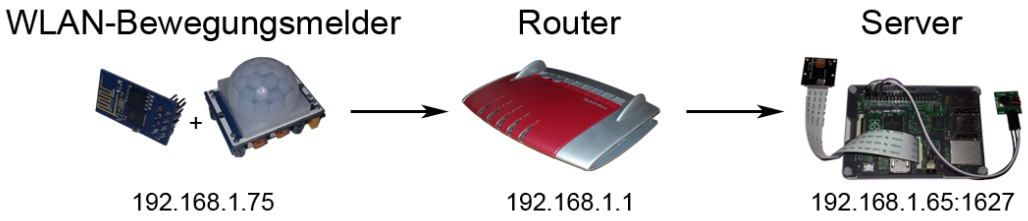


Abbildung 13: Server-Client Kommunikation im Heimnetz

Angenommen, der WLAN-Bewegungsmelder (Client), bestehend aus ESP8266 und PIR-Sensor, und der Raspberry Pi (Server) befinden sich im selben lokalen Heimnetzwerk; der ESP8266 erhält ein Signal des PIR-Sensors HC-SR501 und möchte Server-Anwendung auf dem RasPi darüber informieren. Solange dies im lokalen Netzwerk passiert, kann der ESP8266 so programmiert werden, dass eine Textnachricht an die lokale IP-Adresse (z.B. 192.168.1.65 an den TCP Port 1627) des Servers geschickt wird. Eine Portweiterleitung in der Routerkonfiguration ist nicht notwendig, da sich alle am Router angeschlossenen Geräte untereinander erkennen und ansprechen können.
Befindet sich der ESP8266 mit Bewegungsmelder allerdings in einem anderen Netzwerk an einem anderen Ort, muss das WLAN-Modul den Umweg über das Internet nehmen, um den Server auf dem RasPi im Heimnetzwerk zu erreichen (Abbildung 14).

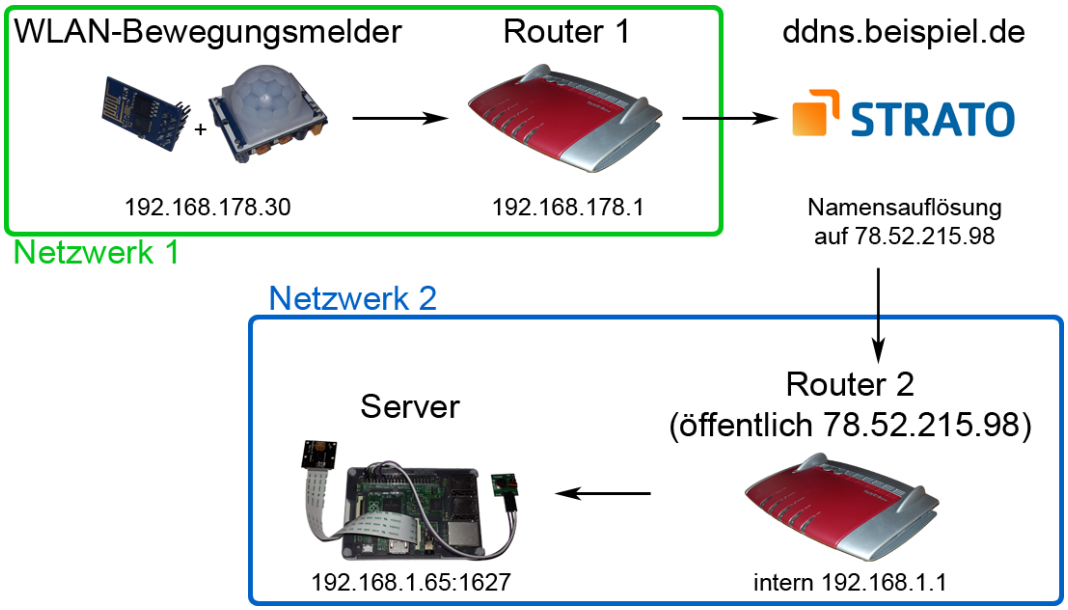


Abbildung 14: Server-Client Kommunikation über DynDNS

Der ESP8266 wird so programmiert, dass er eine Nachricht an die Domain ddns.beispiel.de an Port 1627 schickt. Da ddns.beispiel.de über DynDNS an den Router, und der Port 1627 laut Routerkonfiguration an die IP-Adresse des RasPi weitergeleitet wird, erreicht der Bewegungsmelder den Java-Server aus der Ferne.

Die Art der ortsunabhängigen Kommunikation mit dem Server bietet den Vorteil, mehrere Objekte gleichzeitig zu überwachen und im Falle eines Einbruchs die Beschädigung des Servers samt Überwachungsdaten, wie Protokolle oder Bilder einer Webcam, zu verhindern.

2.2.7 SSH

Durch die Protokollierung von Verbindungen und Bewegungen auf Server-Seite bietet es sich an, Einsicht aus der Ferne in die Textausgaben des Java-Servers

zu haben. Mit Hilfe des Netzwerkprotokolls „SSH" (engl. „Secure Shell")[1, S. 97f] ist es möglich, auf die kommandozeilenbasierte Benutzung, die Shell, des RasPi zuzugreifen. Die Shell auf dem RasPi mit Linux-Distribution Raspian heißt „Terminal" und ist vergleichbar mit der Windows Eingabeaufforderung, mit der ebenfalls Kommandos in Textform eingegeben werden können, um beispielsweise die Konfigurationen des Systems vorzunehmen oder Programme zu starten.

Damit ein anderes System, wie zum Beispiel ein Windows-Rechner oder Android-Smartphone, über SSH auf den RasPi Server zugreifen kann, ist zunächst der SSH-Dienst zu aktivieren.

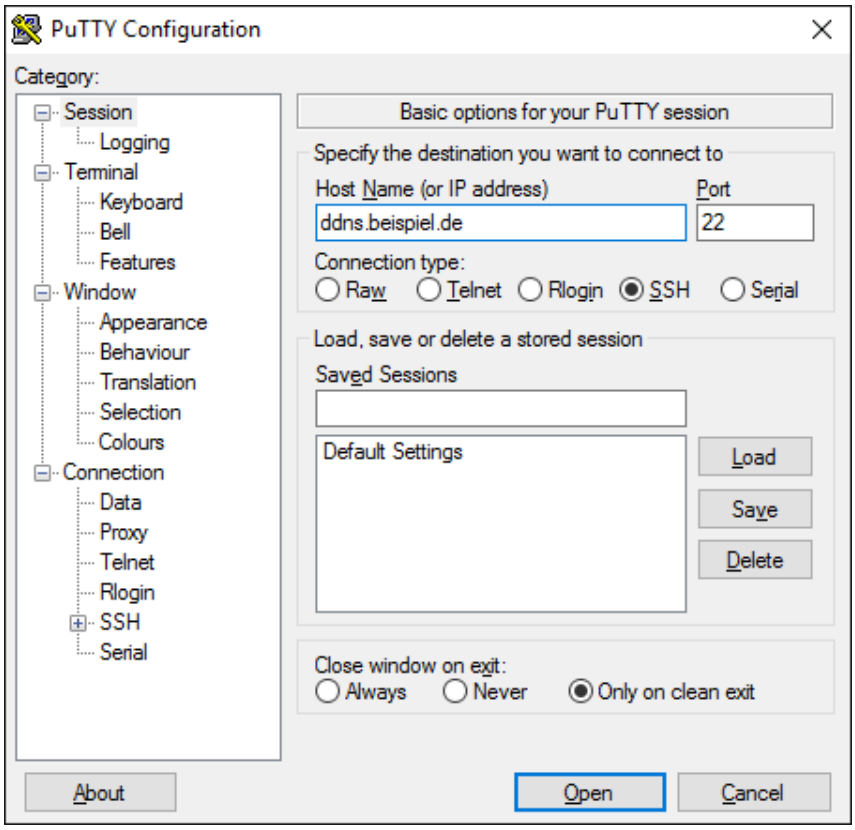


Abbildung 15: PuTTY Konfiguration

Nun bedarf es einer SSH-Client Software für Windows wie dem kostenlosen Programm „PuTTY"(Abbildung 15). Befinden sich Windows-Rechner und Server im gleichen Heimnetzwerk, kann die lokale IP-Adresse des Servers verwendet werden. Findet der SSH-Zugriff aus einem anderen Netzwerk statt, ist an Stelle der lokalen IP-Adresse die aktuelle öffentliche, oder wie in Unterunterabschnitt 2.2.5 beschrieben, die DynDNS-Adresse des Server-Netzwerks anzugeben. Im letzteren Fall muss zusätzlich noch eine Portweiterleitung (Unterunterabschnitt 2.2.6) im Router des Server-Netzwerkes erfolgen. Der Port 22 ist ein speziell für SSH vorbehaltener Port und sollte sowohl bei interner

als auch externer Verbindung beibehalten werden[9, S. 110]. Das Starten einer Java-Anwendung über eine SSH-Verbindung mit dem Befehl[2, S. 34]

```
1 java -jar \Pfad\zur\Serverdatei.jar
```

birgt folgendes Problem: Sobald der SSH-Client geschlossen wird, schließen sich damit einhergehend auch alle Programme, die über die SSH-Verbindung gestartet wurden. Im Idealfall sollte die Java-Anwendung, einmal gestartet, ohne weitere Aufsicht im Hintergrund arbeiten. Möchte man zu einem späteren Zeitpunkt auf die geöffnete Anwendung über SSH zugreifen, ist dies ohne Weiteres nicht möglich. Die Zusatzsoftware „Screen"[13] schafft Abhilfe für dieses Problem. Mit ihr lässt sich auf dem RasPi eine virtuelle Umgebung erzeugen, auf die mit einem Befehl über das Terminal immer wieder zugegriffen werden kann. In dieser Umgebung, dem Screen, wird dann die Anwendung gestartet, sodass diese Instanz des Programms bei jeder SSH-Verbindung abgerufen werden kann.

