



Hochschule für angewandte Wissenschaften Coburg
Fakultät Elektrotechnik und Informatik

Studiengang: Informatik

Projektdokumentation

AutoCount: Kamera-basierte Erfassung der Parkhausbelegung

Hannes Lüer und Marvin Jakob

Abgabe der Arbeit: 02.03.2024

Betreut durch: Prof. Dr. Matthias Mörz, Hochschule Coburg

Inhaltsverzeichnis

Abbildungsverzeichnis	3
Tabellenverzeichnis	4
1 Idee	5
2 Grundlagen	7
3 Technische Umsetzung	9
3.1 Sensor	9
3.1.1 Variante 0: einfache Objekterkennung	9
3.1.2 Variante 1: Richtung des Bewegungsvektors	10
3.1.3 Variante 2: Überschreiten einer Linie	12
3.2 Server	13
3.3 App	15
4 Anwendung und Test	17
4.1 Nutzung des Systems	17
4.2 Installation und Testaufzeichnung	17
4.3 Vergleich der beiden Zählverfahren	17
5 Fazit	20
Literaturverzeichnis	22
Ehrenwörtliche Erklärung	

Abbildungsverzeichnis

Abb. 1:	Raspberry Pi Camera V2.1	8
Abb. 2:	Schematische Darstellung der System-Architektur	9
Abb. 3:	Fahrzeugerkennung Variante 1 Beispiel	11
Abb. 4:	Fahrzeugerkennung Variante 2 Beispiel	12
Abb. 5:	Screenshots der App	15
Abb. 6:	Darstellung der Boxplots der benötigten Zeit pro Frame für die zwei entwickelten Verfahren	19

Tabellenverzeichnis

Tab. 1: Ergebnisse des Tests	18
--	----

1 Idee

Das Parkhaus an der Hochschule Coburg, welches direkt am Campus positioniert ist, umfasst 530 Parkplätze für Studierende und Mitarbeiter [Cob23]. Dabei sind die Parkmöglichkeiten für Studenten und Mitarbeiter durch eine Schranke getrennt. Die Haupteinfahrt ist jedoch für beide Parkflächen identisch. Es gibt außerdem einen Parkplatz in der Sonneberger Straße [Wis23a]. Dieser ist jedoch aufgrund der Lage für Autofahrer unattraktiver als das Parkhaus. Im vergangenen Wintersemester (2022/23) gab es 4890 immatrikulierte Studenten und 518 Mitarbeiter [Wis23b].

Wenn man als Student eine Vorlesung zu den Stoßzeiten besuchen möchte und auf das Auto angewiesen ist, stellt man häufig fest, dass die Auslastungs-Anzeige des Parkhauses „BESETZT“ darstellt. Das Problem dieser Anzeige ist, dass man sich nie sicher sein kann, was diese Anzeige genau darstellt. Ist gar kein Parkplatz mehr frei, sind nur noch wenige frei oder wird sogar eine Fehlinformation angezeigt? Häufig lassen sich trotz der Anzeige, dass das Parkhaus besetzt wäre, noch freie Parkplätze finden.

Im Rahmen des Moduls „Hardware cyber-physischer Systeme“ soll eine Möglichkeit gefunden werden, eine exakte Anzeige der Parkhausbelegung umzusetzen. Dafür soll — wie in größeren Parkhäusern üblich — dargestellt werden, wie viele Parkplätze noch frei sind. Hierfür sind verschiedene Ansätze möglich. Zum einen ist eine Umsetzung denkbar, bei welcher am Eingang des Parkhauses eine Lichtschranke installiert wird. Diese soll jeweils Autos zählen, welche sich über die Einfahrt in das Parkhaus hinein- bzw. durch die Ausfahrt hinausbewegen. Ein fundamentales Problem bei diesem Ansatz ist, dass das System potenziell jedes Objekt oder jede Person detektieren würde. Somit wäre die Anzeige der Parkhausbelegung fehlerhaft. Außerdem wäre eine Umsetzung denkbar, bei welcher auf jedem Parkplatz ein Näherungssensor angebracht wird, welcher erfasst, ob sich auf dem jeweiligen Parkplatz ein Objekt befindet oder nicht. Dieses Konzept würde vermutlich gut funktionieren, bringt jedoch einen hohen Kostenaufwand mit sich, weswegen diese Variante verworfen wird.

Als potenziell beste Lösung hat sich eine Kamera-basierte Umsetzung herausgestellt. Bei dieser Idee wird eine Kamera an der Einfahrt des Parkhauses platziert. Die Aufnahmen werden mit Objekt-Erkennung analysiert. Dabei werden lediglich Autos betrachtet, welche in das Parkhaus

1 Idee

hinein bzw. aus dem Parkhaus hinaus fahren. Durch die Objekterkennung kann ausgeschlossen werden, dass ungewünschte Objekte oder Personen vom System erfasst werden.

Die Parkplätze der Mitarbeiter müssen separat gezählt werden. Eine Möglichkeit wäre die Aktivität der Schranke mit einem Neigungssensor zu erfassen. Es kann jedoch potenziell dazu kommen, dass eine Schranke geöffnet wird und kein Auto durchfährt oder mehrere Autos gleichzeitig durch die geöffnete Schranke fahren. Deshalb wird für die Zählung der Auslastung der Mitarbeiterparkplätze dieselbe Variante verwendet, welche für die Erfassung der Studentenparkplätze vorgeschlagen wurde.

2 Grundlagen

In diesem Kapitel wird beschrieben, welche Hardware- und Software-Komponenten für die Umsetzung des Projektes nötig sind. Außerdem wird darauf eingegangen, welche Anforderungen diese erfüllen müssen.

