Universität Bremen

Erkennung humanoider Roboter mittels Deformable Part Models

Masterarbeit in der Informatik
Fachbereich 3
Universität Bremen

von

René Schröder

Betreuer:
Dr. Tim Laue
Prof. Dr. Thomas Schneider

Tag der Anmeldung: 12.10.2018
Tag der Abgabe: 28.04.2019

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Bremen, den 28.04.2019

Inhaltsverzeichnis

1. Einleitung

In diesem Kapitel wird die Motivation für die Bearbeitung dieses Themas gegeben. Außerdem werden die Ziele der Arbeit und die Anforderungen an das System definiert. Abschließend gibt es eine Übersicht über die Arbeit.

1.1 Motivation

Die Detektion von Objekten in Bildern ist ein komplexer Themenbereich. Die große Menge von Teilproblemen, wie z.B. Verdeckung durch andere Objekte, Änderungen in der Beleuchtung, Spiegelungen und Reflexionen, verschiedene Skalierungen, Verformungen von Objekten, Rauschen der Sensoren und klasseninterne Variationen machen den Reiz dieses Themen-Gebietes aus.

Grundsätzlich lässt sich die Objekt-Erkennung in zwei Gebiete unterteilen: Klassifikation und Detektion. Die Klassifikation beschränkt sich auf die Frage, **was** sich in einem Bild befindet und ordnet somit einem Bild eine Klasse zu. Oftmals ist das mit diversen Vorverarbeitungs-Schritten verbunden, um den Bildausschnitt möglichst genau auf das zu klassifizierende Objekt zuzuschneiden. Die Detektion erweitert das Problem der Klassifikation um die Frage, **wo** sich Objekte im Bild befinden. Dadurch ist das Problem der Objekt-Detektion wesentlich schwerer als die reine Klassifikation, denn bei der Detektion ist nicht immer gegeben, dass das zu detektierende Objekt das dominante Objekt im momentanen Bild ist (siehe dazu Abbildung 1.1). Die zusätzliche Lokalisierung der Objekte verlangt außerdem nach wesentlich komplexeren Algorithmen.

Im Allgemeinen gewinnt die Bildverarbeitung, insbesondere die Objekt-Detektion, immer mehr an Bedeutung. Smartphones werden immer leistungsfähiger und bieten damit immer mehr Möglichkeiten, um selbst mit aufwendigen Algorithmen arbeiten zu können. Die Forschung im Bereich der Robotik macht ebenso rasante Fortschritte. Bildverarbeitungs-Algorithmen finden außerdem viel Anwendung im Gebiet des autonomen Fahrens. Passanten müssen auf Fußwegen und der Straße detektiert werden, um dementsprechend reagieren zu können. Auch Straßenschilder müssen zuverlässig detektiert und klassifiziert werden.

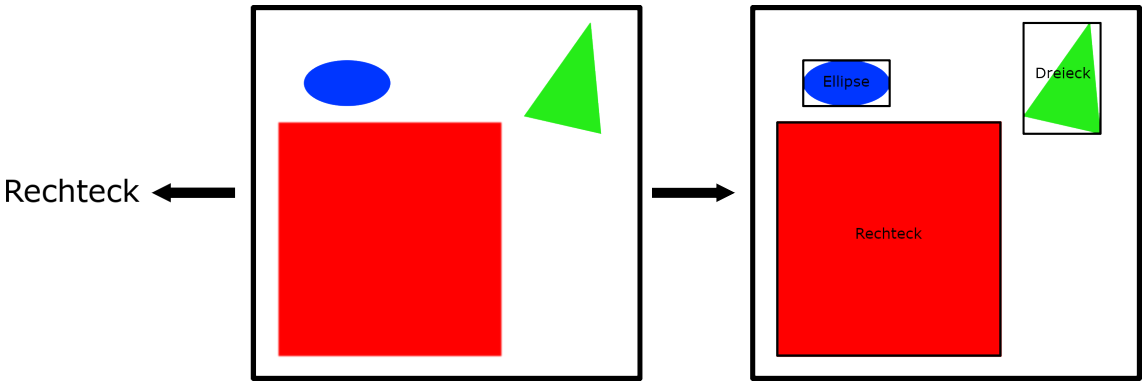


Abbildung 1.1: Beispiel für eine Klassifikation (links) und eine Detektion (rechts). Die Klassifikation liefert die Klasse des Bildes. Eine Detektion kann nicht nur multiple Objekte mit einbeziehen, sondern liefert neben den Klassen (hier: *Rechteck, Dreieck, Ellipse*) auch die Positionen. In diesem Falle sind die Positionen durch schwarze Boxen dargestellt.

Dabei ergeben sich je nach Objekt und Anwendungsbereich neue Probleme. Wenn man die Detektion von Personen und humanoiden Robotern in Bildern betrachtet, so fällt auf, dass diese Objekte diverse klasseninterne Variationen aufweisen. Klasseninterne Variationen sind in diesem Fall beispielsweise verschiedene Posen[1] der Körperteile und Größen im Bild. Diese Variationen sind es, die die Detektion von Personen und humanoiden Robotern besonders schwer machen.

Diese Arbeit entstand im Kontext des Roboterfußball-Teams *B-Human* (siehe Kapitel 3.3). Die Detektion der *NAO*-Roboter (siehe Kapitel 3.2) auf dem Spielfeld bezieht dabei fast alle Teilprobleme der Objekt-Detektion mit ein, denn das System muss sowohl in bekannten, als auch vorher unbekannten Bedingungen zuverlässig funktionieren. Die Lichtverhältnisse z.B. schwanken stark zwischen verschiedenen Spielfeldern und sogar der Position und Rotation des aktuellen Roboters. Der Umstieg von künstlichem Licht auf reale Lichtverhältnisse wird aktiv vorangetrieben. So gab es auf dem *RoboCup 2016* (siehe Kapitel 3.1) erstmals einen Wettbewerb, bei dem unter realen Lichtverhältnissen gespielt wurde.

Auf Farben basierende Algorithmen stoßen bei solchen Situationen schnell an ihre Grenzen. Da die zu verarbeitenden Bilder von einem Roboter auf dem Spielfeld stammen, können sich Roboter in diversen Größen im Bild befinden – insbesondere kommt es oft vor, dass sich Roboter nicht in Gänze im Bild befinden. Ein System, welches diese Roboter detektieren soll, muss also damit zurecht kommen, dass Teile der Roboter verdeckt sind oder außerhalb des sichtbaren Bereiches liegen und die Größe der Roboter stark variiert.

Es gibt verschiedene Wege, um das Problem der Objekt-Detektion zu lösen. Im Allgemeinen werden so genannte Merkmale auf einem Bild berechnet und mit bekanntem Wissen über das zu detektierende Objekt abgeglichen. Die Merkmale können beliebig komplex sein und reichen von einfachen Farb- oder Größen-Informationen bis hin zu sehr komplexen Deskriptoren, welche um ein Vielfaches größer als das Ausgangsbild sind. Eines der einfachsten Merkmale im Kontext der Detektion von weißen Robotern in einem Farbbild ist die Anzahl von weißen Pixeln über einem bestimmten Bereich im Bild.

[1] Eine Pose ist definiert durch eine Position und eine Rotation.

Dieses Merkmal stößt aber an seine Grenzen, soweit der Roboter nicht mehr das einzige weiße Objekt auf dem Spielfeld ist. Farbbasierte Verfahren haben außerdem den Nachteil, dass sie anfällig gegenüber verschiedenen Beleuchtungs-Verhältnissen und Bildrauschen sind. Selbst kleine Änderungen in den Farbwerten können zu großen Problemen führen. Um dieses Problem zu umgehen, wird oftmals vor der Berechnung der Merkmale eine *Farb-Segmentierung*[2] durchgeführt. Segmentierungs-Algorithmen lösen die Probleme allerdings nicht gänzlich. Hinzu kommt, dass sie oftmals abhängig sind von einer vorherigen Kalibrierung oder von Modellen, welche für unbekannte Umgebungen immer neu gelernt werden müssen.

In Anbetracht der zu lösenden Probleme bezüglich der variierenden Lichtverhältnisse und der Verdeckung durch andere Roboter werden in dieser Arbeit *Deformable Part Models* zur Detektion von humanoiden *NAO*-Robotern evaluiert.

Deformable Part Models beschreiben ein Objekt als eine Menge von *Teilen*. Diese *Teile* können relativ zu einer *Wurzel* verschoben werden, wobei ein Fehlerwert entsteht. Diese Repräsentation ermöglicht es, eine Vielzahl von klasseninternen Variationen darzustellen. Insbesondere ist das Verfahren in der Lage, Objekte zu detektieren, auch wenn sich einzelne *Teile* nicht im Bild befinden oder von anderen Objekten verdeckt werden. Die *Teile* beschreiben Gewichte für einen gradientenbasierten *Histogram of Gradients*-Deskriptor. Dieses von *Pedro F. Felzenszwalb et al.* entwickelte Modell hat sich bereits bei der Detektion von Personen bewiesen [Felzenszwalb *et al.*, 2008]. Folglich wäre es interessant zu erfahren, wie sich dieses modellbasierte Verfahren bei humanoiden Robotern verhält. Im Zuge dieser Arbeit soll die Frage beantwortet werden, wie gut sich dieses Verfahren auf die Detektion von humanoiden Robotern adaptieren lässt.

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist die Entwicklung einer Software, welche humanoide Roboter in Bildern mittels *Deformable Part Models* detektiert. Da es sich bei den *Deformable Part Models* um ein modellbasiertes Verfahren handelt, müssen im Zuge dieser Arbeit Modelle auf annotierten Datensätzen gelernt werden. Dabei sollen genau vier Modelle evaluiert werden:

1. Ein Modell, welches auf realen Bildern von humanoiden *NAO*-Robotern gelernt wurde.

2. Ein Modell, welches auf Bildern von Personen gelernt wurde. Hier ist besonders interessant, wie wichtig die gelernten Eigenschaften von Personen in Bildern sind. Wenn dieses Modell genauso gute oder vielleicht sogar bessere Ergebnisse als das erste Modell liefert, eröffnet das die Möglichkeit, Bilder von den öffentlich verfügbaren Datensätzen von Personen zu verwenden, um Modelle für humanoide *NAO*-Roboter zu lernen.

3. Ein Modell, welches auf Bildern von humanoiden *NAO*-Robotern und Personen gelernt wurde. Für dieses Modell gilt dasselbe wie für das vorherige. Wenn es

[2] Bei einer *Farb-Segmentierung* werden alle Farbwerte eines Bildes in vorher festgelegte Farbwerte klassifiziert.

möglich ist, Modelle von Personen mit denen von humanoiden *NAO*-Robotern nachzulernen, kann man auf schon existierende und bereits evaluierte Modelle zurückgreifen.

4. Ein Modell, welches auf simulierten Bildern von humanoiden *NAO*-Robotern gelernt wurde. In diesem Falle wird das Modell mit Bildern mit perfekten Umgebungs-Verhältnissen gelernt. Wenn dieses Modell akzeptable Ergebnisse erzielt, heißt dies, dass *Deformable Part Models* die Transferleistung von einer simulierten Umgebung in echte Umgebungen stemmen können. In diesem Falle könnte man beliebig viele Trainings-Datensätze automatisch generieren, was den Trainings-Aufwand drastisch reduzieren würde.

1.3 Anforderungen an den Detektor

Das beste Modell soll auf einem ausgewählten Evaluations-Datensatz mindestens eine Erkennungsrate (siehe Definition von *Precision* in Kapitel 4.5.2) von 80% erreichen. Der Detektor muss in der Lage sein, Roboter bei verschiedensten Licht-Bedingungen zuverlässig zu detektieren. Da es wichtig ist, dass auch Roboter auf größeren Entfernungen erkannt werden, sollen diese Eigenschaften auf einer maximalen Entfernung von 3,5 Metern nachgewiesen werden.

Zusätzlich muss der Detektor robust gegenüber Verdeckungen durch andere Roboter oder sonstige Objekte sein. Ein Roboter, der neben einem weißen Objekt steht, soll ebenso detektiert werden, wie auch mehrere Roboter, die voreinander oder nebeneinander stehen. Da diese Arbeit nur die Möglichkeit untersucht, ob *Deformable Part Models* verwendet werden können, um humanoide Roboter zu detektieren, wird keine Vorgabe bezüglich der Laufzeit gegeben.

1.4 Aufbau der Arbeit

Im folgenden Kapitel wird eine Übersicht über verwandte Arbeiten gegeben. Dabei werden Arbeiten betrachtet, welche die Detektion von humanoiden *NAO*-Robotern behandeln, wie auch solche, welche *Deformable Part Models* zur Detektion von Objekten anwenden. Im dritten Kapitel werden die Hintergründe dieser Arbeit erläutert. Das vierte Kapitel gibt einen Überblick über einige Grundlagen, welche für das Verständnis der Arbeit benötigt werden. Im fünften Kapitel wird ein Überblick über das entwickelte Gesamt-System und anschließend eine detaillierte Beschreibung der einzelnen Komponenten gegeben. Das System wird im sechsten Kapitel evaluiert, ehe im siebten Kapitel abschließend ein Fazit und ein Ausblick gegeben werden.

2. Verwandte Arbeiten

Im folgenden Kapitel wird auf verwandte Arbeiten eingegangen. Dabei liegt der Fokus auf Arbeiten, welche sich mit der Erkennung humanoider Roboter beschäftigen und auf solche, welche Probleme in der Detektion von Objekten in Bildern mit *Deformable Part Models* gelöst haben.

2.1 Erkennung humanoider Roboter

In der *SPL* (siehe Kapitel 3.2) scheint die Verwendung von *Deformable Part Models* nicht weit verbreitet zu sein. Tatsächlich konnte ich keine Arbeit finden, die humanoide *NAO*-Roboter mit *Deformable Part Models* detektiert. Ich habe nur zwei Arbeiten im Bereich der *RoboCupRescue*-Liga gefunden, aber aus diesen konnte ich leider keine Informationen für diese Arbeit ableiten, da die eine von Çakmak *et al.* [2016] in türkisch verfasst ist und die andere von Yavuz *et al.* [2016] keine konkreten Informationen über die Implementierung enthält. Zusätzlich steht in dieser Liga nicht die Detektion von Robotern im Fokus, sondern die von Personen, welche durch Puppen dargestellt werden. Aufgrund der abgebildeten Formeln in Yavuz *et al.* [2016] gehe ich davon aus, dass die ursprüngliche Version von Felzenszwalb *et al.* [2010b] verwendet wurde, welche auch im Verlauf dieser Arbeit implementiert wurde.

Der Trend im Bereich der Roboter-Detektion in der *SPL* zeigt eindeutig in die Richtung der *neuronalen Netze*. Das kann man sich damit erklären, dass die ehemals verwendeten Algorithmen bei den stetig erschwerten Bedingungen in der *SPL* nicht mehr die gewünschten Erkennungsraten liefern.

Als Beispiel dieser Entwicklung kann man die Arbeit von *Bernd Poppinga* betrachten. Dieser hat sich in seiner Masterarbeit an der *Universität Bremen* mit der Frage beschäftigt ob es möglich sei, einen auf *Deep Learning* basierenden Objekt-Detektor zu entwickeln, welcher auf mobilen Robotern in Echtzeit läuft [Poppinga, 2018]. Dabei wurde auf Basis von schon existierenden Netz-Architekturen ein neues *neuronales Netz* konzipiert und anschließend mit den ursprünglichen Architekturen verglichen.

Der Ansatz mit *neuronalen Netzen* offenbarte dabei Schwächen bei der Detektion von liegenden Robotern. Die besten Ergebnisse mit einer Erkennungsrate von über

90% erreichte der Detektor bei Robotern in einem Abstand von maximal drei Metern. Die berechneten *Bounding-Boxes*[1] seien recht ungenau, was in weiteren Verfahren zu Problemen führen könnte. Außerdem wies der Detektor einige Falsch-Erkennungen auf, welche wohl durch Bild-Artefakte und Personen auf dem Spielfeld entstanden. Hintereinander stehende Roboter führten ebenfalls zu Problemen.

Der entwickelte Detektor erreichte bis auf die *Precision* (siehe Kapitel 4.5.2) alle Vorgaben und konnte mit einer Erkennungsrate von 72% bei Robotern mit einer Entfernung bis sechs Metern und 90% bei einer Entfernung bis drei Metern punkten.

Vor den *neuronalen Netzen* wurden oftmals farbbasierte Ansätze genutzt. Diese nutzen den Fakt aus, dass die Roboter neben den Toren die einzigen großen weißen Objekte auf dem Spielfeld sind. Die momentan in *B-Human* (siehe Kapitel 3.3) genutzte Implementierung verfolgt einen Ansatz, in dem weiße Regionen in einem farbsegmentierten Bild gesucht werden [Röfer *et al.*, 2018, Kapitel 3.2.3]. Dieser Ansatz wird auch zur Detektion von anderen weißen Hindernissen auf dem Spielfeld, wie z.B. den Torpfosten, genutzt.

Zu Beginn werden potentielle Kandidaten im Bild gesucht, indem dieses in horizontalen Abständen von oben nach unten abgetastet wird. Während dieses Verfahrens werden die weißen bzw. grünen Pixel gezählt und in zwei Ergebnissen verrechnet. Der niedrigste Punkt an jeder abgesuchten x-Koordinate, bei dem die Ergebnisse über einem gegebenen *Schwellwert* liegen, wird als potentieller Kandidat gewählt. Anschließend werden nebeneinander liegende Kandidaten zusammengefasst. Dabei müssen diese bestimmten Anforderungen entsprechen, um beispielsweise Feldlinien ausschließen zu können. Entspricht die so gefundene Fläche in der Größe einem Roboter oder einem anderen Hindernis, wird sie akzeptiert (siehe Abbildung 2.1).

Um die Anforderungen von *neuronalen Netzen* zu senken, gibt es auch Teams, welche diese mit klassischen Detektions-Algorithmen kombinieren.

Dario Albani et al. von der *Sapienza University of Rome* haben ein System implementiert, welches auf *Deep Learning*-Algorithmen basiert [Albani *et al.*, 2017]. Dabei werden zuerst interessante Regionen von einem farbsegmentierten Bild extrahiert, ähnlich zu dem Verfahren von *B-Human*. Entspricht eine Region der Größe eines Roboters, wird diese mit Hilfe eines *neuronalen Netzes* evaluiert. Dabei findet nur eine Klassifikation des extrahierten Ausschnittes statt.

Das entwickelte System konnte sehr gute Ergebnisse vorweisen, hatte aber Defizite bei der Laufzeit. Insbesondere das *neuronale Netz* zeigte eine sehr lange Laufzeit.

Schaut man sich die betrachteten Arbeiten an, kann man einige wichtige Punkte extrahieren. Zum einen müssen die Bereiche im Bild stark begrenzt werden, auf denen aufwendige Algorithmen ausgeführt werden, um der Anforderung der Echtzeitfähigkeit gerecht zu werden. Albani *et al.* [2017] haben weiße Flächen wie im *B-Human*-Framework [Röfer *et al.*, 2018, Kapitel 3.2.3] extrahiert. *Bernd Poppinga* hingegen verwendet nur eine stark komprimierte Version des originalen Bildes. Roboter die dicht beieinander stehen sind ebenfalls ein grundlegendes Problem.

Wie gut dicht beieinander stehende Roboter mit *Deformable Part Models* detektiert werden können, wird im Laufe dieser Arbeit evaluiert. Da Echtzeitfähigkeit des

[1] Eine *Bounding-Box* ist das kleinste Rechteck, das ein Objekt in einem Bild vollständig umschließt.

Abbildung 2.1: Die Abbildung zeigt einen *NAO*-Roboter mit einer *Bounding-Box* (weiß), welche aus der aktuellen Implementierung stammt. Die große Schwäche des genutzten Ansatzes ist, dass nur Fußpunkte genutzt werden, um die Größe des Roboters im Bild zu ermitteln. Die Arme werden nicht berücksichtigt, was zu ungenauen *Bounding-Boxes* führt.

entwickelten Systems keine definierte Vorgabe ist, muss der Bereich im Bild nicht begrenzt werden, auf dem Berechnungen ausgeführt werden. Der Ansatz von *Bernd Poppinga* ist in dem hier vorgestellten Verfahren gar nicht anwendbar, da *Deformable Part Models* nur auf Bildern einer gewissen Mindestgröße funktionieren können.

2.2 Anwendung von Deformable Part Models

In der Detektion von Personen waren *Deformable Part Models* einige Zeit der *State-Of-The-Art*. Das zeigte sich vor allem in den Ergebnissen der *VOC*-Challenge 2008 [Felzenszwalb *et al.*, 2010b, Table 3]. Diese Ergebnisse führten dazu, dass *Deformable Part Models* zur Detektion verschiedenster Objekte verwendet wurden.

2.2.1 Gesichts-Erkennung

Die Eigenschaften von *Deformable Part Models* sind gut geeignet zur Detektion von Objekten mit einer großen Anzahl von klasseninternen Variationen. Bei menschlichen Gesichtern zeigen sich diese Variationen hauptsächlich in verschiedenen Abständen zwischen Augen, Mund und anderen Merkmalen. Unregelmäßige Beleuchtungen und

Verdeckung durch Kleidung und Haare sind weitere Probleme. Auf Basis dieser Eigenschaften haben Forscher *Deformable Part Models* zur Detektion von menschlichen Gesichtern verwendet.

Ranjan et al. von der *University of Maryland* haben ein System zur Detektion von menschlichen Gesichtern entwickelt [Ranjan *et al.*, 2015]. In diesem System wurde der *Histogram of Gradients*-Deskriptor durch ein *Convolutional Neural Network* namens *Deep Pyramid CNN* ersetzt. Für jedes *Level* der Pyramide ergeben sich so am Ende 256 verschiedene *Filter*. Da festgestellt wurde, dass die ersten extrahierten *Filter* nicht skalierungsinvariant sind, müssen diese noch normalisiert werden, eher sie mit einem gelernten *Deformable Part Model* ausgewertet werden.

Die Limitierungen vom *HoG*-Deskriptor, wie z.B. Anfälligkeiten gegen verschiedene Rotationen von Merkmalen, sollen so umgangen werden. Außerdem hat das Vorgehen den Vorteil, dass es schon sehr viele vortrainierte Netze gibt, auf deren Merkmals-Extraktion zurückgegriffen werden kann. Das entwickelte System wurde auf vier Datensätzen evaluiert und konnte mit den *State-Of-The-Art* Algorithmen mithalten.

Ghiasi et al. von der *University of California, Irvine* haben in ihrer Arbeit den Fokus darauf gelegt, die Lokalisierung von Merkmalen in teilweise verdeckten menschlichen Gesichtern zu optimieren [Ghiasi & Fowlkes, 2014].

Zu diesem Zwecke werden *Hierarchical Part Models* verwendet. Bei diesen Modellen haben die *Teile* eine Relation zueinander und sind nicht, wie bei den in dieser Arbeit verwendeten Modellen, relativ zu einer *Wurzel* platziert. Die zu lokalisierenden Merkmale sind durch *Keypoints* dargestellt, welche als *HoG*-Deskriptor vorliegen. Eine Verdeckung wird so modelliert, dass die verdeckten *Keypoints* auf 0 gesetzt werden. Das Training der Modelle wurde so implementiert, dass ein positives Bild an zufälligen Stellen verdeckt wurde und dann die *HoG*-Deskriptoren der anderen *Keypoints* mit einer normalen *SVM* trainiert wurden.

Das System konnte vorher veröffentlichte Systeme übertreffen und zeigte robuste Ergebnisse auf verschiedenen Training- und Testdatensätzen.

2.2.2 Bestimmung von Körperhaltungen und Aktionen

Die Platzierungen der *Teile* von *Deformable Part Models* können entweder im Training gelernt werden oder durch die annotierten Trainingsdaten vorgegeben werden. Im zweiten Fall ist genau bekannt, welcher *Teil* welches Merkmal beschreibt. Bei der Detektion kann man die Positionen der *Teile* verwenden, um Körperhaltungen und Aktionen zu bestimmen.

Yi Yang und *Dave Ramanan* von der *University of California* haben eine Variante von *Deformable Part Models* genutzt, um Posen von Personen in statischen Bildern zu ermitteln [Yang & Ramanan, 2011]. In diesem Fall wurde die ursprüngliche Definition von *Deformable Part Models* erweitert, sodass *Teile* ebenfalls aus *Mixture-Models* bestehen können. Außerdem kommt diese Variante ohne eine *Wurzel* aus. Stattdessen werden *Deformations-Kosten* (auch: *deformation costs*) zwischen *Teilen* definiert.

Diese neuen Modelle können nun genutzt werden, um eine Vielzahl von verschieden rotierten *Teilen* zu modellieren. Das ermöglicht unter Anderem die Detektion von verschiedenen Hand-Posen. Zusätzlich wurden Vorgaben in das Modell integriert. Ein Unterarm hat beispielsweise immer eine ähnliche Rotation in Relation zum Oberarm.

