

DHBW Karlsruhe

TINF12B5

Studienarbeit

**Entwicklung eines Komplettsystems zur
Überwachung und Beleuchtung von Innen-
und Außenbereichen mit Raspberry Pi und
iOS App**

Autor:
Timo Höting
2185611

Betreuer:
Stefan Lehmann

Erklärung

Gemäß §5(3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 22. September 2011.

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Ort, Datum

Unterschrift

Inhaltsverzeichnis

1	Einleitung	3
2	Teilprojekte	4
3	Projektmanagement	5
3.1	Meilensteintrendanalyse	5
3.2	Meilensteine und Erfolgsprüfung	5
4	LED-Pixel	8
4.1	Bewertungskriterien	8
4.2	Evaluierung	8
4.3	Teststellung	9
5	Bewegungssensor	12
5.1	Bewertungskriterien	12
5.2	Evaluierung	12
5.3	Teststellung	13
6	Kamera	15
6.1	PI-Kamera vs. Netzwerkkamera	15
6.2	Herangehensweise Ansteuerung	15
6.3	Auswertung des RTSP/RTP-Protokolls mit Python	16
6.4	RTSP mit Python	16
6.5	Speicherung des Video-Streams mit FFmpeg	17
6.6	Speicherung des Bild mittels Screenshot	18
6.7	Speicherung des Bild mittels HTTP-Request	19
7	Verschlüsselung	22
7.1	SSL vs. TLS	22
7.2	Vor- und Nachteile TLS	22
7.3	TLS Handshake	22
7.4	StartTLS	23
7.5	Server Zertifikat	23
7.6	Apple Zertifikat	23
7.7	Wireshark Trace	24
8	Python-Server und Protokoll	25
8.1	Protokoll	25
8.2	Server-Framework	26
8.3	Beispielimplementierung Webserver	26
8.4	Implementierung Webserver	27
8.5	Hashfunktion	28
9	Anwendungsstruktur Serverapplikation	29
9.1	Klassen und ihre Funktionen	29
9.2	Auswertung empfangener Daten	30
9.3	Konfiguration	33

9.4	Apple Push Notification	35
9.5	Unit-Test	37
9.6	Threads	37
10	Konfiguration und Installation	40
10.1	Installation	40
10.2	Konfiguration	40
11	iOS App	41
11.1	Swift	41
11.2	Übertragung	41
11.3	CoreData	43
11.4	Konzept	44
11.5	Aufbau	45
12	Praktische Umsetzung	46
13	Fazit	47
14	Abbildungsverzeichnis	48
	Listings49	

1 Einleitung

Diese Studienarbeit wird im Zuge des Studiums Bachelor of Engineering - Informationstechnik an der DHBW Karlsruhe erstellt.

Im dieser Studienarbeit soll ein Komplettsystem entwickelt werden, dass sowohl die Überwachung als auch die Steuerung der Beleuchtung von Innen- und Außenbereichen ermöglicht. Das System soll nach der Entwicklung universell einsetzbar und leicht konfigurierbar sein.

Die Beleuchtung soll mit adressierbaren LED-Pixeln umgesetzt werden, da diese sehr leicht steuer- und erweiterbar sind. Für die Erkennung von Bewegungen sollen klassische Bewegungsmelder eingesetzt werden. Mittels einer Kamera sollen Bilder aufgerufen und gespeichert werden können. Die gesamte Steuerung soll mittels einer iOS-App über einen Raspberry Pi möglich sein. Die Implementierung dieser App soll in Swift erfolgen und die der Server-Anwendung in Python.

Es müssen passende Bauteile und Produkte evaluiert und getestet werden. Diese müssen vom Raspberry Pi ansteuerbar sein. Des weiteren muss die Architektur der Serveranwendung und iOS-App ausgearbeitet werden. Zur Fertigstellung müssen beide Anwendungen implementiert werden und die funktionsfähige Anwendung an einem Beispielobjekt in Betrieb genommen werden.

Es gibt drei verschiedene Modi in denen sich das System befinden kann:

- Beleuchtung wird durch Bewegungsmelder ausgelöst
- Beleuchtung wird manuell vom Benutzer über App gesteuert
- Bewegungsmelder als Alarmanlage, beim Auslösen wird der Benutzer benachrichtigt und ein Bild der Kamera als Notification auf dem Smartphone angezeigt

2 Teilprojekte

- LED-Pixel und Bewegungssensoren evaluieren / ansteuern
- Implementierung der Ansteuerung aller Bauteile
- Implementierung der Netzwerkkommunikation
- Implementierung der iOS App
- praktische Umsetzung an einem Beispielobjekt

3 Projektmanagement

3.1 Meilensteintrendanalyse

Die Meilensteintrendanalyse ist eine Art des Projektmanagements. Hauptaufgabe ist die Überwachung des Projektfortschritts und die frühe Erkennung von Terminverzögerungen. Hierfür werden bei Projektbeginn Meilensteine festgelegt, die Inhalt, Dauer und Endzeitpunkt enthalten. Im Laufe eines Bearbeitungszeitraums können mögliche Verzögerungen erkannt und entsprechend darauf reagiert werden. Um große Verzögerungen zu vermeiden, sollten realistische Sicherheitspuffer eingeplant werden.

Bei Beendigung eines Meilensteins kann ein Fazit aus dessen Ablauf gezogen werden. Zum Beispiel kann bei aufgetretener Verzögerung Ursachenforschung betrieben werden, um in weiteren Schritten solche Verzögerungen zu vermeiden.

Bemerkung

- Die schriftliche Ausarbeitung ist nicht Teil der Meilensteine. Sie erfolgt parallel zu den durchgeführten Aufgaben.

3.2 Meilensteine und Erfolgsprüfung

1. Planung der Architektur (15. September - 30. September 2014)

- Entwurf Anwendungsstruktur
 - Der Entwurf der Anwendungsstruktur für Serverimplementierung und Mobile-Implementierung konnte erfolgreich abgeschlossen werden.
- Ermittlung notwendiger Hardware für die einzelnen Anwendungsfälle
 - Dies benötigte nur geringen Aufwand, da nur wenige Bauteile benötigt werden. Die Evaluierung, Beschaffung und Tests fällt in den 3. Meilenstein.
- Ausarbeitung Übertragungsprotokoll
 - Das Protokoll wurde erfolgreich ausgearbeitet. Die Details sind in 2.5.1 dargestellt.
- Einarbeitung in Python
 - Es wurden die Grundlagen der Sprache Python im Bezug auf OOP, Funktionen und Datentypen erarbeitet. Die Kenntnisse haben sich im Laufe des Projekts weiter verbessert.

2. Funktionsfähiger Prototyp Webserver (1. Oktober - 19. Oktober 2014)

- Auswahl eines Frameworks für die Implementierung des Webservers
 - Es wurde das Twisted Framework ausgewählt und Testimplementierung der verschiedenen Server-Typen (Socket, SSL, STARTTLS, HTTP, HTTPS) durchgeführt.
- Implementierung der für den Webserver nötigen Klassen
 - Implementierung der Socketübertragung. Im Laufe des Projekts zeigte sich, dass dies nicht die optimale Lösung ist. In einem späteren Meilenstein wurde ein HTTPS-Webserver mit Twisted implementiert.

- Testen der Funktionen
 - Testen mithilfe von Clients, die in Python implementiert wurden.
3. Auswahl Hardwarekomponenten (LEDs, Sensoren, Kamera) (20. Oktober - 31. Oktober 2014)
- Evaluierung
 - Die Evaluierung von LEDs und Sensoren konnte erfolgreich abgeschlossen werden. Für die Wahl der Netzwerkkamera konnte keine Lösung gefunden werden, da noch nicht klar war, in welcher Form die Bilder abgerufen werden können. Dieser Aufgabenteil ist in Meilenstein 7 verschoben worden.
 - Beschaffung
 - Die Beschaffung von LEDs und Sensoren verlief mit erfolgreich mit geringem Aufwand.
 - Testen und Testimplementierung
 - Die Testimplementierungen erfolgreich durchgeführt werden. Der Quellcode und die zugehörigen Schaltbilder befinden sich in den Punkten 2.1 und 2.2.
4. Implementierung Serveranwendung (1. November - 30. November 2014)
- Implementierung Sensorerkennung
 - Die Implementierung der Sensorerkennung war erfolgreich.
 - Implementierung Ansteuerung LED
 - Die Implementierung der Ansteuerung der LEDs war erfolgreich.
 - Implementierung der Konfigurationsmöglichkeiten (hat nicht geklappt -> Dezember, Januar)
 - Die vollständige Implementierung der Konfigurationsdateien, -lesern und -schreibern, sowie der Übertragung wurde nicht abgeschlossen. Grund dafür war fehlende Kenntniss über den Empfang auf dem Client und über JSON.
 - Implementierung des Übertragungsprotokolls
 - Die Auswertung der empfangenen Daten wurde erfolgreich implementiert.
 - Komplette Implementierung
 - Die Implementierung der gesamten Serveranwendung wurde soweit fertiggestellt, dass ein Betrieb möglich war. Einige kleine Änderungen oder nachträgliche Erweiterungen wurden im Laufe des Projekts hinzugefügt. Dies waren meistens Dinge, die vorher nicht bedacht wurden, oder an die mobile App angepasst werden mussten.
5. Funktionsfähiger Prototyp iOS-Anwendung (1. Januar - 31. Januar 2015)
- Einarbeitung Swift und XCode
 - In dieser Zeitphase wurde der Umgang mit XCode und der Sprache Swift erlernt.
 - Erstellung Prototyp der Anwendung in XCode

- Es wurde die Anwendungsstruktur in XCode erstellt.
 - Auswahl von nötigen Frameworks
 - Es wurden Frameworks für Menüführung, Sicherheit und FTP-Verbindung ausgewählt und hinzugefügt.
 - Übertragungen mit dem Webserver
 - In dieser Phase wurde festgestellt, dass die Übertragung über Sockets nicht optimal ist. Eine Übertragung mittels HTTP Protokoll bietet eine deutlich einfachere und sicherere Implementierung. Aufgrund dieser Feststellung wurde die Implementierung des Webserver in diesem Meilenstein verändert.
 - Refactoring Webserver
 - Der Webserver wurde auf HTTPS umgestellt.
 - Viele kleine Veränderungen im Server.
6. Implementierung iOS-Anwendung (1. Februar - 31. März 2015)
- Server-Client Kommunikation
 - Die Übertragung wurde entsprechend dem Anwendungsprotokoll implementiert.
 - User-Interface
 - Das User-Interface wurde erstellt und mit Funktionen versehen.
 - Implementierung konsistente Speicherung
 - Der Zugriff auf CoreData wurde implementiert.
7. Abschluss der Arbeit (1. April - 11. Mai 2015)
- Implementierung Netzwerkkamera
 - Die Anbindung der Netzwerkkamera war einer der aufwändigsten Punkte in diesem Projekt. Nach verschiedenen Ansätzen wurde eine gute Lösung erarbeitet.
 - Beispielobjekt
 - Das gesamte Projekt wurde in einem Treppenhaus installiert.
 - Fertigstellung Ausarbeitung
 - Abgabe

4 LED-Pixel

4.1 Bewertungskriterien

Die Beleuchtung soll durch einzelne LED-Pixel stattfinden. Ein Pixel bedeutet ein Chip auf dem sowohl die LED und der nötige Treiber sitzt. Für die Evaluierung werden folgende Kriterien gewählt:

- **RGB-Farbraum**
Die LED muss den gesamten RGB-Farbraum darstellen können.
Gewichtung: 5, KO-Kriterium
- **Ansteuerung**
Da der Raspberry Pi an einigen seiner Pins Pulsweitenmodulation¹ (PWM) bietet, sollten die LED-Pixel ohne extra Hardware ansteuerbar sein. Eine extra Stromversorgung ist aber bei größerer Anzahl an LEDs unabdingbar.
Gewichtung: 10
- **Framework**
Hier wird bewertet ob der jeweilige Hersteller ein fertiges Framework zu seinen Produkten anbietet.
Gewichtung: 10
- **Kosten**
Es werden nur die reinen Produktkosten, also ohne Versand und Zoll, bewertet.
Gewichtung: 5
- **Extras**
An dieser Stelle können mögliche Extras eines Herstellers einfließen.
Gewichtung: 5

4.2 Evaluierung

Auflistung der möglichen Bauteile

- **Adafruit, Neopixel**
<https://www.adafruit.com/neopixel>
LED-Pixel in unzähligen Ausführungen.
Sitz der Firma in Tampa, Florida, USA
RGB: Chip ist der WS2801, <http://www.adafruit.com/datasheets/WS2801.pdf> ->
Hat volle Abdeckung des RGB-Farbraums
Ansteuerung: Findet über PWM-Pin des Raspberry Pi statt.
Framework: Framework von Adafruit, welches eine sehr leichte Ansteuerung ermöglichen soll.
Kosten: 4 LEDs 7\$, 25 LEDs zusammen 39\$, durch Lieferung aus USA sehr hohe Versandkosten (50\$)
Extras: Händler bietet verschiedene Formen und fertige Ketten an.

