# KI-gestützter QR-Code Reader

Fritz-Hopf-Technikerschule Nördlingen

Technikerarbeit im Fachbereich Elektrotechnik

08/2023 – 01/2024

**Autoren:**
Johannes Förschner / Tobias Vogt
TE2 2023/2024

**Betreuer:**
Herr Michael Pfäffl

FRITZ-HOPF-
TECHNIKERSCHULE
NÖRDLINGEN

# 1. Konzept

## 1.1 Problemstellung

QR-Codes sind inzwischen fester Bestandteil einer sich immer stärker digitalisierenden Welt. Jedoch weisen auch diese Fehler bzw. Probleme auf. Beispielsweise sind QR-Codes nicht mehr auslesbar, wenn sie weit entfernt und die einzelnen Module des Codes nicht mehr klar unterscheidbar sind oder wenn der Code aus einem zu steilen Blickwinkel aufgenommen wird. Während klassische Bildverarbeitung hier an seine Grenzen kommt nutzen wir künstliche Intelligenz um nicht mehr erfass- oder auslesbare Codes grafisch aufzubereiten und dadurch wieder lesbar zu machen.

## 1.2 Idee

Die weit entfernten Codes werden durch eine Skalierungs-KI nahezu verlustfrei skaliert. Die schrägen Codes werden mit Hilfe einer Keypoint-KI vor der Weitergabe an den Reader aufgerichtet. Zur Bereitstellung des Live-Bildes wird eine Webcam verwendet. Diese hat ein geringes Blickfeld von 68°-75°, durch welches die Pixeldichte auf der horizontalen Achse gesteigert ist. Diese höhere Pixeldichte ist für die spätere Bearbeitung wichtig.

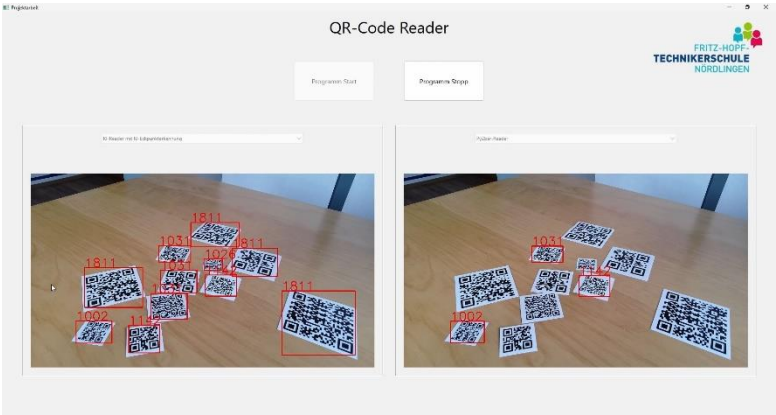
*Im Projekt verwendete Kamera*

Ziel unserer Projektarbeit ist es für die oben genannten Probleme Lösungen bzw. Möglichkeiten der Optimierung zu finden. Hierfür wurden mehrere Systeme entwickelt, in einer eigens erstellten Testumgebung ausgewertet und im Anschluss miteinander verglichen.

Im Folgenden wird nur auf die erfolgreichste Version eingegangen.

