Institut für Visualisierung und Interaktive Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2288

Bewegungserkennung mithilfe von Rechteckmerkmalen

Rudolf Netzel

Studiengang:	Informatik
Prüfer:	Prof. Dr. Gunther Heidemann
Betreuer:	Dipl.-Inf. Benjamin Höferlin
begonnen am:	22. Juni 2010
beendet am:	22. Dezember 2010
CR-Klassifikation:	I.5.2, I.5.1

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Die Nachfrage nach Systemen, die in der Lage sind bestimmte Aktionen in Bildsequenzen aufzufinden, steigt. Bereits heute gibt es Bereiche, in denen solche Systeme zum Einsatz kommen, bzw. in denen es von Nutzen wäre diese zum Einsatz zu bringen. Ein Beispiel hierfür ist die Analyse von Videostreams, bei denen beispielsweise Bewegungen von Menschen erkannt und klassifiziert werden können. Sei es nun bei der Überwachung von Patienten in Krankenhäusern, der Personenüberwachung durch diverse Institutionen oder zur Verbesserung der Mensch- Maschine- Interaktion.
Die Vorstufe hierzu bildeten Forschungsergebnisse im Bereich der Objekterkennung in Bildern, durch die leistungsfähige Methoden hierfür entwickelt werden konnten. Die Erfahrungen, die dabei gesammelt wurden, sind nun im Bereich der Aktionenerkennung von großem Nutzen. Einige der Vorgehensweisen konnten erfolgreich adaptiert werden, um auch hier erstaunliche Erfolge zu erzielen. Diese Entwicklung wird heutzutage durch Plattformen wie YouTube oder die Filmindustrie stark unterstützt. Denn diese liefern eine Vielzahl an heterogenen Videos mit verschiedensten Aktionen in unterschiedlichen Umgebungen. Diese Videos können dann als Test- und Trainingsmenge fungieren, wodurch man dem zeitraubenden Akt der Erstellung solcher Videos entgehen kann.
Die im Laufe der Zeit entstanden Methoden im Bereich der Bewegungserkennung können in zwei Klassen aufgeteilt werden. Zum einen in modellbasierte und zum anderen in anzeichenbasierte Methoden. Bei Ersterem wird, entsprechend der Bezeichnung, ein Modell der Bewegung erstellt. Dies könnte zum Beispiel die Silhouette eines Menschen sein. Anhand von diesem Modell werden nun Videoszenen untersucht. Falls die Szene mit dem Modell übereinstimmt, so ist es wahrscheinlich die gesuchte Bewegung. Auf diese Weise ist es auch möglich Körperteile aufzufinden. Durch die Verfolgung ihrer Trajektorien kann dann die Art der Bewegung klassifiziert werden.
Bei den anzeichenbasierten Verfahren werden lokale Merkmale zur Klassifikation genutzt. Falls bestimmte Kombinationen dieser Merkmale vorliegen, so deutet dies auf die Präsenz einer bestimmten Aktion hin. Hier teilen sich die Methoden wiederum in jene auf, welche starke lokale Merkmale nutzen und andere die schwache lokale Merkmale verwenden. Stark bedeutet, dass diese Merkmale beispielsweise eine Invarianz in ihrer Skalierung aufweisen. Solche Merkmale liefert die „Scale Invariant Feature Transform“ (SIFT). Bei den schwachen Merkmalen handelt es sich um Merkmale die beispielsweise Kanten detektieren. Hier gilt auch, dass eines dieser Merkmale nicht für eine Klassifikation ausreicht, jedoch stiegt die Wahrscheinlichkeit einer richtigen Detektion durch die Kombination mehrerer dieser Merkmale.

Für welche Verfahrensklasse oder Merkmale man sich auch immer entscheidet, es wird immer Punkte geben, welche die Klassifikation erschweren und die berücksichtigt werden sollten.
Diese wären:

- Individualität des Menschen
  - Bewegungscharakteristika
  - Kleidung
  - Aussehen
  - Haltung
- Perspektive und Bewegung der Kamera
- Hintergrund
- Helligkeitsänderungen
- Rauschen

Ziel dieser Arbeit ist es, ein Verfahren zu implementieren, das schwache Merkmale (zweidimensionale Rechteckmerkmale) nutzt und dessen Leistungsfähigkeit mit Verfahren zu vergleichen, welche komplexere (volumetrische) Merkmale gebrauchen.
Nun stellt sich die Frage: Warum sollte man den überhaupt schwache Merkmale verwenden? Die Antwort darauf liegt in der Performanz der Berechnung. Schwächere Merkmale lassen sich deutlich schneller berechnen, wodurch auch die Detektionsgeschwindigkeit steigen würde.
Vielversprechende Verfahren nutzen heutzutage Merkmale, die auf interest-points (starken lokalen Merkmalen) beruhen und sind daher je nach Implementierung mehr oder weniger schnell.
Der Versuch der Aktionenerkennung mithilfe von schwachen Merkmalen wurde motiviert durch die Ergebnisse der Arbeit von Paul Viola und Michael Jones, die unter Verwendung zweidimensionaler Rechteckmerkmale sehr gute Ergebnisse bei der Objektserkennung in Bildern vorweisen können.

2. Grundlagen

2.1. Objekterkennung mit Rechteckmerkmalen

Paul Viola und Michael Jones veröffentlichten 2001 mit ihrer Arbeit „Robust Real-time Object Detection“ [VJ02] eine Methode zur Objekterkennung. Ihr Ziel war es zum einen eine hohe Detektionsrate zu erzielen und zum anderen den Erkennungsprozess sehr schnell durchzuführen. Vier Schwerpunkte stechen in Ihrer Arbeit dabei hervor.

- die Verwendung von Rechteckmerkmalen
- die Form der Bildrepräsentation als Integral-Image
- die Konstruktionsmethode einer Entscheidungskaskade
- das Arrangement einer Vielzahl an Klassifikatoren in einer Kaskade

Der Entschluss das Rahmenwerk merkmalsbasiert statt pixelbasiert zu gestalten, wurde überwiegend durch zwei Gründe motiviert. Zum einen ist es damit möglich Wissen über die Domäne ad-hoc zu kodieren, und zum anderen ist es schneller. Die verschiedenen Merkmalstypen sind in Abbildung 2.1 dargestellt. Duch diese Merkmale ist es möglich bestimme Strukturen zu erkennen. Hier werden mit ihnen mögliche Kanten detektiert.

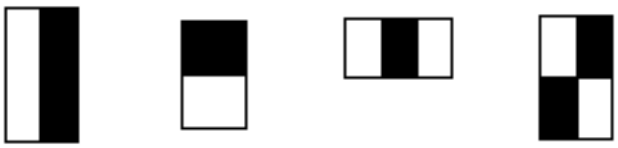


Abbildung 2.1.: Basis aus Rechteckmerkmalen von Viola u. Jones

Dabei wird der Wert eines Merkmals durch die Differenz der Grauwertsummen der schwarzen und weißen Rechteckflächen bestimmt. Dafür kann eine Methode genutzt werden, welche die entsprechenden Flächen schneller berechnet, als dies durch eine Summation der einzelnen Pixelwerte möglich wäre.
An dieser Stelle kommen wir nun zu dem zweiten Schwerpunkt, der Bildrepräsentation

durch ein Integral-Image. Dabei handelt es sich um eine Struktur, mit der die Auswertung eines Bildes beschleunigt werden soll.

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x',y')$$

$ii(x,y)$ entspricht einem Flächeninhalt, welcher der Summation der Pixelwerte eines Bereichs gleichkommt. Der markierte Bereich in Abbildung 2.2 entspricht also dem Flächeninhalt von $ii(Punkt\_1)$. Wobei $i(x',y')$ für den Wert des Pixels (x',y') steht.

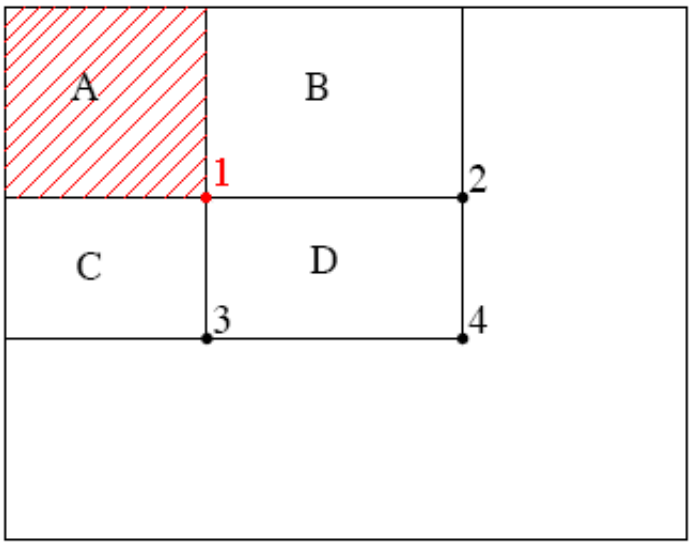


Abbildung 2.2.: Flächenberechnung mit Hilfe des Integral-Image. A-D kennzeichnen Rechtecke, die durch den Punkt $(0,0)$ und einen der angegebenen Punkte $1-4$ gebildet werden. A = $[(0,0); Punkt\_1]$ B = $[(0,0); Punkt\_2]$ C = $[(0,0); Punkt\_3]$ D = $[(0,0); Punkt\_4]$

Die Generierung eines Integral-Image kann, wie im folgenden gezeigt wird, in Linearzeit, zu der Anzahl der Pixel, vollzogen werden. Berechnet werden kann $ii(x,y)$ dann mithilfe folgernder Rekursionsgleichung:

$$\begin{aligned} s(x,y) &= s(x,y-1) + i(x,y) \\ ii(x,y) &= ii(x-1,y) + s(x,y) \\ s(x,-1) &= 0 \\ ii(-1,y) &= 0 \end{aligned}$$

Dadurch ist es nun möglich mit nur vier Zugriffen den Inhalt einer Fläche zu berechnen.

$$A_{1-4} = D - B - C + A$$
$$A_{1-4} = ii(4) - ii(2) - ii(4) + ii(1)$$

Nachdem es nun möglich ist, die verwendeten Merkmale effizient auszuwerten, kann nun ein Klassifikator trainiert werden. Dieser besteht aus einer Kombination verschiedener schwacher Klassifikatoren. Jedem dieser Klassifikatoren ist dabei genau ein Merkmal zugeordnet. Anhand eines Schwellwertes und der Auswertung eines Merkmals kann eine binär Klassifikation erfolgen.
Für die Erstellung eines kombinierten Klassifikators verwenden Viola und Jones AdaBoost.

2.2. Boosting Verfahren

Dabei handelt es sich um Verfahren, welches die Klassifikationsleistung eines Lernalgorithmus verbessern sollen, indem eine Kombination an schwachen Klassifikatoren einen stärkeren bilden.

2.2.1. AdaBoost

Gegeben sei dabei eine Menge an Beispielen $(X_1, Y_1) \dots (X_i, Y_i)$ mit jeweils einem Initialgewicht. X_i ist dabei das $i'te$ Trainingsbeispiel und $Y_i \in 0, 1$ gibt an, ob es negativ (0) oder positiv (1) ist. Die h_j kennzeichen einen schwachen binären Klassifikator

Schritt 1: Normierung der Gewichte

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

Schritt 2: Für jedes Merkmal j wird ein Klassifikator h_j trainiert

Schritt 3: Finden des Merkmals h_i dessen Klassifikationsfehler e_i minimal ist.

$$e_t = \sum_i w_j \left| h_t(x_i) - Y_i \right|$$

Schritt 4: Neugewichten der Beispiele

$$w_{t+1,i} = w_{t,i} * \beta_i^{1-e_i}$$

$$\beta_i = \frac{e_i}{1-e_i}$$

Schritt 5: Gewicht des Merkmals h_i berechnen

$$\alpha_i = log\frac{1}{\beta_i}$$

Dies wird T mal wiederholt, und man erhält somit einen stärkeren Klassifikator $h(X)$.

$$h(X) = (\sum_{i=1}^{T} \alpha_i h_i(x)) \geq (\frac{1}{2}\sum_{i=1}^{T} \alpha_i)$$

Daraus wird ersichtlich, je mehr Merkmale hinzugefügt werden, umso besser ist das Ergebnis. Gleichzeitig steigt jedoch auch der Aufwand der Auswertung. Dieses Dilemma lässt sich nun dadurch lösen, indem man die Anzahl der Merkmale pro Klassifikator einschränkt und damit eine Fehlklassifikation erlaubt. Dafür werden jedoch mehrere Klassifikatoren erzeugt, welche in einer Kaskade angeordnet werden. Dies ist exemplatisch in Abbildung 2.3 dargestellt. Die Aufgabe eines jeden Knotens ist es nun, eine Teilmenge der negativen Beispiele auszusondern, sodass am Ende der Kaskade möglichst nur noch positive Beispiele herausfallen.

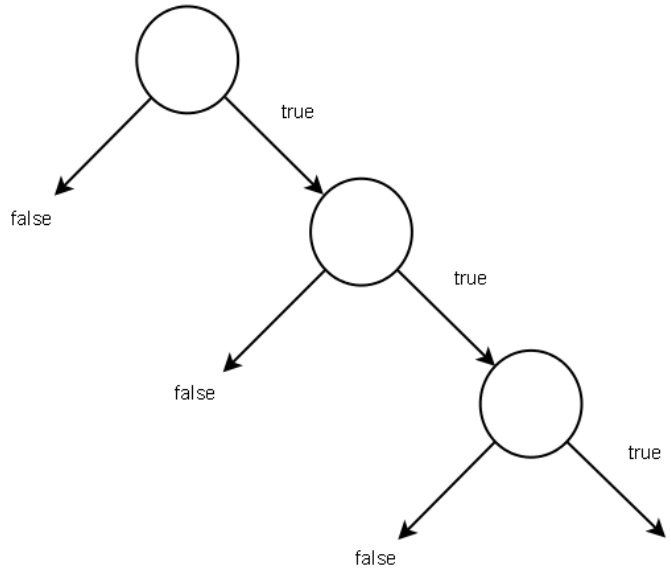


Abbildung 2.3.: Kaskade an Klassifikatoren

2.2.2. Direct-Feature-Selection

Die Direct-Feature-Selection, welche in der Arbeit „Learing a Rare Event Detection Cascade by Direct Feature Selection“ von Jiynxin Wu, James M.Mehg und Matthew D.Mullin

[WRM03] vorgestellt wird, ist eine Methode einen Klassifikator, respektive einen Knoten, einer Entscheidungskaskade zu generieren. Der Vorteil dieser Methode liegt im Vergleich zu AdaBoost darin, dass die Zeit zur Generierung eines Knotens drastisch reduziert wurde, und damit die Erzeugung der gesamten Kaskade deutlich beschleunigt wird. Ihre Untersuchungen haben ergeben, dass die Direct-Feature-Selection c.a. 100 mal schneller ist.
Dies liegt unter anderem daran, dass AdaBoost die Gewichte der Trainingsmenge in jedem Iterationsschritt i neu setzt und damit die Klassifikatoren der Merkmale im Schritt $i+1$ neu trainiert werden müssen (siehe Schritt 2 und Schritt 4 des AdaBoost-Algorithmus).