Das resultierende System konnte die derzeitigen *State-Of-The-Art* Systeme in der Erkennungsrate überbieten.

Yicong Tian et al. von der *University of Central Florida* haben eine Erweiterung der *Deformable Part Models* entwickelt, um Aktionen von Personen zu detektieren [Tian *et al.*, 2013].

Zu diesem Zwecke wurde der *Histogram of Gradients*-Deskriptor so erweitert, dass er auf einer Sequenz von Bildern berechnet werden kann. Statt einen Deskriptor für einen Block von Pixeln (x, y) zu haben, ergibt sich somit ein Deskriptor für ein Volumen (x, y, t). Eine Aktion ist in diesem Falle als eine Abfolge von Verschiebungen von *Teilen* definiert. Der Ansatz konnte *State-Of-The-Art* Ergebnisse erzielen.

2.2.3 Allgemeine Objekt-Detektion

David Weigel hat in seiner Masterarbeit an der *Universität Bremen Deformable Part Models* genutzt, um Hallenfußballspieler in Videoaufnahmen zu detektieren [Weigel, 2015].

Auf Basis dieser Detektionen wurden die Spieler räumlich in einer Sporthalle lokalisiert. Diese Daten konnten anschließend genutzt werden, um das taktische Verhalten der Spieler auf dem Feld zu analysieren. Ziel der Arbeit war es, eine Grundlage für eine individualtaktische Leistungsdiagnose zu schaffen.

Die Erkennung der Hallenfußballspieler lässt sich dabei in zwei Schritte unterteilen:

Im ersten Schritt wird eine Vordergrundmaske erstellt. Dieses binäre Bild hat die selbe Größe wie das Eingabebild – aber Vordergrundpixel werden weiß und Hintergrundpixel schwarz kodiert. Bei Vordergrundpixeln handelt es sich um dynamische Objekte im Bild, wie z.B. die Spieler. Hintergrundpixel sind statische Objekte, wie der Hallenboden.

Aus den Vordergrundpixeln wird anschließend eine *Region of Interest* berechnet, auf welche der Detektor angewendet wird. Die Positionen der Füße werden anschließend genutzt, um die Spieler in der Halle zu lokalisieren. Die Unterscheidung der einzelnen detektierten Spieler geschieht auf Basis eines Farbhistogramms und eines gelernten *k-Nächste-Nachbarn-Klassifikator*.

Das entwickelte System war am Ende der Arbeit in der Lage, einen Spieler in einem Video mit insgesamt drei Spielern in 86% der Fälle eindeutig zu erkennen.

Chai et al. haben einen Ansatz implementiert, bei dem ein *Deformable Part Model* mit einem *GrabCut*-Algorithmus kombiniert wird, um die Lokalisierung von *Teilen* zu verbessern und gleichzeitig eine genaue *Segmentierung* von Objekten in statischen Bildern zu ermöglichen [Chai *et al.*, 2013].

Ein Teil des Systems ist ein klassisches *Deformable Part Model*. Zusätzlich gibt es zu jedem *Teil* eine so genannte *Saliency-Map*, welche mit einem *GrabCut*-Algorithmus gelernt wird und zu jeder Zelle eines *Teiles* die Wahrscheinlichkeit angibt, ob es sich um eine Vordergrund- oder Hintergrundzelle handelt. Abbildung 2.2 zeigt ein solches Paar.

Bei der Platzierung der *Teile* während der Detektion und im Training werden diese nicht nur auf Basis der *deformation costs* platziert, sondern auch auf Basis

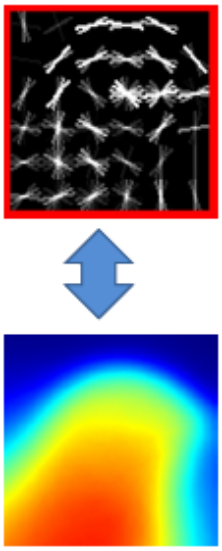


Abbildung 2.2: Ein *HoG*-Deskriptor (oben) mit zugehöriger *Saliency-Map* (unten) (entnommen aus [Chai *et al.*, 2013]).

der jeweiligen *Saliency-Map*. Das System konnte dabei andere *State-Of-The-Art* Algorithmen übertreffen, obwohl laut der Aussage von *Chai et al.* noch Verbesserungen möglich seien.

2.2.4 Zusammenfassung

Aus den hier vorgestellten Arbeiten kann man ebenfalls einige allgemeine Ansätze und Probleme ableiten:

- Keiner der vorgestellten Ansätze läuft in Echtzeit. *Deformable Part Models* scheinen nicht für die Anwendung in Echtzeit konzipiert zu sein oder in dem recherchierten Gebiet ist die Echtzeit allgemein kein großes Kriterium. Stattdessen arbeiten viele der Systeme auf Videos oder statischen Bildern und sind nicht in ein in Echtzeit laufendes System eingebettet.

- *Deformable Part Models* funktionieren in bestimmten Fällen auch bei verdeckten Merkmalen. Die verdeckten *Teile* werden aber unter Umständen falsch platziert. Diese Beobachtung kann man Ghiasi & Fowlkes [2014] entnehmen. Da das Verfahren in dieser Arbeit aber nur zur Detektion verwendet werden soll und die Positionen der *Teile* nicht weiter relevant sind, müssen keine Verdeckungen modelliert werden.

- In Chai *et al.* [2013] wurde der Einfluss von verschiedenen Parametern evaluiert. Dabei handelte es sich um die Anzahl von *Teilen* und Modellen. Diese Evaluation halte ich ebenfalls für sinnvoll und werde sie deshalb später in dieser Arbeit aufgreifen. Diese Parameter können stark anwendungsspezifisch sein und sollten daher überprüft werden.

- Ranjan *et al.* [2015] haben ein *Deep Pyramid CNN* statt dem *Histogram of Gradients* als Merkmal für die *Teile* eines *Deformable Part Models* genutzt. Der Grund dafür war, dass ein *HoG*-Deskriptor nicht die Merkmale eines

menschlichen Gesichtes bei verschiedenen Beleuchtungen und Blickwinkeln erfassen könne. Da es sich bei den hier verwendeten *NAO*-Robotern um größere Objekte handelt, gehe ich allerdings davon aus, dass der *HoG*-Deskriptor für dessen Merkmale ausreicht. Wie stabil die Detektion von humanoiden Robotern ist, wird im Laufe dieser Arbeit ermittelt.

- Der ursprüngliche Algorithmus von Felzenszwalb *et al.* [2010b] scheint das Mittel der Wahl zu sein. Die verschiedenen Ansätze unterscheiden sich hauptsächlich in den Deskriptoren und Vorverarbeitungs-Schritten.

- *David Weigel* hat eine Unterscheidung in Vordergrund- und Hintergrund-Pixel implementiert und dadurch den Rechen-Aufwand der *Deformable Part Models* erheblich reduzieren können. Dieser Ansatz ist in dieser Arbeit aber leider nicht anwendbar, da hier in dynamischen Umgebungen Roboter detektiert werden und somit eine Unterscheidung in Vorder- und Hintergrund beinahe unmöglich ist.

3. Hintergrund

In diesem Kapitel wird ein Überblick über den Kontext der Arbeit gegeben.

3.1 RoboCup

Robot World Cup oder auch *RoboCup* genannt, ist eine 1993 gegründete Initiative, die sich darauf spezialisiert hat, die Forschung im Bereich der KI und Robotik zu fördern [RoboCup Federation, 2019a].

Ursprünglich wurden nur Roboterfußball-Spiele ausgetragen – mittlerweile gibt es aber auch andere Ligen, in denen es darum geht, verschiedenste Probleme mit Robotern zu lösen (*RoboCupRescue*, *RoboCup@Home*, *RoboCupIndustrial*). Selbst Schülern wird mit einer eigenen Liga, namentlich *RoboCupJunior*, der Einstieg in die Robotik ermöglicht.

Die Initiative verfolgt das Ziel, Mitte des 21. Jahrhunderts mit einem Team von humanoiden Robotern gegen den amtierenden FIFA-Weltmeister zu gewinnen. Dieses Ziel bietet einen großen Anreiz, um die Forschung voran zu treiben [RoboCup Federation, 2019b]. Viele der Teams veröffentlichen trotz der Wettbewerbe jährlich ihren Code, damit andere auf diesem aufbauen können. Der Wettbewerbs-Gedanke spielt also meistens eher eine zweitrangige Rolle, wobei der Fortschritt in der Forschung im Vordergrund steht. Neben dem *RoboCup*, welcher inoffiziell als die Weltmeisterschaft des Roboter-Fußballs gilt, gibt es auch kleinere Wettbewerbe, wie die *German Open* oder die *Japan Open*.

3.2 Standard Platform League und NAO

Die *Standard Platform League* (im weiteren Verlauf auch *SPL* genannt) ist eine Liga im *RoboCup*, in der jedes Team den selben Roboter benutzt. Bei diesem Roboter handelt es sich um den *NAO* von der Firma *SoftBank Robotics* (siehe Abb. 3.1a).

Den Teams ist es in dieser Liga nicht erlaubt, die Hardware ihrer Roboter zu ändern. In diesem Falle handelt es sich also um einen reinen Software-Wettbewerb. Die Roboter spielen dabei völlig autonom. Damit das möglich ist, müssen eine Vielzahl

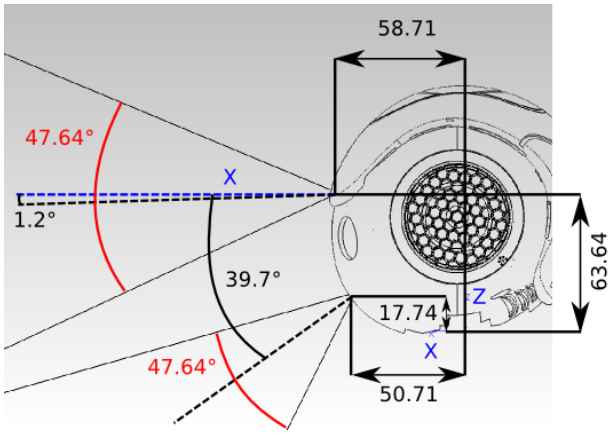
von Problemen aus dem gesamten Feld der Robotik und der KI gelöst werden. Um die Forschung zusätzlich zu fördern und die Teams vor neue Herausforderungen zu stellen, werden jährlich Änderungen an den Regeln vorgenommen – beispielsweise wurde der Ball vor kurzem durch einen neuen Softball ausgetauscht, wodurch sich neue Aufgaben im Bereich der Bildverarbeitung ergaben [RoboCup Federation, 2019c].

Der *NAO* ist ein 57,4 cm hoher humanoider Roboter, welcher seit 2006 von der Firma *SoftBank Robotics* entwickelt wird. In seiner aktuellsten Version (Version 6) verfügt der *NAO* über eine *Atom E3845 Quad core* CPU mit 1,91 GHz Taktrate und 4 GB DDR3 RAM [SoftBank Robotics, 2019b].

Bei dem Betriebssystem handelt es sich um Linux. Über die offizielle Schnittstelle namens *NAOqi* hat man Zugriff auf alle Sensoren und Gelenke mit insgesamt 25 Freiheitsgraden. Der *NAO* verfügt über insgesamt vier Mikrophone, sieben Berührungssensoren, zwei zweidimensionale Kameras, Fußdruck-Sensoren, Inertial-Sensoren (Accelerometer und Gyroskop) und Ultraschall-Sensoren [SoftBank Robotics, 2019d].

Neben einer USB-Schnittstelle verfügen die Roboter außerdem über WIFI. In der *SPL* wird das WIFI dazu verwendet, Informationen zwischen den Robotern eines Teams auszutauschen. Die beiden Kameras befinden sich im Kopf (siehe Abb. 3.1b) und liefern jeweils maximal 30 Bilder pro Sekunde bei einer Auflösung von 640×480 Pixeln.

(a) Die aktuelle Version des *NAO*-Roboters (entnommen aus [SoftBank Robotics, 2019d]).



(b) Positionierung und Ausrichtung der beiden Kameras des *NAO* Roboters (entnommen aus [SoftBank Robotics, 2019c]).

Abbildung 3.1: Übersicht über den *NAO*-Roboter der Firma *SoftBank Robotics*.

3.3 B-Human

B-Human ist ein Projekt des Fachbereichs 3 - Mathematik und Informatik der *Universität Bremen* und des Forschungsbereiches *Cyber-Physical Systems* des DFKI [B-Human, 2019].

Das Ziel des Projektes ist es, Forschung im Bereich der Robotik und KI zu betreiben, als auch Studierende für eine akademische Laufbahn oder die Forschung zu begeistern. So wurden seit 2008 schon etliche Master- und Bachelorarbeiten in diesem Projekt

geschrieben. Das Projekt besteht aus Studierenden und Forschern der *Universität Bremen* und des DFKI. Das Team nimmt seit 2008 an der *SPL* teil und ist seitdem das erfolgreichste Team in dieser Liga mit insgesamt sechs Weltmeister-Titeln [RoboCup Federation, 2019d]. Neben der *SPL* ist das Projekt auch in den *German Open* (acht Titel) und *European Open* (ein Titel) sehr erfolgreich.

4. Grundlagen

In diesem Kapitel werden einige Grundlagen erläutert, welche wichtig sind für das vollständige Verständnis der nachfolgenden Kapitel.

4.1 Lineare Filter

Wenn es darum geht, die Werte eines Bildes zu ändern, gibt es im wesentlichen zwei Möglichkeiten: Punktoperationen und *Filter*.

Punktoperationen werden immer auf einzelne Bildpunkte angewendet. Eine Punktoperation, die einen Wert eines Graustufenbildes invertiert, lässt sich beispielsweise wie folgt definieren:

$$I'(u, v) = 255 - I(u, v),$$

wobei $I(u, v)$ den Wert des Pixels an der Stelle (u, v) im Bild I beschreibt. Da Punktoperationen nur die Werte eines einzelnen Pixels berücksichtigen, kann man allerdings nicht alle Aufgaben mit ihnen lösen. Um auch komplexere Berechnungen auf Bildern auszuführen, gibt es die *Filter*.

Ein *Filter* ist im wesentlichen definiert durch eine Filtermatrix H (siehe Abb. 4.1) mit einem definierten *Hot Spot*. Die Filtermatrix enthält positive und negative Werte, welche auch Koeffizienten genannt werden.

Bei einem linearen *Filter* ist das Ergebnis eindeutig und vollständig bestimmt. Abbildung 4.2 zeigt den Vorgang, um ein neues Bild I' mit einem *Filter* zu berechnen.

Formal lässt sich eine Anwendung eines linearen *Filters* wie folgt definieren:

$$I'(u, v) = \sum_{(i,j) \in R} I(u + i, v + j) \cdot H(i, j), \tag{4.1}$$

wobei R die Region der Filtermatrix darstellt [Burger & Burge, 2005]. Dieser Vorgang wird im folgenden auch *Faltung* genannt.

Bei der Berechnung einer *Faltung* trifft man unmittelbar auf das Problem, dass Teile der Filtermatrix außerhalb des Bildes liegen, wenn der *Hot Spot* beispielsweise an den Bild-Rand gelegt wird. Es gibt mehrere Wege, um mit den Bild-Rändern umzugehen. In diesem Falle wird das Bild an den Rändern so weit mit Nullen aufgefüllt, bis bei jeder möglichen Platzierung des *Hot Spots* auf dem Bild die *Faltung* berechnet werden kann.

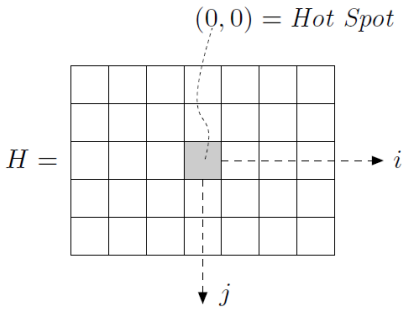


Abbildung 4.1: Beispiel einer Filtermatrix. Die Matrix verfügt über ihr eigenes Koordinatensystem. Der *Hot Spot* liegt üblicherweise in der Mitte an den Koordinaten $(0, 0)$ (entnommen aus Burger & Burge [2005]).

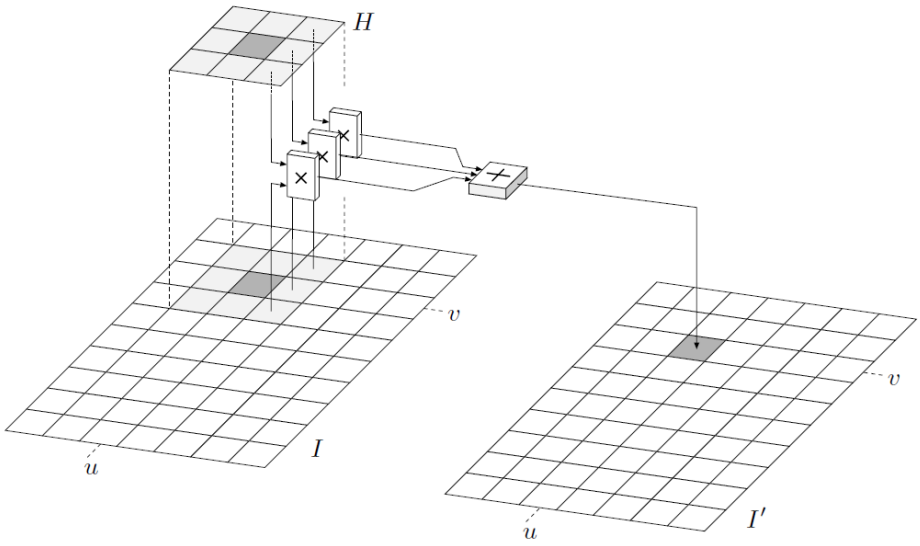


Abbildung 4.2: Anwendung eines linearen *Filters*. An jeder Position (u, v) im Bild wird die Filtermatrix so positioniert, dass der *Hot Spot* auf die aktuelle Position fällt. Anschließend werden alle Bildpunkte mit den darüber liegenden Koeffizienten multipliziert. Die Ergebnisse werden jeweils aufsummiert und am Ende als Ergebnis-Wert an der aktuellen Position (u, v) gespeichert (entnommen aus Burger & Burge [2005]).

4.2 Histogram of Gradients

Der *Histogram of Gradients*-Deskriptor (auch *HoG* genannt) wurde ursprünglich von *Dalal und Triggs* für die Detektion von Personen in Bildern entwickelt und basiert auf Gradienten von Farbwerten [Dalal & Triggs, 2005].

Die Berechnung des Deskriptors beginnt mit der Berechnung der Gradienten in x- und y-Richtung. Von den berechneten Gradienten werden die Stärke (auch *Magnitude*) und die Richtung (auch *Orientation*) berechnet (siehe Abbildung 4.3).

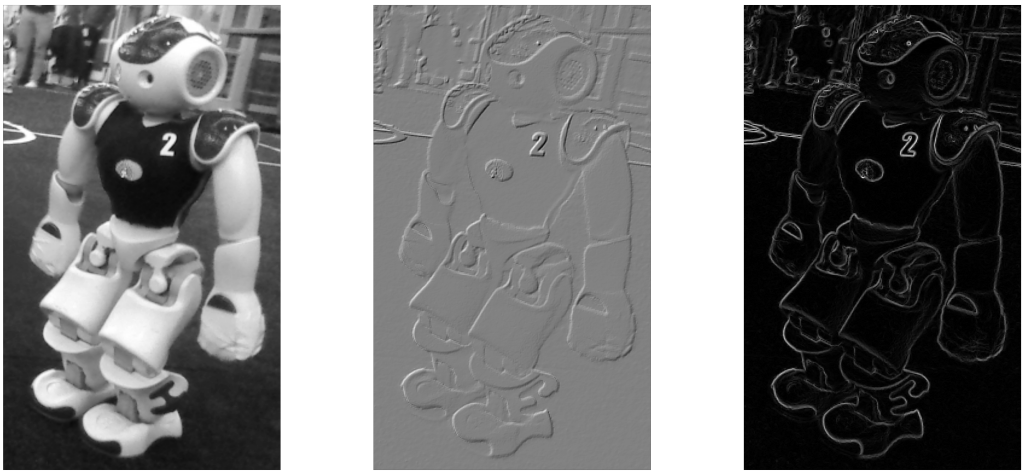


Abbildung 4.3: Die Abbildung zeigt die Kanten (Mitte) und die *Magnitudes* (rechts) des Eingabebildes (links). Starke Kanten werden in weiß (positiv) und schwarz (negativ) dargestellt. Keine oder nur sehr schwache Kanten sind in grau dargestellt. Hohe *Magnitudes* im rechten Bild sind ebenfalls in weiß dargestellt. Hier ist schon gut zu sehen, dass man in Graustufenbildern den Umriss eines *NAO*-Roboters mit diesen Repräsentationen extrahieren kann.

Anschließend wird das Bild in Zellen einer festen Größe unterteilt – in der Regel haben diese Zellen eine Größe von 8×8 Pixeln. Auf Basis dieser Zellen werden Histogramme berechnet (siehe Abbildungen 4.4 und 4.5).

Zum Schluss werden Blöcke von 2×2 benachbarten Histogrammen normalisiert, um eine Invarianz gegen verschiedene Kontraste im Bild zu erhalten.

Die Einträge dieser normalisierten Histogramme werden konkateniert, wodurch sich ein Merkmals-Vektor ergibt mit $4 \times 18 = 72$ Komponenten. Durch eine Hauptkomponentenanalyse konnten *Felzenszwalb et al.* den Merkmals-Vektor auf eine Gesamtlänge von 31 Einträgen reduzieren [Felzenszwalb *et al.*, 2010b, Kapitel 6.2].

Die Größe der Zellen wird dabei so gewählt, dass sie wichtige Merkmale möglichst gut abdecken. Dalal & Triggs [2005] haben eine Größe von 8×8 Pixeln für die Detektion von Personen in statischen Bildern verwendet.

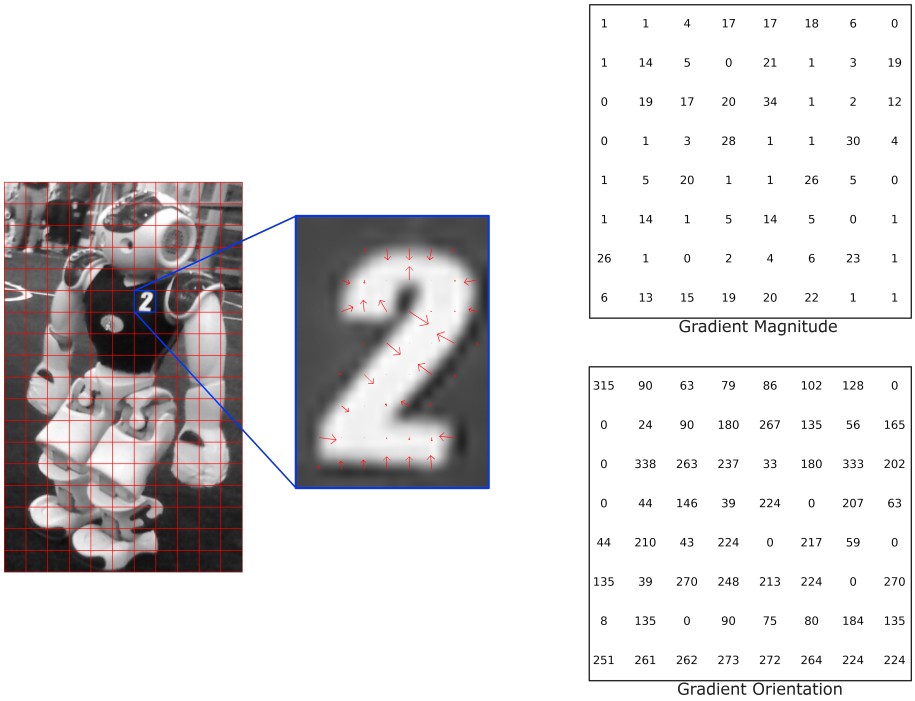


Abbildung 4.4: Die Abbildung zeigt ein Beispiel für die *Magnitudes* und *Orientations* in einer Zelle. Auf der linken Seite ist zu sehen, wie das Eingabebild in Zellen unterteilt wird (in rot dargestellt). In der Mitte sieht man die *Orientations* (rot) einer bestimmten Zelle skaliert nach der *Magnitude*. Auf der rechten Seite sind die *Magnitudes* (oben) und die *Orientations* (unten) der Zelle dargestellt. Aus diesen Zellen werden anhand der *Magnitude* und *Orientation* Histogramme mit 18 Einträgen berechnet. In diesen Einträgen werden die *Magnitudes* interpoliert und auf Basis ihrer *Orientation* aufsummiert.