Zunächst wird ein Rechner benötigt, welcher genug Rechenleistung aufweist, um die Objekterkennung der Autos umsetzen zu können. Es wäre eine Lösung denkbar, bei welcher die Videoaufzeichnungen in eine Cloud gestreamt werden und dort weiterverarbeitet werden. Da die Rechenleistung von Mini-Computern für solche Szenarien jedoch mittlerweile für eine Objekterkennung stark genug ist und eine direkte Verarbeitung des Videomaterials die Menge der zu übertragenden Datenmengen stark einschränkt, wird ein solches System bevorzugt. Dieses benötigt eine Möglichkeit einen Kamera-Sensor anbringen zu können und eine Kommunikationsmöglichkeit über Ethernet oder WLAN. Die Wahl fällt auf einen Raspberry Pi 3 B+, da dieser bereits vorrätig ist und somit kein zusätzlicher finanzieller Aufwand entsteht. Die Rechenleistung sollte mit einem 64 Bit Quad Core 1,4 GHz-Prozessor und 1 GB Arbeitsspeicher für das Projekt ausreichend sein [Ltd23]. Um Autos zu erkennen, wird weder eine hohe Auflösung, noch eine hohe Bildwiederholfrequenz benötigt. Es gibt viele Alternativen, welche für die Umsetzung des Projektes denkbar wären. Falls die Leistung des gewählten Mini-Computers nicht ausreicht, sollten Alternativen, wie Vertreter der NVIDIA Jetson- oder der ASUS Tinker Board-Reihe für das Projekt in Betracht gezogen werden [NVI23] [INC23].

Als Kamera-Sensor wird die, in Abbildung 1 dargestellte, Raspberry Pi Camera in der Version 2.1 verwendet. Diese wurde speziell für den Raspberry Pi entwickelt und liefert mit einem 8 Megapixel-Sensor eine ausreichende Bildqualität. Videos können mit bis zu 1080p aufgezeichnet werden [Ber23]. Der Sensor kann direkt mit dem Mini-Computer, über den dafür vorgesehenen seriellen Schnittstellenanschluss, verbunden werden. Aufgrund der einfachen Kommunikation zwischen Kamera und Raspberry Pi, sowie einer mehr als ausreichenden Bildqualität wurden als Alternativen lediglich andere Versionen der Pi Camera betrachtet. Wegen der Verfügbarkeit und den nicht benötigten Vorteilen der anderen Modelle, wie weitere Blickfelder oder bessere Auflösungen, wurde in dieser Arbeit Version 2.1 der Raspberry Pi Camera verwendet.

Die Anzeige der Belegung des Parkhauses soll für Studenten und Mitarbeiter der Hochschule Coburg über ein beliebiges Endgerät aus erreichbar sein. Aus diesem Grund ist es sinnvoll eine



Abb. 1: Raspbarry Pi Camera V2.1 (Quelle: [Ras18])

Multi-Plattform Applikation zu erstellen. Diese sollte auf den gängigsten Betriebssystemen, wie Android, iOS, Windows oder im Web erreichbar sein. Diese Anwendungen werden häufig durch Frameworks, wie Flutter, Xamarin (mittlerweile .NET MAUI) oder React Native, realisiert. Die Wahl des verwendeten Frameworks fällt auf Flutter, da dies das aktuell modernste und verbreitetste Framework zur Erstellung nativer Multi-Plattform Anwendungen ist und außerdem bereits Erfahrungen mit dessen Programmierung gemacht wurden. Die Realisierung mittels einer reinen Web-Anwendung wäre ebenfalls möglich gewesen. Flutter bietet jedoch die Erstellung von sowohl Web-Anwendungen als auch Applikationen für verschiedene Betriebssysteme.

Als Bindeglied zwischen Raspberry Pi und Applikation für den Nutzer muss ein Server für das Handling der Daten implementiert werden. Die Schnittstelle zum Server kann über HTTP-Anfragen realisiert werden. Dieses Konzept ist weit verbreitet und Go, was zur Implementierung des Servers eingesetzt wurde, bietet hierfür hocheffiziente Bibliotheken. Für die Schnittstelle zur Applikation ist es sinnvoll, eine Methode zu verwenden, welche es dem Server erlaubt, Daten zu schicken, ohne eine Anfrage zu erhalten. So kann sichergestellt werden, dass die Anzeige der Anwendung stets mit dem Auslastungs-Wert des Servers übereinstimmt und hierfür kein Polling notwendig ist. Ein dafür häufig verwendetes Protokoll, das auch in dieser Arbeit verwendet wird, ist MQTT.

3 Technische Umsetzung

Das im Rahmen dieser Arbeit entwickelte System besteht aus drei Teil-Systemen: dem Sensor, dem Server und der App. Dabei wird unter dem Sensor System der Teil angesehen, welcher eine klassische Detektion ersetzen kann. Der Server stellt das Bindeglied der Kommunikation zwischen Sensor und App dar. Für den Nutzer werden die erfassten Daten in der App dargestellt, die vom Server gesendet werden. Die gesamte Architektur ist in Abbildung 2 dargestellt.

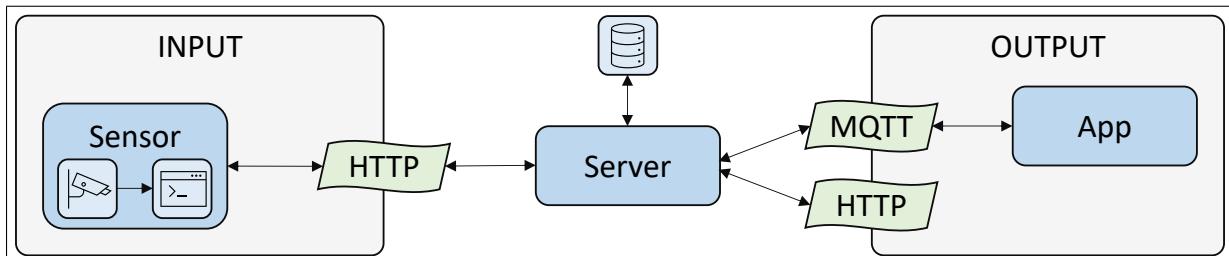


Abb. 2: Schematische Darstellung der System-Architektur mit ihren einzelnen Systemen (blau), den an diesen angebundenen Komponenten (hellblau) und den zur Kommunikation verwendeten Protokollen (grün). (Quelle: eigene Darstellung)

Im Folgenden wird jeweils auf die Umsetzung der einzelnen Teil-Systeme eingegangen.

