Freie Universität Berlin

Institut für Informatik
Studiengang Informatik

GREENGRASS
VISION SYSTEMS

Bachelorarbeit

Entwicklung eines Personenzählsystems auf Basis Haar-ähnlicher Merkmale

Markus Schmidt

Matrikelnummer: [redacted]
m-schmidt-berlin@web.de

10. Juni 2014

Betreuer: Dr. Hamid Mobalegh und Dr. Tim Landgraf

Zusammenfassung

Die Aufgabe eines Personenzählsystems ist die Ermittlung der Anzahl an vorbeigehenden Personen innerhalb eines zu betrachtenden Bereichs. Die daraus gewonnenen Daten sind vor allem für den Einzelhandel von großer Bedeutung, da so optimale Entscheidungen abhängig der Anzahl an Kunden getroffen werden können. In dieser Bachelorarbeit wird die Entwicklung eines kamerabasierten Personenzählsystems vorgestellt, das ein- und ausgehende Personen separat zählt. Als Grundlage dazu dienen Haar-ähnliche Merkmale, wie sie bereits aus der Gesichtserkennung bekannt sind. Dabei werden Trefferquoten von bis zu 92 % bei einer Rate falschen Alarms von unter 7 % erreicht.

Inhaltsverzeichnis

1 Einleitung

Diese Bachelorarbeit befasst sich mit der Planung und der Implementierung eines Personenzählsystems. Unter einem Personenzählsystem ist ein System zu verstehen, welches die Anzahl der vorbeigehenden Personen innerhalb eines bestimmten Bereichs misst. Anwendungsgebiete von Personenzählsystemen liegen im Einzelhandel, um Kassen- und Personaleinsatz zu optimieren, in der Sicherheitsbranche, um beispielsweise einer Überfüllung vorzubeugen, in öffentlichen Einrichtungen, um das Staatsbudget gerecht zu verteilen, oder dienen generell als statistisches Instrument [10].

Neben der nur kurzzeitig anwendbaren Methode, Zählungen manuell durch einen mechanischen Handzähler durchzuführen, gibt es photoelektrische Sensoren (Lichtschranken) und Sensoren, die auf Druck, Vibration oder andere Merkmale reagieren [2]. Solche Sensoren können meist die Bewegungsrichtung der Personen nicht erkennen und versagen bei größeren Ansammlungen von Personen. Für bidirektionale Messungen, also solche, die zwischen eingehenden und ausgehenden Personen unterscheiden, eignen sich kamerabasierte Systeme [2].

In dieser Bachelorarbeit wird ein Personenzählsystem vorgestellt, das Aufnahmen einer einzelnen, senkrecht nach unten zeigenden Kamera auswertet. Hierbei wird eine Methode verwendet, die Haar-ähnliche Merkmale benutzt, um Personen zu erkennen. Diese Methode ist bereits aus der Gesichtserkennung bekannt. Als Tracking wird auf die Ungarische Methode zurückgegriffen. Mit diesen Mitteln ist es möglich, einen Algorithmus zu entwickeln, der zwischen eingehenden und ausgehenden Personen unterscheidet und diese separat zählt. Dabei wird großer Wert auf Datenschutz gelegt.

2 Verwandte Arbeiten

Im Wesentlichen besteht ein kamerabasiertes Personenzählsystem aus einer Erkennungs- und einer Trackingphase. Einige bekannte Personenzählsysteme basieren auf Kameradaten, bei denen jeweils die Differenz der Pixelwerte eines Einzelbildes und seinem Nachfolger [3] oder dem Hintergrund [1,14] als Grundlage für die Erkennung dient. In [25] werden Haar-ähnliche Merkmale benutzt, um ganze Körper aus einer nicht senkrechten Position zu erkennen, wodurch Ausfälle durch Verdeckungen hervorgerufen werden. Für das Tracking werden Template-Matching [25] oder Partikelfilter [33] benutzt. Zudem gibt es weltweit Unternehmen, die sich auf Personenzählsysteme spezialisiert haben. Einige große deutsche Unternehmen lauten Vitracom [32], Visapix [31] und LASE PeCo [12].

Diese Bachelorarbeit stellt ein Personenzählsystem vor, das lediglich eine handelsübliche, videofähige Kamera verlangt, ohne kostenaufwendige Software zu benötigen.

3 Spezifikationen und verwendete Datentypen

Als Softwaregrundlage des gesamten Projekts diente Open Source Computer Vision (OpenCV) [17]. OpenCV ist eine frei verfügbare Ansammlung von Algorithmen, insbesondere aus den Bereichen der Bild- und Videoverarbeitung sowie des maschinellen Lernens, die als Schnittstellen in verschiedenen Programmiersprachen unter allen üblichen Betriebssystemen zur Verfügung gestellt werden. Im Rahmen dieser Bachelorarbeit wurde die C++-Implementierung der Version 2.4.7 genutzt, welche unter Windows 7 mit MinGW [15] durch CMake [4] kompiliert wurde. Durch die Abwärtskompatibilität von OpenCV ist es möglich, spätere Versionen zum Kompilieren der Projektdateien zu benutzen.

Jegliche Maßnahmen, wie das Programmieren, das Training oder Messungen, fanden auf einem AMD-Phenom-II-System mit $4 \times 3{,}6\,\text{GHz}$ und 4 GB Arbeitsspeicher statt. Full-HD-Videoaufnahmen wurden mit einer Canon-Digitalkamera der IXUS-Reihe gemacht, wobei anzumerken ist, dass die HD-Funktion für das entwickelte Personenzählsystem nicht zwingend erforderlich ist.

Neben komplexen Algorithmen bietet OpenCV simple Schnittstellen für Punkte, Rechtecke und andere Datenstrukturen an, von denen in dieser Arbeit Gebrauch gemacht wird.

- Ein Punkt wird in OpenCV als `Point` und in dieser Arbeit mit (x, y) bezeichnet.
- Ein Rechteck ist als `Rect` implementiert, mit der oberen linken Ecke (x, y), der Breite w (engl. width) und der Höhe h (engl. height).
- Die Komponenten eines Punktes oder eines Rechteckes sind natürliche Zahlen, die den Datentyp `int` besitzen.
- Eine Liste eines bestimmten Typs ist in den meisten Fällen durch die C++-Datenstruktur `vector` realisiert und wird in dieser Arbeit ebenfalls Vektor genannt.
- Als Bild oder Bildfenster wird die Datenstruktur `Mat` verwendet. Diese besitzt die Breite W und die Höhe H und enthält in jedem Punkt RGB-Farbinformationen oder einen Grauwert. Der Grauwert Y stellt die Helligkeit eines Pixels dar und lässt sich bestimmen mit $Y = 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B$, wobei R, G und B für den Rot-, Grün- beziehungsweise Blauanteil stehen [29]. Ein Punkt oder ein Rechteck liegen stets innerhalb der Grenzen eines Bildes, es gelten also $0 \leq x < W$, $0 \leq y < H$, $0 < x + w \leq W$ und $0 < y + h \leq H$.

4 Training

4.1 Merkmale

Das Training sowie die Erkennung geschehen auf Basis Haar-ähnlicher Merkmale (engl. Haar-like features). Erstmals wurde diese Methode 1998 von Papageorgiou et al. [24] beschrieben, um Personen, Gesichter und andere Objekte in statischen Bildern zu erkennen. Der Name leitet sich aus der Ähnlichkeit zu Haar-Wavelets ab, welche bereits im Jahr 1909 von Alfréd Haar [9] vorgestellt wurden.

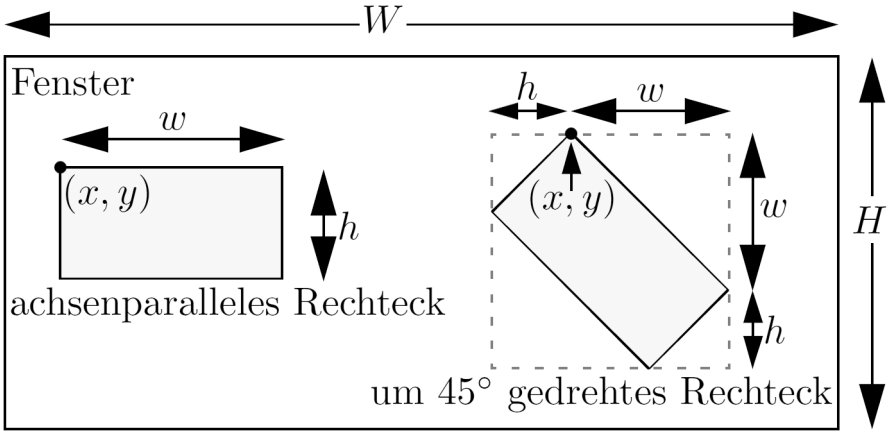

Abbildung 1: Beispielhafte Darstellung eines achsenparallelen und eines um 45° gedrehten Rechteckes innerhalb eines Fensters mitsamt ihren Koordinaten und Längenangaben (übersetzt und leicht verändert aus [13])

Ein Haar-ähnliches Merkmal ist eine Darstellung des Unterschieds zwischen den durchschnittlichen Grauwerten von Teilregionen eines Bildes oder Bildfensters. Das einfachste Merkmal ist eine achsenparallele, rechteckige Region, die in zwei gleich große, rechteckige Teilflächen zerteilt wird. Den Wert des Merkmals erhält man, indem man die Summe der Grauwerte der einen Fläche von der Summe der Grauwerte der anderen Fläche subtrahiert. Ursprünglich benutzten Viola et al. [30] achsenparallele Merkmale mit zwei, drei und vier Flächen. Von Lienhart et al. [13] wurden zudem Merkmale eingeführt, die um 45° gedreht sind, welche die Rate für falschen Alarm bei einer gegebenen Trefferquote senken. Sie alle können in der Breite sowie in der Höhe skaliert werden. In OpenCV sind die ursprünglichen und die um 45° gedrehten Merkmale implementiert, die allesamt von dieser Bachelorarbeit in Anspruch genommen werden. Um Merkmale möglichst effizient zu berechnen, wird jeder Fläche eine Gewichtung zugewiesen. Die Gesamtfläche erhält dabei die Gewichtung -1 und die schwarzen Flächen eine entsprechend höhere Gewichtung (siehe Abbildung 2). Die Summen der Flächen werden mit ihrer jeweiligen

