Abschlussbericht LED-Confusion

Diese Dokumentation beinhaltet die gewonnenen Erkenntnisse aus dem Projekt 2, welches als Vorbereitung zur Bachelorarbeit genutzt wurde.

Projekt 2: Abschlussbericht
Patrik Aebischer, Elia Bösiger
Biel, 18. Januar 2018

Berner Fachhochschule
Hochschule für Technik und Informatik
Studiengang - Informatik

Inhaltsverzeichnis

Änderungsübersicht **1**

1 Einleitung **2**

1.1 Inhalt und Zweck des Dokuments *2*

2 Beschreibung des Projekts 2 ..... **2**

2.1 Idee *2*

2.2 Geplante Umsetzung *2*

3 Kommunikation zwischen Mobiltelefon und Tinkerforge **3**

3.1 Aufbau *3*

3.2 Funktionsweise *4*

3.3 Ergebnisse *4*

3.3.1 Asynchroner MQTT-Client verwenden 4

3.3.2 Benötigte Zeit der Kommunikation 5

3.3.3 Spezieller Methodenaufruf um LEDs einzuschalten 5

4 Erkennen von Lichtpunkten mit einer Kamera **6**

4.1 Technologische Auswahlmöglichkeiten *6*

4.1.1 SL-Project (basierend auf Open CV) 6

4.1.2 Image Play 6

4.1.3 Unity Game Engine 6

4.2 Entscheidung *6*

4.3 Relevante Open CV Klassen *7*

4.4 Arbeitsbericht *7*

4.4.1 Anleitung 7

4.5 Probleme *10*

4.5.1 Problem 1: Lichtverhältnis 10

4.5.2 Problem 2: Lichter nach Farbe erkennen 11

4.6 Aufbau der Lichterkennungs-Applikation *12*

4.6.1 LightDetectionController 12

4.6.2 LightDetectionView 12

4.6.3 Utils 12

5 Aussicht auf Bachelorarbeit **13**

5.1 Struktur des Systems *13*

5.2 Use Case *14*

6 Fazit **15**

Änderungsübersicht

Datum	Autor	Beschreibung der Änderung	Betroffene Kapitel
18.01.2018	Patrik Aebischer Elia Bösiger	Erste Version	Alle Kapitel

1 Einleitung

1.1 Inhalt und Zweck des Dokuments

Dieses Dokument beinhaltet einen Begleitbericht zum Modul "Projekt 2". In unserem Fall wurde das Projekt 2 als Vorbereitung zur Bachelorarbeit genutzt. Dieses Dokument hält fest, was wir alles getan und welche Erfahrungen wir gemacht haben. Weiter soll es uns während der Bachelorarbeit als "Nachschlagewerk" dienen. Während dem Projekt 2 haben wir diverse Tests durchgeführt, welche uns später bei der Bachelorarbeit helfen sollen Fehler zu vermeiden.

2 Beschreibung des Projekts 2

2.1 Idee

Wir haben das Projekt 2 als Vorbereitung zur Bachelorarbeit genutzt. Ziel der Bachelorarbeit wird es sein, mit Hilfe einer Androidapplikation zufällig angeordnete LEDs zu erfassen und zu Mappen. Dazu soll die Kamera des Mobiltelefons auf die LEDs gerichtet werden und auf "Start" gedrückt werden können. Nachfolgende Schritte sollen anschliessend automatisch ablaufen. Das Mobiltelefon meldet dem Controller der LEDs, dass die erste LED eingeschaltet werden soll. Die Kamera erfasst dies und merkt sich dessen Position. Hat sie das getan, soll die erste LED ausgeschaltet und die nächste eingeschaltet werden. Auch diese Position wird wieder von der Kamera erfasst. Dies geschieht so lange, bis alle LEDs durchgeschaltet und ihre Position vermerkt wurde. Die Applikation weiss nun welche LED sich wo befindet, sie wurden gemappt.

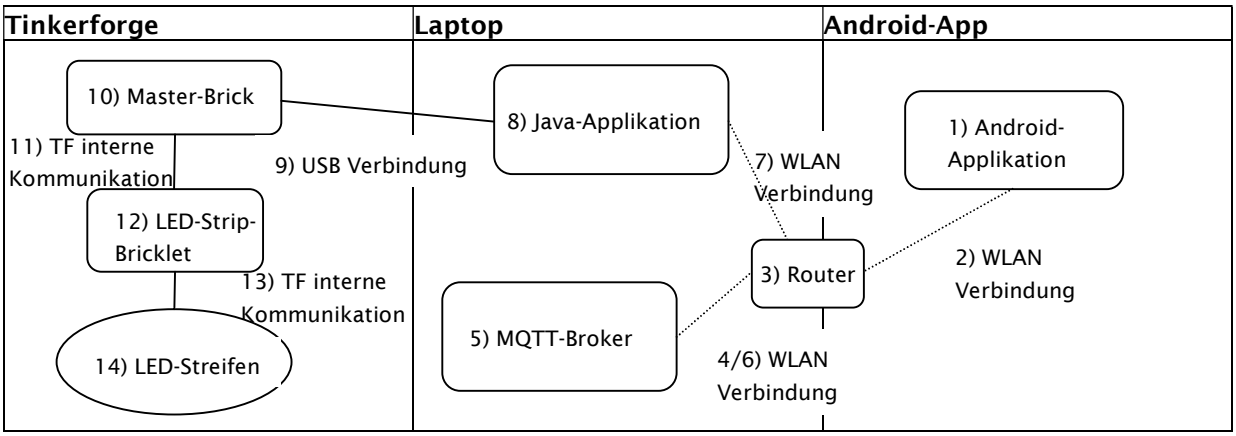
2.2 Geplante Umsetzung

Um das Ziel der Bachelorarbeit zu erreichen, haben wir verschiedene Möglichkeiten. Einige dieser Möglichkeiten werden wir im Verlauf von Projekt 2 testen und auf ihr Eignung untersuchen. Wir verwenden Java als Programmiersprache für die Android-Applikation sowie die Applikation, welche auf einem separaten Rechner laufen wird um die Tinkerforgekomponenten zu steuern. Die LEDs sowie deren eigentliche Ansteuerung kommt von Tinkerforge. Dies ist eine in sich geschlossene Technologie um Hardware anzusteuern. Sie bietet eine Java-Schnittstelle an, welche wir nutzen können. Zur Kommunikation zwischen Mobiltelefon und Tinkerforge soll das Nachrichtenprotokoll MQTT genutzt werden. Ziel ist es, die Kommunikation so zu gestalten, dass auch eine andere Plattform als Android das MQTT-Protokoll nutzen könnte um die LEDs anzusteuern. Ob das Ganze so funktionieren kann wie wir uns das Vorstellen, soll im Verlaufe des Projekts 2 herausgefunden und in diesem Dokument beschrieben werden.