3.1 Sensor

Die Komponente, welche erkennt, wie viele Autos den Parkplatz befahren und verlassen, wird in dieser Arbeit Sensor genannt, besteht aus einem Mini-Computer und einer Kamera. Für die konkrete Anwendung (siehe Kapitel 4) wurde ein Raspberry Pi 3 Model B+ und ein Pi Camera Module v2.1 verwendet. Auf diesem läuft kontinuierlich eine Software, welche den Kamera-Input verarbeitet und hieraus die rein- und rausfahrenden Autos erkennt. Die Herausforderung besteht darin, nicht nur die Erkennung einzelner Fahrzeuge in den einzelnen Frames zu ermöglichen, sondern auch das Ein- und Ausfahren der Fahrzeuge zu identifizieren. Um dieses Problem zu lösen, werden zwei verschiedene Implementierungen, welche im Folgenden erklärt werden, entwickelt und in Kapitel 4 getestet.

3.1.1 Variante 0: einfache Objekterkennung

Zunächst wird versucht, eine Objekterkennung mit dem OpenCV-Framework umzusetzen. Das Verfahren verwendet Computer-Vision-Techniken wie Hintergrundsubtraktion und Konturen-

erkennung, um Fahrzeuge in einem Video zu erkennen und zu zählen. Dabei werden Bewegungsunterschiede zwischen Frames erkannt und die Konturen bewegter Objekte identifiziert, um potenzielle Fahrzeuge zu identifizieren. Die Anzahl der Fahrzeuge wird basierend auf der Position der Konturzentren ermittelt. Nach den ersten Tests stellt sich die Methode allerdings als unbrauchbar heraus. Das Verfahren ist zwar sehr performant, sodass bis zu 30 Bilder pro Sekunde verarbeitet werden können, die Genauigkeit ist für die Umsetzung des Projektes jedoch zu gering. Teilweise werden einzelne Fahrzeuge als mehrere gezählt. Weiterhin gibt es mehrere Objekte und Personen, die von dieser Methode fälschlicherweise als Auto erkannt werden. Deshalb wird diese Variante nicht weiter verfolgt und verbessert.

3.1.2 Variante 1: Richtung des Bewegungsvektors

Um Autos nur einmal zu erkennen wurde ein anderer Ansatz verfolgt, der im Folgenden beschrieben wird.

Die Grundidee des Algorithmus besteht darin, die Richtung zu ermitteln, in die ein Fahrzeug im Bild fährt. Hierbei wurde die vereinfachte Annahme getroffen, dass Autos, die sich im Kamerabild nach oben bewegen, aus dem Parkhaus fahren, während Autos, die sich nach unten bewegen, in das Parkhaus hinein fahren. Diese Annahme gilt nur, wenn die Kamera von oben aus dem Inneren des Parkhauses die Ein- und Ausfahrt filmt. Andernfalls müssten die Richtungen entsprechend angepasst werden.

Zunächst werden die Fahrzeuge in den einzelnen Frames mithilfe von einem vortrainierten Modell, konkret YOLOv5 [Joc20], für die Objekterkennung identifiziert. Die Eckkoordinaten der erkannten Fahrzeuge werden dann verwendet, um die Mittelpunkte der Fahrzeuge zu berechnen. Die Berechnung der Bewegungsrichtung auf Basis der Mittelpunkte erfolgt in mehreren Schritten. Hierfür werden für jeden Punkt im aktuellen Frame die Distanzen zu den Punkten des vorherigen Frames ermittelt. Hiervon wird jeweils der Punkt mit der kürzesten Distanz ausgewählt und der Bewegungsvektor zwischen den zwei Punkten berechnet. Sollte bereits ein passender Bewegungsvektor für den Punkt aus vorherigen Frames vorhanden sein, wird dieser Vektor aufaddiert, da nur das Gesamtergebnis relevant ist. Der hieraus resultierende Vektor zeigt also von der ersten Sichtung des Autos zu seiner aktuellen Position und ist in Abbildung 3 blau dargestellt.

3 Technische Umsetzung



Abb. 3: Beispiel der Fahrzeugerkennung mit Variante 1 (Quelle: eigene Darstellung)

Zur Zählung werden die errechneten Bewegungsvektoren verwendet, wenn das entsprechende Auto das Kamerasichtfeld verlassen hat. Das ist der Fall, wenn Punkte des vorherigen Frames für keinen Punkt im aktuellen Frame die kürzeste Verbindung darstellen, weil jetzt weniger Autos im Bild zu sehen sind als im vorherigen Frame. In diesem Fall werden die zuvor berechneten Bewegungsvektoren der nicht zugeordneten Punkte für die Zählung verwendet. Selbst wenn zwischen den Frames ein neues Auto in den Bildausschnitt fährt, während ein anderes ihn verlässt, stellt dies kein Problem dar, weil dies aufgrund eines Schwellwerts nur für Autos passieren kann, die in entgegengesetzte Richtungen fahren und sich die aufaddierten Bewegungsvektoren folglich ausgleichen, sodass der Zählerwert ebenso unverändert bleibt. In Abbildung 3 ist in Weiß der Vektor des vorherigen Autos zu sehen.

