Change Detection
Segmentierung von bewegten Objekten in Videosequenzen mittels Background Subtraction

Tim Gerrits Stefan Wegener

Orginalbild einer Videosequenz (links). Binärbild nach der Segmentierung durch PBAS∧MOG2 (mitte). Berechnete Bounding Box und Kontur auf dem Orginalbild (rechts).

***Zusammenfassung*—In dieser Arbeit wird ein Algorithmus vorgestellt, welcher automatisch bewegte Objekte in Videosequenzen detektiert. Als Eingabe für den Algorithmus werden Videosequenzen verwendet, welche in Einzelbilder aufgeteilt wurden. Pro Bild wird ein Binärbild erzeugt, wobei mit Weiß Vordergrundobjekte und mit Schwarz Hintergrundobjekte klassifiziert werden. Für die Evaluierung des im Paper beschriebenen Algorithmus wurden Videosequenzen verwendet, welche ebenfalls auf der CVPR 2012 Change Detection Workshop zur Evaluierung verwendet wurden.**

I. Einleitung

Computer Vision leitet sich aus dem Begriff Human Vision (d.h. visuelle Wahrnehmung des Menschen) ab. Ziel von der Computer Vision ist es, Prozesse der visuellen Wahrnehmung durch Algorithmen zu beschreiben. Die Erkennung von bewegten Objekten ist für die menschliche Wahrnehmung problemlos möglich. Dabei kann das menschliche Gehirn sogar im Bruchteil einer Sekunde komplexeste Bewegungen analysieren, um so z.B. in Gefahrensituationen schnell zu reagieren. Eine computergestützte Segmentierung von bewegten Objekten ist ein wichtiger Vorverarbeitungsschritt für viele Methoden, insbesondere im Bereich der Videoüberwachung.
In dieser Arbeit wird die Entwicklung und Implementierung eines solchen Algorithmus vorgestellt, welcher mittels Background Subtraction ein Hintergrundmodell für den jeweiligen Datensatz erstellt, um so für jedes Pixel im Einzelbild zu entscheiden, ob es sich um ein Vordergrund- oder ein Hintergrundpixel handelt.
In dem Organigramm (*Abbildung* **??** *Seite* **??**) ist die Funktionsweise des Algorithmus grob dargestellt. Im weiteren Verlauf der Arbeit wird der Algorithmus anhand der oben genannten Darstellung genauer erläutert.

II. Materialien und Methoden

A. Verwendete Frameworks und Programmiersprachen

In dem hier beschriebenen Algorithmus werden nur Open Source cross-platform C++ Bibliotheken verwendet.
Eine Umsetzung mit anderen Frameworks und Programmiersprachen ist denkbar, da sich der Algorithmus zur Lösung des Problems leicht übertragen lassen sollte.

1) OpenCV

Open CV ist eine Klassenbibliothek mit Algorithmen für die Bildverarbeitung und Computer Vision. Es ist eine Open Source Library, welche unter anderem für C++ verfügbar ist. Sie bietet Zugriff auf Basis-Algorithmen, wie z.B. Erosion und Dilatation, welche in dem hier vorgestellten Algorithmus verwendet wurden.

2) C++ und Qt

Als Klassenbibliothek für das Interface wurde auf Qt zurückgegriffen, da Qt eine komfortable Integration in C++ ermöglicht und diese Klassenbibliothek aktuell der Industriestandard für Interfacedesign ist.
C++ wurde als Programmiersprache gewählt, da diese optimal mit verschieden Frameworks zusammenarbeitet und sich durch Robustheit, Stabilität und Schnelligkeit auszeichnet.

B. Einlesen und Ausgabe von Videosequenzen

Im folgenden Abschnitt wird das Einlesen der Videosequenzen und die Ausgabe der Ergebnisse erläutert. Diese beiden