3 Kommunikation zwischen Mobiltelefon und Tinkerforge

3.1 Aufbau

Um das Ziel unserer Bachelorarbeit überhaupt erreichen zu können. Müssen verschiedene Technologien zusammenspielen. Unter anderem brauchen wir verschiedene Komponenten von Tinkerforge, sowie einen MQTT-Broker. Mit einer Android-App, die in Java geschrieben sein wird, soll das ganze gesteuert werden können. Um die Kommunikation zwischen diesen verschiedenen Komponenten zu testen und uns bereits etwas mit den jeweiligen Technologien vertraut machen zu können, haben wir einen kleinen Testaufbau gemacht. Wir hoffen, dass bereits erste Schwierigkeiten auftauchen werden und wir die gleich schon beheben können, damit wir uns während der eigentlichen Bachelorarbeit nicht mehr darum kümmern müssen. Nachfolgend eine Zeichnung und eine Tabelle mit den verschiedenen Komponenten des Testaufbaus.



Nr	Komponente	Beschreibung
1	Android-Applikation in Java	Android-Applikation, welche das Durchschalten aller LEDs startet oder stoppt.
2	WLAN Verbindung	Eine WLAN-Verbindung zu dem Hausinternen Router.
3	Router	Hausinterner Router
4	WLAN Verbindung	Eine WLAN-Verbindung zu dem Hausinternen Router.
5	MQTT-Broker	Ein MQTT-Broker, welcher in unserem Testaufbau ebenfalls auf dem Laptop lief. Er ist aber zu 100% von der Java-Applikation, welche auch auf dem Laptop lieft getrennt. Dies bedeutet, dass der Broker später nicht am gleichen Ort laufen müsste wie die Java-Applikation. Wie das dann genau sein wird, ist noch zu entscheiden.
6	WLAN Verbindung	Eine WLAN-Verbindung zu dem Hausinternen Router.
7	WLAN Verbindung	Eine WLAN-Verbindung zu dem Hausinternen Router.
8	Java-Applikation	Diese Java-Applikation steuert die Bauteile von Tinkerforge. Sie nimmt Befehle von der Android-Applikation entgegen und führt diese aus. Diese Applikation könnte auch als Schnittstelle für eine andere Anwendung dienen. Zum Beispiel eine Web-Applikation. Diese müsste nur die Korrekten Befehle auf dem MQTT-Broker hinterlegen und könnte dann auch als Steuerung des LED-Strips dienen. Dies ist aber vorerst nicht vorgesehen. Im Testaufbau lief sie auf dem Laptop. Sie könnte aber ohne Probleme auch auf einem Raspberry-Pi laufen.

Nr	Komponente	Beschreibung
9	USB Verbindung	Verbindung der Steuerungsapplikation (Java-Applikation) zu den Tinkerforgebaugruppen.
10	Master Brick	Master Brick von Tinkerforge. Er steuert die einzelnen Bricklets von Tinkerforge und wird immer zwingend benötigt.
11	TF interne Kommunikation	Kommunikation zwischen Master Brick und LED-Strip-Bricklet. Diese Kommunikation wird von Tinkerforge bereitgestellt und funktioniert. Wir haben damit nichts zu tun.
12	LED-Strip-Bricklet	Der LED-Strip-Bricklet steuert den einen oder mehrere LED-Streifen. Er wird von der Java-Applikation über den Master Brick aufgerufen. Ihm wird gesagt, wann die LEDs an- oder ausgeschaltet werden sollen.
13	TF interne Kommunikation	Kommunikation zwischen Master Brick und LED-Strip-Bricklet. Diese Kommunikation wird von Tinkerforge bereitgestellt und funktioniert. Wir haben damit nichts zu tun.
14	LED-Streifen	Das sind die LEDs, welche vom LED-Strip-Bricklet angesteuert werden.

3.2 Funktionsweise

Die Funktionsweise ist verhältnismässig einfach. Auf dem Mobiltelefon soll ein Knopf gedrückt werden können, der das Durchschalten der LEDs startet. Drückt man auf einen Stopp-Knopf wird es wieder beendet. Sobald Start gedrückt wurde, published die App die Message "Next" auf dem Broker. Die Java-Applikation auf dem PC hat auf dem gleichen Topic subscribed und kriegt mit, dass die nächste Aktion durchgeführt werden muss. In diesem Fall ist es die nächste LED einschalten. Hat sie den Befehl an den Led-Strip-Bricklet abgesetzt, published die Java-Applikation ihrerseits ein "done" auf dem Broker. Dies wiederum kriegt die Mobile-App mit und published das nächste "next". Sind alle LEDs eingeschaltet, stellt die Java-Applikation automatisch um und schaltet bei einem "next"-Befehl die nächste LED (in diesem Fall wieder die erste) aus.

