Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Rechnerarchitektur

Prof. Dr. Dietmar Fey

Masterprojekt

Smartcameras
Detektion von Straßenschildern und Pylonen mittels Viola-Jones Algorithmus

von Christoper Werner

und
Maximilian Langohr

März 2018

Inhaltsverzeichnis

1 Motivation **1**

2 Der Viola und Jones Algorithmus **3**

2.1 Vorbereitungen und grundlegender Ablauf 3

2.2 Antrainieren der Klassifizierer 6

2.2.1 Voraussetzungen 6

2.2.2 Antrainieren 7

2.3 Probleme 17

3 VJCMS **20**

3.1 Ziele 20

3.2 Voraussetzungen 21

3.3 Aufbau des Programms 22

3.3.1 VJCMS 23

3.3.2 Viola 23

3.3.3 objectdetector 23

3.3.4 VJ_Logging 23

3.3.5 VJ_Exception 24

3.3.6 monitor.py 24

3.4 Funktionalität 25

3.4.1 detect_and_track 26

3.4.2 detect 26

3.4.3 optimized 27
3.4.4 Weitere Features 27

4 Auswertung **29**

5 Fazit **30**

Literaturverzeichnis **33**

Kapitel 1

Motivation

Autonomes Fahren ist eines der großen Themen der Automobilindustrie der letzten Jahre. Der Traum vom eigenständig fahrenden Auto besteht schon länger, jedoch gestaltet sich die komplett selbstverwaltende Fahrzeugsteuerung als schwierig, da unheimlich viele Faktoren miteinbezogen sowie analysiert werden müssen.
Einer dieser Faktoren ist die erfolgreiche und korrekte Erkennung von Straßenschildern sowie wegweisenden Objekten, in diesem Fall Verkehrshütchen. Ansätze hierzu gibt es viele, jedoch ist die Forschung der Automobilindustrie meist geheim, weshalb eigene Ansätze kreiert werden müssen.
Dieses Projekt stellt den Versuch da, sich diesem Thema mit einer Methode zu nähern. Das Projekt verfolgte die Verwirklichung von Zwei Anforderungen:

1. Es sollen zwei nebeneinanderstehende Verkehrshütchen, auch Pylonen genannt, korrekt erkannt werden und die Position der erkannten Objekte im Bild zurückgegeben werden.

2. 3 Arten von Verkehrszeichen, das Stopschild, Vorfahrt gewähren sowie das Vorfahrtszeichen sollen erkannt werden.

Das Ziel, was hierbei verfolgt wird ist, dass einerseits die Erkennung von Pylonen dazu genutzt werden kann einen Parcours aufzustellen, durch den das Auto eigenständig navigiert, sowie die Erkennung von Verkehrszeichen auf einer Teststrecke für korrektes

Fahrverhalten genutzt werden kann. Die Voraussetzung ist, dass dazu der Algorithmus von Viola und Jones , welcher in einem Paper im Jahr 2001 vorgestellt wurde, für diesen Anweundgsfall adaptiert und genutzt wird.

Kapitel 2

Der Viola und Jones Algorithmus

In diesem Kapitel gilt es die Rahmenbedingungen für das erfolgreiche Nutzen des Viola-Jones Algrithmus abzustecken. Hierbei werden die Schritte für die Vorbereitungen erklärt und ein grober Überblick über den Ablauf mit detailreicheren Erklärungen für die interessanten Teilgebiete gegeben. Abschließend wird auf die Probleme, die bei dem Ablauf des Algorithmus entstehen können, eingegangen.

2.1 Vorbereitungen und grundlegender Ablauf

Die Liste an Vorbereitungen gliedert sich wie folgt:

Zu erkennendes Objekt

Es ist eine klare Vorstellung vom Objekt notwendig, wie dieses auszusehen hat und in welchen Situationen, also beispielswiese verschiedenen Winkeln und Lichtverhältnissen, dieses vorkommen kann.

Bilddatenbanken

Eine Datenbank an Negativbildern ist für den Algorithmus unabdingbar. Je größer die Datenbank ist, desto besser. Jedoch ist ein Limit von 10,000 bildern empfehlenswert, da der Trainingsprozess ansonsten zu lange dauert. Eine Datenbank an Positivbildern ist nicht zwingend erforderlich und hängt von der Art

des zu erkennenden Objekts ab. Ist das Objekt ein rigides Objekt reichen wenige Positivbilder, um das Training erfolgreich durchführen zu können. Im Falle eines nicht-rigiden Objekts, beispielsweise eines Gesichtes, ist es wichtig möglichst viele Formen und Farben zu haben, weshalb eine Datenbank an positiven Bildern notwendig ist.

Geeignete Datei- und Ordnerstruktur

Die Ordnerstruktur, in der die Bilder und Ergebnisse des Trainings abgelegt werden, sollte von vornherein durchdacht sein. In der Ordnerstruktur sollten sich jeweils ein Ordner für die positiven Bilder, einer für die negativen Bilder sowie ein Ordner für die zu speichernden Ergebnisse befinden. Auf der gleichen Ebene, auf der sich diese Ordner befinden sollten die Textdateien für die Pfade zu den Hintergrundbildern, Pfade zu den positiven Bilder, das Vectorfile mit Informationen über alle positiven samples in den positiven Bildern und das info File mit Informationen über das Vorkommen des Objekts in den positiven Bildern befinden.

Programm zum Antrainieren

Das in diesem Projekt verwendete Programm ist eine OpenCV-implementierung, die auf dem Raspberry PI kompiliert wurde. OpenCV liefert 3 weitere Tools, die zum Antrainieren der Klassifizierer verwendet wurden. Genaueres hierzu ist im Abschnitt 2.2.2 zu finden.

Rapsberry PI mit ROS

Für dieses Projekt wurde ein Raspberry PI 3 Modell B 1.2 verwendet. Das Betriebssystem ROS wurde auf eine Micro-SD Karte gespielt und lief standardmäßig auf dem Raspberry PI.