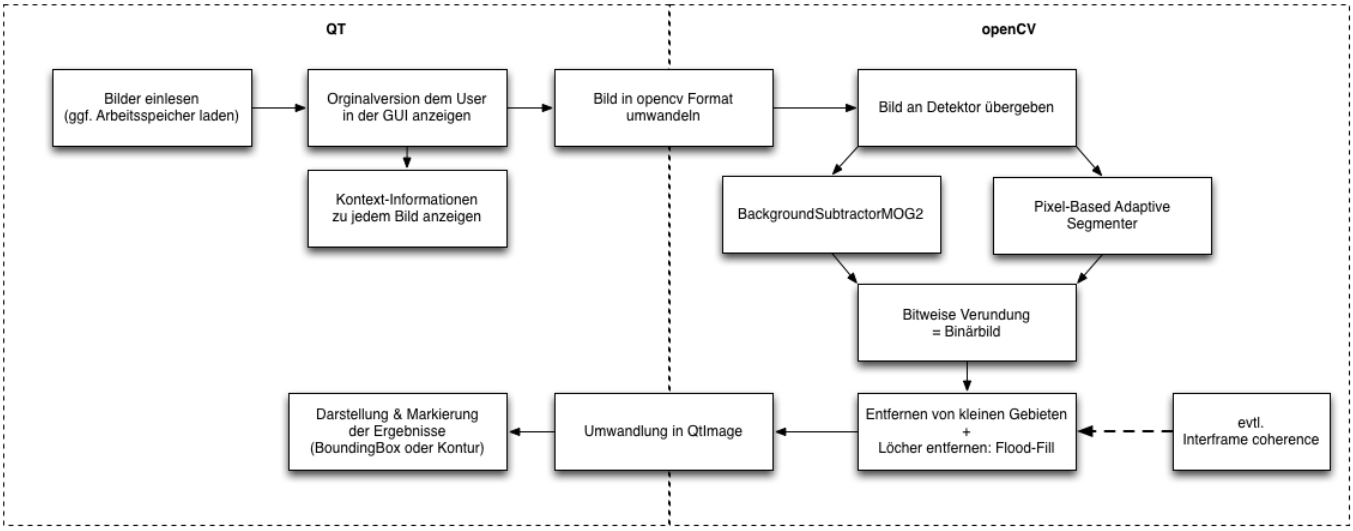

Abb. 1: Übersicht über die Verarbeitungspipeline

Komponenten bilden den Anfang und das Ende der Verarbeitungspipeline.
Das es sich bei den Videosequenzen um eine Reihe von Einzelbildern handelt, werden diese mit dem **cv::imread** eingelesen. Diese Funktion liefert pro Bild eine Matrix zurück. Zur Ausgabe der Ergebnisbilder wird die Bildmatrix in ein **QImage** umgewandelt und in dem Interface angezeigt.

C. Verwendete Segmentierungsalgorithmen

Für die Segmentierung der bewegten Objekte wurden zwei Algorithmen verwendet. Beide Algorithmen werden in den folgenden Punkten genauer erläutert.

1) BackgroundSubtractorMOG2

Zunächst wird das erste Bild der Videosequenz als initiales Hintergrundbild festgelegt.
Jedes weitere Bild der Videosequenz geht gewichtet in das Hintergrundbild ein. Bewegte Objekte berechnen sich hierbei durch:

Hintergrundbild - aktuelles Bild $\rightarrow$ Vordergrundbild

Zoran Zivkovic [**zivkovic2004improved**] beschreibt in seinem Paper die genaue Funktionsweise des Algorithmus.
Für alle getesteten Datensätze lieferten folgende Parameter die besten Ergebnisse:

- history = 800 : Anzahl von Bilder, die in das Hintergrundmodell Einfluss nehmen
- varThreshold = 80 : Distanzmaß, welches angibt wie viel Abstand zwischen zwei Komponenten bestehen muss

Sehr gut funktioniert dieser Algorithmus auf Videosequenzen, in denen der Hintergrund relativ statisch ist.
Störende Bewegungen, wie z.B. Blätterrauschen oder Kamerabewegungen, führen jedoch zu verfälschten Ergebnissen.
Da das Hintergrundbild initial aus dem ersten Bild besteht, verursachen Bewegungen von Objekten, welche sich am Anfang in der Szene befinden, ein verfälschtes Hintergrundbild und wirken sich sehr lange negativ auf die Segmentierung aus.

