Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Visuelle Positionsbestimmung für mobile Roboter im RoboCup Kontext

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)

eingereicht von: Robert Martin
geboren am: 08.05.1996
geboren in: Königs Wusterhausen

Gutachter/innen: Prof. Dr. Verena V. Hafner
Prof. Dr. Hans-Dieter Burkhard

eingereicht am: ..... verteidigt am: .....

Inhaltsverzeichnis

Symbolverzeichnis **ii**

1 Einleitung **1**
1.1 Motivation . . . 2
1.2 Beiträge der Arbeit und Abgrenzung . . . 3
1.3 Gliederung der Arbeit . . . 5

2 Visuelle Erkennung von Landmarken **6**
2.1 Grundlagen . . . 7
2.1.1 Kameramodellierung . . . 7
2.1.2 Projektion von Kamerapixeln auf die Bodenebene . . . 9
2.1.3 Bestimmung des Kamerahorizonts . . . 10
2.1.4 Integralbild . . . 12
2.2 Bilddatensatz . . . 13
2.3 Felderkennung . . . 14
2.3.1 Parameter der Felderkennung auf den Datensätzen . . . 17
2.4 Linienkantenerkennung . . . 19
2.4.1 Adaptive Scanlinien . . . 20
2.4.2 Linienkantenerkennung auf adaptiven Scanlinien . . . 24
2.4.3 Auswertung der Linienkantenerkennung . . . 28
2.5 Extraktion von Linienkandidaten . . . 30
2.5.1 Evaluation der Linienkandidaten . . . 33
2.6 Linien- und Kreiserkennung . . . 34
2.6.1 Linienmodell . . . 35
2.6.2 Kreismodell . . . 35
2.6.3 Modellfilterung . . . 38
2.6.4 Auswertung der Linien- und Kreiserkennung . . . 40
2.7 Vergleich der Linienkandidatenerkennung dieser Arbeit mit dem NaoTH-Framework . . . 43

3 Lokalisierung im RoboCup **45**
3.1 Grundlagen . . . 46
3.1.1 Monte-Carlo Lokalisierung . . . 46
3.2 Sensormodelle . . . 47

4 Zusammenfassung **51**
4.1 Ausblick . . . 52

Literatur **53**

Symbolverzeichnis

τ	Threshold oder Parameter, der an die jeweilige Umgebung angepasst werden muss
$\boldsymbol{R}, \boldsymbol{t}$	Matritzen und Vektoren werden durch fettgedruckte Großbuchstaben, bzw. Kleinbuchstaben gekennzeichnet
$\|\|.\|\|$	Euklidische Norm
O_I	Ursprung des Bildkoordinatensystems
O_C	Ursprung des Kamerakooridnatensystems
O_R	Ursprung des Roboterkoordinatensystems
O_W	Ursprung des Weltkooridnatensystems
$\boldsymbol{p}_I, \boldsymbol{p}_C, \boldsymbol{p}_R, \boldsymbol{p}_W$	Die Urprünge von Punkten der jeweiligen Koordinatensysteme wird durch den Index gekennzeichnet
$\boldsymbol{C}$	Kameramatrix des Roboters
ω	Zustandsvariable, in der Regel paramatrisiert durch den Zeitpunkt t: ω_t

Liste der Algorithmen

2.1 Berechnung des Grünintegralbildes. 12

2.2 Felddetektor auf Basis des Integralbildes. 16

2.3 Feldgrenzenpunktscan 17

2.4 Berechnung der adaptiven vertikalen Scanlinien. 22

2.5 Berechnung einer Menge von Abtastpunkten mit einer adaptiven Abtastrate $r_v(y)$ 23

2.6 Hochkantenintervallsuche auf einer vertikale Scanlinie s_x, adaptiert von Reinhardt [Rei11] 27

2.7 Suche nach Doppeledgelnachbarn auf adaptiven Scanlinien 31

2.8 Abschätzung des Mittelkreises aus zwei Linienkandidaten 37

3.1 Punktlandmarkenmodell im NaoTH-Framework [MSK+18] 49

1 Einleitung

Die Chancen von mobilen Robotiksystemen für Anwendungen in der Wirtschaft, Industrie und im täglichen Leben sind unumstritten. Dazu gehören das autonome Fahren, die medizinische Versorgung von Patienten und die Unterstützung am Arbeitsplatz, um nur einige Beispiele zu nennen. Seit langer Zeit sind industrielle Roboter bei einer Vielzahl stationärer Aufgaben nicht mehr wegzudenken. Fortschritte in der Hardware- und Softwareentwicklung erlauben den Einsatz von Robotern in immer komplexeren Umgebungen. Anstatt Arbeitsabläufe an die Maschine anzupassen, sollen Roboter in Zukunft in der Lage sein, sich nahtlos in die von Menschen geprägte Umwelt einzubringen.

Eine wichtige Komponente dabei ist, dass sich Roboter in ihrer eingesetzten Umgebung bewegen können. Mobile Roboter stehen vor neuen Herausforderungen, wie das Ausweichen von Hindernissen und die Bestimmung der eigenen Position. Einfache GPS basierte Systeme sind für die Navigation in engen Bereichen oft zu ungenau oder für Indooranwendungen ungeeignet. Zum Beispiel bei dem Einsatz von Robotern in Rettungsmissionen, bei denen unbekannte Bereiche keine Seltenheit sind [SAD+14].

Damit ein Roboter eine genaue Position bestimmen kann, muss er im Stande sein, eine Repräsentation der Umgebung zu erlangen. Einige Informationen, wie zum Beispiel die Positionen statischer Objekte, können als Vorwissen im Roboter gespeichert sein, andere muss er mit seinen Sensoren wahrnehmen.

Die visuelle Wahrnehmung wird seit mehr als 30 Jahren in der Robotik angewendet [MGFCGV+14]. Sie spielt eine zentrale Rolle in zahlreichen mobilen Robotiksystemen, zum Beispiel in der Interaktion von Robotern mit Menschen [FP17], der Navigation und Bewegungsplanung [Pap13], der Ziel- und Pfadverfolgung [Ras08], sowie der Kartierung und Lokalisierung. Visuelle Detektion hat den Vorteil, dass viele Arten von Objekten erkannt werden können. Nachteile sind die limitierte Auflösung von Kameras und die Abhängigkeit von Lichtverhältnissen. Dies führt zu Abweichungen oder Fehlerkennungen. Gleichzeitig ist der Anspruch an die Rechenleistung höher als bei anderen Sensoren.

Um Implementationen zu vergleichen und praktisch zu testen, wird in der Robotik oft auf Wettbewerbe zurückgegriffen. So sei zum Beispiel die DARPA Grand Challenge genannt, in der autonome Fahrzeuge langen und schweren Offroad Szenarios ausgesetzt sind [ORB04]. Eine andere Vergleichsmöglichkeit bietet der RoboCup Wettbewerb, an dem jährlich mehrere hundert Teams aus vielen Ländern teilnehmen. RoboCup ist eine in 1995 von Kitano et al. [KAK+95] vorgestellte Initiative, bei der Roboter unter anderem in Rettungsmissionen, Unterstützung im Haushalt und Anwendungen in der Industrie getestet werden.

In RoboCup Soccer treten Teams von autonomen Robotern in einem Fußballspiel gegeneinander an. Nachdem ein Roboter auf das Spielfeld gestellt wird, muss er seine Position bestimmen und über die Spielzeit hinweg verfolgen, um Taktiken auszuführen und letztendlich ein Tor zu schießen. Eine Voraussetzung dafür ist, dass der Roboter

in Echtzeit lokale Eigenschaften auf dem Spielfeld zuverlässig detektieren und verorten kann. Dabei werden vor allem visuelle Information der Umgebung verarbeitet.

In dieser Arbeit wollen wir uns mit der Implementation von Algorithmen zur Wahrnehmung von Objekten in der RoboCup Soccer Umgebung beschäftigen, um damit eine Lokalisierung des Roboters zu realisieren.