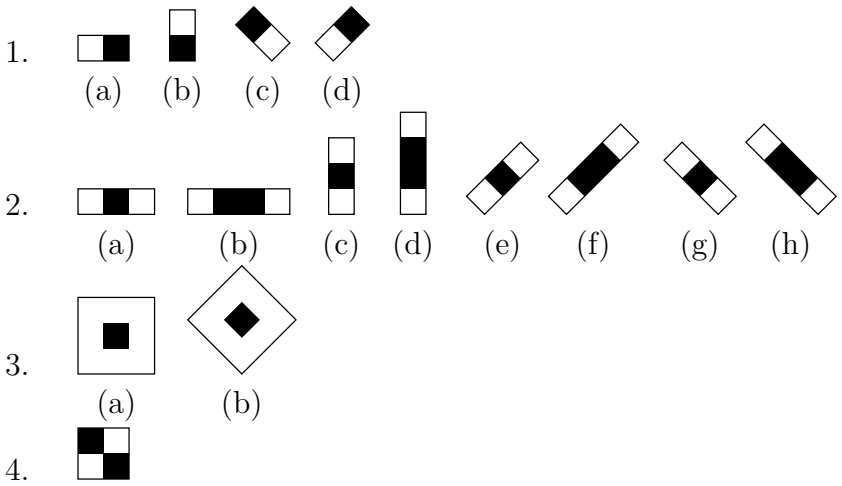

Abbildung 2: Prototypen aller in OpenCV verfügbaren sowie in [13] verwendeten Haar-ähnlichen Merkmale. Die Bezeichnungen stammen ebenfalls aus [13]. Weiße Flächen erhalten eine negative Gewichtung und schwarze Flächen eine positive Gewichtung und werden dann addiert. Man beachte, dass die Merkmale der 1. bis 3. Kategorie mit nur zwei Rechtecken und Merkmale der 4. Kategorie mit nur drei Rechtecken berechnet werden können. Dazu wird die Gesamtfläche eines Merkmals mit -1 und die schwarzen Flächen jeweils mit 2 gewichtet. Um den Größenunterschied auszugleichen, erhalten die schwarzen Flächen in 2. (a), (c), (e) und (g) stattdessen jeweils die Gewichtung 3 und solche in der 3. Kategorie jeweils die Gewichtung 9.

Gewichtung multipliziert und dann addiert. Durch die Benutzung eines Integralbildes, welches im nächsten Abschnitt erläutert wird, geschieht die Berechnung einer Fläche in konstanter Zeit.

4.2 Integralbild

Um die Grauwertsumme einzelner Flächen eines Bildes effizient zu berechnen, wird das Konzept des Integralbildes angewandt. Erstmals wurde es 1984 von Franklin Crow [6] erwähnt und später von Viola et al. [30] aufgegriffen.

Ein Integralbild wird aus einem Originalbild erstellt und besitzt die gleichen Abmessungen wie solches [26]. In jedem Eintrag des Integralbilds wird die Summe der Grauwerte aller Bildpunkte, die oberhalb und links davon liegen, sowie des Punkts selbst, festgehalten. Formal ausgedrückt bedeutet das:

$$i_i(x, y) = \sum_{x' \leq x,\, y' \leq y} i_o(x', y'),$$

wobei $i_i(x, y)$ der Wert des Integralbilds an der Stelle (x, y) und $i_o(x, y)$ der Grauwert

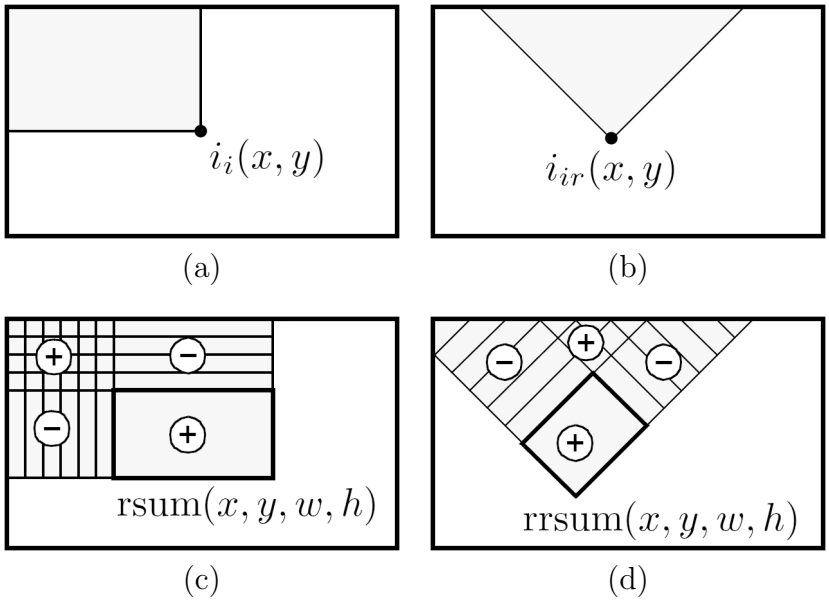

Abbildung 3: Integralbild für (a) achsenparallele und (b) um 45° gedrehte Rechtecke; Berechnungsschema der (c) achsenparallelen und (d) um 45° gedrehten Rechtecke (leicht verändert aus [13])

des originalen Bilds an der Stelle (x, y) ist. In linearer Laufzeit[1] lässt sich das komplette Integralbild erstellen:

$$i_i(x, y) = i_i(x - 1, y) + i_i(x, y - 1) - i_i(x - 1, y - 1) + i_o(x, y),$$

mit

$$i_i(-1, y) = i_i(x, -1) = i_i(-1, -1) = 0.$$

Die Summe der Grauwerte eines gegebenen achsenparallelen Rechtecks (x, y, w, h) lässt sich nun mit vier Zugriffen auf das Integralbild berechnen (siehe Abbildung 3):

$$\begin{aligned}\text{rsum}(x, y, w, h) = {} & i_i(x + w - 1, y + h - 1) - i_i(x - 1, y + h - 1) \\ & - i_i(x + w - 1, y - 1) + i_i(x - 1, y - 1).\end{aligned}$$

Die im letzten Abschnitt angesprochenen um 45° gedrehten Merkmale können ebenfalls effizient mithilfe eines abgewandelten Integralbildes berechnet werden [13]. Ein Eintrag $i_{ir}(x, y)$ in diesem bezeichnet die Fläche eines um 45° gedrehten Rechtecks, dessen untere Ecke im Punkt (x, y) liegt und nach oben verläuft, bis die Grenzen des Gesamtbildes erreicht sind. Die formelle Beschreibung lautet:

$$i_{ir}(x, y) = \sum_{y' \leq y,\, |x - x'| \leq y - y'} i_o(x', y').$$

[1] linear bezüglich der Anzahl der Bildpunkte des originalen Bildes

Auch die Berechnung dieses Integralbilds geschieht in linearer Laufzeit[1]:

$$i_{ir}(x,y) = i_{ir}(x-1,y-1) + i_{ir}(x+1,y-1) - i_{ir}(x,y-2) + i_o(x,y) + i_o(x,y-1),$$

mit

$$i_{ir}(-1,y) = i_{ir}(x,-1) = i_{ir}(x,-2) = i_{ir}(-1,-1) = i_{ir}(-1,-2) = 0.$$

Damit kann die Summe der Grauwerte eines um 45° gedrehten Rechtecks (Bezeichnungen siehe Abbildung 1 auf Seite 7) mit vier Zugriffen auf das Integralbild errechnet werden:

$$\begin{aligned}\mathrm{rrsum}(x,y,w,h) = {} & i_{ir}(x-h+w, y+w+h-1) - i_{ir}(x-h, y+h-1) \\ & - i_{ir}(x+w, x+w-1) + i_{ir}(x, y-1).\end{aligned}$$

4.3 Klassifikator

Damit später Objekte erkannt werden können, wird zunächst ein Klassifikator benötigt. Ein auf Haar-ähnlichen Merkmalen beruhender Klassifikator soll ein Bildfenster fester Größe als Eingabe erhalten und bestimmen, ob es sich dabei um ein Objekt der gewünschten Art handelt. Unterschiedliche Lernverfahren können verwendet werden, um einen solchen Klassifikator zu trainieren. Es bietet sich ein Boosting-Algorithmus an, der viele schwache Klassifikatoren zu einem Gesamtklassifikator zusammenfügt [30]. Dabei ist ein schwacher Klassifikator ein Entscheidungsbaum oder -stumpf, der einzelne Haar-ähnliche Merkmale als Knoten besitzt.

In [13] wurden verschiedene Varianten des AdaBoost-Algorithmus verglichen: Discrete AdaBoost (DAB), Real AdaBoost (RAB) und Gentle AdaBoost (GAB). Dabei war GAB nicht nur der beste (höchste Trefferquote zu gegebener Rate falschen Alarms), sondern auch der schnellste der drei Algorithmen. Aus diesem Grund wird GAB als Lernalgorithmus in dieser Bachelorarbeit benutzt. Weiterhin wurden in [13] verschiedene andere Parameter verglichen. So wurden die besten Ergebnisse bei einer Fenstergröße von 20×20 Pixeln mit einer Entscheidungsbaumhöhe von mehr als 1 erzielt, sodass diese Parameter für diese Arbeit übernommen werden. Eine Baumhöhe von 2 (statt 1) soll zudem einen möglichen visuellen Unterschied zwischen hinauf- und hinabgehenden Personen ausgleichen.