[13]Link: https://wiki.ubuntuusers.de/Screen/

3 Konzept

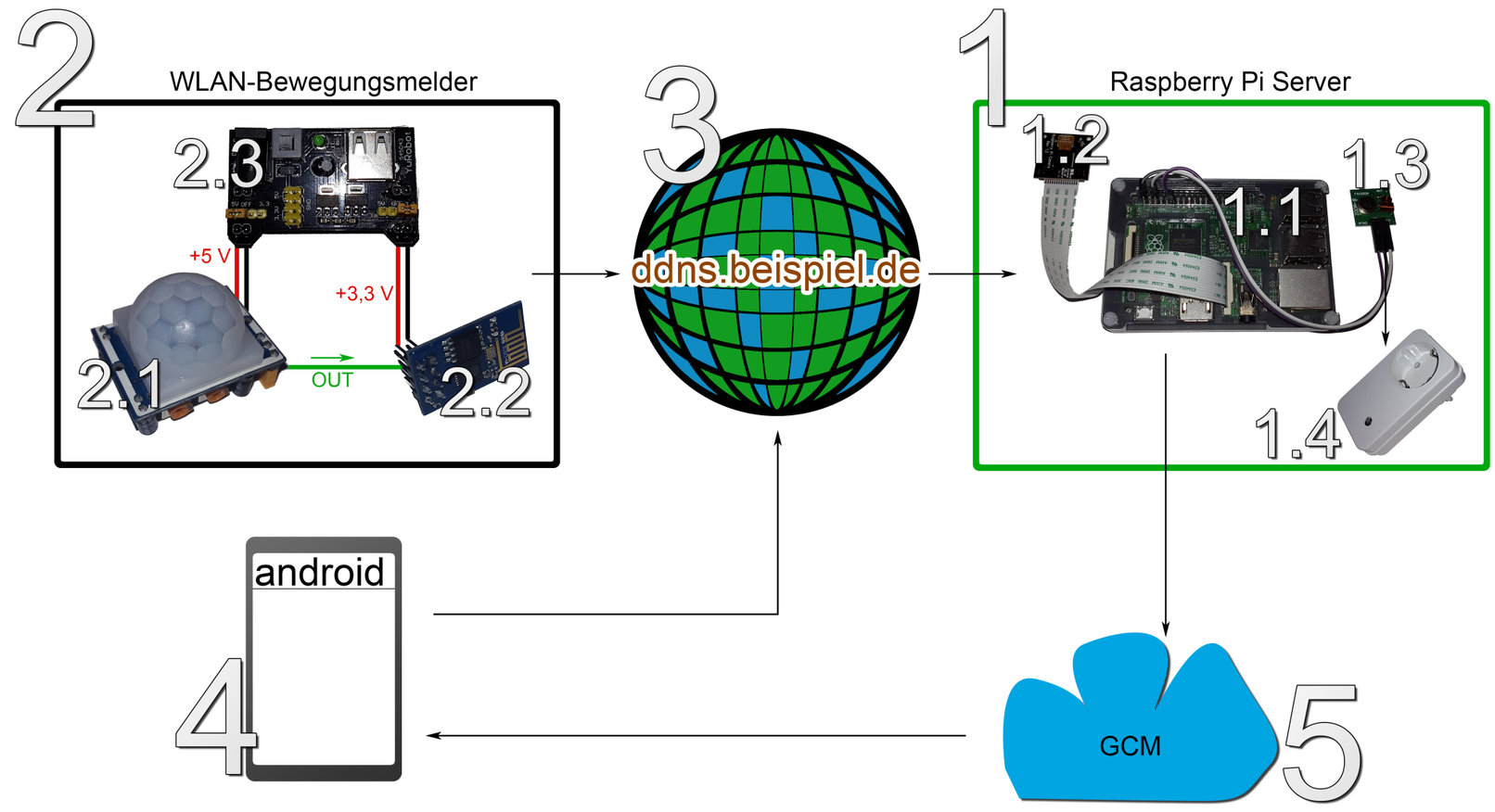


Abbildung 16: Das Konzept

3.1 Raspberry Pi als Server

Der vielseitig nutzbare RasPi(Abbildung 16 Punkt 1.1) dient im Rahmen dieser Arbeit als Server der Überwachungsanlage und bildet somit die zentrale Basisstation(Abbildung 16 Punkt 1).
Als Server zählt es zu seinen Aufgaben, die Kommunikation mit anderen Geräten zu kontrollieren und zu steuern. Erhält der Server eine Textnachricht von einem der verbundenen Geräte, so hat dieser die Nachricht zu analysieren und eine Folgeaktion durchzuführen. Zusätzlich ist der Server in der Lage, Geräte direkt anzusprechen und eine Kommunikation zu starten. Folgende Aufgaben soll der Server erfüllen:

- Textnachrichten empfangen
- Textnachrichten senden
- Kamerabild bereitstellen
- Steckdosen an- und ausschalten
- Benachrichtigungen an Android-Smartphone senden

Bis auf die Bereitstellung eines Kamerabildes lassen sich alle Aufgaben mit Java realisieren. Voraussetzung für ein erfolgreiches Ausführen eines Java Programms ist eine vom Betriebssystem unterstützte Java-Laufzeitumgebung (JRE). Bei allen Linux-Distributionen ist eine JRE bereits vorinstalliert, weshalb es keiner weiteren Installation auf dem RasPi bedarf. Das Kamerabild liefert die in Unterunterabschnitt 2.1.2 vorgestellte Raspberry Pi Kamera (Abbildung 16 Punkt 1.2). In Verbindung mit einem Infrarot-Licht-Scheinwerfer, der an eine Funk-Steckdose (Abbildung 16 Punkt 1.4) angeschlossen werden kann, ist mit der NoIR-Kamera eine Nachtsicht für eine große Fläche, beispielsweise eines

Wohnungseingangsbereiches möglich. Die Funk-Steckdose hat einen integrierten Funk-Empfänger und empfängt Signale mit einer Frequenz von 433 Mhz. In Unterunterabschnitt 2.1.6 bereits erwähnt, ist der Funksender (Abbildung 16 Punkt 1.3), der über die GPIO-Steckerleiste des RasPi angeschlossen werden kann, fähig, Funksignale auf einer Frequenz von 433 Mhz zu senden.
Befindet sich eine Funk-Steckdose in unmittelbarer Nähe zum RasPi, kann diese mit dem Funksender an- oder ausgeschaltet werden. Eine weitere Möglichkeit wäre es, einen akustischen Alarmgeber an eine Steckdose anzuschließen und diesen zu aktivieren, sobald ein unbefugter Eindringling das zu überwachende Gebiet betreten hat. Über die Kamera lässt sich prüfen, ob es sich um einen Fehlalarm handeln könnte.
Damit solche Szenarien funktionieren, muss die zu programmierende Anwendung in der Lage sein, Textnachrichten von Bewegungsmeldern zu bekommen und nach Verarbeitung der Nachricht mit dem Funksender zu kommunizieren. Dies setzt jedoch voraus, dass sich der Bewegungsmelder mit dem RasPi-Server verbinden und auch Nachrichten verschicken kann.

3.2 ESP8266 als Bewegungsmelder

Mit dem in Unterunterabschnitt 2.1.5 vorgestellten HC-SR501 PIR-Sensor ist es möglich, Bewegung zu erkennen. Damit der RasPi-Server die Bewegung verarbeiten kann, muss der PIR-Sensor (Abbildung 16 Punkt 2.1) mit dem Server über das Internet kommunizieren können. Die Kommunikation und Verarbeitung eines Signals bei erkannter Bewegung übernimmt dabei das ESP8266 WLAN-Modul (Abbildung 16 Punkt 2.2). Beide Geräte werden über das MB102-Modul (Abbildung 16 Punkt 2.3) mit Spannung versorgt.
Auf dem ESP8266 muss mittels der Programmiersprache Lua ein Skript da-

für sorgen, dass eine Verbindung zum Server aufgebaut werden kann. Wenn die Verbindung steht, reicht es aus, eine einfache Textnachricht, wie „Bewegung erkannt" zu übermitteln, sobald ein Ausgangssignal des PIR-Sensors den digitalen Eingang des ESP8266 erreicht hat.

3.3 DDNS

Für den Fall, dass WLAN-Bewegungsmelder aus Unterabschnitt 3.2 und Raspberry Pi Server aus Unterabschnitt 3.1 sich in unterschiedlichen, räumlich getrennten Netzwerken befinden, soll trotzdem eine Kommunikation möglich sein. Die dezentrale Verfügung ermöglicht ein unabhängiges Netzwerk aus mehreren Bewegungsmeldern, Kameras oder Steckdosen. So ist es möglich, mehrere Grundstücke oder Gebäude mit nur einem Server zu überwachen. Um dies zu erzielen, muss sich der Server in einem Netzwerk befinden, dessen Router für dynamisches DNS konfigurierbar ist (DDNS siehe Unterunterabschnitt 2.2.5). Der DDNS-Dienst (Abbildung 16 Punkt 3) ermöglicht dem Server, über das Internet von der ganzen Welt aus erreichbar zu sein, um mit anderen Geräten wie einem Smartphone oder Bewegungsmelder zu kommunizieren. Der Server ist damit über eine bestimmte Adresse, im Beispiel als ddns.beispiel.de angegeben, mit definierten Port erreichbar.

