\section{Active Buzzer}

Bei der Umsetzung des Akustischen Signal Tons der Anlage wurde provisorisch auf einen sogenannten aktiven Summer zurückgegriffen. Dieser erklingt zum Beispiel, wenn der Richtige RFID Chip an das Lesegerät gehalten wird und somit die Garage geöffnet oder geschlossen wird. Dabei handelt es sich um ein Oszillator der, wird er einer gewissen Spannung untersetzt, anfängt zu Vibrieren und damit ein Ton erzeugt. Ausgeführt und koordiniert wird der Buzzer über ein Simples Python Skript das mittels der Bibliothek GPIO (General-Purpose Input/Output) in der Lage ist den Buzzer an oder auszuschalten. Dabei wird durch einen definierten Port, in diesem Fall „Port 4“, mittels einer $for$ Schleife die Spannung je nach $positivem$ oder $negativem$ Signal Ton koordiniert und eine andere Tonsequenz abgespielt.

\section{Geometrische Lösung}

Nach mehreren gescheiterten Anläufen der versuchten Implementierung des Darknet Finetuning galt es einen alternativen Ansatz zu finden und umzusetzen. Durch den Mangel an Modellen hinsichtlich der im Paragrafen (Datenbeschaffung) genannten Gründe, des Datenschutzes und der Wirtschaftlichkeit, musste auf eine Methodik fernab des Deep-Leanings zurückgegriffen werden.

Erster Ansätze bestanden dabei aus der dauerhaften Erkennung, ob sich ein Schriftzug innerhalb des Kamerasichtfelds befindet. Die Umsetzung erfolgte dabei mittels des Python Package EasyOCR, das in einem späteren Kapitel näher erläutert wird. Die Erprobung des Ansatzes stellte sich jedoch als ineffizient heraus, da trotz reduzierter Bild Streaming Rate, eine konstante hohe Beanspruchung der begrenzten Rechenkapazitäten von Nöten ist und somit Parallelprozesse einschränkt werden. Zur Umsetzung der Effizienteren Rechenkapazitäts Nutzung bestände der Ansatz außerhalb der Garage einen Sensor zu installieren, der ein herangefahrenes Auto erkennt und somit den Texterkennungsprozess auslöst. Die Lösung würde jedoch ein extraaufwand zur Installation des Sensors mit sich bringen und das Vorhandensein eines weiteren Ultraschallsensors voraussetzen.

Eine alternative, um Rechenleistung einzusparen, wäre die Lokalisierung der auszulesenden Nummernschilder innerhalb des Kamera Blickwinkels. Durch die Angabe, ob sich ein Nummernschild innerhalb des Blickwinkels befindet und wo es sich befindet ist man in der Lage, nicht mehr auf die Überprüfung des Gesamten Bildes angewiesen zu sein und den Prozess dazu nur auszulösen wenn ein Nummernschild erkannt wird. Bei der Umsetzung der Methode wurde in der Praxis auf ein nicht Deep-Learning basierten Ansatz zurückgegriffen. Anders als bei der Verwendung eines trainierten Neuronalen Netzes, was auf die Erkennung von an Autos befestigten Nummernschildern zielt, gilt es bei der verwendeten Methode jedes Rechteck innerhalb einer des Bildes zu detektieren. Potenziell gilt es dabei die Eigenschaft auszunützen, dass die Vehikel mittels Frontaler sich ausschließlich der Konturen des Nummernschildes ein Rechteckt aufweisen.

Kernkonzept der Detektion eines Rechtecks innerhalb des zu Verarbeitenden Bildes ist der Ramer–Douglas–Peucker Algorithmus, dieser wird kurz erläutert. Ursprüngliches Ziel des Algorithmus ist die Kurvenglättung. Dabei hat man eine Kette an Knoten und Vektoren der Länge n. Der Algorithmus schreitet und durchläuft einen Graphen dabei Schrittweise. Als Approximation der Strecke P1 und Pn wird deren Vektor genommen und überprüft welcher Knoten des Graphen sich am weitesten weit weg von dem Vektor P1Pn befindet, liegt dieser außerhalb eines zuvor definierten Toleranzwertes $\epsilon$. Der dabei neu ausgemachte Punkt (Px) dient als neuer Punkt innerhalb des Graphen, alle anderen Punkte, die sich zwischen P1 und Px befinden werden, weg approximiert. Der Prozess wiederholt sich immer wieder bis keine Punkt mehr innerhalb der einzelnen Schritte außerhalb des Toleranzwertes $\epsilon$.\autocite{hershberger1992speeding} Somit ist man in der Lage Ausreißer durch Kurvenglättung auszuschließen und mit der Voraussetzung, dass man nach einem geschlossenen Graphen, der aus vier Vektoren besteht, alle Rechtecke auszumachen die sich innerhalb eines Bildes Befinden auszumachen. Die vier Knotenpunkte des Synthetisierten Rechtecks dienen dabei als Koordinate für die Bildextraktion des Nummernschilds.