3.3 Ergebnisse

3.3.1 Asynchroner MQTT-Client verwenden

Das Ziel dieses Aufbaus war es ja, uns einmal mit den Komponenten, die wir später für die Bachelorarbeit brauchen werden, vertraut zu machen und eventuell bereits erste Probleme zu beseitigen. Dies ist uns bei der Kommunikation mit MQTT auch bereits gelungen. Wir hatten lange ein Problem beim wiederholten verschicken der Messages. Kommt eine Nachricht auf dem Broker an, wird eine Methode auf dem Client aufgerufen, in welcher man sagen kann, was zu tun ist. In unserem Fall wollten wir eine LED schalten und dies gleich mit einer Nachricht bestätigen. Macht man dies jedoch in der gleichen Methode, hängt sich die Applikation nach dem ersten LED auf. Wir haben herausgefunden, dass die von uns verwendeten Methoden von MQTT nicht dafür geeignet sind. Glücklicherweise stellt MQTT eine Lösung bereit. Es gibt eine Klasse, die das aufrufen der Methode "messageArrived" asynchron löst. Da die Klasse dies bereits von sich aus handelt, ist es für uns nicht mehr nötig, eigene Threads zu erstellen, was uns Arbeit abnimmt. Wir werden also für die Bachelorarbeit die Klasse "MqttAsyncClient" verwenden müssen und nicht wie bisher "MqttClient".

3.3.2 Benötigte Zeit der Kommunikation

Nachdem wir dieses Problem behoben hatten, interessierte uns, wie lange es wohl dauert um 50 LEDs ein- und dann wieder auszuschalten. Dies zu messen war relativ einfach und man braucht ziemlich genau 20 Sekunden. Dies ist ein relativ hoher Wert, man muss aber bedenken dass in dieser Zeit 200 Mal eine ankommende Message verarbeitet wurde. Das bedeutet aber, dass der Benutzer der App doch relativ viel Geduld haben muss. 50 LEDs sind schliesslich noch nicht so viele. Da noch die Idee im Raum steht, mehrere LEDs gleichzeitig einzuschalten und sie farblich zu trennen, könnten wir den Wert vielleicht noch herunterbringen. Allerdings haben unsere ersten Tests ergeben, dass es wohl ziemlich schwer sein wird, einzelne Farben zu unterscheiden. Dazu aber mehr im Bericht zur OpenCV-Bibliothek und unseren Tests dort.

3.3.3 Spezieller Methodenaufruf um LEDs einzuschalten

Etwas speziell ist in unseren Augen der Aufruf der Methode "setRGBValues" welche vom LED-Strip-Bricklet zur Verfügung gestellt wird. Die Methode verlangt die Parameter: "index", "length", "r", "g", "b". Hierbei ist "index" die Nummer der LED, wo das Schreiben beginnt, "lengt" die Anzahl LEDs die mit Parametern beschrieben werden sollen und "r", "g" und "b" ergeben die Farbe. Interessant ist, dass die letzten drei Parameter ein Array sind, das immer die Länge 16 haben muss. Nur wenn man wie wir, nur jeweils ein LED beschreiben will und das bei "index" auch so angibt, werden 15 der 16 Werte zwar geschickt, aber nie gebraucht. Das stellt für uns absolut kein Problem dar, hat uns aber etwas irritiert. Wir haben einige Tests vorgenommen, sehen aber keinen Weg das anders zu machen.

4 Erkennen von Lichtpunkten mit einer Kamera

4.1 Technologische Auswahlmöglichkeiten

4.1.1 SL-Project (basierend auf Open CV)

SL steht für Scene Library und ist ein von der Berner Fachhochschule entwickeltes 3D Framework. Es benutzt die Open Source Bibliothek Open Cv und kann für Real Time Rendering und Ray Tracing verwendet werden. Ray Tracing ist ein auf der Aussendung von Strahlen basierender Algorithmus zur Ermittlung der Sichtbarkeit von dreidimensionalen Objekten. Das Framework wurde in C++ entwickelt und läuft auf Windows, Linux, Mac OSX, Android und Apple iOS.

4.1.2 Image Play

Image Play ist ein Rapid Prototyping Tool zum Testen von Bildverarbeitungsalgorithmen und wurde ebenfalls von der Berner Fachhochschule entwickelt. Es bietet Möglichkeiten zum schnellen Zusammenklicken von verschiedenen Algorithmen. Die Oberfläche ist grafisch und bietet verschiedene Bausteine zum Bearbeiten einer Bilddatei.

4.1.3 Unity Game Engine

Unity Game Engine ist eine Laufzeit- und Entwicklungsumgebung für Spiele des Unternehmens Unity Technologies. Unity verwendet für das Erfassen von 3 dimensionalen Objekten ein Shading Verfahren. Nebst dem Erfassen von Objekten bietet Unity auch das Darstellen von Animationen und die Entwicklungsumgebung für die Programmlogik an.

4.2 Entscheidung

Die C++ Library Image Play würde sich für einen Prototypen zwar anbieten, bietet aber keine Schnittstelle für Android. Unity Game Engine ist ebenfalls nicht geeignet, da es in C# geschrieben und kostenpflichtig ist. Wir haben uns für SL entschieden. Da wir schon verschiedene Tests mit Open CV gemacht haben und SL bereits eine Schnittstelle für Android bietet, ist SL perfekt auf unser Projekt zugeschnitten.
Im Projekt 2 haben wir uns jedoch nur auf das Erkennen von Lichtpunkten mit einer Kamera begrenzt. Da SL Project auch auf Open CV basiert, benutzen wir in diesem Projekt vorerst die Open CV Bibliothek für Java.

4.3 Relevante OpenCV Klassen

Klasse	Beschreibung
VideoCapture	Mit der *VideoCapture* Klasse können Videodateien und Bildsequenzen gelesen werden.
Mat	Bildinformationen werden in Open CV in dem Matrizen-Datentypen Mat verwaltet. Die Mat Klasse enthält Informationen zu der Dimension des Bildes sowie Farben der einzelnen Pixel.
ImgProc	Die Klasse *ImgProc*stellt (statische) Bildverarbeitungsmethoden zur Verfügung. Zum Beispiel um Konturen zu finden und zu zeichnen.

4.4 Arbeitsbericht

Am Anfang des Projektes stellte sich die Frage, wie man eigentlich Licht am besten erkennen kann. Wir sind auf zwei Ideen gestossen. Erster Gedanke war, dass man helle Punkte auf einem Bild erkennen soll. Da jedoch die Lichtverhältnisse nicht immer gleich sind haben wir uns auch noch mit der Idee befasst Lichter anhand ihrer Farbe zu erkennen. Zum Beispiel eine rote LED auf dem Bild zu suchen. OpenCV bietet Funktionen für beide dieser Ideen. Deshalb haben wir diese gleich beide implementiert.