¹Pulsweitenmodulation: Signalübertragung durch Wechsel zwischen zwei Spannungen (High, Low), Breite des Impulses ist das Signal

- **LED-Emotion GMBH, LED Streifen**

<http://www.led-emotion.de/de/LED-Streifen-Set.html>

LED-Streifen, keine Einzelpixel, nur mit Controller, keine API

RGB: Voller RGB-Farbraum

Ansteuerung: Nur mit Controller

Framework: Keine öffentliche Api, möglicherweise mit Raspberry Pi ansteuerbar

Kosten: 30 LEDs mit Netzteil 79€

Extras: keine

- **DMX4ALL GmbH, MagiarLED Solutions**

<http://www.dmx4all.de/magiar.html>

Spezialisiert auf DMX-Ansteuerung, keine öffentliche API

RGB: Volle Abdeckung RGB-Farbraum

Ansteuerung: Wird über DMX-Controller angesteuert, dieser setzt die Signale um.

Framework: DMX-Ansteuerung über DMX-Controller

Kosten: Streifen mit 72 LEDs = 99€

Extras: viele verschiedene Varianten

- **TinkerForge, RGB LED-Pixel**

<https://www.tinkerforge.com/de/shop/accessories/leds.html>

Scheinen die gleichen wie von Adafruit zu sein, allerdings werden hauptsächlich Controller im Shop angeboten

RGB: Chip WS2801, volle Abdeckung RGB-Farbraum

Ansteuerung: Nach Anfrage an den Anbieter sollen die LEDs baugleich zu denen von Adafruit sein.

Framework: keins, aber Ansteuerung über das Framework von Adafruit

Kosten: 50 LEDs = 59€

Extras: Lieferung aus Deutschland

	Neopixel von Adafruit	LED Streifen von LED-Emotion GmbH	MagiarLED Solutions, DMX4ALL GmbH	RGB LED-Pixel von TinkerForge
RGB-Farbraum (5)	5	5	5	5
Ansteuerung (10)	10	3	3	8
Framework (10)	10	0	0	5
Kosten (5)	0	3	3	5
Extras (5)	1	0	2	1
Summe	26	11	13	24

Abbildung 1: Ergebnisse der LED-Evaluierung

Fazit: In der Evaluierung schneiden die Produkte von Adafruit und TinkerForge am besten ab. Für eine erste Teststellung werden die einzelnen LED-Pixel von Adafruit aus den USA bestellt (Neopixel). An diesen soll vor allem die Ansteuerung getestet werden. Falls sie sich bewähren, wird für den endgültigen Aufbau auf die LED-Ketten von Tinkerforge zurück gegriffen.

4.3 Teststellung

Für einen ersten Test wurde das in 4.2 ausgewählte Produkt als einzelne Pixel bestellt. Der Hersteller Adafruit bietet hier 4er-Packungen an. Diese können leicht in eigene Schaltungen eingelötet oder auf Experimentier-Boards gesteckt werden. Bei geringer Anzahl LEDs reicht die 5V-Stromversorgung des Raspberry Pi aus.

Technische Daten Neopixel:

- Maße: 10.2mm x 12.7mm x 2.5mm
- Protokollgeschwindigkeit: 800 kHz
- Spannung: 5-9VDC (bei 3,5V gedimmte Helligkeit)
- Strom: 18,5mA / LED, 55mA / Pixel

Framework:

- RPI_WS281X (https://github.com/jgarff/rpi_ws281x)
- Sprache: Python
- Entwickelt für Raspberry Pi
- Voraussetzung: Python 2.7

Ablauf des Tests:

- **Aufbau der Schaltung**

An die einzelnen LED-Pixel wurden Stecker angelötet, damit sie auf das Experimentierboard aufgesteckt werden können. Dann wird die Schaltung nach folgendem Schaltbild verbunden. Wichtig ist, dass beim Raspberry Pi nur Pins verwendet werden können, welche PWM bieten.

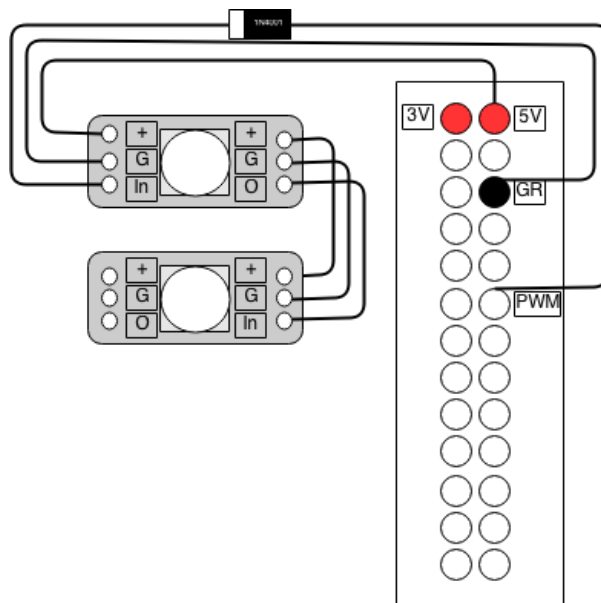


Abbildung 2: Schaltung für LED-Test

- **Installation des Frameworks**

```
1 wget https://github.com/tdicola/rpi_ws281x/raw/master/python/dist/rpi_ws281x
  -1.0.0-py2.7-linux-armv6l.egg
2 sudo easy_install rpi_ws281x-1.0.0-py2.7-linux-armv6l.egg
```

Listing 1: Installation Framework ws281x

- **Testcode**

Der dargestellte Testcode erzeugt ein NeoPixel-Objekt mit den angegebenen Eigenschaften. Im Anschluss werden alle vorhandenen Pixel mit Farbe angesteuert (im RGB-Format).

```
1 from neopixel import *
2
3 LED_COUNT = 4 # Number of LED pixels.
4 LED_PIN = 18 # GPIO pin connected to the pixels (must support PWM!).
5 LED_FREQ_HZ = 800000 # LED signal frequency in hertz (usually 800khz)
6 LED_DMA = 5 # DMA channel to use for generating signal (try 5)
7 LED_INVERT = False # True to invert the signal (when using NPN)
8
9 strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA,
  LED_INVERT)
10
11 strip.begin()
12 strip.setPixelColor(0, Color(255, 255, 255))
13 strip.setPixelColor(1, Color(255, 255, 255))
14 strip.setPixelColor(2, Color(255, 255, 255))
15 strip.setPixelColor(3, Color(255, 255, 255))
16 strip.show()
```

Listing 2: Testcode zur Ansteuerung der LEDs

Fazit Die einzelnen Pixel sind sehr leicht anzusteuern, unterstützen auch das automatische Abschalten nach einer bestimmten Zeit und haben eine sehr hohe Leuchtkraft. Die Evaluierung hat zu einer guten Produktwahl geführt.

Nach einer weiteren Nachfrage an Tinkerforge wurde versichert, dass deren LED-Ketten Baugleich zu denen von Adafruit sind. Aufgrund der hohen Versandkosten werden für die endgültige Teststellung die Produkte von Tinkerforge gewählt. XX Quelle Email !!!!

5 Bewegungssensor

In einem der Modi soll die Beleuchtung durch den Bewegungsmelder ausgelöst werden. Hierfür sind zuverlässige und weitreichende Bewegungssensoren notwendig.

5.1 Bewertungskriterien

- **Ansteuerung**
Die Anbindung an den Raspberry Pi soll möglichst leicht realisierbar sein. Wünschenswert ist, dass der Sensor einfach ein High-Signal bei Bewegungserkennung ausgibt.
Gewichtung: 5, KO-Kriterium
- **Reichweite**
Die Reichweite oder Sensivität des Sensors soll ausreichend und regelbar sein.
Gewichtung: 3
- **Kosten**
Es werden nur die reinen Produktkosten, also ohne Versand und Zoll, bewertet.
Gewichtung: 1
- **Extras**
An dieser Stelle können mögliche Extras eines Herstellers einfließen.
Gewichtung: 3

5.2 Evaluierung

- **PIR (MOTION) Sensor, Adafruit**
Link: <http://www.adafruit.com/product/189>
Ansteuerung: Gibt High-Signal an einem Pin aus.
Reichweite: 7m, 120 Grad
Kosten: 9,95\$ + Versand aus USA
Extras: Kabel inklusive
- **PIR Infrared Motion Sensor (HC-SR501)**
Link: <https://www.modmypi.com/pir-motion-sensor>
Ansteuerung: Gibt High-Signal an einem Pin aus.
Reichweite: 5-7m, 100 Grad
Kosten: 2,99\$ + Versand aus UK
Extras: keine
- **Infrarot PIR Bewegung Sensor Detektor Modul**
Link: http://www.amazon.de/Pyroelectrische-Infrarot-Bewegung-Sensor-Detektor/dp/B008AESDSY/ref=pd_cp_ce_0
Ansteuerung: Gibt High-Signal an einem Pin aus.
Reichweite: 7m, 100 Grad
Kosten: 5 Stück = 7,66€
Extras: keine

	PIR (MOTION) Sensor Adafruit	PIR Infrared Motion Sensor (HC-SR501)	Neopixel von AdafruitInfrarot PIR Bew. Sens. Detekt. Modul
Ansteuerung (10)	10	10	10
Reichweite (3)	3	1	3
Kosten (1)	0	0	1
Extras (3)	1	0	0
Summe	14	11	14

Abbildung 3: Ergebnisse der Motion-Sensor-Evaluierung

Fazit Die meisten Infrarot-Bewegungssensoren sind von der Bauweise nahezu identisch. Die Unterschiede liegen meist nur in der Empfindlichkeit. Da die Reichweite in diesem Fall nicht von großer Bedeutsamkeit ist, kann eigentlich jedes der Produkte bestellt werden. Auf Ebay und Amazon ist die Anzahl angebotener Sensoren nahezu unbegrenzt, es wurde für die Teststellung also die oben evaluierte Variante von Amazon bestellt.

5.3 Teststellung

Der in Punkt X.X.X gewählte Bewegungssensor wurde beim Hersteller bestellt. In der Teststellung reicht die Stromversorgung des Raspberry Pi.

Technische Daten Sensor:

- Die Empfindlichkeit und Haltezeit kann eingestellt werden
- Reichweite: ca. 7m
- Winkel: 100 Grad
- Spannung: DC 4,5V- 20V
- Strom: $\leq 50\mu A$
- Ausgangsspannung: High 3V / Low 0V
- Größe: ca. 32mm x 24mm

Ablauf des Tests:

- **Aufbau der Schaltung**

Der Sensor wird in der Teststellung direkt vom Raspberry Pi mit Strom versorgt. Für die Datenleitung kann jeder beliebige Pin gewählt werden.

- **Testcode**

Um eine Änderung am Datenpin festzustellen werden zwei Variable angelegt: `current_status` und `previous_status`. Das Programm wird in einer Dauerschleife geschickt, in der bei jedem Durchlauf die beiden Status überprüft. Wenn der neue Status (`current_status`) High ist und das vorherige Signal (`previous_state`) Low, dann wird eine Bewegung erkannt. Der Code wird mittels Kommentare erklärt.

```

1 import RPi.GPIO as GPIO
2 import time
3 GPIO.setmode(GPIO.BCM)
4 # Pin definieren

```

```
5 | MOTION_PIN1 = 7
6 | # Diese als Input definieren
7 | GPIO.setup(MOTION_PIN1,GPIO.IN)
8 | # Status definieren um verschiedene Änderungen zu erkennen
9 | Current_State = 0
10 | Previous_State = 0
11 |
12 | try:
13 |     # Loop zur Erkennung einer Bewegung
14 |     # Sensor erkennt Bewegung -> Signal = High
15 |     # Wartet 3 Sekunden und setzt Signal = Low
16 |     while True :
17 |         Current_State = GPIO.input(MOTION_PIN1)
18 |         if Current_State == 1 and Previous_State == 0:
19 |             print "Motion_detected!"
20 |             Previous_State=1
21 |         elif Current_State == 0 and Previous_State == 1:
22 |             print "Ready"
23 |             Previous_State=0
24 |             time.sleep(0.01)
25 |
26 | except KeyboardInterrupt:
27 |     print "Quit"
28 |     GPIO.cleanup()
```

Listing 3: Testcode zur Bewegungserkennung mit Sensor

Bei der Endversion des Systems sollen mehrere Bewegungssensoren integriert werden. Bei Auslösen des ersten Sensors sollen die LEDs angeschaltet werden und nach auslösen eines weiteren Sensors wieder ausgeschaltet werden.