# 2. Programm

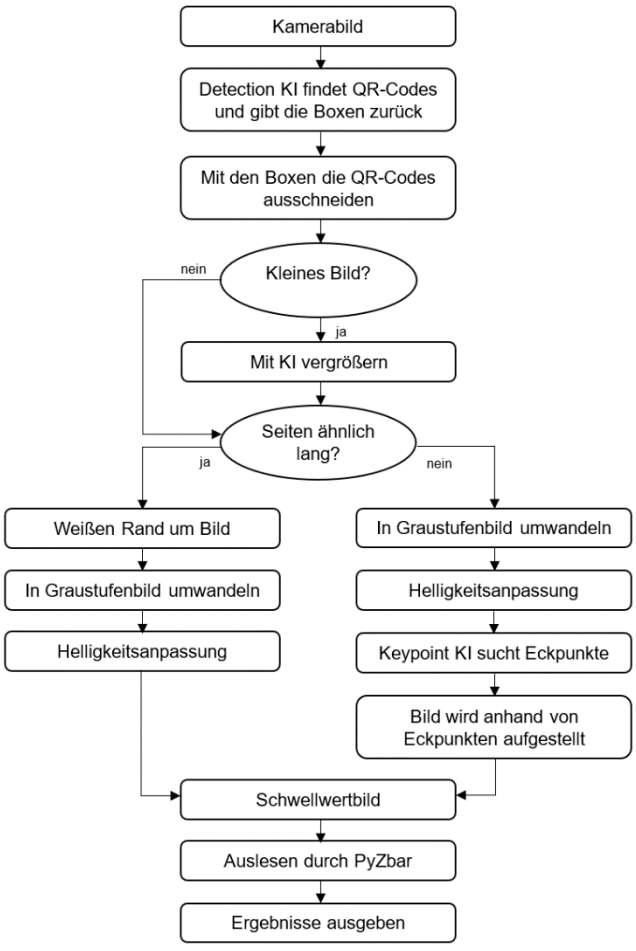
Für die Veranschaulichung des Projekts wurde das Programm mit eigener Benutzeroberfläche in Python erstellt. Um eine einfache Bedienung des Programms zu ermöglichen wird dieses in eine EXE-Datei (Windows Anwendung) umgewandelt. Somit kann das Programm mit dem Aufruf der Datei gestartet werden, ohne dass Python und die benötigten Bibliotheken installiert werden.




*Grafische Benutzeroberfläche der selbst programmierten Testumgebung*

## 2.1 Programmablauf

Das Programm wird über die Benutzeroberfläche gestartet und befindet sich anschließend in einer Dauerschleife. Zu Beginn eines jeden Zyklus wird das aktuelle Kamerabild erfasst. Dieses Bild wird an unsere „Detection KI“ weitergegeben, welche im Bild nach QR-Codes sucht, diese durch Rechtecke markiert und deren Koordinaten zurückgibt. Durch die Koordinaten der Boxen werden die QR-Codes aus dem Originalbild ausgeschnitten und anschließend einzeln weiterverarbeitet. Da die KI auch QR-Codes, welche weiter entfernt sind, erkennt, muss das Bild auf seine Größe überprüft und gegebenenfalls durch eine weitere KI




*Visuelle Darstellung des Programmablaufs*

vergrößert werden. Im nächsten Schritt wird das Bild wird nun in ein Graustufenbild umgewandelt. Anschließend muss die Helligkeit des Graustufenbildes, für die spätere Umwandlung in ein Schwellwertbild, angepasst werden. Nun werden Bildhöhe und Breite miteinander verglichen. Aufgrund des quadratischen Aufbaus eines QR-Codes ist bei einer starken Abweichung auf einen schrägen Code zu schließen. Das Bild des QR-Codes muss für die weitere Verarbeitung also aufgerichtet werden. Hierfür müssen zuerst die Eckpunkte in einer bestimmten Reihenfolge identifiziert werden. Diese Aufgabe übernimmt eine extra dafür trainierte Keypoint-KI. Anhand der erkannten Eckpunkte wird ein neues Bild erstellt, in welchem diese an definierte Punkte im neuen Bild übertragen werden. Die weiteren Pixel des alten Bildes werden anschließend passend transformiert und in das neue Bild geschrieben. Sind die Seitenverhältnisse bereits bei der Lageerkennung passend, wird um das Bild eine weiße Randzone hinzugefügt. Anschließend findet eine Umwandlung in ein Grauwertbild statt, an welchem eine Helligkeitsanpassung vorgenommen wird. Nach beiden Vorgängen wird das resultierende Bild in ein Schwellwertbild konvertiert und durch den Open Source PyZbar QR-Code Reader ausgelesen.
Die Ergebnisse des Readers werden in das Kamerabild eingetragen und über die Benutzeroberfläche dargestellt. Anschließend startet der Zyklus von neuem.

