Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Frederic Alexander Hermann Adler

Kamerabasierter Algorithmus zur Zählung von Fahrradfahrern im Straßenverkehr unter Verwendung von HOG-Deskriptoren

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Frederic Alexander Hermann Adler

Kamerabasierter Algorithmus zur Zählung von Fahrradfahrern im Straßenverkehr unter Verwendung von HOG-Deskriptoren

Masterthesis eingereicht im Rahmen der Masterprüfung
im Studiengang Informations- und Kommunikationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Hans Peter Kölzer
Zweitgutachter : Prof. Dr.-Ing. Lutz Leutelt

Abgegeben am 3. Mai 2017

Frederic Alexander Hermann Adler

Thema der Masterthesis

Kamerabasierter Algorithmus zur Zählung von Fahrradfahrern im Straßenverkehr unter Verwendung von HOG-Deskriptoren

Stichworte

Computer Vision, Histogram of Oriented Gradients, Non-Maximum Suppression, Maschinelles Lernen, Support Vector Machine, Objektverfolgung, Kalman-Filter, OpenCV

Kurzzusammenfassung

Diese Masterthesis beschreibt die Entwicklung und Verifikation eines Algorithmus für die Zählung von Fahrradfahrern im Straßenverkehr. Der entwickelte Algorithmus berechnet dabei zunächst die HOG-Deskriptoren innerhalb einer Videodatei bzw. eines Videostreams und prüft mittels einer Support Vector Machine auf eine erfolgreiche Objekterkennung. Die Objektverfolgung durch Einsatz eines Kalman-Filters ermöglicht abschließend die eigentliche Zählung der Radfahrer.

Frederic Alexander Hermann Adler

Title of the paper

Camera based algorithm for counting cyclists in road traffic environment using HOG descriptors

Keywords

Computer Vision, Histogram of Oriented Gradients, Non-Maximum Suppression, Machine Learning, Support Vector Machine, Object Tracking, Kalman filter, OpenCV

Abstract

This master thesis describes the development and verification of an algorithm for counting cyclists in typical traffic situations. The developed algorithm initially calculates the HOG descriptors and uses a support vector machine for the following object detection. Finally an object tracking using Kalman filters enables the actual counting.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

Diese Masterthesis beschreibt die Entwicklung und Untersuchung eines Algorithmus zur Zählung von Radfahrern im Straßenverkehr. In diesem ersten Kapitel wird zunächst die Motivation der Arbeit erläutert. Dabei wird ein Vergleich zu einem Projekt der Stadt Hamburg gezogen, in welchem sogenannte Zählsäulen installiert wurden, um passierende Radfahrer zu zählen und die Ergebnisse zur Motivation der Verkehrsteilnehmer zu visualisieren. Abschnitt 1.2 beschreibt abschließend kurz den Inhalt der jeweiligen Kapitel und bietet so einen Gesamtüberblick über die Arbeit.

1.1. Motivation

Im Rahmen der gegenwärtigen Diskussion um ein nachhaltiges Leben insgesamt nimmt auch das Thema Radfahren in der Stadt einen hohen Stellenwert ein. An vorderster Stelle für einen gesteigerten Radverkehr steht dabei die Entlastung des Kfz-Verkehrs und die damit einhergehende Reduzierung von Lärmbelästigung und Luftverschmutzung. Dabei versteht sich die Stadt Hamburg zunehmend als Fahrradstadt und bietet eine großes Informationsspektrum zum Thema Fahrradfahren über das Portal der Behörde für Inneres und Sport (BIS) an.[1] Ein zufriedenstellender Ausbau ist jedoch noch Ziel der gegenwärtigen Politik und so kann der aktuelle Stand hin zur Fahrradstadt dem Fortschrittsbericht [BWV15b] der Behörde für Wirtschaft, Verkehr und Innovation (BWVI) entnommen werden.

Im Gegensatz zu Fußgängern nutzen Radfahrer in vielen Szenarien unmittelbar die gleiche Infrastruktur wie der Kfz-Verkehr. Trotz zunehmender Anzahl von Ausbauten der Fahrradwege steigt hier bei ebenfalls zunehmender Anzahl von Radfahrern auch das Potential für Gefahrensituation. Die Tabelle 1.1 zeigt für die Jahre 2014, 2015 und 2016 die Unfallstatistik für Unfälle mit Radfahrerbeteiligung in Hamburg. Es fällt auf, dass die Zahlen trotz aller Anstrengungen hinsichtlich der Verbesserung der Fahrradinfrastruktur nicht signifikant rückläufig sind und daher Bedarf an einer Steigerung der Verkehrssicherheit besteht. Laut einem Pressebericht der BWVI ist die sogenannte Zählsäule dabei ein wesentliches Werkzeug [BWV15a] zur Analyse des Radfahreraufkommens in der Stadt Hamburg. Abbildung 1.1 zeigt eine solche Säule, hier am Standort Gurlitt-Insel an der Außenalster.

[1] Für nähere Information sei auf das Portal unter `http://www.hamburg.de/fahrrad/` verwiesen.

Jahr	2014	2015	2015 zu 2014	2016	2016 zu 2015
Verunglückte Radfahrer	2420	2362	-2,4 %	2412	2,1 %
Leichtverletzte	2185	2126	-2,7 %	2192	3,1 %
Schwerverletzte	224	234	4,5 %	217	-7,3 %
Getötete	11	2	-81,8 %	3	50 %

Tabelle 1.1.: Verkehrsunfälle mit Radfahrern 2014 bis 2016 [BIS16] und [BIS17]

Der Hamburger Senat sieht in den Zählsäulen nicht nur ein wertvolles Analysewerkzeug, sondern auch eine Informationsquelle für die Bürger, durch die ein „tatsächliches Abbild des täglichen Radverkehrsgeschehens“ visualisiert wird [NDR15a]. Bei einem Anschaffungspreis von rund 32.000 € besteht jedoch große Kritik, u.a. seitens des Bundes der Steuerzahler [NDR15b]. Zudem wird die Genauigkeit der Objekterfassung kritisiert. [NDR15c], sowie [St.14] zufolge werden demnach Fahrradfahrer doppelt gezählt und das Ergebnis so verfälscht. Einige Objekte, wie z.B. Tretroller, werden hingegen gar nicht berücksichtigt. Auf den ersten Blick scheint die Zählsäule also ein gutes Hilfsmittel bei der Umsetzung des Projekts Fahrradstadt Hamburg zu sein, welcher auf den zweiten Blick jedoch an der Umsetzung und am Preis zu scheitern droht.

Abbildung 1.1.: Zählsäule der Stadt Hamburg am Standort Gurlitt-Insel an der Außenalster

Aktuelle Systeme der Bildverarbeitung werden, bei steigender Rechenleistung, immer zugänglicher und kostengünstiger. Es besteht daher der Wunsch nach einem kompakten kamerabasierten System, welches die Zählsäule potentiell ersetzen kann. Abbildung 1.2 zeigt dabei eine schematische Darstellung des geplanten Systems. Dieses soll grundsätzlich in der Lage sein, autark installiert zu werden und über einen effizienten Algorithmus eine zuverlässige Zählung der passierenden Radfahrer zu gewährleisten. Dabei soll das System kontinuierlich Bilddaten einer Kamera aufnehmen und zunächst ggfs. vorverarbeiten. Anschließend soll das zu detektierende Objekt, hier also der Radfahrer, über eine angepasste Merkmalsextraktion und eine robuste Klassifizierung erkannt und die Zählung durch eine Objektverfolgung realisiert werden. Im ersten Schritt kann die benötigte Recheneinheit durch einen Laptop oder gar einen Desktop-PC gestellt werden. Die Portierung auf einen Einplatinencomputer realisiert im Anschluss ein kompaktes und preisgünstiges System, welches grundsätzlich autark einsetzbar ist. Für ein speziell auf die gestellten Anforderungen angepasstes System bietet sich zudem die anschließende Portierung des Algorithmus auf ein Embedded System[2] an.

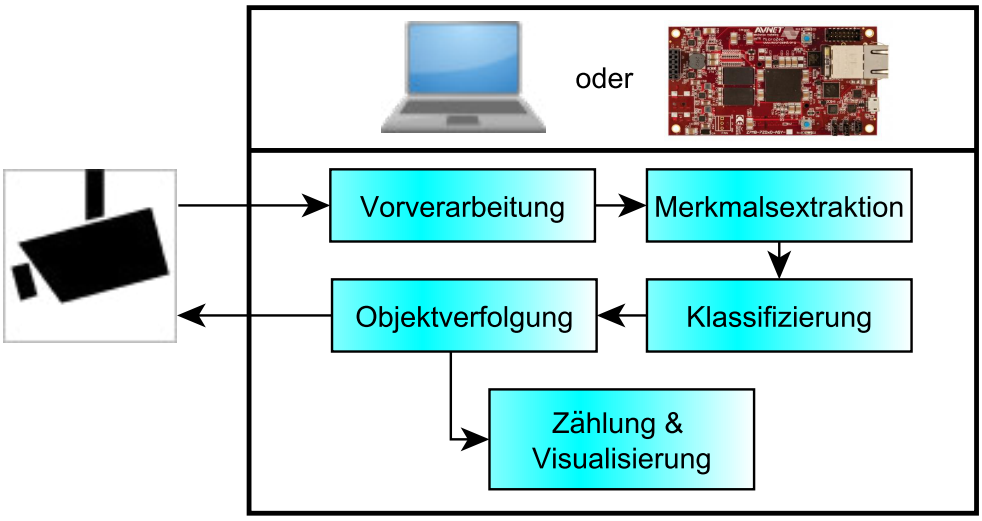


Abbildung 1.2.: Schematische Darstellung des Systems zur Radfahrerzählung

[2] Als Embedded System wird hier das Avnet MicroZed Board [Avn15] herangezogen. Vgl. auch Abschnitt 6.4

1.2. Kapitelübersicht

Nachdem dieses Kapitel die Motivation und eine schematische Darstellung für das zu entwickelnde System aufgezeigt hat, beschreibt *das zweite Kapitel* den Prozess der Analyse bis zum Konzept der Arbeit. Zunächst wird dabei eine Anforderungsanalyse aufgestellt und der aktuelle Stand der Forschung aufgezeigt. Abschließend wird das Grobdesign bzw. Konzept des Algorithmus dargestellt.

Das dritte Kapitel erläutert die für die Umsetzung der Arbeit relevanten Grundlagen der Softwarekomponenten. Das Kapitel behandelt dabei in Reihenfolge des realisierten Algorithmus die Themenbereiche Merkmalsextraktion, Klassifizierung und Objektverfolgung.

Das vierte Kapitel gibt einen Überblick über den im Rahmen der Arbeit erstellten Objektdatensatz. Dabei werden die Rahmenbedingungen der Videosequenzen erläutert und die Auslegung der Trainingssamples vorgestellt.

Das fünfte Kapitel geht auf die Umsetzung des Konzepts ein und zeigt die Parametrisierung und Struktur des realisierten Algorithmus. Neben der Struktur des Deskriptors wird hier der Trainings- und und Entscheidungsprozess der Klassifikation dargestellt. Anschließend wird der Ablauf des Multi-Object-Trackings erläutert.

Im sechsten Kapitel werden die Untersuchungsergebnisse in Bezug auf Erkennungsrate und Echtzeitfähigkeit des Algorithmus vorgestellt und unter Berücksichtigung der Anforderungen bewertet. Abschließend wird zudem ein Lösungsansatz für die Portierung auf ein dediziertes Embedded System vorgestellt.

Das letzte Kapitel fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick für mögliche Verbesserungen des Algorithmus und zukünftige Entwicklungsschritte.

2. Analyse und Anforderungen

Nachdem in der Einleitung die grundlegende Motivation aufgezeigt wurde, analysiert dieses Kapitel die Anforderungen und definiert die Ziele der Arbeit. In der Projektplanung ist die Analysearbeit ein wichtiger Schritt für eine kostengerechte Lösung, dessen Qualität auch nur mittels der anfänglich aufgestellten Spezifikation bewertet werden kann. Im Abschnitt 2.2 wird anschließend zur Zieldefinition der aktuelle Stand der Forschung aufgezeigt. Der Vergleich derzeit eingesetzter Lösungsansätze führt dann zum Abschnitt 2.3, in dem das Konzept des entwickelten Systems bzw. des Algorithmus erläutert wird.

2.1. Anforderungsanalyse

Die wesentliche Hauptanforderung des Systems ergibt sich zunächst aus dem Vergleich mit der Funktionsweise der Zählsäule, welche bereits in der Einleitung erwähnt wurde (vgl. Abbildung 1.1). Diese analysiert über fest im Boden installierte Induktionsschleifen den „elektromagnetischen Fingerabdruck“ der Fahrräder [Eco17]. Ausschlaggebend für die Restriktionen des Zählsystem ist dabei vor allem der stationäre Aufbau und der auf den Radweg beschränkte Detektionsraum. Für das zu entwickelnde System gilt also, dass die Kamera einen festen Ausschnitt des Radwegs aufnehmen soll. Dadurch bleiben die Größenrelationen der Radfahrer größtenteils invariabel, solange Sie auf dem Radweg bleiben. Demnach können dann alle Radfahrer gezählt werden, welche sich korrekterweise auf dem Radweg bewegen. Diese Anforderung deckt sich mit der Zählsäule, welche durch den Einsatz von Induktionsschleifen ebenfalls nur Radfahrer zählen kann, welche sich über die Induktionsschleifen bewegen. Abbildung 2.1 zeigt das Funktionsschema der Zählsäule und die Installation zweier Induktionsschleifen.

(a) Funktionsschema der Zählsäule (hier mit einer Schleife)

(b) Zwei Induktionsschleifen in der Installationsphase

Abbildung 2.1.: Funktionsschema und Induktionsschleifen der Zählsäule Eco-Counter ZELT Urban [Eco17]

Die stationäre Installation der Kamera gibt letztendlich auch den Blickwinkel vor und stellt somit in sich eine weitere Anforderung in Bezug auf den Spielraum der Erkennung. Im zugrunde liegenden Fall sollen nur diejenigen Radfahrer erfasst werden, welche sich orthogonal zur Blickrichtung über den Radweg bewegen. Für die Objektdetektion ergibt sich daraus eine Fallunterscheidung mit zwei Fällen, nämlich entweder Radfahrer von links nach rechts fahrend oder von rechts nach links fahrend zu erkennen.

Zunächst soll außerdem nur der Einsatzfall bei Tag betrachtet werden. Wie später in Abschnitt 3.1 gezeigt, ist der gewählte Algorithmus zur Detektion weitestgehend robust gegenüber Kontrastschwankungen. Damit bleibt eine mögliche Nachtaufnahme vor allem eine Aufgabe in Bezug auf die Hardware, genauer gesagt die Beleuchtung durch Infrarot Leuchtquellen und soll hier nicht näher behandelt werden.

Ein weiteres grundlegendes Ziel der Entwicklung ist die Echtzeitfähigkeit des Algorithmus. Spezifiziert wird diese durch die Bildrate (engl.: Framerate) von 30 Frames per Second (FPS). Diese ist eine typische Rate für kommerziell, wie auch industriell erhältliche Kameras. Die Framerate ist außerdem groß genug um eine flüssige Darstellung für die menschliche Wahrnehmung zu gewährleisten. Aus der Vorgabe eine Bildrate von 30 FPS zu gewährleisten, kann dann die Rechenzeit für einen Iterationsschritt des Algorithmus bestimmt werden (vgl. Gleichung 2.1). Dabei meint T_{frame} die maximal zu Verfügung stehende Rechenzeit zwischen zwei Frames.

$$T_{\text{frame}} = \frac{1}{30}\text{s} \approx 33,3\,\text{ms} \tag{2.1}$$

Insbesondere Algorithmen zur Klassifizierung mit vorangehender auf Pixeloperationen beruhender Merkmalsextraktion fallen potentiell sehr rechenintensiv aus. Daher soll der Detektionsraum auf einen statischen Teilbereich des Kamerabilds begrenzt werden, welcher als Region of Interest (ROI) bezeichnet wird. In dem zugrunde liegenden Fall ist das der sichtbare Teil des Radwegs, also nur der Bildteil auf dem sich die Radfahrer bewegen.

Neben der Entwicklung des Algorithmus auf einem leistungsstarken PC, soll außerdem die spätere Portierung auf ein kompaktes und autark arbeitenden Systems nicht außer Acht gelassen werden. Die Auswahl einer geeigneten Hardware wird daher in Abschnitt 2.3 näher behandelt.

Abschließend seien die Hauptanforderungen an die Arbeit an dieser Stelle noch einmal zusammengefasst:

- Stationäres System mit Blickrichtung orthogonal zum Radweg
- Größenrelation der Fahrradfahrer größtenteils invariabel, d.h. Größe (Breite x Höhe) der Samples kann statisch bleiben
- Es sollen alle Radfahrer erkannt werden, welche sich auf dem Radweg und von links nach rechts oder rechts nach links innerhalb des Detektionsraums bewegen
- Der Detektionsraum soll auf eine ROI, d.h. den sichtbaren Teil des Radwegs, begrenzt werden
- Die Echtzeitfähigkeit im Rahmen der zugrunde liegenden Hardware (30 FPS) soll gewährleistet werden
- Das System soll kompakt sein und grundsätzlich autark arbeiten können

2.2. State of the Art

Das Ziel durch Objekterkennung und -verfolgung automatisierte Systeme zu entwickeln, welche dem Menschen Arbeit abnehmen oder Unterstützung bieten, besteht seit jeher. Durch die steigende Vernetzung von komplexen verteilten Systemen steht das Forschungsfeld Maschinelles Lernen (engl.: Machine Learning) im Mittelpunkt vieler aktueller Forschungen. Insbesondere der Automotive Sektor weist zudem ein umfangreiches Maß an Arbeiten zum Thema Objekterkennung auf. Häufig sind dabei Klassifikatoren nutzbar, welche z.B. Merkmale basierend auf dem Farbspektrum des Objekts [Tör16] nutzen oder Objekte durch Detektion der Bewegung anhand von Differenzbilder lokalisieren. Im zugrunde liegenden Fall der Arbeit kann jedoch kaum auf das Differenzbildverfahren zurückgegriffen werden, da das Hintergrundrauschen durch Straßen- und Personenverkehr einen zu großen Störfaktor darstellt. Auch Farbinformationen eignen sich aufgrund der sich stark unterscheidenden Radfahrer nur bedingt als Basisinformation für die Klassifikation.

Ein weiterer Lösungsansatz an dieser Stelle ist eine Modellbildung des Objekts. In [Sch05] wurde solch eine Modellbildung u.A. für Radfahrer erstellt und konnte für das Testvideo gute Ergebnisse liefern. Das Modell wird hier durch die Erkennung von Reifen und Kopf und deren Relationen zueinander definiert bzw. klassifiziert. Dabei besteht eine große Herausforderung darin, das Modell exakt genau zu parametrisieren, um eine allgemein gültige Detektion zu gewährleisten. Der Lösungsansatz einer Modellbildung führt letztendlich über die Definition des Objekts durch den Menschen. Die für den Menschen charakteristischen Merkmale eignen sich dabei möglicherweise jedoch nur bedingt für einen in Bezug auf die Rechenzeit effizienten oder robusten Algorithmus. Im folgenden Abschnitt 2.3 wird näher auf eine mögliche Modellbildung im Rahmen dieser Arbeit eingegangen.

Ein gutes Ergebnis für die Erkennung von Personen bietet die Arbeit [Dal06]. Diese implementiert erstmals einen Algorithmus, welcher eine Merkmalsextraktion mit HOG-Deskriptoren durchführt. Diese bilden ein Histogramm örtlich begrenzter Gradienten innerhalb des Samples ab und glätten letztendlich durch die Berechnung über mehrere Pixel die Werte des Merkmalsvektors. Zudem wird die Dimension einer zweidimensionalen Matrix, also die mathematische Abbildung des Videobildes auf eine 2D-Matrix, auf einen eindimensionalen Merkmalsvektor projiziert. Über ein Sliding-Window-Verfahren wird das „Testfenster“ (engl.: Detection Window) über das Bild geschoben und der HOG-Deskriptor berechnet. Anschließend erfolgt die Klassifikation durch eine Support Vector Machine (SVM).

