Eine Deep-Learning-basierte Bildverarbeitung für humanoide Roboter zur Erkennung von Merkmalen auf der Bodenebene

Masterarbeit

Felix Thielke

Matrikelnummer: 2909855

2. September 2019

Fachbereich Mathematik / Informatik
Studiengang Informatik

1. Gutachter: Dr. Tim Laue
2. Gutachter: Prof. Dr. Rolf Drechsler

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig angefertigt und nicht anderweitig zu Prüfungszwecken vorgelegt habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel verwendet. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Bremen, den 2. September 2019

.....

(Felix Thielke)

Inhaltsverzeichnis

Kapitel 1

Einleitung

Damit ein autonomer Roboter mit seiner Umgebung interagieren kann, muss er wissen, wo in dieser Umgebung er sich befindet, insbesondere relativ zu den Objekten, auf die er Einfluss nehmen soll. Zur Orientierung in bekannten Umgebungen werden bei mit Kameras ausgestatteten Robotern häufig Bildverarbeitungsalgorithmen genutzt, um bekannte Landmarken zu erkennen und anhand der so gefundenen Referenzpunkte die eigene Position zu ermitteln.

Während die entsprechenden Landmarken in der Regel feste Objekte sind, gibt es einige Domänen, in denen die markantesten Landmarken aus Markierungen auf der Bodenebene bestehen. Eine solche Domäne ist der Roboterfußball – hierbei müssen sich Roboter anhand der auf dem Spielfeld befindlichen Feldlinien orientieren.

In diesem Kontext wird in der vorliegenden Arbeit ein Bildverarbeitungssystem entwickelt, welches Robotern ermöglichen soll, auf einem Fußballfeld die eigene Position und Orientierung zu bestimmen.

1.1 Hintergrund

Die vorliegende Abschlussarbeit entstand im Kontext des Teams B-Human, einem studentischen Projekt der Universität Bremen in Kooperation mit dem Deutschen Forschungszentrum für Künstliche Intelligenz. Das Team B-Human tritt seit 2009 regelmäßig bei den internationalen Forschungswettbewerben der RoboCup Federation in der Standard Platform League an, bei denen Roboter des Typs *NAO* gegeneinander Fußball spielen.

1.1.1 RoboCup

Die alljährlich stattfindenden Wettbewerbe der RoboCup Federation wurden im Jahr 1993 zur Förderung der Forschung in den Bereichen Robotik und künstliche Intelligenz ins Leben gerufen und finden seit 1997 regelmäßig statt (vgl. The Robocup Federation, o.D.(a)). Das Ziel im Forschungskontext ist dabei, kontrollierte Umgebungen und Standardprobleme zu

schaffen, um Theorien, Algorithmen und Softwarearchitekturen in diesen Domänen evaluieren zu können. Darüber hinaus wurde das übergreifende Ziel definiert, im Jahr 2050 ein Team vollständig autonomer humanoider Roboter stellen zu können, welches in der Lage ist, den dann amtierenden menschlichen Fußballweltmeister nach den offiziellen Regeln der FIFA zu schlagen (siehe The Robocup Federation, o.D.(b)).

Im RoboCup gibt es zur Erreichung dieser Ziele Wettbewerbe in verschiedenen Ligen, welche neben Roboterfußball noch einige weitere Domänen wie zum Beispiel Logistik, Rettung und Mensch-Maschine-Interaktion abdecken.

Der Kontext dieser Arbeit ist dabei die RoboCup Soccer Standard Platform League (SPL). In dieser wird mit Robotern Fußball gespielt, wobei die verwendete Hardware vorgeschrieben und nicht modifizierbar ist, sodass die Spiele einen reinen Softwarewettbewerb bilden. Nach den zum Zeitpunkt dieser Arbeit aktuellen Regeln (siehe RoboCup Technical Committee, 2019) spielen dabei zwei Teams von je fünf Robotern des Typs *NAO* gegeneinander in zwei zehnminütigen Halbzeiten auf einem Spielfeld der Größe 9×6 Meter. Das Feld besteht dabei wie in Abb. 1.1 zu sehen aus einem grünen Teppich mit weißen Feldlinien und Toren und gespielt wird mit einem weiß-schwarzen Schaumstoffball mit 10 cm Durchmesser.

Abbildung 1.1 Ein Fußballspiel in der RoboCup SPL. Bild von Tim Laue (CC-BY-SA 4.0).

1.1.2 NAO

Das in der SPL verwendete Robotermodell ist der in Abb. 1.2 gezeigte humanoide *NAO* von Softbank Robotics, der aktuell in der sechsten Version vorliegt.

Dieser ist mit zwei Kameras mit einem horizontalen und vertikalen Öffnungswinkel von 56,3° beziehungsweise 43,7° ausgestattet, wobei die untere Kamera im „Kinn“ des Roboters so ausgerichtet ist, dass sie im Wesentlichen den Nahbereich vor den Füßen des Roboters sieht, während die obere Kamera in der „Stirn“ positioniert ist und daher in den Fußballspielen in den meisten Situationen einen Überblick über das Spielfeld hat. Jede der Kameras nimmt

Abbildung 1.2 Der in der RoboCup SPL verwendete Roboter *NAO* V6. Bild von Softbank Robotics (o.D.).

pro Sekunde 30 Bilder auf.

Der im *NAO* verbaute Prozessor, der die in dieser Arbeit entwickelten Algorithmen ausführen soll, ist ein Intel Atom E3845 mit einer Taktrate von 1,91 GHz, dem 4 GB Arbeitsspeicher zur Verfügung stehen.

1.2 Motivation

Ein wichtiger Faktor, damit Roboter effektiv Fußball spielen können, ist, dass sie ihre eigene Position auf dem Feld kennen. Diese kann in der vorliegenden Umgebung am besten durch Orientierung an den Eigenschaften des Spielfelds bestimmt werden, welche in den Kamerabildern erkannt werden.

In den vergangenen Jahren wurde das resultierende Bildverarbeitungsproblem in der Regel durch analytische Ansätze gelöst, die durch Ausnutzung von Domänenwissen und vereinfachte Annahmen in ihrer Komplexität sehr einfach gehalten wurden, um auf der beschränkten Hardware der verwendeten Roboter echtzeitfähig zu funktionieren, das heißt für jedes aufgenommene Kamerabild ein Ergebnis zu liefern.

Diese Rahmenbedingungen haben sich jedoch mit der aktuellen Robotergeneration verändert: im Vergleich zum zuvor genutzten *NAO* V5 ist der Prozessor um etwa 20 % schneller getaktet, durch das Vorhandensein mehrerer echter CPU-Kerne kann die Bildverarbeitung beider Kameras parallel stattfinden und es steht viermal so viel Arbeitsspeicher zur Verfügung. Damit liegt nun eine Situation vor, in der es sinnvoll ist, die bisher genutzten Herangehensweisen an die Bildverarbeitung zu überdenken.

Überdies ist mittlerweile in vielen Bereichen der Bildverarbeitung die Nutzung von neuronalen Netzen die präferierte Herangehensweise, die in vielen Domänen bestehende analytische Ansätze ersetzt hat. Auch in den verschiedenen Ligen des RoboCup werden in der kürzeren Vergangenheit mehr und mehr Bildverarbeitungsprobleme durch maschinelles Lernen gelöst. Somit bietet es sich an, ein derartiges Verfahren auch für das genannte Problem der Selbstlokalisierung zu nutzen.

1.3 Zielsetzung

Basierend auf dem oben genannten Umstieg auf eine neue Robotergeneration soll diese Arbeit unter Betrachtung von sowohl analytischen als auch auf neuronalen Netzen basierenden neuen Herangehensweisen die Frage beantworten:

Kann durch die Nutzung rechenintensiverer Bildverabeitungsalgorithmen die Selbstlokalisierung von Robotern in der RoboCup SPL verbessert werden?

Hierzu wird ein Bildverarbeitungssystem entwickelt, das als Eingabe von Fußball spielenden Robotern aufgenommene Kamerabilder erhält und die Positionen und Orientierungen von für die Lokalisierung des Roboters verwendbaren Merkmalen des Spielfelds ausgibt. Hierfür verwendbar sind alle Merkmale, die sich statisch an festen Positionen auf dem Feld befinden – in der SPL sind dies Linien, Linienkreuzungen, Strafstoßpunkte, Feldecken, Mittelkreise und Torpfosten.

Das entwickelte System soll dabei auf der vorgegebenen Hardwareplattform ausreichend schnell sein, um während eines Roboterfußballspiels eingesetzt werden zu können, sowie Ergebnisse liefern, die so genau wie möglich sind. Insbesondere sollte die Qualität der Ausgaben für eine genaue Selbstlokalisierung auf dem Spielfeld besser geeignet sein als die von der bisher vom Team B-Human verwendeten Software.

1.4 Aufbau der Arbeit

Um den wissenschaftlichen Kontext dieser Arbeit zu verdeutlichen, enthält das folgende Kapitel 2 zunächst eine Übersicht an relevanten Veröffentlichungen zu Themen, die mit der vorliegenden Problemstellung verwandt sind. Daraufhin beschreiben Kapitel 3, 4 und 5 die zur Lösung des Problems implementierten Ansätze. Anschließend werden diese entwickelten Verfahren in Kapitel 6 hinsichtlich ihrer Tauglichkeit zur Lösung der genannten Problemstellung beschrieben, bevor schlussendlich in Kapitel 7 ein Fazit gezogen wird und Anregungen für Verbesserungen und Erweiterungen der hier entwickelten Methoden gegeben werden.

Kapitel 2

Verwandte Arbeiten

Das Erkennen von Merkmalen auf der Bodenebene ist insbesondere in den verschiedenen Ligen des RoboCup ein relevantes Forschungsthema, da in den jeweiligen Umgebungen ebenjene häufig die nützlichsten Merkmale für die Selbstlokalisierung des Roboters darstellen.

Allerdings stellt sich das Problem auch in anderen Domänen der Bildverarbeitung, in denen die relative Position und Ausrichtung der Kamera zum Boden bekannt ist und wichtige Informationen aus der Detektion bekannter Merkmale auf der Bodenebene gewonnen werden können. Insbesondere findet viel Forschungsarbeit im Bereich des autonomen Fahrens statt, in dem sich das Problem in Form der Erkennung von Straßenmarkierungen zeigt.

Im Folgenden ist eine Auswahl relevanter verwandter Arbeiten vorgestellt.

2.1 Erkennung von Lokalisierungsmerkmalen im RoboCup

Da in den Fußballligen des RoboCup die Spielfeldmarkierungen ein wichtiges Merkmal zur Selbstlokalisierung der Roboter darstellen, gibt es zur Erkennung dieser Merkmale seit Jahren viele Forschungsansätze.

Die von **Lu, L. Liu und Zhao (2012)** vorgestellte Methode zur Extraktion von Spielfeldmerkmalen aus Kamerabildern geht davon aus, dass zur Selbstlokalisierung der Roboter das Wissen über den Verlauf von Feldlinien relativ zum Roboter ausreicht. Um diese in einem Graustufenbild zu erkennen, werden zunächst Kantenpunkte, deren Helligkeit sich von ihren Nachbarn um mindestens einen Schwellwert unterscheidet, aus dem Bild extrahiert. Aufgrund der Beobachtung, dass in Kamerabildern der verwendeten Roboter mit zunehmender Entfernung Linien zwar dunkler wirken, die Helligkeitswerte des grünen Spielfelds sich aber nicht unterscheiden, ist der dabei verwendete Schwellwert linear von der Distanz der jeweiligen Punkte auf dem Spielfeld zum Roboter abhängig. Zu jedem der so erhaltenen Kantenpunkte wird dann durch lokale Anwendung des Sobel-Operators die Orientierung seiner Kante ermittelt, woraufhin durch eine Hough-Transformation Linien bestehend aus Kantenpunkten ähnlicher Ausrichtung gefunden werden. Die resultierenden Linien werden dann paarweise zu

Feldlinien zusammengefasst.
Ein Problem dieses Ansatzes ist, dass dadurch aussagekräftige Feldmerkmale wie der Mittelkreis nicht erkannt werden. Auch wurden keine Experimente durchgeführt, um die Robustheit des Algorithmus bei anderen weißen Objekten im Bild – zum Beispiel Ball oder andere Roboter – zu prüfen.

Andere Verfahren versuchen, weitere Merkmale des Spielfelds zu erkennen, um so mehr Informationen für die Lokalisierung des Roboters zu erhalten. **Kar, Jain und Rout (2016)** verlassen sich hierfür auf die Erkennung von Feldlinienkreuzungen. Diese werden hierbei anhand ihrer Form in drei Kategorien unterteilt: L-, T- und X-Kreuzungen, welche jeweils aus Linien bestehen, die ausgehend von der Kreuzung in entsprechend zwei, drei oder vier Richtungen verlaufen. In Bezug auf das Spielfeld umfassen L-Kreuzungen somit die Außenecken sowie die Ecken der Strafräume, T-Kreuzungen liegen zwischen Mittellinie und Seitenlinien und Strafräumen und Torlinien und X-Kreuzungen bezeichnen die Kreuzungen des Mittelkreises mit der Mittellinie. Das zur Detektion dieser Kreuzungen im Kamerabild verwendete Verfahren gliedert sich dabei in fünf Schritte:

1. Bestimmen der Region of Interest durch Bildung der konvexen Hülle des größten grünen Flecks im Bild.
2. Finden potentieller Linienpixel durch Binarisierung des Graustufenbilds mit einem Schwellwert.
3. Skellettieren der erhaltenen Linien durch konnektivitätserhaltende Erosion.
4. Bilden eines Graphen aus verbundenen Linienpixeln.
5. Bestimmen der Positionen von Feldlinienkreuzungen durch Finden von Knoten des Grads Drei (T), Vier (X) oder Zwei (L), wobei jeweils ungefähr ein rechter Winkel zwischen den inzidenten Kanten sein muss.

Gudi u. a. (2013) verwenden hingegen zusätzlich zu den Feldlinienkreuzungen noch den Mittelkreis und die Torpfosten. Hierzu gehen sie davon aus, dass sich der Roboter zu jeder Zeit auf dem Spielfeld befindet. Unter dieser Annahme erkennen sie zunächst den Rand des Spielfeldes, um damit eine Region of Interest für die Feldmerkmalserkennung zu erhalten. Dies geschieht durch über das Kamerabild verteiltes vertikales Scannen nach Ansammlungen von grünen Pixeln, wobei Grün als ein festgelegter Farbbereich im HSV-Farbraum definiert ist. Das unterhalb des Feldrands befindliche verbleibende Bild wird dann binarisiert, indem pro Pixel geprüft wird, ob seine Farbe innerhalb des definierten HSV-Farbbereichs für Weiß liegt. Im resultierenden Binärbild sucht ein Algorithmus dann nach potentiell auf Feldlinien liegenden Punkten, um im nächsten Schritt eine Menge von Feldlinien durch Verbinden dieser Punkte zu erhalten. Indem die Lage von so ermittelten Feldlinien zueinander analysiert wird, können dann die Positionen von L-, T- und X-Kreuzungen bestimmt werden. Anschließend wird im binarisierten Linienbild auch der Mittelkreis durch Suche nach Ellipsen mit

der Methode von Pătrăucean (2012) gefunden. Das letzte im vorgestellten Ansatz extrahierte Spielfeldmerkmal sind die Torpfosten. Hierbei wird die Tatsache genutzt, dass diese zum damaligen Zeitpunkt in der SPL gelb statt wie heute weiß waren. Mit diesem Wissen kann das Kamerabild erneut binarisiert werden, diesmal mit der Farbe Gelb als Vordergrund, und im resultierenden Binärbild nach Torpfosten und zugehöriger Torlatte gesucht werden.
Neben der genannten Methode der Merkmalserkennung schlagen Gudi u. a. (2013) außerdem erstmals die Anwendung von Inverse Perspective Mapping (IPM, vgl. Kapitel 3) als Ausgangspunkt der Merkmalserkennung im Roboterfußball vor, durch die ein Bild generiert wird, das die Ansicht der Umgebung des Roboters aus der Vogelperspektive zeigt, nutzen dieses allerdings nicht, da es ihnen nicht möglich war, die genannten Verfahren direkt auf das resultierende IPM-Bild anzuwenden.

Auch **Farazi, Allgeuer, Ficht u. a. (2017)** behandeln das Problem der Erkennung von Feldlinien und dem Mittelkreis. Allerdings verwenden sie dazu keinerlei vorgegebenen Farbwert- oder Helligkeitsschwellwerte, sondern wenden den Canny-Algorithmus zur Kantendetektion auf das von der Kamera erhaltene Graustufenbild an. Im resultierenden Kantenbild finden sie dann Liniensegmente durch eine probabilistische Hough-Transformation, aus denen sie durch Ausgleichsrechnung Feldlinien sowie den Mittelkreis berechnen.

Qian und Lee (2017) stellten ein genaues Verfahren zur Erkennung des Spielfeldrandes vor und argumentieren, dass es damit möglich ist, diese Information nicht nur zur Bestimmung des relevanten Bildbereiches, sondern bei Kenntnis der genauen Form des Spielfelds auch selbst zur Unterstützung der Lokalisierung zu nutzen. Hierzu finden sie ebenso wie eines der oben genannten Verfahren zunächst potentielle Feldrandpixel im Kamerabild an den höchsten Stellen, an denen beim vertikalen Scannen jeweils eine bestimmte Anzahl grüner Pixel gefunden wird. Im Gegensatz zu den bisher genannten Verfahren wird aber der Farbbereich für Grün nicht vorgegeben, sondern durch Bildung eines Histogramms von Farbwerten in der unteren Kamera des verwendeten *NAO*-Roboters bestimmt, unter der Annahme, dass diese stets so ausgerichtet ist, dass das gesamte Kamerabild auf der Bodenebene und innerhalb des Spielfelds liegt. Aus den so gefundenen potentiellen Feldrandpixeln wird dann ein konkretes Modell des Feldrands erhalten durch Bestimmen von Linien durch diese Punkte mithilfe des RANSAC-Algorithmus (vgl. Fischler und Bolles, 1981). Ein ähnliches Verfahren wird auch vom Team B-Human seit der zu diesem Thema verfassten Abschlussarbeit von Schröder (2017) genutzt (vgl. Röfer, Laue, Hasselbring u. a., 2018).

Im Gegensatz zu den bisher genannten Arbeiten stützt sich das aktuell vom Team B-Human für die Selbstlokalisierung genutzte Bildverarbeitungsverfahren (siehe **Röfer, Laue, Bülter u. a., 2017**) auf die Detektion einer Reihe von komplexeren Feldmerkmalen auf Basis erkannter Linien, Linienkreuzungen, Strafstoßpunkte und dem Mittelkreis. Dieser Ansatz wird später in Kapitel 4.1 genauer beschrieben und soll zum Teil auch für die in dieser Arbeit entwickelten Verfahren als Grundlage dienen.

2.2 Erkennung von Bodenmerkmalen

Außerhalb des RoboCups ist das Erkennen bekannter Merkmale auf der Bodenebene insbesondere im Bereich des autonomen Fahrens relevant; entsprechend kommen alle im Folgenden genannten Arbeiten aus diesem Kontext. Hierbei ist die Problemstellung in der Regel die Erkennung von Straßenmarkierungen, wobei vereinfacht von einem flachen Untergrund ausgegangen wird. Im Gegensatz zum Roboterfußball gibt es dabei auf Straßen nicht nur gerade, sondern auch gebogene Linien; außerdem sind Kreuzungen von Linien weniger relevante Merkmale als lange, zusammengehörende Spurmarkierungen.

Aly (2008) verwendet zur Erkennung von Spurmarkierungen auf Straßen ein Graustufenbild, zu welchem er zunächst ein IPM-Bild berechnet. Nach einer Glättung per Gauß-Filter wird dieses Bild dann durch Verwendung eines Helligkeitsschwellwerts binarisiert. Im resultierenden Binärbild werden dann in mehreren Schritten Linien gefunden und verfeinert:

1. Durchführung einer vereinfachten Variante der Hough-Transformation zur Extraktion von Linien
2. Ausschneiden eines Bereichs um jede der gefundenen Linien und anschließendes Finden der am besten zu den weißen Pixeln in diesem Bereich passenden Linie per RANSAC.
3. Ausschneiden eines Bereichs um jede der gefundenen Linien und anschließendes Finden des am besten zu den weißen Pixeln in diesem Bereich passenden Splice per RANSAC, wobei die zuvor gefundene Linie als Initialisierung für das Modell genutzt wird.
4. Rückprojektion der gefundenen Splines ins Kamerabild.
5. Anpassung der Positionen und Verläufe der Splines an die Gradienten der Helligkeitswerte im Bild.
6. Verlängerung der Splines durch Verfolgung der entsprechenden Linien im Bild.
7. Ausfilterung von zu kurzen und zu stark gebogenen Splines.

Guo, Mita und McAllester (2010) hatten zur Lösung des gleichen Problems eine Stereokamera und damit zwei Kamerabilder statt einem zur Verfügung. Auch in ihrem Ansatz wird dazu zunächst für beide Bilder eine IPM angewendet. Diese hat in diesem Fall zusätzlich den Vorteil, dass die beiden Kamerabilder nicht registriert werden müssen, da die Pixel in den IPM-Bildern sich bereits im selben Koordinatensystem befinden. Als nächstes werden „Hinweise“ für Straßenmarkierungen pro Pixel mit zwei verschiedenen Verfahren gesucht. Das eine dieser Verfahren sucht nach geometrischen Hinweisen, indem zunächst pro Bild mit dem Canny-Algorithmus ein Kantenbild generiert wird und anschließend eine Kaskade von Filtern auf dieses angewendet wird. Das andere Verfahren hingegen ist intensitätsbasiert und findet Hinweise für Straßenmarkierungen durch Berechnung der Kreuzkorrelation pro Pixel

zwischen den beiden IPM-Bildern und Extraktion der Pixel mit Werten über einem definierten Schwellwert. Die so gefundenen Pixel werden anschließend noch durch ein kleines neuronales Netz, das jeweils einen 9×3 Pixel großen Ausschnitt um sie als Eingabe erhält, in die Klassen feste und aufgemalte Straßenmarkierung eingeordnet. Nachdem auf beide Weisen Hinweise für Straßenmarkierungen gefunden wurden, werden diese in Form einer Konfidenz pro Pixel zusammengefasst. Hierbei erhalten aufgemalte Straßenmarkierungen eine höhere Konfidenz, um im Falle von temporären Änderungen des Verkehrsverlaufs nur die geänderten Spuren zu erkennen. Basierend auf den Konfidenzen werden schlussendlich durch Nutzung von Partikelfiltern Splines gefunden, die die Linienverläufe optimal modellieren sollen.

Im von **Lipski u. a. (2008)** umgesetzten System hingegen stehen Bilder von vier Kameras zur Verfügung, die einen großen Bereich vor und neben dem Fahrzeug abdecken, auf dem die Software läuft. Sie integrieren die Kamerabilder miteinander, indem sie das IPM-Bild jeder Kamera bilden und die resultierenden Bilder zusammenfügen, um so ein Bild aus der Vogelperspektive zu erhalten, welches einen 30×12 Meter großen Bereich vor dem Fahrzeug vollständig enthält. Um in diesem Bild nun Fahrbahnmarkierungen zu erkennen, wird es in 8×8 Pixel große Regionen aufgeteilt und in jeder dieser Regionen nach einem Linienverlauf gesucht. Diese Suche funktioniert derart, dass pro Region je ein Histogramm der Helligkeits- und der Sättigungswerte im HSV-Farbraum gebildet wird. Bei einem ausreichend großen Unterschied zwischen den beiden größten Histogrammklassen wird davon ausgegangen, dass die hellere der beiden Klassen die Farbe einer Straßenmarkierung und die andere die Farbe des Hintergrunds ist. Nun kann die den Verlauf der Straßenmarkierung am besten beschreibende Linie durch die jeweilige Region bestimmt werden, indem der Schwerpunkt und die Richtung der größten Varianz aller Vordergrundpixel berechnet werden.
Nachdem nun eine Menge von Linienpunkten mit Ausrichtungen vorliegt, werden zu diesen um die Bewegung des Fahrzeugs korrigierte Linienpunkte hinzugefügt, die in einem bestimmten Zeitraum in vorherigen Kamerabildern gefunden wurden. Abschließend werden dann mit dem RANSAC-Algorithmus abschnittsweise zu den Linienpunkten passende Linien berechnet, die somit die erkannten Spurmarkierungen beschreiben.
Das hier verwendete Verfahren zur Erkennung von auf Markierungen befindlichen Punkten ist sehr interessant, aber für die Problemstellung in dieser Arbeit vermutlich nicht anwendbar, da es durch die aufwändigen Berechnungen pro Region im Bild zwar sehr gut für die Ausführung auf auf parallele Datenverarbeitung spezialisierten Plattformen wie GPUs geeignet ist, aber bei sequenzieller Ausführung auf der CPU zum Beispiel eines *NAO* sehr teuer ist.

Poggenhans, Hellmund und Stiller (2016) beschreiben im Gegensatz zu den genannten Erkennungsverfahren eine Methode, um verschiedene gesehene Linienabschnitte zu einem Markierungsverlauf zu kombinieren. Dies ist im Bereich des autonomen Fahrens insbesondere relevant, wenn gestrichelte Fahrbahnmarkierungen erkannt werden sollen. Hierzu schlagen sie vor, die einzelnen Abschnitte basierend auf ihrer paarweisen Entfernung zueinander gemäß des jeweils vorliegenden Linienmodells zu Clustern zusammenzufassen. Anschließend kann dann jedes Cluster als Graph betrachtet werden, dessen Knoten Liniensegmente sind und dessen

Kanten mit der Entfernung zwischen diesen Segmenten gewichtet sind. Der die Segmente jedes Clusters am besten beschreibende Spurverlauf wird dann durch die Berechnung des minimal aufspannenden Baums erhalten.

Insgesamt zeigt sich, dass im Anwendungsbereich des autonomen Fahrens die IPM ein sehr gern angewendetes Verfahren ist, um Ausgangsdaten für die Bildverarbeitung im Hinblick auf Bodenmerkmale zu erhalten.

2.3 Deep Learning im RoboCup

Vor allem innerhalb der vergangenen zwei Jahre gab es im RoboCup viele Entwicklungen in Richtung der Nutzung von Deep Learning, also tiefen neuronalen Netzen, zur Lösung verschiedener Bildverarbeitungsprobleme. Nachfolgend sind einige der daraus resultierenden Veröffentlichungen zu diesem Thema genannt.

2.3.1 Klassifikation extrahierter Bildbereiche

Einer der ersten Ansätze stammt von **Albani u. a. (2017)**, deren Ziel die Erkennung anderer *NAO*-Roboter in der RoboCup SPL in Kamerabildern war. Hierzu extrahierten sie Bildbereiche, in denen sich jeweils möglicherweise ein Roboter befindet, und ließen durch ein trainiertes neuronales Netz bewerten, ob der entsprechende Bereich tatsächlich ein Roboter befindet. Das zur Klassifikation verwendete Netz hat dabei eine sehr einfache Architektur: es besteht aus nur zwei Convolutional-Layern mit jeweils anschließendem Pooling und zwei Fully-Connected-Layern, von denen der zweite zwei Ausgaben hat, die eine Wahrscheinlichkeitsverteilung über „ja“ und „nein“ bilden.

Das von **Leiva u. a. (2019)** vorgestellte Verfahren zur Erkennung von Robotern und Bällen in der RoboCup SPL funktioniert hierzu recht ähnlich. Auch hierbei werden zunächst Bildbereiche mit Kandidaten separat für Roboter und Bälle aus dem Kamerabild extrahiert und jeweils von einem Convolutional Neural Network (CNN, vgl. Kapitel 5.1) klassifiziert. Allerdings funktioniert die Ballerkennung hierbei zweistufig durch mehrere aufeinanderfolgende neuronale Netze, während auf die Erkennung von Robotern noch eine Bestimmung ihrer Orientierung folgt. Dazu werden analog zum von Mühlenbrock und Laue (2018) vorgestellten Ansatz die Konturen an den Füßen des Roboters gesucht, um aus diesen die Orientierung des Roboters zu bestimmen. Dies erfolgt aber im Gegensatz zum ursprünglichen Verfahren nicht durch eine Heuristik, sondern durch zwei kaskadierte neuronale Netze. Dadurch erzielen sie insgesamt bessere Ergebnisse. Die Architektur sämtlicher beim genannten Verfahren verwendeter neuronaler Netze ist dabei vom SqueezeNet (siehe Iandola u. a., 2016) inspiriert; sie sind dabei aber vergleichsweise wenig komplex – die fünf verwendeten Netze haben insgesamt unter 2 500 gelernte Parameter, während in vielen Anwendungen neuronaler Netze in der Bildverarbeitung Zehn- bis Hunderttausende oder gar Millionen Parameter üblich sind.

