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

1.1 Vorwort

1.2 Projektbeschreibung

Im Zuge 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 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 iPhone-App über einen Raspberry Pi erfolgen. 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. Ob die Überwachungskamera direkt am Raspberry Pi angeschlossen wird oder sich nur im selben Netzwerk befindet, wird im Laufe dieses Projekts erarbeitet. Es gibt drei verschiedene Modi in denen sich das System befinden kann:

- Beleuchtung wird durch Bewegungsmelder ausgelöst (Reaktion darauf kann vom User definiert werden)
- Beleuchtung wird manuell vom Benutzer über App gesteuert (Color Chooser, Bereichsauswahl, Leuchteffekte)
- Bewegungsmelder als Alarmanlage, beim Auslösen wird der Benutzer benachrichtigt und Bild der Kamera als Notifcation auf dem Smartphone angezeigt

1.3 Teilprojekte

- LED-Pixel und Bewegungssensoren evaluieren / ansteuern
- Implementierung der Ansteuerung / des Protokolls (mit den drei verschiedenen Modi)
- Implementierung der iOS App
- Vollständige praktische Umsetzung an einem Beispielobjekt

1.4 Projektmanagement

1.4.1 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.

Bemerkungen

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

1.4.2 Meilensteine und Erfolgsprüfung

1. Planung der Architektur (15. September - 30. September 2014)
    - Entwurf Anwendungsstruktur
        - Der Entwurf der Anwendungssturktur 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 Anwendungsstrktur 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 Webservers in diesem Meilenstein erneut verändert.
- Refactoring Webserver
  - Der Webserver wurde auf HTTPS umgestellt.

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 konsistene Speicherung
     - Der Zugriff auf CoreData wude implementiert.
7. Abschluss der Arbeit (1. April - 11. Mai 2015)
   - Implementierung Netzwerkkamera
     -
   - Fertigstellung Ausarbeitung
     -
   - Abgabe
     -

2 Hauptteil

2.1 LED-Pixel

2.1.1 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 Pulsweitenmodulation[1] (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

2.1.2 Evaluierung

- 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.

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

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.

- 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-Ansteurung 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 2.1: 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.

2.1.3 Teststellung

Für einen ersten Test wurde das in XXX 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
- Vorraussetzung: 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.

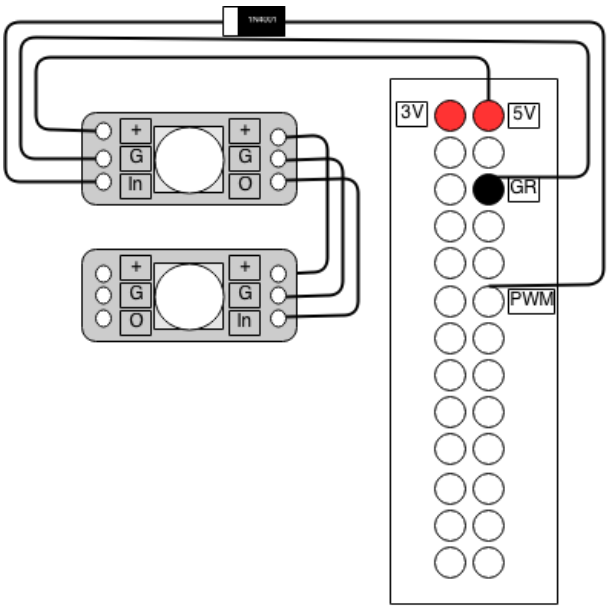


Abbildung 2.2: Schaltung für LED-Test

- **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 2.1: Installation Framework ws281x