Bei der Recherche nach dem aktuellen Stand der Forschung fällt auf, dass sich die Arbeiten überwiegend mit der Erkennung von Personen, insbesondere unter dem Aspekt der Sicherheit im Straßenverkehr befassen. Viele Arbeiten greifen auf den Ansatz von Dalaal zurück und erweitern diesen. In [ZAYC06] wird die Klassifikation beispielsweise durch HOG-Deskriptoren mit einem AdaBoost-Verfahren kombiniert. AdaBoost (Kurzform für „Adaptive

Boosting“) beschleunigt die Rechenzeit mittels einer Reduktion der Dimensionalität durch Vorauswahl geeigneter Merkmale im Lernprozess. Arbeiten die sich explizit mit der Erkennung von Radfahrern beschäftigen sind jedoch rar, obwohl sich Radfahrer häufig dieselbe Infrastruktur mit Kraftfahrzeugen im Straßenverkehr teilen. Dabei sind sie kaum besser geschützt als Fußgänger. Es besteht also insbesondere für den Bereich automatisiertes Fahren ein gesteigertes Interesse Radfahrer zuverlässig zu erkennen und ggfs. bei einer Gefahrensituation zu handeln. Die zum Zeitpunkt dieser Arbeit aktuellste Veröffentlichung ist [TL15] aus dem Jahr 2015. Im Rahmen des Forschungsprojekts KITTI des Karlsruhe Institute of Technology (KIT) und dem Toyota Technological Institute at Chicago (TTIC) wurde hier eine echtzeitfähige Erkennung von Radfahrern angestrebt, indem erst sogenannte Decision Forests[1] eine Vorauswahl der HOG-Merkmale durchführen und danach eine SVM zum Einsatz kommt, welche durch die Vorauswahl weniger große Merkmalsvektoren verarbeiten muss.

In [ZM10] werden Local Binary Patterns (LBP)- und HOG-Deskriptoren für die Klassifikation benutzt. Durch eine anschließende Principal Component Analysis (PCA) wird die Dimensionalität des Merkmalsraums reduziert, indem dieser mit sogenannten Hauptkomponenten dargestellt wird, welche ausschließlich relevante Merkmale berücksichtigen und irrelevante ignorieren. Dadurch kann der Merkmalsvektor für die Berechnung durch die SVM reduziert werden.

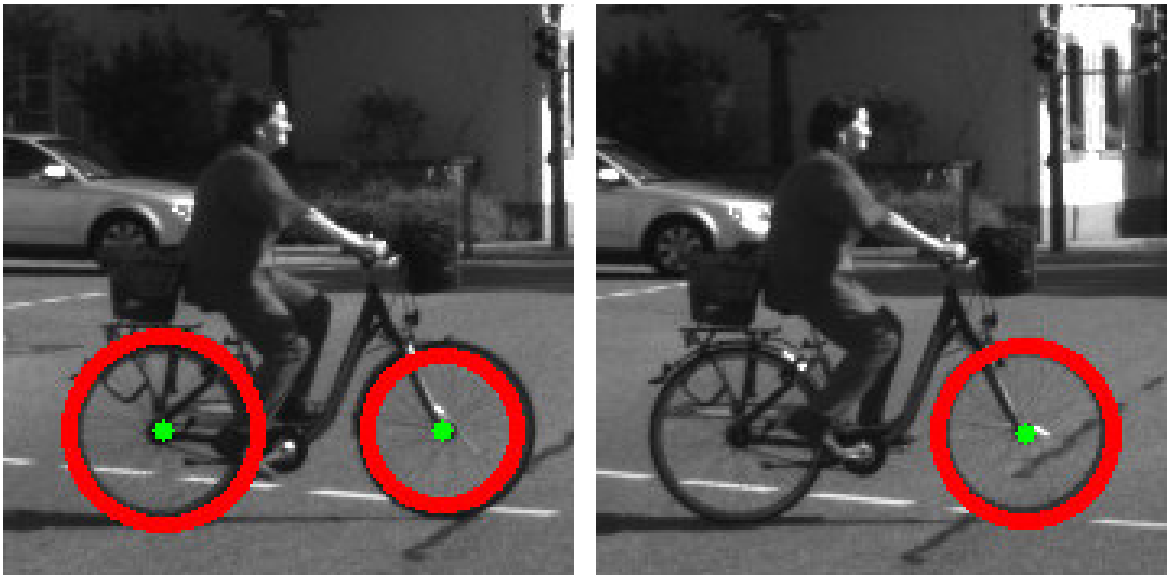
2.3. Konzept

Zu Beginn dieser Arbeit wurde zunächst ein Lösungsansatz verfolgt, der eine Modellbildung des Objekt, d.h. eines Radfahrers voraussetzt. Für die Detektion eines Radfahrers wurden zunächst zwei Kreise mittels Hough-Transformation [Bor03] gesucht und deren Abstand verglichen. Es zeigte sich jedoch, dass der Algorithmus bereits bei guten Sichtverhältnissen und angepassten Parametern, hier der Durchmesser eines Kreisausschnitts, eine begrenzte Robustheit aufweist. Abbildung 2.2 zeigt zwei Frames eines Videos mit guten Sichtverhältnissen, bei denen die Erkennung bei gleichen Parametereinstellungen zunächst gelingt und im folgenden Frame misslingt.

Das Konzept der Arbeit [TL15], welches bereits in Abschnitt 2.2 vorgestellt wurde, erzielte insbesondere für die Detektion von Radfahrern gute Ergebnisse. Es wird daher ein ähnlicher Lösungsansatz verfolgt, an dessen erster Stelle dabei die Berechnung des HOG-Deskriptors als robustes Merkmal für das Objekt steht. Unter den in Abschnitt 2.1 gegeben Anforderungen kann zudem eine Vereinfachung der Detektionsstruktur vorgenommen werden. Wie

[1] Bei den Decision Forests handelt es sich um eine Hintereinanderschaltung von Entscheidungsbäumen. Die Entscheidung auf ein Objekt erfolgt über die Abfrage eines bestimmten Attributs an jedem Knoten.

erwähnt bleiben für stationäre Aufbauten die Größenrelationen der Objekte im legalen Detektionsraum größtenteils invariabel. Außerdem soll nur ein Ausschnitt der Videoaufnahme analysiert werden. Während in einem Szenario für automatisiertes Fahrens auf jede Bewegungsrichtung eingegangen werden muss, sind hier nur die orthogonal kreuzenden Radfahrer zu berücksichtigen. Die in [TL15] aufgestellten acht Blickwinkel können hier also auf zwei reduziert werden. Der Einsatz einer ROI und die Reduktion auf zwei Richtungen gewährleisten dann eine erhebliche Einsparung von Rechenzeit.



(a) Frame F_x mit 2 korrekten Treffern (b) Frame F_{x+1} mit nur einem Treffer

Abbildung 2.2.: Zwei aufeinander folgende Frames für die Suche nach Kreisen über eine Hough-Transformation (Bild aus dem Datensatz von [GLSU13])

Wie bereits die Arbeit [Dal06] zeigt, eignet sich für zwei Klassen mit einem eindimensionalen Klassifikator eine SVM. Auch [TL15] verwendet eine SVM, wenngleich die hohe Trainingszeit bemerkt wird. Durch die Begrenzung auf einen statischen Bildausschnitt, die unter Berücksichtigung der Anforderungen angenommen wurde, kann allerdings auch die Trainingsphase erheblich verkürzt werden. Für den Kernel der SVM wird ein linearer gewählt (siehe auch Abschnitt 3.2), welcher im Vergleich zu anderen Kernmethoden der schnellste ist. Hinzu kommt, dass die Anbindung innerhalb der Softwarebibliothek OpenCV (näheres auf Seite 20 dieses Abschnitts) ausreichend dokumentiert ist und bereits vorangegangene Arbeiten die Portierung der Software auf ein Einplatinensystem realisiert haben (vgl. [Tör16]).

Unter den gegebenen Anforderungen kann postuliert werden, dass sich die Radfahrer mit einer nahezu linearen Geschwindigkeit innerhalb der ROI bewegen. Die Objektverfolgung kann durch diese Rahmenbedingungen hinreichend genau durch Einsatz eines Kalman-Filters realisiert werden. Die Arbeit von [Sch05] bestätigt diese Annahme im Vergleich zum Einsatz von Partikelfiltern für die Objektverfolgung. Der Partikelfilter eignet sich vor allem bei nicht linearen Bewegungen wohingegen der Kalman-Filter in der Arbeit sogar bessere Ergebnisse für das gezeichnete Szenario einer linearen Bewegung dokumentiert (vgl. [Sch05, S. 81]).

Abschließend zeigt Abbildung 2.3 schematisch das vorgestellte Konzept des Algorithmus in Form eines vereinfachten Ablaufdiagramms.

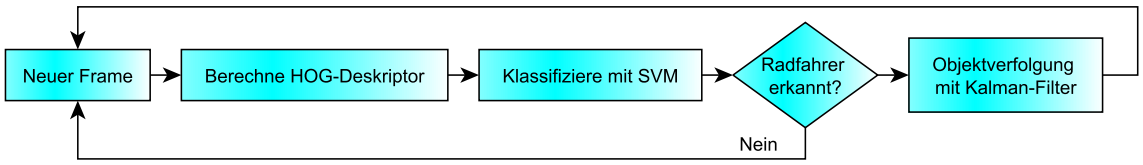


Abbildung 2.3.: Schematische Darstellung für das Konzept des Algorithmus

Im vorgestellten Konzept wird jeweils ein Frame bzw. die gewählte ROI des Frames analysiert. Für den Algorithmus ist es daher unerheblich, ob das Bild aus einer Videodatei oder einem Videostream extrahiert wird. Dieser Umstand ermöglicht zunächst die Implementierung des Algorithmus auf einem leistungsstarken Rechnersystem, auf welchem die in Kapitel 4 vorgestellten Videosequenzen als Datengrundlage analysiert werden können. Für die spätere Portierung auf ein autarkes Embedded System wurde der Raspberry Pi 2 Modell B [RAS17b] gewählt. Aus einem Bachelorprojekt der Hochschule für angewandte Wissenschaften (HAW) steht bereits die Hardware zu Verfügung. Zudem besteht der Wunsch, die Hardware unter der zugrunde liegenden Aufgabe auf ihre Rechenleistung zu untersuchen. Ein proprietäres Kameramodul inklusive Treiberunterstützung [RAS17a] ermöglicht eine einfache Einbindung einer Kamera für den Livestream-Betrieb. Dieses bereits in Abschnitt 2.1 erwähnte Modul bietet eine Framerate von 30 FPS bei einer Auflösung von 720p und erfüllt somit die Anforderung für echtzeitfähige Videoaufnahmen. Im Verbund mit einem für das Kameramodul vorgesehenen Gehäuse und dem Einsatz eines Akkus ist das System dann grundsätzlich autark. Abbildung 2.4 zeigt das beschriebene System. An dieser Stelle sei erwähnt, dass für ein autarkes System unter Realbedingungen ein wetterfestes Gehäuse bereit gestellt werden müsste. Zudem ist die Laufzeit durch den Einsatz eines Akkus begrenzt. Für einen Dauerbetrieb wäre eine Versorgung durch eine fest installierte Stromversorgung oder Solarpanel möglich.

Abbildung 2.4.: Einplatinencomputer Raspberry Pi 2 B mit Kameramodul, WLAN-Stick und Powerbank

Bereits am Entwicklungsbeginn des Algorithmus wurde ersichtlich, dass das im Rahmen des Studiums häufig verwendete Simulationstool Matlab einen für die zugrunde liegende Aufgabe erheblichen Nachteil aufweist. Die nötige Computer Vision Systems Toolbox für die bereits optimierten Funkionen für die Berechnung eines HOG-Deskriptors, `extractHOGFeatures(I)`, ist nicht in der den Studenten zugänglichen Version enthalten. Eine komplette Eigenentwicklung der benötigten Algorithmen würde einen erheblichen Zeitaufwand und die Gefahr vieler unzureichend optimierter Code-Abschnitte mit sich bringen. Eine gute Alternative für die Entwicklung von Algorithmen in der Bildverarbeitung bzw. Computer Vision bietet hier OpenCV [2]. Es handelt sich dabei um eine umfangreiche Bibliothek von laufzeitoptimierten Funktionen, welche häufig im Zuge einer Entwicklung im Bereich Bildverarbeitung verwendet werden. Die Programmierung der Quelldateien erfolgte in C++, wobei eine Unterstützung der Skriptsprache Python zur Ansteuerung ebenfalls gewährleistet ist. Die Hochsprachen C und C++ werden im Studium der HAW intensiv behandelt, weshalb auf einen guten Erfahrungsstand zurückgegriffen werden kann. Daneben bietet OpenCV als Open Source Projekt eine aktive Community und daher bereits viele Dokumentationen im Internet. Des weiteren wird die spätere Portierung auf den Einplatinencomputer Raspberry Pi ermöglicht, indem der entwickelte C++ Quellcode durch Einsatz der bekannten GNU Compiler Collection (GCC) auf dem Zielsystem compiliert werden kann.

[2] Homepage des OpenCV-Teams: `http://opencv.org/`

3. Grundlagen

Dieses Kapitel erläutert die relevanten Grundlagen für die Entwicklung des Algorithmus. Der Aufbau leitet sich dabei grundlegend vom vorgestellten Konzept (vgl. Abbildung 2.3) ab. Abschnitt 3.1 geht zunächst auf die Berechnung der HOG-Deskriptoren ein. Anschließend werden in Abschnitt 3.2 die Grundlagen einer SVM mit linearem Kernel vorgestellt und der Einsatz einer Non-Maximum Suppression (NMS) in Abschnitt 3.3 erläutert. Abschließend beschreibt Abschnitt 3.4 die Grundlagen einer Objektverfolgung durch Einsatz eines Kalman-Filters.

3.1. Histogram of Oriented Gradients

Ein Histogram of Oriented Gradients (HOG) bezeichnet die Anordnung der Gradienten eines Samples innerhalb eines Histogramms. Die Arbeit von [Dal06] stellte das Verfahren erstmals im Rahmen der Doktorarbeit vor und erzielte bis dahin sehr gute Ergebnisse. Als Merkmal für eine Objektklassifizierung werden die Gradienten interpretiert, d.h. die Richtung und Stärke von Kanten in einem festgelegten Bildausschnitt. Die Gradienten werden für Teilausschnitte des Samples in einem Histogramm nach ihrer Ausrichtung angeordnet. Die Betrachtung des entstehenden Vektors reduziert zum einen die Dimensionalität des 2D-Bildes auf einen 1D-Vektor und begünstigt so schnellere Rechenzeiten der Klassifikation. Zum anderen werden durch die Anordnung der Merkmale in einem Histogramm die Farb- und Intensitätswerte des Bildes quasi geglättet, was durch eine Normalisierung der Gradienten gestärkt wird.

Die Berechnung der Gradienten erfolgt zunächst durch die Faltung des Bildausschnitts A mit einem Sobel Operator (vgl. auch [DK15]) der 1. Klasse. Gleichung 3.1 zeigt die Sobel Operatoren für die Berechnung der x- sowie y-Richtung.

$$g_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * A, \qquad g_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * A \tag{3.1}$$

Anschließend werden Absolutwert (engl.: Magnitude) (vgl. Gleichung 3.2) und Ausrichtung (vgl. Gleichung 3.3) der Gradienten, unter einbeziehen beider Richtungen, bestimmt.

$$g = \sqrt{g_x^2 + g_y^2} \tag{3.2}$$

$$\theta = \arctan \frac{g_y}{g_x} \tag{3.3}$$

Nach [Dal06] wird der zu untersuchende Bildausschnitt in 8 x 8 Pixel umfasste Zellen unterteilt, in denen also $8 \cdot 8 = 64$ Gradienten bestimmt werden. Die Ausrichtungen der Gradienten wiederum werden nun in ein Histogramm mit neun Unterteilungen bzw. Klassen (engl.: Bins) des Absolutwerts nach angeordnet. Abbildung 3.1 zeigt eine Anordnung des ersten und vierten Pixels anhand von Beispielwerten. Dabei sei erwähnt, dass die Ausrichtung vorzeichenfrei (Grad von 0 bis 180) gewählt ist und für die negative Repräsentation der Ausrichtung der gespiegelte Wert angenommen wird.

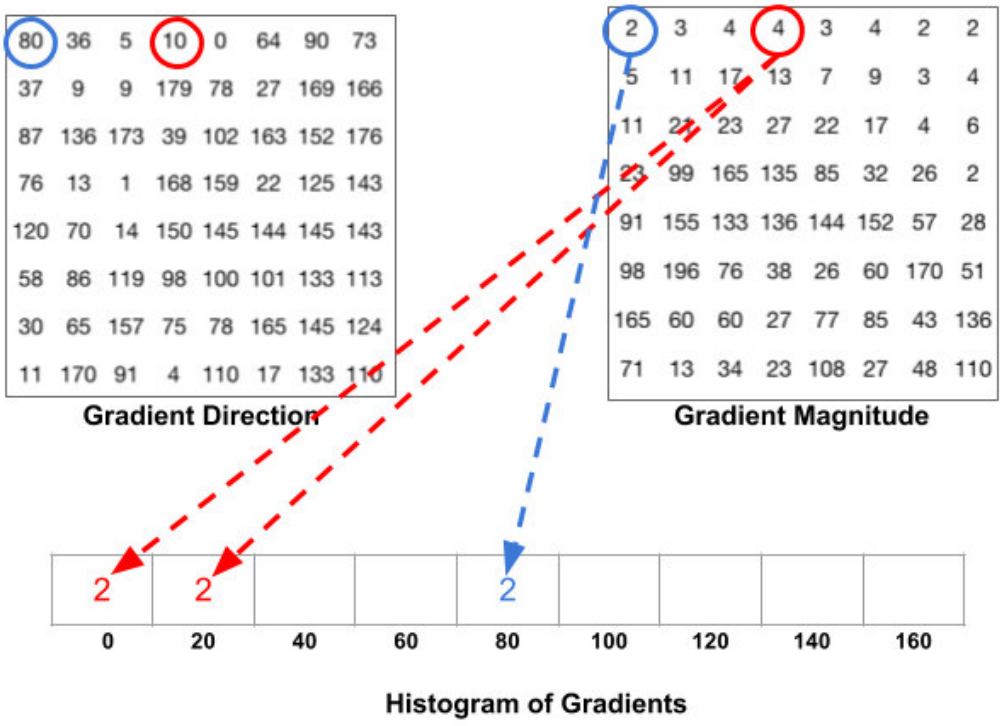


Abbildung 3.1.: Anordnung eines 9-bin HOG anhand von Beispielwerten einer 8 x 8 Zelle [Mal16]

Abschließend werden jeweils vier Zellen in einem 16 x 16 Pixel Block normalisiert. Bei der Normalisierung handelt es sich um eine L^2-Norm (vgl. [Wol17]), dessen Berechnung in Gleichung 3.4 dargestellt wird.

$$|x| = \sqrt{\sum |x_k|^{2^n}_{k=1}} \tag{3.4}$$

Abbildung 3.2 zeigt den Ablauf der angesprochenen Berechnungsschritte. Die Gammakorrektur bietet laut [Dal06, S. 36f] kaum Performance Gewinn, welche standardmäßig nicht angewendet und daher hier auch nicht näher behandelt werden soll.