2.3.2 Semantische Segmentierung

Einen anderen Ansatz zur Nutzung neuronaler Netze im RoboCup beschreiben **van Dijk und Scheunemann (2019)**. Statt nur zuvor im Bild gefundene Kandidaten zu evaluieren, führt das von ihnen entwickelte CNN eine semantische Segmentierung des Kamerabilds durch, das heißt die Ausgabe des Netzes ist ebenfalls ein Bild, dessen Pixel als Wert allerdings jeweils eine Klasse haben, der sie zugehören. In diesem Fall sind die verwendeten Klassen Ball, Fußpunkt eines Torpfostens und Hintergrund. Die verwendete Netzarchitektur ist wie auch in den vorangegangenen Arbeiten einfach gehalten. Es handelt sich dabei um eine dem SegNet (siehe Badrinarayanan, Kendall und Cipolla, 2017) ähnliche Architektur der Tiefe Drei.

Houliston und Chalup (2019) entwickelten dieses Verfahren weiter, indem sie ein CNN zur semantischen Segmentierung des Bilds nach den Klassen Ball beziehungsweise nicht Ball trainierten, das als Eingabe nicht direkt das Kamerabild des Roboters erhält, sondern ein sogenanntes Visual Mesh. Dieses wird durch einen Graphen beschrieben, dessen Knoten alle den Grad 6 haben und jeweils Punkten auf der Bodenebene entsprechen. Jeder dieser Punkte hat dabei einen gleichmäßigen Winkelabstand von den ihm adjazenten Punkten relativ zur Kameraposition entsprechend der Größe des zu erkennenden Balls. Dies führt dazu, dass der Ball unabhängig von seiner Position relativ zum Roboter im Visual Mesh stets die gleiche Größe, Form und Textur hat, sodass das Training des neuronalen Netzes theoretisch besser funktionieren sollte. Dies wurde auch experimentell im Vergleich mit der Nutzung anderer CNNs auf dem Kamerabild bestätigt; auf beliebige Distanzen liefert das mit dem Visual Mesh trainierte Netz sehr gute Ergebnisse, wobei die Laufzeit zur Inferenz relativ gering ist.

Schnekenburger u. a. (2017) weiteten die Idee der semantischen Segmentierung aus, indem sie versuchten, mit einem einzigen neuronalen Netz direkt Informationen über alle für sie relevanten Objekte und Bodenmerkmale in einem Kamerabild zu erhalten. Die von ihnen genutzten Klassen sind daher Fußpunkt eines Torpfostens, L-Kreuzung, T-Kreuzung, X-Kreuzung, Strafstoßpunkt, Ball, Fußpunkt eines Roboters und Fußpunkt eines sonstigen Hindernisses. Eine Besonderheit dabei ist, dass das genutzte Netz keine Wahrscheinlichkeitsverteilung über diese Klassen lernt, sondern als Ausgabe pro Klasse eine Heatmap mit Werten im Bereich [0,1] liefert. Entsprechend wurde hierbei zum Training als Lossfunktion die mittlere quadratische Abweichung der aktuellen zur gewünschten Ausgabe genutzt. Mit diesem Ansatz wurden allgemein gute Ergebnisse erzielt, nur die Erkennung anderer Roboter war eher schlecht.
In Bezug auf das in dieser Arbeit zu lösende Problem ist dieses Verfahren grundsätzlich gut geeignet, allerdings fehlt zu allen erkannten Spielfeldmerkmalen jegliche Information über ihre Orientierung, welche zur Selbstlokalisierung des Roboters sehr hilfreich ist.

2.3.3 Bounding-Box-Detektion

Neben den vorgestellten Anwendungen von neuronalen Netzen wurden kürzlich auch verschiedene Ansätze zur Nutzung von Detektionsnetzen präsentiert, die für zu erkennende Objekte

in einem Bild jeweils eine Menge von minimal umschließenden Rechtecken (Bounding Boxes) liefern. **Szemenyei und Estivill-Castro (to appear)** nutzten diesen Grundsatz, um auf Basis von Tiny YOLOv3 – einer vereinfachten Variante der Architektur YOLOv3 (Redmon und Farhadi, 2018) – Bälle, Feldlinienkreuzungen, Roboter und Torpfosten in Eingabebildern der Größe 512×384 Pixel zu finden. Die verwendete Netzarchitektur wurde dabei zunächst sehr komplex mit relativ vielen Filtern pro Convolutional-Layer gewählt und anschließend beim Training durch sogenanntes Pruning reduziert, indem die für die Detektion unwichtigsten gelernten Parameter schrittweise entfernt wurden. Damit das Training trotz relativ weniger annotierter Bilder im Eingabedatensatz gut funktioniert, wurde sich dazu entschlossen, einen Teil der Trainingsdaten synthetisch zu generieren. Das resultierende neuronale Netz lieferte in den durchgeführten Experimenten gute Ergebnisse, jedoch ist seine Inferenz auf dem verwendeten *NAO* mit über 400 ms pro Bild sehr langsam, sodass nur etwa zwei Bilder pro Sekunde verarbeitet werden können.
Ein weiteres Problem des trainierten Netzes im Hinblick auf die Erkennung von Feldmerkmalen ist, dass die erkannten Feldlinienkreuzungen nicht in verschiedene Arten von Kreuzungen differenziert werden, sodass die Information, dass eine Kreuzung vorliegt, weniger Wert für die Selbstlokalisierung des Roboters hat.

Von **Poppinga und Laue (to appear)** hingegen wurde eine verwandte, ebenfalls grundlegend auf YOLOv3 basierende, Netzarchitektur mit einer deutlich schnelleren Ausführungszeit vorgestellt – sie ist vollständig echtzeitfähig, das heißt mit ihr können auf dem *NAO* sämtliche zur Verfügung stehende Kamerabilder verarbeitet werden. Dazu präsentierten sie eine Anwendung dieser Architektur für ein Detektionsnetz, das Roboter in Graustufenbildern der Größe 80×60 Pixel findet. Die Netzarchitektur nutzt dabei separierbare Convolutional-Layer, die schneller in der Ausführung sind und weniger gelernte Parameter haben als gewöhnliche Convolutional-Layer. Darüber hinaus wird auch hier beim Training Pruning eingesetzt, um so die Parameteranzahl noch weiter zu reduzieren. Weiterhin verwendet auch das hier vorgestellte Verfahren synthetische Daten zur Verbesserung des Trainings, jedoch wurde mit diesen wie in einem von Goodfellow u. a. (2014) vorgestellten Generative Adversarial Network gleichzeitig ein Diskriminator-Netz trainiert, welches reale von synthetischen Daten unterscheiden soll. Dies führt dazu, dass die Unterschiede zwischen realen und simulierten Daten weniger Einfluss auf das resultierende Modell haben.

Kapitel 3

Inverse Perspective Mapping

Sämtliche durch das in dieser Arbeit entwickelte Bildverarbeitungssystem zu erkennende Umgebungsmerkmale befinden sich auf der Bodenebene. Ein für derartige Aufgabenstellungen häufig verwendetes Verfahren ist das Generieren eines Bilds mithilfe von Inverse Perspective Mapping (IPM), wie es ursprünglich von Mallot u. a. (1991) als Vorverarbeitung zur Berechnung des optischen Flusses vorgestellt wurde. Hierbei wird aus einem Kamerabild ein Bild eines Ausschnitts der Bodenebene berechnet, welches dann den Anschein hat, als wäre es von oben herab aus der Vogelperspektive aufgenommen. IPM hat für die weitere Bildverarbeitung den Vorteil, dass Bodenmerkmale unabhängig von der Kameraperspektive und ihrer Position im Bild stets die gleiche Form haben.

Grundsätzlich bezeichnet Inverse Perspective Mapping eine Abbildung, die Punkte im Kamerabild auf Punkte auf der Bodenebene im Raum abbildet (vgl. Mallot u. a., 1991). Sie ist damit die Umkehrung der perspektivischen Abbildung des Kameramodells, welche jedem Punkt im Raum einen Punkt im Kamerabild zuordnet, mit der Einschränkung, dass die IPM nur auf Punkte auf der Bodenebene abbildet.

Das IPM-Bild bezeichnet dann das Bild, das erhalten wird, indem jeder Bildpunkt, für den eine solche Abbildung definiert ist, mit der IPM auf den entsprechenden Punkt auf der Bodenebene abgebildet wird. Die Abbildung ist dabei für einen Bildpunkt genau dann definiert, wenn dieser unterhalb des Horizonts liegt und somit sein zugehöriger Punkt im Raum auf der Bodenebene liegen kann.

Abb. 3.1 zeigt ein IPM-Bild zu einem Kamerabild, das von einem *NAO* während eines Spiels bei der RoboCup Weltmeisterschaft 2019 aufgenommen wurde. Hierbei fällt insbesondere auf, dass im Gegensatz zum Kamerabild im IPM-Bild Linien an Linienkreuzungen im rechten Winkel zueinander stehen und der Mittelkreis kreisförmig statt nur ellipsoid aussieht. Daher liegt es nahe, dass die Vorverarbeitung durch IPM sowohl für eine Linienmerkmalserkennung in der RoboCup SPL allgemein als auch im speziellen für eine Verarbeitung durch neuronale Netze von Vorteil ist. Bei der weiteren Verwendung des IPM-Bilds ist allerdings zu beachten, dass nicht alle Pixel des IPM-Bilds gültige Informationen enthalten, da ein Teil von ihnen

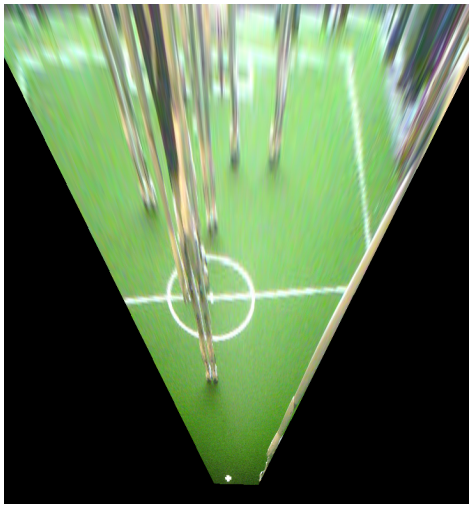



Abbildung 3.1 Links: Von der oberen Kamera eines *NAO* aufgenommenes Bild.
Rechts: Aus dem Kamerabild per Inverse Perspective Mapping erhaltenes Bild.

Punkten außerhalb des Kamerabilds entspricht. In dieser Arbeit werden die entsprechenden Pixel im Bild schwarz gezeichnet.

Praktisch kann das IPM-Bild berechnet werden, indem zunächst der Bereich der Bodenebene, der in ihm erhalten sein soll, sowie die Bildauflösung festgelegt wird. Anschließend wird mithilfe der perspektivischen Abbildung jedem Bildpunkt des IPM-Bilds ein Bildpunkt im Kamerabild zugeordnet.

3.1 Koordinatensysteme im B-Human-System

Für die nachfolgend beschriebenen Berechnungen zur Durchführung der Inverse Perspective Mapping sowie die weiteren Berechnungen von Positionen relativ zum Roboter im Rest dieser Arbeit ist die Definition einiger Koordinatensysteme vonnöten. Diese werden dabei dem Kontext der Arbeit entsprechend analog zu den im B-Human-System verwendeten Koordinatensystemen festgelegt (vgl. Röfer, Laue, Bülter u. a., 2017).

Die für diese Arbeit relevanten Koordinatensysteme sind das zur Roboterposition relative Feldkoordinatensystem sowie das Kamerabildkoordinatensystem. Ersteres ist wie in Abb. 3.2 gezeigt ein rechtshändiges dreidimensionales Koordinatensystem, wobei der Ursprung auf der Bodenebene genau unterhalb des Mittelpunkts der Hüfte des Roboters liegt. Die x-Achse verläuft dann vorwärts entlang der Ausrichtung des Roboters, während die y-Achse vom Roboter aus nach links zeigt. Die z-Achse verläuft dann entsprechend senkrecht zur Bodenebene nach oben.

Im in Abb. 3.3 zweidimensionalen Koordinatensystem des Kamerabilds hingegen liegt der Ursprung in der oberen linken Ecke des Bilds, wobei die x-Achse nach rechts und die y-Achse nach unten verläuft.

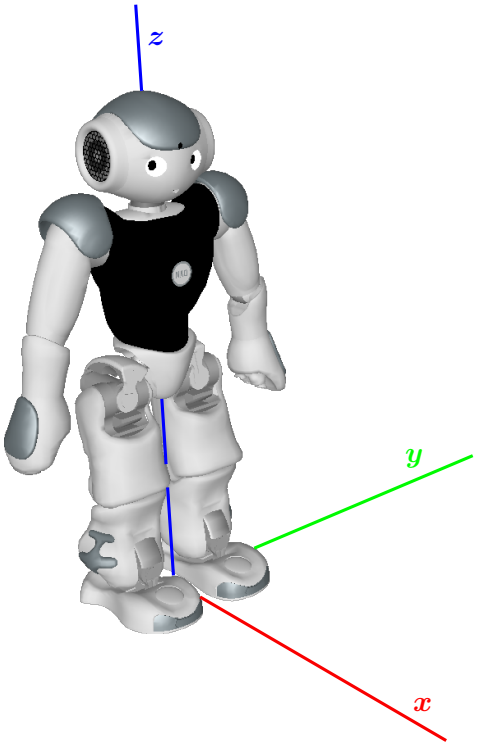


Abbildung 3.2 Veranschaulichung des in dieser Arbeit verwendeten Feldkoordinatensystems.

Abbildung 3.3 Veranschaulichung des in dieser Arbeit verwendeten Kamerabildkoordinatensystem.

3.2 Projektion von Koordinaten auf der Bodenebene ins Bild

Im B-Human-System ist die 3D-Pose jeder Kamera bekannt – diese besteht aus ihrer jeweiligen Position und Orientierung relativ zum Mittelpunkt der Füße des Roboters auf der Bodenebene. Dies ist möglich, da die Position und Orientierung der Kamera im Roboter sowie die Positionen und Orientierungen aller Gelenke zueinander bekannt sind und so die Kamerapose aus der kinematischen Kette bestimmt werden kann.

Angenommen, die Pose der Kamera im Feldkoordinatensystem liegt in Form der Rotationsmatrix R und dem Translationsvektor $\mathbf{t}$ vor. Dann beschreibt Formel 3.1 die Transformation von Koordinaten relativ zur Kamera $\mathbf{a}'$ zu Koordinaten im Feldkoordinatensystem $\mathbf{a}$. Entsprechend zeigt Formel 3.2 die inverse Transformation dazu, welche zu Koordinaten im Feldkoordinatensystem die Koordinaten des entsprechenden Punkts relativ zur Kamera ermittelt.

Zur Berechnung des IPM-Bilds liegt nun ein Punkt $\mathbf{p} = \begin{pmatrix} p_x & p_y & 0 \end{pmatrix}^T$ auf der Bodenebene relativ zur Ausrichtung der Kamera vor und gesucht ist der dazugehörige Punkt im Kamerabild p_i. Hierzu werden zunächst wie in Formel 3.3 gezeigt mithilfe der genannten Transformation die Koordinaten $\mathbf{p}'$ des entsprechenden Punkts relativ zur Kamera ermittelt, wobei $\mathbf{p}$ zuvor durch Drehung um die z-Achse um die Rotation der Kamera θ_z in das Feldkoordinatensystem überführt werden muss.

Um dabei ausgehend von R den Winkel θ_z zu ermitteln, der die Rotation der Kamera um die z-Achse im Feldkoordinatensystem beschreibt, wird in Formel 3.4 und Formel 3.5 die von Day (2014) vorgestellte Methode angewendet.

Schlussendlich kann der gesuchte Punkt im Kamerabild $\mathbf{p}_i$ wie in Formel 3.6 gezeigt durch eine Zentralprojektion (vgl. Duda und Hart, 1973) bestimmt werden. Hierbei ist $\begin{pmatrix} c_x & c_y \end{pmatrix}^T$ das optische Zentrum der Kamera, während f_x und f_y ihre Brennweite jeweils multipliziert mit der horizontalen beziehungsweise vertikalen Bildgröße angeben.

$$\mathbf{a} = R \cdot \mathbf{a}' + \mathbf{t} \tag{3.1}$$

$$\mathbf{a}' = R^T \cdot (\mathbf{a} - \mathbf{t}) \tag{3.2}$$

$$\mathbf{p}' = R^T \cdot \left(\begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \mathbf{p} - \mathbf{t} \right) \tag{3.3}$$

$$\theta_x = \operatorname{arctan2}(R_{32}, R_{33}) \tag{3.4}$$

$$\theta_z = \operatorname{arctan2}\left(\sin(\theta_x)R_{13} - \cos(\theta_x)R_{12}, \cos(\theta_x)R_{22} - \sin(\theta_x)R_{23}\right) \tag{3.5}$$

$$\mathbf{p}_i = \begin{pmatrix} c_x & -f_x & 0 \\ c_y & 0 & -f_y \end{pmatrix} \cdot \frac{\mathbf{p}'}{\mathbf{p}'_x} \tag{3.6}$$

3.3 Kompensierung der Kamerabewegung

Die im *NAO* V6 verbauten Kameras bestehen aus einfachen CMOS Sensoren, bei denen der Rolling-Shutter-Effekt auftritt (siehe Softbank Robotics, o.D.). Dies bedeutet, dass die einzelnen Pixel des Kamerabilds stets zeilenweise nacheinander ausgelesen werden, während die anderen Pixel weiter belichtet werden, und führt dazu, dass bei Bewegungen der Kamera Verzerrungen im Bild entstehen.

Im Kontext des Roboterfußballs wurde dieser Effekt von Röfer (2008) beschrieben, der auch eine Möglichkeit zur Kompensation vorschlug, welche in dieser Form auch heute noch im B-Human-System genutzt wird. Hierbei werden zu Bildkoordinaten $\begin{pmatrix} x & y \end{pmatrix}^T$ die wahren, unverzerrten Koordinaten $\begin{pmatrix} x' & y' \end{pmatrix}^T$ wie in Formel 3.7 gezeigt abhängig von der Bildzeile und Belichtungsdauer bestimmt (vgl. Röfer, 2008). Dabei bezeichnen α und β die relativen Kipp- beziehungsweise Schwenkwinkel zwischen der aktuellen und der vorherigen Kameraausrichtung, Δt die Zeit, die seit der letzten Bildaufnahme vergangen ist, t_r die Zeit, die die Kamera benötigt, um ein Bild aufzunehmen und t_d die Verzögerung zwischen Aufnahme und Verarbeitung des Kamerabilds.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} c_x - f_x \tan\left(\arctan\frac{c_x - x}{f_x} - d\alpha\right) \\ c_y + f_y \tan\left(\arctan\frac{y - c_y}{f_y} - d\beta\right) \end{pmatrix} \quad \text{mit} \tag{3.7}$$
$$d = \frac{\frac{y}{\text{Bildhöhe}} \cdot t_r - (t_r + t_d)}{\Delta t}$$

Für die Kompensation der durch Kamerabewegung verursachten Verzerrung im IPM-Bild ist die Vorgehensweise jedoch umgekehrt: Durch die in Formel 3.6 gezeigte Transformation werden die unverzerrten Bildkoordinaten eines Punkts erhalten und gesucht sind die dazugehörigen verzerrten Koordinaten im tatsächlichen Kamerabild. Entsprechend wird also die Umkehrung von Formel 3.7 verwendet:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} c_x - f_x \tan\left(\arctan\frac{c_x - x'}{f_x} + d\alpha\right) \\ c_y + f_y \tan\left(\arctan\frac{y' - c_y}{f_y} + d\beta\right) \end{pmatrix} \tag{3.8}$$

Hierbei besteht allerdings das Problem, dass der Faktor d abhängig von der gesuchten verzerrten Koordinate y ist. Daher verwendet das B-Human-System in diesem Fall ein iteratives Verfahren, welches d zunächst mit y' statt y berechnet und dann in mehreren Iterationen mit dem jeweils durch Formel 3.8 bestimmten y neu berechnet.

3.4 Wahl der Auflösung

Die Wahl der Auflösung des in den weiteren Schritten zur Bodenmerkmalserkennung verwendeten IPM-Bilds stellt einen Kompromiss zwischen Genauigkeit und Ausführungszeit dar. Je

größer die Dimensionen des IPM-Bilds sind, desto länger dauert einerseits seine Berechnung und andererseits das Finden von Merkmalen in ihm.

Der übliche Bereich relativ zur Position der oberen Kamera eines *NAO*-Roboters, in der in der SPL Bodenmerkmale des Spielfelds gesehen werden können, ist etwa 8,60 Meter breit und lang. Demnach sollte das IPM den entsprechenden Bereich abdecken.

Da die Merkmale des Spielfelds in der Regel aus Feldlinien bestehen, muss der Abstand der Abtastung kleiner als deren Breite, also 5 cm, sein. Um idealerweise alle Merkmale im IPM-Bild korrekt abbilden zu können, sollte gemäß des Abtasttheorems (vgl. Nyquist, 1928) der Abstand der Abtastung sogar kleiner als die Hälfte dieser Länge sein. Damit würde man ein IPM-Bild von circa 360×360 Pixeln erhalten.

In der Praxis zeigt sich jedoch, dass bei üblichen Kamerabildern eine geringere Abtastrate vollkommen ausreicht, um sämtliche Feldmerkmale im Bild eindeutig zu erkennen. Daher wurde der Abtastabstand zum Sparen von Rechenzeit stark erhöht auf 4,50 cm, wodurch sich ein IPM-Bild der Auflösung 192×192 Pixel ergibt. Dieses Bild wird in den folgenden Kapiteln als Grundlage für die Bodenmerkmalserkennung genutzt.

Kapitel 4

Analytische Merkmalserkennung

Um das Ziel dieser Arbeit zu erreichen, ein Bildverarbeitungssystem zur Merkmalserkennung zu entwickeln, welches das von B-Human bisher verwendete System durch bessere Nutzung der zur Verfügung stehenden Ressourcen übertrifft, wird zunächst ein analytischer Ansatz vorgestellt.

Dieser orientiert sich in seiner Grundidee am bisherigen Verfahren und nutzt einige der bisherigen Verarbeitungsschritte, verwendet allerdings als Grundlage nicht direkt die Kamerabilder des *NAO*, sondern die daraus entsprechend dem im vorangegangenen Kapitel beschriebenen Verfahren berechneten IPM-Bilder.

4.1 Bisheriger Ansatz bei B-Human

Die bestehende Methode zur Erkennung von Feldmerkmalen im B-Human-System besteht, wie von Röfer, Laue, Bülter u. a. (2017) beschrieben, aus mehreren Schritten, die im Folgenden genannt werden:

1. Klassifikation jedes Pixels des Eingabebilds in eine der Farbklassen Weiß, Schwarz, Feldfarbe und Andere.
2. Findung von horizontalen und vertikalen Regionen gleicher Farbe im Bild durch entsprechend horizontales beziehungsweise vertikales Scannen.
3. Erkennung der Spielfeldgrenze zur Einschränkung der Regionen auf den Inhalt des Felds.
4. Kombination adjazenter weißer Regionen zu Liniensegmenten.
5. Projektion der Liniensegmente von Bild- zu Feldkoordinaten.
6. Bildung von Linien durch Ausgleichsrechnung der Koordinaten der Liniensegmente, wobei die Rückprojektion der Linie ins Bild jeweils zu einem bestimmten Anteil weiß sein muss.

7. Bestimmung von Mittelkreiskandidaten durch Bildung von Clustern der Mittelpunkte von gebogenen Liniensegmenten.
8. Berechnung der Position des tatsächlichen Mittelkreises durch Ausgleichsrechnung der Punkte auf dem Kreis des größten Clusters.
9. Berechnung von Kreuzungen der berechneten Linien.
10. Erkennung des Strafstoßpunkts im Bild.
11. Bestimmung von Position und Rotation der Feldmerkmale Strafraum, Mittelkreis, Mittelkreuzung und Außenecke durch Kombination der zuvor erkannten Merkmale.

4.2 Neuer Ansatz

Der in den nachfolgenden Unterabschnitten beschriebene neu entwickelte Ansatz nutzt den Großteil der in Kapitel 4.1 beschriebenen bestehenden Schritte und tauscht lediglich den Teil der Linien- und Kreiserkennung aus, sodass dieser auf Basis des IPM-Bilds arbeitet.

Hierbei sind sämtliche in den nachfolgenden Abschnitten genannten Parameter der verwendeten Algorithmen als Größen auf dem Spielfeld definiert, sodass die Verfahren unabhängig von der Auflösung des IPM-Bilds sind. Dadurch kann zu einem späteren Zeitpunkt entschieden werden, vom in Kapitel 3.4 gewählten Abtastabstand auf dem Feld abzuweichen, um zum Beispiel durch eine Erhöhung der Auflösung mehr Genauigkeit bei größerem Rechenaufwand zu erhalten, ohne dass die hier implementierten Algorithmen oder ihre Parameter dafür angepasst werden müssten.

4.2.1 Berechnung des IPM-Bilds

Da der neue Ansatz in seiner Grundidee konsistent zum bisherigen sein soll, verwendet auch er eine vorgegebene Klassifizierung der Farben des Eingabebilds zu Farbklassen. Entsprechend enthält auch das dem Algorithmus zugrundeliegende IPM-Bild keine Helligkeitsinformationen, sondern Farbklassen.

Um dies umzusetzen, wären zwei Herangehensweisen möglich. Einerseits kann erst jeder Pixel des Eingabebilds einer Farbklasse zugeordnet werden und im zweiten Schritt zur Berechnung des IPM-Bilds pro Pixel diejenige Farbklasse gewählt werden, die zum Pixel im farbklassifizierten Bild gehört, dessen Koordinaten den projizierten Bildkoordinaten am nächsten sind. Die andere Möglichkeit wäre, zunächst das IPM-Bild mit vollen Farbinformationen aus dem Kamerabild zu berechnen und anschließend die Farben der Pixel des erhaltenen Bilds zu Farbklassen zu klassifizieren.

Dabei hat die zweite Option einerseits den Vorteil, dass bei der Berechnung des IPM-Bilds die Farben der um die projizierten Koordinaten liegenden Pixel interpoliert werden können,

um eine insgesamt genauere Farbklassifizierung zu erhalten. Da die Anzahl der Pixel im IPM-Bild außerdem geringer als die des gesamten Kamerabildes ist, wäre bei der zweiten Option der Schritt der Farbklassifizierung überdies performanter.

Allerdings sind in der gegebenen Umgebung beide Vorteile nicht relevant: Einerseits ist eine zum Beispiel bilineare Interpolation der Pixel bei der Berechnung des IPM-Bilds nicht vorgesehen, da dies pro Ausgabepixel viermal so viele Speicherzugriffe erfordern würde und damit deutlich mehr Laufzeit in Anspruch nähme, ohne dabei einen großen Vorteil zu bieten. Andererseits wird im B-Human-System das vollständig farbklassifizierte Bild ohnehin für weitere Aufgaben verwendet – etwa zum Finden von potentiellen Bällen –, sodass im Gesamtsystem betrachtet die Farbklassifizierung eines zuvor projizierten IPM-Bilds mit Farbinformationen keinen Performanzvorteil bringt, sondern im Gegenteil noch zusätzlichen Aufwand bedeutet.

Das resultierende farbklassifizierte IPM-Bild zeigt Abb. 4.1.

