

2020

Sudoku-Solver

PYTHON-PROJEKT

JULIAN JANY, GERALD MITTERECKER, MICHAEL NEUHOLD

Sudoku-Solver

Projekt	2
Definition	2
Anforderungen	2
Muss-Anforderungen	2
Kann-Anforderungen	2
Feature-Requests (nach live-demo ;)	2
Entwicklungsteam	2
Installations- und Inbetriebnahmeanleitung	3
Projektverzeichnis	3
Abhängigkeiten installieren	3
Ausführung	3
Projekt-Architektur	4
Main	4
Sudoku-Library	4
Digit-Recognition	5
Backtracking (Solve Sudoku)	5
Debug-Ausgaben	6
Ausgabe des gelösten Sudokus	8
Optimierung mit Cython	9
Umsetzung / Entscheidungen	10
Testen	10
Wartung / Code-Pflege	11

Projekt

Definition

Es wurde ein Bild-basierter Sudoku-Solver entwickelt. Dabei wird die Web-Cam des Notebooks verwendet, um Sudokus einzulesen. Durch bekannte Verfahren der Bildverarbeitung werden die Sudoku-Felder erkannt und extrahiert. Die vorausgefüllten Felder werden mit Hilfe eines trainierten Netzes analysiert und in einer Datenstruktur gespeichert. Die Datenstruktur repräsentiert das Sudoku-Feld und wird durch einen Backtracking-Algorithmus gelöst. Die Benutzeroberfläche zeigt den Original-Input der Kamera sowie die mit der Lösung überlagerte Variante.

Anforderungen

Nachstehend werden die Anforderungen des Sudoku-Solver dargelegt und beschrieben. Es wird dabei zwischen Muss- und Kann-Anforderungen unterschieden. Es wurden sowohl alle Muss- als auch Kann-Anforderungen umgesetzt. Die Feature-Requests nach erfolgreichen Live-Demo wurden aus zeitlichen Gründen nicht mehr umgesetzt.

Muss-Anforderungen

- Erkennen eines Sudoku-Feldes auf einem Bild
- Transformieren des Sudoku-Feldes (90° - Winkel)
- Nachzeichnen der erkannten Kanten im Originalbild
- Ziffern extrahieren
- Trainieren eines Netzes
- Lösung mit Originalbild überlagern

Kann-Anforderungen

- Benutzeroberfläche mit Kamera-Livestream
- Ausgabe des gelösten Sudokus (überlagertes Originalbild)
- Optimierte Bildbearbeitung (Berechnungszeit reduzieren)
- Live-Darstellung

Feature-Requests (nach live-demo ;)

- Wenn Sudoku nicht lösbar, dann soll ein roter Rahmen angezeigt werden
- Ein spezifisches Feld auswählen, welches gelöst werden soll

Entwicklungsteam

Julian Jany (S1810307081@students.fh-hagenberg.at)

Michael Neuhold (S1810307094@students.fh-hagenberg.at)

Gerald Mitterecker (S1810307091@students.fh-hagenberg.at)

Installations- und Inbetriebnahmeanleitung

Projektverzeichnis

1. Variante (GitHub):

Der gesamte Source-Code kann dem GitHub-Repository entnommen werden. Das Team sowie die Übungsleiter wurden dem Repository als Mitglieder hinzugefügt. Der Link zum Repository: <https://github.com/michael-neuhold/sudoku-solver.git>

Bei diesem Projekt wurde ein trainiertes Modell verwendet. Dieses ist jedoch **nicht** im GitHub-Repository enthalten (File zu groß). Bei Bedarf wird das Modell aber gerne übermittelt. Diesbezüglich bitte per Mail beim Entwicklungsteam melden:

- Julian Jany (S1810307081@students.fh-hagenberg.at)
- Gerald Mitterecker (S1810307091@students.fh-hagenberg.at)
- Michael Neuhold (S1810307094@students.fh-hagenberg.at)

Nach Übermittlung muss das Modell im Verzeichnis „digit_recognition“ abgelegt werden.

2. Variante 2 (zip-Datei)

Der gesamte Source-Code, inklusive dem trainierten Modell, kann auf Anfrage übermittelt werden. Anfrage bitte per Mail an das Entwicklungsteam.