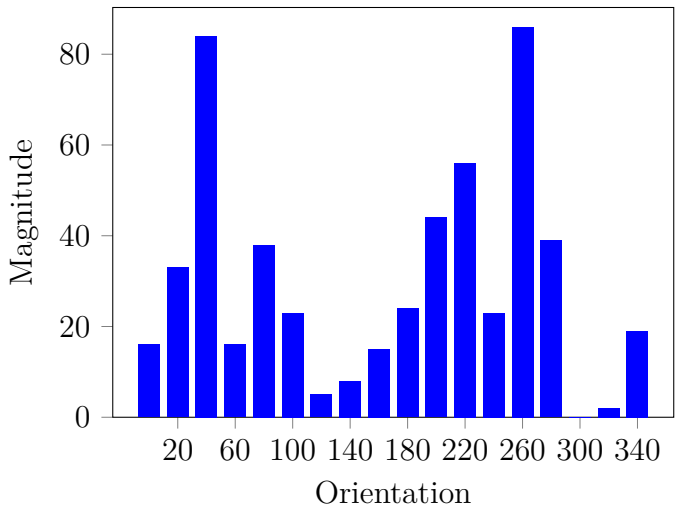


Abbildung 4.5: Histogramm der hervorgehobenen Zelle aus Abbildung 4.4. Die x-Achse zeigt die Einträge, welche die *Orientations* von 0 bis 360 Grad gleichmäßig unterteilen. Eine *Orientation* von 0 und eine von 360 Grad werden hier als gleich angesehen. Die y-Achse zeigt die aufsummierten *Magnitudes*. Dieses Histogramm repräsentiert einen (unnormalisierten) *HoG*-Deskriptor.

4.3 Deformable Part Model

Ein *Deformable Part Model* repräsentiert ein Objekt als eine Menge von *Teilen* und einer *Wurzel*. Die *Teile* unterliegen bestimmten Einschränkungen, was die Verschiebung relativ zur *Wurzel* angeht. Diese Einschränkungen nennen sich *Deformations-Kosten* (auch *deformation costs* genannt). Abbildung 4.6 zeigt ein Beispiel-Modell, welches trainiert wurde, um Personen in Bildern zu detektieren.

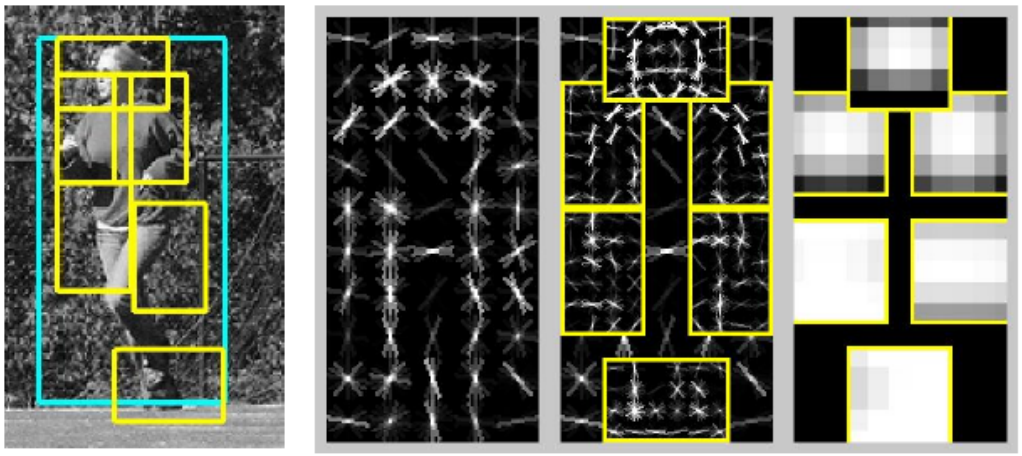


Abbildung 4.6: Beispiel für ein *Deformable Part Model*. Das Modell wird definiert durch eine *Wurzel* (2. von links) und mehreren höher aufgelösten *Teilen* (2. von rechts). Auf der rechten Seite sind die *deformation costs* abgebildet. Links ist eine beispielhafte Detektion einer Person zu sehen. Die *Wurzel* wird in blau dargestellt, die anderen *Teile* in gelb (entnommen aus [Felzenszwalb *et al.*, 2008]).

Ein *Deformable Part Model* mit n *Teilen* ist ein $(n+2)$-Tupel $(F_0, P_1, \ldots, P_n, b)$. F_0 ist in diesem Fall die *Wurzel* des Modells und $b \in \mathbb{R}$ ein *Bias*. P_i beschreibt den i-ten *Teil*. Der *Bias* beschreibt eine Gewichtung des Modells in einem *Mixture-Model*.

Jeder *Teil* P_i eines Modells besteht aus einem 3-Tupel (F_i, v_i, d_i), wobei F_i den *Filter*, $v_i \in \mathbb{R}^2$ den *Anker* und $d_i \in \mathbb{R}^4$ die *deformation costs* für den *Teil* P_i darstellt [Felzenszwalb *et al.*, 2008]. Bei einem *Filter* handelt es sich um Gewichte für einen *Histogram of Gradients*-Deskriptor. Der *Anker* beschreibt die grundsätzliche Verschiebung des *Teiles* gegenüber der *Wurzel*. Ein Teil, der z.B. den Kopf eines Roboters beschreibt, hat eine positive Verschiebung auf der y-Achse. Die *deformation costs* sind ein 4-Tupel (dx, dy, dx^2, dy^2). Ein d_i von $(0, 0, 1, 1)$ würde beispielsweise die quadratische Abweichung zwischen der tatsächlichen Position des *Teiles* und des *Ankers* beschreiben.

Eine Objekt-Hypothese z spezifiziert die Positionen aller *Teile* eines Modells in einer *Merkmal-Pyramide* (siehe Kapitel 5.4):

$$z = (p_0, \ldots, p_n), \tag{4.2}$$

wobei $p_i = (x_i, y_i, l_i)$ die Position und das *Level* des i-ten *Teiles* spezifiziert. Dabei ist vorgegeben, dass das *Level* der Teile die doppelte räumliche Auflösung des *Levels* der *Wurzel* darstellt (siehe Abbildung 4.7).

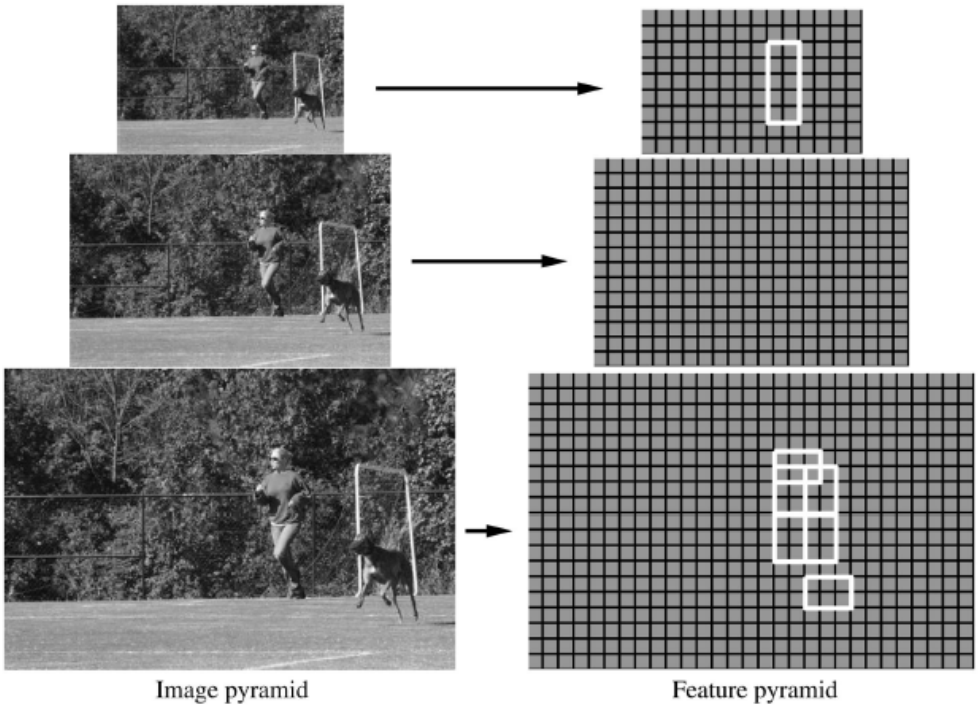


Abbildung 4.7: Beispiel für eine Objekt-Hypothese. Die *Teile* (weiße Boxen im unteren Teil) befinden sich auf der doppelten räumlichen Auflösung der *Wurzel* (im oberen Teil zu sehen) (entnommen aus [Felzenszwalb *et al.*, 2010b]).

Die Bewertung einer Objekt-Hypothese setzt sich dabei aus der Bewertung jedes einzelnen *Teiles* an der jeweiligen Position zusammen. Zusätzlich wird pro *Teil* ein Fehlerwert aufgerechnet, welcher sich aus den *deformation costs* und der Verschiebung relativ zur *Wurzel* ergibt:

$$score(p_0, \ldots, p_n) = \underbrace{\sum_{i=0}^{n} F_i' \cdot \phi(H, p_i)}_{\text{Filter-Antwort}} - \underbrace{\sum_{i=1}^{n} d_i \cdot \phi_d(dx_i, dy_i)}_{\text{Platzierung des Teiles}} + b, \tag{4.3}$$

wobei die *Filter*-Antwort eine *Faltung* mit einem *Filter* F und den Merkmalen ϕ,

$$(dx_i, dy_i) = (x_i, y_i) - (2(x_0, y_0) + v_i) \tag{4.4}$$

die Verschiebung vom i-ten *Teil* gegenüber der *Wurzel* und

$$\phi_d(d_x, d_y) = (dx, dy, dx^2, dy^2) \tag{4.5}$$

die *deformation costs* sind (Formeln aus [Felzenszwalb *et al.*, 2010b]).

Die Idee ist also, dass es eine optimale Position aller *Teile* gibt, bei der sie nur nach den im *Anker* vorgegebenen Werten relativ zur *Wurzel* verschoben sind. Jede Verschiebung gegenüber dieser Position wird durch die *deformation costs* bestraft und resultiert in einer niedrigeren Bewertung.

4.4 Lochkamera-Modell

Das grundlegende Kameramodell dieser Arbeit ist das Modell der Lochkamera (siehe Abb. 4.8).

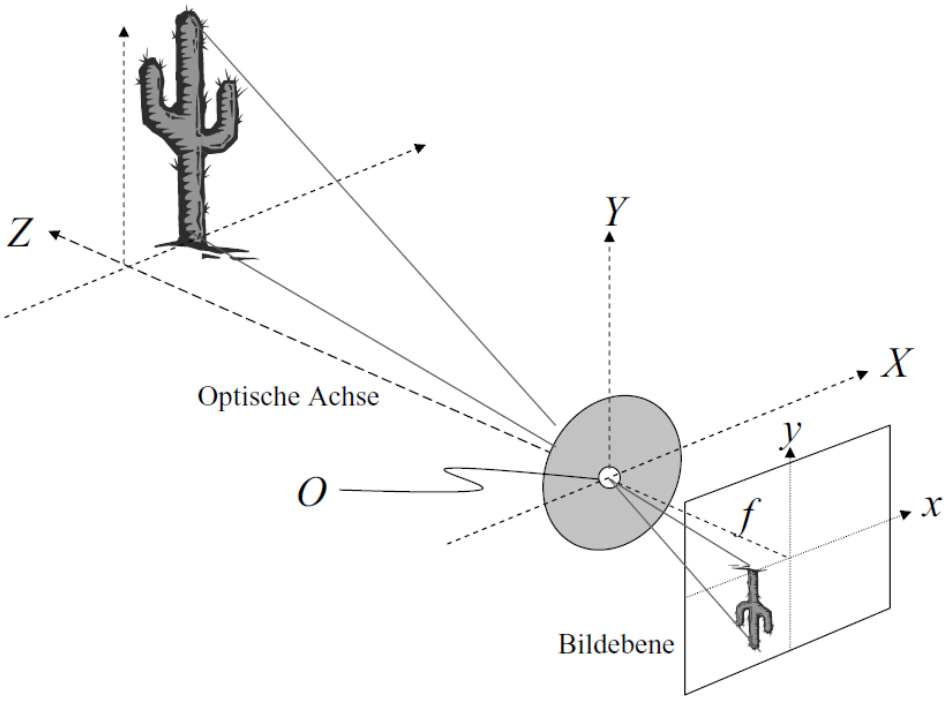


Abbildung 4.8: Darstellung des Lochkamera-Modells. Bei diesem Modell besteht die Kamera aus einer geschlossenen Box mit einer sehr kleinen Öffnung an der Vorderseite. Die Bildebene befindet sich an der gegenüberliegenden Seite. Lichtstrahlen, die durch die Öffnung fallen, werden geradlinig an die Bildebene projiziert, womit ein verkleinertes, horizontal gespiegeltes Abbild an der Bildebene entsteht (entnommen aus [Burger & Burge, 2005, Seite 7]).

Bei der Projektion eines dreidimensionalen Punktes (X, Y, Z) auf die Bildebene wird der Vergleich der ähnlichen Dreiecke angewendet. Z beschreibt dabei die Distanz eines Objektes von der Lochebene und Y den vertikalen Abstand über der optischen Achse. Die optische Achse läuft gerade durch die Lochöffnung und steht orthogonal zur Bildebene. Die Höhe der zugehörigen Projektion y ist dann durch zwei Parameter bestimmt: die Tiefe der Kamerabox f (auch *focal length* oder Brennweite genannt) und dem Abstand Z vom Koordinatenursprung (Lochöffnung).

Durch den Vergleich der ähnlichen Dreiecke, die somit auf beiden Seiten der Lochöffnung entstehen, ergeben sich folgende Formeln:

$$y = -f\frac{Y}{Z} \text{ und } x = -f\frac{X}{Z}. \tag{4.6}$$

In der Realität findet das Lochkamera-Modell keine Anwendung mehr. Stattdessen nutzt man Linsen, um die Lichtstrahlen zu brechen (siehe Abb. 4.9).

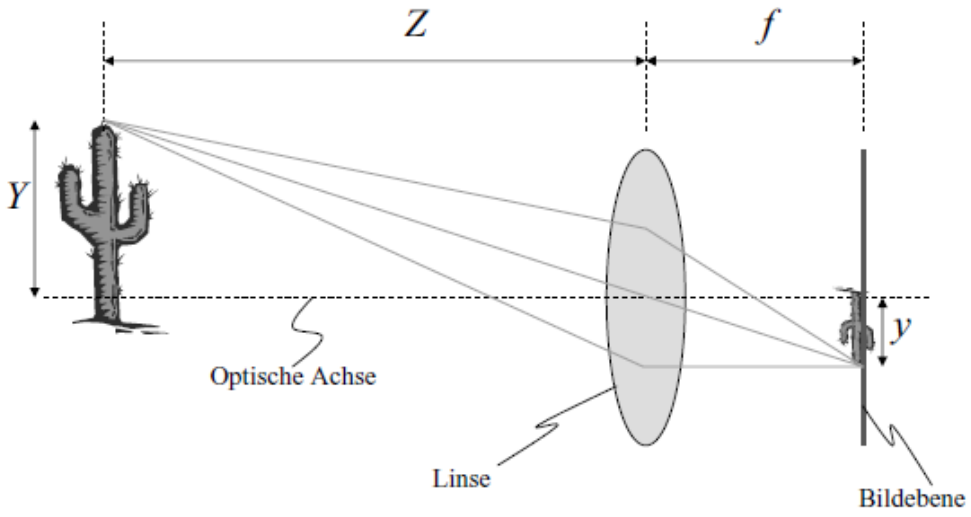


Abbildung 4.9: Darstellung der "dünnen Linse". Das entstehende Modell ist dem Lochkamera-Modell sehr ähnlich. Die Linse wird dabei als symmetrisch und unendlich dünn angenommen. Lichtstrahlen werden an einer virtuellen Ebene in der Linsenmitte gebrochen, wobei sich die gleiche Abbildungsgeometrie wie bei der Lochkamera ergibt (entnommen aus [Burger & Burge, 2005, Seite 9]).

Auch dieses Modell ist allerdings noch nicht vollständig, da Kriterien wie Schärfe, Verzerrungen (z.B. durch runde Linsen) und andere reale Effekte nicht berücksichtigt werden.

4.5 Metriken

Die Grundlage für die meisten Metriken ist die *Konfusionsmatrix*. Diese Matrix gibt an, welche Vorhersagen für eine ausgewählte Menge getroffen wurden und unterteilt diese insgesamt in vier Kategorien: *True positives* (TP), *False positives* (FP), *True negatives* (TN), *False negatives* (FN) (siehe Tabelle 4.1). Zur besseren Lesbarkeit werden diese Klassen im weiteren Verlauf der Arbeit abgekürzt.

		Tatsächliche Klasse	
		Positive	Negative
Vorhergesagte Klasse	Positive	TP	FP
	Negative	FN	TN

Tabelle 4.1: *Konfusionsmatrix* für ein binäres Klassifikations-Problem.

Da es sich bei dem in dieser Arbeit behandelten Problem um ein Detektions-Problem handelt, müssen die Vorhersagen (in diesem Falle sind es *Bounding-Boxes*[1]) auf irgendeine Weise in die Klassen der *Konfusionsmatrix* diskretisiert werden. Zu diesem Zwecke wird die *Intersection over Union*-Metrik genutzt, welche im folgenden Kapitel erläutert wird.

[1]Eine *Bounding-Box* ist das kleinste Rechteck, dass ein Objekt in einem Bild vollständig umschließt.

4.5.1 Intersection over Union (IoU)

Intersection over Union (auch: *IoU*) ist eine Metrik, welche das Verhältnis zwischen zwei Mengen beschreibt. Der Wertebereich dieser Metrik liegt zwischen 0 und 1, wobei 1 eine totale Übereinstimmung der beiden Mengen beschreibt und bei 0 gar keine Übereinstimmung vorliegt. Diese Metrik findet bei der Evaluation von Objekt-Detektoren viel Anwendung, wobei es sich in diesem Fall bei den zu vergleichenden Mengen um die tatsächlichen *Bounding-Boxes* (auch *ground-truth* genannt) und die von einem Detektor vorhergesagten *Bounding-Boxes* handelt. Abbildung 4.10 zeigt ein Beispiel für eine Detektion eines humanoiden *NAO*-Roboters. Die *IoU* ist dabei definiert als Verhältnis zwischen der überlappenden Fläche und der Gesamt-Fläche der beiden Mengen.

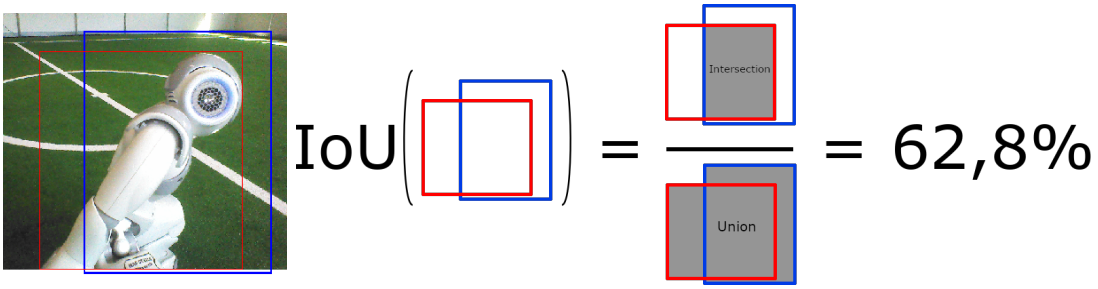


Abbildung 4.10: Beispiel für eine *IoU*-Berechnung. Die *ground-truth Bounding-Box* wird in rot dargestellt, die Detektion des Roboters in blau. Die für die Berechnung relevanten Flächen sind grau markiert.

Um Vorhersagen mittels der *IoU*-Metrik zu diskretisieren, muss also ein *Schwellwert* t definiert werden, ab dem eine Vorhersage als *Positive* angesehen wird. Auf Basis dieses *Schwellwerts* sind die Einträge der *Konfusionsmatrix* für den in dieser Arbeit behandelten Fall wie folgt definiert:

- **True Positive**: Eine Vorhersage, dessen *IoU* mit einer *ground-truth Bounding-Box* $\geq t$ ist.
- **False Positive**: Eine Vorhersage, dessen *IoU* mit allen *ground-truth Bounding-Boxes* $< t$ ist.
- **True Negative**: Das Fehlen von Vorhersagen auf einem Bild ohne Roboter.
- **False Negative**: Das Fehlen einer Vorhersage mit einem *IoU* $\geq t$ für eine *ground-truth Bounding-Box*.

Nachdem die Vorhersagen mittels der *IoU*-Metrik diskretisiert wurden, können andere Metriken auf diesen berechnet werden.

4.5.2 Precision-Recall-Curve

Die *Precision-Recall-Curve* ist eine Metrik zur Evaluation von binären Klassifikatoren. Diese Metrik baut auf der *Konfusionsmatrix* auf, berücksichtigt aber nicht die *True Negatives*. Es wird also nur untersucht, wie gut der Klassifikator die positiven Klassen richtig vorhersagt.

Eine *Precision-Recall-Curve* ist ein Graph, der in der x-Achse den *Recall* und in der y-Achse die *Precision* des Klassifikators für verschiedene *Schwellwerte* enthält. *Precision* und *Recall* haben jeweils einen Wertebereich von 0 bis 1 und sind gegeben durch die Formeln:

$$Recall = \frac{TP}{TP + FN} \tag{4.7}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.8}$$

Die *Precision* gibt also an, wie akkurat die Vorhersagen sind. Ein Klassifikator mit einer *Precision* von 0,7 liegt beispielsweise bei einer Klassifikation zu 70 % mit der Vorhersage richtig. Die *Precision* wird in dieser Arbeit auch mit der Erkennungsrate gleichgesetzt. Der *Recall* gibt an, wie gut der Klassifikator darin ist, die positive Klasse zu finden. Ein Klassifikator mit einem *Recall* von 0,5 findet beispielsweise 50% der positiven Klassen. Die optimalen Werte für *Precision* und *Recall* sind jeweils 1,0.

Die Idee hinter der *Precision-Recall-Curve* ist, die *Precision* und den *Recall* für verschiedene *Schwellwerte* zu berechnen und darzustellen. Dafür werden die Vorhersagen absteigend nach ihren Bewertungen sortiert und anschließend für eine immer größer werdende Menge die beiden Metriken berechnet. Abbildung 4.11 zeigt drei beispielhafte *Precision-Recall-Curves*.

Zum Vergleich von verschiedenen *Precision-Recall-Curves* ist es nützlich, diese in einem Wert zusammenzufassen. Eine Möglichkeit, das zu tun, ist die *Average Precision* (im folgenden auch AP genannt) [Jonathan Hui, 2018]. Die Idee hinter der AP ist es, den Durchschnitt über alle *Recalls* in der Menge $R = \{0{,}0, \ldots, 1{,}0\}$ zu bilden. Die Menge R wird dabei in Schritten von 0,1 diskretisiert, wobei sich 11 Werte ergeben. Für jeden *Recall* $r \in R$ wird die maximale *Precision* nach der folgenden Formel gewählt:

$$p_{interp}(r) = \max_{\widetilde{r} \geq r} p(\widetilde{r}), \tag{4.9}$$

wobei p die *Precision* aus der *Precision-Recall-Curve* für den *Recall* $r \in R$ liefert. Die *Precision-Recall-Curve* wird also begradigt. Die AP ist letztendlich definiert als der Durchschnitt aller *Precisions*:

$$AP = \frac{1}{11} \sum_{r \in R} p_{interp}(r). \tag{4.10}$$

Der Wertebereich der AP liegt ebenfalls zwischen 0,0 und 1,0.

4.6 B-Human Framework

Das Framework *B-Human* bietet viele Werkzeuge, um Software für den *NAO* zu entwickeln und zu testen. Das Herzstück ist der Simulator *SimRobot*, welcher auch in der Entwicklung der in dieser Arbeit beschriebenen Software benutzt wurde [Laue & Röfer, 2008]. Neben der Simulation von Robotern in einem dreidimensionalen Raum, kann man sich auch mit einem physischen Roboter verbinden und über *SimRobot* in Echtzeit die Sensordaten einsehen.