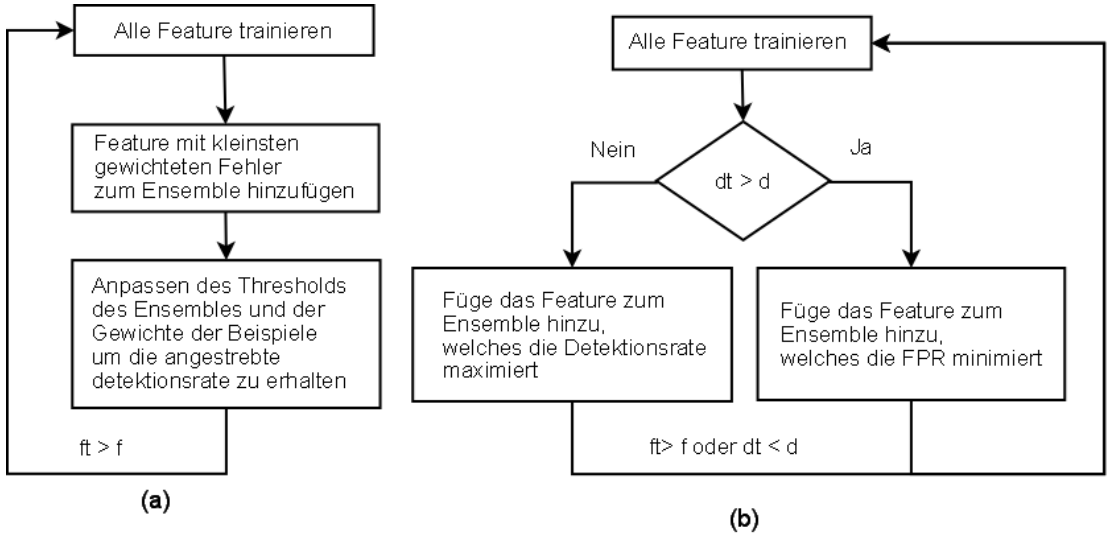


Abbildung 2.4.: Ablauf der Knotengenerierung (a)AdaBoost (b)Direct-Feature-Selection
d entspricht der minimalen angestrebten Erkennungsrate des Knotens und f der maximalen Falschdetektionsrate
d_t und f_t entsprechen der Detektions bzw. Falschdetektionsrate im Iterationsschritt i

Hier ist dies nicht der Fall. Die Merkmale werden einmalig auf Grundlage einer positiven Menge P und einer negativen Menge N vor der Knotengenerierung trainiert. Dies bedeutet, dass für jedes Merkmal ein optimaler Schwellwert auf Grundlage von P und N gesucht wird, sodass seine Falschdetektionsrate kleiner oder gleich dem vorgegebenen Wert f ist.
Sowohl bei Viola und Jones, als auch bei der Direct-Feature-Selection ändert sich die Menge N nach der Erstellung des Kontens, indem die Elemente entfernt werden, welche durch den Knoten korrekt eingestuft wurden. N wird danach mit Elementen aufgefüllt, die von der bisher erzeugten Kaskade als korrekt klassifiziert werden, jedoch negativ sind. Dies geschieht, damit P und N die gleiche Anzahl an Elementen besitzen.
Die Selektion der Merkmale, die dem Ensemble des Knotens hinzugefügt werden, verläuft nach Abbildung 2.4 (b).
Solange die Detektionsrate d_t nicht größer ist als die Vorgabe d und die Fehlerrate nicht

kleiner als die der Vorgabe F, wird ein Merkmal hinzugefügt. Falls $d_t < d$ wird ein Merkmal hinzugefügt, welches d_{t+1} maximiert. Wenn $d_t \geq d$ wird ein Merkmal hinzugefügt, das f_{t+1} minimiert.
Nachdem das Ensemble erstellt wurde, werden nun noch die Gewichte der einzelnen Merkmale bestimmt. Dies geschieht nach dem gleichen Prinzip wie bei AdaBoost. Zu Letzt wird der Schwellwert des Knotens ermittelt. Er wird so eingestellt, dass der Knoten die Erkennungsrate d erreicht.

2.3. Blockmatching

Unter Blockmatching versteht man die Zuordnung eines Bildblocks in Frame j zu einm Bildblock in Frame i (o.B.d.A $i > j$), unter Verwendung eines Abstandsmaßes. Hierfür wird in den meisten Verfahren eine Art quadratischer Fehler der Pixelintensitäten der zu testenden Bereiche ermittelt. Ein Einsatzgebiet des Block-Matchings ist die Verfolgung von Trajektorien. Im laufe der Zeit sind einige Block-Matching-Algorithmen (BMA), wie zum Beispiel der

- Full Search
- Cross Search
- oder der Diamond Search

entstanden.
Der Unterschied dieser Algorithmen besteht hauptsächlich in der Reihenfolge, in der Blöcke miteinander verglichen werden.
Beim Full Search werden einfach alle umliegenden Blöcke in einem Suchraster mit dem ursprünglichen Block verglichen. Danach wird das Suchraster in den Punkt verschoben, an dem der geringste Fehler errechnet wurde. An diesem Punkt wiedeholt sich das Ganze. Dies wird solange fortgesetzt, bis das Zielkriterium erfüllt ist.
Der Cross Search (Abbildung 2.5) hingegen testet vier mögliche Blöcke die in einer X-Formation angeordnet sind. Das Suchraster wird dann an dem Punkt mit dem geringsten Fehler neu gesetzt, und die Breite des Suchrasters halbiert. Sobald die Höhe bzw. Breite Null beträgt, endet die Suche.

Wohingegen der Diamond Search (Abbildung 2.6) zwei Suchmuster nutzt. Das „Large Diamond Search Pattern“ (LDSP) und das „Small Diamond Search Pattern“ (SDSP).

Zuerst wird das LDSP verwendet. Dabei werden die Blöcke an jedem Punkt des Gitters ausgewertet und dann mit dem ursprünglichen Block verglichen. Das Zentrum des LDSP wird dann in den Punkt verlegt, indem der errechnete Fehler minimal ist. Falls dies das Zentrum selbst sein sollte, so wechselt man zum SDSP. Der vom SDSP ermittelte Punkt ist nun der, an dem die beste Übereinstimmung gefunden wurde.

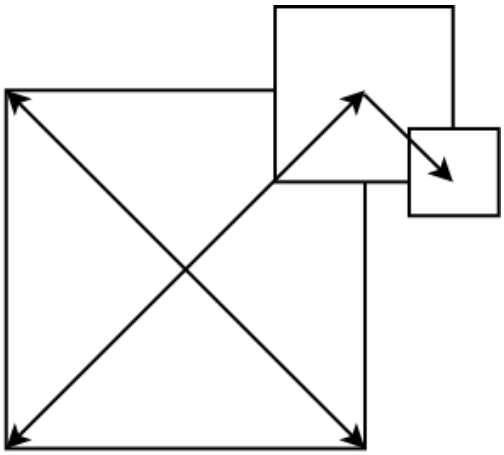


Abbildung 2.5.: Cross Search Suchablauf

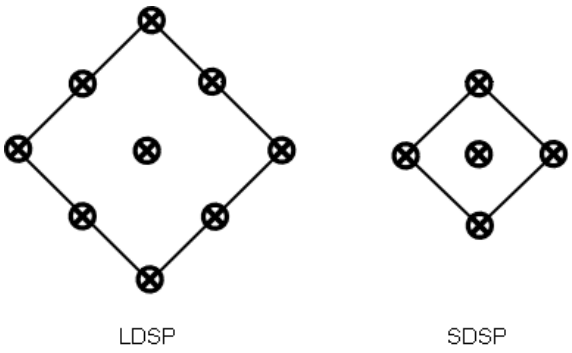


Abbildung 2.6.: Diamond Search Suchraster

3. Related Work

In diesem Kapitel werden einige der vielzähligen Arbeiten im Bereich der Aktionenerkennung näher erläutert. Dabei werden drei unterschiedliche Typen an Merkmalen verwendet.

- Volumetrische Merkmale in Verbindung mit Histogrammen räumlicher Gradienten
- Interest Points in Verbindung mit Visual Words
- Dreidimensionale Rechteckmerkmale

2007 veröffentlichten Ivan Laptev und Patrick Pérez in ihrer Arbeit „Retrieving actions in movies“ [LP07] ein Verfahren um Ereignisse in Videos aufzufinden. Der erste Schritt besteht in der Annotation der Videodaten. Hierbei wird jedem Video ein sogenanntes Keyframe zugewiesen. Dieses beschreibt den Zeitpunkt an dem ein Ereignis eintritt. Bei der Handlung „Trinken“ wäre dies beispielsweise das Frame, indem das Trinkgefäß den Mund berührt. Nun wird ein Klassifikator trainiert, der diese Keyframes erkennt. Das alleine reicht jedoch nicht aus um die dazugehörige Bewegung zu erkennen. Daher werden jetzt noch Informationen aus vorherigen und folgenden Frames hinzugezogen. Hierfür wurden quaderförmige Bereiche um das Keyframe herum gelernt, welche in Kombination die Bewegung als Ganzes erfassen können.
Das Training erfolg auf den Histogrammen der räumlichen Gradienten der quaderförmigen Bereiche.
Der Test, ob eine Bewegung in einem Video vorkommt, läuft wie folgt ab. Das Video wird Frame für Frame eingelesen und mithilfe des Keyframe-Classifier wird entschieden ob es sich um ein Keyframe handeln könnte. Sofern dies der Fall ist, wird mit Hilfe der Histogramm-Blöcke überprüft, ob gelernte Bewegungskriteria erfüllt werden. Damit wurden Detektionsraten von bis zu 90% erreicht bei einer Fehldetektionsrate von bis zu 9%.

Eine weitere Arbeit, mit dem Titel „Learning realistic human actions from movies“ [LMSR08], erschien 2008. Hierbei werden zwei Schwerpunkte gesetzt. Der eine beinhaltet das automatisierte Auffinden von Aktionen in Videos, und der Andere das Erlernen dieser. Zu Beginn wird der Zeitraum ermittelt, indem die Aktion stattfindet. Dies geschieht unter Zuhilfenahme von Untertitelinformationen und dem Drehbuch eines entsprechenden Films. Durch die Untertitel erhält man nun zeitliche Informationen über Dialoge, während das Drehbuch nicht nur die Dialoge enthält, sondern auch Angaben zu den Aktionen enthält, die währenddessen

ausgeführt werden. Somit lässt sich der exakte Zeitraum einer Aktion ermitteln. Ein Beispiel hierfür ist in Abbildung 3.1 dargestellt.

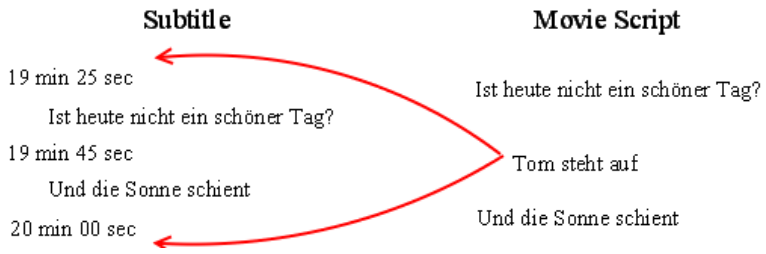


Abbildung 3.1.: Zeitspanne einer Aktion

In dieser Zeitspanne werden nun Positionen raumzeitlich Merkmale in verschiedenen raumzeitlichen Skalierungen berechnet. An jedem dieser Punkte wird dann ein Volumen aufgespannt, dessen Größe abhängig von der Skalierungsstufe ist, in der das Merkmal detektiert wurde. Jedes dieser Volumen wird nun in ein Gitter an Quader aufgeteilt und von jedem Quader wird nun ein Histogramm der Gradienten sowie des optischen Verlaufts erstellt. Durch die Konkatenation der normalisierten Gradientenhistogramme erhält man einen Beschreibungsvektor der Gradienten für das gesamte Volumen, sowie einen Vektor für den optischen Fluss. Nun wird auf Grundlage von hunderten an Merkmalsvektoren, die zu einer Aktion gehören, ein bag-of-feature erstellt. Dies bedeutet es werden visuelle Wörter extrahiert, indem ein k-means Clustering durchgeführt wird. Die entstanden Clusterzentren entsprechen nun den visuellen Wörtern.
Nun ist es möglich ein Histogramm der visuellen Wörter einer Szene zu erstellen, indem die Merkmalsvektoren generiert und dann dem Clusterzentrum zugeordnet werden, zudem die Distanz am geringsten ist. Die Klassifikation erfolgt hier mittels einer nichtlinearen Support-Vektor-Maschine. Auf Grundlage des KTH Datasets, konnte so eine Detektionsrate von c.a 90% erzielt werden.

Ein weiterer Ansatz, der nicht auf starken lokalen Merkmalen beruht, wird in der Arbeit „Efficient Visual Event Detection using Volumetric Features“ [KSH05] dargestellt. In dieser Methode werden volumetrische Merkmale verwendet. Diese sind kubisch und beinhalten positive und negative Breichstypen. Jeder Bereich beinhalten beispielsweise das Integral der Pixelwerte oder des optischen Flusses. Um ein Gesamtmerkmal zu erhalten, werden zuerst jeweils die Integrale alle positiven und negativen Bereiche addiert. Danach wird die Differenz der Integrale der Bereichstypen gebildet. Der Vorteil dieses Merkmalstypus beruht, unter Zugrundelegung eines Integral-Videos, darauf, dass die Auswertung schnell und einfach mit nur acht Zugriffen vollzogen werden kann. Das Integral-Video ist die zeitliche Erweiterung des Integral-Image, welches von Viola und Jones beschrieben wird.

$$iv(x,y) = \sum_{x' \leq x} \sum_{y' \leq y} \sum_{t' \leq t} i(x',y',t')$$

Berechent wird das Integral-Video rekursiv in Linearzeit in bezug auf die Anzahl der Pixel.

$$s_1(x,y,t) = s_1(x,y-1,t) + i(x,y,t)$$
$$s_2(x,y,t) = s_2(x-1,y,t) + s_1(x,y,t)$$
$$iv(x,y,t) = iv(x,y,t-1) + s_2(x,y,t)$$

Um nun eine Aktion in einem Video aufzufinden, wird das Integral-Video sowohl räumlich als auch zeitlich, an verschiedenen Stellen mithilfe eines Detektor ausgewertet. Dabei beinhaltet der Detektor eine Entscheidungskaskade, welche die Merkmalskombinationen enthält, mit denen eine Aktion mit hoher Wahrscheinlichkeit erkannt werden kann. Um die Kaskade zu erstellen wird hier, anders als bei Viola und Jones, die Direct-Feature-Selection anstatt AdaBoost verwendet.
Auf diese Weise wurden folgende Ergebnisse erzielt:

Aktion	Trefferquote in %	Falschdetektionsrate pro Minute
Sit-down	86	0.0
Stand-up	90	0.6
close-laptop	78	0.5
grab-cup	92	0,3

Tabelle 3.1.: „Efficient Visual Event Detection using Volumetric Features“ Testresultate

Die Testvideos hatten dabei eine Größe von 160x120 Pixeln. Des weiteren wurde die Falschdetektionsrate ermittelt, indem der jeweilige Detektor auf je 40 Minuten Film angesetzt wurde, indem die Aktion selbst nicht vorkam.

4. Volumetische Ansatz

Als Vorlage für den volumetrischen Ansatz dient die Arbeit von Yan Ke, Rahul Sukthankar und Martial Hebert [KSH05]. Darauf basierend ist das in Abbildung 4.2 dargestellte System entstanden. Bei dem die einzelnen Komponenten die Fassetten der Originalarbeit abdecken. Darunter Klassen und Funktionen, die Videodaten in ein geeignetes Format konvertieren, bei dem es sich um ein Integral-Video handelt. Des weiteren wurde ein Detektor implementiert, der hier gleichzeitig dem Suchfenster entspricht, mit dem ein zu testendes Video abgetastet wird. Ebenso wurde eine Vererbungsstruktur für die verwendeten Merkmale angelegt. Auch für die Erstellung der Kaskade wurden Strukturen erstellt. Eine besondere Herausforderung stellt die Erzeugung von geeigneten Beispielen dar. Dabei soll zum einen die Menge der positiven Beispiele, auf Grundlage der Vorhanden, vergrößert werden, um zum anderen sollen automatisiert negativ Beispiele generiert werden. Desweiteren wurden zwei Methoden der Schwellwertanpassung getestet, von denen die ausgewählt wurde, mit der die besseren Resultate erzielt wurden. Die eine Methode die Schwellwerte zu erlernen entspricht der, die bei der Direkt-Feature-Selection verwendet wird. Die andere optimiert die Schwellwerte mithilfe einer ROC-Kurve.