4.4.1 Anleitung

Das *VideoCapture*-Objekt in unser Applikation liefert uns alle 30 Millisekunde ein Bild um auf diesem die hellsten Regionen zu erkennen. Um die Lichter besser zu erkennen konvertieren wir das Bild (in OpenCV: *Mat*-Objekt) zuerst in ein Schwarz-Weiss Bild. Um das Rauschen auf dem Bild zu reduzieren, sodass die Umrandungen eines Lichtes besser erkennt werden können, verwenden wir zusätzlich einen Verwischungs-Filter.

```
// Das sourceMat-Objekt wird verwischt und im destinationMat abgelegt
Imgproc.blur(sourceMat, destinationMat, new Size(11, 11));

// Das sourceMat-Objekt wird schwarz-weiss gefärbt und im destinationMat abgelegt
Imgproc.cvtColor(sourceMat, destinationMat, Imgproc.COLOR_BGR2GRAY);
```

Der Output sieht wie folgt aus:

Auf der linken Seite ist das originale Bild und auf der rechten Seite das leicht verschwommenem und schwarz-weisse Bild.

Um die zwei Lichter ausfindig zu machen werden nun alle Pixel aus dem Bild entfernt, die nicht hell genug sind. Wobei in OpenCV die Wert 0 schwarz und 255 weiss entsprechen. Im folgenden Beispiel werden alle Pixel die eine Farbe unter 220 haben entfernt.

```
// Das srcMat-Objekt entfernt Pixel unter 220 und legt das Ergebnis im  destMat ab
Imgproc.threshold(srcMat, destMat, 220, 255, Imgproc.THRESH_BINARY);
```

Der Output sieht wie folgt aus:

Auf der linken Seite ist das originale Bild und auf der rechten Seite das bearbeitete. Nun werden alle hellen Punkte weiss angezeigt und der Rest wird schwarz eingefärbt.

Die zwei Lichter sind gut auf dem Bild zu erkennen. Jedoch bilden beide Lichter sogenannte Geisterflecke die jeweils leicht links unterhalb des Lichtes platziert sind. Um diese zu entfernen verwenden wir die Funktionen *erode* und *dilate*.

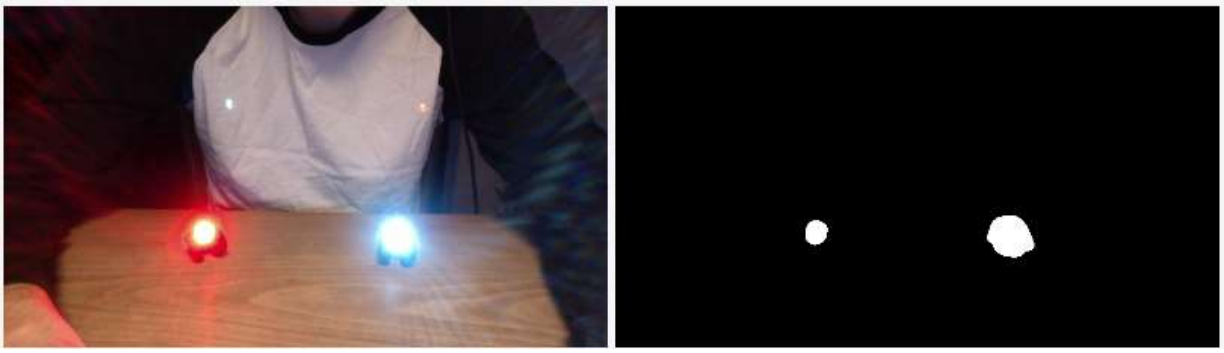
```
// dilate und erode Element
Mat dilateElement = Imgproc.getStructuringElement(Imgproc.MORPH_ELLIPSE);
Mat erodeElement = Imgproc.getStructuringElement(Imgproc.MORPH_ELLIPSE);

// 2 mal schrumpfen
Imgproc.erode(srcMat, mat1, erodeElement);
Imgproc.erode(mat1, mat2, erodeElement);

// bestehende geschrumpfte Flecken wieder ausdehnen
Imgproc.dilate(mat2, mat3, dilateElement);
```

Um die Geisterflecken definitiv zu entfernen wird das *Mat*-Objekt gleich zweimal durch die *erode*-Funktion geführt. Somit sind alle Flecken und auch die zwei Lichter entfernt. Um die zwei Lichter (grosse Flecken) wieder darzustellen wird die *dilate*-Funktion verwendet.

Das Ergebnis kann sich sehen lassen:



Der letzte Schritt ist die Lichter auf dem originalen Bild zu markieren. Um die Umrandungen der Lichter zu finden stellt die Klasse *ImgProc* die statische Methode *findContours* zur Verfügung. Pro Umrandung eines Lichtes wird ein Eintrag in der ArrayList *counters* gemacht.

```
List<MatOfPoint>contours = new ArrayList<>();
Mat hierarchy = new Mat();

// findet die Umrandungen und schreibt sie in die ArrayList countours
// Die Punkte der Umrandungen befinden sich im Mat-Objekt hierarchy
Imgproc.findContours(maskedImage, contours, hierarchy,
Imgproc.RETR_EXTERNAL, Imgproc.CHAIN_APPROX_SIMPLE);
```

Mit der Funktion *drawContours*der Klasse *ImgProc* können nun die Umrandungen eingezeichnet werden.

```
// Die Umrandung wird mit Blau auf dem Mat-Objekt „frame“ eingezeichnet
// Der letzte Parameter der Funktion ist die Dicke der Linie
Imgproc.drawContours(frame, contours, contours[i], new Scalar(250, 0, 0), 10);
```

Das Ergebnis sieht nun wie folgt aus:

Das Licht wird auf dem originalen Bild mit blauer Farbe umrandet.

