

Hochschule für Technik und Wirtschaft Berlin
Studiengang Internationale Medieninformatik
Fachbereich 4: Wirtschaftswissenschaften II

Abschlussarbeit zur Erlangung des Akademischen Grades
Bachelor Of Science – B.Sc.

Entfernung von bewegten Objekten aus Videosequenzen

Name: Bernard Christopher Jollans
Matrikelnummer: 539851
Erstprüfer: Prof. Dr. Klaus Jung
Zweitprüfer: Prof. Dr. Kai Barthel
Datum: 20.07.2015



Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Inhaltsverzeichnis

Abkürzungsverzeichnis.....	4
1.Einleitung.....	5
1.1.Kontext.....	5
1.2.Motivation.....	5
1.3.Aufbau dieser Arbeit.....	6
2.Grundlagen.....	7
2.1.Background Subtraction und Background Initialization.....	7
2.1.1.Überblick.....	7
2.1.2.Nackground Subtraction vs. Background Initialization.....	8
2.1.3.RauschProbleme.....	8
2.1.4.Ghosts.....	9
2.1.5.Datensets.....	9
2.1.6.Algorithmen.....	10
2.1.7.Smoothness Assumption.....	11
2.1.8.Aperture Problem.....	11
2.2.OpenCV.....	12
2.3.Markov Random Fields.....	12
2.3.1.Erläuterung.....	12
2.3.2.Iterative Conditional Modes.....	13
3.Algorithmen.....	14
3.1.Median Filtering.....	14
3.2.Smoothering.....	15
3.3.Single Gaussian Mixture Model.....	15
3.4.Mixture of Gaussian.....	17
3.5.Benutzung der Nachbarpixel.....	19
3.5.1.Erklärung.....	19
3.5.2.Markov Random Field Postprocessing.....	20

4.Implementierung	23
4.1.Einführung	23
4.2.Anforderungen	23
4.3.Struktur	24
4.3.1.En/Decoding	24
4.3.2.Algorithmen	26
4.3.3.User Interface	27
4.4.Details	27
4.4.1.Wahl der Programmiersprache	27
4.4.2.Bilddaten	28
4.4.3.Pixelnachbarschaft	29
4.5.Einstellen der Parameter	30
4.5.1.Median Filter	30
4.5.2.Smoothing Filter	31
4.5.3.GMM	32
4.5.4.MOG	33
4.5.5.MRF	36
5.Vergleich	37
5.1.Wahl der Videos	37
5.2.Speicheraufwand	38
5.3.Geschwindigkeitsvergleich	38
5.4.Qualitätsvergleich	40
5.5.Schlussfolgerung	42
Literaturverzeichnis	43
Abbildungsverzeichnis	44
Diagrammverzeichnis	44
Tabellenverzeichnis	44

Abkürzungen

BS = Background Subtraction

BI = Background Initialization

MRF = Markov Random Field

ICM = Iterative Conditional Modes

MM = Mixture Model

GMM = Gaussian Mixture Model

PDF = Probability Density Function (Wahrscheinlichkeitsverteilung)

MOG = Mixture of Gaussian

WB = Wahrscheinlichkeitsbild

R = Rot

G = Grün

B = Blau

Abb = Abbildung

Diag = Diagramm

1.Einleitung

1.1.Kontext

Den Hinter- und Vordergrund eines Videos zu Unterscheiden ist eine relativ altes Problem. Anwendungen der Computer Vision, wie Bewegungsverfolgung oder Bewegungsfeststellung werden mit diesem Wissen wesentlich erleichtert. Geht es darum, den Vordergrund zu extrahieren spricht man oft von Background Subtraction, geht es darum den Hintergrund zu extrahieren, spricht man dagegen von Background Initialization. Man kann hier auch von Entfernung bewegter Objekte aus Videosequenzen sprechen. Beide Disziplinen haben bereits eine Geschichte welche letztes Jahrhundert begonnen hat und sind heute in mehreren Teilgebieten der Computer Vision relevant. Besonders in kameragesteuerter Human Computer Interaction oder automatisierter Videoüberwachung ist oft der erste Schritt, den Hinter- vom Vordergrund zu trennen.

Background Subtraction und Background Initialization hängen eng zusammen und werden im späteren Verlauf dieser Arbeit noch beide genannt werden. Die meisten Methoden der Background Subtraction kann man ebenfalls für Background Initialization benutzen und umgekehrt.

In dieser Arbeit sollen verschiedene Herangehensweisen an diese Probleme implementiert und verglichen werden, mit dem jeweiligen Ziel den Hintergrund eines Videos zu extrahieren.

1.2.Motivation

Die ursprüngliche Motivation zu dieser Arbeit kam von der Frage ob es möglich ist, an einem Sonntag ein Bild von einer Sehenswürdigkeit wie dem Brandenburger Tor zu schießen, ohne Touristen auf dem späteren Bild zu erkennen. Eine Lösung des Problems wäre sicherlich, alle Touristen dazu zu bringen aus dem Weg zu gehen. Dies lässt sich jedoch leider nicht bei jeder Sehenswürdigkeit anwenden.

Ein zweiter denkbarer Lösungsansatz wäre, statt einem Foto ein Video aufzunehmen, zu warten, bis jeder Teil des Brandenburger Tors sichtbar geworden ist und diese sichtbaren Teilen aus dem Video später manuell zu einem Bild zusammenzufügen. Die Frage dieser Arbeit war, ob sich dies auch automatisch erledigen lässt.

Nach etwas Recherche zeigte sich, dass dieses Problem nicht neu ist. Ähnliche Probleme

wurden bereits gelöst. Beispielsweise die Erkennung und Verfolgung von Autos mit einer Überwachungskamera wird wesentlich erleichtert durch die Entfernung des Hintergrundes der Videosequenz.

Diese und andere ähnliche Probleme werden unter den Titeln Background Subtraction oder Background Initialization beschrieben und gelöst. In dieser Arbeit sollen verschiedene Verfahren und Ansätze zu diesen Problemen erklärt und verglichen werden, nach ihrer Effektivität und Leistungsfähigkeit.

1.3.Aufbau dieser Arbeit

Diese Arbeit ist nach folgendem Schema aufgebaut. Zuerst soll es eine kurze Erläuterung der für diese Arbeit gebrauchten Grundlagen geben. Danach sollen einige Algorithmen zur Lösung des Problems vorgestellt werden. Diese sollen danach in einer Anwendung implementiert und getestet werden. Die Ergebnisse der Tests sollen dann vorgestellt und verglichen werden, um Entscheidungen für spätere ähnliche Arbeiten zu erleichtern.

2.Grundlagen

2.1.Background Subtraction und Background Initialization

2.1.1.Überblick

Background Subtraction (BS) ist ein wichtiger Teil von Computer Vision. Sie wird benutzt zur Vordergrund Extraktion und Bewegungsdetektion bei Videos. Diese Videos sind in der Regel weitgehend unbewegt, wie bei Überwachungskameras, live Webcams oder Kameras für Human Computer Interaction. Oft ist das Ziel, die genauere Erkennung von Menschen oder Fahrzeugen zur genaueren nachherigen Analyse. Der Großteil aller automatisierten Überwachungsalgorithmen beginnt mit einer Art BS. Als Ergebnis der BS wird zwar oft der Eigentliche bewegte Vordergrund angesehen, das Endprodukt kann aber auch einfach nur der Umriss des Vordergrundes in einem Binärvideo sein (siehe Abb.1). BS ist ein noch eher junges Thema, das erst seit etwa drei-fünf Jahren Anwendung findet und erforscht wird. Dennoch gibt es bereits einige Standardverfahren an BS heranzugehen. Die wichtigsten Verfahren werden später in dieser Arbeit genauer beleuchtet.

Background Initialization (BI) könnte als das Gegenteil von BS betrachtet werden. Bei BS ist das Ziel den Vordergrund zu extrahieren, wobei es bei dieser Arbeit, sowie bei BI darum geht den Hintergrund zu extrahieren. Beide Teilgebiete sind relevant füreinander sowie für diese Arbeit. Extraktion des Hintergrundes wird in der Literatur oft Background Model Initialization (BMI) genannt, wobei der extrahierte Hintergrund Background Model (BM) genannt wird.



Abb.1 Ergebnis einer Vordergrund-Segmentierung der Background Models Challenge(BMC)[1]

2.1.2. Background Subtraction vs. Background Initialization

Der BS Vorgang kann oft als etwas wie eine Pipeline verstanden werden. In dieser Pipeline ist der erste Schritt die Modellierung des Hintergrundes. Der zweite Schritt ist die Subtraktion des Hintergrundes aus der Videosequenz. Der erste Schritt hierbei ist gleichzusetzen mit dem Prozess der BI.

Alternativ kann der Vorgang mit einer Segmentierung des Videos in Vordergrund- und Hintergrund Bereiche beginnen. Hierbei wäre der zweite Schritt das Maskieren der einzelnen Frames mit einem aus der Segmentierung resultierenden Bild (siehe Abb.1).

Umgekehrt kann eine BI Pipeline beschrieben werden. Analog zu der zweiten Art von BS Pipeline kann auch eine BI mit der Segmentierung in Vorder- und Hintergrund beginnen. Allerdings kommt der zusätzliche Schritt der Hintergrundschätzung hinzu, für die Stellen, an denen sich ein Vordergrundobjekt befindet.

Die Ähnlichkeit der beiden Prozesse macht ihre Relevanz für einander deutlich.

2.1.3. Rauschprobleme

Ein großes Problem bei der BS/BI ist das Rauschen in Videos. Dieses Rauschen kann durch sich bewegende Blätter eines Baumes, Wasser oder Veränderung der Helligkeit entstehen. Bei einer simplen Bewegungserkennung durch das Bilden der Differenz zweier aufeinanderfolgenden Frames in Abb.2 zeigt sich dieses Problem deutlich.

Das rechte Bild ist aus einem Differenzbild der linken beiden entstanden. Statt nur die Umrisse der Bewegung der Menschen zu sehen, sieht man viel Bewegung überall. Diese Bewegung kann auf die Veränderung in der Helligkeit zurückgeführt werden. Das Problem dieser Helligkeitsveränderung tritt in allen Videos auf, welche unter freiem Himmel gemacht wurden, jedoch weniger bei denen, welche beispielsweise in einem Labor entstanden sind. Bei dem Vergleich verschiedener Methoden zur BS/BI ist es wichtig auf diese beiden unterschiedlichen Situationen einzugehen.



Abb.2 Bilddifferenz von zwei Frames [1]

2.1.4.Ghosts

Ein ähnliches Problem bei BI, welches in dieser Arbeit noch vorkommen wird, stellen sogenannte Ghosts dar. Diese Ghosts sind überbleibende Helligkeitsveränderungen nach der BI, welche den bewegten Objekten im Video zuzuordnen sind. Der Name Ghost kommt daher, dass sie wie Geister der entfernten Objekte aussehen.

2.1.5.Datensets

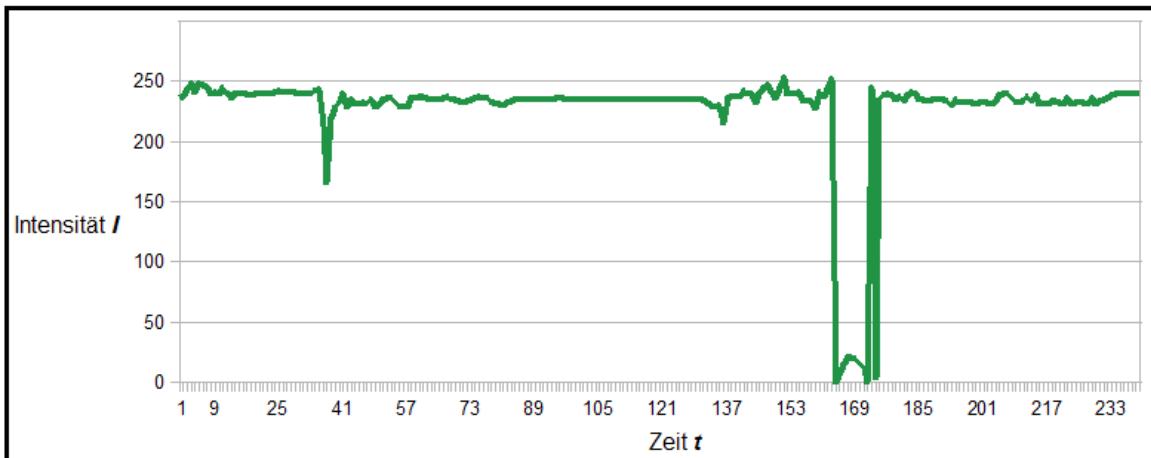
Es gibt einige im Internet kostenlos zur Verfügung stehende Datensets, welche in wissenschaftlichen Arbeiten bereits für BS/BI benutzt wurden. Alle Videos in diesen Datensets wurden mit stationären unbewegten Kameras aufgenommen, meistens von Überwachungskameras. Es gibt einige große Unterschiede zwischen Datensets. Einige liefern zusätzlich zu Videomaterial noch ein manuell aufgenommenes Hintergrundbild, oder den manuell segmentierten Vordergrund, um es möglich zu machen lernende Algorithmen zu implementieren.

Für diese Arbeit wird lediglich Videomaterial benötigt. Allerdings sollte ein Teil der Videos unter freiem Himmel gemacht worden sein, ein anderer Teil nicht (siehe Rauschproblem). Als Testdatensets werden hier folgende benutzt.

1.BCM[1] steht für Background Modeling Challenge. Für diese Herausforderung werden einige Videos zur Verfügung gestellt, mit denen man seine Background Modeling Algorithmen testen kann. Das Video namens "Wandering Students" wird in einigen Abbildungen dieser Arbeit verwendet.

