Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Institut für Simulation und Grafik

Bachelorarbeit

Computervisualistik

Erstellung einer Annotationssoftware mit Videotracking zum Generieren von Trainingsdaten für Deep-Learning

Autor:

Martin Zettwitz

20. März 2017

Betreuer:

Prof. Dr.-Ing. Bernhard Preim

Institut für Simulation und Grafik

Dr.-Ing. Rene Waldmann

Volkswagen Aktiengesellschaft

Zettwitz, Martin:
Erstellung einer Annotationssoftware mit Videotracking zum Generieren von Trainingsdaten für Deep-Learning
Bachelorarbeit, Otto-von-Guericke-Universität Magdeburg, 2017.

Inhaltsangabe

Maschinelle Lernverfahren, im Speziellen Neuronale Netze für Deep-Learning, benötigen sehr viele Trainingsdaten zum Erzeugen eines akzeptablen Lösungskandidaten. Zum Generieren von Trainingsdaten für Lernverfahren mit visuellen Eingängen (z.B. Bilddaten für automatisiertes Fahren eines KfZ) eignen sich besonders Videodaten. Diese Arbeit untersucht verschiedene Methoden und Software für die Annotation von Videosequenzen. Darauf basierend wird ein neues System mit alternativen Ansätzen entwickelt. Ein besonderer Schwerpunkt liegt hierbei auf der Verwendung von aktuellen Tracking-Algorithmen, wie sie in der Computervision Anwendung finden. Die Tracker werden vor allem verwendet, um ruckartige, nicht-lineare Bewegungen auszugleichen und die Arbeitsgeschwindigkeit beim manuellen Annotieren signifikant zu erhöhen.

Die Ergebnisse, Meinungen und Schlüsse dieser Arbeit sind nicht notwendigerweise die der Volkswagen AG.

Schlüsselwörter: Annotation, Videosequenzen, Tracking, Trainingsdaten

Inhaltsverzeichnis

1. Einführung

Maschinelles Lernen stellt einen aktuell sehr wichtigen interdisziplinären Forschungsschwerpunkt dar. Viele komplexe Probleme, wie z.B. Objekterkennung, Datenanalyse und Mustererkennung, lassen sich erst mit maschinellen Lernverfahren effizient und zufriedenstellend lösen [KBB15]. Je nach Anwendungsgebiet existiert eine Vielzahl von verschiedenen Verfahren, wie z.B. Bayes'sche Netze oder Neuronale Netze. Die Grundintention dieser Arbeit bezieht sich auf Neuronale Netze und im Speziellen auf Deep Learning, kann jedoch für weitere, auf Bilddaten basierende, Verfahren angewendet werden.

Neuronale Netze

Diese Systeme sind dem biologischen Gehirn nachempfunden. Es werden künstliche Neuronen miteinander verknüpft, um ein Informationsaustausch bzw. einen Informationsverknüpfung zu erzielen. Ihr Prinzip beruht auf iterativem Training eines Netzes, um den Lösungskandidaten auf unbekannte Daten anzuwenden. Im einfachsten Fall bestehen diese Netze aus einer Eingabeschicht, (k)einer Verarbeitungsschicht und einer Ausgabeschicht. Je nach Problemstellung können diese unterschiedlich komplex ausfallen. So verwendet man bei bilddatenbasierten Netzen beispielsweise oftmals ein Eingangsneuron pro Pixel.
Verwendet man viele versteckte Schichten, um beispielweise nicht-lineare Probleme (High-level Beziehungen zwischen Low-level Features) lösen zu können, spricht man von Deep Neural Networks (DNNs). Eine besondere Anwendungsform hierbei sind Convolutional Neural Networks (CNNs), welche, ähnlich des Filterkernels in der Bildverarbeitung, eine Faltung (Konvolution) mit mehreren Neuronen berechnen. Anwendungsbeispiele hierfür sind vor allem bildbasierte Objektdetektion und Segmentierung.

Trainingsdaten

Gemeinsam haben viele Lernverfahren, dass ihr Erfolg signifikant von den Trainingsdaten abhängt. Verwendet man zu wenig oder zu schlechte Trainingsdaten, ist das

Netz nur in der Lage, die wenigen vorhandenen Situationen oder Objekte zu erkennen bzw. klassifiziert diese nur ungenügend [ASH12]. Es stellt somit eine sehr wichtige Aufgabe dar, Trainingsdaten in großen Mengen und von guter Qualität zu generieren. Für bildbasierte Verfahren eigenen sich hierfür, durch die hohen Bildraten, vor allem Videosequenzen. Je nach benötigter Genauigkeit müssen die Objekte der Einzelbilder pixelgenau oder mit Bounding Boxen/Polygonen segmentiert und klassifiziert werden.
Ein aktuell wichtiger Schwerpunkt in der angewandten Forschung, unter Verwendung von DNNs, ist das sogenannte Deep Driving. Hierbei sollen Fahrzeuge, durch DNNs, in der Lage sein Fahrverhalten zu erlernen, um autonom fahren zu können. Beispielhaft hierfür ist unter anderem die Prädiktion von Verkehrssituationen, wo Verkehrsteilnehmer im richtigen räumlichen, temporalen und semantischen Kontext zu einander stehen müssen. Da Fahrzeuge mit installierter Kamera selber am Verkehr teilnehmen, ist die Kamera nicht statisch. Es kommt zu ruckartigen Bewegungen in der Szene, z.B. durch Bremsmanöver oder die Kamera selber, ausgelöst durch Fahrbahnunebenheiten. Hieraus resultierend ist ein hoher Arbeitsaufwand durch häufiges, manuelles Korrigieren des Anwenders [KSC13].

Objekttracking

Gängige Praxis bei Annotationssystemen ist bilineare Interpolation zwischen Einzelbildern [VR11, MD03, YRCT09], diese versagen jedoch häufig bei eben genannten Situationen. Abhilfe schaffen hier optische Trackingssysteme, welche ein markiertes Objekt über längere Zeit räumlich verfolgen und somit obige Probleme ausgleichen [KSC13]. Der Anwender soll hierdurch halb-automatisiert unterstützt werden, um den Bearbeitungsaufwand, durch seltenere Korrektur, signifikant zu reduzieren. Im Idealfall müssen Objekte, bei entsprechender Behandlung von Verdeckung, Abwesenheit und Skalierung/Rotation, nur einmalig beim ersten Auftreten markiert und die Annotation(Attribute) angepasst werden.

1.1 Motivation und Zielstellung der Arbeit

In dieser Arbeit werden, unter Berücksichtigung der obigen Situation des Deep Drivings bzw. allgemein dynamischer Kameras, verschiedene Ansätze [Mul10, VR11, RE11] zur Annotation von Videosequenzen verglichen und diskutiert. Im Detail werden vor allem bilineare Interpolation und optisches Tracking [YJS06] als unterstützende Systeme gegenübergestellt. Hierauf basierende Softwarelösungen [VPR13, RE11, YRCT09, MD03, Mul10] werden anhand ihrer Features und Limitierungen bewertet und sich ergebende Probleme erläutert. Eine im Zuge dieser Arbeit entwickelte Softwarelösung mit optischem Trackingansatz und Multithreading wird vorgestellt. Bestehende, in der weitverbreiteten Bibliothek *OpenCV* [Its16] implementierte, Algorithmen des Objekttrackings [KMM12, GGB06, HTS16, KMM10, HCMB15, DKFv14, BYB09] werden anhand von *Geschwindigkeit, Genauigkeit und Robustheit* evaluiert [MYV07, BS08, BER03]. Weiterhin wird der Einsatz von *Parallelisierung* in Form von Multithreading zur Performancesteigerung und Echtzeitfähigkeit der Tracking-Algorithmen betrachtet.

1.2 Gliederung der Arbeit

Im folgenden 2. Kapitel werden verwandte Arbeiten, deren Methoden und Softwarelösungen untersucht. Das 3. Kapitel beschäftigt sich mit den thematischen Grundlagen bezüglich Annotation und optischem Tracking. Hierbei wird auf Intention und Arten der Annotation eingegangen sowie auf deren Möglichkeiten und Restriktionen für den Anwender. Optisches Tracking wird Interpolation als alternative Unterstützung des Anwenders gegenübergestellt. Des Weiteren werden verschiedene Arten von Trackingverfahren vorgestellt und regionenbasierte Tracking-Algorithmen aus OpenCV erläutert. Im 4. Kapitel wird der schematische Entwurf eines Annotationssystems, speziell das User-Interface, die Datenstruktur und der Datenexport beschrieben. Das anschließende 5. Kapitel widmet sich der Implementierung eines Annotationssystems. Hierbei werden das Frontend (GUI) und das Backend (Tracking, Multithreading, Speicher) sowie die verwendeten Bibliotheken im Detail beleuchtet. In Kapitel 6 befindet sich eine Evaluierung der verwendeten Verfahren bezüglich *Tracking-Algorithmen* (Art + Parameter), *Parallelisierung*, sowie eine *Anwenderstudie* über die Arbeitsgeschwindigkeit beim Annotieren. Das 7. Kapitel liefert eine Zusammenfassung der Arbeit sowie einen Ausblick über offene Probleme, mögliche Erweiterungen und zukünftige Perspektiven von Annotationssystemen.

2. Verwandte Arbeiten

Sowohl im Bereich der Annotation, als auch dem Tracking und der Evaluierung von Tracking-Algorithmen wird seit langem geforscht. Bereits in den frühen 90er Jahren, zu den Anfängen der graphischen Benutzeroberflächen, werden Design Empfehlungen für Annotationssysteme ausgesprochen [HB92]. Diese beziehen sich auf Nutzeranforderungen. Hierbei wird auf Interaktionsmöglichkeiten mit den Daten, während des laufenden Videos und der Nachbearbeitung, sowie auf Möglichkeiten der Annotation und den Anforderungen an ein Userinterface eingegangen. Die aufgestellten Empfehlungen werden heute in vielen aktuellen Systemen verwendet.

2.1 Applikationen

Es existieren zahlreiche Annotationssysteme, von denen eine Vielzahl auf Interpolation basiert. Mihalcik et al. stellten mit ViPER[MD03] ein Tool vor, welches neben der Annotation auch eine Evaluierung mit einer Ground Truth vornehmen kann. Es werden verschiedene Interaktionsmöglichlkeiten wie Zoom, Panning und eine Historie unterstützt. Für das Objekttracking wird auf bilineare Interpolation zwischen Keyframes gesetzt, zusätzlich ist Frame-by-Frame Labeling möglich sowie das Verschieben einer Bounding Box zur Laufzeit(live dragging). Das VIA Tool[Mul10] nutzt lediglich Frame-by-Frame Labeling und live-dragging. Das webbasierte Tool LabelMe Video[YRCT09] nutzt komplexe Polygone für eine vergleichsweise genaue Markierung der Objekte. Diese Polygone können in Gruppen zusammengefasst und gemeinsam für einen Zeitabschnitt annotiert werden. Um Objekte in der richtigen räumlichen Reihenfolge anzuordnen, werden Tiefeninformationen generiert, aus welchen sich ein 3D-Model erzeugen lässt. Für eine beschleunigte Annotation wird ein spezieller Interpolationsansatz gewählt, welcher die 2D-Interpolation in 3D simuliert und somit Bewegungen plausibler macht. Gegen leichte Kameraschwankungen findet eine Vorverarbeitung statt, um das Bild ruhiger zu halten. In [VR11, VPR13] wird das ebenfalls webbasierte Tool *VATIC* vorgestellt. Objekte werden durch bilineare Interpolation verfolgt. Es wird eine Kombination aus aktivem Lernen und dynamischer Programmierung genutzt, um Frames zu erkennen, welche eine starke Änderung der Bounding Box hervorrufen. Hierdurch soll effektiv der Nutzereingriff

verringert werden. In komplizierten Szenen ergibt sich jedoch kein Mehrwert. *VATIC* teilt die Arbeit dabei explizit auf mehrere Personen auf: Videosequenzen werden in kurze Abschnitte geteilt, welche online über Amazon Mechanical Turk zur Bearbeitung weltweit verteilt und anschließend wieder verbunden werden. Ritter et al. stellen in [RE11] ein System vor, welches einen einfachen Tracker mit Block-Matching zur Verfolgung einer Bounding Box nutzt. Für eine pixelgenaue Markierung findet eine Segmentierung statt. Der Inhalt der Bounding Box dient dabei als positiver und die Umgebung als negativer Bereich. Für eine schnellere Navigation innerhalb des Videos wurde eine sogenannte Shot-Boundary Detection implementiert, welche ein repräsentatives Frame aus einem kurzen Videoabschnitt wählt und somit Navigationspunkte in der Sequenz bietet.

2.2 Verfahren

Tracking-Algorithmen haben sich in den letzten Jahrzehnten stark entwickelt. Mit wachsender Rechenleistung werden diese immer präziser und schneller. Viele Verfahren basieren auf Online-Lernverfahren, bei welchen ein Klassifikator erstellt wird. Einer der ersten Online-Tracker wurde in [GGB06] vorgestellt. Es wird ein Boosting Ansatz verfolgt, wodurch während des Trackings das Objektmodell zur Laufzeit aus einem positiven und mehreren negativen Beispielen, erstellt wird. Ein ähnlicher Ansatz wird auch in [BYB09] verwendet, allerdings wird das Lernen nicht auf einzelne positive Beispiele beschränkt, sondern auf mehrere um die aktuelle Position herum. Somit soll eine Degenerierung des Modells verhindert werden. In [HCMB15] wird das Objektmodell durch Regression einer Vielzahl von synthetischen Beispielen aus zyklische Verschiebungen effizient im Fourierraum berechnet. Durch die hohe Anzahl an Trainingsbeispielen soll das Modell sehr robust gegenüber dem Hintergrund werden. Ein einfacher Ansatz ohne Training wird in [KMM10] vorgeschlagen. Er basiert auf der Annahme, dass Tracking unabhängig von der Richtung sein muss. Ein Tracker muss vorwärts und rückwärts das gleiche Ergebnis liefern. Das Tracking basiert auf einem Fehlermaß dieser Annahme. Kalal et al. stellen ebenfalls einen Tracker vor, welcher das Tracking in drei Prozesse aufteilt [KMM12]. Hierbei wird zwischen Tracking, Lernen und Detektieren unterschieden. Der Tracker wird durch einen Detektor korrigiert, welcher wiederum online trainiert wird. In [HTS16] wird ein Verfahren vorgestellt, welches auf neuronalen Netzen basiert. Es wird nicht explizit ein Modell bzw. ein Objekt gelernt, sondern die Veränderungen eines Objektes, wenn es sich bewegt.
Es existieren zahlreiche Veröffentlichung über die Evaluierung von Trackern. In [WLY13] werden 29 Tracking-Algorithmen evaluiert. Verglichen werden die Bounding Boxen durch die Position ihres Zentrums und ihrer Überlagerung. Die Tracker werden an verschiedenen Frames und Positionen initialisiert. Die Testfälle werden bestimmten Situationen zugeordnet, um Stärken und Schwächen eines Trackers aufzustellen. Perazzi et al. stellen einen aktuellen Datensatz vor[PPTM+16], welcher pixelgenaue HD Aufnahmen aus verschiedenen Situationen enthält. Eine Evaluierung findet anhand der Silhouette des Objektes bezüglich der Form, Fläche und der Stabilität der Kontur statt. In [MYV07] werden Metriken auf Basis von der Überlagerung der Boundig Boxen mit einem Schwellenwert verwendet. Daraus resultieren verschiedene Metriken aus falsch negativen/positiven und tatsächlich negativen/positiven Ergebnissen sowie Verhältnisse aus der Anzahl der kompletten Frames und

der getrackten Frames. Black et al. schlagen in [BER03] die Erzeugung einer pseudo-synthetischen Ground Truth vor, um einen möglichst großen Evaluierungsdatensatz zu erhalten. Als Metriken werden Konsistenz des Trackings bezüglich der Richtung und Geschwindigkeit, die Farbkohärenz(Histogramme), die Formkohärenz(Fläche) und verschiedene Kombinationen aus den Ergebnissen dieser Metriken genutzt, wie z.B. der Gesamtanteil an richtig getrackten Objekten. Weiterhin wird eine gesonderte Bewertung von Verdeckung und Abweichung pro Objekt sowie eine Betrachtung der Fragmentierung der Ergebnisse in Bezug auf die Ground Truth. Bernadin et al. betrachten in [BS08] explizit Multi-Tracker für mehrere Objekte. Es werden einfache Metriken für Präzision und zeitliche Korrektheit aufgestellt. Aufbauend auf diesen werden Metriken bezüglich der Korrektheit von Treffern(MOTA) und der Präzision der Treffer(MOTP) bezügliche mehrerer Objekte aufgestellt.

3. Grundlagen

3.1 Neuronale Netze

Neuronale Netze Wie eingangs erwähnt, stellen diese Systeme eine künstliche Nachbildung des biologischen Hirns dar. Eine gute thematische Übersicht und Einführung in die biologischen Grundsätze ist in [KBB15] zu finden. An dieser Stelle wird eine kurze Einführung in das Grundprinzip gegeben. Die Verknüpfung von Neuronen und deren Informationen ermöglicht es, High- und Low-Level Features zu erkennen bzw. zu erzeugen. Im einfachsten Fall besitzt ein einzelnes Neuron m Eingänge und n Ausgänge ($m, n \in \mathbb{N}$). Eingänge können verschiedene Informationen enthalten, bei Bilddaten basierten Verfahren beispielsweise ein Pixel pro Eingabeneuron oder weitere Informationen, wie z.B. Relationen in einem Bild oder zusätzliche technische Informationen (z.B. Geschwindigkeit des Fahrzeugs, aus dem die Bilddaten stammen). Aus- bzw. Eingänge haben ein spezifisches Gewicht $\mathbf{w}$, welche in Summe ein Gesamteingangsgewicht bilden. Jedes Neuron besitzt einen (Bias-)Schwellenwert θ für das Gesamteingangsgewicht, ab welchem es, je nach verwendeter Aktivierungsfunktion (z.B. Sprungfunktion, logistische Funktion), aktiviert wird. Die Ausgänge stellen nun wieder Eingänge für die nächste Schicht dar. Jede Schicht besteht aus $o \in \mathbb{N}$ Neuronen. Die erste Schicht ist die Eingabeschicht, die Eingänge ihrer Neuronen haben kein Gewicht, sondern nur ihre Ausgänge. Die folgende Schicht ist eine versteckte Schicht (Hidden Layer) und stellt die eigentliche Verarbeitungsschicht dar. Die letzte Schicht ist die Ausgabeschicht mit der Besonderheit, dass jedes Neuron nur einen Ausgang hat. Ein solches Netz nennt man vorwärts gerichtet (feed-forward). Um eine Art Kurzzeitgedächtnis zu erhalten, ist es möglich Neuronen mit sich selbst bzw. den vorherigen Schichten zu verknüpfen, in diesem Fall spricht man von rekurrenten Netzen. Ist jedes Neuron einer Schicht mit jedem Neuron der Folgeschicht verknüpft, spricht man von voll-verknüpften (fully-connected) Schichten. Der vereinfachte Aufbau eines vorwärts gerichteten Netzes wird in Abbildung 3.1 verdeutlicht.

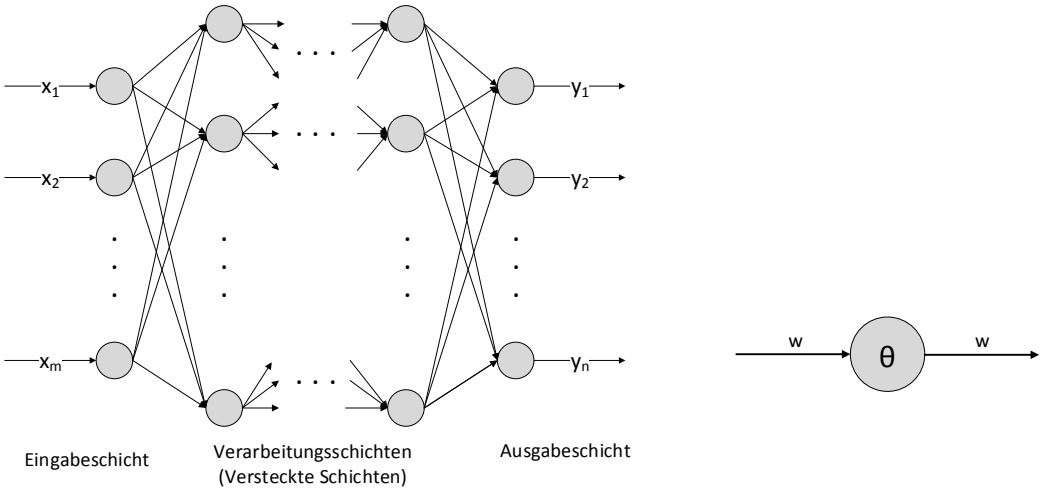

Abbildung 3.1: **Links:** Grundprinzip eines mehrschichtigen, vorwärts-gerichteten neuronalen Netzes. Zur besseren Übersicht wurden die Gewichte **w** und Schwellenwerte θ weggelassen. **Rechts:** Der Aufbau eines einzelnen Neurons mit den Gewichten **w** und dem Schwellenwert θ.

Die Gewichte und Schwellenwerte der einzelnen Neuronen können automatisiert trainiert werden. Hierbei werden die Parameter iterativ solange angepasst, bis $k \in \mathbb{N}$ Iterationen oder ein Schwellenwert (minimale Änderung) erreicht wurden. Eine (dynamische) Lernrate gibt an, wie stark der Wert pro Iteration angepasst wird. Das Training eines Netzes setzt Trainingsdaten voraus, anhand derer die Gewichte angepasst werden. Das trainierte Netz wird anschließend an einem unbekannten Datensatz validiert. Um bezüglich der Eingangsdaten möglichst robuste Lösungskandidaten zu erhalten, ist es von besonderer Bedeutung, eine große Anzahl möglichst vielfältiger Trainingsbeispiele zu haben [ASH12].
Deep Learning bzw. Deep Neural Networks (DNNs) funktionieren hierbei prinzipiell wie die einfachen Netze. Die Besonderheit liegt in der erhöhten Anzahl der versteckten Schichten. Die vorderen versteckten Schichten extrahieren meist Low-Level Features(z.B. Kanten in Bildern) aus den Eingangsdaten, die tieferen Schichten verbinden diese Basis-Informationen zu High-Level Features(z.B. Körperformen oder Beziehungen zwischen Objekten). Deep Learning wird vor allem für komplexe Probleme genutzt, welche nicht ohne a priori Wissen lösbar sind. Eine Analogie besteht hierbei zu dem Erfahrungsschatz von Lebewesen: beispielhaft muss ein Mensch erst lernen, wie bestimmte Dinge aussehen und was sie bedeuten. Es kann ohne Vorwissen nicht (sicher) bestimmt werden, ob ein Pilz giftig ist oder was ein Schriftzeichen bedeutet, er kann es lediglich wahrnehmen. Eine besondere Form von DNNs sind die Convolutional Neural Networks (CNNs), deren Aufbau dem Prinzip einer Konvolution bzw. Faltung aus der Bildverarbeitung nachempfunden sind und sich daher für bild- oder sprachbasierte Ansätze besonders eignen. Der klassische Aufbau von CNN besteht im Wesentlichen aus Konvolutions-Schicht(en) (Faltungsoperation/Low-Level), Pooling-Schicht(en) (Vereinfachen/Reduzieren der Daten) und voll-verknüpften Schicht(en) (Verknüpfen der Informationen/High-Level).