Optional: Kamera und Bildbearbeitungstool

In unserem Fall wurden alle positiven Samples mit einer Canon EOS 600D aufgenommen. Die anschließende Bearbeitung der Bilder erfolgte mit Photoshop

CC. Hierbei wurden die Bilder zurechtgeschnitten, Hintergründe geglättet und Unreinheiten im Bild beseitigt.

Der allgemeine Ablauf besteht aus der Vorbereitung der Trainingsdaten und Kaskadentraining. Diese beiden Hauptthemen können jedoch noch feiner untergliedert werden. Die Untergliederung für die Vorbereitung der Trainingsdaten sieht wie folgt aus:

1. **Erstellen der Liste der Negativbilder**

   Die Liste negativer Bilder wird über den Befehl

   ```
   find negative/ -iname "*.jpg" > bg.txt
   ```

   von uns erstellt. Dieser Befehl extrahiert sämtliche jpg-Dateinamen aus dem Verzeichnis '/negative' und schreibt diese in das Textfile 'bg.txt'

2. **Erstellen der Liste der Positivbilder**

   Die Liste der Positivbilder wird in der Regel per Tool erstellt, da diese ebenfalls die Objektvorkommnisse im Bild enthalten muss.

Anschließend gilt es die erstellten Trainingsdaten zu verwerten:

1. **Erstellen von positiven Samples**

   Positive Samples werden erstellt, indem das Tool Opencv_createsamples mit den entsprechenden Kommandoparametern gestartet wird. Hierbei wird die Datenbank an positiven Bildern erstellt, aus der anschließend das positive Vectorfile erstellt wird. Die hierbei erstellten Bilder unterscheiden sich von den rein positiven Bildern, da diese Bilder eine Kombination aus rein positiven Bildern und negativen Bildern darstellen und somit eine realere Situation für das Vorkommen eines Objektes im Bild darstellen.

2. **Erstellen des positiven Vectorfiles**

   Das positive Vectorfile fasst sämtliche Vorkommnisse von Objekten in den positiven Samples aus vorherigem Schritt in einem File zusammen. Dieses File ist essenziell für den weiteren Trainingsprozess.

3. **Kaskadentraining**

   Als letzter Schritt im Ablauf gilt das Antrainieren der Kaskaden. Hierbei wird das positive Vectorfile verwendet um in verschiedenen Stages mit schwachen Klassifizierern, auch weak classifiers, eine Kaskade zu erstellen, die das Objekt möglichst einwandfrei erkennt.

Nach der groben Beschreibung des Prozesses erfolgt nun die Erläuterung der einzelnen und wichtigeren Teilschritte in den folgenden Abschnitten.

2.2 Antrainieren der Klassifizierer

2.2.1 Voraussetzungen

Um einen Klassifizierer für den Viola Jones Algorithmus anlernen zu können, braucht dieser folgende Dinge:

1. Einen Datensatz an originalen positiven Bildern. Positiv heißt, dass in diesem Bild das zu erkennende Objekt enthalten ist.

2. Optional: Einen Datensatz an künstlichen positiven Bildern. Dieser kann mit entsprechenden Tools aus obigen originalen positiven Bildern und einem Datensatz an Negativbildern erzeugt werden.

3. Zu dem Datensatz mit positiven Bildern muss es eine Liste geben, die alle Vorkomnisse an Objekten in einem positiven Bild enthält. Diese wird meist als 'info.txt' abgespeichert und entspricht der Form

   [**Verzeichnis/Bildname**] [**Anzahl vorkommender Objekte**] [**X-Position Y-Position Breite Höhe**].

   Wobei der Ursprung des Koordinatensystem in der linken oberen Ecke ist und das Koordinatensystem sich über die normalerweise negative Y-Achse aufspannt.

4. Ein Datensatz an negativen Bildern. Negativ bedeutet, dass in diesem Bild das zu erkennende Objekt nicht enthalten ist. Es kann also irgendein wahlfreies Bild

sein, ohne das Vorkommen des Objekts. Hierzu muss ebenfalls eine Liste erstellt werden, die alle Bilder vereint. Diese Liste wird in der Regel 'bg.txt' genannt. Die Form hierfür ist
[**Verzeichnis/Bildname**].

5. Ein Programm, mit dem ein Haar-like Klassifizierer angelernt werden kann. Das in dieser Arbeit verwendete Programm ist OpenCV.

Die Anzahl an zu verwendenden Bildern wurde absichtlich weggelassen. Dies liegt daran, dass die Anzahl an benötigten Bildern stark variieren kann und vom Objekt abhängt. Im Beispiel eines rigiden Objekts können gute Ergebnisse mit wenigen Originalbildern erzielt werden, in unserem Fall ein Verkehrspylon. Im Falle eines nichtrigiden Objekts, beispielsweise einem Gesicht, kann es wesentlich mehr Abweichungen von der Norm geben, weshalb eine wesentlich größere Anzahl an originalen Positivbildern erzeugt werden muss.
Hat man obige Kriterien erfüllt, kann mit dem Lernprozess begonnen werden.

2.2.2 Antrainieren

Der Prozess des Antrainierens gestaltete sich zu Anfangs sehr schwierig. Das Problem hierbei war, dass die Dokumentation durch OpenCV an Stellen teils sehr dürftig ist. Die grundlegenden Schritte des Antrainierens eines Klassifizierers werden im Folgenden genau erörtert.

Opencv Annotations

Der erste Schritt zum Antrainieren qualifizierter Klassifizierer ist eine simple Vorbereitung der zu verwendenden Bilder, bei der es darum geht, die zu erkennenden Objekte in einem Bild zu markieren. Die Verwendung ist wie folgt:

```
opencv_annotations -annotations [info.txt] -images [Bilder]
```

Dieses Tool öffnet die in '-images' spezifizierten Bilder. Als Paramter kann hier ein Ordner, aus dem dann alle Bilder sequentiell geöffnet werden, oder eine einzelne Bilddatei angegeben werden. Die Angabe '[info.txt]' ist hierbei die Datei, in der die entsprechenden Angaben niedergeschrieben werden. Das gespeicherte Format wurde bereits im Abschnitt 3 erklärt.
Startet man das Programm via Konsole, erscheint ein Fenster in dem das spezifizierte Bild, bzw. das erste Bild im Ordner, angezeigt ist. Mit dem ersten Linksklick wird die Aufnahme der annotation gestartet. Das Bewegen der Maus erzeugt ein rotes Rechteck, dass vom ursprünglichen Mausklickpunkt startet. Ist das Rechteck korrekt gezogen, bestätigt ein zweiter Mausklick das gezogene Rechteck. Die Eingabe wird mit 'c' bestätigt. Werden anschließend weitere Objekte im Bild markiert, wird das vorherige rote Rechteck grün gefärbt. Dies indiziert, dass das Rechteck gespeichert wurde. Die gespeicherte Eingabe kann nicht mehr rückgängig gemacht werden.
Bei Eingabe von 'n' wird das nächste Bild im Ordner geladen, falls ein Ordner angegeben wurde, ansonsten wird der Prozess beendet und die Info Datei mit den entsprechenden Paramtern erstellt.