Abhängigkeiten installieren

- NumPy → pip install numpy
- TensorFlow (requires python version 3.5–3.8) -> pip install tensorflow
- Keras → pip install keras
- QT → pip install PyQt5
- OpenCV → pip install opencv-python

Ausführung

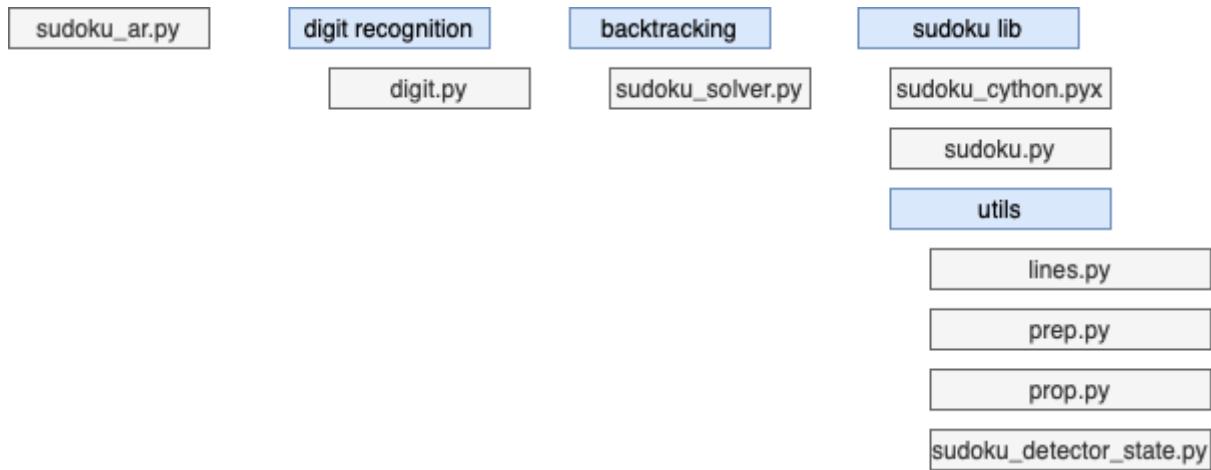
Bevor das Programm gestartet werden kann muss das File „sudoku_cython“ im Verzeichnis „sudoku_lib“ kompiliert werden. In diesem File befindet sich eine ausgelagerte Funktion um die Performance zu verbessern. Der Befehl für das Kompilieren ist:

„cythonize -i package.pyx“

Im Anschluss kann das File „sudoku_ar.py“ ausgeführt werden. → python sudoku_ar.py (Benutzeroberfläche wird geöffnet, Kamerastream wird für die Bildeingabe verwendet)

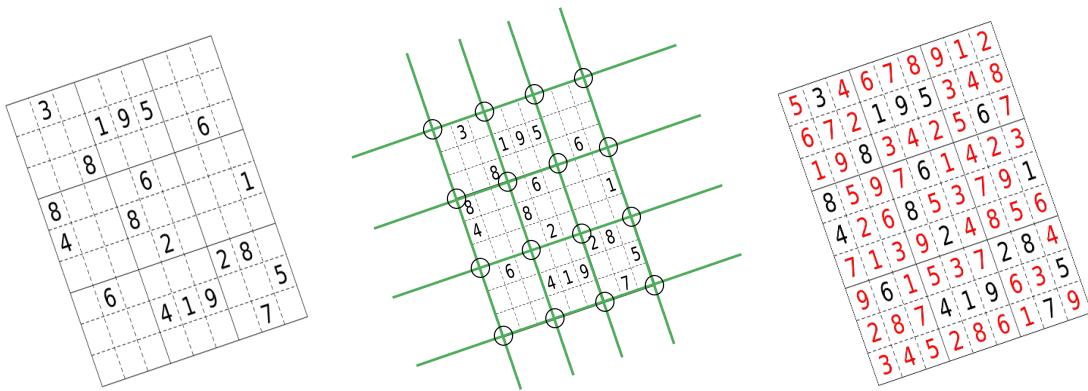
WICHTIG: Das Programm verwendet die im Betriebssystem als Standard eingetragene Kamera. Diese darf nicht von anderen Programmen zeitgleich verwendet werden da das Programm sonst nicht gestartet werden kann.

Projekt-Architektur



Main

Das File “`sudoku_ar.py`” repräsentiert den Startpunkt des Programms. Es wird ein UI-Fenster durch PyQt5 geöffnet und ein Stream der installierten Kamera angezeigt (Die Kamera kann nur von einem Programm verwendet werden!). Mit Hilfe der Funktionen in den anderen Verzeichnissen wird das Sudoku erkannt, entzerrt, gelöst, rücktransformiert und im Originalbild angezeigt.