- **Testcode**

```
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 2.2: 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.

2.2 Bewegungssensor

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

2.2.1 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

2.2.2 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/B008AESD
  Ansteuerung: Gibt High-Signal an einem Pin aus.
  Reichweite: 7m, 100 Grad
  Kosten: 5 Stück = 7,66€
  Extras: keine

	PIR (MOTION) Sendor Adafruit	PIR Infrared Motion Sensor (HC-SR501)	Neopixel von AdafruitInfarot 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 2.3: Ergebnisse der Motion-Sensor-Evaluierung

Fazit Die meisten Infarot-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.

2.2.3 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: $<$ 50uA
- Ausgansspannung: 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
4  GPIO.setmode(GPIO.BCM)
5
6  # Pin definieren
7  MOTION_PIN1 = 7
8
9  # Diese als Input definieren
10 GPIO.setup(MOTION_PIN1,GPIO.IN)
11
12 # Status definieren um verschiedene nderungen zu erkennen
13 Current_State = 0
14 Previous_State = 0
15
16 try:
17         # Loop zur Erkennung einer Bewegung
18         # Sensor erkennt Bewegung -> Signal = High
```

```
19          # Wartet 3 Sekunden und setzt Signal = Low
20          while True :
21                  Current_State = GPIO.input(MOTION_PIN1)
22                  if Current_State == 1 and Previous_State == 0:
23                          print "Motion detected!"
24                          Previous_State=1
25                  elif Current_State == 0 and Previous_State == 1:
26                          print "Ready"
27                          Previous_State=0
28                  time.sleep(0.01)
29
30  except KeyboardInterrupt:
31          print "Quit"
32          GPIO.cleanup()
```

Listing 2.3: Testcode zur Bewegungserkennung mit Sensor

Bei der Endversion des Systems sollen mehrere Beweungssensoren 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 Tststellung bestätigt werden.

2.3 Kamera

2.3.1 PI-Kamera vs. Netzwerkkamera

Bei Auslösen des System im Überwachungsmodus soll ein aktuelles Bild der Überwachungskamera an das jeweilige Sartphone gepusht werden. Es gibt zwei mögliche Kameratechniken, entweder eine direkt an den Raspberry Pi Angeschlossene oder eine, die im Netzwerk erreichbar ist.

Raspberry Pi Cam Die Kameras für den Raspberry Pi können direkt an das Gerät angeschlossen werden. Meistens werden sie direkt über die GPIO Pins verbunden. Der Vorteil dieser Kameras ist, dass sie keine externe Stromversorgung benötigen und durch viele verschiedene Frameworks leicht anpassbar und verwaltbar sind. Der große Nachteil ist allerdings, dass die Kamera an dem Raspberry Pi angeschlossen werden muss, auf welchem auch der Server läuft. Da dieser aber möglich wettergeschützt (im Außenbereich) oder unauffällig (im Innenbereich) angebracht ist, lässt sich von diesen Positionen kaum eine effektive Überwachung realisieren.
Als Beispiel wäre ein von der Raspberry Pi Foundation empfohlene Kamera zu nennen: //TODO Beispielkamera

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 irgendwo befinden kann, solange sie im selben Netzwerk ist. Somit

kann zum Beispiel eine wetterfeste Kamera im Außenbereich angebracht werden und der Server kann sich im geschützten Innenbereich befinden.
Der Nachteil besteht bei IP-Kameras darin, dass es keine einheitliche API zum Abgreifen des Videomaterials gibt. Die meisten IP-Kameras bieten die Möglichkeit, die Aufgenommenen Bilder auf einem FTP-Server abzulegen. Weiter wäre eine mögliche Lösung das Laden der HTML Seite über einen HTTP-Request und darauffolgend das Ausfiltern des Bildmaterials. Über diese Variante kann aber kein Video sondern nur temporäre Bilder geladen werden. Dies würde aber für eine Notification auf dem Smartphone ausreichen.

Für dieses Projekt wird eine IP-Kamera aufgrund von oben genannten Vorteilen verwendet.

2.3.2 Ansteuerung

Testcode HTTP-Request

Implementierung

2.4 Verschlüsselung

2.4.1 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 nur 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 POODLE[2], 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 BEAST[3] Angriffsmethoden.
TODO: NACH HINTEN SCHIEBEN 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.
Wenn ein Server implementiert wird, so muss er alle Techniken unterstützen, beim Client kann der Entwickler selbst entscheiden. Ein Entwickler sollte immer die höchst mögliche Verschlüsselungstechnik einsetzen.

2.4.2 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.

[2] SIcherheitslücke in SSL v3
[3] Sicherheitslücke in TLS v1.0

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.

2.4.3 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.
Ein Wireshark Trace zu diesem Handshake befindet sich in 2.3.6.