3.2 Annotation

3.2.1 Intention

Unter Annotation versteht man das Hinzufügen von Informationen. Für Multimediadaten bedeutet dies, den Bild-, Audio- oder allgemein Sensordaten zusätzliche Informationen, wie Markierung von einzelnen Objekten oder Abschnitten, Zusammenhänge oder Attribute, zuzuweisen. Über solche Methoden können nicht direkt sichtbare Informationen verdeutlicht werden. In einer Audiodatei ist es somit möglich, gesprochenen Sätzen jeweils Gefühlslagen durch Betonung oder gar eine andere inhaltliche Bedeutung als die eigentliche Aussage zuzuweisen. In Bildern kann man den Fokus auf bestimmte Objekte lenken und diese mit zusätzlichen kontextbezogenen Informationen versehen. Die für diese Arbeit wichtige Annotation von Videodaten ist insofern besonders interessant, als dass man, im Gegensatz zu einzelnen Bildern, zusätzliche Informationen über den temporalen Verlauf der Sequenz erlangt. In Bezug auf eine Videosequenz aus einem Fahrzeug heraus ist erst hier der Verkehrsfluss erkennbar, in einem einzelnen Bild könnten theoretisch alle Verkehrsteilnehmer still verharren.
Annotierte Daten werden für unterschiedliche Situationen genutzt. Somit kann in einem Fußballspiel ein interessanter Spieler markiert werden, um ihn im Spielfluss von den anderen Spielern unterscheiden zu können und seinen Zug zu erläutern. Hieraus können auf der einen Hand beispielsweise Lehrvideos für andere Spieler entstehen oder auf der anderen Hand einfach unterhaltende Informationen für den Zuschauer gegeben werden. Ein wichtiges Anwendungsgebiet in Hinsicht auf Lehrvideos stellen jedoch nicht nur die Menschen, sondern auch die Maschinen dar. Intelligente Systeme mit maschinellen Lernverfahren können diese Daten als Trainingsmaterial nutzen, um Situationen neu zu bewerten und für den Menschen unsichtbare oder schwer erkenntliche Informationen zu gewinnen oder um Situationen vorherzusehen (Prädiktion). Letzteres ist im Bereich des autonomen Fahrens in Verbindung mit Deep Learning(Deep Driving) besonders interessant. So könnte ein Auto, durch das Trainieren an zahlreichen Unfalldaten, dazu in der Lage sein, Unfälle vorherzusehen und Brems- oder Ausweichmanöver rechtzeitig einzuleiten. Da es, wie in Abschnitt 3.1 beschrieben, von großer Wichtigkeit ist, dass man sehr viele Trainingsdaten hat, ist es relevant, diese Trainingsdaten möglichst effizient zu generieren. Gerade in autonomen Fahrzeugen ist es besonders wichtig, dass das Fahrzeug in jeder Situation reagieren kann, um Unfälle zu vermeiden, da es schnell zur Gefährdung von Menschenleben kommen kann. Aber auch unabhängig von Unfallvermeidung bietet das Training an Situationen, welche einem menschlichen Autofahrer nicht häufig begegnen, Vorteile. Menschen können Situationen zum ersten Mal erfahren und wissen nicht, wie sie reagieren sollen, ein Computer kann auf einen großen Trainings-Datensatz zugreifen und somit auch in seltenen Situationen korrekt reagieren.

3.2.2 Annotationssysteme

Um solch große Datensätze zu erzeugen, ist eine effiziente Software unerlässlich. In einer frühen Arbeit [HB92] werden bereits Anforderungen bzw. Empfehlungen bezüglich des Designs von Annotationssystemen ausgesprochen. So wurde detailliert auf die Interaktionsmöglichkeiten eingangen. Zu den Kernfunktionen gehören das Markieren von Videoabschnitten, Definieren von Tastaturkürzeln, Steuer-, Navigations-,

und Abspielelemente, verschiedene Abspielmodi(z.B. Schleifen), sowie Datenexport. Die Markierung von Objekten soll direkt auf dem Videofenster erfolgen und wieder abspielbar sein. Das Interface sollte zudem modifizierbar sein. Annotationen müssen auf Gesten, Objekte, Gefühle und Zusammenhänge erfolgen und zusätzlich durchsucht werden könnnen. Viele dieser Eigenschaften finden sich in aktuellen Systemen wieder. Besonders die Verwendung von Tastaturkürzeln und Steuerelementen ist ein wichtiger Punkt, um die Annotation zu beschleunigen. Aktuelle Systeme, wie [VPR13, YRCT09], nutzen vor allem bilineare Interpolation zwischen Keyframes, um ein aufwändiges Frame-by-Frame Labeln zu beschleunigen. Lineare Interpolation ist rechen-technisch nicht aufwändig und somit auf allen Systemen in Echtzeit verfügbar. Für das Markieren von Objekten werden zwei Arten unterschieden: eine grobe Markierung mit einer Bounding Box oder genaue Markierungen durch Polygone. Der Bedarf richtet sich nach dem Anwendungsfall. Sind grobe Annotationen ausreichend, so kann die Verwendung von Bounding Boxen, durch den geringeren Nutzeraufwand, zu signifikanten Beschleunigungen führen. Für eine pixelgenaue Markierung ist ohne weitere Verarbeitung, wie in[RE11], das Verwenden von Polygonen notwendig. Die vorgegebenen Labels(Kategorien, Attribute) können vorgegeben sein oder zur Laufzeit durch den Nutzer erzeugt werden, ebenso wie die Vorgabe der Zeitschritte, in welchen gelabelt werden soll. Generell gilt: je weniger Möglichkeiten ein Nutzer hat, desto weniger ist er irritiert bzw. effizienter kann er arbeiten[VPR13, PD10]. Beide Systeme nutzen Server-basierte Webanwendungen. [VPR13] verwendet den Dienst *Amazon MTurk*, um die Annotation von großen Datenmengen in kleine Arbeitspakete aufzuteilen und weltweit bearbeiten zu lassen. Somit ist es möglich, in kurzer Zeit und mit geringem Kostenaufwand große Datenmengen zu annotieren. Rechtlich ergeben sich in Deutschland bei der Verteilung von Daten in das außereuropäische Ausland für kommerzielle Zwecke jedoch große Probleme. Sofern die Daten nicht anonymisiert sind oder die explizite Einwilligung der identifizierbaren Personen eingeholt wurde, verstößt das Hochladen dieser Daten gegen das Bundesdatenschutzgesetz (BDSG)[Bun15] und die EU-Datenschutz-Grundverordnung (EU-DSGVO)[Eur16]. Das Anonymisieren der Daten ist für große Mengen, sofern nicht zuverlässig automatisiert, sehr aufwändig und muss in jedem einzelnen Frame erfolgen. Sofern dies nicht erfolgt, ist die Nutzung dieser Systeme sehr eingeschränkt und nur unter Verwendung einer Datenschutzerklärung möglich. In Bezug auf die angesprochene lineare Interpolation gibt es jedoch auch Alternativen, wie in [RE11]. Hier wird optisches Tracking verwendet, um das Objekt automatisiert verfolgen zu lassen. Bei einem zuverlässigen Tracker wäre der Nutzereingriff somit noch geringer. Der Einsatz von Trackern wird in den folgenden Abschnitten diskutiert.

3.3 Tracking

Tracking beschreibt die Langzeitverfolgung von Signalen. In diesem Anwendungsfall handelt es sich um Objekte, also Bildausschnitte, in Videosequenzen. Der Tracker erstellt dabei eine Trajektorie(Bewegungspfad) des Objekts. Somit können z.B. Verhaltensmuster(Flugverhalten von Vögeln, Strömungen) und Messdaten bestimmt oder, wie in dieser Arbeit, Objekt-Annotation vorgenommen werden, um Menschen (mit Lehrvideos) oder Maschinen zu trainieren. Ein idealer Tracker ist dabei robust gegenüber der Objektrepräsentation und unterschiedlichen Belichtungsverhältnissen. Er ermöglicht eine genaue Lokalisierung des Objekts, kann Verdeckung und

Abwesenheit behandeln, ist robust gegenüber schnellen Bewegungen und kann das Objekt auch mit neuen Informationen, wie Rotation oder Bewegung von Gliedmaßen, verfolgen und skalieren. Für Echtzeitfähigkeit muss er des Weiteren möglichst effizient zu berechnen sein.

3.3.1 Interpolation vs. Tracking

Interpolation wird, wie in Abschnitt 2.1 und Abschnitt 3.2.2 geschildert, aufgrund seiner einfachen Implementierung und weichen Trajektorie oft verwendet. Betrachten wir Interpolation mit Hinsicht auf komplexe Kamera-Situationen, wie Deep Driving, entstehen allerdings Probleme. (Bi-)lineare Interpolation kann nur lineare Bewegungen abbilden. Betrachten wir beispielhaft als Situation ein Fahrzeug mit eingebauter Kamera zum Aufnehmen und späteren Verarbeiten von Verkehrsdaten. Wir können nicht mehr annehmen, dass die Kamera statisch, sondern Teil des Fahrzeugs und somit dynamisch ist. Dies schließt zugleich auch weitere Verfahren wie Hintergrundsubtraktion aus, da sich dieser permanent ändert. Durch die dynamische Kamera kommt es zu zusätzlichen Problemen bei nicht-linearen Bewegungen, wie z.B. Bremsmanövern oder Beschleunigungen, welche unabhängig von der dynamischen Kamera durch den Verkehrsfluss anderer Fahrzeuge auch entstehen würden. Weiterhin kommt es durch Fahrbahnunebenheiten, wie Schlaglöcher, Spurrillen, Gullydeckel, Gleise oder Kopfsteinpflaster, zu starken, unvorhergesehenen Bewegungen der Kamera, welche bei Interpolation nur durch eine erhöhte Anzahl von Interpolationspunkten auszugleichen ist. Hierdurch steigt der ohnehin zeitlich hohe Aufwand bei dem Annotieren weiterhin stark an. Eine Veranschaulichung der Probleme wird in Abbildung 3.2 gezeigt.

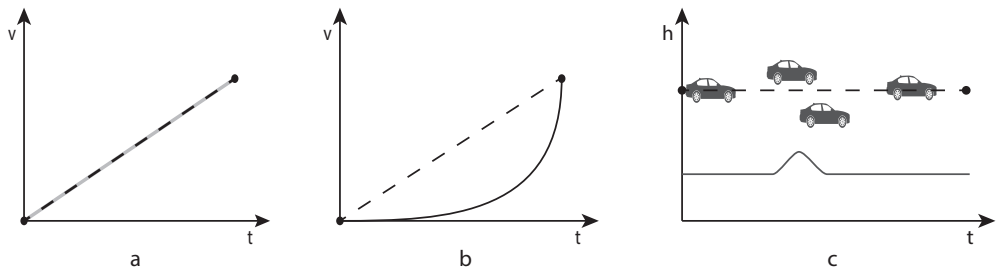

Abbildung 3.2: Probleme von (bi-)linearer Interpolation bei nicht-linearen Bewegungen. Die gestrichelte Linie stellt die Interpolation dar, die stetige Linie die Bewegung. **(a)** Interpolation deckt sich mit linearer Bewegung, **(b)** Interpolation einer nicht-linearen Bewegung führt zu Abweichungen, **(c)** die untere Linie stellt die Fahrbahnoberfläche dar, die obere Linie die daraus resultierende Kamerabewegung im Bild. Durch Stoßdämpfer kommt es zu Schwankungen nach oben und unten.

Der Vorteil von Tracking besteht in der Unabhängigkeit von der Kamera. Ein Tracker verfolgt ein Objekt von einem Frame zu dem nächsten. Dabei ist es irrelevant, ob die gesamte Bewegung linear ist oder nicht, da immer nur ein Zeitschritt verfolgt wird. Weiterhin sind auch unvorhergesehene Bewegungen durch Fahrbahnunebenheiten kein Problem, da diese ebenfalls nur, durch die hohe Framerate der Kamera, eine geringe Veränderung zwischen einzelnen Frames bedeutet. Zusätzlich kann ein Tracker das Objekt automatisch verfolgen, sobald er es initialisiert hat. Ein idealer

Tracker ermöglicht somit eine stark beschleunigte Annotation, da nur das Objekt initial markiert und im weiteren Verlauf die Labels angepasst werden müssen. Leider existiert ein idealer Tracker (nocht) nicht in der Praxis, wodurch das Tracking wieder eigene Probleme hervorruft, welche in Abschnitt 6.1 genauer erklärt werden. In den folgenden Abschnitten erfolgt eine theoretische Einführung in Trackingsysteme.

3.3.2 Arten von Tracking

In [YJS06] wird eine große Übersicht über unterschiedliche Trackingsysteme geboten. Diese unterscheiden sich in der Objekt-Repräsentation anhand von Formmodellen, dem Erscheinungsbild und den verwendeten Features zum Beschreiben dieser Eigenschaften. Weiterhin unterscheiden sich die Systeme in ihrer Initialisierung durch manuelle Markierung oder die Verwendung eines Detektors, sowie durch die Art des Trackings, basierend auf dem Form-Modell.

Form-Modelle Form-Modelle beschreiben die Form des zu trackenden Bereiches. Sie unterscheiden sich in Größe, Bewegungs-Modellen innerhalb des Modells und der benötigten Vorverarbeitung. Die Präzision und Rechengeschwindigkeit eines Trackers ist abhängig von der Wahl des Form-Modells. Es folgt eine Übersicht über häufig verwendete Formmodelle, sowie eine Veranschaulichung in Abbildung 3.3:

- Punkt: einzelner Punkt zum Tracken sehr kleiner Regionen
- Primitive: Rechtecke oder Ellipsen für entsprechend einfache Formen oder Approximierung von komplexen Formen
- Silhouette/Kontur: genaue Formbeschreibung des Modells anhand der Kanten(Kontur) oder inneren Textur(Silhouette)
- Gelenke: Verbindung mehrerer Primitive über Schnittstellen mit spezifischem Bewegungs-Modell (z.B. max. Öffnungswinkel)
- Skelett: Skelettstruktur eines Körpers, sowohl für einfache als auch komplexe Modelle geeignet

Erscheinungsbild-Modelle Diese Modelle beschreiben das Aussehen(Appearance) eines Objektes. Das Modell beinhaltet Informationen zu Objekteigenschaften(Features) und deren Repräsentation. Es trägt somit direkt zur Unterscheidung des Objektes vom Hintergrund und anderen Objekten bei. Wichtige Erscheinungsbild-Modelle sind:

- Wahrscheinlichkeitsdichten: beschreiben die Verteilung von Merkmalen anhand von (gemischten) Gauß-Verteilungen oder Histogrammen, oft verwendete Features sind Farbe und Textur
- Templates: stellen eine explizite Beschreibung des gesuchten Objektes dar und beinhalten sowohl Form als auch Aussehen

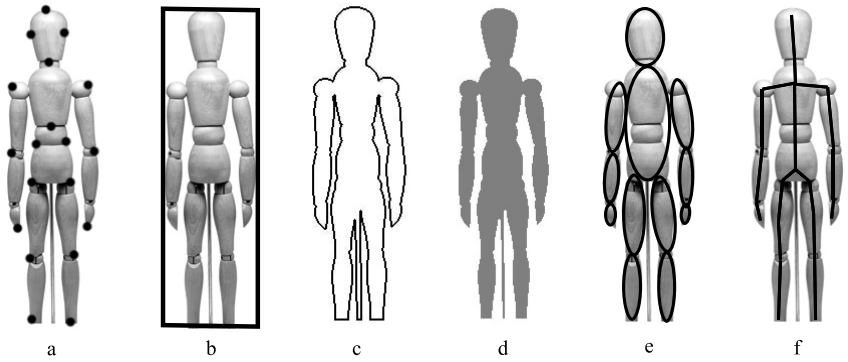

Abbildung 3.3: Übersicht über oft verwendete Form-Modelle. **(a)** Punkte, **(b)** Primitive, **(c)** Kontur, **(d)** Silhouette, **(e)** Gelenke, **(f)** Skelett

- Active Appearance-Modell: ist ein aktiv lernendes System, basierend auf Landmarken, aus welchen ein sogenannter Appearance-Vektor erstellt wird
- Multi-View: beschreibt ein offline trainiertes Modell, welches an verschiedenen Perspektiven des Objektes trainiert wird und z.B. durch eine Hauptkomponenten-Analyse erstellt wird

Featureklassen Die Erscheinung eines Objektes wird durch Features beschrieben. Diese können einfache Informationen wie Farbe enthalten oder komplexere Modelle, wie spezielle Histogramme enthalten. Einzelne Features beschreiben dabei das Objekt nicht vollständig, sondern nur kleine Teilbereiche. Eine vollständige Beschreibung des Objekts wird durch einen Featurevektor, welcher die Features aller Teilbereiche zusammenfasst, erhalten. Auf spezielle Features wird in Abschnitt 3.3.3 weiter eingegangen, oft verwendete Featureklassen sind Farbintensitäten, Kanten, optischer Fluss und Texturen. *Farbintensitäten* sind empfindlich gegenüber schwankender Belichtung, weiterhin sind sie in schwarz/weiß Bildern weniger diskriminativ. *Kanten* beschreiben die Form des Objektes anhand der Konturen durch Gradientenbildung. Sie sind robust gegenüber schwankender Belichtung. Der *optische Fluss* ist ein Feld aus lokalen Fluss- bzw. Verschiebevektoren für Translation von Pixeln zwischen Frames. Er wird daher besonders bei bewegungs-orientierten Modellen, wie dem Median-Flow Tracker in Abschnitt 3.3.3 verwendet. *Texturen* sind aufgrund ihrer Komplexität und der Nachbarschaftsbeziehungen robuster gegenüber Belichtung als reine Farbintensitäten. Auf Features bauen zwei Systeme auf. Das erste ist ein Filter, welches korrelierende Features findet und diese zu einem Featurevektor verbindet. Das zweite ist ein Wrapper, welcher die besten Features aus einem Pool selektiert und ebenfalls verbindet.

Detektion Das zu trackende Objekt muss initialisiert werden. Hierfür gibt es verschiedene Möglichkeiten: das Objekt kann vom Nutzer markiert werden oder automatisiert über einen (trainierten) Detektor gefunden werden. Für die manuelle Initialisierung wird oft ein, auf das Objekt-Modell basierender, Detektor verwendet.

Die Initialisierung ist sehr wichtig für die Perfomance des Trackers[WLY13], da sie direkt vorgibt, wie nah das Modell an den tatsächlichen Daten ist. Detektoren werden auch genutzt, um die Ergebnisse des Trackers zu verbessern[Kal11] oder um vollautomatisiert Objekte zu verfolgen, da eine Nutzereingabe nicht notwendig ist. Es wird zwischen verschiedenen Ansätzen unterschieden. *Punkt-basierte* Detektoren suchen nach Punkten, welche das Objekt gut beschreiben und sich gut differenzieren lassen, wie z.B. der Harris-Corner Detektor. Durch *Hintergrundsubtraktion* differenzieren sich bewegte Objekte vom Hintergrund, für den ein Hintergrund-Modell erstellt wird. Um kleine Bewegungen, wie z.B. Blätter im Wind oder leichtes Wackeln der Kamera auszugleichen, werden nicht allein die Pixel, sondern deren Nachbarschaft betrachtet. Ein weiterer Ansatz ist die *Segmentierung* des Bildes in zusammenhängende Segmente, z.B. über die Wasserscheidentransformation. *Überwachtes Lernen* wird genutzt, um ein Modell anhand von positiven und negativen Trainingsbeispielen zu trainieren, welches als Klassifikator ein spezifisches Objekt im Bild detektieren kann. Wichtig sind hierbei vor allem die Anzahl der Trainingsbeispiele, damit das Objekt möglichst gut vom Hintergrund unterschieden werden kann.

Trackingverfahren Der eigentliche Trackingprozess erstellt die Trajektorie des Objektes. Je nach verwendetem Modell wird eine andere Art des Trackings verwendet. Gemeinsam haben alle, dass sie ein Bewegungsmodell nutzen, um den Suchraum des Objektes einzugrenzen. Oftmals beruht dieses Modell auf der hinreichenden Bedingung, dass das Objekt sich zwischen zwei Frames nicht zu stark bewegt und der Suchraum in der Nähe der alten Position sein muss. *Punkt-basierte* Methoden nutzen den vorherigen Status des Objektes. Es wird unterschieden zwischen deterministischen und statistischen Ansätzen. Deterministische Ansätze verwenden Heuristiken, basierend auf Kostenfunktionen, um ein Bewegungsmodell zu erstellen. Statistische Ansätze verwenden Messdaten und ergänzen diese um Unsicherheiten, wodurch Bildrauschen berücksichtigt wird. Es wird ein statistisches Modell aus allen Beobachtungen erstellt, ein solches Verfahren ist z.B. das Kalman-Filter in Abschnitt 3.3.3. Sobald größere Objekte verfolgt werden, nutzen beide Ansätze ein automatisiertes Clustering von mehreren Punkten, um das Objekt zu beschreiben. Methoden, welche die *Silhouette* des Objektes verfolgen, basieren auf der genauen Form von geometrisch komplexen Objekten. Hierfür werden vor allem Farbhistogramme und Kanten bzw. Konturen genutzt. Man unterscheidet zwischen Form-basierten und Kontur-basierten Ansätzen. Bei Form-basierten Ansätzen wird die Silhouette des Objektes über Template-, im Speziellen Shape-, Matching gesucht. Hierfür wird z.B. die Hausdorff-Distanz zwischen zwei Objekten minimiert. Kontur-basierte Ansätze verändern die ursprüngliche Kontur und passen sie, unter Nutzung von Zustandsräumen oder Energie-Minimierung der Kontur, an das aktuelle Frame an. Auf Kernel-basierte Verfahren wird im nächsten Abschnitt genauer eingegangen und es werden die in OpenCV Kernel-basierten Tracker vorgestellt.

3.3.3 Kernelbasierte Tracking-Algorithmen

Diese Art von Trackingverfahren nutzt geometrische Primitive, wie Ellipsen, aber vor allem Rechtecke, um das Objekt einzugrenzen. Das Bewegungsmodell basiert auf der Transformationen der Objekte, vor allem auf Translation, wo stets die neue

Position des Kernels gesucht wird. Bei dem Erscheinungsbild-Modell wird zwischen Template- bzw. Dichte-basierten und Multiview Modellen unterschieden. *Template-basierte* Ansätze sind rechen-technisch aufwändiger, da über Template-Matching ein Bruteforce-Ansatz zur Lokalisierung der neuen Position gewählt wird, welche durch das Bewegungs-Modell eingegrenzt wird. Wichtige Features sind vor allem die Pixelintensitäten, Kanten und Histogramm-basierte Features, wie in Abschnitt 3.3.3 beschrieben. *Dichte-basierte* Ansätze nutzen die Flussdichte bzw. den optischen Fluss des Bildes, um einen Translations- bzw. Flussvektor aufzustellen. Für die Berechnung des lokalen Flusses werden oft die Helligkeit oder Gradienten verwendet. Trackingsysteme, wie der [LK81] in Abschnitt 3.3.3, minimieren hierbei den Fehler, welcher durch die Verschiebung des Objektes entsteht. *Multiview-basierte* Ansätze nutzen, wie schon bei der Beschreibung der Erscheinungsbild-Modelle, Trainingsdatensätze aus verschiedenen Perspektiven, um einen Klassifikator zu erstellen. Das Training erfolgt durch eine Hauptkomponenten-Analyse oder den Einsatz einer Support Vector Machine (SVM). In den nächsten Abschnitten wird auf die mathematischen Grundlagen eingegangen, um anschließend die verwendeten Tracking-Algorithmen zu erläutern.

Mathematische Grundlagen

Lucas-Kanade Tracker Der Lucas-Kanade Tracker ist ein auf den optischen Fluss basierendes Trackingverfahren [LK81]. Für den optischen Fluss zwischen zwei Frames werden die Gradienten in einem Gleichungssystem
$Av = b$ berechnet :

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \qquad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \qquad b = \begin{bmatrix} -I_t(q_1) \\ \vdots \\ -I_t(q_n) \end{bmatrix} \tag{3.1}$$

Wobei v der lokale Flussvektor, q_i das aktuelle Pixel und $I_x(q_i), I_y(q_i), I_t(q_i)$ die partiellen Ableitungen des Bildes bezüglich der Dimensionen und der Zeit sind. Dieses über-bestimmte Gleichungssystem kann nun mithilfe von Sum of Squared Differences (SSD) gelöst werden:

$$v = (A^T A)^{-1} A^T b \tag{3.2}$$

und wir erhalten:

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} I_x(q_i)^2 & \sum_{i=1}^{n} I_x(q_i) I_y(q_i) \\ \sum_{i=1}^{n} I_y(q_i) I_x(q_i) & \sum_{i=1}^{n} I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_{i=1}^{n} I_x(q_i) I_t(q_i) \\ -\sum_{i=1}^{n} I_y(q_i) I_t(q_i) \end{bmatrix} \tag{3.3}$$

Kalman-Filter Das einfache Kalman-Filter (KF)[Kal60] wird allgemein in linearen Systemen eingesetzt, um Messrauschen zu korrigieren oder in der Statistik, um Verteilungen zu schätzen. Für lineare Modelle ist es optimal. In der Computervision

findet es Anwendung im Tracking. Es gibt grundlegend zwei Iterationsschritte: Messung(Update) und Schätzung(Prädiktion). Die Messung in Zeitschritt t wird durch die Schätzung aus $t-1$ ergänzt und bildet die neue Schätzung für t. Es wird davon ausgegangen, dass das Rauschen einer Gaußverteilung unterliegt. Der Erwartungswert μ gibt den zu berechnenden Wert an, die Varianz σ^2 wie sicher das Ergebnis ist, also wie stark das Rauschen ist. Inital entspricht die erste Schätzung S_1 der ersten Messung M_1, wobei M und S dem jeweiligen Erwartungswert μ_M bzw. μ_S entsprechen. Im eindimensionalen Fall erhalten wir in dem Zeitschritt t eine Messung M_t mit Varianz $\sigma^2_{M_t}$. Für die neue Schätzung S_t der Varianz $\sigma^2_{S_t}$ berechnen wir nun ein nach Varianzen gewichtetes Mittel. Hierfür definieren wir das Kalman-Gain KG:

$$KG = \frac{\sigma^2_{S_{t-1}}}{\sigma^2_{S_{t-1}} + \sigma_{M_t^2}} \tag{3.4}$$

$$S_t = S_{t-1} + KG(M_t - S_{t-1}) \tag{3.5}$$

$$\sigma^2_{S_t} = \frac{1}{\frac{1}{\sigma^2_{S_{t-1}}} + \frac{1}{\sigma^2_{M_t}}} \tag{3.6}$$

Die neue geschätzte Varianz $\sigma^2_{S_t}$ ist somit geringer als die beiden einzelnen, die Sicherheit der Schätzung größer. Im zweidimensionalen Fall wird das Filter um Matrizen für die Kovarianz P, den Prozess mit Dynamikmatrix A, die Prozessrausch-Kovarianzmatrix Q, die Messmatrix H und die Messrausch-Kovarianzmatrix R erweitert. Wichtig ist das Größenverhältnis der Werte in Q und R.