4.1. Ablauf des Trainings

Das Training beginnt mit der Erstellung des Detektors. Dazu werden zunächst die Detektorparametern aus einer Datei geladen. Danach werden Merkmale generiert, die sich innerhalb des Detektorvolumens befinden.
Hierfür wird zunächst eine Punktwolke erstellt, welche die Punkte enthält an denen Merkmale positioniert werden dürfen. Diese Punkte sind äquidistant, anhand der minimalen Ausmaße eines Volumen-Feature, im Volumen des Detektors angeordnet. Dann wird an jedem Punkt jeder Merkmalstyp, in jeder möglichen Skalierung, ausgehend von der minimal Größe, erstellt. Jede Dimension wird dabei einzeln mit dem Faktor $1,5$ skaliert. Ausgehend von einem Merkmal mit der Maßen $4x4x4$ erhält man so beispielsweise Merkmale der Form $6x4x4$, $9x4x4$ oder $4x6x4$ usw.
Tests haben gezeigt, dass unter Verwendung von Merkmalen mit ungerader Höhe, Breite und Tiefe bessere Ergebnisse erzielt werden können, da dann die beiden Teilvolumina der Merkmale gleich groß sind.
Als nächstes wird ein Videolisthander erstellt. Dieser lädt die vom Benutzter spezifizierte

Liste an Beispielen. Diese kann positive und negative Elemente enthalten. Die negativen Elemente dienen dazu dem Klassifikator nur auf eine Aktion zu trainieren. Falls er auf „Winken“ trainiert werden soll, so kennzeichnen die negativen Elemente beispielsweise „Gehen“, „Rennen“ usw. Jedes Tupel der Liste das geladen wurde, wird als ein „Original“ gekennzeichnet. Dies ist wichtig, da jedes dieser Originale vervielfältigt wird. Dabei werden sie nicht nur kopiert, sondern jedes einzelne wird in seiner Größe und Position verändert, wobei die Abweichung von dem Original nur wenige Pixel beträgt. Dies geschieht, um die Robustheit des Verfahrens gegenüber einer wackelnden Kammera oder Rauschen zu erhöhen.
Mit der bisherigen Menge an negativen Beispielen könnte der Klassifikator jedoch fälschlicherweise einen Bereich im Hintergrund als positiv erkennen. Um dies zu vermeiden, werden nun zusätzlich Negativbeispiele erzeugt, die sich im Hintergrund befinden. Dabei ist eine Teilüberdeckung der Originalbeispielen erlaubt, denn Tests haben gezeigt, dass die Erkennung dadurch deutlich verbessert wird. Hiernach folgt die Erzeugung der LookUpTable.
Nun kann die Generierung der Kaskade beginnen. Folgende Parameter werden dafür benötigt:

- Die maximale False-Positiv-Rate F_{gesamt} der Kaskade.
- Die maximale False-Positiv-Rate f eines Kotens.
- Die minimale Detection-Rate d eines Knotens.

Es werden zwei Mengen P und N gebildet. P enthält alle positiven Beispiele, während N ebenso viele negative enthält. Mit diesen Mengen wird ein Knoten mithilfe der Direkt-Feature-Selection erzeugt. Jedoch erfolgt das Training der Merkmalsschwellwerte hier anhand einer ROC-Kurve. Diese wird erstellt, indem für äquidistant verteilte Schwellwerte jeweils die Detektionsrate und die Falschdetektionsrate eines Merkmals berechnet werden. Dann wird der Punkt auf der ROC-Kurve mit dem geringsten Anstand zum Punkt $(0,1)$ gesucht. Der entsprechende Schwellwert ist nun der Beste unter den gegebenen Mengen P und N. Abbildung 4.1 stellt dies exemplarisch dar.
Nach der Erzeugung eines Knotens, werden alle Elemente aus N entfernt, die dieser Knoten korrekt als falsch einstuft. Die verbleibenden Elemente wurden sowohl von dem aktuellen Knoten, als auch von allen vorherigen noch nicht als falsch erkannt und bleiben daher in N bestehen. Sie werden für die Generierung des nächsten Knotens verwendet. Zusätzlich wird N mit negativen Beispielen aufgefüllt, welche durch die bisher erzeugte Kaskade als positiv erkannt werden. Vor dem nächsten Iterationsschritt wird die neue Falschdetektionsrate der Kaskade ermittelt. Denn falls diese nun kleiner als F_{gesamt} sein sollte, bricht die Erzeugung ab.
Die Kaskade wird dann exportiert. Dabei werden zum einen die Schwellwerte der Knoten gesichert und zum anderen die Liste der Feature-ID’s, die die Knoten beinhaltet. Ebenso werden alle Merkmale des Detektors exportiert, welche bei der Konstruktion der Kaskade einem Knoten zugeordnet wurden. Für jedes Merkmal werden alle relevanten Informationen gespeichert.

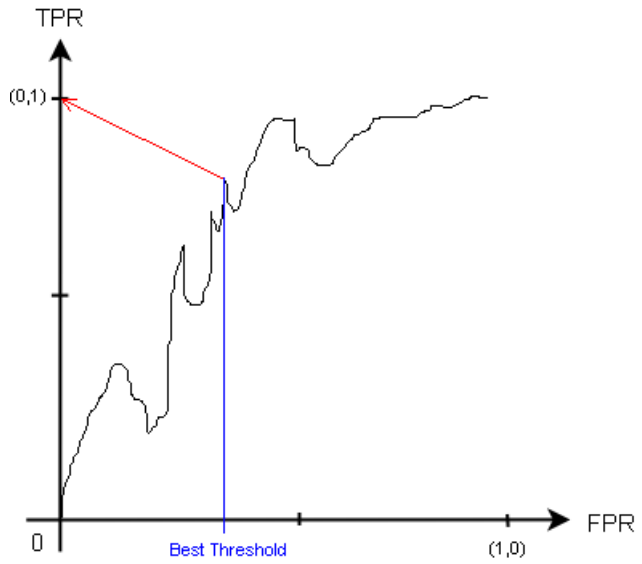


Abbildung 4.1.: Jeder Schwellwert stellt einen Punkt in der ROC-Kurve vor. Der Schwellwert, der zu dem Punkt, mit dem minimalsten Abstand zu $(0,1)$ gehört, ist optimal. Der minimale Abstand ist hier mit einem roten Pfeil gekennzeichnet.

4.2. Ablauf des Testens

Nachdem das Training der Kaskade beendet ist, und als Resultat eine Liste der Kaskadenknoten, sowie die zugehörige Merkmalsliste vorliegt, können diese zu einem späteren Zeitpunkt wieder eingelesen werden, um die Kaskade zu rekonstruieren.
Dabei wird auch hier zunächst der Detektor initialisiert. Dieser importiert dabei die Merkmalsliste. Danach erfolgt der Aufbau der Kaskade mithilfe der Knotenliste.
Es wird nur ein Video zur selben Zeit untersucht, daher reicht es aus, ein Video vom System verwaltet zu lassen.
Nachdem die Kaskade geladen ist, kann der Testvorgang beginnen. Dabei wird das Video mit dem Detektor ausgewertet. Dies bedeutet, dass er an verschiedenen Punkten, sowie in unterschiedlichen Größen positioniert wird, und die Kaskade in diesem Teilvolumen ausgewertet werden kann.
Die Position des Detektors verändert sich dabei jeweils um ein Δx (Abhängig von der Breite des Detektors) oder Δy (Abhängig von der Höhe des Detektors). Wurde der Detektor nun an allen möglichen Positionen ausgewertet, wird er vergrößert, und das ganze wiederholt, bis auch alle Größen getestet wurden.
Ist die Auswertung der Kaskade positiv, so wird das Suchfenster als Aktionsregion gekennzeichnet.

4.3. Implementierung

Im Folgenden werden die einzelnen Komponenten, deren Bestandteile und Aufgaben näher erläutert. Die entsprechenden UML-Diagramme der einzelnen Klassen können in Anhang A eingesehen werden.

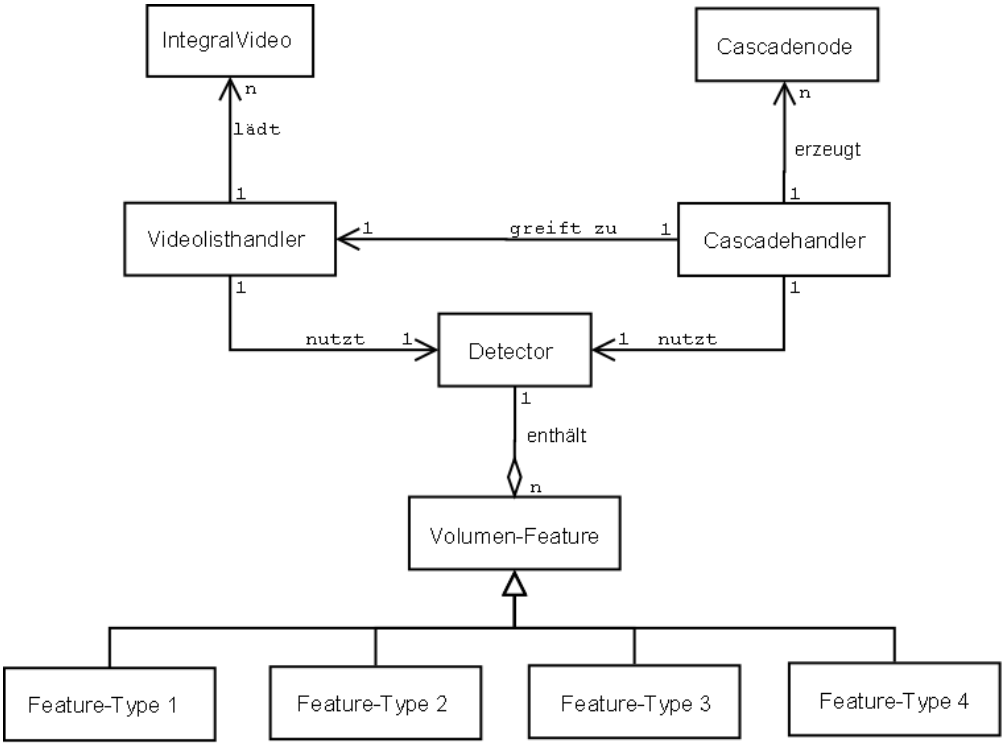


Abbildung 4.2.: Architektur

4.3.1. Komponenten

Das gesamte Programm besteht aus sechs Teilen.

- IntegralVideo
- Videolisthandler
- Detector
- Volumen-Feature
- Cascadehandler
- Cascadenode

Integral-Video

Die Aufgabe dieser Komponente ist die Verwaltung eines Videos in Form eines Integral-Videos (IV). Einem Objekt dieser Klasse ist es daher möglich dem bestehenden IV ein Frame eines Videos hizuzufügen. Dabei sind zwei Modi nutzbar. Zum einen der Insert-Modus, bei dem das neue Frame einfach hinten eingefügt wird, und zum anderen der Shift-Modus. Bei dem das IV über eine Maximallänge verfügt. Wird diese überschritten, so wird das Frame, das sich an erster Stelle befindet entfernt, und dafür das Neue hinten angefügt. Solange die Maximallänge nicht erreicht ist befindet man sich im Insert-Modus.
Um einen Überlauf des Datentyps zu vermeiden wird hier in regelmäßigen Abständen ein Refresh durchgeführt. Dabei wird das Frame, welches sich an erster Stelle befindet, von allen anderen Frames subtrahiert.

Videolisthandler

Dabei handelt es sich um den Teil des Programms, der das Management über die Trainingsmenge übernimmt. Darunter fallen folgende Aufgaben:

- das Auslesen der Daten, welche die Trainingsmenge beschreiben,
- das Laden von Videos als IV,
- das Befüllen einer LookUpTable,
- die Vervielfältigung der initialen Trainingsmenge,
- sowie die Generierung negativer Beispiele während des Trainings.

Die Trainingsmenge wird dabei durch Tupel der Form

$$(String\ URL,\ int\ pos\_x,\ int\ pos\_y,\ int\ height,\ int\ width,\ bool\ positiv,\ bool\ original)$$

beschrieben.
Die URL enthält dabei einen Pfad zu einem Video. Pos_x, pos_y, $heigth$ und $width$ geben das Volumen an, in dem sich das Beispiel befindet. $Positiv$ kennzeichnet dabei ob es sich um ein positives Beispiel handelt und $original$ ob es durch Vervielfältigung dynamisch entstanden ist oder nicht.
Die Vervielfältigung der Beispiele, hat den Vorteil, dass die Anzahl der vorhandenen Videos nicht übermäßig groß sein muss. Das gleiche Argument gilt für die Generierung negativer Beispiele. Zusätzlich hat dies den Vorteil, dass keine Negativ-Beispiele spezifiziert werden müssen.
Bei der LookUpTable handelt es sich um eine MxN Matrix, welche an der Stelle (j,i) die

Auswertung des *j*'ten Merkmals im *i*'ten Video enthält. *M* ist dabei die Anzahl der Merkmale und *N* die der Trainingsbeispiels. Somit ist es nicht nötig alle Videos über den gesamten Trainingsprozess hinweg im Speicher zu halten.

Detektor

Der Detektor übernimmt die Verwaltung der Merkmale. Dazu gehört

- die Generierung der Merkmale,
- das Skalieren der Merkmale bei einer Größenveränderung des Detektors,
- das Training der Merkmale,
- Lade- und Speicheroperationen.

Es existieren zwei Ladefunktionen. Die eine ist für das Laden von Detektorparametern (Höhe, Breite, Tiefe, sowie eine minimale Größe der Volumen-Feature). Die andere lädt, sofern man sich im Test-Modus befindet, eine Merkmalliste, welche Details über die Merkmale enthält, die eine zuvor erlernte Kaskade nutzt.
Dem gegenüber stehen die Speicheroperationen, welche, sofern man sich im Lern-Modus befindet, die Merkmaldetails sichert, die bei der Kaskadengenerierung in einen Knoten aufgenommen wurden.
Merkmale zu trainieren bedeutet hier, einen möglichst optimalen Schwellwert für jedes Merkmal zu finden, sodass eine Menge an Beispielen möglichst gut binär klassifiziert werden kann.
Für die Skalierung des Detektors stehen einige Funktionen zur Verfügung, welche es erlauben Skalierungen in X- und Y Richtung sowie der Tiefe vorzunehmen. Dabei gehen diese Funktionen immer von den Originalwerten des Detektors aus. Soll der Detektor beispielsweise auf die Breite 100 skaliert werden, so wird zuvor der Faktor ermittelt, mit dem die Originalbreite multipliziert werden muss. Dieser Faktor wird dann auch genutzt, um alle Merkmale in der Breite zu skalieren.

Volumen-Feature

Dies ist die Überklasse aller Merkmalstypen und hat folgende Bestandteile:

Diese wären:

- Eine Punkteliste, zur Formgebung des Merkmals
- Ein Gewicht
- Möglichkeiten eine Skalierung in X,Y und Z Richtung vorzunehmen

- Eine virtuelle Funktion, die es jedem Merkmalstyp ermöglicht, eine individuelle Auswertung in einem Integral-Video vorzunehmen
- Ein Schwellwert der für letzteres genutzt wird

Die eben erwähnte virtuelle Funktion sowie die Punkteliste definieren den Typ des Merkmals.

Cascadehandler

Diese Komponente ist für die Generierung und Verwaltung der Kaskade verantwortlich. Für letzteres beinhaltet sie eine Liste an Kaskadenknoten (Cascadenode), in welche neu erstellte Knoten immer am Ende hinzugefügt werden um die Reihenfolge der Knoten zu wahren.
Des weiteren beinhaltet der Cascadehandler Funktionen für

- die Erzeugung von Knoten
- das Setzen von Gewichten ‚der Merkmale, die einem Knoten zugewiesen wurden
- das Lernen der Schwellwerte der Knoten
- die Evaluation der Kaskade

Cascadenode

Diese Klasse enthält ein Ensemble, welches eine Menge an Merkmalen beinhaltet. Gemeinsam kann diese Menge an Merkmalen eine gegebene Trainingsmenge binär klassifizieren. Das Ensemble wurden während der Knotengenerierung erstellt.
Auf Grundlage dieses Ensembles kann nun ein Knoten auf verschiedene Weisen für oder gegen die Präsenz erlernter Eigenschaften einer Aktion votieren.
Während der Generierung des Knotens geschieht dies mittels eines Mehrheitsentscheids. Wohingegen die Entscheidung im Test-Modus mithilfe der gewichteten Summe der Merkmalsaktivitäten gebildet wird.