Auswertung Das High-Signal des Sensors lässt sich mit dem Raspberry Pi sehr leicht auswerten. Auch die Auswertung von mehreren Sensoren stellt kein Problem da. Das Ergebnis der Evaluierung konnte in dieser Teststellung bestätigt werden. Wichtig für die weitere Implementierung ist, dass die While-Schleife auch abgebrochen werden kann.

6 Kamera

6.1 PI-Kamera vs. Netzwerkkamera

Bei Auslösen des System im Überwachungsmodus soll ein aktuelles Bild der Überwachungskamera an das jeweilige Smartphone gepusht werden. Es gibt zwei mögliche Kameratechniken, entweder direkt mit dem Raspberry Pi verbunden oder über das Netzwerk erreichbar.

Raspberry Pi Cam Die Kameras für den Raspberry Pi können direkt an das Gerät angeschlossen werden. Meistens werden sie direkt über Erweiterungsplatinen mit den GPIO Pins verbunden. Der Vorteil dieser Anbindung ist, dass sie keine externe Stromversorgung benötigen und durch viele verschiedene Frameworks leicht ansteuerbar und verwaltbar sind. Der große Nachteil ist allerdings, dass die Kamera direkt an dem Raspberry Pi angeschlossen werden muss. Da dieser möglichst wettergeschützt (im Außenbereich) oder unauffällig (im Innenbereich) angebracht ist, lässt sich von diesen Positionen kaum eine effektive Videoüberwachung realisieren.

Netzwerkkamera Eine Netzwerkkamera oder auch IP-Kamera genannt befindet sich im Netzwerk und kann über eine Website oder App eingesehen und gesteuert werden. Der Vorteil ist, dass sie sich an einem beliebigen Ort befinden kann, solange sie im selben Netzwerk ist. Somit kann zum Beispiel eine wetterfeste Kamera im Außenbereich angebracht werden und der Raspberry Pi kann im geschützten Innenbereich stehen. Der Nachteil bei IP-Kameras besteht darin, dass es wenige einheitliche APIs zum Abgreifen des Videomaterials gibt. Die meisten IP-Kameras bieten die Möglichkeit, die Aufgenommenen Bilder auf einem FTP-Server abzuladen. Weiter wäre eine mögliche Lösung das Laden der HTML-Code über einen HTTP-Request und darauffolgend das Ausfiltern des Bildmaterials. Über diese Variante kann kein Video sondern nur statische Bilder geladen werden. Eine weitere Möglichkeit ist das Verwenden eines Videostreams. Dieser könnte mit dem Raspberry Pi ausgewertet und in Bilder umgewandelt werden.

Da in diesem Projekt nicht garantiert ist, dass der Raspberry Pi an einer passenden Position angebracht ist wird für dieses Projekt eine IP-Kamera verwendet. Dies trägt außerdem zur universellen Einsatzbarkeit bei.

6.2 Herangehensweise Ansteuerung

Aus Recherche und Überlegung haben sich die folgenden Möglichkeiten ergeben:

Auswertung des RTSP-Protokolls mit Python Die meisten Netzwerkkameras bieten einen Video-Stream über das Real Time Streaming Protokoll (RTSP) an. Dieser könnte mit Python ausgelesen und interpretiert werden.

Speicherung des Video-Streams mit FFmpeg Außerdem ist es möglich mit dem Tool FFmpeg den Stream auszuwerten und zu bearbeiten.

Speicherung des Bild mittels Screenshot oder HTTP-Request Der erste und theoretisch einfachste Ansatz ist die Anfertigung eines Screenshots des Kamerabildes oder das Abrufen des Bildes mit einem HTTP-Request.

6.3 Auswertung des RTSP/RTP-Protokolls mit Python

Das Real-Time-Streaming-Protokoll ist ein Netzwerkprotokoll zur Steuerung von kontinuierlichen Übertragungen in Netzwerken. Dagegen werden über das Real-Time-Protokoll die tatsächlichen Video- und Tondaten übertragen. So werden mit RTSP die Übertragungsdetails festgelegt um im Anschluss mit RTP die tatsächlichen Daten zu übertragen.

RTSP Handshake Der Handshake zwischen Client und Server basiert auf Strings und ist somit leicht auszuwerten. Zusätzlich ist der Ablauf sehr kurz und übersichtlich gehalten. Grundsätzlich lässt sich der Handshake in folgende Schritte aufteilen (Server = Kamera):

1. Der Client sendet eine Anfrage an den Server. Diese enthält seine IP und den Port.
2. Als Antwort beschreibt der Server seine Eigenschaften und Funktionen.
3. Daraufhin sendet der Client eine Nachricht in der er die Eigenschaften des Streams festlegt (Format, Ports etc).
4. Der Server bestätigt dies.
5. Der Client sendet ein 'Play' um die Übertragung zu starten.

Im Anschluss startet die Übertragung des Streams über RTP.

6.4 RTSP mit Python

Eine Mögliche Implementierung des Protokolls in Python hat Sergey Lalov im Jahr 2011 vorgenommen. An diesem Code habe ich Reengineering betrieben.

Es wurde ein Server mit dem Twisted Framework implementiert. Dieser empfängt und sendet die Daten der Netzwerkkamera. Das RTSP-Protokoll wird in den einzelnen Schritten abgearbeitet, wobei zu Beginn des Programms die IP, Portbereiche und Userdaten definiert werden. Die einzelnen Protokollschritte werden anhand der 'CSeq' (Identifier für die einzelnen Protokollschritte) identifiziert. Es wird ein Video- und ein Audio-Stream gestartet. Die einzelnen Server-Anfragen sind in Abbildung 4 dargestellt.

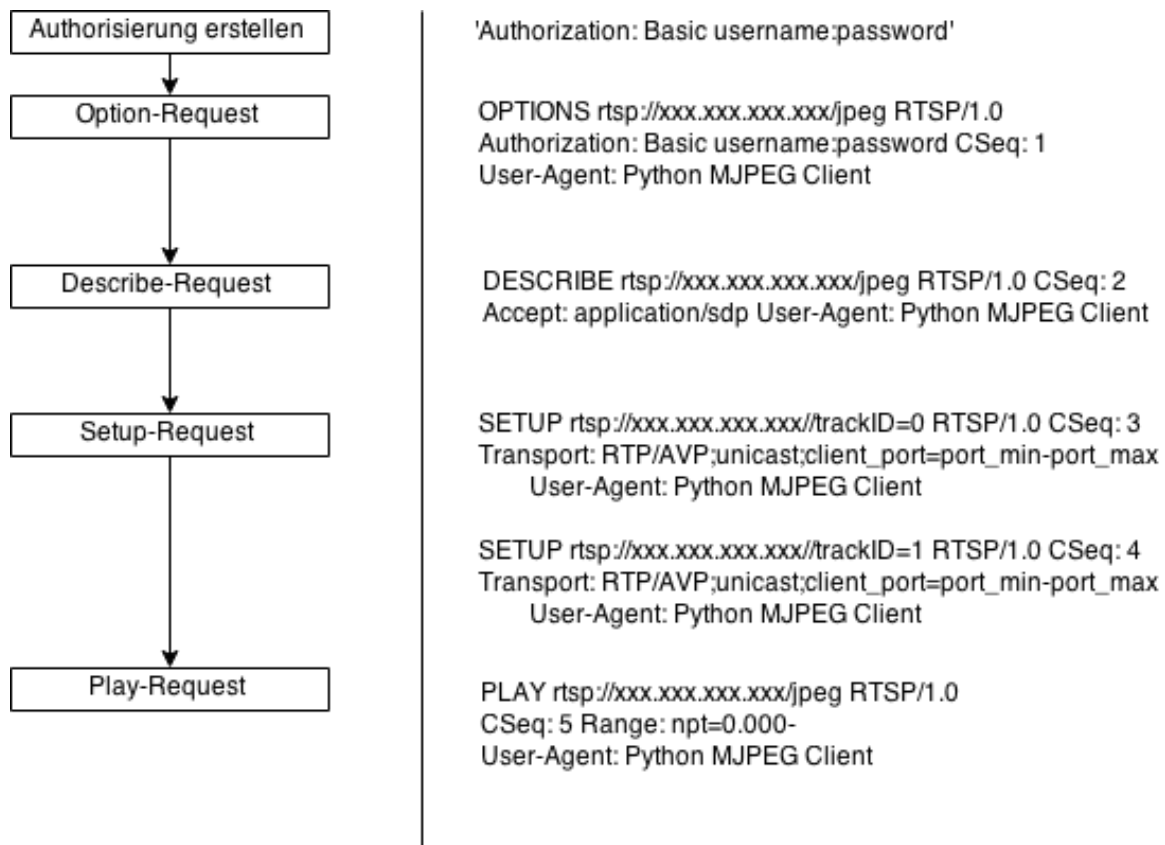


Abbildung 4: RTSP Request-Strings

Nach dem PLAY-Request werden zwei neue UDP-Server-Instanzen generiert, die Daten des RTP-Streams auswerten (Audio / Video). Die Implementierung von Sergey Lalov liest an dieser Stelle die einzelnen Bilder eines MJPEG-Stream aus. Bei MJPEG werden in jedem Paket Einzelbilder gesendet. Dies führt dazu, dass aus einem einzelnen Paket ein Bild gewonnen werden kann.

Die Kamera in diesem Projekt hat im RTP-Stream den Videostream im Format H264 ausgegeben. Da dieses Format verschiedene Arten der Komprimierung anwendet, ist nicht in jedem Paket ein einzelnes Bild enthalten. Die Rekonstruktion des Bildmaterials wäre somit sehr aufwändig (keine bestehende Library).

Die Auswertung des RTP-Streams konnte nicht erfolgreich abgeschlossen werden. Der Originalcode ist unter Google-Code verfügbar (<https://code.google.com/p/python-mjpeg-over-rtsp-client/>).

6.5 Speicherung des Video-Streams mit FFmpeg

Die Speicherung des Streams ist über eine externe Software möglich. Das Tool FFmpeg ist eine Software-Lösung um Audio- und Videostreams aufzunehmen und zu konvertieren. Es bietet die Möglichkeit, einen Stream aufzuzeichnen und direkt im Anschluss in ein anderes Format zu Wandeln.

Ablauf Im ersten Schritt wird mit FFmpeg der Stream eingelesen und pro Sekunde 3 Bilder daraus erzeugt.

```
ffmpeg -i rtsp://user:pw@$ip:554 -f image2 -vf fps=3 %03d.jpg -loglevel quiet
```

Listing 4: Aufnahme mit FFmpeg

Diese Bilder werden in ein Verzeichnis gespeichert. Parallel dazu läuft ein Script, welches alle 60 Sekunden die ältesten Bilder aus diesem Verzeichnis löscht. Dadurch entsteht keine große Datenmenge, es sind nur die aktuellsten Bilder gespeichert. Wenn der Bewegungsmelder auslöst, werden die Bilder aus den letzten 10 Sekunden in ein weiteres Verzeichnis gelegt. Hierauf hat die iOS-App Zugriff über FTP.

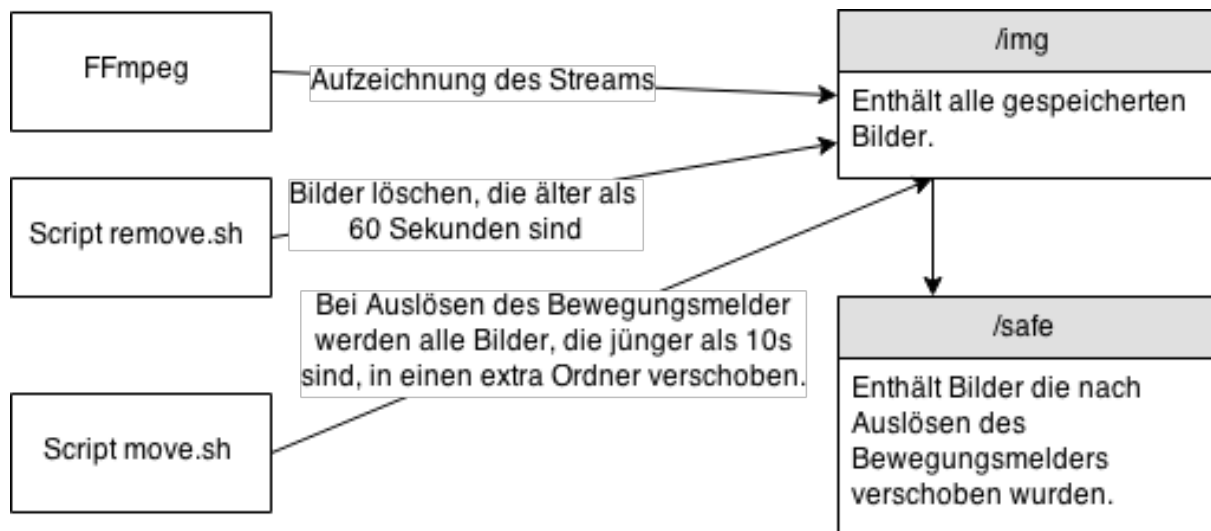


Abbildung 5: Ablauf der Aufzeichnung mit FFmpeg

Vorteil Ein großer Vorteil ist die hohe Variabilität der Konvertierung mit FFmpeg. Es kann nahezu jedes Format eingelesen werden und in variablen Frameraten ausgegeben werden.