- $Q > R$: schnelle Reaktion auf Änderungen
- $Q < R$: Glättung des Rauschens

Da das einfache KF nur für lineare Modelle geeignet ist, wurde ein Erweitertes Kalman-Filter (EKF) für nicht-lineare Modelle entwickelt, wie in [CZR10] beschrieben. Würde man das einfache KF auf nicht-lineare Modelle anwenden, wäre die resultierende Verteilung nicht mehr normalverteilt. Das EKF funktioniert grundlegend ähnlich, allerdings werden die nicht-linearen Bewegungen zum Messzeitpunkt t linearisiert und dadurch approximiert, wodurch das Filter nicht mehr optimal ist. Vereinfacht wird für die Linearisierung eine Jacobimatrix des aktuellen Zustands(Mittelwert und Kovarianz) erstellt, wodurch man den Anstieg der Funktion erhält. Diese wird in eine Taylor-Reihe 1. Ordnung eingesetzt und man erhält den geschätzten Zustand.

Fourier-Transformation Die Fourier-Transformation beschreibt die Abbildung eines Signals aus dem ursprünglichen Ortsraum in den Frequenzraum bzw. Fourierraum [Tön05]. Hierbei werden komplexe, orthogonale Basisfunktionen b in Form von Sinus- und Kosinus-Funktionen genutzt. Die Abbildung ist umkehrbar, sodass die Repräsentation im Frequenzraum durch die inverse Transformation verlustfrei wie-

der in den Ortsraum überführt werden kann. Für den diskreten zweidimensionalen, für die Bildverarbeitung relevanten, Fall ergibt sich folgende Berechnung:

$$F(u,v) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) b_{u,v}(m,n) \qquad b_{u,v}(m,n) = \exp\left[-i2\pi\left(\frac{um}{M} + \frac{vn}{N}\right)\right] \tag{3.7}$$

Wobei F, u, v die Repräsentation von f, m, n im Frequenzraum sind und die Sinus- bzw. Kosinusfunktion als Exponentialfunktion dargestellt wird. In der Praxis wird die Fast Fourier Transformation(FFT) als schnelle Berechnung genutzt. Ein Bild wird also als Überlagerung von Wellen dargestellt. Im Frequenzraum hat die Funktion folgende Eigenschaften:

- Frequenz: Abstand der Position (u, v) zum Ursprung
- Richtung: Richtung des Vektors (u, v)
- Amplitude: (Grau-)Wertvariation der Welle im Bild
- Phase: Winkel der Welle, entscheidend für Position im Ortstraum

Wesentliche(homogene) Bereiche bzw. Informationen eines Bildes werden durch niedrige Frequenzen repräsentiert und liegen im Zentrum des Fourierraums. Hohe Frequenzen (Außenbereich des Frequenzraums) stellen meist Rauschen dar. Die Fouriertransformation ist vor allem für die Faltungs-($\star$) bzw. Korrelationsoperation($\circ$) interessant. Im Ortsraum muss für die Konvolution eine Filtermaske g über jedes Pixel des Bildes f summiert werden und stellt somit einen großen Aufwand dar. Im Frequenzraum hingegen muss die Filtermaske G nur mit dem Bild F multipliziert werden. Äquivalent gilt dies für die Korrelation, wobei hier die komplex-konjugierte Form von G genutzt wird.

Normalisierte Kreuzkorrelation Die Kreuzkorrelation oder nur Korrelation, wie in [Tön05] beschrieben, berechnet die Verschiebung zweier Signale f und g. Sie wird somit genutzt, um Ähnlichkeiten zwischen zwei Bildern bzw. einem kleineren Modell t zu entdecken. Unter der Annahme, dass t ebenso groß ist wie f, t am Ende des Modells bis zu dem Rand mit Nullen aufgefüllt ist und der Mittelwert von t gleich Null ist, wird die diskrete Korrelationsfunktion wie folgt definiert:

$$[f \circ g](m,n) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f(i+m, j+n) t(i,j) \tag{3.8}$$

Eine Normalisierung wird vorgenommen, um unterschiedliche Belichtungen auszugleichen. Der normalisierte Korrelationskoeffizient cc unter Benutzung der Varianz σ^2 ist definiert als:

$$cc_{f,g}(m,n) = \frac{1}{\sigma_f^2 \sigma_g^2} [f \circ g](m,n) \tag{3.9}$$

Da die Kreuzkorrelation eng verwandt mit der Konvolution ist, lässt sich diese ebenfalls über eine Multiplikation im Frequenzraum der Fourier-Transformation berechnen:

$$\mathcal{F}\left([f \circ g](m,n)\right) = F(u,v) \cdot G^*(u,v) \tag{3.10}$$

$\mathcal{F}$ beschreibt die Fourier-Transformation, die Großbuchstaben F, G sind die Repräsentation im Frequenzraum und G^* ist die komplex-konjugierte Form von G. Das Ergebnis ist eine Abbildung, deren Helligkeitsverteilung die Ähnlichkeit der beiden Bilder darstellt (confidence/response map). Je heller ein Bereich ist, desto ähnlicher sind sich beide Bilder an dieser Stelle.

Log-Likelihood Die Log-Likelihood, wie in [Kre02] beschrieben, entspricht der normalen Likelihood-Methode $\mathcal{L}$ zur Schätzung eines Parameters, jedoch mit logarithmierten Werten für eine einfachere Berechnung, da die Nullstellen (Maxima) erhalten bleiben. Für die Zufallsvariable X mit der von dem Parameter ϑ abhängenden Wahrscheinlichkeitsfunktion f mit einem Stichprobenumfang von n gilt:

$$\mathcal{L}(\vartheta) = \prod_{i=1}^{n} f_{X_i}(x_i;\vartheta) \tag{3.11}$$

$$\ell(\vartheta) = \ln \mathcal{L}(\vartheta) = \sum_{i=1}^{n} \ln f_{X_i}(x_i;\vartheta) \tag{3.12}$$

Für eine Normalverteilung $\mathcal{N}$ mit dem Erwartungswert μ, der Varianz σ^2 und der Wahrscheinlichkeitsdichte $f(x;\mu,\sigma^2)$, lässt sich für n unabhängige Ereignisse X mit unbekanntem Erwartungswert $\hat{\mu}$ und unbekannter Varianz $\hat{\sigma}^2$ der Parameter ϑ der unbekannten Verteilung schätzen. Die Log-Likelihood Funktion ist nun definiert als:

$$\ell(\hat{\mu};\hat{\sigma}^2;X) = -\frac{n}{2}\ln(2\pi\sigma^2) - \frac{\sum_{i=1}^{n}(x_i-\hat{\mu})^2}{2\hat{\sigma}^2} \tag{3.13}$$

Sie gibt an, wie nah die Schätzung an den echten Daten liegt. Für die Schätzung von $\hat{\mu}$ und $\hat{\sigma}^2$ werden die partiellen Ableitungen von $\ell(\hat{\mu};\hat{\sigma}^2;X)$ nach $\hat{\mu}$ und $\hat{\sigma}$ berechnet und gleich null gesetzt. Durch Lösen der nicht-linearen Gleichungssysteme ergeben sich:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3.14}$$

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i-\hat{\mu})^2 \tag{3.15}$$

Bayes'sches Entscheidungskriterium Das Bayes'sche bzw. Schwarz-Bayes Kriterium [Sch78] wurde eingeführt, um zu verhindern, dass bei geschätzten Verteilungen zu großen Entscheidungsmodellen mit vielen Parametern tendiert wird, in

dem diese Parameter bestraft werden. Sei θ ein unbekannter Parameter und p eine unbekannte Dichtefunktion, so kann bei einer Maximum-Likelihood Schätzung die Dichtefunktion als $q(\theta)$ definiert werden, deren geschätzter Parameter $\hat{\theta}$ ist. Weiterhin seien n der Stichprobenumfang und k die Anzahl der zu schätzenden Parameter. Das Bayes'sche Entscheidungskriterium BIC_ℓ gibt nun an, wie nah die Schätzung $q(\theta)$, unter Berücksichtigung der Strafterme, an p liegt. Hierzu wird die negative Log-Likelihood Funktion ℓ verwendet:

$$BIC_\ell = -2\ell(\hat{\theta}) + \log(n)k \tag{3.16}$$

Ridge Regression Ridge Regression ist auch bekannt als Regularized Least Squares (RLS)[HCMB12]. Ziel ist es einen linearen Klassifikator $f(\mathbf{z}) = \mathbf{w}^T\mathbf{z}$ zu erzeugen, welcher den Fehler zwischen den Samples $\mathbf{x}_i$ und dem Regressionsziel $\mathbf{y}_i$ minimiert:

$$\min_w \sum_i \left(f(\mathbf{x}_i) - y_i\right)^2 + \lambda\|\mathbf{w}\|^2 \tag{3.17}$$

Das Vorgehen ist hierbei ähnlich einer SVM, unterscheidet sich aber durch die Kostenfunktion, in diesem Fall den quadratischen Fehler. Ein weiterer Unterschied ist die Verwendung des Skalars λ, welches eine Überanpassung (overfitting) verhindern soll. $\mathbf{w}$ stellt somit einen Gewichtsvektor dar, welcher als Hyperebene zum Trennen des Datensatzes X in einem n-Dimensionalen Raum dienen soll:

$$\mathbf{w} = \left(X^TX + \lambda I\right)^{-1} X^T\mathbf{y} \tag{3.18}$$

Für komplexe Zahlen $\mathbb{C}$ nutzen wir die hermitische Transponierte ^{H} statt der normalen Transponierten ^{T}. Wie auch bei der SVM können wir den Kernel-Trick nutzen, um eine nicht-lineare Regression vorzunehmen, wir sprechen nun von Regularized Least Squares with Kernels (KRLS). Die Kernelfunktion κ erzeugt hierbei eine Abbildung in einen höherdimensionalen, nicht-linearen Raum $\varphi(\mathbf{x})$ (Dualspace), in welchem sich die Daten linear separieren lassen. Wenn wir die Daten über unsere Kernelfunktion wieder zurück projizieren, erhalten wir eine Separierung im Ortsraum. Unser Gewichtsvektor $\mathbf{w}$ lässt sich somit als implizite Linearkombination von Support-Vektoren des Dualspace α darstellen:

$$\mathbf{w} = \sum_i \alpha_i\varphi(\mathbf{x}_i) \tag{3.19}$$

Diese Gleichung lässt sich ebenfalls als Skalarprodukt darstellen:

$$\varphi(\mathbf{x}) = \varphi^T(\mathbf{x})\varphi(\mathbf{x}') = \kappa(\mathbf{x}, \mathbf{x}') \tag{3.20}$$

Dadurch lassen sich alle Samplepaare von Skalarprodukten in einer Kernelmatrix K mit den Elementen $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ darstellen. Nun können wir Gleichung 3.18 im Dualspace aufstellen:

$$\boldsymbol{\alpha} = (K + \lambda I)^{-1}\mathbf{y} \tag{3.21}$$

Der Vorteil dieser Methode ist, dass wir keine explizite Form der Abbildung $\varphi(\mathbf{x})$ berechnen müssen, sondern nur die implizite Repräsentation im Dualspace nutzen. Somit ergibt sich der Klassifikator als:

$$f(\mathbf{z}) = \mathbf{w}^T\mathbf{z} = \sum_{i=1}^{n} \alpha_i\kappa(\mathbf{z}, \mathbf{x}_i) \tag{3.22}$$

n stellt dabei die Anzahl der Trainingssamples dar, wovon die Komplexität der Berechnung direkt abhängt. Vergleichen wir Gleichung 3.18 und Gleichung 3.19, fällt auf, dass wir $\mathbf{w}$ mit nicht-linearem Kernel nicht mehr explizit angeben können, sondern nur noch unter Zuhilfenahme der Support Vektoren bzw. Samples.

Features

Features werden genutzt, um Bildinformationen zu kodieren. Sie können Informationen kompakter als reine Pixel darstellen und sind wesentlich schneller zu verarbeiten. Weiterhin können mit ihnen Bildinformationen zusammenhängend beschrieben werden. Features beziehen sich dabei meist auf einfache lokale Eigenschaften, welche in Summe als Featureset das Bild bzw. einen Ausschnitt beschreiben. Alle vorhandenen Features beschreibt man als Featurepool.

HAAR-like HAAR-like Features wurden in [VJ01] vorgestellt und stellen eine einfache regionale Beschreibung des Bildes dar. Sie basieren auf der lokalen Nachbarschaft von Rechtecken in einem Bildausschnitt. Es existieren drei verschiedene Nachbarschaften:

- zwei Rechtecke: Differenz zwischen der Summe zweier benachbarter horizontal oder vertikal angeordneten Rechtecke
- drei Rechtecke: Differenz zwischen den Summen zweier äußerer Rechtecke und der Summe eines inneren in einer horizontalen Anordnung
- vier Rechtecke: Differenz zwischen den Summen von paarweise diagonal angeordneten Rechtecken.

Die Summe der Pixel innerhalb eines Rechteckes wird effizient auf Basis eines Integralbildes berechnet, welches in einem Durchlauf jedem Pixel die Summe der vorangegangenen Pixel in dem Rechteck von $(0, 0)$ bis (x, y) zuweist und somit für die Summe eines Rechteckes nur die vier Eckpunkte benötigt, um diese direkt zu berechnen. Rekursiv lässt sich die Summe I eines Pixels i beschreiben als:

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{3.23}$$
$$I(x, y) = I(x - 1, y) + s(x, y) \tag{3.24}$$

HOG Histogram of Oriented Gradients (HOG) wurden in [McC86] vorgestellt, eine Anwendung für Computervision wird in [DT05] beschrieben. Sie beschreiben Objekte anhand ihrer Form durch die Verteilung von Gradienten. Sie sind somit sehr diskriminativ für Objektformen. Das Grundkonzept teilt das Bild dabei in sich überlappende Blöcke (rechteckig, radial) ein, welche jeweils nochmal in Zellen unterteilt werden. Für die Pixel einer Zelle werden nun einfache Gradienten ($[-1, 0, 1]$,$[-1, 0, 1]^T$) bestimmt und daraus ein orientiertes Histogramm erstellt, wobei jedes Pixel ein Gewicht entsprechend der Gradientenstärke erhält. Das Histogramm besteht aus verschiedenen Kanälen, entsprechend einer gleichmäßigen Verteilung von Winkeln, in

welche die Pixel bzw. Gradienten eingetragen werden. Die Zellhistogramme können nun, um robuster gegenüber Belichtungsunterschieden zu sein, innerhalb eines Block normalisiert werden. Die Histogramme aller Blöcke werden abschließend zu einem Feature-Vektor konkateniert.

LBP Local Binary Patterns (LBPs) wurden in [OPH94] vorgestellt. Sie basieren auf Nachbarschaftsinformationen eines Pixels. Ähnlich wie bei HOG in Abschnitt 3.3.3 wird das Bild in Zellen unterteilt. Für jedes Pixel einer Zelle wird nun die Information über die Nachbarschaft gebildet. Hierfür werden kreisförmig z.B. 8 Nachbarn um das zentrale Pixel bestimmt. Nun wird in mathematisch positive oder negative Richtung(konsistent) jeder Nachbar verglichen. Ist der Wert des zentralen Pixels größer, wird eine 0 eingetragen, anderenfalls eine 1. Somit entsteht ein binärer Vektor, welcher in eine Dezimalzahl überführt werden kann. Über alle binären Vektoren bzw. deren Dezimalwert einer Zelle wird ein Histogramm angelegt. Das Histogramm kann für eine bessere Robustheit normalisiert werden. Die Histogramme aller Zellen werden anschließend zu einem Feature-Vektor konkateniert.

Median-Flow Tracker

Der Median-Flow Tracker wurde 2010 von Kalal et al. [KMM10] veröffentlicht. Er basiert auf den im Paper vorgestellten Forward-Backward error (FB). Die Grundidee beruht auf der Annahme, dass ein korrektes Tracking richtungsunabhängig sein muss und somit sowohl vorwärts als auch rückwärts eine identische Trajektorie liefert. Für das Tracking zwischen k Frames zum Zeitpunkt t wird eine Trajektorie T_f^k vorwärts($t \rightarrow t_{+k}$) und eine Validierungstrajektorie T_b^k rückwärts($t \leftarrow t_{+k}$) berechnet. Der FB misst die Differenz zwischen den beiden Punktpositionen($x_t \in T_f^k, \hat{x}_t \in T_b^k$) der Trajektorien zum Zeitpunkt t. Um Pixel zu finden, die sich gut tracken lassen(konstant sichtbar), wird ein Bruteforce-Ansatz genutzt: 1. verfolge alle Pixel, 2. messe die Fehler der Trajektorien, 3. weise jedem Pixel den entsprechenden Fehlerwert zu. Daraus wird eine Fehlerkarte(error map) erstellt und über einen Schwellenwert die besten Pixel ausgewählt. Das Prinzip wird Abbildung 3.4 veranschaulicht.
Der Median-Flow Tracker verwendet diesen Ansatz in einer vereinfachten Form. Eine ROI wird über eine Bounding Box ausgewählt und die enthaltenen Pixel initialisiert. Diese Pixel werden ein Frame t_{+1} mit einem Lucas-Kanade Tracker vorwärts und rückwärts verfolgt. Für den FB wird der euklidische Abstand verwendet. Zusätzlich wird die Normalisierte Kreuzkorrelation (NCC) verwendet, da gezeigt wurde, dass beide unabhängig voneinander Pixel bewerten können und die Performance signifikant gesteigert wird. Diese beiden Metriken (FB und NCC) werden auf die Trajektorien angewendet, um eine Fehlerkarte zu erstellen, anhand derer zusammen mit einem Schwellenwert die besten Pixel ausgewählt werden. Die neue Position (Zentrum) der Region of Interest (ROI) ergibt sich aus den Median der übrigen Punkte in jeder Dimension. Für die Skalierung werden Punktepaare gebildet. Über diese Paare wird das Verhältnis zwischen dem Abstand in den Frames t und t_{+1} bestimmt. Der Skalierungsfaktor ergibt sich aus dem Median über diese Verhältnisse.

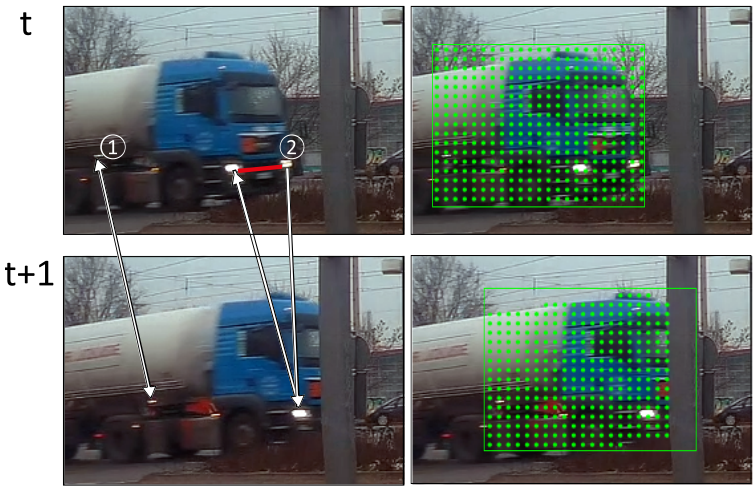

Abbildung 3.4: **Links:** Punkte (1),(2) werden von t zu t_{+1} und zurück verfolgt. Bei (1) ist die Position identisch, der Fehler ist Null. Bei (2) ergibt sich durch Verdeckung eine fehlerhafte Trajektorie, der Fehler(Abstand) wird durch eine rote Linie gekennzeichnet. **Rechts:** Die zugehörigen Punktmasken. Fehlerhafte Punkte werden, durch einen zu großen Fehler, nicht verfolgt.

Online Boosting Tracker

In [GGB06, GB06] wird eine online Variante des AdaBoost Frameworks [Fre95] vorgestellt. Das AdaBoost Framework für binäre Klassifikation basiert auf der Annahme, dass eine Kombination aus guten schwachen Klassifikatoren (low-level Features) einen starken Klassifikator (high-level Feature) bilden kann. Zum Trainieren der Klassifikatoren werden neben dem positiven Sample $\langle \mathbf{x}, y = 1 \rangle$ gleichgroße negative Samples$\langle \mathbf{x}, y = -1 \rangle$ aus dessen Umgebung genommen, um eine möglichst diskriminative Beschreibung zwischen Objekt und Umgebung zu erhalten. Somit soll ein Drift der ROI verhindert werden. Hierfür benötigen wir folgende Definitionen:

Schwacher Klassifikator h^{weak}: Die Fehlerrate für eine Entscheidung in einem binären Problem liegt leicht unter der von zufälligem Raten ($< 50\%$). Ein schwacher Klassifikator stellt bezüglich eines Features die Hypothese h^{weak} auf. Jeder schwache Klassikator besitzt zwei Gewichte $\lambda^{corr}, \lambda^{err}$, welche die Güte des Klassifikators angeben.