3.4 Android App als Client

Eine Android-App auf einem Smartphone (Abbildung 16 Punkt 4) hat folgende Anforderungen zu erfüllen:

Aktiv

- Befehle an Server schicken
- Live-Stream der Webcam anzeigen

Passiv

- Push-Benachrichtigungen des Servers darstellen

Im aktiven Zustand, also beim Öffnen der App durch einen Benutzer wird über die DDNS-Domain ddns.beispiel.de (Abbildung 16 Punkt 3) automatisch eine Verbindung zum Server aufgebaut. Die App kann Befehle wie „Steckdose;1;an" oder „kamera;deaktivieren" in Form von Textnachrichten senden, welche vom Server erkannt und mit weiteren Aktionen verknüpft werden können. Sobald der Server eine Nachricht eines Clients erhalten hat, bekommt der Client eine Bestätigung vom Server über den Empfang der Nachricht, die Kommunikation läuft demnach bidirektional.
Über ein in der App integriertes Fenster kann das Bild der Kamera des RasPi angezeigt werden, das über eine Internetseite zur Verfügung gestellt wird. Parallel zur Betrachtung des Bildes der Kamera kann der Funksender des Servers angesprochen werden, sodass zum Beispiel ein Scheinwerfer, der an eine Funk-Steckdose angeschlossen ist, angeschaltet werden kann. Die Reaktion eines Eindringlings auf ein plötzliches Ausleuchten am überwachenden Ort kann somit direkt betrachtet und zu Beweiszwecken auch aufgezeichnet werden.

Ist die App nicht geöffnet, befindet sie sich im passiven Zustand. Bei Erkennung einer Bewegung am überwachten Ort ist eine zeitnahe Benachrichtigung notwendig, um kurzfristiges Handeln im Notfall zu ermöglichen. Die passive Benachrichtigung hat den Vorteil, dass sich der Nutzer nicht aktiv alle 15 Minuten von dem Zustand seines überwachten Ortes überzeugen muss. Ein möglicher Einbruch kann direkt gemeldet und aufgezeichnet werden. Das Versenden der Push-Benachrichtigung durch den Server an das Smartphone erfolgt über Googles Dienst „Google Cloud Messaging" (Unterunterabschnitt 2.2.3), in Abbildung 16 als Punkt 5 bezeichnet.

Der Ablauf für eine Benachrichtigung auf dem Smartphone bei Bewegung verläuft wie folgt:

1. Der PIR-Sensor (Abbildung 16 Punkt 2.1) registriert Bewegung und gibt ein 3,3 V Signal über den „OUT"-Pin an das ESP8266 Wifi-Modul (Abbildung 16 Punkt 2.2)

2. Der ESP8266 registriert das Signal am digitalen Eingang und versucht eine Verbindung zum Server (Abbildung 16 Punkt 1) über die DDNS-Domain (Abbildung 16 Punkt 3) aufzubauen. Nach erfolgreicher Verbindung schickt der ESP8266 eine Textnachricht mit der Identifikation des Bewegungsmelders an den Server.

3. Die Server-Software, die auf dem Raspberry Pi läuft, erkennt den Eingang einer Textnachricht und verarbeitet diese. Da es sich um eine Bewegung handelt, wird eine Nachricht an den GCM-Dienst (Abbildung 16 Punkt 5) geschickt, dass Bewegung an einem bestimmten Bewegungsmelder stattgefunden hat.

4. Der GCM-Dienst prüft, welche Smartphones berechtigt sind, diese Nachricht zu empfangen und schickt die Push-Benachrichtigung an alle Android-Geräte, die mit der Server-Software verknüpft sind.

5. Die Android-App (Abbildung 16 Punkt 4) erkennt den Eingang einer GCM-Nachricht und verarbeitet diese in Form einer Benachrichtigung mit Vibration und Klingelton auf dem Smartphone.

6. Der Nutzer hat die Möglichkeit, die App zu starten und sich die Situation vor Ort über das Kamerabild anzusehen.

4 Realisierung

4.1 Casa Control - Server

Das Software-Konzept trägt den Namen „Casa Control". Casa ist spanisch und bedeutet „Haus", das englische „control" ist mit Kontrolle zu übersetzen. Die Server-Anwendung „Casa Control Server" (künftig „CC-Server") ist die Software, die auf der Server-Hardware, also dem Raspberry Pi läuft. Als Betriebssystem ist die vom Hersteller empfohlene Linux-Distribution Raspbian installiert. Der CC-Server ist in Java programmiert und bietet eine mit JavaFX generierte grafische Benutzeroberfläche

4.1.1 Java-Anwendung

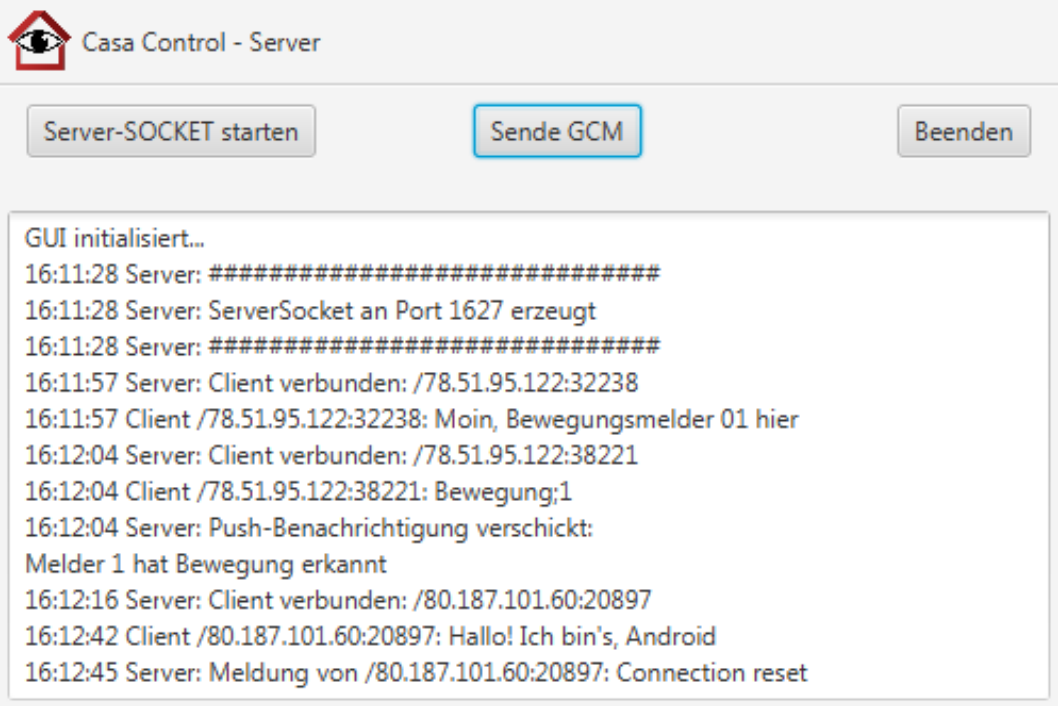


Abbildung 17: Casa Control Server auf dem Raspberry Pi

Weil der RasPi Java-Anwendungen und JavaFX-Oberflächen starten kann, ist auf der Server-Hardware keine weitere Einrichtung erforderlich. In Abbildung 17 ist im unteren Teil des Fensters das Protokoll zu sehen. Hier findet eine Dokumentation von hergestellten, sowie getrennten Verbindungen, ankommenden und ausgehenden Textnachrichten und ausgeführten Befehlen statt.
Beim Start des CC-Servers wird automatisch ein „ServerSocket" erzeugt, der alternativ auch über die Schaltfläche „Server-SOCKET starten" neu gestartet werden kann. Die Schaltfläche „Sende GCM" schickt testweise eine Benachrichtigung an alle registrierten Android-Geräte. Ein Socket ist die Standard-

Schnittstelle für TCP/IP („Transmission Control Protocol / Internet Protocol"[29]), dem Protokoll, das als Grundlage des Internets dient. Ein ServerSocket ist mit einem USB-Anschluss vergleichbar, der auf den Anschluss eines USB-Gerätes wartet. Wie der USB-Anschluss auf ein Gerät, wartet der ServerSocket auf Klienten, die Kommunikation mit dem ihm aufnehmen möchten (USB-fähige Geräte), auch ClientSockets oder Sockets genannt.
Die Initialisierung eines ServerSockets sieht im Java-Quellcode wie folgt aus:

Listing 1: ServerSocket initialisieren

```
1 int port = 1627;
2 ServerSocket serverSocket = new ServerSocket(port);
```

Mit der Erzeugung eines ServerSocket-Objekts ist ein ServerSocket an Port 1627 gestartet worden. Der ServerSocket wartet damit aktiv auf eingehende Sockets von anderen Geräten, die an Port 1627 gerichtet sind. Sobald eine Verbindung zwischen Socket und ServerSocket hergestellt wurde, ist der ServerSocket blockiert um Datenpakete vom Socket zu empfangen oder sie zum Socket zu schicken. Weil es vorkommen kann, dass mehrere externe Geräte über Sockets auf den ServerSocket zugreifen möchten, sollte ermöglicht werden, mehrere ServerSockets parallel laufen zu haben. Dies geschieht über das Erzeugen eines neuen „Threads":

Listing 2: Mehrere ServerSockets parallel mit Threads

```
1 while (true) {
2         Socket clientSocket = serverSocket.accept();
3         Runnable socketManager = new SocketManager(
             clientSocket);
4         new Thread(socketManager).start();
5 }
```

Ein Thread ist ein nebenläufiger Prozess, mit dessen Hilfe eine parallele Programm-Verarbeitung stattfinden kann[8, S. 263]. In Listing 2 Zeile 2 wird mit der accept()-Methode jeder ankommende Socket akzeptiert, der dann dem Runnable „SocketManager" übergeben wird. Der SocketManager wird als Thread gestartet, dessen einzige Aufgabe es ist, empfangende Nachrichten des übergebenden Sockets zu verarbeiten (Listing 3). Damit sofort nach dem Starten eines SocketManager-Threads auf einen neuen, eingehenden Socket gewartet werden kann, sind die Zeilen 2-4 in Listing 2 von einer nicht-endenden while-Schleife ummantelt.

