Segmentierung von Bildern in Vorder- und Hintergrund mithilfe von Gaussian Mixture Models

Marvin Suhr

Studiengang Angewandte Informatik, Hochschule RheinMain
msuhr001@student.hs-rm.de

Abstract

Dieses Paper befasst sich grundlegend mit Gaussian Mixture Models und deren Anwendung, speziell im Kontext der Segmentierung von Bildern in Vorder- und Hintergrund. Zusätzlich wird der Expectation-Maximization-Algorithmus (EM) betrachtet, der verwendet werden kann um die GMMs zu "trainieren", also Parameter zu finden, die vorerst unbekannt waren. Es stellt sich heraus, dass der EM-Algorithmus bezogen auf Videoüberwachung sehr resourcenlastig ist und sich nur bedingt umsetzen lässt.

I. Einführung

Ein immer wiederkehrendes Problem im Bereich der Videoüberwachung ist das Einteilen der aufgezeichneten Bewegtbilder in Vorder- und Hintergrund, um die irrelevanten Daten schon im Voraus aussortieren zu können. Wenn bei einer Videoüberwachung beispielsweise festgestellt wird, dass sich lediglich Dinge im Hintergrund des Bildes bewegt haben (z.B. die Blätter eines Baumes), so sollten die entsprechenden Bilder automatisch ausgeschlossen werden. Ein solches Überwachungssystem sollte zusätzlich robust gegenüber möglichst vielen Änderungen der Szene (z.B. Lichtverhältnisse, Wetterlage) sein, damit es nach einer einmaligen Initialisierung ohne Überwachung durch Menschen laufen kann. Außerdem sollen relevante Objekte (z.B. Menschen, Autos, Tiere, ...) über mehrere Bilder hinweg erkannt und verfolgt werden können.

Der Fortschritt der Technik zur Zeit der Veröffentlichung des Papers von Stauffer [1] ermöglichte dies nun besser als je zuvor, da komplexere Modelle bzw. Berechnungen angewendet werden konnten. Es existierten bereits mehrere Ansätze, die jedoch alle ihre Nachteile hatten. Hier wird nun ein neuer Ansatz beschrieben, welcher die Stärken der alten Versuche zusammenführt und ihre Schwächen minimieren soll. Dies wird mit Gaussian Mixture Models erreicht, die auf die einzelnen Pixel der Bilder angewendet werden. Der Hintergrund des Bildes wird dann heuristisch bestimmt.

Des Weiteren setzt sich dieses Paper mit den Grundlagen von Gaussian Mixture Models, dem Expectation-Maximization-Algorithmus und der Anwendung des Algorithmus auf die GMMs auseinander. Schließlich versuchen wir festzustellen, ob der EM-Algorithmus sich im Gebiet der Videoüberwachung anwenden lässt, um die Pixel in Voder- und Hintergrund einzuteilen.

II. Gaussian Mixture Models

Ein Gaussian Mixture Model (GMM) ist eine gewichtete Summe mehrerer Gauß-Funktionen, das in den verschiedensten Bereichen der Statistik angewendet werden kann. Es wird angenommen, dass alle Werte der Stichprobe von einer Mischung von Gauß-Verteilungen mit unbekannten Parametern erzeugt wurden. GMMs werden vor allem beim Clustering und bei der multi-modalen Dichteschätzung

eingesetzt und dadurch auch in Bereichen der Spracherkennung verwendet.

Die Grundlage für GMMs bildet die Normal- bzw. Gauß-Verteilung mit der Wahrscheinlichkeitsdichte

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad (1)$$

mit Mittelwert μ und Standardabweichung σ. Das Plotten dieser Funktion mit $\mu = 0$ und $\sigma = 1$ führt zu der wohlbekannten Gaußschen Glockenkurve. Im Gegensatz zu normalen Gauß-Funktionen werden bei GMMs mehrere dieser Funktionen kombiniert und verschieden gewichtet, um Verteilungen in mehreren Dimensionen darzustellen. Ein GMM wird beschrieben durch eine multivariate Funktion

$$p(x) = \sum_{i=1}^{n} w_i \cdot \mathcal{N}(x|\mu_i, \Sigma_i)$$

wobei $p(x)$ die Wahrscheinlichkeitsdichte einer bestimmten Beobachtung ist, n die Anzahl der Gauß-Verteilungen, w_i die Gewichtung der Verteilung mit $\sum_{i=1}^{n} w_i = 1$, μ_i der Mittelwert der i-ten Gauß-Funktion und Σ_i die Kovarianzmatrix der i-ten Gauß-Funktion. $\mathcal{N}$ ist eine d-variate Normalverteilung mit

$$\mathcal{N}(x|\mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \cdot e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

wobei d die Anzahl der Dimensionen und $|\Sigma|$ die Determinante der Kovarianzmatrix ist. [2] Im Grunde ist dies eine Erweiterung der Wahrscheinlichkeitsdichtefunktion aus Gleichung 1 für mehrdimensionale Stichproben. In Figure 1 sieht man beispielsweise den Plot einer zweidimensionalen Gauß-Verteilung.