Selektor h^{sel}: Wählt aus einer Menge M von schwachen Klassifikatoren den mit der für $h_m^{weak}(m \in M)$ geringsten Fehlerwahrscheinlichkeit e_m.

$$h^{sel}(\mathbf{x}) = h_{m_{opt}}^{weak}(\mathbf{x}), \qquad m_{opt} = \arg min_i(e_i) \tag{3.25}$$

Starker Klassifikator h^{Strong}: Stellt eine Linearkombination von n Selektoren aus der Menge N der zugehörigen schwachen Klassifikatoren dar. α ist das spezifische Gewicht eines Selektors, sodass gute Selektoren, entsprechend ihres geschätzten Fehlers, stärker gewichtet werden.

$$h^{Strong} = \text{sign}\left(\sum_{n=1}^{N} \alpha_n \cdot h_n^{sel}(\mathbf{x})\right) \tag{3.26}$$

Weiterhin haben wir ein Signifikanz-Gewicht λ, welches die Signifikanz eines Samples, in Abhängigkeit des Fehlers des Selektors, darstellt. Initial ist für jedes Sample $\lambda = 1$.
Der Unterschied zwischen online und offline Boosting besteht in der Art des Trainings. Bei dem offline Verfahren werden die Klassifikatoren/Selektoren an einem einzigen Zeitpunkt an vielen klassifizierten Samples vor-trainiert, wohingegen bei dem online Verfahren die Selektoren zu jedem Zeitpunkt t lediglich an dem einen positiven Sample und dessen negativer Umgebung trainiert werden. Das online Verfahren ist somit robust gegen Veränderungen der Objektrepräsentation oder des Hintergrundes zur Laufzeit.
Die schwachen Klassifikatoren werden aus Performance-Gründen alle in einem zentralen Pool verwaltet, somit müssen diese pro Sample nur einmal aktualisiert werden. Die Selektoren haben jeweils vollen Zugriff auf den gesamten Pool. Mit jeder Iteration $n \in N$ für einen Selektor $h_n^{sel}(\mathbf{x})$ werden die schwachen Klassifikatoren anhand der Features des aktuellen Samples bewertet:

$$\lambda_{n,m}^{corr} \mathrel{+}= \begin{cases} \lambda, & \textbf{wenn } h_{n,m}^{weak}(\mathbf{x}) = y \\ 0, & \textbf{sonst} \end{cases} \qquad \lambda_{n,m}^{wrong} \mathrel{+}= \begin{cases} \lambda, & \textbf{wenn } h_{n,m}^{weak}(\mathbf{x}) \neq y \\ 0, & \textbf{sonst} \end{cases} \tag{3.27}$$

und die Fehlerwahrscheinlichkeit $e_{n,m}$ unter Beachtung von λ aufgestellt:

$$e_{n,m} = \frac{\lambda_{n,m}^{wrong}}{\lambda_{n,m}^{corr} + \lambda_{n,m}^{wrong}} \tag{3.28}$$

Der Selektor h_n^{sel} wählt nun entsprechend Gleichung 3.25 den besten schwachen Klassifikator aus und erhält das Gewicht α_n:

$$\alpha_n = \frac{1}{2} \ln \frac{1 - e_n}{e_n} \tag{3.29}$$

wobei e_n der auf den Selektor übertragene Fehler des gewählten Klassifikators ist. Anschließend wird λ angepasst, um bei dem nächsten Selektor schwer zu klassifizierenden Samples mehr Gewicht zu verleihen:

$$\lambda = \begin{cases} \lambda \cdot \frac{1}{2(1-e_n)}, & \textbf{wenn } h_n^{sel}(\mathbf{x}) = y \\ \lambda \cdot \frac{1}{2e_n}, & \textbf{sonst} \end{cases} \tag{3.30}$$

Somit wird verstärkt darauf trainiert, Fehler zu vermeiden.
Um möglichst vielseitige schwache Klassifikatoren zu erhalten, wird der schlechteste am Ende der Iteration gegen einen zufälligen neuen ersetzt. Nachdem alle Selektoren berechnet wurden und der Prozess für alle Samples durchgeführt wurde, erhalten wir den neuen starken Klassifikator h^{Strong} entsprechend Gleichung 3.26. Anzumerken ist hierbei, dass die Selektoren mit jedem Sample überschrieben werden, es kommt jedoch zu keinem Informationsverlust, da am gesamten Datensatz trainiert wird und die Selektoren unabhängig von dem aktuellen Sample arbeiten. Der Algorithmus wird in Quelltext 1.1 zusammengefasst.
Für das Tracking wird nun mit dem starken Klassifikator durch Template Matching in der Suchregion um das positive Sample eine *confidence map* erstellt, dessen Maxima die neue Position des Trackers darstellt. Für die erste Initialisierung ($t = 0$) wird

der Algorithmus mehrfach durchlaufen, um ein erstes, stabiles Modell zu erhalten. Durch das regelmäßige Update der Selektoren auf andere Feature(-klassen) ist das Verfahren robust gegenüber einer Veränderung der Objektrepräsentation oder des Hintergrundes. Eine Veranschaulichung des Prinzips wird in Abbildung 3.5 beschrieben.

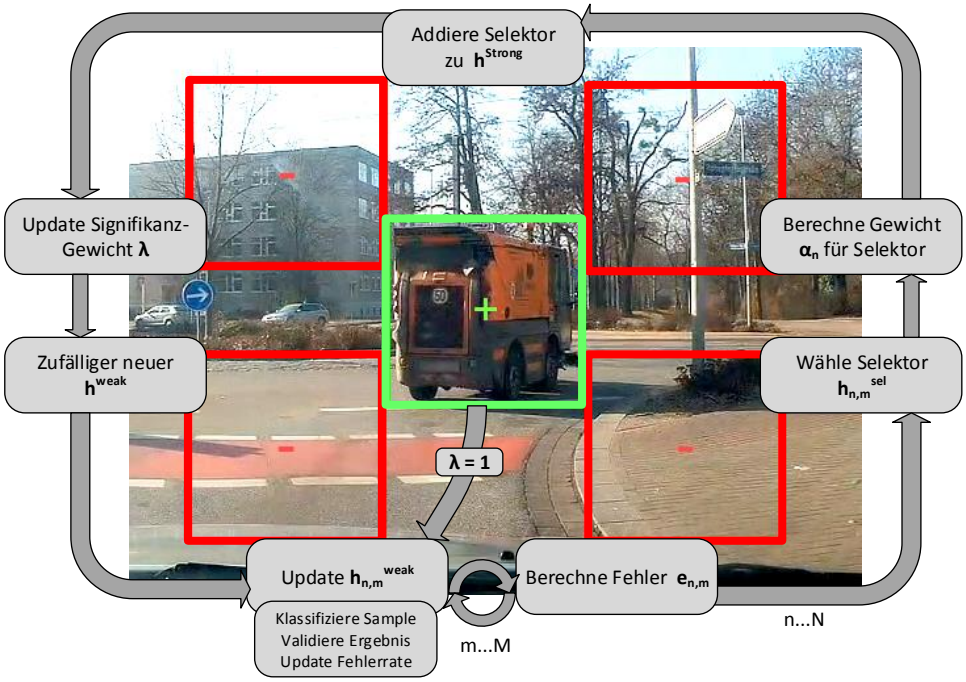

Abbildung 3.5: Der Update Algorithmus von Online-Boosting. Es werden Samples um die Trackerposition genommen und daran das Objektmodell trainiert und aktualisiert.

Verwendete Features

Zur Beschreibung der Samples dienen die in Abschnitt 3.3.3 vorgestellten HAAR-like Features, HOG und LBPs. Um eine Hypothese für h^{weak} zu generieren, wird eine Wahrscheinlichkeitsverteilung für positive und negative Samples erstellt. Die Wahrscheinlichkeitsdichte wird durch eine Kalman-Filter Technik geschätzt. Dabei wird für die HAAR-like Features ein einfacher Schwellenwert oder Bayes'sches Entscheidungskriterium und für HOG und LBPs ein *nearest neighbor* Lernverfahren verwendet. Eine genaue Beschreibung der Metriken wird in [GB06] erläutert.

MIL-Track

Babenko et al. stellten in [BYB09] eine online Variante des bis dahin offline genutzten MILBoost Algorithmus vor. Dieser basiert auf Mulitple Instance Learning (MIL) und einer wie in Abschnitt 3.3.3 adaptiven online Anpassung des Objektmodells. MIL basiert, im Gegensatz zu Abschnitt 3.3.3 nicht auf einzelnen Trainingsbeispielen, sondern auf Sets bzw. Bags. Diese Bags stellen eine Sammlung von einzelnen Trainingsinstanzen(Beispielen) dar. Das Multiple-Instance-Problem besteht nun darin,

dass eine Bag positiv kategorisiert wird, sobald mindestens eine Bag-Instanz positiv ist, anderenfalls negativ. Die Kategorien der Bag-Instanzen sind jedoch unbekannt. Somit muss zum Lösen des Problems festgestellt werden, welche Instanz(en) positiv sind. Um Trainingsbeispiele zu generieren, werden um die aktuelle Position positive Samples $\langle \mathbf{x}, y \rangle$ (ROI enthalten) in einem kleinen Radius r ausgeschnitten. Diese positiven Samples kommen in eine gemeinsame positive Bag $(X^r, y = 1)$. Nun werden in einem größeren Radius β negative Samples ausgeschnitten und jeweils in eine negative Bag $(X^{r,\beta}, y = 0)$ gelegt. Die Features entsprechen Haar-like Features, Bag-Instanzen sind somit Featurevektoren. Der Vorteil ist, dass die Position des Trackers nicht optimal sein muss, um das Objekt als positives Beispiel zu erfassen, und somit robuster ist. Das Prinzip ist in Abbildung 3.6 zu sehen.

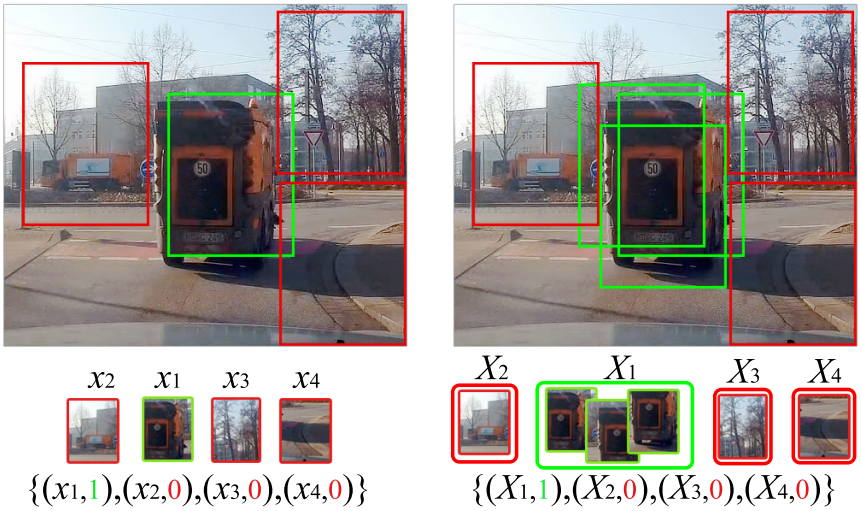

Abbildung 3.6: **Links:** klassisches Sampling mit nicht optimaler Position der ROI. **Rechts:** MIL Sampling mit Bags. Es werden mehrere positive Samples genommen.

Ähnlich zu Abschnitt 3.3.3 gibt es M schwache Klassifikatoren h^{weak} und einen starken Klassifikator h^{Strong}, welcher sich aus einer Linearkombination von N Klassifikatoren schwachen zusammensetzt. Da die Kategorie und Wahrscheinlichkeit eines Samples zur Trainingszeit nicht bekannt ist, muss diese geschätzt werden, um so die Wahrscheinlichkeit der Bag zu schätzen. Der Algorithmus geht hierbei wie folgt vor: für jedes Sample $\langle \mathbf{x}_{i,j}, y_i \rangle$, wobei i die Bag ist und j die Baginstanz, werden alle M schwachen Klassifikatoren aktualisiert:

$$\mu_1 \leftarrow \gamma\mu_1 + (1-\gamma)\frac{1}{n}\sum_{i|y_i=1} f_m(x_i) \tag{3.31}$$

$$\sigma_1 \leftarrow \gamma\sigma_1 + (1-\gamma)\sqrt{\frac{1}{n}\sum_{i|y_i=1}(f_m(x_i)-\mu_1)^2} \tag{3.32}$$

Wobei f_m das Feature zu h_m^{weak} ist und dessen Wahrscheinlichkeit als Normalverteilung $\mathcal{N}_1(\mu_1, \sigma_1)$ modelliert wird. $\mathcal{N}_1(\mu_1\sigma_1)$ wird entsprechend $y = 1$ dabei für positive Samples und $\mathcal{N}_0(\mu_0\sigma_0)$ für negative verwendet und γ stellt die Lernrate dar. Somit besteht Ähnlichkeit zu dem Lernmodell der schwachen Klassifikatoren aus Abschnitt 3.3.3. Nun werden N aus M schwachen Klassifikatoren gewählt, ähnlich

der Selektoren aus Abschnitt 3.3.3, weshalb diese, der Einfachheit halber, im Folgenden so bezeichnet werden. Für alle $n \in N$ Selektoren h_n^{sel} wird für alle $m \in M$ schwachen Klassifikatoren iterativ die Instanz-Wahrscheinlichkeit $p_{i,j}^m$ bestimmt:

$$p_{i,j}^m = \sigma\left(h_{i,j}^{Strong} + h_m^{weak}(x_{i,j})\right) \tag{3.33}$$

$\sigma(x)$ ist die Sigmoid Funktion, wodurch das Gewicht von h_m^{weak} direkt angegeben wird:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.34}$$

Die Wahrscheinlichkeit $h_m^{weak}(x_{i,h})$ wird durch das logarithmierte Wahrscheinlichkeitsverhältnis der beiden Einzelwahrscheinlichkeiten(Sample = positiv, Sample = negativ), unter dem Satz von Bayes, gebildet:

$$h_m^{weak}(x) = \log\left[\frac{p_t(y=1|f_m(x))}{p_t(y=0|f_m(x))}\right] \tag{3.35}$$

Dabei gilt $p_t(f_t(x)|y=1) \sim \mathcal{N}(\mu_1, \sigma_1)$ und äquivalent für $y=0$. Ausgehend von der Instanz-Wahrscheinlichkeit können wir die Bag-Wahrscheinlichkeit p_i^m über das Noisy-OR(NOR) Modell berechnen:

$$p_i^m = 1 - \prod_j (1 - p_{i,j}^m) \tag{3.36}$$

Ist die Wahrscheinlichkeit einer Instanz hoch, ist es auch die der gesamten Bag. Nun wird die Log-Likelihood ℓ^m des Klassifikators über die Wahrscheinlichkeit aller Bags gebildet:

$$\ell^m = \sum_i \left(y_i \log(p_i^m) + (1-y_i)\log(1-p_i^m)\right) \tag{3.37}$$

Wenn dies für alle $m \in M$ geschehen ist, wird der beste schwache Klassifikator $h_{m_{opt}}^{weak}$ anhand von

$$m_{opt} = \arg max_m \ell^m \tag{3.38}$$

als Selektor h_n^{sel} gewählt und dem starken Klassifikator hinzugefügt. Der gesamte Vorgang wird für alle N Selektoren wiederholt. Der starke Klassifikator wird daher gebildet durch:

$$h^{Strong}(x_{i,j}) = \sum_n h_n(x_{i,j}) \tag{3.39}$$

Anzumerken ist, dass, wie schon bei Abschnitt 3.3.3, der starke Klassifikator mit jedem Sample neugebildet wird, jedoch, durch das Training am ganzen Datensatz und dem Speicher der schwachen Klassifikatoren, der starke Klassifikator nicht für ein Sample spezifisch ist. Der Algorithmus wird in Quelltext 1.2 zusammengefasst.

Bei dem Tracking wird nun von Zeitschritt t zu $t+1$ anhand des Klassifikators h^{Strong} die neue Position $l(t+1)$, in einem Suchfenster s um die alte Position $l(t)$, über Template Matching gesucht. Die neue Position $l(t+1)$ ergibt sich über das Maxima aus dem Template Matching. Anschließend wird das Objektmodell mit neuen Samples von $l(t+1)$ aktualisiert.

KCF

In [HCMB12, HCMB15] wird ein Trackingansatz ohne klassisches Sammeln von Samples vorgestellt. Der Hauptkritikpunkt an den vorher beschriebenen Algorithmen besteht in dem Undersampling von negativen Beispielen für das Training. Der Ansatz von Kernalized Correlation Filters (KCFs) besteht in dem Sammeln von extrem großen Mengen negativer Samples. Dies wird durch zyklische Verschiebungen des positiven Samples in einem Suchraum realisiert. Ein geeigneter Klassifikator wird mit Hilfe einer Kernelregression gefunden. Die neue Position des Trackers wird durch Korrelation gesucht. Um diese Operationen bei den großen Datenmengen effizient zu berechnen, finden diese im Fourierraum statt. Die Grundlage ist das sogenannte 'dichte' (dense) Sampling. Es werden nicht nur einige (wenige) Samples um die ROI herangezogen, sondern eine Vielzahl aus einem Fenster um die aktuelle Position. Hierfür werden zirkulante bzw. zyklische Matrizen genutzt.

Zyklische Verschiebung: Das positive Basissample ist hierbei doppelt so groß wie die ROI und wird als Vektor $\mathbf{x}$ dargestellt. Für eine verständliche Erklärung werden nun 1D-Signale betrachtet, das Verfahren lässt sich aber ebenso auf höherdimensionale Signale anwenden. $\mathbf{x}$ stellt somit einen $n \times 1$ Vektor dar. Um negative Samples zu generieren, wird das Signal $\mathbf{x}$ durch Multiplikation mit einer Permutationsmatrix P um eine Stelle zyklisch verschoben. Um über größere Distanzen u Verschiebungen zu erhalten, nutzen wir die Potenz $P^u(\mathbf{x})$. Nach n Verschiebungen erhalten wir wieder das ursprüngliche Signal. Auf dieser Basis können wir nun eine zirkulante Matrix X erstellen:

$$X = C(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \\ x_n & x_1 & x_2 & \dots & x_{n-1} \\ x_{n-1} & x_n & x_1 & \dots & x_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_2 & x_3 & x_4 & \dots & x_1 \end{bmatrix} \tag{3.40}$$

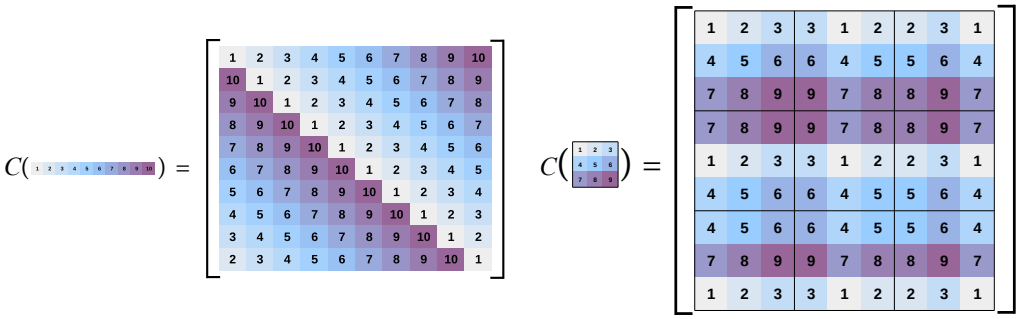

Abbildung 3.7: Visualisierung von zirkulanten Matrizen. **Links:** zirkulante Matrix aus 1D Vektor. **Rechts:** Block-zyklische zirkulante Matrix aus 2D Matrix.

Anzumerken ist, dass $C(\mathbf{x})$ nicht explizit berechnet werden muss, da die Matrix allein durch $\mathbf{x}$ und dessen Permutation definiert wird. Eine Veranschaulichung ist in Abbildung 3.7 zu sehen. Im zweidimensionalen Fall ergibt dies eine "Block-zyklische zirkulante Matrix die einzelnen zirkulanten Bilder(Matrizen) werden zyklisch in einer Matrix angeordnet. Eine zirkulante Matrix wird durch die Diskrete Fourier Transformation (DFT) diagonalisiert. Da die DFT eine lineare Operation ist, kann sie durch

eine konstante und vom Datensatz unabhängige Matrix F beschrieben werden. Für einen Vektor $\mathbf{x}$ im Fourierraum verwenden wir die Bezeichnung $\hat{\mathbf{x}} = \mathcal{F}(\mathbf{x}) = \sqrt{n}F\mathbf{x}$. Die Matrix X lässt sich somit darstellen als Eigenwertzerlegung:

$$X = F \operatorname{diag}(\hat{\mathbf{x}}) \, F^H \tag{3.41}$$

Wobei F^H die hermitische (komplex konjugierte) transponierte Matrix von F ist. Der Vorteil von diagonalisierten Matrizen ist, dass wir alle Operationen elementweise und somit sehr effizient ausführen können.

Kernelregression: Durch die zirkulante Matrix erhalten wir einen enorm großen Datensatz, auf welchem wir durch Regression einen Klassifikator berechnen, welcher in der Lage ist, diese Daten zu unterscheiden. Hierfür wird die Kernelized Ridge Regression bzw. KRLS verwendet. Unter der Bedingung, dass die Kernelmatrix K zirkulant ist, können signifikante Beschleunigungen vorgenommen werden. In [HCMB15] wird bewiesen, dass für bestimmte Kernel gilt: wenn die Eingangsdaten zirkulant sind, so ist es auch die Kernelmatrix K. Zu diesen Kernelklassen zählen unter anderem Radiale Basis Funktion (RBF) Kernel, wie der häufig verwendete Gauß-Kernel. Zirkulante Matrizen werden durch die DFT, wie oben beschrieben, diagonalisiert und können Operationen elementweise ausführen. Unter diesem Umstand ist es möglich, die KRLS signifikant zu beschleunigen. Wenden wir die DFT auf Gleichung 3.18 an, erhalten wir aus Gleichung 3.41:

$$\hat{\mathbf{w}} = \left(F \operatorname{diag}(\hat{\mathbf{x}}^*) \, F^H F \operatorname{diag}(\hat{\mathbf{x}}) \, F^H + \lambda I\right)^{-1} \left(F \operatorname{diag}(\hat{\mathbf{x}}^*) \, F^H\right)^H \hat{\mathbf{y}} \tag{3.42}$$

$F^H F$ bildet die Einheitsmatrix I und die hermitische Transponierte einer zirkulanten Matrix X bildet die komplex-konjugierte Form $\hat{\mathbf{x}}^*$. Weiterhin können wir die Multiplikation zweier Diagonalmatrizen elementweise mit $\odot$ durchführen:

$$\hat{\mathbf{w}} = \frac{\hat{\mathbf{x}}^* \odot \hat{\mathbf{y}}}{\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}} + \lambda} \tag{3.43}$$

Übertragen auf einen nicht-linearen Kernel aus Gleichung 3.21 erhalten wir:

$$\hat{\boldsymbol{\alpha}} = \frac{\hat{\mathbf{y}}}{\hat{\mathbf{k}}^{xx} + \lambda} \tag{3.44}$$

Wobei $\hat{\mathbf{k}}^{xx}$ die erste Zeile der Matrix $K = C(\mathbf{k}^{xx})$ und die Kernelkorrelation (ähnlich NCC in Abschnitt 3.3.3) eines Vektors $\mathbf{x}$ mit sich selbst darstellt, weil die zirkulante Matrix K die Verschiebung eines Vektors $\mathbf{x}$ beschreibt:

$$k_i^{xx} = \kappa(\mathbf{x}, P^{i-1}\mathbf{x}) = \varphi^T(\mathbf{x})\varphi(P^{i-1}\mathbf{x}) \tag{3.45}$$

In Bezug auf die Korrelation spricht man auch von Autokorrelation eines Signales mit sich selbst, in diesem Fall Kernel-Autokorrelation. Statt einer $n \times n$ Matrix K muss nun nur noch ein $n \times 1$ Vektor im Fourierraum berechnet werden. Die Kernel selber können effizient im Fourierraum berechnet werden, da die zugrundeliegenden Werte aus der zirkulanten Matrix stammen und somit diagonalisiert werden, wodurch elementweise Operationen möglich sind. Somit ergibt sich für einen Gauß-Kernel:

$$\mathbf{k}^{xx'} = \exp\left(-\frac{1}{\sigma^2}\left(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\mathcal{F}^{-1}(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}}')\right)\right) \tag{3.46}$$

Durch die Exploration der Daten im Dualspace ist es möglich Mehrkanal-Features, wie HOG, zu nutzen. Im Fourierraum müssen hierfür die Kanäle c lediglich addiert werden, somit wird Gleichung 3.45 abgewandelt in:

$$\mathbf{k}^{xx'} = \exp\left(-\frac{1}{\sigma^2}\left(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\mathcal{F}^{-1}\left(\sum\nolimits_c \hat{\mathbf{x}}_c^* \odot \hat{\mathbf{x}}_c'\right)\right)\right) \tag{3.47}$$

Eine weitere Variante ist die Verwendung von einem linearen Kernel im Dualspace, welcher zu einem stark verringerten Rechenaufwand und schlechterem Klassifikator führt. In der Praxis sind die Unterschiede des Klassifikators jedoch gering [HCMB15], sofern mehrere Kanäle, wie z.B. HOG, verwendet werden. Diese Variante des Filters wird Dual Correlation Filter(DCF) genannt. Der lineare Kernel wird für mehrere Kanäle ebenfalls durch die Summe gebildet:

$$\mathbf{k}^{xx'} = \mathcal{F}^{-1}\left(\sum\nolimits_c \hat{\mathbf{x}}_c^* \odot \hat{\mathbf{x}}_c'\right) \tag{3.48}$$

Detektion: Aufgrund der obigen Berechnungen, ist es nun möglich, die neue Position des Objektes zum Zeitpunkt $t+1$ zu berechnen. Hierfür nutzen wir den Klassifikator aus der Kernelregression $f(\mathbf{z})$. $K^{\mathbf{z}}$ ist die zirkulante Kernelmatrix zwischen allen Trainingssamples und allen Lösungskandidaten, welche aus zyklischen Verschiebungen der Basisvektoren $\mathbf{x}$ und $\mathbf{z}$ hervorgehen. Somit lässt sich, analog zu Gleichung 3.45, die Kernelkorrelation $K^{\mathbf{z}} = C(\mathbf{k}^{\mathbf{xz}})$ definieren. Aus Gleichung 3.22 ergibt sich damit:

$$\mathbf{f}(\mathbf{z}) = (K^{\mathbf{z}})^T\boldsymbol{\alpha} \quad \Leftrightarrow \quad \hat{\mathbf{f}}(\mathbf{z}) = \hat{\mathbf{k}}^{\mathbf{xz}} \odot \hat{\boldsymbol{\alpha}} \tag{3.49}$$

Die neue Position ergibt sich aus dem Maximum von $\hat{\mathbf{f}}(\mathbf{z})$ bzw. dessen inverser Fourier-Transformation. An der neuen Position wird das Modell nun mit neuen Samples aktualisiert. Der Tracker verfügt hierbei über ein Gedächtnis, da die neuen und alten Werte für $\boldsymbol{\alpha}$ und $\mathbf{x}$ linear interpoliert werden.
Anzumerken ist, dass die Fourier-Transformation periodisch ist. Somit müssen Diskontinuitäten an den Signalgrenzen verhindert werden. Dies wird durch ein Fenster auf Basis einer (Ko-)Sinus Funktion erreicht, welches die Signalgrenzen mit Null gewichtet. x^{raw} stellt hierbei das ursprüngliche 2D-Signal(Bild) dar.

$$x_{ij} = (x_{ij}^{raw} - 0.5)\sin(\frac{\pi i}{n})\sin(\frac{\pi j}{n}) \tag{3.50}$$

Weiterhin werden auf die Regressionsziele $\mathbf{y}$ Gaußfunktionen angewendet, wodurch es sich nicht mehr um ein binäres Klassifikationsproblem handelt, sondern reelle Werte ausgegeben werden. Dies ist möglich, da die KRLS kontinuierliche Daten berechnen kann. Hierdurch steigt die Präzision der Detektion und zusätzlich werden Ringing-Artefakte in dem Fourierraum verhindert.