1.1 Motivation

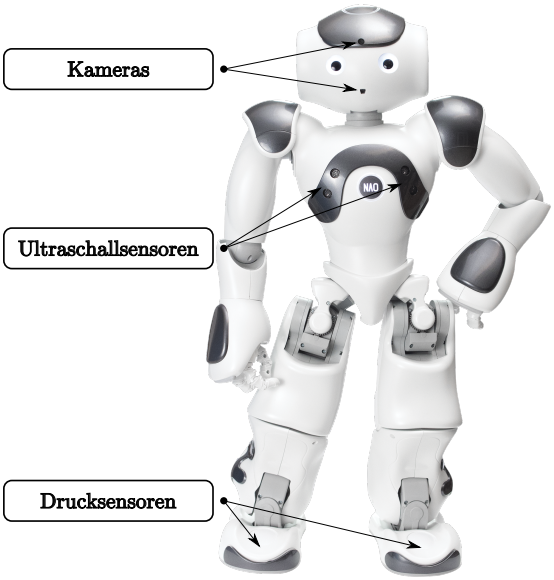


Abbildung 1.1.1: Dargestellt ist der Nao V6 Roboter[1] von Softbank Robotics, wie er in der Standard Plattform Liga verwendet wird. Im Kopf sorgen zwei übereinander angeordnete Kameras für einen großen vertikalen Sichtbereich. Horizontal ist der Öffnungswinkel auf etwa 72.6° begrenzt. Mit den Ultraschallsensoren im Brustkorb können Hindernisse in unmittelbarer Entfernung wahrgenommen werden. Mit Drucksensoren in den Füßen können wir feststellen, ob der Roboter steht oder angehoben wird. Aus dem Trägheitssensor und den Lagesensoren der Gelenke kann die Lage des Roboters im Raum bestimmt werden.

In der Standard Plattform Liga (SPL) von RoboCup Soccer werden baugleiche humanoide Roboter eingesetzt. Der Focus liegt dabei nicht auf der Hardwareentwicklung, sondern auf dem Vergleich von implementierten Algorithmen. Der verwendete Roboter Nao von Softbank Robotics verfügt über zwei Kameras, mit denen er seine Umgebung wahrnehmen kann (Abbildung 1.1.1).

[1]Bild vom Nao: `https://www.softbankrobotics.com/emea/themes/custom/softbank/images/full-nao.png`, letzter Zugriff: 24.08.2019

In der SPL ist die Lokalisierung des Roboters auf dem Spielfeld ein zentrales Problem. Sie umfasst die visuelle Objekterkennung von wiedererkennbaren Eigenschaften auf dem Spielfeld, den sogenannten Landmarken, sowie die anschließende Bestimmung und Verfolgung seiner relativen Position. Die Fehleranfälligkeit von Sensoren stellt einen besonderen Anspruch an die Robustheit von Lokalisierungsmethoden. Sie erfordert eine umfangreiche Filterung.

Das Team Berlin United - NaoTH nimmt regelmäßig an Tunieren in der SPL teil. Im NaoTH-Framework [MSK+18] ist ein Partikelfilter zur Bestimmung der Roboterposition auf dem Spielfeld implementiert. Der Partikelfilter ist ein Algorithmus der probabilistischen Lokalisierung. Dabei wird iterativ eine Menge von Hypothesen durch wiederholte Messungen von Landmarken verfeinert und die wahrscheinlichste Position berechnet. Die Methode besitzt eine gewisse Robustheit gegenüber Fehlerkennungen, weil der Messfehler explizit modelliert wird.

Als Landmarken für die Lokalisierung verwendet das Team NaoTH im Kamerabild detektierte, gerichtete Feldlinienpunkte. Die Richtungen der Punkte entsprechen den Ausrichtungen von Feldlinien auf denen sie gefunden wurden. Bei der Positionsbestimmung werden die Linienpunkte mit einer Karte des Spielfeldes verglichen. Abgesehen von möglichen Fehlerkennungen können Linienpunkte nicht eindeutig auf dem Spielfeld zugeordnet werden. Das kann dazu führen, dass für eine Menge von detektierten Punkten mehrere Positionen wahrscheinlich sind, was die fehlerhafte Lokalisierung in einem lokalen Maxima oder den Verlust einer gefundenen Position bedeuten kann.

1.2 Beiträge der Arbeit und Abgrenzung

Das Ziel der vorliegenden Arbeit ist die Steigerung der Robustheit bei der Selbstlokalisierung des Roboters im NaoTH-Framework [MSK+18] durch die schrittweise Verbesserung der Objekterkennung. Die Beiträge der Arbeit fassen sich wie folgt zusammen:

1. Implementierung einer robusten Erkennung des Spielfeldes auf Basis einer Integralbild-Repräsentation der Kamerabilder.

   Die meisten relevanten Objekte für das Spiel befinden sich innerhalb der Spielfeldgrenzen. Durch die Einschränkung der Objekterkennung auf das Spielfeld können Fehlerkennungen außerhalb der Grenzen vermieden und Rechenleistung gespart werden. Das Spielfeld zeichnet sich als grüne Region im Bild aus. Im NaoTH-Framework [MSK+18] wird bei der Feldgrenzenerkennung auf sogenannten Scanlinien nach Kanten gesucht, an denen grüne Bereiche enden. Da auf den Scanlinien nur eine lokale Suche möglich ist, kann die Methode anfällig gegenüber Fehlern in der Farbklassifikation sein. In dieser Arbeit wollen wir eine Felderkennung auf Basis des Integralbildes [VJ+01] vorstellen, mit der wir die Umgebung von Pixeln genauer betrachten und lokale Fehler in der Grünerkennung ausgleichen können.

2. Verbesserung der Kantendetektion, sodass mehr Linienpunkte gefunden werden können.

   Klassische Bildverarbeitungsmethoden zur Kantenerkennung wie der Canny Edge Detector [Can87] können auf dem Roboter aufgrund mangelnder Rechenleistung nicht eingesetzt werden. Im NaoTH-Framework ist die Kantenerkennung auf Scanlinien realisiert. Dabei wird das Bild gleichmäßig innerhalb nur weniger vertikaler Linien abgetastet. Dadurch ist die Anzahl erkennbarer Linienpunkte limitiert.

   Feldlinien im Bild nehmen durch die perspektivische Projektion unterschiedlich große Bereiche auf der Bildebene ein. Im unteren Bildbereich können Linien schon mit einer kleinen Abtastrate erfasst werden, während sie in höheren Bildbereichen nur aus wenigen Pixeln bestehen. Um einen gleichmäßigere Abtastung des Bildes in Bezug zur Bodenebene zu erreichen, schlagen Härtl et. al. [HVR13] ein adaptives Scanlinienverfahren unter Beachtung der perspektivische Projektion vor. Anders als im NaoTH-Framework [MSK+18] verwenden Härtl et. al. Scanlinien um Farbregionen im Bild zu identifizieren und eine anschließende Objekterkennung zu ermöglichen. Dieser Ansatz erfordert die Klassifizierung von Farben im Bild, welche durch unterschiedliche Lichtverhältnisse beeinflusst wird. In dieser Arbeit stellen wir eine Adaption der Kantenerkennung des NaoTH-Frameworks auf den adaptiven Scanlinien von Härtl et. al. vor, um die Menge an detektierbaren Linienpunkten zu erhöhen.

3. Implementierung einer Kreis- und Linienerkennung auf den detektierten Linienpunkten zur Anwendung als Landmarken der Lokalisierung für den bestehenden Partikelfilter im NaoTH-Framework.

   Durch die Erkennung von Kreis und Linien können falsch erkannte Linienpunkte ausgesiebt und das Rauschen reduziert werden. Außerdem sind Feldlinien einfacher in der Karte zuzuordnen. Der Mittelfeldkreis ist ein einzigartiges Feature auf dem Spielfeld, sodass wir bei erfolgreicher Erkennung neue verlässliche Positionshypothesen generieren können.

   In dieser Arbeit stellen wir einen Feldlinienerkennung auf Basis der Random sample consensus (RANSAC) Methode [FB81] vor, um die Linien und Kreise in den Linienpunkten zu identifizieren. Anschließend betrachten wir die Modellierung des Messfehlers der erkannten Features, um sie als Landmarken im Partikelfilter zu verwenden.

4. Erstellung eines Evaluierungsdatensatzes vergangener RoboCup Veranstaltungen mit annotierten Mittelfeldkreis-, Feldlinien- und Spielfeldbereichfeatures.

   Für die Auswertung der Feldlinien- und Kreiserkennung haben wir jeweils 150 Bilder der oberen und unteren Kamera von Feldern des RoboCup 2018 in Montreal und der German Open 2018 aufgenommen. Darauf haben wir die Kantenerkennung

dieser Arbeit mit der des NaoTH-Frameworks verglichen und die Güte der Linien und Kreiserkennung auf den resultierenden Linienpunkten evaluiert.