Der Vorteil von GMMs gegenüber "normalen" multivariaten Gauß-Funktionen wird klar, wenn man Figure 2 und 3 vergleicht. Die multivariate Gauß-Verteilung kann nicht "erkennen", dass es sich um zwei Gruppierungen von Daten handelt und behandelt die Daten, als ob sie nur ein Cluster wären. Das GMM hingegen verwendet pro Ansammlung eine eigene multivariate Gauß-Verteilung. Jede

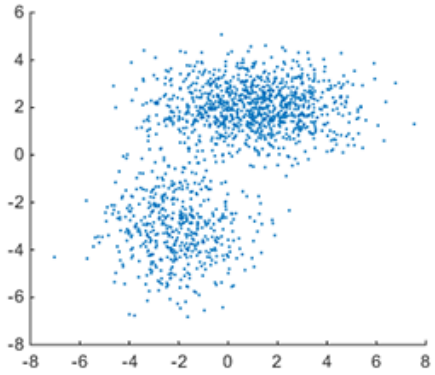

Figure 1: Beispiel für eine zweidimensionale Gauß-Verteilung.

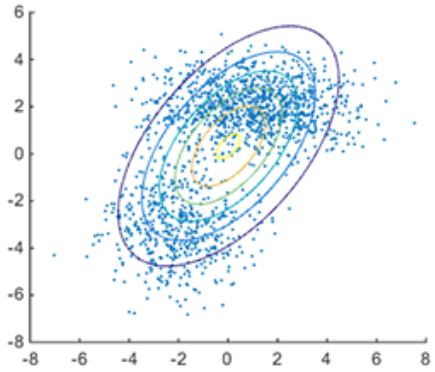

Figure 2: Multivariate Gauß-Verteilung, angepasst an die Werte aus Figure 1. Man erkennt sofort, dass das Modell nicht wirklich auf die Daten passt.

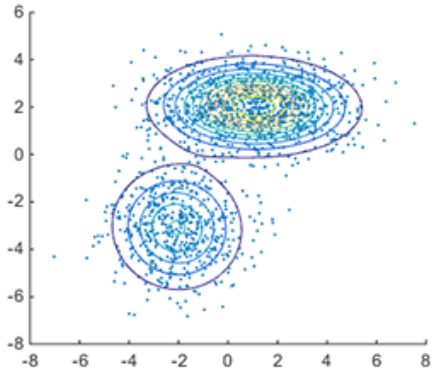

Figure 3: GMM, mithilfe des EM-Algorithmus angepasst an die Werte aus Figure 1. (Bildquelle (Figure 1 - 3): [3])

Komponente des GMMs repräsentiert dann ein Cluster von Daten.

Mithilfe von verschiedenen Machine Learning-Konzepten wie z.B. dem EM-Algorithmus können nun die Parameter des GMMs bestimmt werden ("Trainieren" des GMMs), um mit höherer Wahrscheinlichkeit zu sagen, welche Daten zu welchem Cluster gehören.

II.1 Der EM-Algorithmus

Im Gebiet der Statistik passiert es oft, dass wir zwar eine Stichprobe von Daten haben, jedoch nicht wissen mit welcher Art von Verteilung oder vor allem mit welchen Werten für die Parameter diese Daten entstanden sind. Grundsätzlich sind uns einige Werte gegeben, andere sind uns jedoch unbekannt. Eigentlich würde es Sinn machen, hier die Maximum-Likelihood-Methode (ML) zu verwenden um die Parameter möglichst gut (entsprechend den gegebenen Werten) zu schätzen [4]. Uns stellt sich jedoch das Problem, dass wir sowohl die Parameter (z.B. μ und σ bzw. Σ) als auch einige "versteckte Werte" nicht kennen. Das bedeutet, wir können weder die Parameter bestimmen (wir müssten die versteckten Werte kennen) noch die versteckten Werte direkt berechnen bzw. schätzen (wir müssten die Parameter kennen). Für diese Problematik existiert der Expectation-Maximization-Algorithmus.