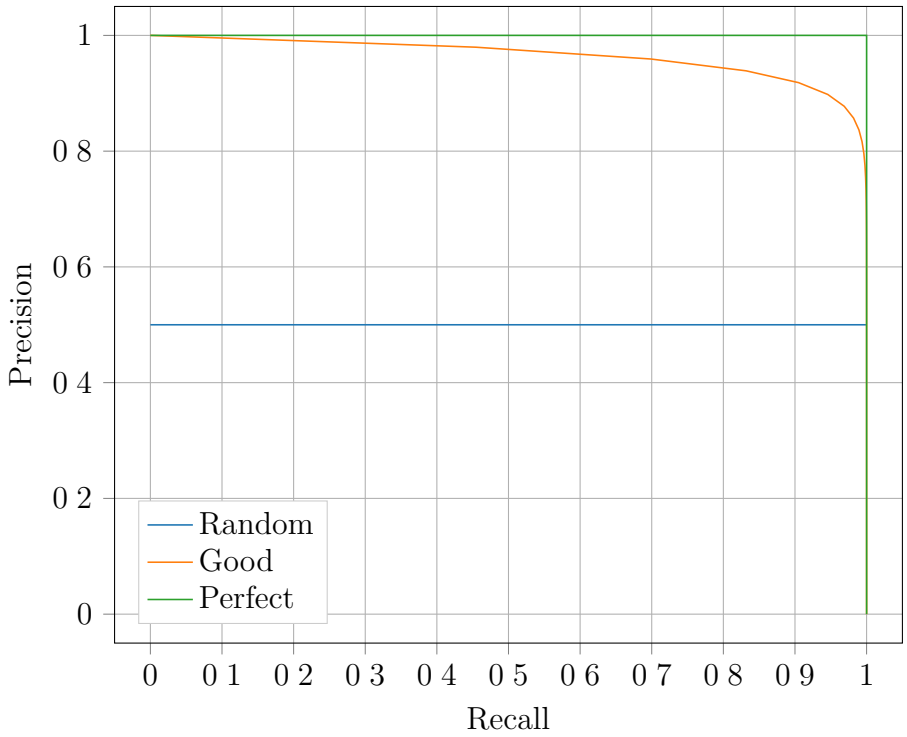


Abbildung 4.11: Die Abbildung zeigt drei Beispiele für *Precision-Recall-Curves*. Die x-Achse beschreibt den *Recall* und die y-Achse beschreibt die *Precision*. Ein perfekter Klassifikator liefert durchgehend eine *Precision* von 1,0 und fällt erst gegen Ende stark ab (grün). Ein Klassifikator, welcher nicht besser ist als der reine Zufall, liefert Ergebnisse, welche einer horizontalen Diagonalen entsprechen. Bei einem perfekt ausbalancierten Datensatz befindet sich diese auf der Höhe 0,5 (blau). Ein guter Klassifikator liefert durchgehend eine *Precision*, welche über dem Mittelwert liegt und fällt erst gegen Ende ab (orange).

B-Human baut auf einem *Modul-Framework* auf, bei dem *Module* so genannte *Repräsentationen* bereitstellen. Außerdem können *Module Repräsentationen* anfordern, die von anderen *Modulen* bereitgestellt werden.

Bei *Repräsentationen* handelt es sich um Informationen, auf dessen Basis weitere Berechnungen gemacht werden sollen. Im Falle eines *Moduls*, welches Roboter in Bildern detektiert, kann zum Beispiel über die *Repräsentation* die Positionen der Roboter kommuniziert werden. Das *Modul*, welches im Zuge dieser Arbeit entstanden ist, wurde auch in diesem Framework entwickelt.

5. Umsetzung

In diesem Kapitel wird auf alle Einzelheiten des entwickelten Systems eingegangen.

5.1 Ein- und Ausgaben

Im Zuge dieser Arbeit habe ich einen Detektor im *B-Human*-Framework entwickelt. Mir standen dabei alle Werkzeuge dieses Frameworks zur Verfügung.

Bei den Eingaben handelt es sich dementsprechend um die *Repräsentationen*, welche vom Detektor vorausgesetzt werden.

Dabei gibt es zum einen die intrinsischen und extrinsischen Kamera-Parameter, welche benötigt werden, um *Transformationen* zwischen Koordinaten-Systemen durchführen zu können und Objekt-Größen auf bestimmten Entfernungen zu berechnen (siehe Kapitel 5.3).

Die *Feldgrenze* wird ebenfalls benutzt[1]. Diese begrenzt den Bereich, auf dem sich Roboter bewegen können. Projiziert man diese aus dem Bild auf das Spielfeld, hat man also eine sehr gute Abmessung davon, auf welcher maximalen Distanz sich derzeit Roboter befinden können. Diese wird in der Vorverarbeitung benutzt (siehe Kapitel 5.3).

Das aktuelle Bild liegt in einer *Repräsentation* als Graustufenbild vor. Das Bild der oberen Kamera hat dabei eine Größe von 640×480 Pixeln.

Die Ausgabe des Detektors ist eine Liste von *Bounding-Boxes*. Eine *Bounding-Box* ist dabei definiert durch vier Punkte im Bild und beschreibt das kleinste umschließende Rechteck für einen humanoiden *NAO*-Roboter im Bild (siehe Abbildung 5.1).

5.2 Überblick

Das Framework *B-Human* ist so konzipiert, dass jedes *Modul* eine *Update*-Funktion hat, welche in jedem Durchlauf aufgerufen wird. In jedem Durchlauf stehen immer

[1] Die Feldgrenze ist die äußerste Grenze des Spielfeldes, welche durch den Teppich begrenzt wird.

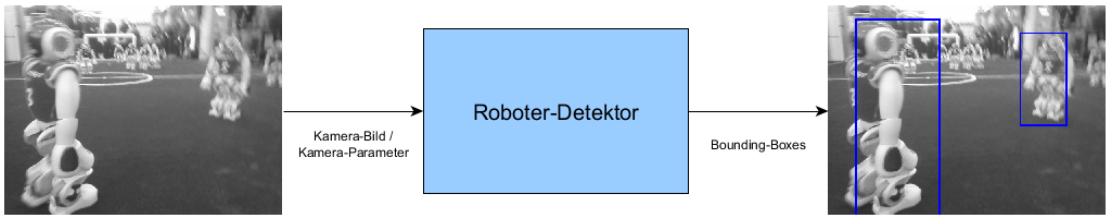


Abbildung 5.1: Beispiel für Ein- und Ausgaben des entwickelten Detektors.

die aktuellen *Repräsentationen* zur Verfügung. Ein Aufruf des entwickelten Roboter-Detektors lässt sich dabei in mehrere Abschnitte unterteilen:

Vorbereitungen: Hier werden die Vorbereitungen für den restlichen Ablauf des *Moduls* getroffen. Neben der Initialisierung der wichtigen Parameter wird hier die maximale Entfernung ermittelt, in der sich momentan ein Roboter befinden kann. Dafür werden u.a. die Kamera-Parameter verwendet. Auf Basis dieser Berechnung werden die *Level* einer *Merkmal-Pyramide* gewählt, die in einem Durchlauf berechnet werden sollen.

Matching: Hier werden die *Deformable Part Models* verwendet, um Roboter im momentanen Bild zu detektieren. Zu diesem Zwecke wird zuerst eine *Merkmal-Pyramide* auf den vorher ausgewählten *Leveln* berechnet. Anschließend werden die Antworten der *Wurzeln* berechnet, eher die Merkmale auch mit den *Teilen* gefaltet werden. Durch Verrechnen der Antworten ergeben sich Hypothesen. Schlechte Hypothesen werden dabei durch einen *Schwellwert* ausgefiltert.

Nachbereitung: In der Nachbereitung werden die übrigen Hypothesen verarbeitet. Das Verfahren, das mit *Deformable Part Models* Objekte detektiert, hat die Eigenheit, mehrere Detektionen für dasselbe Objekt zu liefern. Mit einem Verfahren namens *Nonmax-Suppression* werden diese multiplen Detektionen aussortiert.

Abbildung 5.2 zeigt einen Überblick über das entwickelte System. In den folgenden Abschnitten wird im Detail auf die einzelnen Schritte eingegangen.

5.3 Vorbereitung

Die wichtigste Aufgabe der Vorbereitungen ist es, den Suchbereich zu reduzieren, auf dem die aufwendigen Operationen ausgeführt werden müssen. Neben einem verringerten Speicher-Verbrauch und einer besseren Laufzeit kann man mit den richtigen Vorbereitungen auch Falsch-Erkennungen eindämmen. Im Falle der *Deformable Part Models* gibt es mehrere Möglichkeiten, wie man dieses Ziel erreichen kann. Zum einen könnte man die Bereiche im Bild begrenzen, auf denen die Berechnungen gemacht werden. Zusätzlich könnte man die Anzahl der *Level* begrenzen, die berechnet werden müssen.

Wenn man vor der Detektion den Bereich im Bild stark eingrenzt, kann man das Problem der Detektion zu einer Klassifikation verringern. Diesen Ansatz haben z.B. Albani *et al.* [2017] verwendet.

Eine Reduzierung des Suchbereiches im Hinblick auf die *Level* der *Merkmal-Pyramide* (siehe Kapitel 5.4.1) könnte in diesem Anwendungsbereich besonders interessant sein,

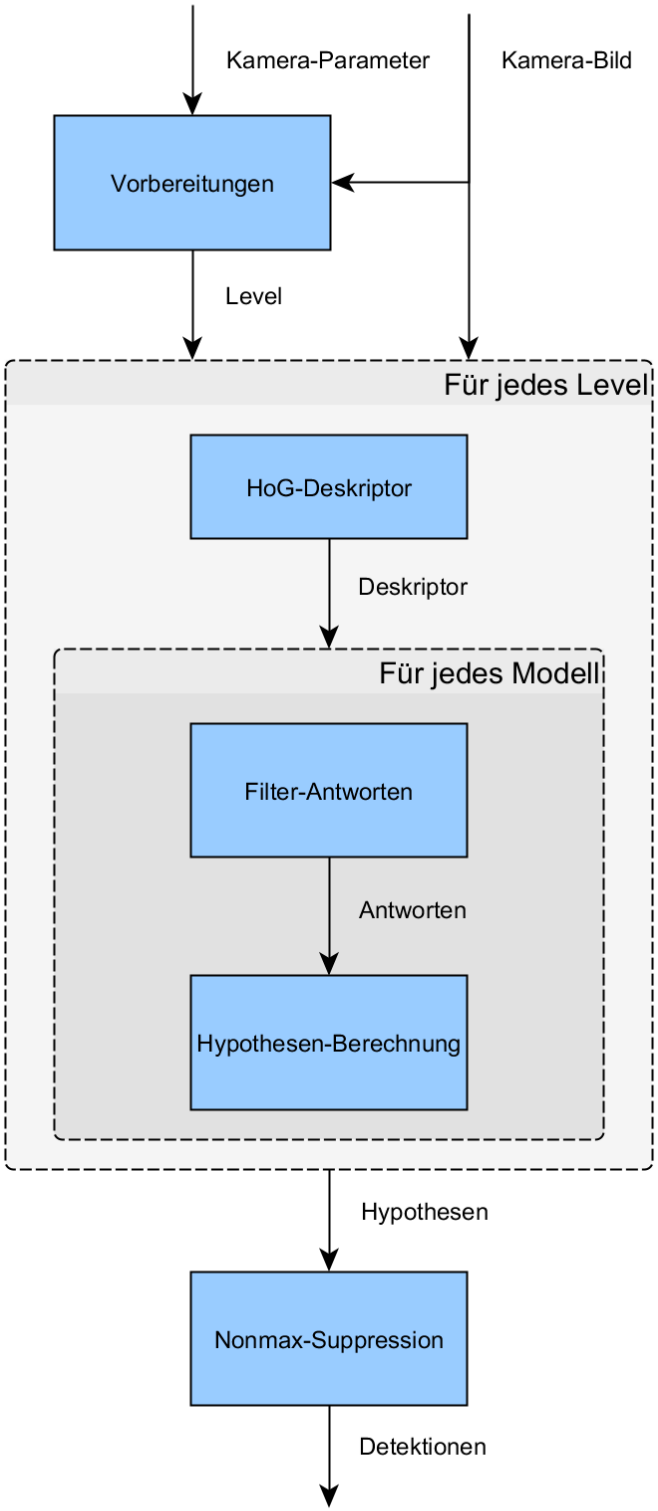


Abbildung 5.2: Überblick über das entwickelte System.

da so Falsch-Erkennungen von Objekten außerhalb des Spielfeldes minimiert werden könnten. Die Gefahr könnte bei den *Deformable Part Models* besonders groß sein, da nicht auszuschließen ist, dass auch Personen außerhalb des Spielfeldes als Roboter detektiert werden.

Aus diesem Grund nehme ich eine Reduzierung des Suchbereiches in den *Leveln* der *Merkmal-Pyramide* vor. Hier wird also kontrolliert die Reichweite des Detektors eingedämmt. Der originale Algorithmus von Felzenszwalb *et al.* [2010b] enthielt keine solche Reduzierung. Ein Grund dafür könnte sein, dass die *Deformable Part Models* darauf ausgelegt sind, auf Bildern als einzige Eingabe funktionieren zu können. Im *B-Human*-Framework haben wir allerdings auch die Möglichkeit, Entfernungen von Objekten im Bild relativ zum Roboter zu berechnen, da die relative Position der Kameras bekannt ist.

Diese Reduzierung basiert auf der Annahme, dass eine *Wurzel* immer das gesamte Objekt im Bild abdeckt (siehe dazu Kapitel 4.3). Im Training der Modelle werden die *Filter*-Größen so gewählt, dass sie das zu modellierende Objekt möglichst gut im Bild abdecken. Abbildung 5.3 zeigt die Aufteilung der *Level* einer *Merkmal-Pyramide* (siehe Kapitel 5.4.1) auf dem Spielfeld. Insbesondere erkennt man dabei, dass der abgedeckte Bereich einer *Wurzel* mit jedem *Level* der *Pyramide* größer wird. Projiziert man also eine *Wurzel* mit der obigen Annahme auf das Spielfeld und berechnet die Entfernung zum momentanen Roboter, erhält man den optimalen Abstand für die jeweilige *Wurzel*. Dies lässt sich für beliebige *Level* der *Merkmal-Pyramide* umsetzen, indem man vorher die Dimensionen der *Wurzel* passend zur Bildgröße skaliert.

Für diesen Zweck wähle ich allerdings nicht die kleinste *Wurzel* aus dem *Mixture-Model*, sondern einen Durchschnittswert aller *Wurzel*-Größen. Außerdem benötigt man die Größe des zu detektierenden Objektes und die Brennweite der zu verwendenden Kamera (auch: *Focal length*).

Die Idee hinter den folgenden Berechnungen nennt sich *Ähnlichkeitssatz* (auch: *Triangle similarity*) und wurde bereits in Kapitel 4.4 erwähnt. Wie mitunter in Abbildung 4.8 und 4.9 zu sehen ist, entstehen zwei ähnliche Dreiecke bei der Projektion eines Objektes auf eine Bildebene. Die Verhältnisse zwischen den einzelnen Seiten dieser Dreiecke sind die selben. Es gilt also:

$$\frac{Z}{Y} = \frac{f}{y}, \tag{5.1}$$

wobei y die Größe des Objektes auf dem Sensor, f die Brennweite, Y die reale Größe des Objektes und Z die Distanz des Objektes zur Kamera ist [Rosebrock, 2015].

Wenn die Brennweite bekannt ist, kann man Formel 5.1 nach Z umstellen:

$$Z = \frac{Y \times f}{y}. \tag{5.2}$$

Die reale Größe Y der Roboter ist gegeben durch die Dokumentation von *Softbank Robotics* [SoftBank Robotics, 2019a]. Die Größe auf dem Kamera-Sensor y ergibt sich aus der Größe der *Wurzel* eines Modells. Die Brennweite f ist durch das Framework *B-Human* gegeben. Für jedes *Level* kann man also die optimale Entfernung berechnen,

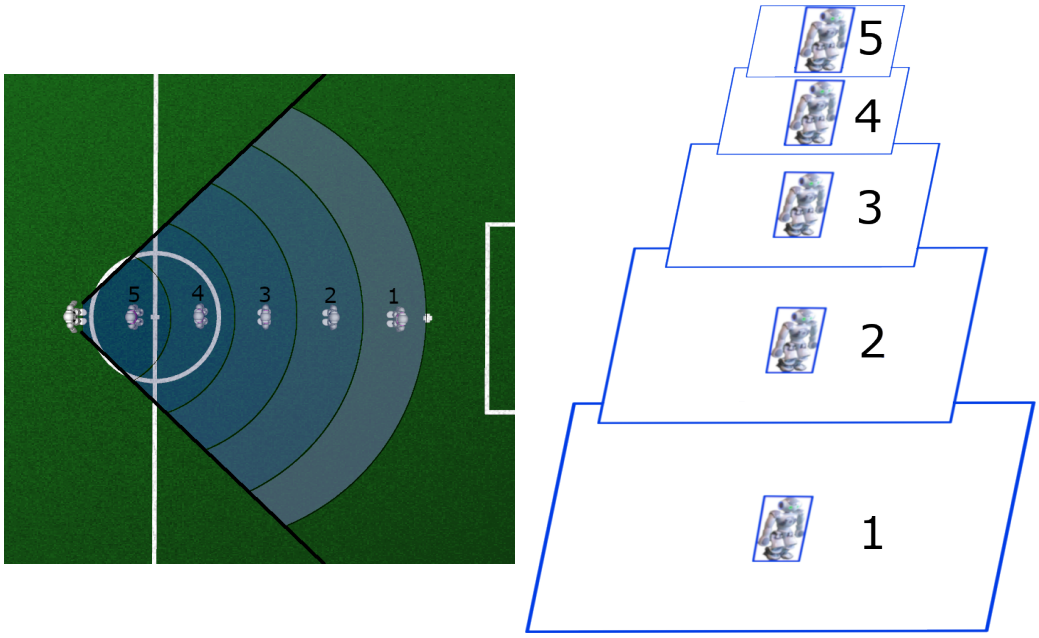


Abbildung 5.3: Beispiel-Zeichnung für 5 *Level*. In der linken Abbildung begrenzen die schwarzen Linien den Sichtbereich des Roboters. Die *Level* sind in verschiedenen Blautönen eingefärbt und durchnummeriert dargestellt. *Level* 1 beschreibt hier das erste *Level* einer *Merkmal-Pyramide* und somit das voll aufgelöste Bild. Je weiter man nun die *Pyramide* durchläuft, desto kleiner wird das verarbeitete Bild und somit auch der Abdeckungs-Bereich der *Teil-Filter* – wie auch in der rechten Abbildung zu sehen ist. Die Übergänge zwischen den einzelnen Bereichen sind fließend. Das heißt, dass mehrere *Level* für den selben Roboter hohe Bewertungen erreichen können. Befinden sich einer oder mehrere der Bereiche außerhalb des Spielfeldes, werden sie nicht berücksichtigt.

indem man die durchschnittliche Größe der *Wurzeln* entsprechend skaliert – da sie auf jedem *Level* eine andere Fläche einnehmen – und alles in die Formel 5.2 einsetzt.

Um nun *Level* ausschließen zu können, benötigt man außerdem die maximale Entfernung im aktuellen Bild. Um die momentane maximale Entfernung zu ermitteln, wird die *Feldgrenze* verwendet, welche durch das Framework zur Verfügung steht. Diese wird in regelmäßigen Abständen abgetastet, wobei die Punkte immer auf Feld-Koordinaten umgerechnet werden, um den Abstand zum momentanen Roboter zu bestimmen. Der maximal ermittelte Abstand wird dabei gewählt. Unterschreitet die maximale Entfernung auf dem Spielfeld die optimale Entfernung der *Wurzeln*, welche nach Formel 5.2 berechnet wurde, werden dementsprechend *Level* aussortiert und nicht berechnet.

5.4 Matching

Den Matching-Prozess habe ich so implementiert, wie er in Felzenszwalb *et al.* [2010b] beschrieben wurde. Folglich stammen auch die folgenden Formeln aus dieser Arbeit.

Die Idee hinter dem Matching ist es, Bewertungen für *Wurzel*-Positionen in Verbindung mit einer optimalen Platzierung der *Teile* zu berechnen:

$$score(p_0) = \max_{p1,\ldots,p_n} score(p_0, \ldots, p_n), \tag{5.3}$$

wobei $p_n = (x, y, l)$ die Position (x, y) des n-ten *Teiles* im l-ten *Level* einer *Merkmal-Pyramide* ist. Die Bewertung ergibt sich aus den *Teil*-Antworten und den *deformation costs* (siehe Formel 4.3).

Wurzel-Positionen mit hohen Bewertungen stellen Detektionen dar. Eine *Wurzel*-Position mit definierten *Teil*-Positionen stellt eine Hypothese dar (siehe Formel 4.2).

5.4.1 Merkmal-Pyramide

Die Ausgangslage ist eine *Merkmal-Pyramide*. Diese besteht aus mehreren verschieden skalierten Versionen des Eingabebildes, welche auch als *Level* bezeichnet werden. Eine *Oktave* beschreibt die Anzahl von *Leveln*, die man in der *Pyramide* durchlaufen muss, um ein Bild mit der halben Auflösung des Originals zu erhalten. Die einzelnen *Level* werden dabei durch wiederholtes Glätten und Unterabtasten berechnet.

Die Glättung ist besonders wichtig, da ansonsten durch die Unterabtastung sehr feine Gradienten verloren gehen könnten. Sie darf allerdings nicht zu stark sein, da das ansonsten den selben Effekt haben könnte. In der Praxis habe ich die *Level* so implementiert, dass ich beim Unterabtasten zwischen benachbarten Pixeln interpoliert habe. Abbildung 5.5 zeigt einige verschieden skalierte Versionen eines Eingabebildes.

Im nächsten Schritt werden die *Histogram of Gradients*-Deskriptoren wie in Kapitel 4.2 beschrieben auf jedem Bild der *Merkmal-Pyramide* berechnet. Abbildung 5.4 zeigt ein Beispiel.

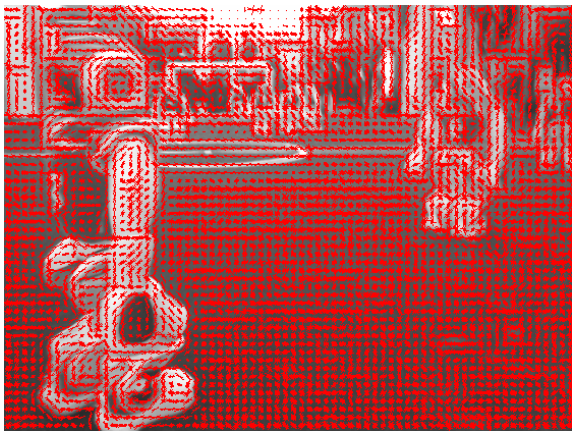


Abbildung 5.4: *Histogram of Gradients*-Deskriptoren. Zur besseren Veranschaulichung wurden die Gradienten um 90° gedreht.

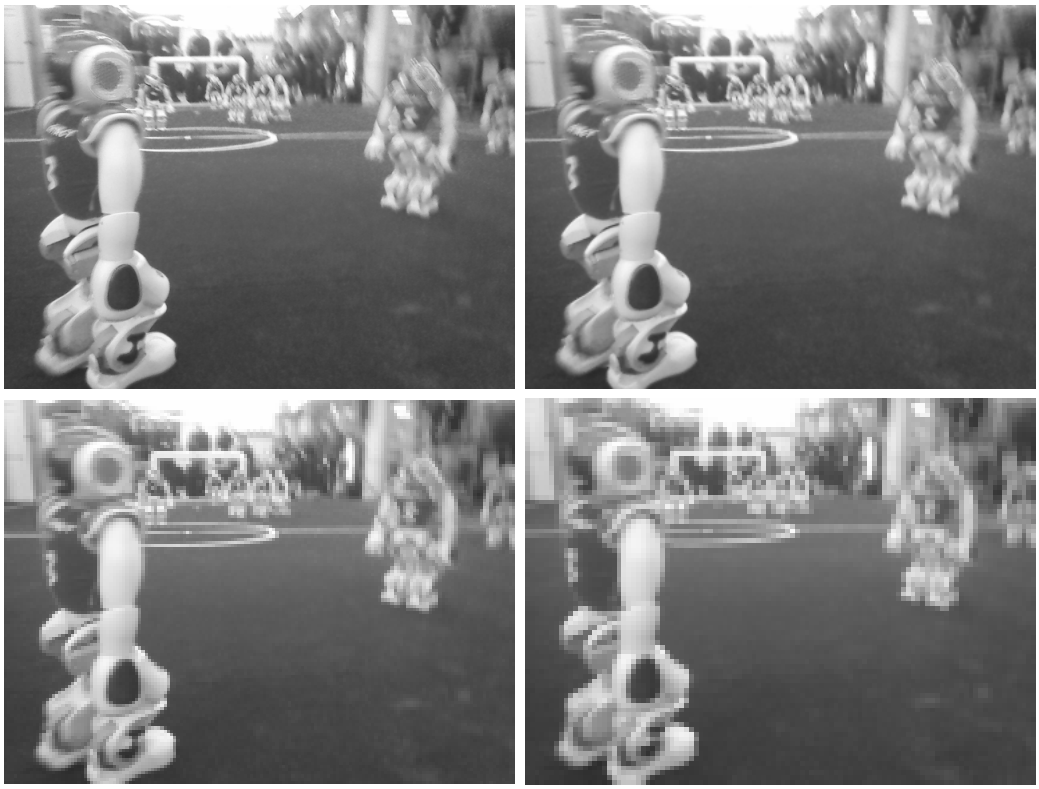


Abbildung 5.5: Verschieden skalierte Versionen eines Eingabebildes. Selbst bei sehr kleinen Skalierungen gehen nur wenige Gradienten verloren. Die Skalierungen (Vielfache der Original-Auflösung) von links oben nach rechts unten: 1, 0,57, 0,25, 0,16.