Die Pipeline zur Nummernschilderknunng besteht aus einem Preprocessing Part und der Detektion die mittels des Ramer–Douglas–Peucker Algorithmus als Kernmodul der Erkennung fungiert. Umgesetzt wird der Prozess mittels der Python Bibliothek $opencv$. Zuerst werden die von der Kamera gestreamten Bilder durch mehrere Filter der Bildverarbeitung durchlaufen. Dabei wird das Bild in Graustufen umgewandelt der Noise reduzieren und des Weiteren mit gegebenen Kernels Kanten detektiert. Die Kanten werden dabei in Vektoren erfasst. Durch die Beschaffenheit der Vektoren kann mittels des Peucker Algorithmus jene Vektoren gefunden werden, die in der Summer 0 ergeben und somit einen Geschlossenen Graphen bilden. Durch die Detektion jener Vektoren die approximiert eine Summe von vier ergeben ist man in der Lage innerhalb des Bildes Rechtecke zu finden. Bei gegebener Detektion eines Rechtecks wird diese extrahiert und erst dann mittels der Texterkennungsprozesse ausgelesen, jene werden im nächsten Kapitel erklärt.

\section{OCR}

OCR (Optical Character Recognition) ist das automatische Erkennen von Text innerhalb eines Bildes. Diese Technologie ermöglicht es, den Text in einem Bild oder einem gescannten Dokument in für Menschen verständlichen Text umzuwandeln. Für dies artige Anwendungen gibt es eine Vielzahl von Anwendungsfällen in der heutigen Zeit. Dazu gehört beispielsweise die Umwandlung Informationen, die ausschließlich in einem Print-Medium zu finden sind, in ein digitales Medium, die einfache Erkennung eines Coupons oder auch das Auslesen eines schriftlich ausgefüllten Formulars mit Hilfe des Computers.\autocite{ocr1}[Vgl.][S. 81]

Für die Optische Texterkennung gibt es mittlerweile eine Vielzahl von angebotenen Algorithmen und Services. Darunter beispielsweise Textract, welches von Amazon bereitgestellt und verkauft wird. Allerdings gibt es auch Open-Source Algorithmen dafür wie EasyOCR und Tesseract. Nach der Erkennung und Extraktion zur Weiterverarbeitung eines potenziellen Nummernschildes, wie im vorherigen Kapitel erläutert wurde, gilt es dieses mittels weiterer Modelle weiter zu verarbeiten und auszulesen. Der erste Ansatz des Projekts galt der Pythonbibliothek EasyOCR. Da jene den Momentanen Sate-of-the Art bereitstellt, Ressourcen schonen ist und eine simple Implementierung innerhalb der Pipeline ermöglicht. Nach der Erkennung des Nummernschildes wird das dabei extrahierte Bild im nächsten Pipeline Schritt durch das Package ausgelesen und der Text in Schriftform generiert. Bezüglich der Funktionalität und der Hintergrundprozesse wird nun in Folge eingegangen.

Das Modell besteht aus zwei Schritten, zuerst die Detektion der einzelnen Buchstaben, danach deren Erkennung. Für erstere Aufgabe wird sich des CRAFT (Character Region Awareness for Text Detection) Algorithmus bedient. \autocite{baek2019character} Durch die Ausmachung der einzelnen Charaktere weist das Modell besonders bei verzerrten oder nicht linearen Texten bessere Ergebnisse, im Gegensatz zu Detektion von Wort Blöcken, auf. Das Modell basiert dabei auf einem Convolutional Neuronal Network mit skip-connections und Charakteristiken eins U-Netz. Diese wird mit Texten jeglicher Form und Druckschrift trainiert. Nach erfolgreichem Training ist es in der Lage für die Einzelne Buchstaben ein "region score" und ein "Affinitäts score" auszugeben. Ersterer beziehet sich dabei, wie der Name schon ausdrückt, auf die Koordinaten, die die Buchstabe lokalisieren, zweiteres auf die Wahrscheinlichkeit des Zusammenhangs einzelner Charaktere in einem Wort. Durch das Anpassen des Modells auf auch nicht lineare Texte gilt es als vielversprechenden bei der Umsetzung des $SmartGarage$ Projekts da bei dessen Umsetzung die Kamera unwahrscheinlich Frontal und auf das Nummernschild zeigen wird. Wahrscheinlicher ist ein Winkel und damit eine Verzehrte Schrift die das OCR Modell zu meistern hat.

Im Folgenden Bild sind die einzelnen Zusammenhänge der Detektion eines Wortes dargestellt:

(hier bild von whatsapp baek2019character)

Nach dessen Lokalisierung und der daraus gewonnen Koordinaten, werden die gewonnen Informationen in EasyOCR weiterverarbeitet und mittels eines CRNN (Convolutional Recurrent Neural Network) Netzes bestehend aus drei Komponenten ausgelesen. Es handelt sich dabei um eine Feature Extraktion, des Gesuchten Inputs mittels eines RestNets. Eines LSTM (Long-short term memory) Netzes für das sequenzielle Labeling von Charakteren innerhalb eines Wortes. LSTM Netzte eignen sich dabei besonders für die Verarbeitung sequenzieller Daten. Die dritte Komponente besteht aus einem CTC (connection temporal classification) Netz und ist für das Decoding der Outputs innerhalb des LSTM Netzes Zuständig ist \autocite{JaidedAI70}.