2.4.4 Zertifikat und Key

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 2.4: private Key

2. Certificate Signing Request

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

Listing 2.5: 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 2.6: Self Signed Certificate

Apple Zertifikat Um die Apple Push Notifications einzusetzen ist ein Apple-Developer Zertifikat notwendig, welches nur über das Apple-Developer Portal erzeugt werden kann.

2.4.5 Wireshark Trace

Im folgenden ist ein Trace eines TLS Handshakes zwischen einem Client und dem implementierten Server 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 2.4: Wireshark Trace TLS Handshake

2.5 Python-Server und Protokoll

2.5.1 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 weis mit höchster Leuchstärke.
- Konfiguration
  Damit können einzelne Elemente der Serverkonfiguration verändert werden. Zum Beispiel die Leuchtdauer der LEDs, wenn sie durch den Bewegunsmelder ausgelöst wurden.
- Hash
  Überprüfung ob die Übertragung erfolgreich war, mittels eines Hashwertes. Es wird der SHA-224-Algorithmus eingesetzt.

Übertragungsbeispiel:

```
auth:pw:control:ledNo:rangeStart:rangeEnd:red:green:blue:modus:effectcode:config:hash
user:password:X01:0:0:49:255:255:255:::Hashvalue
```

Listing 2.7: Beispielübertragung des Protokolls

Dies würde die LEDs 0 bis 49 einschalten (Farbe weis 255,255,255). Anstelle des 'Hashvalue' würde der Hashwert der gesamten Übertragung gesendet.

2.5.2 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 simplen Server 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.

2.5.3 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.

```
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 2.8: Testcode Echoserver mit Twisted Framework

2.5.4 Implementierung Webserver

Der Server wird in einem neuen Thread gestartet, damit er beim Empfang von Daten keine anderen Abläufe aufhält. Zusätzich wird ihm eine Instanz der Klasse RRecvdData“übergeben. Diese verarbeitet die empfangene Nachricht. Anhand der “:“werden die empfangenen Daten gesplittet und in ein Array abgelegt. Zur besseren Verständlichkeit 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.

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