2.VISOR[2] steht für Video Surveillance Online Repository. Es ist eine online Sammlung von Überwachungsvideos vieler Art. Da dieses Set mehr für automatische Überwachungsanwendungen verwendet wird, sind nicht alle Videos auf diese Arbeit zugeschnitten. Es gibt einen Teil dieses Sets, der sich "Shadows" nennt. Die Videos aus diesem Teil wurden speziell für das Testen von Algorithmen, welche etwas mit Schatten zu tun haben herausgesucht. In dieses Videos sind auch immer bewegte Objekte, weshalb sie auch in dieser Arbeit benutzt werden.

3.EWAP[3] steht für ETH Walking Pedestrians und ist ein Datenset von der ETH Zürich. In diesem Datenset sind zwei Videos enthalten, auf denen man Fußgänger in Bewegung sieht. Diese Videos wurden nicht primär für BS/BI erstellt, sind jedoch dafür geeignet.



Diag.1 Pixelintensitäten I zur Zeit t

2.1.6.Algorithmen

Da BS und BI sehr viel miteinander zu tun haben, teilen sie auch ihre Herangehensweisen. Für beide gibt es ein paar sehr häufig verwendete Verfahren, welche an verschiedenen Stellen der Prozesse eingesetzt werden. Der Anspruch an BS/BI Verfahren ist in der Regel, dass das jeweilige Video mit einer stationären Kamera aufgenommen wurde. Diesen Anspruch hat auch diese Arbeit.

In Diag.1 sieht man die Intensität $I(x|y)$ eines Pixels an einer zufälligen Stelle $(x|y)$ zur Zeit t . Zu den Zeiten $t \approx 30$ und $t \approx 169$ kann eine starke Veränderung der Intensität beobachtet werden. Dies ist dem bewegten Vordergrund zuzuschreiben. Dieses Diagramm soll dazu dienen Die Vorstellung der Prozesse in dieser Arbeit zu erleichtern.

Es gibt verschiedene Ansätze zum BS/BI und dazu passend viele verschiedene Arbeiten. Die meist vorkommenden Algorithmen approximieren für jeden Pixel ein Model, welches den Hintergrund beschreibt. Diese Approximation geschieht auf Grundlage der vorhergehenden Frames. Fast alle in dieser Arbeit vorgestellten Algorithmen funktionieren auf diese Art. Die Approximation ist zwangsläufig ein Teil aller solcher Algorithmen, die Frames auf deren Grundlage diese geschieht sind jedoch variabel. Am genauesten wäre es wahrscheinlich mit einer Analyse aller Frames zu beginnen. In vielen Fällen (z.B. Videoüberwachung), stehen jedoch noch nicht alle Frames zur Verfügung. Außerdem müsste man bei gleichzeitiger Analyse aller Frames den Zugriff auf alle Frames ermöglichen. Da es viel zu kostspielig wäre alle decodierten Frames im Arbeitsspeicher zu halten müsste man bei Neugebrauch jedes einzelnen Frames während der Analyse, das Video neu decodieren. Dies würde zu lange dauern. In dieser Arbeit werden deswegen nur Algorithmen behandelt, welche eine vorher festgelegte Zahl an vorherigen Frames als Grundlage der Hintergrundapproximation nehmen.

2.1.7.Smoothness Assumption

Die Smoothness Assumption (dt. Glätte-Annahme) spielt oft eine große Rolle in Computer Vision. Sie besagt, dass Pixel die nahe beieinander liegen mit höherer Wahrscheinlichkeit zum selben Objekt gehören, als Pixel die weit auseinander liegen. Diese Annahme folgt natürlicher Intuition. Sie hilft bei der Bestimmung von Abhängigkeiten verschiedener Pixel. Außerdem gilt diese Annahme nicht nur in den beiden räumlichen Dimensionen, sondern auch in der zeitlichen. Im Falle eines Videos von Straßenverkehr bedeutet das zum Beispiel, dass die Autos als genau eine Fläche beschrieben werden können (räumliche Dimensionen) und nicht von der einen Seite des Bildes zur Anderen springen, ohne die Mitte zu passieren (zeitliche Dimension)

2.1.8.Aperture Problem

Das Aperture Problem beschreibt eine Problematik, welche entsteht wenn man nur einen Ausschnitt eines Bildes sehen kann. Bewegt sich eine Diagonale Linie durch ein Bild, auf dem man nicht alles sehen kann, weiß man oft nicht genau wie diese Bewegung aussieht. Auf der Abbildung oben sieht man dieses typische Beispiel für das Aperture Problem. Links sieht man nur einen Ausschnitt des Bildes und denkt, dass sich die Diagonale nach unten rechts bewegt. Rechts sieht man jedoch, dass sich die Diagonale eigentlich nur nach rechts bewegt.

In der Bewegungserkennung, sowie bei BS/BI sieht man dieses Problem besonders, wenn sich monotonfarbene große Objekte durch das Bild bewegen. In diesem Fall ist es mit den Daten eines einzelnen Pixels, über welches sich das Objekt bewegt, unmöglich die Richtung der Bewegung oder gar überhaupt eine Bewegung zu erkennen.

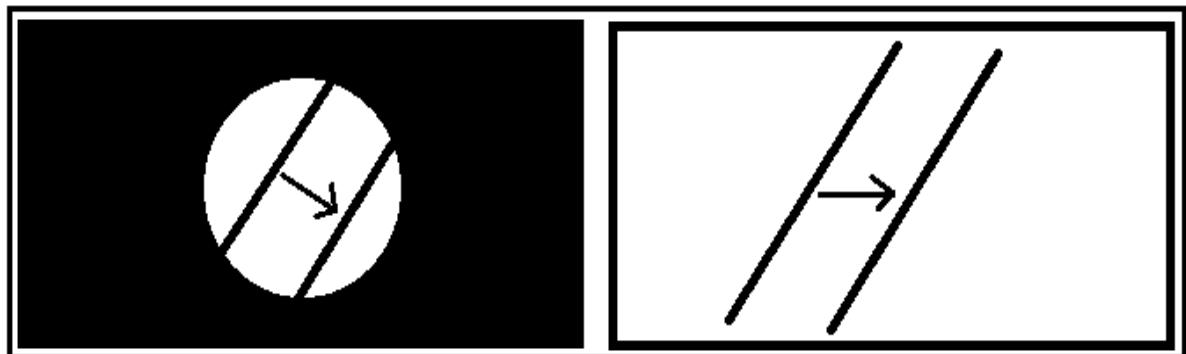


Abb.3 Aperture Problem

2.2.OpenCV

OpenCV (Open Computer Vision) ist eine open source Bibliothek, welche Algorithmen für Bild- und Video-Verarbeitung bereitstellt. Sie wurde von Intel angefangen, wird zur Zeit von Willow Garage aufrechterhalten und ist mittlerweile das verbreitetste Tool ihrer Art. Die Bibliothek wurde in C++ geschrieben und wird hauptsächlich in dieser Sprache benutzt. Sie kann jedoch auch in Java benutzt werden mithilfe von Wrappern. In dieser Arbeit wird die Wrapper Bibliothek JavaCV benutzt, welche in folgendem Github Repository gefunden werden kann <https://github.com/bytedeco/javacv> [27.06.2015].

Obwohl hier bereits einige hilfreiche Algorithmen der Bildverarbeitung realisiert wurden, wird OpenCV in dieser Arbeit lediglich als Encoder/Decoder der Videodaten benutzt. Alle benötigten Algorithmen und Teilalgorithmen sollen selbst implementiert werden, um die Arbeit nachvollziehbarer zu gestalten.

2.3.Markov Random Fields

2.3.1.Erläuterung

Markov Random Fields (MRF) sind endliche ungerichtete Graphen, welche oft genutzt werden um Bilder zu segmentieren. Simon A. Barker hat 1998 als erster eine solche Bildsegmentierung beschrieben [4]. Die Knoten des Graphen sind dabei Zufallsvariablen. Der Graph ist ausgezeichnet durch sogenannte Cliques. Diese Cliques sind Zusammenschlüsse aus benachbarten Knoten. Was ein MRF auszeichnet ist, dass jeder Knoten unabhängig ist von allen Anderen, außer seinen Nachbarknoten. Bei einer Menge von Knoten X und einem beliebigen Knoten $X_n \in X$ mit einer Knotennachbarschaft X_N , lässt sich sagen:

$$P(X_n | X - X_n) = P(X_n | X_N)$$

Bei dem Gebrauch eines MRF geht es oft darum die gesamte Energie des Feldes zu minimieren. Weil sich diese aus der Energie der einzelnen Cliques zusammensetzt, geht es in der Praxis darum die Energie der Cliques zu minimieren. Die Energie der Cliques definiert sich durch eine Energiefunktion $E(X_N, X_n)$. Wie diese Energiefunktion aussieht ist von Anwendung zu Anwendung unterschiedlich. Bei den meisten Problemen der Bildsegmentierung, welche mit MRFs gelöst werden jedoch sieht sie nach diesem Schema aus:

$$E(X_N, X_n) = E_P(X_N, X_n) * E_S(X_N, X_n)$$

Wobei $E_P(X_N, X_n)$ die Energie des eigentlichen MRFs ist und $E_S(X_N, X_n)$ die Energie des Bildes ist. Die Energien richten sich in der Regel nach der Glätte bzw. Ähnlichkeit von Nachbarn in dem jeweiligen Graphen. In einem Bild heißt das, dass benachbarte Pixel eher ähnlich aussehen sollen (siehe Smoothness Assumption). In dem MRF heißt das, dass benachbarten Pixeln ähnliche Wahrscheinlichkeiten zugrunde liegen. Eine typische Energiefunktionen in einem Bild mit zwei Graustufen Pixeln wäre[5]:

$$E(x) = e^{\frac{-1}{T}(x_1 - x_2)^2}$$

Wobei x_1 und x_2 die Intensitäten der Pixel sind. Hat man eine Energiefunktion gewählt gilt es mithilfe dieser, die Energie des gesamten MRFs zu minimieren, bzw. die Energie der einzelnen Cliques. Für diesen Vorgang gibt es mehrere mögliche Algorithmen. Ein sehr beliebter ist zum Beispiel der sogenannte Gradient Descent, welcher mit der Ableitung einer Energiefunktion die Energie der Eingangsparameter minimiert. Gradient Descent wird für viele Probleme genutzt und wäre auch hier anwendbar. Jedoch gibt es eine andere weniger kostspielige Methode, namens Iterative Conditional Modes (ICM)

2.3.2. Iterative Conditional Modes

Der Iterative Conditional Modes[6] ist ein Algorithmus zur Optimierung der Energie eines MRF. Er wurde entwickelt um die hohen Rechenkosten anderer Algorithmen, mit dem selben Ziel, zu umgehen. Die Idee ist sehr naheliegend. Für jeden Knoten X_n im MRF wird mithilfe aller Informationen seiner Nachbarschaft X_N der Wert gefunden, welcher die Energiefunktion am weitesten minimiert. Dies wird so oft durchgeführt, bis keine starke Veränderung der Knoten mehr erkennbar ist. Das dies passiert ist, anders als bei anderen Algorithmen garantiert, da in keinem Schritt die Energiefunktion erhöht werden kann und deshalb Zyklen unmöglich sind. In der Praxis, reichen etwa sechs Durchläufe dieses Algorithmus aus[6].

3.Algorithmen

Im Folgenden werden verschiedene Verfahren der BS/BI vorgestellt werden, welche in dieser Arbeit realisiert und verglichen werden sollen.

3.1.Median Filtering

3.1.1.Erklärung

Median Filter werden außerhalb von BS und BI viel genutzt. Sie fungieren dazu den Medianwert einer Gruppe von Werten zu ermitteln indem Sie ihren Median bestimmen. Hier wie auch in anderen Anwendungsgebieten, versucht man mit einem Medianfilter das Rauschen zu minimieren. Das Rauschen im diesem Falle sind Pixelintensitäten die bewegtem Vordergrund zuzuschreiben sind. Der Median Filter hat auf den ersten Blick den deutlichen Vorteil der algorithmischen Einfachheit.

3.1.2.Berechnung

In dieser Arbeit wird der Median eines Frames des Ausgangsvideos zu der Zeit t mithilfe der n Frames des Eingangsvideos von den Zeiten $t-n$ bis t ermittelt. Die Variable n ist dabei also die Menge der Eingangsframes.

Der Median muss für jeden Frame noch einmal von vorne berechnet werden. Für jeden Frame müssen also die Pixel n anderer Frames sortiert werden. Dies ist relativ kostspielig und dauert somit vergleichsweise lange. Wie Cuchira et. al in [8] allerdings experimentell gezeigt haben, funktioniert ein Median Filter hier immer noch gut, wenn man als input nur etwa jeden zehnten Frame nimmt. In dieser Arbeit wird dies jedoch nicht getan. Jeder Frame soll für die Berechnung gebraucht werden.

3.2.Smoothing

3.2.1.Erklärung

Smoothing (dt.:Glättung) ist ein mit dem Medianfilter vergleichbares Verfahren. Es wird ebenfalls in vielen Bereichen genutzt unter anderem um Rauschen zu minimieren. Zu oft genutzten Smoothing Filtern gehören der Mittelwerts-Filter sowie der Gauss-Filter. Normalerweise wird ein Smoothing Filter lediglich als Teil des Preprocessing genutzt. In dieser Arbeit soll jedoch beleuchtet werden, wie gut er für die eigentliche BI/BS zu gebrauchen ist.

3.2.2.Berechnung

Genauso wie beim Medianfilter werden n Frames zur Hilfe genommen um den Mittelwert der Pixelintensitäten der Frames zu errechnen. Hier wird ebenfalls keine rechnerische Optimierung betrieben, weshalb die Berechnung des Mittelwertes μ zur Zeit t an der Stelle $(x|y)$ folgendermaßen aussieht:

$$\mu = \sum_{i=(t-n)}^t \frac{I_i(x|y)}{n}$$

I ist hierbei die Intensität des Pixels.