Sudoku-Library

Der Ordner “`sudoku_lib`” beinhaltet die Funktionalitäten für die Verarbeitung der Bilder. Die Vorbereitung für das Erkennen der Ziffern steht dabei im Mittelpunkt. Im Unterverzeichnis “`util`” befinden sich 3 Files, die Hilfsfunktionen für Linienerkennung, Bildfilterung, sowie Funktionen für das Extrahieren und Entzerren von Bildteilen enthält.

Für weitere Entwicklungsschritte ist eine “debug” Funktionalität verfügbar. Diese kann durch ein Flag aktiviert werden. Bei Aktivierung werden zusätzliche Ausgaben produziert und Bildbearbeitung-Zwischenschritte gespeichert.

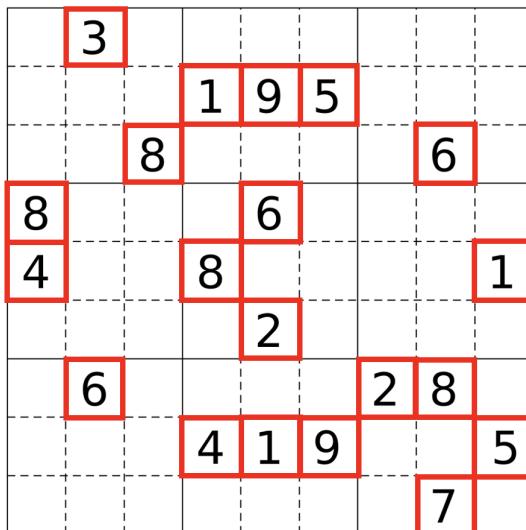
Digit-Recognition

Das Verzeichnis “digit_recognition” enthält ein Python-File “digit.py” welches das trainierte Modell lädt und eine Funktion für die Ziffern-Erkennung bereitstellt. Dieser Funktion kann eine List mit vorbereiteten Ziffern-Bildern übergeben werden (siehe nachstehende Bilder). Die gleichzeitige Übergabe durch eine Liste führt zu einer Performancesteigerung.



Die Funktion returniert eine Liste mit den durch das Netz erkannten Ziffern. Das Netzwerk wurde mit eigenen Bildern trainiert. Dafür wurden gelabelte Testbilder entsprechend vorbereitet.

Für eine bessere Performance werden jene Feldelemente vorausgewählt, in denen eine Ziffer eingetragen ist. Dies wird durch die Standardabweichung in Bezug auf den Schwarz bzw. Weißwert berechnet.