Listing 3: Nachrichtenverarbeitung in SocketManager

```
1  PrintWriter out = new PrintWriter(neuerClient
2                  .getOutputStream(), true);
3  BufferedReader in = new BufferedReader(
4                  new InputStreamReader(neuerClient
5                  .getInputStream()))
6  String inputLine;
7  while ((inputLine = in.readLine()) != null) {
8          textausgabe(inputLine, "Client␣"
9          +neuerClient.getRemoteSocketAddress());
10         out.println("Habe␣Nachricht␣erhalten:␣"+inputLine);
11         if (inputLine.toLowerCase().equals("lampe␣an"){
12                 java.lang.Runtime.getRuntime()
13                 .exec("sudo␣raspberry-remote/send␣11111␣3␣1")
                      ;
14         } else if (inputLine.startsWith("Bewegung;")) {
15                 String bewegungsmelderID = inputLine.split(";
                      ")[1];
16                 sendeGCM("Melder␣"+bewegungsmelderID
17                 +"hat␣Bewegung␣erkannt", "Casa␣Control");
```

```
18              }
19  }
```

Die Klassen PrintWriter (Listing 3 Zeile 1) und BufferedReader (Listing 3 Zeile 3) stellen den Nachrichten-Eingangs-, bzw. den Nachrichten-Ausgangsstrom dar. Über diese werden Nachrichten vom verbundenen Socket entgegengenommen (BufferedReader „in") und herausgegeben (Printwriter „out").
In Zeile 12-13 ist die Aktion zu betrachten, die durchgeführt wird, falls eine eingehende Nachricht eines verbundenen ClientSockets dem Text „lampe an" entspricht (siehe Bedingung in Zeile 11). Ist dies der Fall, wird über das Terminal auf dem RasPi der Befehl zum Aussenden eines Funksignals gegeben. Hierfür ist auf dem RasPi eine Installation der Software „Raspberry-Remote"[14] durchzuführen. Über den send-Befehl von Raspberry-Remote kann eine spezifische Steckdose ein- oder ausgeschaltet werden. Die ersten fünf Ziffern nach dem Befehl „send" stehen für den Systemcode der Steckdose, gefolgt von der Steckdosen-Nummer („Unitcode") und einer 1 für an und 0 für aus. Systemcode und Unitcode lassen sich in jeder Funk-Steckdose definieren und ermöglichen ein eindeutiges Schalten einer bestimmten Steckdose.
Damit der RasPi den Funksender über die richtigen GPIO-Pins betreibt, ist zusätzlich die Programmbibliothek „wiringPi"[15] vonnöten[3].

4.1.2 GCM - serverseitig

Das Verschicken einer Push-Benachrichtigung an einen Android-Client ist anhand der Methode „sendeGCM()" erläutert:

Listing 4: Push-Benachrichtigung an Android-App schicken

[14]Link: `github.com/xkonni/raspberry-remote`
[15]Link: `http://wiringpi.com/`

```
1  private void sendeGCM(String msg, String title) {
2     Sender sender = new Sender("API-SCHLUESSEL");
3     Message message = new Message.Builder()
4        .addData("message", msg)
5        .addData("title", title)
6        .build();
7     try {
8        Result result = sender.send(message,
9              "REGISTRIERUNGS-ID", 5);
10       textausgabe("Push-Benachrichtigung verschickt:\n"
11          +msg,"Server");
12    } catch (IOException e) {
13       e.printStackTrace();
14    }
15 }
```

Der für das Senden benötigte, in Unterunterabschnitt 2.2.3 erläuterte API-Schlüssel wird dem „Sender"-Objekt in Listing 4 Zeile 2 übergeben. In den Zeilen 3-6 wird ein „Message"-Objekt verwendet, um eine Nachricht mit mehreren Elementen zu generieren. Das Element „title" (Zeile 5) stellt beim Empfang in der Android-Benachrichtigung die Überschrift, das Element „message" die eigentliche Nachricht dar. Über das „Result"-Objekt in Zeilen 8 und 9 kann die Nachricht zusammen mit der Registrierungs-ID des empfangenden Android-Smartphones über die send-Methode des Sender-Objekts zum GCM-Server übertragen werden.

4.1.3 Server-Kamera

Der Zugriff auf das Bild der Raspberry Pi Kamera über das Internet ist mit der Zusatzsoftware „RPi-Cam-Web-Interface"[16] für den RasPi realisiert.

[16]Link: `http://elinux.org/RPi-Cam-Web-Interface`

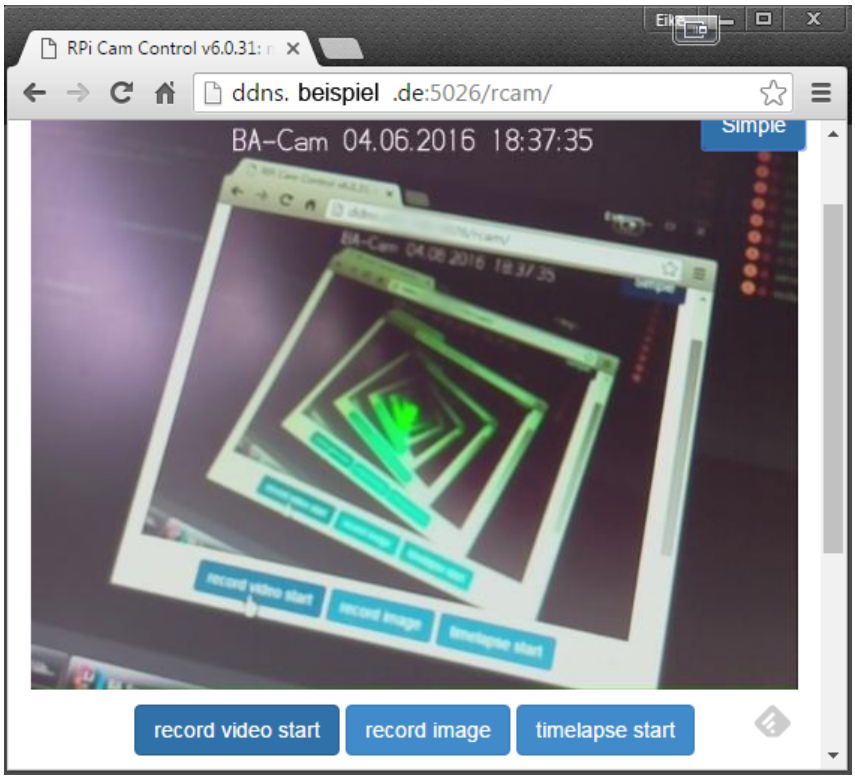


Abbildung 18: Externer Zugriff auf Kamera

Die Software erstellt einen Webserver, der von einem einen beliebigen Internet-Browser aus aufgerufen werden kann (siehe Abbildung 18). Über die Internetseite kann das Bild der Kamera eingesehen und es können diverse Einstellungen vorgenommen werden. Auch lassen sich über die Seite Video- und Bildaufnahmen starten und ansehen. Damit das nicht nur im selben Netzwerk, sondern von überall aus funktioniert, muss DynamicDNS im Router, der sich im selben Netzwerk wie der RasPi befindet, aktiviert sein. Da der Kamera-Webserver über einen anderen Port als der des CC-Servers erreichbar ist, muss dieser neben dem Port des Casa Control Servers in die Liste der freigegebenen Ports aufgenommen werden (Abbildung 19).

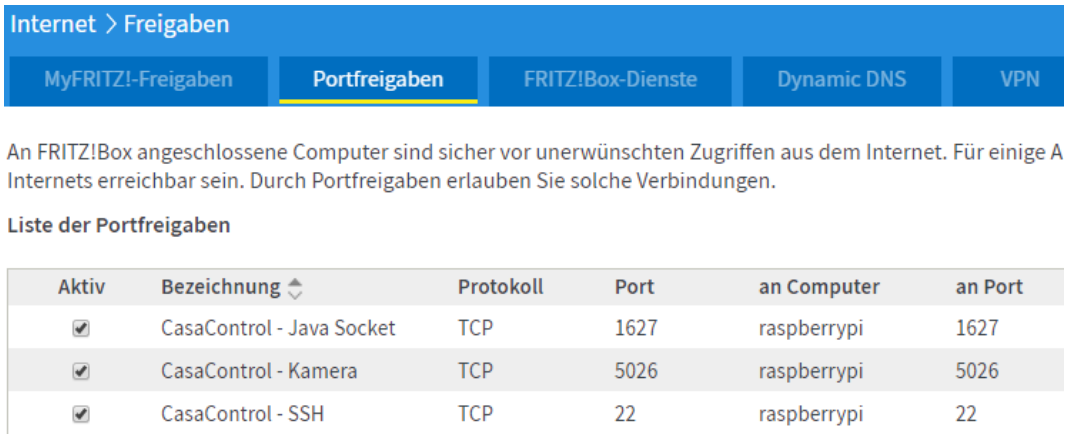


Abbildung 19: Portfreigabe in einem Router

Findet nun von außen ein Zugriff auf die DDNS-Domain (siehe Adressleiste in Abbildung 18) in Verbindung mit Port 5026 statt, leitet der Router den Zugriff weiter an die netzinterne Adresse des „raspberrypi" weiter.

4.1.4 Zugriff via SSH

Die im vorherigen Unterunterabschnitt 4.1.1 gezeigte grafische Oberfläche ist sicht- und bedienbar, wenn am RasPi ein Bildschirm und Peripheriegeräte wie Maus und Tastatur angeschlossen sind. Der Server-Administrator kann das Protokoll vor Ort am Server einsehen und hat die Möglichkeit, Einstellungen vorzunehmen. Wenn ein Einblick in das Protokoll von außerhalb des Heimnetzes geschehen soll, ermöglicht das Protokoll SSH (Unterunterabschnitt 2.2.7) einen Fernzugriff auf die Konsole des Raspberry Pi. Wenn eine JavaFX-Anwendung, im konkreten Fall der Casa Control Server, aus der Ferne über SSH gestartet wird, kann die grafische Oberfläche der Software nicht übertragen werden. Einzig das Mitlesen von Textausgaben und Eingeben von Konsolen-Befehlen erlaubt die SSH-Verbindung. Aus diesem Grund ist eine Klasse „KonsolenEinga-

ben" (Listing 6) im CC-Server eingebunden, die über einen Thread aufgerufen wird (Listing 5). Die Methode „readLine()" des Konsolen-Objekts (Listing 6 Zeile 6) wartet auf eine Eingabe des Benutzers in der Konsole, welche dann in einen String übergeben wird. Da diese Methode das gesamte Programm anhalten würde, bis eine mit Return bestätigte Eingabe erfolgt, muss das Abwarten eines Befehls in einem parallel zum Hauptprogramm laufenden Thread passieren.