2) Pixel-Based Adaptive Segmenter

Martin Hofmann et. al [**PBAS**] beschreiben in ihrem Paper einen ganz anderen Ansatz.
Dabei wird nicht wie bei dem MOG2 ein Hintergrund für die ganze Videosequenz erzeugt, sonderen pro Pixel eine History abspeichert. In der History wird gespeichert, wann das Pixel Hintergrund bzw. Vordergrund war.
Außerdem werden Informationen über die benachbarten Pixel gesammelt, welche bei der Entscheidung ob das aktuelle Pixel ein Vordergrund- oder Hintergrundpixel berücksichtigt werden.
Dabei ist es unwahrscheinlich, dass das aktuelle Pixel ein Vordergrundpixel ist, wenn alle benachbarten Pixel Hintergrundpixel darstellen.
Dadurch ist der Algorithmus viel stabiler gegenüber Bildrauschen, da z.B. Pixel, welche ab und zu durch Blätterrauschen überdeckt werden durch die History der Pixel erkannt und entfernt werden.
Da allerdings viel mehr Berechnungen notwendig sind, steigt die Berechnungszeit und auch der Speicheraufwand.
In unserer Variante des PBAS spalten wir das Bild in die 3 Farbkanäle auf und lassen diese jeweils in einem eigenen Thread laufen. Dadurch reduziert sich die Verarbeitungszeit erheblich. Anschließend wird das Ergebnis der 3 Farbkanäle zu einem Ergebnisbild verodert.
Im Gegensatz zum MOG2 arbeit er sehr gut mit dynamischen Hintergründen. Auch gehen Objekte, die sich kurz nicht mehr bewegen, nicht gleich verloren.
Jedoch ist eine schlagartige Bewegungsänderung mit dem PBAS Algorithmus schwer zu segmentieren. Hier liegt der größte Nachteil des Algorithmus.

3) Interframe coherence

Die Interframe-Kohärenz stellt einen einfachen und schellen Algorithmus dar. Hierbei werden bewegte Objekte durch folgende Berechnung detektiert:

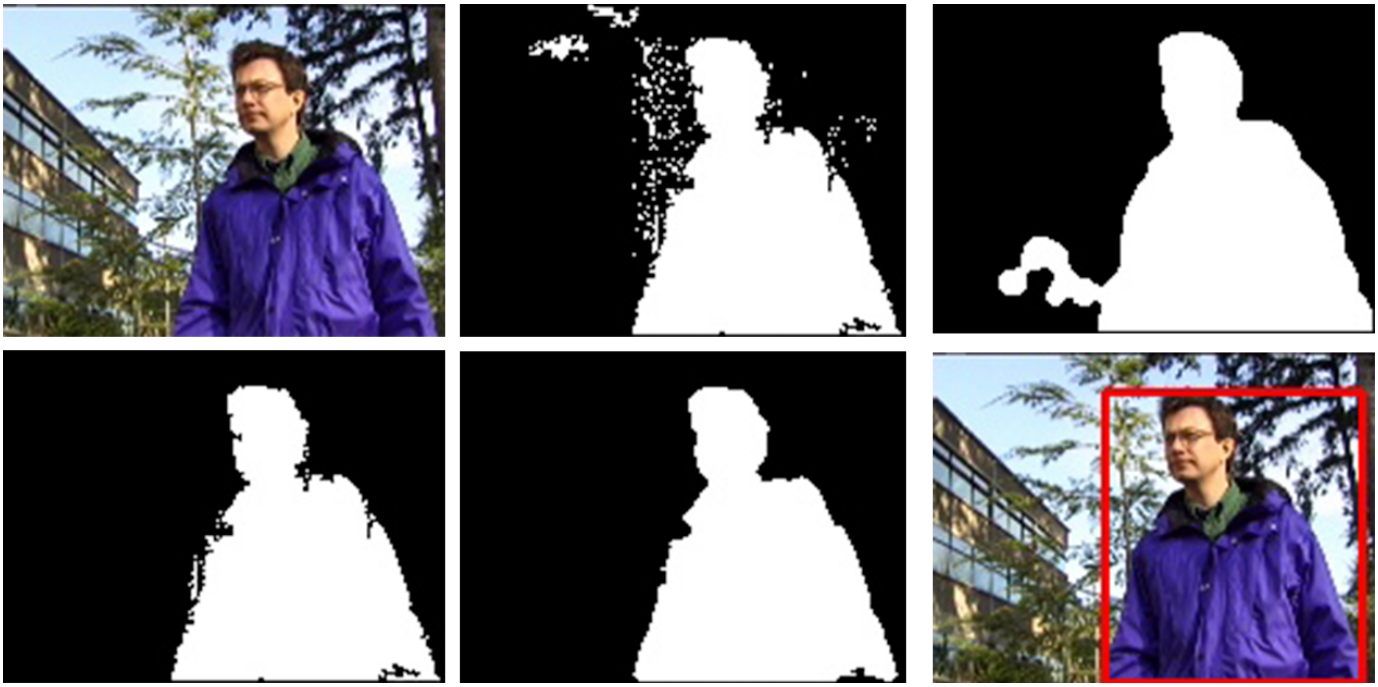

Abb. 2: Originalbild (oben links). Segmentierung durch MOG2 (oben mitte). Segmentierung durch PBAS (oben rechts). Verundung: PBAS∧MOG2 (unten links). Endergebnis Binärbild (unten mitte). Ergebnisbild mit Boundingbox. (unten rechts).