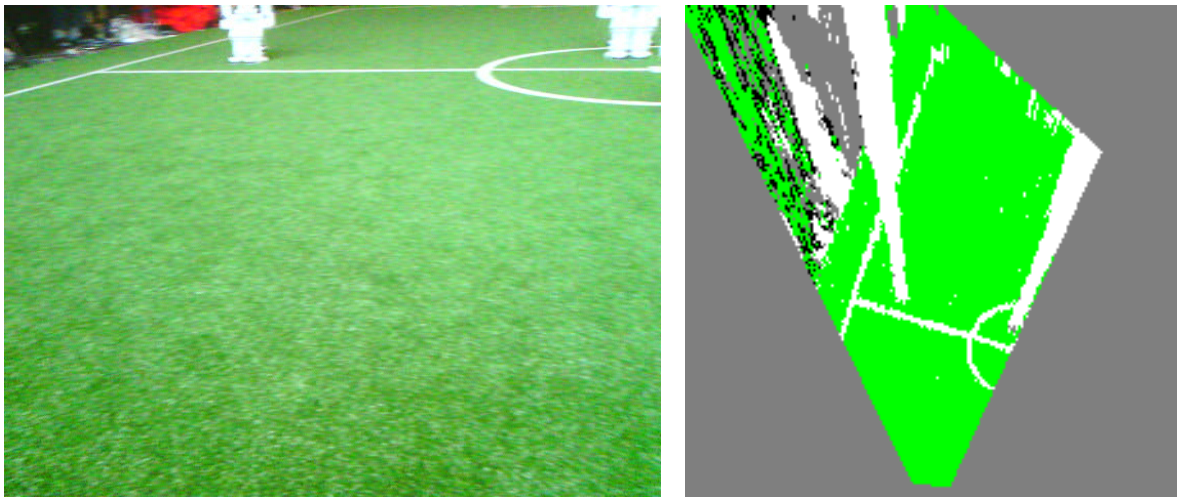


Abbildung 4.1 Links: Von der oberen Kamera eines *NAO* aufgenommenes Bild.
Rechts: Daraus berechnetes farbklassifiziertes IPM-Bild, das in den folgenden Schritten zur Merkmalserkennung verwendetet werden soll.

4.2.2 Bildung von Feldlinienketten

Dem bisherigen Ansatz entsprechend beginnt auch der neue mit dem Finden potentieller Linienabschnitte durch das horizontale und vertikale Abtasten des Bilds. Hierbei werden jeweils Zeilen beziehungsweise Spalten mit einer bestimmten Entfernung voneinander zur Abtastung gewählt. Diese sind in Abb. 4.2 veranschaulicht und werden im Folgenden als Scanlines bezeichnet.

Pro Scanline werden dabei Folgen von als weiß klassifizierten Pixeln gesucht, die von als feldfarben klassifizierten Pixeln umrandet sind, wobei die Folgen eine festgelegte Breite nicht überschreiten dürfen. Allerdings dürfen die Folgen von einer bestimmten Anzahl nicht weißer und nicht feldfarbener Pixel unterbrochen sein. Dies soll eine Erkennung von Linien auch

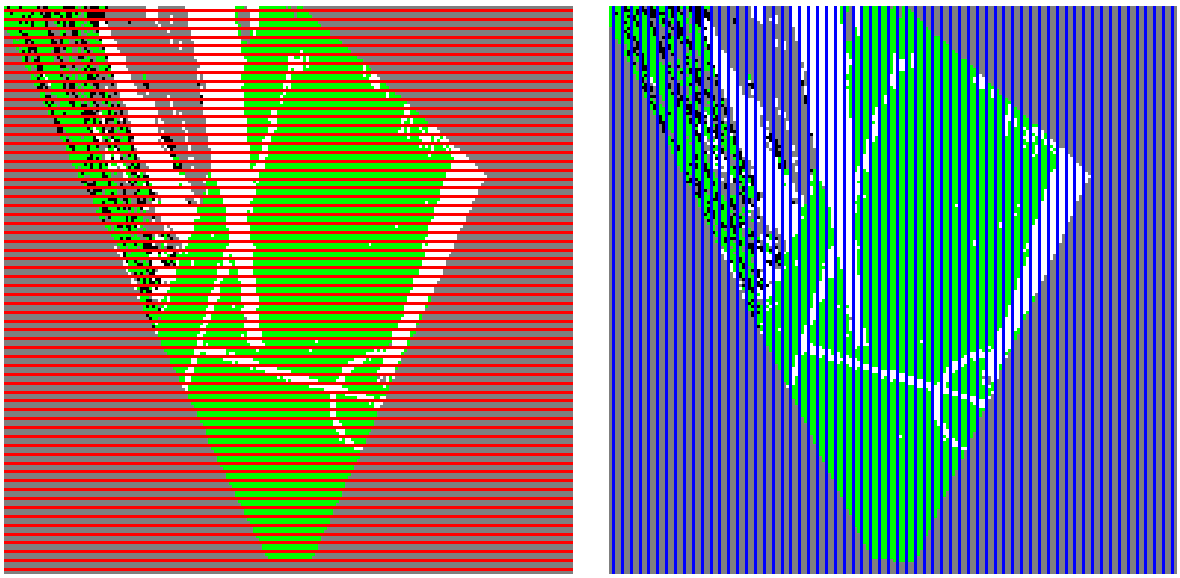


Abbildung 4.2 Horizontale (links) und vertikale Scanlines (rechts), entlang denen nach potentiellen Linienabschnitten gesucht wird.

dann ermöglichen, wenn Konturen im Bild fälschlicherweise als schwarz oder andersfarben klassifiziert werden.

Wurde eine Folge weißer Pixel gefunden, wird davon ausgegangen, dass sie einen Abschnitt einer senkrecht zur Scanline verlaufenden Feldlinie beschreibt. Entsprechend wird angenommen, dass das Zentrum dieses Abschnitts ein Punkt auf der Mitte der Feldlinie ist. Um sicherzustellen, dass der gefundene Punkt wirklich auf einer Feldlinie liegt, wird als nächstes geprüft, ob mindestens einer der senkrecht zur Scanline adjazenten Pixel dieses Feldlinienpunkts ebenfalls weiß ist. Dadurch kann ein Großteil der fälschlich in Bildrauschen gefundenen Linienpunkte ausgeschlossen werden. Abb. 4.3 zeigt die so detektierten Linienpunkte im Bild.

Nachdem nun pro Scanline eine Menge von Linienpunkten vorliegt, die jeweils auf Linien senkrecht zur Scanline liegen, wird im nächsten Schritt versucht, aus diesen Punkten Ketten in Richtung ihres Linienverlaufs zu bilden. Dazu wird jeweils pro Linienpunkt versucht, eine Verknüpfung mit Linienpunkten auf der vorherigen Scanline zu bilden. Um eine solche Verknüpfung durchzuführen, gibt es zwei Bedingungen: Zum Einen muss, um der Annahme des Linienverlaufs zu entsprechen, der Winkel zwischen den Punkten relativ zur Normalen der Scanline kleiner als 45° sein. Zum Anderen müssen die beiden Punkte im IPM-Bild ohne Unterbrechung durch weiße Pixel verbunden sein.

Bei der Bildung von Ketten kann es sowohl vorkommen, dass mehrere Linienpunkte mit demselben Punkt auf der vorherigen Scanline verknüpft werden, als auch dass ein Linienpunkt mit mehreren Punkten auf der vorherigen Scanline verknüpft wird. In beiden Fällen wird jeweils pro Verlauf der Kette eine separate Kette erstellt.

Somit liegen schlussendlich zwei in Abb. 4.4 visualisierte Listen von Linienketten vor: jeweils eine für Ketten, die horizontal beziehungsweise vertikal im IPM-Bild verlaufen.

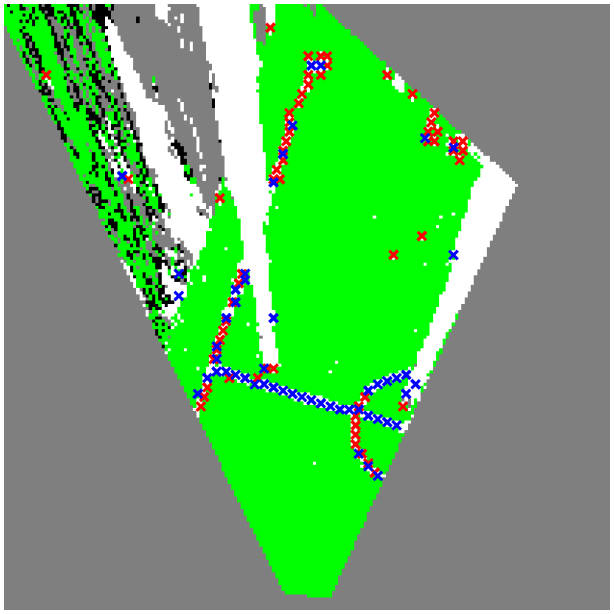


Abbildung 4.3 Durch Scannen des Bilds gefundene Linienpunkte. Beim horizontalen Scannen gefundene Punkte sind dabei rot, während die beim vertikalen Scannen gefundenen blau markiert sind.

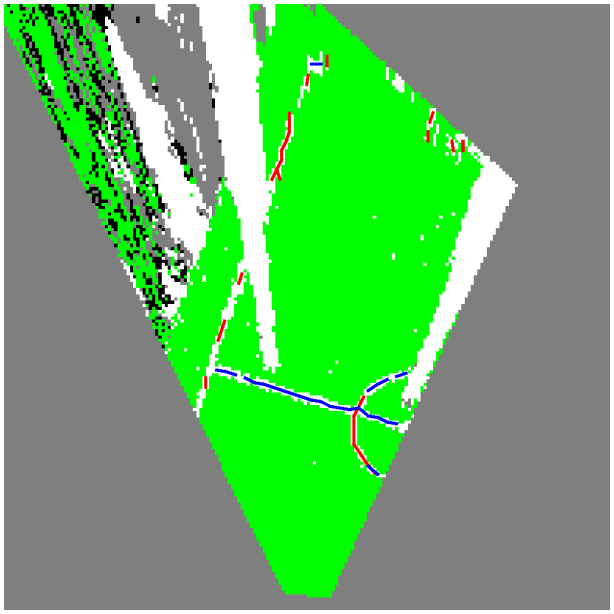


Abbildung 4.4 Resultierende Linienketten. Rote Ketten wurden durch horizontales Scannen gefunden und verlaufen daher vertikal, während blaue Ketten entsprechend umgekehrt durch vertikales Scannen gefunden wurden und daher horizontal verlaufen.

4.2.3 Feldlinienerkennung

Im nächsten Schritt wird nun versucht, aus den Ketten von Linienpunkten tatsächliche Feldlinien zu ermitteln.

Dazu werden erst einmal nur diejenigen Ketten betrachtet, die aus mindestens drei Punkten bestehen. Kurze Ketten aus nur zwei Punkten werden zunächst ausgeschlossen, da bei diesen die Wahrscheinlichkeit größer ist, dass sie durch Bildrauschen entstanden sind, sodass sie mit geringerer Wahrscheinlichkeit auf echten Feldlinien liegen.

Nun wird zuerst pro Kette versucht, sie mit einem existierenden Feldlinienkandidaten zu verbinden. Dies ist möglich, wenn einerseits die N Linienpunkte p_i der Kette zum in Hessescher Normalform als n_0 und d vorliegenden Linienmodell passen. Dieses Kriterium wird sichergestellt, indem die in Formel 4.1 gezeigte mittlere quadratische Abweichung (mean squared error, MSE) der Kette zum Modell einen festgelegten Schwellwert nicht überschreiten darf.

$$\mathrm{MSE}(p, n_0, d) = \frac{1}{N} \sum_{i=1}^{N} \left(\langle n_0, p_i \rangle - d \right)^2 \tag{4.1}$$

Ist dies der Fall, muss andererseits die durch Verbindung der Kette mit dem bestehenden Linienkandidaten im Bild erhaltene Strecke zu mindestens einem vorgegebenen Anteil aus als weiß klassifizierten Pixeln besteht. Dazu werden die Linienpixel allerdings nicht im IPM-Bild geprüft, sondern stattdessen die Linie ins Kamerabild zurückprojiziert, um die Farben von gleichmäßig im Kamerabild auf der Linie verteilten Pixeln zu prüfen. Dies wird deshalb getan, da sonst die geringe Auflösung des IPM-Bilds in vielen Fällen dazu führt, dass die geprüften Pixel leicht neben der Linie liegen und damit fälschlicherweise nicht weiß sind.

Konnte eine Kette nicht mit einem bestehenden Feldlinienkandidaten verbunden werden, wird aus ihr ein neuer Feldlinienkandidat erstellt. Hierzu wird zunächst das am besten zu den Linienpunkten passende Linienmodell durch Ausgleichsrechnung unter Nutzung der Methode der kleinsten Quadrate erhalten. Wie von Duda und Hart (1973, Seiten 332-335) gezeigt, kann diese Minimierung der quadratischen Abweichungen der Linienpunkte zum jeweils nächsten Punkt auf der Ausgleichslinie erfolgen, indem die 2×2-Kovarianzmatrix dieser Linienpunkte berechnet wird. Der Normalenvektor n_0 der Linie entspricht dann dem Eigenvektor dieser Matrix mit dem kleineren Eigenwert, während die Distanz d wie in Formel 4.2 gezeigt aus dem Wissen erhalten werden kann, dass der Mittelwert der Linienpunkte auf der Ausgleichslinie liegen muss.

$$d = \left\langle n_0, \frac{1}{N} \sum_{i=1}^{N} p_i \right\rangle \tag{4.2}$$

Um sicherzustellen, dass die Punkte der Kette wirklich zum erhaltenen Linienmodell passen, wird der neue Feldlinienkandidat nur gespeichert, wenn die mittlere quadratische Abweichung (vgl. Formel 4.1) der Linienpunkte zum Modell unterhalb eines festgelegten Schwellwerts liegt.

Nachdem nun eine Liste von Feldlinienkandidaten existiert, werden im nächsten Schritt auch kurze Ketten berücksichtigt, die aus nur zwei Punkten bestehen. Für diese wird jeweils versucht, sie mit bestehenden Feldlinienkandidaten zu verbinden, wobei jeweils das gleiche Verfahren wie zuvor genutzt wird. Das heißt, dass auch hierbei die mittlere quadratische Abweichung von ihnen zum Modell des Kandidaten unter einem festgelegten Schwellwert liegen muss und die verbundene Linie bei einer Rückprojektion ins Kamerabild zu einem gegebenen Anteil weiß sein muss.

Durch die vorangegangenen Schritte kann es vorkommen, dass noch immer mehrere Linienkandidaten auf derselben Feldlinie liegen. Um diesen Fall abzudecken, wird noch einmal versucht, alle gefundenen Linienkandidaten jeweils mit allen anderen zu verbinden. Dies erfolgt, wenn die folgenden drei Bedingungen erfüllt sind: Erstens müssen die Linien in dieselbe Richtung verlaufen, also die Differenz der Winkel ihrer Normalen unterhalb eines gegebenen Schwellwerts liegen. Zweitens müssen die Linien ungefähr kollinear sein, also die Winkeldifferenzen der Differenzen ihrer Linienendpunkte zum mittleren Winkel des Linienverlaufs unterhalb eines Schwellwerts sein. Drittens gilt auch hier wieder das Kriterium, dass die ins Kamerabild zurückprojizierte Linie zu mindestens einem gegebenen Anteil weiß sein muss. Abb. 4.6a zeigt die so gefundenen Feldlinien.

Prinzipbedingt treten im IPM-Bild wie in Abb. 4.5 gezeigt Fehlerkennungen von Linien insbesondere in Objekten auf, die nicht auf der Bodenebene liegen, also im Kontext des Roboterfußballs in Robotern, Torpfosten und dem Ball. Allerdings gilt für diese falsch erkannten Linien in der Regel, dass die Gerade durch sie auch durch die Kameraposition verläuft. Daher können sie leicht ausgefiltert werden, indem alle Linienkandidaten entfernt werden, deren Winkelunterschied zwischen ihrem Linienverlauf und dem Winkel der Kameraposition zu einem der Linienendpunkte einen festgelegten Schwellwert nicht überschreitet.

Nach all diesen Schritten verbleibt nun eine Liste von Feldlinienkandidaten, von denen zu einer hohen Wahrscheinlichkeit jede auf genau einer tatsächlichen Feldlinie liegt. Um nun noch die volle Länge der Feldlinien abzudecken, obwohl an manchen Stellen einer Feldlinie möglicherweise keine Linienketten gefunden wurden, wird als letzter Schritt wie auch schon im bisherigen System versucht, die Linien zu verlängern. Dazu werden die Endpunkte jeder Linie in das Kamerabild zurückprojiziert und die Linie in beide Richtungen entlang ihres Verlaufs verlängert, solange jeder Pixel auf dieser Strecke als weiß klassifiziert wurde. Die resultierenden Erkennungen sind in Abb. 4.6b zu sehen.

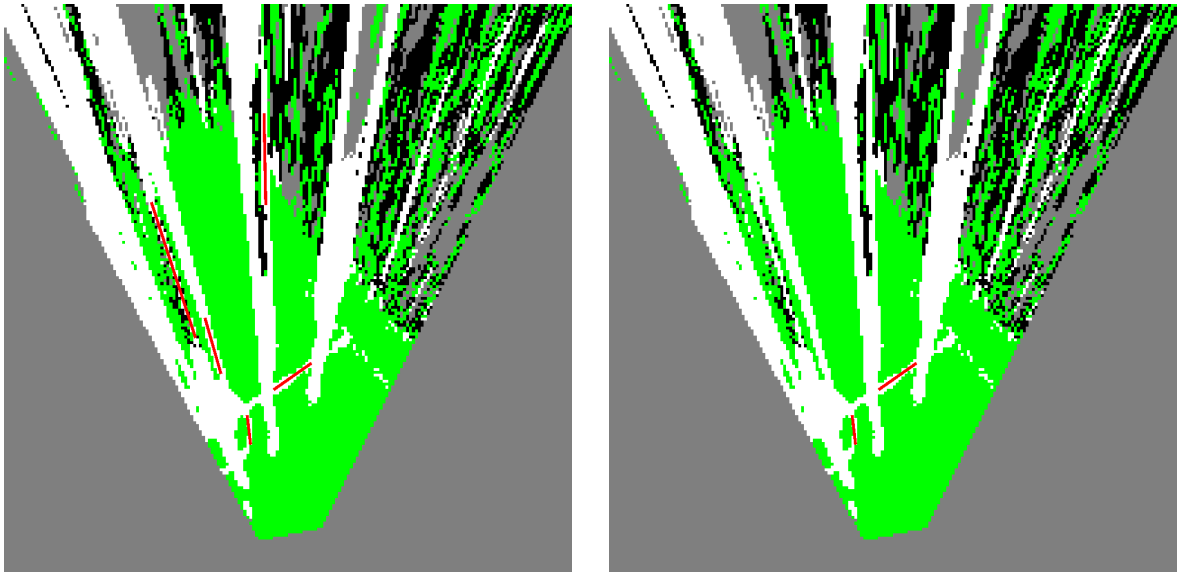
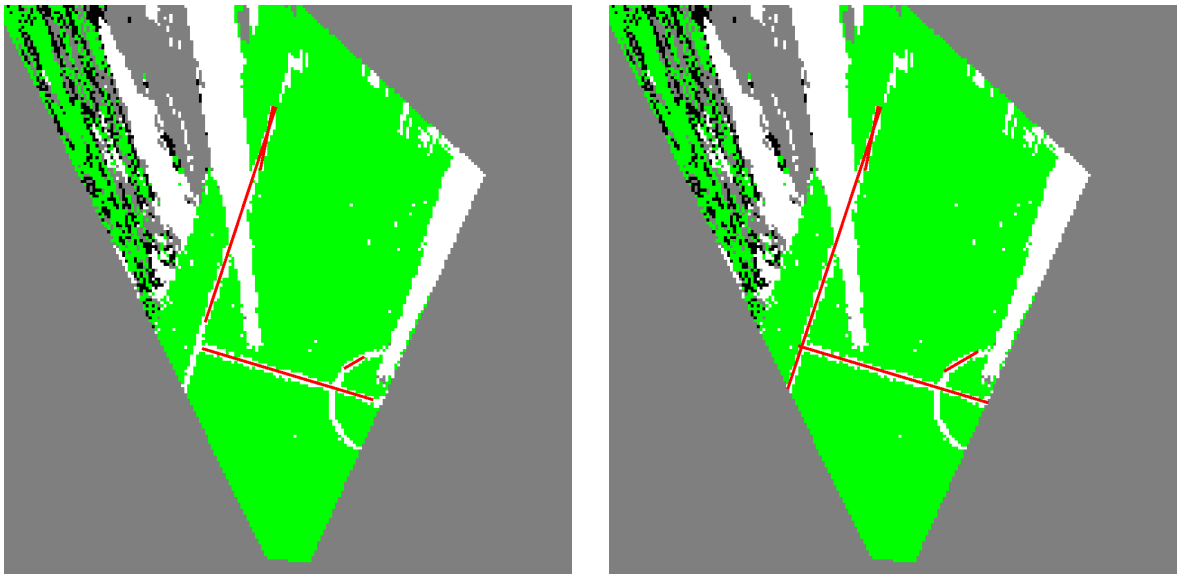


Abbildung 4.5 Links: Fehlerkennungen des Algorithmus zur Linienerkennung in Robotern. Rechts: Verbleibende Linien, nachdem die in Richtung der Kameraposition verlaufenden Linien ausgefiltert wurden.



(a) **(b)**

Abbildung 4.6 Mit dem beschriebenen Algorithmus gefundene Feldlinien bevor **(a)** und nachdem **(b)** sie durch Verfolgen ihres Verlaufs im Kamerabild verlängert wurden.

4.2.4 Mittelkreiserkennung

Neben der Feldlinienerkennung können die zuvor gefundenen Ketten von Linienpunkten auch genutzt werden, um den Mittelkreis des Spielfelds im IPM-Bild zu finden.

Hierzu wird ein der oben beschriebenen Erkennung von Feldlinien grundlegend ähnliches Verfahren verwendet. Eine zur Erkennung des Mittelkreises sehr hilfreiche Eigenschaft des IPM-Bilds ist, dass er in diesem Bild kreisförmig erscheint. Somit ist das Ziel des nachfolgend beschriebenen Algorithmus, Kreise in den Linienpunktketten zu finden. Da ein Kreis durch drei nicht kollineare Punkte beschrieben wird beziehungsweise aus mehr als drei Punkten per Ausgleichsrechnung bestimmt werden kann, werden auch bei diesem Verfahren zunächst nur Ketten aus mindestens drei Linienpunkten betrachtet.

Für jede dieser Ketten wird analog zur Feldlinienerkennung zunächst versucht, einen bestehenden Mittelkreiskandidaten zu finden, zu dem sie passen. Hierzu wird ebenfalls ein Schwellwert für die mittlere quadratische Abweichung der Linienpunkte zum durch Mittelpunkt c und Radius r beschriebenen Kreismodell festgelegt, die hierbei wie in Formel 4.3 gezeigt berechnet wird.

$$\mathrm{MSE}(p, c, r) = \frac{1}{N} \sum_{i=1}^{N} \left(|p_i - c| - r\right)^2 \tag{4.3}$$

Konnte eine Linienpunktkette nicht einem bestehenden Mittelkreiskandidaten zugeordnet werden, wird für sie ein neuer erstellt. Besteht die Kette aus genau drei Punkten p_1, p_2, p_3, so können die Parameter des Kreises wie in Abb. 4.7 gezeigt durch diese Punkte direkt bestimmt werden.

Hierzu werden entsprechend der von Bourke (1990) beschriebenen Methode zunächst die in Formel 4.4 und Formel 4.5 gezeigten Gleichungen der beiden Geraden durch p_1 und p_2 beziehungsweise p_2 und p_3 abhängig von der x-Koordinate c_x des Mittelpunkts aufgestellt. Deren Steigungen $m_a = \frac{p_{2_y} - p_{1_y}}{p_{2_x} - p_{1_x}}, m_b = \frac{p_{3_y} - p_{2_y}}{p_{3_x} - p_{2_x}}$ sind dabei durch die Definition der Strecken bekannt. Weiterhin lassen sich die in Formel 4.6 und Formel 4.7 gezeigten Gleichungen der beiden Mittelsenkrechten bestimmen.

$$y_a = m_a(c_x - p_{1_x}) + p_{1_y} \tag{4.4}$$

$$y_b = m_b(c_x - p_{2_x}) + p_{2_y} \tag{4.5}$$

$$y'_a = -\frac{1}{m_a}\left(c_x - \frac{p_{1_x} + p_{2_x}}{2}\right) + \frac{p_{1_y} + p_{2_y}}{2} \tag{4.6}$$

$$y'_b = -\frac{1}{m_b}\left(c_x - \frac{p_{2_x} + p_{3_x}}{2}\right) + \frac{p_{2_y} + p_{3_y}}{2} \tag{4.7}$$

Somit kann schließlich der Mittelpunkt des Kreises bestimmt werden, indem die Gleichungen der Mittelsenkrechten wie in Formel 4.8 gezeigt nach c_x aufgelöst werden und anschließend c_y

durch Einsetzen von c_x in die Gleichung einer der Mittelsenkrechten erhalten wird. Der Radius des Kreises entspricht dann dem Abstand eines der Kreispunkte zum Kreismittelpunkt, also zum Beispiel $r = |p_1 - c|$.

$$c_x = \frac{m_a m_b (p_{1_y} - p_{3_y}) + m_b (p_{1_x} + p_{2_x}) - m_a (p_{2_x} + p_{3_x})}{2(m_b - m_a)} \tag{4.8}$$

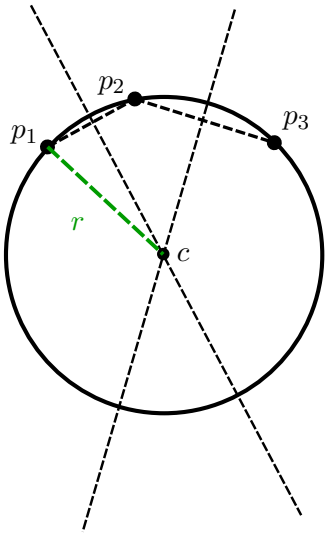


Abbildung 4.7 Berechnung eines Kreises mit den Parametern c, r aus drei Punkten p_1, p_2, p_3.

Sind hingegen mehr als drei Linienpunkte in der betrachteten Kette vorhanden, wird durch Ausgleichsrechnung der beste Kreis durch diese Punkte gefunden. Auch hierbei wird wieder die Methode der kleinsten Quadrate angewendet, wobei in diesem Fall die quadrierten Abstände der Punkte zum nächsten Punkt auf dem Kreisbogen minimiert werden.

Wie von Bullock (2006) gezeigt, können die Kreisparameter für diesen besten Kreis bestimmt werden, indem zunächst der Kreismittelpunkt relativ zum Mittelwert der Punkte $\bar{p}$ durch Lösung des in Formel 4.9 gezeigten Gleichungssystems bestimmt wird.

$$\begin{pmatrix} S_{xx} & S_{xy} \\ S_{xy} & S_{yy} \end{pmatrix} \bar{c} = \frac{1}{2} \begin{pmatrix} S_{xxx} + S_{xyy} \\ S_{yyy} + S_{yxx} \end{pmatrix} \quad \text{mit} \tag{4.9}$$

$$\bar{p} = \frac{1}{N} \sum_{i=1}^{N} p_i \qquad \bar{c} = c - \bar{p}$$

$$S_{xx} = \sum_{i=1}^{N} \left(p_{i_x} - \bar{p}_x\right)^2 \qquad S_{xy} = \sum_{i=1}^{N} \left(p_{i_x} - \bar{p}_x\right)\left(p_{i_y} - \bar{p}_y\right)$$

$$S_{yy} = \sum_{i=1}^{N} \left(p_{i_y} - \bar{p}_y\right)^2$$

$$S_{xxx} = \sum_{i=1}^{N} \left(p_{i_x} - \bar{p}_x\right)^3 \qquad S_{xyy} = \sum_{i=1}^{N} \left(p_{i_x} - \bar{p}_x\right)\left(p_{i_y} - \bar{p}_y\right)^2$$

$$S_{yyy} = \sum_{i=1}^{N} \left(p_{i_y} - \bar{p}_y\right)^3 \qquad S_{yxx} = \sum_{i=1}^{N} \left(p_{i_y} - \bar{p}_y\right)\left(p_{i_x} - \bar{p}_x\right)^2$$

Der echte Kreismittelpunkt kann dann einfach umgekehrt als $c = \bar{c} + \bar{p}$ erhalten werden, während der Radius des Kreises entsprechend der Methode von Bullock (2006) mittels Formel 4.10 bestimmt wird.

$$r = \sqrt{\langle \bar{c}, \bar{c} \rangle + \frac{S_{xx} + S_{yy}}{N}} \tag{4.10}$$

Der so durch Ausgleichsrechnung gefundene Kreis wird nur dann als Mittelkreiskandidat hinzugefügt, wenn einerseits die mittlere quadratische Abweichung (siehe Formel 4.3) der Kreispunkte zum gefundenen Modell nicht zu hoch ist und andererseits der ermittelte Radius nicht zu stark vom bekannten Radius des Mittelkreises auf dem Spielfeld abweicht.

Nachdem nun alle Ketten mit mehr als zwei Linienpunkten zu Mittelkreiskandidaten gewandelt wurden, werden im nächsten Schritt auch die kurzen Ketten berücksichtigt. Hierzu werden die Linienpunkte dieser kurzen Ketten zu den gefundenen Mittelkreiskandidaten hinzugefügt, sofern sie jeweils innerhalb einer festgelegten Distanz vom zugehörigen Kreisbogen liegen.