Der EM-Algorithmus ist eine gute Methode, eben diese fehlenden Parameter einer unbekannten Verteilung mit unbekannten Daten zu finden und zu verbessern, also näher an die wirklichen Werte zu bringen. Der Algorithmus besteht aus zwei Schritten, die iterativ wiederholt werden, bis die geschätzten Parameter schließlich konvergieren. Im ersten Schritt (Expectation step) werden die fehlenden Werte anhand der gegebenen Daten geschätzt. Im zweiten Schritt (Maximization step) wird angenommen, die Werte aus dem ersten Schritt seien die wirklichen Werte der Verteilung. Auf diese Werte wird nun ML angewendet um die eigentlich gesuchten Parameter zu berechnen. Zu Beginn des Algorithmus werden die Parameter einmalig geraten.

Der EM-Algorithmus kann sehr vielseitig benutzt werden, wird jedoch hauptsächlich auf Mischverteilungsmodellen (wie z.B. GMMs) angewendet, worauf wir in II.2 zurückkommen werden. Generell haben wir die Dichtefunktion $p(x|\Theta)$ gegeben, wobei x die beobachteten Daten und Θ ein Vektor der Parameter der Verteilung sind. Der Ausdruck lässt sich folgendermaßen umformen:

$$p(x|\Theta) = \prod_{i=1}^{n} p(x_i|\Theta) = \mathcal{L}(\Theta|x)$$

Ganz rechts steht eine Likelihood-Funktion, die die Wahrscheinlichkeit der Parameter entsprechend der gegebenen Daten beschreibt. Gesucht ist logischerweise ein Θ, für welches diese Wahrscheinlichkeit maximiert wird:

$$\Theta^* = \underset{\Theta}{argmax}\ \mathcal{L}(\Theta|x)$$

Oftmals wird stattdessen $log(\mathcal{L}(\Theta|x)$ maximiert, da es analytisch einfacher ist. [5] In günstigen Fällen reicht es, $\mathcal{L}$ abzuleiten und 0 zu setzen um unser Maximum auszurechnen. Wenn dies nicht möglich ist, müssen wir uns mit anderen Verfahren wie z.B. dem EM-Algorithmus auseinandersetzen, der jetzt genauer erklärt wird.

Da wir einen unvollständigen Datensatz betrachten, uns also nicht alle Daten bekannt sind, beginnen wir mit

$$p(z|\Theta) = p(x, y|\Theta) = p(y|x, \Theta) \cdot p(x|\Theta)$$

wobei z der vollständige Datensatz ist, x der unvollständige und y der unbekannte "Rest". Θ ist nach wie vor ein Vektor der Parameter unserer Verteilung (z.B. μ und σ^2 bei einer Normalverteilung). Im ersten Schritt wurde z lediglich zu x, y umgeformt. Im zweiten Schritt wurde der Satz der bedingten Wahrscheinlichkeit angewendet um x und y zu trennen. Zu unserer neuen Dichtefunktion gibt es auch eine entsprechende Likelihood-Funktion:

$$\mathcal{L}(\Theta|z) = \mathcal{L}(\Theta|x, y) = p(x, y|\Theta)$$

Da die fehlenden Daten y unbekannt und zufällig sind, ist auch die gesamte Funktion eine Zufallsvariable. Zuerst findet der EM-Algorithmus den Erwartungswert zur log-likelihood der gerade berechneten Funktion. Wir definieren:

$$Q(\Theta, \Theta^{(i-1)}) = E[logp(x, y|\Theta)|x, \Theta^{(i-1)}]$$

wobei $\Theta^{(i-1)}$ die momentane Schätzung der Parameter und Θ die neu zu berechnenden Parameter (zur Erhöhung von Q) beschreibt. Da sich x (die bekannten Daten) nicht mehr verändert und $\Theta^{(i-1)}$ die von uns geschätzten Parameter sind, handelt es sich bei beiden um Konstanten. Θ ist eine Variable die erst neu berechnet bzw. geschätzt wird und y ist eine Zufallsvariable die abhängig von x und $\Theta^{(i-1)}$ ist. Die rechte Seite der vorherigen Gleichung lässt sich umschreiben:

$$\begin{aligned} &E[logp(x, y|\Theta)|x, \Theta^{(i-1)}] \\ &= \int_{y \in \mathcal{Y}} logp(x, y|\Theta) \cdot f(y|x, \Theta^{(i-1)}) dy \end{aligned}$$

wobei $\mathcal{Y}$ den Raum aller Werte, den y einnehmen kann, beschreibt. Dies ist der Expectation step. Das Ergebnis dieser Gleichung ist ein vorläufiger Erwartungswert der gesuchten Verteilung. Im zweiten Schritt (Maximization step) wird die Maximum-Likelihood-Methode angewandt um über die im ersten Schritt geschätzten Werte die eigentlich gesuchten Parameter Θ zu berechnen bzw. besser zu schätzen. Wir suchen also ein Θ welches unsere Q-Funktion maximiert:

$$\Theta^{(i)} = \underset{\Theta}{\operatorname{argmax}}\ Q(\Theta, \Theta^{(i-1)})$$