5. Ansatz mit Bewegungsvorhersage

Auf Grundlage des volumetrischen Ansatzes ist eine neue Systemarchitektur entstanden. Die in Abbildung 5.3 enthaltenen Komponenten übernehmen hier die selben Aufgaben, wie jene des volumetrischen Ansatzes. Einige der Elemente mussten jedoch gemäß der neuen Struktur angepasst werden. Die Modifikationen die vorgenommen wurden, werden im späteren Verlauf genauer erläutert.
Zunächst wenden wir uns jedoch den zu Grunde liegenden Restriktionen und den damit verbundenen Vorbemerkungen zu. Die Einschränkung hier betrifft die Gestalt der Merkmale. Genutzt werden zweidimensionale Rechteckmerkmale um Aktionen in Framesequenzen zu erkennen. Dadurch kann die Auswertung dieser theoretisch schneller vollzogen werden als dies mit anderen Merkmalen möglich wäre. Die in Abbildung 2.1 dargestellten Merkmale werden genutzt.
Ein solches Merkmal ist jedoch nur in einer Momentaufnahme aussagekräftig, da es ohne weitere Maßnahmen nicht in der Lage ist zeitliche Informationen zu kodieren. Daher wurde die neue Komponente Stronger Classifier (Stcf) eingeführt um dieses Problem zu lösen. Jede Instanz dieser Klasse kodiert den zeitlichen Verlauf der Auswertung eines Merkmals. Anhand einer Beispielmenge wird für jedes Merkmal eine Schwellwertkurve trainiert.

Auf Grundlage der Stronger classifier wird nun ein Boosting durchgeführt und man erhält wiederum eine Entscheidungskaskade.
Beim Test eines Suchfensters in einem Video, kann nun durch Block-Matching ermittelt werden in welche Richtung sich das Fenster bewegt, bzw. in welcher Richtung der Fehler, zu den erlernten Kurven, minimal ist. In dieser „besten“ Richtung kann nun die Kaskade überprüfen, ob die Aktion tatsächlich stattgefunden hat. Dadurch sollte die Auswertung eines Suchfensters durch die Kaskade beschleunigt werden, wodurch die Detektionsrate und die Testgeschwindigkeit gesteigert werden sollte. Eine Geschwindigkeitssteigerung sollte auch dadurch erzielt werden, indem die Berechnung des Integral-Videos entfällt. Stattdessen müssen nur noch Integral-Images der einzelnen Frames berechnet werden.

5.1. Annotation der Beispieldaten

Jedem Beispielvideo muss ein Datentupel zugewiesen werden, das folgende Elemente besitzt:

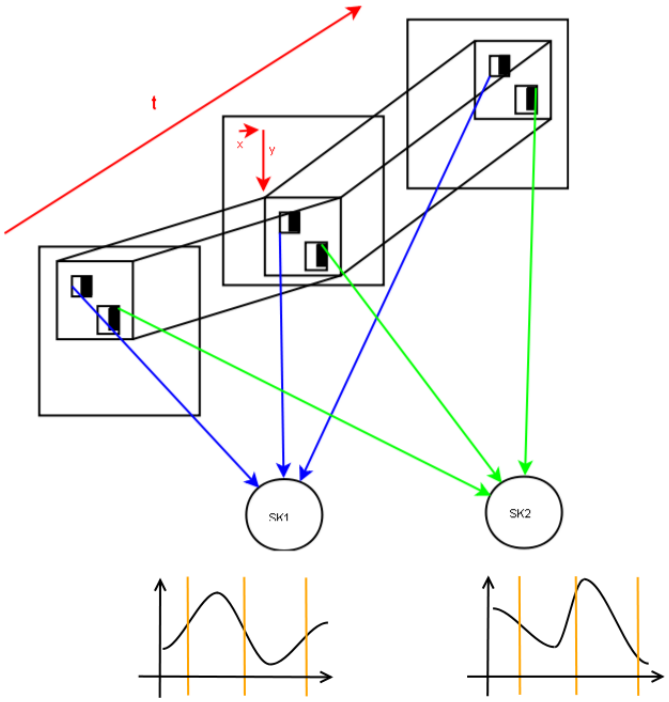


Abbildung 5.1.: Positionsänderung eines Suchfensters und daraus resultierende Schwellwertkurven der Stronger Classifier. Orange Linen entsprechen den Schwellwerten, die ein Merkmal in den dargestellten Frames aufweist. Mit diesen Schwellwerten wird die Wertekurve interpoliert.

- *String url*
- *bool positiv_negativ*
- *int x_start, y_start*
- *int x_end, y_end*
- *int width, int height*
- *bool original*

Die *URL* enthält dabei einen Pfad zu einem Video. *X_start*, *y_start*, *heigth* und *width* geben den Bereich an, in dem sich der Anfang der Aktion befindet. Wohingegen *x_end*, *y_end*, *heigth* und *width* den Bereich markieren, in dem die Aktion endet. *Positiv_negativ* kennzeichnet dabei ob es sich um ein positives Beispiel handelt und *original* ob es durch Vervielfältigung dynamisch entstanden ist. Abbildung 5.2 stellt die Annahme einer linearen Bewegungsrichtung da. Diese Annahme kann hier getroffen werden, da die Trainingsmenge aus kurzen Videosequenzen besteht.

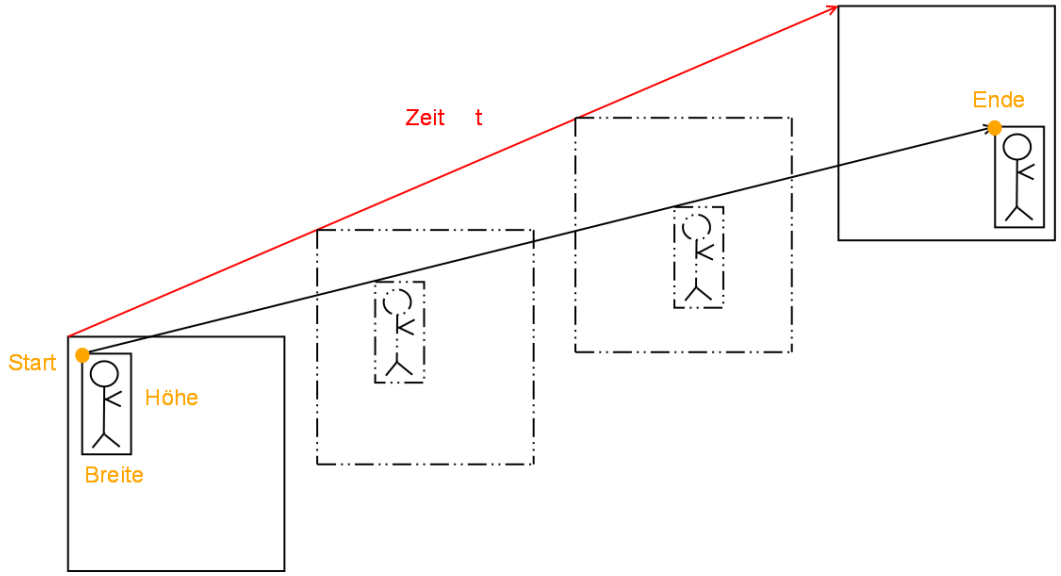


Abbildung 5.2.: Visualisierung der Annotation

5.2. Ablauf des Trainings

Auch hier beginnt alles mit dem Einlesen der Detektorparameter, gefolgt von der Erzeugung der Merkmale. Neu ist an dieser Stelle, dass zum Einen eine FrameInfoListe generiert wird und zum Anderen die Initialisierung der Statistikmatrix.
Die FrameInfoListe besteht aus Tupeln der Form

$$(Framenummer, Punkt(x, y)).$$

Diese Liste wird im weiteren Verlauf dazu genutzt, die Interpolationspunkte zwischen Start- und Endfarme zu speichern. Zu diesem Zeitpunkt ist es jedoch nur möglich die Frames zu berechnen, in denen Auswertungen der Merkmale stattfinden sollen. Dabei sind immer mindestens zwei Elemente in der Liste enthalten. Das erste Frame $(0, k)$ und das letzte $(depth - 1, k)$. Dabei ist k irgendein Punkt und $depth$ enspricht der zeiltichen Ausdehnung des Detektors.
Als nächstes folgt die Initialisierung der Komponenten, welche für die Verwaltung der Videodaten verantwortlich ist. Dabei wird zunächst die Beispielliste geladen. Danach werden, wie beim volumetrischen Ansatz die Originalbeispiele vervielfältigt und negative Beispiele erzeugt.
Nun wird auch eine Lookuptable angelegt und anschließend mit Werten gefüllt. Für jedes Trainingsbeispiel wird dabei dir Detektor in Höhe un Breite skaliert. Die Tiefe wird auf die Anzahl der Frames des Videos gesetzt. Dies bedeutet hier allerdings, dass auch die Framenummern der FrameInfoList proportional verschoben werden müssen, um ihre relative

Position beizubehalten. Dann werden, mithilfe der Punkte *start* und *end* des Beispieltupels, die Zwischenpunkte berechnet, an denen der Detektor ausgewertet werden soll. Diese Punkte werden dann in die entsprechenden Felder der Tupel der FrameInfoListe eingefügt. Mit dieser Liste können nun alle Merkmale in allen nötigen Frames ausgewertet werden. Danach wird die Statistik des Detektors aktualisiert, falls das aktuelle Beispiel positiv sein sollte.
Der letzte Schritt ist nun die Generierung der Kaskade. Dies geschieht analog zum volumetrischen Ansatz, ebenso wie das Sichern der Kaskade und das aller genutzten Merkmale.

5.3. Ablauf des Testens

Der Ablauf beim Test eines Suchfensters eines zu testenden Videos erfolgt analog zu dem Testablauf des volumetrischen Ansatztes.
Die einzige Neuerung ist das Blockmatching, das in der *eval*() Funktion des NewCascadehandler pro Knoten ausgeführt wird, um möglicherweise einen früheren Abbruch der Evaluation der Kaskade zu erzielen.

5.4. Implementierung

Im Folgenden werden nun die einzelnen Komponenten, deren Bestandteile und Aufgaben näher erläutert. Die entsprechenden UML-Diagramme der einzelnen Klassen können in Anhang B eingesehen werden.

5.4.1. Komponenten

Das gesamte Programm besteht aus sieben Teilen.

- NewIntegralVideo
- Feature2D
- NewVideolisthandler
- Stronger Classifier (Stcf)
- NewDetector
- NewCascadehandler
- NewCascadenode

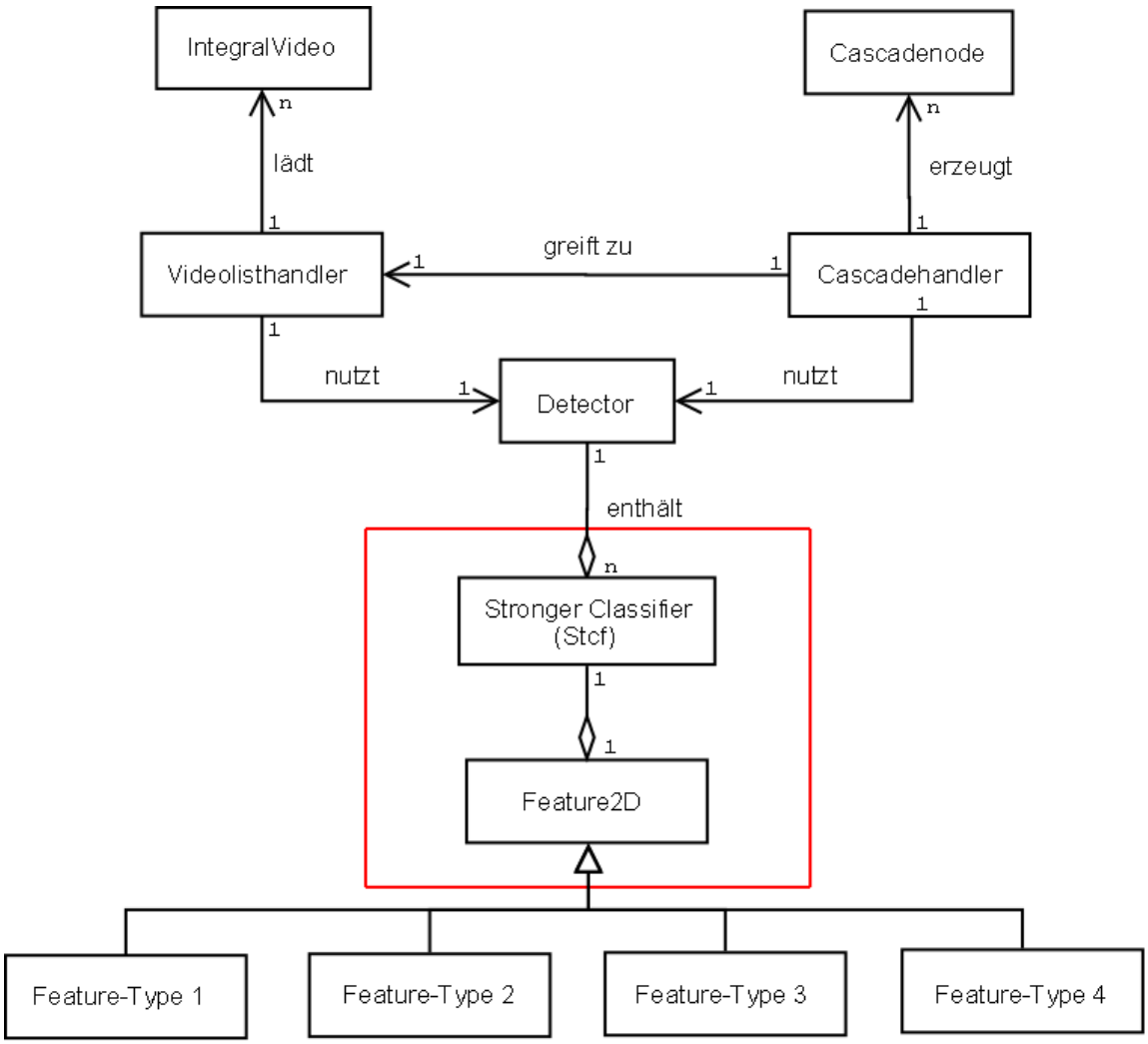


Abbildung 5.3.: Neue Systemarchitektur. Neu hinzugekommene Komponenten sind rot umrandet.

NewIntegralVideo

Diese Klasse wurde lediglich modifiziert, und zwar wurde hier eine Liste an Integral-Images angelegt. Wenn nun der Aufruf von *addFrame*() erfolgt, so wird das Integral-Image des übergebenen Frame berechnet. Danach wird es am Ende der Integral-Image-Liste eingefügt. Auch hier gibt es wieder den Shift- und den Insert-Modus.

Feature2D

Dies ist die Überklasse aller Merkmalstypen, von der sie alle ihre Eigenschaften erben.

- Eine Punkteliste, zur Formgebung des Merkmals
- Möglichkeiten eine Skalierung in X- und Y- Richtung vorzunehmen
- Eine virtuelle Funktion, die es jedem Merkmalstyp ermöglicht, eine individuelle Auswertung in einem NewIntegralVideo vorzunehmen

Die eben erwähnte virtuelle Funktion sowie die Punkteliste definieren den Typ des Merkmals, die in Abbildung 2.1 dargestellt sind.

NewVideolisthandler

Der NewVideolisthandler erfüllt die gleiche Aufgabe wie der Videolisthandler, jedoch wurde die Funktionsweise einiger Methoden angepasst. So musste in $jitterExamples()$ bei der Vervielfältigung der Originalbeispiele darauf geachtet werden, dass sich das Rechteck im letzten Frame noch im Video befindet. Bei der Erzeugung der negativen Beispiele musste sich das Rechteck analog im ersten und letzten Frame innerhalb des Videos befinden.
Auch hier existiert wieder eine Lookuptable. Es handelt sich wiederum um eine Matrix. Die Elemente dieser Matrix sind jedoch Listen von evaluierten Werten. Daher befindet sich an der Stelle (i, j) eine Liste an Werten, die durch das Merkmal i im j'ten Beispiel errechnet wurde. Dahingehend wurde die $evalVideo()$ Funkion angepasst um dies zu bewerkstelligen.

Stronger Classifier (Stcf)

Wie beschrieben kodiert ein Objekt dieser Klasse eine Schwellwertkurve eines Merkmals, das dieser Instanz zugewiesen wurde. Die Kurve wird als Liste gespeichert, welche die erlernten Schwellwerte enthält. Das heißt, dass i'te Element entspricht dem Schwellwert des Merkmals, das zur Zeit i, anhand einer positiven Menge P und einer negativen N, ausgewertet wurde.