3.3.Single Gaussian Mixture Model

3.3.1.Erklärung

Mixture Models (MM) sind statistische Modelle, welche versuchen eine bestimmte Untergruppe einer Gruppe von Daten zu beschreiben. Diese Beschreibung erfolgt mithilfe einer bestimmten Wahrscheinlichkeitsverteilung (eng. Probability Density Function PDF). Bei BS/BI wird oft versucht mit einem solchen MM den Hintergrund eines Pixels des jeweiligen Videos zu beschreiben. Der Pixel $I(x|y)$ an der Stelle $(x|y)$ kann dabei als Zufallsvariable der jeweiligen PDF angesehen werden. $I(x|y)$ beschreibt dabei die Wahrscheinlichkeit des Auftretens der verschiedenen Pixelintensitäten an der Stelle $(x|y)$. Es könnten verschiedene PDFs benutzt werden, doch in der Praxis findet eine Gauß-Verteilung meistens Anwendung. Ein MM mit einer solchen Verteilung wird gerne Gaussian Mixture Modell (GMM) genannt. Das Verfahren der BS/BI das ein GMM zum Modellieren des Hintergrundes nutzt wird gerne auch einfach Single Gaussian genannt.

3.3.2.Berechnung

Das Single Gaussian Verfahren, funktioniert folgendermaßen. Bei jedem bearbeiteten Frame wird die PDF an das Video angepasst. Da die genutzte PDF eine Gaußsche Normalverteilung ist, sieht sie für den Wert X folgendermaßen aus:

$$\eta(X|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} * e^{-\frac{(X-\mu)^2}{2\sigma^2}}$$

Für die Beschreibung dieser PDF werden zwei Werte benötigt:

1.Der Mittelwert μ . Um diesen Wert zu erhalten bräuchte man wie beim Smoothing die letzten n Frames. Um Speicher und Rechenzeit zu sparen, wird der Mittelwert nur approximiert. Die Approximation zum Zeitpunkt t erfolgt wie in [8] durch

$$\mu_t = (1-\alpha)*\mu_{t-1} + \alpha*I_t$$

Wobei I_t die Pixelintensität zur Zeit t ist. Und α die Lernrate ist mit der sich der Mittelwert an Veränderungen im Video anpassen soll.

2.Die Varianz σ . Diese würde sich normalerweise ähnlich kostspielig wie der Mittelwert des Smoothing berechnen, mit einer Iteration über die letzten n Frames. Um diese Kosten ebenfalls zu sparen wird σ genauso wie μ mithilfe der Lernrate α approximiert.

$$\sigma_t = (1-\alpha)*\sigma_{t-1} + \alpha*(\mu_t - I_t)^2$$

Für jeden neuen Pixel im Video muss getestet werden, ob er zum Hintergrund oder Vordergrund gehört. Naheliegend wäre es, die Wahrscheinlichkeit der Zugehörigkeit zu der jeweiligen Gauß-Verteilung zu berechnen. Ab einer bestimmten Größe dieses Wertes, wird der Pixel als Hintergrund gesehen. Eine einfachere Berechnung erfolgt in dieser Arbeit nach dem Vorbild von Stauffer und Grimson[7], indem man den Abstand der Intensität des Pixels von dem Mittelwert der jeweiligen Verteilung mit der Varianz vergleicht. Es gilt wenn

$$[I(x|y) - \mu(x|y)]^2 > \phi \sigma^2$$

dann wird der Pixel dem Vordergrund zugeordnet, sonst dem Hintergrund. ϕ Ist hierbei ein Faktor für den in dieser Arbeit, nach dem Vorbild von [7], gilt $\phi=2.5$.

Die Intensitäten bei diesem GMM sind die Intensitäten der R, G und B Werte der einzelnen Pixel unabhängig voneinander. Eine genauere Beschreibung der Intensitäten würden Vektoren im RGB-Raum wiedergeben, da dort die Einzelteile der Farben eben nicht unabhängig sind. Die Dimension der PDF des MMs würde damit von der Ersten auf die Dritte steigen, was Berechnungen um einiges komplexer und kostspieliger machen würde. Um diese Kosten zu sparen, wird hier lediglich mit eindimensionalen GMMs gearbeitet.

Dies geschieht nach dem Vorbild und auf Basis der Arbeit von Stauffer und Grimson [7], die gezeigt haben, dass man hierdurch keinen starken Verlust an Qualität erleidet.

3.4.Mixture of Gaussian

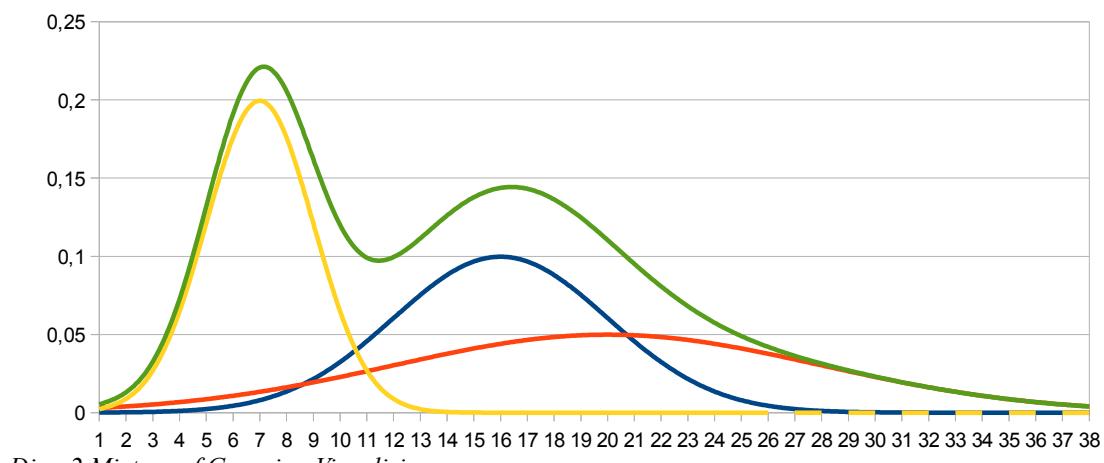
3.4.1.Erklärung

Mixture of Gaussian (MOG) könnte als Mixture of Gaussian Mixture Models ausgeschrieben werden. Auch hier werden Gauß-Verteilungen zur Beschreibung des Hintergrundes genutzt, diesmal jedoch nicht nur eine, sondern 3-5 um den Hintergrund bei vielen Veränderungen immer noch genau beschreiben zu können. In Diag.2 kann man sehen, wie man sich einen MOG vorstellen kann. Die gelbe, blaue und rote Linie beschreiben Gauß-Verteilungen mit verschiedenen Mittelwerten und Varianzen. Die grüne Linie ist eine Mischung dieser drei Verteilungen.

Die Methode des MOG wurde erstmals von Stauffer und Grimson benutzt [7]. Sie war nicht als Methode des BI sondern für BS gedacht, um Verkehr zu überwachen. Heute ist Stauffer und Grimsons Arbeit eine der meist gebrauchten Methoden des BS und wird in vielen anderen Arbeiten als Grundlage genutzt.

3.4.2.Berechnung

Die Berechnung des MOG erfolgt nach Vorbild von Stauffer und Grimson[7]. Genauso wie beim vorherigen GMM, sind auch hier die PDFs eindimensional um Rechenzeit zu sparen. Zu den gebrauchten Werten beim GMM kommt hier noch ein zusätzlicher Wert des Gewichtes hinzu. Das Gewicht soll beschreiben, wie relevant die jeweilige PDF für den Hintergrund ist.



Die Berechnung der Werte beginnt mit der Entscheidung welchen Verteilungen sich die Pixelintensität I_t zum Zeitpunkt t zuordnen lässt. Genauso wie beim GMM lässt sie sich zuordnen, wenn:

$$(I_t - \mu_{t-1})^2 < \phi * \sigma_{t-1}$$

Wobei ϕ ein Faktor ist, für den in dieser Arbeit, genauso wie beim GMM stets $\phi = 2,5$ gilt. In Diag.3 ist der Datenbereich eingerahmt, welcher nach dieser Bedingung zu der Gauß-Verteilung im Bild gehören würde:

Die Wahl des Wertes kommt ebenfalls aus [7], aber findet in dieser Visualisierung noch einmal Begründung

Die Werte werden nun wie folgt berechnet:

1. Der Mittelwert μ zu der Zeit t errechnet sich für alle Verteilungen, denen I_t zuzuordnen ist nach der Formel:

$$\mu_t = (1-p)\mu_{t-1} + pI_t$$

2. Die Varianz σ zu der Zeit t errechnet sich für alle Verteilungen, denen I_t zuzuordnen ist nach der Formel:

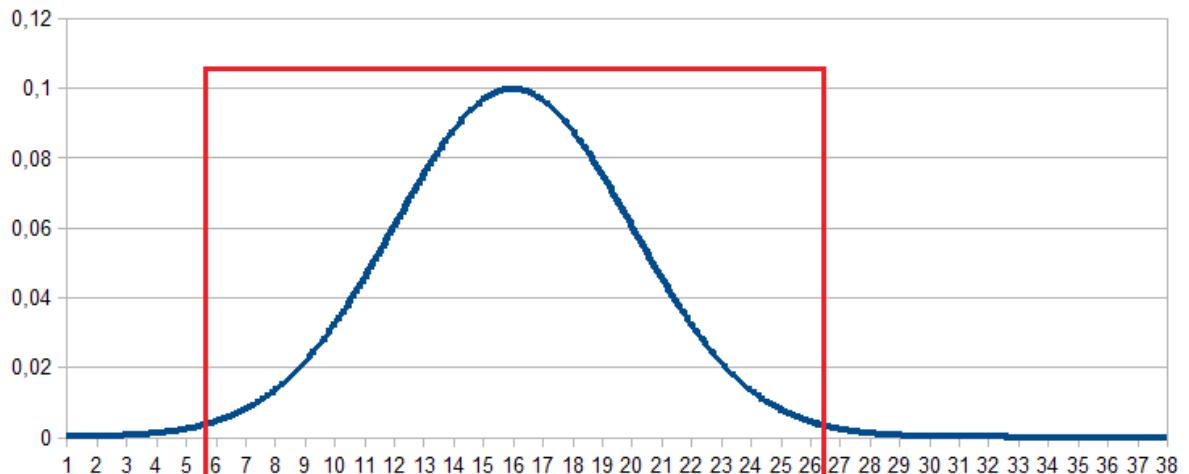
$$\sigma_t^2 = (1-p)\sigma_{t-1}^2 + p(I_t - \mu)^2$$

Wobei p die Wahrscheinlichkeit der Zugehörigkeit von I_t zu dem jeweiligen GMM ist, verrechnet mit der Lernrate α .

$$p = \alpha \eta(I_t | \mu, \sigma)$$

Die Berechnung der Varianz und des Mittelwertes sollte als eine Art Interpolation zwischen alten und neuen Werten verstanden werden.

Für alle Verteilungen denen I_t nicht zugeordnet werden kann, bleiben μ , sowie σ gleich.



Diag.3 Visualisierung der Zuordnung zu einer Gauß-Verteilung

3.Das Gewicht ω zur Zeit t berechnet sich für das GMM nummer k nach der Formel:

$$\omega_{t,k} = (1-\alpha)\omega_{k,t-1} + \alpha(M_{k,t})$$

Wobei $M_{k,t}=1$ für alle GMMs denen I_t zugeordnet wurde und $M_{k,t}=0$ für alle Anderen. Nach der Aktualisierung der Gewichte, werden sie normalisiert um dannach als Prozente angesehen werden zu können.

Lässt sich I_t keinem GMM zuordnen, so wird der Mittelwert, des GMMs mit dem niedrigsten ω , gleich I_t gesetzt und die Varianz einer relativ großen Zahl gleichgesetzt. Nach der Zuweisung muss entschieden werden, ob I_t dem Hinter- oder Vordergrund zugewiesen wird. Um dies zu ermitteln wird folgendermaßen vorgegangen. Zuerst werden die einzelnen GMMs geordnet nach der Größe des Wertes $\frac{\omega}{\sigma}$. Dieses Kriterium kommt daher, dass ein kleineres Gewicht und eine größere Varianz beide darauf hinweisen, dass die Verteilung eher neu ist.

Nach der Ordnung der Verteilungen, werden die ersten B GMMs ausgewählt nach dem Kriterium:

$$B = \operatorname{argmin}_b \left(\sum_{k=0}^b w_k > T \right)$$

Wobei T ein Wert zwischen 0 und 1 ist, der vorgibt was für einen Anteil der Hintergrund am Video haben soll. Ist das GMM dem I_t zugeordnet wurde unter den ersten B Verteilungen, ist es ein Teil des Hintergrundes, sonst ein Teil des Vordergrundes.

Sollte I_t zum Vordergrund gehören, wird es im Ausgangsvideo ersetzt, durch den Mittelwert des, durch das Bewertungskriterium $\frac{\omega}{\sigma}$ am höchsten bewertete GMM.

3.5.Benutzung der Nachbarpixel

3.5.1.Erläuterung

Ein großer Kritikpunkt an den bisher vorgeschlagenen Methoden ist, dass sie jeden Pixel unabhängig von seinen Nachbarn betrachten. Es ist jedoch wohl bekannt, dass in Bildern benachbarte Pixel ähnliche Eigenschaften haben (siehe Smoothness Assumption). Aus der Nachbarschaft der Pixel könnte man also zusätzliche Information ziehen.

Methoden, welche auf diesem Punkt aufbauen versuchen also das Problem der BS/BI nicht nur mit zeitlich sequentiellen Hinweisen, sondern auch mit räumlichen Hinweisen zu lösen. Viele solcher Methoden sind sehr Komplex in ihrem Rechenaufwand, was ihnen einen Nachteil einbringt. Selbstverständlich haben sie dafür den Vorteil, dass sie genauer sind.