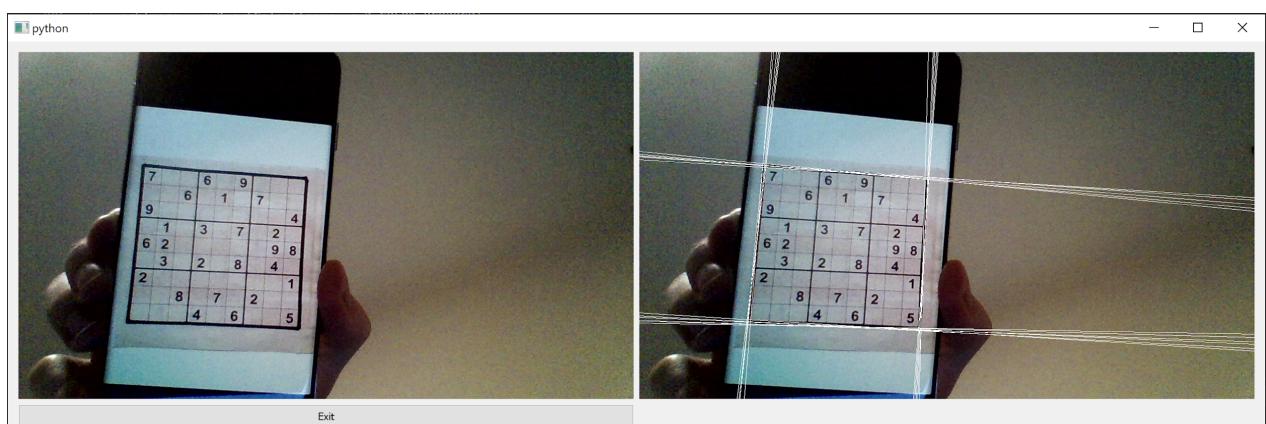
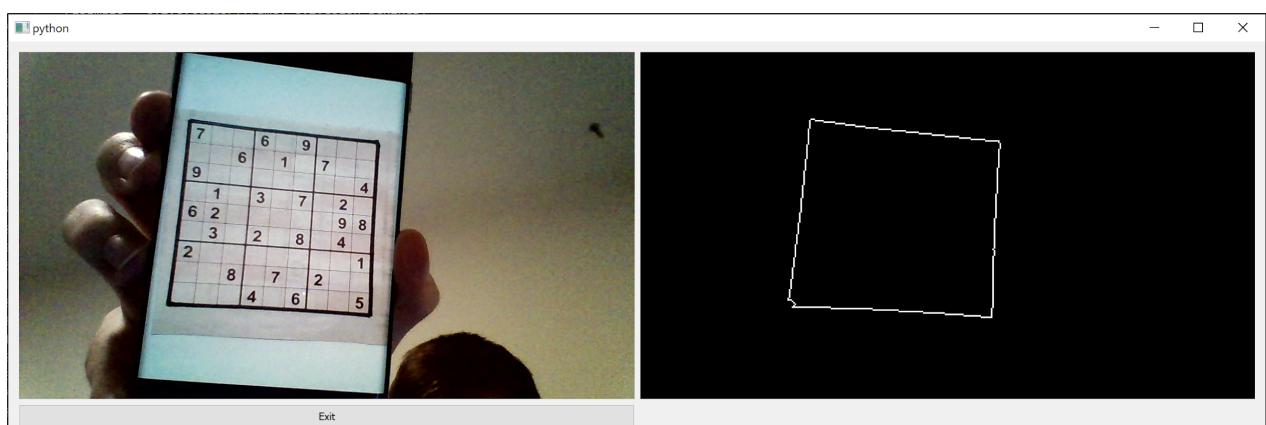
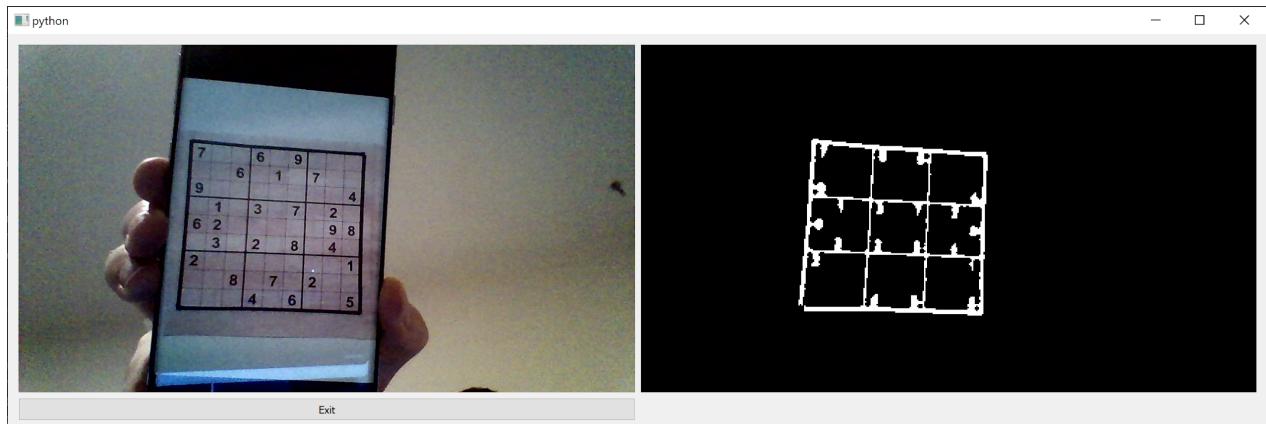


Backtracking (Solve Sudoku)