Daraus ergibt sich ein neues Θ ($\Theta^{(i)}$), welches im folgenden Durchlauf im ersten Schritt (statt $\Theta^{(i-1)}$) verwendet wird, um Θ immer besser zu schätzen. Der Algorithmus wird nun wiederholt, bis die errechneten Parameter sich nicht mehr stark ändern. Das Ergebnis ist eine immer bessere Schätzung der gesuchten Parameter.

II.2 Der EM-Algorithmus für GMMs

Da unser ursprüngliches Ziel das Einteilen von Bildern in Vorder- und Hintergrund mithilfe von GMMs ist, ist für uns vor allen Dingen auch interessant, wie sich der EM-Algorithmus auf GMMs anwenden lässt. Die "versteckten Werte" die wir mit dem EM-Algorithmus suchen, beziehen sich hier darauf, welche Ansammlung von Daten zu welchem Gaussian aus unserem Modell passt. Wir wollen also einteilen, welches Cluster von welcher Gauß-Verteilung erzeugt wurde. Genauer ausgedrückt ist unser Problem, dass uns neben dieser Information zusätzlich die Parameter (μ, Σ und w) der Verteilungen fehlen.

Wir stecken fest: Wenn wir die wirklichen Mittelwerte, Kovarianzen und Gewichtungen der Verteilungen kennen würden, könnten wir eine sichere Aussage dazu treffen, welche Samples zu welcher Gauß-Verteilung gehören. Man könnte anhand der Parameter für jedes Sample relativ leicht ausrechnen, mit welcher Wahrscheinlichkeit es von jedem der Gaussians stammt. Um diese Parameter für die jeweilige Verteilung zu berechnen, müssten wir jedoch wissen, welche Samples zu welcher Verteilung gehören. Wenn wir dieses Wissen hätten, wäre es sehr einfach die Parameter der jeweiligen Gaussians zu berechnen. Wir bräuchten quasi eine der Antworten, um auf die andere Antwort zu kommen. Da uns jedoch beide fehlen, kommen wir vorerst nicht weiter. Wir stehen also vor einer Art Henne-Ei-Problem. Es bietet sich der EM-Algorithmus an, um diese Problematik zu lösen. Ein passender Pseudocode befindet sich auf der nächsten Seite oben links (Algorithm 1).

Anstatt eines unserer Problem direkt lösen zu wollen, versuchen wir sozusagen beide Probleme schrittweise parallel zu lösen indem wir abwechselnd die Schritte des EM-Algorithmus durchlaufen. Da wir keinen Ansatzpunkt haben, müssen wir die Mittelwerte, Kovarianzen und Gewichtungen für jeden Gaussian in unserem GMM vor dem er-

Data: Daten des GMMs $x = (x_1, ..., x_k)$
Result: Optimierter Datenvektor $\Theta = (\mu_1, ..., \mu_n, \Sigma_1, ..., \Sigma_n, w_1, ..., w_n)$
Initialisiere Datenvektor $\hat{\Theta} = (\hat{\mu}_1, ..., \hat{\mu}_n, \hat{\Sigma}_1, ..., \hat{\Sigma}_n, \hat{w}_1, ..., \hat{w}_n)$ mit Zufallswerten (μ: Mittelwerte, Σ: Kovarianzmatrizen, w: Gewichtungen);
while Θ *nicht konvergiert* **do**
  E-step: Für jedes x mithilfe der Parameter $\hat{\Theta}$ berechnen, wie wahrscheinlich es zu jedem der n Gaussians passt;
  M-step: Die Parameter $\hat{\Theta}$ mit den neu eingeteilten Clustern neu schätzen;
  $\Theta = \hat{\Theta}$;
end
return Θ;