Die Richtung des Fahrzeugs wird anhand der y-Koordinate des Bewegungsvektors bestimmt. Wenn die y-Koordinate des Vektors positiv ist, fährt das Fahrzeug nach unten, andernfalls fährt es nach oben, was durch die bereits erwähnte Vereinfachung einen Rückschluss auf die tatsächliche Fahrtrichtung des Autos ermöglicht. Der interne Zähler des Sensors wird daraufhin entsprechend inkrementiert oder dekrementiert und über eine HTTP PUT-Request im, in Kapitel 3.2 beschrieben Format, als JSON an den Server übertragen.

3.1.3 Variante 2: Überschreiten einer Linie

Die grundlegende Idee des Algorithmus besteht darin, Fahrzeuge in den Frames eines Videos zu identifizieren und dann ihre Bewegung in Bezug auf eine festgelegte Linie zu verfolgen, um die Fahrtrichtung zu bestimmen. Die Linie wird dabei horizontal über der Einfahrt des Parkhauses festgelegt, wie dies in Abbildung 4 zu sehen ist. Somit wird das Bild in zwei Zonen geteilt. Ändert sich die Position eines Autos von der oberen in die untere Zone des Bildes, so wird die Anzeige der freien Parkplätze um eins verringert, da sich nun ein Auto mehr im Parkhaus befindet. Geschieht eine Änderung der Position von unten nach oben, so wird die Anzeige um eins erhöht.

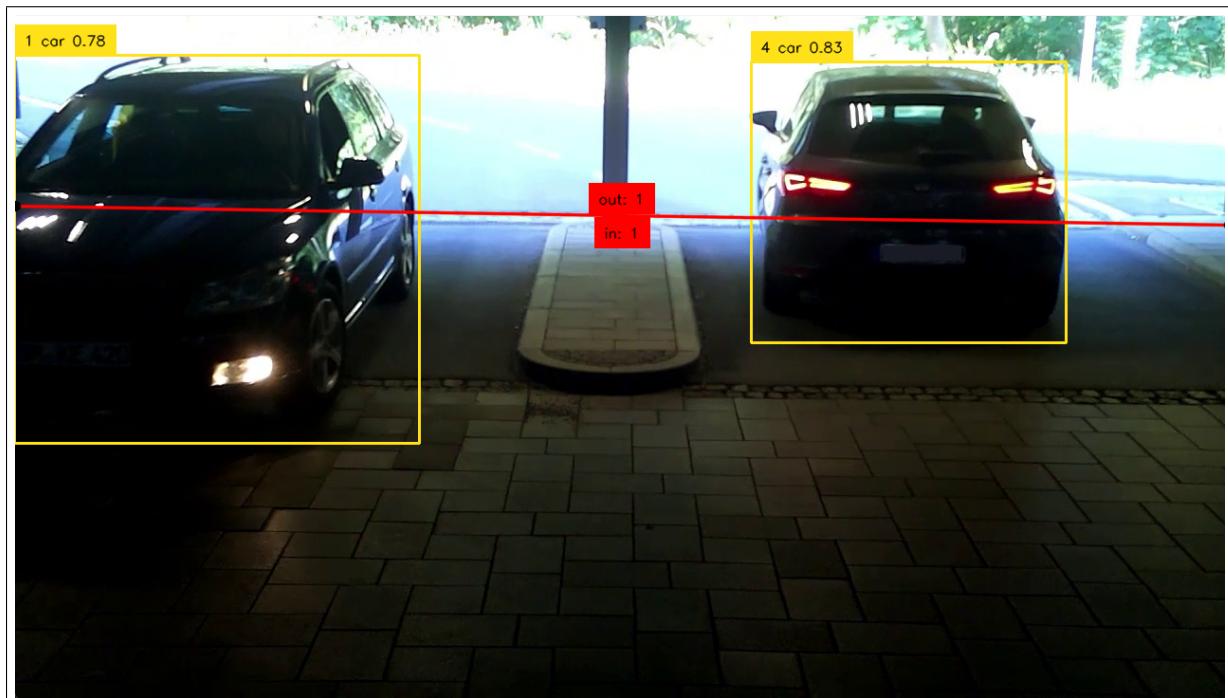


Abb. 4: Beispiel der Fahrzeugerkennung mit Variante 2 (Quelle: eigene Darstellung)

Es wird ein vortrainiertes YOLOv8-Modell eingesetzt, um die Fahrzeuge in den einzelnen Frames des Videos zu erkennen. Der Vorteil des Frameworks ist es, dass Objekten IDs zugeordnet werden können. Somit ist eine eindeutige Zuordnung eines Autos über mehrere Frames hinweg möglich. Die Zuordnung einer ID zu einem konkreten Auto über eine gewisse Dauer geschieht über verschiedene Merkmale des Objektes, wie die Eckpunkte der Objekterkennung, die Geschwindigkeit des Autos oder die Veränderung der Zuverlässigkeitswerte des Algorithmus. Die erkannten Fahrzeuge werden durch ihre Eckkoordinaten repräsentiert. Der daraus errechnete Mittelpunkt gibt Aufschluss über die Position des Fahrzeuges.

Der Algorithmus nutzt die erfassten Zählerwerte, um die aktuelle Auslastung des Parkhauses an einen Server zu übertragen. Dazu wird eine HTTP PUT-Request im JSON-Format verwendet, um die Zählerwerte an den Server zu senden und die Fahrzeuggbewegungen zu protokollieren.

3.2 Server

Der Server wurde in der Programmiersprache Go entwickelt und dient zur Verwaltung und als zentrale Stelle der Kommunikation für die Übermittlung der Zählerwerte von Sensor zum Endnutzer.