Nachteil Das Tool FFmpeg erzeugt eine sehr hohe Last auf dem System, da kontinuierlich eine Stream eingelesen wird.

6.6 Speicherung des Bild mittels Screenshot

Ein weiterer Ansatz wäre das Aufrufen der Weboberfläche der Kamera und das Anfertigen eines Screenshots. Dieser könnte abgespeichert und auf einem FTP-Server bereitgestellt werden.

Hierfür ist es allerdings erforderlich, dass der Raspberry Pi eine grafische Ausgabe hat. In diesem Projekt soll er nur Serverfunktionalitäten haben. Als Work-Around bietet sich ein virtuelles Display an. Die Virtualisierung kann mittels dem Tool Xvfb realisiert werden. Dieses bietet einen virtuellen Framebuffer, der vom System wie ein normales Display angesprochen werden kann.

Die Anfertigung eines Screenshots kann mit dem Tool Selenium erledigt werden. Das Tool Selenium dient zur Automatisierung von Browser-Aktivitäten und wird oft zum Testen von Webanwendungen verwendet. Selenium setzt ein Display voraus, was durch Xvfb

erfüllt wäre. Da Selenium nach Python portiert wurde, könnten die Screenshots direkt aus der Anwendung heraus erstellt werden.

Um diese Möglichkeit zu testen, wurde folgender Testcode implementiert:

```
1 from selenium import webdriver
2 import time
3
4 start = time.time()
5 driver = webdriver.Firefox()
6 driver.get('user:password@ip_of_cam')
7 // get_screenshot_as_file() gibt das Bild als Binärdaten zurück
8 img = driver.get_screenshot_as_file()
9 driver.quit()
10 end = time.time()
11 print 'Benötigte_Zeit:' + str(start - end)
```

Listing 5: Testcode - Aufnahme Screenshot mit Selenium

Mehrere Messungen haben ergeben, dass das Aufnehmen des Screenshots rund 7 Sekunden dauert. Dies ist auf die geringe Rechenleistung des Raspberry Pi zurückzuführen. Außerdem wird mit Selenium ein vollständiger Browser gestartet, was zu einem enormen Overhead führt.

Die Durchführung dieser Methode macht aufgrund der hohen Verzögerung keinen Sinn. Eine Person die den Bewegungssensor auslöst ist längst aus dem Bild verschwunden, bis ein Screenshot aufgezeichnet wird.

6.7 Speicherung des Bild mittels HTTP-Request

Die nächste Variante ist das Laden des Bildes aus dem HTML-Quelltext der Weboberfläche. Die Analyse des Quelltextes führt schnell zu dem Ergebnis, dass das angezeigte Bild mittels Javascript nachgeladen wird. Mit Python kann ohne weitere Probleme der Quelltext einer Webseite abgerufen werden:

```
1 import urllib2
2
3 response = urllib2.urlopen('http://ip_of_cam')
4 html = response.read()
```

Listing 6: Testcode - Aufnahme Screenshot mit Selenium

Hier offenbart sich direkt das Problem, denn mit der Library urllib2 wird nur der HTML-Code abgerufen, ohne auf die Ausführung von Javascript zu warten. Somit kann über den Code zwar das Image-Objekt identifiziert werden, es enthält allerdings nur das Startbild. Mit dieser Version ist keine Lösung erreichbar.

Somit wird ein Framework benötigt, welches zuerst auf die vollständige Ausführen des Javascript-Codes wartet.

- **Selenium** Hier könnte ebenfalls wieder Selenium eingesetzt werden, welches einen Export des HTML-Code bietet. Allerdings würde auch hier der Zeitfaktor zu hoch sein.

- **Scrapy** Dies ist ein open-source Framework, mit dem alle Informationen von Webseiten geladen werden können. Die Installation auf dem Raspberry Pi war nicht erfolgreich.

Im nächsten Schritt wurde ein HTTP-Request an die URL gesendet, der im Javascript-Code als erstes aufgerufen wurde. Die Antwort enthielt ein aktuelles Frame der Kamera im Jpeg-Format. Das Abrufen dieses Bildes stellt in den meisten Sprachen keine große Schwierigkeit dar.

In Python wird das Bild mit dem Framework Requests und StringIO geladen und in ein Bild umgewandelt. Dieses kann beliebig verarbeitet werden.

```
1 from PIL import Image
2 import requests
3 from StringIO import StringIO
4
5 r = requests.get('http://user:password@ip:80/tmpfs/auto.jpg')
6 i = Image.open(StringIO(r.content))
7 i.save("path/to/save/python_test.jpg")
```

Listing 7: Abrufen eines Bildes von einer URL in Python

Auch in Swift stellt dies kein Problem dar. Die Daten werden von der URL geladen und in ein UIImage-Objekt gespeichert.

```
1 let url = NSURL(string: camurl)
2 let data = NSData(contentsOfURL: url!)
3 let image : UIImage! = UIImage(data: data!)
```

Listing 8: Abrufen eines Bildes von einer URL in Swift

Im Gegensatz zu der Implementierung mit FFmpeg muss nicht kontinuierlich ein Stream ausgelesen werden, sondern es wird immer nur ein einzelnes Bild ausgelesen, sobald der Bewegungsmelder auslöst. Es lassen sich somit zwei Ansichten realisieren:

- **Archiv:** Bei Auslösen des Bewegungsmelders wird das aktuelle Bild abgerufen und auf einen FTP-Server gespeichert. Die Daten dieses Servers können in der iOS-App unter 'Archiv' abgerufen werden.
- **Live:** In der iOS-App ist ein Live-View möglich. Dieser liest kein Video-Stream ein, sondern ruft in festgelegten Schritten ein Bild ab und aktualisiert die Ansicht. Um das Bild regelmäßig neu zu laden, wird ein NSTimer genutzt.

```
var timer = NSTimer.scheduledTimerWithTimeInterval(0.5, target: self, selector:  
    Selector("loadImageView"), userInfo: nil, repeats: true)
```

Listing 9: NSTimer in Swift

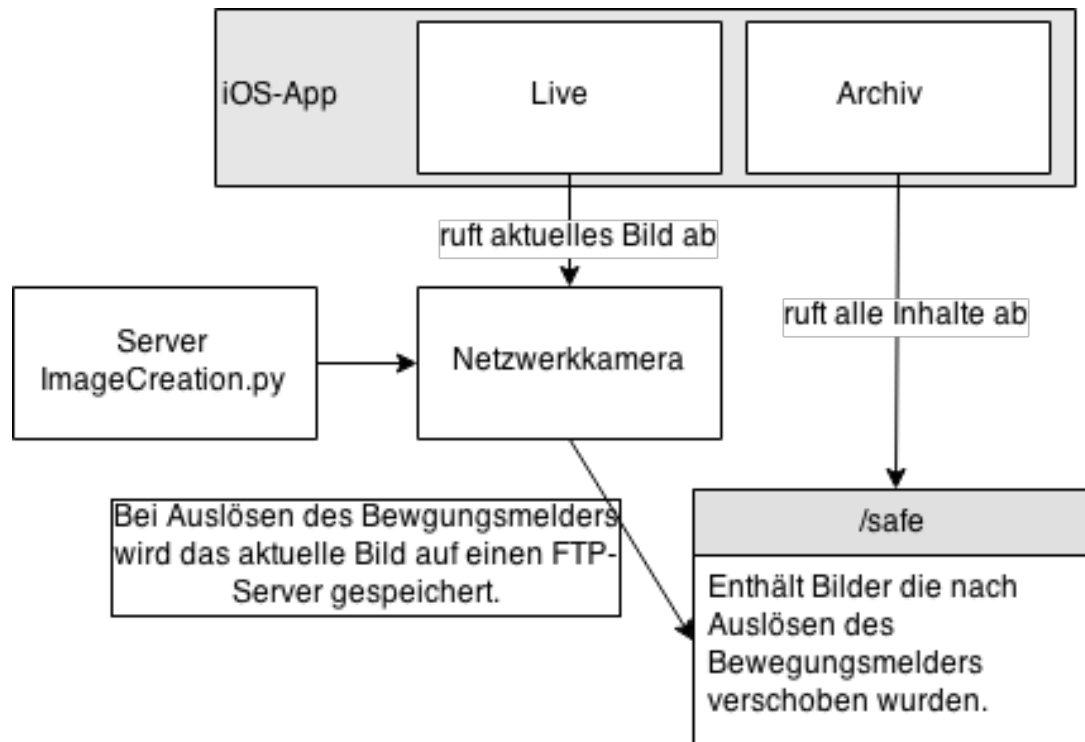


Abbildung 6: Abrufen des Bildes über HTTP-Request

Fazit Das Speichern des Kamerabildes mittels Screenshot und RTSP-Protokoll sind fehlgeschlagen. Die Aufnahme mit FFmpeg funktioniert gut, benötigt allerdings sehr viele System-Ressourcen. Somit ist der gewählte Weg in diesem Projekt das Abspeichern des Bildes über einen HTTP-Request. Diese Lösung ist gut in Python und Swift umzusetzen.

7 Verschlüsselung

7.1 SSL vs. TLS

SSL (Secure Sockets Layer) und TLS (Transport Layer Security) sind Protokolle, die Verschlüsselung und Authentifizierung zwischen zwei Kommunikationspartnern bieten. Die beiden Begriffe SSL und TLS werden umgangssprachlich oft als zwei verschiedene Techniken dargestellt, obwohl TLS eine Weiterentwicklung von SSL ist. SSL v3 ist die Basis von TLS 1.0.

Aufgrund des Alters und einiger Sicherheitslücken wird SSL als unsicher angesehen und soll nicht mehr verwendet werden. Die aktuellste gefundene Lücke ist POODLE², welche das Auslesen von Informationen aus einer verschlüsselten Übertragung erlaubt. Die Weiterentwicklungen TLS 1.1 und 1.2 sind deutlich sicherer und beheben einige Sicherheitslücken. So schützt die richtige Implementierung von TLS 1.2 auch vor den BEAST³ Angriffsmethoden.

7.2 Vor- und Nachteile TLS

Da TLS auf der Transportschicht aufsetzt kann jedes höhere Protokoll darüber übertragen werden, somit ist die Verschlüsselung unabhängig von der genutzten Anwendung.

Der größte Nachteil besteht darin, dass der Verbindungsaufbau serverseitig sehr rechenintensiv ist. Die Verschlüsselung selbst nimmt, abhängig vom Algorithmus, nur noch wenig Rechenleistung in Anspruch.

7.3 TLS Handshake

1. Client Hello

Übertragung von Verschlüsselungsinformationen vom Client an den Server, wie TLS Version oder Verschlüsselungsmöglichkeiten

2. Server Hello

Server sendet seine Informationen und legt Verschlüsselung fest.

3. Server Key Exchange

Server sendet seine Identität in Form seines Zertifikats.

4. Client Key Exchange

Client legt seinen Pre-Shared-Key fest und überträgt ihn verschlüsselt mit dem public Key des Servers.

5. Change Cipher Spec

Aus dem PSK wird ein Master-Secret generiert, mit welchem die folgenden Übertragung abgesichert wird.

6. Application Data

Übertragung der Daten.

²Sicherheitslücke in SSL v3

³Sicherheitslücke in TLS v1.0

7.4 StartTLS

Eine Variante von TLS ist das sogenannte STARTTLS, bei dem zuerst ein unsicheres 'hello' an den Server gesendet wird. Falls im Anschluss eine Verbindung erfolgreich zustande kommt, wird zur sicheren Übertragung gewechselt.

Im ersten Versuch der Server-Client-Kommunikation in diesem Projekt wurden die Daten direkt zwischen Sockets übertragen. Im folgenden Ausschnitt ist der Wechsel zur Verschlüsselung sehr gut erkennbar:

```
1 if line == "STARTTLS":
2     print "--Switching to TLS"
3     self.sendLine('READY')
4     ctx = ServerTLSContext(
5         privateKeyFileName='./certs/server.key',
6         certificateFileName='./certs/server.crt',
7     )
8     self.transport.startTLS(ctx, self.factory)
```

Listing 10: Starttls - Wechsel zur Verschlüsselung

Der vollständige Code ist im Commit unter <https://github.com/hoedding/Studienarbeit-Anwendung/commit/b52d056f55a9d65b9115ead2d2a2c0a549b366b6> zu finden

7.5 Server Zertifikat

Für die Verschlüsselung der Übertragung zwischen Server und Client ist ein Server-Zertifikat notwendig. Dieses wird mit OpenSSL in der neuesten Version generiert.