Algorithm 1: EM-Algorithmus

sten Schritt des Algorithmus zufällig wählen. Dabei spielt es keine Rolle, wie nah diese Werte an den wirklichen Werten liegen, da der EM-Algorithmus immer konvergiert. Im Expectation step nehmen wir nun an, unsere geratenen Werte seien die wirklichen Werte und berechnen daraus den "assignment score"

$$\gamma(z_{nk}) = \frac{w_k \cdot \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} w_j \cdot \mathcal{N}(x_n|\mu_j, \Sigma_j)}$$

zwischen jeder Stichprobe und jedem Gaussian [6]. Für jeden Wert x_n in den Clustern wird berechnet, wie stark der Gaussian k für ihn verantwortlich ist. Dies ist unser Expectation step. Im folgenden Maximization step berechnen wir die Parameter der Verteilungen neu, indem wir die vorher berechneten assignment scores folgendermaßen benutzen:

Mittelwert des k-ten Gaussians:

$$\mu_k = \frac{1}{N_k} \cdot \sum_{n=1}^{N} \gamma(z_{nk}) \cdot x_n$$

Kovarianzmatrix des k-ten Gaussians:

$$\Sigma_k = \frac{1}{N_k} \cdot \sum_{n=1}^{N} \gamma(z_{nk}) \cdot (x_n - \mu_k) \cdot (x_n - \mu_k)^T$$

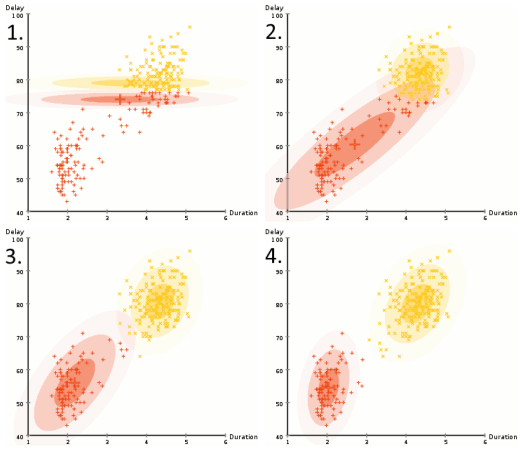

Figure 4: Anwendung des EM-Algorithmus auf ein GMM. Das erste Bild zeigt den ersten Schritt des Algorithmus, bei dem die Parameter mit Zufallswerten initialisiert wurden. Auf den folgenden Bildern sieht man, wie das Modell sich über die Laufzeit des Algorithmus hinweg verändert. Auf dem letzten Bild kann man schließlich erkennen, dass ein passendes Modell für die Daten gefunden wurde. (Bildquelle: [7])

Gewichtung des k-ten Gaussians:

$$w_k = \frac{N_k}{N}$$

wobei

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

und N die Gesamtanzahl aller Werte beschreibt. Daraus folgen bereits besser geschätzte Werte für unsere Parameter. Nun wiederholen wir die beiden Schritte beliebig oft (bzw. bis die Werte der Parameter konvergieren). Im zweiten Durchlauf berechnen wir die assignment scores erneut. Statt Zufallswerten können wir nun die Werte nehmen, die sich aus unserem ersten Durchgang ergeben haben. Aus diesen verbesserten assignment scores folgen logischerweise auch besser geschätzte Parameter. Wenn die beiden Schritte genügend oft wiederholt werden, kovergieren die Werte und bieten eine optimale Schätzung der gesuchten Parameter.

III. Anwendung im Gebiet der Videoüberwachung

Im Folgenden wird betrachtet, wie das Prinzip der Gaussian Mixture Models im Gebiet der Videoüberwachung angewendet werden kann. Der Ansatz von Stauffer ([1]) beschreibt eine Methode, bei der jeder Pixel des Bildes als ein eigenes GMM modelliert wird. So wird versucht zu entscheiden, ob es sich bei dem jeweiligen Pixel um Vorder- oder Hintergrund handelt. Dazu werden die Gaussians nach einer einfachen Heuristik ausgewertet, wenn einer ihrer Parameter aktualisiert wurde. Es wird bestimmt, welche Pixel am wahrscheinlichsten zum Hintergrund gehören. Pixel, die nicht in die Verteilungen passen (also wahrscheinlich nicht zum Hintergrund gehören), werden vorerst als Vordergrund-Pixel eingestuft und über die nächsten Frames hinweg beobachtet um weitere Aussagen treffen zu können.