Zum Entgegennehmen der Werte vom Sensor wird die HTTP-Schnittstelle `PUT /api/v1/c/{site_id}` zur Verfügung gestellt. Die Daten werden für diese im HTTP Body in JSON übertragen. Das vom Server erwartete JSON Objekt, welches nachfolgend beispielhaft dargestellt ist, enthält dabei aktuell lediglich das Attribut `currentCars`, welches den aktuellen Zählerstand als Ganzzahl enthält.



Damit diese HTTP-Request zum Ändern des Zählerstandes nicht von jedem ungeschützt verwendet werden kann, ist eine Authentifikation nötig. Konkret besitzt hierzu jeder Sensor, der mit dem Server verbunden ist, einen eindeutigen Benutzernamen (`site_id`) sowie ein Passwort. Zur Authentifizierung mit dem Server wird hierzu die HTTP Basic Authentication genutzt, bei welcher Nutzernname (`site_id`) und Passwort im HTTP Header als Base64 String kodiert übertragen werden. Auf komplexere Authentifizierungsverfahren, wie beispielsweise JSON Web Tokens (JWT) wurde verzichtet, da die gebotenen Vorteile, wie beispielsweise die Möglichkeit zum clientseitigen Ausloggen, das automatische Ablauen des Tokens oder der Schutz vor Klau des Passworts durch Verwendung des Tokens, in diesem Anwendungsfall als API nicht benötigt werden. Zu Entwicklungszwecken wurde bisher auf die Verwendung von einer verschlüsselten Datenübertragung mittels HTTPS verzichtet. In einer Produktivumgebung sollte zwingend

3 Technische Umsetzung

HTTPS verwendet werden, um eine geschützte Übertragung zu gewährleisten. Besonders wichtig ist dies, weil ohne verschlüsselte Verbindung das Passwort abgefangen werden kann.

Als Ausgabeschnittstelle stellt der Server die Daten sowohl über HTTP als auch über MQTT bereit. Mittels HTTP kann über die Schnittstelle `GET /api/v1/c/{site_id}` das JSON Objekt, welches nachfolgend beispielhaft zu sehen ist, mit den Zählerinformationen abgerufen werden. Der Vorteil von MQTT besteht darin, dass Clients Daten abonnieren können, anstatt sie aktiv abzurufen. Der Server fungiert als MQTT-Broker und Clients können sich mit ihm verbinden, um Topics zu abonnieren und bei Änderungen dieser die Daten zu empfangen. Die `site_id` wird hierbei als Topic des MQTT-Protokolls verwendet. Sobald vom Sensor, wie zuvor beschrieben, mittels HTTP der Zählerstand aktualisiert wird, publiziert der Server diesen auf die entsprechende MQTT-Topic. Beim Verbinden mit dem MQTT-Server wird der letzte bekannte Zustand an den Client gesendet, da das `retain`-Flag auf `true` gesetzt ist. Somit sind direkt aktuelle Daten verfügbar, ohne dass zunächst eine Änderung durch den Sensor gemeldet werden muss. Die Clients haben jedoch keine Berechtigung, Daten zu veröffentlichen (`publishen`). Es gibt daher keine Beschränkungen für die MQTT-Verbindung aufgrund von IP-Adressen oder Anmeldungen mit Passwörtern.

```
Output JSON

{
    "id": "HS_Coburg",
    "displayName": "Parkhaus Hochschule Coburg",
    "currentCars": 100,
    "maxCars": 530
}
```

Zur Speicherung der relevanten Informationen verwendet der Server eine SQLite-Datenbank mit zwei Tabellen. Die erste Tabelle, `counter`, speichert die aktuellen Zählerstände sowie gleichbleibende Daten der Sensoren, wie beispielsweise die maximale Anzahl der Autos, die im JSON der Ausgabeschnittstelle vorhanden sind. Die zweite Tabelle, `sites`, enthält die Zugangsdaten für die einzelnen Sensoren. Das Passwort wird hierbei nur als Hash-Wert gespeichert. Die Einträge zu Zugangsdaten und Details der Parkplätze müssen zuvor manuell in die Datenbank eingetragen werden. Hierzu wäre es in einem finalen Produkt möglicherweise erforderlich, eine zusätzliche

Anwendung für Mitarbeiter bereitzustellen. Diese Anwendung könnte eine Benutzeroberfläche bieten, um neue Sensoren hinzuzufügen, ihre Zugangsdaten einzugeben und weitere Konfigurationen vorzunehmen. Für den Rahmen des aktuellen Prototyps genügt die Oberfläche von Drittanbieter Datenbanksoftware, wie beispielsweise der *DB Browser for SQLite*.

3.3 App

Um die aktuelle Parkhausbelegung für den Nutzer darzustellen, wurde eine App entwickelt, die mithilfe von Flutter und Dart geschrieben wurde und daher verschiedene Plattformen, wie Android, Windows, iOS und macOS, bedient. In Abbildung 5 sind Screenshots der App dargestellt.