Opencv Createsamples

Der zweite Schritt des Antrainierens ist der wichtigste. Hier werden die Samples, mit denen später die Haar-like Features gebildet werden, erstellt. Die Algorithmus ist vielseitig verwendbar, da er verschiedene Kombinationsmöglichkeiten der Paramter betrachtet und anhand derer Samples erstellt.
In dieser Arbeit werden nur die Fälle vorgestellt, die zum erstellen der Samples verwendet wurden. Die erste Methode diente zur Erstellung künstlicher Samples.

```
opencv_createsamples -img Vorfahrtsschilder/Vorfahrt2.png

-info info.txt -bg bg.txt -num 1000 -maxidev 40

-maxxangle 0.2 -maxyangle 0.2 -maxzangle 0.2 -w 32 -h 32
```

```
-bgcolor 125 -bgthresh 10 -show
```

Obiger Befehl ist ein Beispiel, dass in dieser Form zur Erstellung von Samples für das zweite Vorfahrtsschild verwendet wurde. Die einzelnen Komponenten ergeben sich wie folgt:

Parameter	Erklärung
img	Beschreibt das Bild, welches für die Erstellung der künstlichen Bilder als Grundlage dient. Dieses Bild muss cropped vorliegen. Der Hintergrund muss eine einheitliche Farbe sein, damit das Bild in die Negativbilder ohne Rand oder ähnliches eingebettet werden kann.
info	Spezifiziert den Speicherort, an den die Informationen für die generierten Bilder geschrieben werden. Die Datei wird, falls vorhanden, überschrieben. Sie enthält anschließend zu jedem generierten Bild die Bounding Box für das Objekt in diesem Bild.
bg	Ist die Datei in der die Auflistung der Negativbilder steht.
num	Gibt die Anzahl an positiven Samples an, die erstellt werden soll.
maxidev	Gibt die maximale Abweichung des Intensitivitätsgrads an.
maxXYZangle	Maximale Drehung in RAD die für das Erstellen der Samples verwendet wird.
w und h	Geben die Breite und Höhe in Pixel an. Diese Werte sollten entsprechend der Form des Objekts gewählt werden. Beispielsweise bei einem Cone wird circa ein Seitenverhältnis von 1:2 vorhanden sein, weshalb die Höhe zweimal so groß wie die Breite sein sollte.
bgcolor	Gibt den Wert für die Hintergrundfarbe an. Am besten ist eine schwarze oder weiße Hintergrundfarbe, da bgcolor dann auf 0 bzw 255 gesetzt werden kann. Bei richtig gesetztem Wert wird der Hintergrund in der Sample Erstellung transparent.

bgthresh	Der Abweichungswert für die bgcolor Erkennung. Dieser Wert sollte möglichst klein (Maximal 10) sein, um nicht andere Bereiche des Objekts versehentlich als transparent darzustellen.
show	Ist dieser Parameter gesetzt werden Bilder, bevor sie erstellt werden, angezeigt. Es kann folglich überprüft werden, ob Werte wie bgcolor oder mögliche Winkel richtig gesetzt sind und somit das Sample korrekt in die Negativbilder eingebettet wird.

Tabelle 2.1: Opencv Createsamples: Parameter und deren Erklärung

Die obige Anwendung des Programms zeigt, wie künstliche Samples zum anlernen erstellt werden können. Der eigentliche Zweck des Programms ist es jedoch sogenannte Vectorfiles zu erstellen. Diese Vector Files sind für den Lernprozess essenziell, da sie die Vektordaten darüber enthalten, wo die zu erkennenden Objekte in den positiven Bildern liegen. Diese Datei wird anschließend für das antrainieren der Kaskaden genutzt. Der Befehl zum Erstellen dieser Datei sieht wie folgt aus:

```
opencv_createsamples -info info.txt -bg bg.txt

-num 3000 -vec pos.vec
```

Parameter	Erklärung
info	Spezifiziert den Speicherort, aus dem die Informationen zu den positiven Bildern geladen werden.
bg	Gibt erneut die Speicherposition der Negativbilder an.
num	Anzahl an positiven Samples, die erstellt wird.
vec	Speicherort, an dem die Ausgabe des Programms in Form eines vector Files stattfindet.

Tabelle 2.2: Opencv Createsamples: Parameter und deren Erklärung für die 2. Variante

Opencv Traincascade

Im letzten Schritt zum Antrainieren der Klassifizierer wird das Tool 'opencv traincascade' verwendet. Dieses Programm dient dazu, die in den vorherigen Schritten erstellten Samples zum antrainieren eines Klassifizierers zu verwenden. Eine beispielhafte Verwendung sieht wie folgt aus:

```
opencv_traincascade -data obj-classifier/ -vec pos.vec


-bg bg.txt -numPos 2800 -numNeg 2800 -minHitRate 0.999


-maxFalseAlarmRate 0.5 -numStages 14 -numThreads 8


-featureType HAAR -mode ALL -w 32 -h 32


-precalcValBufSize 4096 -precalcIdxBufSize 4096


-nonsym -baseFormatSave
```

Es ist offensichtlich, dass dieser Befehl eine Fülle an Optionen zum antrainieren mit sich bringt. Im Folgenden wird nur auf die wichtigsten und elementaren Aspekte dieses Befehls eingegangen.

Parameter	Erklärung
data	Im '-data' Abschnitt wird der Ordner festgelegt, in welchen die Stages, ein sogenanntes 'params.xml' File und die Kaskade abgelegt werden. Nach jeder Stage im Trainingsprozess wird ein neues File namens 'stageX.xml', wobei 'X' für die entsprechende Stage steht, in dem '-data' Ordner angelegt. Sollte das Training erfolgreich sein, indem der Prozess entweder bis zur in '-numStages' gesetzten Stage gekommen ist oder die entsprechende Erfolgsrate erreicht hat, wird ein File namens 'cascade.xml' erzeugt. Dieses File kann vom Viola-Jones Algorithmus genutzt werden. Das 'params.xml' File ist ein einfaches XML Konstrukt, dass einige Paramater, welche in obiger Codezeile verwendet wurden, speichert, um diese bei einer Fortsetzung des Trainings wiederverwenden zu können. Dies bedeutet natürlich auch, dass sobald der Trainingsprozess mit gesetzten Parametern begonnen wurden, diese nicht mehr geändert werden können. Außnahmen hierzu werden im an diese Tabelle folgenden Abschnitt erklärt.
vec	Gibt den Pfad zum pos.vec File, welches im Abschnitt 2.2.2 bereits erklärt wurde, an.
bg	Gibt den Pfad zur bg.txt Datei an.