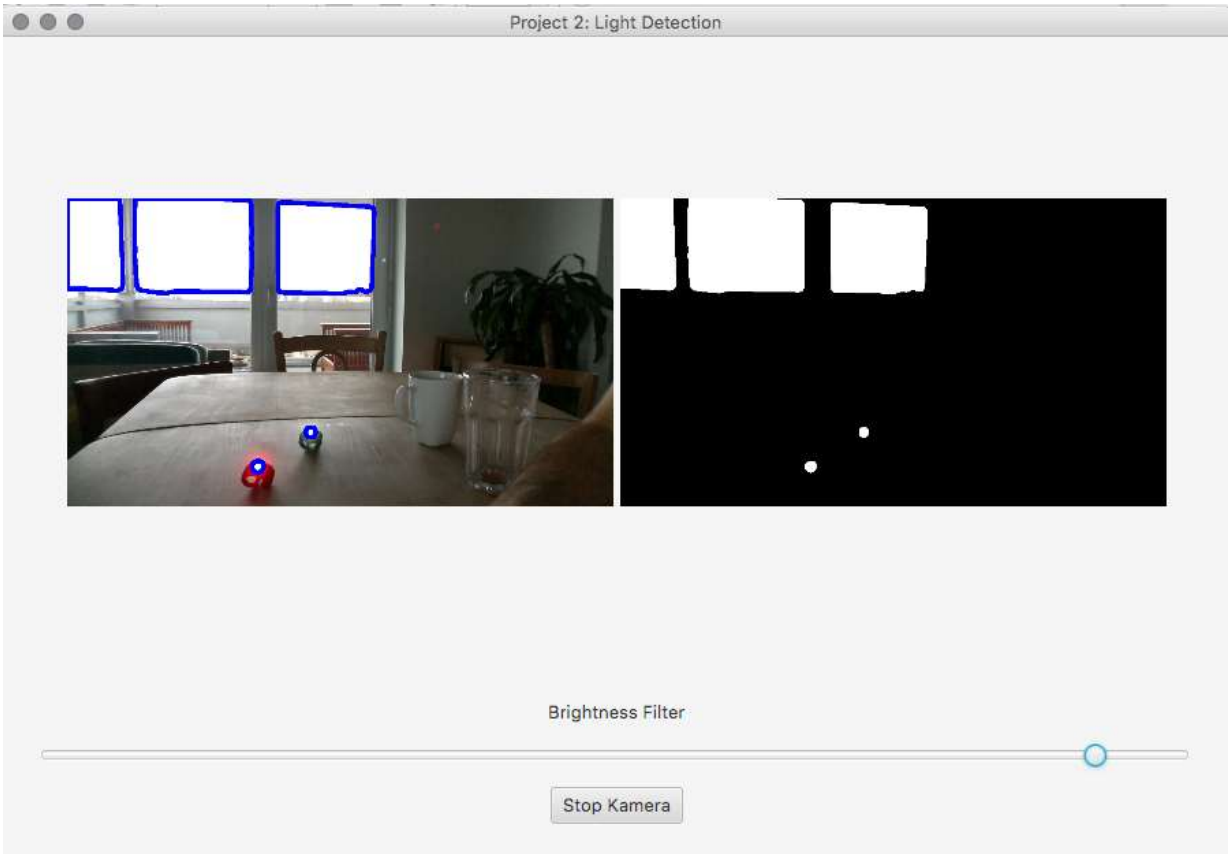
4.5 Probleme

4.5.1 Problem 1: Lichtverhältnis

Das Lichtverhältnis ist nach wie vor ein Problem. Um diesem Problem entgegen zu wirken haben wir einen Regler auf der Applikation definiert. Mit diesem kann man einstellen ab welcher Helligkeit ein Licht erkennt werden soll. So werden nun nicht mehr alle Farben unter 220 entfernt sondern alle Farben unter dem Wert des Reglers.

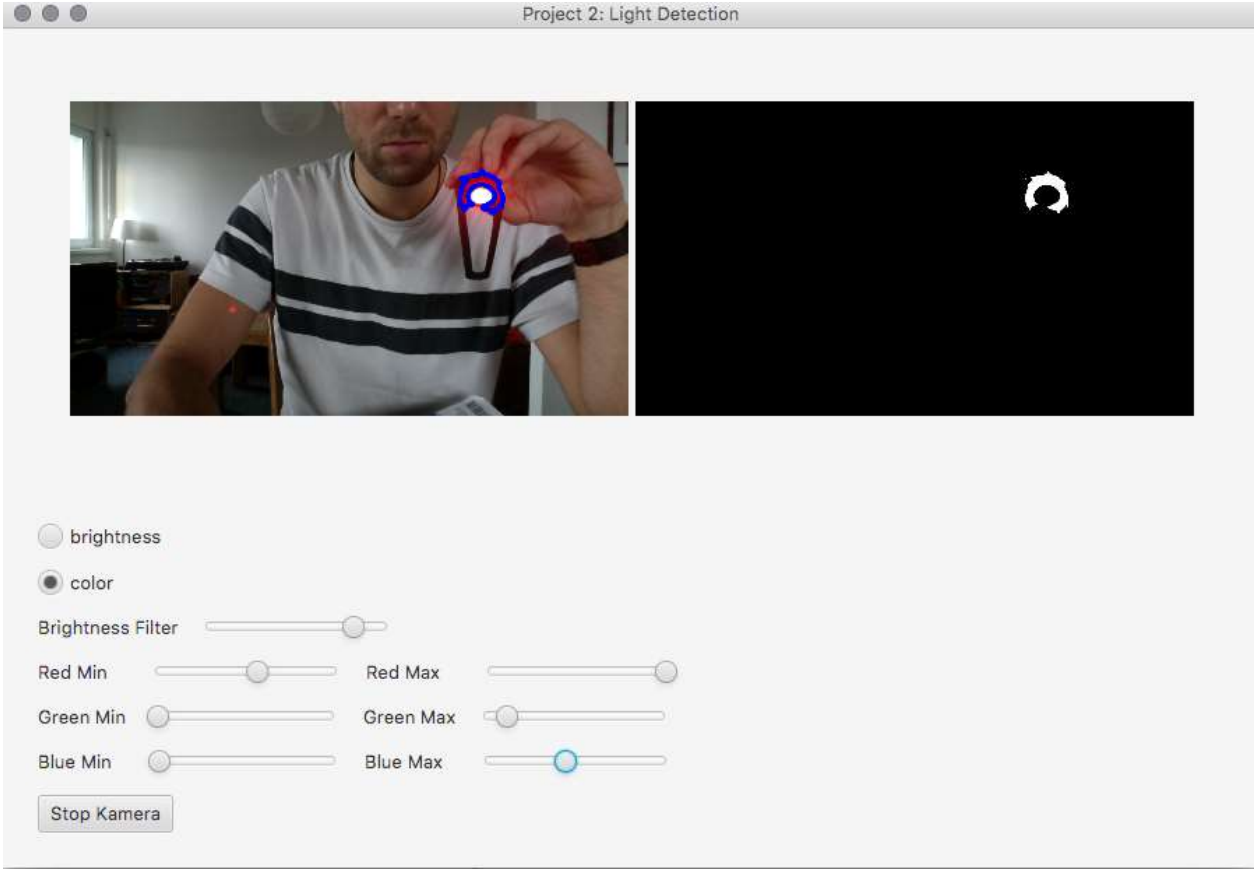
```
// Das srcMat-Objekt entfernt Pixel unter dem Wert des Reglers und legt das Ergebnis
im  destMat ab
Imgproc.threshold(srcMat, destMat, regler.getValue(), 255, Imgproc.THRESH_BINARY);
```