Um die korrekte Klassifizierung zu steigern und dabei die Geschwindigkeit zu erhöhen, wird ein Konzept namens Kaskaden-Klassifikator oder kurz Kaskade verwendet [30]. Hierbei werden mehrere Klassifikatoren hintereinandergeschaltet. Nur wenn der erste Klassifikator ein positives Ergebnis zurückgibt, geschieht die Auswertung des zweiten Klassifikators, erst dann die Evaluierung des dritten und so weiter. Erst wenn die komplette Kaskade mit einem positiven Ergebnis durchlaufen wird, ist das Gesamtergebnis positiv. Sobald das Ergebnis eines Klassifikators negativ ist, gilt das Gesamtergebnis als negativ. Das hängt damit zusammen, dass man davon ausgeht, dass die größten Bereiche eines Bildes keine Objekte enthalten und man diese so schnell wie möglich

verwerfen möchte, um fortzufahren. Im Gegenzug gibt es nur wenige Regionen, die Objekte der gewünschten Art enthalten, deren Präsenz man mithilfe der mehrstufigen Kaskade bestätigen möchte [30]. In [13] wurde eine 20-stufige Kaskade mit einer minimalen Trefferquote von 0,999 sowie einer maximalen Falschen-Alarm-Rate von 0,5 pro Stufe verwendet, sodass man eine Trefferquote von $0{,}999^{20} \approx 0{,}98$ und eine Falsche-Alarm-Rate von $0{,}5^{20} \approx 9{,}6 \cdot 10^{-7}$ erwarten kann. Diese Parameter werden auch für diese Bachelorarbeit verwendet.

4.4 Trainingsablauf

Im Folgenden werden die durchgeführten Schritte von der Vorbereitung bis zur fertig trainierten Kaskade erläutert. Die Objekte, welche vom Kaskaden-Klassifikator gelernt und später erkannt werden sollen, stellen dabei von oben aufgenommene Köpfe dar, weil solche in den meisten Szenen stets unverdeckt sind.

Sammeln negativer Bilder Für das Training werden negative Bilder benötigt. Dies sind beliebige Bilder, die jedoch nicht das zu trainierende Objekt, in diesem Fall Köpfe, enthalten dürfen. Als Grundlage wurde der Satz negativer Bilder von Naotoshi Seo [28] benutzt, zu dem Hintergrundbilder der aufgenommenen Szenen hinzugefügt wurden. Die gesamte Anzahl an negativen Bildern lag damit bei 3037. Die Dateipfade aller negativen Bilder wurden in einer Textdatei zusammengefasst.

Extrahieren der Einzelbilder Um das manuelle Auswählen der Trainingsdaten vorzubereiten, wurde ein simples Programm `extractFrames` geschrieben, welches mit Mitteln der OpenCV-Bibliothek die Einzelbilder eines Videos extrahiert und sie als Bilddateien speichert. Dieses Programm wurde auf alle aufgenommenen Videodateien angewandt.

Auswählen der positiven Bilder Zur Auswahl der positiven Daten wurde ein ebenfalls von Naotoshi Seo entwickeltes Programm namens *ImageClipper* [27] verwendet. Dieses Programm ermöglicht es, aus Bilddateien Ausschnitte per Hand auszuwählen. Die Ausschnitte werden dabei als separate Bilder gespeichert, deren Dateinamen den ursprünglichen Namen sowie die Koordinaten und die Größe des Ausschnitts enthalten, sodass eine nachträgliche genaue Zuordnung möglich ist. Das Programm wurde leicht modifiziert, um nur quadratische Bildausschnitte zuzulassen. Auf diese Weise konnten aus den extrahierten Videobildern Ausschnitte ausgewählt werden, die einen Kopf umrahmen. Insgesamt wurden so 6089 Köpfe aus 3360 Einzelbildern ausgewählt.

Formatieren der positiven Ausschnitte Um das passende Format für den nächsten Schritt zu liefern, bot sich das Perl-Skript `haartrainingformat.pl` [27] an, welches im Umfang von ImageClipper enthalten ist. Da die Dateinamen der Ausschnitte aus dem

vorigen Schritt bereits alle Informationen enthielten, genügte es, deren Pfade in einer Textdatei zu speichern, welche dann als Argument an das Skript übergeben wurde.

Erstellen der positiven Samples Das Trainingsprogramm von OpenCV nimmt positive Samples nur in Form einer sogenannten Vektordatei an. Um eine solche zu erstellen, wird ein separates Programm innerhalb von OpenCV angeboten: `opencv_createsamples`. In diesem Schritt wird ebenfalls die Fenstergröße, in denen sich die Haar-Merkmale befinden werden, festgelegt. Alle eingegebenen Samples werden auf diese Größe skaliert und gegebenenfalls in Graustufen umgewandelt, um in einer Vektordatei mit der Endung `.vec` gespeichert zu werden.

Trainieren der Kaskade Liegen die Textdatei, welche die Pfade zu den negativen Bildern enthält, und die Vektordatei, welche die skalierten positiven Samples beinhaltet, vor, kann das Training mit dem Programm `opencv_traincascade` gestartet werden. Dieses ist eine Weiterentwicklung von `opencv_haartraining`, welches mittlerweile obsolet ist [18].

Ein Parameter von `opencv_traincascade` legt den Pfad zu einem Verzeichnis fest, in dem die trainierte Kaskade als `.xml`-Datei gespeichert wird. Dadurch dass die Stufen der Kaskade ebenfalls gespeichert werden, kann das Training nach einer Unterbrechung bei der letzten Stufe fortgesetzt werden. Das komplette Training dauerte ungefähr anderthalb Tage.

Zusätzlich wurden für diese Arbeit folgende Parameter benutzt:

`-numPos 4500`	In jeder Stufe der Kaskade wurden 4500 positive Samples benutzt.
`-numNeg 3037`	In jeder Stufe wurden alle 3037 negativen Bilder verwendet.
`-numStages 20`	Die Anzahl der Stufen wurde auf 20 gesetzt.
`-w 20 -h 20`	Die Fenstergröße für die Haar-ähnlichen Merkmale betrug 20×20 Pixel.
`-minHitRate 0.999`	In jeder Stufe sollte eine Trefferquote von mindestens 99,9 % erreicht werden.
`-maxFalseAlarmRate 0.5`	Jede Stufe sollte mit einer Falschen-Alarm-Rate von höchstens 50 % abgeschlossen werden.
`-maxDepth 2`	Die maximale Tiefe der Knoten der Entscheidungsbäume betrug 2.
`-mode ALL`	Es wurde aus dem gesamten Arsenal der Haar-ähnlichen Merkmale geschöpft.

4.5 Datenschutz

Nachdem die Kaskade trainiert ist, wird für das eigentliche Personenzählsystem lediglich die entstandene `.xml`-Datei benötigt. Diese enthält keinerlei Daten über Personen, sondern nur die beim Training benutzten Parameter, Informationen über die Stufen der Kaskade sowie die damit verbundenen Haar-ähnlichen Merkmale. Jegliche andere Dateien können somit gelöscht werden. Möchte man erneut ein Training durchführen, so genügt die Vektordatei mit der Endung `.vec`. Sie besteht aus den in Graustufen umgewandelten Bildausschnitten der von oben aufgenommenen Köpfe der Größe 20×20 Pixel, wodurch eine Unkenntlichkeit erreicht wird. So bleibt in jedem Fall die Privatsphäre gewahrt.

5 Der Algorithmus

Der Algorithmus des Personenzählsystems unterteilt sich in drei grobe Teile: die Erkennung, die Gruppierung und das Tracking. Diese werden in einem Programm zusammengefasst.

5.1 Erkennung

Die Erkennung versucht, innerhalb eines Bildes Objekte jener Art, welche bei der Kaskade trainiert wurden, ausfindig zu machen.

Dazu wird ein Fenster mit der beim Training angegeben Größe (hier 20×20 Pixel) an der oberen linken Ecke des Bildes positioniert. An dieser Stelle wird der Kaskaden-Klassifikator aufgerufen. Ist das Ergebnis positiv, wird ein Rechteck ausgegeben, welches die aktuelle Position und Größe des Fensters darstellt. Unabhängig davon, ob ein Objekt gefunden wurde, wird das Fenster um einen Bildpunkt verschoben, wo erneut der Klassifikator aufgerufen wird. Auf diese Weise wird das Fenster weiter verschoben, bis das komplette Bild durchsucht ist. Um jedoch auch Objekte zu erkennen, die größer als die ursprüngliche Fenstergröße sind, wird das Fenster um einen bestimmten Faktor vergrößert[2], welches erneut das gesamte Bild durchläuft. Dieser Vorgang wird so oft wiederholt, bis die Fenstergröße die angegebene maximale Größe oder die Bildgröße überschreitet.

Für die Detektion stellt OpenCV eine Klasse namens `CascadeClassifier` zur Verfügung. Dem Konstruktor übergibt man den Pfad der als `.xml`-Datei gespeicherten Kaskade und ruft dann die Funktion `detectMultiScale` [19] des erstellten Objekts auf. Hierbei werden ein Bild, auf welchem die Detektion durchgeführt werden soll, und ein (leerer) Vektor, in den die Rechtecke um die erkannten Objekte abgelegt werden, als Argumente angegeben. Optional können folgende weitere Argumente angegeben werden:

`scaleFactor` gibt den Wert an, mit welchem die Höhe und die Breite des Fensters nach jedem Durchgang multipliziert werden. Standardmäßig ist `1.1` eingestellt, was bedeutet, dass sich nach jedem Durchgang die Fensterdimensionen um 10 % vergrößern.[2]

`minNeighbors` ist die minimale Anzahl an Nachbarn pro Cluster nach dem Aufruf von `groupRectangles`, welche im nächsten Abschnitt genauer erläutert wird. Als Standardwert ist `3` eingestellt.

[2] Aus Effizienzgründen wird nach jedem Durchlauf jedoch nicht das Fenster vergrößert, sondern das gesamte Bild um den angegebenen Faktor verringert [22].