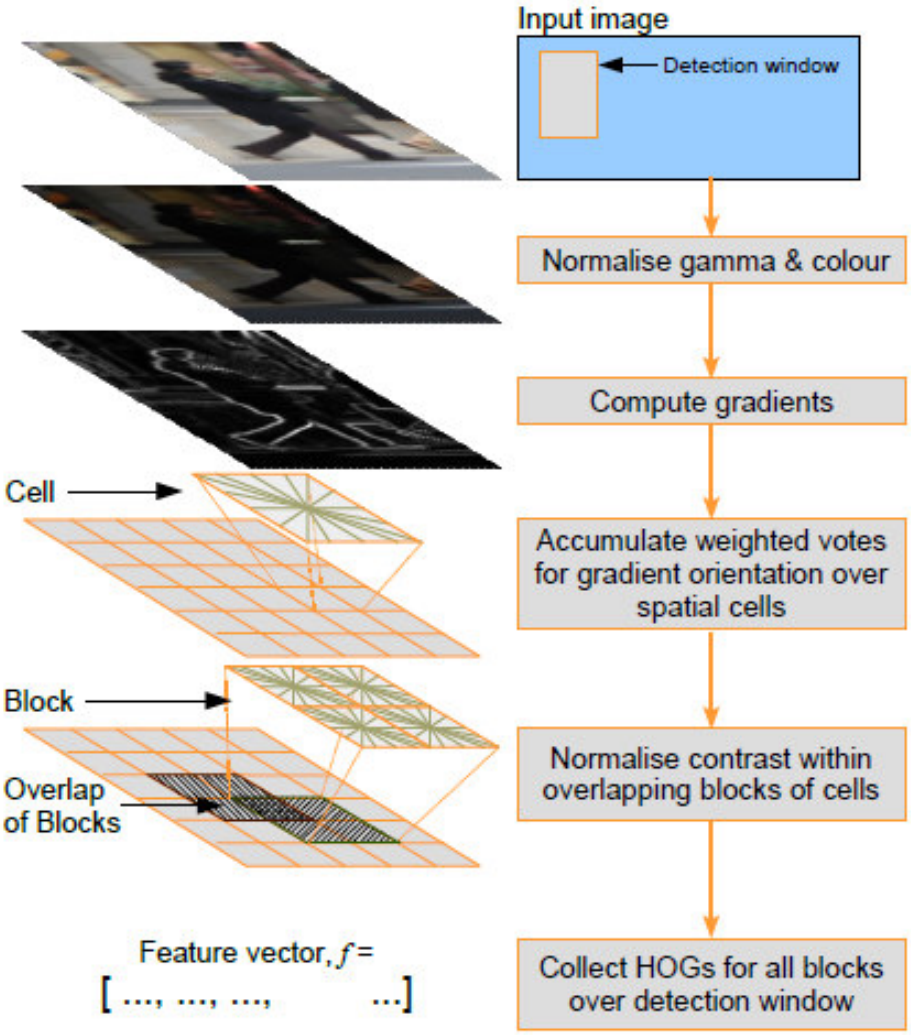
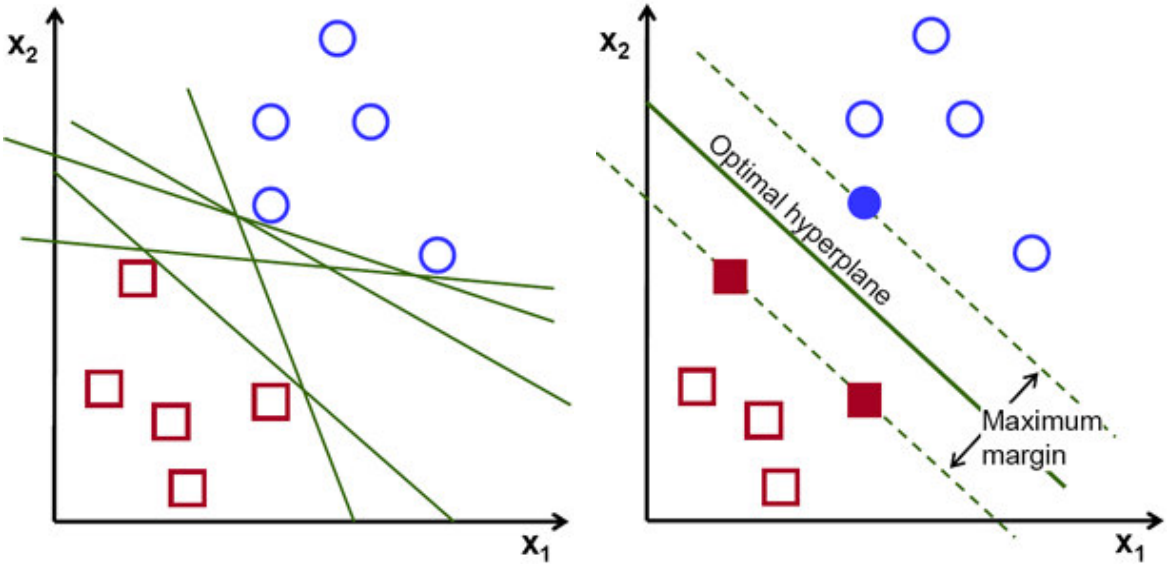


Abbildung 3.2.: Schematischer Ablauf der Berechnung eines HOG-Deskriptors nach [Dal06, S. 23]

Der finale Deskriptor berechnet sich über die Anzahl aller Blöcke eines zu untersuchenden Bildausschnitt (engl.: Detection Window). Für den für die Detektion von Personen entwickelten Deskriptor (vgl. [Dal06]) der Größe 64 x 128 Pixel sind dies 7 Blocks in x- und 15 Blocks in y-Richtung. Insgesamt sind also 105 Blocks zu berechnen, von denen sich jeder zu 4 Zellen mit jeweils einem 9-bin Histogramm aufgliedert. Die Größe des Merkmalvektors beträgt dann $105 \cdot 4 \cdot 9 = 3780$. Für den im Rahmen der Arbeit definierten HOG-Deskriptors der Größe 176 x 176 Pixel berechnet sich die Größe des Vektors dann zu $21 \cdot 21 \cdot 4 \cdot 9 = 15876$.

3.2. Support Vector Machine

Bei einer Support Vector Machine (SVM) handelt es sich um einen binären Klassifikator der zwischen zwei Klassen differenziert. Dabei werden die Klassen nach vorgegebenen Attributen im Raum angeordnet und durch eine Hyperebene (engl.: Hyperplane) voneinander getrennt. Gesucht wird dabei die Hyperebene mit dem größtmöglichen Abstand (engl.: Maximum Margin) zwischen den beiden zu unterscheidenden Klassen. Für eine Klassifikation im zweidimensionalen Merkmalsraum kann die Hyperebenen auch als einfache Trennlinie zwischen den Klassen verstanden werden. Abbildung 3.3 zeigt ein solches Klassifikationsproblem im zweidimensionalen Raum. in Abbildung 3.3b ist die optimale Hyperebene und deren Entfernung zu den Klassen (Maximum Margin) eingezeichnet.



(a) Nicht-Optimale Trennlinien für eine zwei Klassen-Klassifikation

(b) Optimale Hyperplane als Trennlinie für die Klassifikation

Abbildung 3.3.: Nicht-Optimale und optimale Hyperebene einer Regressionsanalyse im zweidimensionalen Merkmalsraum [Ope14a]

Für Berechnung der Support-Vektoren, welche die Hyperebene definieren, können verschiedene Regressionsverfahren und Kernel angewendet werden. Im Rahmen dieser Arbeit wurde eine ϵ-Support Vector Regression (ϵ-SVM) mit einem linearen Kernel angewendet. Bei dieser Regression wird während des Trainings der SVM eine Hyperebene gesucht, dessen Maximum Margin kleiner gleich ϵ beträgt.

Die Implementierung der verwendeten SVM der OpenCV-Bibliothek richtet sich nach [CL13]. Für nähere Informationen zu weiteren Verfahren sei auf diesen Report verwiesen. Gleichung 3.5 zeigt die Summenfunktion der Vektoren x_i eines Datensatzes für die Lösung des Optimierungsproblems der ϵ-SVM. α_i beschreibt die Gesamtheit der Vektoren für eine mögliche Gewichtung der Support Vektoren x_i. Im Fall einer linearen Separierbarkeit der Klassen gilt $\forall \alpha_i = 1$. Der Skalar b wird in der OpenCV-Implementierung auch als ρ bezeichnet und beschreibt einen zusätzlichen Offset der gewichteten Support Vektoren. [Ope14b]

$$\sum_{i=1}^{l} (-\alpha_i + \alpha_i^*) K(x_i, x) + b \tag{3.5}$$

Gleichung 3.6 beschreibt den linearen Kernel $K(x_i, x)$.

$$K(x_i, x) = x_i^T x \tag{3.6}$$

Abschnitt 5.2 geht näher auf den Ablauf der Trainings- und Detektionsphase, sowie auf die Parametrisierung der realisierten SVM ein. Diese Arbeit befasst sich nicht näher mit dem generellen Optimierungsproblem beim Trainieren einer SVM. Für weiterführende Definitionen und Grundlagen sei daher auf [CST00] verwiesen.

3.3. Non-Maximum Suppression

Die Non-Maximum Suppression (NMS) dient dem Filtern der Ergebnismenge durch Löschen nicht-signifikanter Ergebnisse. Verwendung findet der Begriff auch im Canny-Algorithmus [Pri96]. Hier werden Grauwerte eines Kantenbilds nur dann der Ergebnismenge zugeordnet wenn sie das lokale Maximum aufweisen.

Im Rahmen einer Objekterkennung, insbesondere beim Einsatz eines Sliding Window-Verfahren, können für ein Objekt mehrere Ergebnisse produziert werden, d.h. das Objekt wird mehrfach erkannt. Da eine Eindeutigkeit gewünscht ist, kann hier die NMS Abhilfe schaffen, indem die Treffermenge gefiltert wird. Für Bounding Boxes [Wik17c] geschieht dies durch Überprüfung, inwiefern sich die gefunden Objekte überschneiden. Liegt keine Überschneidung vor, wird die lokale Ergebnismenge einem Objekt zugeordnet. Für örtliche Überschneidungen wird dann der Grad der Überschneidung zweier Flächen gemessen. Wird ein festgelegter Grenzwert nicht überschritten, werden die Treffer gelöscht, bis entweder nur ein Ergebnis übrig bleibt oder, im Falle von nahe platzierten Objekten, eine Unterscheidung mehrerer Ergebnisse durch den Vergleich der Flächenüberschneidung abgeschlossen wird.

3.4. Objektverfolgung mit Kalman-Filter

Das Kalman-Filter beschreibt eine Sammlung mathematischer Gleichungen mit denen bei einer fehlerbehafteten Beobachtung bzw. Messung eine Aussage über den Zustand der betrachteten Variable oder des betrachteten Systems ermöglicht wird. Des weiteren lässt der Einsatz eines Kalman-Filters eine Aussage über den zukünftigen Systemzustand zu. In diesem Zusammenhang kann eine Objektverfolgung realisiert werden, indem eine Voraussage der Bewegung eines aus dem Detektionsraum tretenden Objektes getätigt wird. [Wik17b] Die hier erbrachte Definition richtet sich nach [Bal16], wobei für weiterführende Informationen auf [WB06] und [CZR05] verwiesen sei.

Im Rahmen einer Objektverfolgung für zweidimensionale Bilder kann der Systemzustand X wie in Gleichung 3.7 beschrieben werden. Die Anfangsbedingung des Systemzustands wird dann als X_0 definiert. An dieser Stelle kann bereits durch Vorwissen beispielsweise eine geschätzte Geschwindigkeit vorausgesetzt werden, um die Näherung an den Ist-Zustand zu beschleunigen.

$$X = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{matrix} \text{Position x} \\ \text{Position y} \\ \text{Geschwindigkeit in x-Richtung} \\ \text{Geschwindigkeit in y-Richtung} \end{matrix} \tag{3.7}$$

Die Kovarianzmatrix P (vgl. Gleichung 3.8) gibt die Unsicherheit des Ist-Zustands an. Im ersten Schritt kann hiermit eine Unsicherheit über den Zustand X_0 angegeben werden. Ist der Ausgangszustand sicher bekannt, kann die Kovarianzmatrix mit einer geringen Standardabweichung initialisiert werden. Als Beispielswert sei hier eine Abweichung von $\sigma_0 = 10$ angegeben. Je höher die Abweichung dimensioniert wird, desto schneller konvergiert das Filter an den Ist-Zustand.

$$P = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \tag{3.8}$$

Die Bewegungsrichtung des Objekts innerhalb einer Recheniteration kann über die Dynamikmatrix A erfasst werden. Position und Geschwindigkeit des Folgezustands lassen sich dann durch die Multiplikation des Ist-Zustands mit der Dynamikmatrix berechnen. Im zugrunde liegenden Fall wird eine konstante Geschwindigkeit der Radfahrer angenommen. Damit gilt für die Bestimmung von Position und Geschwindigkeit dann $x_{t+1} = \dot{x}_t \cdot t + x_t$ bzw. $\dot{x}_{t+1} = \dot{x}_t$. Gleichung 3.9 zeigt die aufgestellte Dynamikmatrix A. Im Fall diskreter Schritte von einem Frame zum nächsten kann $dt = 1$ gesetzt werden.

$$A = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

Unter Realbedingungen ist die Bewegung eines Objekts meist durch eine Überlagerung eines normalverteilten Rauschens gestört. Die Prozessrauschkovarianzmatrix Q (vgl. Gleichung 3.11) bezieht diese Störung der Objektbewegung durch Angabe einer Standardabweichung ein. Über Gleichung 3.10 kann die Matrix Q aufgestellt werden. Dabei muss für die Standardabweichung der Beschleunigung σ_a vorab ein Wert geschätzt oder durch Vorwissen angegeben werden.

$$Q = GG^T\sigma_a^2, \quad \text{mit } G = [0.5\Delta t^2 \; 0.5\Delta t^2 \; \Delta t \; \Delta t]^T \tag{3.10}$$

$$Q = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\dot{x}} & \sigma_{x\dot{y}} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{y\dot{x}} & \sigma_{y\dot{y}} \\ \sigma_{\dot{x}x} & \sigma_{\dot{x}y} & \sigma_{\dot{x}}^2 & \sigma_{\dot{x}\dot{y}} \\ \sigma_{\dot{y}x} & \sigma_{\dot{y}y} & \sigma_{\dot{y}\dot{x}} & \sigma_{\dot{y}}^2 \end{bmatrix} \tag{3.11}$$

Die Messmatrix H definiert die Messgröße (hier die Geschwindigkeit) und das Verhältnis der Messwerte zum Zustandsvektor. Gleichung 3.12 zeigt die aufgestellte Messmatrix mit einem Verhältnis von 1, bei dem die Messwerte direkt übernommen werden.

$$H = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

Die Messrauschkovarianzmatrix R berücksichtigt die Ungenauigkeit der Messwerte. Die Standardabweichung muss auch hier z.B. anhand der Präzisison der verwendeten Sensoren geschätzt werden. Wird eine sehr genaue Messung angenommen, kann die Matrix mit kleinen Werten initialisiert werden (Beispielswert $\sigma_r = 1$). Gleichung 3.13 zeigt die Messrauschkovarianzmatrix R für den Anwendungsfall einer Messung von Geschwindigkeit in x- und y-Richtung.

$$R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.13}$$

Die Steuermatrix B und Steuergröße u dienen zur Definition von externen Einflüssen. Wird beispielsweise die Bewegung eines Kfz verfolgt, können über Sensoren Messwerte von Bremsen oder Ähnlichem erfasst werden. Im zugrunde liegendem Fall können keine dieser externen Einflüsse berücksichtigt werden, weshalb B und u nicht näher erläutert werden.

Die eigentliche Voraussage des Folgezustands X_{k+1} erfolgt wie erwähnt über die Multiplikation des Ist-Zustands x_k mit der Dynamikmatrix A (vgl. Gleichung 3.14). Die Kovarianzmatrix P_{k+1} wird ebenfalls bei der Voraussage aktualisiert, um die aktuelle Unsicherheit der Messung zu berücksichtigen (vgl. Gleichung 3.15).

$$X_{k+1} = Ax_k \tag{3.14}$$

$$P_{k+1} = AP_kA^T + Q \tag{3.15}$$

Nachdem die aktuellen Messwerte gesammelt wurden, erfolgt die Korrektur des Kalman-Filters. Dabei wird zunächst der Kalman-Gain bestimmt (vgl. Gleichung 3.16). Weicht der vorausgesagte Systemzustand stark von den eigentlichen Messwerten ab, steigt K_k und korrigiert somit den Systemzustand X_k (vgl. Gleichung 3.17).

$$K_k = P_kH^T(HP_kH^T + R)^{-1} \tag{3.16}$$

$$X_k = X_k + K_k(z_k - HX_k) \tag{3.17}$$

Abschließend wird erneut die Kovarianzmatrix P_k aktualisiert (vgl. Gleichung 3.18), um im folgenden Iterationsschritt eine möglichst gute Voraussage zu ermöglichen.

$$P_k = (I - K_k H) P_k \tag{3.18}$$

Für die Funktionsweise des Kalman-Filters ist vor allem das Verhältnis von Prozessrauschkovarianzmatrix Q und Messrauschkovarianzmatrix R entscheidend. Je größer das Verhältnis der Absolutwerte von Q zu R, desto schneller reagiert das Filter auf Änderungen der Messwerte. Die Dynamik steigt also und ermöglicht einen schnellere Konvergenz des Systemzustands. Werden hingegen im Verhältnis größere Werte der Messrauschkovarianzmatrix R gewählt, berücksichtigt das Filter das Systemverhalten anstelle der Messwerte. Messungenauigkeiten können so gefiltert werden und im Falle einer Nichtdetektion wird das Objekt korrekt verfolgt, sofern eine hinreichend genaue Systembeschreibung durch vorangegangene Filteriterationen gegeben ist.

4. Videoaufnahmen der Trainings- und Testdaten

Die Recherche nach geeigneten Daten für die Lern- und Testphase zeigte, dass im Bereich der Fußgängererkennung bereits umfangreiche und strukturierte Datensätze vorhanden sind. Ein wichtiger Datensatz, welcher häufig als Referenz für Algorithmen herangezogen wird, ist der INRIA[1] Datensatz, welcher im Rahmen der Arbeit von [Dal06] entwickelt wurde. Er umfasst 2416 Trainings- und 1126 Testsamples (Windowgröße: 64 x 128) und wurde auch für die Erstellung des in OpenCV ladbaren HOG-Deskriptor, für die Objektdetektion von Personen verwendet (aufrufbar über die Funktion `getDefaultPeopleDetector()`). Im Bereich Radfahrer wurde nach [TL15] der KITTI Datensatz [GLU12] untersucht, welcher in Projektarbeit der KIT und der TTIC erstellt wurde[2]. Dieser beinhaltet verschiedene Objekte (Kfz, Personen, Radfahrer, ...), welche bereits mit Positionsangabe und Objektbezeichnung versehen sind. Es fällt jedoch auf, dass die Skalierung und Ausrichtung der für diese Arbeit relevanten Objekte, also Radfahrer, stark variieren. Einige Radfahrer sind zudem nahezu vollständig verdeckt und eignen sich daher nicht für das Anlernen der SVM.

Um für die Trainingsphase geeignete Samples bereitzustellen, wurden daher eigene Videoaufnahmen erstellt, die unter den Einsatzbedingungen des zu entwickelnden Systems getätigt wurden. Für den Ort der Aufnahme wurde der Standort Gurlitt-Insel in Nähe der Zählsäule (vgl. Abbildung 1.1) gewählt, um zukünftige Vergleiche entwickelter Algorithmen zu gewährleisten. Unter Abschnitt A.2 sind weitere Informationen zum Standort und der verwendeten Hardware für die Videoaufnahmen angegeben. Abbildung 4.1 zeigt einen beispielhaften Frame einer Videoaufnahme der Testdaten mit eingezeichneter ROI. Die für die Trainingsphase verwendeten Samples wurden jeweils aus dem durch die ROI gekennzeichneten Bereich ausgeschnitten. Abbildung 4.2 zeigt einige Beispiele für die gewählten positiven und negativen Trainingssamples. Die Videoaufnahmen sowie die extrahierten Samples liegen der CD dieser Arbeit bei (siehe Abschnitt A.1).