Diese Schwellwerte werden auch hier gewählt, indem der Wert gesucht wird, der in einer ROC-Kurve die geringste Distanz zum Punkt $(0, 1)$ besitzt. Zusätzlich wird ein Schwellwert erlernt, der aussagt, wie oft das Merkmal mit der Schwellwert-Liste mindestens aktiv sein muss.
Des weiteren sind Funktionen zur Skalierung vorhanden, sowie für die Aktivitätsprüfung. Für das Blockmatching ist die Funktion $calsError()$ vorhanden, welche den quadratischen Fehler, im Vergleich mit der gelernten Kurve, eines Bereichs ermittelt.

Dabei wird das Merkmal zur Zeit i an $Punkt(i)$ ausgewertet und es wird überprüft, ob es aktiv ist. Falls ja, so wir die Fehlersumme nicht erhöht. Wenn es jedoch nicht aktiv sein sollte,

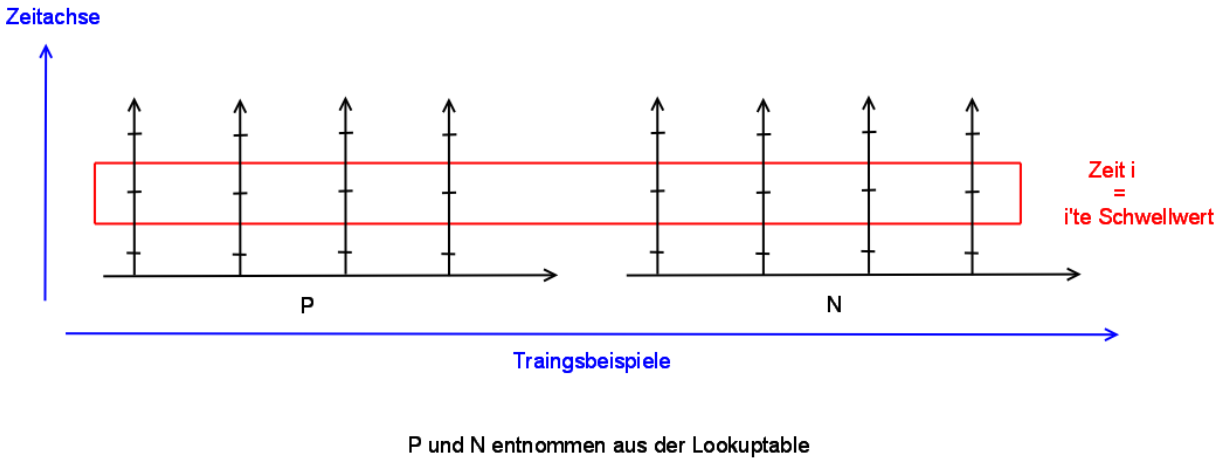


Abbildung 5.4.: Wertemenge zur Berechung eines Schwellwertes entspricht den rot umrandeten Werten. Ein Pfeil entspricht der Auswertung eines Traingsbeispiels zu diskreten Zeitpunkten

so wird die Fehlersumme um das Quadrat der Differenz von Schwellwert und evaluiertem Wert erhöht.

$$error = \sum_{i=0}^{k-1} (threshold\_list(i) - eval\_line(i))^2$$

Die benötigten Tupel $(i, Punkt(i))$ werden der Funkion in Form einer FrameInfoList übergeben. Diese Liste stellt die räumliche Positionsveränderung über die Zeit da. $Punkt(i)$ entspricht daher dem Punkt, an dem sich das Suchfenster zur Zeit i befindet.

NewDetector

Der Aufgabenbereich des Detektors hat sich ausgeweitet auf:

- Die Generierung der Merkmallisten
  - Stcf-Liste
  - Feature2D-Liste
- Training der Stcf-Merkmale
- Skalierung der Merkmale bei Größenveränderung des Detektors
- Laden und Speichern der Merkmallisten
- Blockmatching beim Testen des Bereichs, der durch den Detektor in einem Video überdeckt wird

Die Generierung der Rechteckmerkmale erfolgt auf die gleiche Weiße wie im volumetrischen Ansatz. Jedoch muss hier die Tiefe nicht berücksichtigt werden. Für jedes angelegte Merkmal wird ebenso ein Objekt der Klasse Stcf instantiiert, welches einen Verweis auf das Rechteckmerkmale enthält. Für die Skalierung der Rechteckmerkmale werden die Funktionen zur Skalierung der Klasse Stcf genutzt, welche analog zu früher funktionieren. Ebenso erfolgt das Training über den Aufruf der Funktion *trainThresholds*() der Stcf-Objekte.
Die Speicheroperation *saveDetector*() wurde angepasst, indem nicht nur die Daten eines jeden in der Kaskade genutzten Merkmals gespeichert werden, sondern auch noch die gelernten Informationen des Stcf-Objektes, also die Schwellwertliste sowie die minimale Anzahl der nötigen zeitlichen Aktivität des Merkmals.
Zudem wird ebenso eine Statistik über das Bewegungsverhalten der positiven Beispiele gesichert. Erstellt wird diese, indem die Richtung und die Strecke der Bewegungen registriert werden. Dies ist in Abbildung 5.5 veranschaulicht.

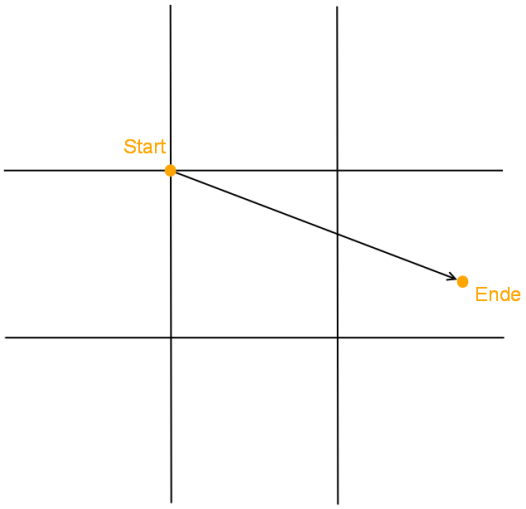


Abbildung 5.5.: Registrierung der Bewegung. Dabei befindet sich das aktuelle Suchfenster im Zentrum und es wird der Quadrant ermittelt, in den sich das Suchfenster verschiebt.

Beim Laden der Merkmalliste werden zunächst die Statistikinformationen geladen und danach in einer Prioritäts-Such-Liste neu angeordnet. Diese enthält an erster Stelle die Richtung, in die eine Aktion am häufigsten verlief, gefolgt von der zweithäufigsten usw. Die Länge des Richtungsvektors entspricht dem Durchschnitt der Entfernungen, die in diese Richtung zurückgelegt wurden.

Nun werden die restlichen Merkmaldaten geladen.
Beim Laden der Detektorparameter musste zusätzlich die Anzahl der Interpolationspunkte geladen werden. Damit wird festgelegt, an wie vielen Stellen ein Video zeitlich ausgewertet wird. Diese Zahl muss größer oder gleich zwei sein, da das erste und letzte Frame immer ausgewertet werden. Je mehr Interpolationspunkte man verwendet, desto genauer können

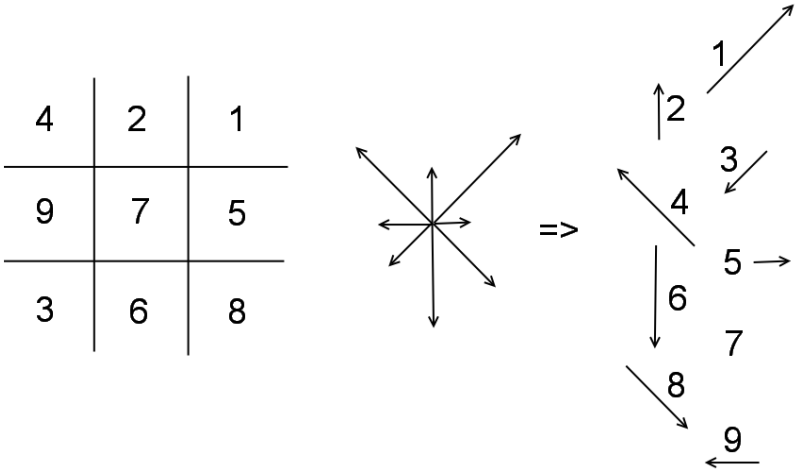


Abbildung 5.6.: Links: Statistikmatrix mit Wahrscheinlichkeitsverteilung der Bewegungsrichtung. Mitte: Durchschnittliche Entfernung die in eine Richtung zurückgelegt wurde. Rechts: Prioritäts-Such-Liste. An erster Stelle steht die Bewegungsrichtung, in die eine Bewegung am häufigsten verlief.

die Schwellwertkurven approximiert werden. Jedoch wird damit auch der Aufwand der Auswertung erhöht. In allen Trainingsdruchläufen wurde hier dieser Wert auf drei gesetzt. Ein neu entstandener Teilbereich ist das Blockmatching. Wird $createBlockmatchedInfoList()$ aufgerufen, wird ein Punkt im Endframe gesucht, in den das aktuelle Detektorsuchfenster verschoben werden kann. Dieser Punkt wird anhand des summierten Fehlers einer Liste an Stcf-Objekten ermittelt.

$$error = \sum_{i=0}^{j} stcf(i).calcError()$$

Dabei werden mögliche Punkte in Richtung und Reihenfolge entsprechend der Prioritäts-Such-Liste ausgewertet. Wird in einem Punkt ein kleinerer Fehler errechnet, so wird an diesem Punkt rekursiv weiter gesucht. Die Länge der Richtingsvektoren der Prioritäts-Such-Liste wird beim rekursiven Aufruf halbiert.
Sobald, nach Abbruch der Rekursion, ein Punkt k mit minimalem Fehler gefunden wurde, wird eine FrameInfoList mit $(i, Punkt(i))$ erstellt, wobei i für das i'te Frame steht. Es gilt $(0, start)$, $(depth - 1, k) \in FrameInfoList$. Hierbei entspricht $start$ dem Punkt, an dem sich das Suchfenster befindet und $depth$ steht für die Tiefe, zeitliche Ausdehnung oder Anzahl der überwachten Frames des Detektors. Somit ist sichergestellt, dass jeweils der erste und letzte Punkt im ersten und letzten Frame enthalten sind. Alle anderen Punkte werden äquidistant auf einer direkten Strecke von $start$ nach k verteilt.

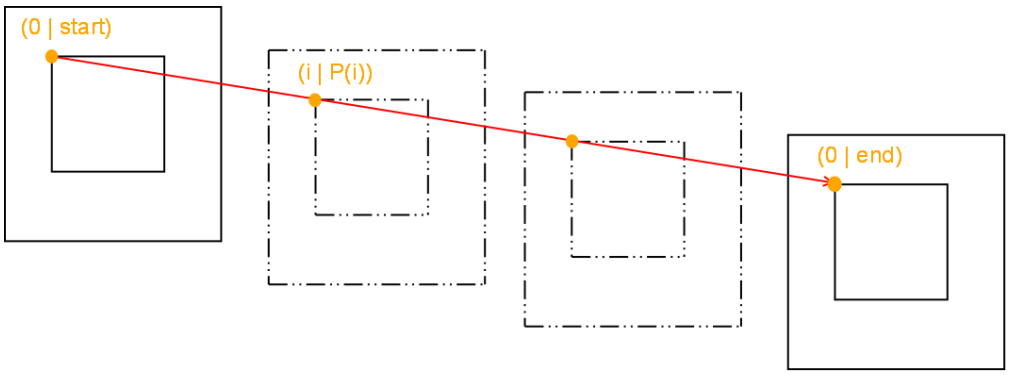


Abbildung 5.7.: Visualisierung der FrameInfoList

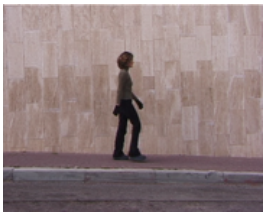
NewCascadehandler und NewCascadenode

Auch die Aufgaben dieser beiden Klassen haben sich nicht geändert. Änderungen wurden am NewCascadehandler lediglich an den Evaluationsfunktionen vorgenommen. Für jeden Knoten der Kaskade wird eine FrameInfoListe mithilfe der Funktion $createBlockmatchedInfoList()$ der Detektors erstellt, um somit jeden Knoten in die am vielversprechendste Richting auszuwerten.
Im NewCascadenode mussten nur die Funktionen $isAktiv()$ der Stcf-Klasse genutzt werden, statt wie bisher die der Merkmale selbst.

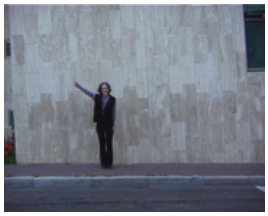
6. Auswertung

In diesem Abschnitt werden die beiden Methoden sowohl im Trainings- als auch im Testmodus untersucht. Dabei werden Zeitmessungen an verschiedenen Stellen der Programme vorgenommen, die eine äquivalente Funktion erfüllen.
Der für den Trainings- und Testmodus verwendete Datensatz entspricht dem Weizmann-Datensatz. Dabei handelt es sich um eine Ansammlung von mehreren Videos, welche bestimmte Aktionsklassen, wie zum Beispiel „Rennen“, „Gehen“ oder das ausführen eines „Hampelmann“, beinhalten. In jedem Video sind jeweils mehrere Perioden der selben Bewegung auf einem homogenen Hintergrund dargestellt.

(a) Rennen

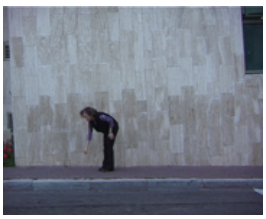


(b) Gehen

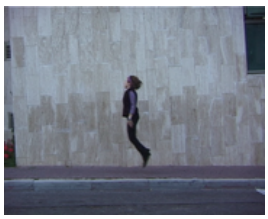


(c) Winken

(d) Hampelman



(e) Bücken



(f) Springen

Abbildung 6.1.: Aktionsklassen des Weizmann-Datensatz

Diese Videos eignen sich in ihrer ursprünglichen Form nicht zum Training der vorgestellten Verfahren, denn es darf hier pro Trainingsbeispiel nur eine Periode einer Aktion vorhanden sein. Daher wurde jedes ursprüngliche Video gemäß der Anforderung in mehrere geteilt. Die Testreihen wurden dann auf Grundlage der originalen Videos durchgeführt.
Analysen werden in folgenden Bereichen durchgeführt:

- der Einfluss der Videorepräsentation
- die Trainings- und Testgesamtdauer sowie die Längen der Kaskaden

- der Einfluss der Granularität als Parametern

Die Auswertung erfolgt anhand von acht Trainingsdurchläufen, welche in Tabelle 6.1 mit den entsprechenden Startparametern dargestellt sind.

Methode	Bezeichung	Overlay in %	maximale Falschdetektionsrate pro Kaskadenknoten
Ansatz mit Bewegungsvorhersage	na1	0	0.2
	na2	30	0.2
	na3	60	0.2
	na4	30	0.5
Volumetrische Ansatz	va1	0	0.2
	va2	30	0.2
	va3	60	0.2
	va4	30	0.5

Tabelle 6.1.: Tabelle der Trainingdsdurchläufe

Overlay gibt dabei an, wieviel ein negatives Beispiel ein positives überdecken darf. Die Trainingsmenge besteht auch 21 Videoclips, von dem jedes eine Wiederholung der Aktion enthält. Durch die Vervielfältigung dieser Beispiele ergibt sich eine Gesamtmenge an positiven Beispielen von c.a. 1200. Ebenso viele negative Beispiele werden pro Erzeugung eines Knotens generiert. Beide Methoden wurden auf der Bewegungsklasse „Hampelmann“ trainiert und getestet.