Die erhaltenen Kandidaten werden zuletzt noch validiert, indem eine Menge von gleichmäßig auf dem Kreisbogen verteilten Punkten in das Kamerabild zurückprojiziert wird und geprüft wird, ob mindestens ein festgelegter Anteil der entsprechenden Pixel im Bild als weiß klassifiziert wurde.

Aus den verbleibenden Kandidaten wird als Ergebnis derjenige Kandidat als echter Mittelkreis ausgewählt, dessen Radius am besten zum bekannten Radius des Mittelkreises auf dem Spielfeld passt. Abb. 4.8 zeigt die in einem IPM-Bild gefundenen Mittelkreise.

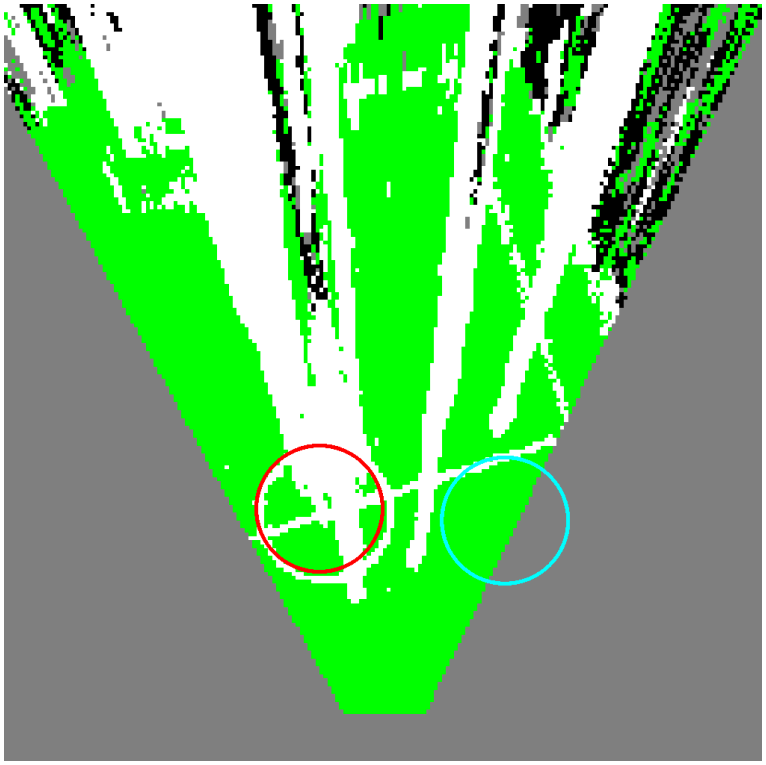


Abbildung 4.8 Gefundene Mittelkreise im IPM-Bild. Der rote Kreis wurde als echter Mittelkreis ausgewählt, da auf dem Kreisbogen des blauen nicht genügend weiße Pixel gefunden wurden.

Kapitel 5

Merkmalserkennung mit neuronalen Netzen

Im Gegensatz zum im vorigen Kapitel beschriebenen analytischen Ansatz zur Merkmalserkennung werden im Folgenden Methoden des maschinellen Lernens als weitere Lösungsansätze für das Problem des Erkennens von Bodenmerkmalen in IPM-Bildern angewendet.

5.1 Grundlagen neuronaler Netze

Das hierbei genutzte Verfahren aus dem maschinellen Lernen bilden neuronale Netze. Diese haben im Vergleich mit anderen Lernverfahren den insbesondere im Bereich der Bildverarbeitung relevanten Vorteil, dass sie einen Ende-zu-Ende-Ansatz bieten, also direkt die vorliegenden Bilddaten ohne einen vorherigen Merkmalsextraktionsschritt erhalten und das gewünschte Ergebnis ausgeben.

Künstliche neuronale Netze bestehen grundsätzlich aus einer aufeinanderfolgenden Reihe von Schichten – nachfolgend nur englisch als Layer bezeichnet –, die Operationen auf den als Eingabe erhaltenen Daten definieren.

Die einfachste Variante einer solchen Schicht ist der Fully-Connected-Layer. Dieser erhält als Eingabe einen Vektor $\mathbf{v}$ und definiert eine Gewichtsmatrix W und einen Bias-Vektor $\mathbf{b}$ sowie eine nichtlineare Aktivierungsfunktion σ (vgl. Kruse u. a., 2015), sodass seine Ausgabe wie folgt ist:

$$\sigma(W \cdot \mathbf{v} + \mathbf{b}) \tag{5.1}$$

Die Nichtlinearität durch die Aktivierungsfunktion erlaubt dem neuronalen Netz dabei, durch entsprechende Wahl der Gewichte und des Bias wie von Hornik (1991) gezeigt durch Kombination mehrerer Fully-Connected-Layer beliebige Funktionen approximieren zu können. Heutzutage wird dazu in der Regel die in Formel 5.2 gezeigte Aktivierungsfunktion ReLU (rectified linear unit) verwendet (vgl. Glorot, Bordes und Bengio, 2011).

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{für } x \geq 0 \\ 0 & \text{sonst} \end{cases} = \max(0, x) \tag{5.2}$$

Die Gewichte und der Bias jedes Layers bilden die lernbaren Parameter eines neuronalen Netzes, die zum Training des Netzes durch ein Optimierungsverfahren, in der Regel eine Variante des Gradientenabstiegs, hinsichtlich einer Fehlerfunktion – der sogenannten Lossfunktion – optimiert werden, welche den Unterschied der aktuellen zur gewünschten Ausgabe des Netzes misst.

Für die Anwendung in der Bildverarbeitung sind Layer von Vorteil, die im Gegensatz zu den Fully-Connected-Layern auf räumlichen Bildbereichen operieren, ohne dass dabei die Position im Bild relevant wäre. Hierzu wurden sogenannte Convolutional-Layer eingeführt, welche als Ein- und Ausgabe dreidimensionale Bilddaten erhalten (siehe LeCun u. a., 1989). Die Dimensionen beschreiben dann Höhe, Breite und Kanäle – bei Ein- und Ausgabedaten etwa Farbkanäle – des Bildes.

Die Anwendung eines Convolutional-Layers funktioniert dann derart, dass jeder mögliche Bildbereich einer festgelegten Größe aus dem Eingabebild extrahiert und sein Inhalt analog der oben genannten Anwendung des Fully-Connected-Layers mit einer Gewichtsmatrix, einem Bias-Vektor und einer Aktivierungsfunktion verrechnet wird, um schließlich je einen Vektor zu erhalten, der den neuen Inhalt der Kanäle des mittleren Pixels des Bildbereiches im Ausgabebild enthält. Die Operation ist also eine Erweiterung der zweidimensionalen mathematischen Faltungsoperation (vgl. Burger und Burge, 2015) um eine Kanaldimension: eine Kernel-Matrix wird auf das gesamte Eingabebild angewendet. In der Regel wird für die meisten Layer eine Kernelgröße von 3×3 Pixel verwendet.

Aus Convolutional-Layern bestehende neuronale Netze werden als Convolutional Neural Networks (CNNs) bezeichnet. Zusätzlich zu Convolutional-Layern enthalten diese häufig noch weitere Layer zur Änderung der Bilddimensionen. Pooling-Layer verkleinern dabei die Bilddimensionen, indem eine Fenstergröße von $m \times n$ Pixel definiert und ein entsprechendes Fenster dieser Größe über das Eingabebild verschoben wird, wobei jeweils alle Pixel innerhalb dieses Fensters zu einem Pixel im Ausgabebild zusammengefasst werden. Übliche Formen sind hierbei Max-Pooling, welches stets den größten Wert im Fenster auswählt, und Average-Pooling, bei dem jeweils der Durchschnitt aller Pixelwerte gebildet wird.

Im Gegensatz dazu werden durch Upscaling-Layer die Bilddimensionen um einen bestimmten Faktor in x- und y-Richtung vergrößert. Während dabei eine Skalierung mit bilinearer Interpolation möglich wäre, werden in den meisten Implementierungen lediglich die Inhalte der Pixel des Bildes entsprechend oft in die jeweiligen Richtungen dupliziert – so auch in der in dieser Arbeit verwendeten Implementierung.

Zum Entwurf und Training neuronaler Netze sind in den vergangenen Jahren viele Bibliotheken und Frameworks entstanden. Diese Arbeit nutzt Keras (Chollet u. a., 2015) zur Definition der verwendeten Netze und Tensorflow (Abadi u. a., 2015), um sie zu trainieren.

5.2 Problemstellung

Konkret sollen zwei verschiedene Ansätze – und damit zwei verschiedene Problemstellungen für die jeweiligen Lernverfahren – betrachtet werden. Beide werden in Form eines neuronalen Netzes realisiert, das als Eingabebild das aus dem Kamerabild berechnete IPM-Bild erhält und dessen Ausgabe das gestellte Problem löst.

Verwendet werden hierbei Convolutional Neural Networks (CNNs), welche aufgrund ihrer Eigenschaft, unter Verwendung relativ weniger gelernter Parameter räumlich im Bild gut abstrakte Konzepte zu erfassen, in vielen Bereichen der Bildverarbeitung mittlerweile das Mittel der Wahl sind (vgl. Ruiz-del-Solar, Loncomilla und Soto, 2018).

5.2.1 Erkennung von Linien mittels neuronaler Netze

Als erster Ansatz soll versucht werden, analog zum analytischen Ansatz den Großteil des bisherigen Bildverarbeitungssystems beizubehalten und lediglich die Erkennung von Feldlinien durch Nutzung des IPM-Bilds zu verbessern. Das Ziel ist hierbei also, ein neuronales Netz zu trainieren, welches Feldlinien in Bildern erkennt. Konkret bedeutet dies, dass aus der Ausgabe des Netzes direkt Linien im Bild abgeleitet werden können sollen. Diese Anforderung eröffnet mehrere Möglichkeiten der Interpretation des Problems im Hinblick auf die Anwendungsmöglichkeit neuronaler Netze.

Zunächst einmal kann die Anwendung als Detektionsproblem betrachtet werden. In der Literatur werden verschiedene Netzarchitekturen verwendet, die direkt Objekte in Bildern finden und je gefundenem Objekt das es minimal umgebende Rechteck ausgeben, etwa Faster R-CNN (Ren u. a., 2017), SSD (W. Liu u. a., 2016) und YOLOv3 (Redmon und Farhadi, 2018). Dieser Ansatz ist allerdings für die gegebene Zielsetzung problematisch, da einerseits Feldlinien keine klassischen Objekte in Bildern sind, sondern je nach Ausrichtung im Bild vollkommen unterschiedlich aussehen, und andererseits die Ausgabe des umgebenden Rechtecks zur Beschreibung des Linienverlaufs nicht sonderlich hilfreich ist.

Stattdessen kann das Problem des Findens von Linien im Bild auch als Problem der Regression der Linienausrichtung pro Pixel betrachtet werden: Ausgabe ist in diesem Fall ein Bild, das in jedem Pixel eine Stärke der Linie jeweils in die beiden Richtungen der Achsen des Koordinatensystems enthält – zum Beispiel jeweils im Bereich $[-1{,}1]$. Hierbei ist zu beachten, dass wie auch in Abb. 5.1 gezeigt nur die Hälfte der so möglichen Ausrichtungen einer Linie eindeutig ist: zum Beispiel beschreiben $(1, 0)$ und $(-1, 0)$ denselben Linienverlauf. Eine mögliche Lösungsmöglichkeit für diese Einschränkung wäre, entweder die x- oder y-Richtung auf nur den positiven oder negativen Bereich zu beschränken.

Eine weitere Interpretation der Problemstellung ist, sie als Problem der Klassifikation der Linienausrichtung pro Pixel beziehungsweise als Problem der semantischen Segmentierung zu betrachten. Dieser Ansatz stellt im Prinzip eine Diskretisierung der oben beschriebenen Regression dar: pro Pixel soll eine Klasse ausgegeben werden, die beschreibt, ob an dieser

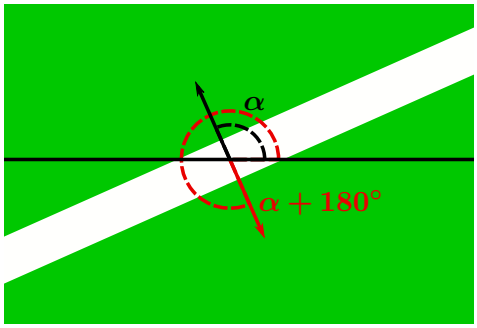


Abbildung 5.1 Illustration der Ausrichtung einer Feldlinie. Da die Linie im zweidimensionalen Raum zwei Normalen hat, kann ihre Orientierung durch zwei verschiedene Winkel beschrieben werden.

Stelle eine Feldlinie verläuft und wenn ja, in welche Richtung sie ausgerichtet ist. Die Richtungen von Feldlinien werden also wie in Abb. 5.2 dargestellt durch eine festgelegte Menge von Klassen beschrieben. Die Betrachtung als semantische Segmentierung hat den Vorteil, dass sich gezeigt hat, dass CNNs für diese Aufgabe besonders gut geeignet sind (vgl. Garcia-Garcia u. a., 2017). Daher wurde für das zu entwickelnde Netz diese Herangehensweise gewählt.

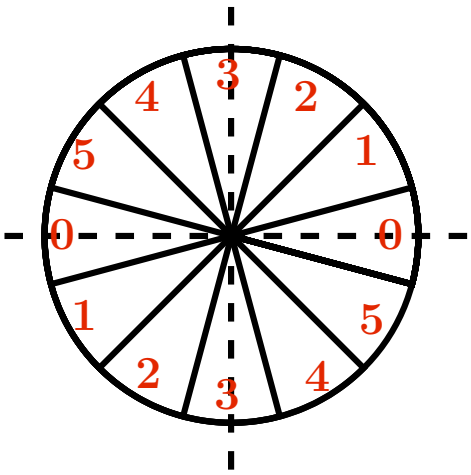


Abbildung 5.2 Unterteilung von Linienausrichtungen in sechs diskrete Klassen. Durch eine solche Diskretisierung der Linienrichtungen können Richtungen von einem neuronalen Netz durch Klassifikation statt Regression angenähert werden.

5.2.2 Erkennung von Merkmalen mittels neuronaler Netze

Der zweite in dieser Arbeit verfolgte Ansatz zur Verwendung von neuronalen Netzen versucht, das Problem der Merkmalserkennung auf der Bodenebene ohne weitere Nachverarbeitung zu lösen. Zu einem gegebenen IPM-Eingabebild soll ein neuronales Netz hierbei direkt die Positionen und Ausrichtungen von Merkmalen in diesem Bild ausgeben.

Die hierfür definierten Merkmale zeigen Tab. 5.1 und Abb. 5.3. Diese Merkmale wurden so gewählt, dass sie einerseits für Menschen intuitiv im Bild erkennbar sind – ein gutes Kriterium, um abzuschätzen, ob ein neuronales Netz sie erkennen könnte – und sie andererseits auf die

aktuell im B-Human-System verwendeten Feldmerkmale abgebildet werden können, sodass das Verfahren zur Selbstlokalisierung des Roboters nicht angepasst werden muss.

Kürzel	Merkmal	Ausrichtung
CC	Zentrum des Mittelkreises	keine
PM	Strafstoßpunkt	keine
OC	Äußere Spielfeldecken	Richtung Diagonale
CI	Kreuzungen von Mittellinie und Mittelkreis	Richtung Feldmittelpunkt
T	T-Kreuzungen von Mittellinie und Seitenlinien	Richtung Feldmittelpunkt
PAT	T-Kreuzungen von Torlinien und Strafraum	Richtung Strafraumlinie
PAC	Ecken der Strafräume	Richtung Diagonale

Tabelle 5.1 Auflistung der definierten Merkmale, die vom neuronalen Netz zur Merkmalserkennung gefunden werden sollen, mit der jeweiligen Bedeutung ihrer Ausrichtung. Die Kürzel entsprechen den verwendeten Markierungen in Abb. 5.3.

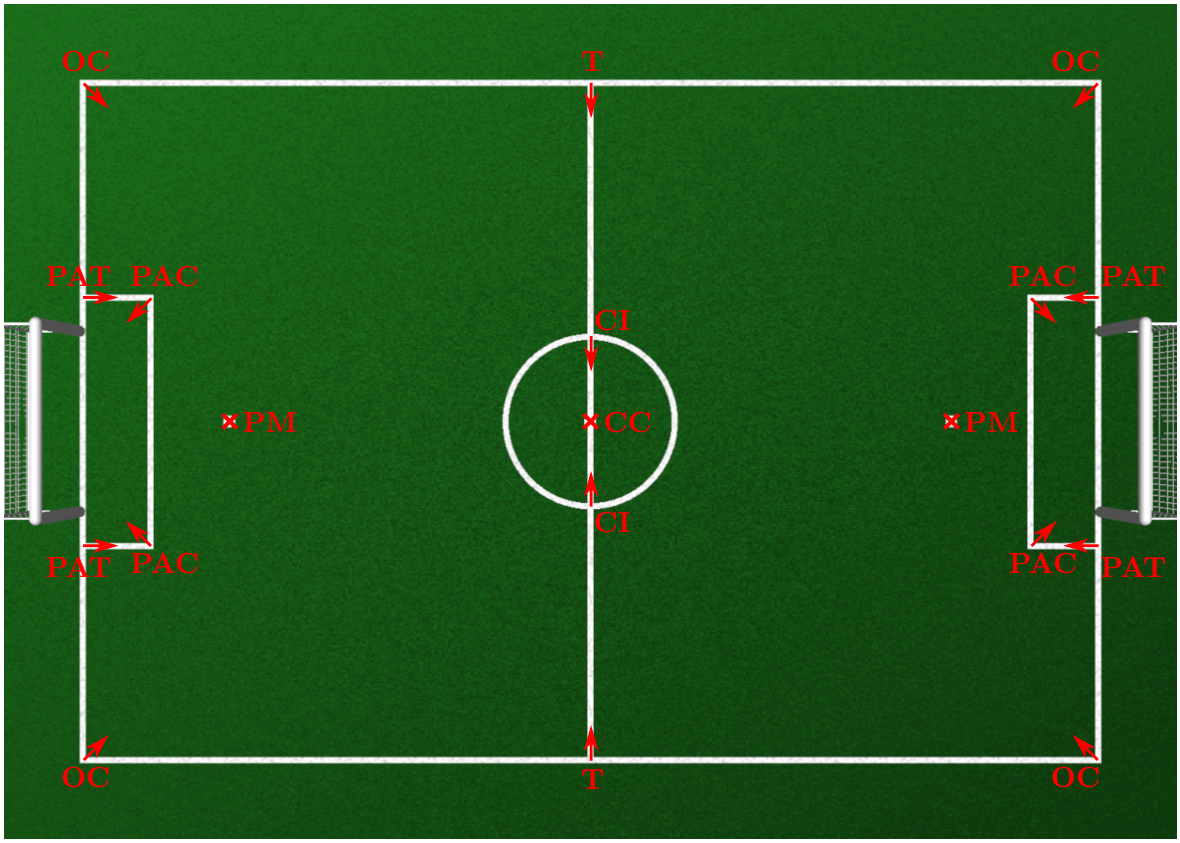


Abbildung 5.3 Visualisierung der definierten Merkmale, die vom neuronalen Netz zur Merkmalserkennung gefunden werden sollen, auf dem Spielfeld.

Auch für diesen Ansatz gibt es mehrere mögliche Herangehensweisen je nach Interpretation der Problemstellung. Diese unterscheiden sich kaum von den oben für den ersten Ansatz genannten Möglichkeiten: Objektdetektion, Regression und semantische Segmentierung.

Objektdetektion im Sinne des Findens von umschließenden Rechtecken ist im Kontext der Merkmalserkennung sinnvoller als zuvor bei der Linienerkennung, allerdings bleibt das Pro-

blem, dass die Ausrichtung des jeweiligen Merkmals nicht ohne Weiteres mit ausgegeben werden kann. Außerdem entsprechen die definierten Merkmale Punkten auf dem Spielfeld statt Bereichen, sodass die Definition des minimal umschließenden Rechtecks noch geklärt werden müsste.

Bei der Betrachtung als Regressionsproblem kann die gleiche Definition der Ausgabe benutzt werden wie oben für die Linienerkennung genannt – eine Stärke der Ausprägung des Merkmals jeweils in x- und y-Richtung. Allerdings hätte das Netzwerk in diesem Fall eine solche Ausgabe pro zu erkennendem Merkmalstyp. Weiterhin muss eine separate Möglichkeit der Ausgabe für diejenigen Merkmale gefunden werden, die keine Ausrichtung haben können. Eine einfache Weise, dieses Problem zu umgehen, ist, für diese Merkmale einen einzelnen Ausgabekanal zu nutzen, welcher lediglich eine symbolische Zahl für das Vorhandensein des Merkmals im Bereich $[0, 1]$ ausgibt.

Zuletzt kann auch dieser Ansatz als Problem der semantischen Segmentierung angegangen werden. Hierbei wird erneut für diejenigen Merkmale, die eine Ausrichtung haben, pro Richtungsbereich eine Klasse erstellt. Dies ist auch in diesem Fall eine vielversprechende Methode, insbesondere auch, weil mit ihr kein Unterschied zwischen dem Lernen von Merkmalen mit und ohne Ausrichtung besteht. Daher wird auch für den Ansatz der direkten Merkmalserkennung die Herangehensweise der semantischen Segmentierung gewählt.

5.2.3 Entwurfskriterien

Für die Entwicklung und das Training der neuronalen Netze wurden einige Rahmenbedingungen aufgestellt, die dem Kontext des Roboterfußballs in der SPL geschuldet sind.

Zunächst einmal sollte die Qualität der Ergebnisse der Modelle relativ unabhängig von der Beleuchtung der Szene im Eingabebild sein, sodass sie unter den verschiedenen von den Regeln vorgegebenen natürlichen und künstlichen Beleuchtungsbedingungen gleichermaßen gut funktionieren.

Weiterhin ist es sowohl beim Erkennen von Linien als auch von größeren Merkmalen wichtiger, dass die erkannten Merkmale richtig sind, als dass alle vorhandenen Merkmale auch erkannt werden. Demnach ist bei der Beurteilung des Trainings und der Wahl der Hyperparameter stets die Genauigkeit (Precision) als Metrik wichtiger als die Trefferquote (Recall). Dies ist dem Ziel des Systems geschuldet, eine Grundlage für die Selbstlokalisierung eines Roboters auf einem Spielfeld zu bilden: wenige, richtige Ausgaben sind hierbei besser als viele Ausgaben mit einem hohen Anteil an Falscherkennungen.

Außerdem stellt der *NAO* mit seinen beschränkten Ressourcen als schlussendliche Ausführungsplattform eine Beschränkung der maximalen Komplexität der Netze dar. Daher sollte nach der initialen Prüfung der Machbarkeit jedes Ansatzes versucht werden, die Komplexität des entsprechenden Modells so stark wie möglich zu reduzieren.

5.3 Datensatz

Für das Training der neuronalen Netze ist ein entsprechender Datensatz mit annotierten Bildern vonnöten, die den Bildern, die die Netze später zur Laufzeit verarbeiten sollen, entsprechen.

5.3.1 Erstellung des Datensatzes

Hierfür wurde eine Menge von 430 Kamerabildern aus mehreren Roboterfußballspielen gewählt, die entsprechend der gewünschten Umgebungs- und Beleuchtungsunabhängigkeit aus verschiedenen Umgebungen mit unterschiedlichen Beleuchtungssituationen stammen. Die konkreten Spiele, aus denen die Bilder stammen, wurden bei der RoboCup German Open 2019 in Magdeburg, auf mehreren Spielfeldern beim RoboCup 2019 in Sydney und als Testspiele an mehreren Orten in der Universität Bremen gespielt.

Da die jeweils zu den Bildern gehörende Position und Ausrichtung der Kamera relativ zu den Füßen des Roboters bekannt ist, konnte zu jedem Bild das IPM-Bild erzeugt werden. Dieses wurde dann sowohl zum Training als auch zum manuellen Annotieren der Bilder genutzt. In jedem der Bilder wurden jeweils manuell alle Linien und Feldmerkmale mit ihren Ausrichtungen unter Nutzung des Programms labelme (Wada, 2016) annotiert.

Gemäß dem Ziel, dass die Ausgaben der gelernten Modelle unabhängig von der Beleuchtung der Umgebung und damit auch der Farbe des Umgebungslichts sein sollen, wurde beschlossen, dass die Eingabedaten keine Farbinformationen, sondern nur Helligkeitswerte enthalten.

Entsprechend der oben gewählten Definition der Ausgabe der Netze wurden dann zunächst die Ausgabeklassen pro Netz als Kombination von Merkmalstyp und Ausrichtung bestimmt. Für eine festgelegte Anzahl möglicher Ausrichtungen pro Merkmal N_{dir} wird dazu die Nummer der diskretisierten Richtungsklasse zu einem gegebenen Winkel $\alpha \in [-\pi, \pi)$ wie in Formel 5.3 gezeigt bestimmt. Analog funktioniert wie in Formel 5.4 gezeigt die Diskretisierung der Linienrichtungen zu Klassen für das Linienerkennungsnetz, wobei hierbei wie schon oben erwähnt und in Abb. 5.1 gezeigt berücksichtigt wird, dass Linienausrichtungen 180°-periodisch sind.

$$\text{Index}_{\text{Merkmale}}(\alpha) = \left\lfloor (\alpha + \pi) \cdot \frac{N_{\text{dir}}}{2\pi} + 0{,}5 \right\rfloor \mod N_{\text{dir}} \tag{5.3}$$

$$\text{Index}_{\text{Linien}}(\alpha) = \left\lfloor (\alpha + \pi) \cdot \frac{N_{\text{dir}}}{\pi} + 0{,}5 \right\rfloor \mod N_{\text{dir}} \tag{5.4}$$

Nachdem nun die gewünschten Ausgabeklassen definiert wurden, können sie allerdings noch nicht direkt zum Training genutzt werden. Idealerweise wären zu jeder Klasse ausreichend Beispiele in den Trainingsdaten vorhanden, sodass die Netze ihre Eigenschaften lernen und sie sinnvoll abbilden können. Praktisch sind allerdings einige der ermittelten Klassen nicht

oder nur selten im annotierten Datensatz vorhanden. Empirisch wurde bestimmt, dass das in den nachfolgenden Abschnitten beschriebene Trainingsverfahren für die ebenfalls später beschriebene Netzwerkarchitektur unter Berücksichtigung des vorhandenen Datensatzes für all jene Klassen funktioniert, die mindestens in 14 % der Bilder des Datensatzes vorkommen. Daher kommen nur diejenigen Klassen als Ausgaben in Betracht, für die dies gilt.

Als Parameter für die zu trainierenden Modelle wurde $N_{\mathrm{dir}} = 8$ gewählt. Nach dem beschriebenen Ausschlussverfahren bleiben demnach 8 Klassen, die das Linienerkennungsnetz erkennen soll, und 16 Klassen für das Merkmalserkennungsnetz.

Bis zu diesem Punkt bestehen die Klassenannotationen aus Linien und Punkten in den Bildern, die jeweils einer Klasse zugeordnet sind. Um aus diesen symbolischen Daten mehrkanalige Bilder zu generieren, die der gewünschten Ausgabe der neuronalen Netze entsprechen, werden sie in zunächst mit 0 gefüllte Bilder mit dem Wert 1 eingezeichnet. Dabei wird um Punktmerkmale jeweils ein Kreis mit einem Radius gezeichnet, der auf der Bodenebene 2,5 cm entspricht, während Linien eine Breite von 2 cm erhalten. Dies hat den Grund, dass die direkt um ein Merkmal liegenden Pixel im IPM-Bild in der Regel noch zum Merkmal dazugehören und es daher nicht sinnvoll wäre, sie als Negativbeispiele zu betrachten. Umso stärker gilt dies im Fall von häufig vorkommenden verwischten Bildern, bei denen ein exakter Mittelpunkt nur schwer festgelegt werden kann. Die so erhaltenen Bilder zeigt Abb. 5.4, wobei hier zur besseren Darstellung allerdings die Bilder aller Klassen zusammengefasst sind.