Listing 5: Thread für Konsolen-Eingaben starten

```
1 Runnable konsolenEingaben = new KonsolenEingaben();
2 new Thread(konsolenEingaben).start();
```

Wie schon zuvor in Listing 2 verwendet, findet auch hier eine endlose while-Schleife Platz (Listing 6 Zeile 5-13), die jedes mal erneut aufgerufen wird, sobald eine Konsolen-Eingabe stattfand.

Listing 6: Reaktion auf Konsolen-Eingabe

```
1  private class KonsolenEingaben implements Runnable {
2  @Override
3  public void run() {
4          Console console = System.console();
5          while (true) {
6                  String eingabe = console.readLine();
7                  if (eingabe.toLowerCase().equals("exit")) {
8                          System.out.println("Beende␣Server");
9                          System.exit(0);
10                 } else textausgabe("Unbekannter␣Befehl:
11 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣\"" + eingabe + "\"", "Server");
12         }
13 }
```

Folgende Befehle sind über die Konsole und somit aus der Ferne implementiert:

- **close** - schließt den ServerSocket
- **restart** - startet den ServerSocket neu
- **port:PORTNUMMER** - ändert den Port des ServerSockets auf PORTNUMMER
- **sendegcm:NACHRICHT** - sendet eine Push-Benachrichtigung mit NACHRICHT an alle registrierten Android Geräte
- **exit** - beendet den Casa Control Server

Ein Starten, Beenden sowie Ändern des Ports über den SSH-Client PuTTY von einem entfernten Windows-PC ist beispielhaft in Abbildung 20 demonstriert:

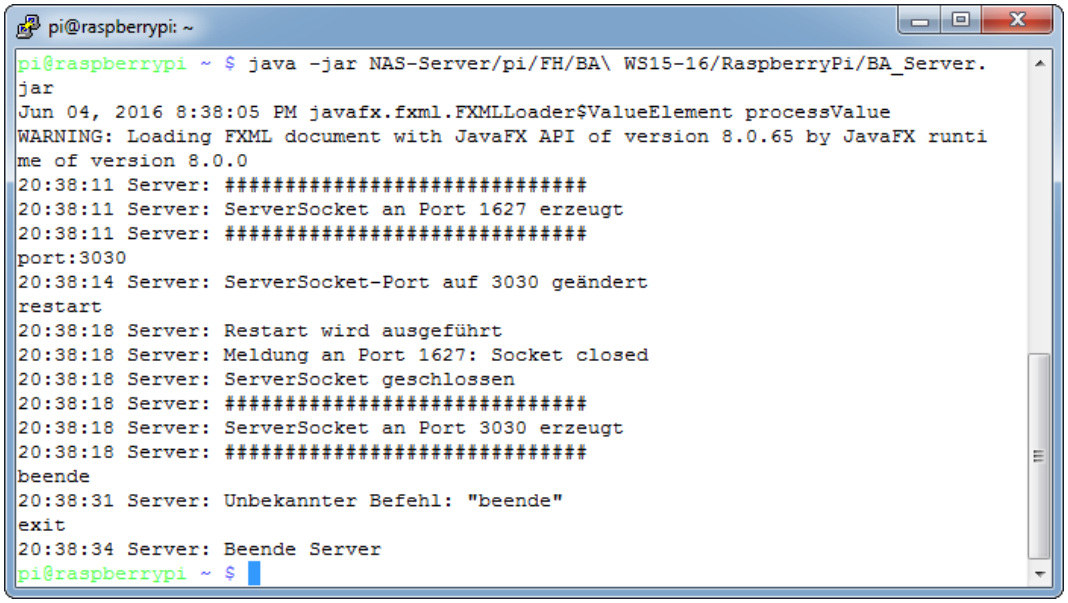


Abbildung 20: Konsolen-Eingaben über SSH aus der Ferne

Das in Unterunterabschnitt 2.2.7 dargelegte Problem, eine über SSH gestartete Anwendung aufrechtzuerhalten ist mit der Software Screen behoben.

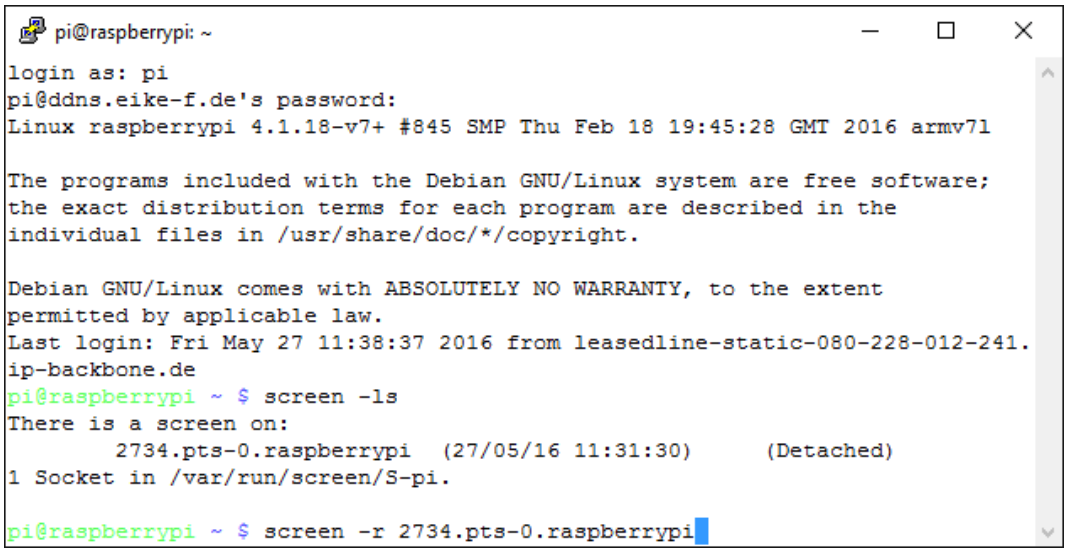


Abbildung 21: SSH mit Screen-Befehl

Abbildung 21: In einer aktiven SSH-Verbindung ist die Liste aller gestarteten Screen-Sitzungen mit

```
1 screen -ls
```

ersichtlich, aus der eine bestimmte Sitzung über den Befehl

```
1 screen -r <Bezeichnung der Sitzung>
```

geöffnet werden kann [19].

Die Screen-Sitzung beendet sich erst nach Aufforderung eines Benutzers und nicht beim Schließen eines SSH-Clients, eine Instanz des CC-Servers kann somit von mehreren SSH-Clients zu jedem Zeitpunkt neu aufgegriffen werden,

um beispielsweise Einlbick in das Protokoll zu haben oder Servereinstellungen vorzunehmen.

4.2 Bewegungsmelder

Die zu programmierende Einheit des WLAN-Bewegungsmelders (Abbildung 16 Punkt 2) ist das ESP8266 WLAN-Modul (Abbildung 16 Punkt 2.2). DerE-SP8266 startet ein in der Programmiersprache Lua geschriebenes Skript beim Anschluss einer Stromquelle.

Zunächst ist eine Verbindung zum Internet notwendig, um Kontakt zur Server-Einheit aufnehmen zu können. Dies geschieht über die integrierte WLAN-Antenne des Moduls und mit folgenden Zeilen im Quellcode des Lua-Skripts:

Listing 7: Verbindung zu WLAN aufbauen

```
1 wifi.setmode(wifi.STATION)
2 wifi.sta.config("SSID","PASSWORT")
3 wifi.sta.connect()
```

In Zeile 2, Listing 7 sind die SSID („Service Set Identifier"[27], Bezeichnung eines WLANs) und das dazugehörige Passwort anzugeben. Da die WLAN-Zugangsdaten fest in das System des ESP8266 einprogrammiert sind, ist ein Vorhandensein des definierten WLANs am Standort nötig, um eine Internetverbindung des Bewegungsmelders zu ermöglichen.

Listing 8: Verbindungsschleife zu WLAN[5]

```
1 tmr.alarm(1, 2000, 1, function()
2      if(wifi.sta.getip()~=nil) then
3           tmr.stop(1)
4           print("Connected!")
5             function raspiAnmeldung()
```

```
6                 conn = net.createConnection(net.TCP, 0)
7                 conn:on("connection", function()
8                   print("connected")
9                   text = "Moin, Bewegungsmelder 01 hier"
10
11                  print(text)
12                  conn:send(text)
13                  conn:close()
14                end)
15                conn:connect(1627,"ddns.beispiel.de")
16              end
17
18              raspiAnmeldung()
19        else
20           print("Connecting...")
21        end
22  end)
```

In Listing 8 Zeile 1 beginnt die Schleife, die versucht eine Verbindung zum definierten WLAN aufzubauen. Alle 2000 ms wird geprüft, ob eine IP-Adresse aus dem WiFi ermittelt werden kann (Zeile 2). Wenn keine Verbindung zu einem WLAN besteht, kann keine IP abgerufen werden, die IP ist somit null („nil"). In diesem Fall ist die Bedingung aus Zeile 2 nicht erfüllt, das Skript fällt in die else-Anweisung in Zeile 19. Entspricht die IP nicht-null, trifft die if-Bedingung aus Zeile 2 zu und die Schleife wird gestoppt. Es folgt eine Verbindung zur DDNS-Domain mit Port des CC-Servers auf dem RasPi (Zeile 15). Im Falle eines erfolgreichen Verbindens zum Server (Zeile 7, „conn:on"), sendet der ESP8266 eine Begrüßungsnachricht (Zeile 9 und 12).