1. Private Key erzeugen

```
1 openssl genrsa -des3 -out server.key 2048
```

Listing 11: private Key

2. Certificate Signing Request

```
openssl req -new -key server.key -out server.csr
```

Listing 12: Certificate Signing Request

3. Self Signed Certificate

Bei einem öffentlichen Server sollte das Zertifikat bei einer CA (Certificate Authority) signiert werden.

```
openssl x509 -req -days 1865 -in server.csr -signkey server.key -out server.crt
```

Listing 13: Self Signed Certificate

7.6 Apple Zertifikat

Um die Apple Push Notifications einzusetzen ist ein Apple-Developer Zertifikat notwendig, welches nur über das Apple-Developer Portal erzeugt werden kann. Hierfür muss eine App-Identität angelegt werden. Im Anschluss wird auf einem Apple-Gerät ein Signing-Request erstellt. Dieser wird in das Developer-Portal hochgeladen. Im Anschluss wird ein

Zertifikat generiert. Dieses muss in das Verzeichnis 'certs' gelegt werden.
Zusätzlich ist auf dem System ein Apple-Root Zertifikat erforderlich.

Die Generierung von Zertifikaten im Developer-Portal ist nur mit gültiger Apple-Developer-Registrierung möglich.

7.7 Wireshark Trace

Im folgenden ist ein Trace eines TLS Handshakes zwischen einem Client und dem implementierten Server (mit Twisted Framework) auf dem Raspberry Pi zu sehen.
Die einzelnen Schritte des Handshakes sind sehr gut erkennbar.

No.	Time	Source	Protocol	Length	Info	Destination
12	10.667623000	127.0.0.1	TLSv1	156	Client Hello	127.0.0.1
14	10.667932000	127.0.0.1	TLSv1	1038	Server Hello, Certificate, Server Hello Done	127.0.0.1
16	10.668541000	127.0.0.1	TLSv1	382	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message	127.0.0.1
18	10.673457000	127.0.0.1	TLSv1	290	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message	127.0.0.1
20	10.673777000	127.0.0.1	TLSv1	289	Application Data, Application Data, Application Data, Application Data	127.0.0.1

Abbildung 7: Wireshark Trace TLS Handshake

8 Python-Server und Protokoll

8.1 Protokoll

Um die LEDs später von einer App aus ansprechen zu können, soll ein auf Strings basierendes Protokoll implementiert werden. Hierfür muss als erstes festgelegt werden, welche Informationen übertragen werden sollen:

- Authentifizierung
Übertragung eines Benutzers und eines Passworts. Das Passwort ist als Hashwert im System gespeichert und kann so überprüft werden. Zum Hashen wird der SHA-224-Algorithmus eingesetzt.
- Control
Unterscheidung zwischen:
 - X00: Alle LEDs ausschalten
 - X01: Eine LED anschalten
 - X02: LED-Bereich anschalten
 - X03: Alle LEDs in einer Farbe anschalten
 - X04: Effekte
 - X05: Modus des Systems verändern
 - X06: Anforderung des Systemstatus
 - X07: Anforderung des LED-Status
 - X08: Konfiguration ändern
 - X09: Login überprüfen

Abhängig von diesem Feld werden die nachfolgenden Werte behandelt.

- LED-Nummer
Falls nur eine LED angesprochen werden soll (Control = X00), so wird hier die Nummer angegeben. Ob sie im gültigen Range liegt wird intern überprüft.
- Bereich Start
Wenn mehrere LEDs gesteuert werden sollen (Control = X01), so wird hier der Beginn des Bereichs angegeben.
- Bereich Ende
Und hier das Ende des Bereichs.
- Rot
Farbwert Rot 0-255
- Grün
Farbwert Grün 0-255
- Blau
Farbwert Blau 0-255
- Modus
An dieser Stelle werden die verschiedenen Modi des Systems dargestellt.

- Effektcode
Hinterlegte, fest programmierte Effekte, zum Beispiel alle LEDs anschalten in weiss mit höchster Leuchstärke.
- Konfiguration
Damit können einzelne Elemente der Serverkonfiguration verändert werden. Zum Beispiel die Leuchtdauer der LEDs, wenn sie durch den Bewegungsmelder ausgelöst wurden.
- Hash
Überprüfung ob die Übertragung erfolgreich war, mittels eines Hashwertes. Es wird der SHA-224-Algorithmus eingesetzt.

Übertragungsbeispiel:

```
user:pw:control:ledNo:rangeStart:rangeEnd:red:green:blue:modus:effectcode:config:hashv  
user:password:X01:::49:255:255:255:::Hashvalue
```

Listing 14: Beispielübertragung des Protokolls

Dies würde die LEDs 0 bis 49 einschalten (Farbe weiss 255,255,255). Anstelle des 'Hash-value' würde der Hashwert der gesamten Übertragung gesendet.

8.2 Server-Framework

Twisted: <https://twistedmatrix.com>

Es wird das Twisted Matrix Framework eingesetzt. Twisted ist eine in Python geschriebene event-getriebene Netzwerkengine. Die meisten gängigen Protokolle wie TCP, IMAP, SSHv3 und viele mehr werden unterstützt. Somit bietet Twisted die ideale Möglichkeit einen eigenen Webserver zu implementieren.

Event-Getrieben (event-based): Die Serveranwendung befindet sich in einer Schleife und wartet auf ein Event. Dieses Event ist in diesem Fall der Connect eines Clients zum Server. Für jeden Connect wird eine neue Instanz angelegt, in welcher empfangene Daten bearbeitet werden können. Die Daten werden als String ausgewertet.

8.3 Beispielimplementierung Webserver

Im Folgenden wird die grundlegende Implementierung eines Webservers mit Twisted gezeigt. Für die einzelnen Funktionen des HTTP-Protokolls werden Methoden deklariert. In diesem Fall wird noch ein SSL-Kontext erzeugt, welcher die Zertifikate einliest und validiert und dafür sorgt, dass die Übertragung verschlüsselt wird.

Der gezeigte Server empfängt Daten vom Client und sendet sie direkt als Antwort zurück.

```
1 from twisted.web.server import Site  
2 from twisted.web.resource import Resource  
3 from twisted.internet import reactor  
4 import cgi  
5 from twisted.internet.protocol import Factory, Protocol  
6 from twisted.internet import reactor  
7
```

```
8 class Webserver(Resource):
9     def render_POST(self, request):
10         print cgi.escape(request.args["data"][0]))
11
12     def render_GET(self, request):
13         print cgi.escape(request.args["data"][0]))
14
15 root = Resource()
16 root.putChild("serv", Webserver())
17 factory = Site(root)
18 sslContext = ssl.DefaultOpenSSLContextFactory(
19     './certs/server.key', './certs/server.crt'
20 )
21 reactor.listenSSL(8000, factory, contextFactory = sslContext)
```

Listing 15: Testcode Echoserver mit Twisted Framework

8.4 Implementierung Webserver

Der Server wird in einem neuen Thread gestartet, damit er beim Empfang von Daten keine anderen Abläufe aufhält. Zusätzlich wird ihm eine Instanz der Klasse 'RecvData' übergeben. Diese verarbeitet die empfangene Nachricht. Anhand der ':' werden die empfangenen Daten gesplittet und in ein Array abgelegt. Zur besseren Lesbarkeit werden die Werte in einzelne Variablen gespeichert.

Im Anschluss wird das Übertragene Passwort und die Korrektheit der Daten überprüft. Falls beides Korrekt ist, so werden die Daten anhand ihres "ControlFeldes ausgewertet. Bevor tatsächlich LEDs angesteuert werden, wird überprüft ob die Farbwert im gültigen Bereich (0-255) liegen und ob die Angabe der LED-Nummer korrekt ist.

Einige Methoden haben Rückgabewerte, die an den Client gesendet werden müssen (zum Beispiel bei der Übertragung von Synchronisationsdaten). Dies geschieht in der Post-Methode (Z. 22).

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3 #####
4 # Author: Timo Höting #
5 # Mail: mail[at]timohoeting.de #
6 #####
7 from twisted.web.server import Site
8 from twisted.web.resource import Resource
9 from twisted.internet import reactor
10 import cgi
11 from twisted.internet.protocol import Factory, Protocol
12 from twisted.internet import reactor
13 import hashlib
14 import threading
15 from twisted.internet import reactor, ssl
16
17 class LightServer(Resource):
```

```
18 def render_POST(self, request):
19     message = datamanager.dataReceived(cgi.escape(request.args["data"][0]))
20     if (message != None):
21         return message
22
23 class StartLightServer(threading.Thread):
24     def __init__(self, d):
25         threading.Thread.__init__(self)
26     global datamanager
27     datamanager = d
28
29     def cleanup(self):
30         reactor.stop()
31
32     def join(self):
33         self.cleanup()
34         threading.Thread.join(self)
35
36     def run(self):
37         root = Resource()
38         root.putChild("serv", LightServer())
39         factory = Site(root)
40         sslContext = ssl.DefaultOpenSSLContextFactory(
41             './certs/server.key', './certs/server.crt'
42         )
43     global reactor
44     reactor.listenSSL(8000, factory, contextFactory = sslContext)
45     reactor.run(installSignalHandlers=False)
```

Listing 16: Implementierung des Webservers in Python

8.5 Hashfunktion

Es wird zu zweierlei Zwecken eine Hashfunktion eingesetzt. Zum einen um die Korrektheit der Übertragung zu überprüfen und zum Anderen um ein Passwort zur Authentifizierung verwenden zu können. Dieses wird als Wort übertragen, ist auf dem Server aber nur als Hash-Wert abgespeichert. Falls es jemand schafft die Konfigurationsdatei abzugreifen, so ist der Passworthash nichts wert.

Hash-Funktion Eine Hashfunktion ist eine Einwegfunktion die aus einer großen Eingabemenge, eine kleinere Zielmenge generiert. Die Ausgabe muss für die selbe Eingabe immer gleich sein. Jedoch soll bei der kleinsten Änderung der Eingabe, eine möglichst große Veränderung in der Ausgabe auftreten.

9 Anwendungsstruktur Serverapplikation

9.1 Klassen und ihre Funktionen

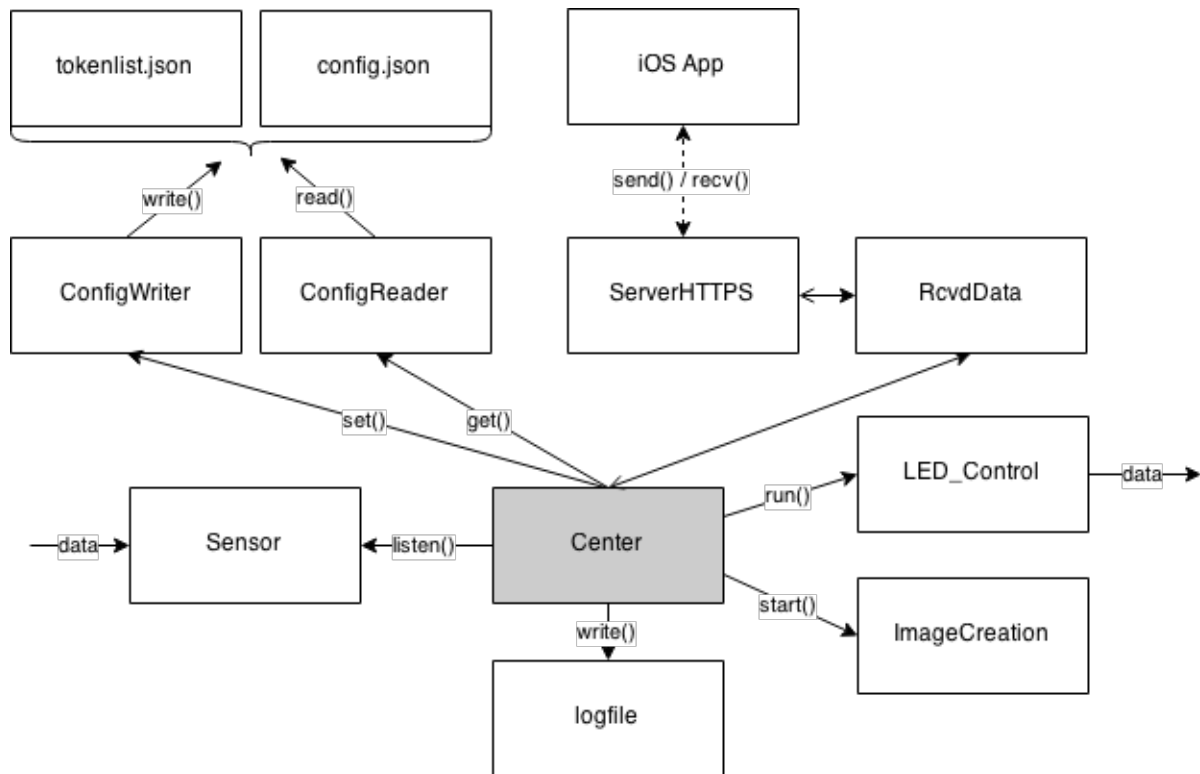


Abbildung 8: Anwendungsstruktur Server-Anwendung

- **Center**
Die Klasse 'Center' stellt die zentrale Stelle in der Anwendung dar, an der alle Informationen zusammen laufen und verwaltet werden.
- **ServerHTTPS**
Hier läuft der Webserver, welcher Nachrichten empfängt und sendet. Empfangene Nachrichten werden an die Klasse 'RcvdData' übergeben.
- **RcvdData**
Hier werden die empfangenen Nachrichten ausgewertet und entsprechende Antworten generiert. Diese werden an den Webserver zurück gegeben und abgesendet. Die Prüfung der Korrektheit der einzelnen Protokollbestandteile findet ebenfalls hier statt. Wenn alle Überprüfungen erfolgreich sind, werden die Befehle an 'Center' weiter gegeben und dort ausgeführt.
- **Sensor**
In der Klasse 'Sensor' werden die einzelnen Bewegungssensoren überwacht. Falls eine Bewegung detektiert wird, so werden in 'Center' die notwendigen Methoden aufgerufen um die LEDs an- oder auszuschalten.

- **LED-Control**
Die Klasse 'LED_Control' verwaltet die eingerichteten LEDs und steuert diese. Hier werden auch die möglichen Effekte gesteuert. Die Methoden in dieser Klasse werden aus der Klasse 'Center' aufgerufen. Ein Zugriff in die andere Richtung ist nicht möglich.
- **ConfigReader / ConfigWriter** Diese beiden Klassen bieten die Möglichkeit die Konfigurationsdatei config.json zu lesen und zu schreiben. In der Konfiguration werden Informationen wie Anzahl der LEDs, Passworthash oder Adresse der Netzwerkkamera abgespeichert. Die Konfigurationsdatei wird beim Installationsvorgang erstellt.
- **ImageCreation**
Abrufen von Bildmaterial von der Netzwerkkamera oder vom Server findet ausschließlich über die Klasse 'ImageCreation' statt. Die Klasse ruft die Informationen ab und filtert das Bildmaterial. Außerdem ist sie fähig, die gespeicherten Bilder zu Verwalten und das FTP-Verzeichnis zu mounten.
- **Logfile**
Im Logfile werden unter anderem auftretende Fehler gespeichert.

9.2 Auswertung empfangener Daten

Da die Daten anhand des ausgearbeiteten Protokolls übertragen werden, können sie als String sehr einfach an den ":" aufgesplittet werden. Im Anschluss werden sie einzelnen Variablen zugewiesen (bessere Lesbarkeit des Codes). Bevor das 'Control'-Feld ausgewertet wird, muss der Hashwert der Übertragung und die Authentifizierung geprüft werden.

RecvData.py Auswertung