numPos	Legt die Anzahl an positiven Samples fest, die für die Stage 0 verwendet werden. Hierbei ist zu beachten, dass je nach Anzahl an zu trainierende Stages ein Wert unterhalb des Limits an positiven Samples, die in pos.vec im vorherigen Schritt generiert wurden, gewählt werden muss. Dies liegt daran, dass pro Stage zusätzliche positive Samples mit in Betracht gezogen werden. Der Grund hierfür ist, dass das Trainingsergebnis auch mit Samples funktionieren soll, mit denen nicht gelernt wurde, also eine automatische Überprüfung der Zwischenergebnisse. Der Wert für '-numPos' sollte maximal entsprechend der Hitrate gewählt werden. Beispielsweise werden bei einer minHitrate von 0.99 pro Stage circa 1% positive Samples für den Trainingsprozess hinzugezogen.
numNeg	Gibt die Zahl an negativen Bildern an. Die Zahl darf nicht größer als die Anzahl an Bildern in der 'bg.txt' Datei sein. In manchen Fällen ist es ratsam, einen kleineren Wert an Negativbildern zu wählen, in der Regel sollte der Wert jedoch genau der Anzahl an Samples in der 'bg.txt' Datei entsprechen.
minHitRate	Setzt die Mindestrate, mit der die positiven Bilder erkannt werden müssen. Bei obigem Beispiel ist die minHitRate auf 0.999 gesetzt, was bedeutet, dass der Trainingsprozess trotzdem erfolgreich ist, wenn 1 aus 1000 Objekten nicht erkannt wurde. Die minHitRate hat direkten, aber nicht proportionalen, Einfluss darauf, wieviele neue positive Samples für die nächste Stage zum trainieren verwendet werden.

maxFalseAlarmRate	Gibt die Rate an, wieviele Weak Classifier einen sogenannten False Positive, also eine vermeintliche korrekte Erkennung des entsprechenden Bildabschnitts, abgeben dürfen. 0.5 entspricht dem Standardwert. Experimente mit diesem Wert wurden nicht unternommen, da alle Ergebnisse mit diesem Wert erzielt wurden.
numStages	Dieser Wert gibt an, bis zu welchem Grad das Training ausgeführt werden soll. Je mehr Stages, desto genauer ist das Ergebnis, also der Klassifizierer. Jedoch kann es passieren, dass im Falle einer zu hoch gewählten Anzahl an Stages das Ergebnis 'Overfitted' ist, also der Klassifizierer nicht mehr allgemein genug ist. Hierbei kann dann das Objekt nur unter ganz bestimmten Bedingungen korrekt erkannt werden. Die AcceptanceRatio ist ein Anhaltungspunkt anhand derer eine passende Anzahl an maximale Anzahl an Stages gewählt werden kann. Die AcceptanceRatio wird nach der Befehlserklärung erklärt.
numThreads	Für den Falle, dass Multithreading auf der Maschine zur Verfügung steht, kann dieser Parameter gesetzt werden. Multithreading führt zu einer deutlichen Zeitersparnis beim Antrainieren des Klassifizierers.
featureType	Für unser Training wurden Haar-like Features verlangt, weshalb dieser Parameter auf 'HAAR' gesetzt wurde. Andere Featuretypen wie beispielsweise LDBP wären ebenfalls möglich, waren jedoch nicht Gegenstand dieses Projekts.
mode	Dieser Parameter gibt die zulässigen Weak Classifier an. Es stehen die Modi 'BASIC', 'CORE' und 'ALL' zur Verfügung. Der Modus 'ALL' war bei uns eine zwingende Wahl, da unsere Objekte schräge Kanten enthalten und nur dieser Modus die entsprechenden Weak Classifier hierzu anwendet.

w und h	Geben die Größe, in Form von Breite und Höhe, der in pos.vec spezifizierten Samples an. Die Werte müssen zwingend den Werten der genutzten Samples entsprechend, da das Programm sonst den Trainingsprozess nicht durchführen kann.
precalcValBufSize	Genutzer Buffer im Hauptspeicher um Vorberechnungen auszulagern und den Trainingsprozess zu beschleunigen. Ein Wert von mindestens '1024' sollte hier gewählt werden, um eine Beschleunigung zu erreichen.
precalcIdxBufSize	Entsprechend dem Vorgänger sollte dieser Parameter den gleichen Wert erhalten, um eine Beschleunigung zu erzielen. Dieser Parameter muss implizit genutzt werden, sollte '-precalcValBufSize' genutzt werden.
nonsym	Lässt man diesen Parameter weg, wird die params.xml Datei nicht in die cascade.xml mit eingebunden. Unsere Implementierung von Viola-Jones braucht diesen Kopf jedoch, weshalb der Parameter verwendet wurde.
baseFormatSave	Dies ist ein Parameter, der die Abwärtskompabilität für Haar-like Features garantiert. Für gewöhnlich werden Haar-like features nicht mehr verwendet und das Programm ist fähig auch neuere Klassifiziererarten zu generieren, die ein neueres Format verwenden. Da in unserem Fall der alte Haar-like Featuretyp verwendet wird, muss dieser Parameter gesetzt werden.

Tabelle 2.3: Opencv Traincascade: Parameter und deren Erklärung

Trainingsprozess

Nachdem im vorherigen Abschnitt die einzelnen Parameter für den Trainingsbefehl erklärt wurden, gilt es nun den Trainingsprozess per se zu erklären.

Zu Beginn wird eine Übersicht der übergebenen Paramater gegeben, woraufhin mit der ersten Stage, Stage 0, der Trainingsprozess beginnt. Hierbei werden zuerst alle positiven Bilder eingelesen.

```
POS count : consumed    2800 : 2800
```

Die obige Codezeile gibt an, dass alle positiven Samples aus dem pos.vec File eingelesen wurden (linker Zahlenwert) und es den aktuellen Klassifizierer mit allen vorherigen Stages auf die Anzahl an zu testenden positiven Bildern (rechter Zahlenwert) anzuwenden gilt. Hierbei muss bei jedem Durchlauf 'N', auch Run genannt, die entsprechende minHitRate 'HR' überschritten werden.

```
NEG count : acceptanceRatio    2800 : 1
```