Listing 9: Reaktion auf Bewegung am Sensor

```
1  inpin=4
2  gpio.mode(inpin,gpio.INT)
3  gpio.mode(inpin,gpio.PULLUP)
4  gpio.trig(inpin,"up",bewegung)
5        function bewegung()
6             conn = net.createConnection(net.TCP, 0)
7                    conn:on("connection", function()
8                      text = "Bewegung;1"
9                      print(text)
10                     conn:send(text)
11                     conn:close()
12                   end)
13         conn:connect(1627,"ddns.beispiel.de")
14       end
```

In Listing 9 ist das Verhalten im Falle einer Bewegung am mit dem ESP8266 verbundenen PIR-Sensor (Abbildung 16 Punkt 2.1) nachzuvollziehen. Der digitale Eingangs-Pin des ESP8266 ist in Zeile 1 definiert. Zeilen 2 und 3 definieren den Eingangs-Pin als „Interrupt" und „Pullup". Der Interrupt-Modus bewirkt, dass das Skript angehalten wird, damit eine Funktion ausgeführt werden kann. Der Pullup-Modus gibt an, eine steigende Flanke am Eingangs-Pin 4 als Signal zu erkennen. Zeile 4 definiert für ein erhaltenes Signal die Funktion „bewegung" (Zeilen 5-14), in der erneut eine Verbindung zum ServerSocket des CC-Servers aufgebaut und anschließend eine Textnachricht mit dem Inhalt „Bewegung;1" übermittelt wird.

4.3 Casa Control - Client

4.3.1 Casa Control - Android App

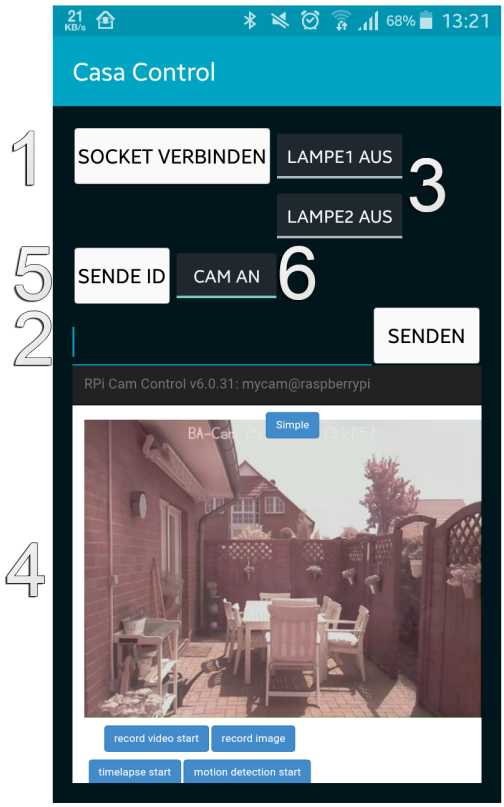


Abbildung 22: App Bedienoberfläche

Der Übersichtlichkeit halber ist die Oberfläche der Android App einfach gehalten. Beim Start der Anwendung findet zunächst ein Verbindungsaufbau zum ServerSocket des CC-Servers statt. Damit die App parallel dazu weiterhin bedient werden kann, erfolgt die Verbindung zum Casa Control Server in einem sogenannten „AsyncTask”:

Listing 10: Verbindung zum CC-Server über Socket

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3          new VerbindeClientSocket().execute();
4  }
5
6  private class VerbindeClientSocket extends AsyncTask {
7  @Override
8     protected Object doInBackground(Object[] params) {
9        String host = ddns.beispiel.de
10       int port = 1627
11       Socket clientSocket = new Socket(host, port);
12       PrintWriter out = new PrintWriter(clientSocket
13          .getOutputStream(), true);
14       BufferedReader in  = new BufferedReader(
15          new InputStreamReader(clientSocket
16          .getInputStream()));
17       return null;
18         }
19 }
```

Ein AsyncTask kann, wie auch ein Thread, Prozesse parallel verarbeiten. In Zeile 3 (Listing 10) ist das Ausführen des AsyncTasks in der beim Start automatisch aufgerufenen Methode „onCreate()" platziert. Der AsyncTask „VerbindeClientSocket" (Zeile 6-16) wird mit Hilfe der überschriebenen Methode „doInBackground()" (Zeile 8) im Hintergrund der App ausgeführt. Alternativ kann das Verbinden zum ServerSocket auch über die Schaltfläche „Socket verbinden" (Abbildung 22 Punkt 1) geschehen.

Wie auch im Listing 3 des Casa Control Servers werden ein PrintWriter (Zeile 12 u. 13) für ausgehende und ein BufferedReader (Zeile 14-16) für eingehende

Nachrichtenkommunikation zwischen ServerSocket (RasPi) und ClientSocket (Android App) verwendet. Eine individuelle Nachricht an den Server ist über ein Textfeld (Abbildung 22 Punkt 2) mit dem Betätigen der Schaltfläche „Senden" möglich.

Die Schaltflächen „Lampe1 Aus" und „Lampe2 Aus" (Abbildung 22 Punkt 3) sind sogenannte ToggleButtons, die eine Zustandsänderung zwischen true & false optisch darstellen. Wird ein ToggleButton gedrückt, ändert sich die Farbe des am unteren Rand angelehnten Streifens und die Bezeichnung.

Listing 11: ToggleButton Lampe

```
1  ToggleButton btnLampe2 = (ToggleButton)
2                          findViewById(R.id.btnLampe2);
3  btnLampe1.setTextOn("Lampe1␣an");
4  btnLampe1.setTextOff("Lampe1␣aus");
5  btnLampe1.setOnClickListener(new View.OnClickListener() {
6     @Override
7     public void onClick(View v) {
8        if (clientSocket != null) {
9           if (clientSocket.isConnected()) {
10             if (((ToggleButton) v).isChecked()) {
11                out.println("Lampe␣an");
12             } else {
13                out.println("Lampe␣aus");
14             }
15          } else Toast.makeText(getApplicationContext(),
16                "Keine␣Verbindung",
17                Toast.LENGTH_SHORT).show();
18       } else Toast.makeText(getApplicationContext(),
19             "Socket␣ist␣null",
20             Toast.LENGTH_SHORT).show();
```

```
21      }
22  });
```

In Listing 11 ist das Verhalten eines ToggleButtons verdeutlicht. In Zeile 3 und 4 wird der Bezeichnung für den An- und Aus-Status des Buttons angepasst. Falls ein Objekt von „clientSocket" existiert, es also nicht-null ist (Zeile 8) und der exisiterende clientSocket verbunden ist (Zeile 9) soll der PrintWriter „out" eine Textnachricht mit dem Inhalt „Lampe1 an" an den Server senden (Zeile 11). Steht der Zustand des ToggleButtons auf true (damit wäre er „checked", Zeile 10) wird die Nachricht „Lampe aus" (Zeile 13) an den ServerSocket übermittelt. In Listing 3 aus Unterunterabschnitt 4.1.1 erfolgt wie beschrieben die weitere Verarbeitung des Befehls.

4.3.2 Webcam Live-Stream

Für das Darstellen des Kamerabildes des Servers wird in die untere Hälfte der App-Oberfläche ein „WebView"-Objekt eingebunden (Abbildung 22 Punkt 4). Eine WebView erlaubt es, eine bestimmte Internetseite in einer definierten Fenstergröße anzuzeigen und mit der Seite zu interagieren.

Listing 12: WebView Kamera

```
1 WebView webView = (WebView) findViewById(R.id.webView);
2 webView.loadUrl("http://ddns.beispiel.de:5026/rcam/");
3 webView.setInitialScale(40);
```

In Listing 12 Zeile 1 wird das WebView-Objekt erzeugt, das in Zeile 2 die Anweisung erhält, die Internetadresse des Kamera-Webservers (Abbildung 18) zu laden. Um die Seite optimal darzustellen ist die Größe der Internetseite auf 40 Prozent reduziert (Zeile 3).

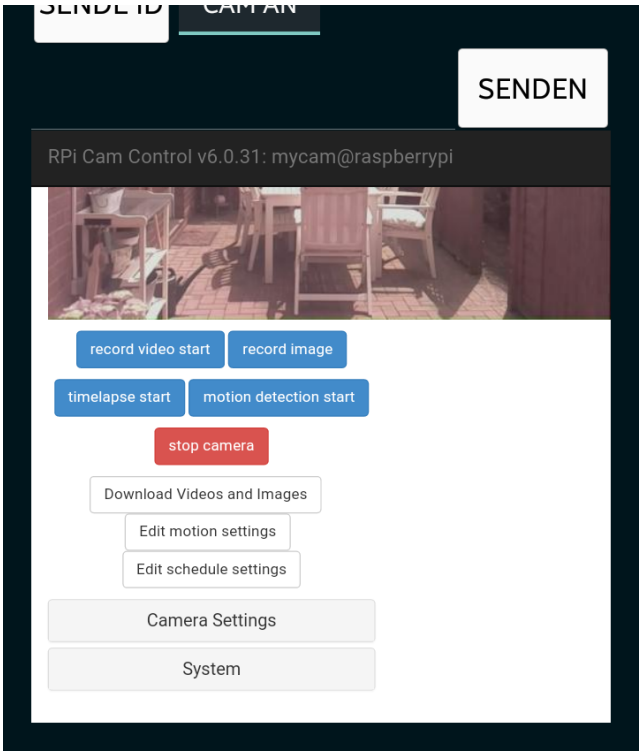


Abbildung 23: Weitere Einstellungen der Kamera

Unter anderem ist eine Videoaufnahme des Kamerabildes über den WebView-Bereich möglich, da die Internetseite diese Funktion zur Verfügung stellt. Weil sich die WebView wie ein Browser-Fenster verhält, kann auf der Internetseite nach unten gescrollt werden, wo sich weitere Einstellungen und Buttons zum Aufrufen diverser Funktionen befinden (Abbildung 23).
Da die Übertragung des Live-Bildes einen hohen Verbrauch an Daten verursacht, ist mit der Schaltfläche „Cam An" (Abbildung 22 Punkt 6) die WebView deaktivierbar. Der Verbindung zur Kamera wird dadurch getrennt, sodass keine weiteren Datenpakete zwischen Server und Android-App ausgetauscht werden.

4.3.3 GCM - clientseitig

Damit das Empfangen von GCM-Nachrichten (Unterunterabschnitt 2.2.3) auf der Seite des Clients ermöglicht werden kann, ist eine Registrierung des Android-Smartphones beim GCM-Server notwendig.

Listing 13: GCM Client Registrierung[6]