Bei Beachtung der Nachbarpixel verwandelt sich das Problem dieser Arbeit in eines, welches sich mit Bildsegmentierung vergleichen lässt, wobei hier die Nachbarpixel eher für Postprocessing genutzt werden. Der entscheidende Unterschied zur Bildsegmentierung ist, dass wir hier bereits Informationen über die Pixel gewonnen haben, durch ihre zeitlichen Vorgänger und die oben genannten Methoden (GMM u. MOG). Diese Information ist die Wahrscheinlichkeit, dass der Pixel zum Hintergrund/Vordergrund gehört. Da alle Pixel räumlich abhängig sind und jeder Pixel eine Wahrscheinlichkeit hat, können die Wahrscheinlichkeiten der Pixel als ein 2D Signal, ähnlich wie ein Bild angesehen werden. Dieses Signal soll Wahrscheinlichkeitsbild (WB) genannt werden.

3.5.2. Markov Random Field Postprocessing

Eine oft in ähnlichen Zusammenhängen genannte Methode ist die Verwendung eines MRF (siehe Markov Random Field) für das Postprocessing des WB [4]. Mit dem MRF soll die Energie des WB minimiert werden. Der erste Schritt dazu, ist die Definition einer Energiefunktion $E(X_N, X_n)$. Diese Energiefunktion sollte abhängig von dem WB sowie von dem eigentlichen Frame sein. Bei großen Unterschieden zwischen benachbarten Werten des WB sollte die Energie groß sein, bei kleinen eher klein. Ein Unterschied zwischen den Werten des WB $P(x|y)$ und $P(x+1|y)$ sollte jedoch mehr Einfluss auf die Energie haben, wenn die jeweiligen Werte $I(x|y)$ und $I(x+1|y)$ im korrespondierenden Frame ähnlich sind. Eine Beschreibung der Energie an dieser Stelle könnte also wie folgt aussehen:

$$\frac{P(x|y) - P(x-1|y)}{I(x|y) - I(x-1|y)}$$

Diese Beschreibung der Energie macht Sinn, weil benachbarte Pixel, welche ähnliche Farben haben wahrscheinlich zum gleichen Objekt gehören (siehe Smoothness Assumption). Ein Unterschied in ihren korrespondierenden Werten des WB, macht deswegen weniger Sinn. In erster Linie geht es hierbei jedoch nicht um die Findung der perfekten Energiefunktion, sondern um die Minimierung der Energie des WB. An dieser Stelle wird ein ICM Algorithmus benutzt (siehe Grundlagen). Das heißt, dass in jeder Iteration über das WB für jeden Knoten des WB anhand seiner Nachbarn der Wert gefunden werden soll, welcher seine Energie maximal minimiert. Um diesen Wert zu berechnen benutzen wir einen Filter mit Gewichten, welche abhängig von dem zum WB korrespondierenden Frame sind:

$\zeta *$	$\frac{1}{(I(x-1 y-1)-I(x y))}$	$\frac{1}{(I(x y-1)-I(x y))^2}$	$\frac{1}{(I(x+1 y-1)-I(x y))}$
	$\frac{1}{(I(x-1 y)-I(x y))^2}$	0	$\frac{1}{(I(x+1 y)-I(x y))^2}$
	$\frac{1}{(I(x-1 y+1)-I(x y))}$	$\frac{1}{(I(x y+1)-I(x y))^2}$	$\frac{1}{(I(x+1 y+1)-I(x y))}$

Diag.4 ICM Filter Kernel

ζ ist hierbei ein Faktor zur Normalisierung der Werte.

Wenn der Farbunterschied zwischen dem Pixel X_n und einem seiner Nachbarpixel groß ist, dann ist der zugehörige Faktor im Filterkernel klein. Also hat dieser Wert dann weniger Einfluss auf die Energie und wird weniger stark berücksichtigt, als ein Pixel dessen Farbunterschied zu X_n klein ist.

Bei der Anwendung dieser Faktoren wurde schnell klar, dass ähnlich gefärbte Pixel zu sehr Vorgezogen wurden.

In dem linken Bild in Abb.4 kann man oben rechts ein kleines bisschen Rauschen erkennen, durch schlechte Entfernung der Menschen. Das rechte Bild, ist das Linke nach einer MRF Nachbearbeitung mit den oben vorgeschlagenen Werten. Da der Einfluss von ähnlich farbenen Pixeln überproportional groß ist, wird eine Gruppe von ähnlich aussehenden falsch erkannten Pixeln durch dieses Verfahren nie korrigiert werden können, wenn sie in sich mehr Ähnlichkeit haben als nach außen hin. Wie man in Abb.4 sieht, wachsen diese Regionen stattdessen

Wenn man sich die Faktoren als eine Funktion $f(x) = \frac{1}{x-\mu^2}$ vorstellt, wo μ eine Konstante und $x \geq 0$ ist. Dann nimmt diese Funktion eine Hyperbel Form an. Die Unterschiede zwischen benachbarten Werten werden also kleiner, je größer der Nenner von



Abb.4 Schlechte ICM Werte

$f(x)$ ist. Eine Funktion wie $g(x) = \frac{1}{(x-\mu)^2 + \lambda}$ würde weniger Unterschiede zwischen Werten erzeugen, da der Nenner von Natur aus um λ größer ist als der Nenner von $f(x)$.

Voraussetzung für das Verfahren mit MRF ist natürlich, dass ein WB existiert. Bei der Single Gaussian Methode ist das der Fall, bei den Anderen hier vorgeschlagenen Methoden aber nicht. Der Median und Smoothness Filter selbstverständlich geben keine Aussage darüber zurück, ob ein Pixel Vorder- oder Hintergrund ist. Sie geben lediglich einen Farbwert zurück und man geht davon aus, dass dieser relativ genau dem Hintergrund entspricht. Das MOG Verfahren berechnet zwar Wahrscheinlichkeiten, jedoch nur der Zugehörigkeit zu einzelnen Gauß-Verteilungen. Ob Der Pixel zum Vorder oder Hintergrund gehört, entscheidet sich durch den Rang der jeweiligen Verteilung. Es gibt einige Möglichkeiten trotzdem MRFs für das MOG Verfahren zu verwenden. Die Idee in dieser Arbeit baut auf Teilen der Arbeit von Shu-Jhen Fan Jiang et al. [9] auf. Statt den Wahrscheinlichkeiten werden Labels genommen, welche entweder 1 oder 0 sein können. 1 steht für Vordergrund und 0 für Hintergrund. Es soll für den Wert X_n der dominante Wert seiner Nachbarn X_N genommen werden. Dabei soll der dominante Wert jedoch genauso wie oben in Abhängigkeit des Farbunterschiedes der jeweiligen Pixel gewichtet werden. Das Verfahren hier bleibt also so wie beim Single Gaussian, die einzige Veränderung ist, dass statt einer weichen Zuordnung zu Vorder-/Hintergrund eine harte Zuordnung genommen wird.

Das eben erklärte Verfahren soll in der späteren Arbeit auch einfach als MRF genannt werden.

4.Implementierung

4.1.Einführung

Wie bereits erwähnt, sollen die im Kapitel "Algorithmen" beschriebenen Verfahren zur BI in dieser Arbeit getestet und verglichen werden. Um dies zu tun soll eine Anwendung implementiert werden, welches ein Video als Input nimmt und aus diesem mit einem Algorithmus nach Wahl den Hintergrund extrahiert. Das Programm soll mit der Kommandozeile gesteuert werden und den extrahierten Hintergrund für jeden Frame des Eingangsvideos in einem neuen Video speichern. Im Folgenden sollen die Anforderungen dieser Anwendung beschrieben werden.

4.2.Anforderungen

4.2.1.Funktionale Anforderungen

1.Videoumwandlung

Die Anwendung soll Videos umwandeln können, so dass in dem Ergebnisvideo nur der errechnete Hintergrund zu sehen ist. Was für ein Video umgewandelt wird und wo das Ausgangsvideo gespeichert wird, soll einstellbar sein. Das Ausgabevideo soll im selben Format wie das Eingabevideo gespeichert werden.

2.Auswahl des Algorithmus

Mithilfe der Benutzeroberfläche soll der Algorithmus mit dem die Videoumwandlung geschieht spezifiziert werden. Es sollen alle im Kapitel "Algorithmen" beschriebene Algorithmen zur Verfügung stehen.

3.Bewertung des Algorithmus

Nach der Durchführung der Videoumwandlung, sollen Messdaten zum Umwandlungsprozess ausgegeben werden, um den jeweils benutzten Algorithmus bewerten zu können. Die Messdaten sollen sein:

- 1.Laufzeit der Umwandlung
- 2.Durchschnittliche Bearbeitungszeit pro Frame

4.Hilfefunktion

Um das Steuern der Anwendung zu vereinfachen, soll es möglich sein mit einem "help"-Befehl genauere Informationen zu erhalten.

4.2.2.Nicht Funktionale Anforderungen

1.Moderater Speicherverbrauch

Der genutzte Speicher sollte minimal gehalten werden, ohne den jeweiligen Algorithmus zu verändern oder beeinträchtigen. Es sollten keine redundanzen gespeichert werden.

2.Minimale Rechenzeit

Die Rechenzeit sollte möglichst klein gehalten werden, es sollten also so wenig Berechnungen wie möglich durchgeführt werden. Diese Berechnungen sollten eine möglichst geringe Komplexität haben. Außerdem sollten Berechnungen nicht doppelt ausgeführt werden.

4.3.Struktur

Der Programmcode ist im Wesentlichen in drei Teile geteilt. Das En-/Decoding, die Algorithmus-Implementierungen und die Benutzeroberfläche. Diese Teile werden im Folgenden näher beleuchtet.

4.3.1.En/Decoding

Der Teil des En-/Decoding wandelt das Eingangsvideo in eine bearbeitbare Form um und erstellt und speichert nach der Bearbeitung das Ausgangsvideo. Dieser Vorgang lässt sich in die zwei offensichtlichen Teile teilen: Decoding und Encoding.

4.3.1.1.Decoding

In dieser Arbeit werden nur Algorithmen behandelt, welche nicht bereits das gesamte Video kennen. Die Algorithmen werden zwar mit vollständigen Videodateien getestet, sollen sich aber ebenfalls auf etwas wie einen Livestream anwenden lassen. Um dieser Idee zu folgen, sollen mit Hilfe der speziellen Implementierung lediglich einzelne Frames nacheinander bearbeitet werden. Außer den Frames dürfen nur vorhergehende Frames zu Hilfe genommen werden. Dies hat einen Fluss von Bilddaten zur Folge, mit dem die Implementierungen der Algorithmen arbeiten können

Das Decoding ist dafür zuständig, die Frames aus den Videos zu extrahieren und

zurückzugeben. Dies soll in einer Weise passieren, dass den Algorithmen die letzten n Frames bekannt sind, jedoch keine danach kommenden.

Wie bereits erwähnt, wurde OpenCV als Codec benutzt in dieser Arbeit. Als Decoder wurde die Klasse `OpenCVFrameGrabber` verwendet, welche es ermöglicht die Frames eines Videos nacheinander zu decodieren und in der Form von Frame-Objekten zu erhalten.

Die Initialisierung des `OpenCVFrameGrabber` erfolgt durch den Aufruf des Konstruktors und der Methode `start()`. Dem Konstruktor muss dabei lediglich die zu decodierende Videodatei übergeben werden. Durch die Methode `grab()` erhält man den nächsten Frame eines Videostreams des übergebenen Videos.

Für die Klasse `OpenCVFrameGrabber` wird eine Wrapperklasse namens `FrameGetter` geschrieben, dessen Methoden in Diag.4 zu sehen sind.

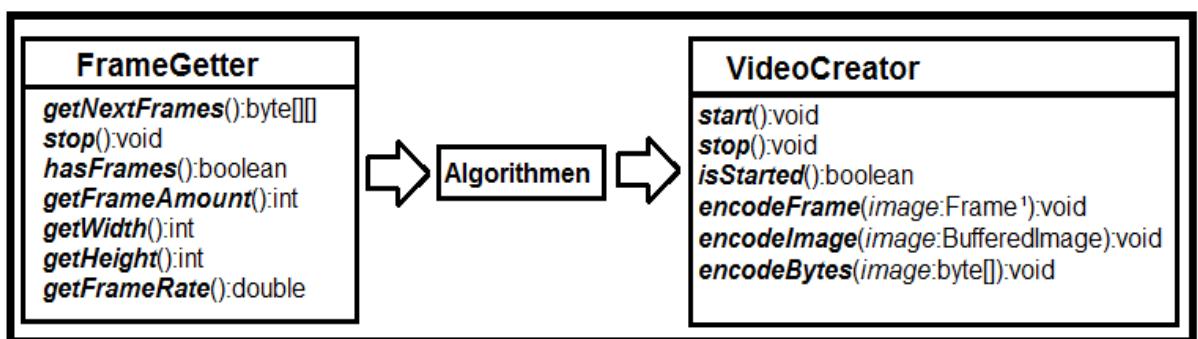
Die Methode `getNextFrames()` gibt dabei die letzten n Frames zurück, wobei n im Konstruktor festgelegt wurde. Die Rückgabe mehrerer Frames erleichtert Algorithmen, welche vergangene Frames benötigen. Für die Algorithmen GMM und MOG gilt $n=1$.

Eine naheliegende Herangehensweise an das Holen der Frames, wäre erst das gesamte Video zu decodieren um beim Bearbeiten des nächsten Frames nicht jedes mal auf den Decoder warten zu müssen. Dieser Vorschlag ist jedoch für größere Videos unrealistisch, da der verbrauchte Speicher stark in die Höhe schnellen würde.