Währenddessen werden verschiedene Modelle des Hintergrundes zwischengespeichert. Das bedeutet, dass die "alten" Hintergründe noch verfügbar sind, falls z.B. ein neues Objekt in den Hintergrund tritt oder sich generell etwas im Hintergrund ändert. Das System kann sich daher schneller erholen, falls Änderungen im Hintergrund auftreten bzw. wenn diese Änderungen wieder rückgängig gemacht werden. Dies führt dazu, dass das System sehr robust gegenüber Änderungen der Szene wie z.B. Wetterwechsel ist. Außerdem ist es möglich, Objekte die sich im Vordergrund befinden zu erkennen und über mehrere Frames hinweg zu verfolgen. Dieses System wurde ohne Änderung der Parameter erfolgreich 16 Monate am Stück laufen gelassen.

Leider ist es in diesem System nicht möglich, den EM-Algorithmus anzuwenden, um die Pixel in Vorder- und Hintergrund einzuteilen, da dieser in jedem Frame auf jeden einzelnen Pixel angewendet werden müsste, was zu teuer bzw. aufwendig wäre. Stattdessen wird eine "on-line K-means approximation" [1] implementiert.

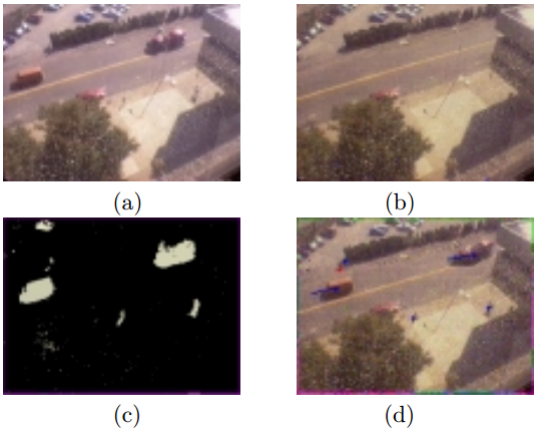

Figure 5: Ausführung des Programms. (a) zeigt das jetzige Bild, (b) den "Durchschnitt" des Hintergrundes bzw. der jetzige Hintergrund, (c) die Pixel die als Vordergrund eingestuft wurden, (d) zeigt zusätzlich noch Informationen zum Tracking. (Bildquelle: [1])

III.1 Funktionsweise

Jeder einzelne Pixel des Bildes wird über die Zeit hinweg, also über mehrere Frames, betrachtet. Wir kennen zu jeder Zeit den "Verlauf" jedes Pixels, also seinen Zustand bzw. seine Farbe (RGB-Werte) in den vorherigen Frames. Uns ist also nicht nur der jetzige Zustand, sondern auch der Verlauf

$$\{X_1, ..., X_t\} = \{I(x_0, y_0, i) : 1 \leq i \leq t\}$$

für jeden Pixel $\{x_0, y_0\}$ bekannt, wobei I die Bildsequenz des Pixels beschreibt. Das bedeutet, wir kennen immer die letzten t Zustände jedes Pixels bzw. seine Farbe in den letzten t Frames. Da das System möglichst schnell auf Änderungen der Lichtverhältnisse und des Hintergrundes reagieren soll, liegt es nahe, dass neuere Beobachtungen bzw. Werte wichtiger in der Schätzung der Gaussian-Parameter sind. Diese sind daher höher gewichtet.

Wir entscheiden uns also für ein passendes Modell, welches unseren Ansprüchen entspricht, nämlich ein GMM ähnlich der Gleichung (II). Dies ist eine gute Wahl, da wir so

im Gegensatz zu einer einzigen Gauß-Funktion noch einen Gewichtungsfaktor w in unserer Gleichung haben. So lässt sich jeder Gaussian aus unserem GMM anders gewichten. Die Wahrscheinlichkeit den derzeitigen Pixelwert zu beobachten wird beschrieben als

$$P(X_t) = \sum_{i=1}^{n} w_{i,t} * \mathcal{N}(X_t, \mu_{i,t}, \Sigma_{i,t})$$

Hier beschreibt n die Anzahl der Verteilungen und wird entsprechend dem verfügbaren Speicher und der Rechenleistung gewählt (zur Zeit der Veröffentlichung waren es zwischen 3 und 5). $w_{i,t}$ ist eine Schätzung der Gewichtung des i-ten Gaussians zur Zeit t, $\mu_{i,t}$ ist der Mittelwert des i-ten Gaussians zur Zeit t, $\Sigma_{i,t}$ ist die Kovarianzmatrix des i-ten Gaussians zur Zeit t und $\mathcal{N}$ ist die Normalverteilung aus (II). Es wird angenommen, die Kovarianzmatrix sei der Form $\Sigma_{i,t} = \sigma_i^2 \cdot I$ (wobei I eine Identitätsmatrix ist), da wir uns so eine teure Matrixinversion sparen können. Dies bedeutet zwar, dass unsere Berechnungen weniger genau sind, was jedoch vernachlässigt werden kann.