[1] Staatliche französische Forschungseinrichtung Institut national de recherche en informatique et en automatique (INRIA), deutsch: Nationales Forschungsinstitut für Informatik und Automatisierung

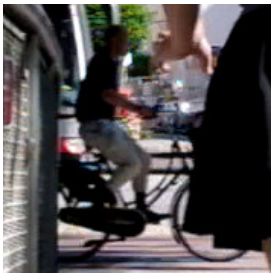
[2] Homepage der KITTI Datensätze: `http://www.cvlibs.net/datasets/kitti/`

Abbildung 4.1.: Beispielhafter Frame (Video: SAM_9128.mp4) mit eingezeichneter ROI

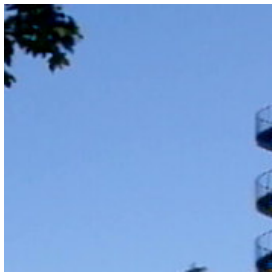
(a) Sample mit freier Sicht auf Objekt

(b) Sample wie (a) einige Frames versetzt

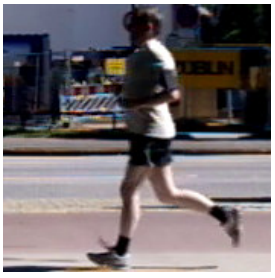
(c) Sample mit leichter Überdeckung



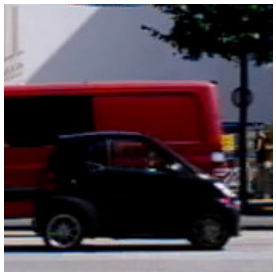
(d) Sample mit starker Überdeckung



(e) Negativsample ohne jegliche Objekte



(f) Negativsample mit Objekt (Person)



(g) Negativsample mit Objekt (Auto)

(h) Negativsample des leeren Radwegs

Abbildung 4.2.: Beispielhafte Auswahl einiger Positiv- und Negativamples der Trainingsdaten

Insgesamt wurden an zwei unterschiedlichen Tagen Videoaufnahmen durchgeführt, welche verschiedene Lichtverhältnisse abdecken. Die ersten Aufnahmen wurden am 21. Juli 2016 aufgenommen und umfassen zwei Videodateien. Dabei dient das erste Video als Quelle für die Trainingsdaten, das zweite als Test für den Algorithmus. In einer ersten Iteration wurden für die Trainingsdaten ausschließlich Radfahrer ausgeschnitten, welche die ROI von links nach rechts passierten. Die Anzahl an Radfahrern im 11:06 dauernden Video, auf denen die genannte Richtung zutrifft, liegt bei 69. Um die Menge der Trainingsdaten zu erhöhen wurden zwei Ausschnitte eines Radfahrers gewählt, wobei der zweite Ausschnitt einige Frames versetzt zum ersten gewählt wurde. Dies gilt nur sofern das Objekt in den folgenden Frames nicht überwiegend verdeckt ist (mehr als 80 % Verdeckung). Die Negativsamples können generell beliebig gewählt werden. In diesem Fall wurden größtenteils zufällig ausgewählte Bereiche der Videosequenz ausgeschnitten. Zudem wurden einige Klassen wie Personen oder Kfzs explizit als Negativsample angegeben.

Die zweite Aufnahme wurde am 14. Februar 2017 aufgenommen und umfasst eine 09:55 andauernde Videosequenz unter abweichenden Lichtverhältnissen im Vergleich zum ersten Aufnahmetag. Das Video dient zum Test des Algorithmus unter variierenden Sichtverhältnissen. Es ist anzumerken, dass die Position der Kamera nicht vollständig kongruent zum ersten Aufnahmetag ist und daher Unterschiede in den Größenrelationen der Objekte auftreten können.

Die Samples sind jeweils 176 x 176 Pixel groß und wurden ohne Vorskalierung aus den Frames des Testvideos ausgeschnitten. Wie in Abschnitt 3.1 erläutert, berechnet sich damit die Größe der HOG-Deskriptoren eines Samples zu 15876. In Anbetracht der Rechenzeit ist dies im Vergleich zu den Standard-Personendeskriptoren ein wesentlich höherer Wert. Diese sind wie bereits erwähnt nur 64 x 128 Pixel groß und weisen damit eine HOG-Deskriptorgröße von 3780 auf. Die Rechenzeit steigt exponentiell durch den nötigen Mehraufwand für Multiplikationen des HOG-Algorithmus. Abschnitt 6.2 und Abschnitt 6.3 gehen auf die Echtzeitfähigkeit des Algorithmus auf den jeweiligen Zielsystem ein. Für leistungsstarke Rechnersysteme ist eine Echtzeitfähigkeit dabei grundsätzlich erreichbar.

	SAM_9127.mp4	SAM_9128.mp4	SAM_9130.mp4
Laufzeit	11:06	06:05	09:55
Rechts passierend	69	37	23
Training-Samples (rechts)	129	-	-
Links passierend	53	33	25
Training-Samples (links)	92	-	-
Radfahrer insg.	122	70	48

Tabelle 4.1.: Objektanzahl (Radfahrer) in den jeweiligen Videoaufnahmen

5. Realisierung des Algorithmus

Dieses Kapitel beschreibt die Entwicklung des Algorithmus und geht näher auf die jeweiligen Teilabschnitte der realisierten Software ein. Die Entwicklung fand zunächst unter Visual Studio 2015 unter Windows 10 statt. Verwendet wurden zudem OpenCV-Bibliotheken der Release-Version 3.2.0. Diese bietet wie in Abschnitt 2.2 erwähnt laufzeitoptimierten Quellcode für Standardanwendungsfälle der Bildverarbeitung und stellt auch die hier benötigten Funktionen für HOG, SVM und Kalman-Filter bereit. Die spätere Kompilierung des entwickelten Codes auf dem Einplatinencomputer Raspberry Pi erfolgte mit der GCC unter Verwendung von CMake[1]. Abschnitt A.3 und Abschnitt A.4 im Anhang der Arbeit bieten eine kurze Anleitung für die Entwicklung weiterer Programme unter Visual Studio oder auf dem Raspberry Pi. Der vollständige Programmcode für den entwickelten Algorithmus, sowie für die im Rahmen der Arbeit erstellte Hilfsprogramme, liegt der CD dieser Arbeit bei (siehe Abschnitt A.1).

Abbildung 5.1 zeigt den vereinfacht dargestellten Programmablauf für den vollständigen Algorithmus zur Radfahrerzählung. Innerhalb der Initialisierung werden zunächst die benötigten Variablen angelegt und die Videodatei oder Videostream geöffnet. Außerdem werden die vorab antrainierte SVM bzw. der Support Vektor und die Parameter des verwendeten Verfahrens geladen. Anschließend erfolgt die Vorverarbeitung, welche sich auf das Ausschneiden der definierten ROI begrenzt. Für jeden neuen Frame bzw. dessen ROI erfolgt dann die Objektdetektion, an dessen erster Stelle die Berechnung der HOG-Desriptoren steht. Abschnitt 5.1 geht näher auf die Parametrisierung dieser Merkmalsvektoren ein. Die Klassifikation der Objekte findet ebenfalls im Funktionsblock Detect Objects statt. Abschnitt 5.2 beschreibt die Parametrisierung der für die Klassifikation zuständigen SVM und geht näher auf den Ablauf der Trainings- und Klassifikationsphase ein. Bei einer erfolgreichen Detektion erfolgt eine Initialisierung und Zuweisung der Kalman-Filter für die jeweiligen Objekte. Anschließend erfolgt die Aktualisierung des Systemzustands der Filter und eine Visualisierung der Objektdetektion und -verfolgung im dargestellten Frame. Der Ablauf für die realisierte Objektverfolgung mittels Kalman-Filter wird im Einzelnen näher in Abschnitt 5.3 erläutert.

[1] Plattformunabhängiges Tool für die Unterstützung von Kompilierung und Linken externer Bibliotheken. Für weitere Informationen sei auf `https://cmake.org/` verwiesen.

Für das entwickelte Hauptprogramm `CyclistCamCount.exe` sind beispielhafte Screenshots im Anhang unter Abschnitt A.5 beigefügt, welche die Visualisierung des aktuellen Frames sowie Messwerte der Programmabschnitte anzeigen.

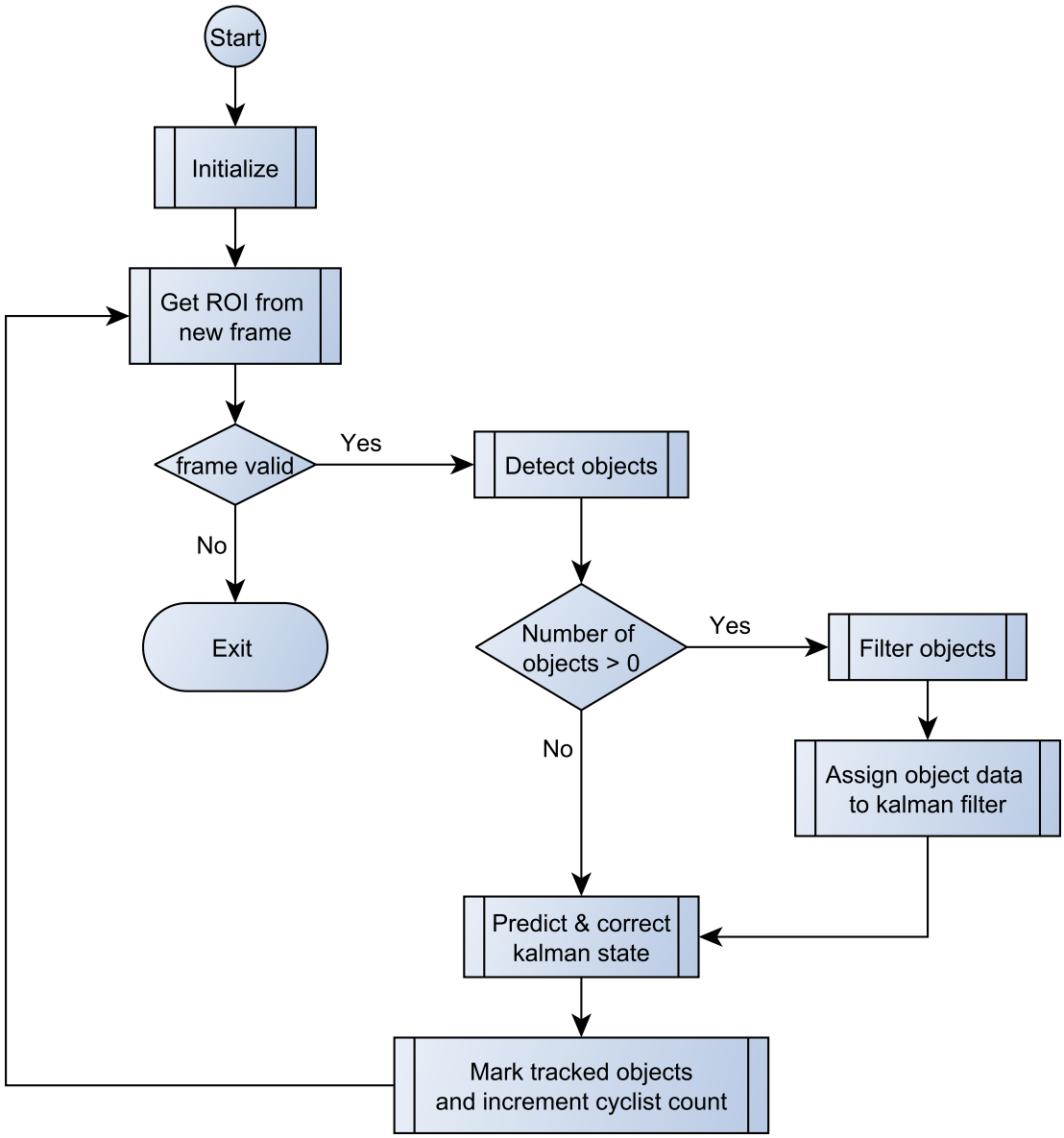


Abbildung 5.1.: Programmablauf des Algorithmus zur Radfahrerzählung

Neben dem Hauptprogramm wurde zudem ein Programm `SampleGrabber.exe` entwickelt um Samples aus einer vorab definierten Videosequenz auszuschneiden. Dabei wird jeder Frame des Videos betrachtet und eine Boundig Box manuell über den Frame bewegt. Geeignete Positve oder zufällig ausgewählte Negativ Samples können dann sukzessive abgespeichert werden und dienen dem Training der SVM.

Generell sei noch auf die Einstellungen des Compilers hingewiesen. Das Kompilieren unter Debug-Parametern (Optimierung deaktiviert: `/Od`) führt keine Optimierung durch und verhindert dadurch ein mögliches „Wegoptimieren“ von Codeabschnitten oder Variablen. Außerdem werden Variablen während der Laufzeit nicht unmittelbar freigegeben, sondern bleiben für die Betrachtung während des Debuggen zwischengespeichert. Weitere Details können in den Einstellungen des Projekts vorgenommen werden. Im Rahmen der Arbeit wurde insbesondere für die Untersuchungen in Kapitel 6 die Kompilierung unter Release-Parametern (Optimierung auf Geschwindigkeit maximieren `/O2`) vorgenommen. Mit einem Faktor von etwa 10 kann die Laufzeit der Programme dadurch stark verringert werden.

5.1. Auslegung des HOG-Deskriptors

Listing 5.1 zeigt die Parameter der für die Auslegung des HOG-Deskriptors zuständige OpenCV-Klasse `HOGDescriptor`. Die Implementierung richtet sich dabei nach [Dal06]. Die Größe der Zellen, Blöcke und Bins die unter Abschnitt 3.1 bereits genannt wurden, sind dabei vordefiniert und in der aktuellen Version OpenCV 3.2.0 nicht manuell konfigurierbar. Die weiteren Parameter für die L^2-Normalisierung, sowie die Standardabweichung `_winSigma` des verwendeten Gaussian Smoothing[2], richten sich ebenfalls nach den im Rahmen der Arbeit von [Dal06] ermittelten Standardeinstellungen.

Die `_blockStride` gibt an, in welchem Abstand das Detection Window über den zu untersuchenden Bildbereich bewegt wird und muss ein Vielfaches der `_cellSize` sein. Für eine möglichst vollständige Überlappung der Blöcke über den untersuchten Bildbereich wird eine `_blockStride` von 8 gewählt. Für die Größe des Detection Windows `_winSize` gilt die unter Kapitel 4 vorgestellte Größe von 176 x 176 Pixel der Samples. Wie bereits unter Abschnitt 3.1 wird der hier verwendete Deskriptor mit vorzeichenfreien Gradienten und ohne Gammakorrektur konfiguriert. Durch die Einsparung von Berechnungen lässt sich dann ein leichter Laufzeitgewinn verzeichnen. Die Klasse 1 des Sobel-Operators für die Berechnung der Gradienten wird mit `_derivAperture = 1` eingestellt.

[2] Bewusstes Weichzeichnen des Bildes über den Einsatz einer Gauß- bzw. Normalverteilung

```
cv::HOGDescriptor::HOGDescriptor(
  Size  _winSize = Size(176, 176),
  Size  _blockSize = Size(16, 16),
  Size  _blockStride = Size(8, 8),
  Size  _cellSize = Size(8, 8),
  int   _nbins = 9,
  int   _derivAperture = 1,
  double  _winSigma = -1,
  int   _histogramNormType = HOGDescriptor::L2Hys,
  double  _L2HysThreshold = 0.2,
  bool  _gammaCorrection = false,
  int   _nlevels = HOGDescriptor::DEFAULT_NLEVELS,
  bool  _signedGradient = false
)
```

Listing 5.1: Klassendefinition des HOG-Deskriptors

5.2. Parameter und Trainingsphase der Support Vector Machine

Listing 5.2 zeigt einen Codeausschnitt der für die Parametrisierung der SVM relevanten Parameter. Dabei folgt Einstellung einem Beispielcode der OpenCV-Bibliothek, welches wiederum die SVM nach [CL13] implementiert. Eingesetzt wird eine ϵ-SVM mit einem linearem Kernel (vgl. Abschnitt 3.2). Die Terminierung des Trainings erfolgt unter den eingestellten Kriterien entweder bei Erreichen des Toleranzwertes `CV_TERMCRIT_EPS` für ϵ oder bei Erreichen der Iterationsgrenze `CV_TERMCRIT_ITER`.

```
  Ptr<SVM> svm = SVM::create();

  /* Default values to train SVM */
  svm->setType(SVM::EPS_SVR); // C_SVC; // EPS_SVR; // may be also
     NU_SVR; // do regression task
  svm->setKernel(SVM::LINEAR);
  svm->setP(0.1); // for EPS_SVR
  svm->setC(0.01); // From paper, soft classifier
  svm->setTermCriteria(TermCriteria(CV_TERMCRIT_ITER + CV_TERMCRIT_EPS,
     1000, 1e-3));

  svm->train(train_data, ROW_SAMPLE, Mat(labels));
```

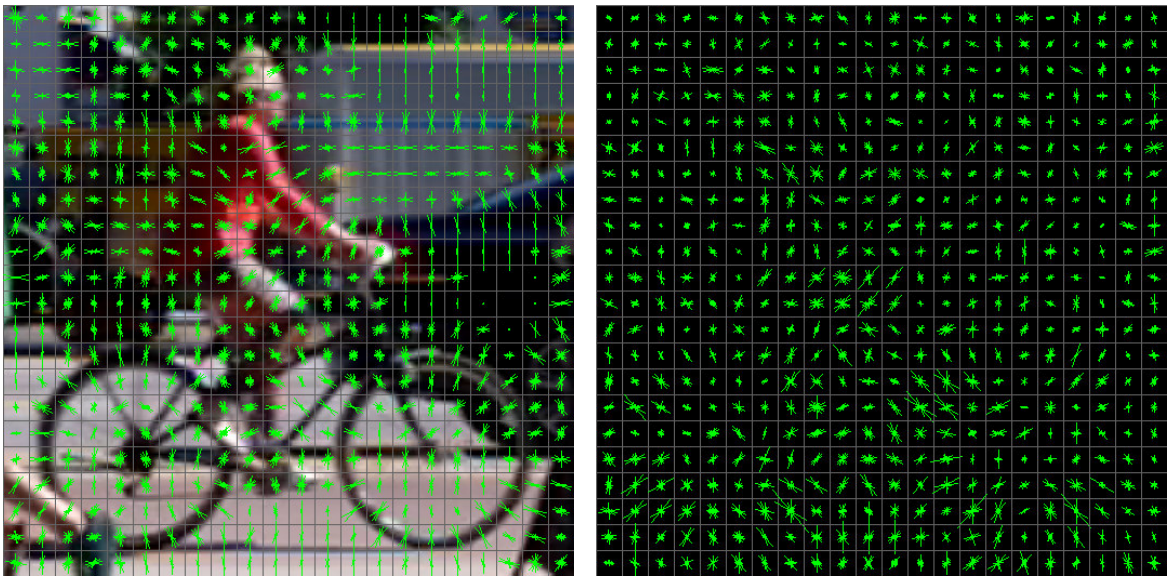
Listing 5.2: Codeausschnitt für die Parametrisierung der SVM

Die Lernphase erfolgte mit den in Kapitel 4 vorgestellten Daten. Angelernt wurde zunächst nur mit rechts passierenden Radfahrern, um zu testen wie robust der Algorithmus mit einer geringen Anzahl an Samples arbeitet. Die Negativsamples wurden zufällig aus dem Hintergrund der Videosequenz extrahiert. Für die Extraktion wurde das am Anfang dieses Kapitels erwähnte Hilfsprogramm `SampleGrabber.exe` verwendet. Mit dem Programm `TrainCyclistSVM.exe`, dessen Quellcode ebenfalls der CD dieser Arbeit beiliegt, kann die SVM dann trainiert werden. Die Funktion `test_svm_cyclist(automationFlag)` bietet zudem die Möglichkeit einen überwachten oder unüberwachten Test zu vollziehen. Bei dem überwachten Test kann der Benutzer für jeden einzelnen Frame eine Detektion innerhalb der ROI oder dem gesamten Bildbereich durchführen und erhält durch Visualisierung potentieller Objektdetektionen unmittelbar Rückmeldung über den Erfolg des Tests. Der unüberwachte Test wird mit dem Übergabeparameter `automationFlag = 1` aktiviert. Dieser führt automatisiert eine Detektion für jede ROI eines Frames der vorab definierten Videosequenz durch und speichert die Ergebnisse als Samples ab. Diese können manuell auf Korrektheit untersucht werden. Im Falle von False Positives, also fälschlicherweise als Radfahrer erkannte Bildausschnitte, können die Samples für eine neue Lerniteration explizit als Negative Sample verwendet werden. Dieses Verfahren wird auch als Hard Negative Training bezeichnet.

Im Fall einer linearen Separierbarkeit der Klassen wird als Ergebnis der Lernphase ein einzelner Support Vektor mit der gleichen Dimensionalität wie die HOG-Deskriptoren abgespeichert. Die im Rahmen der Arbeit erstellten Support Vektoren sind derCompact Disc (CD) der Arbeit beigefügt (vgl. Abschnitt A.1). Für die Klassifikation müssen diese über die Funktion `get_svm_detector(...)` zunächst geladen werden. Die Werte der Support Vektoren werden einem neuen Vektor zugewiesen und der Skalar $\rho = -b$ als letztes Element angehängt. Dieser dient als Offset der durch die Support Vektoren aufgespannten Hyperebene (vgl. Abschnitt 3.2).