`flags` wird nur für eine alte Kaskade benutzt und bleibt bei einer neuen Kaskade unberücksichtigt.[3]

`minSize` gibt die minimale Fenstergröße der Merkmale als `Size(w,h)` an. Objekte kleiner als diese werden ignoriert. Standardmäßig wird die Größe nicht eingeschränkt.

`maxSize` gibt die maximale Fenstergröße der Merkmale als `Size(w,h)` an. Objekte größer als diese werden ignoriert. Wird kein Wert übergeben oder ist eine der Komponenten `0`, werden die Abmessungen des Gesamtbildes als obere Schranke gewählt.

Ist die Erkennungsphase abgeschlossen, liegen die Ergebnisse in Form von Rechtecken vor. Jegliche weitere Informationen des Bildes oder voriger Bilder bleiben unberücksichtigt. So werden auch keine Daten über Personen erhoben.

5.2 Optionales Clustering

Oftmals treten viele Erkennungen in unmittelbarer Nähe zum eigentlichen Objekt auf, sodass es sich anbietet, diese zusammenzufassen [13]. OpenCV bietet dazu eine Funktion namens `groupRectangles` [20] an, die einen Partitionierungsalgorithmus mithilfe einer Union-Find-Struktur [5] in quadratischer Laufzeit[4] implementiert [21]. Sie wird nach der Detektion innerhalb von `detectMultiScale` aufgerufen, wobei der Parameter `minNeighbors` weitergereicht wird. Jedes Cluster, das weniger als `minNeighbors` + 1 Elemente besitzt, wird verworfen. Um volle Kontrolle über alle Erkennungen zu haben, kann man `0` als Wert übergeben, was dazu führt, dass kein Clustering erfolgt und alle Rechtecke erhalten bleiben.

5.3 Gruppierung

In den folgenden Schritten wird lediglich mit den Mittelpunkten der Rechtecke gearbeitet, da für den Algorithmus die Größen der Rechtecke irrelevant und nur die Mittelpunkte dieser entscheidend sind. Im Idealfall stellen diese die Mittelpunkte der Köpfe dar. Die Funktion `getCenters` erwartet einen Vektor mit Rechtecken und gibt die entsprechenden Mittelpunkte ebenfalls als Vektor gleicher Größe zurück. Den Mittelpunkt (x_c, y_c) eines Rechtecks erhält man mit:

$$(x_c, y_c) = \left(x + \left\lfloor \frac{w}{2} \right\rfloor, y + \left\lfloor \frac{h}{2} \right\rfloor \right),$$

wobei auf eine Rundung verzichtet wird.

[3] Hier konnte früher optional unter anderem die Bildskalierung statt der Fensterskalierung aktiviert werden.

[4] quadratisch bezüglich der Anzahl der ursprünglichen Rechtecke

Unabhängig davon, ob bereits ein Clustering stattfand, wird im Programm des Personenzählsystems als nächster Schritt die eigens für diese Arbeit entwickelte Funktion `group` (siehe Algorithmus 1 im Anhang) aufgerufen. Ähnlich dem Clustering hat sie die Aufgabe, jeweils nahe beieinanderliegende Punkte zu einer Person zusammenzufassen. Im Besonderen sollen dadurch erkannte Schultern einer Person zugeordnet werden, anstatt als einzelne Personen aufgefasst zu werden. Der Funktion werden zwei Parameter übergeben: ein Vektor aus Punkten und eine natürliche Zahl `groupingRadius`. Letzterer gibt den maximalen euklidischen Abstand zum Mittelpunkt einer Gruppe an, damit ein Punkt der entsprechenden Gruppe angehört. Zurückgegeben wird ein Vektor, der die Mittelpunkte der Gruppen enthält, welche jedoch als Kreise um den Mittelpunkt mit dem Radius `groupingRadius` aufgefasst werden können.

5.4 Tracking

Nach der Gruppierung liegen die Erkennungen als Vektor aus Punkten vor. Für das Tracking werden der Vektor v_1 eines Einzelbilds und der Vektor v_2 des darauffolgenden Einzelbilds betrachtet. Ziel ist es, jeweils zwei Punkte aus verschiedenen Vektoren so miteinander zu verknüpfen, dass sie in beiden Bildern derselben Person angehören.

Zunächst wurde ein Ansatz gewählt, der stets den kleinsten euklidischen Abstand zwischen zwei freien, vektorfremden Punkten sucht und diese dann verknüpft, bis ein Vektor keine freien Punkte mehr besitzt oder der kleinste Abstand über einem gewissen Schwellenwert liegt. Letztere Eigenschaft verhindert, dass weit entfernte Punkte miteinander verknüpft werden können. Ein großes Problem bei dieser Herangehensweise ist folgendes: Blockiert ein Punkt, der mehrere mögliche Nachbarn besitzt, einen Platz, der die einzige Möglichkeit für einen anderen Punkt darstellt, so wird eine optimale Verteilung verhindert.

Es wurde ein Algorithmus benötigt, der die Punkte der Vektoren möglichst gut verknüpft und dabei die Anzahl an Verknüpfungen maximiert. Die Punkte der zwei Vektoren können ebenfalls als gewichteter bipartiter Graph aufgefasst werden, dessen Kantengewichte dem euklidischen Abstand zwischen den betreffenden Punkten entspricht. Nun gilt es, das Maximum-Matching [5] dieses Graphen zu finden, bei dem das Gesamtgewicht minimal ist. Genau dies erreicht eine Implementierung, die auf der Ungarischen Methode beruht, welche 1955 von Harold Kuhn [11] vorgestellt und 1957 von James Munkres [16] bezüglich der Laufzeit[5] auf $\mathcal{O}(n^3)$ verbessert wurde.

Die zugehörige Funktion lautet `correlate` (siehe Algorithmus 2 im Anhang), welche zwei Vektoren aus Punkten (der zwei Einzelbilder) und eine natürliche Zahl `maxDistance` erwartet. Die Zahl wurde aus der ursprünglichen Implementierung übernommen und dient wie bei der Gruppierung als Schwellenwert: Zwei Punkte, deren euklidischer Abstand

[5] mit n als der Anzahl der zuzuordnenden Elemente. Im Rahmen dieser Bachelorarbeit gilt $n = \max(|v_1|, |v_2|)$.

größer ist als dieser, können nicht miteinander korrelieren. Es wird also davon ausgegangen, dass diese beiden Punkte nicht dieselbe Person darstellen.

5.5 Gebietstracking

In den ersten Versionen des Personenzählsystems wurde eine simple Messlinie genutzt, um Personen zu zählen. Wenn das Tracking des vorigen Schritts ergibt, dass ein Punkt aus v_1 oberhalb oder genau auf der Messlinie mit einem Punkt aus v_2 unterhalb der Linie verknüpft ist, wird der Zähler für die hinablaufenden Personen um eins erhöht. Analog gilt dies für die Zählung der hinauflaufenden Personen: Liegt ein Punkt aus v_1 unterhalb oder auf der Linie und sein Korrelat aus v_2 oberhalb der Linie, dann wird der entsprechende Zähler um eins erhöht. Das führte dazu, dass kleine Änderungen zwischen zwei Einzelbildern, mehr durch die Ungenauigkeit der Erkennung als durch die Bewegung der Person selbst, fälschlicherweise eine Zählung auslösen konnten.

Um dieses Verhalten zu verhindern, wurde die Messlinie durch ein Mess*gebiet* ersetzt, das von Personen zu durchqueren ist, um gezählt zu werden. Im Detail bedeutet das, dass die Spur einer Person tatsächlich verfolgt wird. Falls eine Person das Gebiet von oben betritt, dann erfolgt eine Zählung nach unten erst, wenn sie es im unteren Bereich verlässt. Betritt eine Person das Gebiet von unten, dann wird eine Zählung nach oben erst durchgeführt, wenn sie es im oberen Bereich verlässt. Tritt der Fall ein, dass die Person das Gebiet an einer der drei anderen Seiten verlässt oder die Spur, zum Beispiel wegen mangelnder Erkennung, verloren geht, wird sie nicht weiterverfolgt, bis die Person erneut oben oder unten eintritt. Kann einer Gruppe keine Spur zugeordnet werden und befindet sich der Mittelpunkt der Gruppe innerhalb des Gebiets, so wird die Gruppe verworfen, damit Spuren anderer Personen nicht unterbrochen werden. Anders ausgedrückt beginnt eine Spur stets außerhalb des Gebiets.

5.6 Implementierung

Eine simple Implementierung stellt eine solche dar, welche lediglich ein Einzelbild aus der Eingabe-Videodatei einliest und dann die Funktionen der Erkennung, der Gruppierung und des Trackings hintereinander aufruft. Dabei stellt man jedoch fest, dass der Vorgang der Erkennung auf einem Bild mit einer Auflösung von 1920×1080 Pixeln mehrere Sekunden dauern kann, was für ein Personenzählsystem inakzeptabel ist. Um die Verarbeitungszeit zu verkürzen, hilft es, das Bild vor der Erkennungsphase zu verkleinern, indem man es skaliert. Außerdem ist es nicht nötig, die Erkennung auf dem gesamten Bild durchzuführen, sondern nur in einem Bereich um das Gebiet des Trackings herum. Allein diese Maßnahmen erhöhen die Geschwindigkeit bei der Erkennung und um ein Vielfaches und sind deshalb unverzichtbar im Programm. Weitere Geschwindigkeitsoptimierungen und deren Einbußen in der Erkennungsrate werden in den nächsten Kapiteln erwähnt.

Ein Schnappschuss des Programms ist in der Abbildung 4 zu sehen.

Dem Programm werden folgende Parameter übergeben, welche hintereinander eingegeben und lediglich durch Leerzeichen getrennt werden. Näheres zum Hintergrund der Parameter findet man in den entsprechenden Abschnitten und Hinweise zur Verwendung im Abschnitt 6.2 des nachfolgenden Kapitels.