Diese Codezeile gibt an, wieviele der Negativbilder mit dem aktuellen Klassifizierer bis zur jeweiligen Stage korrekt erkannt worden. In obigem Beispiel ist der Wert 1, da es für Stage 0 noch keine Weak Classifier gibt, die Negativbilder ausschließen könnten. Von Stage zu Stage wird dieser Wert in der Regel kleiner. Ziel ist es, eine AcceptanceRation von circa 0.0001 zu erreichen. Dies bedeutet, dass nur ein Subwindow von 10000 fälschlicherweise das zu erkennende Objekt erkennt.

```
Precalculation time: 64
```

Anschließend wird Auskunft über die Precalculation time, also die genutzte Zeit für Vorberechnungen, ausgegeben. Vorberechnungen ersparen Zeit beim weiteren Trainingsprozess für diese Stage.
Anschließend beginnt der eigentliche Trainingsprozess, bei dem mit jedem Durchlauf auf dieser Stage ein neues Ergebnis generiert wird. Ziel ist es, auf den höheren Stages mit wenigen Runs und wenigen Weak Classifieren die entsprechende Hitrate und die maximale Falsealarmrate zu erhalten. Die Schwierigkeit für den Algorithmus hierbei ist es, die Falsealarmrate zu senken und trotzdem noch die entsprechende Hitrate zu

erfüllen. Es wird versucht die Falsealarmrate iterativ herabzusetzen. Eien Stage ist abgeschlossen, wenn in einem Run die erreichte Hitrate über der minimalen Hitrate und die Falsealarmrate unter der maximalen Falsealarmrate ist.
Der komplette Trainingsprozess gilt als erfolgreich abgeschlossen, sobald der Algorithmus bei der in 'numStages' spezifizierten Stage angelangt ist, oder vorher bereits die Meldung ausgegeben wird, dass

```
Required leaf false alarm rate achieved.
Branch training terminated.
```

Während des Trainingsprozesses kann jedoch eine Vielzahl an Problemen auftreten, die im Folgenden erläutert werden.

2.3 Probleme

Probleme, die während des Trainingsprozesses aufkommen können, sind so gut wie nicht dokumentiert. Es bedurfte einiger Nachforschungen um die Probleme ergründen und verstehen, sowie diese anschließend lösen zu können. Mögliche Probleme umfassen:

1. **Killed**

   Meist ist die Ursache hierfür, dass der Arbeitsspeicher nicht ausreicht. Der Trainingsprozess wird abrupt abgebrochen. Zurückzuführen ist dies auf die Implementierung von OpenCVs Trainscascade. Für den Trainingsprozess werden harte Kopien der Bilder erstellt. Ergo wird bei einer Anzahl von 3000 positive Samples und einer Subwindowgröße von 32x32 Pixeln eine sehr große Zahl an Bildern in den Hauptspeicher geladen. Das Limit kann schnell erreicht sein, weshalb der Prozess einfach abgebrochen und mit einem 'Killed' kommentiert wird.

2. **Branch Training terminated**

   Diese Aussage kann je nach Präfix positiv oder negativ sein. In einen Fall ist, wie bereits im vorherigen Abschnitt beschrieben, die entsprechende Zielrate der Bilderkennung erreicht worden. Eine andere Begründung für dieses Statement

kann jedoch sein, dass der Algorithmus keinen Sinn dahinter sieht, diese Stage zu berechnen, da das Ergebnis zu fein, also overfitted, mit der angegebenen Anzahl an 'numStages' sein wird. Der Prozess wird deshalb abgebrochen und das Training ist unvollständig. Der Prozess kann jedoch anschließend mit einer größeren Anzahl an 'numStages' gestartet werden.

3. **Wenige Runs pro Stage**

   In den ersten Stages ist es in Ordnung, wenn nur wenige Runs gestartet werden, um entsprechende Weak Classifier für diese Stage zu wählen. Je höher die Anzahl an Stages jedoch wird, desto mehr Runs sollten vorhanden sein. Als selbst festgelegte Faustregel kann von mindestens der Stage als Anzahl an Runs ausgegangen werden. Im Beispiel sollten bei der Berechnung für Stage 10 mindestens 10 Runs durchlaufen werden.

4. **cols == rows**

   Dieser Fehler beschreibt den Fall, dass in der pos.vec Datei andere Werte für Höhe und Breite gesetzt wurden, als die beim Training verwendeten Werte. Um diesen Fehler zu korrigieren müssen die Werte für Breite und Höhe für 'opencv createsample' und 'opencv traincascade' angepasst werden.

5. **Overfitting**

   Sollte das Training mit schlecht gewählten positiven Samples durchgeführt worden sein und die Acceptance Ratio einen sehr niedrigen Wert (ca. 0.0001) erreichen, kann der Klassifizierer overfitted, also nur speziell auf diesen Fall antrainiert, sein. Dieser Fehler lässt sich beseitigen indem die positiven Samples mit den von uns vorgeschlagenen Mitteln vorher bearbeitet werden oder eine schlechtere Erkennungsrate mit möglicherweise mehr false positives muss akzeptiert werden.

6. **Out of positive samples**

   Dieses Problem wurde im vorherigen Abschnitt 2.2.2 unter traincascade bereits erklärt. Dieser Fehler tritt auf, wenn beim Trainingsprozess durch iteratives

Erhöhen der positiven Samples der Maximalwert der verfügbaren Samples überschritten wird. Die einfachste Lösung hierzu ist, den Trainingsprozess mit weniger positiven Samples fortzusetzen. Alternativ können durch die bereits beschriebenen Methoden neue positive Samples und ein neues pos.vec File generiert werden. Jedoch muss der Trainingsprozess bei dieser Alternative neu gestartet werden, da sich das pos.vec geändert hat.

7. **Out of Range error**

   Diesem Fehler liegen korrumpierte Negativbilder zugrunde. Eine Sichtung der Negativbilder und löschen der fehlerhaften Bilder behebt diesen Fehler.

8. **nonsym und baseFormatSave**

   Sollten diese Parameter nicht gesetzt sein, ist das Fomrat der generierten Kaskaden nicht für unseren Algorithmus nutzbar. Ein Setzen der Parameter beim Training löst dieses Problem.

9. **Falsch generierte Samples**

   Via createsamples werden die genutzten positiven Samples generiert. Hierbei ist es unabdingbar für den Anfang den Parameter

   `-show`

   zu nutzen. Dieser Parameter zeigt die generierten Samples an. Zu Testzwecken können die Breite und Höhe auf beispielsweise 100 gesetzt werden, um das Testbild besser betrachten zu können. Sind in den angezeigten Bildern schwarze oder weiße Hintergründe zu sehen oder Transparenzen die nicht gewünscht sind, müssen die Werte

   `-bgcolor`

   und

   `-bgthresh`

   angepasst werden.