```
5  # Mail: mail[at]timohoeting.de    #
6  #
       ##########################################################
7
8  from twisted.web.server import Site
9  from twisted.web.resource import Resource
10 from twisted.internet import reactor
11 import cgi
12 from twisted.internet.protocol import Factory, Protocol
13 from twisted.internet import reactor
14 import hashlib
15 from ConfigReader import *
16 import threading
17 from twisted.internet import reactor, ssl
18
19 class LightServer(Resource):
20   def render_POST(self, request):
21     message = datamanager.dataReceived(cgi.escape(request.args["data"][0]))
22     if (message != ""):
23         return message
24
25 class StartLightServer(threading.Thread):
26   def __init__(self, d):
27     threading.Thread.__init__(self)
28     global datamanager
29     datamanager = d
30
31   def run(self):
32     root = Resource()
33     root.putChild("serv", LightServer())
34     factory = Site(root)
35     sslContext = ssl.DefaultOpenSSLContextFactory(
36         './certs/server.key', './certs/server.crt'
37     )
38     reactor.listenSSL(8000, factory, contextFactory = sslContext)
39     #reactor.listenTCP(8000, factory)
40     reactor.run(installSignalHandlers=False)
```

Listing 2.9: Implementierung des Webservers in Python

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  #
       ##########################################################
4  # Author: Timo Hting     #
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         # Protokoll: auth:pw:control:ledNo:rangeStart:rangeEnd:red:green:blue:modus:
               effectcode:config:hashv
20         # Beispiel: admin:w:X00:1:0:0:10:10:10:0:0:w-w:58
               acb7acccce58ffa8b953b12b5a7702bd42dae441c1ad85057fa70b
21         # Ermoeglicht Zuweisung von Farben und Effekten
22         # Ermglicht Abruf von aktuellem Status des Systems und der LEDs
23         #
24         # Ankommende String bei ":" aufsplitten und in Array a[] Speichern:
25         a = data.split(':')
26         print a
27         if len(a) > 1:
28             auth =   a[0]
29             pw =   a[1]
30             control =  a[2]
31             ledNo =  a[3]
32             rangeStart = a[4]
33             rangeEnd =  a[5]
34             red =   a[6]
35             green =  a[7]
36             blue =   a[8]
37             modus =  a[9]
38             effectcode = a[10]
39             config = a[11]
40             hashv = a[12]
41             data = auth + pw + control + ledNo + rangeStart + rangeEnd + red +
                   green + blue + modus + effectcode + config
42             data = data.rstrip('\n')
43             data = data.rstrip('\r')
44             if (self.checkAuthentification(auth, pw) & self.checkTransmissionData(data,
                    hashv)):
45                 if control == 'X00':
46                     ## Alle LEDs ausschalten
47                     center.clearPixel()
```

```
48                  elif control == 'X01':
49                      ## Eine LED anschalten
50                      self.lightUpOneLED(int(ledNo), int(red), int(green), int(blue))
51                  elif control == 'X02':
52                      ## LED Bereich anschalten
53                      self.lightUpLEDRange(int(rangeStart), int(rangeEnd), int(red),
                            int(green), int(blue))
54                  elif control == 'X03':
55                      ## Eine Farbe fr alle LED
56                      self.lightUpAllLED(int(red), int(green), int(blue))
57                  elif control == 'X04':
58                      ## Effekt alle LEDs
59                      self.effectLED(effectcode)
60                  elif control == 'X05':
61                      ## Modus des Systems
62                      self.changeModus(int(modus))
63                  elif control == 'X06':
64                      ## Systemstatus als JSON an den Client
65                      return self.sendStatus()
66                  elif control == 'X07':
67                      ## Status der einzelnen LEDs senden
68                      return self.sendLEDStatus()
69                  elif control == 'X08':
70                      ## Konfiguration ndern
71                      self.changeConfiguration(config)
72                  elif control == 'X09':
73                      ## Login
74                      return "LOGIN:TRUE"
75              else:
76                  print center.writeLog('bertragung_fehlerhaft')