5.4.2 Filter-Antworten

Im nächsten Schritt werden die Antworten $R_{i,l}$ von jedem *Filter* i auf jedem *Level* l berechnet, wobei diese ein Skalarprodukt zwischen dem *Filter* und dem Merkmals-Vektor $\phi(H,(x,y)$ aus der *Merkmal-Pyramide* sind:

$$R_{i,l}(x,y) = F_i \cdot \phi(H,(x,y)) \tag{5.4}$$

Wobei im Skalarprodukt die Werte des *Filters* F_i und des Merkmals-Vektors $\phi(H)$ zeilenweise aufgerechnet werden (siehe Abbildung 5.6).

Da die *Teile* gegenüber der *Wurzel* verschoben sein können, müssen die Antworten in einem weiteren Schritt noch transformiert werden:

$$D_{i,l}(x,y) = \max_{dx,dy}(R_{i,l}(x+dx, y+dy) - d_i \cdot \phi_d(dx,dy)). \tag{5.5}$$

Diese Transformation – *Felzenszwalb* bezeichnet sie als *distance transform* – verteilt hohe *Teil*-Antworten über einen Bereich (dx,dy) und bezieht die *deformation costs* mit ein. d_i bezeichnet dementsprechend die aktuelle Verschiebung $(x-dx, y-dy)$ und ϕ_d bezeichnet die gelernten *deformation costs* des Modells (siehe auch Formel 4.5). Sie wird nur auf *Teil*-Antworten durchgeführt und nicht auf den *Wurzel*-Antworten. Ein Wert in der transformierten Antwort $D_{i,l}(x,y)$ entspricht der maximalen *Teil*-Bewertung des *Teiles* i in einer *Wurzel*-Bewertung, die ihren *Anker* an die Stelle (x,y) setzen würde (siehe auch Abbildung 5.7).

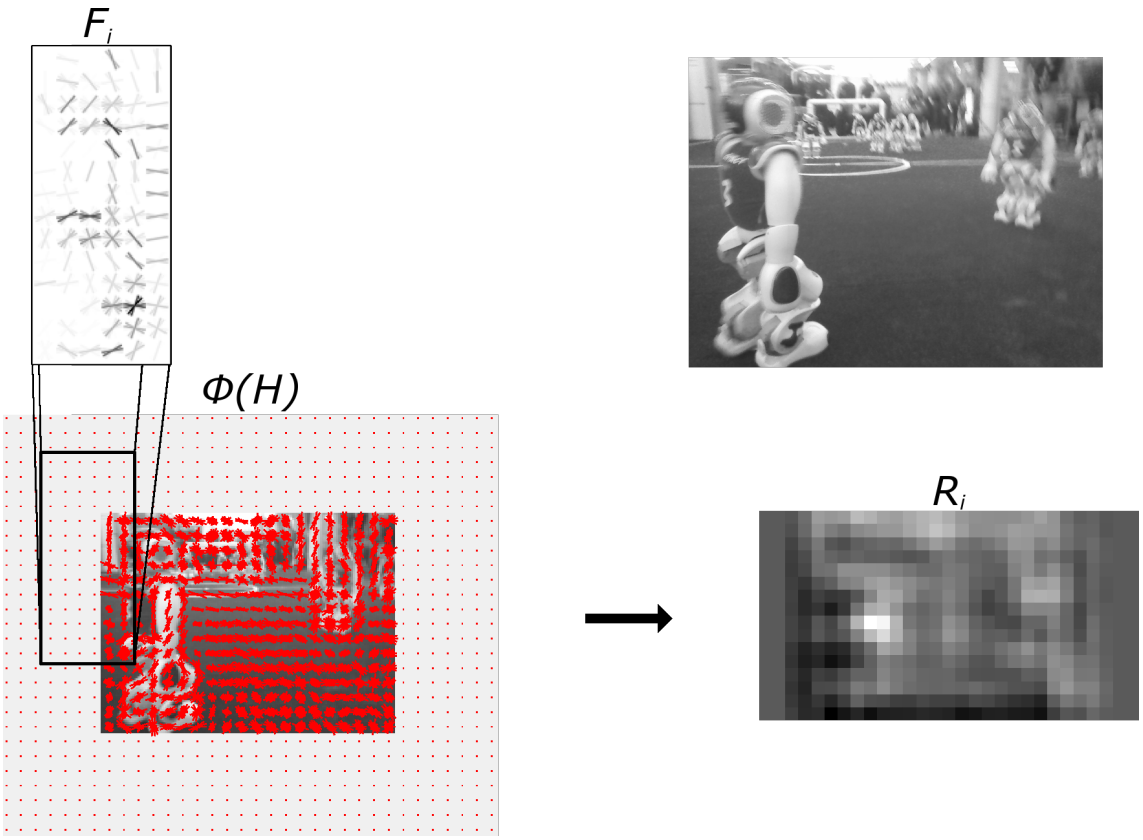


Abbildung 5.6: Beispiel für die Berechnung von *Filter*-Antworten. Der *Filter* F_i wird wie bei einer *Faltung* über die Merkmals-Vektoren $\phi(H)$ geschoben. Der *Hot Spot* befindet sich dabei in der linken oberen Ecke des *Filters*. Da der vordere Roboter in diesem Bild sehr groß ist, ergeben sich erst auf höheren *Leveln* hohe *Filter*-Antworten. Tiefe *Filter*-Antworten sind in schwarz dargestellt – hohe in weiß. In den Antworten auf der rechten Seite sieht man deutlich, wie die Werte an der Position des vorderen Roboters ausschlagen.

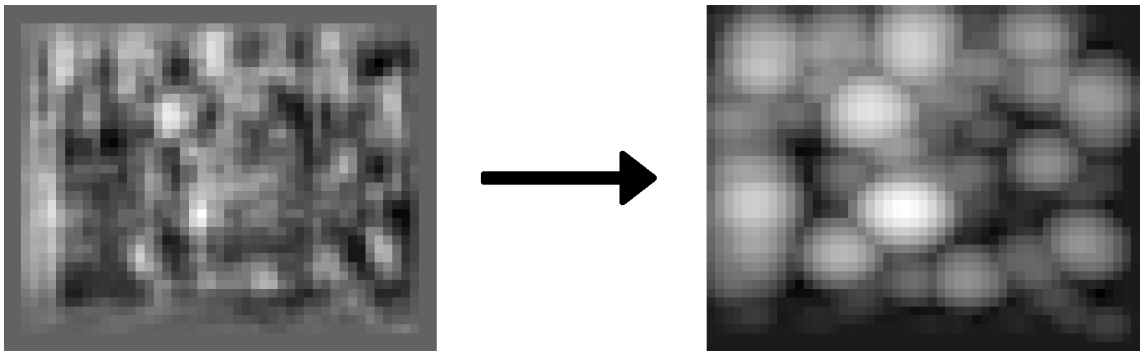


Abbildung 5.7: Beispiel für eine *distance transform*. Wie auf der rechten Seite zu sehen ist, werden hohe Antworten über einen Bereich verteilt und verdrängen dabei niedrigere Antworten.

5.4.3 Bewertungen

Im letzten Schritt werden die einzelnen Antworten aufsummiert, wodurch sich eine Bewertung für die *Wurzel* ergibt:

$$score(x_0, y_0, l_0) = \underbrace{R_{0,l_0}(x_0, y_0)}_{\text{Antwort der Wurzel}} + \underbrace{\sum_{i=1}^{n} D_{i,l_0-\lambda}(2(x_0, y_0) + v_i)}_{\text{Antworten der Teile}} + b, \tag{5.6}$$

wobei λ die Größe der *Oktaven* der *Merkmal-Pyramide* beschreibt und b die Gewichtung des Modells in einem *Mixture-Model* ist (siehe Abbildung 5.8).

Diese Bewertung kann nur auf diese Weise berechnet werden, da die *Filter*-Antworten vorher mit der *distance transform* transformiert wurden, sodass an jeder Position (x, y) in den Antworten D die maximale *Teil*-Bewertung vorhanden ist.

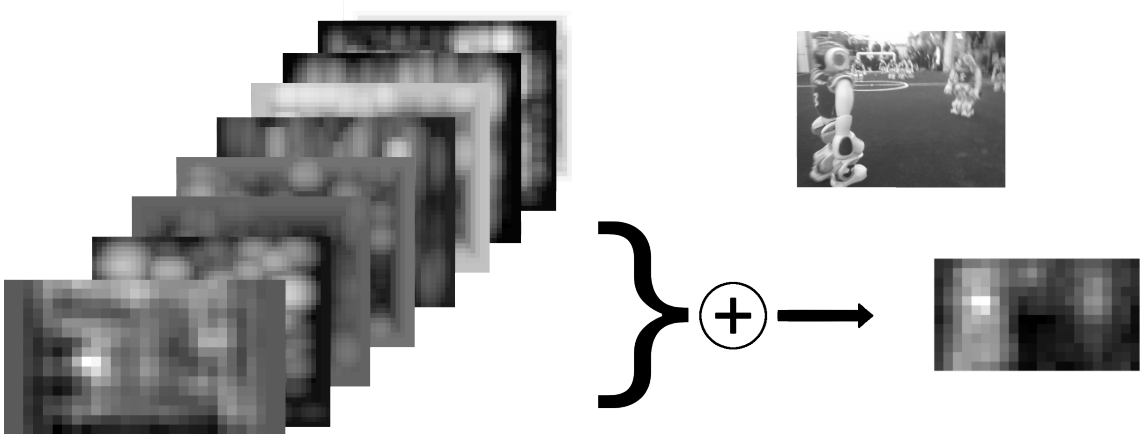


Abbildung 5.8: Berechnung der Bewertungen. Die transformierten *Filter*-Antworten werden nach Formel 5.6 aufgerechnet. Das Eingabe-Bild ist rechts oben dargestellt. In den Bewertungen auf der rechten Seite sieht man deutlich die Ausschläge an den Positionen der Roboter.

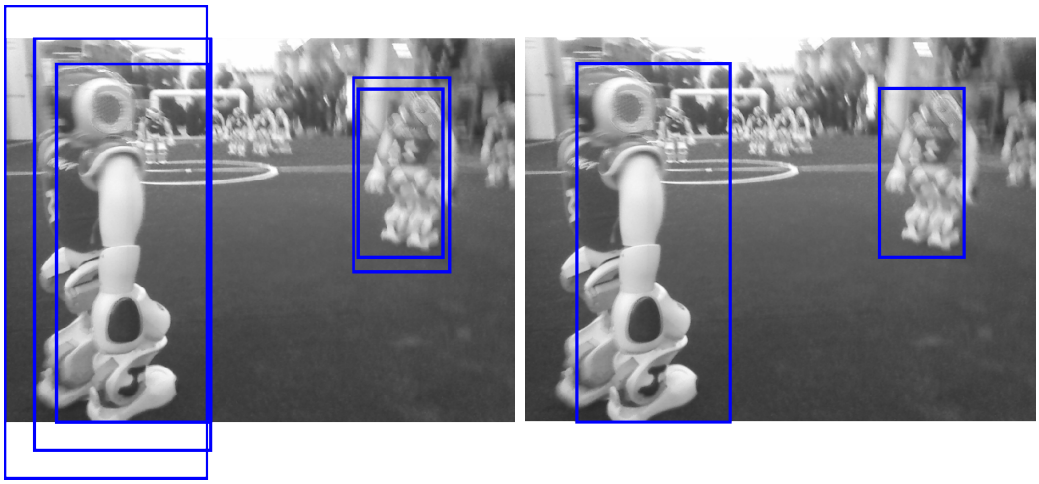
Da diese Bewertungen für jedes Modell auf jedem *Level* erstellt werden, kann es vorkommen, dass mehrere Hypothesen den selben Roboter im Bild beschreiben. Um diese auszusortieren, wird abschließend eine *Nonmax-Suppression* durchgeführt, welche im nächsten Abschnitt erläutert wird.

5.5 Nachbereitung

Da im Matching Bewertungen für *Wurzel*-Positionen berechnet werden, können multiple Objekte in Bildern detektiert werden. Ein Nachteil ist allerdings, dass auch multiple Hypothesen für das selbe Objekt entstehen können (siehe Abbildung 5.9).

Diese multiplen Hypothesen entstehen, weil sich hohe Bewertungen über einen Bereich verteilen können (in Abbildung 5.8 am Ausgabebild zu sehen). Außerdem können mehrere *Level* für einen Roboter hohe Bewertungen erreichen. Das ist beispielsweise dann der Fall, wenn sich der Roboter zwischen zwei *Leveln* befindet.

Diese multiplen Hypothesen müssen aussortiert werden. Schon nach der Berechnung der Bewertungen werden daher in einer 3×3 – Nachbarschaft pro *Level* Hypothesen



(a) Mehrere Detektionen des selben Roboters. (b) Reduzierung auf die beste Detektion.

Abbildung 5.9: Nachbereitung an einem Beispiel-Bild. Bei multiplen Detektionen des selben Objektes wird die mit der höchsten Bewertung ausgewählt.

basierend auf der Bewertung aussortiert. Zusätzlich wird nach Berechnung aller Hypothesen auf allen *Leveln* eine *Nonmax-Suppression* durchgeführt.

Zu Beginn dieser *Nonmax-Suppression* werden Hypothesen nach den Bewertungen sortiert und die *Bounding-Boxes* aus den *Wurzel-Filtern* berechnet. Die Berechnung der *Bounding-Boxes* ist dabei nur eine Skalierung der Dimensionen des *Wurzel-Filters*, passend zum *Level*, auf dem die Hypothese gemacht wurde. Anschließend werden sich überlappende Hypothesen aussortiert, wobei immer die Hypothese mit der höheren Bewertung erhalten bleibt. Bei Berechnung der Überlappung wird eine Variante von *IoU* (siehe Kapitel 4.5.1) verwendet, welche besonders gut ist, um Hypothesen auf verschiedenen *Leveln* auszusortieren. Bei dieser Variante wird das Verhältnis nicht über die Vereinigung gebildet, sondern nur über die *Bounding-Boxes*.

5.6 Auswirkungen der Parameter

Ähnlich wie in der Arbeit von Chai *et al.* [2013] untersuche ich in diesem Kapitel die Auswirkungen von verschiedenen Hyper-Parametern. Die Parameter werden dabei mit Hilfe der *Precision-Recall-Curve* und der *Average Precision* evaluiert (siehe Kapitel 4.5.2).

Um die Parameter zu evaluieren, wird der im folgenden Kapitel 5.7 erläuterte Datensatz in zwei Datensätze aufgeteilt, wobei einer zum Training und der andere zum Testen benutzt wird. Da zu diesem Zeitpunkt die 1494 manuell annotierten Bilder aus Kapitel 5.7 noch nicht existierten, besteht jeder der Datensätze aus 1570 Bildern. Die Evaluation wird mit der Software *FFLD2* [Dubout, 2019] durchgeführt, da hier die Modelle im Fokus stehen und nicht der neu implementierte Detektor.

Die ermittelten Parameter sind dabei nicht auf alle Modelle anwendbar, welche in Kapitel 5.7 trainiert werden. Die Modell-Größe des *Hybrid-Modells* und des *Personen-Modells* kann ich nicht anpassen, da diese bereits vorgegeben sind. Für das *Roboter-Modell* und das *Simulator-Modell* werden die hier ermittelten Modell-Größen als optimal angesehen.

5.6.1 Modell-Größe

Bei dem ersten zu optimierenden Parameter handelt es sich um die Anzahl von Modellen in einem *Mixture-Model* und die Anzahl von *Teilen* pro Modell. Die Anzahl der Modelle wird im folgenden durch M dargestellt, die der *Teile* durch N.

Wie auch schon in Chai *et al.* [2013] beschrieben, sind diese Parameter stark anwendungsspezifisch. Wählt man ein zu großes M, ist es wahrscheinlich, dass viele Modelle die selbe Variation modellieren. Dabei würden unnötig viele Berechnungen gemacht und auch die Zuteilung von Modell zu Detektion wäre unklar, wenn man diese Information in weiteren Schritten weiter verwenden will. Ein zu kleines M würde dazu führen, dass wichtige Variationen nicht modelliert werden, da sie zu einem größeren Modell zusammengefasst wurden.

Wählt man ein zu kleines N, kann das dazu führen, dass durchaus wichtige Merkmale nicht erfasst werden. In bestimmten Fällen könnte das zu falschen Detektionen führen. Ein zu großes N könnte dazu führen, dass unwichtige Dinge modelliert werden. Haben die *Teile* nur eine eingeschränkte Freiheit was die Überlappung angeht, kann es so dazu kommen, dass irrelevante Merkmale modelliert werden.

Ich erwarte dabei, dass eine relativ hohe Anzahl von Modellen nötig ist, um dieses komplexe Problem zu lösen. Im Bezug auf eventuelle zukünftige Weiterentwicklungen, ist das ein Bereich, in dem man Komplexität für weniger Rechenzeit und reduziertem Speicherverbrauch eintauschen könnte.

Abbildung 5.10 zeigt die Ergebnisse der ersten Testreihe.

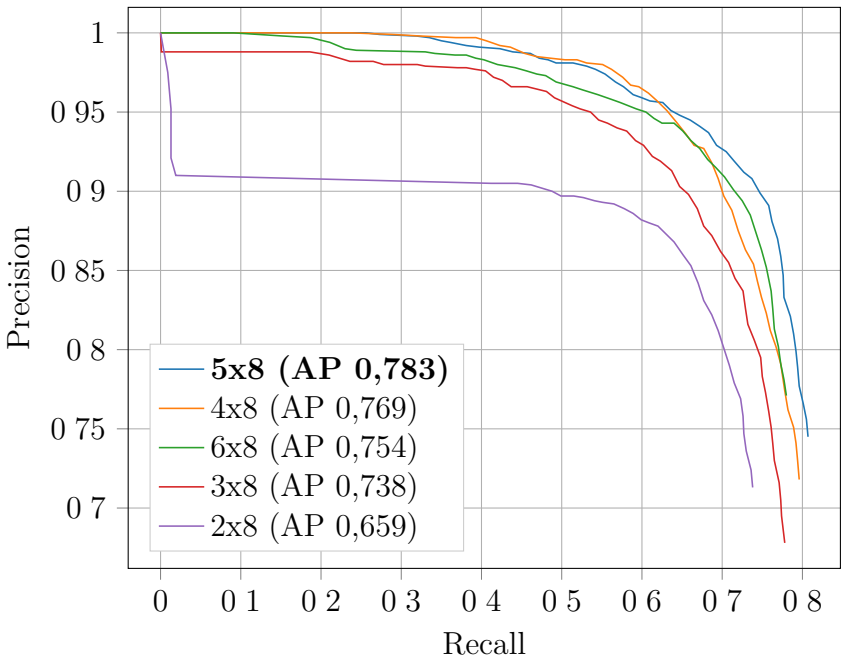


Abbildung 5.10: *Precision-Recall*-Kurve verschiedener Modelle. Die x-Achse beschreibt den *Recall* – die y-Achse die *Precision*. Der Legende sind die getesteten Größen ($M \times N$) und die *Average Precision (AP)* zu entnehmen. Die Modelle wurden bis zu einer Größe von 5×8 immer besser, ehe die Ergebnisse mit einem Modell von einer Größe von 6×8 wieder deutlich schlechter wurden.

Bei dieser Testreihe habe ich nur die Anzahl der Modelle M verändert. Als N wurde ein Ausgangswert von 8 gewählt. Die AP wurden dabei bis zu einer Größe von 5×8 immer besser. 6 Modelle sind anscheinend schon zu viel und die AP sank deutlich von 0,783 auf 0,754. Ein M von 5 wird somit im weiteren Verlauf als optimal angesehen. Da 4 Modelle in dieser Testreihe allerdings auch ein gutes Ergebnis vorweisen konnten, werden auch diese noch weiter evaluiert. In einem weiteren Test wird die Anzahl der *Teile* N auf die selbe Weise überprüft (siehe Abbildung 5.11).

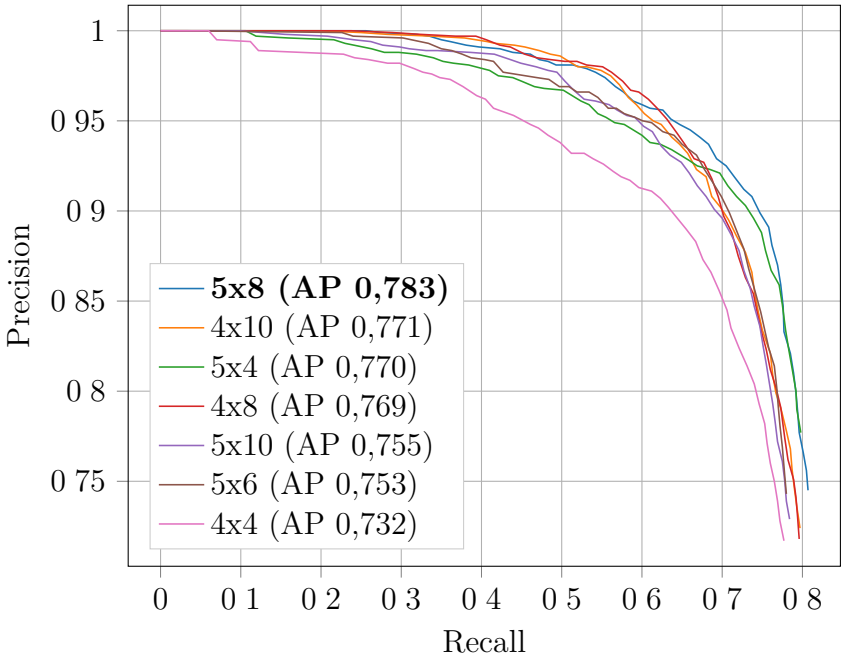


Abbildung 5.11: *Precision-Recall*-Kurve verschiedener Modelle. Die x-Achse beschreibt den *Recall* – die y-Achse die *Precision*. Der Legende sind die getesteten Größen ($M \times N$) und die *Average Precision (AP)* zu entnehmen.

Das Modell mit einer Größe von 5×8 schnitt auch in dieser Testreihe mit einer AP von 0,783 am besten ab – dicht gefolgt von mehreren Modellen mit einer AP von ungefähr 0,77. Wenn die Laufzeit ein großer Faktor wäre, könnte man wohl eines der kleineren Modelle nehmen, ohne große Einbußen in der Erkennungsrate zu haben.

Auf Basis der durchgeführten Tests wird eine Modell-Konfiguration von 5×8 (Modelle $\times$ *Teile*) im weiteren Verlauf der Arbeit als optimal angesehen.

5.6.2 Nonmax-Suppression

Ein weiterer Parameter, der untersucht wird, ist die maximale Überlappung in der *Nonmax-Suppression* (siehe Kapitel 5.5) – hier als O bezeichnet. Der Wertebereich dieses Parameters liegt zwischen 0 und 1. Dieser Parameter bestimmt, ab welcher Überlappung Hypothesen als Hypothesen des selben Objektes angesehen werden. Ein großes O sorgt folglich dafür, dass weniger Hypothesen verworfen werden, dessen *Bounding-Boxes* sich stark überlappen. Ein kleines O hat den gegenteiligen Effekt.

O wird zu Beginn auf 0,5 festgelegt, da dies der Standard ist, welcher von Felzenszwalb *et al.* [2010b] eingeführt wurde. Im Laufe dieser Testreihen habe ich allerdings bemerkt, dass ein kleineres O für diesen Anwendungsfall besser geeignet sein könnte. Ein Test

mit dem Test-Datensatz, dessen Ergebnisse in Tabelle 5.1 einsehbar sind, unterstützt diese Beobachtung, wobei ein O von 0,4 die besten Ergebnisse lieferte, weshalb dieser Wert in Zukunft verwendet wird.

O	0,1	0,2	0,3	**0,4**	0,5
AP	0,775	0,785	0,792	**0,795**	0,783

Tabelle 5.1: *Average Precision (AP)* bei verschiedenen O.

5.7 Training der Modelle

In diesem Kapitel wird erläutert, wie die Modelle trainiert werden, welche in Kapitel 6 evaluiert werden. Nach einem Überblick über die Trainingsdaten-Generierung folgt eine Beschreibung des Trainings der Modelle.