5.3.2 Aufteilung des Datensatzes

Für das Training sowie die Modellselektion werden zwei disjunkte Teilmengen – die Trainings- und die Validierungsmenge – des Datensatzes benötigt. Um sicherzustellen, dass beide Mengen repräsentativ für sämtliche zu lernende Klassen sind, werden einerseits jeweils eigene Unterteilungen des Datensatzes für jedes der beiden Netze durchgeführt und andererseits die Unterteilung jeweils so gewählt, dass die Verteilung des Vorhandenseins der Klassen in den Bildern jeder Untermenge der Verteilung im gesamten Datensatz entspricht. Um dies sicherzustellen, wird das von Szymański und Kajdanowicz (2017a) vorgestellte Verfahren in Form seiner Implementierung in der Bibliothek scikit-multilearn (Szymański und Kajdanowicz, 2017b) genutzt.

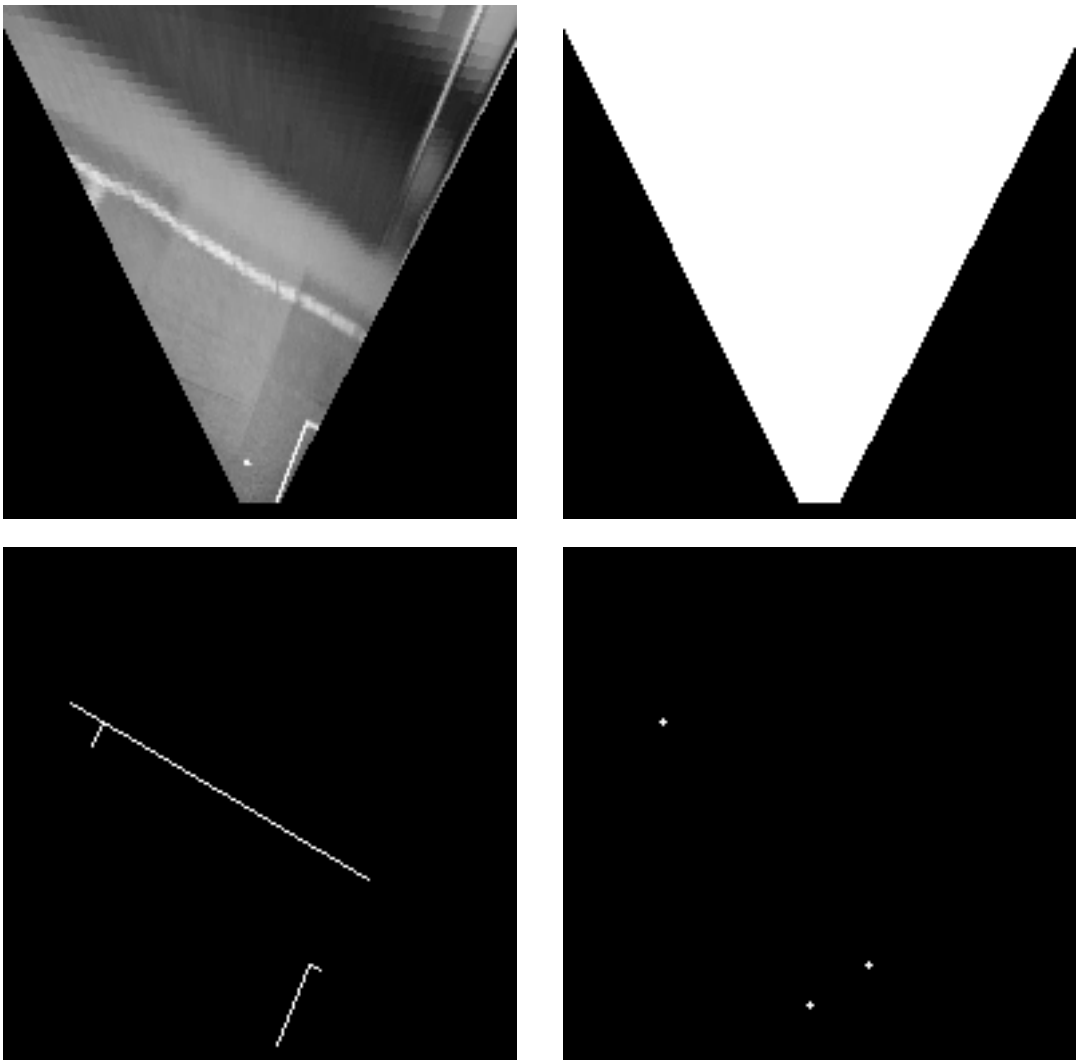


Abbildung 5.4 Überblick über die Bilddaten im erzeugten Datensatz.
Oben links: IPM-Bild als Eingabebild für die neuronalen Netze.
Oben rechts: Bildmaske, die den gültigen Bereich des Eingabebilds auszeichnet.
Unten links: Bild mit annotierten Linien.
Unten rechts: Bild mit annotierten Merkmalen.

5.3.3 Augmentation

Die so erhaltenen Trainingsdatenmengen enthalten zwar durch die Auswahl der Ausgangsdaten eine gewisse Spannweite an verschiedenen Umgebungen und Beleuchtungssituationen. Allerdings ist die Datenmenge noch immer relativ klein und damit für das Training von neuronalen Netzen nicht optimal, da Lernverfahren bei kleinen Trainingsmengen dazu tendieren, die Parameter des Modells an die Trainingsmenge überanzupassen. Wie unter anderem von Perez und Wang (2017), Taylor und Nitschke (2017) und Xu u. a. (2016) gezeigt, bietet die Methode, Daten beim Einspeisen in den Trainingsalgorithmus zu augmentieren, eine Lösung für dieses Problem.

Konkret soll durch die Augmentation erreicht werden, dass die gelernten Modelle nicht überangepasst an die Trainingsdaten, unabhängig von Änderungen der Beleuchtungssituation und unabhängig von Rauschen in den Eingabedaten sind. Hierzu wird für jedes Bild bei der Übergabe an den Trainingsalgorithmus jeweils zufällig entschieden, ob einer der im Folgenden beschriebenen Verarbeitungsschritte durchgeführt wird. Zur Umsetzung der Augmentation beim Training wird dabei die Bibliothek imgaug (Jung, 2019) verwendet.

- **Horizontale Spiegelung**
  Im Kontext der verwendeten IPM-Bilder erzeugt horizontales Spiegeln jeweils ein ebenso realistisches Bild in der vorliegenden Domäne. Hierbei muss allerdings berücksichtigt werden, dass sich dabei auch die Richtungen der Merkmale im Bild ändern, sodass jeweils die Klassen in den annotierten Daten mit geändert werden müssen. Beim Training wird pro Epoche jedes Bild jeweils einmal normal und einmal gespiegelt an das Trainingsverfahren übergeben.

- **Verwischen des Bilds**
  Da die Merkmalserkennung auf mobilen Robotern funktionieren soll, müssen die gelernten Modelle mit durch Bewegung des Roboters verwischten Kamerabildern umgehen können. Deshalb werden die Bilder der Trainingsmenge mit zufälliger Stärke verwischt, wobei jedes Bild entweder mit einem Gauß-Filter, Box-Filter oder einer Operation zur Simulation von Bewegungsunschärfe in eine zufällige Richtung bearbeitet wird.

- **Additives Rauschen**
  Analog zur Bildverwischung kann auch je nach Belichtungssituation und Kameraeinstellung verschieden starkes Rauschen in verschiedener Form in den Kamerabildern des *NAO* auftreten. Um dies in den Trainingsdaten zu simulieren, wird auf jeden Pixel des jeweiligen Bilds in zufälliger Stärke entweder gleichverteiltes oder gaußsches Rauschen addiert.

- **Helligkeits- und Kontraständerung**
  Schließlich kann auch die allgemeine Helligkeit und der Kontrast in Kamerabildern je nach Umgebung stark abweichen. Entsprechend werden zur Augmentation Kamerabilder entweder mit einem zufälligen konstanten Faktor multipliziert, ein zufälliger

konstanter Wert zu ihnen addiert oder ihr Kontrast um einen zufälligen Wert erhöht oder reduziert.

Sämtliche Wertebereiche der zufällig gewählten Parameter in den oben genannten Verfahren sind dabei derart gewählt, dass der Inhalt der resultierenden Bilder von Menschen stets noch erkannt werden kann.

5.4 Entwurf der Modelle

Für die beiden genannten Problemstellungen – Erkennen von Linien beziehungsweise Merkmalen – soll jeweils für ein gegebenes Eingabebild ein ebenso großes Ausgabebild erzeugt werden, wobei ein Problem der semantischen Segmentierung, also ein Klassifikationsproblem pro Pixel, vorliegt. Zur Lösung dieser Art von Aufgaben werden in der Literatur verschiedene Architekturen neuronaler Netze vorgeschlagen.

5.4.1 Segmentierungsarchitekturen in der Literatur

Die einfachste, bis etwa 2014 in der Regel verwendete Methode – siehe zum Beispiel Ning u. a. (2005) und Cirean u. a. (2012) – ist, ein Netz zu nutzen, das jeweils einen Teilbereich des Bildes als Eingabe erhält und in Form eines klassischen Klassifikationsnetzes nach einigen Convolutional- und Pooling-Layern mittels eines Fully-Connected-Layers als Ergebnis die Klasse für genau einen Pixel ausgibt. Dies ist normalerweise der Pixel, der sich in der Mitte des Eingabeausschnitts befindet. Das Ausgabebild wird dann durch wiederholte Anwendung des Netzwerkes mit einem nach und nach über das Eingabebild geschobenen Ausschnitt – dem sogenannten Sliding Window – erhalten.

Shelhamer, Long und Darrell (2017) entwickelten diesen Ansatz weiter, indem sie in einem solchen Sliding-Window-Netz den abschließenden Fully-Connected-Layer durch einen funktionell gleichen Convolutional-Layer ersetzten, um damit ein sogenanntes Fully Convolutional Network zu erstellen. Dadurch konnten sie durch Vergrößerung des Eingabefensters mehrere Pixel, bis hin zum gesamten Eingabebild, zugleich klassifizieren konnten.

Mit dem SegNet (Badrinarayanan, Kendall und Cipolla, 2017) wurde dann eine neue Form von neuronaler Netzarchitektur zur semantischen Segmentierung vorgestellt. Das Netz besteht hierbei aus zwei Teilen, einem Encoder-Teil, welcher wie die bisher genannten Netzarchitekturen aus Convolutional- und Pooling-Layern besteht, und einem Decoder-Teil, der mit abwechselnden Upscaling- und Convolutional-Layern die Daten in den Höhen- und Breitendimensionen wieder hochskaliert, um zum Schluss wieder die Auflösung des Eingangsbilds zu erhalten. Die Idee ist hierbei, dass der Encoder-Teil die für die Lösung des Segmentierungsproblems relevanten Eigenschaften der Bilddaten aus der Domäne des Bilds extrahiert, während der Decoder-Teil die so erhaltenen Informationen in die Domäne des Zielbilds überträgt.

Solche Encoder-Decoder-Netze haben allerdings das Problem, dass ihre Ausgabe, obwohl sie dieselbe Auflösung wie die Eingabe hat, in der Regel deutlich gröber und ungenauer in der Segmentierung ist als die Annotationen der Trainingsdaten. Um die Genauigkeit der Segmentierung zu verbessern, wurde daher das U-Net (Ronneberger, Fischer und Brox, 2015) entwickelt. Dieses fügt zusätzliche Verbindungen (gelegentlich Skip-Verbindungen genannt) zwischen dem Encoder- und dem Decoder-Teil des Netzes hinzu. Diese hängen jeweils wie in Abb. 5.5 gezeigt die Ausgabe des letzten Convolutional-Layers jeder Auflösungsstufe des Encoders entlang der Kanaldimension an die Eingabe des ersten Convolutional-Layers derselben Auflösungsstufe des Decoders an. Dadurch besteht die Möglichkeit, dass im Decoder auch Details beachtet werden können, die durch das Herunterskalieren der Daten im Encoder verloren gehen. Mittlerweile werden in der Regel in Netzen, die die Encoder-Decoder-Architektur nutzen, die im U-Net vorgeschlagenen Skip-Verbindungen stets verwendet (vgl. Ye und Sung, 2019).

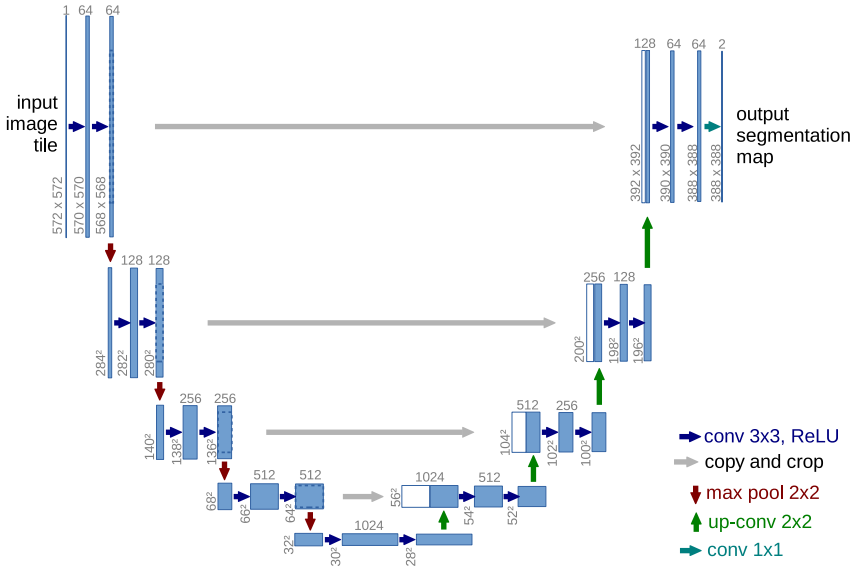


Abbildung 5.5 Architektur des U-Nets. Abbildung aus Ronneberger, Fischer und Brox (2015).

5.4.2 Gewählte Netzarchitektur

Daher wurde entschieden, auch für die in dieser Arbeit entworfenen Netze dem U-Net ähnliche Architekturen zu verwenden.

Wie von Ronneberger, Fischer und Brox (2015) beschrieben, besteht der Encoder-Teil des U-Nets aus wiederholten Abfolgen von zwei 3×3-Convolutional-Layern mit ReLU-Aktivierung und einem 2×2-Max-Pooling-Layer, wobei jeweils die Anzahl der Filter der Convolutional-Layer mit jeder Skalierungsstufe verdoppelt wird. Umgekehrt besteht der Decoder-Teil aus wiederholten Abfolgen von einem 2×2-Upscaling-Layer und zwei 3×3-Convolutional-Layern mit ReLU-Aktivierung, wobei die Anzahl der Filter nach jedem Hochskalieren halbiert wird. Die Skip-Verbindungen konkatenieren dabei jeweils die Ausgabe des letzten Convolutional-

Layers vor jedem Max-Pooling-Layer im Encoder mit der Eingabe des ersten Convolutional-Layers nach dem Upscaling-Layer der gleichen Skalierungsstufe im Decoder.

Dieser Vorlage folgend sind auch das Linien- und das Merkmalserkennungsnetz entsprechend aufgebaut. Allerdings wurden einige Anpassungen vorgenommen. Zunächst einmal wurde zwischen jeden Convolutional-Layer und dessen Aktivierungsfunktion ein Batch-Normalization-Layer (BN) entsprechend der Empfehlung von Ioffe und Szegedy (2015) eingefügt, um die Geschwindigkeit und die Ergebnisse des Trainings der Netze zu verbessern. Weiterhin verwenden die Netze die von Maas, Hannun und Ng (2013) vorgeschlagene Aktivierungsfunktion LeakyReLU (siehe Formel 5.5) statt ReLU, da diese im gegebenen Anwendungsfall bei ansonsten gleicher Netzarchitektur und Lernverfahren bessere Ergebnisse erzielt. Der Parameter α wird dabei auf den Wert 0,3 gesetzt.

$$\text{LeakyReLU}_\alpha(x) = \begin{cases} x & \text{für } x \geq 0 \\ \alpha x & \text{sonst} \end{cases} \tag{5.5}$$

Die Wahl der konkreten Architektur der beiden Netze ist eine Abwägung zwischen der Güte der erreichten Klassifikation und der Kompaktheit, sprich der späteren Ausführungszeit, des jeweiligen Modells. Hierbei sind die relevanten festzulegenden Hyperparameter mit starkem Einfluss auf diese Eigenschaften die Anzahl der Filter pro Convolutional-Layer, sowie die Tiefe des Netzwerks, also die Anzahl von Herunter- und Hochskalierungsschritten.

Die Anzahl der Filter pro Convolutional-Layer bestimmt dabei die Anzahl der Kanäle, die durch den jeweils nächsten Convolutional-Layer zu neuen Informationen kombiniert werden können, sodass sie einen direkten Einfluss auf die Güte der Klassifizierung hat. Allerdings steigt mit der Anzahl der Filter auch der Ressourcenaufwand bei der späteren Ausführung der Netze auf dem *NAO*. Im Gegensatz zum U-Net, in dem die Filter der Convolutional-Layer nach jedem Max-Pooling-Layer verdoppelt werden, wächst daher die Anzahl der Filter in den hier entworfenen Netzen zwar mit den Skalierungsstufen an, allerdings linear und damit deutlich weniger stark. Zu beachten ist dabei, dass für die optimale Ressourcennutzung bei der Inferenz auf dem *NAO* aufgrund des dort verwendeten Verfahrens (vgl. Kapitel 5.6.1) Filteranzahlen stets Vielfache von 4 sein sollten.

Die Tiefe des Netzes beeinflusst hingegen neben der Ausführungszeit insbesondere das rezeptive Feld jedes Ausgabepixels, das heißt die Größe des Fensters um den entsprechenden Pixel im Eingabebild, innerhalb dessen die Eingabewerte einen Einfluss auf die letztendliche Klassifizierung haben. Damit beeinflusst die Netzwerktiefe auch die maximale Größe von Merkmalen im Bild, die erkannt werden können. Dies führt dazu, dass das Merkmalserkennungsnetz tiefer als das Feldlinienerkennungsnetz sein muss, da Feldlinien im IPM-Bild deutlich kleiner als die teilweise komplexen oder auf größeren Bildausschnitten beruhenden definierten Merkmale wie zum Beispiel der Mittelkreis sind.

Die resultierenden Netzarchitekturen für das Finden von Feldlinien beziehungsweise Bodenmerkmalen zeigen Tab. 5.2 und Tab. 5.3.

Layer	**# Filter**	**Ausgabegröße**
Eingabe		$(192, 192, 1)$
3×3-Convolution, BN, LeakyReLU	8	$(192, 192, 8)$
3×3-Convolution, BN, LeakyReLU	8	$(192, 192, 8)$
$\vert$ 2×2-Max-Pooling		$(96, 96, 8)$
$\vert$ 3×3-Convolution, BN, LeakyReLU	12	$(96, 96, 12)$
$\vert$ 3×3-Convolution, BN, LeakyReLU	12	$(96, 96, 12)$
$\vert$ $\vert$ 2×2-Max-Pooling		$(48, 48, 12)$
$\vert$ $\vert$ 3×3-Convolution, BN, LeakyReLU	16	$(48, 48, 16)$
$\vert$ $\vert$ 3×3-Convolution, BN, LeakyReLU	16	$(48, 48, 16)$
$\vert$ $\vert$ $\vert$ 2×2-Max-Pooling		$(24, 24, 16)$
$\vert$ $\vert$ $\vert$ 3×3-Convolution, BN, LeakyReLU	20	$(24, 24, 20)$
$\vert$ $\vert$ $\vert$ 3×3-Convolution, BN, LeakyReLU	20	$(24, 24, 20)$
$\vert$ $\vert$ $\vert$ 2×2-Upscaling		$(48, 48, 20)$
$\vert$ $\vert$ Konkatenation		$(48, 48, 36)$
$\vert$ $\vert$ 3×3-Convolution, BN, LeakyReLU	16	$(48, 48, 16)$
$\vert$ $\vert$ 3×3-Convolution, BN, LeakyReLU	16	$(48, 48, 16)$
$\vert$ $\vert$ 2×2-Upscaling		$(96, 96, 16)$
$\vert$ Konkatenation		$(96, 96, 28)$
$\vert$ 3×3-Convolution, BN, LeakyReLU	12	$(96, 96, 12)$
$\vert$ 3×3-Convolution, BN, LeakyReLU	12	$(96, 96, 12)$
$\vert$ 2×2-Upscaling		$(192, 192, 12)$
Konkatenation		$(192, 192, 20)$
3×3-Convolution, BN, LeakyReLU	8	$(192, 192, 8)$
3×3-Convolution, BN, LeakyReLU	8	$(192, 192, 8)$
1×1-Convolution	8	$(192, 192, 8)$

Tabelle 5.2 Übersicht über die Architektur des Linienerkennungsnetzes. Vertikale Linien entsprechen dabei Skip-Verbindungen.

Layer	**# Filter**	**Ausgabegröße**
Eingabe		$(192, 192, 1)$
3×3-Convolution, BN, LeakyReLU	8	$(192, 192, 8)$
3×3-Convolution, BN, LeakyReLU	8	$(192, 192, 8)$
| 2×2-Max-Pooling		$(96, 96, 8)$
| 3×3-Convolution, BN, LeakyReLU	12	$(96, 96, 12)$
| 3×3-Convolution, BN, LeakyReLU	12	$(96, 96, 12)$
| | 2×2-Max-Pooling		$(48, 48, 12)$
| | 3×3-Convolution, BN, LeakyReLU	16	$(48, 48, 16)$
| | 3×3-Convolution, BN, LeakyReLU	16	$(48, 48, 16)$
| | | 2×2-Max-Pooling		$(24, 24, 16)$
| | | 3×3-Convolution, BN, LeakyReLU	20	$(24, 24, 20)$
| | | 3×3-Convolution, BN, LeakyReLU	20	$(24, 24, 20)$
| | | | 2×2-Max-Pooling		$(12, 12, 20)$
| | | | 3×3-Convolution, BN, LeakyReLU	24	$(12, 12, 24)$
| | | | 3×3-Convolution, BN, LeakyReLU	24	$(12, 12, 24)$
| | | | 2×2-Upscaling		$(24, 24, 24)$
| | | Konkatenation		$(24, 24, 44)$
| | | 3×3-Convolution, BN, LeakyReLU	20	$(48, 48, 20)$
| | | 3×3-Convolution, BN, LeakyReLU	20	$(48, 48, 20)$
| | | 2×2-Upscaling		$(96, 96, 20)$
| | Konkatenation		$(48, 48, 36)$
| | 3×3-Convolution, BN, LeakyReLU	16	$(48, 48, 16)$
| | 3×3-Convolution, BN, LeakyReLU	16	$(48, 48, 16)$
| | 2×2-Upscaling		$(96, 96, 16)$
| Konkatenation		$(96, 96, 28)$
| 3×3-Convolution, BN, LeakyReLU	12	$(96, 96, 12)$
| 3×3-Convolution, BN, LeakyReLU	12	$(96, 96, 12)$
| 2×2-Upscaling		$(192, 192, 12)$
Konkatenation		$(192, 192, 20)$
3×3-Convolution, BN, LeakyReLU	8	$(192, 192, 8)$
3×3-Convolution, BN, LeakyReLU	8	$(192, 192, 8)$
1×1-Convolution	16	$(192, 192, 16)$

Tabelle 5.3 Übersicht über die Architektur des Merkmalserkennungsnetzes. Vertikale Linien entsprechen dabei Skip-Verbindungen.

5.5 Training der Netze

Die Parameter des Training der nun definierten Netze haben eine große Auswirkung auf die spätere Güte der Klassifikation. Daher werden diese sorgfältig abgewogen und gegebenenfalls experimentell bestimmt.

5.5.1 Lossfunktionen

Ein wesentlicher Aspekt, der bestimmt, wie gut ein neuronales Netz die vorliegende Aufgabe löst, ist die zum Training genutzte Lossfunktion. Dies ist diejenige Funktion, die für eine gegebene Ausgabe y_{pred} des Netzes und die durch manuelle Annotation definierte, gewünschte Ausgabe y_{true} einen Wert liefert, der die Güte der Prädiktion des Netzes beschreibt. Die Lossfunktion ist deshalb entscheidend für das Training des Modells, weil sie die Funktion ist, die zum Ermitteln der Aktualisierungen der Netzparameter pro Schritt durch den Optimierer nach den jeweiligen Parametern abgeleitet wird.

In den im Folgenden vorgestellten Funktionen bedeutet dabei die Notation $y_{\text{pred}}(z)_i$ die Ausgabe des Netzes an der Stelle des Pixels z für die i-te der C definierten Klassen; analog ist auch $y_{\text{true}}(z)_i$ definiert.

In der Literatur werden im Kontext der semantischen Segmentierung je nach konkreter Problemstellung verschiedene Lossfunktionen verwendet. Zunächst einmal muss für alle nachfolgend vorgestellten Lossfunktionen die Ausgabe der Netze in den Wertebereich $[0, 1]$ überführt werden. Hierbei stellt sich die Frage, ob die Netze pro Pixel eine einzelne Klasse oder potentiell mehrere Klassen ausgeben können sollen.

Im ersteren Fall wird hierzu die in Formel 5.6 gezeigte Softmax-Funktion genutzt, die dafür sorgt, dass pro Pixel die Summe der Ausgaben aller Klassen 1 ergibt, sodass ihre Ausgabe eine Wahrscheinlichkeitsverteilung über die vorgegebenen Klassen darstellt, die hierbei voneinander abhängige Zufallsvariablen sind. Damit auch der Hintergrund, bestehend aus Pixeln, die keiner Klasse zugeordnet sind, als solcher erkannt werden kann, muss er für diese Vorgehensweise als eigene Klasse hinzugefügt werden. Bei Verwendung des Softmax nutzt man zum Training in der Regel die kategorische Kreuzentropie (CCE) zwischen den Ausgaben der Klassen (Formel 5.7) als Lossfunktion.

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{5.6}$$

$$\text{CCE}(z) = -\sum_{i=1}^{C} y_{\text{true}}(z)_i \log\left(\text{softmax}(y_{\text{pred}}(z)_i)\right) \tag{5.7}$$

Das Lernen einer Wahrscheinlichkeitsverteilung über Klassen pro Pixel wird grundsätzlich in der semantischen Segmentierung gern genutzt, da es die einzelnen Klassen gut voneinander

abgrenzen kann (vgl. u. a. Ronneberger, Fischer und Brox, 2015). Allerdings ist dieses Vorgehen im vorliegenden Fall nicht ideal, weil einzelne Klassen nicht nur verschiedene Objekte, sondern auch gleichartige Objekte in verschiedener Orientierung beschreiben.

Sollen die zu lernenden Klassen voneinander unabhängig sein, verwendet man die in Formel 5.8 gezeigte logistische Funktion – in diesem Kontext auch einfach als Sigmoid-Funktion bezeichnet –, die die Ausgabe separat für jede Klasse pro Pixel in den Wertebereich $[0, 1]$ überführt. Dadurch ist einerseits die Zuordnung mehrere Klassen pro Pixel möglich und andererseits wird das Training jeder Klassenzugehörigkeit nicht von den anderen Klassen beeinflusst.

$$\text{sigmoid}(z_i) = \frac{1}{1 + e^{-z_i}} \tag{5.8}$$

Eine häufig verwendete Lossfunktion ist dann die in Formel 5.9 zu sehende binäre Kreuzentropie (BCE), die jeweils das Vorhandensein jeder Klasse von ihrem Nichtvorhandensein abgrenzt.

$$\begin{aligned} \text{BCE}(z)_i = &- y_{\text{true}}(z)_i \cdot \log\left(\text{sigmoid}(y_{\text{pred}}(z)_i)\right) \\ &- \left(1 - y_{\text{true}}(z)_i\right) \cdot \log\left(1 - \text{sigmoid}(y_{\text{pred}}(z)_i)\right) \end{aligned} \tag{5.9}$$

Die binäre Kreuzentropie hat allerdings das Problem, dass sie das Vorhandensein und das Nichtvorhandensein einer Klasse gleich stark gewichtet, sodass das Netz lernt, die Optimierung einer dieser Eigenschaften zu bevorzugen, sofern sie in den Trainingsdaten nicht gleichmäßig verteilt sind. Bei extremer Ungleichverteilung, wie sie in den hier zum Training verwendeten Datensätzen vorliegt – von über 13 000 000 Pixeln in den Trainingsdaten sind zum Beispiel nur circa 1 300 als Strafstoßpunkt klassifiziert –, führt dies gar dazu, dass Netze bei Verwendung dieser Lossfunktion einfach lernen, sämtliche Pixel als Hintergrund zu klassifizieren.