6.1. Einfluss der Videorepräsentation

In diesem Abschnitt wird die Leistungsfähigkeit der Videorepräsentation der beiden Methoden untersucht. Dabei werden hier die durchschnittlichen Zeiten, die für die Konvertierung der Videos in die jeweils verwendete Darstellung anfallen, für Trainings- und Testdurchlauf, gegenübergestellt. Während eines Tests wird dabei der Shift-Modus ausgewertet, wohingegen während des Trainings der Insert-Modus betrachtet wird.
Die Auswertung von Abbildung 6.2 und 6.3 zeigt, dass die Videorepräsentation des volumetrischen Ansatzes, in beiden Modi, in der Lage ist Videos deutlich schneller laden. Dieser Effekt könnte jedoch von der verwendeten Datenstruktur abhängen. Beim volumetrischen Ansatz wird ein Array verwendet, um Videodaten so speichern, wohingegen beim neuen Ansatz eine Vektor verwendet wird. Die Operationen dieser Klasse sind, wenn auch optimiert, komplexer als Operationen auf einem Array.

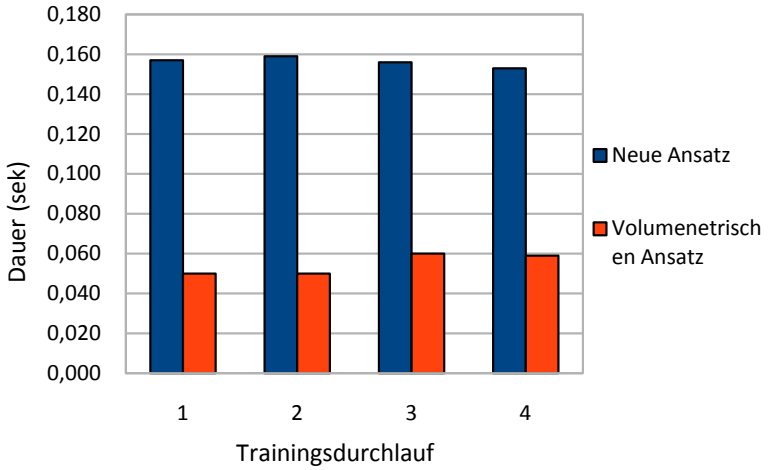


Abbildung 6.2.: Zeitvergleich des Insert-Modus

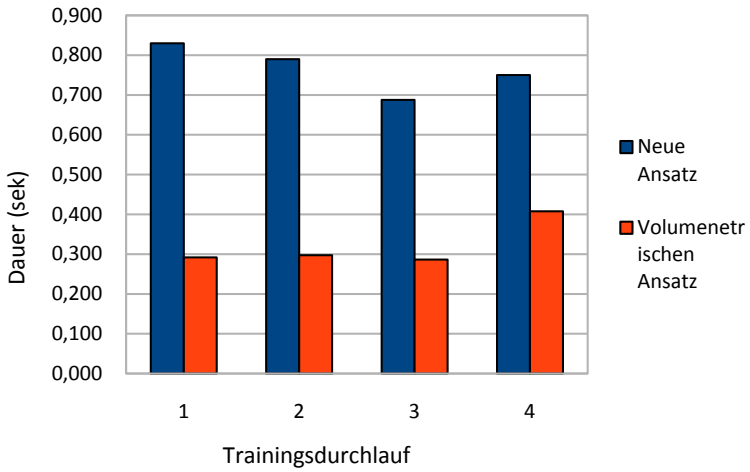


Abbildung 6.3.: Zeitvergleich des Shift-Modus

6.2. Trainings- und Testdauer

In diesem Abschnitt werden nun die durchschnittlichen Test- und Trainingszeiten miteinander verglichen. Start und Stop der Zeitmessungen sind dabei jeweils Beginn und Ende des jeweiligen Modus. Zusätzlich wird die durchschnittliche Anzahl der Merkmale der ersten drei Knoten dargestellt. Denn diese Knoten sind, mit hoher Wahrscheinlich, bei jeder Evaluation der Kaskade beteiligt und tragen daher maßgeblich zur Evaluationsdauer bei.

Trainingdauer

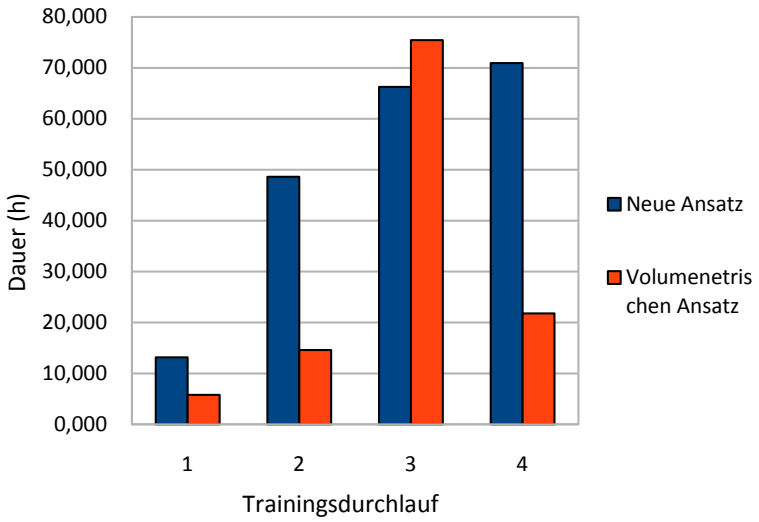


Abbildung 6.4.: Vergleich der Trainingsdauer

Das Diagramm in Abbildung 6.4 zeigt, dass der neue Ansatz deutlich mehr Zeit für das Training benötigt. Der Hauptgrund hierfür liegt in der Verwendung der Stcf-Feature. Bei diesen ist es nötig, eine Liste an Schwellwerten zu erlernen statt nur einem einzigen wie beim volumetrischen Ansatz. Auffällig ist die zunehmende Trainingsdauer. Dieser Effekt wird durch den Overlay-Parameter verursacht, da nur er in den ersten drei Trainingsdurchläufen verändert wird.

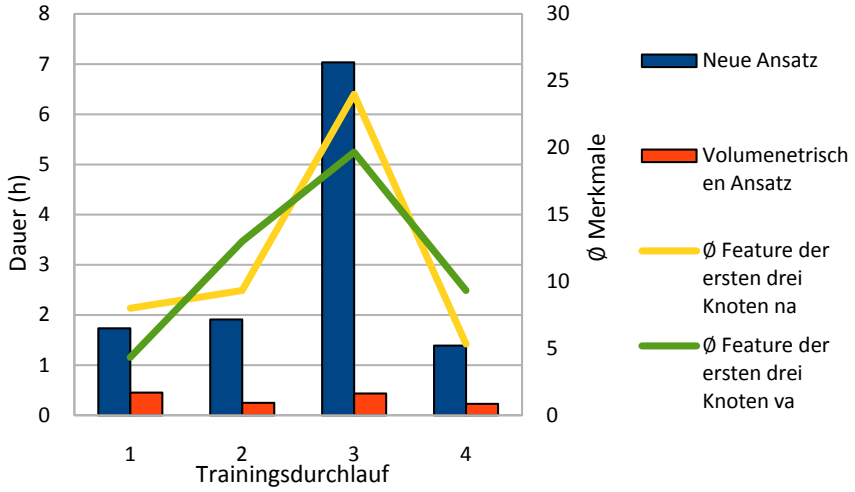


Abbildung 6.5.: Vergleich der Testdauer und durchschnittliche Anzahl an Merkmalen in den ersten drei Knoten

Testdauer

Durch das Diagramm in Abbildung 6.5 wird deutlich, dass der volumetrische Ansatz deutlich weniger Zeit zur Evaluation benötigt als der Ansatz mir Bewegungsvorhersage. Jedoch sind bei diesem überwiegend mehr Merkmale an der Auswertung beteiligt. Dies führt zu einer höheren Evaluationsdauer. Zusätzlich wird bei der Auswertung der Kaskade pro Knoten ein Blockmatching durchgeführt. Da der volumetrische Ansatz nicht über einen derartigen Mechanismus verfügt stellt dieser einen Overhead da, wodurch die Dauer der Auswertung vergrößert wird. Dennoch stellt sich die Frage, um wie viel das Blockmatching das Ganze erhöht. Dies wird im Diagramm der Abbildung 6.6 veranschaulicht. Die gesamte Evaluationdauer setzt sich dabei aus der Dauer des Blockmatchings und der des gewichteten Mehrheitsendscheids der Kaskadenknoten zusammen. In diesem Diagramm ist deutlich zu erkennen, dass das Blockmatching in etwa $\frac{2}{3}$ der Zeit in Anspruch nimmt. Der Mehrheitsendscheid der Kaskadenknoten beansprucht in etwa genau soviel Zeit wie der des volumetrischen Ansatz.

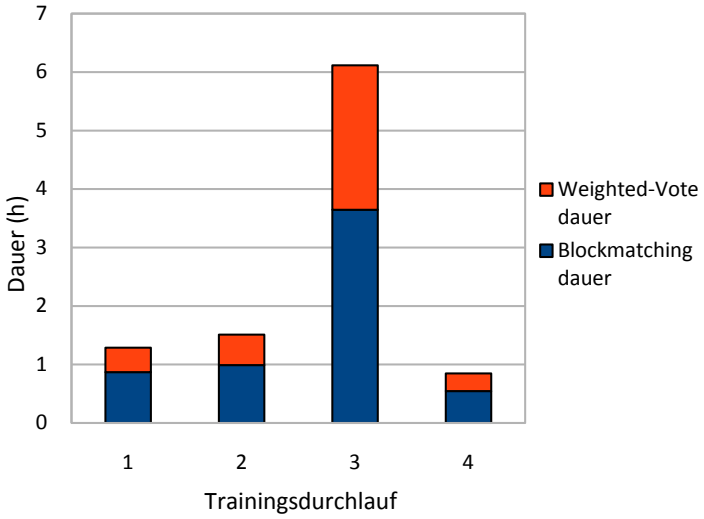


Abbildung 6.6.: Hauptanteile der Evaluationsdauer

6.3. Einfluss der Granularität

Für den Testmodus existiert nur ein Parameter, die Granularität der Suchfenster. Diese sagt aus, wie viele Suchfenstergrößen auftreten können bzw. wie oft das Suchfenster jeweils in X- und Y Richtung vergrößert werden darf. Liegt die Granularität bei vier, so existierten 4^2 Suchfenstergrößen. Andere Parameter werden durch das Training vorgegeben und können nachträglich nicht verändert werden.
Im Trainingsmodus gibt es zwei Parameter, den Overlay und die maximale FPR pro Kaskadenknoten. Die Ergebnisse dazu sind in den Precision-Recall-Diagrammen (a)-(d) der Abbildung 6.8 integriert. Bei jedem Diagramm wird die Granularität variiert und jeder andere Parameter bleibt konstant. Von Diagramm zu Diagramm wird lediglich die Toleranz verändert, mit der ein Suchfenster als positiv erkannt wird. Diese beschreibt, um wieviel Prozent das Suchfenster von der perfekten Detektion abweichen darf.

Allgemein zeigt das Diagramm in Abbildung 6.7, dass die Testdauer mit der Granularität steigt. Dies entspricht genau der Erwartungshaltung, den je mehr Suchfenster untersucht werden müssen, desto länger dauert die Auswertung.

Diagramm (a) der Abbildung 6.8 sagt aus, das kein Suchfenster gefunden wurde, welches einer perfekten Detektion entspricht. Aus den restlichen Kurven der Diagramme ist zu erkennen, dass die Granularität Einfluss auf die Güte der Detektion hat. Wählt man sie zu niedrig oder zu hoch, so sinken oder steigen die Precision oder der Recall. Dies wird durch

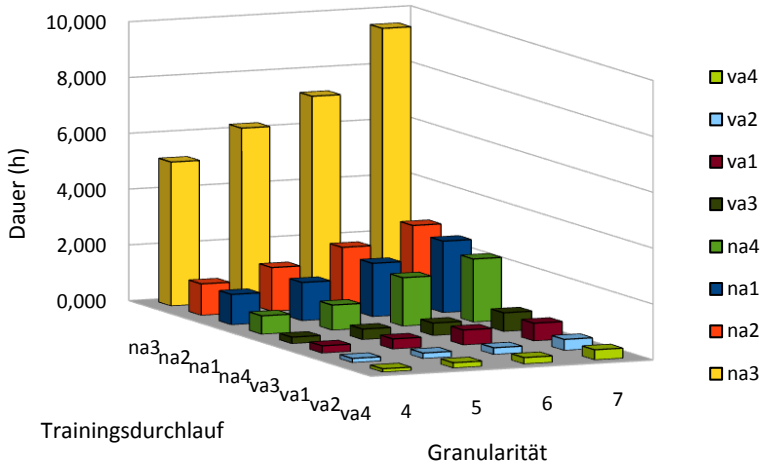
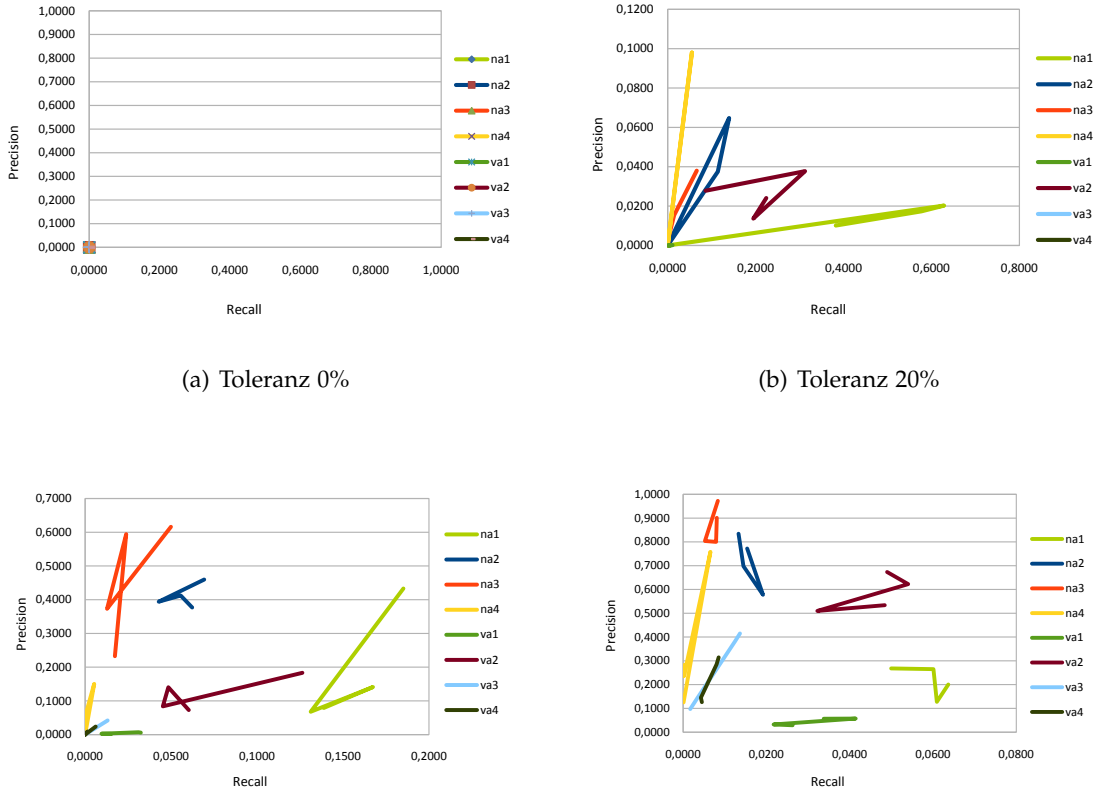


Abbildung 6.7.: Einfluss der Granularität auf die Testdauer eines Videos der Länge fünf Sekunden und 136 Frames.

Kurve *na*2 oder *va*2 in Diagramm (b) deutlich. Wie man deutlich sehen kann ist auch die Toleranz nicht bedeutungslos. Erhöht man sie so verschieben sich die Kurven in Richtung des Punktes $(1, 1)$. Dies bedeutet, dass viele Suchfenster korrekt als richtig oder falsch (die trainierte Aktion ist vorhanden oder nicht) eingestuft werden.
Durch die Lage der Kurven wird auch deutlich, dass unter gleichen Parametern, der Ansatz mit Bewegungsvorhersage bessere oder vergleichbar gute Ergebnisse erzielt als der volumetrische Ansatz.
Es existieren noch weitere Parameter, die verändert werden könnten, jedoch gemäß [WRM03] und [KSH05] gibt es für diese keinen all zu großen Spielraum. Da wäre zum einen die maximale Falschdetektionsrate, welche die gesamte Kaskade erreichen soll, also das Lernziel, und zum anderen die minimale Detectionrate pro Knoten.