`xmlFile` *Klassifikator* – Dateipfad zur trainierten Kaskade, welche als `.xml`-Datei vorliegt

`videoFile` *Allgemein* – Dateipfad zum Video, auf dem die Zählung angewendet werden soll; es werden alle FOURCC-Codecs [8] unterstützt, die auf dem System installiert sind

`scaleFactor` *Erkennung* – Skalierungsfaktor, mit dem die Fensterdimensionen nach jedem Erkennungsdurchgang multipliziert werden

`minNeighbors` *Optionales Clustering* – minimale Anzahl an Nachbarn pro Cluster; ist der Wert `0`, geschieht kein Clustering

`minSize` *Erkennung* – einzelner Wert, der beide Dimensionen der minimalen Fenstergröße festlegt; Objekte kleiner als diese werden ignoriert

`maxSize` *Erkennung* – einzelner Wert, der beide Dimensionen der maximalen Fenstergröße festlegt; Objekte größer als diese werden ignoriert

`trackDist` *Tracking* – maximaler Abstand zwischen zwei aus verschiedenen Einzelbildern stammenden Punkten, damit sie verbunden werden können

`groupRadius` *Gruppierung* – maximaler Abstand eines jeden Punkts einer Gruppe zum Mittelpunkt dieser

`detectHeight` *Allgemein* – Höhe des Gebiets („Erkennungsgebiet“), in dem die Erkennung durchgeführt wird

`trackHeight` *Gebietstracking* – Höhe des Gebiets („Trackinggebiet“), das durchquert werden muss, um eine Zählung auszulösen

`videoScale` *Allgemein* – Faktor, mit dem die Dimensionen jedes Einzelbilds multipliziert werden; dabei werden `minSize`, `maxSize`, `trackDist`, `groupRadius`, `detectHeight` und `trackHeight` ebenfalls skaliert

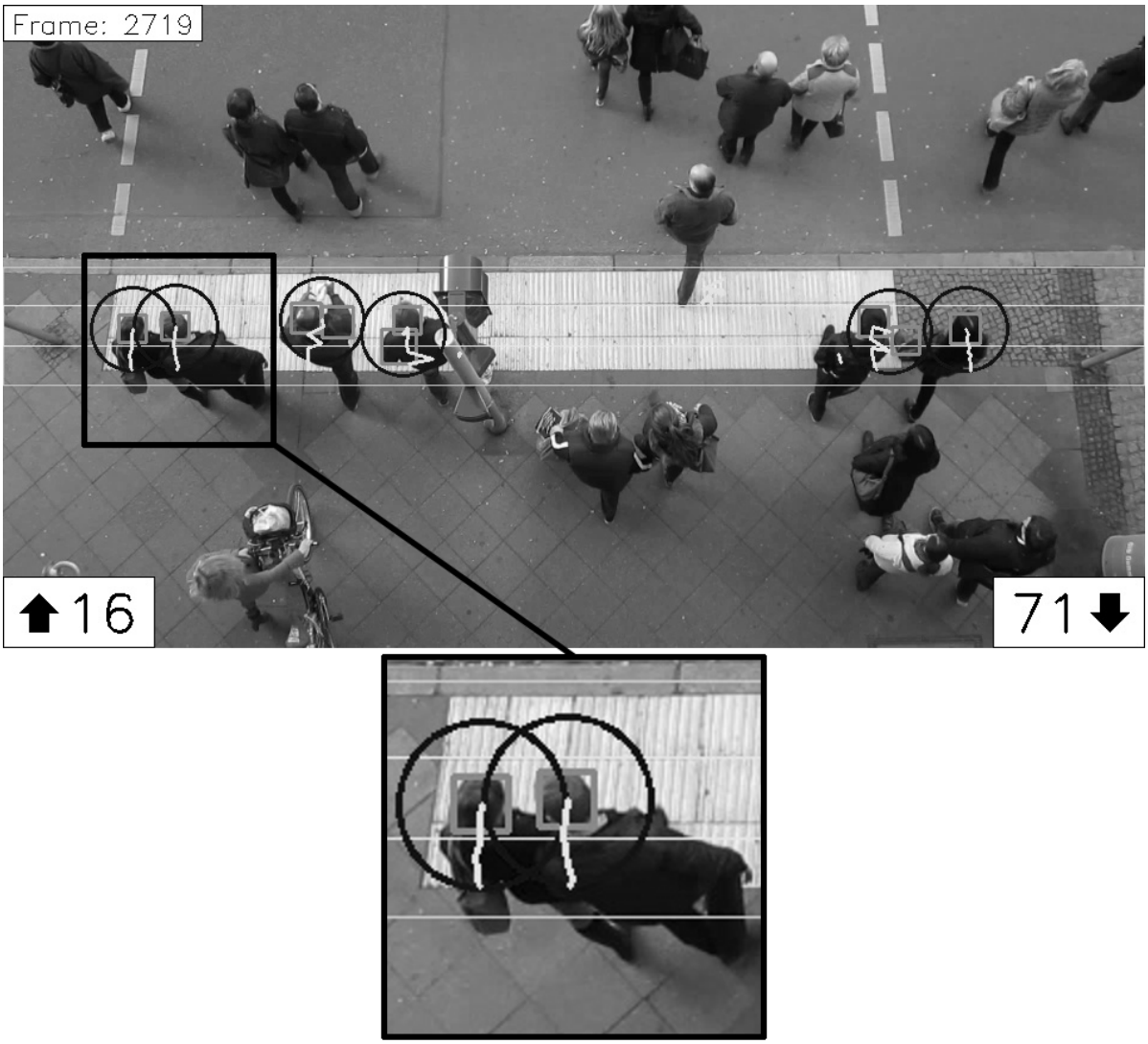

Abbildung 4: Ein Schnappschuss des Programms und die Vergrößerung eines Bildausschnitts daraus. In der oberen linken Ecke ist die derzeitige Nummer des Einzelbildes zu sehen und in der unteren linken/rechten Ecke die gesamte Anzahl der bisher gezählten hinauf-/hinabgegangenen Personen. In der Mitte des Bildes begrenzen die äußeren Geraden das Erkennungsgebiet, die inneren das Trackinggebiet. Durchquert eine Person das komplette innere Gebiet, so löst sie eine Zählung der entsprechenden Richtung aus. Rechtecke zeigen Erkennungen der Köpfe, Kreise die zusammengefassten Erkennungen und Linien deren Spuren.

6 Parametereinstellungen

Es wird auf die Parametereinstellungen des Programms in Bezug auf die Geschwindigkeit bei der Erkennung und die Benutzung eingegangen.

6.1 Beschleunigung der Erkennung

Im Gegensatz zur Gruppierung und zum Tracking, welche zusammen meist weit weniger als eine Millisekunde benötigen, kann die Objekterkennung auf einem Bild viel Zeit in Anspruch nehmen. Um dem entgegenzuwirken, werden hier generelle Maßnahmen zur Erhöhung der Ausführungsgeschwindigkeit bei der Erkennung vorgestellt.

Dadurch dass sich der Parameter `videoScale` auf beide Dimensionen der Einzelbilder auswirkt, kann man erwarten, dass sich bei einer Halbierung von `videoScale` die Ausführungszeit der Erkennung auf ein Viertel reduziert. Eine Halbierung von `detectHeight` halbiert die Größe des Erkennungsgebiets und führt somit dazu, dass die Erkennungsdauer um die Hälfte reduziert wird.

Innerhalb der Erkennungsphase vergrößern sich die Dimensionen des Fensters nach jedem Durchgang jeweils um den angegebenen Faktor s (`scaleFactor`), sodass jede Dimension des Fensters nach n Durchgängen das s^n-Fache der jeweiligen Dimension des ursprünglichen Fensters besitzt. Geht man davon aus, dass die Erkennung bei der minimalen Fenstergröße $l \times l$ (`minSize`) beginnt und endet, sobald die maximale Fenstergröße $u \times u$ (`maxSize`) überschritten wird, folgt für $l \leq u$, dass $\lfloor \log_s(\frac{u}{l}) \rfloor + 1$ Durchgänge absolviert werden, wenn man interne Rundungen unbeachtet lässt.[6] So liegt die exakte Anzahl an Durchgängen bei Benutzung eines Skalierungsfaktors von 1,1 und einer Fenstergröße von 20×20 Pixeln auf einem Bild mit den Abmessungen von 1920×1080 Pixeln bei $\lfloor \log_{1,1}(\frac{1080}{20}) \rfloor + 1 = 42$. Wird die maximale Größe zum Beispiel auf 70×70 Pixel beschränkt, so finden nur noch 14 Durchgänge statt, welche möglicherweise ausreichen, um dennoch alle relevanten Objekte zu erkennen.

Dabei bleibt anzumerken, dass die Anzahl der Berechnungen pro Durchgang leicht sinkt, da sich das Verhältnis der Gesamtgröße zur Fenstergröße verringert. Zudem verlängert sich die Ausführungszeit, je mehr potenzielle Objekte sich im Bild befinden, da mehr Stufen der Kaskade in Anspruch genommen werden und häufiger Rechtecke im Vektor abgelegt werden müssen. Das optionale Clustering hat darauf keinen Einfluss; es werden stets zunächst alle Erkennungen in dem Vektor abgelegt, bevor das Clustering stattfindet, dessen Ausführungsgeschwindigkeit selbst sehr hoch ist.

[6] Tatsächlich wird in OpenCV stets mit der beim Training verwendeten Fenstergröße begonnen und zur Überprüfung werden die Fensterdimensionen innerhalb eines Durchgangs gerundet [23].

6.2 Wahl der Parameterwerte

Für jede separate Szene sollten folgende Überlegungen bei der Einstellung der Parameter für das Personenzählsystem berücksichtigt werden.