## 2.2 Multithreading

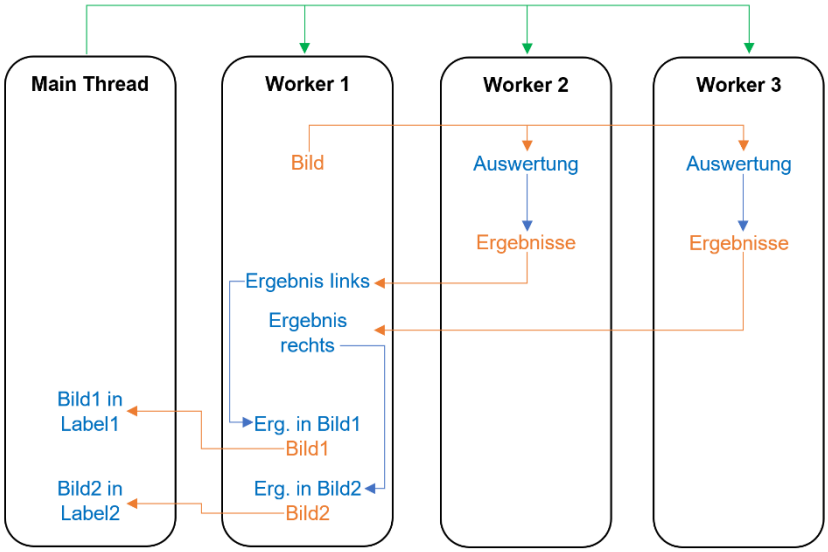
Aufgrund des sequentiellen Ablaufs von Python-Code wird das Programm auf mehrere Threads aufgeteilt. Threads ermöglichen ein paralleles Abarbeiten von Programmabläufen. Ohne Multithreading würde die Benutzeroberfläche während der Bearbeitung des restlichen Programms „einfrieren“ und auf keine Eingabe des Benutzers reagieren, bis der aktuelle Programmteil abgearbeitet ist. Des Weiteren würde die Darstellung der Webcam weniger „flüssig“ erscheinen, da die Bilder bei jedem Programmdurchlauf nur einmal aktualisiert werden würden. Bei einer Programmlaufzeit von 100ms entstünde beispielsweise eine Framerate von 10 fps (frames per second). Kinofilme und Streamingdienste nutzen standardmäßig eine Frame-Rate von 24 fps, um eine flüssige Wahrnehmung zu gewährleisten. Sind in einem Programm mehrere Threads voneinander abhängig, beziehungsweise

benötigen sie Informationen voneinander, kann dies zu folgendem Fehlverhalten führen:

- Dead-Lock: Ein Thread wartet endlos auf Informationen eines anderen Threads und macht währenddessen keinen Fortschritt.

- Live-Lock: Zwei oder mehr Threads sind durch eine unvorteilhafte Informationsweitergabe in Endlosschleifen gefangen und werden ihren Prozess nie beenden.

Aus diesen Gründen kann der Informationsaustausch zwischen den Threads nicht durch das Beschreiben und Lesen einer Variable realisiert werden, sondern muss durch Signale und Slots erfolgen.




*Veranschaulichung der Aufteilung des Programms in mehrere parallele Threads*

Hier wird die Erfassung des Bildes und das Eintragen der Ergebnisse, sowie die Auswertung der Bilder an separate Worker Threads vergeben während der Main Thread für die Benutzeroberfläche zuständig ist.

## 2.3 Programmerstellung

Die Methoden zur Verwendung der KI wurden als Python-Bibliothek eigens von uns erstellt und werden gemeinsam mit weiteren benötigten Bibliotheken am Anfang des Hauptprogramms importiert.