77
78      def changeModus(self, modus):
79          if modus >= 0 & modus < 4:
80              center.setModus(modus)
81
82      def lightUpOneLED(self, ledNo, red, green, blue):
83          # Eine einzelne LED mit den o.g. RGB-Werten dauerhaft anschalten
84          a = self.checkColorRange(red)
85          b = self.checkColorRange(green)
86          c = self.checkColorRange(blue)
87          d = self.checkRange(ledNo)
88          if ( a & b & c & d):
89              center.lightUpOneLED(ledNo, red, green, blue)
90
91      def lightUpLEDRange(self, rangeStart, rangeEnd, red, green, blue):
92          # Einen Bereich von LEDs mit den o.g. RGB-Werten
93          # dauerhaft einschalten
94          # Bereich muss ueberprueft werden mit checkRange()
```

```
95          a = self.checkColorRange(red)
96          b = self.checkColorRange(green)
97          c = self.checkColorRange(blue)
98          d = self.checkRange(rangeStart)
99          e = self.checkRange(rangeEnd)
100         if ( a & b & c & d & e):
101             center.rangePixel(rangeStart, rangeEnd, red, green, blue)
102
103     def lightUpAllLED(self, red, green, blue):
104         # Alle LEDs mit den o.g. RGB-Werten
105         # dauerhaft einschalten
106         # Bereich muss ueberprueft werden mit checkRange()
107         a = self.checkColorRange(red)
108         b = self.checkColorRange(green)
109         c = self.checkColorRange(blue)
110         if ( a & b & c):
111             center.lightUpAllLED(red, green, blue)
112
113     def effectLED(self, code):
114         # Effekte auf einer LED aktivieren
115         center.effectLED(code)
116
117     def checkRange(self, ledNo):
118         # Ueberprueft ob die uebergeben LED-Nummer ueberhaupt im
119         # gueltigen Bereich liegt
120         # Es wird der Eintrag 'number' aus dem Config-File geladen
121         reader = ConfigReader()
122         number = int(reader.getNumberOfLED())
123         if ( ledNo >= 0 & ledNo < number):
124             return True
125         else:
126             return False
127
128     def checkColorRange(self, color):
129         # berprfung ob Farbwert im gltigen Bereich liegt
130         if (color >= 0 & color <= 255):
131             return True
132         return False
133
134     def checkAuthentification(self, auth, pw):
135         # TODO
136         # Authentifizierung berprfen
137         # Eingabewert ist das Passwort aus der bertragung
138         # Dieses wird gehasht und mit dem in der Konfiguration gespeicherten
139         # Hashwert verglichen
140         reader = ConfigReader()
141         hashv = reader.getHashPass()
142         pw = hashlib.sha224(auth).hexdigest()
```

```
143             if ( pw == hashv ):
144                 return True
145             return False
146
147         def checkTransmissionData(self, data, check):
148             # Korrektheit der bertragung mittels Hashvergleich feststellen
149             # Eingabewert sind die gesamten Daten der bertragung
150             hashdata = hashlib.sha224(data).hexdigest()
151             check = check.rstrip('\n')
152             check = check.rstrip('\r')
153             if ( hashdata == check ):
154                 return True
155             # TODO Fr Testbertragung return immer True
156             return True
157
158         def sendStatus(self):
159             # Status des Systems senden
160             reader = ConfigReader()
161             message = 'STATUS:{"ledcount":"' + reader.getNumberOfLED() + '","
                    motionport1":"' + reader.getMotionPin1() + '","motionport2":"' + reader.
                    getMotionPin2() + '","ftp_url":"' + reader.getFTP() + '","camavaible":"'
162             message = message + reader.camAvaible() + '","cam_url":"' + reader.
                    camURL() + '","cam_url_short":"' + reader.camShortURL() + '","
                    timeperiod":"' + reader.getTimePeriod() + '"}'
163             print message
164             return str(message)
165
166         def sendLEDStatus(self):
167             # Farbwerte aller einzelnen LEDs senden
168             ledstatus = center.getLEDStatusAsJson()
169             return ledstatus
170
171         def changeConfiguration(self, config):
172             b = config.split('--')
173             key = b[0]
174             value = b[1]
175             center.changeConfiguration(key, value)
```