```
1 def dataReceived(self, data):
2     # Protokoll: user:pw:control:ledNo:rangeStart:rangeEnd:red:green:blue:modus:
3     #           effectcode:config:hashv
4     # Beispiel: admin:w:X00:1:0:0:10:10:10:0:0:w-w:58
5     #           acb7accce58ffa8b953b12b5a7702bd42dae441c1ad85057fa70b
6     # Ermöglicht Zuweisung von Farben und Effekten
7     # Ermöglicht Abruf von aktuellem Status des Systems und der LEDs
8     #
9     # Ankommende String bei ":" aufsplitten und in Array a[] Speichern:
10    a = data.split(':')
11    print a
12    if len(a) > 1:
13        user = a[0]
14        pw = a[1]
15        control = a[2]
16        ledNo = a[3]
17        rangeStart = a[4]
18        rangeEnd = a[5]
19        red = a[6]
20        green = a[7]
21        blue = a[8]
```



```
20  modus = a[9]
21  effectcode = a[10]
22  config = a[11]
23  hashv = a[12]
24  data = user + pw + control + ledNo + rangeStart + rangeEnd + red + green + blue
    + modus + effectcode + config
25  data = data.rstrip('\n')
26  data = data.rstrip('\r')
27  if (self.checkAuthentication(user, pw) & self.checkTransmissionData(data, hashv)):
28      if control == 'X00':
29          ## Alle LEDs ausschalten
30          center.clearPixel()
31      elif control == 'X01':
32          ## Eine LED anschalten
33          self.lightUpOneLED(int(ledNo), int(red), int(green), int(blue))
34      elif control == 'X02':
35          ## LED Bereich anschalten
36          self.lightUpLEDRange(int(rangeStart), int(rangeEnd), int(red), int(green), int(
            blue))
37      elif control == 'X03':
38          ## Eine Farbe für alle LED
39          self.lightUpAllLED(int(red), int(green), int(blue))
40      elif control == 'X04':
41          ## Effekt alle LEDs
42          self.effectLED(effectcode)
43      elif control == 'X05':
44          ## Modus des Systems
45          self.changeModus(int(modus))
46      elif control == 'X06':
47          ## Systemstatus als JSON an den Client
48          return self.sendStatus()
49      elif control == 'X07':
50          ## Status der einzelnen LEDs senden
51          return self.sendLEDStatus()
52      elif control == 'X08':
53          ## Konfiguration ändern
54          self.changeConfiguration(config)
55      elif control == 'X09':
56          ## Login
57          return "LOGIN:TRUE"
58      else:
59          print center.writeLog('Übertragung_fehlerhaft')
```

Listing 17: Auswertung der empfangenen Daten (RcvdData.py)

Die komplette Klasse ist einsehbar unter: <https://github.com/hoedding/Studienarbeit-Anwendung/blob/master/RaspberryPI/RecvdData.py>

RecvData.py gesamt

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  #####
4  # Author: Timo Höting #
5  # Mail: mail[at]timohoeting.de #
6  #####
7
8  import hashlib
9  from ConfigReader import *
10 import threading
11
12 class RecvdData(threading.Thread):
13     def __init__(self, c):
14         threading.Thread.__init__(self)
15     global center
16     center = c
17
18     def dataReceived(self, data):
19         # Aufteilung der übertragenen Daten
20
21     def changeModus(self, modus):
22         # Modus des Systems ändern.
23
24     def lightUpOneLED(self, ledNo, red, green, blue):
25         # Eine einzelne LED mit den o.g. RGB-Werten dauerhaft anschalten
26
27     def lightUpLEDRange(self, rangeStart, rangeEnd, red, green, blue):
28         # Einen Bereich von LEDs mit den o.g. RGB-Werten
29         # dauerhaft einschalten
30         # Bereich muss ueberprueft werden mit checkRange()
31
32     def lightUpAllLED(self, red, green, blue):
33         # Alle LEDs mit den o.g. RGB-Werten
34         # dauerhaft einschalten
35         # Bereich muss ueberprueft werden mit checkRange()
36
37     def effectLED(self, code):
38         # Effekte auf einer LED aktivieren
39
40     def checkRange(self, ledNo):
41         # Ueberprueft ob die uebergeben LED-Nummer ueberhaupt im
42         # gueltigen Bereich liegt
43         # Es wird der Eintrag 'number' aus dem Config-File geladen
44
45     def checkColorRange(self, color):
46         # Überprüfung ob Farbwert im gültigen Bereich liegt
47
```

```
48 def checkAuthentication(self, user, pw):
49 # Authentifizierung überprüfen
50 # Eingabewert ist das Passwort aus der Übertragung
51 # Dieses wird gehasht und mit dem in der Konfiguration gespeicherten
52 # Hashwert verglichen
53
54 def checkTransmissionData(self, data, check):
55 # Korrektheit der Übertragung mittels Hashvergleich feststellen
56 # Eingabewert sind die gesamten Daten der Übertragung
57
58 def sendStatus(self):
59 # Status des Systems senden
60
61 def sendLEDStatus(self):
62 # Farbwerte aller einzelnen LEDs senden
63
64 def changeConfiguration(self, config):
65 # Konfiguration der Anwendung ändern
```

Listing 18: Implementierung des Nachrichten-Verarbeitung in Python

Der vollständige Code ist unter <https://github.com/hoedding/Studienarbeit-Anwendung/blob/master/RaspberryPI/RecvData.py> einsehbar.

9.3 Konfiguration

Konfigurations-Datei Die Informationen die zum Betrieb notwendig sind, werden in JSON-Format gespeichert.

```
1 {
2   "username": "",
3   "ledport": "",
4   "motionport2": "",
5   "motionport1": "",
6   "cam_url": "",
7   "pw": "",
8   "ledcount": "",
9   "timeperiod": "",
10  "camavaible": "",
11  "ftp_host": "",
12  "ftp_directory": "",
13  "ftp_user": "",
14  "ftp_pw": "",
15  "cam_user": "",
16  "cam_pw": "",
17  "cam_host": "",
18  "cam_dir": ""
19 }
```

Listing 19: Konfigurationsdatei config.json

Einige dieser Informationen können für die Synchronisation des Clients gesendet werden.

```
1 def sendStatus(self):
2     # Status des Systems senden
3     reader = ConfigReader()
4     message = 'STATUS:{"ledcount":"' + reader.getValue("ledcount") + '",'
5         + "motionport1":"' + reader.getValue("motionport1") + '",' + "motionport2":"'
6         + reader.getValue("motionport2") + '",' + "camavaible":"'
7         + reader.getValue("camavaible") + '",' + "timeperiod":"' +
8         + reader.getValue("timeperiod") + '",' + "ftppdir":"' + reader.getValue("
9         + ftp_directory") + '",' + "ftphost":"' + reader.getValue("ftp_host")
10        + "}"
11    return str(message)
```

Listing 20: Senden von Konfigurationsinformationen an den Client

Status der LED Es ist möglich der Status der einzelnen LEDs abzurufen. Hierfür wird ein JSON-Objekt generiert, welches die einzelnen Farben als 24Bit RGB-Werte enthält.

```
1 def getLEDStatusAsJson(self):
2     led_values = led.getLedAsArray()
3     data = 'LED:{"led":['
4     count = 0
5     if len(led_values) > 0:
6         for i in range(0, len(led_values)-1):
7             data = data + '{"l":"' + str(led_values[i]) + '",'
8             count = count + 1
9             data = data + '{"l":"' + str(led_values[len(led_values)-1]) + '"]}'
10            count = count + 1
11    return data
```

Listing 21: Status der LEDs an Client senden

Lesen und Schreiben der Konfiguration Um die Konfiguration lesen und schreiben bearbeiten zu können wird ein ConfigReader und ein ConfigWriter implementiert. Bei jeder neuen Verbindung der iOS-App zum Server wird das Token für die Notification übertragen. Diese wird, falls noch nicht vorhanden, in die Datei tokenlist.json eingetragen. Das neue Passwort muss vor dem Abspeichern noch in einen Hash-Wert umgewandelt werde.

```
1 class ConfigReader():
2     def getValue(self, key):
3         if key == "token":
4             return self.getToken()
5             data = open('config.json')
6             jdata = json.load(data)
7             return jdata[key]
```

```
8
9     def getToken(self):
10         data = open('tokenlist.json')
11         jdata = json.load(data)
12         return jdata["token"]
13
14 class ConfigWriter():
15     def changeConfig(self, key, value):
16         jsonFile = open("config.json", "r")
17         jdata = json.load(jsonFile)
18         jsonFile.close()
19         jdata[key] = value
20         jsonFile = open("config.json", "w+")
21         jsonFile.write(json.dumps(jdata))
22         jsonFile.close()
23
24     def changePassword(self, value):
25         hashpw = hashlib.sha224(value).hexdigest()
26         jsonFile = open("config.json", "r")
27         jdata = json.load(jsonFile)
28         jsonFile.close()
29         jdata["pw"] = hashpw
30         jsonFile = open("config.json", "w+")
31         jsonFile.write(json.dumps(jdata))
32         jsonFile.close()
33
34     def addNewToken(self, token):
35         jsonFile = open("tokenlist.json", "r")
36         jdata = json.load(jsonFile)
37         jsonFile.close()
38         tokens = jdata['token']
39         for element in tokens:
40             if element['t'] == token:
41                 return
42         jdata['token'].append({"t":token})
43         jsonFile = open("tokenlist.json", "w+")
44         jsonFile.write(json.dumps(jdata))
45         jsonFile.close()
```

Listing 22: ConfigReader / ConfigWriter

9.4 Apple Push Notification

Eigenschaften Die Notifications können von nahezu jedem System an den Apple-Server gesendet werden. Dieser gibt Rückmeldung, ob das Format der Notification korrekt ist und sendet sie an das jeweilige Device. Falls das Gerät nicht verfügbar ist, wird sie eine gewisse Zeit zwischengespeichert, bevor sie verworfen wird.

Jede Notification enthält Darstellungseigenschaften für den Client und einen Payload

mit der Größe 2kb (vor iOS 8 sind es 256Bytes). Die Daten werden in Form von JSON übertragen.