(a) Toleranz 0%

(b) Toleranz 20%

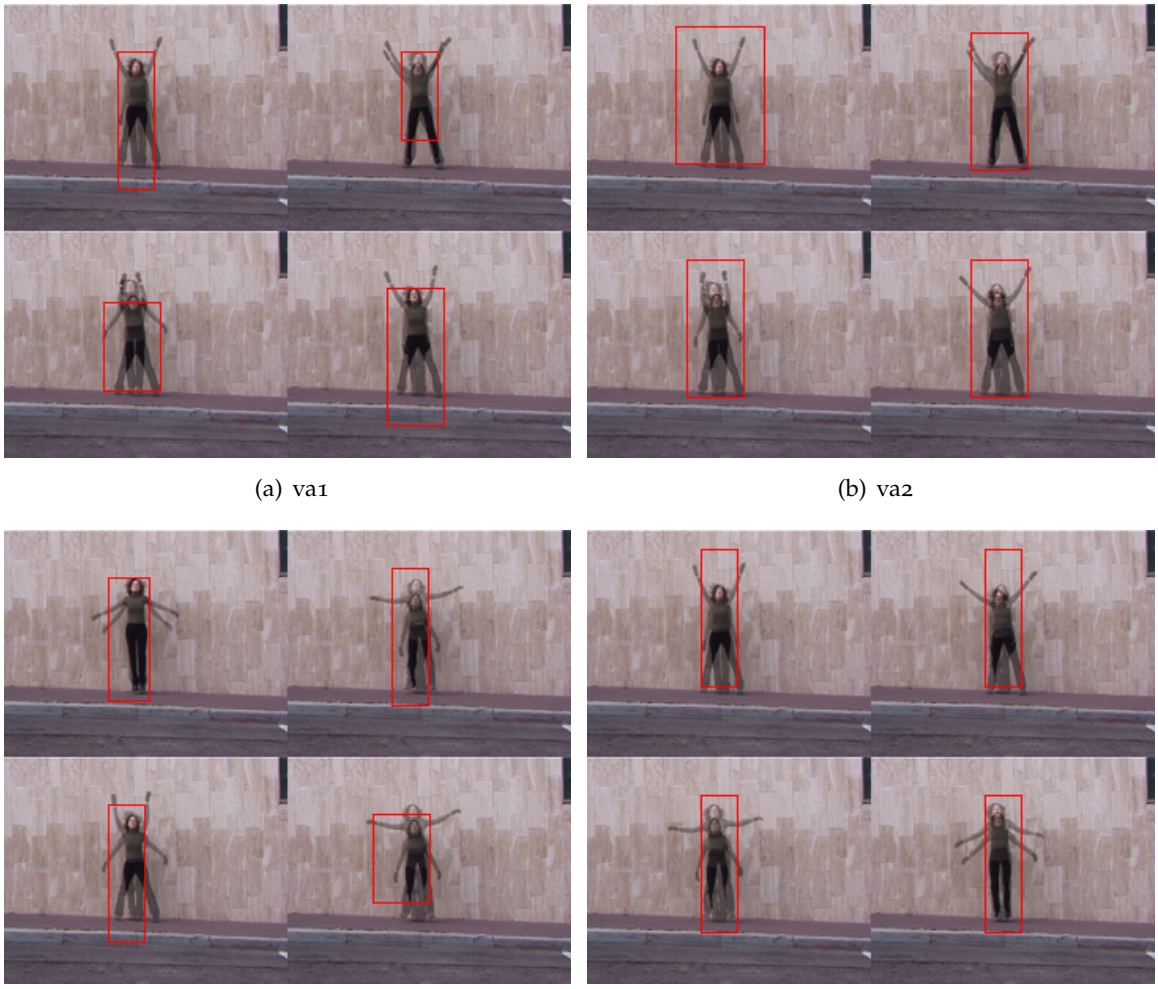
(c) Toleranz 40%

(d) Toleranz 60%

Abbildung 6.8.: Precision-Recall-Diagramme für variierende Granularität

6.4. Testresultate

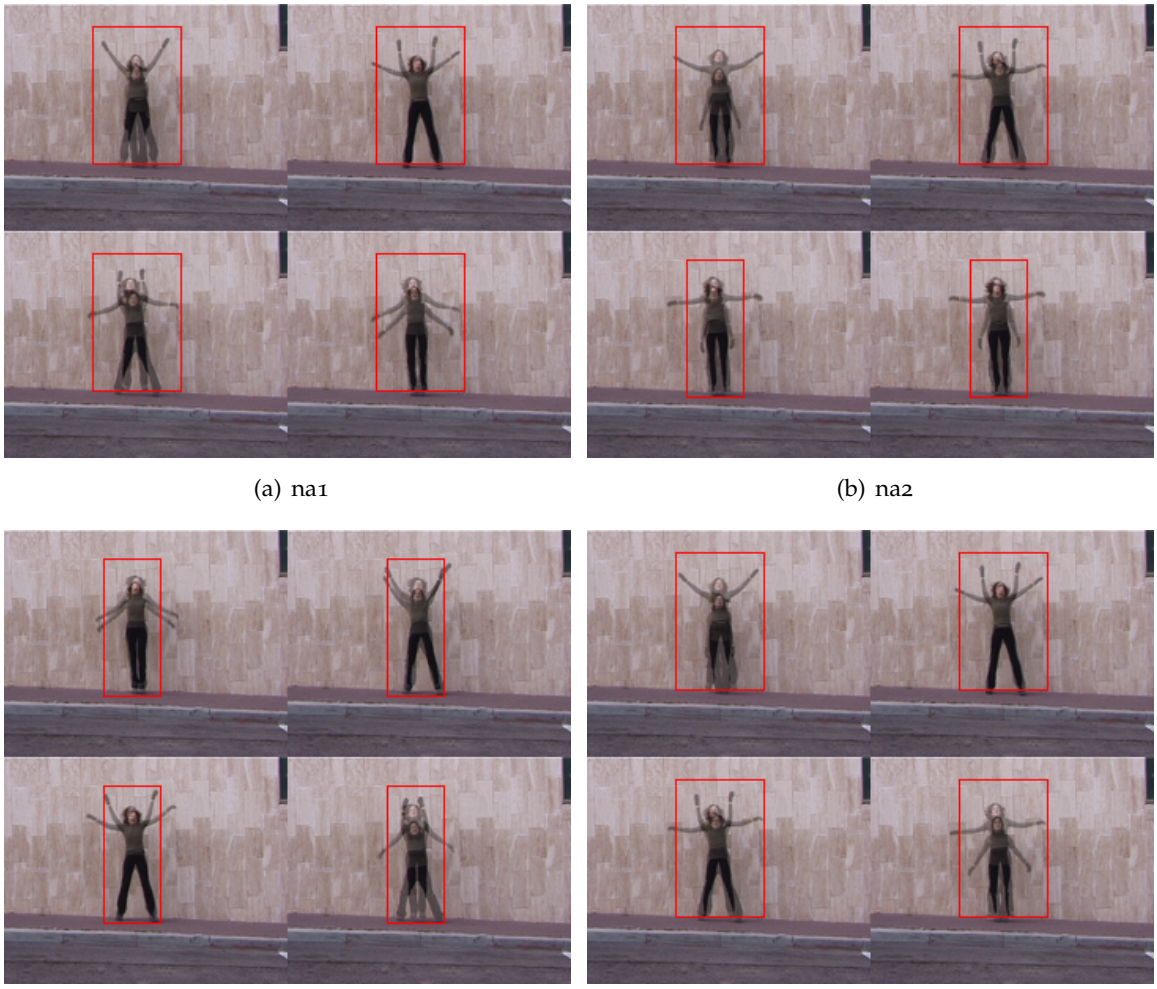
In folgenden Abbildungen sind exemplarisch einige Ergebnisse der verschiedenen Tests der Trainingsdurchläufe dargestellt. Für jeden Trainingsdurchlauf sind dabei vier Bilder mit je einer positiv erkannten Region vorhanden. Die Detektion erfolgte jeweils zu verschiedenen Zeitpunkten. Jedes Bild ist dabei eine Überlagerung des ersten und letzten Frame des Volumens, welches momentan vom Detektor überdeckt wird.
Die Detektionsbilder wurden für jeden Trainingsdurchlauf zu vier Zeitpunkten ausgewählt. Oben links entspricht dem einer Detektion im fünften Frame, obern recht im 45'sten Frame, unten links im 80'sten Frame und unten rechts im 125'sten Frame.



(a) va1 (b) va2

(c) va3 (d) va4

Abbildung 6.9.: Positive Detektionen der Trainingsdurchläufe des volumetrischen Ansatzes



(a) na1 (b) na2

(c) na3 (d) na4

Abbildung 6.10.: Positive Detektionen der Trainingsdurchläufe des neuen Ansatzes

7. Zusammenfassung und Ausblick

In dieser Arbeit ist ein Verfahren zur Erkennung von Aktionen in Videos entstanden, bei dem Rechteckmerkmale verwendet werden. Diese sind zweidimensional und werden zeitlich an mehreren Stellen ausgewertet. Dadurch sollte die Evaluationsdauer gesenkt werden. Diese Merkmale werden während des Trainings in einer Kaskade angeordnet. Um die Dauer der Auswertung eines Videos zu verringern, wurde ein Blockmatching der Suchfensterregionen implementiert. Im Vergleich des neuen Verfahrens mit einer, in der Literatur empfohlenen Methode, zeigte das Verfahren mit Bewegungsvorhersage ein besseres oder gleich gutes Detektionsverhalten. Jedoch dauerte die Auswertung eines Videos um ein vielfaches länger.

Ausblick

Momentan ist das Verfahren mit Bewegungsvorhersage noch nicht völlig ausgereift. Für praktische Anwendungen ist es momentan zu langsam, obwohl das hier zeitraubende Blockmatching generell echzeitfähig ist. Das selbe gilt für die Entscheidungskaskade mit Rechteckmerkmalen.
Wie in dieser Arbeit gezeigt wurde, ist die Hauptursache dafür das Blockmatching. Der nächste Schritte wäre daher diesen Teil zu beschleunigen. Eine Möglichkeit dafür wäre beispielsweise eine parallele Auswertung mehrerer Kaskadenknoten.
Alternativ dazu kann der Blockmatching-Algorithmus modifiziert werden, in dem nur wenige bedeutsame Merkmale genutzt werden um eine Bewegungsrichtung zu bestimmen. Momentan werden alle Merkmale die ein Knoten beinhaltet dazu genutzt. Für die Videorepräsentation sollte, statt einer Vektor-Liste, eine andere Datenstruktur gewählt werden, bei der Elemente schnell hinzugefügt und entfernt werden können. Außerdem sollte sie über eine schnelle Schiebeoperation verfügen.
Das Verfahren zu beschleunigen ist jedoch nicht die einzige Möglichkeit es zu verbessern. Es kann hier, ähnlich wie bei anderen Methoden, eine Nachverarbeitung der erkannten Bereiche durchgeführt werden. Denn alle Verfahren wiesen eine Vielzahl an positiv erkannten Regionen in der Umgebung der Aktion auf. Diese kann dafür verwendet werden, Aktionen mit einer höheren Sicherheit zu klassifizieren.