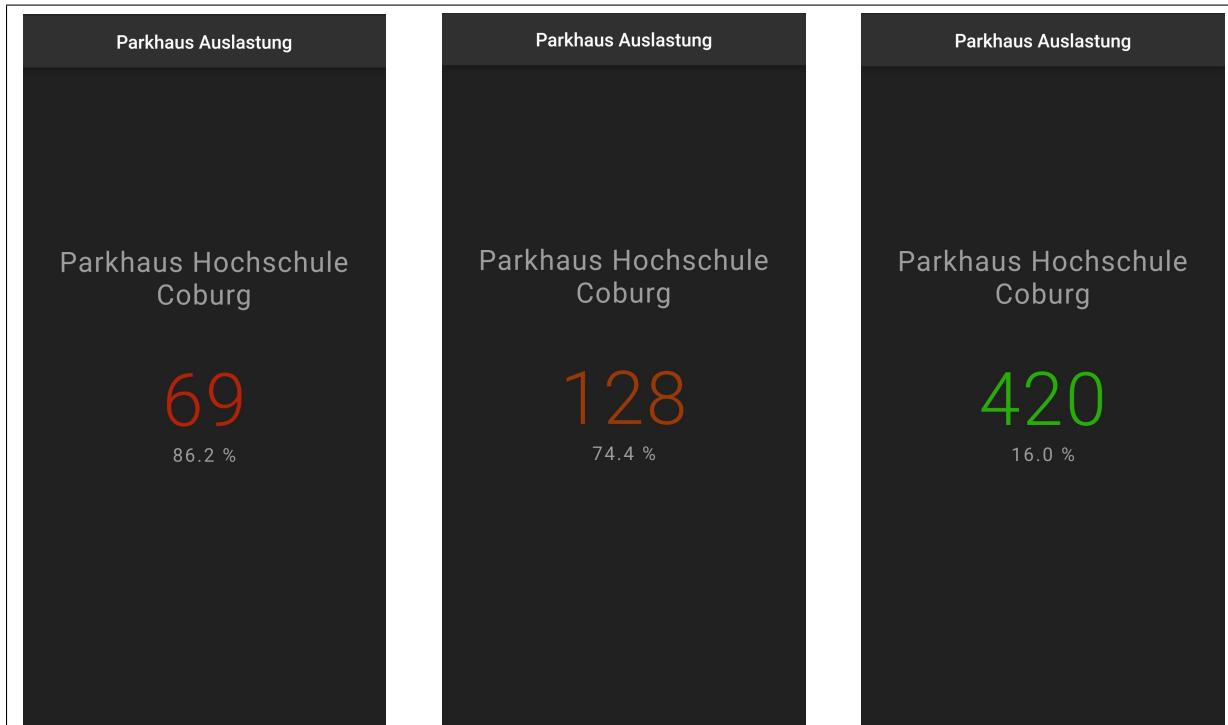


Abb. 5: Screenshots der App (Quelle: eigene Darstellung)

Flutter und Dart stellen ein leistungsstarkes Framework und eine Programmiersprache dar, die von Google entwickelt wurden, um die Entwicklung plattformübergreifender Apps zu vereinfachen. Flutter ermöglicht die Erstellung reaktiver Benutzeroberflächen, während Dart eine effiziente und objektorientierte Programmierung unterstützt. Die Verwendung einer einzigen Codebasis für verschiedene Plattformen verbessert die Wartung und Skalierbarkeit der App erheblich. [AWS] [Dar]

3 Technische Umsetzung

Wie bereits genannt, dient MQTT als leichtgewichtiges und zuverlässiges Kommunikationsprotokoll zum Empfangen der Nachrichten. Die App abonniert die Topic des entsprechenden Parkplatzes beim Broker, um diese Informationen anzuzeigen. Im aktuellen Prototypen ist die Topic hartkodiert.

4 Anwendung und Test

In diesem Kapitel wird die Anwendung und der Test des entwickelten Systems vorgestellt. Zunächst wird die allgemeine Nutzung des Systems erläutert. Anschließend werden die Installation des Systems im Hochschul-Parkhaus und die Aufzeichnung von Testdaten beschrieben sowie ein Vergleich der beiden entwickelten Zählverfahren vorgenommen.

4.1 Nutzung des Systems

Das entwickelte System zur Fahrzeugzählung bietet eine einfache und benutzerfreundliche Anwendung. Für den Betreiber des Parkhauses ist es lediglich nötig die Sensoreinheit an der Ein- und Ausfahrt des Parkhauses zu montieren und für Strom und eine Internetverbindung zu sorgen. Nachdem die Authentifizierungsdaten hinterlegt wurden, kann direkt mit der Zählung begonnen werden. Auf der Seite des Nutzers wird aufgrund der plattformübergreifenden App eine Installation und Nutzung auf verschiedenen Endgeräten, darunter Android, Windows, iOS und macOS, ermöglicht. Die Zählergebnisse werden dabei in Echtzeit angezeigt und können überwacht werden.

4.2 Installation und Testaufzeichnung

Zur Überprüfung der Genauigkeit des entwickelten Systems wurde ein Test durchgeführt. Das System wurde dabei im Parkhaus der Hochschule Coburg installiert, um statt der eigentlichen live Fahrzeugerkennung zunächst ein Video aufzunehmen, welches als Vergleichsmaterial der Verfahren genutzt werden kann. Auch, weil zum aktuellen Zeitpunkt im Parkhaus keine Internetverbindung besteht, wurden alle weiteren Tests auf Basis des aufgezeichneten Videomaterials durchgeführt. Technisch ergibt sich hierdurch kein Unterschied im Vergleich zur Echtzeitanalyse, da die verwendeten Verfahren jeweils nur auf den aktuellen Frame zurückgreifen.

4.3 Vergleich der beiden Zählverfahren

Im Rahmen der Arbeit wurden zwei verschiedene Zählverfahren (siehe Kapitel 3.1) entwickelt und implementiert. Um zu entscheiden, welches dieser Verfahren besser ist, wurden diese

auf Korrektheit und Effizienz verglichen. Hierzu wurde ein 10 Minuten langer Ausschnitt der Aufzeichnung von beiden Verfahren analysiert und mit der manuellen Zählung der Autos verglichen. Dabei ergaben sich folgende Werte.

Verfahren	Autos rein	Autos raus	Autos gesamt
Manuelle Zählung	10	4	6
Verfahren 1 (Kapitel 3.1.2)	10	4	6
Verfahren 2 (Kapitel 3.1.3)	10	4	6

Tab. 1: Ergebnisse des Tests

Demnach erzielen beide Verfahren korrekte Ergebnisse. Während des Testvideos fuhren außerdem drei Autos durch den Bildausschnitt, die nicht ins Parkhaus fuhren oder aus dem Parkhaus kamen. Diese wurden jeweils korrekterweise von den Verfahren nicht beachtet.