Kapitel 3

VJCMS

Dieser Abschnitt des Berichts befasst sich mit der Implementierung unserer Software zur Analyse der Bilddaten des Raspberry Pi's, im weiteren Pi abgekürzt. Der Name VJCMS (Viola-Jones Camshift) kommt von einem Programm, dass wir zu beginn unserer Recherechen gefunden haben. [Pat15] Dieses Programm bzw. Skript, da es sich um ein Python-Skript handelt, bildet funktional die Grundlage unserer Implementierung. Im Verlaufe der Arbeit haben wir das Konzept des Programms aufgenommen und in einer für unsere Zwecke dienlichen Form implementiert. In seiner ursprünglichen Form hat das Skript nicht nur Gesichter erkannt, sondern diese auch für eine vorher definierte Zeit mittels Camshift Algorithmus verfolgt.

3.1 Ziele

Das grundlegende Ziel des Projektes ist es auf einem Raspberry Pi 3, mittels dem Algorithmus von Viola-Jones, drei Verkehrsschilder und Pylonen zu erkennen. Im Anschluss sollen die Koordinaten der Bounding-Boxes (minimal umgebendes Rechteck) und ein entsprechendes Label zurückgegeben werden. Dabei hatten sich folgende Schritte ergeben, die es als erstes zu bewältigen gab:

1. Festlegen auf eine Programier- oder Skript-Sprache zur Implementierung

2. Ermitteln der nötigen Softwarepakete, Bibliotheken und Frameworks um diese Sprache und deren Funktionalitäten auf dem Pi nutzen zu können

3. Installation und einrichten der entsprechendenden Pakete und Bibliotheken

4. (Optional) Finden von Code-Beispielen oder vergleichbarer Implementierungen, um diese als Vorlage zu verwenden

Nachdem das Framework, mit Hilfe dessen das Programm entwickelt werden sollte, definiert wurde, haben wir ein Reihe von Funktionalitäten bestimmt, die wir für die Umsetzung des Projektes als nötig oder wünschenswert erachtet haben:

- Erkennen von Objekten und Ausgabe der Position im Frame
- Ausgabe eines Videostream mit den Bounding-Boxes
- Übermittlung des Video-Streams mittels TCP Verbindung an einen Server oder Client
- Implementierung einer Metrik zur Ermittlung der Performance der verschiedenen Modi

3.2 Voraussetzungen

Für die Implementierung und Ausführung unseres Programms sind folgende vorinstallierten Packeges nötig:

- Python 2.7
- OpenCV 3.3 oder höher für Python
- termcolor 1.1 (Installation durch pip-Installer)

Das Programm wurde in Python 2.7 geschrieben und setzt zu großen Teilen auf die Bibliotheken von OpenCV [OPe17]. OpenCV stellt an dieser Stelle ein Sammlung von Funktionen bereit, die denen von Viola-Jones entsprechen. Es gibt es diese Bibliotheken für verschiedene Programmiersprachen.
Für den Pi gibt es leider kein fertiges Package, dass einfach installiert werden kann, sondern es muss OpenCV selbst kompiliert werden. Hierbei können zwei Probleme auftretten. Das erste Problem ist die fehlende Kühlung der CPU des Pi's. Erst nachdem dieser mittels eines umgebauten Lüfters gekühlt wurde, konnte der Pi überhaupt lang genug in einem akzeptablen Temperaturbereich gehalten werden, um mit diesem arbieten zu können. Das zweite Problem war der mangelnde Arbeitsspeicher. Dieses konnte durch das Erweitern des Swap-Files gelöst werden.
Das Modul termcolor wird lediglich dazu genutzt die Ausgabe in der Konsole farblich aufzuarbeiten.

3.3 Aufbau des Programms

Die beiden Hauptkomponenten des Projekts bilden VJCMS.py und monitor.py. Neben den beiden Skripten gibt es noch eine Sammlung von Klassen, die in den Ordner utils ausgegliedert sind. Diese umfassen das Logging, Exceptions und eine Implementierung von Viola-Jones.

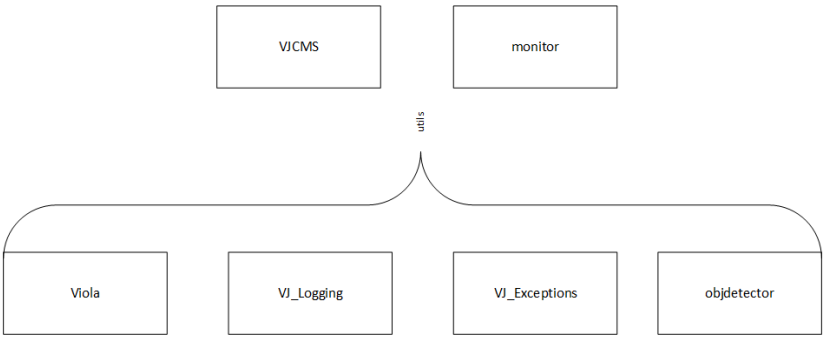

Abbildung 3.1: Programmaufbau

3.3.1 VJCMS

VJCMS bildet den Kern des Programms und übernimmt einen Großteil der Berechnungen. Neben der main-Methode befinden sich in diesem Modul Funktionen zur Ausgabe der Frames und zum Verfolgen der erkannten Objekte. Die Erkennung selbst ist wiederum in die Klasse Viola ausgelagert.

3.3.2 Viola

Die Klasse Viola übernimmt die Aufbereitung der Daten des Viola-Jones Algorithmus. Der Tatsächliche Zugriff auf diese Funktionen findet in der Klasse objectdetector statt. In Viola werden die einzelnen Klassifizierer für die verschiedenen Objekte aufgerufen und die zurückgegebenen Datensätze mit den entsprechenden Labels versehen.

3.3.3 objectdetector

Mittels dieser Klasse wird direkt auf die Funtkionalitäten von OpenCV zugregriffen und der Klassifizierer angewendet.