1.3 Gliederung der Arbeit

Wir beginnen mit einem Überblick über den Aufbau des Spielfeldes und führen in Abschnitt 2.1 für die Wahrnehmung nötige Grundlagen ein. Anschließend stellen wir in Abschnitt 2.3 die in dieser Arbeit implementierte Spielfelderkennung vor, mit der die Suche nach Objekten in nachfolgenden Schritten eingeschränkt werden kann. In Abschnitt 2.4 betrachten wir einen Scanlinienalgorithmus nach Härtl et. al. [HVR13] und die darauf aufbauende Kantenerkennung. Nachfolgend erörtern wir in Abschnitt 2.5 die Extraktion von Linienpunkten aus den Kanten, welche für die Linien- und Kreiserkennung in Abschnitt 2.6 benötigt werden. In Abschnitt 2.7 evaluieren wir die Erkennung auf Datensätzen aus vergangenen RoboCup Veranstaltungen und vergleichen sie mit den Kantenerkennungsmethoden im NaoTH-Framework [MSK+18].

Für die Implementierung der erkannten Features als Landmarken geben wir in Abschnitt 3.1 eine Einführung in den im NaoTH-Framework implementierten Partikelfilter. Anschließend beschäftigen wir uns in Abschnitt 3.2 mit der Modellierung des Messfehlers von Mittelfeldkreis- und Feldlinienfeatures, um sie als Landmarken im Partikelfilter zu verwenden.

2 Visuelle Erkennung von Landmarken

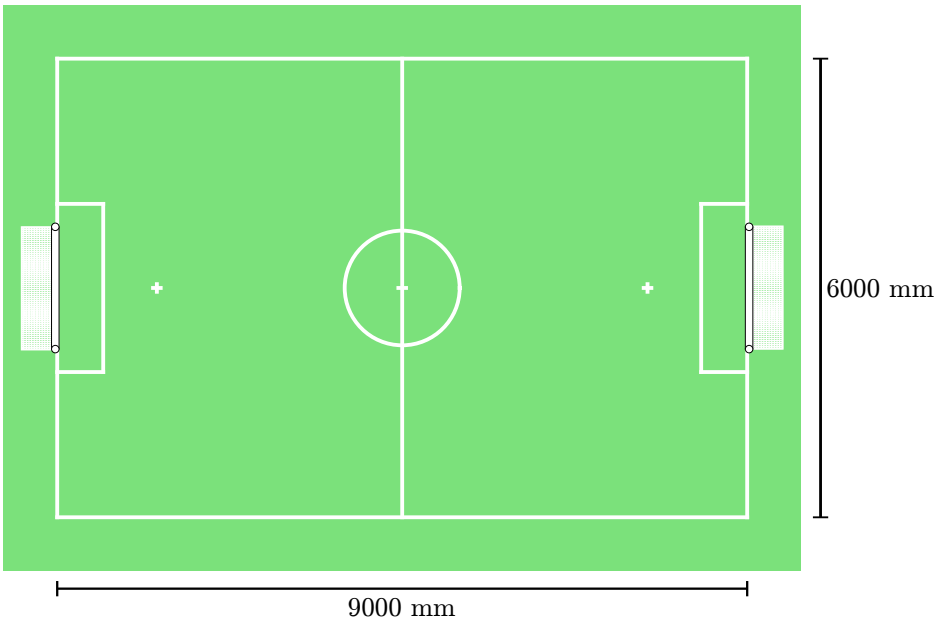


Abbildung 2.0.1: Maßstabsgetreue Darstellung des Spielfeldes, nach den SPL RoboCup Regeln [Com19]. Es besteht aus weißen Linien mit einer Breite von 50 mm, jeweils einem weißen Tor und Strafstoßkreuz pro Spielhälfte und dem Mittelfeldkreis mit einem Durchmesser von 1500 mm. Das Spielfeld ist symmetrisch. Auch wenn der Roboter seine eigene Hälfte kennt, bilden nur Tor, Kreis und Strafstoßkreuz eindeutige Landmarken. Andere Merkmale, wie Linien, müssen gesammelt werden, um eine eindeutige Position für eine Spielfeldhälfte zu bestimmen.

Damit sich der Roboter auf dem Spielfeld lokalisieren kann, muss er Landmarken in den Kamerabildern detektieren und relativ zu sich verorten können. Landmarken sind definiert als eindeutige, stationäre Merkmale der Umgebung, die zuverlässig detektiert werden können [TBF05]. In den Anfängen der RoboCup Liga wurden einzigartige, farbcodierte Beacons an fest definierten Positionen am Spielfeldrand platziert, um als Landmarken zu dienen und die Lokalisierung zu vereinfachen. Seither gab es in der Liga viele Vorstöße, die erleichternden Merkmale des Spielfeldes zu reduzieren, um dem realen Szenario des Fußballs näherzukommen. In diesen Bestrebungen verschwanden viele eindeutige Features, wie die verschiedenfarbigen Tore, was die Lokalisierung im RoboCup um einiges anspruchsvoller macht. Eine eindeutige Zuordnung auf dem Spielfeld ist auch bei der Detektion einzelner Feldmerkmale, wie Tore, Linien oder Linienkreuzungen nur möglich, wenn die eigene Position vorher bekannt ist (Abbildung 2.0.1). Obwohl der Mittelkreis auf dem Spielfeld eindeutig ist, lässt sich auch hier nicht ohne weiteres Vorwissen eine Position bestimmen, weil das Feld symmetrisch ist. In den SPL Regeln [Com19] ist festgelegt, dass der Roboter in seiner eigenen Hälfte startet, sodass Symmetrien aufgelöst werden können. Die Starthälfte muss sich der Roboter

über die gesamte Spielzeit hinweg merken.

Weil Tore in der eigenen Spielhälfte einzigartig sind, können sie, wie der Mittelfeldkreis, ein starker Indikator für eine initiale Position sein und sich für die Generierung von Positionshypothesen eignen. Seitdem die Tore weiß sind, ist eine Detektion deutlich schwieriger geworden. Vor allem, wenn sie vor einem ebenfalls weißen Hintergrund stehen. Die Erkennungsrate von Linien ist höher, weil sie aus jedem Blickwinkel gesehen werden können, wenn der Roboter auf das Feld schaut. Darauf aufbauend kann die Extraktion von Linienkreuzungen den Wiedererkennungswert von Linien erhöhen, deren Erkennung aber nicht Teil dieser Arbeit ist.

Um mit dem Problem der Zuordnung von detektierten Landmarken auf dem Feld umzugehen, werden wir das Weltbild des Roboters bei der Lokalisierung kontinuierlich durch mehrere Messungen verfeinern. In den folgenden Abschnitten beschäftigen wir uns mit der visuellen Erkennungspipeline von Feldgrenzen-, Kreis- und Linienfeatures, die eine Grundlage für die Lokalisierung bilden.

2.1 Grundlagen

Bevor wir auf die jeweiligen Erkennungsalgorithmen näher eingehen, betrachten wir einige Grundlagen, welche deren Implementierung voraussetzten.

2.1.1 Kameramodellierung

Die Kameras im NaoTH-Framework [MSK+18] werden auf Basis einer Lochkamera modelliert. Alle Objekte im Raum lassen sich so über einen Brennpunkt auf die jeweilige Bildebene abbilden. Mit der Umkehrung dieser Abbildung ist es möglich, relative Distanzen und Winkel zu den im Bild detektierten Objekten zu bestimmen.

Für die folgende mathematische Modellierung betrachten wir die Welt in 4 verschiedenen Koordinatensystemen. Dabei ist zu beachten, dass die Bezeichnungen der Koordinatenachsen im NaoTH Framework und in dieser Arbeit von der klassischen Bildverarbeitungsliteratur abweichen.

Das dreidimensionale Kamerakoordinatensystem C beschreibt Objekte in der Umgebung ausgehend vom Brennpunkt der Kamera als Ursprung O_C. In dieser Arbeit beschreiben wir C als Linkssystem. Die Tiefe von Objekten wird über die x-Achse bestimmt und die Höhe mit der z-Achse. Die y-Achse erhalten wir durch das Drehen der x-Achse um 90° gegen den Uhrzeigersinn um die z-Achse. Kamerakoordinaten beschreiben wir mit den Parametern x_C, y_C und z_C.

Im dreidimensionalen Roboterkoordinatensystem R sind alle Objekte relativ zur Roboterposition auf der Bodenebene als Ursprung O_R beschrieben. Auch hier verwenden wir das Linkssystem ähnlich wie bei den Kamerakoordinaten. Die Bodenebene wird

durch die x- und y-Achse aufgespannt. Roboterkoordinaten beschreiben wir durch die Parameter x_R, y_R und z_R.