letzes Bild - aktuelles Bild → Vordergrundbild

Dieser Algorithmus kann sehr gut starke Bewegungsänderungen zwischen zwei Frames detektieren, versagt jedoch bei Bildrauschen. Er wird in unserem Ansatz lediglich als unterstützende Funktion genutzt um die Trägheit des PBAS auszugleichen.

D. Die Verarbeitungspipeline

Zunächst werden die Bilder der Sequenz eingeladen. Jedes Bild wird in zwei parallelen Threads jeweils vom PBAS und MOG2 verarbeit.
Außerdem wird das Bild in die einzelenen Farbkanäle aufgespalten und in parallelen Threads vom jeweilgen Algorithmus bearbeitet.
So ergibt sich eine optimale Auslastung aller verfügbaren Kerne der CPU. Das Ergebnis liegt jetzt als Binärbild vor, wobei weiße Pixel Vordergrundobjekte und schwarze Pixel Hintergrundobjekte darstellen. Beide Binärbilder werden bitweise wie folgt verknüpft:

$$f(n,m) = \begin{cases} 1, & \text{wenn PBAS(n,m)=MOG2(n,m)=1} \\ 0, & \text{wenn PBAS(n,m)=MOG2(n,m)=0} \end{cases}$$

Anschließend werden zusammenliegende Pixel zu einer Region verknüpft und die Fläche der Region berechnet. Ist die Fläche zu klein, wird die Region gelöscht. So können kleine Störungen entfernt werden.
Außerdem wird noch ein FloodFill Algorithmus angewandt, welche Löcher innerhalb einen Segmentes schließt. Zu guter letzt werden die Konturen der Segmente in einer Liste aus Punkten und pro Segment eine BoundingBox gespeichert, welche dann auf dem Originalbild angezeigt werden können.

III. Resultate

A. Testdaten

Um zu testen, wie gut ein Algorithmus bewegte Objekte detektiert, ist es notwendig, seine Arbeitsweise auf möglichst unterschiedlichen Testdaten zu untersuchen. Hierfür wurden Bildsequenzen im .jpeg-Format bereitgestellt, die den verschiedenen Frames einer Kamera entsprechen sollen. Der Datensatz, der im Zusammenhang mit dem Paper *A new change detection benchmark dataset [N. Goyette et al]*[**goyette2012changedetection**] erstellt wurde, liefert folgende Unterteilung:

- Baseline: *Einfache Videos*
- Dynamic Background: *Bewegte Objekte vor dynamischen Hintergründen*
- Camera Jitter: *Starke Kamerabewegung*
- Intermittent Object Motion: *Objekte, die sich nur kurzzeitig bewegen oder rasten*
- Shadow: *Harte, Weiche und unterbrochene Schatten*
- Thermal: *Videos einer Wärmebildkamera*

Dabei enthält jeder Bereich mehrere Testdaten, die sich in Größe, Länge und Inhalt unterscheiden.

B. Messwerte

Für die Auswertung werden die Ergebnisbilder mit mitgelieferten, handsegmentierten Bildern verglichen. Dabei entstehen die Qualitätsmerkmale:

- TP : True Positive
- FP : False Positive
- FN : False Negative
- TN : True Negative
- Re (Recall) : TP / (TP + FN)