Während ein einzelner HOG-Deskriptor eines Bildausschnitts über die Funktion `hog.compute(...)` berechnet werden kann, wird die Klassifikation über Einsatz der Funktion `hog.detect(roi, locationsPoints, foundWeights, hitThreshold)` realisiert. Diese schiebt das Detection Window über den zu untersuchenden Bildbereich und berechnet den zugehörigen Merkmalsvektor bzw. HOG-Deskriptor für jedes Fenster. Anschließend wird das Ergebnis der SVM berechnet und der untersuchte Bildausschnitt ggf. als Objekt klassifiziert. `locationsPoints` und `foundWeights` speichern die Position und die Gewichtung des Ergebnis für die Gesamtheit der gefundenen Objekte. Die Empfindlichkeit der Detektion kann über den Parameter `hitThreshold` variiert werden, welcher als Eingabeparameter der Hauptfunktion `countCyclists(double hitThreshold, ...)` übergeben werden muss.

Abbildung 5.2a zeigt die Visualisierung des HOG-Deskriptors eines Positive Samples. Über die Ausrichtung der Gradienten kann insbesondere eine Kreisstruktur im Bereich der Reifen sowie der Oberkörper des Radfahrers erkannt werden. Abbildung 5.2b zeigt eine Visualisierung des Support Vektors nach gleichem Verfahren. Der Absolutwert der Gradienten ist hier für die Darstellung um Faktor 30 manuell verstärkt worden. Wie bereits bei dem beispielhaft gewählten Positive Sample werden die Strukturen der Reifen und des Oberkörpers erkennbar.



(a) Visualisierung eines positiven HOG-Deskriptors

(b) Visualisierung des Support Vectors (verstärkt um Faktor 30)

Abbildung 5.2.: Visualisierung eines HOG-Deskriptors und des Support Vectors im Vergleich

Im Rahmen dieser Arbeit wurde für die NMS der Algorithmus nach [MGE11] in C portiert und um eine Zentrierung der Ergebnisbox durch hinzufügen der SVM-Gewichtungsfaktoren erweitert. Des weiteren wurde die OpenCV `groupRectangles()` implementiert und die Rechenzeiten der beiden Varianten verglichen. Die Auswertung des Vergleichs wird in Abschnitt 6.2 näher beschrieben.

5.3. Multi Object Tracking mit Kalman-Filter

Im Vorfeld der Untersuchung zur Objektverfolgung wurde zunächst eine Implementierung eines Kalman-Filters für jeweils ein Objekt realisiert. Diese bestätigt die bereits unter Abschnitt 2.3 getätigte Annahme, dass kreuzende Radfahrer mit linearer Bewegung zuverlässig verfolgt werden können. Für den Fall mehrerer Objekte, welche zu derselben Detektionszeit im Bildbereich auftauchen, wird eine Objektverfolgung mit dynamisch angelegten Filtern realisiert. Verwendet wird hier die C++-Klasse Vektor, für die zur Laufzeit neue Elemente angehängt oder alte Elemente gelöscht werden können. Der Systemzustand der Objekte wird durch die Position sowie die korrespondierenden Geschwindigkeiten in in x- und y-Richtung definiert und folgt der Erklärung nach Abschnitt 3.4.

Durch Beobachtung der Messung für jeweils ein Objekt wurde eine Beschleunigung in x-Richtung von etwa 10 bis 20 Pixeln festgestellt. Für den Mittelwert 15 wurde daher die Prozessrauschkovarianzmatrix nach Gleichung 3.10 berechnet. Die Objektverfolgung war insbesondere während des Voraussagemodus jedoch ungenau. Die Voraussage des Systemzustand wich meist stark in y-Richtung ab. Wesentlich bessere Ergebnisse konnten mit dem voreingestellten Wert $\sigma_a = 1^{-4}$ eines OpenCV-Beispiels erzielt werden. Dies lässt sich dadurch erklären, dass die Prozessrauschkovarianzmatrix, wie in Abschnitt 3.4 erwähnt, das Maß einer Störung der Bewegung angibt. Im zugrunde liegenden Fall kann die Störung vernachlässigt werden und daher für die Standartabweichung ein sehr kleiner Wert angenommen werden.

Die Messung der Beschleunigung erfolgt unmittelbar über die Differenz der Positionsdaten der Ergebnismenge des Detektionsalgorithmus. Da diese als genau angenommen werden können, wird für die Standardabweichung der Messrauschkovarianzmatrix R ebenfalls ein kleiner Wert von $\sigma_r = 0.01$ gewählt.

Abbildung 5.3 zeigt den Ablauf des Multi Object Trackings für die Radfahrerzählung. Die dargestellten Berechnungsschritte werden dabei für jeden Frame ausgeführt.

Findet eine Objektdetektion statt, werden für alle Objekte die zugehörigen Entfernungen zu den bereits bestehenden Kalman-Filtern berechnet. Das Objekt mit der geringsten Entfernung wird dann dem korrespondierenden filter zugeordnet, sofern die Entfernung in Pixeln unterhalb eines vorab definierten Grenzwert liegt. Dieser muss per Eingangsparameter der Funktion `countCyclists(..., int seperationDistance, ...)` mitgeteilt werden und wird im Rahmen dieser Arbeit zu `seperationDistance = 80` gewählt. Liegt die Entfernung für jedes Objekt über dem Grenzwert, oder sind keine weiteren Kalman-Filter mehr für die Zuweisung vorhanden, wird für die nicht zugeordneten Objekte jeweils ein neuer Filter initialisiert. Für die definierenden Parameter des Filters gelten dabei die bereits erwähnten Einstellungen, wobei für den Ausgangszustand X_0, die Positionsdaten des zuzuweisenden Objekts

verwendet werden. Zuletzt wird außerdem für jeden neuen Filter der Zähler `cyclistCount` für die eigentliche Radfahrerzählung inkrementiert.

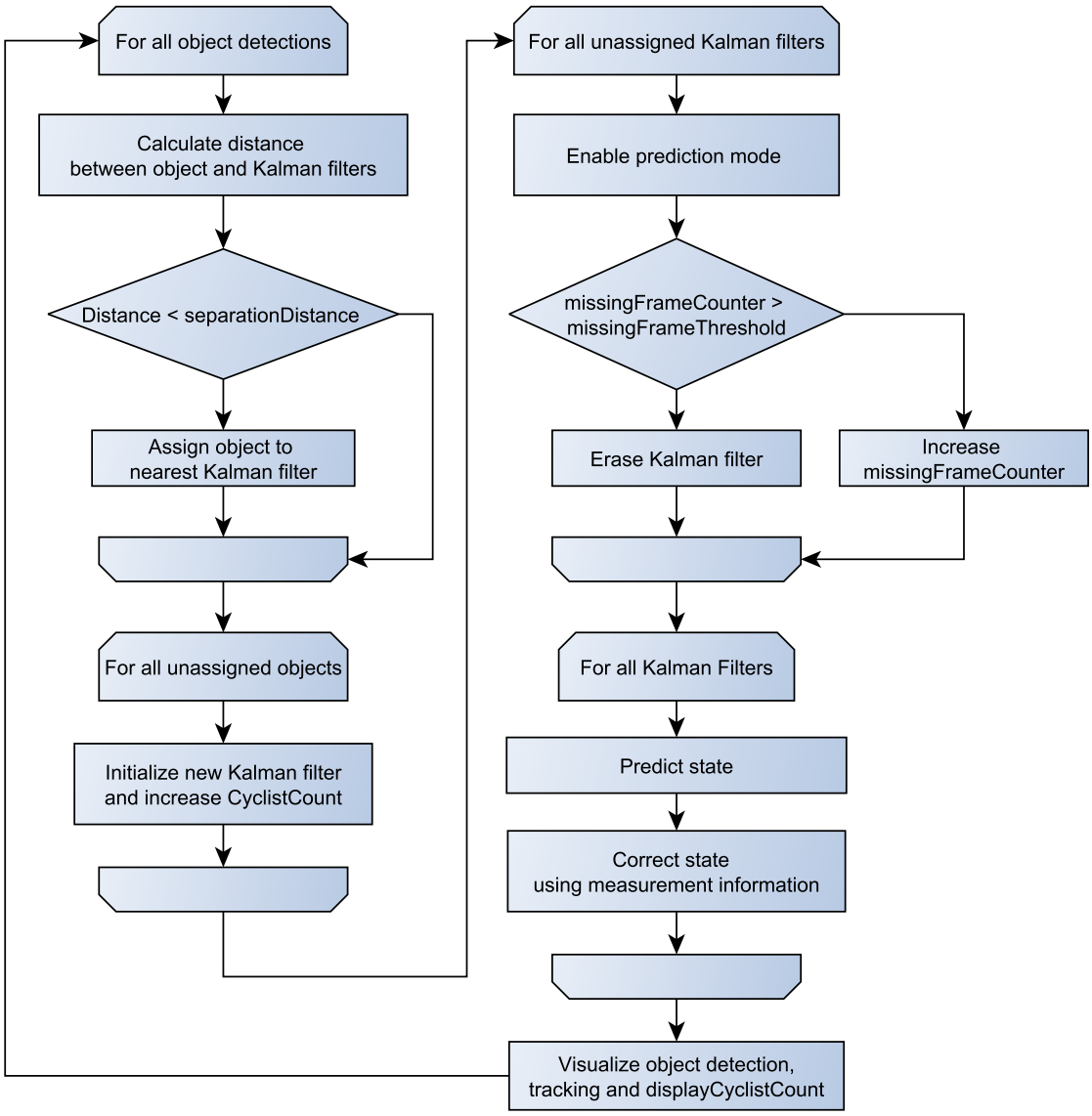


Abbildung 5.3.: Programmablauf des Algorithmus zur Radfahrerzählung

Für alle nicht zugewiesenen Kalman-Filtern wird davon ausgegangen, dass das zugehörige Objekt überwiegend bis gänzlich verdeckt wurde. Das Filter wird nun in den Voraussagemodus gesetzt, indem die Elemente der Messrauschkovarianzmatrix R auf 500 gesetzt werden. In diesem Fall berücksichtigt das Filter den bekannten Systemzustand anstelle der Messwerte für eine Voraussage des Folgezustands. Anschließend wird der

Zähler `missingFrameCounter` inkrementiert, welcher die Anzahl an aufeinander folgenden misslungenen Zuweisungen für das Filter speichert. Steigt dieser über den Grenzwert `missingFrameThreshold = 10`, wird angenommen, das Objekt habe den Detektionsraum verlassen. Das zugehörige Filter wird daraufhin gelöscht. Wird dem Filter nach einer misslungen Zuweisung wieder ein Objekt zugewiesen, wird der Zähler hingegen zurückgesetzt und die Messrauschkovarianzmatrix wieder neu geladen.

Abschließend wird für alle bestehenden Kalman-Filter eine Voraussage über den Systemzustand getroffen. Diese wird im Falle einer korrekten Zuweisung dann durch die Angabe der Messwerte, also der Positionsdaten des Objekts, korrigiert. Zuletzt erfolgt eine Visualisierung der Objektdetektion und -verfolgung im aktuellen Frame (siehe auch Abschnitt A.5).

6. Verifikation des Algorithmus und Auswertung der Ergebnisse

Nachdem die Umsetzung des Algorithmus im vorangegangenen Kapitel vorgestellt wurde, beschäftigt sich dieses Kapitel um die Bewertung der Testergebnisse der Software. Die Bewertung der Erkennungsrate in Abschnitt 6.1 bietet neben einer Aussage über die Qualität der Zählung auch eine generelle Verifikation des Algorithmus. Abschnitt 6.2 untersucht die Echtzeitfähigkeit des Algorithmus auf dem Desktop-PC. Anschließend wird in Abschnitt 6.3 die Portierung auf den Einplatinencomputer Raspberry Pi vorgestellt und die Echtzeitfähigkeit im Vergleich zum Desktop-PC untersucht. Abschnitt 6.4 schließt das Kapitel ab, indem ein Lösungsansatz für die Portierung auf ein dediziertes Embedded System vorgestellt wird.

6.1. Auswertung der Erkennungsrate

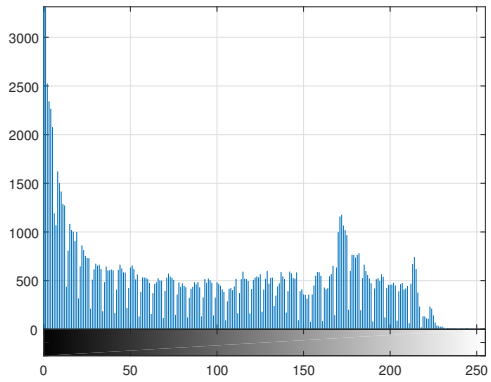
Die Bewertung der Erkennungsrate erfolgt auf dem Desktop-PC (vgl. Kenndaten in Tabelle 6.4) unter Windows 10. Nach der Trainingsphase der SVM wurde vorab ein allgemeiner Funktionstest vollzogen, welcher zeigte, dass ein Hard Negative Training unabdingbar ist. Da nahezu in jedem Frame eine Vielzahl von falschen Klassifikationen (engl.: False Positives) stattfand, konnte die Erkennungsrate kaum eingeschätzt werden. Für die zweite Iteration der Trainingsphase wurden daher 32 False Positives Samples als zusätzliche Negativsamples hinzugefügt. Die Verbesserung war erfolgversprechend und leitete unmittelbar zum eigentlich Test der Erkennungsrate, welche letztendlich auch die Verifikation des Algorithmus darstellt.

Verglichen wurden die unter Kapitel 4 vorgestellten Videosequenzen SAM_9128.mp4 und SAM_9130.mp4. Die erste Sequenz bietet die gleichen Bedingungen hinsichtlich Kameraposition und Lichtverhältnissen wie die des Trainingsvideos. Die zweite wurde im Gegensatz zur ersten nicht im Sommer sondern im Winter gemacht. Zudem ist die Ausrichtung der Kamera nicht absolut kongruent zur ersten Aufnahmeposition. Daher ergeben sich z.T. Abweichungen in der Größenrelation von einigen Pixeln. Abbildung 6.1a und Abbildung 6.1b zeigen

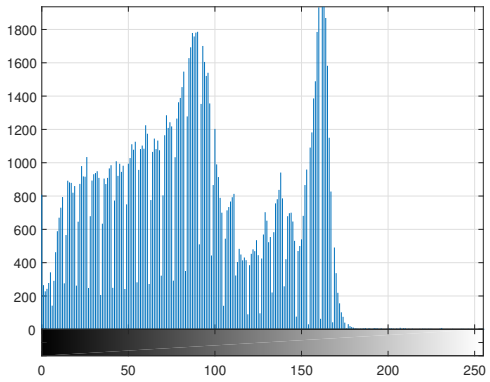
jeweils einen beispielhaften Frame (begrenzt auf die ROI) der jeweiligen Testsequenzen. Abbildung 6.1c und Abbildung 6.1d visualisieren zudem die Lichtverhältnisse der Aufnahmen anhand von Histogrammen für das Grauwertbild des Bildausschnitts. Analysiert wird hier nur die für den Algorithmus relevante ROI. Dabei bestätigt der Vergleich der Histogramme die Vermutung, dass die Aufnahmen im Winter eine weniger starke Ausleuchtung (wenig bis keine Pixel im Bereich der Grauwerte über 200) aufweisen. Über die Schattenbildung weist die Sommeraufnahme wesentlich mehr Pixel im Bereich der Grauwerte nahe 0 auf. Insgesamt können signifikant abweichende Lichtverhältnisse der Aufnahmen durch den Vergleich bestätigt werden.

(a) Beispielhafter Ausschnitt (ROI) der Sequenz SAM_9128.mp4

(b) Beispielhafter Ausschnitt (ROI) der Sequenz SAM_9130.mp4



(c) Histogramm des Grauwertbilds des Bildausschnitts (a)



(d) Histogramm des Grauwertbilds des Bildausschnitts (b)

Abbildung 6.1.: Beispielhafte Ausschnitte (ROI) und korrespondierende Histogramme der Grauwerte für die beiden untersuchten Videosequenzen

Hinsichtlich der Parametrisierung des Algorithmus gelten für die nachfolgenden Tests folgende Einstellungen:

- SVM mit rechts passierenden Radfahrern trainiert: `cyclist_detector_r.yml`
- Empfindlichkeit der Detektion: `hitThreshold = 0.1`
- Maximale Entfernung von Objekt und Kalman-Filter: `seperationDistance = 80`
- Iterationen eines Kalman-Filters ohne Objektzuweisung: `missingFrameThreshold = 10`

6.1.1. Analyse der Erkennungsrate der ersten Testvideosequenz

Tabelle 6.1 zeigt die Ergebnisse der Untersuchungen der Testvideosequenz SAM_9128.mp4. Von den rechts passierenden Radfahrern werden lediglich zwei nicht erkannt. Erwähnt sei jedoch, dass einer dieser Nichterkennungen auf eine zu große Verdeckung (größer als 80 %) des Radfahrers durch Objekte (Personen) im Vordergrund zurückzuführen ist. Dieser Umstand zeigt in jedem Fall eine Grenze des Algorithmus bzw. der Rahmenbedingung der Aufnahme auf. So kann eine Klassifizierung nur unter ausreichend guten Sichtverhältnissen, also einer möglichst ungehinderten Sichtverbindung (engl.: Line-of-sight (LOS)), gewährleistet werden. Steigt der Grad der Verdeckung, wird das Ergebnis der SVM wesentlich ungenauer und führt zu einem Wert, der eine negativ Klassifizierung begünstigt. Die zwei doppelt gezählten Radfahrer können ebenfalls durch den Umstand der Objektverdeckung erklärt werden. Wird das Objekt jeweils an den Rändern der ROI erkannt, im Zentrum jedoch verdeckt, wird der Zähler für eine ausstehende Detektion (`missingFrameThreshold`) hochgezählt, bis das Objekt vom Kalman-Filter freigegeben wird. Durch die erneute Detektion am Randbereich wird dasselbe Objekt dann erneut gezählt. In einem zweiten Test wurde der `missingFrameThreshold = 20` gesetzt. Dadurch konnten die doppelt gezählten Objekte ausgeschlossen werden.

	Rechts passierend	**Links passierend**
Anzahl Radfahrer	37	33
Erkannte Radfahrer	35	30
Erkennungsrate	94,6 %	90,9 %
Nicht erkannt	2	3
Doppelt erkannt	2	2
False Positive	0	0

Gesamtanzahl Frames:	10963	Laufzeit:	00:06:05

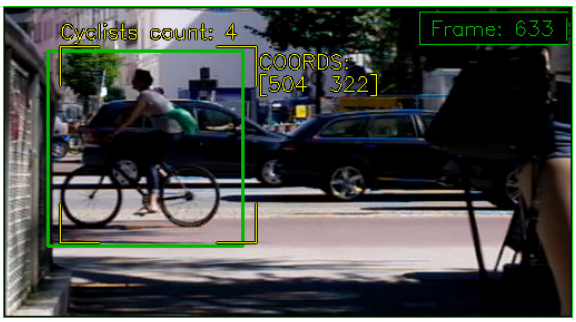
Tabelle 6.1.: Untersuchungsergebnis der Testvideosequenz SAM_9128.mp4

Für die links passierenden Radfahrer können ähnliche Werte festgestellt werden. Die Erkennungsrate liegt bei 90,9 %. Von den drei Radfahren ist einer überdeckt, die weiteren wurden nicht erkannt, da sie nur einige Frames im Randbereich der ROI sichtbar waren. Hier ist zu erwähnen, dass die SVM nur mit rechts passierenden Samples trainiert wurde. Der Test zeigt die bereits hohe Zuverlässigkeit des Algorithmus. Links passierende Radfahrer werden jedoch weniger robust erkannt und begünstigen so nicht erkannte Objekte. Dabei kann postuliert werden, dass die Robustheit durch einen höheren Trainingsdatensatz gesteigert werden kann. Anders als bei den rechts passierenden Radfahrern sind die mehrfach erkannten Objekte hier nicht durch Verdeckung zu erklären.

Durch den bereits erwähnten Mangel an Robustheit der Detektion kann es dazu kommen, dass ein Objekt zunächst nur für einen Frame erkannt wird, in den folgenden Frames jedoch nicht. Der Kalman-Filter hat in diesem Fall zu wenig Messdaten, um eine korrekte Verfolgung zu gewährleisten. Wird das Objekt einige Frames weiter wieder erkannt, fällt die Abfrage bezüglich der Entfernung zum bereits initialisierten Kalman-Filter zu Ungunsten der korrekten Zuweisung aus. In diesem Fall wird angenommen es handelt sich um einen unbekannten Radfahrer. Ein weiterer Kalman-Filter wird initialisiert und dem Objekt zugewiesen. Da eine Erhöhung der `seperationDistance` eine Erkennung von mehreren nah beieinander fahrenden Objekten ausschließen würde, kann hier nicht durch eine einfache Neuparametrisierung Abhilfe geschaffen werden.