3.3.4 VJ_Logging

Das Logging umfasst neben der Ausgabe auf der Konsole auch die Ausgabe in einer Datei, die im Ordner log gespeichert wird. Nachdem sowohl VJCMS als auch monitor auf diese Klasse zugreifen gibt es zwei unterschiedliche Logging-Dateien. Insgesamt gibt es 3 Arten bzw. Klassen von Meldungen, die in diesem Projekt unterschieden werden. Zum einen gibt es INFO. Hierbei handelt es sich um Meldungen die zum Beispiel angeben welche Methode gerade aufgerufen oder welcher Modus verwendet wird. Die Klasse WARNING gibt Meldungen aus, die zwar einen Fehler oder ein Problem darstellen, aber dennoch nicht zum Abbruch des Programms führen würden. Meldungen der Klasse ERROR geben Fehler oder Probleme an, die definitiv zum Abbruch oder vorzeitigen Beenden des Programmes führen. Dies muss allerdings nicht immer auf Grund

einer Exception sein. In der Konsole wird kein Zeitstempel mit angegeben, sondern lediglich um welche Klasse von Meldung es sich handelt und die Meldung selbst. In dem folgenden Auszug ist eine Reihe von Logging-Ereignissen von VJCMS zu sehen. Erkannte Objekte werden ebenfalls in das Log mit eingetragen.

```
[Thu Feb 22 12:21:35 2018][INFO]      [main] starting program
[Thu Feb 22 12:21:35 2018][INFO]      [main] streaming to network
[Thu Feb 22 12:21:35 2018][INFO]      [main] DEBUG 1
[Thu Feb 22 12:21:35 2018][INFO]      [main] MODE 0
[Thu Feb 22 12:21:35 2018][INFO]      [main] VERBOSE 0
[Thu Feb 22 12:21:35 2018][INFO]      [detect_and_track] Begin detect_and_track
[Thu Feb 22 12:21:35 2018][INFO]      [detect_and_track] starting Thread for Viola
[Thu Feb 22 12:21:35 2018][INFO]      [detect_and_track] opening input stream
[Thu Feb 22 12:21:35 2018][OBJECT]    [Cone](351,102) (549,240)
[Thu Feb 22 12:21:35 2018][OBJECT]    [Cone](183,141) (252,180)
```

3.3.5 VJ_Exception

Das Modul Exception enthält lediglich zwei Ausnahmeklassen. Zum einen „CameraNotConnected“ und zum anderen „InvalidMode“.

3.3.6 monitor.py

Monitor.py dient auschließlich der Anzeige des Streams, welcher durch VJCMS gesendet wird, falls eine Ausgabe über das Netzwerk per Parameter angegeben wurde. Dazu muss sowohl im Client als auch im Server die entsprechende IP-Adresse und Port gesetzt werden.

3.4 Funktionalität

Im Folgenden werde die einzelnen möglichen Parameter für den Funktionsaufruf aufgelistet und erklärt.

Option	Wertebereich	Standard	Erklärung
–debug, -d	0,1	1	0 = Als Input wird die PiCamera verwendet, 1 = Ein Video, welches im Ordner videos liegt, wird verwendet
–mode, -m	0,1,2	0	0 = detect_and_track, 1 = detect, 2 = optimized
–network, -n	0,1,2	0	0 = Ausgabe über Monitor, 1 = Ausgabe via Stream, 2 = keine Ausgabe
–ip,-ip	string	local	IP-Adresse des Servers
–port, -port	int	5001	Port des Servers
–verbose, -v	0,1,2	2	0 = INFO, WARNING, ERROR, 1 = WARNING, ERROR, 2 = ERROR
–track, -t	int	50	Anzahl der Frames, für die ein Objekt verfolgt werden soll
–skip, -s	int	5	Anzahl der Frames, in denen der Frame ohne bearbeitung angezeigt wird
–object, -o	string	cone	Liste der Objekte, die erkannt werden sollen
–video	Boolean	False	Wenn diese Option genutzt wird, wir der aktuelle Outputstream als Video auf dem Pi gespeichert

Tabelle 3.1: Liste der Optionen für VJCMS

VJCMS bietet eine Reihe von Funktionen, um Objekte im Videostream erkenen zu können. Es gibt insgesammt 3 Modi, die in verschiedenen Kombinationen mit anderen Paramtern genutzt werden können. Neben diesen Kernfunktionen bietet dieses Skript

noch eine Reihe weitere hilfreiche Features, die vor allem für Debug-Zwecke sinnvoll sein können.

3.4.1 detect_and_track

Der Modus detect_and_track entspricht im Grunde der Funktionalität aus dem ursprünglichen Skript, welches anfangs schon erwähnt wurde. Hinzu kamen nur Elemente, wie das Logging und eine entsprechende Fehlerbehandlung. Dabei wird alle n Frames der Viola-Jones Algorithmus angewendet und die gefunden Objekte für eine gewisse Anzahl an Frames verfolgt. Die Position jedes gefundenen Objektes wird im folgenden Format zurückgegeben.

Listing 3.1: Aufbau des Ergebnis-Array

```
....
{x1, y1, x2, y2, label},
{x1, y1, x2, y2, label}
....
```

Die ersten beiden Koordinaten beschreiben dabei die linke obere Ecke der Bounding-Box und das zweite Koordinaten-Paar die rechte untere Ecke. Das Label gibt an um welches Objekt es sich handelt.

3.4.2 detect

Im Gegensatz zu dem gerade eben erklärten Modus, werden hier Objekte lediglich erkannt und deren Position in dem bekannten Format ausgegeben. Es wird sonst keine weitere Bearbeitung des Frames vorgenommen. Es wird hierbei jeder einzelne Frame untersucht.

3.4.3 optimized

In diesem Modus implementieren wir einige Optimierungen, die dazu dienen sollen, die Performance des Algorithmus auf dem Pi zu verbessern.
Wir nehmen an, dass in einem normalen Verkehrszenario alle für uns relevanten Schilder entweder links oder rechts von unserem Auto am Straßenrand stehen. Dies hätte zur Folge, dass sich in der Mitte unseres Bildes keine relevanten Informationen befinden. Selbst wenn ein Schild aufgrund der Distanz zu unserem Auto, theoretisch dadurch in der Mitte unseres Bilder erscheinen würde, wäre das Schild zu weit weg, als das es zur Erkennung relevant wäre. Demzufolge könnte das mittlere Drittel des Bilder vernachlässigt werden. In diesem Modus wird lediglich das linke und rechte Drittel eines jeden Frames untersucht, wodurch ein Teil der Berechnungen eingespart wird.

3.4.4 Weitere Features

Videostream

Wie eingangs erwähnt besteht die Möglichkeit, einen Videostream zu erzeugen, der die Bounding-Boxes anzeigt. Dazu gibt es verschiedene Ausgabemöglichkeiten. Zum einen direkt auf einem über den HDMI Port angeschlossenen Bildschirm. Zum anderen kann dieses Signal aber auch über TCP an einen entsprechenden Server geschickt werden. Dies ist mit monitor.py möglich. Der Parameter, der dies steuert ist `--network`.