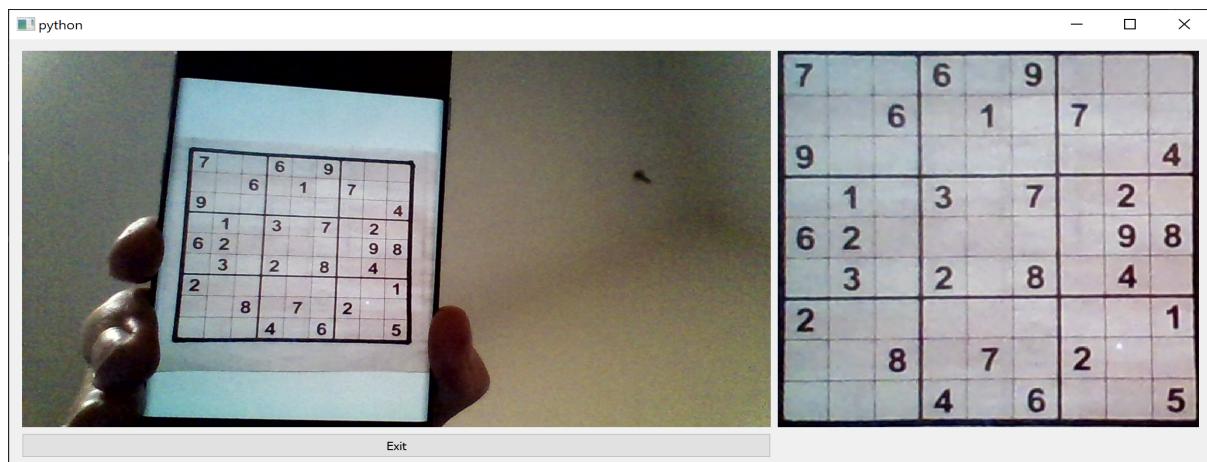
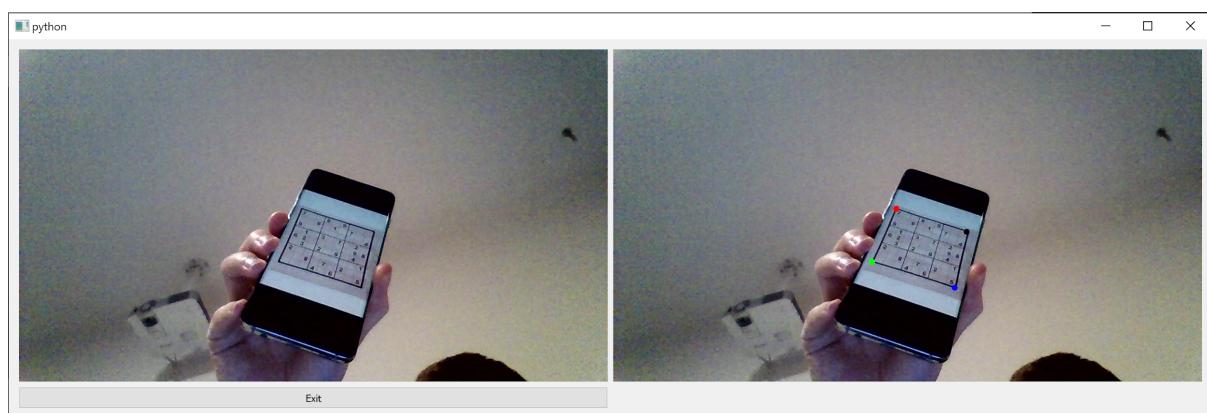
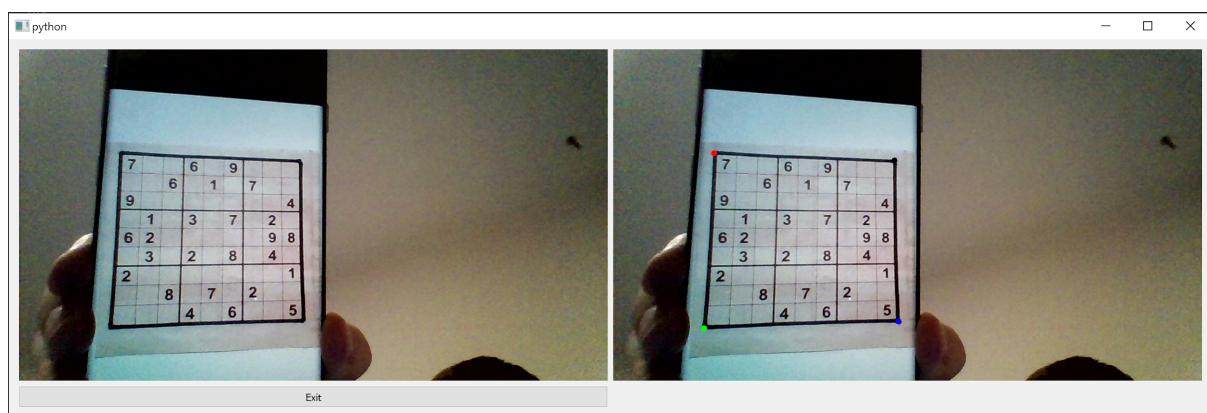
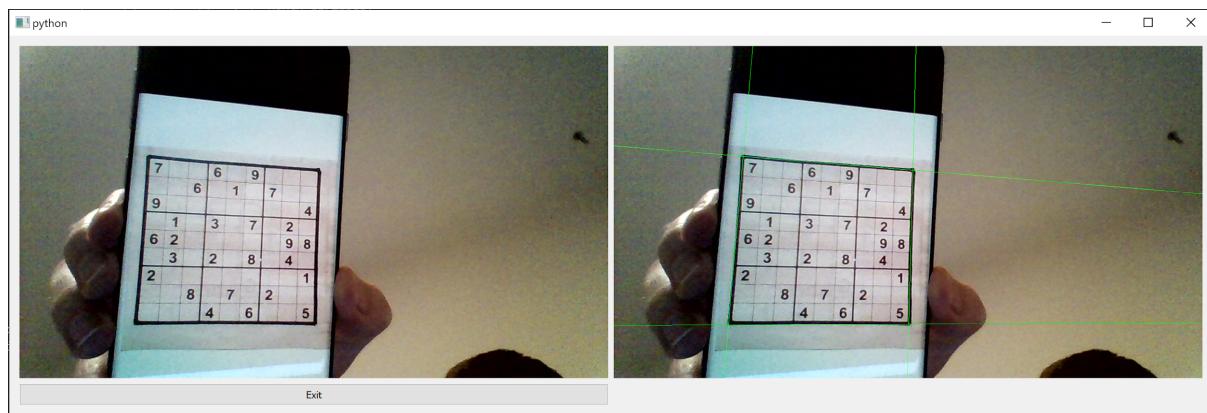
Das Verzeichnis “backtracking” enthält ein File “sudoku_solver.py”, mit dem Sudokus mit Hilfe eines Backtracking-Algorithmus gelöst werden können. Der Funktion “solve_sudoku” kann eine 2-Dimensionale Liste (welche das Sudoku-Feld repräsentiert) übergeben werden. Nicht ausgefüllte Felder werden durch den Wert 0 in der Datenstruktur repräsentiert. Das Sudoku wird rekursiv gelöst. Ist eine valide Lösung in einem rekursiven Abstieg nicht mehr möglich, so wird der Schritt zurückgesetzt und die Lösung mit der nächsten Ziffer getestet. Bei Lösung eines Konnte das Sudoku gelöst werden, so wird die befüllte 2-Dimensionale Liste und der Wahrheitswert True zurückgegeben. Ist das Lösen des Sudokus nicht möglich so wird False returniert.