Bei einer Framerate von $30 \frac{\text{Frames}}{\text{Sekunde}}$, einer Bildgröße von 800×600 und einer Farbtiefe

von $24 \frac{\text{Bits}}{\text{Pixel}}$ würde nach 6 Sekunden bereits mehr als 2GB an Speicher verbraucht.

Der Wrapper `FrameGetter` löst dies damit, dass die Methode `getNextFrames()` asynchron funktioniert. Dies simuliert etwas wie einen Stream von Inputdaten, mit denen die Algorithmen arbeiten können. So müssen lediglich IO Wartezeiten statt zusätzlichen Decoding Wartezeiten in Kauf genommen werden.



Diag.5 FrameGetter und VideoCreator

4.3.1.2.Encoding

Analog zum Decoding, sollen ebenfalls einzelne Frames nacheinander codiert werden um der Implementierung der Algorithmen einen Livestream für den Input sowie für den Output zu geben. Nach der Initialisierung des Encoders, sollen alle Frames des Videos einzeln übergeben werden.

Analog zum Decoding, wird hier ebenfalls eine OpenCV Klasse benutzt. Die Klasse `FFmpegFrameRecorder` braucht zur Initialisierung die Datei des Videos, sowie seine Breite, Höhe, Framerate und sein Dateiformat. Zu der Initialisierung gehört hier ebenfalls das Aufrufen der `start()` Methode. Mit der Methode `record(image:Frame)` kann man dem Video solange Bilder beifügen, bis man die Erstellung des Videos mit der `stop()` Methode beendet. Die Bilder müssen dabei zu der Klasse Frame gehören.

Für diese Klasse wurde ebenfalls ein Wrapper geschrieben. Dieser Wrapper heißt `VideoCreator`(siehe Diag.4). Genauso wie beim Decoding-Prozess des `FrameGetters`, funktioniert das Encoding hier asynchron. So werden ebenfalls jegliche lange Encoding Wartezeiten der Algorithmen, in kürzere IO Wartezeiten umgewandelt.

4.3.2.Algorithmen

Dieser Teil übernimmt, wie der Name schon sagt, die Realisierung der hier genannten Algorithmen. Dieser Teil ist also für das Verarbeiten der decodierten Videodaten zuständig, sowie für die Messung der Verarbeitungen.

Der Übersicht und Ordnung halber, wurde jeder Algorithmus in einer eigenen Klasse implementiert. Dafür wurde hierbei Codedopplung in Kauf genommen. Die Algorithmusklassen sind:

- 1.**MedianAnalyzer**: Implementiert den Medianfilter
- 2.**SmoothingAnalyzer**: Implementiert den Smoothingfilter
- 3.**GMMAnalyzer**: Implementiert ein Single Gaussian bzw. GMM
- 4.**MOGAnalyzer**: Implementiert eine MOG
- 5.**GMMAnalyzerMRF**: Implementiert ein GMM mit anschließendem MRF
- 6.**MOGAnalyzerMRF**: Implementiert eine MOG mit anschließendem MRF
- 7.**MRFOptimizer**: Implementiert das MRF bzw. den ICM Algorithmus

Die letzten wichtigen Klassen sind der **AlgorithmController**, der wie der Name schon sagt als Controller der Algorithmen und als Anbindung an das UI angesehen werden kann sowie der **Measurer**, der Messdaten zu den einzelnen Verfahren aufzeichnet.

4.3.3. User Interface

Das UI ist in dieser Arbeit eher nebensächlich. Dennoch muss es implementiert werden. Die Nutzerinteraktion soll auf der Kommandozeile mithilfe von Befehlen stattfinden. Um dem Nutzer die Bedienung ein wenig zu erleichtern, soll es zu den einzelnen Befehlen auf Anfrage nähere Informationen, sowie eine Befehlsliste geben.

4.4. Details

4.4.1. Wahl der Programmiersprache

Die meisten Implementierungen von Algorithmen der Videoverarbeitung sind in der Programmiersprache C++ geschrieben. Es existiert viel Literatur dazu. Es gibt viele Bibliotheken und durch C++ spezifische Optimierungen, werden die Algorithmen schneller. Die Wahl der Programmiersprache in dieser Arbeit fällt jedoch auf Java. Der Hauptgrund ist, dass der Autor sich mit dieser Sprache am besten auskennt. Da das Hauptziel dieser Arbeit ist Algorithmen zu vergleichen, ist die relative Geschwindigkeit wichtiger als die absolute Geschwindigkeit dieser Algorithmen. Das heißt, dass wenn Algorithmus A in einer C++ Implementierung 5 mal so schnell ist wie Algorithmus B , dann wird das in einer Java Implementierung genauso bleiben, auch wenn beide Algorithmen doppelt so viel Zeit in Anspruch nehmen wie vorher.

Da sich das Endprodukt der Algorithmen ebenfalls nicht verändern wird, ist für diese Anwendung die Programmiersprache Java vollkommen ausreichend.

4.4.2.Bild Daten

Nachdem die Videodaten decodiert wurden, werden sie von OpenCV zunächst als Frames zurückgegeben. Da diese Frameklasse von OpenCV kommt, wird eine ebenfalls von OpenCV kommende Klasse namens `Java2DFrameConverter` genutzt um aus den Frames Java spezifische `BufferedImages` zu machen.

Der nächste Schritt wäre, aus den `BufferedImages` die Rohdaten der Bilder zu gewinnen. Hierbei gibt es in Java zwei oft genutzte Möglichkeiten. Die Eine ist die Methode `getRGB()` von `BufferedImage`, mit der man für jeden Pixel einzeln den Rot, Grün und Blau Wert erhält. Die Andere ist die Daten aus dem `BufferedImage` als Byte Array direkt zu kopieren.

Ein wesentlicher Unterschied dieser beiden Methoden ist der Rückgabewert. Bei der ersten Methode erhält man ein Array aus Integern, die bereits die Farbwerte und einen Alphawert kombiniert enthalten. Bei der zweiten Methode erhält man ein Array aus Bytes, wobei jeder Farbwert unabhängig von den Anderen gespeichert ist.

Ein zweiter wesentlicher Unterschied ist die Geschwindigkeit des Ladens und Speicherns eines Bildes. Nach 100 Tests an drei verschiedenen großen Bildern ergaben sich die Konvertier-Geschwindigkeiten in Diag.5. Wobei Variante 1 die Methode `getRGB()` benutzt und Variante 2 die Rohdaten direkt kopiert.

Die Messungen geben die Rechenzeit der beiden Varianten in $\frac{\text{Nanosekunden}}{\text{Pixel}}$ an. Man sieht, dass Variante 2 wesentlich schneller ist als Variante 1. Aus diesem Grund wurde sie benutzt werden.

Größe	Variante 1	Variante 2
4760x3605	44.2235159372487	0.00100465681418198
1190x902	46.0872679386611	0.0107751681603905
216x120	47.1519413580247	0.281760030864198

Tab.1 `BufferedImage` zu Rohdaten Konvertierungs-Geschwindigkeiten

4.4.3.PixelNachbarschaft

Da wie eben erklärt, die Bilddaten als Byte Array gespeichert werden, mit R, G und B Werten einzeln, ist es wichtig zu erläutern wie die Nachbarschaft eines Wertes aussehen wird. Es würde sicherlich keinen Sinn machen für einen Wert im Array beim Index 4, die Werte mit den Indexen 3 und 5 als Nachbarn zu zählen. In diesem Fall würden alle Werte verschiedene Farben repräsentieren. Es macht mehr Sinn die Nachbarn zu wählen, wie in Abb.5 illustriert wurde. Dabei gehören alle Nachbarn zu der selben Farbe. Statt 3 und 5 wären also 1 und 6 mögliche Nachbarn für den Index 4.

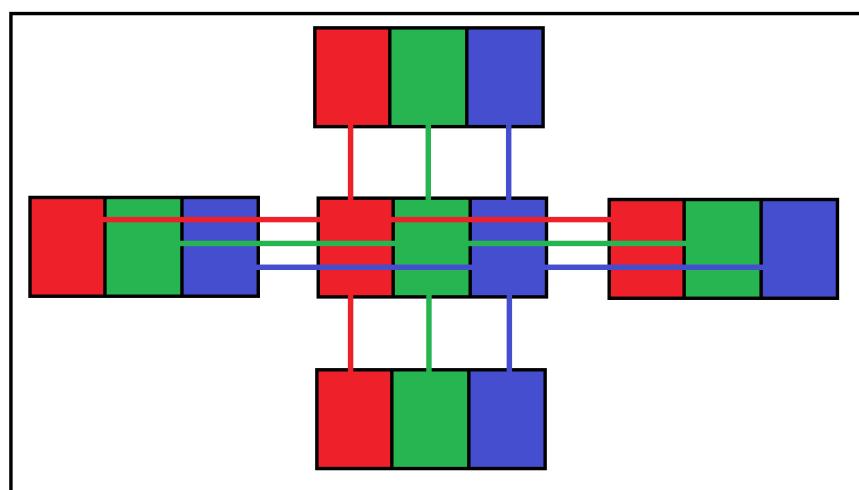


Abb.5 Pixelnachbarschaften visualisiert

4.5.Einstellen der Parameter

Bei den einzelnen vorgestellten Verfahren gibt es verschiedene Werte, welche eingestellt werden müssen. Im Folgenden soll ermittelt werden, welche Werte hier jeweils am besten sind.

4.5.1.Median Filter

Der Wert der sich beim Median Filter verändern lässt ist die Anzahl der Frames n , von welchen der Median genommen wird. Es soll zwischen den Werten $n=10, 30, 50, 100$ entschieden werden.

In Abb.6 sieht man Ausschnitte eines Videos und die dazugehörigen nachbearbeiteten Medianbilder mit verschiedenen großen n . Auf den Bildern lässt sich erkennen, dass die Verfahren mit weniger als 50 Frames viele Objekte nur halb entfernt bekommen. Bei 50 Frames, ist lediglich ein Fehler im Ausschnitt von 0:40. Bei 1:20 bleibt eine Person stehen, was auch überall erkannt wurde.

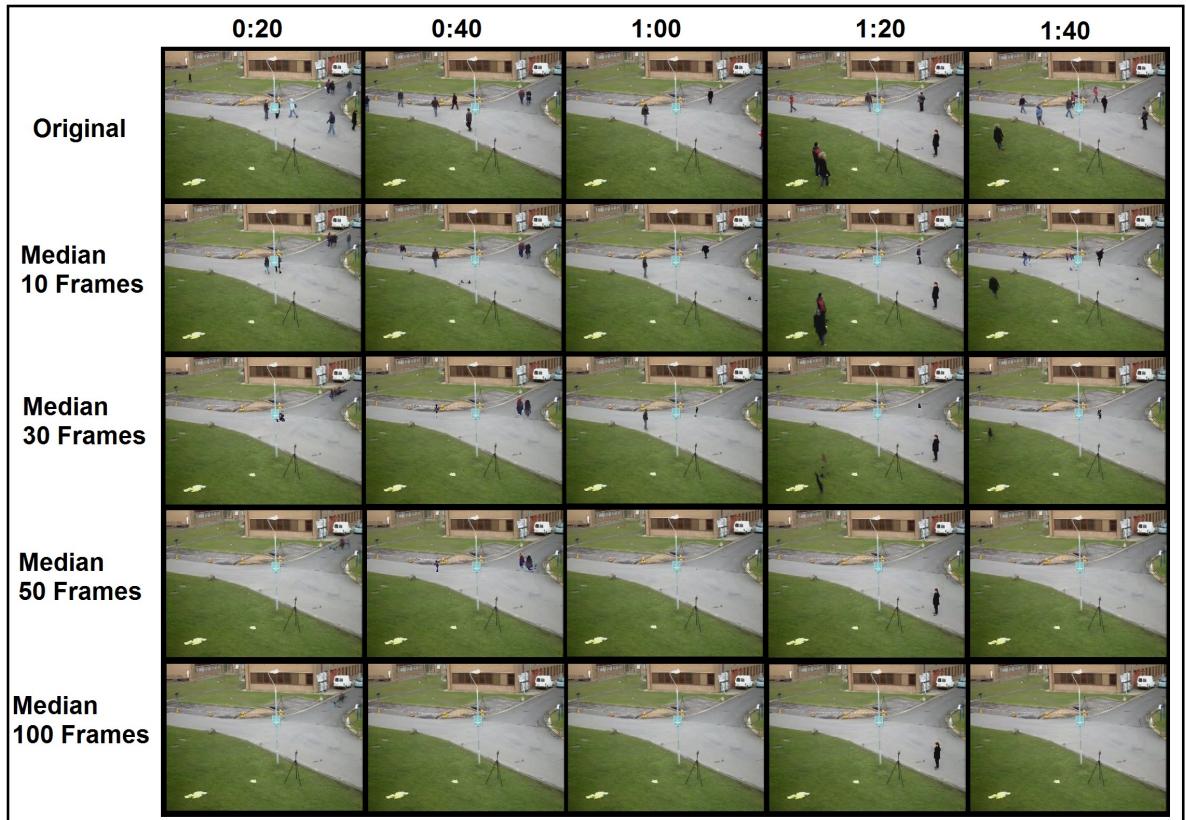


Abb.6 Median Vergleich für n

10 Frames	30 Frames	50 Frames	100 Frames
15 ms	57 ms	124 ms	151 ms

Tab.2 Median Rechenzeit Vergleich

In Tab.2 sind die mittleren Rechenzeiten für die verschiedenen großen n aufgelistet.