Auch die Effizienz der Verfahren wurde verglichen. Hierzu wurde die Zeit ermittelt, die jedes Verfahren pro Frame zur Verarbeitung benötigt. Die Ergebnisse des 10-minütigen Tests wurden als Boxplots in Abbildung 6 dargestellt. Aufgrund des ursprünglich geplanten Einsatzes eines Raspberry Pi, wurde der Test auf diesem durchgeführt. Hierbei ergab sich ein Median für Verfahren 1 von rund 2,04 Sekunden und für Verfahren 2 von rund 1,85 Sekunden. Als Boxplot sind die Messwerte in Abbildung 6a zu sehen, in der aufgrund der Ausreißer deutlich wird, dass Verfahren 2 in seltenen Fällen sehr lange für die Verarbeitung eines einzelnen Frames benötigt. Dies macht sich bereits im Mittelwert bemerkbar, da dieser bei rund 1,92 liegt, während er für Verfahren 1 dem Median entspricht.

Um sicherzugehen, dass die starken Ausreißer nicht an äußeren Umständen des Raspberry Pi, wie beispielsweise dessen Stromversorgung oder passiven Kühlung, liegen, wurde der Test ebenfalls auf einem Rechner mit aktiver Kühlung, stabiler Stromversorgung und mehr Rechenleistung durchgeführt. Erwartungsgemäß konnten allgemein bessere Ergebnisse erzielt werden. Die starken Ausreißer bei Verfahren 2 blieben allerdings bestehen. Im Median erreichte jetzt Verfahren 1 mit rund 0,10 Sekunden sogar einen minimal besseren Wert als Verfahren 2 mit rund 0,11 Sekunden.

Aus den Messdaten wird also ersichtlich, dass Verfahren 2 auf dem Raspberry Pi sowohl im Median als auch im Mittelwert performanter ist als Verfahren 1. Allerdings ist Verfahren 1 im Mittel ähnlich effizient aber besitzt vor allem eine niedrigere Varianz. Daher wird eine konstantere

4 Anwendung und Test

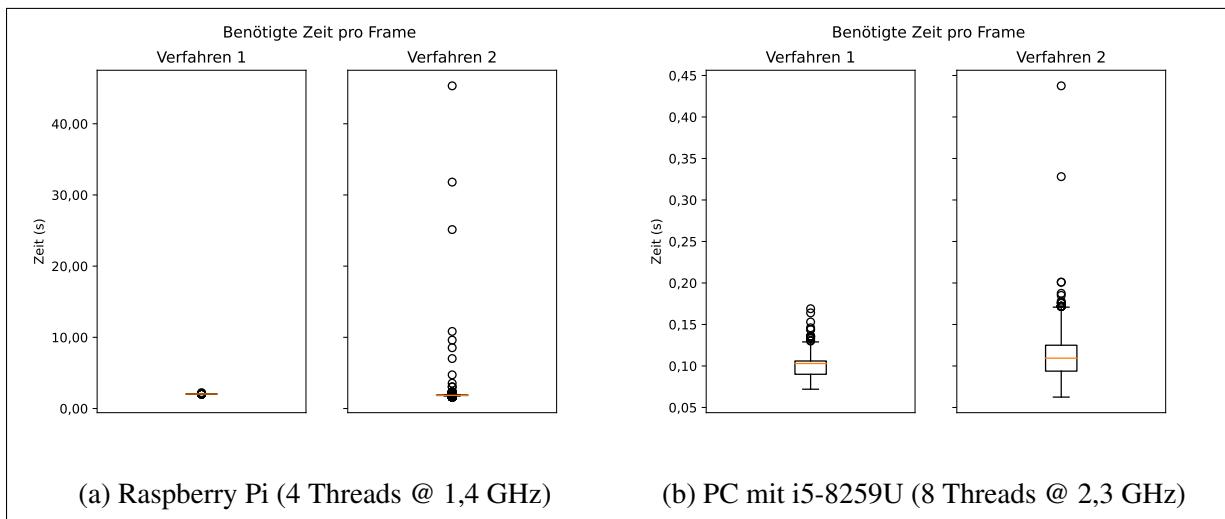


Abb. 6: Darstellung der Boxplots der benötigten Zeit pro Frame für die zwei entwickelten Verfahren auf dem Raspberry Pi 3 B+ und einem Intel NUC NUC8i5BEH (Quelle: eigene Darstellung)

Zeit pro Frame ermöglicht, was die Messung berechenbar macht. Aus diesen Gesichtspunkten wird sich für Variante 1 entschieden.

Da Verfahren 2 die Autos anhand ihrer Merkmale erkennt, um diesen IDs zuzuweisen, müssen Informationen zu diesen gespeichert werden. Daher ist die Vermutung, dass intern Datenstrukturen aufgebaut oder umstrukturiert werden, was dazu führt, dass die Hardware zusätzlich zur eigentlichen Berechnung kurzzeitig stärker beansprucht wird und die Berechnung des Frames infolgedessen deutlich mehr Zeit benötigt.

5 Fazit

Die Anwendung und der Test des entwickelten Fahrzeugzählsystems wurden in einem realen Parkhaus durchgeführt, wobei sich das System als zuverlässig in der Zählung der Fahrzeuge erwies. Der Vergleich der beiden implementierten Zählverfahren ergab, dass beide Verfahren eine hohe Genauigkeit und Robustheit aufweisen, Verfahren 1 jedoch bei ähnlicher Effizienz konstantere Zeiten benötigt. Die Ergebnisse dieser Tests sind vielversprechend und zeigen das Potenzial des Systems für den praktischen Einsatz in der Parkhausauslastungsmessung auf.