Um die Größe der Objekte einzuschränken, sollte zunächst eine Abschätzung über das kleinste und größte in der Szene vorkommende Objekt vorgenommen werden. Im Fall der Kopferkennung stellt ein Kinderkopf meist das kleinstmögliche und eine Frau mit voluminösem Haar meist das größtmögliche Objekt dar. Jedes Einzelbild des Videos sollte nun so skaliert werden, dass die minimale Objektgröße der ursprünglichen Fenstergröße von 20×20 Pixeln entspricht. Den Skalierungsfaktor erhält man, indem man die Höhe oder die Breite des kleinsten Objekts durch die entsprechende Fensterdimension dividiert. Wird so verfahren, bedarf es keiner expliziten Angabe der minimalen Objektgröße und die maximale Objektgröße wird unverändert angegeben, da sie automatisch skaliert wird. Wurde beispielsweise festgestellt, dass im Originalvideo die Objektgröße mindestens 40×40 Pixel und maximal 70×70 Pixel beträgt, dann gibt man als `videoScale 0.5`, als `minSize 0` und als `maxSize 70` an.

Um mehrfache Erkennungen um Köpfe und Schultern zu einer Person zusammenzufassen, wird der Radius für die Gruppierung festgelegt, welcher sich aus der von oben betrachteten Breite der Körper ableitet. Als Maßstab bietet sich meist ein stämmiger Mann an. Ist die Breite beispielsweise 140 Pixel, so setzt man `groupRadius` auf `70`. Abhängig davon wählt man den maximalen Abstand für das Tracking. Er sollte etwas unter dem Gruppierungsradius liegen, damit die Spur einer Person korrekt verfolgt werden kann, ohne dass eine falsche Spur entsteht oder andere Spuren behindert werden. Um das Beispiel fortzuführen, wird `trackDist` auf `60` gesetzt.

Anschließend lässt sich ein Wert für die Höhe des Gebiets festlegen, durch das die Spur einer Person führen muss, damit eine Zählung ausgelöst wird. Die Höhe des Gebiets sollte größer als der Trackingabstand sein, um einer falschen Zählung vorzubeugen, die aus zwei willkürlich auftretenden Erkennungen auf unterschiedlichen Seiten des Gebiets entstehen kann. Daher wird in diesem Beispiel für `trackHeight` der Wert `70` gewählt. Die Höhe des Gebiets, auf dem die Erkennung durchgeführt wird, muss größer als dieser Wert sein, da das Tracking erfordert, dass eine Person mindestens einmal erkannt werden muss, *bevor* sie das Trackinggebiet betritt und *nachdem* sie es verlässt. So dienen der Erkennung zwei Eingangsbereiche, eines über und eines unter dem Trackinggebiet, wobei die Höhe jeweils nicht zu weit unter der maximalen Objektgröße liegen sollte. Rechnet man mit einer Eingangshöhe von etwa der maximalen Objekthöhe, kann man im Beispiel `detectHeight` auf `200` setzen. Danach wird `scaleFactor` für die Erkennung festgelegt; ein Wert von `1.1` deckt viele Objektgrößen ab. Abschließend möchte man `minNeighbors` meist auf `0` setzen, damit der Gruppierungsfunktion alle erkannten Objekte zur Verfügung stehen.

7 Auswertung

7.1 Auswertungsmethode

Zur Auswertung des Personenzählsystems wird zwischen drei verschiedenen Fällen unterschieden:

Richtig positiv Eine Person durchschreitet den Zählbereich und das System zählt diese korrekt.

Falsch positiv Das System zeigt fälschlicherweise eine Zählung an, die keiner Person zuzuordnen ist.

Falsch negativ Eine Person durchschreitet den Zählbereich und wird vom System fälschlicherweise nicht erfasst.

Mithilfe dieser Fälle lassen sich die Trefferquote (engl. hit rate) und die Falsche-Alarm-Rate (engl. false alarm rate) ermitteln, welche die Qualität des Systems beschreiben. Sie sind wie folgt definiert [7]:

$$\text{Trefferquote} = \frac{\text{Anzahl der richtig positiven Fälle}}{\text{Gesamtanzahl der Personen}}$$

$$\text{Falsche-Alarm-Rate} = \frac{\text{Anzahl der falsch positiven Fälle}}{\text{Gesamtanzahl der Zählungen}}$$

Für eine solche Auswertung wurde ein Programm namens `evaluateCounting` entwickelt, das die vollzogene Zählung eines Durchlaufs des Systems mit einer manuell durchgeführten Zählung vergleicht. Dem Programm werden Pfade zu vier Dateien übergeben, welche jeweils die vom System gemachte und manuelle Zählung der jeweils hinab- und hinaufgehenden Personen enthält. Jede Zeile einer solchen Datei enthält folgende durch Leerzeichen getrennte Informationen zu einer einzelnen Zählung: die Einzelbildnummer, die x-Koordinate und die y-Koordinate, wobei in der vorliegenden Version die y-Koordinate unberücksichtigt bleibt. Zusätzlich übergibt man die beim Testdurchlauf angegebene Videoskalierung, welche für die Angleichung der Koordinaten sorgt.

Innerhalb des Programms wird mit der Funktion `findConnections` (siehe Algorithmus 3 im Anhang) versucht, übereinstimmende Zählungen miteinander zu verbinden. Dies geschieht anhand zweier Abstandsmaße, dem Unterschied in den Einzelbildnummern (primär) und dem Abstand der x-Koordinaten (sekundär) von jeweils zwei Zählungen. Die maximalen Werte für diese Abstände werden ebenfalls dem Programm einzeln übergeben. Nach dem Aufruf der Funktion berechnet das Programm die Trefferquote sowie die Falsche-Alarm-Rate, separat für hinaufgehende und für hinabgehende Personen sowie für deren Gesamtheit, und gibt diese aus.

7.2 Evaluierung

Um konkrete Aussagen über die Qualität des Personenzählsystems treffen zu können, wurde ein Beispielvideo erstellt. Dabei handelt es sich um ein Video, das aus mehreren Einzelvideos der gleichen Szene besteht, aus denen Teile ohne sichtbare Personen herausgeschnitten wurden. Die Aufnahmen wurden von oben herab auf einen Gehweg gemacht, auf dem sich eine Fußgängerampel befindet, welche bewusst im Erkennungsgebiet liegt, um deren Einfluss auf das Personenzählsystem zu testen. Die Abmessungen des Videos betragen 1920×1080 Pixel, die Bildwiederholfrequenz ist 23,976 Bilder pro Sekunde und es besitzt eine Gesamtlänge von 2:45 Minuten.

Eine manuelle Zählung wurde auf diesem Video durchgeführt: Die darin vorkommenden Personen wurden per Hand in hinauf- und hinablaufende Personen unterteilt und es wurden zu jeder Person die Nummer des Einzelbilds sowie die Koordinaten, in denen sie das Trackinggebiet verlässt, festgehalten. Maßgeblich für die Zählungen sind die von oben betrachteten Mittelpunkte der Köpfe. So wurden insgesamt 212 Zählungen gemessen, 60 nach oben und 152 nach unten. In Hinsicht auf das Auswertungsprogramm schien für das verwendete Beispielvideo ein maximaler Abstand von 14 Einzelbildern und 200 Pixeln auf der x-Achse angemessen.

Die Parametereinstellung, die im Abschnitt 6.2 *Wahl der Parameterwerte* genannt wurden, führte zu einer Trefferquote von 91,04 % sowie einer Falschen-Alarm-Rate von 11,87 %. Dabei waren die Ausführungszeiten pro Einzelbild im Durchschnitt folgende: Die Vorbereitung[7] benötigte 6,87 ms, die Erkennung 84,4 ms, die Gruppierung 0,15 ms und das Tracking weniger als 10 μs. Die gesamte Ausführungszeit je Einzelbild betrug demnach ungefähr 91 ms und entspricht somit einer Bildwiederholfrequenz von etwa 11 Bildern pro Sekunde. Auf Grundlage dieser Parametereinstellung wurden einzelne Parameter verändert und das Ergebnis jeweils evaluiert. Eine Auswahl ist in der Abbildung 5 sichtbar. Dabei lassen sich folgende Beobachtungen machen:

- Generell kann man eine Korrelation der Trefferquote und der Falsche-Alarm-Rate beobachten: Erhöht sich einer der Werte, so steigt meist auch der andere; verringert er sich, verringert sich der andere ebenso.
- Man erkennt, dass das optionale Clustering sich negativ auf die Erkennungsrate auswirkt. Je höher die minimale Anzahl an Nachbarn ist, desto mehr Erkennungen gehen verloren, was im schlimmsten Fall zum Verlust der Spur führt. Aus diesem Grund ist es sinnvoll, das Clustering ausschalten, um einzig der Gruppierung die Zusammenfassung der Erkennungen zu überlassen.
- Kleine Änderungen des Skalierungsfaktors bei der Erkennung haben einen großen Einfluss auf die Ausführungsgeschwindigkeit, wobei sich die Trefferquote nur leicht verändert.

[7] Zur Vorbereitung gehört das ein Einlesen des Einzelbilds, das Beschränken dieses auf das Erkennungsgebiet, die Konvertierung in Graustufen sowie die Skalierung (siehe Abschnitt 5.6).