Es lässt sich erkennen, dass natürlich die gebrauchte Zeit und die Größe des n proportional zueinander sind. Obwohl bei $n=10$ und $n=30$ die Zeit daher am besten war, schneiden diese beiden Werte in der Qualität zu schlecht ab. $n=100$ schneidet in der Qualität besser ab als $n=50$, braucht nur etwa 1,2 mal so lange zum Rechnen, verbraucht aber aufgrund der doppelten Anzahl Frames auch etwa den doppelten Speicher. Daher wird in dieser Arbeit ab sofort für den Median Filter $n=50$ verwendet.

4.5.2.Smoothing Filter

Beim Smoothing lässt sich ebenfalls wie beim Medianfilter lediglich die Anzahl der Frames n zwischen denen der Mittelwert gebildet wird einstellen. Es soll zwischen den selben Werten wie beim Medianfilter entschieden werden.

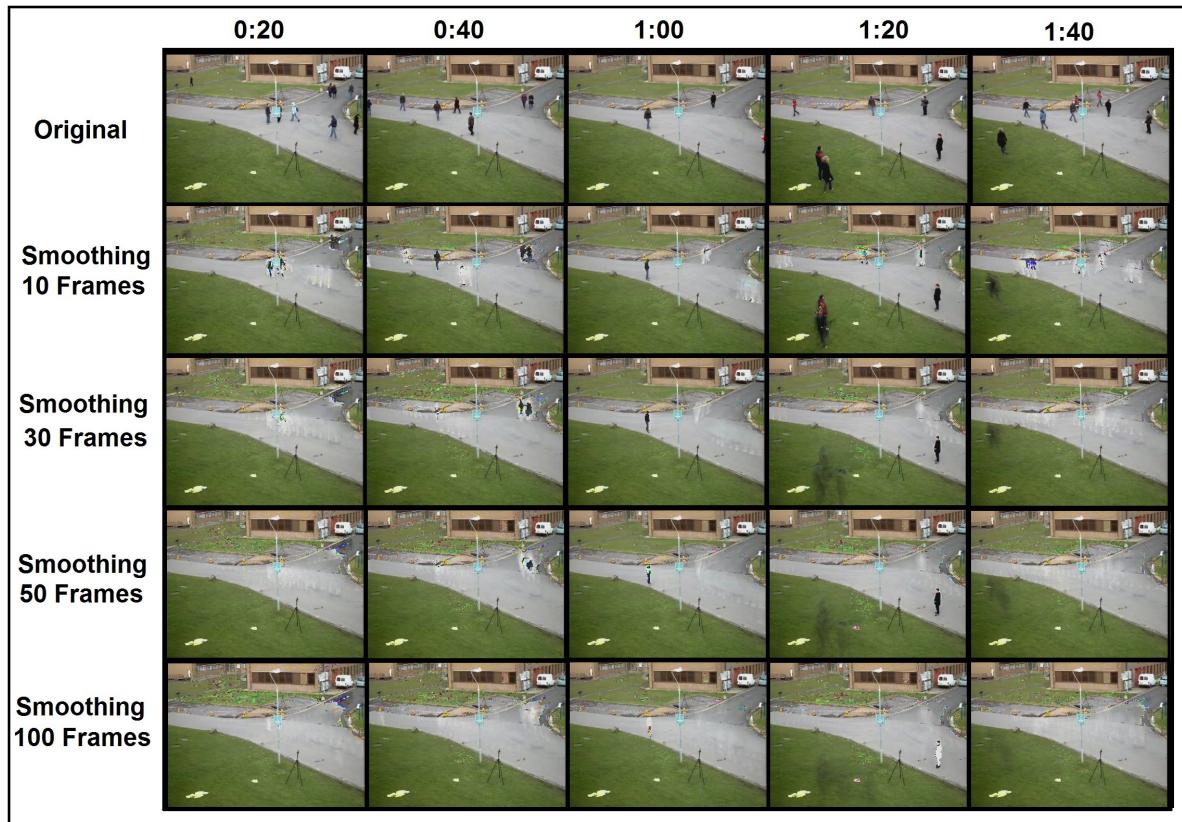


Abb.7 Smoothing Vergleich für n

Die Messungen hier vielen ähnlich wie beim Medianfilter aus. Die Rechenzeiten gehen hoch je mehr Frames zur Berechnung herangezogen werden. Hier ebenfalls liefern $n=50$ und $n=100$ akzeptable Ergebnisse, der Unterschied zwischen $n=50$ und $n=30$ ist jedoch nicht ganz so groß. Deshalb wird in dieser Arbeit für das Smoothing Verfahren $n=30$ gewählt.

4.5.3.GMM

Beim GMM ist der veränderliche Wert die Lernrate α , welche den Einfluss von neuen Farben einstellt. Getestet sollen die Werte $\alpha=0.1, 0.01, 0.003, 0.001$. Jeder Wert darunter würde die Änderung eindeutig zu langsam machen. Werte darüber liegen bereits bei 1. Eine Lernrate von 1 würde 100% Veränderung pro Frame bedeuten, was das Originalvideo zurückgeben würde.

Unten links in Abb.7 wird alles sehr dunkel. Dies liegt daran, dass dieses GMM mit $\mu=0$ beginnt und sich farblich dann erst an das Video anpasst. Man kann also sehen, dass bei der Auswahl des α ein genaues Ergebnisvideo und eine schnelle Anpassung an Veränderungen in einem Kompromiss gegenüberstehen. Da alle GMM den gleichen Speicher- und

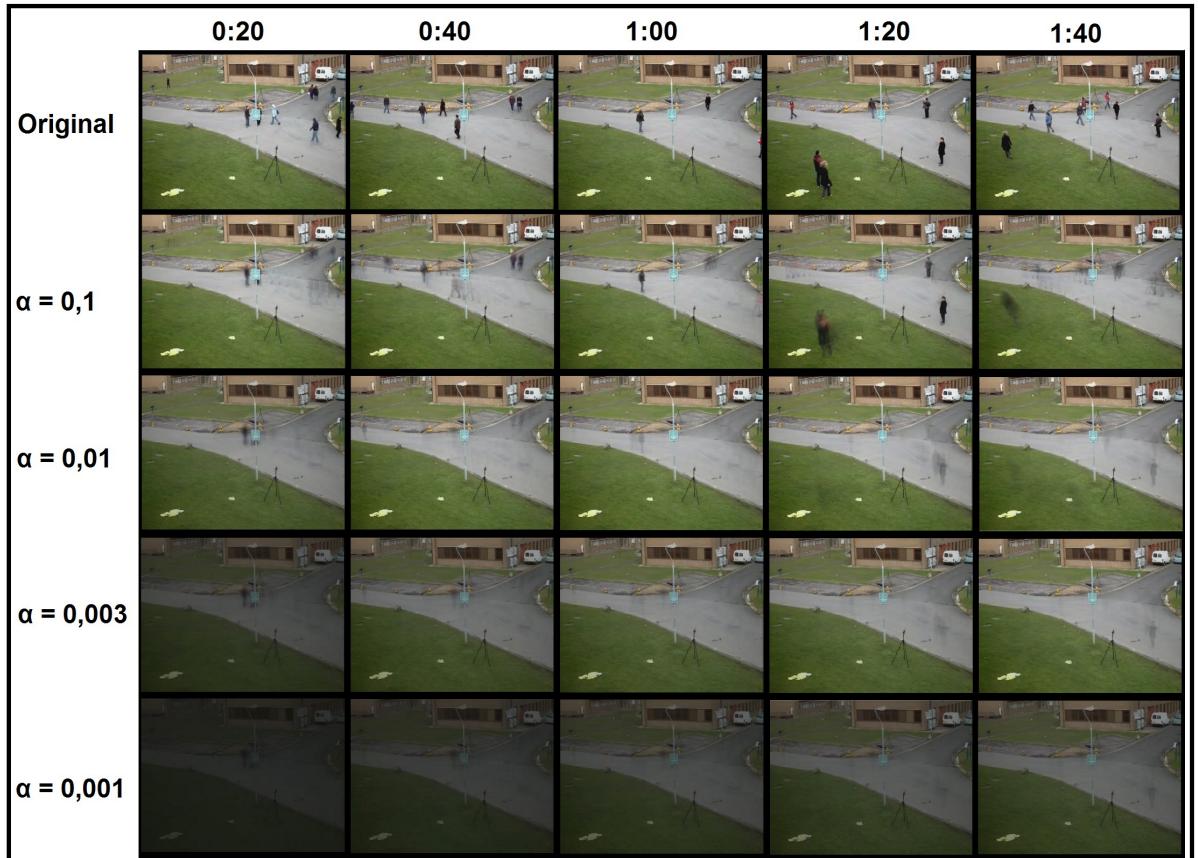


Abb.8 GMM Vergleich für die Lernrate

Rechenbedarf haben, kommt es bei der Wahl des α lediglich auf die Qualität des Ergebnisvideos an. $\alpha=0.003$ und $\alpha=0.01$ geben hier sicherlich den besten Kompromiss. Sie brauchen nicht zu lange um sich an das Video anzupassen und liefern ein vergleichsweise gutes Ergebnis. Die Anpassung bei $\alpha=0.003$ dauert aber immer noch

$\frac{1}{0.003} \rightarrow 333$ Frames, was in diesem Video mehr als 40 Sekunden bedeutet. Das ist zu lang und bringt die Wahl auf $\alpha=0.01$. Neben dem Wert für α gibt es noch den theoretisch einstellbaren Wert ϕ , welcher den Maximalen Abstand eines Wertes vom Mittelwert in Zusammenhang mit der Varianz beschreibt. Wie schon erwähnt, ist in dieser Arbeit immer $\phi=2.5$. Diese Wahl des Wertes wurde empirisch begründet von Stauffer und Grimson in [7].

4.5.4.MOG

Ebenfalls hier wird wie bei GMM $\phi=2.5$ gesetzt. Die anderen Werte, welche bei MOG festgelegt werden müssen sind. Die Lernrate α , das Anfangsgewicht einer neuen Gauß-Verteilung ω_{Anfang} , die Anfangsvarianz der Verteilung σ_{Anfang} , der Anteil des Hintergrundes am Video ρ und die Anzahl der Gauß Verteilungen des MOG.

Die Anzahl der Verteilungen sollte nach [3] bei 3 - 5 liegen. In dieser Arbeit wird 5 verwendet.

Nach einigen Versuchen, hat sich gezeigt, dass sich Unterschiede bei dem Wert ω_{Anfang} bei ausreichend geringer Größe, nicht im Video widerspiegeln.

Deswegen wurde $\omega_{Anfang}=0.001$ gewählt, da nach [3] der Wert möglichst klein gewählt werden sollte, aber $\omega_{Anfang}<0$ gelten muss.

Für σ_{Anfang} haben sich ebenfalls keine Unterschiede gezeigt, diesmal jedoch bei ausreichend hoher Größe. Da die möglichen Werte der Varianz $\{0^2, 1^2, 2^2, \dots, 255^2\}$ sind, wurde ein höherer Wert $\sigma_{Anfang}=300^2$ gewählt, um in jedem Fall vergleichsweise hoch zu sein.

Der Wert für ρ soll zunächst bestimmt werden. Bei diesem Wert muss ein Kompromiss eingegangen werden. Je höher ρ gewählt wird, desto mehr falsch erkannter Vordergrund entsteht im Ausgangsvideo und desto besser wird bewegter Hintergrund wiedergegeben. Da in dem für Vergleiche bisher genutzten Video kein bewegter Hintergrund existiert, wird hier ein anderes Video zum Vergleich hergenommen.

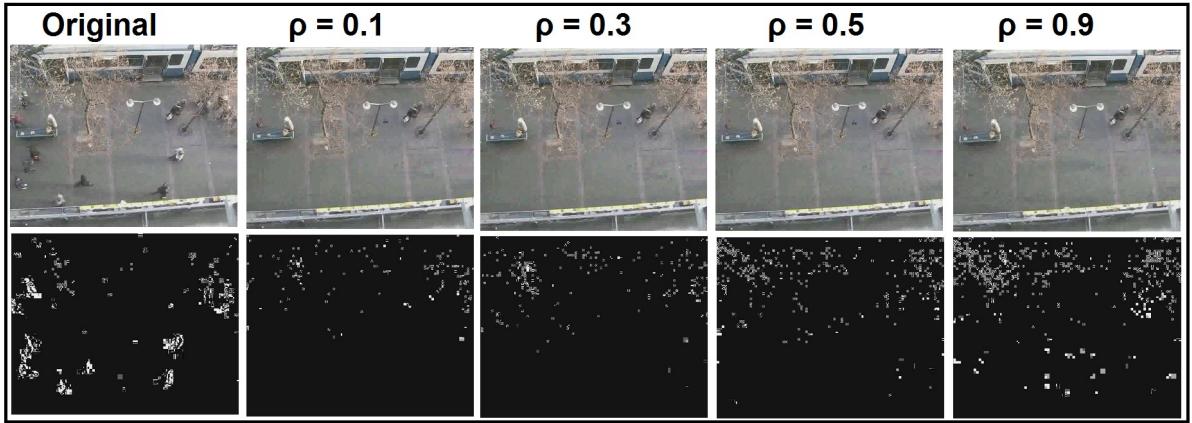


Abb.9 MOG Vergleiche für den Hintergrundanteil

In Abb.8 sieht man einen Frame dieses Videos nachbearbeitet mit $\rho=0.1,0.3,0.5,0.9$. Darunter sieht man das Differenzbild, dieses Frames mit seinem Vorgänger im jeweiligen Ausgangsvideo. Der Sichtbarkeit halber wurde das Differenzbild binarisiert mit einem Threshold von 3.

In dem Differenzbild des Originals erkennt man Bewegungen, welche nirgendwo anders sichtbar sind. Dies sind die Menschen, welche über den Bahnsteig spazieren. Die Bewegung die überall zu erkennen ist, ist die bewegte Hintergrundtextur. Wie man erkennen kann, werden die Bewegungen bei einem höheren Wert für ρ detaillierter. Bei $\rho=0.9$ werden jedoch noch mehr Bewegungen sichtbar. Diese Bewegungen sind Ghosts(siehe Grundlagen). Da bei $\rho=0.5$ lediglich wenige Ansätze von Ghosts zu sehen sind, die Textur jedoch ihre Bewegung sehr detailliert behält, wird dieser Wert als bester Kompromiss behalten.