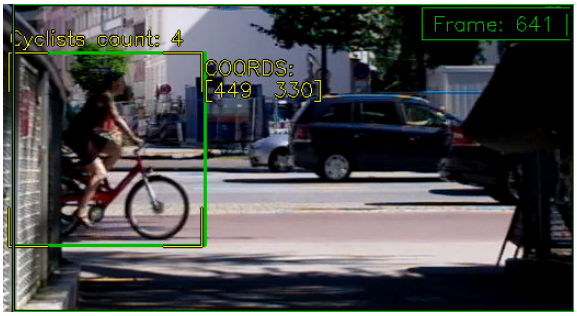
Die Videosequenz beinhaltet einen weiteren Spezialfall, der eine Grenze der eingesetzten Objektverfolgung aufzeigt. Abbildung 6.2 zeigt diesen Fall anhand von vier Teilausschnitten. Im ersten wird zunächst ein links passierendes Objekt erfasst und korrekt mit dem zugewiesenen Kalman-Filter verfolgt. Im Randbereich der ROI misslingt dann die Detektion (vgl. Abbildung 6.2b), das Filter geht in den Voraussagemodus über und bewegt sich weiter zum linken Bildbereich. Unmittelbar beim Verlassen des links passierenden Objekts tritt eine rechts passierende Radfahrerin in die ROI. Das neue Objekt wird nun fälschlicherweise dem bereits angelegten Kalman-Filter zugewiesen, da die Abfrage auf die `seperationDistance` fehlinterpretiert wird. Um solch einen Speziallfall abzufangen, müsste ein weiteres Attribut berücksichtigt werden. Als Beispiel könnte die Richtung des Geschwindigkeitswerts abgefragt werden, wobei bei unzureichend erfassten Objekten (Detektion erfolgt nur in einem einzelnen Frame) diese Information ebenfalls fehlinterpretiert werden kann.

Insgesamt ist hervorzuheben, dass innerhalb der 10963 Frames keine False Positives erkannt wurden, also keine Falschklassifizierung eines Teilausschnitts festzustellen ist.

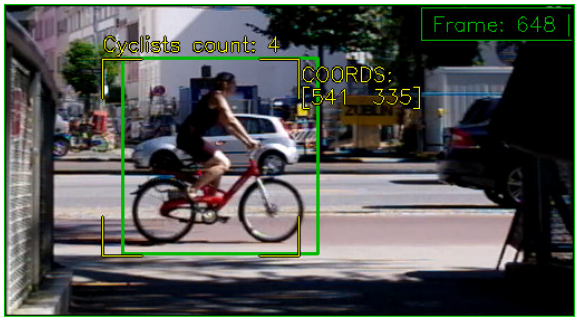


(a) Links passierendes Objekt korrekt zugewiesen.

(b) Detektion des Objekts misslingt, Filter in Voraussagemodus



(c) rechts passierendes Objekt fehlerhaft zugewiesen.



(d) Alter Kalman-Filter konvergiert auf das neue Objekt.

Abbildung 6.2.: Fehlfunktion der Objektverfolgung eines Speziallfalls im Randbereich der ROI

6.1.2. Analyse der 2. Videosequenz mit abweichenden Lichtverhältnissen

Die Ergebnisse der Untersuchung des Sommervideos können überwiegend als positiv bewertet werden. Für den zweiten Test unter anderen Lichtverhältnissen zeigt Tabelle 6.2 das Untersuchungsergebnis der Testvideosequenz SAM_9130.mp4. Für die Erkennungsrate der Objekte wurden auch hier gute Werte ermittelt. Insgesamt traten jedoch mehr fehlerbehaftete Detektionen im Vergleich zum Sommervideo auf. Die drei nicht erkannten rechts passierenden Radfahrer können durch die abweichende Größenrelation des untersuchten Bildausschnitts, durch eine leicht abweichende Kamerapositionierung, erklärt werden. In einem Fall ist zudem eine fast vollständige Objektverdeckung durch einen im Vordergrund fahrenden weiteren Radfahrer der Grund der Nichterkennung.

	Rechts passierend		Links passierend
Anzahl Radfahrer	23		25
Erkannte Radfahrer	20		24
Erkennungsrate	86,96 %		96,00 %
Nicht erkannt	3		1
Doppelt erkannt	2		7
False Positive	3		1
Gesamtanzahl Frames:	17873	Laufzeit:	00:09:55

Tabelle 6.2.: Untersuchungsergebnis der Testvideosequenz SAM_9130.mp4

Bei den insgesamt vier False Positives der Detektion handelt es sich um LKWs bzw. die hinteren Reifen, welche ähnliche Proportionen aufweisen wie die die eines Radfahrers. Abbildung 6.3 zeigt ein Beispiel für eine solche Falschklassifizierung. Das Ergebnis der SVM fällt hier zu Ungunsten der korrekten Klassifikation aus. Diese Falschdetektionen können allerdings ausgeschlossen werden, indem die SVM unter expliziter Angabe der False Positives neu antrainiert wird (Hard Negative Training wie bereits in Abschnitt 6.1 erwähnt).

Abbildung 6.3.: Falschklassifizierung für einen Teilbereich eines LKWs

Die potentiellen Gründe für eine doppelte Detektion eines Radfahrers wurden bereits in Unterabschnitt 6.1.1 angesprochen. In der Untersuchung des Wintervideos fällt auf, dass die Detektion vor allem bei den links passierenden Radfahrern häufig nur für vereinzelte Frames gelingt. Hinsichtlich der geringen Anzahl an Trainingsdaten ist das Ergebnis der Zählung überaus positiv einzuschätzen. Da die SVM jedoch nur mit rechts passierenden Radfahrern trainiert wurde, zeigt sich in diesem zweiten Testvideo die Schwäche bezüglich der Robustheit unter Einsatz des Algorithmus in abweichenden Lichtverhältnissen. Eine Steigerung

kann potentiell durch einen größeren und variationsreicheren Trainingsdatensatz erreicht werden. Ein weiterer Lösungsansatz wäre zudem eine mehrstufige Klassifikation, welche zwischen den rechts und links passierenden Radfahrern unterscheiden kann.

6.2. Auswertung der Echtzeitfähigkeit

Die Bewertung der generellen Echtzeitfähigkeit findet durch eine Untersuchung des Algorithmus auf einem Windows 10 Desktop-PC statt. Vorab sei erwähnt, dass es sich bei dem Betriebssystem Windows 10 nicht um ein Echtzeitbetriebssystem (engl.: Real-Time Operating System (RTOS)) handelt und die Zuweisung von Ressourcen über einen Scheduler nach Prozesspriorität erfolgt (vgl. [Mic17]). Um eine möglichst schnelle Rechenzeit zu gewährleisten, wird daher dem Prozess über den Kommandozeilenaufruf `wmic process where name="CyclistCamCount.exe"CALL setpriority "realtime"` die höchstmögliche Priorität zugewiesen. Da diese Prioritätenklasse jeder anderen übergeordnet wird und bei Programmfehlern schnell zu Problemen führen kann, muss der Aufruf als Administrator erfolgen. Alternativ kann die Priorität eines Prozesses auch im Task Manager geändert werden. Für eine Übersicht der Leistungskenndaten des Rechners sei auf die Tabelle 6.4 im nachfolgenden Abschnitt verwiesen.

Die Laufzeiten variieren durch die dem Algorithmus zugeteilten Ressourcen durch den Scheduler des Betriebssystems. Für die untersuchten Laufzeiten wird daher der Mittelwert über 8192 Frames der Videosequenz SAM_9128.mp4 gebildet. Innerhalb dieser Frames können unterschiedliche Szenarien (mehrere Objekte, teilweise verdeckt, ...) abgedeckt werden. Insgesamt zählt der Algorithmus innerhalb der 8192 Frames 55 Radfahrer.

Zudem sei an dieser Stelle die unter Kapitel 5 bereits genannte Compiler-Einstellung für Release erwähnt. Über die Einstellung *Geschwindigkeit maximieren* `/O2` kann für die Programme eine Laufzeitsteigerung von einem Faktor um etwa 10 erreicht werden. Alle Untersuchungen sind daher unter dieser Konfiguration zu betrachten.

Tabelle 6.3 zeigt die Messergebnisse der Laufzeituntersuchungen der wichtigsten Programmabschnitte.

Bezeichnung des Messwerts	Laufzeit in [ms]
`averageReadingFrameTime`	2,64
`averageDetectionTime`	12,93
`averageTrackingTime`	0,70
`averageFrameToFrameTime`	31,00

Tabelle 6.3.: Untersuchungsergebnis der Laufzeitmessungen der jeweiligen Programmabschnitte

Die Gesamtlaufzeit des Algorithmus ergibt sich aus der Summe der Teilabschnitte *Frame laden und ROI ausschneiden* (`averageReadingFrameTime`), *Merkmalsextraktion und Klassifizierung* (`averageDetectionTime`) und *Objektverfolgung* (`averageTrackingTime`). Die Gesamtlaufzeit beträgt dann 2,64 ms + 12,93 ms + 07 ms = 16,27 ms. Für die Laufzeit zwischen von einem Frame zum nächsten (`averageFrameToFrameTime`) kann eine Zeit von 31,0 ms ermittelt werden. Die verbleibende Differenz 31,0 ms - 16,27 ms = 14,73 ms sind dann Funktionen wie der Visualisierung oder anderen Utility-Funktionen zuzuschreiben. Beispielsweise wartet die Bildanzeige 1 ms auf eine Benutzereingabe, bevor ein neuer Frame geladen wird. Insgesamt berechnet sich die durchschnittliche Framerate dann zu 32,26 FPS, was die Echtzeitfähigkeit des Algorithmus in diesem Versuch bestätigt. Das Video selbst wurde mit einer Framerate von 30 FPS aufgenommen. Der Algorithmus bearbeitet das Video demnach schneller als die Aufnahme selbst getätigt wurde. Wichtig zu erwähnen ist noch, dass die signifikant größte Laufzeit die `averageDetectionTime` ist und dem Sliding Window-Verfahren zuzuschreiben ist. Dieser Abschnitt benötigt mit 41,7 % bereits mehr als ein Drittel der Gesamtzeit `averageFrameToFrameTime`.

Neben den vorgestellten Untersuchungen zeigt Abbildung 6.4 beispielhaft ein Ergebnis der Rechenzeit für die beiden untersuchten Implementationen der NMS. Hier ist die portierte Funktion nach [Tom11] insbesondere bei einer hohen Anzahl an Klassifizierungen, also der Anzahl an Bounding Boxes, wesentlich langsamer als die laufzeitoptimierte Funktion der OpenCV-Bibliothek. Im gegeben Beispiel bearbeitet die Funktion `groupRectangles()` die Filterung der Ergebnismenge in 0,0083 ms und damit um den Faktor 301,2 schneller. Dabei ist die Zentrierung des Ergebnis genauso erfolgreich, ohne dabei auf zusätzliche Gewichtungen der Ergebnisse durch die NMS angewiesen zu sein. Diese Untersuchung bestätigt zumindest teilweise die Vermutung, dass die Funktionen der OpenCV-Bibliothek hinreichend optimiert sind und sich nur mit einem hohen Aufwand noch weiter für das gegebene Problem optimieren lassen.

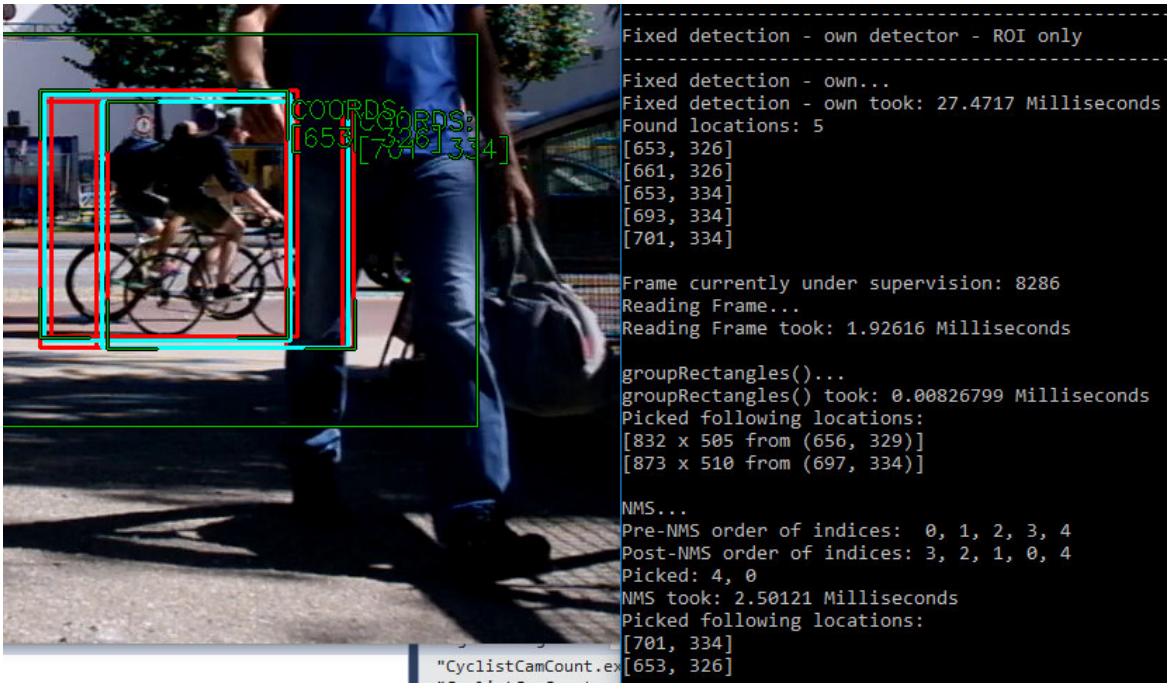


Abbildung 6.4.: Screenshot - Vergleich der Rechenzeit für zwei Ergebnisfilter

6.3. Portierung auf den Einplatinencomputer Raspberry Pi

Nachdem der Algorithmus zunächst auf dem leistungsstarken Desktop-PC verifiziert werden konnte, erfolgt nun die Portierung auf den Einplatinencomputer Raspberry Pi 2 Modell B, welcher bereits in Abschnitt 2.3 vorgestellt wurde. Auch hier handelt es sich mit dem eingesetzten Betriebsystem Raspian 7.8 nicht um ein RTOS. Dem zu untersuchenden Prozess wird daher über den Befehl `sudo renice -n -20 -p [Prozess-ID]` die höchstmögliche Ressourcenpriorität zugewiesen.

Die Gegenüberstellung in Tabelle 6.4 zeigt eine signifikant schwächere Hardware des Einplatinencomputers Raspberry Pi im Vergleich zum Desktop-PC. Im Vergleich zum Desktop-PC handelt es sich bei der CPU hier um einen ARM Cortex-A7 Mikrocontroller mit 32-Bit Architektur. Die 32-Bit Architektur weist eine kürzere Instruktions- und Datenwortbreite als ein 64-Bit System auf und führt so zu einem Anstieg der Rechenzeit durch die zusätzlich benötigte Anzahl an Rechenschritten. Zudem liegt der Takt der Central Processing Unit (CPU) um 25 % unter dem der Intel CPU. Im Vorfeld bestand daher bereits die Vermutung, die Hardware des Raspberry Pi könnte nicht genügen um einen Echtzeitbetrieb zu gewährleisten.

Kenndaten	Desktop-Rechnersystem	Raspberry Pi 2 Modell B
Betriebssystem	Windows 10 64-Bit	
CPU Typ & Architektur	Intel Core i7-4790, 64-Bit	ARM Cortex-A7, 32-Bit
CPU Leistungsangabe	4 x 3,6 GHz	4 x 900 MHz
GPU	Intel HD Graphics 4600	Broadcom Dual Core VideoCore IV
Arbeitsspeicher	16 GB DDR3-SDRAM	1024 MB DDR2-SDRAM

Tabelle 6.4.: Vergleich der wichtigsten Kenndaten der Zielsysteme [Wik17a]

6.3.1. Verifikation der Grundfunktionalität und Analyse der Laufzeitmessung

Abbildung 6.5 zeigt das Ergebnis des Funktionstests des Algorithmus. Für die Detektion eines Radfahrers wurde hier ein Standbild der Videosequenz SAM_9128.mp4 mit der Raspberry Pi Kamera abgefilmt. Die korrekte Detektion verifiziert zunächst die generelle Funktionalität des Algorithmus. Unter der gleicher Parametrisierung wie in den vorangegangenen Tests auf dem Desktoprechner, d.h. mit einer gleichgroßen ROI konnte dabei jedoch lediglich eine durchschnittliche Framerate von 3,16 FPS erzielt werden. Eine derart niedrige Framerate schließt den Echtzeitbetrieb des Systems aus, weshalb die Möglichkeit einer Optimierung erwogen wird.

Der Prozess Manager (vgl. unterer Bereich der Abbildung 6.5) zeigt eine Auslastung der CPU von 97,5 %. Dabei repräsentiert der Wert unter der Spalte `P` den zuletzt verwendeten Kern des Rechners. Die Analyse über einen gewissen Zeitraum bestätigt die Auslastung aller vier Kerne durch den Algorithmus. Da die Ressourcen bereits durch den Scheduler verteilt werden, wird eine mögliche Optimierung durch Ausnutzung von Multi Threading nur als bedingt aussichtsreich eingeschätzt.

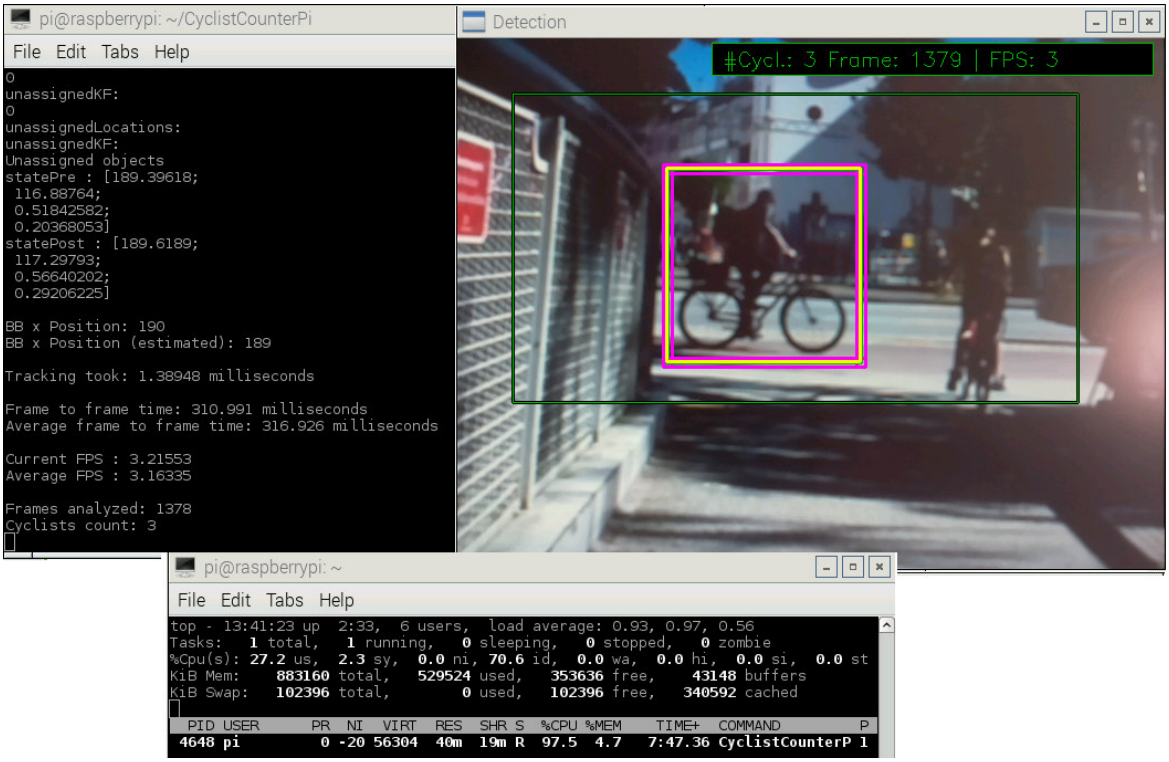


Abbildung 6.5.: Screenshot - Funktionstests des Algorithmus auf dem Einplatinencomputer Raspberry Pi

Im direkten Vergleich durch einen Rendering-Test erzielt die GPU des Raspberry Pi absolute 1252 Punkte, während die Intel GPU 60360 Punkte erzielt [JeG15]. Der geringe Wert kann dadurch erklärt werden das die GPU des Einplatinencomputer lediglich durch einen relativ simples SOC (vgl. [Wik17d]) realisiert ist. Eine Optimierung durch Zuhilfenahme der GPU kann ebenfalls nur bedingt als leistungssteigernd eingeschätzt werden.