```
1  private void registerInBackground() {
2    new AsyncTask<Void, Void, String>() {
3      @Override
4      protected String doInBackground(Void... params) {
5        String msg = "";
6        GoogleCloudMessaging gcm;
7        String regid;
8          if (gcm == null) {
9            gcm = GoogleCloudMessaging.getInstance(context);
10         }
11         regid = gcm.register(SENDER_ID);
12         msg = "Device␣registered,␣registration␣ID=" + regid;
13
14         out.println("RegID_Erhalten;"+regid);
15         storeRegistrationId(context, regid);
16         return msg;
17     }
18   }.execute(null, null, null);
19 }
```

Wie auch das Verbinden mit dem ServerSocket (Listing 10) geschieht die Registrierung des Android-Smartphones mittels eines AsyncTasks im Hintergrund, um ein Blockieren der grafischen Oberfläche zu verhindern. Dem „GoogleCloudMessaging"-Objekt wird eine Instanz übergeben (Listing 13 Zei-

le 9), um in Zeile 11 eine Registrierungs-ID vom GCM-Server zu erhalten. In Zeile 14 wird die zugewiesene Registrierungs-ID dem ServerSocket des CC-Servers über einen PrintWriter mitgeteilt. Damit nicht bei jedem Neustart der App erneut eine die gleiche Registrierungs-ID angefordert werden muss, wird diese im Speicher des Android-Smartphones abgelegt (Zeile 15).

Nach Zustellung einer ankommenden GCM-Nachricht seitens des Servers wird eine Benachrichtigung durch folgende Methode ausgelöst:

Listing 14: Anzeige einer Push-Benachrichtigung

```
1  private void sendNotification(String msg, String title) {
2    mNotificationManager = (NotificationManager)
3      this.getSystemService(Context.NOTIFICATION_SERVICE);
4    PendingIntent contentIntent = PendingIntent.getActivity(
         this, 0,
5      new Intent(this, MainActivity.class), 0);
6
7    Uri alarmSound = RingtoneManager
8      .getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
9
10   NotificationCompat.Builder mBuilder =
11                 new NotificationCompat.Builder(this)
12         .setSmallIcon(R.drawable.ic_stat_cc)
13         .setContentTitle(title)
14         .setStyle(new NotificationCompat.BigTextStyle()
15         .bigText(msg))
16         .setVibrate(new long[] {0, 200, 100, 200,100,200})
17         .setSound(alarmSound)
18         .setContentText(msg);
19
```