So werden schon einige unnötige helle Flecken nicht mehr erkannt. Wenn man jedoch die Lichter vor einem Fenster versucht zu erkennen ist dies zwar möglich, die Helligkeit vom Himmel wird jedoch auch als Licht erkannt:



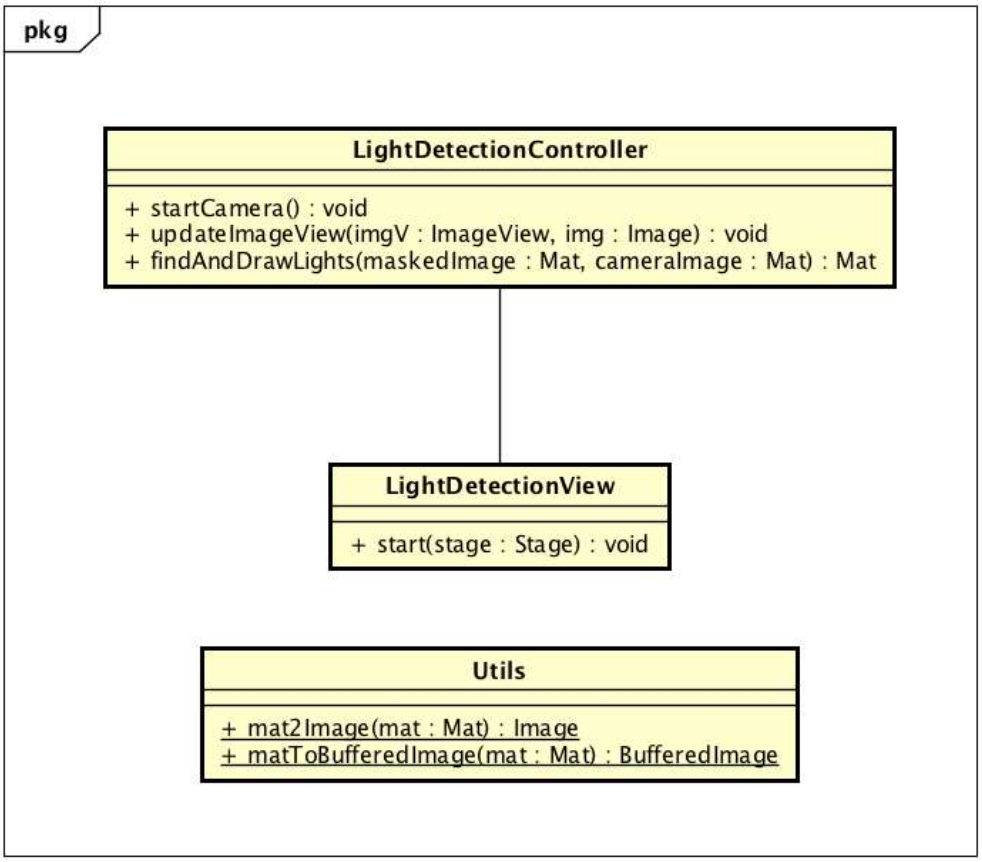
Bei den Tests ist uns zudem aufgefallen, dass es vermutlich besser sein wird, wenn wir die Lichter nicht all zu hell einstellen. Glücklicherweise kann man die Helligkeit der Lampen mit Tinkerforge sehr einfach regeln. Erste Tests haben ergeben, dass es vermutlich besser ist, den Helligkeitswert sehr tief, bei circa Stufe 10 oder 20 von total 255 Stufen zu halten. Die Kamera erkennt dann den Helligkeitsunterschied. Lässt man die LED so stark leuchten wie es möglich ist, filtert bereits das Objektiv das helle Leuchten raus und wir haben somit keine Möglichkeit einzugreifen.
Ein Lösungsweg wäre, einen weiteren Filter hinzuzufügen. Dieses Filter sollte die Grösse des Lichtes definieren. Da alle LED auf dem Bild ungefähr gleich gross sind könnte man eine Grösse mit einer bestimmten Toleranz festlegen. Dadurch sollten nur noch die LEDauf dem Bild erkannt werden.

4.5.2 Problem 2: Lichter nach Farbe erkennen
Lichter anhand ihrer Farbe zu erkennen erwies sich als fast unmöglich. Erstens wird ein rotes Licht nicht als Rot wahrgenommen. Zweitens findet der Filter irgendwo im Bild die gesuchte Farbe (bspw. Auf einem Tisch).
Hier ein Beispiel dazu:



In diesem Beispiel wird nach rotem Licht gesucht. Die Regler unter der Kamera können konfiguriert werden um grünes, rotes oder blaues Licht zu erkennen. Wenn man ein wenig an den Reglern schraubt kann die Applikation das Licht selber nicht erkennen, jedoch wird der rote Schein um das Licht als Rot erkannt.