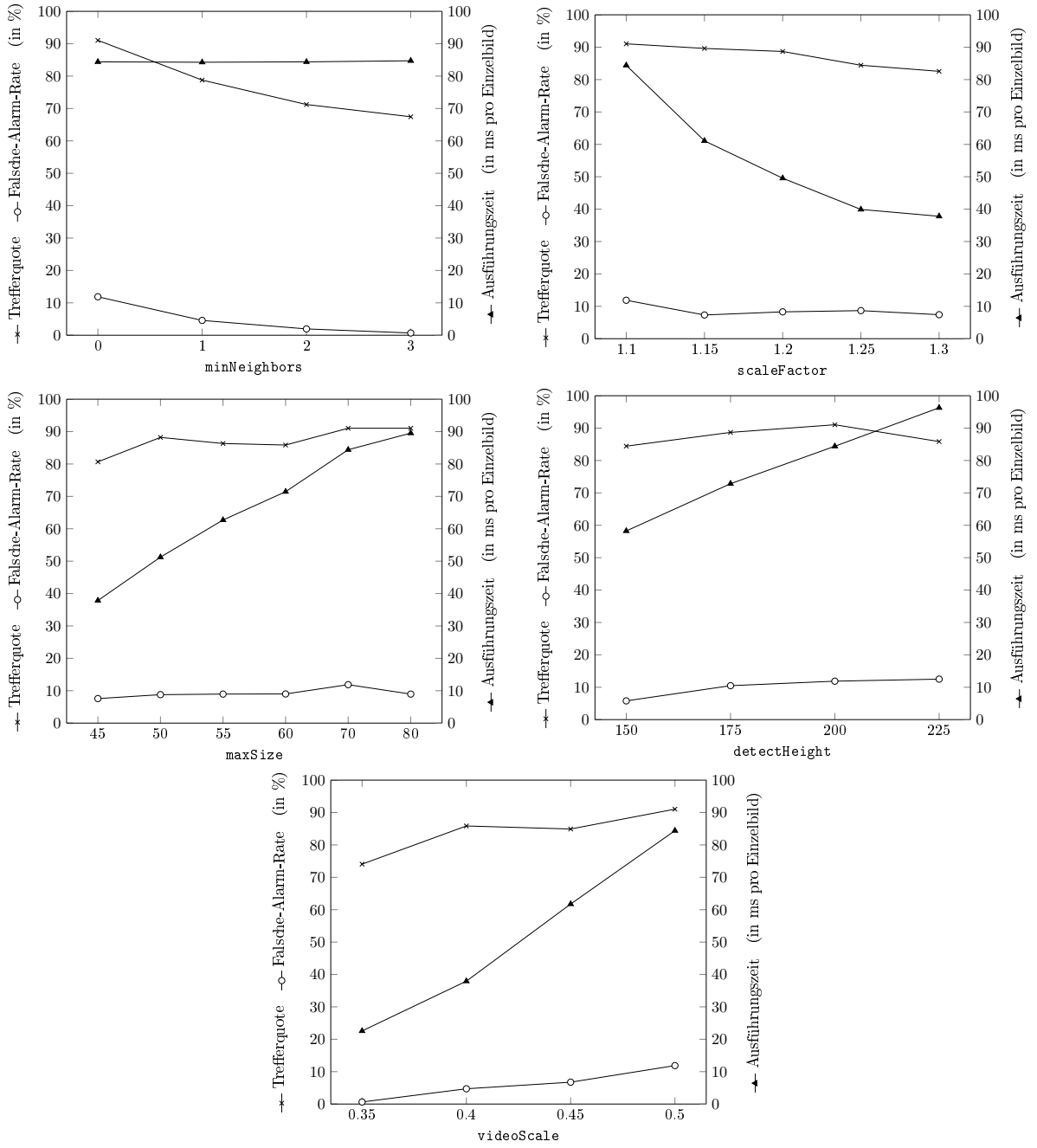

Abbildung 5: Graphen ausgewählter Parametereinstellungen. Die Grundeinstellung stammt aus dem Abschnitt 6.2, wobei jeweils ein Parameter variabel gehalten wird, welcher unter der entsprechenden Grafik benannt ist. Die Ausführungszeit bezieht sich nur auf die Erkennungsphase.

- Auch die maximale Fenstergröße kann die Ausführungsgeschwindigkeit entscheidend beeinflussen; die Trefferquote schwankt dabei. Ist der Wert allerdings zu klein eingestellt, sinkt auch die Trefferquote, da größere Köpfe nicht mehr erkannt werden.

- Bezüglich der Größe des Erkennungsgebiets kann keine klare Struktur in der Erkennungsrate erkannt werden. Die Angabe einer geringen Höhe des Gebiets dient hauptsächlich der Senkung der Ausführungszeit.

- Betrachtet man die Videoskalierung, so stellt man fest, dass hier die Ausführungszeit erheblich gesenkt werden kann. Liegt die dadurch resultierende minimale Fenstergröße allerdings unter der Größe des kleinsten tatsächlich vorkommenden Kopfes, so ist mit Einbußen in der Erkennungsrate zu rechnen.

Es wurden viele weitere Testdurchläufe mit unterschiedlichen Kombinationen der Parameterwerte gestartet. Am Ende erreichte die beste vollzogene Zählung eine Trefferquote von 91,98 % bei einer Falschen-Alarm-Rate von 6,25 %. Es wurde dabei die Parametereinstellung des ursprünglichen Beispiels benutzt, lediglich mit der Änderung von `scaleFactor` auf `1.15` und von `detectHeight` auf `175`. Mit dieser Einstellung dauerte die Ausführung aller Operationen auf einem Einzelbild durchschnittlich ungefähr 60 ms, was einer Bildwiederholfrequenz von etwa 17 Bildern pro Sekunde entspricht.

8 Diskussion

Nach durchgeführten Testläufen wurde nicht nur auf das Ergebnis geachtet, sondern es wurden stets auch manuelle Analysen der Einzelbilder durchgeführt, um Parameter oder sogar den Algorithmus anzupassen.

So wurden im Laufe des Projekts die in vorigen Kapiteln beschriebenen Konzepte der Ungarischen Methode sowie des Gebietstrackings hinzugefügt. Ebenfalls positiv bemerkbar machte sich eine Änderung in der Gruppierungsfunktion, welche die Zugehörigkeit eines einzelnen Punktes zu mehreren Gruppen zulässt. Auf diese Weise entstehen um einen großen Bereich mit vielen Erkennungen mehrere nahe beieinanderliegende Gruppen, aus denen das Tracking die passende Gruppe für eine Spur „auswählen“ kann, wobei die restlichen Gruppen ohne Spur ohnehin verfallen. Dies führt wiederum dazu, dass Spuren gleichmäßiger verlaufen.

Einige Probleme traten jedoch hervor. Hohe Ausführungsgeschwindigkeiten auf dem benutzten Beispielvideo konnten nur durch Einbußen in der Erkennungsrate erreicht werden. So gab es einen Durchlauf, der durchschnittlich etwa 30 ms pro Einzelbild für alle Ausführungsschritte benötigte, was einer hohen Bildwiederholfrequenz von zirka 33 Bildern pro Sekunde entspricht. Die Trefferquote dieses Durchlaufs lag allerdings bei geringen 81,13 % mit einer Rate falschen Alarms von 2,82 %. Unter diesem Aspekt ist anzumerken, dass es Szenen gibt, die einen kleineren Bereich abdecken und somit die minimale anzutreffende Kopfgröße größer ist als in diesem Beispielvideo. Ist dies der Fall, können deutlich höhere Geschwindigkeiten bei guten Erkennungsraten erreicht werden. Im Umkehrschluss kann die Ausführungszeit oder die Erkennungsrate inakzeptabel werden, wenn ein zu großer Bereich abgedeckt werden soll.

Die erwähnte Ampel überdeckte im Beispielvideo einige Personen, was die vollständige Erkennung und damit die Zählung dieser Personen verhinderte. Personen, die nur teilweise verdeckt wurden, konnten meist dennoch eine korrekte Zählung bewirken. Die Spur der Person verschob sich dabei lediglich etwas zur Ampel. Es konnten durchgängig Erkennungen der Ampel selbst gesichtet werden, die jedoch eigenständig keine Zählung auslösten. Unter diesem Gesichtspunkt zeigte das Konzept des Gebietstrackings seine Wirkung. Die Benutzung einer einfachen Messlinie führte nämlich häufig dazu, dass die Ampel fälschlicherweise als Person gezählt wurde. Trotzdem sollte beim Positionieren der Kamera stets darauf geachtet werden, freie Sicht auf die Szene zu haben.

Vor allem kann man feststellen, dass Ansammlungen von Personen die Zählung erschweren. Dadurch dass der Gruppierungsradius statisch ist, kann nicht immer korrekt zwischen einer „breiten“ Person und zwei „schmalen“ Personen unterschieden werden. Ähnliches gilt für Personen, die sich entgegenkommen und einander passieren. Wird der Abstand zwischen den Personen zu gering, ist es möglich, dass das System die Personen als eine einzelne Person auffasst und somit eine Zählung auslässt.

9 Ausblick

In dieser Bachelorarbeit wurde ein bidirektionales Personenzählsystem vorgestellt, das durch Haar-ähnliche Merkmale unter OpenCV realisiert wurde. Dabei wurden Trefferquoten von etwa 92 % bei einer Falschen-Alarm-Rate von unter 7 % erreicht.

Trotz mehrerer erfolgreicher Maßnahmen zur Beschleunigung der Erkennung, liegt noch immer ein Defizit in der geringen Ausführungsgeschwindigkeit der Erkennung. Um diese zu erhöhen, könnte man das Gebiet der Erkennung beschränken, indem beispielsweise nur zwei Eintrittsgebiete durch die Erkennung abgedeckt würden und danach ein anderer Mechanismus die Verfolgung der Spur übernähme. Für dieses Verfahren wäre jedoch ein Klassifikator mit hoher Genauigkeit erforderlich, damit keiner falschen Spur gefolgt würde. Ein anderer Ansatz wäre, nicht jedes verfügbare Einzelbild zu nutzen. Würden die Informationen aus lediglich jedem zweiten Bild für das System ausreichen, so würde sich die Geschwindigkeit effektiv verdoppeln. Erste Versuche dazu wurden bereits durchgeführt, konnten jedoch im Rahmen dieser Bachelorarbeit nicht vertieft werden. Weiterhin ist es in der vorliegenden Version des Personenzählsystems nicht möglich, Kameraaufnahmen direkt zu verwenden. Es bedarf einer vorigen Speicherung als Videodatei, welche dann dem Programm übergeben wird. Die Möglichkeit, Kameraaufnahmen direkt zu übermitteln, würde erlauben, das Personenzählsystem überall für vielseitige Zwecke zu verwenden.