Das dreidimensionale Weltkoordinatensystem W stellt eine Beziehung zwischen Objekten relativ zur Roboterposition und deren Positionen in der realen Welt her. Den Ursprung O_W der Weltkoordinaten legen wir auf den Mittelpunkt des Mittelfeldkreises auf dem Spielfeld fest. Für die Lokalisierung sind alle Landmarken auf einer Karte in Weltkoordinaten eingezeichnet. Weltkoordinaten werden durch die Parameter x_W, y_W und z_W bestimmt.

Das zweidimensionale Bildkoordinatensystem I beschreibt Objekte in Kamerakoordinaten projiziert auf die Bildebene. Bei der Einteilung des Bildes in Pixeln nehmen wir eine Diskretisierung dieser Ebene vor. Den Ursprung O_I des Bildkoordinatensystems legen wir auf die linke obere Ecke des Bildes fest. Über die y-Achse werden die Reihen des Bildes adressiert und die Spalten über die x-Achse. Bildkoordinaten bestimmen wir durch die Parameter x_I und y_I.

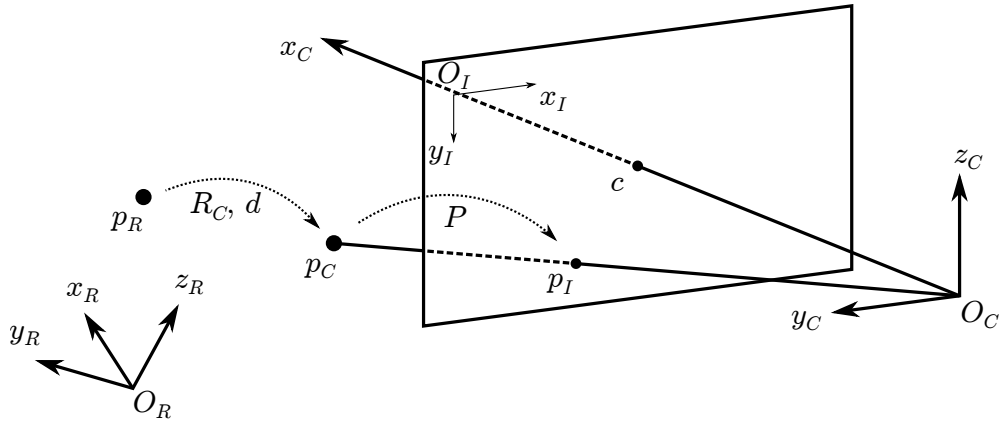


Abbildung 2.1.1: Ein beliebiger Punkt $\boldsymbol{p}_R$ im Roboterkoordinatensystem mit dem Ursprung O_R wird mithilfe einer euklidischen Transformation mit Rotationsmatrix $\boldsymbol{R}_C$ und Translation $\boldsymbol{d}$ auf den Punkt $\boldsymbol{p}_C$ im Kamerakoordinatensystem mit dem Ursprung O_C abgebildet. Schließlich wird eine Projektion $\boldsymbol{P}$ angewendet um den Punkt $\boldsymbol{p}_C$ in den Punkt $\boldsymbol{p}_I$ in das Bildkoordinatensystem mit dem Ursprung O_I zu überführen.

Sei $\boldsymbol{q}, \theta$ die Position und Ausrichtung auf der Bodenebene des Roboters in Weltkoordinaten und $\boldsymbol{d}, \boldsymbol{R}_C$ die Position und Rotationsmatrix der Kamera in Roboterkoordinaten.

Die Abbildung eines beliebigen Punktes $\boldsymbol{p}_W$ in Weltkoordinaten auf einen Punkt $\boldsymbol{p}_I$ auf der Bildebene erfolgt über eine Transformation in den Punkt $\boldsymbol{p}_R$ in Roboterkoordinaten, sodass

$$\boldsymbol{p}_R = \boldsymbol{R}_\theta \cdot \boldsymbol{p}_W + \boldsymbol{q}, \tag{2.1.1}$$

wobei $\boldsymbol{R}_\theta$ die Rotationsmatrix von der Ausrichtung θ des Roboters ist, einer anschließenden Transformation von $\boldsymbol{p}_R$ in Kamerakoordinaten

$$\boldsymbol{p}_C = \boldsymbol{R}_C \cdot \boldsymbol{p}_R + \boldsymbol{d}, \tag{2.1.2}$$

und schließlich einer perspektivischen Projektion

$$\boldsymbol{p}_I = \boldsymbol{P} \cdot \boldsymbol{p}_C \quad \text{mit} \quad \boldsymbol{P} = \begin{bmatrix} u_0 & f & 0 \\ v_0 & 0 & f \\ 1 & 0 & 0 \end{bmatrix}, \quad \text{sodass} \quad \boldsymbol{p}_I = \begin{bmatrix} x_C \cdot x_I \\ x_C \cdot y_I \\ x_C \end{bmatrix} \tag{2.1.3}$$

[Sch05] (Abbildung 2.1.1). Die Werte x_I und y_I ergeben die Koordinaten auf der Bildebene. $\boldsymbol{P}$ ist die perspektivische Projektionsmatrix, welche die extrinsischen Parameter der Kamera enthält. f bezeichnet die Brennweite, die den Abstand zwischen Bildebene und Brennpunkt angibt. u_0 und v_0 beschreiben die Translation des Bildursprungs I. Diese Verschiebung ist notwendig, da der Bildkoordinatenursprung im NaoTH-Framework [MSK+18] in der linken oberen Ecke des Bildes festgelegt ist.

Die inverse Transformation von Bildkoordinaten in Weltkoordinaten erfolgt durch

$$\boldsymbol{p}_C = \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix} = \boldsymbol{P}^{-1} \cdot \boldsymbol{p}_I = \begin{bmatrix} -\frac{u_0}{f} & \frac{1}{f} & 0 \\ -\frac{v_0}{f} & 0 & \frac{1}{f} \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_C \cdot x_I \\ x_C \cdot y_I \\ x_C \end{bmatrix} \tag{2.1.4}$$

$$\boldsymbol{p}_R = \boldsymbol{R}_C^{-1} \, (\boldsymbol{p}_C - \boldsymbol{d}) \tag{2.1.5}$$

$$\boldsymbol{p}_W = \boldsymbol{R}_\theta^{-1} \, (\boldsymbol{p}_R - \boldsymbol{q}). \tag{2.1.6}$$

Für Rotationsmatrizen gilt $\boldsymbol{R}^{-1} = \boldsymbol{R}^T$.

Die Tiefe x_C eines Punktes in Bildkoordinaten ist eine unbekannte Größe. Der Roboter verfügt über keine Stereo-Bild Systeme oder dergleichen aus denen Tiefeninformationen extrahiert werden können. Da wir aber die Position der Roboterkameras im Raum aus den Orientierungen seiner Gelenke berechnen können, ist eine Projektion von Pixeln auf den Boden möglich.

2.1.2 Projektion von Kamerapixeln auf die Bodenebene

Sei $\boldsymbol{p}_I$ ein Punkt auf der Bildebene und $\boldsymbol{p}_C$ der gleiche Punkt in Kamerakoordinaten, sodass

$$\boldsymbol{p}_C = \begin{bmatrix} f \\ x_I - u_0 \\ y_I - v_0 \end{bmatrix}. \tag{2.1.7}$$

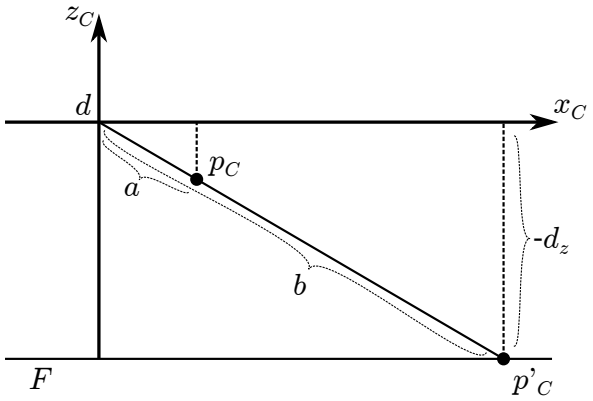


Abbildung 2.1.2: Projektion eines Bildpunktes $\boldsymbol{p}_C$ auf einen Punkt $\boldsymbol{p}'_C$ auf der Bodenebene F, die in unserem Fall parallel zur xy-Ebene in Kamerakoordinaten orientiert ist. Die Translation $\boldsymbol{d}$ von Roboter- in Kamerakoordinaten entspricht der Kameraposition im Roboterkoordinatensystem. Aus der Abbildung geht hervor, dass wir den Vektor $\boldsymbol{p}_C$ um den Faktor b/a verlängern müssen. Nach dem Strahlensatz gilt $b/a = -d_z/p_{C,z}$.