Der Wert für α soll genauso wie beim GMM bestimmt werden.

Auf Abb.8 erkennt man auf dem Frame zu der Zeit 0:20 ab einem $\alpha=0.003$ eine bessere Zuordnung des Hintergrundes. Sonst kann überall leider kaum ein Unterschied erkannt werden, weshalb es bei $\alpha=0.003$ bleiben wird.

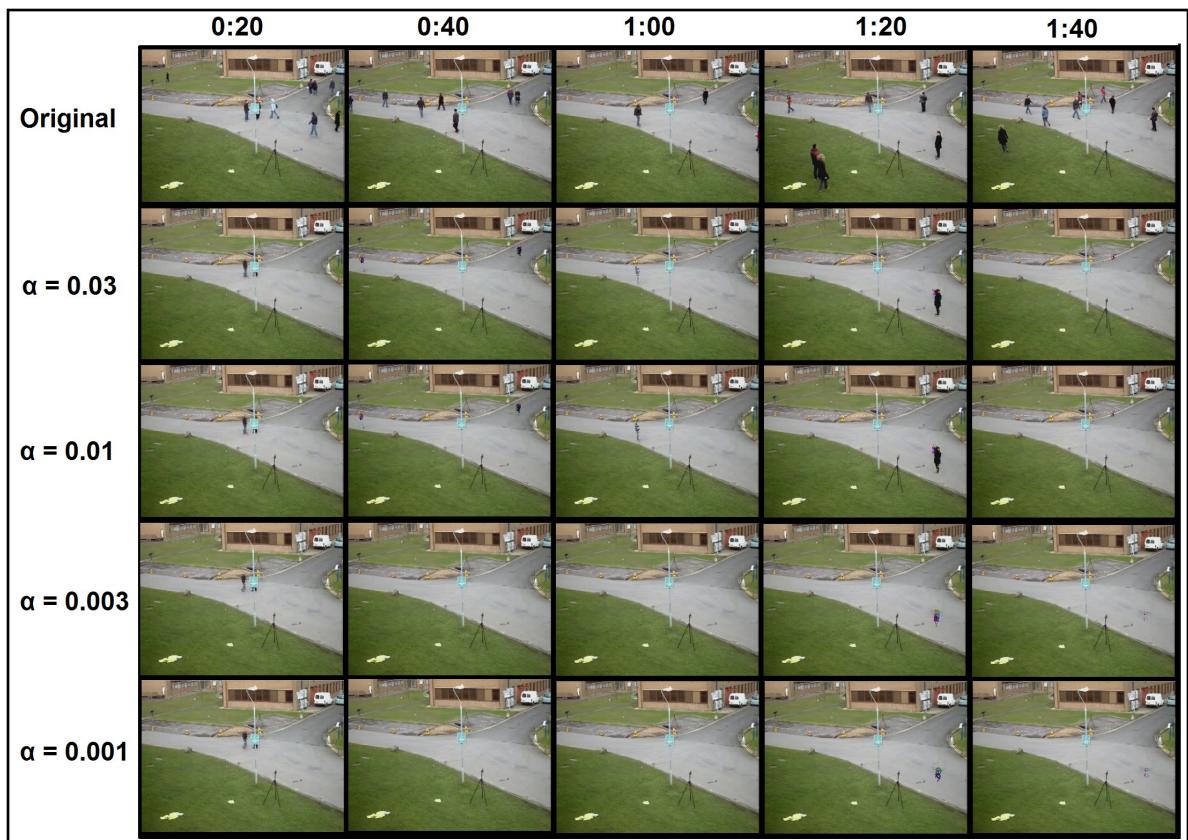
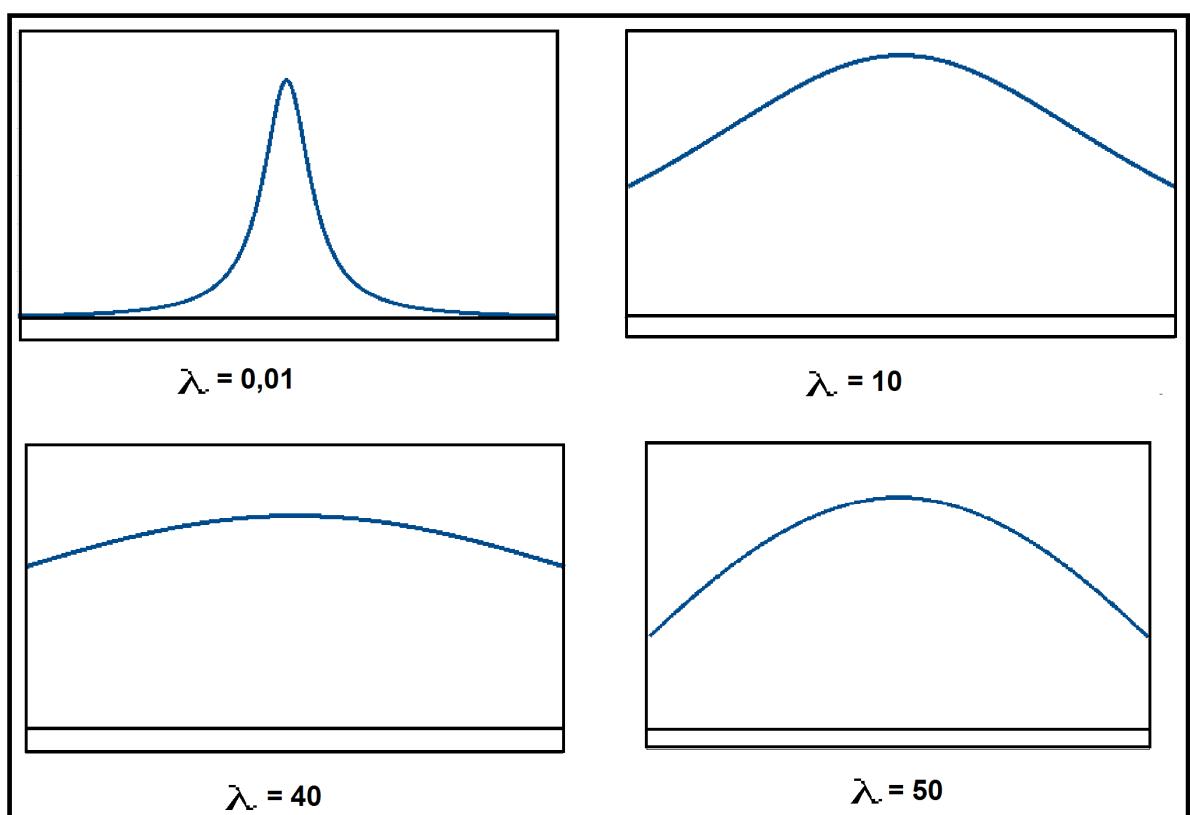


Abb.10 MOG Vergleich für die Lernrate



Diag.6 MRF Vergleich für Lambda

4.5.5.MRF

Hier muss der Wert des λ bestimmt werden, sowie die Anzahl der Iterationen über das Wahrscheinlichkeitsbild. Die Anzahl der Iterationen bleibt in dieser Arbeit immer 6. Die Wahl dieser Zahl kommt von den Forschungsergebnissen von Julian Besag [6]. Der Wert des λ soll simpler bestimmt werden als alle Werte vorher. λ war ein Wert, welcher die Gewichte der Nachbarn X_N eines Pixels X_n bei der Neuberechnung von X_n mitbestimmt. Die Berechnung erfolgte durch die Funktion

$$g(x) = \frac{1}{(x - \mu)^2 + \lambda}$$

wobei x die Intensität eines Mitglieds von X_N und μ der Mittelwert der Intensitäten von X_N war. Die Funktion sieht für verschiedene λ verschieden Flach aus (siehe Diag.5). Da die Unterschiede zwischen ähnlichen Werten möglichst gleich bleiben sollen, ist $\lambda=40$ die beste Wahl.

5. Vergleich

5.1. Wahl der Videos

Im Folgenden sollen die Algorithmen miteinander verglichen werden. Zunächst werden die dafür verwendeten Videos vorgestellt.

1. "Wandering students"-BCM 320x240

Ausschnitte aus diesem Video wurden bereits in dieser Arbeit verwendet. In dem Video sieht man einige Studenten über einen Hof laufen. Der Hintergrund ist hier weitgehend unbewegt und der Vordergrund hat eine relativ geringe Fläche, was dieses Video einfach zu bearbeiten macht. Es treten allerdings Luminanzschwankungen auf.

2. "Hotel Sequence"-EWAP 720x576

Dieses Video wurde ebenfalls bereits benutzt bei Vergleichen. Es ist ein Video des EWAP Walking Pedestrians Datasets der ETH Zürich. In dem Video sieht man Fußgänger an einer Tramstation. Die Besonderheit hier ist, dass sich der Hintergrund in dem Video auf zwei Weisen verändert. In der oberen Hälfte des Videos bewegen sich Bäume leicht im Wind und während des Videos trifft eine Tram ein und bleibt stehen. Das Original des Videos ist etwa 13 Minuten lang. In dieser Arbeit werden nur die ersten 23 Sekunden zum Testen verwendet.

3. "HighwayI"-VISOR 320x240

Dieses Video wurde im VISOR (Video Surveillance Online Repository) unter der Kategorie "Shadows" zur Verfügung gestellt. Wie diese Kategorie schon andeutet, ist das besondere an diesem Video, die große Menge an Schatten

4. "Intelligent Room"-VISOR 320x240

Dieses Video findet man ebenfalls unter der Kategorie "Shadows" auf der VISOR Homepage. Dieses Video ist das einzige der Verwendeten, welches in einem Gebäude aufgenommen wurde, weshalb es auch das Einzige ist, bei dem keine Luminanzschwankungen im Hintergrund auftreten.

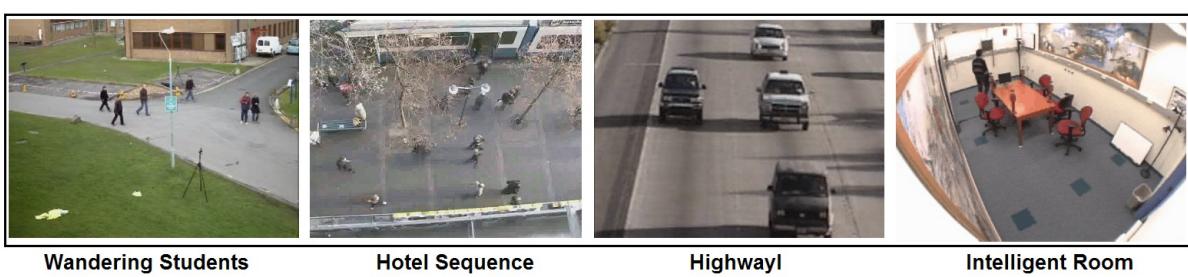


Abb.11 Testvideos

5.2.Speicheraufwand

Wenn man die Verfahren genauer betrachtet, fallen schnell die Unterschiede in Sachen Speicherverbrauch auf. Der Median und Smoothing Filter müssen jeweils n Frames speichern um zu funktionieren. In dieser Arbeit speichert der Median Filter 50 Frames und der Smoothing Filter 30 Frames(siehe Einstellen der Paramter).

Das GMM speichert die Mittelwerte und Varianzen der einzelnen Pixel. In dieser Arbeit werden diese als float-arrays gespeichert. Da ein float in Java eine Größe von 4byte hat und die Frames in dieser Arbeit als byte-Arrays gespeichert werden, verbrauchen die Mittelwerte und Varianzen zusammen genau soviel Speicher wie 8 Frames.

Der MOG speichert in dieser Arbeit 5 einzelne Gauß-Verteilungen mit jeweils einem Mittelwert, einer Varianz und einem Gewicht. Das sind 1,5 mal so viele Werte pro Gauß-Verteilung wie beim GMM. Insgesamt braucht der MOG also $1,5*5=7,5$ mal so viel Speicher wie das GMM. In dieser Arbeit entspricht das dem Speicher von 60 Frames.

Beim Postprocessing mit MRFs tritt kein weiterer Speicherverbrauch auf.

In Tab.3 sind die Verfahren mit ihrem Speicherverbrauch aufgelistet. Der Speicherverbrauch wird dabei an der Größe g gemessen, welche dem Speicherverbrauch eines Frames entspricht. Die Verfahren sind in der Tabelle bereits geordnet.

GMM	8 g
Smoothing	30 g
Median	50 g
MOG	60 g

Tab.3 Speicherverbrauch der Algorithmen

5.3.Geschwindigkeitsvergleich

Die Videos wurden einzeln von allen Verfahren verarbeitet. Die Ausgabevideos wurden gespeichert und die jeweiligen Rechenzeiten gemessen. In Tab.4 sieht man die Rechenzeiten

der einzelnen Verfahren in $\frac{\text{Millisekunden}}{\text{Frame}}$.

Die Tabelle ist bereits der Geschwindigkeit nach sortiert. Überraschend fällt dabei auf, wie viel schneller GMM ist als die anderen Verfahren. Ebenfalls überraschend ist, wie viel langsamer der Median Filter ist.