PBAS	RE	SP	FPR	FNR	PWC	FM	Precision
baseline	0.8543	0.9951	0.0049	0.1457	0.6550	0.7520	0.6717
dynamicBackground	0.8626	0.9968	0.0032	0.1374	0.4020	0.8266	0.8268
intermittentObjectMotion	0.4611	0.9867	0.0133	0.5389	5.3648	0.4842	0.6264
shadow	0.7995	0.9917	0.0083	0.2005	1.4896	0.7802	0.7813
PBAS∧MOG2	RE	SP	FPR	FNR	PWC	FM	Precision
baseline	0.6698	0.9985	0.0015	0.3302	0.5396	0.7467	0.8440
dynamicBackground	0.5559	0.9992	0.0008	0.4441	0.4371	0.6632	0.9488
intermittentObjectMotion	0.2933	0.9976	0.0024	0.7067	5.6217	0.3923	0.7725
shadow	0.5365	0.9965	0.0035	0.4635	2.0693	0.6575	0.8911
PBAS∧MOG2∨IF	RE	SP	FPR	FNR	PWC	FM	Precision
baseline	0.7056	0.9980	0.0020	0.2944	0.5479	0.7516	0.8061
dynamicBackground	0.6159	0.9985	0.0015	0.3841	0.4494	0.6971	0.9155
intermittentObjectMotion	0.3222	0.9970	0.0030	0.6778	5.4434	0.4217	0.7621
shadow	0.5729	0.9955	0.0045	0.4271	2.0366	0.6797	0.8728

Abb. 3: Qualitätsmesswerte der verschiedenen Algorithmen auf den gegebenen Datensätzen

- Sp (Specificity) : TN / (TN + FP)
- FPR (False Positive Rate) : FP / (FP + TN)
- FNR (False Negative Rate) : FN / (TP + FN)
- PWC (Percentage of Wrong Classifications) : 100 * (FN + FP) / (TP + FN + FP + TN)
- F-Measure : (2 * Precision * Recall) / (Precision + Recall)
- Precision : TP / (TP + FP)
- FPR-S : Average False positive rate in hard shadow areas

Für die Abgabe waren jedoch nicht alle Datensätze notwendig, wurden von uns aber für Testzwecke verarbeitet. Diese Werte werden genutzt, um eine Reihenfolge unter den teilnehmenden Algorithmen zu erstellen. Der Vergleich wird von einem ebenfalls bereitgestellten Programm ausgeführt, der die Ergebnisse in einer Textdatei ablegt.

C. Resultate

In der dargestellten Tabelle (*Tabelle* **??**, *S.* **??**) haben wir die Ergebnisse unseres Algorithmus eingetragen.

D. Auswertung

Zuerst betrachten wir die Ergebnisse des *Pixel-Based Adaptive Segmenters*. Auf den ersten Blick liefert dieser Ansatz durchgängig gute Ergebnisse, zeigt aber auf den Datensätzen der intermittent Object Motion deutliche Verluste.
Die Spezifität (SP) bleibt hier zwar weitgehend stabil, dafür ist ein rapider Anstieg beim Prozentsatz der falsch-klassifizierten Pixel, sowie ein starker Einbruch bei der Sensitivität (Re) zu verzeichnen. Dies lässt sich durch die Eigenschaft des PBAS erklären, Pixel eine Hintergrund-/ oder Bewegungswahrscheinlichkeit zu geben, die Abhängig seiner Umgebung ist.
Im Datensatz der intermittent Object Motion nehmen bewegte Objekte teilweise das gesamte Kamerabild ein, sodass alle Pixel als bewegte gekennzeichnet werden und beeinflussen sich daher auch noch lange weiter, wenn das bewegte Objekte verschwunden ist. So noch lange nach dem Verschwinden des Objektes Bewegung detektiert.
Sehr positiv fällt jedoch auf, dass dynamische Hintergrundelemente gut erkannt und somit nicht als Bewegung gesetzt werden.

Die Ergebnisse der Verundung von MOG2 und PBAS sind leider nicht eindeutig als besser oder schlechter einzustufen, wenn man sie mit dem einfachen PBAS vergleicht. So sieht man eine deutliche Verbesserung der Präzision, sowie einen Rückgang der falsch-positiven Pixel. Dem entgegen steht ein starker Verlust von Spezifität und eine erhöhte falsch-negativ Rate.
Diese lassen sich auf das Verhalten des MOG2 zurückführen, der oftmals viele Lücken in zusammenhängenden bewegten Flächen lässt. Die bitweise Verundung sorgt nun dafür, dass einige durch den PBAs korrekt als positiv gekennzeichneten Pixel als negativ angegeben werden. Andererseits hilft die Verundung, um mit kamerafüllenden Objekten umzugehen, da der MOG2 schneller darauf reagiert, wenn ein Hintergrund erscheint.