Um die Position des Punktes $\boldsymbol{p}'_C$ auf der Bodenebene zu berechnen, der durch $\boldsymbol{p}_I$ auf der Bildebene abgebildet wird, müssen wir $\boldsymbol{p}_C$ verlängern, damit

$$\boldsymbol{p}'_C = v \cdot \boldsymbol{p}_C, \tag{2.1.8}$$

Sei $\boldsymbol{d}$ die Translation von Roboter- in Kamerakoordinaten. Unter Verwendung des Strahlensatzes (Abbildung 2.1.2) lässt sich v bestimmen als

$$v = -\frac{d_z}{y_I - v_0}. \tag{2.1.9}$$

2.1.3 Bestimmung des Kamerahorizonts

Pixel im Bild können nur unterhalb des Horizonts auf die Bodenebene projiziert werden. Der Kamerahorizont sei definiert als die Überschneidung zweier Ebenen I und H, wobei I die Bildebene der Kamera und H eine Ebene parallel zum Boden bezeichnet, die durch den Ursprung C des Kamerakoordinatensystems verläuft [JHL03]. Sei w die Breite des Kamerabildes. Gesucht sind die Werte $z_{l/r}$ der Punkte

$$\boldsymbol{b}_{l/r} = \begin{bmatrix} f \\ \pm\frac{w}{2} \\ z_{l/r} \end{bmatrix}, \tag{2.1.10}$$

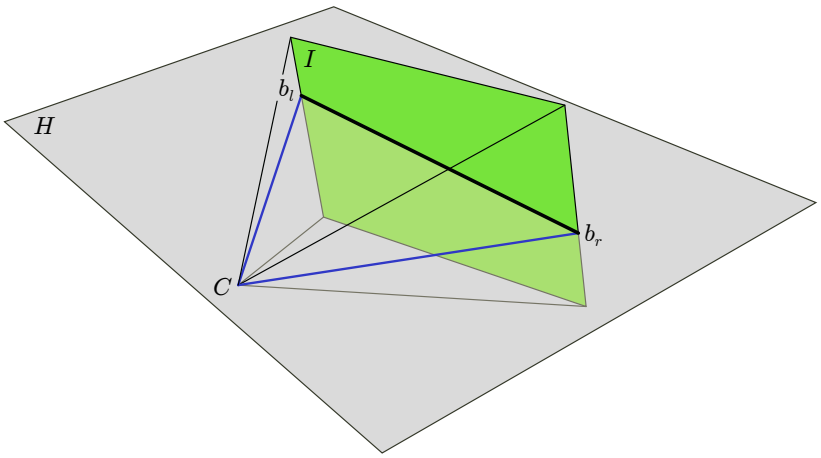


Abbildung 2.1.3: Darstellung der Horizontebene H. Die Punkte $\boldsymbol{b}_l$ und $\boldsymbol{b}_r$ definieren die Schnittlinie des Horizonts mit der Bildebene I (Bild adaptiert von Jüngel et al. [JHL03]).

auf der linken und rechten Bildgrenze in Kamerakoordinaten, die auf der Horizontebene H liegen, sodass gilt

$$\boldsymbol{R}_C \cdot \boldsymbol{b}_{l,r} = \boldsymbol{h} \quad \text{mit} \quad \boldsymbol{h} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}, \tag{2.1.11}$$

wobei $\boldsymbol{R}_C$ die Rotationsmatrix der Kamera in Roboterkoordinaten und $\boldsymbol{h}$ einen Punkt auf H definiert (Abbildung 2.1.3). Ausgeschrieben lautet die Gleichung 2.1.11

$$\begin{aligned} R_{11} \cdot f + R_{12} \cdot (\pm\frac{w}{2}) + R_{13} \cdot z_{l/r} &= x \\ R_{21} \cdot f + R_{22} \cdot (\pm\frac{w}{2}) + R_{23} \cdot z_{l/r} &= y \\ R_{31} \cdot f + R_{32} \cdot (\pm\frac{w}{2}) + R_{33} \cdot z_{l/r} &= 0 \end{aligned} \tag{2.1.12}$$

Umstellen der dritten Zeile von Gleichung 2.1.12 nach $z_{l/r}$ ergibt schließlich

$$z_{l/r} = \frac{\pm R_{32} \cdot \frac{w}{2} - R_{31} \cdot f}{R_{33}}. \tag{2.1.13}$$

Seien $\boldsymbol{b}'_l$ und $\boldsymbol{b}'_r$ die Punkte $\boldsymbol{b}_{l,r}$ projiziert auf die Bildebene. Die Horizontlinie $l_H(t)$ auf der Bildebene ist definiert durch

$$l_H(t) = t \cdot \boldsymbol{d}_H + \boldsymbol{s} \quad \text{mit} \quad \boldsymbol{d}_H = \frac{\boldsymbol{b}'_r - \boldsymbol{b}'_l}{||\boldsymbol{b}'_r - \boldsymbol{b}'_l||} \quad \text{und} \quad \boldsymbol{s} = \boldsymbol{b}'_l. \tag{2.1.14}$$

Für diese Arbeit definieren wir die minimale y-Position im Bild an Position x begrenzt durch die Horizontlinie l_h durch die folgende Funktion, sodass

$$l_y(x) = \frac{x - s_x}{d_x} \cdot d_y + s_y \tag{2.1.15}$$

$$G(x) = \begin{cases} 0 & \text{wenn } l_y(x) < 0, \\ h & \text{wenn } l_y(x) > h, \\ l_y(x) & \text{sonst,} \end{cases} \tag{2.1.16}$$

wobei h die Bildhöhe ist.

2.1.4 Integralbild

Das Integralbild wurde ursprünglich von Viola und Jones [VJ+01] als Teil eines Frameworks für die schnelle Detektion von Gesichtern in Kamerabildern entwickelt. Es ermöglicht die Berechnung von Pixelsummen innerhalb rechteckiger Bereiche des Bildes in konstanter Zeit. Das Integralbild

$$ii(x, y) = \sum_{x'<x, y'<y} i(x', y'). \tag{2.1.17}$$

enthält die Summe von vorangegangen Pixeln $x' < x, y' < y$ im Ursprungsbild an Position x, y. Die Erstellung des Integralbildes ist in Algorithmus 2.1 am Beispiel für grün klassifizierte Pixel gegeben. Damit können wir die Pixelsumme eines Rechteckes R

Algorithmus 2.1 : Berechnung des Grünintegralbildes.

Eingabe : Originalbild $i(x, y)$, Bildbreite w, Bildhöhe h,

$$f_{\text{grün}}(p_{x,y}) = \begin{cases} 1, & \text{falls Pixel } p \text{ am Punkt } x, y \text{ grün,} \\ 0, & \text{sonst.} \end{cases}$$

Ausgabe : Grünintegralbild $ii(x, y)$

```
1 für alle y ∈ [0 : h − 1] tue
2     σ = 0;
3     für alle x ∈ [0 : w − 1] tue
4         σ = σ + f_grün(i(x, y));
5         wenn y > 0 dann
6             ii(x, y) = σ + ii(x, y − 1);
7         sonst
8             ii(x, y) = σ;
```

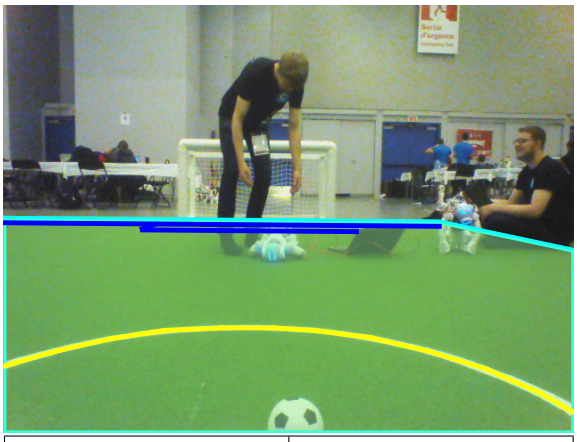
mit den Koordinaten $(x_{min}, y_{min}, x_{max}, y_{max})$ mit nur vier Referenzen im Bild berechnen,

sodass

$$\sum R = ii(x_{max}, y_{max}) + ii(x_{min}, y_{min}) - (ii(x_{min}, y_{max}) + ii(x_{max}, y_{min})). \quad (2.1.18)$$

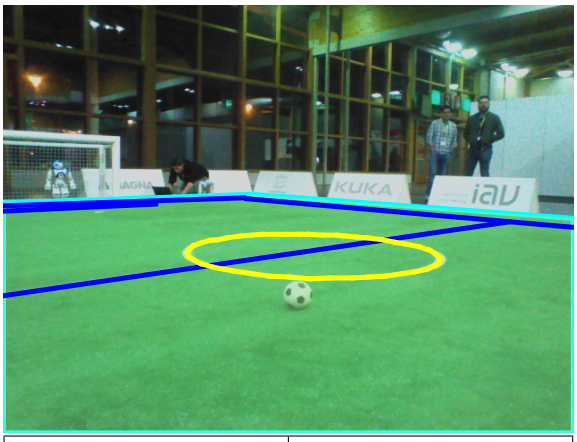
Das Integralbild wird im NaoTH Framework aktuell zur Ballerkennung verwendet [MSK+18]. Es kann aber für weitere Zwecke wiederverwertet werden, wie zum Beispiel bei der Felderkennung, die wir in Abschnitt 2.3 vorstellen.