TLD

Tracking-Learning-Detection (TLD) wurde 2011[Kal11] bzw. 2012[KMM12] vorgestellt. Es basiert auf einem Ansatz, welcher das Tracking in drei Aufgaben unterteilt: Tracken, Lernen, Detektieren. Alle drei Teilbereiche sollen ineinandergreifen und sich somit ergänzen. Die verwendete OpenCV[Its16] Implementierung bietet leider nur

eine sehr schlechte Performance (Geschwindigkeit und Tracking, siehe Evaluierung in Abschnitt 6.1) und ist somit für das Ziel, ein echtzeitfähiges Annotationssystem zu erstellen, ungeeignet. An dieser Stelle wird deshalb nur auf abstrakter Ebene auf den Algorithmus eingegangen.
Die Aufgaben der drei Teilgebiete stellen sich wie folgt auf:

- Tracker: klassisches Trackingsystem zum Erstellen einer Trajektorie
- Lernen: bewertet Detektor-Fehler und aktualisiert diese
- Detektor: korrigiert Tracker

Der Tracker verfolgt hierbei das Objekt von Frame zu Frame unter Nutzung des Median-Flow Trackers aus Abschnitt 3.3.3. Der Detektor sucht im gesamten Bild nach bekannten (erlernten) Objekten. Beide Module werden durch den Integrator verbunden: das Objekt wird als nicht-sichtbar klassifiziert, wenn keines der Module das Objekt erkennt, anderenfalls bestimmt das Modul mit der größeren Wahrscheinlichkeit die aktuelle Position, wobei der Tracker hier primär bevorzugt wird und erst bei größeren Abweichungen(abhängig von der Überlappung) der Detektor akzeptiert wird. Es wird das erlernte a priori Wissen genutzt, um möglichst schnell negative Kandidaten zu verwerfen. Dafür wird ein dreistufiges System verwendet, um möglichst effizient, vor allem den Hintergrund, zu verwerfen. Jede Stufe ist dabei spezifischer als die vorherige, somit muss die erste bereits einen Großteil der negativen Daten verwerfen können. Die erste Stufe basiert dabei auf den Unterschieden der Pixelwerte in Patches (Patch Variance). Die zweite Stufe besteht aus einer Komposition von schwachen Klassifikatoren, ähnlich Abschnitt 3.3.3. Die letzte Stufe verwendet einen Nearest-neighbor Klassifikator, basierend auf der Ähnlichkeit von benachbarten Patches. Beim Lernen wird der Fehler des Detektors, ähnlich der Selektoren in Abschnitt 3.3.3, bewertet und dessen Modell aktualisiert/korrigiert. Der Lernschritt wird als P-N-Lernen beschrieben. Hierbei gibt es zwei Klassifikatoren: P- und N-Experte. Der P-Experte ist hierbei spezialisiert auf positive Kandidaten, er untersucht daher alle negativen Kandidaten auf falsch negative. Er basiert auf dem Tracker und der Annahme, dass ein Objekt eine bestimmte Trajektorie verfolgt und somit in dessen Nähe sein muss. Der N-Experte ist hingegen spezialisiert auf negative Kandidaten und untersucht alle positiven Kandidaten auf falsch positive. Er basiert auf dem Detektor und sucht das Objekt mit der größten Wahrscheinlichkeit für das Objektmodell. Der P-Experte erhöht dabei die Sensitivität und der N-Experte die Spezifität. Das Objektmodell besteht aus positiven und negativen Patches. Die positiven sind eine Untermenge der ROI, die negativen eine Untermenge der Umgebung. Während des Trainings werden gelabelte Trainingsdaten in ein gemeinsames Trainingsset gelegt und die Parameter für den Klassifikator werden erzeugt. Zusätzlich gibt es noch nicht-klassifizierte Daten. Ein Klassifikator klassifiziert nun die unbekannten Daten. Die P-N Experten bewerten(schätzen) anschließend alle (unbekannten) klassifizierten Daten und fügen diese mit einem neuen Label (positiv|negativ) dem Trainingsset zu, sofern sie falsch klassifiziert wurden. Das Training wird nun auf dem aktualisierten Datensatz ausgeführt und neue Parameter werden erzeugt. Der Prozess wird nun solange iterativ wiederholt, bis es konvergiert oder

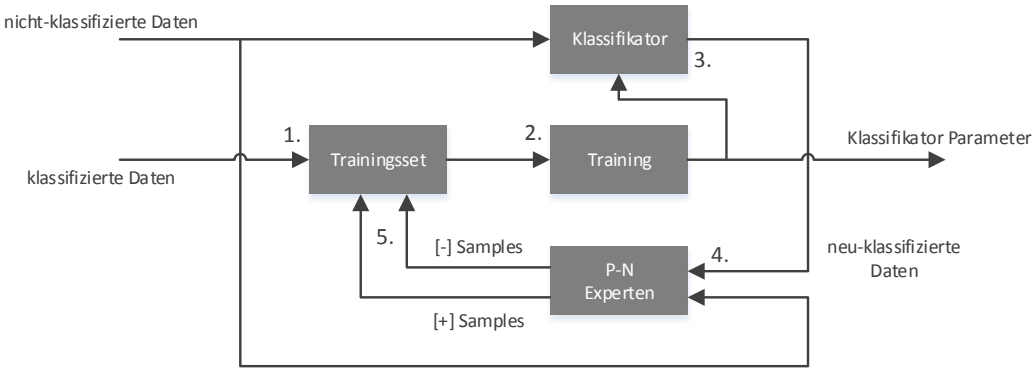

Abbildung 3.8: P-N-Lernen. **1:** Initialisieren des Trainingssets. **2:** Trainieren eines Klassifikators(Parameter) mit dem Trainingsset. **3:** Klassifizieren der unbekannten Daten. **4:** Bewerten(schätzen) von fehlerhaft klassifizierten Daten. **5:** Hinzufügen von umgelabelten, fehlerhaft klassifizierten Daten zu dem Trainingsset.

ein Haltekriterium erreicht wurde. Abbildung 3.8 zeigt eine Übersicht über das P-N-Lernprinzip.

GOTURN

Generic Object Tracking Using Regression Networks (GOTURN) ist ein im Jahr 2016 veröffentlichter Tracker [HTS16]. Er basiert als einziger, in dieser Arbeit betrachteten Tracker, auf CNNs und stellt somit eine interessante Brücke zwischen dem Anwendungsziel der Arbeit, dem Erzeugen von Trainingsdaten für Deep Learning und der Basis zum Erzeugen von Trainingsdaten, dem Tracking, dar. Die in *Opencv*[Its16] implementierte Variante ist CPU-basiert und bietet durch die mangelnde Parallelisierung, wie auch in der CPU-Version in [HTS16], nur eine geringe Geschwindigkeit von ca. 2 Bildern pro Sekunde. Zusätzlich enthält die aktuelle Version in *OpenCV* noch Fehler, wodurch eine Evaluierung nicht möglich war. Deshalb wird hier, wie auch bei TLD, nur eine abstrahierte Übersicht gegeben.
Das Grundprinzip basiert auf einem vor-trainierten CNN, welches nicht die Objekte selbst, sondern ein generisches Bewegungsmodell erlernt. Auf Basis dieses Modells wird anschließend durch Regression die genaue Position in einem Suchraum um die ROI lokalisiert. Das Netz erhält hierbei als Input das aktuelle Frame t und das vorherige $t-1$. In beiden Frames wird eine Region an der Position der ROI von $t-1$, aber größer als diese, ausgeschnitten. Die ausgeschnittene Region dient als Objektmodell und stellt somit das zu trackende Objekt dar. Die beiden Ausschnitte gehen jeweils in getrennte Konvolutions-Schichten, welche unabhängig die High-Level Features zur Objektbeschreibung generieren. Diese Objektmodelle werden in einem einzelnen Feature-Vektor konkateniert und in voll-verknüpfte Schichten geleitet. Durch ein vor-trainiertes generisches Bewegungsmodell werden die Features-Sets verglichen und durch Regression die Position in t gefunden. Die Ausgabe des Netzes sind die beschreibenden Eckpunkte der Bounding Box. Der Aufbau des Netzes wird in Abbildung 3.9 verdeutlicht.

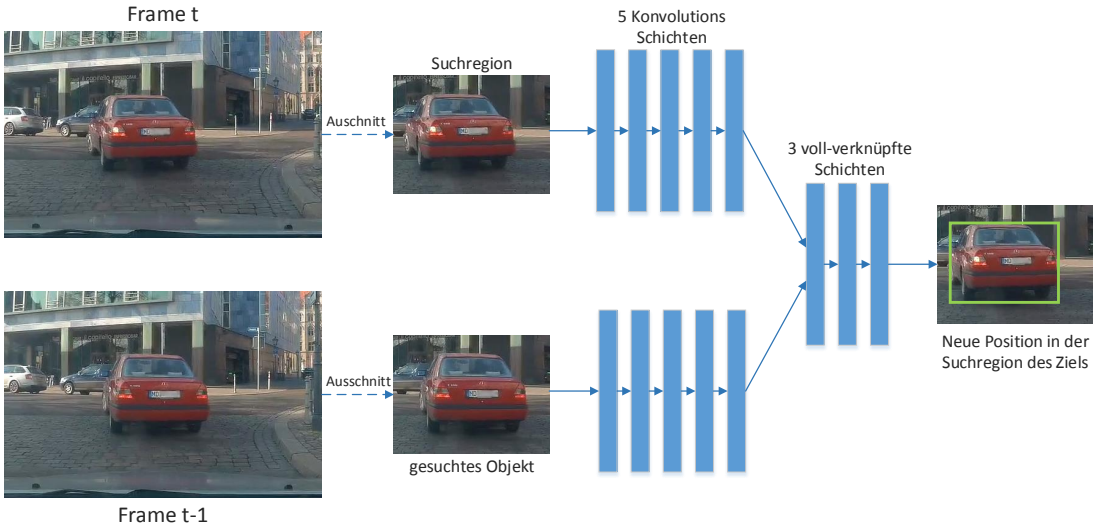

Abbildung 3.9: Aufbau und Funktionsweise des CNNs von GOTURN

Das Netz basiert auf der Annahme, dass Bewegungen weich sind und somit zwischen zwei Frames keine erheblichen lokalen Unterschiede auftreten. Dementsprechend wird der Suchraum gewählt. Er bestimmt somit direkt die Komplexität des Netzes. Um möglichst robust gegenüber Transformationen und Belichtungsunterschieden zu sein, wurde das Netz offline trainiert. Dafür wurden Videos und transformierte Einzelbilder genutzt. Die Videos lieferten hierbei natürliche Bewegungsänderungen, wohingegen die Einzelbilder die Robustheit bzw. Generalisierbarkeit erhöhen. Die Einzelbilder wurden hierfür, ausgehend von weichen Bewegungen, durch eine Laplace Verteilung unterschiedlich ausgeschnitten, skaliert und verschoben. Da der Tracker auf einem neuronalen Netz basiert, ist die Berechnung, trotz offline Trainings, entsprechend aufwändig und nur auf einer GPU in Echtzeit zu gewährleisten.

Die eben vorgestellten Tracking-Algorithmen wurden in einem Annotationsystem in Abschnitt 5.2 implementiert. Auf Basis der Evaluierung in Abschnitt 6.1 wird eine Empfehlung für die Verwendung spezifischer Tracker ausgesprochen.

4. Entwurf eines Annotationssystems

Bei dem konzeptionellen Entwurf eines Annotationssystems ist es wichtig, sich auf die Kernelemente zu fokussieren. Dazu zählen das Graphical User Interface (GUI), die interne Datenstruktur und das Datenexport-Format. Das GUI muss möglichst intuitiv sein und den Nutzer effektiv unterstützen, um die Arbeitsgeschwindigkeit zu verringern. Die interne Datenstruktur muss konsistent und schlank gehalten werden, da Videosequenzen tendenziell hohe Datenmengen bereithalten. So hat ein 5 minütiges Video mit einer Bildwiederholrate von 60 Bildern pro Sekunde bereits 18.000 Einzelbilder, zu welchen jeweils mehrere Annotationen gespeichert werden müssen. Der Datenexport muss ein Format haben, welches einfach zu importieren ist und ebenfalls eine schlanke Datenstruktur bietet.

4.1 User Interface

Das GUI stellt die Schnittstelle zwischen System und Anwender dar. Um diesem den Arbeitsfluss möglichst angenehm zu gestalten, müssen im Vorfeld bestimmte Anforderungen geklärt werden. Preim und Dachselt schlagen in [PD15] verschiedene Methoden vor. So wurde eine Beobachtung durchgeführt, in der mit bestehenden Systemen [VPR13, MD03, Mul10] repräsentative Videosequenzen mit Verkehrsdaten annotiert wurden. Hierbei wurden Probleme der linearen Interpolation(Abschnitt 3.2.1) ersichtlich, aber auch positive und negative Eigenschaften des Interfaces selbst. So werden in [VPR13] alle Objekte der Sequenz zu jeder Zeit in einer Auswahlliste angezeigt, was bei längeren oder komplexeren Sequenzen schnell zu Problemen der Übersicht führen kann. Weiterhin ist es dort nur möglich vordefinierte Labels zu erstellen. Das hat zum einen den Vorteil, dass der Nutzer nicht überfordert wird und klare Anweisungen erhält, zum Anderen verweigert man Experten-Nutzern die Möglichkeit, Informationen nachträglich hinzuzufügen. Weiterhin wurden zu den eigenen Beobachtungen potenzielle Nutzer aus verschiedenen Berufszweigen bzw. Studiengängen interviewt, um spezielle Anforderungen und Funktionswünsche zu

definieren. Dazu zählen die allgemeine Struktur des Datenexports, Interaktionsmöglichkeiten und Anordnung der Elemente für einen angenehmen Arbeitsfluss. Aus diesen Anforderungen wurde ein Prototyp entwickelt, welcher nach dem Feedback aus Testphasen zu einer Release-Version ausgebaut wurde. In [PD10] werden Empfehlungen bezüglich des Interface-Designs aufgestellt. So wurde, um die kognitive Belastung des Nutzers zu minimieren, das Interface möglichst einfach gehalten. Dazu zählen eine dezente, dunkle Farbgestaltung, um den Fokus auf die Videosequenzen zu lenken und die Augen des Anwenders bei intensiver und langer Nutzung zu schonen. Expertenfunktionen, wie der Import von verschiedenen Tracker-Konfigurationen, sind über den untersten Punkt der Menüleiste erreichbar. Sie sind für den Anwender somit nicht direkt ersichtlich, da sie einen unerfahrenen Nutzer schnell überfordern können und den Fokus von der eigentlichen Aufgabe ablenken. Wichtige Kernfunktionen hingegen wurden offensichtlich und nah bei einander platziert, um die Bewegungen innerhalb des Interfaces zu minimieren. Zusätzlich wurden Tastaturkürzel(Shortcuts) definiert, um ein beidhändiges Nutzererlebnis zu ermöglichen und die Bewegung signifikant zu verringern. Der Anwender nutzt dabei die primäre Hand für die präzise Interaktion mit dem Grafikfenster und die sekundäre Hand, um innerhalb der Videosequenz zu navigieren oder Objekte hinzuzufügen. Bei Nutzertests wurde diese Art der Interaktion sehr positiv aufgefasst. Für eine intuitive Bedienung wurden die Navigations- und Bedienelemente des Videofensters an die eines Media-Players angepasst, da diese sehr stark verbreitet sind und der Nutzer sein antrainiertes Wissen intuitiv anwenden kann. Bezüglich der Adaptierbarkeit ist das GUI skalierbar, wobei das Videofenster bewusst nicht direkt mit der Gesamtanwendung skaliert, sondern nur mit dessen Höhe. Somit kann der Nutzer bei Bedarf bestimmte, nicht notwendige, Elemente ausblenden, in dem das Videofenster diese Elemente überdeckt. Weiterhin existiert ein Splitter, um die Breite des Videobereichs bzw. der Listboxen(Objekte, Labels) anzupassen oder komplett auszublenden. ROIs können per Maus als Rechteck ausgewählt werden. Sie sind in ihrer Größe nachträglich skalier- und verschiebbar. In Abbildung 4.1 wird eine Übersicht des GUIs gezeigt.

Menüleiste Die Menüleiste beinhaltet grundlegende Steuerelemente, welche aber nicht für den fließenden Prozess notwendig sind oder für die Expertenwissen benötigt wird. So finden sich dort als Aktionen für Grundfunktionen das Laden von Videos, der Import von vorher gespeicherten Annotationsdaten und der Export von Annotationsdaten, wahlweise mit den Einzelbildern des Videos. Als Expertenfunktion ist der Import von Konfigurationsdaten für die implementierten Tracker hinterlegt.

Videobereich Der Videobereich stellt das zentrale Element der Anwendung dar. Hier befindet sich das Grafik- bzw. Videofenster, in welchem die Sequenz abgespielt wird. Um dieses herum befinden sich Elemente für die Steuerung des Videos. Diese beinhalten einen Schieberegler zum Steuern der Abspielgeschwindigkeit, eine Anzeige mit Eingabe für die aktuelle Framenummer und eine Trackbar mit Position und Zeitangaben für die aktuelle Stelle des Videos und zum schnellen Suchen/Springen zu Stellen. Springt der Nutzer an eine Stelle, wird für dieses Frame der Speicher geladen und eventuell getätigte Annotationen sind verfügbar. Weiterhin befindet

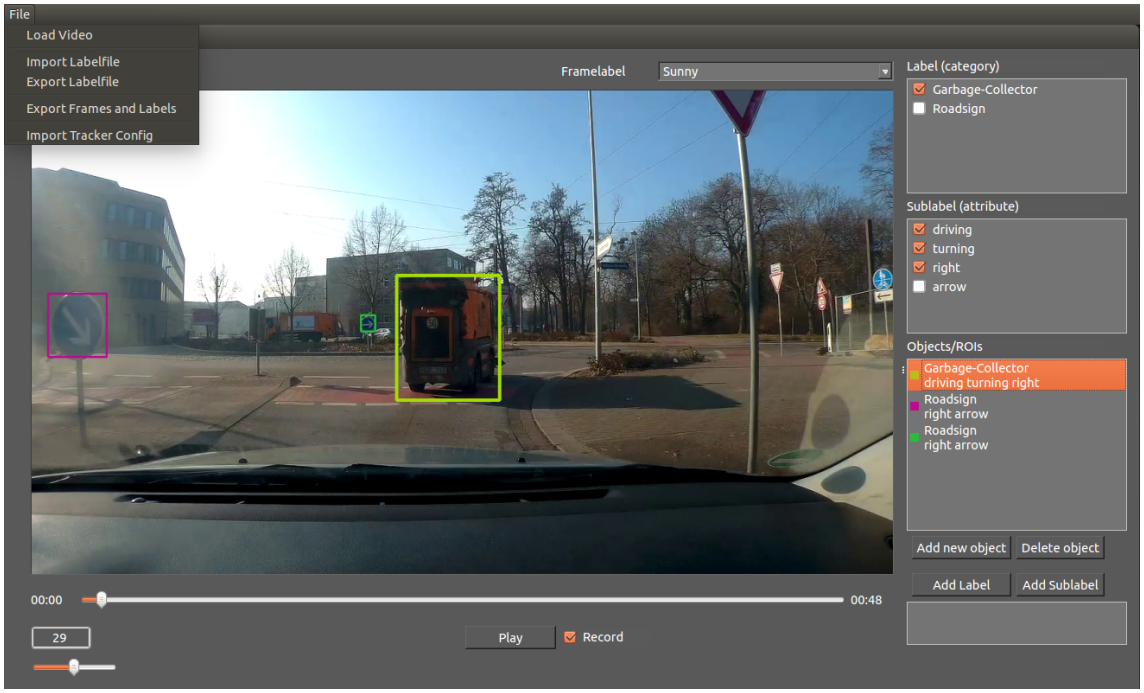

Abbildung 4.1: Übersicht über das Interface. **Oben-Links:** ausgeklappte Menüleiste, **Zentrum:** Videofenster mit ausgewählter ROI(gelb-grün), **Unten:** Steuerelemente, **Rechts:** Listen und zugehörige Buttons für Objekte und Labels

sich dort ein Play-Button zum Starten und Stoppen der Wiedergabe, sowie eine Checkbox(*Record*). Hier kann gewählt werden, ob der Tracker aktiv sein soll und der Speicher in dem aktuellen Frame be- bzw. überschrieben wird(*Tracking Modus*) oder ob der interne Speicher einfach abgespielt wird(*Wiedergabe Modus*). Der Nutzer kann sich somit bereits annotierte Szenen nochmals anschauen und gegebenenfalls Veränderungen vornehmen. Anzumerken ist, dass nur in dem *Tracking Modus* Änderungen bzw. das Hinzufügen neuer ROIs möglich ist. Eine ROI wird hier durch ein farbiges Rechteck visualisiert. Die Farbe wird beim Erstellen der ROI zufällig gewählt. Das Verändern der Geometrie einer gewählten ROI ist auf zwei Arten möglich. So kann der Anwender durch Ziehen an den Ecken oder Kanten diese skalieren oder bei gedrückter Shift-Taste und Ziehen der ROI diese verschieben. Beide Operationen sind durch den Bildrand begrenzt, sodass ein Verschieben oder Skalieren über diesen hinaus die ROI beschneidet. Ein letztes Element befindet sich über dem Videofenster und zählt praktisch schon zu dem Objektbereich, ist aber für eine bessere Handhabung neben diesem platziert wurden. Bei dem Element handelt es sich um eine Dropdown-Liste der Framelabels. Hier kann der Nutzer ein Label für das gesamte Frame auswählen bzw. neu eingeben. Somit ist es möglich eine Situation zu beschreiben, die nicht über einzelne ROIs erkenntlich wäre, wie z.B. das Wetter(Regen, Nebel) oder Zusammenhänge(Einweisen eines LKW).

Objektbereich Dieser Bereich ist von dem Videobereich getrennt und lässt sich durch einen Splitter in der Breite skalieren oder ganz ausblenden. Hier befinden sich zwei Listboxen mit integrierten Checkboxen für die *Mainlabels*(Hauptkategorien, z.B. Fahrzeug, Fußgänger) und *Sublabels*(Eigenschaften, z.B. fahrend, verdeckt).

Unter diesen befindet sich eine mehrspaltige Listbox für die Objektauswahl. Diese beinhaltet pro Objekt(ROI) ein Icon in der Farbe des Objekts sowie als Text die Main- und Sublabels. Unter dieser Listbox befinden sich Buttons zum Hinzufügen und Entfernen von Objekten und Labels sowie ein zugehöriges Texteingabefeld. Labels können, um Inkonsistenzen im Speicher zu verhindern, bewusst nicht global gelöscht werden. Die Objektauswahl erfolgt über die Objekt-Listbox. Im *Tracking Modus* wird das ausgewählte Objekt im Videofenster hervorgehoben und steht für Interaktionen oder Verändern der Labels über Auswahl aus den Label-Listboxen bereit. Die aktivierten Elemente der Label-Listboxen passen sich dabei interaktiv dem gewählten Objekt an.