Debug-Ausgaben

Die Implementierung stellt Debug-Ausgaben zur Verfügung, mit denen die einzelnen Bearbeitungsschritte ausgegeben werden können. Diese Ausgaben können durch das Setzen des entsprechenden Wahrheitswertes im Quelltext aktiviert werden. Die nachstehenden Screenshots zeigen die erkannten Kanten, das erkannte Sudoku-Feld, die Zusammenfassung der Linien, sowie die äußenen Schnittpunkte. Durch die äußenen Schnittpunkte kann das Sudoku-Feld mit einer OpenCV Funktion entzerrt werden.

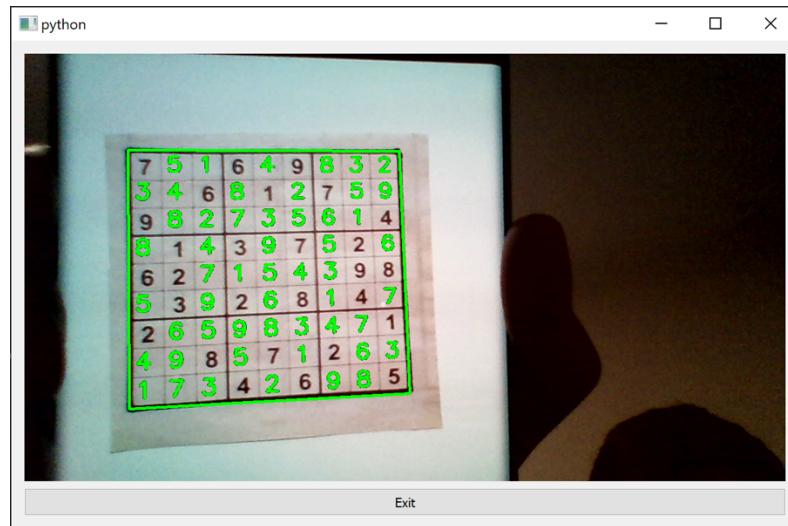


Sudoku-Solver

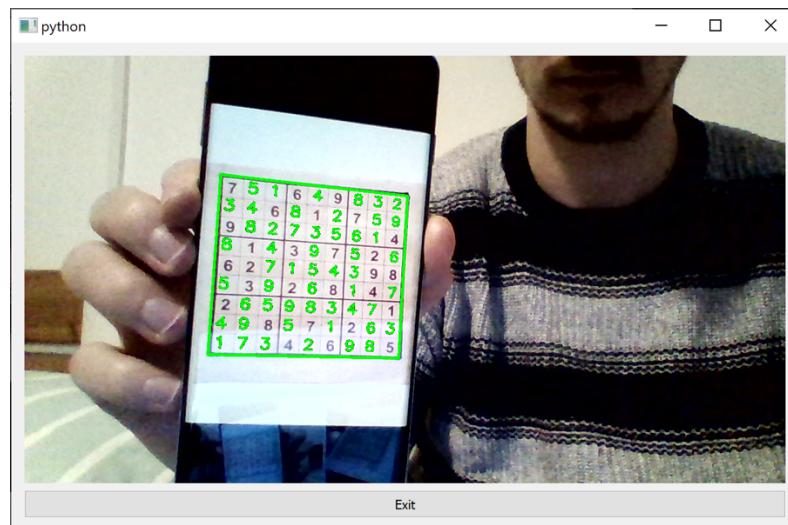


Ausgabe des gelösten Sudokus

Die nachstehenden Screenshots zeigen das End-Produkt, bei dem ein UI-Fenster geöffnet wird, indem ein „live“-Stream der Kamera angezeigt wird. Wird ein Sudoku-Feld erkannt, so wird dieses angezeichnet und durch das Erkennen der Ziffern gelöst. Ist eine Lösung vorhanden, so wird diese in den gleichen Stream geschrieben.

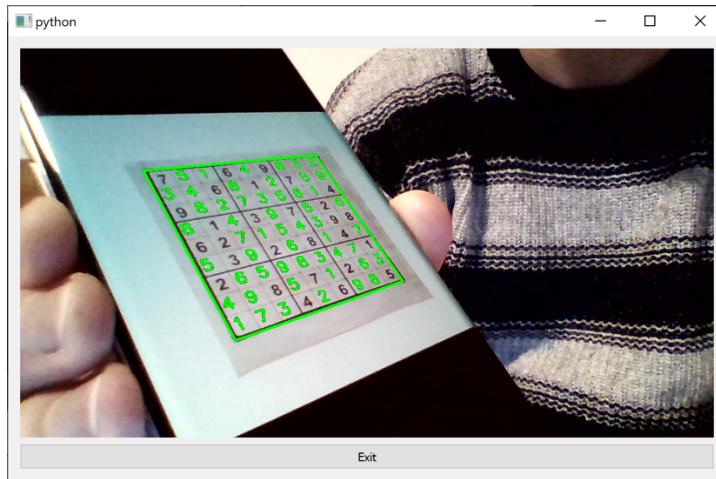


Der Algorithmus zur Erkennung der Sudoku-Felder ist für unterschiedliche Entfernungen und Ausrichtungen optimiert.



Sudoku-Solver

Der nachstehende Screenshot zeigt eine Demo mit stark verzerrtem Sudoku-Feld, welches erkannt und gelöst werden kann.



Optimierung mit Cython

Durch „Cython“ können C – Funktion für Python-Projekte geschrieben werden. Die Funktionalität wird in einer eigenen Syntax geschrieben und im Anschluss zu C/C++ umgewandelt. Für die Ausführung wird der Code als Python-Extension-Module kompiliert. In der folgenden Funktion muss über das gesamte Bild iteriert werden, daher ist eine derartige Optimierung erforderlich.

```
Sudoku_lib/sudoku_cython.pyx
import cython

@cython.boundscheck(False)
cpdef unsigned char[:, :] calc_component_bound(unsigned char[:, :], unsigned char[:, :])
bound_img:
    cdef int y, x, width, height
    height = comp_img.shape[0]
    width = comp_img.shape[1]
    # vertical lines
    for y in range(height):
        x = 0
        while (x < width and comp_img[y][x] == 0):
            x += 1
        if x < width:
            bound_img[y][x] = 255
            x = width-1
            while (x > 0 and comp_img[y][x] == 0):
                x -= 1
            if x > 0:
                bound_img[y][x] = 255

    # horizontal lines
    for x in range(width):
        y = 0
        while (y < height and comp_img[y][x] == 0):
            y += 1
        if y < height:
            bound_img[y][x] = 255
            y = height-1
            while (y > 0 and comp_img[y][x] == 0):
                y -= 1
            if y > 0:
                bound_img[y][x] = 255
return bound_img
```