2.2 Bilddatensatz



Annotation	Anzahl
Linien	421
Kreislinienzüge	194

(a) RoboCup 2018



Annotation	Anzahl
Linien	443
Kreislinienzüge	173

(b) German Open 2018

Abbildung 2.2.1: Beispiele aus den Evaluierungsdatensätzen. Feldlinien wurden durch ihren im Bild sichtbaren Anfangs- und Endpunkt markiert, sodass die entstandenen Strecken (blau) möglichst mittig auf der Linie liegen. Die Strecken wurden durchgezogen, auch wenn die Feldlinien teilweise verdeckt waren. Der Mittelfeldkreis wurde durch einen Linienzug (rot) markiert und der Spielfeldbereich durch ein Polygon (türkis).

Für die Evaluation der Feld-, Kreis- und Liniendetektion wurden zwei Bilddatensätze aus Aufnahmen von Spielfeldern der RoboCup Weltmeisterschaft 2018 in Kanada und der German Open 2018 erstellt. Dabei handelt es sich um jeweils 150 Bilder der oberen und unteren Kamera pro Datensatz. Um Ground Truth Daten zu erzeugen, haben wir zu jedem Bild die Kreis-, Feldlinien und Feldbereiche annotiert (Abbildung 2.2.1). Eine Aufnahme von Bildern während eines gesamten RoboCup Spieles ist im NaoTH-Framework derzeit aufgrund mangelnder Prozessorleistung und zu kleinem Hauptspeicher des Roboters nicht möglich. Die Bilder stammen aus Testaufnahmen, die abseits der regulären Spiele während der Meisterschaften erstellt wurden. Um eine repräsentative Menge von Beispielen zu erzeugen, haben wir Bilder zufällig ausgewählt.

Da Felder zwischen den regulären Spielen auch von anderen Teams genutzt werden, zeichnen sich einige Bilder durch Objekte aus, die normalerweise nicht auf dem Spielfeld auftreten, zum Beispiel Laptops und LAN-Kabel (Abbildung 2.2.1a). Zu jedem Bild wurde die Position und Rotation der Kamera aufgenommen, sodass wir die Positionen von Pixeln auf der Bodenebene berechnen können.

2.3 Felderkennung

(a) (b) (c) (d)

Abbildung 2.3.1: Dargestellt sind Kamerabilder des Roboters vom Spielfeld. Felder im Bild liegen an der unteren Bildgrenze an (a). Fehler in der Feldfarbendetektion sind wahrscheinlich, unter anderem durch Reflexionen in Tischbeinen oder grünen Objekten außerhalb des Feldes (c). Grüne Regionen können durch weiße Linien unterbrochen werden (d) oder auch als benachbartes Spielfeld außerhalb der eigenen Feldgrenzen auftreten (b).

In den offiziellen Regeln der Standard Plattform Liga [Com19] ist der Aufbau des Spielfeldes detailliert festgelegt. Es werden aber nur wenige Aussagen darüber getroffen, wie die Umgebung außerhalb der Feldgrenzen auszusehen hat. Wir müssen also damit rechnen, dass die Umgebung rund um das Spielfeld jegliche Formen annehmen kann. Seien es Bälle, die am Spielfeldrand vergessen wurden, gestreifte T-Shirts von Zuschauern, die Feldlinien ähneln oder benachbarte Spielfelder, die nicht durch Banden verdeckt werden, wenn sie mindestens 3 Meter voneinander entfernt sind. Wollen wir Fehlerkennungen vermeiden und Rechenaufwand sparen, lohnt es sich, die Suche von Objekten im Bild auf eine sogenannte Region of Interest (ROI) einzugrenzen.

Da sich jegliche für das Spiel relevante Objekte und Landmarken innerhalb der Feldgrenzen flach auf der Ebene befinden, oder zumindest auf dem Feld stehen, bildet das eigene Spielfeld im Bild eine natürliche ROI. Der Roboter steht während der gesamten Spielzeit auf dem Feld und ist somit von der grünen Feldfarbe umgeben. Daraus folgt, dass zumindest im unteren Bildbereich eine grüne Region zu erkennen ist, vorausgesetzt der Roboter blickt auf das Spielfeld (Abbildung 2.3.1). Wir charakterisieren das Feld im Bild als konvexe Region, die hauptsächlich grüne Pixel enthält und an der unteren Bildgrenze anliegt. Die Feldgrenze sei definiert als Übergang dieser Region in Bereiche, die überwiegend andersfarbig sind.

Im Folgenden betrachten wir das Problem der Feldgrenzenerkennung im Bild auf Basis einer bestehenden Farbklassifikation. Im NaoTH Framework [MSK+18] ist eine

Funktion $f_{\text{grün}}$ gegeben, sodass

$$f_{\text{grün}}(p_{x,y}) = \begin{cases} 1, & \text{falls Pixel } p \text{ am Punkt } x, y \text{ grün,} \\ 0, & \text{sonst.} \end{cases} \tag{2.3.1}$$

Wir beschreiben den Fehler der Grünerkennung durch eine Zufallsvariable $\epsilon_{\text{grün}}$.

Beim aktuell im NaoTH Framework [MSK+18] implementierten Verfahren wird das Problem der Feldgrenzendetektion als Teil der regulären Kantenerkennung betrachtet. Dabei ist das Bild vertikal gleichmäßig in sogenannte Scanlinien aufgeteilt. Auf diesen Linien wird das Bild lokal von unten nach oben abgetastet und nach Kanten gesucht, an denen grüne Bereiche enden. Ähnliche Ansätze werden zum Beispiel auch von den Teams Nao Team HTWK [HTW18], B-Human [RLH+18] und HULKs [AFG+18] implementiert. Ein Scanlinienverfahren zur Kantenerkennung werden wir noch ausführlich in Abschnitt 2.4 betrachten.

Weil auf den Scanlinien nur lokal aufeinanderfolgende Pixel abgetastet werden, ist die Methode anfällig für Fehler in der Gründetektion. Es ist zu erwarten, dass einige Feldgrenzenpunkte fehlerhaft detektiert werden. Um diese Fehler zu reduzieren wenden die oben genannten Teams noch ein umfangreiches Ausreißerfilterungsverfahren an.

In dieser Arbeit implementieren wir eine Methode, die robuster gegenüber Fehlern in der Grünerkennung ist, unabhängig von der regulären Kantenerkennung funktioniert und damit eine ROI für alle folgenden Objekterkennungsschritte bildet. Dafür ist es nötig, das Umfeld von einzelnen Pixeln näher zu betrachten.

Anstatt der Verwendung von Scanlinien, teilen wir das Bild gleichmäßig in Zellen $Z_{i,j}$ auf. Ziel ist es, die Zelle innerhalb jeder Spalte i zu finden, welche die Feldgrenze enthält. Wir nehmen an, dass Feldgrenzenzellen einen geringen Grünanteil haben und von Zellen mit hohem Grünanteil prädestiniert sind. Der Grünanteil $\rho_{\text{grün}}$ einer Zelle Z kann wie folgt berechnet werden:

$$\rho_{\text{grün}} = \frac{\sum_{\text{grün}} Z}{(x_{max} - x_{min}) \cdot (y_{max} - y_{min})}. \tag{2.3.2}$$

Die Variablen im Nenner bezeichnen die Ausmaße von Z auf der x bzw. y-Achse. Die Summe grüner Pixel $\sum_{\text{grün}} Z$ können wir effizient mit Hilfe des Grünintegralbildes berechnen (siehe Abschnitt 2.1.4).

Eine Zelle wird als grün markiert, wenn

$$\rho_{\text{grün}} \geq 1 - \bar{\epsilon}_{\text{grün}}, \tag{2.3.3}$$

wobei $\bar{\epsilon}_{\text{grün}}$ die mittlere Fehlerwahrscheinlichkeit der Gründetektion beschreibt. Wir legen $1 - \bar{\epsilon}_{\text{grün}}$ als Threshold $\tau_{\text{grün}}$ fest. Die Suche nach der Feldgrenze ist in Algorithmus 2.2 angegeben.