Der bereits in II behandelte EM-Algorithmus würde sich hier typischerweise anbieten, um die Wahrscheinlichkeit der betrachteten Daten zu maximieren. Das Problem ist jedoch, dass der EM-Algorithmus in jedem Frame auf jeden Pixel angewendet werden müsste, da die Pixelverläufe sich über die Zeit hinweg verändern. Dies wäre vom Rechenaufwand extrem teuer und zumindest zur Zeit der Veröffentlichung des Systems von Stauffer unrealistisch umzusetzen. Daher müssen wir eine andere Methode anwenden; es bietet sich eine "on-line K-means approximation" [1] an.

Während der Laufzeit des Programms wird jeder neu beobachtete Pixelwert X_t gegen die K vorhandenen Gauß-Verteilungen dieses Pixels geprüft, bis eine Übereinstimmung gefunden wird. Als Übereinstimmung wird ein Wert innerhalb von 2,5 Standardabweichungen zum Mittelwert bezeichnet. Wenn für den neuen Pixel innerhalb der K Verteilungen eine Übereinstimmung gefunden wird, wird der Pixel (immer noch) als Hintergrund betrachtet. Die übrigen Verteilungen werden dann nicht mehr geprüft. Falls keine der K vorhandenen Verteilungen eine ausreichend gute Übereinstimmung liefert, wird der Pixel als Vordergrund betrachtet. In diesem Fall wird die unwahrscheinlichste Verteilung aus unserem GMM gelöscht und wie folgt ersetzt:

- Mittelwert $\mu = X_t$
- Varianz σ^2: ein hoher Wert
- Gewicht w: ein niedriger Wert

Das bedeutet, dass bereits existierenden Modelle des Hintergrundes nicht gelöscht werden, wenn etwas neues dem Hintergrund hinzugefügt wird. Lediglich die unwahrscheinlichste Verteilung wird ersetzt, was bedeutet, dass stets K Modelle des Hintergrundes verfügbar sind bzw. geprüft werden.

Die bisherigen Gewichtungen der K Verteilungen, $w_{k,t}$, werden wie folgt angepasst:

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

α beschreibt hier die "learning rate" und $M_{k,t}$ ist 1 für das übereinstimmende und 0 für alle anderen Modelle. Dies bedeutet, dass das Gewicht der übereinstimmenden Verteilung erhöht, das Gewicht aller anderen Verteilungen jedoch reduziert wird, was auch sinnvoll ist. Die Parameter μ und σ der nicht übereinstimmenden Verteilungen werden nicht verändert. Für das übereinstimmende Modell werden die Werte folgendermaßen angepasst:

$$\mu_t = (1 - \rho) \cdot \mu_{t-1} + \rho X_t$$

$$\sigma_t^2 = (1 - \rho) \cdot \sigma_{t-1}^2 + \rho \cdot (X_t - \mu_t)^T \cdot (X_t - \mu_t)$$

mit

$$\rho = \alpha \mathcal{N}(X_t | \mu_k, \sigma_k)$$

Der Mittelwert μ wird also näher an X_t angepasst und die Varianz σ^2 wird reduziert.

Einer der hauptsächlichen Vorteile dieser Methode ist, dass zu jeder Zeit mehrere Modelle des Hintergrundes existieren. So ist das System robust gegenüber allen möglichen

Änderungen der Szene (z.B. Änderung der Lichtverhältnisse, Einführung bzw. Entfernen von Objekten des Hintergrundes). Falls z.B. ein neues Objekt in die Szene eingeführt wird, dort dann so lange verharrt bis es Teil des Hintergrundes wird, dann aber wieder entfernt wird, so existiert trotzdem noch das "alte" Modell des Hintergrundes in unserem GMM. Die älteren Modelle des Hintergrundes bleiben ein Teil des Mixture Models bis sie zum K-t wahrscheinlichsten Modell werden und dann eine neue Farbe beobachtet wird. Damit wird diese Verteilung dann schließlich durch eine neue ersetzt.