4.6 Aufbau der Lichterkennungs-Applikation



4.6.1 LightDetectionController
Der *LightDetectionController* implementiert die Applikationslogik. Er reagiert auf den Start/Stopp Button der Kamera und zeigt so den Video Stream auf der grafischen Benutzeroberfläche mit den gefundenen und markierten Lichtern.

Funktion	Beschreibung
startCamera()	Die Funktion „startCamera“ startet die im Computer integrierte Kamera und zeigt den Stream auf der grafischen Benutzeroberfläche an.
updateImageView (ImageView, Image)	Die Funktion „updateImageView“ zeigt ein bearbeitetes Image an. Wenn man Beispielsweise auf einem Bild ein Licht gefunden hat, umrandet man das Licht auf einem Image. Das neue Image kann nun auf der ImageView platziert und schlussendlich angezeigt werden.
findAndDrawLights (Mat, Mat)	Die Funktion findAndDrawLights sucht auf dem bearbeitetem Mat-Objekt nach gefundenen Lichtern (weisse Objekte) und umrandet diesen Bereich auf dem originalen Mat-Objekt (unbearbeitete Kameraübertragung).

4.6.2 LightDetectionView
Die Klasse *LightDetectionView* ist eine in JavaFX programmierte grafische Benutzeroberfläche. Der Hauptteil der Benutzeroberfläche ist es den Video-Livestream anzuzeigen. Die *start*-Methode wird von JavaFX verwendet um die Applikation zu starten.

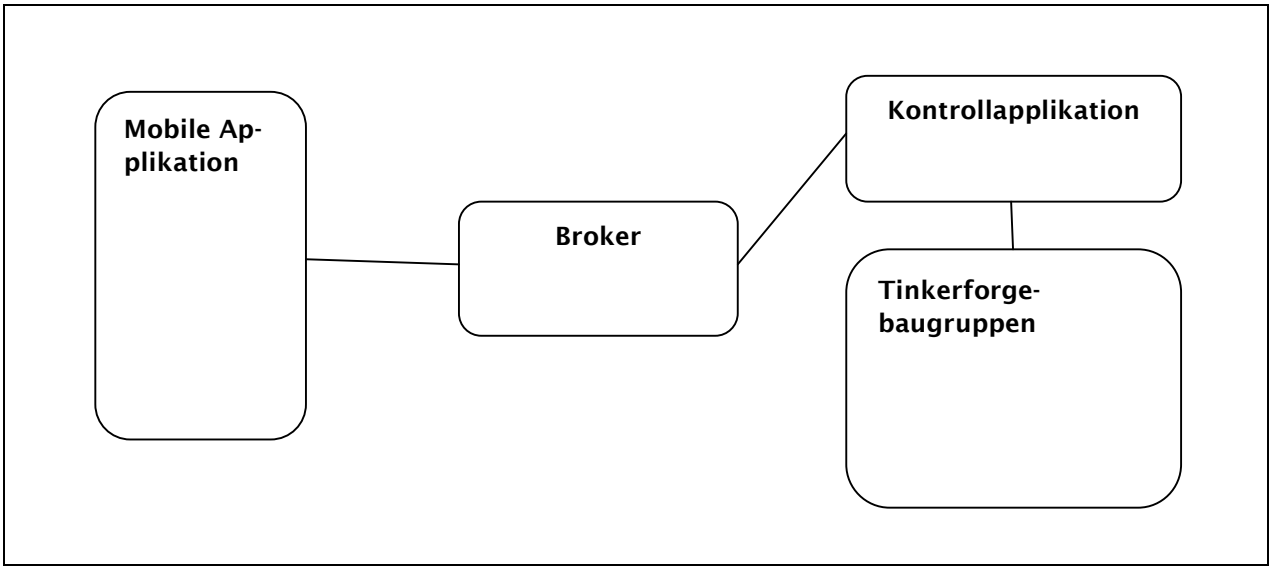
4.6.3 Utils
Die *Utils*-Klasse wird verwendet um die Konversation von OpenCV zu JavaFXzu gewährleisten. Die Funktionen der *Utils*-Klasse sind statisch und werden vom *LightDetectionController* aufgerufen.

5 Aussicht auf Bachelorarbeit

Bereits haben wir einige technische Details gefunden, auf die wir bei der Bachelorarbeit Acht geben müssen. Hier wollen wir uns noch einige Gedanken darüber machen, wie die Projektstruktur der Bachelorarbeit aussehen könnte und was für UseCases es gibt. Diese Gedanken sollen explizit als Brainstorming gelten und sind **nicht** verbindlich für die Bachelorarbeit.

5.1 Struktur des Systems

Dies ist die voraussichtliche Struktur unseres Systems.