Daher gibt es Lossfunktionen, die versuchen, diese Eigenschaft durch unterschiedliche Gewichtung des Vorder- und Hintergrundes auszugleichen. Die einfachste dieser Varianten stellt die gewichtete binäre Kreuzentropie (WBCE) dar, die wie in Formel 5.10 gezeigt dem positiven Term einen Faktor ω hinzufügt, sodass der Einfluss von positiven und negativen Beispielen ausgeglichen werden kann. Damit positive und negative Beispiele insgesamt den gleichen Einfluss auf das Training haben, sollte ω_i dabei pro Klasse entsprechend Formel 5.11 gewählt werden.

$$\begin{aligned}\mathrm{WBCE}(z)_i = &- \omega_i \cdot y_{\mathrm{true}}(z)_i \cdot \log\left(\mathrm{sigmoid}(y_{\mathrm{pred}}(z)_i)\right) \\ &- (1 - y_{\mathrm{true}}(z)_i) \cdot \log\left(1 - \mathrm{sigmoid}(y_{\mathrm{pred}}(z)_i)\right)\end{aligned} \tag{5.10}$$

$$\omega_i = \frac{\#\text{ Negative Beispiele für Klasse } i}{\#\text{ Positive Beispiele für Klasse } i} \tag{5.11}$$

Eine andere Herangehensweise für das Problem ungleich verteilter Klassen wurde von Lin u. a. (2017) vorgestellt: der Focal Loss (FL; siehe Formel 5.12) fügt der binären Kreuzentropie einen weiteren Term hinzu, der dafür sorgt, dass der Loss deutlich stärker als bei der binären Kreuzentropie wächst, je weiter die Ausgabe y_{pred} vom richtigen Wert in y_{true} entfernt ist. Dies soll den „Fokus“ des Trainings weg vom leicht zu klassifizierenden Hintergrund und hin zu den schwer zu klassifizierenden Beispielen bewegen, wobei die Stärke des Effekts durch den eingeführten Exponenten γ gesteuert werden kann. Zusätzlich wird ein Skalierungsfaktor $\alpha \in [0, 1]$ zur Steuerung des Verhältnisses der Berücksichtigung positiver und negativer Beispiele hinzugefügt.

$$\begin{aligned}\mathrm{FL}(z)_i = &- \alpha_t (1 - p_t)^\gamma \log(p_t) \quad \text{mit} \\ p_t = &\begin{cases}\mathrm{sigmoid}(y_{\mathrm{pred}}(z)_i) & \text{für } y_{\mathrm{true}}(z)_i = 1 \\ 1 - \mathrm{sigmoid}(y_{\mathrm{pred}}(z)_i) & \text{sonst}\end{cases} \\ \alpha_t = &\begin{cases}\alpha & \text{für } y_{\mathrm{true}}(z)_i = 1 \\ 1 - \alpha & \text{sonst}\end{cases}\end{aligned} \tag{5.12}$$

Zum Training der in dieser Arbeit entwickelten neuronalen Netze kommen daher sowohl die gewichtete binäre Kreuzentropie als auch der Focal Loss in Frage. Um die besser geeignetere der beiden Lossfunktionen zu ermitteln, wird für beide Netze jeweils die gleiche Netzarchitektur mit gleichen Trainingsdaten je einmal mit beiden Lossfunktionen trainiert, um schlussendlich das Modell mit den insgesamt besten Klassifizierungseigenschaften auf dem Validierungsdatensatz auszuwählen (siehe Kapitel 5.5.5).

5.5.2 Bildmasken

Wie schon in Kapitel 3 beschrieben, werden bei der Berechnung des IPM-Bilds prinzipbedingt einige Pixel auf Punkte außerhalb des Kamerabilds projiziert, sodass Teile des IPM-Bilds keine gültigen Informationen enthalten.

Daher sollte beim Training der Netze beachtet werden, dass diejenigen Pixel, die außerhalb des Kamerabilds liegen, keinen Einfluss auf das Training haben. Dieses Ziel kann erreicht werden, indem das Ergebnis der Lossfunktion pro Pixel mit einer binären Bildmaske multipliziert wird, deren Einträge genau dann 1 sind, wenn sie gültigen Pixeln im Kamerabild entsprechen.

Da bei der Durchführung des Inverse Perspective Mapping leicht ermittelt werden kann, welche Pixel gültig sind, wird diese Bildmaske pro Bild im Trainingsdatensatz direkt bei der Erstellung des Datensatzes generiert (vgl. Kapitel 5.3.1, Abb. 5.4).

Auch bei der späteren quantitativen Beurteilung der Güte der Netze auf dem Validierungsdatensatz werden nur diejenigen Pixel in die Berechnung der jeweiligen Metriken einbezogen, die entsprechend der zugehörigen Bildmaske gültig sind.

5.5.3 Optimierungsverfahren

Zum Training neuronaler Netze wird in der Regel eine Variante des stochastischen Gradientenabstiegs zur Optimierung der Parameter des Netzes über die Ableitung der Lossfunktion verwendet.

In den letzten Jahren wurden verschiedene Erweiterungen dieser Methode vorgestellt. Aus dem Vergleich dieser Verfahren von Ruder (2016) geht hervor, dass die Effektivität der einzelnen Algorithmen je nach Quelle schwankt und demnach von den jeweils verwendeten Modellen abhängt. Für das Training der in dieser Arbeit entwickelten Netze wurden die Optimierungsverfahren RMSProp und Adam (Kingma und Ba, 2014) qualitativ evaluiert, wobei sich herausgestellt, dass sie unter ansonsten gleichen Umständen zu vergleichbar guten Ergebnissen führen, Adam dazu aber weniger Optimierungsschritte benötigt.

Daher werden die letztendlich genutzten Modelle mit Adam trainiert.

5.5.4 Entscheidungsfunktionen

Nach dem Training der neuronalen Netze liefert jedes von ihnen für ein gegebenes Bild pro Pixel pro mögliche Klasse einen Wert, die sogenannte Aktivierung. Um jetzt jedem Pixel tatsächlich eine oder mehrere Klassen zuzuordnen, braucht es eine Entscheidungsfunktion E, die für eine gegebene Ausgabe x eines Netzes und eine Klasse i einen booleschen Wert liefert, der jeweils besagt, ob der klassifizierte Pixel der entsprechenden Klasse zugehörig ist. Diese Funktion kann wie in Formel 5.13 gezeigt einfach durch Vergleich des Werts mit einem Schwellwert t_i realisiert werden.

$$E(x, i) = x \geq t_i \tag{5.13}$$

Durch die Wahl des Schwellwerts pro Klasse können innerhalb eines gewissen Ausmaßes die Eigenschaften der Klassifikation gesteuert werden: je geringer der Schwellwert, desto mehr der Klasse zugehörige Pixel werden gefunden, allerdings kommt es auch zu umso mehr Falscherkennungen. Demnach sollte der Schwellwert so gewählt werden, dass er einen idealen Kompromiss zwischen Genauigkeit und Trefferquote bietet.

Um diese Entscheidung zu automatisieren, wird das Verhalten der trainierten Netze auf dem jeweiligen Validierungsdatensatz betrachtet. Pro Klasse wird dabei der Wertebereich der Ak-

tivierungen ermittelt und eine gleichverteilte Menge von potentiellen Schwellwerten innerhalb dieses Wertebereichs bestimmt. Für jeden dieser potentiellen Schwellwerte wird dann die Konfusionsmatrix bei Klassifikation des Validierungsdatensatzes mit der resultierenden Entscheidungsfunktion ermittelt und darauf basierend mittels einer Heuristik der beste Schwellwert ausgewählt.

Die Konfusionsmatrix besteht dabei jeweils aus der Anzahl der richtig klassifizierten Vordergrundelemente (true positives, TP), der Anzahl der richtig klassifizierten Hintergrundelemente (true negatives, TN) und entsprechend den Anzahlen der falsch klassifizierten Vorder- (false negatives, FN) und Hintergrundelementen (false positives, FP). Aus diesen können dann die Metriken Precision (Formel 5.14) und Recall (Formel 5.15) berechnet werden, die quantitative Angaben von Genauigkeit beziehungsweise Trefferquote darstellen.

$$\text{Precision} := \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5.14}$$

$$\text{Recall} := \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.15}$$

Die Heuristik zur Wahl des besten Schwellwerts funktioniert nun – entsprechend des Entwurfskriteriums, dass im angestrebten Anwendungsbereich Precision wichtiger als Recall ist – derart, dass stets derjenige Schwellwert gewählt wird, der unter den Schwellwerten mit einem Recall größer als 10% die höchste Precision erzielt.

Erreicht keiner der Schwellwerte eine Precision von mindestens 10%, so wird angenommen, dass die Güte der Klassifikation für die entsprechende Klasse nicht ausreichend ist und der entsprechende Schwellwert auf Unendlich gesetzt.

5.5.5 Modellselektion

Nachdem die beiden definierten neuronalen Netze jeweils mit den beiden gewählten Lossfunktionen – gewichtete binäre Kreuzentropie und Focal Loss – viele Iterationen lang trainiert wurden, gilt es als nächstes, für jedes der Netze das beste der gelernten Modelle auszuwählen.

Beim Training wurden in regelmäßigen Abständen von 50 Epochen die aktuellen Modellparameter gespeichert, wobei eine Epoche die Anzahl der Iterationen bezeichnet, innerhalb derer jedes Beispiel des Trainingsdatensatzes je einmal zum Training verwendet wurde.

Um aus allen so erhaltenen Modellen pro Netz dasjenige mit den besten Klassifikationseigenschaften zu erhalten, findet eine quantitative Auswertung unter Nutzung des Validierungsdatensatzes statt. In der Literatur werden zu diesem Zweck häufig die Metriken Accuracy (siehe Formel 5.16) oder F_1-Score (siehe Formel 5.17) verwendet, wobei letzterer das harmonische Mittel aus Precision und Recall darstellt.

$$\text{Accuracy} := \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{5.16}$$

$$F_1 := 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.17}$$

Allerdings stößt insbesondere die Accuracy auf Probleme, wenn wie im vorliegenden Fall die Anzahl der den Vorder- beziehungsweise Hintergrundklassen zugeordneten Beispiele nicht ausgewogen ist, da sie TP und TN gleich stark gewichtet. Wie von Chicco (2017) gezeigt, ist auch der F_1-Score in solchen Fällen nicht optimal. Stattdessen schlägt er die Nutzung des in Formel 5.18 gezeigten Matthews Correlation Coefficient (MCC, vgl. Boughorbel, Jarray und El-Anbari, 2017) vor.

$$\text{MCC} := \frac{\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}} \tag{5.18}$$

Der Gesamtwert jeder der verwendeten Metriken für ein Modell wird dabei als Makro-Durchschnitt (vgl. Sokolova und Lapalme, 2009) berechnet, das heißt, dass zuerst die jeweilige Metrik pro Klasse berechnet wird und anschließend der Durchschnitt dieser Zwischenergebnisse gebildet wird. Alternativ wäre die Berechnung des Mikro-Durchschnitts möglich gewesen, wobei die Metrik nur einmal für die Summe der Konfusionsmatrizen aller Klassen berechnet werden würde, allerdings hat der Makro-Durchschnitt den Vorteil, dass dabei alle Klassen unabhängig von der Anzahl der ihnen zugehörigen Beispiele im Datensatz gleich stark gewichtet werden.

Jedes der Netze wurde mit jeder der Lossfunktionen jeweils 25 000 Epochen lang trainiert. Tab. 5.4 und Tab. 5.5 zeigen die jeweils besten Modelle in Hinblick auf die Werte der genannten Metriken für jedes der beiden neuronalen Netze. Gemäß der oben genannten Vorteile des MCC wurde dabei letztlich jeweils das Modell ausgewählt, für das diese Metrik den höchsten Wert hat.

Lossfunktion	**Trainingsepochen**	**Accuracy**	**F_1-Score**	**MCC**
FL	1 700	0,99946	0,16858	0,17338
FL	13 800	0,99937	0,18618	0,18998
WBCE	9 550	0,99950	0,15783	0,19341
WBCE	20 900	0,99944	0,19784	**0,19962**

Tabelle 5.4 Auflistung der jeweils besten gelernten Modelle des Linienerkennungsnetzes, aufsteigend sortiert nach MCC auf dem Validierungsdatensatz.

Lossfunktion	Trainingsepochen	Accuracy	F_1-Score	MCC
FL	8 200	0,78568	0,13062	0,15242
FL	5 700	0,73807	0,14953	0,16922
WBCE	4 350	0,95234	0,16760	0,18721
WBCE	9 100	0,85711	0,18519	0,19391
WBCE	5 850	0,86901	0,18113	**0,20070**

Tabelle 5.5 Auflistung der besten gelernten Modelle des Merkmalserkennungsnetzes, aufsteigend sortiert nach MCC auf dem Validierungsdatensatz.

5.6 Auswertung und Nachverarbeitung

Nachdem in den vorherigen Abschnitten zwei neuronale Netze entworfen und trainiert wurden, die Spielfeldmerkmale im IPM-Bild erkennen, beschreiben die nachfolgenden Abschnitte nun, wie diese Netze auf dem *NAO* angewendet und ausgewertet werden, sodass die Selbstlokalisierung des Roboters ihre Ergebnisse nutzen kann.

5.6.1 Inferenz

Gemäß der in Kapitel 5.2 definierten Problemstellung wurde jedes der beiden neuronalen Netze derart entwickelt, dass es ein IPM-Bild als Eingabe und ein Bild gleicher Auflösung als Ausgabe liefert.

Zur Inferenz der Netze auf dem *NAO* wird die im Kontext von B-Human entwickelte Bibliothek CompiledNN (Thielke und Hasselbring, to appear) genutzt. Diese lädt die Netze und ihre gelernten Parameter bei der ersten Anwendung zur Laufzeit ein und übersetzt sie in jeweils eine einzige C++-Funktion, die dann wiederholt für jedes neue Kamerabild aufgerufen wird. Dabei werden die Rechenoperationen mittels Instruktionen aus dem Streaming SIMD Extensions (SSE)-Befehlssatz für x86-Prozessoren umgesetzt, die in den zugehörigen XMM-Registern des Prozessors jeweils vier Fließkommazahlen auf einmal verarbeiten können (vgl. Intel Corporation, 2019). Deshalb ist es auch wichtig, dass die Anzahlen der Filter der Convolutional-Layer in den verwendeten Netzen jeweils Vielfache von vier sind, um eine optimale Performanz zu ermöglichen.

Wie in Kapitel 5.5.4 genannt, wurde pro Klasse jedes Netzes eine Entscheidungsfunktion definiert, die jeweils die entsprechenden Ausgaben der Netze mit einem Schwellwert vergleicht. Da somit ein Schwellwertvergleich pro Klasse gewünscht ist, kann dieser zur Laufzeit ohne zusätzlichen Aufwand realisiert werden, indem beim Kompilieren der Netze die Biases des letzten Convolutional-Layer jedes Netzes um die entsprechenden Schwellwerte reduziert werden. Die boolesche Klassenzugehörigkeit pro Pixel kann dann einfach durch Ermittlung des Vorzeichens jeder Aktivierung erhalten werden, für vier Klassen zugleich etwa durch Nutzung der SSE-Operation `movmskps`.

5.6.2 Auswertung des Linienerkennungsnetzes

Wie zuvor festgelegt ist die Ausgabe des Netzes zur Feldlinienerkennung ein Bild, dessen Pixel jeweils mehreren Feldlinienrichtungsklassen angehören können; der Normalfall ist allerdings, dass jeder Pixel genau einer oder keiner Klasse zugehörig ist.

Um aus diesen Daten nun die Positionen von Feldlinien ableiten zu können, werden alle Pixel betrachtet, die mindestens einer Klasse zugeordnet wurden. Nachdem für jeden dieser Pixel seine Feldlinienrichtung durch Bildung des Durchschnitts seiner Klassen ermittelt wurde, liegt wie in Abb. 5.6 gezeigt als Ergebnis eine Menge von Punkten vor, für die bekannt ist, dass sie auf einer Feldlinie mit einer bestimmten Ausrichtung liegen.

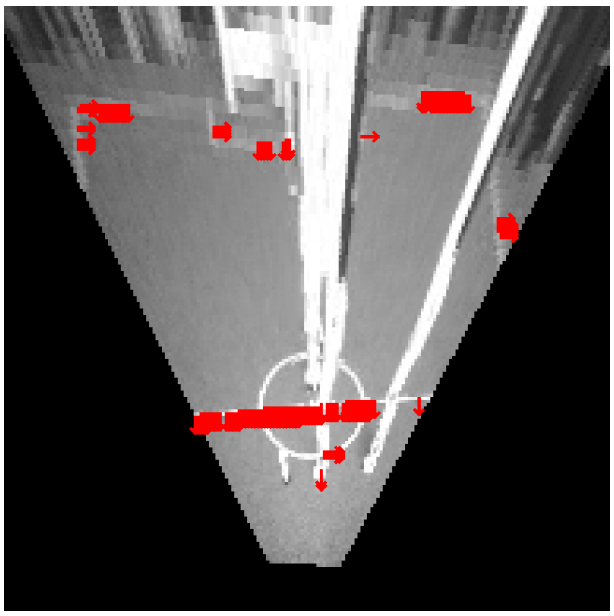


Abbildung 5.6 Vom neuronalen Netz erkannte Linienpunkte mit Orientierungen. Die Pfeilbasis liegt dabei auf dem jeweiligen Punkt, während die Pfeilspitze senkrecht zur Feldlinie zeigt, auf der der Punkt liegt.

Diese Linienpunkte werden im nächsten Schritt zu Linien zusammengefügt. Hierzu wird zunächst pro Punkt ein Linienkandidat gebildet, dessen beide Endpunkte an diesem Punkt liegen. Haben nun zwei Kandidaten eine bis auf einen Schwellwert ähnliche Richtung und liegen ihre Endpunkte sowohl nahe beieinander als auch in der Nähe der Geraden durch die jeweils andere Linie, so können diese zu einem gemeinsamen Linienkandidaten zusammengefasst werden. Dabei wird die optimale Linie durch die Punkte der beiden Linienkandidaten analog zum in Kapitel 4.2.3 beschriebenen Verfahren durch Ausgleichsrechnung mittels der Methode der kleinsten Quadrate erhalten. Die so erhaltenen Feldlinien zeigt Abb. 5.7.

Somit ist das Ergebnis schlussendlich wie auch beim in Kapitel 4.2.3 vorgestellten analytischen Ansatz eine Menge von Feldlinien, die an das restliche B-Human-System übergeben wird, welches aus diesen dann Linienkreuzungen und komplexere Feldmerkmale bildet.

Im Gegensatz sowohl zum analytischen Ansatz als auch zum Netz zur Merkmalserkennung

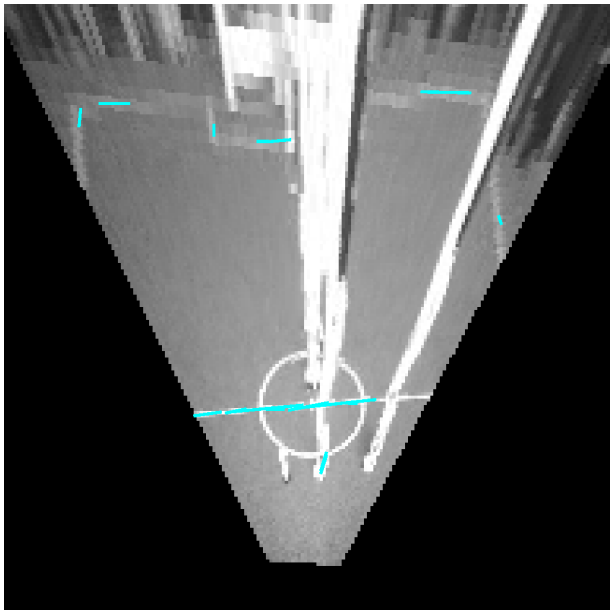


Abbildung 5.7 Aus den vom neuronalen Netz erkannten Linienpunkten gebildete Feldlinien.

kann mit diesem Verfahren der Mittelkreis nicht erkannt werden. Möglich wäre zwar, zu versuchen, analog zur Methode in Kapitel 4.2.4 in den Feldlinienkandidaten Kreise zu suchen, allerdings erkennt das Linienerkennungsnetz in der Regel auf dem Mittelkreis keine Feldlinienpunkte, da dieser auch in den Trainingsdaten dieses Netzes nicht als Feldlinie annotiert wurde.

5.6.3 Auswertung des Merkmalserkennungsnetzes

Das zweite trainierte neuronale Netz, welches direkt Merkmale in IPM-Bildern erkennt, gibt wie zuvor festgelegt ebenfalls ein Bild aus, dessen Pixel verschiedenen Klassen angehören können. In diesem Fall bedeuten die Klassen allerdings jeweils einen Merkmalstyp und eine Ausrichtung.

Die Auswertung dieses Ergebnisses besteht dabei aus zwei Schritten. Zunächst wird aus der Ausgabe des Netzes eine Liste von konkreten Merkmalspositionen und -ausrichtungen berechnet. Im zweiten Schritt müssen dann die so erhaltenen Bodenmerkmale noch zu den vom B-Human-System definierten Spielfeldmerkmalen umgerechnet werden, um sie an die bestehenden Verfahren zur Selbstlokalisierung übergeben zu können. Eine alternative Option hierzu wäre, den Selbstlokalisierungsteil des Roboterprogramms so anzupassen, dass er direkt die hier gefundenen Merkmalstypen verwendet; dies liegt jedoch außerhalb der Zielsetzung dieser Arbeit.

5.6.3.1 Extraktion von Merkmalen aus dem Ausgabebild

Das Verfahren zur Extraktion von Positionen und Richtungen einzelner Merkmale aus dem Ausgabebild des neuronalen Netzes beginnt nun damit, zunächst diejenigen Pixel zu identifizieren, die mindestens einer Klasse zugeordnet wurden. In der hier vorliegenden Implementierung wird dabei für Pixel, die mehreren Merkmalen zugeordnet sind, aus diesen nur eines ausgewählt. Dies ist dabei einfach dasjenige mit dem geringsten Index.

Als Ergebnis liegt nun pro Merkmalstyp wie in Abb. 5.8 zu sehen eine Menge von Punkten mit Ausrichtungen vor. Davon liegen häufig viele gleichartige Merkmale direkt nebeneinander – dies ist neben Ungenauigkeit in der Klassifikation auch der Entscheidung geschuldet, im Trainingsdatensatz für jedes Merkmal jeweils alle Pixel innerhalb eines bestimmten Radius um dieses als ihm zugehörig zu markieren (siehe Kapitel 5.3.1).

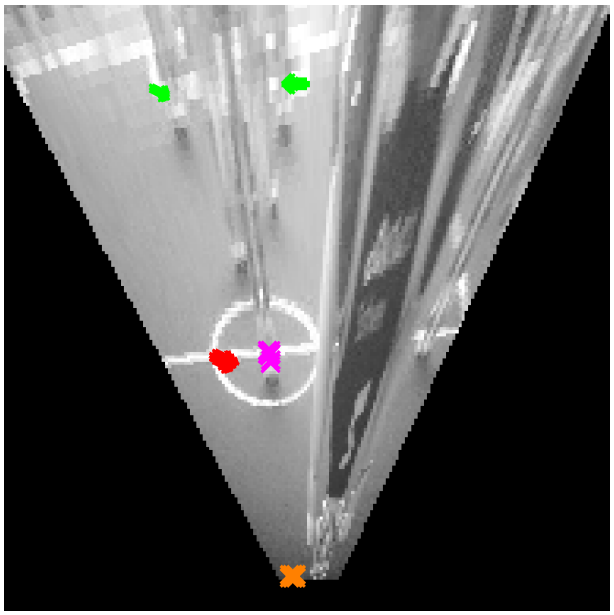


Abbildung 5.8 Vom neuronalen Netz erkannte Merkmalspunkte mit Ausrichtungen. Wie man sieht, werden viele gleichartige Merkmale nah beieinander gefunden.

Um eindeutige Merkmalspositionen zu erhalten, werden daher Cluster von gefundenen Merkmalen gebildet. Dabei wird jeder Merkmalspunkt jeweils dem ihm nächsten Cluster aus gleichartigen Merkmalen hinzugefügt, sofern die Distanz zum Clustermittelpunkt einen Schwellwert nicht übersteigt. Konnte ein Punkt keinem bestehenden Cluster zugeordnet werden, so wird ein neues für ihn erstellt.

Nachdem nun alle Merkmalspunkte zu Clustern gehören, wird pro Cluster jeweils der Durchschnitt aller Positionen und Ausrichtungen von den ihm zugeordneten Punkten gebildet und dieser als Pose eines echten Merkmals angesehen. Somit liegt nach diesem Schritt wie in Abb. 5.9 zu sehen die gewünschte Menge von Merkmalen vor, die im IPM-Bild erkannt wurden.

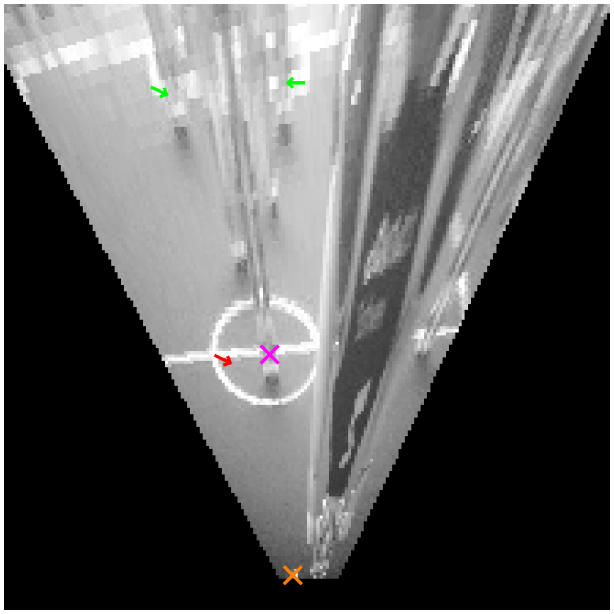


Abbildung 5.9 Merkmale mit Ausrichtungen, die nach dem Bilden von Clustern aus dem Ergebnis des neuronalen Netzes übrig bleiben. In diesem Fall hat der Torwartroboter von seiner Position aus einen Strafstoßpunkt, eine Mittelkreiskreuzung, das Mittelkreiszentrum und zwei Strafraumecken an den korrekten Orten detektiert.

5.6.3.2 Konvertierung der gefundenen Merkmale zu B-Human-Spielfeldmerkmalen

Zuletzt müssen aus den gefundenen Merkmalen die im B-Human-System bekannten Spielfeldmerkmale Mittelkreis, Mittelkreuzung, Strafraum, Strafstoßpunkt und Außenecke (vgl. Kapitel 4.1) gebildet werden.

Zunächst wird versucht, den Mittelkreis zu finden. Hierzu wird für jeden erkannten Mittelkreismittelpunkt ein Mittelkreiskandidat erstellt, wobei als Radius der bekannte Radius des Kreises auf dem Spielfeld verwendet wird. Im nächsten Schritt wird versucht, die erkannten Kreuzungen des Mittelkreises mit der Mittellinie den entsprechenden Kandidaten anhand ihrer Position und Ausrichtung zuzuordnen. Konnte zu einer solchen Kreuzung kein Mittelkreiskandidat gefunden werden, wird ein neuer erstellt. Da beobachtet wurde, dass die Erkennung von Mittelpunkten des Kreises durch das neuronale Netz genauer ist als die Erkennung der Kreuzungen, werden in der anschließenden Auswahl des echten Mittelkreises aus den Kandidaten diejenigen bevorzugt, die einen erkannten Mittelpunkt enthalten. Davon abgesehen funktioniert die Auswahl einfach derart, dass derjenige Kandidat gewählt wird, für den die meisten erkannten Merkmale sprechen. Gibt es zum gewählten Kandidaten mindestens eine Kreuzung mit der Mittellinie, so kann zusätzlich zu seiner Position auch seine Ausrichtung relativ zum Roboter bestimmt werden.

Die Abbildung der Mittelkreuzungen hingegen ist relativ einfach, da dieses vom B-Human-System verwendete Merkmal direkt vom Netz erkannt wird. Allerdings wurde beobachtet, dass das trainierte Netz gelegentlich Falscherkennungen in den Kreuzungen des Mittelkreises

mit der Mittellinie hat. Um diese zu eliminieren, werden erkannte Mittelkreuzungen nur übernommen, wenn sie mindestens einen festgelegten Abstand zum zuvor gefundenen Mittelkreis haben.