10 Anhang

10.1 Algorithmen

Funktion `group`

Eingabe: `vector<Point> points, int groupingRadius`

Ausgabe: `vector<Point> grouped_points`

1. Speichere für jeden Punkt aus `points` die Anzahl der Nachbarn innerhalb `groupingRadius` und den akkumulierten euklidischen Abstand zu ihnen.
2. Solange ungruppierte Punkte verbleiben:
   a) Wähle denjenigen Punkt aus `points`, der die meisten Nachbarn besitzt; gibt es mehrere davon, wähle den mit dem geringsten akkumulierten Abstand. Dieser Punkt und alle seine Nachbarn stellen eine Gruppe dar.
   b) Berechne den Mittelpunkt der Gruppe anhand des arithmetischen Mittels der jeweils x- und y-Koordinaten aller Punkte der Gruppe und lege diesen in `grouped_points` ab.
   c) Entferne alle Punkte der Gruppe aus `points`. Die Punkte bleiben für bisher ungruppierte Punkte als Nachbarn erhalten.

Algorithmus 1: Funktion `group` zur Gruppierung von Punkten

Funktion `correlate`

Eingabe: `vector<Point> v1, vector<Point> v2, int maxDistance`

Ausgabe: `vector<int> indices`

1. Setze den Standard-Abstand auf das Doppelte von `maxDistance`.
2. Setze die Dimension auf das Maximum der Anzahl der Elemente in `v1` und `v2`.
3. Erstelle eine quadratische Matrix C mit dieser Dimension als Zeilen- und Spaltenanzahl, wobei jeder Eintrag c_{ij} den euklidischen Abstand zwischen `v2[i]` und `v1[j]` enthält, wenn dieser `maxDistance` nicht überschreitet, ansonsten setze den Eintrag auf den Standard-Abstand.
4. Subtrahiere von jeder Zeile und Spalte jeweils das Minimum.
5. Kennzeichne jede Null mit einem Stern, in dessen Zeile und Spalte noch kein Stern steht.
6. Solange die Anzahl der Sterne kleiner ist als die Dimension:
   a) Markiere jede Spalte, die einen Stern enthält.
   b) Falls das Minimum aller nicht randmarkierten Elemente größer als 0 ist, subtrahiere es von allen nicht randmarkierten Elementen und addiere es zu allen doppelt randmarkierten Elementen.
   c) Kennzeichne eine nicht randmarkierte Null mit einem Strich.
   d) Steht in dieser Zeile bereits ein Stern, so markiere die Zeile, lösche die Spaltenmarkierung des Sterns und springe zum Schritt 6. b).
   e) Kennzeichne die Null mit einem Stern.
   f) Steht in dieser Spalte bereits ein Stern, wähle die gekennzeichnete Null in der Zeile dieses Sterns und springe zum Schritt 6. e).
   g) Lösche sämtliche Striche und Zeilenmarkierungen.
7. Fülle `indices` so, dass für jeden Index j `indices[j]` der Position des Sterns in der Spalte j entspricht, falls der ursprüngliche Abstand der damit verbundenen Punkte nicht größer ist als `maxDistance`, ansonsten trage den Wert -1 ein.

Algorithmus 2: Funktion `correlate` zur Zuordnung von Punkten aus zwei Vektoren, welche auf der Ungarischen Methode aufbaut. Sterne stellen Spaltenmarkierungen und Striche Zeilenmarkierungen dar. Nach jedem Durchlauf der Schleife im Schritt 6 ist ein zusätzliches Element zugeordnet.

Funktion `findConnections`

Eingabe: `vector< pair<int,Point> > counts_test,`
`vector< pair<int,Point> > counts_comparison,`
`int maxFrameDist, int maxXDist`

Ausgabe: `vector<int> connections_test,`
`vector<int> connections_comparison`

1. Speichere die Abstände zwischen jedem Eintrag aus `counts_test` und jedem Eintrag `counts_comparison`, die weder `maxFrameDist` noch `maxXDist` überschreiten, zusammen mit ihren Indizes in einem Vektor.
2. Sortiere diesen Vektor nach dem Abstand der Einzelbildnummern und anschließend nach dem Abstand der x-Koordinaten.
3. Initialisiere in `connections_test` und `connections_comparison` jedes Element mit -1.
4. Für jeden Eintrag aus dem sortierten Vektor:

   Sei i der Index von `counts_test` und j der Index von `counts_comparison`. Wenn keiner der beiden Zählungen bereits zugeordnet wurde, setze `connections_test[i] = j` und `connections_comparison[j] = i`.

Algorithmus 3: Funktion `findConnections` zum Vergleich einer vom System ermittelten Zählung (`test`) und der manuellen Zählung (`comparison`). Jedes Element der Eingabe-Vektoren enthält die Nummer des Einzelbildes sowie den Punkt einer Zählung. Die Ausgabe-Vektoren werden als Referenzen übergeben.

Literaturverzeichnis

[1] Borislav Antic, Dragan Letic, Dubravko Culibrk und Vladimir Crnojevic: *K-means Based Segmentation for Real-Time Zenithal People Counting.* In: *Image Processing (ICIP), 2009 16th IEEE International Conference*, Seiten 2565–2568. IEEE, 2009.

[2] Gordon Cessford, Stuart Cockburn und Murray Douglas: *Developing New Visitor Counters and their Applications for Management.* Monitoring and management of visitor flows in recreational and protected areas, Seiten 14–20, 2002.

[3] Chao-Ho Chen, Tsong-Yi Chen, Da-Jinn Wang und Tsang-Jie Chen: *A Cost-Effective People-Counter for a Crowd of Moving People Based on Two-Stage Segmentation.* J Inform Hiding Multimedia Signal Process, 3(1):2073–4212, 2012.

[4] CMake. `http://cmake.org/`.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest und Clifford Stein: *Introduction to Algorithms, Third Edition.* The MIT Press, 2009.

[6] Franklin C. Crow: *Summed-area Tables for Texture Mapping.* SIGGRAPH Comput. Graph., 18(3):207–212, Januar 1984.

[7] Tom Fawcett: *ROC Graphs: Notes and Practical Considerations for Data Mining Researchers.* 2003.

[8] FOURCC: *Video Codecs by FOURCC.* `http://fourcc.org/codecs.php`.

[9] Alfréd Haar: *Zur Theorie der orthogonalen Funktionensysteme: Inaugural-Dissertation zur Erlangung der Doktorwürde einer hohen philosophischen Fakultät der Georg-Augusts-Universität zu Göttingen.* Druck der Dieterich'schen Univ.-Buchdruckerei, 1909.

[10] Peter Klausmann, Claus Wetzel und Bertram Anderer: *Personenzählung für kundenorientiertes Shop-Controlling.* Eul, 2009.

[11] Harold W. Kuhn: *The Hungarian Method for the Assignment Problem.* Naval research logistics quarterly, 2(1-2):83–97, 1955.

[12] LASE PeCo Systemtechnik GmbH. `http://peoplecounter.de/`.

[13] Rainer Lienhart, Alexander Kuranov und Vadim Pisarevsky: *Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection.* MRL Technical Report, 2002.

[14] Osama Masoud und Nikolaos P. Papanikolopoulos: *A Novel Method for Tracking and Counting Pedestrians in Real-Time Using a Single Camera.* Vehicular Technology, IEEE Transactions on, 50(5):1267–1278, 2001.

[15] Minimalist GNU for Windows. `http://www.mingw.org/`.

[16] James Munkres: *Algorithms for the Assignment and Transportation Problems.* Journal of the Society for Industrial & Applied Mathematics, 5(1):32–38, 1957.

[17] Open Source Computer Vision Library. `http://opencv.org/`.

[18] OpenCV Documentation: *Cascade Classifier Training.* `http://docs.opencv.org/doc/user_guide/ug_traincascade.html`.

[19] OpenCV Documentation: *detectMultiScale.* `http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html#cascadeclassifier-detectmultiscale`.

[20] OpenCV Documentation: *groupRectangles.* `http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html#grouprectangles`.

[21] OpenCV Documentation: *partition.* `http://docs.opencv.org/modules/core/doc/clustering.html#partition`.

[22] OpenCV on the Cell: *Facedetect: Performance.* `http://cell.fixstars.com/opencv/index.php/Facedetect#Performance`.

[23] OpenCV Source Code: *cascadedetect*, Zeilen 1122–1124. `https://github.com/Itseez/opencv/blob/2.4.7/modules/objdetect/src/cascadedetect.cpp#L1122-L1124`.

[24] Constantine P. Papageorgiou, Michael Oren und Tomaso Poggio: *A General Framework For Object Detection.* In: *Computer vision, 1998. sixth international conference*, Seiten 555–562. IEEE, 1998.

[25] Chirag I. Patel und Ripal Patel: *Object Counting in Video Sequences.* International Journal of Computer & Electrical Engineering, 4(4), 2012.

[26] Sebastian Schmitt: *Real-Time Object Detection With Haar-Like Features*, 2010.

[27] Naotoshi Seo: *ImageClipper.* `https://code.google.com/p/imageclipper/`.

[28] Naotoshi Seo: *OpenCV haartraining Tutorial.* `http://note.sonots.com/SciSoftware/haartraining.html`.

[29] Stemmer Imaging GmbH: *Grauwert.* `http://www.stemmer-imaging.de/de/grundlagen/5455`.

[30] Paul A. Viola und Michael J. Jones: *Rapid Object Detection using a Boosted Cascade of Simple Features.* In: *CVPR (1)*, Seiten 511–518. IEEE Computer Society, 2001.

[31] Visapix GmbH. `http://www.visapix.de/`.

[32] Vitracom AG. `http://www.vitracom.de/`.

[33] J. M. Wang, S. W. Chen und C. S. Fuh: *People Counting System Based on Particle Filter with Memory States for Improvement.*