5.7.1 Erzeugung der Trainingsdaten

Da es sich bei den Robotern um ein sehr spezifisches Modell handelt, war die Suche nach Trainingsdaten nicht leicht. *Maschinelles Lernen* steht im Projekt *B-Human* noch an seinen Anfängen und wird erst seit kurzem effektiv vorangetrieben. Daher gab es zum Zeitpunkt dieser Arbeit noch keinen nennenswerten Datensatz mit annotierten Robotern, welcher im Projekt zur Verfügung steht. Teams in anderen Wettbewerben sind den Teams in der *SPL* in diesem Gebiet teilweise weit voraus – auch, weil sie ganz andere Möglichkeiten haben, was die Hardware der Roboter angeht.

Die *bit-bots* aus Hamburg haben eine Plattform namens *ImageTagger* entwickelt, auf der Datensätze erzeugt und ausgetauscht werden können [Fiedler *et al.*, 2018]. Obwohl die *bit-bots* nicht in der *SPL* antreten, wird die Plattform auch von Teams aus der *SPL* genutzt. Ein Teil des Datensatzes, welcher zum Trainieren der Modelle verwendet wurde, stammt von dieser Plattform.

Die Datensätze bestehen dabei zum einen aus den *Annotationen*, welche für jedes Bild die *Bounding-Boxes* der Roboter angeben und den Bildern im *png*-Format. Die *Annotationen* liegen als *xml*-Dateien vor. Ein Datensatz ist damit mit den *VOC*-Datensätzen kompatibel und kann auch von der Software zum Lernen namens *FFLD2* [Dubout, 2019] verwendet werden.

Um Modelle für *NAO*-Roboter zu trainieren, habe ich mehrere Datensätze von der Plattform *ImageTagger* zu einem größeren Datensatz zusammengeführt. Außerdem habe ich 1494 Bilder aus einem Testspiel von *B-Human* mit einem Tool namens *labelme* [Wada, 2019] manuell annotiert. Der resultierende Datensatz besteht aus insgesamt 4634 annotierten Bildern und liegt im *VOC*-Format vor.

Die Bilder enthalten dabei verschiedenste Spielszenen mit Robotern in diversen Posen, welche so auch im echten Spiel vorkommen können. Außerdem wurde Wert darauf gelegt, Bilder mit verschiedenen Beleuchtungs-Bedingungen im Datensatz zu haben. Es handelt sich dabei ausschließlich um Bilder der oberen Kamera mit einer Auflösung von 640×480 Pixeln. Abbildung 5.12 zeigt einige Beispiel-Bilder des verwendeten Datensatzes. Tabelle 5.2 zeigt eine genaue Aufstellung der zum Training verwendeten Daten.

Zusätzlich musste ich einen Datensatz zusammenstellen, um das Modell für simulierte *NAO*-Roboter zu lernen. Die Bilder für dieses Modell habe ich mit dem Tool aus der Arbeit von *Bernd Poppinga* [Poppinga, 2018] generiert. Dabei handelt es sich um 4000 Bilder der oberen Kamera, welche während eines simulierten Spieles annotiert und exportiert wurden.

Klasse	Anzahl
Spielszenen in normaler Beleuchtung	3302
Aufstehende Roboter	562
Spielszenen in schlechter Beleuchtung	770
Insgesamt	4634

Tabelle 5.2: Aufstellung des Datensatzes, welcher zum Trainieren der Modelle verwendet wurde.

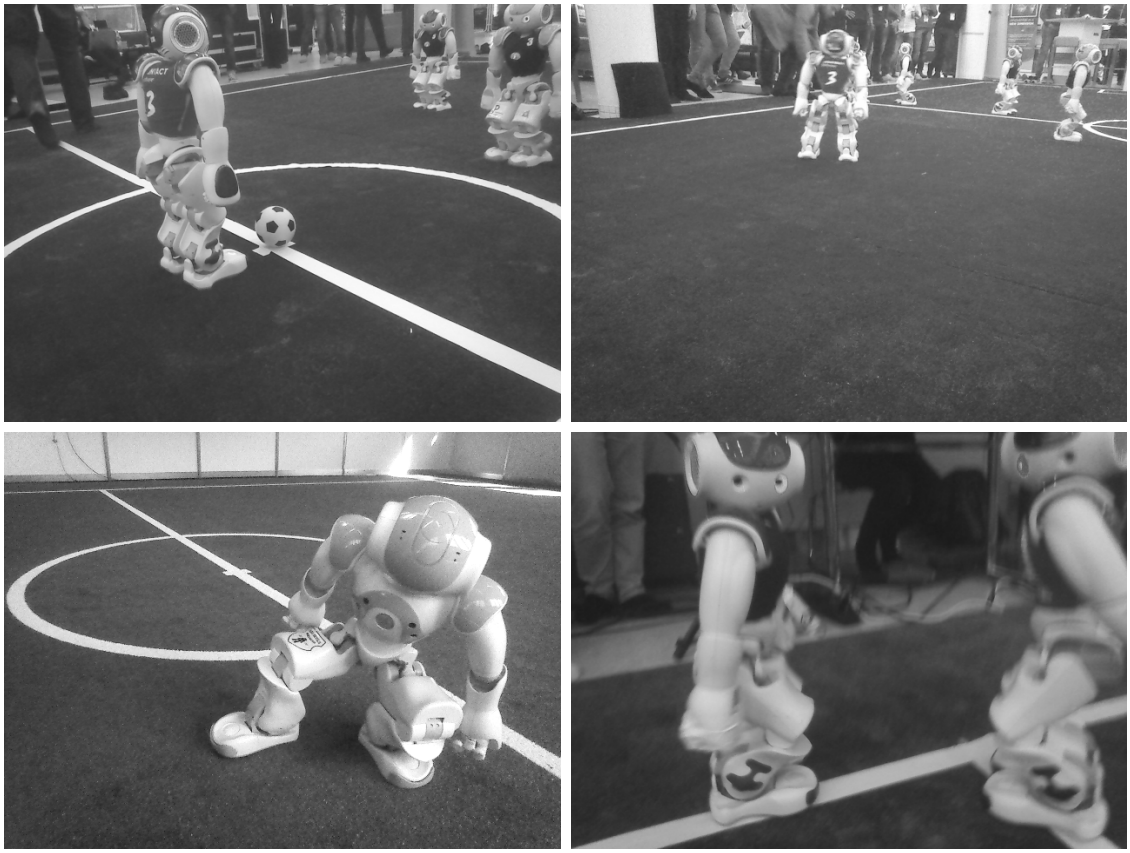


Abbildung 5.12: Beispiel-Bilder aus dem Trainings-Datensatz.

5.7.2 Training

Für die Evaluation werden genau vier Modelle benötigt:

1. Ein Modell, welches auf *NAO*-Roboter trainiert wurde. Dieses Modell wurde auf dem Datensatz der *NAO*-Roboter gelernt (auch: *Roboter-Modell*).

2. Ein Modell, welches auf Personen trainiert wurde. Dieses Modell wurde dem *Release 4* von *P. Felzenszwalb et al.* entnommen [Felzenszwalb *et al.*, 2010b,a]. Dieses hat eine Größe von 3×8 und konnte auf dem *PASCAL VOC 2009* Datensatz eine *AP* von 0,438 bei der Detektion von Personen erreichen (auch: *Personen-Modell*).

3. Ein Hybrid, welcher auf *NAO*-Roboter und Personen trainiert wurde. Dieser wurde trainiert, indem das Modell, welches auf Personen trainiert wurde, mit dem Datensatz der *NAO*-Roboter nachgelernt wurde. In diesem Falle gibt das Ausgangsmodell die Größe vor, welche beim Nachtrainieren nicht mehr geändert werden kann. Daher handelt es sich bei dem Hybriden um ein Modell mit der Größe 3×8 (auch: *Hybrid-Modell*).

4. Ein Modell, welches auf simulierten Bildern von *NAO*-Robotern trainiert wurde. Dieses Modell wurde auf dem Datensatz der simulierten *NAO*-Roboter gelernt (auch: *Simulator-Modell*).

Zur besseren Lesbarkeit werden die Modelle im weiteren Verlauf der Arbeit abgekürzt.

Das Training habe ich mit der Software *ffld2* durchgeführt [Dubout, 2019], welche die Algorithmen von Felzenszwalb *et al.* [2010b] implementiert. Das Training ist ein iterativer Prozess, bei dem die Positionen und Gewichte der *Filter* so lange angepasst werden, bis sie einem Konvergenzkriterium genügen. Das Training kann auch auf Basis eines bereits trainierten Modells stattfinden. In diesem Fall wird die Konfiguration (*Filter*-Positionen und Anzahl) übernommen und nur die Gewichte werden angepasst. Die resultierenden *Mixture-Models* sind achsensymmetrisch, was heißt, dass bei einer definierten Modell-Anzahl von 5 das resultierende *Mixture-Model* 10 Modelle enthält. Abbildung 5.13 zeigt einige der gelernten Modelle.

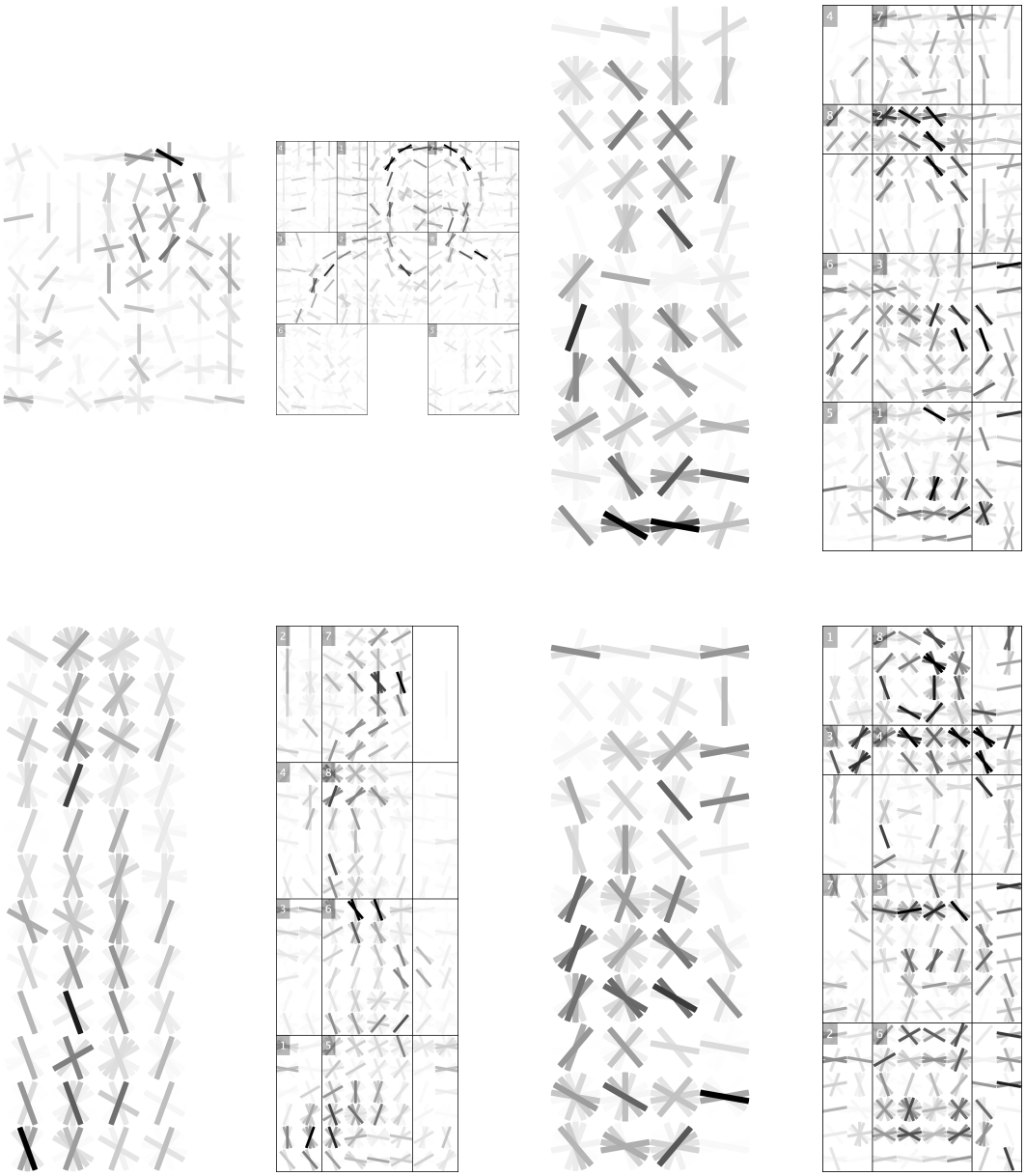


Abbildung 5.13: Die Abbildung zeigt vier der gelernten Modelle. Von oben links nach unten rechts: *Personen-Modell*, *Roboter-Modell*, *Simulator-Modell*, *Hybrid-Modell*. Es sind jeweils eine *Wurzel* (linke Seite) und die zugehörigen *Teile* abgebildet. Die *Teile* werden durch schwarze Boxen repräsentiert. Die *deformation costs* sind hier nicht abgebildet.

6. Evaluation

In diesem Kapitel werden die Modelle evaluiert, welche in Kapitel 5.7 trainiert wurden. Die Evaluation untersucht die folgenden Aspekte:

- **Erkennungsrate bei verschiedenen Beleuchtungen:** Da *Deformable Part Models* ein gradientenbasiertes Verfahren sind, könnte dieses weniger anfällig für Änderungen in der Beleuchtung einer Szene sein. Diese Annahme soll in der Evaluation untersucht werden.
- **Erkennungsrate bei verdeckten Robotern:** Verdeckte Roboter stellen bei der aktuellen Implementierung im *B-Human*-Framework ein Problem dar. Die Erkennungsrate in diesem Fall ist also auch besonders interessant.
- **Erkennungsrate bei nebeneinander stehenden Robotern:** Wie bei dem vorherigen Punkt ist auch dieser Fall ein Extremfall und muss daher evaluiert werden.
- **Erkennungsrate auf verschiedenen Entfernungen:** *Deformable Part Models* sind darauf ausgelegt, Objekte auf diverse Entfernungen zu detektieren. Diese Eigenschaft ist in diesem Anwendungsbereich besonders nützlich und sollte daher überprüft werden.

Grundlage für die Bewertung sind die *Precision*, der *Recall* und die *Average Precision* (siehe Kapitel 4.5).

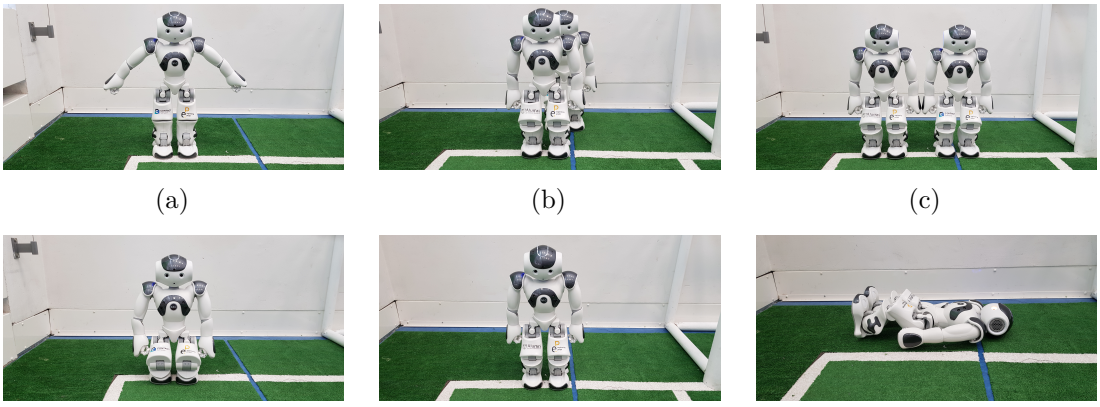
6.1 Allgemeine Erkennungsrate

Dieses Experiment überprüft die Erkennungsrate bei verschiedenen Entfernungen und Posen. Hier wird insbesondere überprüft, wie robust die gelernten Modelle gegen die Verschiebung verschiedener Körperteile und diversen Rotationen in Relation zur Kamera sind.

Diese Versuchsreihe wird auf den in Kapitel 5.7 gelernten Modellen durchgeführt. Die aktuell im *B-Human*-Framework genutzte Implementierung (im Folgenden wird diese als "*B-Human*-Detektor" abgekürzt) dient als Referenzwert.

6.1.1 Versuchs-Aufbau

Dieses Experiment enthält einen detektierenden Roboter und bis zu zwei Roboter, die detektiert werden sollen. Die Roboter wurden dabei in verschiedenen Posen (siehe Abbildung 6.1) auf dem Feld platziert (siehe Abbildung 6.2).



(a) (b) (c)

(d) (e) (f)

Abbildung 6.1: Alle möglichen Posen, welche im ersten Experiment auf verschiedenen Distanzen evaluiert wurden. Pose *a* soll untersuchen, welche Erkennungsrate bei einer vergrößerten Körperfläche und ausgestreckten Armen erreicht wird. Außerdem steht der Roboter durch die durchgedrückten Knie höher als normal. Pose *b* stellt einen Extremfall dar, bei dem zwei Roboter voreinander stehen. Hier sollte wenigstens der vordere Roboter zuverlässig detektiert werden. Der hintere Roboter testet die Detektion von Robotern mit verdeckten *Teilen*. Pose *c* stellt den Fall dar, bei dem zwei Roboter nebeneinander stehen. Hier sollen beide Roboter detektiert werden. Pose *d* beschreibt einen sitzenden Roboter. Diese Pose könnte ein Roboter beispielsweise während einer Aufsteh-Bewegung einnehmen. Pose *e* beschreibt einen normal stehenden Roboter und Pose *f* einen liegenden Roboter.

Zusätzlich wurden folgende Aspekte in den Durchgängen variiert.

1. **Entfernung zum erkennenden Roboter:** Das Experiment wurde mehrmals mit den Entfernungen 0,3 m, 1,1 m, 1,9 m, 2,7 m und 3,5 m durchgeführt.

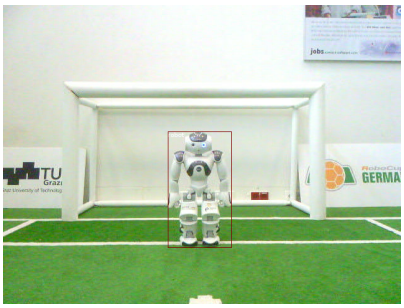
2. **Rotation zum erkennenden Roboter:** Bei den Durchgängen, wo sich nur ein einzelner Roboter stehend im Bild befindet (siehe Abbildung 6.1e), wurde dessen Rotation in Relation zum erkennenden Roboter variiert. Insgesamt wurden vier Rotationen im Bereich von 0 bis 270 Grad evaluiert.

Es wurde ein Datensatz mit insgesamt 5069 Bildern erzeugt, welcher Roboter in den oben beschriebenen Posen auf allen erwähnten Entfernungen enthält. Es handelt sich dabei nur um Bilder der oberen Kamera bei einer Auflösung von 640×480 Pixeln. Die Roboter wurden in allen Bildern mit *Bounding-Boxes* annotiert.

6.1.2 Durchführung und Ergebnisse

Im Zuge des Experiments wurde der in dieser Arbeit vorgestellte Detektor mit den Modellen aus Kapitel 5.7 auf dem erzeugten Datensatz angewandt. Außerdem wurde

(a) Beispiel für zwei Roboter mit einer Entfernung von 1,9 m. Die Bilder für den Versuch stammen vom Roboter auf der linken Seite.



(b) Roboter mit annotierter *Bounding-Box* (rot).

Abbildung 6.2: Aufbau des Experimentes zur Ermittlung der allgemeinen Erkennungsrate.

ein weiterer Durchlauf mit dem momentan im *B-Human*-Framework verwendeten Detektor durchgeführt. Dabei wurde für jeden Durchlauf die AP berechnet. Die *Precision*, der *Recall* und die durchschnittliche IoU wurde über alle Posen ermittelt.

Für die in dieser Arbeit vorgestellten Modelle wurde ein IoU von 50% gewählt. Der *B-Human*-Detektor wurde mit einer IoU von 30% evaluiert.

Der *B-Human*-Detektor lieferte grundsätzlich gute Ergebnisse (siehe Tabelle 6.1). Die Reduzierung auf Fußpunkte führte allerdings wie erwartet zu *Bounding-Boxes*, welche nicht den ganzen Roboter abdecken (siehe Abbildung 6.3 zum Vergleich der *Bounding-Boxes*). Besonders gut sichtbar war das bei den Durchgängen mit der Pose *a*. Die ausgestreckten Arme sorgten dabei dafür, dass keine der Detektionen akzeptiert wurde. Insgesamt zeigt der *B-Human*-Detektor eine circa 30% schlechtere IoU, als der hier vorgestellte Ansatz, was aber so auch zu erwarten war.

Beim Betrachten der durchschnittlichen AP fällt auf, dass die Stärke des *B-Human*-Detektors bei relativ nahen Robotern liegt. So erzielte er bei einer Entfernung von 1,1 Metern gemittelt über alle Posen in seiner Versuchsreihe die beste AP, zeigte aber schlechtere Ergebnisse für die Entfernungen 1,9 und 2,7 Meter, ehe auf 3,5 Metern gar keine richtige Detektion mehr vorlag. Dieser Detektor ist der einzige Ansatz, der auf einer Entfernung von 0,3 Metern gute Ergebnisse vorweisen konnte.

Pose	0,3 m	1,1 m	1,9 m	2,7 m	3,5 m	Precision	Recall	IoU
a	0	0	0	0	0	0	0	0
b	0,545	0,324	0,454	0,409	0	0,929	0,353	0,47
c	0,545	0,893	0,725	0,449	0	0,98	0,494	0,45
d	1,0	0,909	0,902	0,909	0	0,994	0,717	0,57
e	0,493	0,97	0,89	0,53	0	0,9	0,615	0,56
Average	0,516	0,619	0,594	0,459	0	0,76	0,435	0,41

Tabelle 6.1: Ergebnisse für Experiment 1 für den *B-Human*-Detektor.

Das *Hybrid-Modell* konnte in diesem Experiment mit rund 2% höherer *Precision* und *Recall* das *Roboter-Modell* überbieten (siehe Tabelle 6.2). Die über alle Posen

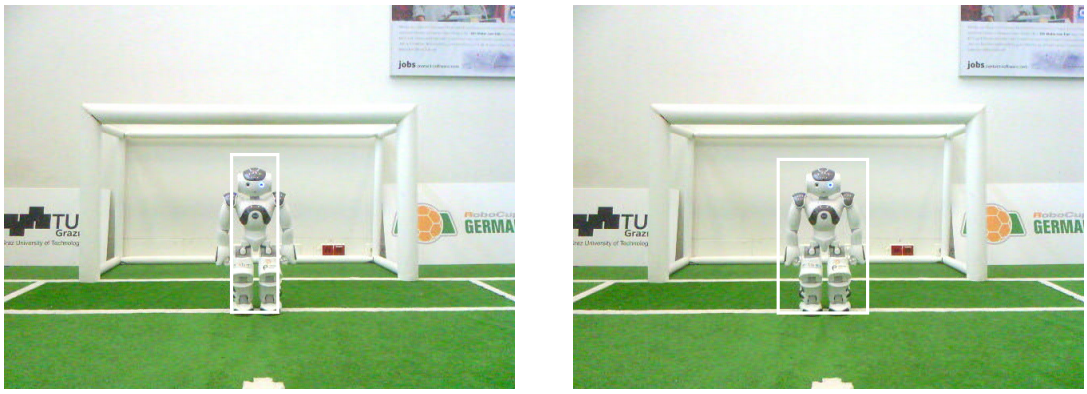


Abbildung 6.3: Die Abbildung zeigt eine Detektion des momentan verwendeten Detektors (links) und eine Detektion des in dieser Arbeit vorgestellten Detektors (rechts). Es wurde das *Hybrid-Modell* verwendet.

gemittelte *IoU* befindet sich mit 70% weit über der gesetzten Schranke von 50% für akzeptierte Detektionen. In Pose *a* lieferte das *Hybrid-Modell* in seiner Versuchsreihe die schlechteste *IoU* mit rund 58%. Das ist darauf zurückzuführen, dass bei der aktuellen *Bounding-Box*-Berechnung keine Teil-Positionen mit einbezogen werden. Auffallend ist hier, dass bei den Versuchen mit Pose *b* durchgängig nur der vordere Roboter detektiert wurde – dies aber mit perfekter *Precision*. Auch nebeneinander stehende Roboter (Pose *c*) lieferten eine perfekte *Precision*, wobei hier in den meisten Fällen beide Roboter detektiert wurden, was sich in einem hohen *Recall* niederschlägt. Bei sitzenden Robotern (Pose *d*) zeigte das Modell kleinere Probleme auf einer Entfernung von 1,1 Metern. Auf größerer Entfernung wurden die Ergebnisse zunehmend besser. Bei stehenden Robotern (Pose *e*) zeigte das Modell die besten Ergebnisse auf einer Entfernung von 1,9 Metern. Insgesamt konnte das *Hybrid-Modell* in diesem Experiment den momentan in *B-Human* verwendeten Ansatz bei der *Precision* um 14%, bei dem *Recall* um 20% und beim *IoU* um circa 30% überbieten.