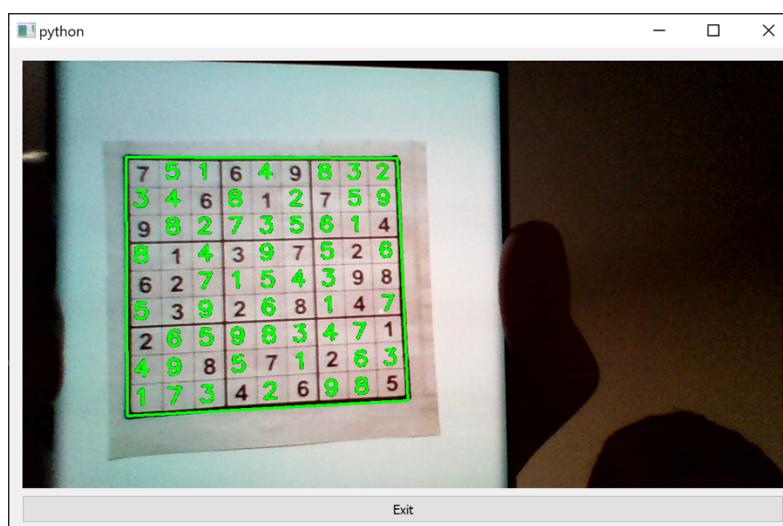
Umsetzung / Entscheidungen

Wie bereits zuvor beschrieben, wurden **alle Anforderungen**, die bei der Projektdefinition besprochen wurden, **umgesetzt**.

- Für die Umsetzung der Bildverarbeitungsoperationen wurde die Python Bibliothek *OpenCV* verwendet. Die Entscheidung ist auf den umfangreichen Funktionalitätsumfang sowie die Integration mit *NumPy* zurückzuführen.
- Für die Erkennungen der Ziffern wurde die Bibliothek *Keras* verwendet. Es wurde ein eigener Trainingsdatensatz erstellt, um für die Sudokus eine optimale Erkennung zu garantieren. Es wurde ungefähr mit 1600 Bildern trainiert.
- Für die Darstellung in einer Benutzerdarstellung wurde *pyQt5* verwendet. Durch diese Bibliothek konnte eine einfache UI für die Darstellung der Kamera-Streams erstellt werden.
- Die Feature-Requests (nach live-Demo 10.02.2021) wurde aus zeitlichen Gründen nicht mehr umgesetzt.

Testen

Die entwickelten Funktionen wurden durch die Debug-Ausgaben (zuvor gezeigt) getestet. Sudoku-Eingaben funktionieren sowohl auf Papier als auch auf Displays (zB Handy). Für das Testen wurden zusätzlich etliche Zwischenschritte in Form von Bildern im Filesystem gespeichert.



Wartung / Code-Pflege

Das GitHub-Repository wird nach der Beurteilung „public“ geschalten, damit es auch von anderen Interessierten weiterentwickelt werden kann. Für eine leichte Einarbeitung sind für den Großteil der Funktionen eine In-Code Dokumentation (Doc-Strings) vorhanden. Der nachstehende Quellcode-Ausschnitt zeigt ein Beispiel der In-Code-Dokumentation.

```
def find_digits(sudoku_img) -> Tuple:
    """
    Detects which cells of Sudoku image contain a number and which are empty

    Parameter:
    sudoku_img: [][]numpy_array Image of Sudoku

    Returns:
    ([[]numpy_array, (int, int), (int, int))]
    First element: Image of the digit
    Second elemnt: Position of digits
    Third element: Position of empty cells
    """

    digits = []
    digit_pos = []
    empty_pos = []

    for y in range(9):
        for x in range(9):
            # determine if tile contains digit
            stddev = (
                post_process.calc_stddev(
                    post_process.extract_detector_region(sudoku_img, (x, y)))

            # stddev:
            # - empty: [35, 145]
            # - digit: [1458, 2103]

            if stddev >= 800: # contains digit
                cnn_input = (
                    post_process.post_process_cnn_input(
                        post_process.extract_cnn_input(sudoku_img, (x, y))))
                digits.append( cnn_input )
                digit_pos.append( (x, y) )
            else: # empty
                empty_pos.append( (x, y) )

    return digits, digit_pos, empty_pos
```