Es können zum Beispiel folgende Informationen übergeben werden:

- Titel der Meldung
- Inhal der Meldung
- Nummer für das App-Icon
- Der abzuspielende Sound

Implementierung

Für die Implementierung wird die Library PyAPNs <https://pypi.python.org/pypi/apns/> eingesetzt. Diese bietet alle Möglichkeiten, auf einfache Art und Weise, Notifications zu senden.

Bei der Initialisierung werden die gespeicherten Tokens aus der Datei tokenlist.json geladen. Im Anschluss kann über die push()-Methode die Notification an alle Geräte gesendet werden.

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  #####
4  # Author: Timo Höting      #
5  # Mail: mail[at]timohoeting.de  #
6  #####
7
8  import sys, time
9  from apns import APNs, Frame, Payload
10 from ConfigReader import *
11
12 class ApplePush():
13     def __init__(self):
14         reader = ConfigReader()
15         tokenlist = reader.getValue("token")
16         global token
17         token = []
18         # Alle Tokens werden aus der Liste geladen
19         for i in tokenlist:
20             token.append(i['t'])
21
22     # Das Apple-Device bekommt eine Message gepusht
23     def push(self, message):
24         # Developer Zertifikat für iOS Push Benachrichtigung
25         apns = APNs(use_sandbox=True, cert_file='certs/Studienarbeit-APN.
                crt.pem', key_file='certs/Studienarbeit-APN.key.pem')
26         payload = Payload(alert="Bewegung_erkannt!", sound="default",
                badge=1)
27         for i in token:
28             apns.gateway_server.send_notification(i, payload)
```

Listing 23: ConfigReader / ConfigWriter

9.5 Unit-Test

In Python ist es möglich Unit-Tests zu schreiben. Mit diesen wird hauptsächlich die Initialisierung der einzelnen Klassen geprüft. So kann schnell herausgefunden werden, ob in diesen Implementierungsfehler vorliegen. Außerdem werden ConfigReader und ConfigWriter getestet.

Eine Ausgabe sieht so aus:

```
1 root@raspberrypi:/home/timo/Studienarbeit# python UNIT_Test.py
2 test_Center (__main__.TestSequenceFunctions) ... ok
3 test_Effects (__main__.TestSequenceFunctions) ... ok
4 test_LEDControl (__main__.TestSequenceFunctions) ... ok
5 test_Sensor (__main__.TestSequenceFunctions) ... ok
6 test_Server (__main__.TestSequenceFunctions) ... ok
7 test_Status (__main__.TestSequenceFunctions) ... ok
8 test_camAdress_MUST_FAIL (__main__.TestSequenceFunctions) ... FAIL
9 test_camAvaible (__main__.TestSequenceFunctions) ... ok
10 test_getHashPass (__main__.TestSequenceFunctions) ... ok
11 test_getMotionPin1 (__main__.TestSequenceFunctions) ... ok
12 test_getNumberOfLED (__main__.TestSequenceFunctions)... ok
13
14 =====
15 FAIL: test_camAdress_MUST_FAIL (__main__.TestSequenceFunctions)
16 -----
17 Traceback (most recent call last):
18   File "UNIT_Test.py", line 67, in test_camAdress_MUST_FAIL
19     self.assertEqual(resultTest, resultCorrect)
20 AssertionError: '192.168.2.205' != '123'
21
22 -----
23 Ran 11 tests in 1.873s
```

Listing 24: Ausgabe der Klasse UNIT_Test

9.6 Threads

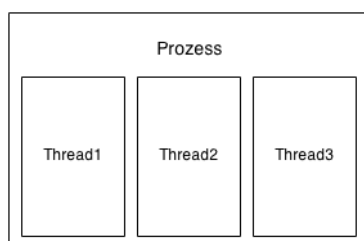


Abbildung 9: Prozess und Threads

Problem Die Server-Klasse und Sensor-Klasse befinden sich in einer Endlosschleife, da sie dauerhaft auf eine Eingabe warten. Beim Server sind dies Empfangene Daten und beim Sensor Bewegungssignale. Würden alle Klassen in einem Thread ablaufen, so würde nur eine Klasse gestartet werden und der Anwendungsablauf in dieser bleiben.

Lösung Die beiden oben genannten Klassen, sowie weitere Klassen wie die LED-Steuerung, werden in eigene Threads ausgelagert. Threads sind Unterprozesse im Hauptprozess, die es ermöglichen mehrere Aufgaben in einem Programm gleichzeitig abzuarbeiten. Zwischen den einzelnen Threads kann Datenaustausch statt finden und es ist möglich übergreifende

Funktionen aufzurufen.

Zur Implementierung wird das Modul 'threading' genutzt. Eine Klasse, die in einem Thread gestartet werden soll, benötigt eine init- und eine run-Methode.

Beispielcode Für eine Funktionsdarstellung der Threads mit Python werden drei Klassen angelegt, eine zur Steuerung und zwei, die in einem Thread laufen sollen. Die Klasse 'Testcenter' initialisiert die Klassen als Threads und startet diese.

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  #####
4  # Author: Timo Höting #
5  # Mail: mail[at]timohoeting.de #
6  #####
7  import threading
8  from TestThread import *
9  from TestThread1 import *
10
11 class TestCenter():
12     def newThread(self):
13         global testthread
14         global testthread1
15         testthread = TestThread('thread0', self)
16         testthread1 = TestThread1('thread1', self)
17         testthread.start()
18         testthread1.start()
19
20     def dosth(self):
21         print 'dosth'
22
23     def dosth2(self):
24         print 'dosth2'
25
26     def dosth3(self):
27         testthread1.calledFromMain('--dosth3')
28
29 if __name__ == "__main__":
30     newThread = TestCenter()
31     newThread.newThread()
```

Listing 25: Klasse Testcenter

Die beiden TestThread-Klassen enthalten beide eine init- und eine run-Methode. Die Klasse Thread1 enthält zusätzlich noch eine Methode die von anderen Klassen ausführbar ist.

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  #####
4  # Author: Timo Höting #
```



```
5 # Mail: mail[at]timohoeting.de #
6 #####
7
8 import threading
9 import time
10 import datetime
11
12 class TestThread1(threading.Thread):
13     def __init__(self,ms,c):
14         threading.Thread.__init__(self)
15         global center
16         center = c
17         global message
18         message = ms
19
20     def run(self):
21         print message
22         center.dosth2()
23
24     def calledFromMain(self, message):
25         print 'calledFromMain' + message
```

Listing 26: Klasse TestThread1

Die `init()`-Methoden werden bei der Erzeugung des Threads aufgerufen und die `run()`-Methode wenn er gestartet wird. Danach können die Methoden wie bei normalen Methodenaufrufen benutzt werden.

Cleanup Um die einzelnen Threads korrekt zu beenden sind Methoden zum Aufräumen notwendig. Dies ist vor allem bei Threads wichtig, die sich in einer Endlosschleife befinden oder Server-Anwendung beherbergen.

- **Webserver:** Im Thread des Twisted-Webservers wird eine Methode implementiert, die den Server korrekt herunterfährt. Somit wird auch der reservierte Port frei gegeben.
- **Sensorauswertung:** Die Sensorauswertung befindet sich in einer Endlosschleife, welche beendet werden muss. Die Schleife wird abhängig einer Variable implementiert. In der Cleanup-Methode wird diese Variable auf false gesetzt.
- **LED-Ansteuerung:** Beim Stoppen der Anwendung müssen alle LEDs ausgeschaltet werden.
- **Kameraaufnahme:** Da zu Beginn der Aufnahme ein FTP-Verzeichnis gemountet wird, muss dieses am Ende frei gegeben werden. Andernfalls können beim neuen Verbinden Fehler entstehen.

10 Konfiguration und Installation

10.1 Installation

Eine gesonderte Installation ist nicht notwendig, es ist nur erforderlich die Anwendung aus Git zu klonen.

```
git clone https://github.com/hoedding/Studienarbeit-Anwendung.git
```

Listing 27: Git Clone der Studienarbeit

10.2 Konfiguration

Die Konfiguration der Anwendung erfordert nur das Ausfüllen der Konfigurationsdatei. Dies ist leicht mit einem Frage-Antwort-Dialog umzusetzen. Dieser wird ebenfalls in Python implementiert. Für das Schreiben der Konfiguration kann die schon vorhandene ConfigWriter-Klasse verwendet werden.

```
1 def raspberryConfig(self):
2     writer = ConfigWriter()
3     print '#####Konfiguration_Raspberry_Pi#####'
4     print '#####Benutzer:admin#####'
5     print '#####Passwort:password#####'
6     ledport = raw_input("Port_der_LED? ")
7     ledcount = raw_input("Anzahl_der_angeschlossenen_LEDs? ")
8     motionport1 = raw_input("Port_Bewegungssensor_1? ")
9     motionport2 = raw_input("Port_Bewegungssensor_2? ")
10    timer = raw_input("Zeitdauer_bei_Bewegungsmelder? ")
11    writer.changeConfig("username", "armin")
12    writer.changeConfig("pw", "
        d63dc919e201d7bc4c825630d2cf25fdc93d4b2f0d46706d29038d01")
13    writer.changeConfig("ledport", ledport)
14    writer.changeConfig("ledcount", ledcount)
15    writer.changeConfig("motionport1", motionport1)
16    writer.changeConfig("motionport2", motionport2)
17    writer.changeConfig("timeperiod", timer)
```

Listing 28: Implementierung setup.py

Im Anschluss müssen noch die erforderlichen Zertifikate (Server-Zertifikat und APN-Zertifikat) erzeugt werden und in den Ordner 'certs' gespeichert werden.

11 iOS App

11.1 Swift

Swift ist eine neue, von Apple entwickelte, Programmiersprache für die Entwicklung von iOS und Mac Apps. Sie ist eine Weiterentwicklung aus der Sprache Objective-C und kann deren Code problemlos ausführen. Laut Apple ist Swift rund 2,6 mal so schnell wie Objective-C.

11.2 Übertragung

Sockets Der erste Ansatz in diesem Projekt, war die Übertragung der Daten über Sockets. Diese können genutzt werden um sämtliche Daten bidirektional über Netzwerke zu senden. Ein Socket wird serverseitig an eine Adresse und einen Port gebunden. Ein Client kann sich zu einem diesen Socket verbinden und Daten austauschen.

Die Implementierung einer Socket-Verbindung funktioniert in Swift problemlos, allerdings ist sie mit einem großen Aufwand verbunden. Große Probleme stellt die Verschlüsselung der zu sendenden Daten dar. Diese müsste selbst implementiert werden. (QUELLEN !!!!).

```
1
2 private var inputstream : NSInputStream!
3     private var outputstream : NSOutputStream!
4     private var host : String = ""
5     private var port : UInt32 = 0
6
7 func connect() {
8     // Initialisierung des Input- und Outputstreams
9 }
10
11 internal func stream(aStream: NSStream, handleEvent eventCode: NSStreamEvent) {
12     // Behandeln der einzelnen Stream-Events
13
14     switch (eventCode){
15     case NSStreamEvent.ErrorOccurred:
16         // Fehler beim Empfang oder Senden
17     case NSStreamEvent.EndEncountered:
18         // Ende der Übertragung
19     case NSStreamEvent.HasBytesAvailable:
20         // Es sind Daten auf dem Stream verfügbar
21     case NSStreamEvent.OpenCompleted:
22         // Stream erfolgreich geöffnet
23     case NSStreamEvent.HasSpaceAvailable:
24         // Space am Ende der Übertragung
25     default:
26     }
27 }
```

Listing 29: Implementierung einer Socketverbindung in Swift (Schematische Darstellung)

Die vollständige Implementierung von mir ist auf Github einsehbar (<https://github.com/hoedding/Studienarbeit/blob/master/iOS-App/Studienarbeit/ConnectServerTCP.swift>).

HTTP Aufgrund der aufwändigen Implementierung und Schwierigkeiten mit der Verschlüsselung bei der Socketverbindung wurde als zweiter Ansatz die Übertragung der Daten im HTTP-Protokoll gewählt. Dieses Protokoll wird im Internet für die Datenübertragung zu Webservern verwendet. Ein Webserver wartet auf eingehende Anfragen von Clients. Beim Verbindungsaufbau wertet er die Daten aus und antwortet dementsprechend. Eine Anfrage kann beliebige Daten enthalten.

In Swift ist die Funktionalität des Clients in der Klasse `NSURLSession` implementiert und ist einfach einzusetzen. Die Verschlüsselung findet automatisch statt, wenn eine Zieladresse mit dem Protokollidentifizier 'https' beginnt.

Da ein HTTP-Request ein Asynchrone Request ist, muss der Methode `sendMessageViaHttpPostWithCompletion()` eine andere Methode (`completionClosure : (s : NSString) ->()`) übergeben werden, die aufgerufen wird, wenn die Abfrage beendet ist. Mit dem Framework 'IJReachability' kann überprüft werden ob eine Internetverbindung verfügbar ist und von welchem Typ (WLAN, Mobile) diese ist.