Tastaturkürzel Wie Anwendungstests in Abschnitt 6.3 und Literatur[PD10] beweisen, profitieren Nutzer signifikant von einer beidhändigen Eingabe über Maus und Tastatur. So wurden folgende Kürzel definiert:

←, A: Springe ein Frame rückwärts (lade vorheriges Frame)

→, D: (*Tracking Modus*: tracke, *Wiedergabe Modus:* lade) das nächste Frame

Shift: Ausgewählte ROI lässt sich mit der Maus verschieben

+: Neue ROI lässt sich mit der Maus in dem Videofenster hinzufügen

4.2 Datenstruktur

Dieser Abschnitt beschreibt die Gestaltung der internen Datenstruktur hinsichtlich ihrer Klassen und Zugriffe. Grundlegend wird zwischen dem Frontend, mit dem Hauptfenster(*MainWindow*), und dem Backend(*Player*), mit der Speicherverwaltung und den Trackern, unterschieden.

Frontend Das Frontend beinhaltet das GUI und stellt somit die Schnittstelle zwischen dem Anwender und der internen Software dar. Hier werden sämtliche Daten(inkl. des Backends) initialisiert und Nutzereingaben verarbeitet und angezeigt. So beinhaltet das Hauptfenster alle in Abschnitt 4.1 erwähnten Steuer- und Anzeigeelemente des GUIs und verwaltet Callbacks dieser Elemente an das Backend. Als Schnittstelle zwischen dem Frontend und dem Backend dient das in dem Hauptfenster eingebettete Grafikfenster. Dieses verarbeitet, unabhängig von dem Hauptfenster, Interaktionen mit dem Video und zeigt die im Backend verarbeiteten Frames des *Players* an.

Backend Das Backend kommuniziert über Threads mit dem Frontend. So stellt das Frontend regelmäßig Anfragen zum Aktualisieren der Frames an das Backend. Eine Aktualisierung beinhaltet während der Video-Wiedergabe das Laden des nächsten Frames und, abhängig von dem aktivierten Modus, das Tracken der Objekte. In einem für den Nutzer nicht standardmäßig verfügbaren Evaluierungsmodus werden zusätzlich Evaluierungen zwischen einer vorher geladenen Ground Truth und dem

aktuellen Track erstellt. Diese werden in einem zusätzlichen internen Speicher verwaltet. Sowohl Frontend als auch Backend haben Zugriff auf die jeweils andere Klasse, um auf Strukturelemente oder den Speicher zugreifen zu können. Das Backend verwaltet die Videodatei und kann in dieser navigieren. Weiterhin werden hier die Trackerdaten (Typ, zugehörige ROI) und der interne Speicher verwaltet. Der interne Speicher setzt sich zusammen aus einzelnen Arrays von verfügbaren Mainlabels, Sublabels, Framelabels und einem Framespeicher. Der Framespeicher besteht pro Frame aus dem Framelabel und Informationen über alle in dem Frame verfügbaren ROIs. In jedem Frame werden, um den Speicherbedarf zu verringern, nur die IDs(Position im jeweiligen Label-Array) der Labels gespeichert, anstelle eines redundanten Strings. Eine ROI setzt sich zusammen aus Geometrie, einzigartiger ID, Farbe, Main- und Sublabel IDs. Ein Überblick der Datenstruktur wird in Abbildung 4.2 dargestellt.

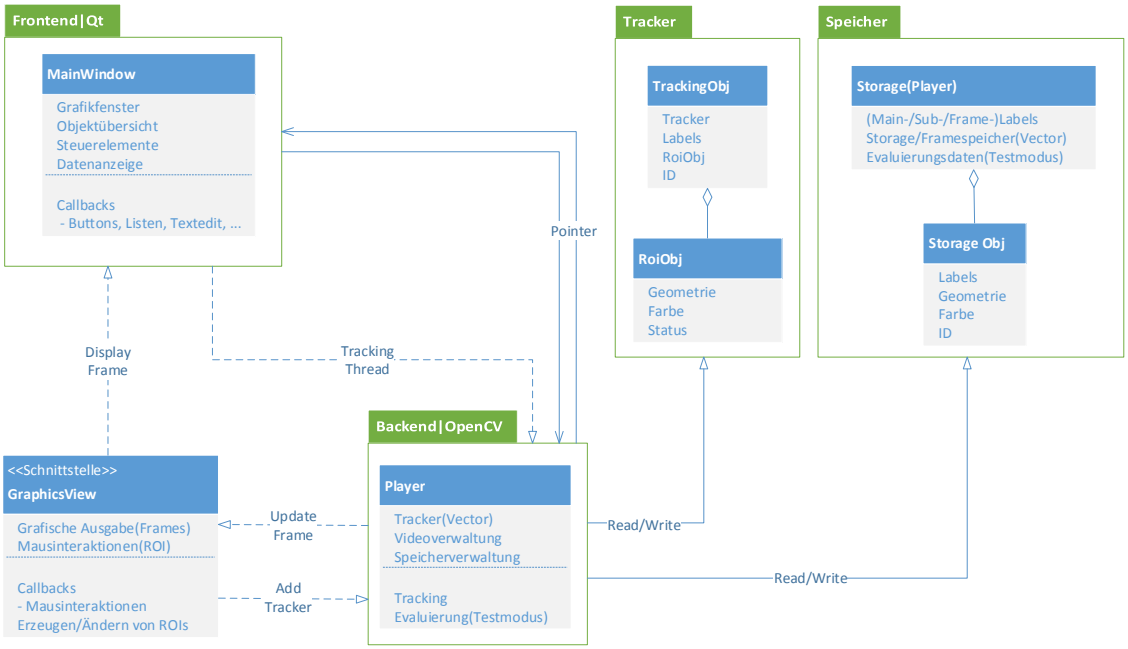

Abbildung 4.2: Übersicht der Datenstruktur. Wesentliche Elemente sind das Frontend(Hauptfenster) und das Backend(*Player*) mit Trackern und Speicher. Das visuelle Feedback aus dem *Player* erfolgt über den *GraphicsView* als Schnittstelle.

4.3 Datenexport

Der Datenexport dienst als externer Datenspeicher, um seinen Fortschritt zu speichern oder Ground Truth Daten zu erzeugen und für die Kommunikation mit externen Anwendungen. Die Exportdatei muss alle nötigen Informationen aus dem Setup der Szene bzw. dessen Annotationen enthalten. Für einen möglichst einfachen Aufbau und eine einfache Verarbeitung wird das Extensible Markup Language (XML)-Format genutzt. Aufgrund der Tatsache, dass wir, wie eingangs erwähnt, tendenziell große Datenmengen speichern müssen, wird die Exportdatei, wie auch zuvor der Speicher, möglichst schlank gehalten. So werden alle verfügbaren Labels einmalig gespeichert und später nur per ID(Position im jeweiligen Label-Array) in den ROIs referenziert. Es werden zudem nicht alle Frames gespeichert, sondern nur

die bereits annotierten. Pro annotierten Frame werden die Framenummer, das Framelabel(ID) und die zugehörigen ROIs gespeichert. Pro ROI werden die gleichen Informationen wie auch in dem internen Speicher in Abschnitt 4.2 gespeichert.
Zusätzlich können die Evaluierungsergebnisse im entsprechenden Modus exportiert werden, um sie anschließend mit einer Tabellenkalkulations-Software auszuwerten. Sie enthalten neben Setup-Informationen die Ergebnisse der angewandten Metrik in jedem Frame. Weiterhin ist es möglich, verschiedene Tracker-Konfigurationen zu verwenden. Diese können nur importiert werden, für ein besseres Verständnis wird der Aufbau trotzdem hier erwähnt. Sie bestehen ebenfalls aus XML-Dateien und enthalten neben dem Trackertyp die Typ-spezifischen Parameter. Der Aufbau beider XML-Dateien wird in Abbildung 4.3 gezeigt.

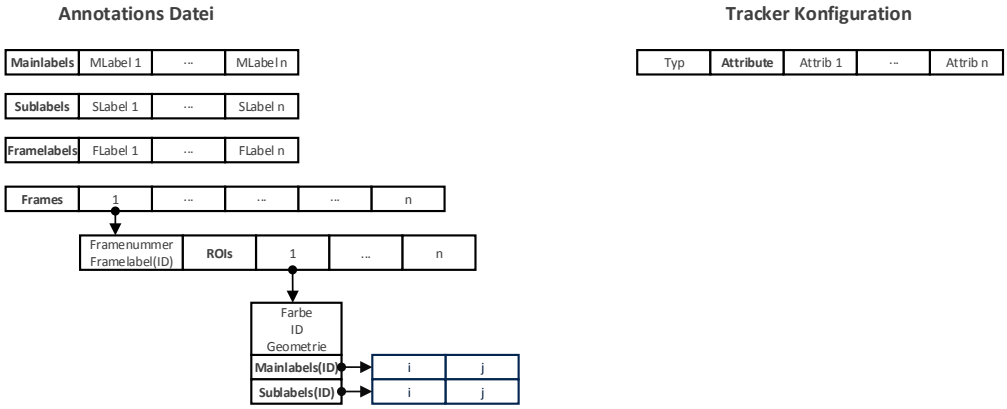

Abbildung 4.3: Strukturelle Übersicht der Export Dateien. Dick-gedruckte Bereiche stellen Arrays bzw. mehrere Einträge dieses Typs dar.

5. Implementierung

In diesem Kapitel wird auf Implementierungsdetails eingegangen. Es werden einführend die technischen Details erwähnt. So wurde die Software in Plattform-unabhängigen C++ geschrieben. Für eine einfache Speicherverwaltung ohne Memory Leaks wurden *Smartpointer* des C++11 Standards verwendet. Als Build-Tool wurden sowohl *CMake* 2.9 als auch *QMake* in Zusammenhang mit der Integrated Development Environment (IDE) *QtCreator* 4.1 und dem Compiler *GCC* 4.9 genutzt. Als externe Bibliotheken kamen *OpenCV* 3.2[Its16] und *Qt* 4.8.6[Qt14] zum Einsatz.

5.1 Frontend

Das Frontend stellt die Graphische Schnittstelle zum Nutzer bereit. Hierfür wurde die weit verbreitete Bibliothek *Qt* genutzt. *Qt* umfasst neben graphischen Elementen für das Interface auch viele nützliche Funktionen, wie XML-Verarbeitung, Zeitmessung und Thread-Management. Das Frontend besteht aus der in Abbildung 4.2 dargestellten Klasse *MainWindow*, welche alle weiteren Objekte initialisiert. Das GUI wurde mit *QtCreator* in Qt Meta-object Language (QML) erstellt.

Elemente und Callbacks

Die Steuerelemente setzen sich zusammen aus Eingabefeldern, Schiebereglern, Listboxen, (Dropdown-)Menüs und Checkboxen. Sie alle stellen in *Qt QWidgets* dar. Für diese lassen sich spezifische Callbacks definieren. Callbacks werden in bestimmten Situationen ausgelöst, wie z.B. das Aktivieren einer Checkbox oder das Wählen eines Elements aus einer Listbox. Diese werden von *Qt* abgefangen und ausgewertet. So werden z.B. die mit einem ausgewählten Objekt verknüpften Labels verändert und in den Speicher geschrieben. Es ist daher wichtig, das Speicherlayout so zu gestalten, dass über die Callbacks auf benötigte Daten zugegriffen werden kann und die Daten konsistent gehalten werden. Wichtige Callbacks sind unter anderem:

- **Zustandsänderungen:** Setzen von Zustandsvariablen, z.B. *Tracking Modus*

- **Selektion:** Auswählen von bestimmten Objekten oder Attributen, z.B. ROI
- **Texteingabe:** Beenden/Übermitteln einer Text-Eingabe, z.B. Framenummer
- **Aktionen:** Drücken von Buttons/Feldern, z.B. Wiedergabe starten

Weiterhin werden Tastatur-Events ausgewertet, über welche sich Shortcuts auswerten lassen. Für eine spezifische Auswertung von Tastatur-Events wurde ein *Eventhandler* eingerichtet. Dieser leitet Tastatureingaben nur dann weiter, wenn z.B. kein Eingabefenster ausgewählt wurde. Somit wird verhindert, dass Shortcuts ausgelöst werden, wenn die Intention des Nutzers eine Dateneingabe ist. Da es durch die Suchleiste bzw. Trackbar, Framenummer und Shortcuts möglich ist, zwischen Frames zu springen, werden die zu dem neuen Frame zugehörigen Daten aus dem Speicher geladen. Hierdurch werden auch die Tracker reinitialisiert, wodurch das antrainierte Modell verloren geht und neu trainiert werden muss.

Grafische Ausgabe

Die grafische Ausgabe besteht aus einem umgeschriebenen *QGraphicsView*. Dieser ist, gegenüber einfachen Elementen für eine Bildausgabe, dazu in der Lage, grafische Primitive in *Qt* zu verwalten. So verfügt jede ROI über eine allgemeine Geometrie mit Bildkoordinaten und eine *Qt*-spezifische Geometrie (*QGraphicsRectItem*). Diese Primitive lassen sich interaktiv in dem Grafikfenster verschieben und skalieren. Weiterhin wurden Maus-Events definiert, um eine Interaktion mit dem Grafikfenster zu ermöglichen. Diese beziehen sich auf die Zustände der linken Maustaste und die Bewegung der Maus. So kann eine neue ROI (bei aktiviertem Zustand) bei gedrückter Maus ausgewählt werden und wird beim Loslassen dem Speicher hinzugefügt. Weitere Maus-Events beziehen sich auf das Skalieren und Verschieben einer ROI. So lassen sich die Kanten bzw. Ecken an der Mausposition skalieren. Für eine verbesserte Nutzung wird ein Offset um die Mausposition herum verwendet, da ein pixel-genaues Auswählen die Nutzbarkeit stark einschränken und den Anwender frustrieren würde. Ebenso lässt sich die gewählte ROI beim gleichzeitigen Drücken der Shift- und linken Maustaste verschieben. Beide Interaktionen senden bei dem Drücken, Bewegen und Loslassen der Maus(-taste) Signale. Bei dem Loslassen der Maustaste wird die Änderung in die ROI des Trackers und in den Speicher geschrieben. Weiterhin wird mit jeder Aktualisierung des *Players* das aktuelle Frame aktualisiert, in dem die Bounding Boxen der ROIs in das Frame geschrieben werden und durch das Grafikfenster visualisiert werden.

5.2 Backend

Das Kernelement des Backends ist der *Player*. Er verwaltet den Speicher, die Tracker, die Videoverwaltung und führt den Trackingprozess aus. Für die Videoverwaltung und das Tracking wird die weitverbreitete Bibliothek *OpenCV* verwendet. Für die Videoverwaltung wird das Video in ein *VideoCapture* geladen, welches Meta-Informationen auslesen und das Video dekodieren kann. Somit ist es möglich, in dem Video zu navigieren und einzelne Frames auszulesen.

Speicher

Der Speicher gliedert sich im Wesentlichen in den Labelspeicher, Framespeicher und optional den Evaluierungsspeicher. Der Labelspeicher setzt sich zusammen aus je einem String-Vector für Main-, Sub- und Framelabels, sodass alle verwendeten Labels als kompletter String einmalig global gespeichert werden. Alle weiteren lokalen Labels eines Frames oder einer ROI speichern lediglich eine ID, welche die Position des vollwertigen Strings in dem jeweiligen String-Vektor darstellt. Hierdurch muss pro annotiertem Label lediglich ein Integer oder Short statt einem kompletten String gespeichert werden. Gleiches gilt, wie zuvor in Abschnitt 4.3 erwähnt, auch für die Export-Datei. Der Framespeicher ist ebenfalls ein Vector und beinhaltet, wie in Abschnitt 4.2 erläutert, für jedes Frame ein *Pair*. Dieses besteht aus einem Vector von *StorageObjects* und dem Framelabel(ID). Ein *StorageObject* beinhaltet alle benötigten Informationen für eine ROI: Labels(IDs), Geometrie, Farbe und die einzigartige ID der ROI. Um Reallokierungen des Framespeichers(Vector) zur Laufzeit zu verhindern, wird er nach dem Laden eines neuen Videos direkt mit der Gesamtanzahl aller Frames initialisiert. Zusätzlich wird die Konsistenz des Framespeichers hierdurch gewährleistet. Der *Evaluierungsmodus* ist erst verfügbar, nachdem eine Compiler-Flag im Code gesetzt wurde. Die Evaluierung findet während des Trackings statt. Die Evaluierungsergebnisse werden pro Frame in den Evaluierungsspeicher geschrieben. Dieser speichert für jede ROI die aus der verwendeten Evaluierungsmetrik entstandenen Ergebnisse.

Tracking

Für das Objekttracking wird die Tracking API von *OpenCV* aus dem *Contrib-Repository* genutzt. Für jede ROI in dem aktuellen Frame existiert ein *TrackerObject*, welches in einem Vector gespeichert wird. Ein *TrackerObject* setzt sich zusammen aus dem Tracker und den Informationen einer ROI(Labels, Geometrie, Farbe, ID) sowie dem Status der ROI. Der Status gibt Auskunft über den Zustand der Kanten. So passt sich dieser an, wenn eine spezifische Kante im Grafikfenster verändert wird, um anschließend den Tracker mit der neuen Bounding Box zu reinitialisieren. Eine Reinitialisierung ist notwendig, da ansonsten das (antrainierte) Modell des Trackers degenerieren würde. Während der Wiedergabe wird im *Tracking Modus* jeweils das aktuelle Frame in den Tracker geladen, wodurch der Tracker seine Position der ROI aktualisiert. In dem Framespeicher wird anschließend das Element des aktuellen Frames (Position = Framenummer) mit den neuen Informationen überschrieben. Der Typ des Trackers ist standardmäßig KCF, es lassen sich jedoch Konfigurationsdaten aller in *OpenCV* 3.2 implementierten Tracker über das *MainWindow* laden. Bei einer (Re-)Initialisierung wird ein Tracker mit der zuletzt geladenen Trackerkonfiguration erstellt. Somit ist es möglich, dass mehrere Trackermodelle parallel verwendet werden. In Abbildung 5.1 wird der Zusammenhang zwischen Speicher, Tracker und Frame verdeutlicht.

Multithreading

OpenCV nimmt keine Optimierung für Multitracker vor, sondern speichert diese lediglich in einem Vector und aktualisiert diese sequenziell. Da Tracking ein rechenintensiver Prozess ist, leidet die Performance stark unter einer sequentiellen

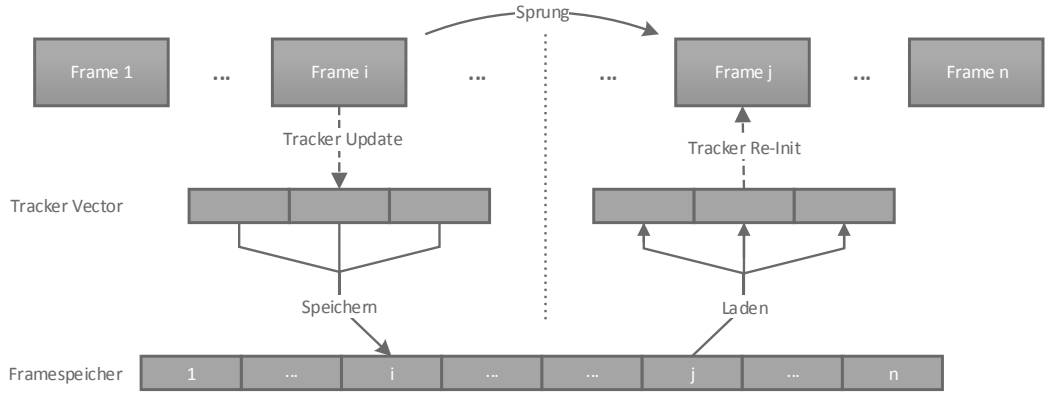

Abbildung 5.1: Zusammenhang zwischen Frame, Tracker und Speicher. **Links** wird von Frame $i-1$ zu i getracked, **rechts** wird zu Frame j gesprungen und der Tracker aus dem Speicher generiert.

Ausführung, wie in Abschnitt 6.2 gezeigt wird. Somit wurde auf die Multitracker-Implementierung von *OpenCV* verzichtet und eine eigene Variante implementiert. Diese sieht ebenfalls einen Vector aus Trackern vor. Sie werden jedoch nicht sequenziell, sondern parallelisiert aufgerufen. Für die Parallelisierung wurden die in C++11 standardmäßig verfügbaren Threads genutzt. Für jeden Tracker wird nun ein neuer Thread erstellt. Die Verteilung ist hierbei simpel und erfolgt in Intervallen: pro CPU-Kern wird für einen Tracker ein Thread erstellt. Nach den Berechnungen werden die Threads synchronisiert. Der Vorgang wird wiederholt bis alle Tracker aktualisiert wurden. Die Dauer t eines Updates von n Objekten mit c CPU-Kernen ergibt sich daher aus der Summe der einzelnen Update-Intervalle i:

$$t(n) = \sum_{i=1}^{\lceil \frac{n}{c} \rceil} \max t(i) \tag{5.1}$$

Dabei ist anzumerken, dass ein Update-Intervall i durch die Synchronisierung maximal so schnell ist wie dessen langsamste Instanz, welche von der Größe der Bounding Box und der Art des verwendeten Trackers abhängt.

6. Evaluierung

In der Evaluierung betrachteten wir drei verschiedene Faktoren. So werden die Tracking-Algorithmen hinsichtlich ihrer Geschwindigkeit und Genauigkeit geprüft, um anschließend eine Empfehlung für spezifische Tracker in spezifischen Szenen zu geben. Weiterhin wird der Einfluss von Parallelisierung auf die Geschwindigkeit des Systems untersucht. Zuletzt folgt eine Nutzerstudie hinsichtlich der benötigten Zeit für das Annotieren einzelner Szenen zwischen *VATIC* und dem hier vorgestellten Tool.

6.1 Tracking-Algorithmen

Die Tracker werden anhand einzelner Szenen getestet. Diese werden in bestimmte Szenarien unterteilt, da es (noch) nicht den perfekten Tracker für jede Situation gibt. Hierdurch werden die Stärken und Schwächen der einzelnen Tracker aufgezeigt und für jedes Szenario eine Empfehlung gegeben. Als Szenarios wurden Fahrzeugdaten in folgenden Situationen gewählt: nächtliche Kreuzung bei Regen, Verkehrsfluss bei Dämmerung, Größenänderung eines Fahrzeugs, Belichtungsunterschiede durch Gegenlicht, Verdeckung eines Fahrzeuges, Verdeckung von mehreren Verkehrsteilnehmern. Die Evaluierung orientiert sich an [WLY13]. So werden ebenfalls verschiedene Szenarien erstellt, jedoch werden, statt der Initialisierung in verschiedenen Frames und Positionen(Offsets), unterschiedlich große Bounding Boxen verwendet. Somit soll geprüft werden, wie ein Tracker auf mehr oder weniger Informationen reagiert. Um eine messtechnisch genaue Gesamtperformance zu evaluieren, sind verschiedene Initialisierungen wie in [WLY13] notwendig, da hier aber nur das Verhalten in kurzen, einzelnen Szenen das Ziel der Untersuchung ist, wird dieses vernachlässigt. Gewählt wurden zwei Metriken: 1. die Überlagerung der Flächen beider ROIs und 2. der euklidische Abstand der Zentren. So sind diese zwischen Ground Truth GT und Track T definiert als:

$$M_{area}(GT, T) = \frac{|\text{Area}(GT) \cap \text{Area}(T)|}{|\text{Area}(GT) \cup \text{Area}(T)|} \tag{6.1}$$

$$M_{dist}(GT, T) = \|\mathbf{gt}_{center} - \mathbf{t}_{center}\| \tag{6.2}$$

Wobei $\mathbf{gt}_{center}$ und $\mathbf{t}_{center}$ den jeweiligen Mittelpunkt der Bounding Box als Punktvektor repräsentieren. Beide sind notwendig, um zwei unabhängige Metriken zu haben, da nur der TLD- und Median-Flow-Tracker eine Skalierung der ROI beherrschen und somit einen Vorteil bei der alleinigen Betrachtung der Fläche hätten. Die nicht-skalierenden Tracker können in dem Distanz-Maß aufzeigen, dass sie das Objekt noch verfolgen können, obwohl die Bounding Box nicht mit dem Objekt skaliert, aber trotzdem noch zentriert ist. Letzteres ist interessant, wenn die Größe der Bounding Box von geringerer Bedeutung, die Position jedoch entscheidend ist. Weiterhin wurden, sofern sinnvoll, verschiedene Parametersetups der Tracker getestet. Die Änderungen der Parameter können den Tabellen der Evaluierung entnommen werden. Einzig bei TLD wurden, aufgrund der schlechten Gesamtperformance, keine zusätzlichen Skalierungen getestet, dies wurde in den Tabellen der Auswertung mit einem * gekennzeichnet. Die Tracker wurden nur einmalig initialisiert und verfolgen die Objekte mit dieser Initialisierung über die gesamte Sequenz. Getestet wurde auf einem Intel Core i7-3630QM(2.4GHz, 4 native Kerne + 4 virtuelle Kerne) und 8GB RAM.
Allgemein wurden die Daten der Ground Truth mit genau passenden Bounding Boxen erstellt: Objekte verändern distanzabhängig ihre Größe und erhalten rotationsbedingt neue Informationen. Ein nicht-optimales Ergebnis für einen Tracker stellt somit kein Ausschlusskriterium dar, da eine Re-Initialisierung der Bounding Box letzteren Umstand beheben würde und ersterer den Vorteil von skalierenden Trackingsystemen deutlich macht. In Abbildung 6.1 werden die einzelnen Szenen gezeigt, auf welche nachfolgend eingegangen wird.