Für eine Weiterentwicklung des Systems wurden folgende Aspekte identifiziert.

Um die Performance des Systems zu erhöhen, sollten Alternativen für den Mini-Computer in Betracht gezogen werden. Der Raspberry Pi 3 B+ hat sich für diese Aufgabe als nicht leistungsstark genug herausgestellt. Vertreter der Jetson-Reihe von NVIDIA oder Single-Board-Computer der ASUS Tinker-Board-Reihe bieten beispielsweise eine deutlich höhere Rechenleistung. Gleichzeitig werden alle anderen Anforderungen für die Objekterkennung und die Kommunikation zum Server von den genannten Boards erfüllt.

Um die Sicherheit und den Datenschutz zu gewährleisten, sollte eine Umstellung auf HTTPS erfolgen. Dies würde die Übertragung der Daten verschlüsseln und die Vertraulichkeit der Informationen gewährleisten. Dadurch wird sichergestellt, dass die erfassten Messdaten und vor allem Zugangsdaten vor unbefugtem Zugriff geschützt sind.

Im aktuellen Prototypen müssen Parkhäuser initial manuell in der Datenbank angelegt werden. Eine für den großflächigen Einsatz nötige Erweiterung wäre die Implementierung eines Systems, über welches Mitarbeiter bequem und fehlerfrei neue Stationen anlegen können.

Eine Weiterentwicklung der Anwendung könnte eine dynamische Parkplatzauswahl in der App sein. Statt fester Standortzuweisungen könnten die verfügbaren Parkplätze und Parkhäuser in der App angezeigt werden, wobei die Darstellung bei großflächigem Einsatz auf einer Karte erfolgen könnte. Dies würde den Benutzern ermöglichen, freie Parkplätze leichter zu finden und die Effizienz des Parkplatzmanagements zu steigern. Außerdem wäre so nicht für jedes Parkhaus eine eigene App nötig.

Die Einbindung der Mitarbeiterparkplätze in das Zählsystem wäre eine sinnvolle Erweiterung. In der Umsetzung des Prototyps wird nicht beachtet, dass innerhalb des Parkhauses ein separater

5 Fazit

Bereich für Mitarbeiter der Hochschule existiert. Durch Anbringung eines zweiten Sensors und Subtrahieren der Zählergebnisse der Mitarbeiterparkplätze von den Gesamtzählergebnissen kann auch eine Differenzierung zwischen den Bereichen vorgenommen werden.

Zusammenfassend eröffnet die entwickelte Fahrzeugzählungsanwendung eine Vielzahl von Möglichkeiten für zukünftige Verbesserungen und Erweiterungen. Die Kombination dieser Aspekte könnte das Fahrzeugzählungssystem zu einem sinnvollen Werkzeug für ein effizientes und intelligentes Parkplatzmanagement machen. Bereits der Prototyp konnte zeigen, wie ohne den Einsatz spezieller Hardware eine Zählung ermöglicht wird.

Literaturverzeichnis

- [AWS] AWS. *Was ist Flutter? – Flutter-App erklärt* – AWS. Amazon Web Services, Inc. URL: <https://aws.amazon.com/de/what-is/flutter/> (besucht am 28.07.2023).
- [Ber23] BerryBase. *Raspberry Pi Camera Module 8MP v2*. 29.07.2023. URL: <https://www.berrybase.de/raspberry-pi-camera-module-8mp-v2?c=341>.
- [Cob23] Hochschule Coburg. *Parkhaus , Campus Friedrich Streib - Hochschule Coburg - Audioguide bei guidemate*. 29.07.2023. URL: <https://guidemate.com/station/Parkhaus-60d9b6bb355bd50c936d0f73?selectedGuideLocale=de>.
- [Dar] Dart. *Dart Overview*. URL: <https://dart.dev/overview.html> (besucht am 28.07.2023).
- [INC23] ASUSTeK COMPUTER INC. *Tinker Board - AIoT & Industrial Solution* - ASUS Deutschland. 29.07.2023. URL: <https://www.asus.com/de/networking-iot-servers/aiot-industrial-solutions/all-series/tinker-board/>.
- [Joc20] Glenn Jocher. *YOLOv5 by Ultralytics*. Version 7.0. 2020. DOI: [10.5281/zenodo.3908559](https://doi.org/10.5281/zenodo.3908559). URL: <https://github.com/ultralytics/yolov5>.
- [Ltd23] Raspberry Pi Ltd. *Buy a Raspberry Pi 3 Model B+ – Raspberry Pi*. 28.07.2023. URL: <https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/>.
- [NVI23] NVIDIA. *Eingebettete Systeme von NVIDIA für autonome Maschinen der nächsten Generation*. 27.07.2023. URL: <https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/>.
- [Ras18] the Raspberry Pi Foundation. *Raspberry Pi Camera Module v2 with ribbon*. 22. März 2018. URL: https://commons.wikimedia.org/wiki/File:Raspberry_Pi_Camera_Module_v2_with_ribbon.jpg (besucht am 02.03.2024).
- [Wis23a] Hochschule für angewandte Wissenschaften Coburg. *Anfahrt :: Hochschule Coburg*. 29.07.2023. URL: <https://www.hs-coburg.de/anfahrt.html>.
- [Wis23b] Hochschule für angewandte Wissenschaften Coburg. *Hochschule Coburg in Zahlen :: Hochschule Coburg*. 29.07.2023. URL: <https://www.hs-coburg.de/ueber-uns/zahlen-daten-fakten.html>.

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich meine/n Projektdokumentation mit dem Titel

AutoCount: Kamera-basierte Erfassung der Parkhausbelegung

selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie nicht an anderer Stelle als Prüfungsarbeit vorgelegt habe.

Ort

Datum

Unterschrift

Ort

Datum

Unterschrift