	Wandering Student	Hotel Sequence	HighwayI	Intelligent Room
GMM	2	10	2	3
Smoothing	3	19	4	3
MOG	29	163	29	30
GMM-MRF	55	280	52	53
MOG-MRF	66	366	65	62
Median	260	1140	205	185

Tab.4 Geschwindigkeitsvergleich

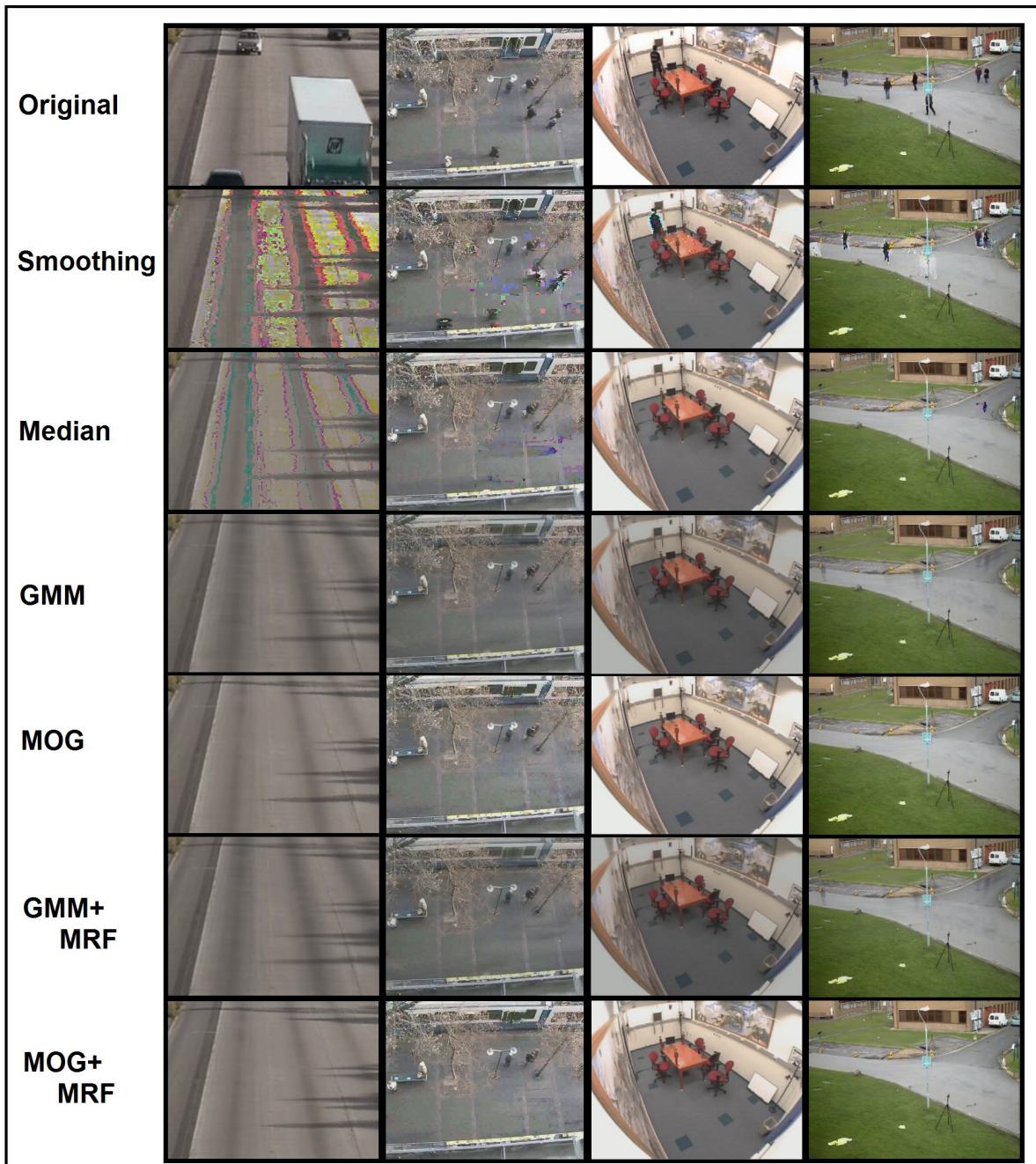


Abb.12 Qualitätsvergleich

5.4.Qualitätsvergleich

In Abb.10 kann man einen groben Überblick über die einzelnen Methoden und ihre qualitative Wirksamkeit bekommen. Im Folgenden sollen die einzelnen Verfahren evaluiert werden:

5.4.1.Smoothing

Hier erkennt man sehr schnell, dass man dies nicht für chromatische Bilder verwenden kann. Auf jedem Bild erkennt man verrauschte Farben. Auf dem ersten Bild wurden die bewegten Objekte erfolgreich entfernt, auf den anderen nicht. Dies ist darauf zurückzuführen, dass sich die Objekte hier schneller bewegt haben. Man könnte daraus schließen, dass man mit einer größeren Anzahl gepufferten Frames auch langsamere Objekte entfernen könnte. Wie wir aber bei der Einstellung der Werte festgestellt haben, führt dies zu aller erst zu einem sich immer länger ziehenden Schweif.

5.4.2.Median

Das Median Verfahren ist erstaunlich robust. Gelegentlich sieht man wie beim Smoothing verrauschte Farben, jedoch werden die Objekte immer weitgehend entfernt. Bewegen sich die Objekte allerdings zu langsam, schafft es dieses Verfahren nicht, sie ganz zu entfernen. Ein Beispiel dafür erkennt man im oberen rechten Teil des Ausschnittes aus dem "Wandering Students"-Video.

Bleibt ein Objekt hier stehen, wird es von innen nach außen zu einem Teil des Hintergrundes. Bewegt es sich wieder, wird es ebenso von außen nach innen abgebaut.

5.4.2.GMM

Dieses Verfahren schafft es alle Objekte zu entfernen und hinterlässt als Erstes keine rauschenden Farben. Allerdings verwandelt sich jedes Objekt in ein Ghost (siehe Grundlagen). Je langsamer sich etwas bewegt, desto deutlicher erkennt man es als Ghost wieder. Dies ist jedoch ein wenig gravierender Nachteil, da man trotz Ghosts immer den Hintergrund erkennen kann und die Ghosts sehr wenig sichtbar sind.

Wenn ein Objekt hier stehen bleibt, verliert dessen Ghost stetig an Transparenz bis es undurchsichtig und somit ein Teil des Hintergrundes geworden ist. Bewegt es sich wieder, wird dieser Prozess umgedreht und es wird immer transparenter.

5.4.3.MOG

Dieses Verfahren schafft es ebenfalls alle Objekte zu entfernen. Es hinterlässt dabei keine Ghosts. Leider sieht man immer wieder vereinzelte Pixel, deren Farbe verrauscht wurde. Dies ist darauf zurückzuführen, dass die R, G und B Werte der Pixel unabhängig voneinander behandelt werden. Wenn ein B Wert dem Hintergrund zugeschrieben wird, kann der R Wert des selben Pixels gleichzeitig dem Vordergrund zugeschrieben werden.

Wenn ein Objekt hier stehen bleibt, fangen an der Stelle wo es steht die Farben an stärker zu rauschen und mit der Zeit zu dem Objekt zu werden. Bewegt es sich wieder, passiert dies rückwärts.

5.4.4.GMM+MRF

Im Vergleich zum GMM lässt sich hier leider kein Unterschied erkennen.

5.4.5.MOG+MRF

Im Gegensatz zum GMM+MRF lässt sich hier zum MOG ein deutlicher Unterschied erkennen. Das farbliche Rauschen tritt nach diesem Verfahren weniger auf. Einige Ausläufer des Rauschens bleiben zwar übrig, der Großteil wurde jedoch entfernt. In Abb.11 lässt sich dies anhand eines Ausschnittes des "Wandering Students" Videos erkennen.



Abb.13 MOG vs. MOG+MRF

5.5.Schlussfolgerungen

5.5.1.Smoothing

Dieses Verfahren ist leider für farbige Videos nicht zu gebrauchen. Selbst bei achromatischen Videos kommt es mindestens auf eine hohe Geschwindigkeit der Objekte an bzw. eine hohe Zahl an gespeicherten Frames.

5.5.2.Median

Dieses Verfahren erzielt bei Videos ohne viel Luminanzveränderung keine schlechten Ergebnisse. Die Rechenzeit und der Speicherverbrauch machen es jedoch in der Praxis unbrauchbar.

5.5.3.GMM

Dieses Verfahren kann besonders mit seiner Geschwindigkeit und dem geringen Speicherverbrauch punkten. Die Ergebnisse sind vergleichsweise gut. Es treten zwar Fehler auf, da diese jedoch kein Rauschen sind, sind sie angenehmer für das Auge und stören somit weniger. Durch die extreme Schnelligkeit ist es denkbar einen ähnlichen Algorithmus für mobile Anwendungen zu implementieren. Das automatische Entfernen von Touristen auf Bildern von Sehenswürdigkeiten wurde am Anfang als Motivation erwähnt. Für diese Aufgabe wäre das GMM am besten geeignet.

5.5.4.MOG

Dieses Verfahren hat sich über die letzten 16 Jahre einen wohl verdienten Namen gemacht. Es liefert ein sehr gutes Ergebnis und rechnet schnell genug um bei einer Liveberechnung und einer Bildgröße von 320x240 eine Framerate von 30fps zuzulassen. Das einzige Problem bietet der vorher eingegangene Kompromiss zwischen Qualität und Rechenzeit, wegen welchem die RGB-Werte der Pixel einzeln behandelt werden.

5.5.5.MOG+MRF

Dieses Verfahren liefert von allen das beste Ergebnis. Trotzdem ist es noch schnell genug um eine Framerate von 15fps zuzulassen bei einer Bildgröße von 320x240 . Es stellt sich jedoch die Frage ob sich diese Einbußen in Rechenzeit lohnen für die vergleichsweise kleine Verbesserung an einem bereits guten Ergebnis des MOG, welche wahrscheinlich auch mit einem Medianfilter erreichbar wären.

Literaturverzeichnis

- [1] Antoine Vacavant, Thierry Chateau, Alexis Wilhelm and Laurent Lequievre. A Benchmark Dataset for Foreground/Background Extraction. In ACCV 2012, Workshop: Background Models Challenge, LNCS 7728, 291-300. November 2012, Daejeon, Korea.
- [2] R. Vezzani, R. Cucchiara, "Video Surveillance Online Repository (ViSOR): an integrated framework" (available online) in Multimedia Tools and Applications, DOI 10.1007/s11042-009-0402-9, 2009
<http://imagelab.ing.unimore.it/Pubblicazioni/pubblicazioni/Visorjournal.pdf> [18.07.2015]
- [3] S. Pellegrini, A. Ess, K. Schindler, L. van Gool, "You'll Never Walk Alone: Modeling Social Behavior for Multi-target Tracking"
http://www.igp.ethz.ch/photogrammetry/publications/pdf_folder/pellegrini09iccv.pdf [18.07.2015]
- [4] Simon A. Barker "Image Segmentation using Markov Random Field Models"
<http://www.maia.ub.es/~cerquide/research/GraphicalModelsForVolumeSegmentation/MarkovRandomFields/Cambridge/Image%20Segmentation%20Using%20Markov%20Random%20Field%20Models> [18.07.2015]
- [5] http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0809/ORCHARD/
[18.07.2015]
- [6] Julian Besag "On the Statistical Analysis of Dirty Pictures", pp.7
<https://stat.duke.edu/~scs/Courses/Stat376/Papers/GibbsFieldEst/BesagDirtyPicsJRSSB1986.pdf>
[08.07.15]
- [7] Chris Stauffer, W.E.L Grimson "Adaptive background mixture models for real-time tracking"
http://www.ai.mit.edu/projects/vsam/Publications/stauffer_cvpr98_track.pdf [18.07.2015]
- [8] R.Cucchiara, C.Gрана, M.Piccardi, A.Prati "Statistic and Knowledge-based Moving Object Detection in Traffic Scenes" draft version
<http://imagelab.ing.unimore.it/imagelab/pubblicazioni/itsc2000.pdf> [18.07.2015]
- [9] Shu-Jhen Fan Jiang, Kahlil Muchtar¹, Chih-Yang Lin¹, Li-Wei Kang² and Chia-Hung Yeh³
Background subtraction by modeling pixel and neighborhood information
http://www.apsipa.org/proceedings_2012/papers/230.pdf [18.07.2015]

Abbildungsverzeichnis

Abb.1:.....	Ergebnis einer Vordergrund-Segmentierung der BMC.....	7
Abb.2:.....	Bilddifferenz von zwei Frames [1].....	8
Abb.3:.....	Aperture Problem.....	11
Abb.4:.....	Schlechte ICM Werte.....	21
Abb.5:.....	Pixelnachbarschaften visualisiert.....	29
Abb.6:.....	Median Vergleich für n.....	30
Abb.7:.....	Smoothing Vergleich für n.....	31
Abb.8:.....	GMM Vergleich für die Lernrate.....	32
Abb.9:.....	MOG Vergleich für den Hintergrundanteil.....	34
Abb.10:.....	MOG Vergleich für die Lernrate.....	35
Abb.11:.....	Testvideos.....	37
Abb.12:.....	Qualitätsvergleich.....	39
Abb.13:.....	MOG vs. MOG+MRF.....	41

Diagrammverzeichnis

Diag.1:.....	Pixelintensitäten I zur Zeit t.....	10
Diag.2:.....	Mixture of Gaussian Visualisierung.....	17
Diag.3:.....	Visualisierung der Zuordnung zu einer Gauß-Verteilung.....	18
Diag.4:.....	ICM Filter Kernel.....	21
Diag.5:.....	FrameGetter und VideoCreator.....	25
Diag.6:.....	MRF Vergleich für Lambda.....	35

Tabellenverzeichnis

Tab.1:.....	BufferedImage zu Rohdaten Konvertierungs-Geschwindigkeiten.....	28
Tab.1:.....	Median Rechenzeit Vergleich.....	31
Tab.1:.....	Speicherverbrauch der Algorithmen.....	38
Tab.1:.....	Geschwindigkeitsvergleich.....	39

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Ort, Datum

Unterschrift