Video

Neben der Ausgabe des Videostreames gibt es auch die Möglichkeit diesen gleich auf dem Pi als Videodatei (.avi) zu speichern. Über die Option `--video` kann dieses Feature aktiviert werden.

Debug

Eien weitere sehr hilfreiche Option, besonders während der Entwicklung, ist der Debug Modus. In diesem Modus wird keine Kamera als Videoquelle verwendet, sondern ein

Testvideo, dass sich in dem Ordner videos befindet. Dieses kann natürlich nach belieben ausgetauscht werden. Das Feature ermöglicht es einem Nutzer an verschiedenen Rechnern und Geräten zu arbeiten, ohnen einen Pi zum testen parat haben zu müssen.

Kapitel 4

Auswertung

In einem Kurztest haben wir untersucht, wie gut die Performance des Pi's mit unserem Programm ist. Dazu haben wir eine Teststrecke aufgebaut und sind diese sowohl mit dem Pi als auch mit einem Laptopt mit angeschlossener Webcam abgefahren. Zum Vergleich die Spezifikationen der beiden Geräte:

Lenovo Yoga 13	Raspberry Pi 3 Model b
Intel i5 2 x 1,8 GHz	Broadcom BCM2837 4 x 1,2 GHz
8 GB RAM	1 GB RAM

In dem Test wurde der Modus 1 verwendet. D.h. es wird jedes Frame nach Objekten untersucht. Ebenfalls wurde in dem Test nach allen Objekten (Cones, Vorfahrt, Stopp Schild, Vorfahrt gewähren) gesucht. In diesem Test konnte der Pi maximal 6 FPS und der Laptop 30 FPS erzielen. Das gleiche Ergebnis konnte unter Verwendung des Modus 2 ereicht werden. Der Grund für die relativ schlechte Performance auf dem Pi liegt vermutlich am mangelnden Arbeitsspeicher. Ein weiterer Aspekt ist, dass Python-Skripte aufgrund ihres Parsings nicht performant operieren.

Kapitel 5

Fazit

Nach lang anhaltenden Schwierigkeiten mit dem Traingsprozess, der auf lückenhafte Dokumentation sowie nicht vorhandene Erläuterungen zur Benutzung der durch OpenCV bereitgestellten Tools zurückzuführen ist, gelang es uns für die Verkehrsschilder solide Klassifizierer zu erstellen. Die Dauer des Trainingsprozesses nahm von Objekt zu Objekt stark ab. Dauerte es noch mehrere Monate den ersten guten Klassifizierer für Pylonen zu erstellen, gelang es uns den Klassifizierer für das Vorfahrt gewähren Schild binnen weniger Stunden, plus der Trainingszeit für den Algorithmus, zu erstellen.
Die trainierten Klassifizierer erkennen die Verkehrsschildobjekte mit einer sehr hohen Trefferrate aus verschiedenen Sichtwinkeln. Einziger Negativpunkt hierbei ist die auf die Kamera einfallende Lichtintensität. Ist diese zu hoch, stößt der Klassifizierer schnell an seine Grenzen. Dunkel belichteter Videoinput scheint für unsere Klassifizierer kein Problem darzustellen. Ein Grund weshalb die Verkehrsschilder gut erkannt werden können ist ihre Form. Die genutzten Weak Classifier passen gut auf die Kanten, welche bei allen Drei von uns angelernten Verkehrsschildern vorkommen. Die 45° Kanten werden durch die Erweiterung des Modus beim Trainingsprozess als weiteres Indiz zum Erkennen genutztm weshalb eine sehr hohe Trefferrate mit wenigen Falsepositives erreicht werden konnte.
Einzig die Pylonen stellen eine Besonderheit dar. Aufgrund ihrer ungewöhnlichen Form, die durch die Kanten im 70° Winkel zustande kommt, sowie ihrer Art der Verzerrung

sollten sie von der Seite betrachtet werden, konnte kein eindeutiger Klassifizierer von uns erstellt werden. Das Muster welches vorwiegend durch den Viola und Jones Algorithmus erkannt wird, ist hierbei durch die Farbwechsel von rot zu weiß und andersherum, beziehungsweise in Graustufen auf Ebene des Algorithmus, geprägt. Diese Art Muster kann je nach Belichtungsverhältnis häufig und leicht in anderen Oberflächen gefunden werden, weshalb eine sichere Nichterkennung von Falsepositives nicht gewährleistet kann. Ein weiterer Grund für potenzielle Falsepositives ist, dass die Weak Classifier nicht in ihrer vollen Fähigkeit zum tragen kommen. Wie bereits oben erwähnt besitzen Pylonen eine steile Kante von circa 70°. Schwache Klassifizierer sind jedoch für 0°, 45° und 90° angepasst, weshalb diese im Lernprozess kaum zum Tragen kommen. Die Erkennung von Pylonen beruht somit alleinig auf dem rot-weiß Muster, welches das Charakteristikum der Eindeutigkeit gegenüber anderen Objekten im Alltag leider nicht vorweist.

Zusammenfassend kann gesagt werden, dass der Algorithmus nach einiger Einarbeitungszeit eine solide Basis zur Verkehrsschildererkennung darstellt und für Pylonen nur bedingt geeignet ist. Um die Erkennung von Pylonen sinnvoll zu nutzen wird vorgeschlagen weitere logische Bedingungen einzufügen, sodass ein sinnvolles Geleit durch die Pylonenpaare gewährleistet werden kann. Die Umsetzung auf dem Raspberry ist als erster Versuch und somit Übergangslösung anzusehen, da die gemessene Performance auf diesem Gerät nicht überzeugt. Um diese verbessern zu können gibt es zweierlei Ansätze. Zum einen besteht die Möglichkeit eine Hochsprache für die Implementierung zu verwenden, die ein besseres Resourcenmanagement erlaubt und insgesamt performanter ausgeführt wird. Zum anderen ist eine leistungsfähigere Hardware einzusetzen. Empfehlenswert wäre an dieser Stelle die Implemenierung einzelner Schritte bzw. Berechnungen auf dem Z-Board, um besonders die rechenintensive Erkennung von Objekten zu beschleunigen. Damit sollte dann eine deutlich bessere Performance und somit ein höhere FPS zu erreichen sein.

Literaturverzeichnis

[OPe17] OPENCV: *Face Detection using Haar Cascades*. https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html. Version: August 2017

[Pat15] PATELL, Rahull: *Real time face detection using Viola-Jones and Camshift in Python*. https://rahullpatell.wordpress.com/2015/04/21/real-time-face-detection-using-viola-jones-and-camshift-in-python-i/. Version: April 2015