Algorithmus 2.2 : Felddetektor auf Basis des Integralbildes.

Eingabe : Grünintegralbild $ii(x, y)$, Integralbildbreite w, Integralbildhöhe h, Zellengröße τ_s, $\tau_{grün}$, τ_{Grenze}, τ_{min}, τ_{max}

Ausgabe : Feldgrenzenkandidaten p_i

1 Spaltenanzahl $n = w/s$;
`# Berechne` $(x_{i,min}, x_{i,max})$ `für alle Spalten` S_n
2 $S_0 = (0, \tau_s)$, $S_i = (max(x_{i-1}), max(x_{i-1}) + \tau_s)$;
3 **für alle** *Spalten* S_i *mit* $i = [0, \ldots, n-1]$ **tue**
`# Berechne Anzahl von Zellen einer Spalte unter dem Horizont`
4 $m = \frac{h - max(G(x_{min}), G(x_{max}))}{\tau_s}$;
`# Berechne` $(y_{j,min}, y_{j,max})$ `für alle Zellen` Z_m `in der Spalte` S_i
5 $Z_{0,y} = (h - \tau_s, h)$, $Z_{j,y} = (y_{j-1,min} - \tau_s, y_{j-1,min})$;
`# Initialisiere Zähler für aufeinanderfolgendes Grün.`
6 $\sigma_{\text{grün}} \leftarrow 0$, $\sigma_{\neg\text{grün}} \leftarrow 0$
7 **für alle** *Zellen* Z_j *mit* $j = [0, \ldots, m-1]$ **tue**
`# Berechne Grünanteil von` Z_j `aus` ii`, vgl. Gleichung 2.3.2`
8 $\rho_{\text{grün}} = \frac{ii(x_{i,max}, y_{j,max}) + ii(x_{i,min}, y_{j,min}) - ii(x_{i,min}, y_{j,max}) - ii(x_{i,max}, y_{j,min})}{\tau_s^2}$;
9 **wenn** $\rho_{grün} \geq \tau_{grün}$ **dann**
10 inkrementiere $\sigma_{\text{grün}}$;
11 **wenn** $\sigma_{grün} \geq \tau_{min}$ **dann**
12 $Z_{\text{Grenze}} \leftarrow Z_j$;
13 $s_{\neg\text{grün}} \leftarrow 0$;
14 **sonst**
15 inkrementiere $s_{\neg\text{grün}}$;
16 **wenn** $\sigma_{\neg grün} > \tau_{max}$ **dann**
17 $\sigma_{\text{grün}} \leftarrow 0$;
18 **wenn** $Z_{Grenze} = Z_{j-1}$ *und* $\rho_{grün} \geq \tau_{Grenze}$ **dann**
19 $Z_{\text{Grenze}} \leftarrow Z_i$;
20 $p_i \leftarrow$ finde Feldgrenzenpunkt in Z_{Grenze} ;

Der Parameter τ_{Grenze} gibt den minimalen Grünanteil einer Feldgrenzenzelle an. τ_{min} beschreibt die Mindestanzahl von Zellen die vor einer Feldgrenzenzelle als grün markiert sein müssen. τ_{max} ist die maximale Anzahl von andersfarbigen Zellen, die als eine Unterbrechung von grünen Bereichen durch zum Beispiel Linien zugelassen werden.

In Zeile 4 berechnen wir die Anzahl der Zellen unter dem Horizont m mit der Funktion $G(x)$, die wir in Gleichung 2.1.16 in Abschnitt 2.1.3 definiert haben.

Der Zähler $\sigma_{\text{grün}}$ in Zeile 6 sorgt dafür, dass mehrere grüne Zellen hintereinander detektiert werden müssen, bevor wir einen Kandidat für die Feldgrenzenzelle bestimmen. Dies soll verhindern, dass einzelne grüne Bereiche außerhalb des Feldes, wie zum Beispiel

in Abbildung 2.3.1b, beachtet werden.

Bevor wir den Zähler in Zeile 17 zurücksetzen, dürfen einzelne andersfarbige Zellen übersprungen werden, die zum Beispiel weiße Linien enthalten können. Aufeinanderfolgende Zellen mit zu geringem Grünanteil werden über den Zähler $\sigma_{\neg\text{grün}}$ akkumuliert.

Wenn eine Zelle die Voraussetzung erfüllt einen Teil der Feldgrenze zu enthalten, aber nicht als grün markiert wurde, wird sie erneut mit einem weniger strengen Threshold überprüft (Zeile 18). Wir nehmen an, dass Feldgrenzenzellen überwiegend Pixel enthalten, die nicht Teil des Feldes sind.

Mithilfe der Feldgrenzenzellen und deren Gründichte ist nur eine Abschätzung der Feldgrenze möglich. Wollen wir genauere Ergebnisse erzielen, können wir ähnlich wie bei dem Scanlinienansatz innerhalb der Zelle nachscannen. In Algorithmus 2.3 ist ein Verfahren zur Berechnung der größten vorangehenden Gründifferenz ρ_{max} gegeben, welches wir in Zeile 20 von Algorithmus 2.2 verwenden um den Punkt auf der Feldgrenze zu bestimmen.

Algorithmus 2.3 : Feldgrenzenpunktscan

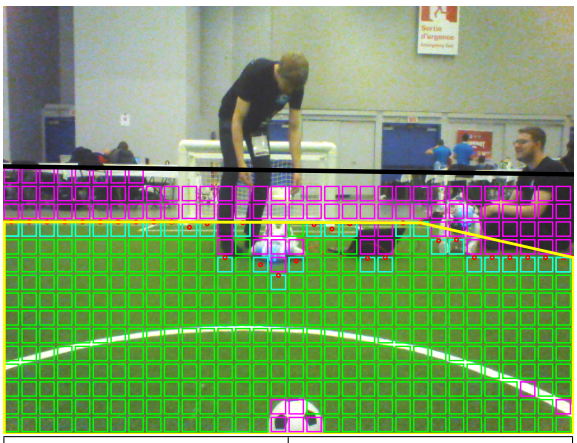
Eingabe : Feldgrenzenzelle $Z_{\text{Grenze}} = (x_{min}, x_{max}, y_{min}, y_{max})$
Ausgabe : Feldgrenzenpunkt p_{Grenze}

```
1  x = (x_min + x_max)/2;
2  ρ_max = f_grün(p_{x,y_max});
3  y_Grenze = y_max;
4  y = y_max − 1;
5  solange y ≥ y_min tue
6  |  ρ = ρ + f_grün(p_{x,y}) − (1 − f_grün(p_{x,y}));
7  |  wenn ρ ≥ ρ_max dann
8  |  |  ρ_max = ρ;
9  |  |  y_Grenze = y;
10 |  dekrementiere y;
11 p_Grenze ← {x, y_Grenze};
```

Um das Feldpolygon aus den Feldgrenzendpunkten zu berechnen, projizieren wir die linken und rechten Feldgrenzenpunkte auf die untere Bildgrenze und berechnen die Konvexe Hülle mit zum Beispiel dem Monotone Chain Convex Hull Algorithmus [And79]. Dadurch werden zu niedrige Feldgrenzenkandidaten ausgeglichen, die zum Beispiel in Robotern gefunden werden, wenn diese die Feldgrenze verdecken.