Listing 2.10: Implementierung des Nachrichten-Verarbeitung in Python

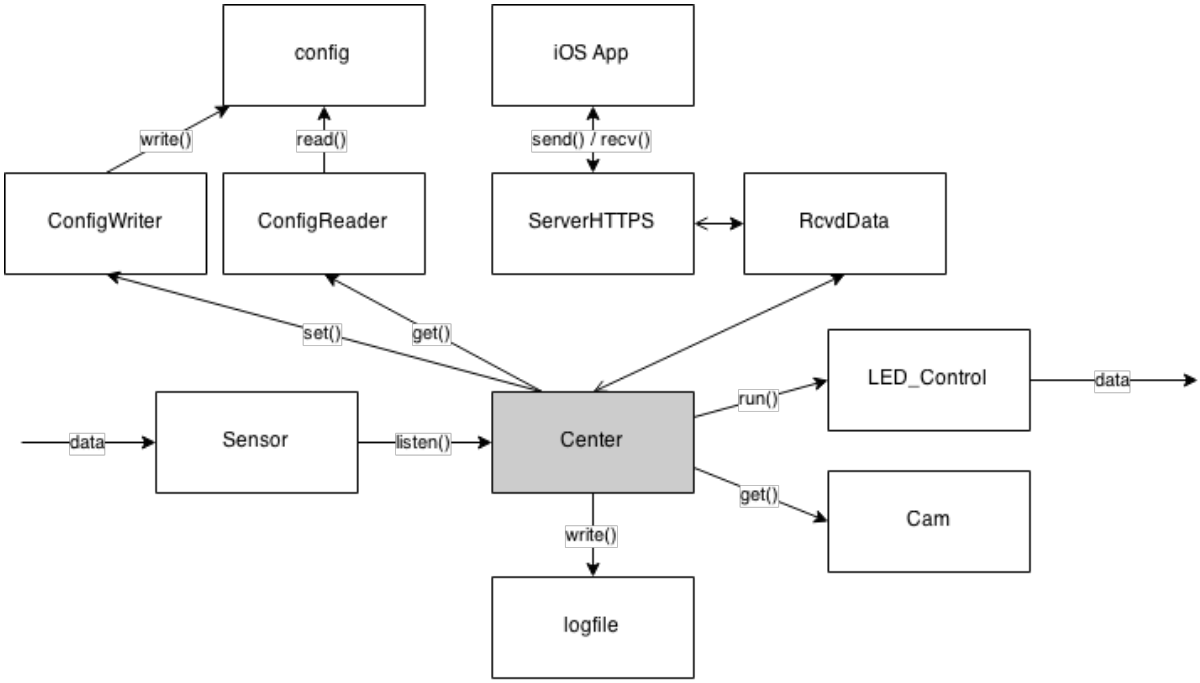
2.5.5 Hashfunktion

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, auf dem Server aber nur als Hash-Wert abgespeichert. Falls es also jemand schafft die Konfirgurationsdatei 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.

2.6 Anwendungsstruktur Webserver

2.6.1 Klassen und ihre Funktionen



- 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.
- Cam
  Abrufen von Bildmaterial von der Netzwerkkamera oder vom Server findet ausschließlich über die Klasse 'Cam' statt. Die Klasse ruft die Informationen ab und filtert das Bildmaterial.
- UNIT-Test
  Einzelne Rückgabetypen von Methoden, sowie die Initialisierung von allen Klassen kann mit 'UNIT_Test' getestet werden.
- Logfile
  Im Logfile werden unter anderem auftretende Fehler gespeichert.

2.6.2 Auswertung empfangener Daten

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

```
1  class RecvdData(threading.Thread):
2      def dataReceived(self, data):
3          # Protokoll: auth:pw:control:ledNo:rangeStart:rangeEnd:red:green:blue:modus:
               effectcode:config:hashv
4          # Beispiel: admin:w:X00:1:0:0:10:10:10:0:0:w-w:58
               acb7acccce58ffa8b953b12b5a7702bd42dae441c1ad85057fa70b
5          a = data.split(':')
6          if len(a) > 1:
7              auth =   a[0]
8              pw =   a[1]
9              control =  a[2]
10             ledNo =  a[3]
11             rangeStart = a[4]
12             rangeEnd =  a[5]
13             red =   a[6]
14             green =  a[7]
15             blue =   a[8]
16             modus =  a[9]
17             effectcode = a[10]
18             config = a[11]
19             hashv = a[12]
20             data = auth + pw + control + ledNo + rangeStart + rangeEnd + red +
                   green + blue + modus + effectcode + config
```

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

Listing 2.11: Auswertung der empfangenen Daten (RcvdData.py)

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

2.6.3 Konfiguration

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

```
1  {
2    // Anzahl der LEDs
```

```
3   "ledcount": "2",
4   // Name des Benutzers
5   "username": "testuser",
6   // Hash des Nutzerpassworts
7   "passhash": "58acb7acccce58ffa8b953b12b5a7702bd42dae441c1ad85057fa70b",
8   // Port an dem ein Bewegungssensor angeschlossen wird
9   "motionport1": "7",
10  "motionport2": "8",
11  // Verfgbarkeit der Netzwerkkamera
12  "camavaible": "1",
13  // URL der Kamera
14  "cam_url": "test.de",
15  "cam_url_short": "test2.de",
16  // Zeitdauer der LED-Beleuchtung
17  "timeperiod": "10",
18  // FTP Details
19  "ftp_url": "test3.de",
20  "ftp_directory":"",
21   "ftp_user":"",
22   "ftp_pw":""
23 }
```

Listing 2.12: Konfigurationsdatei config.json

Diese Informationen können auch vom Client abgerufen werden:

```
1       def sendStatus(self):
2           # Status des Systems senden
3           reader = ConfigReader()
4           message = 'STATUS:{"ledcount":"' + reader.getNumberOfLED() + '","
                motionport1":"' + reader.getMotionPin1() + '","motionport2":"' + reader.
                getMotionPin2() + '","ftp_url":"' + reader.getFTP() + '","camavaible":"'
5           message = message + reader.camAvaible() + '","cam_url":"' + reader.
                camURL() + '","cam_url_short":"' + reader.camShortURL() + '","
                timeperiod":"' + reader.getTimePeriod() + '"}'
6           return str(message)
```

Listing 2.13: 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-Werte enthält.