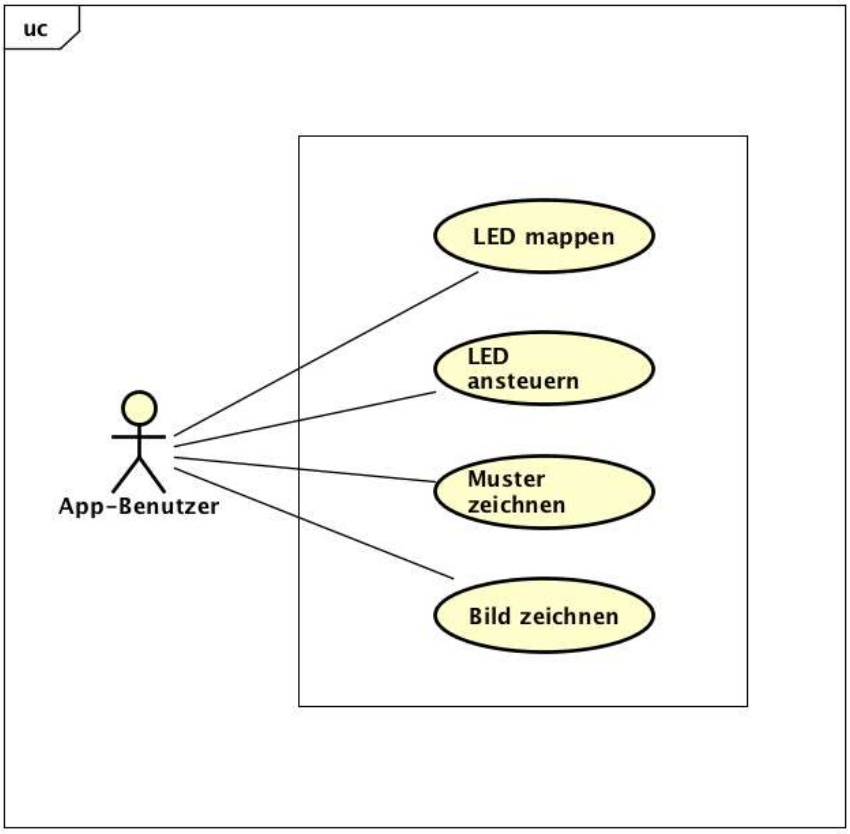
- **Mobile Applikation:** Diese Applikation, welche auf dem Smartphone des Benutzers läuft, wird in Java geschrieben sein und mit Android Studio entwickelt werden. Sie muss mindestens einen MQTT-Client enthalten, sowie die Daten verwalten, welche über die Kamera erfasst werden und uns von der Bibliothek OpenCV bereitgestellt werden. Sie beinhaltet zudem ein gut ausgearbeitetes GUI, auf welchem der Benutzer sämtliche Einstellungen vornehmen kann (Einzelne LED einschalten, Lauflicht, einfache Muster etc). Die Koordinaten der einzelnen LED werden von dieser Applikation errechnet und gespeichert. Der MQTT-Client kommuniziert mittels eines **Brokers** mit der **Kontrollapplikation**.

- **Kontrollapplikation:** Diese Applikation steuert die **Tinkerforgebaugruppen**. Auch sie hat einen MQTT-Client, der mit dem gleichen Broker kommuniziert wie die **Mobile Applikation**. Sie läuft nicht auf einem Handy sondern irgendwo im Hintergrund. Höchst wahrscheinlich wird sie auf einem RaspberryPi installiert werden, welches sich örtlich dort befinden muss, wo auch die **Tinkerforgebaugruppen** verbaut sind. Dies kommt daher, dass die **Tinkerforgebaugruppen** nicht ohne weiteres über ein WLAN kommunizieren können.

- **Broker:** Der Broker wird entweder auf demselben RaspberryPi laufen wie die **Kontrollapplikation** oder, was sehr viel schöner wäre, auf einem eigens für ihn eingerichtetem Server mit einer eigener Domain. Ob das zu realisieren ist, ist noch in Abklärung.

- **Tinkerforgebaugruppen:**Unter die Tinkerforgebaugruppen fällt alles, was von Tinkerforge kommt. Dazu zählen die LED-Strips, sowie die Bricks und Bricklets. Alle diese Baugruppen sind fest miteinander verdrahtet und müssen sich deshalb am gleichen Ort befinden.

5.2 UseCase



- **LED-Mapping:** Der Benutzer kann die Kamera auf eine oder mehrere LED richten und diese automatisch mappen lassen.

- **LED ansteuern:** Der Benutzer kann einzelne LED individuell ansteuern. Er kann zum Beispiel sagen, diese LED hier oben in der rechten Ecke, will ich grün einschalten.

- **Muster zeichnen:** Der Benutzer kann möglichst einfach verschiedene Muster von der Lichterkette darstellen lassen. Als Muster gelten zum Beispiel, die LED in den Regenbogenfarben anzeigen, sie blinken oder durchschalten lassen. Welche Muster das genau sein werden, ist noch nicht definiert.

- **„Bild zeichnen“:** Der Benutzer kann eine Bilddatei (welches Format ist noch nicht definiert) nehmen und dieses Bild von den LED anzeigen lassen.

6 Fazit

Zum Abschluss unserer Projektarbeit ist es an der Zeit das Modul „Projekt 2“ zu analysieren und die geeigneten Lehren zu ziehen.
Die zweite Projektarbeit hat uns einiges abverlangt. Die geleistete Arbeit ist jedoch mit neuem Wissen und interessanten Erkenntnissen belohnt worden. Die Hauptpunkte daraus sind nachfolgend aufgelistet:

- Die **Zusammenarbeit im Team** hat sehr gut funktioniert. Auch die Meetings mit Reto Koenig halfen uns jeweils die Aufgabe und das Ziel nicht aus den Augen zu verlieren.

- Das **laufende Nachtragen der Dokumentation** erwies sich als herausfordernde Aufgabe und wurde etwas vernachlässigt. Dennoch ist es uns gelungen alle Erkenntnisse vollständig und korrekt in der Dokumentation festzuhalten.

- Die **Tinkerforge-Komponenten** können nun mit Java angesprochen werden. Mit einer Java Applikation auf einem Android-Gerät kann eine Lichterkette von LEDs durchgeschalten werden.

- Die **Kommunikation mit dem MQTT-Protokoll** zwischen einem Android-Gerät und den Tinkerforge LEDs hat uns keine grosse Mühe bereitet und funktioniert einwandfrei.

- Wir sind begeistert von der **Open CV Library**. Sie bietet viele interessante Funktionen im Bereich vom „Image Processing“. Lichter können nun über unsere Java Applikation erkannt werden. Es hat sich herausgestellt, dass die Methode Lichter anhand ihrer Helligkeit zu erkennen besser ist als die Methode mit den Farben. Für die Bachelorarbeit gilt es nun die Funktionen mit Java auf einem Android-Gerät zum Laufen zu bringen.

Schlussendlich können wir sagen, dass wir uns intensiv mit den Technologien auseinandergesetzt haben und bereits erste kritische Punkte und zum Teil auch schon gelöst haben. Für diejenigen Punkte, für welche wir noch keine Lösung haben, sind wir optimistisch, im Verlauf der Bachelorarbeit noch eine zu finden. Wir gehen davon aus, dass wir die Bachelorarbeit grundsätzlich mit diesen Technologien realisieren können. Natürlich kann es vorkommen, dass es noch die eine oder andere Änderung gibt, das sollte jedoch kein Problem darstellen.