```
20          mBuilder.setContentIntent(contentIntent);
21          mNotificationManager
22            .notify(NOTIFICATION_ID, mBuilder.build());
23      }
```

In Listing 14 Zeile 7 und 8 wird die akustische Benachrichtigung für den Erhalt einer GCM-Nachricht festgelegt - hier die Standard-Einstellung des Smartphones. In den Zeilen 10-18 findet die Erstellung der Benachrichtigung mit Bild (Zeile 12), Überschrift (Zeile 13), Inhalt (Zeile 15 u. 18), Klingelton (Zeile 17) und Vibrationsmuster (Zeile 16) statt.

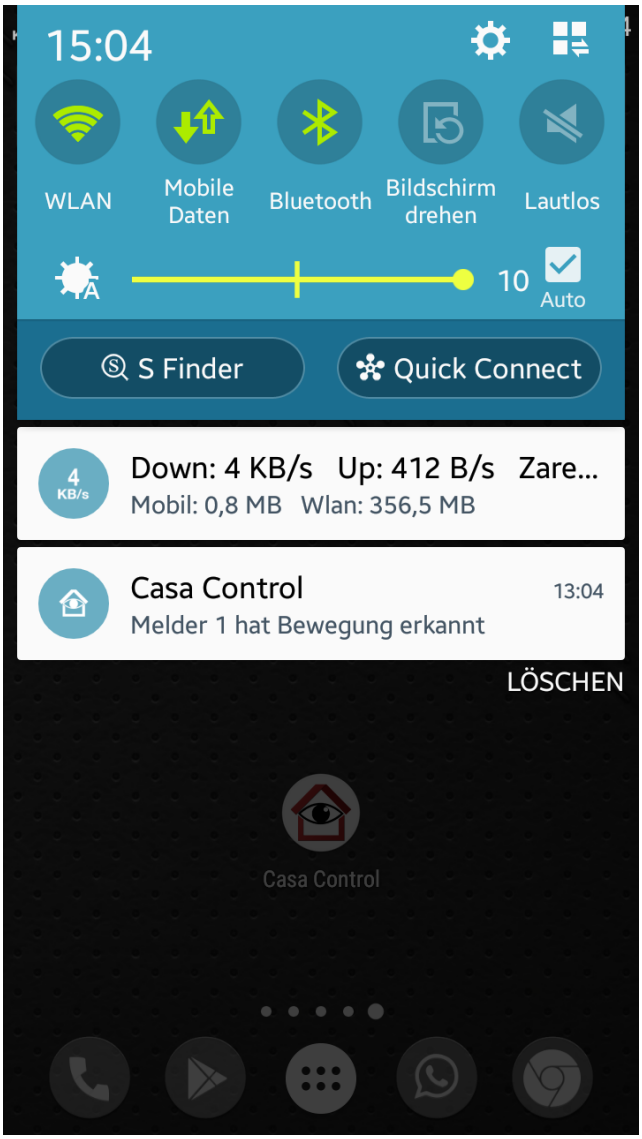


Abbildung 24: Push-Benachrichtigung

In Abbildung 24 ist ein Beispiel für eine Benachrichtigung dargestellt. Falls das Smartphone mit einer Android Smartwatch verbunden ist, erscheint die Benachrichtigung auch hierüber (Abbildung 25).

(a) Vorschau der Benachrichtigung (b) Benachrichtigung erweitert

Abbildung 25: Casa Control Benachrichtigung auf Android Smartwatch

5 Zusammenfassung

5.1 Ergebnis

Nach Zusammenstellung der Komponenten und Realisierung des Quellcodes ergab ein Test der Überwachungsanlage ein zufriedenstellendes Ergebnis. Die Kommunikation zwischen Sockets verläuft ohne Verzögerung oder Fehler, die Erreichbarkeit des Servers ist zuverlässig, die Android-App arbeitet ohne Abstürze und der Zugriff auf die Kamera ist zuverlässig.

Der internetfähige Bewegungsmelder, speziell das ESP8266 WLAN-Modul erfüllt weitestgehend seinen Zweck, benötigt jedoch in unkontrollierten Abständen einen Neustart. Vermutlich ist der Fehler auf ein Spannungsproblem zurückzuführen, das mit einem hinzufügen eines Kondensators behoben werden könnte. An dieser Stelle bedarf es noch weiterer Verbesserungen.

Darüberhinaus ist es in der Praxis umständlich, das Protokoll des Servers nur

über einen SSH-Client einsehen zu können, eine automatische Übertragung und Darstellung des Protokolls über den Socket würde ein einfacheres Prüfen ermöglichen und Zeit sparen.
Dadurch, dass die Raspberry Pi Kamera am Server keinen Infrarotfilter besitzt, werden Farben am Tag leicht verfärbt dargestellt (siehe Abbildung 22). Ein Kamera-Modul mit Infrarotfilter ist für eine Überwachungsanlage nicht ratsam, da eine Nachtsicht mit Infrarotscheinwerfer damit nicht möglich wäre. Alternativ könnten mehrere Kameras bereitgestellt werden, die unterschiedlichen Anforderungen gerecht werden. Ein breiter Bereich der überwacht werden soll, könnte mit einer schwenkbaren Kamera ausgestattet werden, während ein Bereich, der weit von der Kamera entfernt ist, eine hohe Auflösung oder eine Zoom-Funktion der Kamera benötigt.

Das Ziel der Arbeit war es, ein umfangreiches, unabhängiges Überwachungssystem mit möglichst energie- und materialsparenden Komponenten zu entwickeln. Mit dem Raspberry Pi Einplatinencomputer mit 3 W Leistungsaufnahme unter Vollast und dem ESP8266 WLAN-Modul mit 0,5 W Leistungsaufnahme beträgt der Energieverbrauch des gesamten Systems pro Jahr im Dauerbetrieb lediglich 30,66 kWh[17]. Das ist im Gegensatz zu käuflich erwerblichen Produkten nicht nur deutlich stromsparender, sondern gleichzeitig unabhängig und sicherer. Sicherer, weil die gespeicherten Daten, wie zum Beispiel Bewegungsprotokolle oder Kamera-Aufnahmen nicht auf einem zentralen, zu einem externen Dienstleister gehörenden Speicher abgelegt werden, sondern sich im privaten Netzwerk des Benutzers befinden. Dieses zu hacken ist für einen Kriminellen nicht von großem Nutzen, da es weitaus lukrativer wäre, eine große Datenbank eines Unternehmens zu berauben. Des Weiteren hat die Sicherung

[17] $3,5W \cdot 24h \cdot 365 = 30660Wh$

auf privaten Speichermedien den Vorteil, durchgehende Kontrolle über die Daten zu haben und den externen Zugriff auf diese im Notfall durch Trennen der Internetverbindung zu verhindern.

5.2 Ausblick

Dank des immer größer werdenden IoT-Gemeinwesens gibt es zahlreiche Möglichkeiten, das System mit Komponenten zu erweitern. Zum Beispiel kann am zu überwachenden Ort ein RFID-Lesegerät („radio-frequency identification", deutsch „Identifizierung mit Hilfe elektromagnetischer Wellen"[26]) zum Lesen von RFID-Chips oder ein Tastenfeld mit Ziffern platziert werden, um eine Identifikation autorisierter Personen zu ermöglichen, die ein Auslösen eines Alarms unterbinden können. Über ein GSM-Modul („Global System for Mobile Communications"[23]) könnten Anrufe getätigt werden, die vor Ort mit Lautsprecher und Mikrofon als Fernsprechanlage dienen könnten. Über ein GSM-Modul könnte auch die mobile Verbindung zum Internet erfolgen, was im Zusammenhang mit den stromsparenden Komponenten eine Insellösung ermöglichen würde. Ein abgelegenes Objekt ohne Strom- und Internetleitung kann dann über einen mobilen Bewegungsmelder, bestehend aus Batterie („Powerbank" mit USB), PIR-Sensor, ESP8266 und GSM-Modul betrieben werden, um entfernte Besitzer über ungebetene Gäste zu informieren.

Um das in Abschnitt 1 angesprochene smarte Verhalten des Systems zu erzielen, könnte der Bewegungsmelder mit einer Lebenszeichen-Funktion programmiert werden. An den Server würde dann alle fünf Minuten eine Nachricht „Melder 5 lebt" gesendet werden. Der Server prüft durchgehend, ob die Lebenszeichen aller Bewegungsmelder ankommen. Falls nicht, könnte mit Hilfe einer

Funksteckdose der Bewegungsmelder neu gestartet werden, was voraussetzen würde, dass sich Server und Bewegungsmelder in unmittelbarer Nähe befinden. In der vorgestellten Realisierung ist die Nachrichtenkommunikation zwischen ClientSocket und ServerSocket unverschlüsselt. Eine SSL-Verschlüsselung („Secure Sockets Layer"[11, S. 820]) würde die Sicherheit des Systems und das Mitlesen durch dritte verhindern.

6 Anhang

6.1 Abkürzungsverzeichnis

Abkürzung	Bezeichnung	Erstnutzung
AT	Attention	S. 15
API	application programming interface	S. 23
App	Applikation	S. 19
CSI	Camera Serial Interface	S. 11
DDNS	Dynamisches DNS	S. 27
DNS	Domain Name System	S. 27
GCM	Google Cloud Messaging	S. 21
GPIO	general purpose input/output	S. 11
HDMI	High Definition Multimedia Interface	S. 10
IDE	integrated development environment	S. 19
IoT	Internet of Things	S. 6
JDK	Java Development Kit	S. 18
JRE	Java Runtime Envirnoment	S. 12
LAN	Local Area Network	S. 26
PIR	passive infrared	S. 16
RasPi	Raspberry Pi	S. 10
SD	Secure Digital	S. 11
SoC	System-on-a-Chip	S. 15
SPI	Serial Peripheral Interface	S. 15
SSID	Service Set Identifier	S. 51
TCP	Transmission Control Protocol	S. 15
TCP/IP	Transmission Control Protocol/Internet Protocol	S. 42
UART	Universal Asynchronous Receiver Transmitter	S. 15
USB	Universal Serial Bus	S. 10
WLAN	Wireless Local Area Network	S. 6

6.2 Abbildungsverzeichnis

6.3 Listingverzeichnis

6.4 Literaturverzeichnis

[1] E. Bartmann. *Die elektronische Welt mit Raspberry Pi entdecken.* Köln: O'Reilly Germany, 2013. ISBN: 978-3-955-61110-1.

[2] S. Chin und J. Weaver. *Raspberry Pi with Java: Programming the Internet of Things (IoT)* -. 1. Aufl. Madison: McGraw Hill Professional, 2015. ISBN: 978-0-071-84202-0.

[3] Einplatinencomputer.com. *433 Mhz Funksteckdose schalten.* `http://www.einplatinencomputer.com/raspberry-pi-433-mhz-funksteckdose-schalten/`. [Online; Aufgerufen am 04.06.2016 17:56].

[4] E. F. Engelhardt. *Hausautomation mit Raspberry Pi - Alarmanlage, Heizung, Smart Home, W-LAN & Co: 20 Projekte, die Ihr Leben leichter machen.* 3. Auflage. München: Franzis Verlag GmbH, 2014. ISBN: 978-3-645-60313-3.

[5] eps8266-projects.com. *ESP8266 – PIR Motion sensor detection.* `http://www.esp8266-projects.com/2015/03/esp8266-pir-motion-sensor-detection.html`. [Online; Aufgerufen am 23.04.2016 22:46].

[6] Github – ton1n8o. *GCMApp.* `https://github.com/ton1n8o/GCMApp`. [Online; Aufgerufen am 04.05.2016 21:12].

[7] Golem. *Firmware - Golem.de.* `http://www.golem.de/specials/firmware/`. [Online; Aufgerufen am 25.05.2016 17:23].

[8] W. Höfer. *Raspberry Pi programmieren mit Java.* 1. Aufl. Heidelberg: MITP-Verlags GmbH & Co. KG, 2016. ISBN: 978-3-958-45057-8.

[9] C. Immler. *Raspberry Pi für Maker. Mach's einfach! - DIE KOMPAKTESTE GEBRAUCHSANWEISUNG MIT 222 ANLEITUNGEN*. 1. Aufl. München: Franzis Verlag GmbH, 2015. ISBN: 978-3-645-60351-5.

[10] C. Kühnel. *Building an Iot Node for Less Than 15 $ - Nodemcu & Esp8266*. 1. Aufl. Skript Verlag Kuhnel, 2015. ISBN: 978-3-907-85730-4.

[11] M. Kofler, C. Kühnast und C. Scherbeck. *Raspberry Pi - Das umfassende Handbuch, komplett in Farbe - aktuell zu Raspberry Pi 2 - inkl. Schnittstellen, Schaltungsaufbau, Steuerung mit Python u.v.m.* 2. Aufl. Bonn: Rheinwerk Verlag GmbH, 2015. ISBN: 978-3-836-23795-6.

[12] mb-v2 datasheet.pdf. *docdroid.net*. `https://www.docdroid.net/9oyf/mb-v2-datasheet.pdf.html`. [Online; Aufgerufen am 02.06.2016 15:01].

[13] Mikrocontroller.net. *ESP8266 – Mikrocontroller.net*. `http://www.mikrocontroller.net/articles/ESP8266`. [Online; Aufgerufen am 01.06.2016 17:19].

[14] Mikrocontroller.net. *Serial Peripheral Interface – Mikrocontroller.net*. `http://www.mikrocontroller.net/articles/Serial_Peripheral_Interface`. [Online; Aufgerufen am 01.06.2016 17:13].

[15] Mikrocontroller.net. *UART – Mikrocontroller.net*. `http://www.mikrocontroller.net/articles/UART`. [Online; Aufgerufen am 01.06.2016 17:09].

[16] Raspberry Pi Forum. *Programmiersprachen auf dem Raspberry Pi*. `http://www.forum-raspberrypi.de/Thread-faq-welche-programmiersprachen-kann-ich-auf-dem-raspberry-pi-nutzen`. [Online; Aufgerufen am 4.06.2016 20:49].

[17] M. Schwartz. *Home Automation with the ESP8266 - Build Home Automation Systems Using The Powerful & Cheap ESP8266 WiFi Chip*. 1. Aufl. Altendorf, Schweiz: Open Home Automationl, 2015. ISBN: 978-1-523-81005-5.

[18] Seriousandroiddeveloper.in. *Google Cloud Messaging Tutorial*. http://www.seriousandroiddeveloper.in/listoffile/google-cloud-messaging-tutorial. [Online; Aufgerufen am 02.05.2016 16:55].

[19] Raspberry Pi StackExchange. *Will Terminating an SSH Connection Also Terminate any Program running?* http://raspberrypi.stackexchange.com/questions/8085/will-terminating-an-ssh-connection-also-terminate-any-program-running/8086. [Online; Aufgerufen am 26.05.2016 19:38].

[20] TIOBE. *TIOBE Index*. http://www.tiobe.com/tiobe_index. [Online; Aufgerufen am 31.05.2016 11:04].

[21] Wikipedia. *AT-Befehlssatz – Wikipedia*. https://de.wikipedia.org/wiki/AT-Befehlssatz. [Online; Aufgerufen am 30.05.2016 10:47].

[22] Wikipedia. *Bewegungsmelder – Wikipedia*. https://de.wikipedia.org/wiki/Bewegungsmelder#Funktionsprinzip. [Online; Aufgerufen am 24.05.2016 12:33].

[23] Wikipedia. *Global System for Mobile Communications – Wikipedia*. https://de.wikipedia.org/wiki/Global_System_for_Mobile_Communications. [Online; Aufgerufen am 05.06.2016 16:31].

[24] Wikipedia. *Java-Laufzeitumgebung – Wikipedia*. https://de.wikipedia.org/wiki/Java-Laufzeitumgebung. [Online; Aufgerufen am 28.05.2016 20:43].

[25] Wikipedia. *Raspberry Pi – Wikipedia*. https://de.wikipedia.org/wiki/Raspberry_Pi#Namensgebung_und_Logo. [Online; Aufgerufen am 28.05.2016 20:58].

[26] Wikipedia. *RFID – Wikipedia*. https://de.wikipedia.org/wiki/RFID. [Online; Aufgerufen am 05.06.2016 16:25].

[27] Wikipedia. *Service Set – Wikipedia*. https://de.wikipedia.org/wiki/Service_Set#Service_Set_Identifier. [Online; Aufgerufen am 06.06.2016 22:03].

[28] Wikipedia. *Transmission Control Protocol – Wikipedia*. https://de.wikipedia.org/wiki/Transmission_Control_Protocol. [Online; Aufgerufen am 06.06.2016 20:01].

[29] Wikipedia. *Transmission Control Protocol/Internet Protocol – Wikipedia*. https://de.wikipedia.org/wiki/Transmission_Control_Protocol/Internet_Protocol. [Online; Aufgerufen am 06.06.2016 21:34].

6.5 Quellcode

Die Quellcodes der programmierten Software befinden sich auf der beiliegenden DVD.