Anschließend findet im Hauptprogramm die Realisierung der Benutzeroberfläche und des Programmablaufs statt. Während das Programm anschließend in Python bereits

ausführbar ist, muss für die EXE-Datei eine zusätzliche Funktion einprogrammiert werden. Diese wird benötigt, da bei der Umwandlung Probleme entstehen. Bei dieser wird der Programmcode von der Programmiersprache Python zu C übersetzt.

# 3. Trainingsdaten

## 3.1 Datensätze erstellen

Um die KIs zu trainieren, wurden Trainingsdatensätze mit eigenen Bildern erstellt. Bei den Codes handelt es sich um computergenerierte QR-Codes welche ausgedruckt und mit dem jeweiligen Inhalt und der Versionsnummer beschriftet werden. Anschließend wurden mit einem eigens entwickelten Programm die zum Annotieren benötigten Bilder aufgenommen. Hierbei wurde auf eine deutliche Variation der Blickwinkel und der Entfernung der Codes geachtet. Das Programm ist in der Lage in einem vorgegebenen Speicherpfad eigene Ordner zu erstellen und in diese die Bilder, mit angegebenen Namen der aktuell fotografierten QR-Codes, zu speichern. Durch die genaue Datierung kann das trainierte System überprüft und mögliche Mängel festgestellt werden.

## 3.2 Datensätze annotieren

*Beispiel für ein annotiertes Bild*

Die erfassten Datensätze müssen vor dem Trainieren noch annotiert werden. Das bedeutet, es müssen auf allen Bildern alle QR-Codes einzeln mit einer Box umrandet

werden, um die Position im Bild zu kennzeichnen. Zusätzlich müssen in jedem Code die Eckpunkte markiert werden. Bei diesen muss darauf geachtet werden, dass sie immer in derselben Reihenfolge mit demselben Startpunkt markiert werden. Das Annotieren kann über frei verfügbare Programme durchgeführt werden. Von dort können die annotierten Datensätze dann, im für die KI richtigen Format, exportiert und für das Training genutzt werden. Um ein erfolgreiches Trainieren zu gewährleisten, werden große Datensätze benötigt. In diesem Projekt wurden ca. 1500 Bilder mit jeweils einem oder mehreren QR-Codes annotiert. Gesamt nahm der Prozess des Trainings mit der Erstellung der Datensätze rund 60 Stunden ein.

# 4. Fazit

Für die abschließende Auswertung wurden zwei Datensätze mit jeweils ca. 700 Bildern erstellt, wobei ein Datensatz ausschließlich schräge, der andere weit entfernte QR-Codes enthielt. Die Datensätze wurden den Open Source Readern von OpenCV und PyZbar, sowie unserem finalen Reader „KeypointAI“ und dessen Vorgängerversion „KonturAI“ zum Auswerten gegeben.

Die Ergebnisse zeigen, dass das im Rahmen dieser Arbeit entwickelte System die Auslesequote deutlich erhöht. Da das konventionelle Ausleseverfahren bei gut erkennbaren Codes bereits sehr gut ist und durch die hohe Fehlertoleranz selbst bei Lichtreflexionen und Verschmutzung fast keine Probleme auftreten, sind bei geraden QR-Codes nahezu keine Verbesserungen durch die KI-Unterstützung möglich. Im Vergleich zu Readern ohne unsere Unterstützung konnten wir die Ausleserate bei schrägen oder weit entfernten Codes allerdings um ca. 135% erhöhen. Selbst im Vergleich zu öffentlich verfügbaren, ebenfalls durch KI unterstützten QR-Code Readern haben wir eine deutlich höhere Ausleserate.

```
Beurteilung bei schrägen QR-Codes:
        OpenCV: 41
        PyZbar: 171
        KeypointAI: 396
        KonturAI: 333
```

*Beurteilung bei schrägen QR-Codes*

```
Beurteilung bei entfernten QR-Codes:
        OpenCV: 204
        PyZbar: 237
        KeypointAI: 563
        KonturAI: 563
```

*Beurteilung bei fernen QR-Codes*