Der vorangegangene Test in Abschnitt 6.2 zeigte bereits, dass der rechenlastigste Teil die Berechnung der HOG-Deskriptoren und der Einsatz des Sliding Window-Verfahrens ist. Tabelle 6.5 zeigt die Laufzeit des gesamten Algorithmus und resultierende Framerate für eine Reihe von unterschiedlich großen ROIs. Abbildung 6.6 visualisiert zudem die Messung der resultierenden Framerate für unterschiedliche Breiten der ROI. Festzustellen ist eine exponentiell steigende Laufzeit für eine linear steigende Breite der ROI. Da die Berechnung selbst aber deterministisch ist, d.h. im Vorfeld bekannt ist wie viel Ressourcen für die Multiplikationen benötigt werden, besteht die Möglichkeit einer Leistungssteigerung durch Parallelisierung dieses Rechenabschnitts. Die Parallelisierung kann durch Verwendung eines Field Programmable Gate Array (FPGA)s erreicht werden. Der nachfolgende Abschnitt 6.4 zeigt daher einen Lösungsansatz für eine mögliche Optimierung durch die Portierung auf ein dediziertes Embedded System.

ROI Größe [px]	CPU [%]	Laufzeit [ms]	Framerate [FPS]
176 x 176	77	28,1	34,38
184 x 176	82	31,2	32,46
192 x 176	80	33,1	30,40
200 x 176	85	34,2	29,30
176 x 184	78	31,3	32,25
192 x 184	90	36	28,8
344 x 232	98	127,0	7,90
512 x 280	98	312,0	3,25

Tabelle 6.5.: Laufzeit und resultierende Framerate für verschiedene ROI

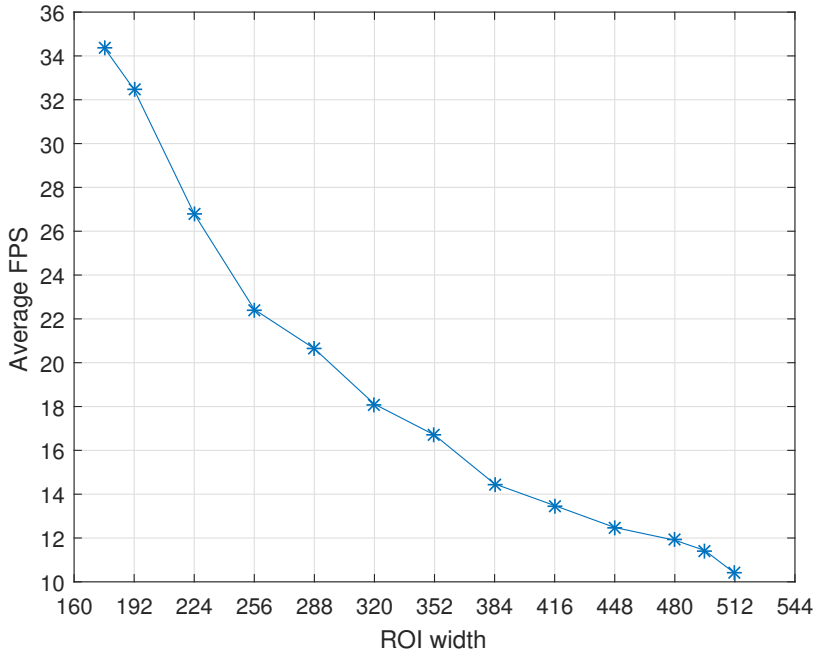


Abbildung 6.6.: Durchschnittliche Framerate für verschiedene Breiten der ROI

6.3.2. Stromverbrauch des Einplatinencomputers im Livebetrieb

Hinsichtlich einer möglichen autarken Installation des Einplatinencomputers soll an dieser Stelle kurz auf den Stromverbrauch des Systems eingegangen werden. Unter der vollen Auslastung durch das realisierte Programm benötigt der Raspberry Pi 440 mAh bis 480 mAh. Die Messung wurde direkt an der Eingangsbuchse des Einplatinencomputers durchgeführt (5 V Versorgungsspannung). Angeschlossen waren dabei ein Monitor über den HDMI-Anschluss, sowie eine Tastatur, eine Maus und ein WLAN-Dongle über USB. Ohne diese Peripherie kann die Stromaufnahme auf 380 mAh bis 420 mAh gesenkt werden. Für den Einsatz als autarkes System steht wie in Abbildung 2.4 gezeigt die Powerbank EC Technology B30224 [EC 16] mit 22400 mAh zur Verfügung. Mit einem geschätzten Mittelwert von etwa 400 mAh, kann das System für etwa 56 Stunden eingesetzt werden. Dabei ist zu Beachten das der Wert mit einer Kapazität von 100 % als Richtwert zu betrachten ist. Unter Realbedingungen liegt diese unter 100 % und führt zu einer geringeren Laufzeit.

6.4. Mögliche Portierung auf ein dediziertes Embedded System

Das vorangegangene Kapitel hat bereits gezeigt das die Portierung auf den Raspberry Pi möglich ist, die Architektur und Leistung des Systems jedoch unzureichend für die Echtzeitfähigkeit des entwickelten Algorithmus. Die Detektionsphase stellt dabei den Flaschenhals bezüglich Ressourcenbedarf dar. Für die Merkmalsextraktion, also das Erstellen eines HOG-Deskriptors, sind viele Multiplikationen nötig und das Sliding-Window-Verfahren führt zu einer exponentiell steigenden Rechenzeit für eine steigende ROI (vgl. Abbildung 6.6). Die Anforderungen an die nötigen Ressourcen für die Multiplikation sind jedoch deterministisch, also vor Ausführung bekannt. Dieser Umstand legt die Entscheidung für eines dedizierten Embedded Systems für den Algorithmus nahe. Die Portierung soll selbst nicht Inhalt dieser Arbeit sein, jedoch soll an dieser Stelle aufgezeigt werden, inwiefern ein FPGA-System ein geeignetes Zielsystem darstellen kann.

Die Wahl fällt auf einen Zynq der 7000er Familie FPGA der Firma Xilinx Inc. [Xil17a], da bereits Expertise an der HAW vorhanden ist. Als Entwicklungsplattform steht das MicroZed 7020 Entwicklungsboard [Xil17b] zur Verfügung, welches bereits kurz in Abbildung 1.2 dargestellt wurde. Der Zynq besitzt zwei ARM Cortex-A9 Prozessoren, dessen Programmierung in den Hochsprachen C und C++ auf einen hohen Abstraktionslevel effizient erledigt werden kann. Die rechenintensiven Pixeloperationen der HOG-Detektion kann dann auf den FPGA-Teil des SOC ausgegliedert werden. Ein weiterer entscheidender Punkt ist die Unterstützung

von OpenCV durch die Entwicklungsumgebung Vivado des Herstellers Xilinx. Über den Einsatz der sogenannten High-level Synthesis (HLS) kann der mit C++ entwickelte Programmcode mithilfe der angebotenen Tools von Vivado für den FPGA-Teil synthetisiert werden. Über den internen Advanced eXtensible Interface (AXI)-Bus der Advanced Microcontroller Bus Architecture (AMBA) kommunizieren ARM-Prozessor und Logikteil dann miteinander und stellen so ein leistungsstarkes und effizientes System. Abbildung 6.7 zeigt schematisch die beschriebene Aufgabenteilung auf einem solchen SOC. Für eine zukünftige Entwicklungsarbeit sei auf die Einführung [NLW15] verwiesen.

Neben den Kernfunktionen des Chips (vgl. [Xil17a]) selbst besitzt das MicroZed Entwicklungsboard weitere die Entwicklung unterstützende Peripherie (vgl. [Xil17b]). Unter anderem wird eine USB 2.0 - Schnittstelle bereit gestellt, mit der z.B. eine Webcam angeschlossen werden kann, um Liveaufnahmen zu machen. Die Stromversorgung über einen Standard 5 V Micro USB-Anschluss macht, wie beim Raspberry Pi, einen Einsatz eines autarken Systems über beispielsweise einen Akku möglich.

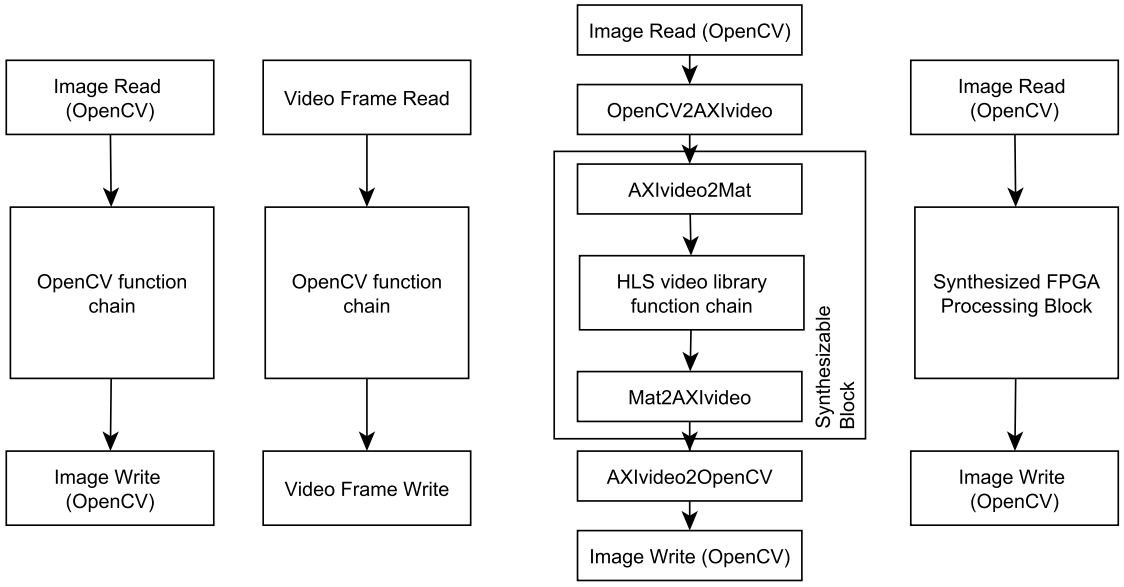


Abbildung 6.7.: Schematische Darstellung der Aufgabenteilung auf dem Zynq SOC [NLW15]

7. Zusammenfassung und Ausblick

Hervorzuheben ist zunächst, dass das die unter den Anforderungen gestellten Ziele erreicht werden konnten. Für die Zählung unter den gleichen Bedingungen wie der Trainingsphase konnten Erkennungsraten von über 90 % erzielt werden. Bemerkenswert ist, dass für das Training eine im Vergleich zu bekannten Datensätzen lediglich 129 Samples verwendet wurden (vgl. Kapitel 4). Dabei erkennt der Algorithmus auch links passierende Radfahrer relativ gut, obgleich nur mit rechts passierenden Samples trainiert wurde.

Für die Laufzeit des Hauptprogramms konnte festgestellt werden, dass bei Verwendung eines leistungsstarken Rechners eine Echtzeitfähigkeit von 30 FPS grundsätzlich gegeben ist, sofern eine Begrenzung des zu untersuchenden Bereichs eingesetzt wird. Der Einplatinencomputer Raspberry Pi hingegen verfügt nicht über genügend Ressourcen und kann nur unter stark eingrenzender ROI eine Framerate von 30 FPS gewährleisten. Diese Eingrenzung berücksichtigt einen zu geringen Ausschnitt des Bildbereichs, wodurch ein effektiver Einsatz unter Realbedingen nicht gewährleistet ist.

Die Erkennungsrate des Algorithmus wird vor allem durch die Sichtverhältnisse bedingt. Für eine Verbesserung ist daher ein Aufbau denkbar, welcher eine Kameraaufnahme in Vogelperspektive ermöglicht. Eine mögliche Verdeckung der Objekte wird damit nahezu ausgeschlossen. In diesem Fall müsste ein potentieller Klassifikator wie eine SVM mit einem neuen Datensatz antrainiert werden. Vorausgesetzt wird außerdem, dass das Hinzufügen von Trainingsdaten die Robustheit des Algorithmus steigern kann. Der Einsatz unter leicht abweichenden Lichtverhältnissen zeigte, dass der Algorithmus nicht ohne Einbuße der Erkennungsrate auf ein generelles Szenario anwendbar ist. Auch hier kann postuliert werden, dass eine größere Datenmenge eine flexiblere Einsatzmöglichkeit ermöglicht. Unter Einbuchung der Laufzeit kann zudem eine den Bildausschnitt skalierende Funktion (engl.: Multi-Scale Detection) eingesetzt werden, um Größendifferenzen auszugleichen.

Für eine Verbesserung der Laufzeit kann vor dem SVM-Klassifikator auf eine Vorentscheidungen gesetzt werden. Wie in Abschnitt 2.2 erwähnt, können beispielsweise binäre Entscheidungsbäume für eine Vorauswahl eingesetzt werden. Eine ähnliche Implementierung könnte im Rahmen nachfolgender Arbeiten die Laufzeit des Algorithmus senken. Des weiteren ist auch der Einsatz einer Hauptkomponentenanalyse (engl.: PCA) denkbar. Diese reduziert die Dimensionalität des Merkmalsraums, indem signifikante Werte des HOG-

Deskriptors ermittelt und nur diese zur Klassifikation berücksichtigt werden. Eine auf den neuen Merkmalsraum angepasste SVM kann dadurch potentiell Rechenzeit einsparen.

Die Verifikation des Algorithmus zeigte einige Spezialfälle, bei denen die korrekte Zählung der Radfahrer misslingt. Beispielsweise kann die Implementierung des Kalman-Filters an seine Grenze gelangen, wenn ein verfolgtes Objekt ein neu auftauchendes Objekt im Randbereich kreuzt. Das Kalman-Filter weist dann das neue Objekt dem alten Systemzustand des Filters zu, ohne zu registrieren, dass es sich um einen neuen Radfahrer handelt (vgl. Abbildung 6.2 in Unterabschnitt 6.1.1). Solche Spezialfälle müssen zunächst durch manuelle Tests ermittelt werden und anschließend ggf. berücksichtigt oder durch Hinzufügen einer Berücksichtigung neuer Attribute ausschlossen werden.

Wie bereits erwähnt zeigte sich, dass die Hardware des Einplatinencomputers ohne grundlegende Modifikation des Algorithmus nicht für ein echtzeitfähiges System ausreicht. Abschnitt 6.4 zeigt den Lösungsansatz auf, ein dediziertes Embedded System zu verwenden. Dieses kann den rechenintensiven Teil für die Berechnung der HOG-Deskriptoren durch Verwendung eines FPGA parallelisieren. Denkbar ist dann zudem ein autarker Einsatz unter Bereitstellung eines wetterfesten Gehäuses sowie einer Stromversorgung durch ein Solarpanel.

Literaturverzeichnis

[Avn15] Avnet, Inc. MicroZed Produktseite, 2015. URL: http://zedboard.org/product/microzed, [letzter Zugriff am 17.03.2017].

[Bal16] Paul Balzer. Das Kalman Filter einfach erklärt [Teil 2], 2016. URL: http://www.cbcity.de/das-kalman-filter-einfach-erklaert-teil-2, [letzter Zugriff am 12.04.2017].

[BIS16] BIS - Behörde für Inneres und Sport Hamburg. Kurzfassung der Verkehrsunfallstatistik 2015, 2016. URL: http://www.hamburg.de/contentblob/5776060/3d6b814f84177aa6df85e7fb3bb12a90/data/2016-04-12-bis-pm-dl-kurzfassung-2015-1).pdf, [letzter Zugriff am 17.03.2017].

[BIS17] BIS - Behörde für Inneres und Sport Hamburg. Kurzfassung der Verkehrsunfallstatistik 2016, 2017. URL: http://www.hamburg.de/contentblob/8284760/c77016926f661546a82edd6d7d9c7941/data/2017-03-02-bis-pm-dl-kurzfassung-verkehrsunfallbilanz-2016.pdf, [letzter Zugriff am 17.03.2017].

[Bor03] Jaroslav Borovicka. Circle detection using hough transforms. Documentation, Image Processing and Computer Vision, University of Bristol, 2003. URL: http://www.borovicka.org/files/research/bristol/hough-report.pdf, [letzter Zugriff am 28.03.2017].

[BWV15a] BWVI - Behörde für Wirtschaft, Verkehr und Innovation Hamburg. Fortschrittsbericht 2015 Hamburg auf dem Weg zur Fahrradstadt. *Pressearchiv*, 2015. URL: http://www.hamburg.de/pressearchiv-fhh/4538548/2015-06-23-bwvi-fahrradstadt/, [letzter Zugriff am 17.03.2017].

[BWV15b] BWVI - Behörde für Wirtschaft, Verkehr und Innovation Hamburg. Radverkehrsstrategie für Hamburg - Fortschrittsbericht 2015, 2015. URL: http://www.hamburg.de/contentblob/4538022/f80b2806d74a33dba4f404dd319d10ce/data/fortschrittsbericht-2015.pdf, [letzter Zugriff am 17.03.2017].

[CL13] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A Library for Support Vector Machines. Technical report, National Taiwan University, 2013. URL: http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf, [letzter Zugriff am 22.04.2017].

[CST00] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 1 edition, 3 2000.

[CZR05] Erik Cuevas, Daniel Zaldivar, and Raul Rojas. Kalman filter for vision tracking. Technical report, Freie Universität Berlin, 2005. URL: http://www.diss.fu-berlin.de/docs/servlets/MCRFileNodeServlet/FUDOCS_derivate_000000000473/2005_12.pdf, [letzter Zugriff am 18.04.2017].

[Dal06] Navneet Dalal. *Finding People in Images and Videos*. PhD thesis, Institut National Polytechnique De Grenoble, 2006. URL: http://lear.inrialpes.fr/people/dalal/NavneetDalalThesis.pdf, [letzter Zugriff am 17.03.2017].

[DK15] Jörg Dahlkemper and Hans Peter Kölzer. Angewandte industrielle Bildverarbeitung. Script, 2015. Auf Nachfrage erhältlich bei Herrn Prof. Dr.-Ing. Jörg Dahlkemper oder Herrn Prof. Dr.-Ing. Hans Peter Kölzer.

[EC 16] EC Technology LLC. EC Technology Portable 2nd Gen Deluxe 22400mAh 3 USB Power Bank, 2016. URL: http://www.iectechnology.com/product/portable-2nd-gen-deluxe-22400mah-3-usb-power-bank-blackred.html, [letzter Zugriff am 20.04.2017].

[Eco17] Eco-Counter GmbH Deutschand. Eco-Counter - ZELT Urban, 2017. URL: http://www.eco-compteur.com/de/produkte/zelt-produkte/urban-zelt, [letzter Zugriff am 21.03.2017].

[GLSU13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[JeG15] JeGX - Geeks3D Blog. Quick Benchmark of the Raspberry Pi 2 GPU (VideoCore IV), 2015. URL: http://www.geeks3d.com/20150603/quick-benchmark-of-the-raspberry-pi-2-gpu-videocore-iv/, [letzter Zugriff am 12.04.2017].

[Mal16] Satya Mallick. Learn OpenCV - Histogram of Oriented Gradients, 2016. URL: http://www.learnopencv.com/histogram-of-oriented-gradients/, [letzter Zugriff am 23.04.2017].

[MGE11] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A. Efros. Ensemble of exemplar-svms for object detection and beyond. In *ICCV*, 2011.

[Mic17] Microsoft. SetPriority method of the Win32_Process class, 2017. URL: https://msdn.microsoft.com/en-us/library/aa393587(v=vs.85).aspx, [letzter Zugriff am 12.04.2017].

[NDR15a] NDR. Rot-Grün will mehr Fahrradzählstationen. *Pressearchiv*, 2015. URL: http://www.ndr.de/nachrichten/hamburg/Rot-Gruen-will-mehr-Fahrradzaehlstationen,fahrradzaehlstation102.html, [letzter Zugriff am 21.03.2017].