Abbildung 6.1: Übersicht der Szenen. **(1)** kleiner werdendes Fahrzeug, **(2)** teilweise verdecktes Fahrzeug, **(3)** mehrere verdeckte Fahrzeuge, **(4)** Belichtungsunterschiede, **(5)** Dämmerung und langer Verkehrsfluss, **(6)** Nacht und Regen

Größenänderung

Die Szene in Abbildung 6.1(1) beschreibt ein Fahrzeug, welches kurz am linken Bildrand mit höherer Geschwindigkeit vorbeifährt und mit zunehmender Distanz kleiner wird. Die Auswertung in Tabelle 1.1 zeigt, dass der Median-Flow-Tracker in Situationen mit weichem Bildlauf und Skalierung der Objektgröße die mit Abstand besten Ergebnisse bringt. Ausschlaggebend für das Ergebnis ist die Initialisierung der

Bounding Box, diese sollte sich idealerweise innerhalb der Objektgrenzen befinden, um eine Degenerierung zu verhindern. Weiterhin ist zu sehen, dass besonders der KCF-Tracker empfindlich auf die Objektgröße reagiert und nur noch sehr langsam ist, obwohl er sonst einer der schnellsten und zuverlässigsten Tracker ist.

Verdeckung eines Objektes

Szene (2) in Abbildung 6.1 stellt ein kleiner werdendes Fahrzeug dar, welches mehrfach kurz von Verkehrsschildern und Masten verdeckt wird. Der KCF-Tracker kann das Objekt bei geringer Rechenzeit sehr exakt verfolgen, wie der geringe Distanz-Fehler in Tabelle 1.2 zeigt. Wichtig ist, dass bei Verdeckung genug Informationen über das Objekt vorhanden sind, da kleinere Bounding Boxen meist schlechtere Ergebnisse liefern.

Verdeckung mehrerer Objekte

In dieser kurzen Szene steht das Kamerafahrzeug still an einer Kreuzung, während andere Verkehrsteilnehmer(Radfahrer, Autos) die Kreuzung mit konstanter Geschwindigkeit queren und sich dabei gegenseitig Verdecken, wie in Abbildung 6.1(3) zu sehen ist. Die Ergebnisse sind in Tabelle 1.3 zu sehen. Szenen dieser Art stellen allgemein eine große Herausforderung für Tracker dar, da teils ähnliche Informationen überlagert werden und somit schwer zu unterscheiden sind. Die besten Ergebnisse liefern die stark lernenden Algorithmen Boosting und MIL, wobei klassisches online Boosting mit weniger Features und einer vergrößerten Bounding Box am stabilsten läuft. Diese Szene ist mit linearer Interpolation grundsätzlich besser zu verfolgen, da diese unabhängig von den Bilddaten arbeitet und somit keinen Schwierigkeiten bei Verdeckung unterliegt.

Belichtungsunterschiede

Stark unterschiedliche Belichtungen in einer Szene stellen Herausforderungen hinsichtlich der verwendeten Features dar, da die Farbinformationen sich stark verändern und Kanteninformationen bei mangelndem Kontrast verloren gehen. Die Szene in Abbildung 6.1(4) stellt solche Probleme dar. Ein vorausfahrendes Fahrzeug ist anfangs gut zu identifizieren, durch eine Fahrt in einem Kreisverkehr kommen rotationsbedingt neue Informationen hinzu. Anschließend fährt es durch einen Schattenbereich, das Kamerafahrzeug fährt jedoch mit Gegenlicht, wodurch das vordere Fahrzeug sich kaum noch von dem Hintergrund unterscheidet. Auch hier sind die stark lernenden Algorithmen MIL und Boosting im Vorteil, wobei nur MIL das Objekt durch die wechselnde Belichtung am Ende relativ genau verfolgen kann, wie der geringere Distanzfehler in Tabelle 1.4 belegt. Größere Bounding Boxen lassen das Objekt dabei besser verfolgen, allerdings wird die Genauigkeit mit einem hohen Rechenaufwand erkauft, da besonders der MIL-Tracker rechen-technisch sehr aufwändig ist.

Dämmerung und Langzeitverfolgung

Diese längere Szene wurde während der Abenddämmerung aufgenommen, wie in Abbildung 6.1(5) zu sehen ist. Sie enthält zwei vorweg fahrende Fahrzeuge auf kurviger Strecke, wobei das rechte Fahrzeug seine Distanz und somit auch Größe mit

der Zeit verändert. Durch die gestiegene Empfindlichkeit der Kamera auf weniger Licht wirken sich Bremslichter verstärkt auf das Bild aus, wodurch ein bremsendes Fahrzeug seine Repräsentation verändert. Tabelle 1.5 zeigt, dass der Median-Flow-Tracker mit kleineren Bounding Boxen die Objekte gut verfolgen kann und die Größe anpasst, allerdings nicht über die gesamte Sequenz, wie der Distanzfehler aufzeigt. Der Boosting-Tracker mit angepassten Parametern weißt hingegen einen sehr geringen Distanzfehler bei gleichzeitig guter Rechenzeit auf.

Nacht und Regen

In Abbildung 6.1(6) wird eine kurze nächtliche Szene mit Regen gezeigt. Das Kamerafahrzeug steht dabei still an einer Kreuzung, ein Auto aus dem Querverkehr soll verfolgt werden. Dieses Szenario stellt eine besonders große Herausforderung dar, da die Kamera durch die geringe Lichtmenge nur wenig Informationen erhält und gleichzeitig die Lichtempfindlichkeit stark erhöht ist, wodurch Lichtquellen überbelichtet werden. Durch den Regen werden die Lichter zudem reflektiert, die Gischt verfälscht die Objekte und die Informationen des Objektes werden unterrepräsentiert. Ein zufriedenstellendes Ergebnis stellt somit kein Tracker dar, wie Tabelle 1.6 zu entnehmen ist. Lediglich die stark lernenden Algorithmen Boosting(mit angepassten Parametern) und MIL nähern sich der Ground Truth an, jedoch nur, wenn die Bounding Box verkleinert wird. Allgemein lässt sich eine solche Situation nur schwer tracken, lineare Interpolation würde hier, durch die Unabhängigkeit von den Bilddaten, bessere Ergebnisse liefern.

Zusammenfassend lässt sich sagen, dass stets abgewogen werden muss, ob ein Tracker mit der Objektgröße skalieren oder die Position gut verfolgen soll. Für viele Situationen bietet der KCF-Tracker ein gutes Verhältnis aus Rechengeschwindigkeit, Genauigkeit und Robustheit. Er ist somit in Situationen, wo das Objekt nicht zu groß ist, seine Größe behält und sich nicht zu stark mit anderen Objekten überschneidet, eine gute Wahl. Bei stärkeren Veränderungen des Objektes, z.B. durch Belichtung oder Verdeckung mit ähnlichen Objekten, stellen stark lernende Algorithmen, wie Boosting und MIL, oft die beste Wahl dar. Boosting profitiert besonders von vielen Iterationen während der Initialisierung und einer verringerten Anzahl an Features, da Objekte (speziell Verkehrsobjekte) oft eine einfache Form und homogene Oberfläche haben und diese somit gut angelernt wird. Für einfache und weiche Bewegungen mit sich in der Größe verändernden Objekten eignet sich der Median-Flow-Tracker besonders gut, da er die Größe der Bounding Box skaliert und somit keine weiteren Eingriffe des Nutzers notwendig sind. Der Parametereinfluss ist bei Median-Flow marginal, wichtiger ist es, die Bounding Box innerhalb der Objektgrenzen zu positionieren, da eine Bounding Box außerhalb des Objektes den Hintergrund verfolgt und dazu neigt zu degenerieren und eine zu kleine Initialisierung nicht korrekt mit den Objektgrenzen skaliert. Die Implementierung des TLD-Trackers schnitt bei jedem Test sehr schlecht ab. Die Rechenzeit ist sehr hoch und die Genauigkeit sehr gering. Auffällig ist, dass der Detektor des TLD-Trackers stark innerhalb des Bildes springt, also Objekte falsch erkennt, wodurch auffällig hohe Distanzfehler zustande kommen.

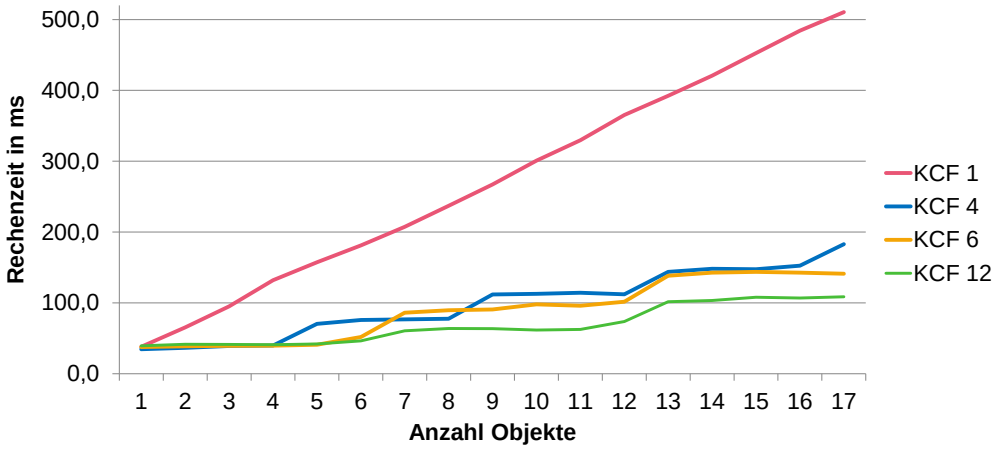

Abbildung 6.2: Skalierung der Rechenzeit in Abhängigkeit von der Anzahl der verfolgten Objekte. Die Zahl hinter KCF steht für die Anzahl der verwendeten CPU-Kerne.

6.2 Parallelisierung

Der Aufwand des Trackings einzelner Objekte ist bereits, je nach Tracker und Szene, sehr hoch. Werden mehrere Objekte verfolgt, ist eine Parallelisierung notwendig, um weiterhin echtzeitfähig zu sein. Der Einfluss von mehreren CPU-Kernen wurde mit dem KCF-Tracker mit einem Intel Core i7-5930K(3.5GHz, 6 native Kerne + 6 virtuelle Kerne) und 32GB RAM untersucht. Abbildung 6.2 zeigt den Einfluss der CPU-Kerne auf die durchschnittliche Rechenzeit in Abhängigkeit von der Anzahl der verfolgten Objekte: wie deutlich zu sehen ist, wird die Performance signifikant durch die Anzahl der CPU-Kerne beeinflusst. Mit 12 CPU-Kernen arbeitet das System um den Faktor 5 schneller als mit einem Kern. Ein Anstieg der Rechenzeit erfolgt etappenweise, immer genau dann, wenn die Anzahl der Objekte die Anzahl der CPU-Kerne um ein Vielfaches übersteigt. Interessant sind zudem zwei Beobachtungen: die erste bezieht sich auf die Überschneidung der Graphen von 4 und 6 Kernen, wobei 6 Kerne zeitweise einen höheren Rechenaufwand benötigen. Eine Vermutung liegt in der Vollauslastung aller Kerne des Prozessors, da bei 4 Kernen mehr verfügbar sind als benötigt werden. Die zweite interessante Beobachtung bezieht sich auf die Skalierung mit virtuellen Kernen. Die CPU verfügt über 6 native und 6 virtuelle Kerne. Sind die nativen Kerne ausgelastet, steigt der benötigte Rechenaufwand leicht an, obwohl noch virtuelle Kerne verfügbar sind. Je weniger Kerne ein System zur Verfügung hat, desto schneller kommt es zu einem Anstieg der Rechenzeit. Der Unterschied zwischen 6 und 4 Kernen scheint nicht groß zu sein, der Unterschied in der Praxis liegt auch hier in der Anzahl der Objekte, wo zwei Objekte mehr einen signifikanten Unterschied ausmachen können.

6.3 Nutzerstudie

In der Nutzerstudie wurde das hier vorgestellte System mit *VATIC*[VPR13] verglichen. Gemessen wurde die benötigte Zeit, um verschiedene Videosequenzen zu annotieren. Als Qualitätskontrolle wurde jede annotierte Sequenz begutachtet und

es musste, wenn nötig, nachgebessert werden. Das hat den Vorteil, dass jede Sequenz qualitativ gut annotiert werden musste. Eine Messung, wie genau eine Markierung ist, wurde nicht vorgenommen. Ein Objekt galt als richtig markiert, wenn die Bounding Box von dem Objekt nur minimal abwich. Wir kommen somit zu Ergebnissen, welche die tatsächlich benötigte Zeit widerspiegeln und nicht, wie gut die durchschnittliche Qualität ist, da diese von der Einzelperson abhängig ist und eine hohe Qualität in dieser Studie vorausgesetzt wurde. Die Studie wurde an acht Probanden aus unterschiedlichen Fachbereichen durchgeführt. Diese mussten jeweils drei verschiedene Videosequenzen annotieren, welche in Abbildung 6.3 dargestellt werden.

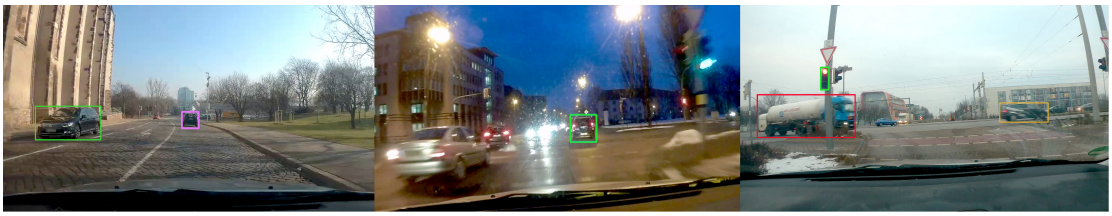

Abbildung 6.3: Übersicht der für die Nutzerstudie verwendeten Szenen. **(1)** Kopfsteinpflaster **(2)** Asphalt bei Dämmerung **(3)** Kreuzung mit Gleisen

In Szene (1) wurde das vorausfahrende Fahrzeug und der Gegenverkehr bis zu einer entfernten Kreuzung annotiert. Die Position und der Abstand zu den Fahrzeugen verändern sich während der Szene mehrfach. Durch Kopfsteinpflaster kommt es zu starken Schwankungen der Kamera. Szene (2) beinhaltet eine kurze Strecke auf Asphalt über eine Kreuzung mit schlechteren Lichtverhältnissen (Dämmerung). Da Asphalt auch nicht frei von Unebenheiten ist, kommt es auch hier zu Schwankungen der Kamera. Verfolgt wird nur das vorausfahrende Fahrzeug. In Szene (3) steht das Kamerafahrzeug anfangs still an einer Kreuzung und biegt anschließend an dieser nach links ab. Inmitten der Kreuzung befinden sich Gleise, was zu kurzzeitigen Schwankungen der Kamera führt. Annotiert werden die Leuchtanlagen, der Querverkehr und das erste Fahrzeug aus dem Gegenverkehr.
Den Probanden wurde jeweils die zu verwendende Software erklärt und für eine kurze Einarbeitung zur Verfügung gestellt. Dadurch sollten die Probanden an das jeweilige System gewöhnt werden, um einen anfänglichen Lerneffekt während des Tests zu vermeiden und eine Software zu bevorteilen. Weiterhin wurde im Vorfeld jeweils die komplette Videosequenz gezeigt und erklärt, welche Objekte markiert werden sollen. Um einen einschlägigen Lerneffekt zu vermeiden, hat die Hälfte der Probanden mit *VATIC* angefangen, die andere Hälfte mit der hier vorgestellten Anwendung. Erst nachdem alle Sequenzen mit einer Software fertig gestellt wurden, wurde nach einer Pause die andere Software verwendet und die Sequenzen in der gleichen Reihenfolge bearbeitet. Dadurch sollte bewirkt werden, dass die Probanden einen durchgehenden Lerneffekt in einer Software ausnutzen können, keinen Ermüdungseffekt durch zu lange durchgehende Arbeit erleiden und dass kein Gedächtnis bezüglich der einzelnen Sequenzen aufgebaut wird. Das Ergebnis ist in Abbildung 6.4 zu sehen.
Mit der hier vorgestellten tracking-basierten Methode benötigten die Probanden in den ersten beiden Szenen im Durchschnitt nur die Hälfte der Zeit im Vergleich zu

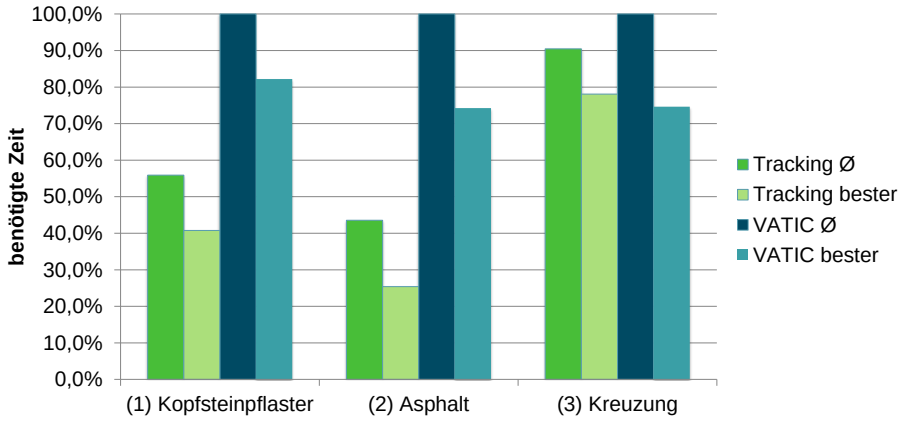

Abbildung 6.4: Ergebnis der Nutzerstudie. Dargestellt ist die relative benötigte Zeit für die jeweilige Szene. Gezeigt wird jeweils die durchschnittliche und die schnellste Zeit.

VATIC und waren somit signifikant schneller. Durch die Unebenheiten der Fahrbahn kommt es dort oft zu Situationen, wo die starre Bounding Box der linearen Interpolation in *VATIC* stark von dem Objekt abweicht, wie bereits in Abbildung 3.2 veranschaulicht wurde. Somit muss der Nutzer vermehrt eingreifen und die Position korrigieren. Interessant ist, dass der schnellste Proband der kurzen zweiten Szene nur ein Viertel der durchschnittlichen Zeit und ein Drittel der schnellsten Zeit von *VATIC* benötigte. In Szene drei wird deutlich, dass *VATIC* durch die lineare Interpolation in linearen Szenen deutlich schneller ist. Die Tracker müssen oft korrigiert werden, sobald neue Informationen hinzukommen oder entfernt werden, was am Bildrand durch neu hinzukommende oder verschwindende Objekte oft der Fall ist. Hier kann in *VATIC* der Anwender bei linearen Bewegungen die Bounding Box in großen Schritten vergrößern und das Objekt anschließend durch wenige Stützpunkte verfolgen. Erst mit Beginn der Bewegung über die Kreuzung und den damit verbundenen nicht-linearen Bewegungen (Beschleunigung während des Anfahrens) kann das trackingbasierte System wieder aufschließen. Durch die Gleise und die damit verbundenen Kamerabewegungen wird in *VATIC* mehr Zeit für Korrekturen benötigt, wodurch die Probanden in beiden Systemen insgesamt im Durchschnitt fast gleich viel Zeit benötigten. Der schnellste Proband benötigte mit *VATIC* sogar weniger Zeit. Würde eine Sequenz keine Fahrt über Unebenheiten oder nur linearen Verkehr beinhalten, wäre eine Annotation mit *VATIC* deutlich schneller.
Weiterhin berichteten die Probanden, dass sie es, auch bei gleicher benötigter Zeit, als angenehmer empfanden mit dem tracking-basierten System zu arbeiten. Es wurde als frustrierend wahrgenommen, wenn bereits fertige Abschnitte nachträglich korrigiert werden mussten, was bei der linearen Interpolation vor allem bei Unebenheiten auftrat. Dadurch könnte ein eintretender Ermüdungseffekt möglicherweise verringert werden. Als besonders effektiv gestaltet sich die Kombination aus Springen einzelner Frames und dem Verändern der ROI, wenn sich Objekte im Randbereich in oder aus der Szene bewegen. Viele Probanden haben das Springen einzelner Frames (bzw. das stetige Drücken der Taste) der Wiedergabe über den Play-Button vorgezogen, da sie so eine präzisere Kontrolle haben und schneller reagieren können.

7. Zusammenfassung und Ausblick

In dieser Arbeit wurde gezeigt, dass optisches Tracking den Nutzer bei der Annotation von Videosequenzen signifikant unterstützen kann und wie ein Annotationssystem mit *Qt* und *OpenCV* aufgebaut sein kann. Auch wenn die Evaluierung der Tracker in Abschnitt 6.1 oder [WLY13] gezeigt hat, dass an diesen noch weiterer Forschungsbedarf besteht, so zeigt doch die Nutzer-Evaluierung in Abschnitt 6.3, dass Trackingsysteme den Anwender gut unterstützen können und die Nutzer teilweise deutlich schneller sind als mit Interpolations-basierten Methoden, wie [VPR13]. So wurde gezeigt, dass bei guten Sichtverhältnissen, ohne starke Verdeckung und vor allem bei Fahrbahnunebenheiten, Systeme mit Tracking die Annotation von Videosequenzen, im Speziellen mit Verkehrsdaten, signifikant beschleunigen können. Die Anzahl der Nutzereingriffe wird stark verringert und Objekte können unter guten Bedingungen fast vollständig ohne Nutzereingriff verfolgt werden. In einigen Situationen, vor allem in Randgebieten, bei schlechten Sichtverhältnissen oder Verdeckung von Objekten, bietet (bi-)lineare Interpolation durch die Unabhängigkeit von den Bilddaten jedoch noch Vorteile. Mit der weiteren Entwicklung von Tracking-Algorithmen werden die Nachteile gegenüber der Interpolation weiter verringert werden. Zu dem jetzigen Zeitpunkt wäre eine Kombination beider Systeme jedoch sinnvoll, um situationsabhängig die beste Methode nutzen zu können. Weiterhin wurde in Abschnitt 6.2 gezeigt, dass der Einsatz von Parallelisierung durch Multithreading zu einer erheblichen Beschleunigung führt, welche für den Einsatz von Tracking-Algorithmen unter Echtzeit eine Notwendigkeit darstellt. Die Verwendung der Tracker in *OpenCV* 3.2 sollte für Verkehrsdaten situationsbedingt gewählt werden. So bringt der Median-Flow-Tracker gute Ergebnisse mit einer Skalierung der ROI bei guten Sichtverhältnissen und weichen Bewegungen. Die Bounding Box sollte leicht innerhalb der Objektgrenzen sein, um ein Degenerieren zu verhindern. Der KCF-Tracker ist sehr stabil, beherrscht leichte Verdeckung und ist schnell, sofern die ROI nicht zu groß gewählt wird. Für Probleme mit schlechter Sicht oder mittelstarker Verdeckung können der Boosting- oder MIL-Tracker verwendet werden. Da diese Systeme starke Lernkomponenten besitzen, sind sie nach genügend Iterationen robust gegenüber Belichtungsunterschieden und Verdeckung. Der Boosting-Tracker profitiert sowohl in der Rechenzeit als auch in der Genauigkeit von einem verrin-

gerten Featurepool und weniger Selektoren. Allerdings sind diese Trackingsysteme gegenüber Interpolation bei schlechten Bilddaten noch unterlegen.