Um den Strafraum zu finden, liegen drei durch das neuronale Netz gefundene Merkmalstypen vor: die Kreuzungen der Torlinie mit dem Strafraum, die Strafraumecken und der Strafstoßpunkt. Die Kreuzungen von Strafraum und Torlinie sowie die Ecken des Strafraums können dabei jeweils zu genau zwei verschiedenen potentiellen Strafräumen gehören, in denen sie sich jeweils auf der linken oder rechten Seite befinden. Um den am besten zu den gefundenen Merkmalen passenden Strafraum zu finden, werden daher all diese möglichen Strafräume generiert. Die so erhaltene Menge von Kandidaten wird dann reduziert, indem in Position und Ausrichtung ähnliche Kandidaten vereint werden. Schließlich werden alle erkannten Strafstoßpunkte betrachtet. Passt einer von diesen zu einem der verbleibenden Strafraumkandidaten, so wird angenommen, dass dies der wahre Strafraum ist. Ansonsten wird der Kandidat mit den meisten ihm zugeordneten erkannten Merkmalen ausgewählt, wobei dies mindestens zwei Punkte sein müssen – wie bereits erwähnt, kann jede Strafraumecke und -kreuzung zu mehreren potentiellen Strafräumen gehören, sodass die Zuordnung des Strafraums zu einem Merkmalspunkt nicht eindeutig wäre. Wurde hingegen mit dieser Methode kein Strafraum gefunden, so werden die erkannten Strafstoßpunkte direkt an die Selbstlokalisierung weitergegeben, da sie auch allein Informationen über den Standort des Roboters auf dem Feld liefern können.

Die äußeren Ecken des Spielfelds werden ebenso wie die Mittelkreuzungen direkt vom trainierten neuronalen Netz erkannt. Allerdings ist die Ausrichtung dieser Merkmale im B-Human-System stets relativ zu einem Strafraum, sodass aus einem solchen Merkmal die Position des Roboters auf einer Spielfeldhälfte bereits eindeutig bestimmt werden kann. Daher werden die durch das Netz gefundenen Ecken mit dem durch das vorangegangene Verfahren bestimmten Strafraum verglichen, um ihre relative Ausrichtung zu ermitteln, bevor sie an das Selbstlokalisierungsverfahren weitergegeben werden.

Kapitel 6

Evaluation

Nachdem in den vorangegangenen Kapiteln drei verschiedene Verfahren zur Bestimmung von Positionen und Ausrichtungen von Bodenmerkmalen implementiert wurden, stellt sich nun die Frage, wie gut diese tatsächlich in der Lage sind, Spielfeldmerkmale in der RoboCup SPL zu erkennen.

Hierzu werden sie im Folgenden durch die Durchführung von Experimenten in Hinblick auf ihre Güte und Genauigkeit sowie die Laufzeit, die sie zur Ausführung benötigen, evaluiert. Dabei werden die Ergebnisse stets auch mit dem bisher im B-Human-System genutzten Verfahren verglichen.

6.1 Testverfahren

Um die Güte der verwendeten Verfahren qualitativ zu messen, wird ein Experiment durchgeführt, in dem alle vier Verfahren auf einem Roboter ausgeführt werden, der auf einem den Regeln der RoboCup SPL entsprechenden Spielfeld steht.

Allerdings stand für die Durchführung der Evaluation lediglich ein Spielfeld mit den Dimensionen 5×4 Meter zur Verfügung – im Gegensatz zum für Spiele in der SPL verwendeten Feld der Größe 9×6 Meter (vgl. RoboCup Technical Committee, 2019). Da sich jedoch weder das Aussehen noch die Größe der zu erkennenden Bodenmerkmale unterscheiden, kann davon ausgegangen werden, dass die Ergebnisse des Experiments auch für das größere Spielfeld gelten.

6.1.1 Versuchsaufbau

Zur Durchführung des Experiments wird ein Roboter an vorher definierten Positionen mit festgelegten Ausrichtungen auf dem Spielfeld platziert. Diese werden so gewählt, dass sie sowohl eine Übersicht über das Spielfeld ermöglichen, sodass der Roboter potentiell Merkmale sehen kann, als auch übliche Orte sind, an denen Roboter sich während eines Fußballspiels

häufig befinden. Die hierfür gewählten Orte sind wie in Abb. 6.1 gezeigt die Ecken des Spielfelds mit Blick nach innen, die Torlinie, das Zentrum des Spielfelds sowie die Mitte einer Feldhälfte jeweils einmal mit Ausrichtung in Richtung jedes der beiden Tore.

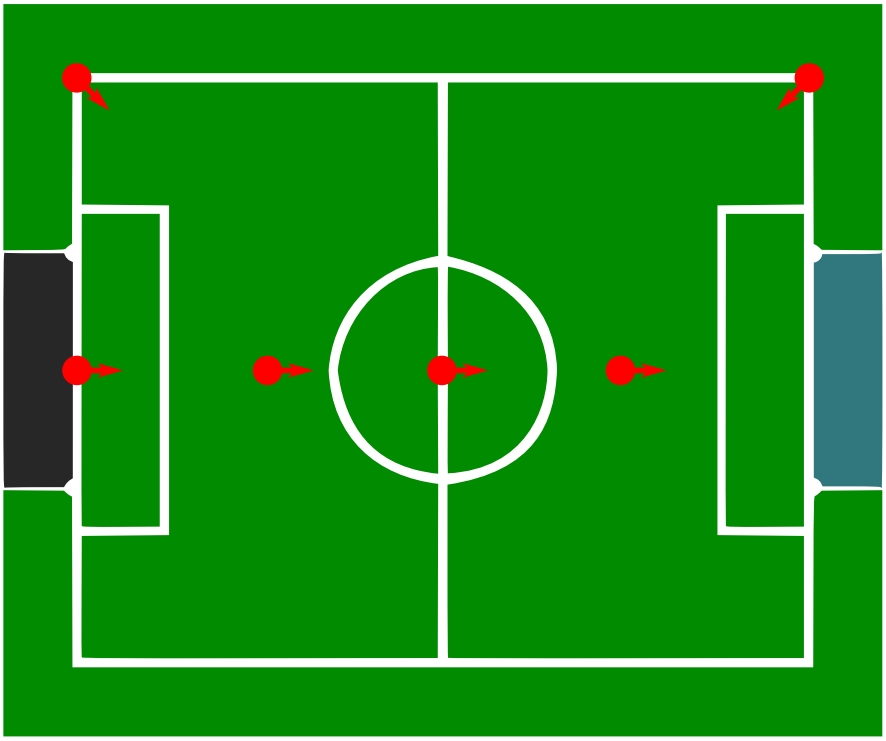


Abbildung 6.1 Übersicht über alle zur Evaluation genutzten Positionen eines Roboters auf dem Spielfeld. Die Pfeilrichtung gibt jeweils die Blickrichtung des Roboters an.

Dadurch, dass die Positionen und Ausrichtungen des Roboters auf dem Feld stets genau definiert sind, können durch Ausmessen des Spielfelds Ground-Truth-Daten für alle potentiell zu sehenden Bodenmerkmale berechnet werden, mit denen später die von den zu testenden Verfahren tatsächlich erkannten Merkmale verglichen werden.

Auf jeder der Positionen auf dem Spielfeld schaut der Roboter jeweils einen kurzen Zeitraum lang abwechselnd nach links und rechts, mit der gleichen Bewegung, die er auch in einem Fußballspiel verwenden würde. Während dieser Umschaubewegung werden jeweils die vier Verfahren zur Merkmalserkennung ausgeführt und ihre Ergebnisse ausgewertet. Dadurch, dass so ein Großteil der für den Versuch genutzten Kamerabilder in der Bewegung aufgenommen werden, sind diese entsprechend den meisten realen Situationen in einem Roboterfußballspiel verwischt und durch den Rolling-Shutter-Effekt verfälscht.

Überdies werden zwei weitere Roboter auf dem Feld als Hindernisse platziert. Diese sollen eine Ablenkung für die verwendeten Bildverarbeitungsalgorithmen entsprechend der Situationen in echten Spielen darstellen, ohne jedoch potentielle Erkennungen unmöglich zu machen. Daher sind ihre Positionen so gewählt, dass sie keines der Spielfeldmerkmale verdecken.

Einen Überblick über den Versuchsaufbau zeigt Abb. 6.2.

Abbildung 6.2 Versuchsaufbau für eine der Konfigurationen des Experiments. In der hier gezeigten Konfiguration ist der Roboter in der linken hinteren Ecke derjenige, auf dem das Evaluationsprogramm läuft, während die beiden anderen Ablenkungen für die Merkmalserkennung darstellen sollen.

6.1.2 Auswertung

Da sich in der RoboCup SPL gezeigt hat, dass die Erkennung von Objekten in Kamerabildern bei den meisten Algorithmen abhängig von der Entfernung zum Objekt ist, werden die Ergebnisse nach Entfernungen des Roboters vom jeweiligen Merkmal gruppiert. Hierzu werden die potentiellen Spielfeldmerkmale in drei Bereiche unterteilt: Nahbereich bis 2,5 m, mittlere Distanz von 2,5 m bis 5 m und größere Entfernungen als 5 m.

Zur Auswertung der gemessenen Daten wird nun pro Spielfeldmerkmal pro Distanzbereich eine Konfusionsmatrix (vgl. Kapitel 5.5.4) berechnet. Im B-Human-System kann aktuell allerdings nur maximal ein Merkmal pro Typ pro Bild als Ergebnis der Bildverarbeitung ausgegeben werden, auch wenn mehrere desselben Typs im Bild existieren – dies kommt zum Beispiel bei Strafstoßpunkten oder Außenecken vor. Daher ist die Definition der Elemente der Konfusionsmatrix wie folgt beschrieben entsprechend angepasst.

Pro Bild, Distanz und Merkmalstyp wird maximal ein Element der Konfusionsmatrix hochgezählt, und zwar:

- **True Positives**, wenn das Merkmal an einer Stelle gesehen wurde, an der sich wirklich ein solches befindet.

- **False Positives**, wenn das Merkmal an einer Stelle gesehen wurde, an der sich kein solches befindet.
- **False Negatives**, wenn das Merkmal im aktuellen Bild nicht gesehen wurde, obwohl es in der entsprechenden Distanz vorhanden ist.
- **True Negatives**, wenn das Merkmal im aktuellen Bild nicht gesehen wurde und es auch nicht im Bild vorhanden ist.

Als Toleranzbereich für die Zuordnung von erkannten Merkmalen zur Ground Truth wurde dabei ein Radius von einem Meter gewählt.

Damit die so erhaltenen Konfusionsmatrizen sinnvoll verglichen werden können, werden sie jeweils durch den Matthews Correlation Coefficient (MCC) zusammengefasst (vgl. Kapitel 5.5.5).

Um neben der Güte der Merkmalserkennung auch die Genauigkeit der einzelnen Verfahren beurteilen zu können, wird weiterhin für alle gefundenen True Positives ihre Abweichung vom entsprechenden Ground-Truth-Merkmal hinsichtlich der Position und Orientierung bestimmt.

Neben der gemeinsamen Auswertung der vier Verfahren wird anschließend zusätzlich das trainierte Merkmalserkennungsnetz noch einmal für sich evaluiert, indem die oben genannte Auswertung für die von diesem gefundenen Merkmale direkt angewendet wird. Dies hat den Hintergrund, dass die Umwandlung der durch das neuronale Netz gefundenen Feldmerkmale zu den vom B-Human-System verwendeten Merkmalstypen wie in Kapitel 5.6.3 genannt lediglich durchgeführt wird, um kompatibel zum aktuell verwendeten Selbstlokalisierungsverfahren zu sein; sollte sich in der Zukunft entschieden werden, das hier trainierte neuronale Netz zu verwenden, so sollten dessen erkannte Merkmale direkt vom Selbstlokalisierungsalgorithmus genutzt werden.

Diese vom neuronalen Netz gefundenen Merkmalsarten können überdies im Gegensatz zu den oben beschriebenen Merkmalstypen auch mehrfach pro Bild gesehen werden, sodass sich hierbei die Bestimmung der Konfusionsmatrix geringfügig ändert:

- **True Positives** sind alle gesehenen Merkmale, die an Stellen gesehen wurden, an denen sich wirklich ein entsprechendes Feldmerkmal befindet.
- **False Positives** sind alle gesehenen Merkmale, die an Stellen gesehen wurden, an denen sich kein entsprechendes Feldmerkmal befindet.
- **False Negatives** sind alle tatsächlich vorhandenen Merkmale, auf die kein detektiertes Merkmal abgebildet werden kann.
- **True Negatives** bezeichnen die Anzahl der Ereignisse, dass im entsprechenden Distanzbereich kein Merkmal gesehen wurde und auch keines existiert.

6.2 Ergebnisse

Bei der Durchführung des Experiments wurden mit einem Roboter an sechs Positionen auf dem Spielfeld insgesamt 4 550 Kamerabilder aufgenommen und ausgewertet. Die Tabellen 6.1, 6.2 und 6.3 zeigen die Ergebnisse des durchgeführten Experiments, welche im Folgenden qualitativ beurteilt werden.

6.2.1 Analytischer Ansatz

Im Hinblick auf den analytischen Ansatz zeigt sich, dass er in der Lage ist, alle Arten von Spielfeldmerkmalen auf alle Distanzen zu erkennen – abgesehen von den schon bei der Implementierung des Verfahrens nicht berücksichtigten Strafstoßpunkten. Allerdings ist er im direkten Vergleich in allen Fällen, in denen das bisherige Verfahren Merkmale erkennt, schlechter als dieses. Der große Vorteil des neuen analytischen Ansatzes zeigt sich daher darin, dass er auch über größere Distanzen als das alte Verfahren funktioniert.

Die Abweichungen der Positionen und Orientierungen erkannter Merkmale sind dabei stets mit dem bestehenden Verfahren vergleichbar; dies ist allerdings auch nicht verwunderlich, da der neue Ansatz nur Mittelkreise direkt erkennt und die restlichen Merkmale aus erkannten Feldlinien mit demselben Algorithmus und daher unter den gleichen Bedingungen wie beim bestehenden Verfahren generiert werden.

6.2.2 Linienerkennungsnetz

Die Ergebnisse des neuronalen Netzes zur Linienerkennung in Tab. 6.1 fallen auf, da sie stets gleich Null sind: bei der Durchführung des Experiments wurden durch Nutzung dieses Verfahrens keinerlei Spielfeldmerkmale erkannt. Daher konnte auch keine Auswertung der Abweichungen erkannter Merkmalspositionen und -orientierungen durchgeführt werden.

Betrachtet man die durch das neuronale Netz gefundenen Feldlinien, fällt auf, woran diese schlechten Ergebnisse liegen: die gefundenen Linien sind sehr kurz und umfassen wie in Abb. 6.3 zu sehen in der Regel nur kurze Abschnitte von tatsächlichen Feldlinien. Da außerdem ausgerechnet an Feldlinienkreuzungen, also an für Spielfeldmerkmale besonders aussagekräftigen Punkten, nur sehr selten eine Aktivierung des Netzes vorliegt, ist es nicht verwunderlich, dass keinerlei Bodenmerkmale aus den gefundenen Linien rekonstruiert werden können.

Allerdings fällt bei einer qualitativen Betrachtung der Ausgaben des neuronalen Netzes auch auf, das die gefundenen Feldlinienabschnitte nur an Orten liegen, an denen tatsächliche Feldlinien sich befinden, und zwar unabhängig von ihrer Entfernung zur Kamera. Teilweise gelingt es sogar, von anderen Robotern verdeckte Linien durchgängig zu rekonstruieren (siehe Abb. 6.3).

Abbildung 6.3 Vom Linienerkennungsnetz gesehene, in das Kamerabild zurückprojizierte Feldlinien. Es zeigt sich, dass Linienabschnitte zwar korrekt erkannt werden – teils sogar wie im rechten Bild durch Hindernisse hindurch –, diese aber in der Regel zu kurz sind, um aus ihnen komplexere Bodenmerkmale zu bilden.

Detektion (MCC)	**Bisherig**	**Analytisch**	**NN Linien**	**NN Merkmale**
Bis 2,5 m entfernt				
Mittelkreis	**0,986117**	0,477544	-	0,713739
Mittelkreuzung	**0,882038**	0,510237	0	0,379731
Außenecke	**0,326616**	0,095481	0	0,270217
Strafraum	**0,248207**	0,210672	0	−0,009052
Strafstoßpunkt	**0,452100**	-	-	0,264012
Gesamt	**0,579016**	0,323483	0	0,323729
2,5 m – 5 m entfernt				
Mittelkreis	0,559096	0,376663	-	**0,826498**
Mittelkreuzung	**0,473356**	0,364068	0	0,217202
Außenecke	0	0,127926	0	**0,273466**
Strafraum	0	**0,475619**	0	0,037923
Strafstoßpunkt	0	-	-	**0,583929**
Gesamt	0,206490	0,336069	0	**0,387804**
Über 5 m entfernt				
Außenecke	0	0,029803	0	**0,316151**
Strafraum	0	**0,118277**	0	0,087918
Gesamt	0	0,074040	0	**0,202035**

Tabelle 6.1 Matthews Correlation Coefficient der Erkennung von Feldmerkmalen für die bisherige Methodik sowie jeden der drei in dieser Arbeit implementierten Ansätze, gruppiert nach der Entfernung der jeweiligen Merkmale vom Roboter.
Weder der analytische Ansatz noch das neuronale Netz zur Linienerkennung sind aktuell in der Lage, Strafstoßpunkte zu finden; überdies können mit dem Linienerkennungsnetz auch Mittelkreise nicht erkannt werden.

Abw. Position (cm)	**Bisherig**			**Analytisch**			**NN Merkmale**		
	Min	Max	∅	Min	Max	∅	Min	Max	∅
Bis 2,5 m entfernt									
Mittelkreis	0,5	13,8	5,8	0,9	10,0	5,6	0,6	30,4	**4,1**
Mittelkreuzung	1,2	12,4	7,0	2,3	10,6	7,2	0,9	7,7	**4,7**
Außenecke	1,4	8,4	4,9	4,2	7,3	5,7	0,8	9,7	**4,7**
Strafraum	25,8	29,1	**27,0**	24,5	43,3	34,0	-	-	-
Strafstoßpunkt	4,3	7,4	**5,5**	-	-	-	0,5	99,6	27,3
2,5 m – 5 m entfernt									
Mittelkreis	4,4	20,6	**11,0**	3,8	21,3	11,5	3,2	69,4	21,9
Mittelkreuzung	3,2	42,4	17,2	6,2	38,3	**16,2**	1,0	48,8	17,9
Außenecke	-	-	-	8,0	49,0	**21,1**	3,5	85,0	22,2
Strafraum	-	-	-	8,1	30,7	**20,9**	22,5	42,5	37,5
Strafstoßpunkt	-	-	-	-	-	-	4,9	89,0	**29,4**
Über 5 m entfernt									
Außenecke	-	-	-	21,7	29,5	**27,0**	17,5	99,4	55,8
Strafraum	-	-	-	7,9	20,9	**14,7**	14,0	36,9	23,8

Tabelle 6.2 Abweichungen der Positionen von korrekt erkannten Spielfeldmerkmalen zu ihren bekannten echten Positionen in Zentimeter für die bisherige Methodik sowie zwei der in dieser Arbeit implementierten Ansätze, gruppiert nach der Entfernung der jeweiligen Merkmale vom Roboter.
Das neuronale Netz zur Linienerkennung ist ausgeschlossen, da mit ihm keinerlei Merkmale erkannt wurden.

Abw. Orientierung (°)	**Bisherig**			**Analytisch**			**NN Merkmale**		
	Min	Max	∅	Min	Max	∅	Min	Max	∅
Bis 2,5 m entfernt									
Mittelkreis	0,00	2,36	0,57	0,00	2,15	**0,55**	0,00	20,21	18,59
Mittelkreuzung	0,00	3,94	**1,02**	0,00	3,43	1,10	0,00	17,30	14,06
Außenecke	0,00	1,33	**0,55**	0,00	1,93	1,14	0,00	154,55	81,63
Strafraum	0,00	1,32	**0,48**	0,00	4,39	1,87	-	-	-
2,5 m – 5 m entfernt									
Mittelkreis	0,00	5,86	**2,22**	0,00	16,92	2,24	0,00	53,81	19,81
Mittelkreuzung	0,00	8,49	1,95	0,00	8,91	**1,93**	0,00	80,69	29,65
Außenecke	-	-	-	0,00	5,63	**1,32**	0,00	143,86	33,81
Strafraum	-	-	-	0,00	6,48	**1,49**	0,00	127,13	31,71
Über 5 m entfernt									
Außenecke	-	-	-	0,00	1,21	**0,55**	0,00	142,1	28,70
Strafraum	-	-	-	0,00	7,08	**2,35**	0,00	20,23	15,50

Tabelle 6.3 Abweichungen der Orientierungen von korrekt erkannten Spielfeldmerkmalen zu ihren bekannten echten Ausrichtungen in Grad für die bisherige Methodik sowie zwei der in dieser Arbeit implementierten Ansätze, gruppiert nach der Entfernung der jeweiligen Merkmale vom Roboter.
Das neuronale Netz zur Linienerkennung ist ausgeschlossen, da mit ihm keinerlei Merkmale erkannt wurden.

6.2.3 Merkmalserkennungsnetz

Im Gegensatz zum Linienerkennungsnetz zeigt sich, dass das andere neuronale Netz, welches direkt Spielfeldmerkmale in IPM-Bildern findet, diese auf dem gesamten Spielfeld gut erkennen kann.

Während die Ergebnisse dieses Ansatzes zwar im Nahbereich durchgängig schlechter sind als das bisherige Verfahren, sind sie für Distanzen von mehr als 2,5 Meter Entfernung allerdings besser als die aller anderen getesteten Verfahren; insbesondere werden Mittelkreise, Strafstoßpunkte und Außenecken teilweise deutlich besser als von den anderen Ansätzen gefunden.

Lediglich Strafräume werden im Vergleich zu den beiden analytischen Ansätzen sehr viel seltener erkannt – im Nahbereich sind sogar die drei einzigen gesehenen Strafräume False Positives, was in der Auswertung zu einem negativen MCC führt.

Außerdem schwanken die Positions- und Orientierungsabweichungen gesehener Merkmale deutlich stärker und sind abgesehen vom Nahbereich allgemein höher. Insbesondere die Orientierungen von Außenecken sind nicht verlässlich, da sie durchschnittlich um circa 80° von der Ground Truth abweichen. Bei Strafstoßpunkten kommt außerdem das Problem hinzu, dass in der Nähe von echten Merkmalen häufig weitere False Positives gesehen werden.

Betrachtet man nun die in Tab. 6.4 gezeigte Evaluation der vom neuronalen Netz direkt gefundenen Merkmale, so fällt zunächst auf, dass das Zentrum des Mittelkreises in allen Distanzen sehr gut erkannt wird, die Kreuzungen von Mittelkreis und Mittellinie jedoch nicht – insbesondere im Nahbereich ist deren Erkennung sehr schlecht, sodass die Orientierung des Mittelkreises nur auf größere Distanzen überhaupt bestimmt werden kann.

Da in dieser Auswertung sämtliche potentielle Merkmale statt nur einem pro Typ pro Bild berücksichtigt werden, lässt sich aus Tab. 6.4 weiterhin ableiten, dass die Erkennung von Strafstoßpunkten schlechter ist, wenn es nicht genügt, nur einen von beiden zu finden.

Außerdem zeigt sich, dass die Erkennung von Strafraumecken und Kreuzungen von Strafraum und Torlinie deutlich besser ist, als man durch die schlechten Werte des Strafraums in Tab. 6.1 glauben könnte. Dass im Experiment aus diesen an den richtigen Orten gefundenen Teilmerkmalen des Strafraums nur selten ein gemeinsames Merkmal gebildet werden konnte, scheint auch in diesem Fall an der eher schlechten Detektion der Orientierung der Merkmale zu liegen: diese weicht bei Strafraumecken in den Testdaten um bis zu fast 180° von der Ground Truth ab.

Insgesamt lässt sich sagen, dass das Merkmalserkennungsnetz am besten auf mittlere Distanz funktioniert. Dies legt die Vermutung nahe, dass im zum Training des Modells verwendeten Datensatz zu wenige Merkmale in kurzer oder weiter Entfernung vorhanden sind.

NN Merkmale	MCC	Abw. Position (cm)			Abw. Orientierung (°)		
		Min	Max	∅	Min	Max	∅
Bis 2,5 m entfernt							
Mittelkreiskreuzung	0,089816	2,0	6,4	3,3	0,00	20,21	18,59
Strafraumecke	0,307734	0,3	85,8	10,4	0,00	177,04	40,73
Strafraumkreuzung	0,164628	0,9	92,9	17,8	0,00	61,25	39,89
Außenecke	0,309349	0,8	9,7	4,7	0,00	96,84	81,39
Mittelkreuzung	0,384543	0,9	7,7	4,7	0,00	17,30	14,06
Mittelkreiszentrum	0,712651	0,6	9,9	3,4	-	-	-
Strafstoßpunkt	0,231113	0,5	99,6	28,9	-	-	-
2,5 m − 5 m entfernt							
Mittelkreiskreuzung	0,411118	3,4	33,5	10,5	0,00	54,25	23,59
Strafraumecke	0,489455	2,0	81,9	26,3	0,00	177,15	55,41
Strafraumkreuzung	0,365213	2,5	40,7	15,9	0,00	100,5	53,91
Außenecke	0,407452	3,5	99,2	22,3	0,00	119,12	69,20
Mittelkreuzung	0,221524	1,0	48,8	17,9	0,00	80,69	29,65
Mittelkreiszentrum	0,820318	2,0	26,7	13,3	-	-	-
Strafstoßpunkt	0,334392	4,9	89,0	27,5	-	-	-
Über 5 m entfernt							
Strafraumecke	0,427694	21,1	85,3	46,9	0,00	62,95	20,97
Strafraumkreuzung	0,341467	8,5	96,4	34,5	0,00	104,86	65,20
Außenecke	0,370768	17,5	99,4	55,3	0,00	124,92	76,88

Tabelle 6.4 Quantitative Auswertung der Detektion von Spielfeldmerkmalen durch das Merkmalserkennungsnetz, gruppiert nach der Entfernung der jeweiligen Merkmale vom Roboter. Gezeigt sind der Matthews Correlation Coefficient als Metrik für die Güte der Erkennung sowie die Abweichungen der korrekt erkannten Bodenmerkmale zu ihren bekannten echten Positionen und Orientierungen.

6.2.4 Zusammenfassung

Die Ergebnisse des Experiments zeigen, dass der bisher vom Team B-Human verwendete Ansatz zur Erkennung von Bodenmerkmalen auf kurze Distanzen bis 2,5 Meter Entfernung vom Roboter ungeschlagen ist. Für weiter entfernte Feldmerkmale sind aber sowohl der neue analytische Ansatz als auch das neuronale Netz zur Merkmalserkennung besser geeignet, wobei die von ersterem erkannten Positionen und Orientierungen eine deutlich höhere Genauigkeit haben, aber dafür das neuronale Netz eine bessere Erkennungsrate hat.

Weiter entfernte Spielfeldmerkmale erkennen zu können ist insbesondere deshalb von Vorteil, da auf dem üblicherweise in Spielen der RoboCup SPL verwendete Spielfeld – im Gegensatz zum für das Experiment verwendeten kleineren Spielfeld – für die Selbstlokalisierung der Roboter relevante Merkmale derart weit auseinanderliegen, dass häufig keines von ihnen innerhalb von 2,5 Metern Entfernung um den Roboter ist (siehe Dimensionen des Spielfelds in RoboCup Technical Committee, 2019). Eine verlässliche Detektion über größere Distanzen würde dem Roboter in solchen Situationen erlauben, sich dennoch gut auf dem Feld zu lokalisieren.

Das neuronale Netz zur Linienerkennung hingegen ist im aktuellen Zustand ungeeignet für die

Verwendung mit der bisherigen Form von Merkmalserkennung im B-Human-System. Da aber die von ihm gefundenen Linien in der Regel korrekten Feldlinienabschnitten entsprechen, wäre es vermutlich möglich, dieses Verfahren zu nutzen, um eine bestehende Selbstlokalisierung zu verbessern, indem die erkannten sicheren Feldlinienabschnitte dem Selbstlokalisierungsverfahren zur Verfügung gestellt werden.

6.3 Performanz

Da die Zielplattform der in dieser Arbeit implementierten Verfahren ein autonomer Roboter ist, der mit seinen beschränkten Ressourcen noch viele weitere Aufgaben mehrfach pro Sekunde leisten muss, um erfolgreich Fußball zu spielen, stellt die Laufzeit der Implementierung einen weiteren relevanten Faktor zur Beurteilung der Verfahren dar.