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

Listing 2.14: Status der LEDs an Client senden

Lesen und Schreiben der Konfiguration Um die Konfiguration lesend und schreiben bearbeiten zu können wird ein ConfigReader und ein ConfigWriter implementiert:

```
1  class ConfigReader():
2          def getValueOfKey(self, key):
3                  data = open('config.json')
4                  jdata = json.load(data)
5                  return jdata[key]
6
7  class ConfigWriter():
8          def changeConfig(self, key, value):
9                  jsonFile = open("config.json", "r")
10                 jdata = json.load(jsonFile)
11                 jsonFile.close()
12                 jdata[key] = value
13                 jsonFile = open("config.json", "w+")
14                 jsonFile.write(json.dumps(jdata))
15                 jsonFile.close()
```

Listing 2.15: ConfigReader / ConfigWriter

2.6.4 Apple Push Notification

TODO

2.6.5 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 2.16: Ausgabe der Klasse UNIT_Test

2.6.6 Threads

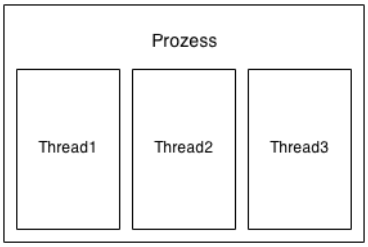


Abbildung 2.5: Prozess und Threads

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 Hting #
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 2.17: 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 Hting #
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 2.18: 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.

2.7 Konfiguration und Installation

2.7.1 Konfiguration

2.7.2 Installation

2.8 iOS App

2.8.1 Swift

2.8.2 Übertragung

Sockets Der erste Ansatz in diesem Projekt, war die Übertragung der Daten über Sockets. Diese können genutzt werden um Daten (wie TCP, UDP) bidirektional über Netzwerke zu senden. Ein Socket wird an eine Adresse und einen Port gebunden. Ein Client kann sich zu einem serverseitigen Socket verbinden.
Die Implementierung einer Socket-Verbindung funktioniert in Swift problemlos, allerdings ist sie mit einem großen Aufwand verbunden. Das Problem ist dabei, dass innerhalb von Swift dieser Datenstream nicht einfach verschlüsselt werden kann (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 bertragung
19         case NSStreamEvent.HasBytesAvailable:
20                 // Es sind Daten auf dem Stream verfgbar
21         case NSStreamEvent.OpenCompleted:
22                 // Stream erfolgreich geffnet
```

```
23          case NSStreamEvent.HasSpaceAvailable:
24                  // Space am Ende der bertragung
25          default:
26          }
27      }
```

Listing 2.19: Implementierung einer Socketverbindung in Swift

Die vollständige Implementierung ist auf Github einsehbar (https://github.com/hoedding/Studienarbeit Anwendung/blob/master/iOS-App/Studienarbeit/ConnectServerTCP.swift) oder unter http://timohoe to-socketbase-ios-app-mit-swift/

HTTP Aufgrund der aufwändigen Implementierung und Schwierigkeiten mit der Verschlüsselung wurde als zweiter Ansatz die Übertragung der Daten im HTTP-Protokoll gewählt. Hier wartet ein Webserver auf eingehende Anfragen von Clienten. Diese können beliebige Daten enthalten und werden oft zur Abfrage von Logins oder Suchergebnissen genutzt.
In Swift können Anfragen an Webserver sehr leicht durchgeführt werden, wobei auch die Verschlüsselung (HTTPS) ohne Probleme funktioniert.

```
1       func sendMessageViaHttpPostWithCompletion(message : NSString,
            completionClosure : (s : NSString) -> ()) {
2           // Diese Methode bertrgt die Message ber das HTTP-Protokoll
3           // und erhlt eine Funktion als bergabgeparameter, die sie nach
4           // Beendigung der bertragung aufrufen kann.
5       }
```

Listing 2.20: Implementierung der Übertragung mittels HTTP in Swift

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

2.8.3 Konzept

2.8.4 Aufbau

2.8.5 ...

3 Praktische Umsetzung

4 Kostenaufstellung

5 Fazit

Abbildungsverzeichnis

Literaturverzeichnis

[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