A. UML-Diagramme des volumetrischen Ansatzes

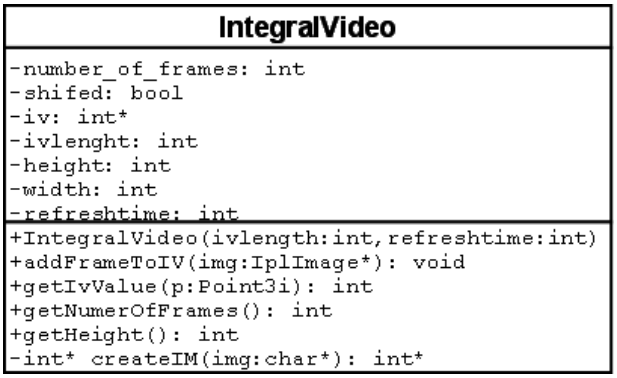


Abbildung A.1.: Klasse IntegralVideo

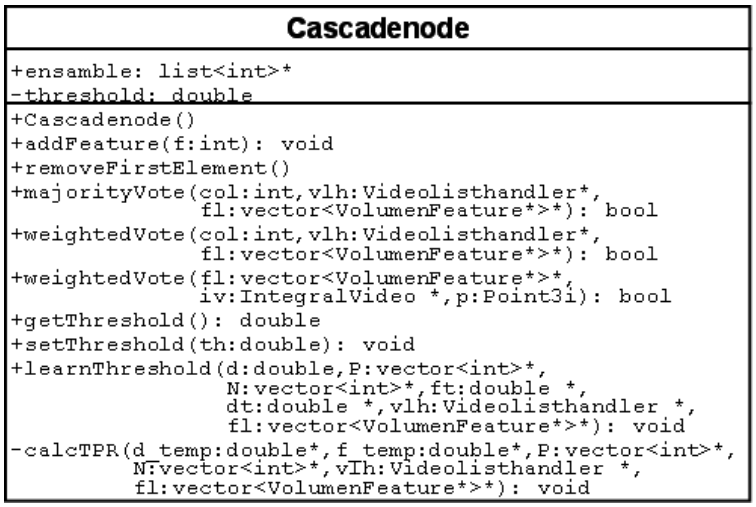


Abbildung A.2.: Klasse Cascadenode

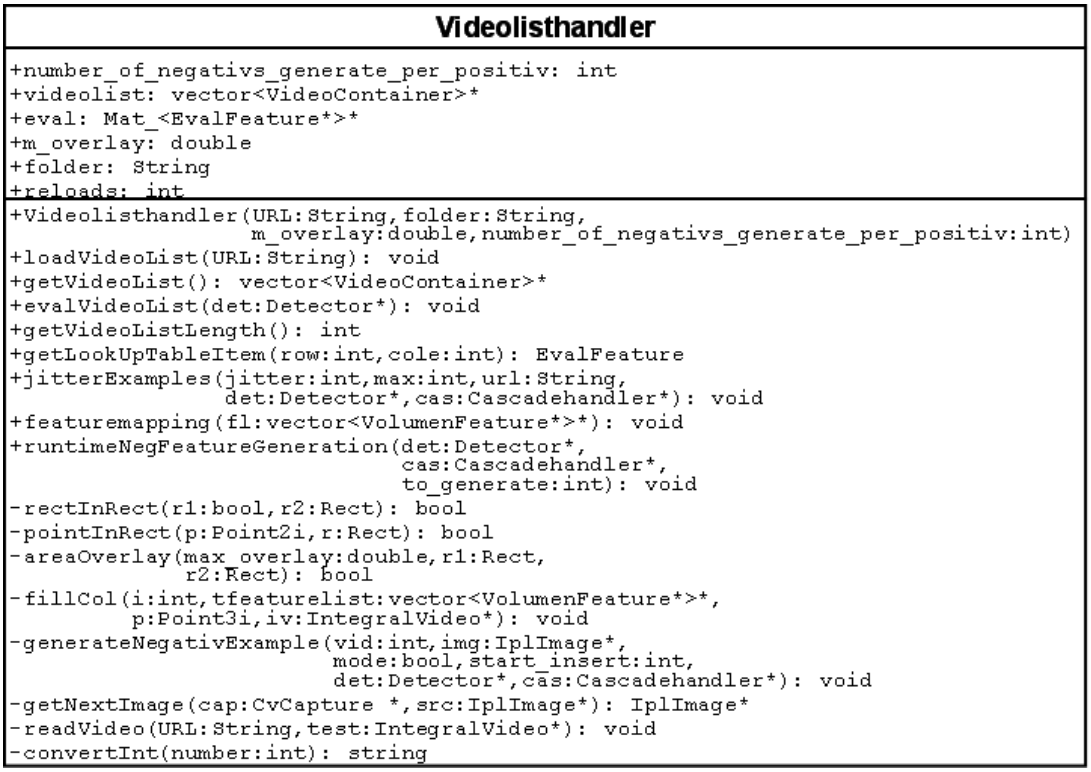


Abbildung A.3.: Klasse Videolisthandler

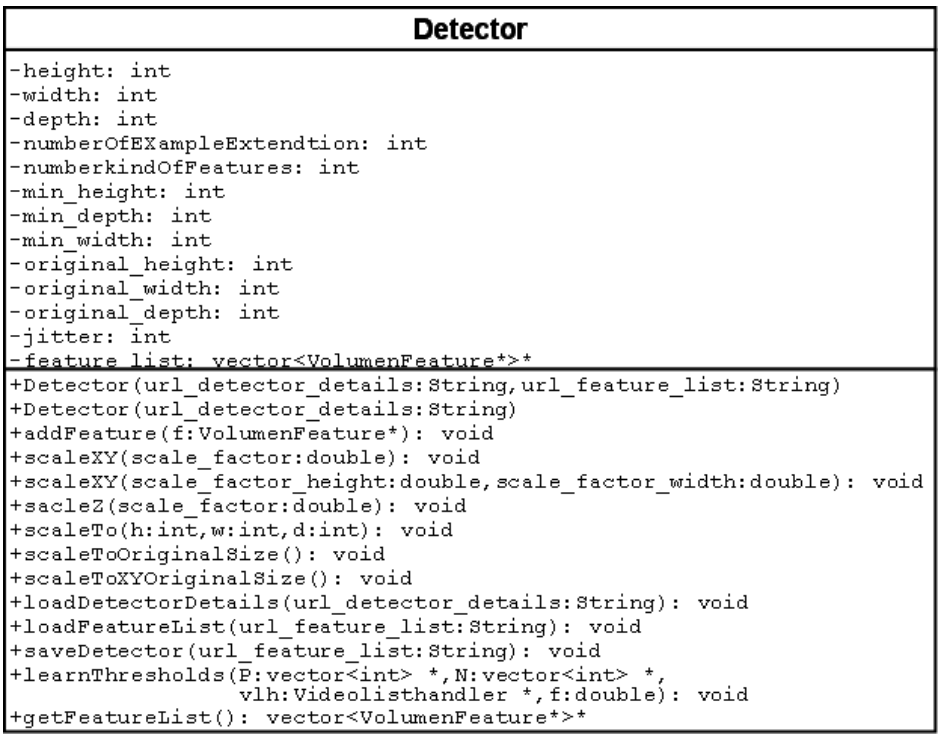


Abbildung A.4.: Klasse Detector

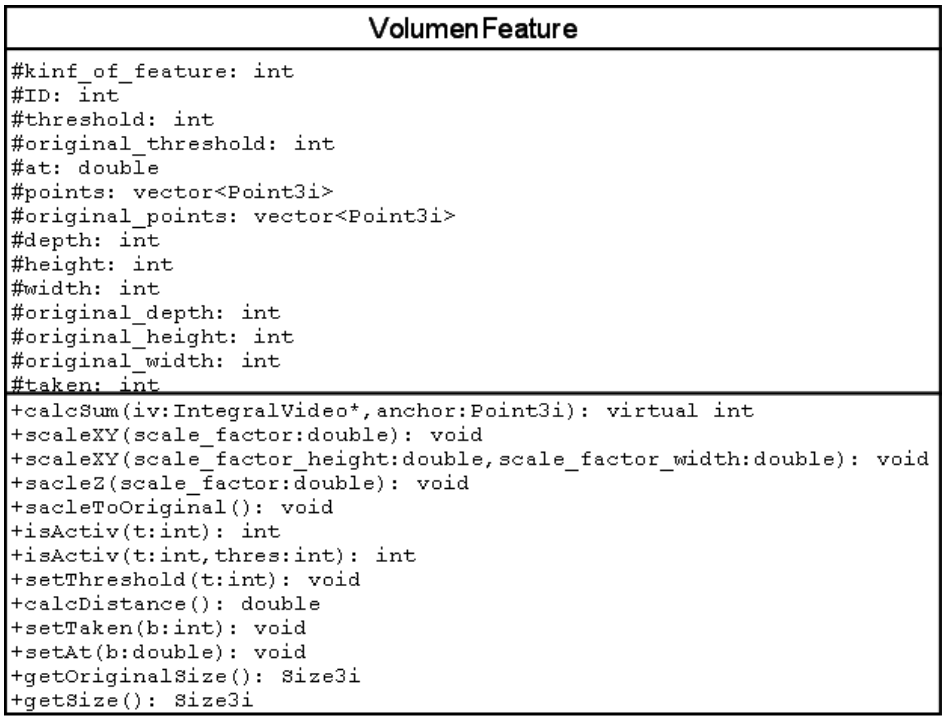


Abbildung A.5.: Klasse Volumen-Feature

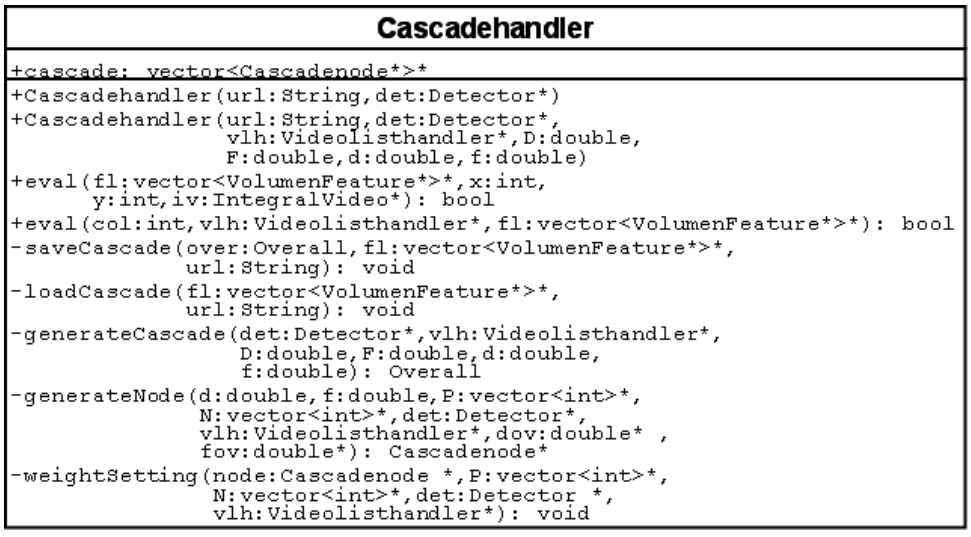


Abbildung A.6.: Klasse Cascadehandler

B. UML-Diagramme des Ansatzes mit Bewegungsvorhersage

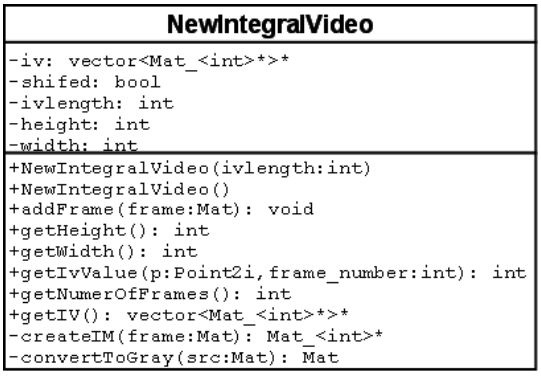


Abbildung B.1.: Klasse NewIntegralVideo

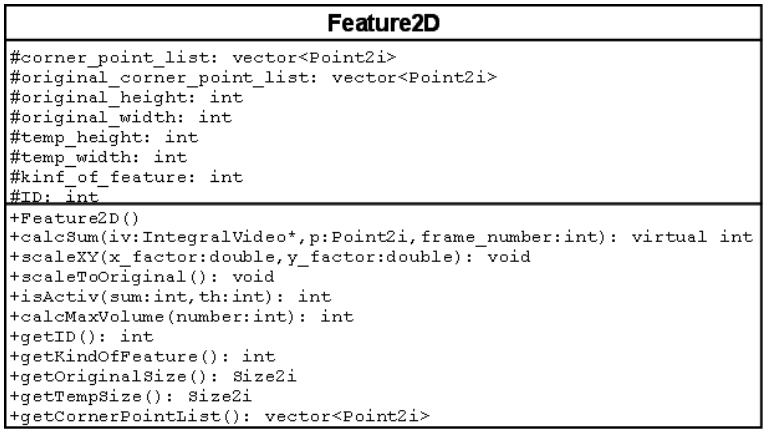


Abbildung B.2.: Klasse Feature2D

Abbildung B.3.: Klasse NewVideolisthandler

Abbildung B.4.: Klasse Stcf

Abbildung B.5.: Klasse NewDetector

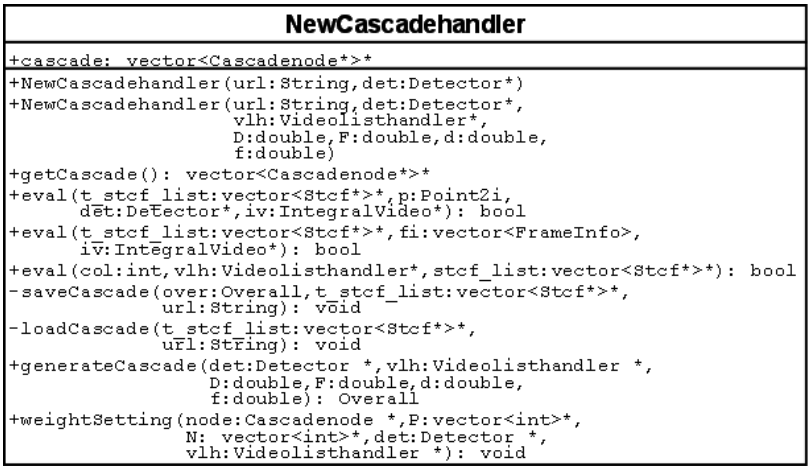


Abbildung B.6.: Klasse NewCascadehandler

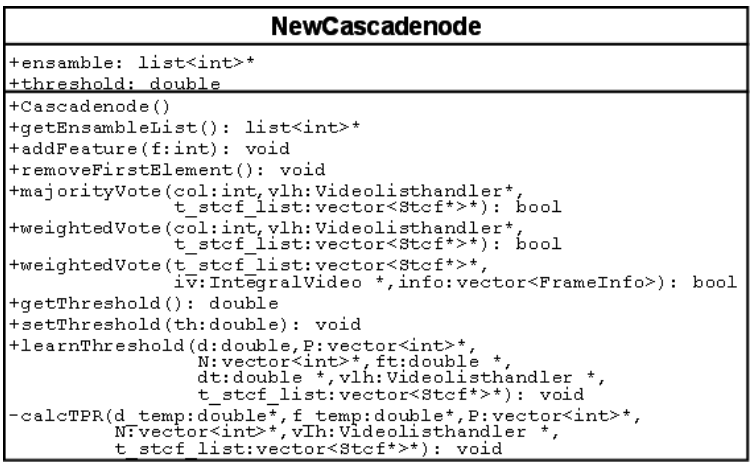


Abbildung B.7.: Klasse NewCascadenode

Literaturverzeichnis

[DLS+09] O. Duchenne, I. Laptev, J. Sivic, F. Bach, J. Ponce. Automatic annotation of human actions in video. In *Proc. Int. Conf. Comp. Vis.(ICCV'09)*. Kyoto, Japan, 2009. URL http://www.di.ens.fr/~fbach/iccv09_duchenne.pdf.

[DRCB05] P. Dollár, V. Rabaud, G. Cottrell, S. Belongie. Behavior recognition via sparse spatio-temporal features. In *In VS-PETS*, pp. 65–72. 2005. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.5712&rep=rep1&type=pdf.

[Gha90] M. Ghanbari. The cross-search algorithm for motion estimation. *IEEE Transactions on Communications*, 38:950 – 953, 1990. URL http://www.di.unipi.it/~lonetti/PhDCourse/ME/CrossSearch.pdf.

[HBC08] A. Hervieu, P. Bouthemy, J.-P. L. Cadre. Video event classification and detection using 2D trajectories. In *PROC. OF THE INT. CONF. ON COMPUTER VISION THEORY AND APPLICATIONS, VISAPP'08*. 2008. URL http://www.irisa.fr/vista/Papers/2008-visapp-hervieu.pdf.

[KSH05] Y. Ke, R. Sukthankar, M. Hebert. Efficient Visual Event Detection using Volumetric Features. In *International Conference on Computer Vision*, volume 1, pp. 166–173. 2005. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.4874&rep=rep1&type=pdf. (Zitiert auf den Seiten 18, 21 und 45)

[Les05] G. Leshem. Improvement of Adaboost Algorithm by using Random Forests as Weak Learner and using, 2005. URL http://shum.huji.ac.il/~gleshem/Guy_Leshem_Proposal.pdf.

[LKP03] R. Lienhart, E. Kuranov, V. Pisarevsky. Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection. In *In DAGM 25th Pattern Recognition Symposium*, pp. 297–304. 2003. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.9688&rep=rep1&type=pdf.

[LM02] R. Lienhart, J. Maydt. An Extended Set of Haar-Like Features for Rapid Object Detection. In *IEEE ICIP 2002*, pp. 900–903. 2002. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.9433&rep=rep1&type=pdf.

[LMSR08] I. Laptev, M. Marszalek, C. Schmid, B. Rozenfeld. Learning realistic human actions from movies. In *Proc. Conf. Computer Vision and Pattern Recog. (CVPR'08)*, pp. 1–8. Anchorage, Alaska, 2008. URL http://www.irisa.fr/vista/Papers/2008_cvpr_laptev.pdf. (Zitiert auf Seite 17)

[LP07] I. Laptev, P. Pérez. Retrieving actions in movies. In *Proc. Int. Conf. Comp. Vis.(ICCV'07)*, pp. 1–8. Rio de Janeiro, Brazil, 2007. URL http://www.irisa.fr/vista/Papers/2007_iccv_laptev.pdf. (Zitiert auf Seite 17)

[LT96] Y.-C. Lin, S.-C. Tai. Fast Full-Search Block-Matching Algorithm for Motion Compensated Video Compression. *Pattern Recognition, International Conference on*, 3:914, 1996. doi:http://doi.ieeecomputersociety.org/10.1109/ICPR.1996.547301. URL http://www.stanford.edu/class/ee398b/handouts/papers/Lin-FastMotionCompensation.pdf.

[PM96] L.-M. Po, W.-C. Ma. A Novel Four-Step Search Algorithm for Fast Block Motion Estimation. *IEEE Trans. Circuits Syst. Video Technol*, 6:313–317, 1996.

[VJ02] P. Viola, M. Jones. Robust Real-time Object Detection. *International Journal of Computer Vision*, 57(2):137–154, 2002. URL http://research.microsoft.com/en-us/um/people/viola/Pubs/Detect/violaJones_IJCV.pdf. (Zitiert auf Seite 9)

[WBTVG09] G. Willems, J. Becker, T. Tuytelaars, L. Van Gool. Exemplar-based action recognition in video. In *BMVC09*. 2009. URL http://class.inrialpes.fr/pub/willems-bmvc09.pdf.

[WRM03] J. Wu, J. M. Rehg, M. D. Mullin. Learning a Rare Event Detection Cascade by Direct Feature Selection. In *In NIPS*. MIT Press, 2003. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.8770&rep=rep1&type=pdf. (Zitiert auf den Seiten 13 und 45)

[WTG08] G. Willems, T. Tuytelaars, L. J. V. Gool. An Efficient Dense and Scale-Invariant Spatio-Temporal Interest Point Detector. In D. A. Forsyth, P. H. S. Torr, A. Zisserman, editors, *ECCV (2)*, volume 5303 of *Lecture Notes in Computer Science*, pp. 650–663. Springer, 2008. URL http://class.inrialpes.fr/pub/willems-eccv08.pdf.

[ZM00] S. Zhu, K.-K. Ma. A new diamond search algorithm for fast block-matching motion estimation. *IEEE Transactions on Image Processing*, 9(2):287–290, 2000. URL http://www3.ntu.edu.sg/home/ekkma/1_Publications_files/AwACTIONSf.

[ZTD09] W. Zhang, R. Tong, J. Dong. Boosting 2-Thresholded Weak Classifiers over Scattered Rectangle Features for Object Detection. *Journal of Multimedia*, 4:397–404, 2009. URL http://ojs.academypublisher.com/index.php/jmm/article/viewFile/0406397404/1302.

Alle URLs wurden zuletzt am 20.12.2010 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Rudolf Netzel)