Um diese zu testen, wird ein Roboter in einer dem oben beschriebenen Experiment entsprechenden Umgebung über das Spielfeld laufen gelassen, wobei nacheinander jeweils einen kurzen Zeitraum von etwa 30 Sekunden lang jedes der drei implementierten Verfahren sowie zum Vergleich die bisherige Lösung ausgeführt wird. Dabei werden jeweils die Laufzeiten der einzelnen Verfahren pro verarbeitetes Kamerabild gemessen.

Das Ergebnis dieses Experiments zeigt Tab. 6.5. Hierbei zeigt sich zunächst, dass die Zeit zur Inferenz der neuronalen Netze stets ungefähr gleichbleibend ist, unabhängig vom Bildinhalt. Dies ist zu erwarten, da dabei für jedes Eingabebild stets dieselben Berechnungen entsprechend der gelernten Modelle durchgeführt werden. Die Ausführungszeit der IPM hingegen schwankt, abhängig davon, wie viel vom das Bild ausmachenden Bereich im Kamerabild zu sehen ist.

Andererseits ist zu sehen, dass die beiden Verfahren, die neuronale Netze benutzen, um circa Faktor 25 langsamer als die anderen Verfahren sind. Bei Nutzung dieser Verfahren können von den 30 pro Sekunde vorliegenden Kamerabildern nur etwa neun bis zehn verarbeitet werden. Es ist zwar davon auszugehen, dass dieser Takt für eine erfolgreiche Selbstlokalisierung des Roboters vollkommen ausreicht; allerdings bedeutet diese lange Laufzeit auch, dass es sich nur lohnt, eines dieser Verfahren einzusetzen, wenn dadurch deutlich bessere Ergebnisse als durch andere Verfahren erhalten werden.

Laufzeiten (ms)	**Min**	**Max**	**⌀**
Bisherig	2,46	3,97	2,81
Analytisch	2,44	2,76	2,52
NN Linien	77,96	80,35	78,83
Inferenz	69,99	71,31	70,58
Auswertung	7,87	9,13	8,21
NN Merkmale	89,26	90,54	89,83
Inferenz	73,64	74,82	74,21
Auswertung	15,45	15,71	15,58
IPM	0,53	1,69	1,54

Tabelle 6.5 Auflistung der minimalen, maximalen und durchschnittlichen Laufzeiten der vier betrachteten Verfahren zur Merkmalserkennung auf dem *NAO* in einer typischen Spielsituation. Die Berechnung des IPM-Bilds ist dabei separat aufgelistet, da dieses für alle drei neu implementierten Ansätze gebraucht wird.

Kapitel 7

Zusammenfassung und Ausblick

In der vorliegenden Abschlussarbeit wurde nach Lösungen für eine Fragestellung im Kontext der Bildverarbeitung im Roboterfußball gesucht. Nachdem mehrere Herangehensweisen an dieses Problem umgesetzt und evaluiert wurden, findet nachfolgend eine Auswertung der erhaltenen Ergebnisse in Bezug auf die ursprüngliche Fragestellung statt und es wird ein Ausblick auf mögliche Erweiterungen oder Verbesserungen der entwickelten Methoden gegeben.

7.1 Zusammenfassung

Die Ausgangsfrage dieser Arbeit war, ob durch rechenintensivere Bildverarbeitungsalgorithmen als die bisher in der RoboCup SPL verwendeten eine Verbesserung der Selbstlokalisierung möglich ist.

Um diese zu klären, wurde zunächst als Grundlage dieser neuen Algorithmen die Inverse Perspective Mapping eingeführt, welche ein Bild erzeugt, in dem die für die Selbstlokalisierung relevanten Bodenmerkmale unabhängig von ihrer Position relativ zum Roboter stets ähnlich aussehen.

Basierend darauf wurden drei verschiedene Verfahren implementiert:

- Ein analytischer Ansatz, der dieselbe Grundidee wie der bereits zuvor von B-Human verwendete Ansatz verfolgt und diese auf das IPM-Bild anwendet.
- Ein auf maschinellem Lernen basierender Ansatz, der durch Anwendung eines neuronalen Netzes Feldlinien im IPM-Bild erkennt und sich für das Finden von Lokalisierungsmerkmalen in diesen Linien auf das bestehende B-Human-System verlässt.
- Ein auf maschinellem Lernen basierender Ansatz, der durch Anwendung eines neuronalen Netzes direkt für die Selbstlokalisierung des Roboters brauchbare Merkmale im IPM-Bild findet.

Durch Evaluation dieser Verfahren zeigt sich, dass die Ausgangsfrage grundsätzlich mit „ja“

beantwortet werden kann: sowohl der neue analytische Ansatz als auch das neuronale Netz zur Merkmalserkennung sind aufgrund ihrer Eigenschaft, Spielfeldmerkmale auch über größere Distanzen mit ausreichender Genauigkeit zu detektieren, gut dazu geeignet, die Selbstlokalisierung zu verbessern. Allerdings sollte keines der beiden Verfahren im aktuellen Zustand ohne weitere Verbesserungen das bestehende Verfahren ersetzen, da letzteres im Nahbereich durchgängig bessere Ergebnisse als die neuen Methoden erzielt.

7.2 Ausblick

Jedes der drei in dieser Arbeit implementierten Verfahren bietet noch Potenzial für Verbesserungen. Einige Ideen hierzu sind im Folgenden beschrieben.

7.2.1 Linienerkennungsnetz

Zunächst soll hier auf das neuronale Netz zur Linienerkennung eingegangen werden, da es bei der Evaluation die schlechtesten Ergebnisse lieferte. Das Hauptproblem dieses Ansatzes ist, dass er wenige, kurze, aber korrekte Liniensegmente als Ergebnis liefert.

Daher kann einerseits versucht werden, beim Training der Modelle weniger Wert auf die Precision zu legen, sondern stattdessen durch Wahl der Schwellwerte in der Entscheidungsfunktion einen höheren Recall zu erreichen. Dadurch würden zwar auch falsche Linienpixel erkannt, aber es würden insgesamt mehr Linien gesehen werden, aus denen dann potentiell komplexere Merkmale gebildet werden können.

Andererseits könnte das vom neuronalen Netz zu lösende Problem statt als Klassifikationsproblem als Regression einer Heatmap wie bei Schnekenburger u. a. (2017) definiert werden und folglich der mittlere absolute Fehler oder der mittlere quadratische Fehler beim Training als Lossfunktion verwendet werden, um so das Training des Netzes zu vereinfachen und als Resultat weniger genaue, aber besser definierte Linienverläufe zu erhalten.

Eine weitere Option wäre, das Verfahren im aktuellen Zustand zu belassen, aber es nur zum Finden von sicheren Linienabschnitten zu benutzen, die im Anschluss durch einen weiteren Algorithmus noch verlängert werden.

Neben den genannten Verbesserungsansätzen wäre außerdem das Erkennen von Mittelkreisen eine sinnvolle Erweiterung des Linienerkennungsnetzes. Dies könnte relativ einfach umgesetzt werden, indem für das Training auch Kreisbögen im verwendeten Datensatz annotiert werden und anschließend bei der Nachverarbeitung der Ausgabe aus Linienpunkten mit Orientierung in Richtung des gleichen Mittelpunktes per Ausgleichsrechnung die Position des Mittelkreises bestimmt wird.

7.2.2 Analytischer Ansatz

Der neu implementierte analytische Ansatz funktionierte grundsätzlich gut, jedoch erkennt er insgesamt weniger Merkmale als das bisherige analytische Verfahren. Dies liegt teilweise daran, dass eigentlich erkannte Linien durch die Heuristik ausgefiltert werden, die Fehlerkennungen in Robotern oder Bällen vermeiden soll. Daher sollte hierfür eine bessere Heuristik gefunden werden – bei ausreichend guter Ball- und Robotererkennung könnten zum Beispiel minimal umfassende Rechtecke um diese Objekte beim Detektieren von Linienpunkten ausgeschlossen werden.

Außerdem können durch Anpassung der in den verwendeten Algorithmen definierten Parameter vermutlich noch bessere Ergebnisse erzielt werden. Auch kann man verschiedene Auflösungen des verwendeten IPM-Bilds evaluieren.

7.2.3 Merkmalserkennungsnetz

Bei der Evaluation des Merkmalserkennungsnetzes zeigte sich, dass dessen Qualität vermutlich von einer größeren Anzahl Trainingsdaten insbesondere von Merkmalen in kurzer und weiter Distanz profitieren würde. Für ein neues Training sollten also mehr Daten annotiert werden; vermutlich wäre es außerdem sinnvoll, wie Szemenyei und Estivill-Castro (to appear) oder Poppinga und Laue (to appear) synthetische Daten zum Training zu benutzen, da diese mit wenig Aufwand erzeugt und zugleich annotiert werden können.

Außerdem hatte dieser Ansatz Probleme mit der korrekten Bestimmung der Orientierung von erkannten Spielfeldmerkmalen. Daher sollte das Verfahren zum Bestimmen von Richtungen überdacht werden. Eine Alternative zum genutzten Verfahren, verschiedene Orientierungen als verschiedene Klassen der gewünschten Klassifikation zu behandeln, wäre zum Beispiel, zwei Ausgabekanäle für das Netz zu definieren, wobei ein Kanal jeden Pixel einer Klasse zuordnet und der andere bei Vorhandensein eines Merkmals dessen Orientierung schätzt.

7.2.4 Laufzeit der neuronalen Netze

Die Evaluation der implementierten Verfahren hat gezeigt, dass bei Verwendung eines der Verfahren, die ein neuronales Netz nutzen, aufgrund der langen Laufzeit nicht alle verfügbaren Kamerabilder ausgewertet werden können.

Dies ist zur Selbstlokalisierung grundsätzlich nicht schlimm, jedoch würde eine kürzere Laufzeit gerade bei sich bewegenden Robotern zu einer stabileren Positionsbestimmung führen. Außerdem blockiert aufgrund der aktuellen Architektur des B-Human-Systems die langsame Auswertung der neuronalen Netze auch andere Bildverarbeitungsaufgaben wie die Erkennung des Balls, was zu Nachteilen im Fußballspiel führen kann.

Allerdings wären verschiedene Möglichkeiten zur Verbesserung der Laufzeiten denkbar. Zunächst einmal liefern die aktuell verwendeten Netze als Ausgabe ein Bild mit den gleichen

Dimensionen wie die Eingabe; vermutlich ist die Erkennung von Merkmalen aber auch bei einer kleineren Ausgabe bereits hinreichend genau. Die Entfernung des letzten Hochskalierungsschritts in den Netzen würde viel Zeit bei der Inferenz sparen, da dadurch die letzten Operationen des Netzes jeweils auf einem Bilds mit nur einem Viertel der Pixel ausgeführt werden müssen.

Während davon abgesehen die Architektur der verwendeten neuronalen Netze bereits sehr klein ist und daher nicht reduziert werden sollte, könnten entsprechend des Ansatzes der MobileNets (siehe Howard u. a., 2017) separierbare Convolutional-Layer verwendet werden, um die Anzahl der gelernten Parameter und damit der Speicherzugriffe bei der Auswertung zu reduzieren.

Eine weitere Methode zur Reduktion der Speicherzugriffe ist die Quantisierung der Eingaben und Gewichte des Netzes wie zum Beispiel von Krishnamoorthi (2018) genannt, sodass die Berechnungen zur Inferenz nicht mit Fließkommazahlen, sondern Ganzzahlen durchgeführt werden. Bei Nutzung von 8-Bit-Ganzzahlen etwa würde nur ein Viertel des Speicherplatzes für Gewichte und Zwischenergebnisse gebraucht, wobei dank der 128 Bit breiten XMM-Register des verwendeten Prozessors dadurch auch nur ein Viertel der Speicherzugriffe stattfindet.

Weiterhin kann in Zukunft der Auswertungsschritt, der ebenfalls einen nicht zu vernachlässigenden Anteil an der Ausführungszeit der beiden neuronale Netze nutzenden Methoden hat, direkt in die Inferenz der Netze integriert werden. Dies würde zu Verbesserungen der Laufzeit führen, da bei der Berechnung des letzten Layers jeweils direkt die Kanäle mit Aktivierungen identifiziert werden könnten, während ihre Werte sich ohnehin noch in den Registern des Prozessors befinden.

Literatur

Abadi, Martín u. a. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software erhältlich unter tensorflow.org. URL: https://www.tensorflow.org/.

Albani, Dario u. a. (2017). „A Deep Learning Approach for Object Recognition with NAO Soccer Robots“. In: RoboCup 2016: Robot World Cup XX. Hrsg. von Sven Behnke u. a. Cham: Springer International Publishing, S. 392–403. ISBN: 978-3-319-68792-6.

Aly, Mohamed (Juni 2008). „Real time detection of lane markers in urban streets“. In: 2008 IEEE Intelligent Vehicles Symposium. DOI: 10.1109/ivs.2008.4621152. URL: http://dx.doi.org/10.1109/IVS.2008.4621152.

Badrinarayanan, Vijay, Alex Kendall und Roberto Cipolla (Dez. 2017). „SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation“. In: IEEE Transactions on Pattern Analysis and Machine Intelligence 39.12, S. 2481–2495. ISSN: 1939-3539. DOI: 10.1109/tpami.2016.2644615. URL: http://dx.doi.org/10.1109/TPAMI.2016.2644615.

Boughorbel, S, Fethi Jarray und Mohammed El-Anbari (Juni 2017). „Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric“. In: PLOS ONE 12, e0177678. DOI: 10.1371/journal.pone.0177678.

Bourke, Paul (1990). Equation of a Circle from 3 Points (2 dimensions). http://paulbourke.net/geometry/circlesphere/. Abgerufen am 12.08.2019.

Bullock, Randy (2006). „Least-squares circle fit“. In: Developmental Testbed Center. URL: https://dtcenter.org/met/users/docs/write_ups/circle_fit.pdf.

Burger, Wilhelm und Mark James Burge (2015). Digitale Bildverarbeitung: eine algorithmische Einführung mit Java. 3., vollständig überarb. und erw. Aufl. X.media.press. XXIII, 801 S. ; 242 mm x 193 mm : Ill., graph. Darst. Berlin: Springer Vieweg. ISBN: 3642046037 and 9783642046032 and 9783642046032. URL: http://dx.doi.org/10.1007/978-3-642-04604-9.

Chicco, Davide (Dez. 2017). „Ten quick tips for machine learning in computational biology“. In: BioData Mining 10, S. 35. ISSN: 1756-0381. DOI: 10.1186/s13040-017-0155-3. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC5721660/.

Chollet, François u. a. (2015). Keras. https://keras.io.

Cirean, Dan u. a. (Jan. 2012). „Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images“. In: Proceedings of Neural Information Processing Systems 25.

Day, Mike D. (2014). Extracting Euler Angles from a Rotation Matrix. URL: https://www.semanticscholar.org/paper/Extracting-Euler-Angles-from-a-Rotation-Matrix-Day/668137fa4b875d890f446e689eea1e334bcf6bf6.

Duda, Richard O. und Peter E. Hart (1973). Pattern Classification and Scene Analysis. John Wiley & Sons, Inc.

Farazi, Hafez, Philipp Allgeuer und Sven Behnke (2018). „A Monocular Vision System for Playing Soccer in Low Color Information Environments“. In: CoRR abs/1809.11078. arXiv: 1809.11078. URL: http://arxiv.org/abs/1809.11078.

Farazi, Hafez, Philipp Allgeuer, Grzegorz Ficht u. a. (2017). „RoboCup 2016 Humanoid TeenSize Winner NimbRo: Robust Visual Perception and Soccer Behaviors“. In: RoboCup 2016: Robot World Cup XX. Hrsg. von Sven Behnke u. a. Cham: Springer International Publishing, S. 478–490. ISBN: 978-3-319-68792-6.

Fischler, Martin A. und Robert C. Bolles (Juni 1981). „Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography“. In: Commun. ACM 24.6, S. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: http://doi.acm.org/10.1145/358669.358692.

Garcia-Garcia, Alberto u. a. (2017). A Review on Deep Learning Techniques Applied to Semantic Segmentation. arXiv: 1704.06857 [cs.CV].

Glorot, Xavier, Antoine Bordes und Yoshua Bengio (Nov. 2011). „Deep Sparse Rectifier Neural Networks“. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. Hrsg. von Geoffrey Gordon, David Dunson und Miroslav Dudík. Bd. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, S. 315–323. URL: http://proceedings.mlr.press/v15/glorot11a.html.

Goodfellow, Ian u. a. (2014). „Generative adversarial nets“. In: Advances in neural information processing systems, S. 2672–2680.

Gudi, Amogh u. a. (2013). Feature Detection and Localization for the RoboCup Soccer SPL. URL: https://www.dutchnaoteam.nl/publications/gudi2013feature.pdf.

Guo, Chunzhao, Seiichi Mita und David McAllester (Okt. 2010). „Lane detection and tracking in challenging environments based on a weighted graph and integrated cues“. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, S. 5543–5550. DOI: 10.1109/IROS.2010.5650695.

Hornik, Kurt (1991). „Approximation capabilities of multilayer feedforward networks“. In: Neural Networks 4.2, S. 251–257. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90009-T. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T.

Houliston, Trent und Stephan K. Chalup (2019). „Visual Mesh: Real-Time Object Detection Using Constant Sample Density“. In: RoboCup 2018: Robot World Cup XXII. Hrsg. von Dirk Holz u. a. Cham: Springer International Publishing, S. 45–56. ISBN: 978-3-030-27544-0.

Howard, Andrew G. u. a. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv: 1704.04861 [cs.CV].

Iandola, Forrest N. u. a. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv: 1602.07360 [cs.CV].

Intel Corporation (2019). Intel® 64 and IA-32 Architectures Software Developer Manuals. https://software.intel.com/en-us/articles/intel-sdm.

Ioffe, Sergey und Christian Szegedy (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv: 1502.03167 [cs.LG].

Jung, Alexander (2019). imgaug – Image augmentation for machine learning experiments. URL: https://github.com/aleju/imgaug.

Kar, P., A. Jain und B. K. Rout (Juli 2016). „Effective localization of humanoid with fish-eye lens using field line detection". In: 2016 Asia-Pacific Conference on Intelligent Robot Systems (ACIRS), S. 72–78. DOI: 10.1109/ACIRS.2016.7556191.

Kingma, Diederik P. und Jimmy Ba (2014). Adam: A Method for Stochastic Optimization. arXiv: 1412.6980 [cs.LG].

Krishnamoorthi, Raghuraman (2018). Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv: 1806.08342 [cs.LG].

Kruse, Rudolf u. a. (2015). „Allgemeine neuronale Netze". In: Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze. Wiesbaden: Springer Fachmedien Wiesbaden, S. 33–42. ISBN: 978-3-658-10904-2. DOI: 10.1007/978-3-658-10904-2_4. URL: https://doi.org/10.1007/978-3-658-10904-2_4.

LeCun, Y. u. a. (Dez. 1989). „Backpropagation Applied to Handwritten Zip Code Recognition". In: Neural Comput. 1.4, S. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. URL: http://dx.doi.org/10.1162/neco.1989.1.4.541.

Leiva, Francisco u. a. (2019). „Playing Soccer Without Colors in the SPL: A Convolutional Neural Network Approach". In: RoboCup 2018: Robot World Cup XXII. Hrsg. von Dirk Holz u. a. Cham: Springer International Publishing, S. 122–134. ISBN: 978-3-030-27544-0.

Lin, Tsung-Yi u. a. (Okt. 2017). „Focal Loss for Dense Object Detection". In: 2017 IEEE International Conference on Computer Vision (ICCV). DOI: 10.1109/iccv.2017.324. URL: http://dx.doi.org/10.1109/ICCV.2017.324.

Lipski, C. u. a. (März 2008). „A Fast and Robust Approach to Lane Marking Detection and Lane Tracking". In: 2008 IEEE Southwest Symposium on Image Analysis and Interpretation, S. 57–60. DOI: 10.1109/SSIAI.2008.4512284.

Liu, Wei u. a. (2016). „SSD: Single Shot MultiBox Detector". In: Lecture Notes in Computer Science, S. 21–37. ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

Lu, Si, Li Liu und Shuai Zhao (Dez. 2012). „Field Line Detection Based on Local-precise Extracting and Modified Hough Transform". In: 2012 12th International Conference on Control Automation Robotics Vision (ICARCV), S. 383–389. DOI: 10.1109/ICARCV.2012.6485189.

Maas, Andrew L., Awni Y. Hannun und Andrew Y. Ng (2013). „Rectifier nonlinearities improve neural network acoustic models“. In: ICML Workshop on Deep Learning for Audio, Speech and Language Processing.

Mallot, Hanspeter A. u. a. (Jan. 1991). „Inverse perspective mapping simplifies optical flow computation and obstacle detection“. In: Biological Cybernetics 64.3, S. 177–185. ISSN: 1432-0770. DOI: 10.1007/BF00201978. URL: https://doi.org/10.1007/BF00201978.

Mühlenbrock, Andre und Tim Laue (2018). „Vision-based Orientation Detection of Humanoid Soccer Robots“. In: RoboCup 2017: Robot World Cup XXI. Lecture Notes in Artificial Intelligence. Springer.

Ning, Feng u. a. (Sep. 2005). „Toward Automatic Phenotyping of Developing Embryos from Videos“. In: Trans. Img. Proc. 14.9, S. 1360–1371. ISSN: 1057-7149. DOI: 10.1109/TIP.2005.852470. URL: http://dx.doi.org/10.1109/TIP.2005.852470.

Nyquist, H. (Apr. 1928). „Certain Topics in Telegraph Transmission Theory“. In: Transactions of the American Institute of Electrical Engineers 47.2, S. 617–644. ISSN: 0096-3860. DOI: 10.1109/T-AIEE.1928.5055024.

Pătrăucean, Viorica (Jan. 2012). „Detection and identification of elliptical structure arrangements in images: theory and algorithms“. Diss. Institut National Polytechnique de Toulouse. URL: http://oatao.univ-toulouse.fr/6985/.

Perez, Luis und Jason Wang (2017). The Effectiveness of Data Augmentation in Image Classification using Deep Learning. arXiv: 1712.04621 [cs.CV].

Poggenhans, Fabian, Andre-Marcel Hellmund und Christoph Stiller (Nov. 2016). „Application of line clustering algorithms for improving road feature detection“. In: 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), S. 2456–2461. DOI: 10.1109/ITSC.2016.7795951.

Poppinga, Bernd und Tim Laue (to appear). „JET-Net: Real-Time Object Detection for Mobile Robots“. In: RoboCup 2019: Robot World Cup XXIII. Lecture Notes in Artificial Intelligence. Springer.

Qian, Yongbo und Daniel D. Lee (2017). „Adaptive Field Detection and Localization in Robot Soccer“. In: RoboCup 2016: Robot World Cup XX. Hrsg. von Sven Behnke u. a. Cham: Springer International Publishing, S. 218–229. ISBN: 978-3-319-68792-6.

Redmon, Joseph und Ali Farhadi (2018). YOLOv3: An Incremental Improvement. arXiv: 1804.02767 [cs.CV].

Ren, Shaoqing u. a. (Juni 2017). „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks“. In: IEEE Transactions on Pattern Analysis and Machine Intelligence 39.6, S. 1137–1149. ISSN: 2160-9292. DOI: 10.1109/tpami.2016.2577031. URL: http://dx.doi.org/10.1109/TPAMI.2016.2577031.

RoboCup Technical Committee (2019). RoboCup Standard Platform League (NAO) Rule Book (2019 rules, as of May 22, 2019). http://spl.robocup.org/wp-content/uploads/downloads/Rules2019.pdf.

Röfer, Thomas (2008). „Region-Based Segmentation with Ambiguous Color Classes and 2-D Motion Compensation“. In: RoboCup 2007: Robot Soccer World Cup XI. Hrsg. von Ubbo Visser u. a. Bd. 5001. Lecture Notes in Artificial Intelligence. Springer, S. 369–376.

Röfer, Thomas, Tim Laue, Yannick Bülter u. a. (2017). B-Human Team Report and Code Release 2017. Online verfügbar unter: http://www.b-human.de/downloads/publications/2017/coderelease2017.pdf.

Röfer, Thomas, Tim Laue, Arne Hasselbring u. a. (2018). B-Human Team Report and Code Release 2018. Online verfügbar unter: http://www.b-human.de/downloads/publications/2018/CodeRelease2018.pdf.

Ronneberger, Olaf, Philipp Fischer und Thomas Brox (2015). „U-Net: Convolutional Networks for Biomedical Image Segmentation“. In: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, S. 234–241. ISSN: 1611-3349. DOI: 10.1007/978-3-319-24574-4_28. URL: http://dx.doi.org/10.1007/978-3-319-24574-4_28.

Ruder, Sebastian (2016). An overview of gradient descent optimization algorithms. arXiv: 1609.04747 [cs.LG].

Ruiz-del-Solar, Javier, Patricio Loncomilla und Naiomi Soto (2018). „A Survey on Deep Learning Methods for Robot Vision“. In: CoRR abs/1803.10862. arXiv: 1803.10862. URL: http://arxiv.org/abs/1803.10862.

Schnekenburger, Fabian u. a. (2017). „Detection and Localization of Features on a Soccer Field with Feedforward Fully Convolutional Neural Networks (FCNN) for the Adult-Size Humanoid Robot Sweaty“. en. In: Proceedings of the 12th Workshop on Humanoid Soccer Robots, 17th IEEE-RAS International Conference on Humanoid Robots. Birmingham: IEEE, S. 1–6. URL: http://nbn-resolving.de/urn:nbn:de:bsz:ofb1-opus4-26557.

Schröder, René (2017). „Feldrand-Erkennung im Roboterfußball mittels Random Sample Set Consensus“. Bachelorarbeit. Universität Bremen.

Shelhamer, Evan, Jonathan Long und Trevor Darrell (Apr. 2017). „Fully Convolutional Networks for Semantic Segmentation“. In: IEEE Transactions on Pattern Analysis and Machine Intelligence 39.4, S. 640–651. ISSN: 2160-9292. DOI: 10.1109/tpami.2016.2572683. URL: http://dx.doi.org/10.1109/TPAMI.2016.2572683.

Softbank Robotics (o.D.). NAO - Documentation – Aldebaran 2.8.6.6 documentation. http://doc.aldebaran.com/2-8/home_nao.html. Abgerufen am 02.09.2019.

Sokolova, Marina und Guy Lapalme (Juli 2009). „A Systematic Analysis of Performance Measures for Classification Tasks“. In: Inf. Process. Manage. 45.4, S. 427–437. ISSN: 0306-4573. DOI: 10.1016/j.ipm.2009.03.002. URL: http://dx.doi.org/10.1016/j.ipm.2009.03.002.

Szemenyei, Marton und Vladimir Estivill-Castro (to appear). „ROBO: Robust, Fully Neural Object Detection for Robot Soccer“. In: RoboCup 2019: Robot World Cup XXIII. Springer.

Szymański, Piotr und Tomasz Kajdanowicz (2017a). „A Network Perspective on Stratification of Multi-Label Data“. In: Proceedings of the First International Workshop on Learning with Imbalanced Domains: Theory and Applications. Hrsg. von Luís Torgo u. a. Bd. 74.

Proceedings of Machine Learning Research. ECML-PKDD, Skopje, Macedonia: PMLR, S. 22–35.

Szymański, Piotr und Tomasz Kajdanowicz (2017b). „A scikit-based Python environment for performing multi-label classification“. In: CoRR abs/1702.01460. arXiv: 1702.01460. URL: http://arxiv.org/abs/1702.01460.

Taylor, Luke und Geoff Nitschke (2017). Improving Deep Learning using Generic Data Augmentation. arXiv: 1708.06020 [cs.LG].

The Robocup Federation (o.D.[a]). A Brief History of RoboCup. https://www.robocup.org/a_brief_history_of_robocup. Abgerufen am 02.09.2019.

— (o.D.[b]). Objective. https://www.robocup.org/objective. Abgerufen am 02.09.2019.

Thielke, Felix und Arne Hasselbring (to appear). A JIT Compiler for Neural Network Inference.

van Dijk, Sander G. und Marcus M. Scheunemann (2019). „Deep Learning for Semantic Segmentation on Minimal Hardware“. In: RoboCup 2018: Robot World Cup XXII. Hrsg. von Dirk Holz u. a. Cham: Springer International Publishing, S. 349–361. ISBN: 978-3-030-27544-0.

Wada, Ketaro (2016). labelme: Image Polygonal Annotation with Python. https://github.com/wkentaro/labelme.

Xu, Yan u. a. (2016). Improved Relation Classification by Deep Recurrent Neural Networks with Data Augmentation. arXiv: 1601.03651 [cs.CL].

Ye, Jong Chul und Woon Kyoung Sung (2019). Understanding Geometry of Encoder-Decoder CNNs. arXiv: 1901.07647 [cs.LG].