Die bei der Praktische n Umsetzung erlangten Resultate sind in dem Folgenden Unterkapitel aufgeführt. Manko bei der jeweiligen Implementierung ist jedoch die Notwendigkeit eines 64Bit Systems für die Implementierung, da die Bibliothek größtenteils auf Pytoch für die Implementierung der Netze zurückgreift. Die Ursprüngliche Hardwarekomponente mittels dessen das Projekt jedoch umgesetzt wird beläuft sich auf ein Rasbery Pi 3. Da für jenen bei der Umsetzung des Projektes (16.12.21-10.02.22) lediglich ein 32Bit Systeme zur Verfügung stand galt es Umsetzung mittels EasyOCR zu verwerfen. Mittlerweile steht jedoch ein 64Bit Systems zur Verfügung, dieses könnten in einer weiterführenden Arbeit die Nutzung des Packages bei Bedürfnis mit der selbigen Hardware erlauben.

Alternativ zu der Erkennung der Schriftzeichen mittels EasyOCR wurde auf die Bibliothek Tesseract 5zurückgegriffen. Diese bedient sich auch eines LSTM Netztes wie die vorherig gesehene Bibliothek, benötigt jedoch keines 64Bit Systems. Tesseract ist dabei ein open source Projekt, dass seit dem Jahr 2005 von Google verwaltet wird und seither immer weiterentwickelt wird. Bei der Umsetzung wird auf $Pytesseract$ und ein vortrainiertes englisches Modell der Texterkennung zurückgegriffen. \autocite{Tesserac98}

Bei der Umsetzung wird auch wieder auf ein Preprocessing gesetzt, dass das Bild in Graustufen konvertiert. Des Weiteren wird versucht mittels des Gauß-Verfahrens und gegebenen Filtern, Noise herauszufiltern. Dritter Schritt des Preprocessings ist das hervorhebenden der Kanten mittels der Konvertierung des Bildes in sogenannte $Blobs$, dabei handelt es sich um die Umwandlung eines Graubildes mittels eines festgelegten Schwellenwerts in Binäre Werte. \autocite{8974469}

Das Modell versucht mittels Vektoren Linien auszumachen, auf denen sich die Schriftzüge befinden. Danach geht Tesseract dazu über Zeilenweise Wörter mittels der Bemessung von Abständen auszumachen und Wörter in Bounding-Boxes zu fassen. Die Erkennung der Texte geschieht über das vortrainierte englische LSTM Netzt. Dieses weist bei der Umsetzung jedoch deutlich schlechtere Resultate auf als das zuvor verwendete CRNN (EasyOCR). Die Funktionalität und Genauigkeit beider Modelle wird in einem separaten Kapitel ausgewertet. Da beide Modelle jedoch funktionieren ist es nicht von Nöten das schlechtere zu verwerfen, durch mehrere Versuche das Nummernschild zu lesen gelingt es beiden Modellen das richtige zu lesen. Lediglich die Halbwertszeit ist invers proportional zur Güte der Modelle. Somit ist man auch in der Lage mittels pytesseract die Nummernschild Erkennung durchzuführen und die Resultate mit einer White-list abzugleichen. Bei einer Übereinstimmung öffnet sich somit das Tor.

@inproceedings{baek2019character,

title={Character region awareness for text detection},

author={Baek, Youngmin and Lee, Bado and Han, Dongyoon and Yun, Sangdoo and Lee, Hwalsuk},

booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition},

pages={9365--9374},

year={2019}

}

@misc{JaidedAI70,

author = {},

title = {JaidedAI/EasyOCR: Ready-to-use OCR with 80+ supported languages and all popular writing scripts including Latin, Chinese, Arabic, Devanagari, Cyrillic and etc.},

howpublished = {\url{https://github.com/JaidedAI/EasyOCR}},

month = {},

year = {},

note = {(Accessed on 02/19/2022)}

}

@INPROCEEDINGS{4376991,

author={Smith, R.},

booktitle={Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)},

title={An Overview of the Tesseract OCR Engine},

year={2007},

volume={2},

number={},

pages={629-633},

doi={10.1109/ICDAR.2007.4376991}}

@article{hershberger1992speeding,

title={Speeding up the Douglas-Peucker line-simplification algorithm},

author={Hershberger, John Edward and Snoeyink, Jack},

year={1992},

publisher={Citeseer}

}

@misc{Tesserac98,

author = {},

title = {Tesseract User Manual | tessdoc},

howpublished = {\url{https://tesseract-ocr.github.io/tessdoc/#Tesseract-with-lstm}},

month = {},

year = {},

note = {(Accessed on 01/29/2022)}

}

@INPROCEEDINGS{8974469,

author={Singh, Jaskirat and Bhushan, Bharat},

booktitle={2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)},

title={Real Time Indian License Plate Detection using Deep Neural Networks and Optical Character Recognition using LSTM Tesseract},

year={2019},

volume={},

number={},

pages={347-352},

doi={10.1109/ICCCIS48478.2019.8974469}}