Mit dieser Methode können die Vordergrundpixel erkannt werden. Wir müssen jedoch immer noch herausfinden, welche Gaussians unseres GMMs die Hintergrundpixel darstellen. Wir wollen also Verteilungen mit einem hohen Gewicht (über lange Zeit hinweg die selbe Farbe beobachtet) und einer niedrigen Varianz (geringe Anzahl Änderungen des Pixels) finden. Um diese Verteilungen zu finden, werden die Gaussians des GMMs jedes Pixels nach ihrem Wert für w/σ sortiert. Dieser Wert steigt mit wachsendem Gewicht w und sinkt mit steigendern Standardabweichung σ, also genau was gesucht ist.

In dieser Ordnung steigen die Pixel, die am wahrscheinlichsten zum Hintergrund gehören also an die Spitze, während die anderen in der Liste sinken und ggf. von neuen Werten ersetzt werden. Die Gewichte in dieser Ordnung werden so lange addiert, bis ein vorher festgelegter Schwellwert T überschritten wird. Ein höheres T bedeutet hier, dass mehrere Pixelwerte als Hintergrund erkannt werden. Schließlich werden die Mittelwerte der Gaussians in dieser Summe als die Farben gewertet, die den Hintergrund repräsentieren. Wir haben nun also sowohl die Pixel markiert, die den Vordergrund bilden, als auch ein Modell gefunden, welches den Hintergrund repräsentiert.

IV. Auswertung

Das System von Stauffer wurde 16 Monate lang mit einer Auflösung von 120x160 Pixeln bei 11-13 Bildern pro Sekunde erfolgreich getestet. Das System hat gelegentlich Probleme, wenn sich Licht- und Wetterverhältnisse plötzlich ändern, funktioniert sonst aber sehr stabil. Es kann dadurch kurzzeitig durcheinander gebracht werden, erholt sich jedoch relativ schnell wieder, besonders weil mehrere Modelle des Hintergrundes zwischengespeichert sind. Außerdem ist es möglich Objekte zu verfolgen bzw. teilweise sogar zu erkennen (z.B. Autos und Menschen). Fehler treten nur auf, wenn sich viele Objekte gleichzeitig in der Szene bewegen, sie sich zu nahe sind oder sich ihre Wege kreuzen.

IV.1 Vorherige Arbeit und Mängel

Stauffer vergleicht sein System in seiner Ausarbeitung mit einigen andere Versuchen, ein ähnliches System zu implementieren. Diese sind jedoch alle auf bestimmte Weise mangelhaft oder anfällig für Fehler, vor allem bei Änderungen der Lichtverhältnisse. Andere Ansätze die genannt wurden:

- Den Durchschnitt der Pixelwerte bilden und daraus den Hintergrund modellieren (Problem: Fehler bei vielen bzw. sich langsam bewegenden Objekten)
- Jeden einzelnen Pixel mit einem Kalman-Filter modellieren (robuster gegenüber Änderungen der Lichtverhältnisse, erholt sich jedoch nur langsam gegenüber Änderungen des Hintergrundes)
- Jeden einzelnen Pixel mit einer Gauß-Funktion modellieren (Relativ gute Ergebnisse, jedoch nicht im Freien getestet)

IV.2 Heutiger Stand

Durch erhöhte Rechenleistung könnte man ein ähnliches System heutzutage natürlich mit einer weitaus höheren Auflösung sowie einer

höheren Zahl von Bildern pro Sekunde einsetzen. Zusätzlich könnte es möglich sein, den resorucenlastigen EM-Algorithmus auf die GMMs des Systems anzuwenden. Dies würde womöglich zu einer besseren bzw. genaueren Einteilung in Vorder- und Hintergrund führen. Außerdem könnte man eine höhere Anzahl von Gauß-Funktion in den GMMs benutzen (hier waren es nur 3 - 5).

References

[1] Chris Stauffer and W Eric L Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2. IEEE, 1999.

[2] Douglas Reynolds. Gaussian mixture models. In *Encyclopedia of Biometrics*, pages 659–663. Springer, 2009.

[3] Robert Poehlmann. Introduction to gaussian mixture models. Available from `http://recognize-speech.com/basics/introduction-to-gaussian-mixture-models/14-basics` (retrieved: December 2015), December 2014.

[4] Michael Baumann. Expectation maximization. Available from `http://recognize-speech.com/basics/introduction-to-gaussian-mixture-models/14-basics` (retrieved: December 2015), December 2014.

[5] Jeff A Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.

[6] "petitegeek". Expectation maximization and gaussian mixture models. Available from `http://www.slideshare.net/petitegeek/expectation-maximization-and-gaussian-mixture-models` (retrieved: January 2016), June 2010.

[7] Wikipedia. Expectation-maximization algorithm. Available from `https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm` (retrieved: January 2016), December 2015.