Der letzte Ansatz, der zusätzlich zur Verundung noch einen Vergleich des aktuellen und des letzten Frames einbringt, liefert in allen Bereichen leider nur mittelmäßige bis schlechte Ergebnisse im Vergleich zu den anderen Ansätzen. Betrachtet man jedoch die Ergebnisse im Speziellen sind diese teilweise sogar besser. Dies gilt es in zukünftigen Arbeiten

einzubeziehen.

E. Laufzeitanalyse

In der folgenden Tabelle ist die durchschnittliche Laufzeit pro Bild der Vidosequenz dargestellt. Vergleichend wurden hierbei die von uns verwendeten Algorithmen gegenübergestellt. Aufgrund der Auslagerung einzelner Verarbeitungsschritte in Threads ist eine Verarbeitung der Bildsequenzen in Echtzeit möglich ($\geq$ 540 FPS).

Bildgröße (320 * 240)	Core 2 Duo (2,8 GHz)	Core i7 2630QM (2,0GHz)
MOG2	0.19s	0.025s
PBAS	0.09s	0.033s
PBAS$\wedge$MOG2	0.105s	0.06s
(PBAS$\wedge$MOG2)$\vee$IF	0.111s	0.045s

IV. Probleme

Abb. 4: Nicht gefundene, zusammenhängende Fläche

Auch wenn unsere Ergebnisse im Vergleich zu den meisten online einsehbaren Algorithmen einen guten Eindruck machen, so sind doch signifikante Schwächen zu sehen. Vor allem die große Zahl der falsch-negativen Werte zeigt, dass beim Versuch, dynamische Hintergrundelemente zu eliminieren, auch gewollte Bewegung ausgeschlossen wurden. Der Ansatz der Interframe-Kohärenz konnte dies zwar teilweise beheben, scheitert dann aber in anderen Bereichen.
Daher ist der letzte Ansatz in dieser Form noch nicht auf alle Datensätze anwendbar und muss angepasst werden.

V. Zusammenfassung und Ausblick

A. Zusammenfassung

Unser Algorithmus nutzt verschiedene Ansätze, um Bewegte Objekte in Videosequenzen zu finden zu zu segmentieren. Dabei haben wir versucht, die Vorteile verschiedener Algorithmen zu kombinieren, um dynamische Hintergründe, sowie unterbrochene Bewegungen korrekt zu detektieren.

Der einfache Background Subtractor MOG2 liefert gute Ergebnisse, auch bei Objekten, die lange im Hintergrund verharren, deklariert jedoch auch dynamische Hintergrundelemente als bewegte Objekte.
Der Pixel-Based Adaptive Segmenter hingegen unterdrückt diese dynamischen Elemente sehr gut und findet auch zusammenhängende Flächen, da die Nachbarinformationen sowie die History einzelner Pixel betrachtet werden.
Um beide Vorteile zu nutzen haben wir daher eine bitweise Verundung der Ergebnisbilder durchgeführt. Diese führt jedoch zu einer starken Erhöhung der falsch-negativen Pixel, sodass wir zusätzlich eine Interframe-Koheränz nutzen, um entstandene Lücken wieder zu füllen.

B. Future Work

Die berrechneten Ergebnisdaten liefern bisher zwar gute, aber nicht optimale Resultate, sodass eine Weiterenwicklung der Algorithmen wünschenswert ist Dabei wird erst einmal von kleinen Erweiterungen wie das Erkennen von Schatten abgesehen.
Da die einzelnen Algorithmen jeweils Vor-und Nachteile für verschiedene Qualitätsmesswerte besitzen, gilt es, die einzelnen Stärken genauer zu isolieren und auf den Datensatz angepasst zu verwenden. Dabei steht vor allem die Entwicklung einer Steuereinheit im Vordergrund, die für den jeweiligen Datensatz oder gar Bereich im Bild entscheidet, welcher Algorithmus verwendet werden soll, um optimal auf das aktuelle Geschehen zu reagieren.
Dabei wäre es denkbar, Unruhe für ein Bild oder Bereiche eines Bildes zu berrechnen um dann auf den jeweiligen Bereich zu reagieren. Auch die Vorverarbeitung bietet Raum für Verbesserungen. Weitere Testdaten, die wir zusätzlich getestet haben zeigen beispielsweise eine große Schwäche unserer Algorithmen bei wackelnden Kamerabildern. Hier würde eine Bildstabiliserung helfen, die vor der Bewegungsdetektion einsetzt.
Nicht zuletzt würde eine Tracking-Funktion rastende Objekte speichern, um sie so nicht in den Hintergrund aufnehmen zu lassen.