```
1 func sendMessageViaHttpPostWithCompletion(message : NSString, completionClosure :  
    (s : NSString) -> ()) {  
2     self.initServerConnection()  
3     var result : NSString = ""  
4     if !IJReachability.isConnectedToNetwork() {  
5         return  
6     }  
7     let request = NSMutableURLRequest(URL: NSURL(string: server)!)  
8     request.HTTPMethod = "POST"  
9     let postString = "data=" + (message as String)  
10    request.HTTPBody = postString.dataUsingEncoding(NSUTF8StringEncoding)  
11  
12    var configuration = NSURLSessionConfiguration.defaultSessionConfiguration()  
13    var session = NSURLSession(configuration: configuration, delegate: self,  
        delegateQueue: NSOperationQueue.mainQueue())  
14    var task = session.dataTaskWithRequest(request) {  
15        data, response, error in  
16  
17        if error != nil {  
18            println("error=\(error)")  
19            return  
20        }  
21        result = NSString(data: data, encoding: NSUTF8StringEncoding)!  
22        completionClosure(s: result)  
23    }  
24    task.resume()  
25 }
```







































Listing 30: Implementierung HTTP-Request in Swift

Die vollständige Implementierung ist auf Github einsehbar (<https://github.com/hoedding/Studienarbeit-Anwendung/blob/master/iOS-App/Studienarbeit/ConnectServerHTTP.swift>)

11.3 CoreData

Mit CoreData wird eine Framework zur persistenten Speicherung von Daten geboten. Die Datenbank basiert auf einem tabellenbasierten relationalen Datenbankmodell. Grundsätzlich sollen hier die selben Daten wie in der Serveranwendung gespeichert werden. Die Entität 'Config' enthält folgende Attribute:

▼ Attributes

Attribute ^	Type	
 authWithoutPW	String	
 camavaible	String	
 camdir	String	
 campassword	String	
 camurl	String	
 camuser	String	
 ftp	String	
 ftpdir	String	
 ftppassword	String	
 ftpuser	String	
 host	String	
 ledcount	String	
 modus	String	
 motionport1	String	
 motionport2	String	
 port	String	
 pw	String	
 timeperiod	String	
 user	String	

+ —

Abbildung 10: Entität Config

Um einen schnellen Zugriff auf die Daten zu ermöglichen wurden Methoden zum Lesen und Schreiben implementiert. Unter anderem mit der Methode `changeValueWithEntityName()` kann ein Eintrag geändert werden:

```

1 func changeValueWithEntityName(entityName : String, key : String, value : AnyObject
  ) {
2     var err : NSError? = nil
3     var appDel : AppDelegate = (UIApplication.sharedApplication()).delegate as!
      AppDelegate)
4     var context : NSManagedObjectContext = appDel.managedObjectContext!
5     var request = NSFetchRequest(entityName: entityName)
6     request.returnsObjectsAsFaults = false
7     var result : Array = context.executeFetchRequest(request, error: &err)! as Array
8     if (result.count == 1){
9         for res in result {
10             res.setValue(value, forKey: key)

```

```

11     }
12 }
13 context.save(&err)
14 if (err != nil) {
15     println(err)
16 }
17 }
    
```

Listing 31: Implementierung HTTP-Request in Swift

11.4 Konzept

Beim Start der App soll die Verbindung zum Server aufgebaut werden. Hierbei sollen die Zugangsdaten überprüft werden und alle Daten synchronisiert werden. Das bedeutet es werden bei jeder Verbindung mit korrektem Login alle Konfigurationsdaten des Servers an den Client gesendet. Des weiteren wird der aktuelle Status der LEDs übertragen. Nach dem Verbindungsaufbau ist es dem Benutzer möglich, den Modus des Systems zu verändern. Außerdem kann er alle anderen Funktionen in der App nutzen.

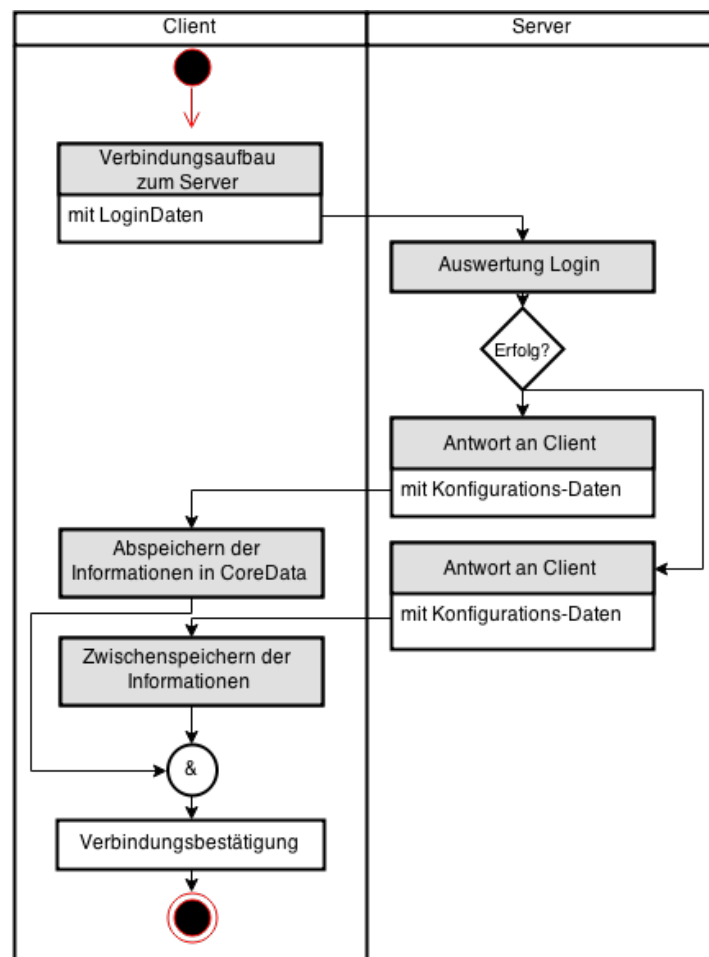


Abbildung 11: Konzept der Server-Client-Kommunikation

11.5 Aufbau

In einer Standard iOS-App werden die einzelnen Seiten Klassenweise implementiert. In einem visuellen Editor können Grundelemente wie Buttons und Textfelder hinzugefügt werden. Diese werden über Outlets mit den zugehörigen Klassen verbunden und können direkt im Code angesprochen werden. Es wurden weitere Klassen für die Kommunikation mit der Serveranwendung und CoreData angelegt. Um zwischen den einzelnen Views zu wechseln, wird ein Menü verwendet, welches sich durch eine Wisch-Bewegung vom linken Rand öffnet.

Frameworks Es wurden einige Frameworks eingesetzt:

- **CryptoSwift:** Bietet viele Hashalgorithmen.
- **VIPhotoView:** Notwendig für die Bildansicht im Archiv.
- **WhiteRacoon:** FTP-Verbindung und Management.
- **IJReachability:** Überprüft welche Internetverbindung vorhanden ist.
- **SideMenu:** Slide-Menü der Anwendung.

12 Praktische Umsetzung

Eines der Ziele dieses Projekts war die praktische Umsetzung an einem Beispiel. Als Beispielobjekt wurde ein innenliegendes Treppenhaus gewählt.

Der Raspberry Pi wurde am oberen Ende der Treppe installiert und der LED-Streifen verlief an einer Seite der Treppe nach unten. Jeweils am Ende des Streifens wurde ein Bewegungssensor angebracht. Die Kamera wurde im oberen Bereich an der Wand befestigt, von wo aus ein guter Blickwinkel auf die Treppe geboten war. Als FTP-Server wurde ein vorhandener NAS eingesetzt.

Bis zum Aufbau wurden die einzelnen Komponenten über ein Experimentierboard verbunden. Da dies sehr fehleranfällig war, wurde eine kleine Platine entworfen. Diese sollte ein Steckfeld darstellen, auf dem die Komponenten leicht und sicher verbunden werden können. Der Entwurf wurde mit der Software EAGLE angefertigt, welche auch in professionellen Bereichen eingesetzt wird. Sie bietet die Möglichkeit zum Erstellen von komplexen Schaltungen und automatischem Routing von Platinen.

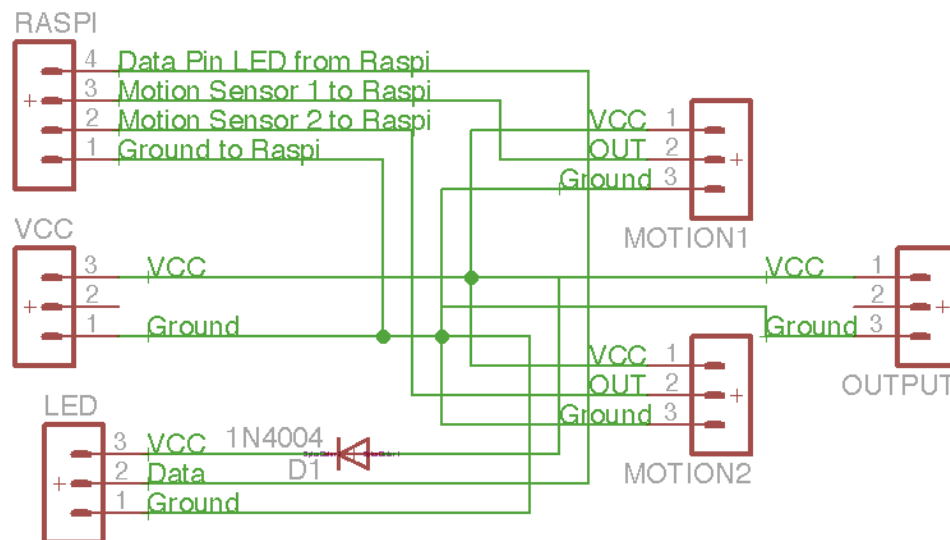


Abbildung 12: Schaltplan der Steckplatine

Das zugehörige Layout und Bilder der Platine befinden sich im Anhang.

Fazit Die Anwendung war einige Tage in Betrieb und funktionierte gut. Es wurden alle Funktionen getestet. Die Benachrichtigung auf das Smartphone funktionierte problemlos und innerhalb des Netzwerks konnte direkt zum Live-View gewechselt werden.

13 Fazit

inhalt:

- arbeit beginnen um etwas neues zu lernen
- arbeit ohne kenntnisse in python + swift
- noch nie netzwerkkommunikation programmiert
- ziel war, dass alle komponenten zusammen arbeiten

14 Abbildungsverzeichnis

Abbildungsverzeichnis

1	Ergebnisse der LED-Evaluierung	9
2	Schaltung für LED-Test	10
3	Ergebnisse der Motion-Sensor-Evaluierung	13
4	RTSP Request-Strings	17
5	Ablauf der Aufzeichnung mit FFmpeg	18
6	Abrufen des Bildes über HTTP-Request	21
7	Wireshark Trace TLS Handshake	24
8	Anwendungsstruktur Server-Anwendung	29
9	Prozess und Threads	37
10	Entität Config	43
11	Konzept der Server-Client-Kommunikation	44
12	Schaltplan der Steckplatine	46

Listings

1	Installation Framework ws281x	11
2	Testcode zur Ansteuerung der LEDs	11
3	Testcode zur Bewegungserkennung mit Sensor	13
4	Aufnahme mit FFmpeg	18
5	Testcode - Aufnahme Screenshot mit Selenium	19
6	Testcode - Aufnahme Screenshot mit Selenium	19
7	Abrufen eines Bildes von einer URL in Python	20
8	Abrufen eines Bildes von einer URL in Swift	20
9	NSTimer in Swift	20
10	Starttls - Wechsel zur Verschlüsselung	23
11	private Key	23
12	Certificate Signing Request	23
13	Self Signed Certificate	23
14	Beispielübertragung des Protokolls	26
15	Testcode Echoserver mit Twisted Framework	26
16	Implementierung des Webservers in Python	27
17	Auswertung der empfangenen Daten (RcvdData.py)	30
18	Implementierung des Nachrichten-Verarbeitung in Python	32
19	Konfigurationsdatei config.json	33
20	Senden von Konfigurationsinformationen an den Client	34
21	Status der LEDs an Client senden	34
22	ConfigReader / ConfigWriter	34
23	ConfigReader / ConfigWriter	36
24	Ausgabe der Klasse UNIT_Test	37
25	Klasse Testcenter	38
26	Klasse TestThread1	38
27	Git Clone der Studienarbeit	40
28	Implementierung setup.py	40
29	Implementierung einer Socketverbindung in Swift (Schematische Darstellung)	41
30	Implementierung HTTP-Request in Swift	42
31	Implementierung HTTP-Request in Swift	43

Literatur

- [I.] “SWR Info - Zahlen, Daten, Fakten über den SWR“, <http://www.swr.de/unternehmen/unternehmen/kennzahlen/kennzahlen-organisation/-/id=12213420/did=12302978/nid=12213420/eqq46v/index.html>, 13.12.2013