Pose	0,3 m	1,1 m	1,9 m	2,7 m	3,5 m	Precision	Recall	IoU
a	0	0	0,709	0,883	0,415	0,603	0,448	0,58
b	0	0,545	0,545	0,545	0,545	1,0	0,423	0,72
c	0	0,909	1,0	1,0	1,0	1,0	0,795	0,77
d	0	0,434	0,818	0,948	1,0	0,946	0,72	0,71
e	0	0,945	1,0	0,961	0,87	0,969	0,797	0,73
Average	0	0,566	0,814	0,867	0,766	0,903	0,636	0,7

Tabelle 6.2: Ergebnisse für Experiment 1 für das Hybrid-Modell.

Das *Roboter-Modell* lieferte ähnliche Ergebnisse im Vergleich zum *Hybrid-Modell* (siehe Tabelle 6.3). Insgesamt zeigte dieses allerdings eine 2% bessere *IoU* und lieferte ab einer Entfernung von 1,9 Metern für Pose *a* bessere Ergebnisse, als das *Hybrid-Modell*. Die gemittelten *Precision*- und *Recall*-Ergebnisse liegen allerdings jeweils rund 2% unter denen des *Hybrid-Modells*. Insgesamt konnte aber auch dieses Modell den *B-Human*-Detektor um rund 12% und 18% überbieten.

Pose	0,3 m	1,1 m	1,9 m	2,7 m	3,5 m	Precision	Recall	IoU
a	0	0	1,0	1,0	1,0	0,728	0,558	0,55
b	0	0,545	0,545	0,545	0,542	0,999	0,422	0,8
c	0	0,545	1,0	1,0	0,863	0,969	0,712	0,76
d	0	0,409	1,0	1,0	1,0	1,0	0,712	0,75
e	0	0,918	0,898	0,443	0,736	0,751	0,678	0,76
Average	0	0,483	0,886	0,797	0,828	0,889	0,616	0,72

Tabelle 6.3: Ergebnisse für Experiment 1 für das Roboter-Modell.

Das *Simulator-Modell* lieferte durchgängig schlechtere Ergebnisse, als das *Hybrid-Modell* und das *Roboter-Modell* (siehe Tabelle 6.4). Die über alle Posen gemittelte *IoU* ist rund 10% schlechter, als die des *Roboter-Modells* und überbietet dabei noch die des *B-Human*-Detektors um rund 20%. Die *Precision* (rund 30% schlechter) und der *Recall* (rund 6% schlechter) können aber nicht mehr mit dem *B-Human*-Detektor mithalten. Die besten Ergebnisse lieferte dieses Modell auf einer Entfernung von 3,5 Metern.

Pose	0,3 m	1,1 m	1,9 m	2,7 m	3,5 m	Precision	Recall	IoU
a	0	0	0	0	0,056	0,039	0,019	0,51
b	0	0,006	0,563	0,235	0,323	0,422	0,360	0,59
c	0,075	0	0,924	0,738	0,738	0,535	0,72	0,68
d	0	0	0	0	0,971	0,57	0,258	0,66
e	0,058	0,231	0,356	0,722	0,859	0,552	0,537	0,69
Average	0,026	0,047	0,368	0,339	0,589	0,423	0,378	0,626

Tabelle 6.4: Ergebnisse für Experiment 1 für das Simulator-Modell.

Das *Personen-Modell* lieferte die schlechtesten Ergebnisse in diesem Experiment (siehe Tabelle 6.5). Dieses konnte den *B-Human*-Detektor nur in der gemittelten *IoU* um 10% überbieten. *Precision* und *Recall* sind deutlich schlechter. Die schlechten Ergebnisse könnten auf den Datensatz zurückzuführen sein, der zum Trainieren des Modells verwendet wurde, welcher auf ganz andere Anwendungsbereiche zugeschnitten wurde.

Pose	0,3 m	1,1 m	1,9 m	2,7 m	3,5 m	Precision	Recall	IoU
a	0	0	0	0	0	0	0	0
b	0	0,065	0,026	0,045	0,022	0,136	0,034	0,65
c	0,202	0	0	0,159	0,068	0,356	0,127	0,63
d	0	0	0,124	0,014	0,090	0,092	0,058	0,6
e	0	0,368	0,337	0,393	0,526	0,45	0,442	0,67
Average	0,04	0,086	0,097	0,122	0,141	0,206	0,132	0,51

Tabelle 6.5: Ergebnisse für Experiment 1 für das Personen-Modell.

Keines der Modelle konnte in Verbindung mit dem in dieser Arbeit vorgestellten Detektor liegende Roboter detektieren (siehe Tabelle 6.6). Das ist darauf zurückzuführen, dass der Trainings-Datensatz nur sehr wenige Bilder von liegenden Robotern

enthält. Die *Deformable Part Models* sind nicht invariant hinsichtlich der Rotation der *Teile*, was dazu führt, dass die Detektion von liegenden Robotern über ein Modell stattfinden muss, welches extra auf diesen Extremfall ausgelegt ist. Der *B-Human*-Detektor konnte nur auf 1,1 Metern und 2,7 Metern einige Detektionen vorweisen. Da dieser Extremfall weder vom *B-Human*-Detektor noch von den im Verlauf dieser Arbeit gelernten Modellen modelliert wird, wird er gesondert behandelt.

Modell	0,3 m	1,1 m	1,9 m	2,7 m	3,5 m	Precision	Recall	IoU
Hybrid	0	0	0	0	0	0	0	0
Roboter	0	0	0	0	0	0	0	0
Simulator	0	0	0	0	0	0	0	0
Personen	0	0	0	0	0	0	0	0
B-Human	0	0,183	0	0,049	0	0,294	0,036	0,48

Tabelle 6.6: Ergebnisse für liegende Roboter (Pose f).

6.2 Erkennungsrate bei verschiedenen Beleuchtungs-Bedingungen

Da *Deformable Part Models* ein gradientenbasiertes Verfahren sind, erwarte ich, dass dieses zumindest gegen geringe Änderungen in der Beleuchtung robust ist. Das gilt allerdings nur soweit, wie die grundlegende Struktur der Roboter in den Deskriptoren erhalten bleibt. Bei stark überbelichteten Bildern erwarte ich einen Einbruch in der Erkennungsrate. Dieser Versuch wurde nur mit dem in dieser Arbeit vorgestellten Detektor durchgeführt.

6.2.1 Versuchs-Aufbau

Für dieses Experiment habe ich einen Datensatz des *SPL*-Teams *NaoDevils* von *ImageTagger* verwendet. Dieser Datensatz enthält insgesamt 2614 Bilder der oberen Kamera mit einer Auflösung von 640×480 Pixeln. Die Bilder enthalten einen *NAO*-Roboter in verschiedenen Posen und Entfernungen zur Kamera. Zusätzlich variieren die Beleuchtungs-Bedingungen. Einige Bilder wurden bei normaler Beleuchtung aufgenommen, einige unter Einfluss von Sonnenlicht und einige bei gedimmten Licht. Der Roboter wurde in jedem Bild mit einer *Bounding-Box* annotiert. Abbildung 6.4 zeigt einige Bilder aus dem verwendeten Datensatz.

Abbildung 6.4: Beispiel-Bilder aus dem Evaluations-Datensatz. Die Bedingungen von links nach rechts: Einfluss von Sonnenlicht; gedimmtes Licht; Normale Beleuchtungs-Bedingungen.

6.2.2 Durchführung und Ergebnisse

Für diesen Versuch wurde nur der in dieser Arbeit vorgestellte Detektor mit allen Modellen aus Kapitel 5.7 verwendet. Wie im vorherigen Versuch auch, wurden die *AP*, die *Precision* und der *Recall* für jeden Durchlauf notiert.

Die Ergebnisse für die Bilder, welche unter Einfluss von Sonnenlicht aufgenommen wurden, sind erwartungsgemäß schlecht (siehe Tabelle 6.7). Nur das *Hybrid-* und das *Roboter-Modell* konnten einige richtige Detektionen vorweisen. Die allgemein schlechte Performance schließe ich darauf zurück, dass bei so starker Beleuchtung viele Bildbereiche auf den Maximalwert gesetzt werden, wodurch auf bestimmten Bildbereichen jegliche Gradienten verloren gehen (siehe Abbildung 6.5). Dadurch fehlen viele Details auf den Körperteilen, welche aber in den Modellen mit modelliert wurden. Verringert man den *Schwellwert* für die Detektionen, tauchen einige Falsch-Erkennungen auf und die Anzahl der Detektionen für den Roboter nehmen nicht spürbar zu.

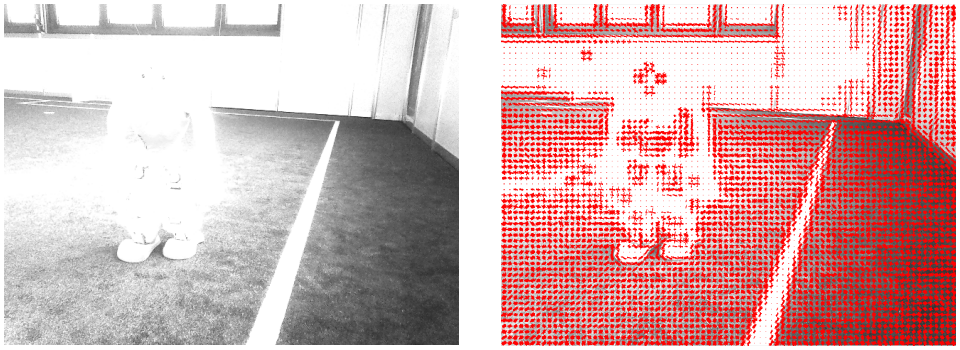


Abbildung 6.5: Die Abbildung zeigt ein stark überbelichtetes Bild (links) und einen zugehörigen *HoG*-Deskriptor (rechts). Die Normalisierung in den *HoG*-Deskriptoren sorgt dafür, dass einige Gradienten trotz starker Überbelichtung erhalten bleiben. Trotzdem fehlen insbesondere viele Details. Einige Teile, wie die Arme oder der Kopf, sind außerdem nicht vollständig. Die Kombination von fehlenden Teilen und Details sorgt dafür, dass die Detektion fehlschlägt.

Modell	AP	Precision	Recall	IoU
Hybrid	0,024	0,166	0,013	0,57
Roboter	0,018	0,201	0,014	0,72
Simulator	0	0	0	0
Personen	0	0	0	0

Tabelle 6.7: Ergebnisse für die Bilder, welche bei Einfluss von Sonnenlicht aufgenommen wurden. Die Tabelle enthält die *AP*, die durchschnittliche *Precision*, *Recall* und die *IoU* für alle Modelle.

Die Ergebnisse für die Bilder, welche bei gedimmter Beleuchtung aufgenommen wurden, sehen sehr vielversprechend aus (siehe Tabelle 6.8). Das *Hybrid-Modell* erreichte die insgesamt besten Ergebnisse bei einer *Precision* von 99,8% und einem *Recall* von 94,1%. Der *IoU* liegt mit 79% weit über der gesetzten Schranke von 50% für akzeptierte Detektionen. Da sich in diesem Datensatz auch Bilder von Robotern befinden, welche weiter als die vorher evaluierten 3,5 Meter entfernt

sind, verstärkt dies die Vermutung, dass die Modelle auf größere Entfernungen ausgelegt sind. Das *Roboter-Modell* lieferte im Vergleich zum *Hybrid-Modell* eine leicht schlechtere *Precision* (rund 4%) und einen ebenfalls schlechteren *Recall* (rund 5%). Das *Simulator-* und das *Personen-Modell* lieferten auch in diesem Durchlauf die schlechtesten Ergebnisse, wobei das *Simulator-Modell* noch eine weit schlechtere *AP* (rund 17% schlechter) und *Precision* (rund 50% schlechter) aufwies.

Modell	AP	Precision	Recall	IoU
Hybrid	0,948	0,998	0,941	0,79
Roboter	0,861	0,957	0,889	0,76
Simulator	0,084	0,181	0,167	0,64
Personen	0,253	0,682	0,277	0,63

Tabelle 6.8: Ergebnisse für die Bilder, welche bei gedimmten Licht aufgenommen wurden. Die Tabelle enthält die *AP*, die durchschnittliche *Precision*, *Recall* und die *IoU* für alle Modelle.

Die Ergebnisse auf Bildern unter normaler Beleuchtung sind ebenfalls gut (siehe Tabelle 6.9). Das *Hybrid-Modell* und das *Roboter-Modell* liefern eine 3%, bzw. 2% schlechtere *IoU*, als auf den Bildern bei gedimmten Licht. Das *Hybrid-Modell* verschlechterte sich um rund 20% im *Recall*, was auch zu einem deutlich niedrigeren *AP* von 0,798 führt. Das *Roboter-Modell* hingegen lieferte einen besseren *Recall* und damit auch eine bessere *AP*, als im vorherigen Versuch. Die *Precision* und der *Recall* des *Simulator-Modells* verbesserten sich deutlich um rund 24%, bzw. 9%. Das *Personen-Modell* zeigte ebenfalls einen um rund 14% besseren *Recall*, was sich in einer um 4% besseren *AP* niederschlug.

Modell	AP	Precision	Recall	IoU
Hybrid	0,798	0,983	0,762	0,76
Roboter	0,899	0,946	0,971	0,74
Simulator	0,159	0,425	0,237	0,70
Personen	0,294	0,616	0,416	0,63

Tabelle 6.9: Ergebnisse für die Bilder, welche bei normalen Beleuchtungs-Bedingungen aufgenommen wurden.

6.3 Erkennungsrate im Spiel

Dieser Versuch untersucht die Erkennungsrate während eines Spieles. In einem Spiel ergeben sich diverse Situationen, welche nicht durch die vorherigen Tests abgedeckt wurden. Außerdem befinden sich Roboter auf verschiedensten Entfernungen und in diversen Posen in dem verwendeten Datensatz. Dieser Versuch wird auf den neu gelernten Modellen und dem aktuell in *B-Human* verwendeten Detektor durchgeführt.

6.3.1 Versuchs-Aufbau

Für diesen Versuch habe ich einen Datensatz aus einem Testspiel von *B-Human* annotiert. Insgesamt handelt es sich dabei um 774 Bilder der oberen Kamera. Diese Bilder wurden über einen Zeitraum von 10 Minuten aufgenommen. Das Testspiel fand

auf einem großen Feld statt, was heißt, dass sich auch sehr weit entfernte Roboter in diesem Datensatz befinden. In diesem Versuch wurde, wie in den vorherigen Versuchen auch, die *AP*, die *Precision*, der *Recall* und die *IoU* berechnet.

6.3.2 Durchführung und Ergebnisse

Die Ergebnisse dieses Experimentes sind Tabelle 6.10 zu entnehmen.

Modell	AP	Precision	Recall	IoU
Hybrid	0,348	0,946	0,354	0,74
Roboter	0,371	0,927	0,401	0,75
Simulator	0,057	0,168	0,263	0,67
Personen	0,048	0,129	0,122	0,63
B-Human	0,232	0,737	0,278	0,48

Tabelle 6.10: Ergebnisse auf Bildern aus einem Testspiel von *B-Human*. Dargestellt sind die *AP*, die *Precision*, der *Recall* und der *IoU* gemittelt über alle Bilder für alle Modelle und den aktuell in *B-Human* verwendeten Detektor.

Das *Roboter-Modell* lieferte in diesem Experiment insgesamt die besten Ergebnisse – dicht gefolgt vom *Hybrid-Modell*. Diese beiden Modelle heben sich vor allem durch ihre hohe *Precision* ab, wobei beide den *B-Human*-Detektor jeweils um rund 20% überboten. Bei der *IoU* lieferten diese beiden Modelle das beste Ergebnis, wobei diese mit 74%, bzw. 75% weit über der gesetzten Grenze von 50% liegen. Den besten *Recall* lieferte das *Roboter-Modell*, wobei dieses mit seinen 40% rund 13% mehr Roboter detektieren konnte, als der *B-Human*-Detektor. Der *B-Human*-Detektor lieferte eine erwartungsgemäß schlechte *IoU* und detektierte rund 27% aller Roboter in den Bildern. In der *AP* konnte der *B-Human*-Detektor das *Simulator-Modell* und das *Personen-Modell* um jeweils rund 20% überbieten. Das *Simulator*- und das *Personen-Modell* lieferten auch in diesem Experiment die schlechtesten Ergebnisse. Das *Simulator-Modell* kann bei dem *Recall* noch mit dem *B-Human*-Detektor mithalten. Die *Precision* ist aber mit rund 16% um einiges schlechter.

6.4 Weiterführende Untersuchungen

In diesem Abschnitt werden Beobachtungen aus den vorangegangenen Versuchen ausgewertet.

6.4.1 Fehler in den Detektionen

Bei der Durchführung vom ersten Experiment ist aufgefallen, dass die vorhergesagte *Bounding-Box* bei dem *B-Human*-Detektor in der Größe und der Positionierung sehr stark variiert. Die *Bounding-Boxes* der in Kapitel 5.7 trainierten Modelle in Verbindung mit dem neuen Detektor scheinen hingegen sehr konsistent zu sein. Um diese Beobachtung zu untersuchen, habe ich während des Versuches auf einer Entfernung von 1,9 Metern mit der Pose *e* die Vorhersagen und die annotierten *Bounding-Boxes* jeweils auf das Feld projiziert und einen Fehler zwischen diesen berechnet.

Der *B-Human*-Detektor weist dabei eine leichte Tendenz zur linken Seite auf (siehe Abbildung 6.6). Auf der x-Achse hält sich der Fehler fast durchgängig unter 20 Millimeter, wobei auf der y-Achse größere Fehler sichtbar werden. Diese sind vermutlich auf das Rauschen zurückzuführen, wobei die Wahl der Startpunkte an den Füßen inkonsistent wird.

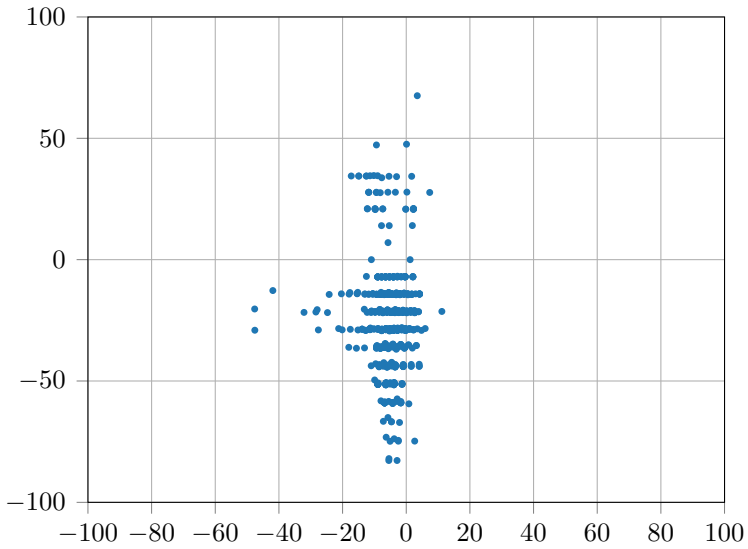


Abbildung 6.6: Die Abbildung zeigt den Fehler der Vorhersagen des *B-Human*-Detektors (in blau) in Relation zur tatsächlichen Position des Roboters an Koordinate $(0, 0)$. Einheiten in Millimeter.

Wenn man den Fehler des *Hybrid-Modells* darstellt, wird ein gewisses Muster sichtbar (siehe Abbildung 6.7). Die Fehler scheinen sich an bestimmten Positionen zu sammeln. Wenn man während des Versuches die *Level* beobachtet, auf denen die höchste Bewertung erreicht wurde, dann fällt auf, dass diese mit dem Fehler korrelieren. In diesem konkreten Fall erreichen *Level* 10 und 11 der *Merkmal-Pyramide* durchgängig hohe Bewertungen an der Position des Roboters im Bild. Die meiste Zeit ist die Bewertung auf *Level* 10 am höchsten und nur hin und wieder wird diese von *Level* 11 überboten. Da die beiden *Level* auf verschieden skalierten Versionen des Original-Bildes basieren, variiert dementsprechend die *Bounding-Box* in der Größe, was zu einem Fehler auf der y-Achse führt.

Einen grundsätzlichen Fehler auf beiden Achsen führe ich auf die Verwendung einer *Merkmal-Pyramide* zurück. Dadurch, dass auf den verschiedenen *Leveln* der *Merkmal-Pyramide* auf skalierten Versionen des Eingabe-Bildes gearbeitet wird, geht Genauigkeit bei der letztendlichen Lokalisierung des Objektes im Bild verloren. Ein Pixel auf dem *Level* mit der halben Auflösung des Original-Bildes deckt beispielsweise einen Bereich von 2×2 Pixeln auf dem voll aufgelösten Bild ab. In diesem Bereich von 2×2 Pixeln ist eine genaue Lokalisierung somit unmöglich. Diese Situation verschlimmert sich, je weiter man die *Merkmal-Pyramide* durchläuft.

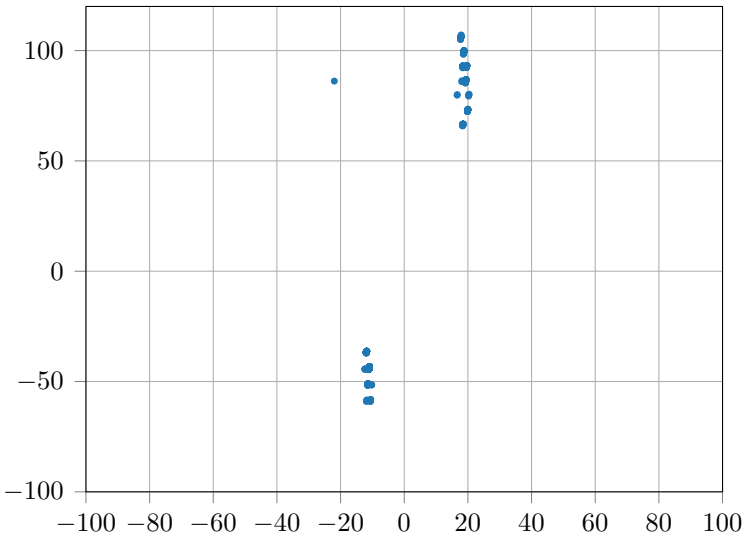


Abbildung 6.7: Die Abbildung zeigt den Fehler der Vorhersagen des *Hybrid-Modells* (in blau) in Relation zur tatsächlichen Position des Roboters an Koordinate $(0, 0)$. Einheiten in Millimeter.

6.4.2 Untersuchung des Hybrid-Modells

Das *Hybrid-Modell* hat über alle Experimente hinweg gute Ergebnisse erzielen können und hat in einigen Fällen sogar das *Roboter-Modell* überboten. Im Gegensatz dazu enttäuschte das *Personen-Modell*. Die Vermutung liegt also nahe, dass die gelernten klasseninternen Variationen im *Personen-Modell* für sich genommen auf humanoide *NAO*-Roboter übertragbar sind. Die gelernten Gewichte sind allerdings nicht universell anwendbar, was auch an verschiedenen Anwendungsbereichen liegen könnte, die durch den Trainings-Datensatz festgelegt wurden. Abbildung 6.8 zeigt eines der Modelle im *Personen-Modell*. Dieses ist offensichtlich für eine Frontal-Aufnahme gedacht, wobei sich das Modell eher auf den Oberkörper konzentriert. Der Kopf und die Schultern sind klar erkennbar.

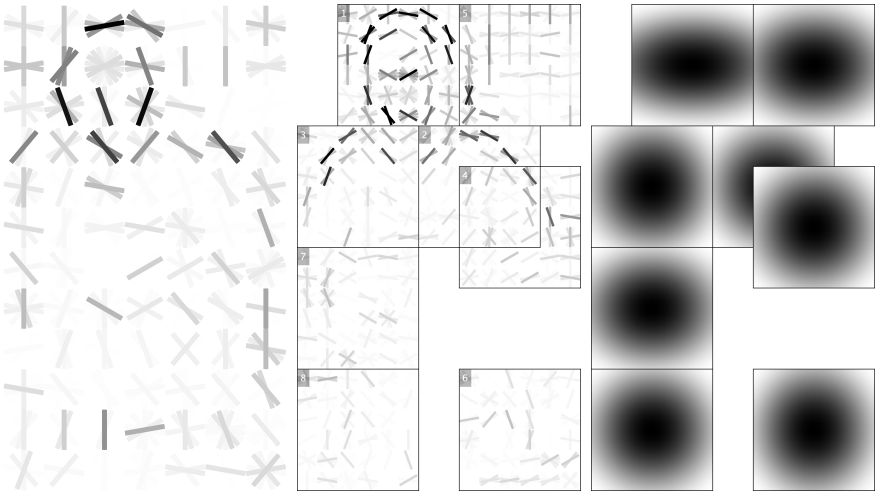


Abbildung 6.8: Die Abbildung zeigt die *Wurzel* (links), die *Teile* (Mitte) und die *deformation costs* (rechts) eines Modells aus dem *Personen-Modell* (*Release 4* aus Felzenszwalb *et al.* [2010b,a]).