Ausblick

Das hier vorgestellte Annotationssystem stellt nur eine einfache Implementierung dar, so sind zahlreiche Optimierungen und Erweiterungen denkbar. Aus technischer Sicht wäre die wohl wichtigste Erweiterung der Einsatz von GPU-basierten Tracking-Algorithmen, welcher die Rechenzeit der Tracker enorm reduzieren und gleichzeitig das Tracking von mehreren Objekten über das Vermögen von CPUs hinaus und die Verwendung von sehr hochauflösenden 4K-Videos ermöglichen würde. Weiterhin stieg die Rechenleistung von GPUs im Vergleich zu CPUs in den letzten Jahren sehr stark an, wodurch die Systeme mit zukünftiger Hardware besser skalieren würden. Eine einfache Erweiterung der Tracker ist verfügbar, sobald ein Patch für den GOTURN-Tracker in *OpenCV* 3.2 verfügbar ist, wobei besonders hier eine GPU-Implementierung interessant wie in [HTS16] ist. Für bessere Ergebnisse der Tracker könnte zusätzlich eine Vorverarbeitung der Bilddaten durch Histogramm-Linearisierung oder eine Korrektur der Bewegungsunschärfe vorgenommen werden. Eine Erweiterung durch Segmentierung der ROI, wie in [RE11], würde eine pixelgenaue Beschreibung der Objekte ermöglichen. Weiterhin wäre eine weitere Automatisierung durch den Einsatz von spezialisierten Detektoren denkbar. So könnte ein Detektor das Objekt automatisch erkennen und selber oder über einen Tracker verfolgen, der Anwender müsste nun nur noch die Labels anpassen.
Auf Seiten der Benutzerfreundlichkeit wäre eine Mehrfachauswahl von Objekten eine wichtige Erweiterung, so könnten die Labels von mehreren Objekten zeitgleich angepasst werden. Weiterhin wäre ein selektives Update der Tracker sinnvoll, da somit einzelne oder über Mehrfachauswahl gewählte Objekte unabhängig getracked würden und der Anwender sich stärker auf einzelne Objekte fokussieren könnte. Dies hätte, wie die Ergebnisse in [VPR13] zeigen, eine positive Auswirkung auf die Arbeitsgeschwindigkeit der Anwender. Zusätzlich sollte eine Auswahlliste an verfügbaren Trackern direkt im Interface integriert werden, um in einer spezifischen Situation schnell den passenden Tracker wählen zu können. Jedes Objekt sollte dabei eine eigene Tracker-Konfiguration besitzen und nicht wie bisher eine globale, die bei Re-Initialisierung überschrieben wird. Somit wäre es möglich, dass für jedes Objekt der beste Tracker gewählt wird. Ein Zusatz dazu wäre die Integration von linearer Interpolation neben dem Tracking. Über einen Schalter könnte für jedes Objekt zwischen Tracking und Interpolation gewechselt werden, wodurch in für Tracker schwierigen Situationen auf Interpolation gewechselt werden könnte. Eine weitere denkbare Erweiterung wäre die Möglichkeit, den Speicher, mit wenigen Frames Versatz, transparent im Hintergrund laufen zu lassen. Dadurch könnte der Nutzer, wenn er den Speicher zwischendurch überschreibt, sehen, wie lange er ein Objekt verfolgen muss, bis er wieder die passenden Ergebnisse erreicht hat.

Quelltextverzeichnis

```
Input: Datensatz aus Samples ⟨x, y⟩

1.  Initialisiere Signifikanz-Gewicht λ = 1
2.  for(k = 1...K)          // alle Selektoren
    {
3.      for(m = 1...M)      // alle schwachen Klassifikatoren
        {
4.          bewerte aktuelles Sample ⟨x_i, y_i⟩ mit h^weak_{n,m}      (Gleichung 3.27)
5.          berechne Fehler e_{n,m} des Klassifikators h^weak_{n,m}   (Gleichung 3.28)
        }
6.      waehle besten Klassifikator h^weak_{n,m} als Selektor h^sel_n (Gleichung 3.25)
7.      weise Selektor h^sel_n das Gewicht α_n zu                     (Gleichung 3.29)
8.      passe Signifikanz-Gewicht λ an                                (Gleichung 3.30)
9.      ersetze schlechtesten h^weak durch zufaelligen neuen
    }

Output: Klassifikator h^Strong als Summe der Selektoren               (Gleichung 3.26)
```

Quelltext 1.1: Online-Boost Algorithmus

```
Input: Datensatz aus Samples {X_i, y_i}, wobei X_i ={x_i1, ..., x_ij, y_i}

1.  Aktualisiere alle M schwachen Klassifikatoren mit
    dem aktuellen Sample {x_ij, y_i}                                  (Gleichung 3.31)

2.  Initialisiere starken Klassifikator h^Strong_ij = 0
3.  for(k = 1...K)          // alle Selektoren
    {
4.      for(m = 1...M)      // alle schwachen Klassifikatoren
        {
5.          berechne Instanz-Wahrscheinlichkeit p^m_ij                (Gleichung 3.33)
6.          berechne Bag-Wahrscheinlichkeit p^m_i                     (Gleichung 3.36)
7.          berechne Log-Likelihood ℓ^m der Bags                      (Gleichung 3.37)
        }
8.      waehle besten schwachen Klassifikator als Selektor            (Gleichung 3.38)
9.      addiere Selektor zu starkem Klassifikator
    }

Output: Klassifikator h^Strong als Summe der Selektoren               (Gleichung 3.39)
```

Quelltext 1.2: Online-MIL-Boost Algorithmus

Tabellenverzeichnis

Tracker	Parametersetup	Skalierung	Ø Rechenzeit in ms	Ø gemeinsame Fläche in %	Ø Abstand in Pixeln
Boosting	Standard		100,30	0,21	132,83
		kleiner	88,41	0,19	123,05
		größer	129,00	0,23	129,69
	weniger Features/Klassifikatoren, geringerer Suchradius, mehr Iterationen bei der Initialisierung		**29,29**	0,22	109,21
		kleiner	**22,85**	0,18	139,20
		größer	**35,64**	0,22	123,77
KCF	Standard		381,24	0,22	76,18
		kleiner	295,16	0,20	61,47
		größer	455,72	0,24	87,36
Median-Flow	Standard		42,25	**0,80**	**9,26**
		kleiner	36,76	0,22	125,58
		größer	39,60	0,70	18,19
	erhöhte Pixeldichte		47,40	**0,72**	**14,18**
		kleiner	48,36	**0,71**	**10,55**
		größer	46,81	0,67	22,67
MIL	Standard		114,23	0,22	71,90
		kleiner	122,16	0,20	67,39
		größer	119,95	0,24	83,76
	mehr Features, geringerer Suchradius		136,33	0,22	73,79
		kleiner	136,99	0,20	69,06
		größer	134,16	0,24	114,53
TLD*	Standard		334,36	0,22	246,74

Tabelle 1.1: Evaluierungs-Ergebnisse einer Szene mit Größenänderung eines Fahrzeugs. Bester Kandidat: Median-Flow(Standard)

Tracker	Parametersetup	Skalierung	Ø Rechenzeit in ms	Ø gemeinsame Fläche in %	Ø Abstand in Pixeln
Boosting	Standard		88,23	**0,47**	**8,59**
		kleiner	45,35	0,31	14,12
		größer	162,70	0,26	52,75
	weniger Features/Klassifikatoren, geringerer Suchradius, mehr Iterationen bei der Initialisierung		26,69	0,46	9,44
		kleiner	**19,78**	0,05	147,10
		größer	42,61	0,34	60,22
KCF	Standard		**19,28**	**0,47**	**4,41**
		kleiner	**23,84**	0,04	223,33
		größer	28,79	**0,58**	**7,12**
Median-Flow	Standard		33,73	0,15	119,02
		kleiner	39,25	0,10	111,37
		größer	39,81	0,32	36,18
	erhöhte Pixeldichte		48,67	0,16	108,57
		kleiner	49,97	0,10	110,65
		größer	47,62	0,29	31,92
MIL	Standard		106,59	0,06	155,51
		kleiner	121,20	0,32	12,21
		größer	112,68	0,07	146,04
	mehr Features, geringerer Suchradius		137,43	0,06	146,51
		kleiner	136,75	0,05	152,68
		größer	129,59	0,10	99,25
TLD*	Standard		359,07	0,06	555,17

Tabelle 1.2: Evaluierungs-Ergebnisse einer Szene mit mehrfacher Verdeckung eines Fahrzeugs. Bester Kandidat: KCF

Tracker	Parametersetup	Skalierung	Ø Rechenzeit in ms	Ø gemeinsame Fläche in %	Ø Abstand in Pixeln
Boosting	Standard		32,32	0,50	70,87
		kleiner	35,67	0,49	90,53
		größer	43,32	**0,71**	**23,36**
	weniger Features/Klassifikatoren, geringerer Suchradius, mehr Iterationen bei der Initialisierung		**11,54**	0,54	75,02
		kleiner	**11,45**	0,48	**30,74**
		größer	12,59	**0,67**	39,44
KCF	Standard		11,58	0,49	85,15
		kleiner	**9,84**	0,41	73,15
		größer	17,51	0,43	113,43
Median-Flow	Standard		15,45	0,52	67,85
		kleiner	15,80	0,49	55,39
		größer	16,30	0,41	132,28
	erhöhte Pixeldichte		20,09	0,53	62,81
		kleiner	19,90	0,50	50,37
		größer	19,67	0,40	134,59
MIL	Standard		51,84	**0,57**	54,35
		kleiner	50,85	0,43	70,73
		größer	52,82	0,45	99,88
	mehr Features, geringerer Suchradius		61,97	0,53	54,79
		kleiner	57,30	0,44	**33,20**
		größer	60,80	0,47	94,14
TLD*	Standard		183,08	0,19	238,11

Tabelle 1.3: Evaluierungs-Ergebnisse einer Szene mit Verdeckung mehrerer Objekte. Bester Kandidat: Boosting(Standard)

Tracker	Parametersetup	Skalierung	Ø Rechenzeit in ms	Ø gemeinsame Fläche in %	Ø Abstand in Pixeln
Boosting	Standard		48,17	0,19	84,40
		kleiner	34,18	0,25	56,40
		größer	82,99	**0,52**	**25,35**
	weniger Features/Klassifikatoren, geringerer Suchradius, mehr Iterationen bei der Initialisierung		19,24	0,29	92,59
		kleiner	**17,80**	0,07	272,89
		größer	24,45	0,43	52,58
KCF	Standard		29,79	0,34	55,35
		kleiner	**15,59**	0,18	133,31
		größer	**17,48**	0,36	57,39
Median-Flow	Standard		25,07	0,21	236,68
		kleiner	24,91	0,06	342,51
		größer	26,54	0,16	219,57
	erhöhte Pixeldichte		31,56	0,22	235,71
		kleiner	31,50	0,15	238,44
		größer	30,28	0,18	162,70
MIL	Standard		98,17	0,37	62,12
		kleiner	103,80	0,26	56,18
		größer	98,71	**0,56**	**20,20**
	mehr Features, geringerer Suchradius		117,23	0,40	49,96
		kleiner	120,94	0,25	57,21
		größer	119,08	**0,56**	**20,86**
TLD*	Standard		312,60	0,07	475,13

Tabelle 1.4: Evaluierungs-Ergebnisse einer Szene mit sich verändernder Belichtung. Beste Kandidaten MIL(Standard), Boosting(Standard)

Tracker	Parametersetup	Skalierung	Ø Rechenzeit in ms	Ø gemeinsame Fläche in %	Ø Abstand in Pixeln
Boosting	Standard		45,32	0,52	**12,46**
		kleiner	53,60	0,44	16,71
		größer	45,37	0,51	42,46
	weniger Features/Klassifikatoren, geringerer Suchradius, mehr Iterationen bei der Initialisierung		**14,96**	0,51	**14,05**
		kleiner	**14,78**	0,40	16,56
		größer	**14,52**	**0,55**	14,39
KCF	Standard		24,64	0,42	42,49
		kleiner	23,11	0,45	16,59
		größer	23,15	0,43	149,51
Median-Flow	Standard		16,52	0,53	76,23
		kleiner	16,28	**0,61**	26,39
		größer	16,57	0,52	40,42
	erhöhte Pixeldichte		19,98	0,54	72,75
		kleiner	19,86	**0,63**	27,76
		größer	20,17	0,53	40,44
MIL	Standard		60,88	0,39	32,86
		kleiner	61,87	0,43	**13,82**
		größer	59,03	0,43	138,20
	mehr Features, geringerer Suchradius		72,52	0,46	24,95
		kleiner	72,18	0,44	17,16
		größer	69,76	0,47	138,62
TLD*	Standard		177,91	0,18	116,89

Tabelle 1.5: Evaluierungs-Ergebnisse einer Szene mit Dämmerung und langem Verkehrsfluss. Beste Kandidaten: Boosting(angepasst), Median-Flow(angepasst)

Tracker	Parametersetup	Skalierung	Ø Rechenzeit in ms	Ø gemeinsame Fläche in %	Ø Abstand in Pixeln
Boosting	Standard		57,77	0,11	252,98
		kleiner	43,09	0,07	238,09
		größer	93,42	0,29	248,27
	weniger Features/Klassifikatoren, geringerer Suchradius, mehr Iterationen bei der Initialisierung		**18,28**	0,17	219,69
		kleiner	**18,13**	**0,43**	**46,50**
		größer	25,03	**0,31**	**160,05**
KCF	Standard		19,57	0,14	233,31
		kleiner	**14,53**	0,12	242,82
		größer	22,11	0,17	209,39
Median-Flow	Standard		24,71	0,06	452,43
		kleiner	26,14	0,06	451,94
		größer	23,82	0,07	452,43
	erhöhte Pixeldichte		33,35	0,06	452,43
		kleiner	31,11	0,06	452,43
		größer	31,05	0,07	452,43
MIL	Standard		108,10	0,17	181,05
		kleiner	102,50	**0,46**	**36,14**
		größer	105,40	0,19	290,62
	mehr Features, geringerer Suchradius		126,47	0,14	324,73
		kleiner	122,94	0,15	294,21
		größer	124,77	0,17	301,76
TLD*	Standard		87,75	0,17	230,19

Tabelle 1.6: Evaluierungs-Ergebnisse einer Szene mit Regenfall und Nacht. Bester Kandidat: MIL(Standard), Boosting(angepasst)

Abkürzungsverzeichnis

BDSG	Bundesdatenschutzgesetz
CNN	Convolutional Neural Network
DFT	Diskrete Fourier Transformation
DNN	Deep Neural Network
EKF	Erweitertes Kalman-Filter
EU-DSGVO	EU-Datenschutz-Grundverordnung
FB	Forward-Backward error
GOTURN	Generic Object Tracking Using Regression Networks
GUI	Graphical User Interface
HOG	Histogram of Oriented Gradients
IDE	Integrated Development Environment
KCF	Kernalized Correlation Filter
KF	Kalman-Filter
KRLS	Regularized Least Squares with Kernels
LBP	Local Binary Pattern
MIL	Mulitple Instance Learning
NCC	Normalisierte Kreuzkorrelation
QML	Qt Meta-object Language
RBF	Radiale Basis Funktion
RLS	Regularized Least Squares

ROI	Region of Interest
SSD	Sum of Squared Differences
SVM	Support Vector Machine
TLD	Tracking-Learning-Detection
XML	Extensible Markup Language

Literaturverzeichnis

[ASH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012. (zitiert auf Seite 2 und 10)

[BER03] James Black, Tim Ellis, and Paul Rosin. A Novel Method for Video Tracking Performance Evaluation. In *IEEE Int. Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance: VS-PETS*, pages 125–132, 2003. (zitiert auf Seite 2 und 7)

[BS08] Keni Bernardin and Rainer Stiefelhagen. Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics. *EURASIP Journal on Image and Video Processing*, 2008:1–10, 2008. (zitiert auf Seite 2 und 7)

[Bun15] Bundesministerium der Justiz und für Verbraucherschutz. Bundesdatenschutzgesetz: BDSG, 2015. (zitiert auf Seite 12)

[BYB09] B. Babenko, Ming-Hsuan Yang, and S. Belongie. Visual Tracking with Online Multiple Instance Learning. In *Conference on Computer Vision and Pattern Recognition: CVPR*, 2009. (zitiert auf Seite 2, 6 und 26)

[CZR10] Erik Cuevas, Daniel Zaldivar, and Raul Rojas. Kalman filter for vision tracking: Technical Report B 05-12, 2005-08-10. (zitiert auf Seite 18)

[DKFv14] Martin Danelljan, Fahad Shahbaz Khan, Michael Felsberg, and Joost van de Weijer. Adaptive Color Attributes for Real-Time Visual Tracking. In *Conference on Computer Vision and Pattern Recognition: CVPR*, pages 1090–1097, 2014. (zitiert auf Seite 2)

[DT05] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *Conference on Computer Vision and Pattern Recognition: CVPR*, pages 886–893, 2005. (zitiert auf Seite 22)

[Eur16] Europäisches Parlament. Verordnung (EU) 2016/679 des Europäischen Parlaments und des Rates vom 27. April 2016 zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten, zum freien Datenverkehr und zur Aufhebung der Richtlinie 95/46/EG (Datenschutz-Grundverordnung): DSGVO, 2016. (zitiert auf Seite 12)

[Fre95] Yoav Freund. Boosting a Weak Learning Algorithm by Majority. *Information and Computation*, 121(2):256–285, 1995. (zitiert auf Seite 24)

[GB06] Helmut Grabner and H. Bischof. On-line Boosting and Vision. In *Conference on Computer Vision and Pattern Recognition: CVPR*, volume 1, pages 260–267, 2006. (zitiert auf Seite 24 und 26)

[GGB06] H. Grabner, M. Grabner, and H. Bischof. Real-Time Tracking via On-line Boosting. In *Proceedings of the British Machine Vision Conference: BMV*, pages 6.1–6.10, 2006. (zitiert auf Seite 2, 6 und 24)

[HB92] B. L. Harrison and R. M. Baecker. Designing Video Annotation and Analysis Systems. In *Proceedings Graphics Interface: PGI*, pages 157–166, 1992. (zitiert auf Seite 5 und 11)

[HCMB12] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the Circulant Structure of Tracking-by-detection with Kernels. In *European Conference on Computer Vision: ECCV*, 2012. (zitiert auf Seite 21 und 29)

[HCMB15] Joao F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-Speed Tracking with Kernelized Correlation Filters. *IEEE transactions on pattern analysis and machine intelligence*, 37(3):583–596, 2015. (zitiert auf Seite 2, 6, 29, 30 und 31)

[HTS16] David Held, Sebastian Thrun, and Silvio Savarese. Learning to Track at 100 FPS with Deep Regression Networks. In *European Conference on Computer Vision: ECCV*, pages 749–765. 2016. (zitiert auf Seite 2, 6, 33 und 54)

[Its16] Itseez. Open Source Computer Vision Library Version 3.2.0, 2016. (zitiert auf Seite 2, 31, 33 und 41)

[Kal60] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. (zitiert auf Seite 17)

[Kal11] Zdenek Kalal. *Tracking Learning Detection*. Dissertation, University of Surrey - Centre for Vision, Speech and Signal Processing, Guildford, Surrey, England, 2011. (zitiert auf Seite 16 und 31)

[KBB15] Rudolf Kruse, Christian Borgelt, and Christian Braune. *Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. Springer Vieweg, Wiesbaden, 2. aufl. 2015 edition, 2015. (zitiert auf Seite 1 und 9)

[KMM10] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-Backward Error: Automatic Detection of Tracking Failures. In *International Conference on Pattern Recognition: ICPR*, pages 2756–2759, 2010. (zitiert auf Seite 2, 6 und 23)

[KMM12] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-Learning-Detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1409–1422, 2012. (zitiert auf Seite 2, 6 und 31)

[Kre02] Ulrich Krengel. *Einführung in die Wahrscheinlichkeitstheorie und Statistik*, volume 59 of *vieweg studium Aufbaukurs Mathematik*. Vieweg+Teubner Verlag, Wiesbaden, 6. verbesserte auflage edition, 2002. (zitiert auf Seite 20)

[KSC13] Kurt Keutzer, Byung Gon Song, and John Chuang. Driving dataset in the wild: Various driving scenes, 2016-05-13. (zitiert auf Seite 2)

[LK81] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence: IJCAI*, pages 674–679, 1981. (zitiert auf Seite 17)

[McC86] R. K. McConnell. Method of and apparatus for pattern recognition, 1986. US Patent 4,567,610. (zitiert auf Seite 22)

[MD03] David Mihalcik and David Doermann. The Design and Implementation of ViPER, University of Maryland, Institute for Advanced Computer Studies, 2003. (zitiert auf Seite 2, 5 und 35)

[Mul10] Multimedia Knowledge And Social Media Analytics Laboratory. VIA Video Image Annotation Tool, 2010. (zitiert auf Seite 2, 5 und 35)

[MYV07] Dimitrios Makris, Fei Yin, and Sergio Velastin. Performance evaluation of object tracking algorithms. In *Workshop on Performance Evaluation of Tracking and Surveillance: PETS*, 2007. (zitiert auf Seite 2 und 6)

[OPH94] T. Ojala, M. Pietikainen, and D. Harwood. Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. In *Conference on Computer Vision and Pattern Recognition: CVPR*, pages 582–585, 1994. (zitiert auf Seite 23)

[PD10] Bernhard Preim and Raimund Dachselt. *Interaktive Systeme, Band 1: Grundlagen, Graphical User Interfaces, Informationsvisualisierung*. eXamen.press. Springer and Springer-Verlag Berlin Heidelberg, Berlin, 2. aufl. edition, 2010. (zitiert auf Seite 12, 36 und 38)

[PD15] Bernhard Preim and Raimund Dachselt. *Interaktive Systeme, Band 2: User Interface Engineering, 3D-Interaktion, Natural User Interfaces*. eXamen.press. Springer Vieweg, Berlin, 2. aufl. edition, 2015. (zitiert auf Seite 35)

[PPTM+16] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung. A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation. In *Conference on Computer Vision and Pattern Recognition: CVPR*, 2016. (zitiert auf Seite 6)

[Qt14] Company Qt. Qt Library Version 4.8.6, 24. April 2014. (zitiert auf Seite 41)

[RE11] Marc Ritter and Maximilian Eibl. An Extensible Tool for the Annotation of Videos Using Segmentation and Tracking. In *Design, User Experience, and Usability. Theory, Methods, Tools and Practice*, volume 6769 of *Lecture Notes in Computer Science*, pages 295–304. 2011. (zitiert auf Seite 2, 6, 12 und 54)

[Sch78] Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464, 1978. (zitiert auf Seite 20)

[Tön05] Klaus D. Tönnies. *Grundlagen der Bildverarbeitung*. Informatik. Pearson Studium, München, 2005. (zitiert auf Seite 18 und 19)

[VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Conference on Computer Vision and Pattern Recognition: CVPR*, pages I–511–I–518, 2001. (zitiert auf Seite 22)

[VPR13] Carl Vondrick, Donald Patterson, and Deva Ramanan. Efficiently Scaling up Crowdsourced Video Annotation - A Set of Best Practices for High Quality, Economical Video Labeling. *International Journal of Computer Vision*, 101(1):184–204, 2013. (zitiert auf Seite 2, 5, 12, 35, 49, 53 und 54)

[VR11] Carl Vondrick and Deva Ramanan. Video Annotation and Tracking with Active Learning. In *Neural Information Processing Systems: NIPS*. 2011. (zitiert auf Seite 2 und 5)

[WLY13] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online Object Tracking: A Benchmark. In *Conference on Computer Vision and Pattern Recognition: CVPR*, 2013. (zitiert auf Seite 6, 16, 45 und 53)

[YJS06] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object Tracking: A Survey. *ACM Computing Surveys*, 38(4):13–es, 2006. (zitiert auf Seite 2 und 14)

[YRCT09] Jenny Yuen, Bryan Russell, Ce Liu, and Antonio Torralba. LabelMe video: Building a video database with human annotations. In *Conference on Computer Vision and Pattern Recognition: CVPR*, pages 1451–1458, 2009. (zitiert auf Seite 2, 5 und 12)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den 20. März 2017