[NDR15b] NDR. Steuerzahlerbund prangert Geldverschwendung an. *Pressearchiv*, 2015. URL: http://www.ndr.de/nachrichten/hamburg/Steuerzahlerbund-prangert-Geldverschwendung-an,schwarzbuch186.html, [letzter Zugriff am 21.03.2017].

[NDR15c] NDR extra 3. Realer Irrsinn: Fahrradzählstation in Hamburg. *Mediathek*, 2015. URL: http://www.ndr.de/fernsehen/sendungen/extra_3/Realer-Irrsinn-Fahrradzaehlstation-in-Hamburg,extra10190.html, [letzter Zugriff am 21.03.2017].

[NLW15] Stephen Neuendorffer, Thomas Li, and Devin Wang. *Accelerating OpenCV Applications with Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries*. Xilinx Inc., 2015. URL: https://www.xilinx.com/support/documentation/application_notes/xapp1167.pdf, [letzter Zugriff am 03.04.2017].

[Ope14a] OpenCV dev team. Introduction to Support Vector Machines, 2014. URL: http://docs.opencv.org/3.0-beta/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html, [letzter Zugriff am 21.03.2017].

[Ope14b] OpenCV dev team. Support Vector Machines, 2014. URL: http://docs.opencv.org/3.0-beta/modules/ml/doc/support_vector_machines.html, [letzter Zugriff am 21.04.2017].

[Pri96] Sarah Price. Edges: The Canny Edge Detector . Technical report, Institute for Computer Based Learning - Heriot-Watt University, 1996. URL: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/edges/canny.htm, [letzter Zugriff am 18.04.2017].

[RAS17a] RASPBERRY PI FOUNDATION. CAMERA MODULE V2, 2017. URL: https://www.raspberrypi.org/products/camera-module-v2/, [letzter Zugriff am 12.04.2017].

[RAS17b] RASPBERRY PI FOUNDATION. RASPBERRY PI 2 MODEL B, 2017. URL: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/, [letzter Zugriff am 12.04.2017].

[Sch05] Gregor Schwarzenberg. Objektverfolgung mit Partikel-Filtern. Diplomarbeit, Universität Karlsruhe, 2005. URL: http://publica.fraunhofer.de/documents/N-53669.html, [letzter Zugriff am 25.03.2017].

[St.14] St. Pedali - Der Fahrrad-Blog aus Hamburg. Skandal: Fahrradbarometer an der Alster zählt falsch und übertreibt. *Blogeintrag*, 2014. URL: http://st-pedali.blogspot.de/2014/12/skandal-fahrradbarometer-der-alster.html, [letzter Zugriff am 21.03.2017].

[TL15] Wei Tian and Martin Lauer. Fast Cyclist Detection by Cascaded Detector and Geometric Constraint. Technical report, Karlsruhe Institute of Technology, 2015. URL: http://www.mrt.kit.edu/z/publ/download/Fast_Cyclist_Detection_by_Cascaded_Detector_and_Geometric_Constraint_tian.pdf, [letzter Zugriff am 21.03.2017].

[Tom11] Tomasz Malisiewicz. blazing fast nms (non-maximum suppression), 2011. URL: https://gist.github.com/quantombone/1144423#file-nms_fast-m, [letzter Zugriff am 10.04.2017].

[Tör16] Isac Törnberg. Real time object tracking on Raspberry Pi 2. Bachelorthesis, KTH - Skolan för industriell teknik och management, 2016. URL: http://www.diva-portal.org/smash/get/diva2:957920/FULLTEXT01.pdf, [letzter Zugriff am 25.03.2017].

[WB06] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical report, University of North Carolina, 2006. URL: https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf, [letzter Zugriff am 18.04.2017].

[Wik17a] Wikipedia - die freie Enzyklopädie. Raspberry Pi - Hardware, 2017. URL: https://de.wikipedia.org/wiki/Raspberry_Pi#Hardware, [letzter Zugriff am 12.04.2017].

[Wik17b] Wikipedia - The Free Encyclopedia. Kalman filter, 2017. URL: https://en.wikipedia.org/wiki/Kalman_filter, [letzter Zugriff am 20.04.2017].

[Wik17c] Wikipedia - The Free Encyclopedia. Minimum bounding box, 2017. URL: https://en.wikipedia.org/wiki/Minimum_bounding_box, [letzter Zugriff am 20.04.2017].

[Wik17d] Wikipedia - The Free Encyclopedia. VideoCore, 2017. URL: https://en.wikipedia.org/wiki/VideoCore, [letzter Zugriff am 20.04.2017].

[Wol17] Wolfram Research, Inc. L2-Norm, 2017. URL: http://mathworld.wolfram.com/L2-Norm.html, [letzter Zugriff am 23.04.2017].

[Xil17a] Xilinx, Inc. All Programmable SoC with Hardware and Software Programmability, 2017. URL: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html#productTable, [letzter Zugriff am 28.03.2017].

[Xil17b] Xilinx, Inc. MicroZed Evaluation Kit, 2017. URL: https://www.avnet.com/opasdata/d120001/medias/docus/178/PDP-AES-Z7MB-7Z010-G-V1.pdf, [letzter Zugriff am 28.03.2017].

[ZAYC06] Qiang Zhu, Shai Avidan, Mei-Chen Yeh, and Kwang-Ting Cheng. Fast Cyclist Detection by Cascaded Detector and Geometric Constraint. Technical report, University of California, 2006. URL: http://www.eng.tau.ac.il/~avidan/papers/IntegralHoG.pdf, [letzter Zugriff am 18.04.2017].

[ZM10] Chengbin Zeng and Huadong Ma. Robust Head-shoulder Detection by PCA-Based Multilevel HOG-LBP Detector for People Counting. Technical report, Beijing University of Posts and Telecommunications, 2010. Zugriff über IEEE Xplore, [letzter Zugriff am 25.03.2017].

Abkürzungsverzeichnis

Bezeichnung	Beschreibung	Seiten
ϵ-SVM	ϵ-Support Vector Regression	25, 36
AMBA	Advanced Microcontroller Bus Architecture	55
ARM	Advanced RISC Machines	50, 51, 54, 55
AXI	Advanced eXtensible Interface	55
BIS	Behörde für Inneres und Sport	9
BV	Bildverarbeitung	11
BWVI	Behörde für Wirtschaft, Verkehr und Innovation	9
CD	Compact Disc	30, 33, 37, 66
CPU	Central Processing Unit	50, 51
FPGA	Field Programmable Gate Array	52, 54, 55, 57
FPS	Frames per Second	14, 15, 49, 51, 53, 56, 70
GCC	GNU Compiler Collection	20, 33
GPS	Global Positioning System	66
GPU	Graphics Processing Unit	51, 52
HAW	Hochschule für angewandte Wissenschaften	19, 20, 54

Bezeichnung	**Beschreibung**	**Seiten**
HDMI	High Definition Multimedia Interface	54
HLS	High-level Synthesis	55
HOG	Histogram of Oriented Gradients	7, 16, 17, 20–24, 30, 32, 33, 35, 37, 38, 52, 54, 56, 57
INRIA	Institut national de recherche en informatique et en automatique	30
Kfz	Kraftfahrzeug	9, 28, 30, 32
KIT	Karlsruhe Institute of Technology	17, 30
LBP	Local Binary Patterns	17
LKW	Lastkraftwagen	7, 47
LOS	Line-of-sight	44
NMS	Non-Maximum Suppression	21, 26, 38, 49
PC	Personal Computer	11, 15, 42, 48, 50
PCA	Principal Component Analysis	17, 56
ROI	Region of Interest	6, 7, 15, 18, 19, 30, 32, 33, 37, 43–46, 49, 51–54, 56

Bezeichnung	**Beschreibung**	**Seiten**
RTOS	Real-Time Operating System	48, 50
SOC	System On Chip	8, 52, 54, 55
SVM	Support Vector Machine	16–18, 21, 24, 25, 30, 33, 35–38, 42, 44, 45, 47, 56, 57, 66
TTIC	Toyota Technological Institute at Chicago	17, 30
USB	Universal Serial Bus	54, 55
WLAN	Wireless Local Area Network	54

A. Anhang

A.1. Inhalt Datenträger

An dieser Stelle folgt eine Übersicht über die Inhalte der der Arbeit beigelegten CD. Der Datenträger kann auf Nachfrage bei Erst- und Zweitprüfer dieser Thesis eingesehen werden.

- **1_PDF** - Enthält die Masterthesis abgespeichert als .pdf-Datei.
- **2_Quellcode** - Enthält den C++-Quellcode für das Hauptprogramm `CyclistCamCount.cpp`, sowie das Hilfstool `SampleGrabber.cpp` zum Ausschneiden von Testsamples und das Programm `TrainCyclistSVM.cpp` zum Antrainieren der SVM. Eine modifizierte Version des Hauptprogramms `CyclistCamCount_PI.cpp` für die Portierung auf den Raspberry Pi ist ebenfalls beigefügt.
- **3_Matlab** - Enthält den für die Vorabuntersuchung verwendeten Matlab-Code `CyclistDetectionHoughCirc.m` und ein Hilfstool `NameChanger.m` für die Namensänderung von Testsamples
- **4_Datensatz** enthält den im Rahmen der Arbeit erstellten Datensatz. Dieser beinhaltet die Videosequenzen und verwendeten Testsamples (vgl. Kapitel 4). Des weiteren sind die Support Vektoren der SVM beigelt.

A.2. Zusatzinformationen zu den Trainings- und Testaufnahmen

Für den Aufnahmestandort wurde die Brücke zur Gurlitt-Insel gewählt. Der Standort in Nähe der Zählsäule bietet einen guten Vergleich zu einem möglichen Einsatzszenario des Zählsystems. Abbildung A.1 zeigt den Aufnahmestandort im Hintergrund der Zählsäule und den Stativaufbau für die Videoaufnahmen. Die GPS-Koordinaten für den Ort der Aufnahmen lauten wie folgt:

Breitengrad: 53.559115
Längengrad: 10.008068

Für die Aufnahmen wurde eine Samsung NX 11 mit 14.6 Megapixel verwendet. Die Einstellungen der Kamera lauten wie folgt: Manueller Fokus, Belichtung Zentrum, Auflösung 1280x720, Hohe Qualität (Bildsensor), Blendregler Off.

(a) Blick auf die Brücke (Aufnahmestandort rot markiert) mit Zählsäule im Vordergrund

(b) Aufnahmestandort auf der Brücke für die Videoaufnahmen mit Kamerastativ

Abbildung A.1.: Fotoaufnahmen des Standorts der Videoaufnahmen für die Trainings- und Testdaten

A.3. Erstellung eines OpenCV-Projekts mit Microsoft Visual Studio 2015

Für mögliche weitere Arbeiten mit OpenCV unter Windows bietet diese Kurzeinführung eine Anleitung zur Erstellung eines OpenCV-Projekts unter Visual Studio. Zunächst sollten sowohl OpenCV[1] als auch Microsoft Visual Studio[2] installiert werden. Im Rahmen der Arbeit wurde hauptsächlich mit Visual Studio 2015 gearbeitet, weshalb diese Version empfohlen wird.

Als nächsten Schritt muss unter Windows eine Umgebungsvariable, für den Ordnerpfad der OpenCV-.dlls angelegt werden:

- Neue Systemvariable anlegen unter **Systemsteuerung** → **System und Sicherheit** → **System**
- Link **Erweiterte Systemeinstellungen** anklicken
- Link **Umgebungsvariablen** anklicken

[1] `http://opencv.org/downloads.html`
[2] `https://www.visualstudio.com/de/downloads/`

- Unter **Path** auf **Bearbeiten...** und dann neuen Pfad zu den OpencV-Dateien `opencv_world310.dll` und `opencv_world310d.dll` angeben
- Beispielpfad: **C:\[OpenCV-Pfad]\opencv\build\x64\vc14\bin**

In Microsoft Visual Studio kann nun ein Projekt angelegt werden:

- **Datei** → **Neu** → **Projekt** → **Win32-Konsolenanwendung** und aussagekräftigen Namen angeben
- **Ok** → **Weiter**, bei zusätzliche Optionen **Leeres Projekt** markieren, dann **Fertig stellen** anklicken

Zuletzt müssen nun noch einige Projekteinstellungen vorgenommen werden:

- **Projekt** → **Einstellungen** auswählen
- Unter **C/C++** → **Allgemein** → **Zusätzliche Includeverzeichnisse** den include-Pfad von OpenCV angeben
- Beispielpfad: **C:\[OpenCV-Pfad]\opencv\build\include**
- Unter **Linker** → **Allgemein** → **Zusätzliche Bibliotheksverzeichnisse** den lib-Pfad von OpenCV angeben
- Beispielpfad: **C:\[OpenCV-Pfad]\opencv\build\x64\vc14\lib**
- Unter **Linker** → **Eingabe** → **Zusätzliche Abhängigkeiten** → **Bearbeiten...** die jeweilige .lib-Datei angeben. Für Debug `opencv_world310d.lib`, für Release `opencv_world310.lib`

A.4. Einrichtung für die OpenCV-Entwicklung in C/C++ auf dem Raspberry Pi

Listing A.1 zeigt die nötigen Befehle, für die Einrichtung von OpenCV unter Linux. Zunächst werden die benötigten Werkzeuge installiert und das OpenCV git-Verzeichnis geklont. Im Anschluss muss OpenCV dann auf dem Zielsystem, also dem Raspberry Pi, kompiliert werden. Dieser Vorgang dauert mehrere Stunden und sollte daher beispielsweise über Nacht ausgeführt werden.

```
sudo apt-get install build-essential cmake git-core libgtk2.0-dev pkg-
    config python-dev python-numpy libavcodec-dev libavformat-dev
    libswscale-dev

mkdir opencv-build
cd opencv-build

git clone https://github.com/Itseez/opencv.git
cd opencv
cmake .
make
sudo make install
```

Listing A.1: Bash-Skript für die Installation und Compilierung von OpenCV unter Linux

Ist OpenCV korrekt eingerichtet, kann nun die `CMakeLists.txt` Datei angelegt werden, um die Projekteinstellungen zu definieren, mit denen kompiliert werden soll. Unter `target\_link\_libraries(...)` können zusätzliche Parameter angegeben werden, wie hier `lr` für die Verwendung von Timing-Funktionen unter Linux.

```
cmake_minimum_required(VERSION 2.8)
project( ProjectName )
find_package( OpenCV REQUIRED )
add_executable( ProjectName ProjectName.cpp )
target_link_libraries( ProjectName ${OpenCV_LIBS} lr)
```

Listing A.2: Auszug der Datei `CMakeLists.txt`

Im selben Ordner kann dann der Befehl `cmake .` ausgeführt werden um eine Makefile zu erstellen, welche anschließend mit dem Befehl `make` ausgeführt werden kann. Damit wird die in `CMakeLists.txt` angegebene Codedatei kompiliert. Das Programm kann nun mit `./ProjectName` ausgeführt werden.

Für die Verwendung der Raspberry Pi Kamera muss diese über `raspi-config` aktiviert werden. Mit dem Befehl `sudo modprobe bcm2835-v4l2` werden zudem die nötigen v4l2-Treiber geladen um die Kamera direkt über OpenCV anzusprechen.

A.5. Screenshots des erstellten Hauptprogramms zur Radfahrerzählung

An dieser Stelle sollen zwei zur Laufzeit aufgenommene Screenshots beispielhaft eine Übersicht über die Funktion des erstellten Hauptprogramms zur Radfahrerzählung geben. Abbildung A.2 zeigt dazu den betrachteten Frame des Sommertestvideos. Gezeigt werden die Nummer des aktuellen Frames 844, die aktuelle Framerate mit 30 FPS, sowie eine Fortschrittsanzeige bezogen auf die Gesamtzahl der Frames der Videosequenz (Bildbereich oben rechts). Im mittleren oberen Bildbereich wird zudem die aktuelle Anzahl an gezählten Radfahrern angegeben. Im betrachteten Frame wird gerade der sechste Radfahrer verfolgt. Die grüne Umrandung steht dabei für die erfolgreiche Detektion. Die gelbe Umrandung zeigt die Verfolgung und die momentane Position des Systemzustands des Objekts an.

Abbildung A.3 zeigt für denselben Frame Nummer 844 die Ausgabe der Messdaten für den jeweiligen Programmabschnitt (Framegewinnung, Detektion und Verfolgung).

Abbildung A.2.: Screenshot - Hauptprogramm mit Visualisierungen der Objektverfolgung während der Untersuchung des Sommervideos

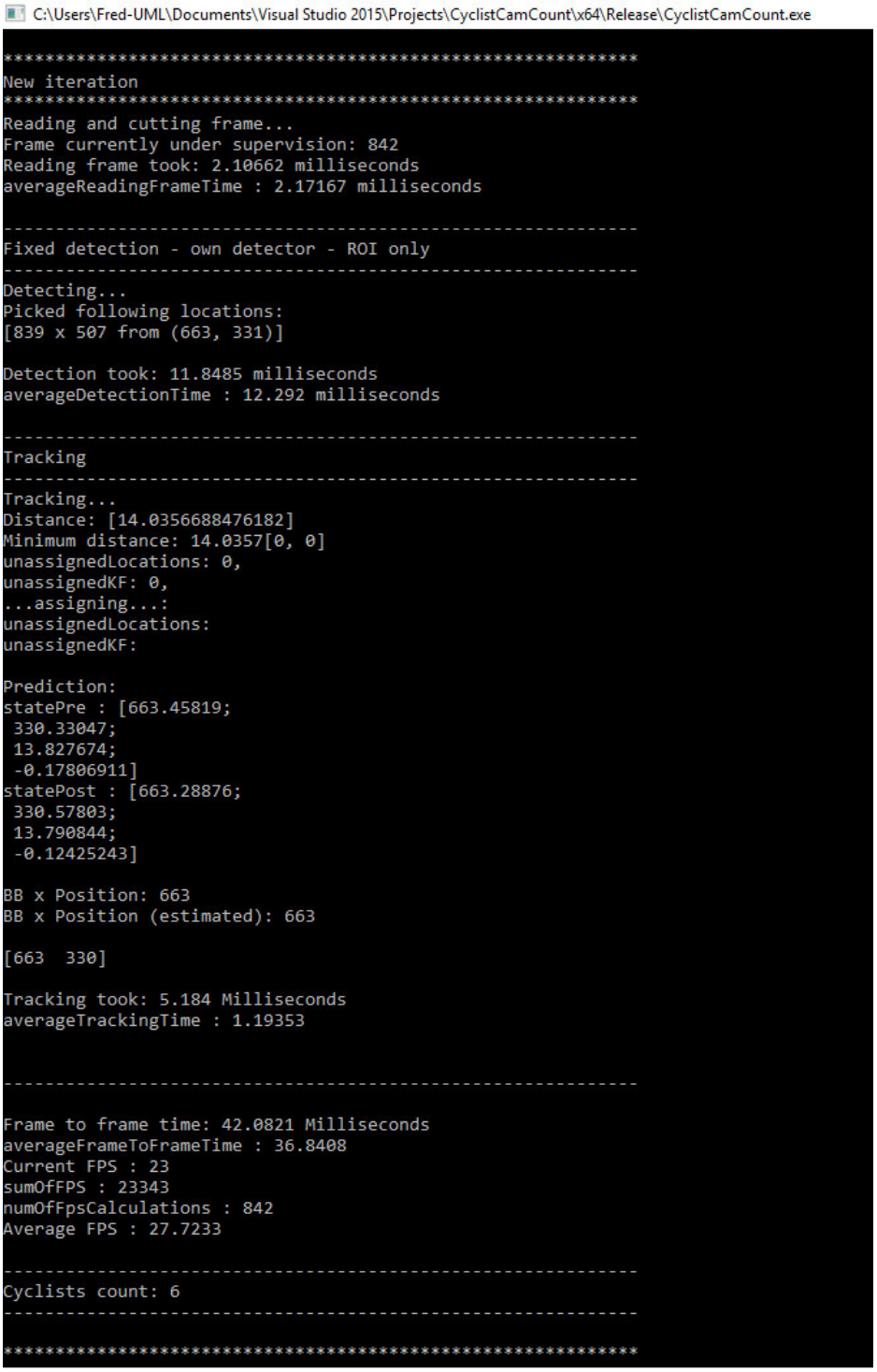


Abbildung A.3.: Screenshot - Messdatenausgabe des Hauptprogramms während der Untersuchung des Sommervideos

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 3. Mai 2017	
Ort, Datum	Unterschrift