Abbildung 6.9 zeigt dasselbe Modell, wie in Abbildung 6.8 – allerdings wurde dieses mit Bildern von humanoiden *NAO*-Robotern nachgelernt. In der Abbildung sieht man, dass insbesondere in der *Wurzel* viele Details hinzukamen. In der unteren Region des *Filters* ist jetzt eine Neigung zu horizontalen Gradienten-Richtungen zu sehen. Vorher war der untere Bereich wesentlich undefinierter, was vermutlich daran liegt, dass im ursprünglichen Trainings-Datensatz bei vielen der Personen der Unterkörper durch Tische oder andere Gegenstände verdeckt war. Die Region um den Kopf blieb in der *Wurzel* und den *Teilen* hauptsächlich unverändert. Die Veränderungen in den *deformation costs* sind leider nicht gut darstellbar, aber auch hier sind Veränderungen sichtbar – vor allem bei den Kosten für *Teil* 1. Einige Dynamiken des grundsätzlichen *Personen-Modells* gingen also verloren, da diese bei den *NAO*-Robotern nicht benötigt werden. Viele der ursprünglichen Gewichte in den *Filtern* blieben ebenfalls erhalten. Das *Hybrid-Modell* profitiert also vor allem aus der Positionierung der *Teile* und deren Gewichte. Das Nachtrainieren besteht dann im wesentlichen aus dem Anpassen der Gewichte. In den *deformation costs* verschwinden zusätzlich ungenutzte Informationen und neue werden modelliert.

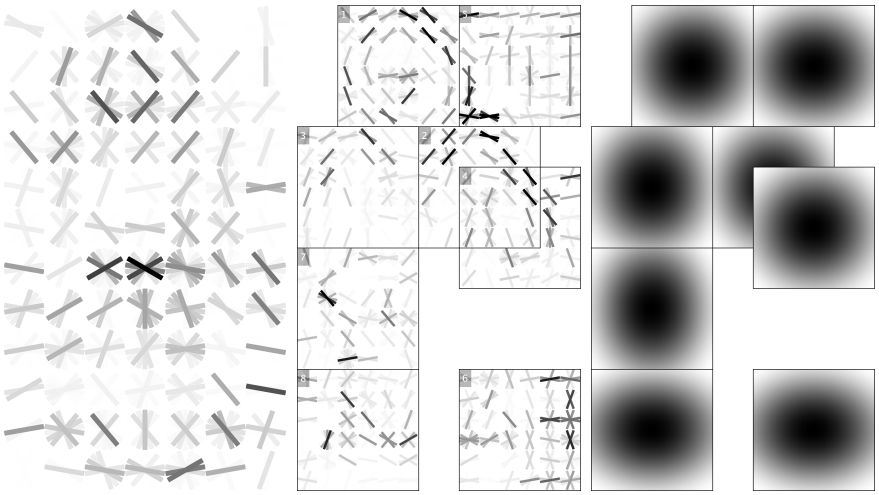


Abbildung 6.9: Die Abbildung zeigt die *Wurzel* (links), die *Teile* (Mitte) und die *deformation costs* (rechts) eines Modells aus dem *Hybrid-Modell*.

Basierend auf diesen Beobachtungen soll das *Simulator-Modell* mit einem Teil des Trainings-Datensatzes aus Kapitel 5.7 nachgelernt werden. Meine Vermutung dabei ist, dass dieses – genau wie das *Personen-Modell* – grundsätzlich die richtigen Variationen gelernt hat, die Gewichte allerdings nicht auf echte Bilder anwendbar sind. Zur Evaluation des neuen Modells wird der Datensatz aus dem Testspiel (Versuch 3) verwendet. Ich erwarte dabei eine spürbare Verbesserung der Erkennungsrate – auch schon bei einem kleinen Trainings-Datensatz.

Tabelle 6.11 zeigt die Ergebnisse von Versuch 3 für das originale und das nachgelernte *Simulator-Modell*. Die *AP* hat sich um rund 30% gesteigert. Auch der *Recall* (rund 10% mehr) und der *IoU* haben sich verbessert. Besonders stark hat sich die *Precision* verändert. Die nun erreichten 90% können sogar mit dem *Roboter-Modell* und dem *Hybrid-Modell* mithalten. Insgesamt ist das nachgelernte *Simulator-Modell* im *Recall* und der *AP* sogar leicht besser als das *Hybrid-Modell*.

Modell	AP	Precision	Recall	IoU
Simulator-Modell (Original)	0,057	0,168	0,263	0,67
Simulator-Modell (Nachgelernt)	0,349	0,908	0,360	0,69

Tabelle 6.11: Ergebnisse auf Bildern aus einem Testspiel von *B-Human*. Dargestellt sind die *AP*, die *Precision*, der *Recall* und der *IoU* gemittelt über alle Bilder für das originale *Simulator-Modell* und die nachgelernte Variante.

6.5 Diskussion

Das *Hybrid-Modell* und das *Roboter-Modell* haben in allen Vergleichen den aktuell in *B-Human* verwendeten Detektor übertroffen. Besonders bemerkenswert ist die hohe *Precision* über alle Experimente hinweg. Das spricht dafür, dass der Ansatz nur sehr wenige *False Positives* liefert. Bei dem *Recall* ist noch Luft nach oben. Die Untersuchung in Kapitel 6.4.2 hat aber gezeigt, dass es noch Möglichkeiten gibt, bessere Modelle zu trainieren. Die Positionen der *Teile* und die *deformation costs* könnte man über einen beliebig großen generierten Datensatz aus dem Simulator trainieren und anschließend die Gewichte mit einem wesentlich kleineren Datensatz echter Bilder anpassen. Die Untersuchung hat gezeigt, dass auch das *Simulator-Modell* die richtigen Informationen gelernt hat, aber die Gewichte aus dem Simulator nicht auf echte Bilder anwendbar sind.

Im ersten Experiment haben die *Deformable Part Models* gute Ergebnisse für die Posen *a* bis *e* gezeigt. Der hintere Roboter in Pose *b* konnte allerdings erst detektiert werden, nachdem der *Schwellwert* für Detektionen erheblich reduziert wurde. Dies brachte aber gleichzeitig auch einige Falsch-Erkennungen mit sich. *Deformable Part Models* können also auch Roboter mit verdeckten *Teilen* detektieren, allerdings fällt die letztendliche Bewertung dieser aufgrund der fehlenden *Teile* wesentlich geringer aus. Hier muss man also abwägen, ob man die geringere *Precision* mit dem höheren *Recall* durch die Detektion von verdeckten Robotern eintauschen will. Handelt es sich bei dem verdeckenden Objekt um einen zweiten Roboter ist es außerdem wahrscheinlich, dass die zweite Detektion aufgrund der schlechteren Bewertung in der *Nonmax-Suppression* aussortiert wird. Der *B-Human*-Detektor zeigte im ersten Experiment eine vergleichbar gute Erkennungsrate, da die gemittelten Durchschnittswerte durch die Ergebnisse für Pose *a* leicht verfälscht wurden. Auch hier zeigte der *B-Human*-Detektor eine annehmbare Anzahl von Detektionen – durch die ausgestrecktem Arme wurde aber der *IoU-Schwellwert* von 0,3 nicht überschritten.

Die *Deformable Part Models* können in der jetzigen Form keine liegenden Roboter detektieren. Das liegt an der Rotationsinvarianz der *Teile*, wodurch ein spezielles Modell gelernt werden müsste, welches nur diesen Extremfall modelliert.

Die erste Untersuchung hat einen grundlegenden Fehler in der Lokalisierung der Detektionen der *Deformable Part Models* festgestellt. Dieser ist auf die *Merkmal-Pyramide* zurückzuführen, wobei durch das Unterabtasten des Eingabebildes über die *Level* hinweg Präzision verloren geht. Detektionen auf verschiedenen *Leveln* sorgen ebenfalls dafür, dass die berechneten *Bounding-Boxes* variieren. Der *B-Human*-Detektor ist anfälliger gegen das Bildrauschen. Allerdings fallen die Variationen in den *Bounding-Boxes* hier nicht so stark aus. Da die *IoU* der *Deformable Part Models* über alle Experimente hinweg trotzdem deutlich höher als die definierte Schranke

von 50% war, hat dieser grundlegende Fehler keine großen Auswirkungen auf die allgemeine Erkennungsrate gezeigt.

Es hat sich eine Schwäche der *Deformable Part Models* bei sehr nahen Robotern offenbart. Diese ist auf den Trainings-Datensatz zurückzuführen. Dieser enthielt auch Roboter, welche sich auf einer wesentlich höheren Distanz als 3,5 Meter befanden, wodurch die gelernten *Filter* wohl auf einen Bereich ansprechen, der als Minimum eine Distanz hat, die größer als 0,3 Meter ist. Der *B-Human*-Detektor lieferte auf einer Entfernung von 0,3 Metern gute Ergebnisse, hatte aber Probleme bei 3,5 Metern. Da die Detektion von sehr nahen Robotern auch sehr wichtig ist, kann man das als Punktgewinn für den *B-Human*-Detektor ansehen. Die Detektion von Robotern in einer Entfernung von 3,5 Metern kann im Kontext des Roboter-Fußballs allerdings auch sehr wichtig sein, zur besseren Planung von zukünftigen Spielzügen oder Torschüssen.

Im zweiten Experiment wurde eine Robustheit der *Deformable Part Models* gegen Änderungen der Beleuchtung nachgewiesen. Stark überbelichtete Bilder lieferten dabei ein erwartet schlechtes Ergebnis. Dem liegt zugrunde, dass bei solchen Bildern zu viele Gradienten verloren gehen, da überbelichtete Bildbereiche auf den selben Pixelwert abgebildet werden. Leicht unterbelichtete Bilder stellen allerdings kein Problem dar. Die Detektion war in diesen Fällen sogar teilweise besser, was darauf zurückzuführen ist, dass viele Gradienten auf dem Feld verloren gehen und der Roboter sich so besser vom Hintergrund abhebt.

Im dritten Experiment haben die *Deformable Part Models* die guten Ergebnisse aus den vorausgegangenen Experimenten noch ein mal bestätigt. Die hohe *Precision* wird noch bemerkenswerter, wenn man bedenkt, dass hier der IoU mit 50% wesentlich restriktiver gewählt wurde, als beim *B-Human*-Detektor. Obwohl der Fehler in der Lokalisierung bei den *Deformable Part Models* größer ist, kann das *B-Human*-Framework von den genaueren *Bounding-Boxes* profitieren.

7. Fazit und Ausblick

Dieses Kapitel enthält ein abschließendes Fazit. Außerdem wird ein Ausblick gegeben, wie die Ergebnisse und Ansätze dieser Arbeit in bestehende Systeme eingepflegt und erweitert werden könnten.

7.1 Fazit

Ziel dieser Arbeit war die Überprüfung, ob *Deformable Part Models* auf humanoide *NAO*-Roboter anwendbar sind. Zu Beginn wurde eine Recherche durchgeführt, welche Arbeiten im Bereich der *Deformable Part Models* und der Detektion von *NAO*-Robotern enthielt. Auf Basis dieser Arbeiten wurde ein System entwickelt, welches *Deformable Part Models* zur Erkennung von *NAO*-Robotern benutzt. Während dieser Entwicklung wurden neue Konzepte entwickelt und wichtige Beobachtungen gemacht. Dies enthält die Implementierung der *Deformable Part Models* in das *B-Human*-Framework mit einer Reduzierung in den *Leveln* der *Merkmal-Pyramide* zur Vermeidung von Falsch-Erkennungen außerhalb des Spielfeldes, die Erzeugung diverser Modelle, die Ermittlung eines optimalen Parameter-Satzes, und die Beobachtung, dass ein vorgelerntes Modell auf einem kleineren Trainings-Datensatz an neue Begebenheiten angepasst werden kann.

Dass die Beobachtungen und Neuentwicklungen Konstanz haben, hat sich in der Evaluation gezeigt. Der neu entwickelte Detektor zeigte eine bessere Erkennungsrate als der momentan im *B-Human*-Framework verwendete Detektor und zeigte gleichzeitig nur sehr wenige Falsch-Erkennungen. Außerdem wurde nachgewiesen, dass die Erkennungsrate der Modelle noch verbessert werden kann.

Abschließend wird eine Bewertung des entwickelten Detektors auf Basis der Vorgaben aus der Einleitung vorgenommen.

- **Erkennungsrate:** Die vorgegebene *Precision* von 80% wurde in allen Experimenten vom *Hybrid-Modell* und dem *Roboter-Modell* erreicht.
- **Detektion bei verschiedenen Lichtbedingungen:** Die Evaluation hat nachgewiesen, dass die Detektion zuverlässig bei schwächerer Beleuchtung arbeitet. Überbelichtung führt allerdings zu großen Problemen.

- **Detektion auf verschiedenen Entfernungen:** Die *Deformable Part Models* haben zuverlässige Erkennungsraten auf der vorgegebenen Entfernung von 3,5 Metern geliefert und überboten dabei in allen Fällen den aktuell in *B-Human* verwendeten Detektor. Auf sehr kurze Entfernungen offenbarte der neu entwickelte Detektor Probleme.
- **Verdeckte Roboter und nebeneinander stehende Roboter:** Die *Deformable Part Models* zeigten Probleme bei hintereinander stehenden Robotern, wobei nur der vordere Roboter mit hoher *Precision* detektiert wurde. Nebeneinander stehende Roboter stellten keine Probleme dar, wie das erste Experiment gezeigt hat.

Der entwickelte Detektor und die im Zuge dieser Arbeit gelernten Modelle erfüllen die meisten Vorgaben. Als nicht erfüllt gelten die Detektionen auf überbelichteten Bildern und die zuverlässige Detektion von hintereinander stehenden Robotern. Abschließend kann man also sagen, dass *Deformable Part Models* zur Detektion von humanoiden *NAO*-Robotern verwendet werden können. Der in dieser Arbeit entwickelte Detektor eignet sich allerdings nicht für die Anwendung auf einem Roboter. Grund dafür ist die zu lange Laufzeit. Obwohl die Implementierung schon stark optimiert wurde und aufwendige Berechnungen auf einer Grafikkarte stattfinden, benötigt das *Hybrid-Modell* im Schnitt 250 ms auf einer GTX 970, was in einem System, das auf Echtzeit ausgelegt ist, nicht tragbar ist. Es wird daher eine Anwendung empfohlen, bei der die Laufzeit keinen großen Faktor darstellt.

Die in dieser Arbeit vorgestellten Ergebnisse sind nicht nur für den Roboter-Fußball relevant. Es wurde gezeigt, dass *Deformable Part Models* auch ohne große Anpassungen in anderen Anwendungsbereichen gute Erkennungsraten liefern können und sich diese nicht nur auf die Detektion von Personen beschränken.

7.2 Ausblick

Es gibt verschiedene Beobachtungen und Ansätze, welche weiter verfolgt und evaluiert werden könnten.

Zum einen kann man weiter evaluieren, wo die Grenzen des Nachlernens und insbesondere des *Personen-* und *Simulator-Modells* liegen. Kann man Modelle beliebig kombinieren? Vielleicht ist es sogar möglich, das *Personen-* und das *Simulator-Modell* zu kombinieren und anschließend mit echten Bildern nachzulernen. An dieser Stelle müsste ermittelt werden, inwiefern mehrere Kombinationen sinnvoll sind und ab welcher Kombination zu viele Informationen aus vorherigen Modellen verloren gehen.

In Girshick *et al.* [2014] wird beschrieben, wie man *Deformable Part Models* in neuronale Netze umformen kann. Diese neuronalen Netze könnten evaluiert werden, da sie laut der Arbeit eine bessere Erkennungsrate liefern und wesentlich schneller berechnet werden können. Erste Ansätze mit diesen so genannten *Deep Pyramid Deformable Part Models* können Ranjan *et al.* [2015] entnommen werden. Vielleicht wird der Detektor mit diesem Ansatz echtzeitfähig und kann doch auf einem Roboter verwendet werden.

Eine Verwendung des Detektors auf einem Roboter halte ich im jetzigen Zustand für unwahrscheinlich. Wenn man aber einen anderen Anwendungsbereich – wie z.B.

die Analyse von Videos von einem Spiel – betrachtet, könnte der Detektor dort eine Anwendung finden. In diesem Falle würde man von der hohen *Precision* und dem guten IoU des Detektors profitieren. Befindet sich die Kamera am Spielfeldrand, gibt es außerdem nicht das Problem, dass sehr nahe Roboter detektiert werden müssen. Als Beispiel kann man das System von Stöwing [2018] nennen.

Man könnte auch die Anwendung des Detektors auf ein anderes Objekt evaluieren. Das Tor oder einzelne Torpfosten bieten sich da an. Diese weißen Objekte heben sich in den meisten Fällen ebenfalls sehr gut vom Hintergrund ab. Die *deformation costs* würden in dem Fall die perspektivischen Verzerrungen modellieren.

Literaturverzeichnis

Çakmak, F., Uslu, E., Altuntaş, N., Marangoz, S., Balcılar, M., Amasyalı, M. F., & Yavuz, S. 2016 (may). Deformable Part Model and Deep Learning Comparison on Victim Detection. *Pages 1513–1516 of: 2016 24th Signal Processing and Communication Application Conference (SIU).*

Albani, Dario, Youssef, Ali, Suriani, Vincenzo, Nardi, Daniele, & Bloisi, Domenico Daniele. 2017. A Deep Learning Approach for Object Recognition with NAO Soccer Robots. *Pages 392–403 of:* Behnke, Sven, Sheh, Raymond, Sarıel, Sanem, & Lee, Daniel D. (eds), *RoboCup 2016: Robot World Cup XX.* Lecture Notes in Computer Science. Springer International Publishing.

B-Human. 2019. *B-Human Homepage.* https://www.b-human.de/index-de.html. Zuletzt aufgerufen: 16.04.2019.

Burger, Wilhelm, & Burge, Mark James. 2005. *Digitale Bildverarbeitung: eine Einführung mit Java und ImageJ.* eXamen.press. Berlin: Springer. OCLC: 249181488.

Chai, Yuning, Lempitsky, Victor, & Zisserman, Andrew. 2013. Symbiotic Segmentation and Part Localization for Fine-Grained Categorization. *Pages 321–328 of: 2013 IEEE International Conference on Computer Vision.* Sydney, Australia: IEEE.

Dalal, N., & Triggs, B. 2005. Histograms of Oriented Gradients for Human Detection. vol. 1. IEEE. http://ieeexplore.ieee.org/document/1467360/. Zuletzt aufgerufen: 26.04.2019.

Dubout, Charles. 2019. *Ffld.* http://www.dubout.ch/en/code/ffld_v2.tar.bz2. Zuletzt aufgerufen: 16.04.2019.

Felzenszwalb, P. F., Girshick, R. B., & McAllester, D. 2010a. *Discriminatively Trained Deformable Part Models, Release 4.* http://people.cs.uchicago.edu/~pff/latent-release4/. Zuletzt aufgerufen: 01.03.2019.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. 2010b. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **32**(9), 1627–1645.

Felzenszwalb, Pedro, McAllester, David, & Ramanan, Deva. 2008. A Discriminatively Trained, Multiscale, Deformable Part Model. *Pages 1–8 of: 2008 IEEE Conference on Computer Vision and Pattern Recognition.* Anchorage, AK, USA: IEEE.

Fiedler, Niklas, Bestmann, Marc, & Hendrich, Norman. 2018. ImageTagger: An Open Source Online Platform for Collaborative Image Labeling. *In: RoboCup 2018: Robot World Cup XXII*. Springer.

Ghiasi, Golnaz, & Fowlkes, Charless C. 2014. Occlusion Coherence: Localizing Occluded Faces with a Hierarchical Deformable Part Model. *Pages 1899–1906 of: 2014 IEEE Conference on Computer Vision and Pattern Recognition*. Columbus, OH, USA: IEEE.

Girshick, Ross, Iandola, Forrest, Darrell, Trevor, & Malik, Jitendra. 2014. Deformable Part Models are Convolutional Neural Networks. http://arxiv.org/abs/1409.5403. Zuletzt aufgerufen: 26.04.2019.

Jonathan Hui. 2018. *mAP (mean Average Precision) for Object Detection*. https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173. Zuletzt aufgerufen: 16.04.2019.

Laue, Tim, & Röfer, Thomas. 2008. SimRobot - Development and Applications. *In:* Amor, Heni Ben, Boedecker, Joschka, & Obst, Oliver (eds), *The Universe of RoboCup Simulators - Implementations, Challenges and Strategies for Collaboration. Workshop Proceedings of the International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR 2008)*.

Poppinga, Bernd. 2018. *Nao You See Me: Echtzeitfähige Objektdetektion für mobile Roboter mittels Deep Learning*. Master's Thesis, University of Bremen.

Ranjan, Rajeev, Patel, Vishal M., & Chellappa, Rama. 2015. A Deep Pyramid Deformable Part Model for Face Detection. *arxiv:1508.04389 [cs]*, aug. arXiv: 1508.04389.

RoboCup Federation. 2019a. *A Brief History of RoboCup*. http://www.robocup.org/a_brief_history_of_robocup. Zuletzt aufgerufen: 16.04.2019.

RoboCup Federation. 2019b. *RoboCup - Objective*. http://www.robocup.org/objective. Zuletzt aufgerufen: 16.04.2019.

RoboCup Federation. 2019c. *RoboCupSoccer - Standard Platform*. http://www.robocup.org/leagues/5. Zuletzt aufgerufen: 16.04.2019.

RoboCup Federation. 2019d. *Standard Platform League Overview*. https://spl.robocup.org/history/. Zuletzt aufgerufen: 16.04.2019.

Röfer, Thomas, Laue, Tim, Hasselbring, Arne, Heyen, Jannik, Poppinga, Bernd, Reichenberg, Philip, Roehrig, Enno, & Thielke, Felix. 2018. *B-Human Team Report and Code Release 2018*. https://github.com/bhuman/BHumanCodeRelease/raw/master/CodeRelease2018.pdf. Zuletzt aufgerufen: 26.04.2019.

Rosebrock, Adrian. 2015. *Find Distance from Camera to Object/Marker Using Python and OpenCV*. https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/. Zuletzt aufgerufen: 16.04.2019.

SoftBank Robotics. 2019a. *NAO - Dimensions*. http://doc.aldebaran.com/2-8/family/nao_technical/dimensions_naov6.html. Zuletzt aufgerufen: 16.04.2019.

SoftBank Robotics. 2019b. *NAO - Motherboard*. http://doc.aldebaran.com/2-8/family/nao_technical/motherboard_naov6.html. Zuletzt aufgerufen: 16.04.2019.

SoftBank Robotics. 2019c. *NAO - Video camera*. http://doc.aldebaran.com/2-1/family/robots/video_robot.html#robot-video. Zuletzt aufgerufen: 16.04.2019.

SoftBank Robotics. 2019d. *NAO the humanoid robot*. https://www.softbankrobotics.com/emea/en/nao. Zuletzt aufgerufen: 16.04.2019.

Stöwing, Alexander. 2018. *Automatisierte Videoanalyse zur Generierung von Referenzdaten für Spiele in der RoboCup Standard Platform League*. Master's Thesis, University of Bremen.

Tian, Yicong, Sukthankar, Rahul, & Shah, Mubarak. 2013. Spatiotemporal Deformable Part Models for Action Detection. *Pages 2642–2649 of: 2013 IEEE Conference on Computer Vision and Pattern Recognition*. Portland, OR, USA: IEEE.

Wada, Kentaro. 2019. *labelme*. https://github.com/wkentaro/labelme. Zuletzt aufgerufen: 16.04.2019.

Weigel, David. 2015. *Videobasierte Lokalisierung von Hallenfußballspielern*. Master's Thesis, University of Bremen.

Yang, Y., & Ramanan, D. 2011 (jun). Articulated Pose Estimation with Flexible Mixtures-Of-Parts. *Pages 1385–1392 of: CVPR 2011*.

Yavuz, Sırma, Amasyali, M Fatih, Uslu, Erkan, & Marangoz, Salih. 2016. RoboCup Rescue 2016 Team Description Paper yildiz.