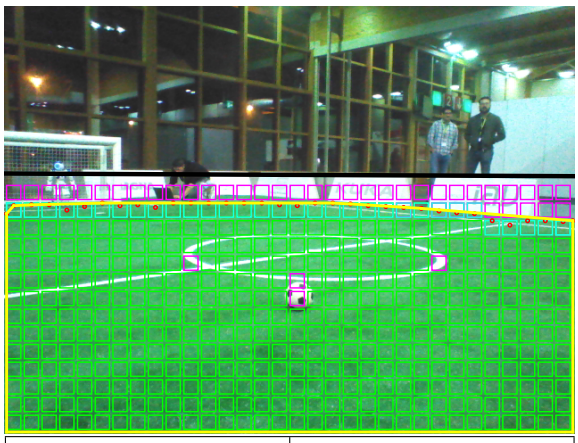
2.3.1 Parameter der Felderkennung auf den Datensätzen

Die Fehlerwahrscheinlichkeit bei der Feldfarbenklassifikation ist abhängig von den Lichtbedingungen, von der Farbintensität des Spielfeldes und dem Aussehen von Objekten außerhalb der Spielfeldgrenzen. In diesem Abschnitt wollen wir uns mit der Anpassung



τ_{max}	0
τ_{min}	4
$\tau_{\text{grün}}$	0.5
τ_{Grenze}	0.5
τ_s obere Kamera	20
τ_s untere Kamera	30

(a) RoboCup 2018



τ_{max}	1
τ_{min}	3
$\tau_{\text{grün}}$	0.5
τ_{Grenze}	0.3
τ_s obere Kamera	20
τ_s untere Kamera	30

(b) German Open 2018

Abbildung 2.3.2: Felderkennung auf den Evaluierungsdatensätzen. In den zwei Beispielbildern sind die Horizontlinie (schwarz), die Feldgrenzenpunkte (rot), das resultierende Feldpolygon (gelb) und die Zellen für die Berechnung des Grünanteils eingezeichnet. Eine Zelle wurde grün markiert, wenn für den zugehörigen Grünanteil $\rho_{\text{grün}}$ gilt, dass $\rho_{\text{grün}} \geq \tau_{\text{grün}}$, andernfalls rot. Feldgrenzenzellen sind blau markiert. Die Tabellen enthalten die Thresholdwerte der jeweiligen Datensätze. τ_s gibt die Höhe und Breite von Zellen in Pixeln an, τ_{min} die Mindestanzahl an grün markierten Zellen vor einer Feldgrenzenzelle, τ_{max} die maximale Anzahl an Zellen die grün unterbrechnen dürfen und τ_{Grenze} den minimalen Grünanteil einer Feldgrenzenzelle. In Bild a haben Objekte teilweise die Feldgrenze verdeckt. Feldgrenzenpunkte wurden in Objekten erkannt, aber durch die Berechnung der konvexen Hülle ausgeglichen. In Bild b wurde das Feld teilweise zu klein erkannt.

von Parametern der vorgestellten Felderkennungsmethode auf den Evaluierungsdaten beschäftigen. In Abbildung 2.3.2 sind die Thresholdwerte für das RoboCup 2018 Feld und das German Open 2018 Feld zusammengefasst.

Von der Zellengröße τ_s ist abhängig, wie viel Kontext im Bild betrachtet werden kann. Wenn wir τ_s vergrößern, kann die Methode robuster sein, andere Objekte auf dem Spielfeld, wie zum Beispiel Roboter, haben dann jedoch einen verstärkten Einfluss auf die Genauigkeit der Erkennung. Zellen werden unter Umständen nicht als Grün markiert wenn Teile von Objekten enthalten sind. Im oberen Kamerabild sind in der Regel mehr Objekte enthalten. Für eine Verbesserung der Genauigkeit haben wir uns bei der oberen Kamera für eine kleine Zellengröße entschieden.

In Datensatz a haben wir das häufige Auftreten von grünklassifizierten Pixeln außerhalb

der Feldgrenzen beobachtet. Damit das Feld nicht durch einzelne grün markierte Zellen außerhalb der Feldgrenzen zu groß erkannt wird, haben wir festgelegt, dass Feldgrenzenzellen von mindestens 4 grünen Zellen prädestiniert werden müssen (τ_{min}). Außerdem haben wir die Möglichkeit ausgeschaltet, dabei Zellen zu überspringen (τ_{max}) und den Grünanteil von Feldgrenzenzellen genauso wie von normalen Feldzellen betrachtet ($\tau_{Grenze} = \tau_{grün}$).

In Datensatz b dagegen sind Fehler der Grünerkennung außerhalb der Feldgrenzen seltener. Damit das Feld nicht zu klein erkannt wird, haben wir den Threshold für die Gründichte von Feldgrenzen τ_{Grenze} reduziert.

2.4 Linienkantenerkennung

Feldlinien können durch helle, weiße Bereiche im Bild charakterisiert werden, die von zwei parallelen Linienkanten eingeschlossen sind. In diesem Abschnitt wollen wir Kantenelemente (Edgels) der Feldlinien und des Mittelfeldkreises im Bild identifizieren und Linienkandidaten sammeln.

Die Bildverarbeitung auf beiden Kameras ist eine der rechenintensivsten Aufgaben des Roboters. Kamerabilder werden im NaoTH-Framework [MSK+18] mit einer Auflösung von jeweils 640×480 Pixeln verarbeitet. Die CPU des Roboters ist nicht leistungsstark genug, um bei der Kantensuche alle Pixel der Kamera zu betrachten. Um damit umzugehen, wird oft auf zwei klassische Methoden zurückgegriffen: Die Unterabtastung des Bildes und die Einteilung in Scanlinien.

In den SPL-Regeln [Com19] ist festgelegt, dass die Breite einer Feldlinie 50 mm beträgt. Dies führt dazu, dass Linien besonders in nahen und mittleren Entfernungen mehrere Pixel einnehmen. Es liegt nahe, eine sogenannte Unterabtastung des Bildes vorzunehmen und horizontal und vertikal nur jeden u-ten Pixel zu scannen. Dies verringert den Rechenaufwand um das u-fache und hat darüber hinaus den Vorteil, dass die gesamte Erkennung robuster wird. Bereiche im Bild, die fälschlicherweise durch den Einfluss von Fehlerfaktoren wie unterschiedlichen Lichtverhältnissen oder Rauschen einer Linie ähneln, sind oft klein, sodass sie bei steigendem u eher übersehen werden. Allerdings verringert sich die Genauigkeit bei der Positionsbestimmung von detektierten Objekten. Vor allem in weiten Entfernungen bilden einzelne Pixel größere Bereiche auf der Bodenebene ab, was einen hohen Abstand zwischen benachbarten projizierten Pixeln auf dem Feld zur Folge (Abbildung 2.4.1) hat.

Eine weitere Möglichkeit ist die Einschränkung der Suche auf sogenannte Scanlinien. Dabei werden nur die Pixel entlang weniger horizontaler und vertikaler Linien im Bild untersucht. Die Lokalisierung von Objekten, die von Scanlinien geschnitten werden, können somit genauer erfolgen. Der Nachteil ist, dass Scanlinien eindimensional sind und somit nur ein kleiner Teil der Umgebung betrachtet werden kann [LDHB+09]. Dies führt dazu, dass die Erkennung stärker von Rauschen beeinflusst wird.

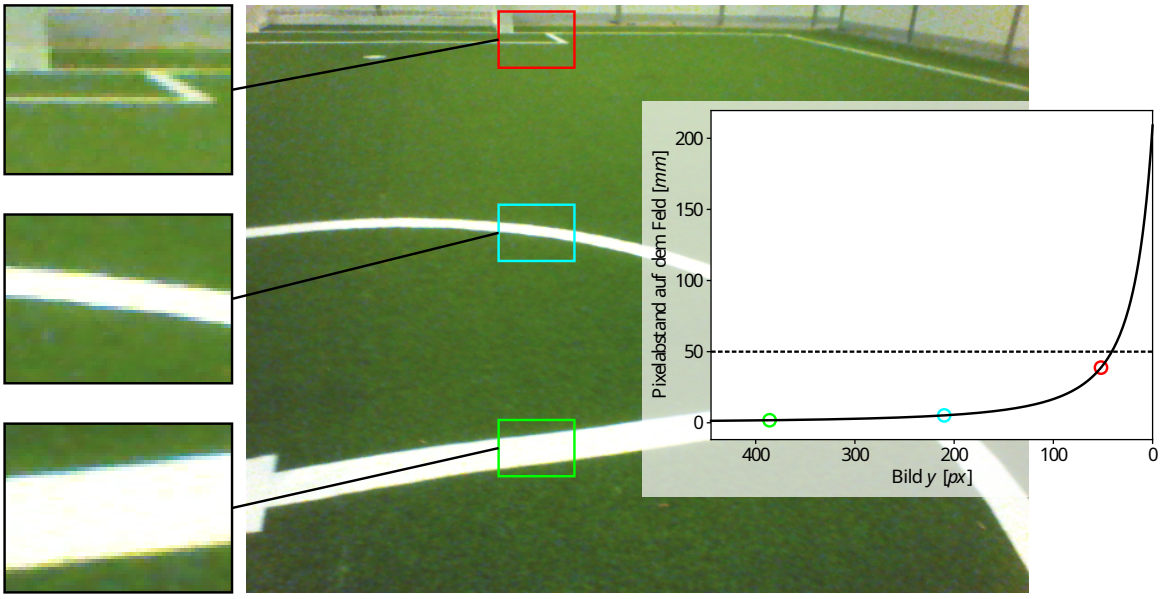


Abbildung 2.4.1: 64×48 Pixelausschnitte von Feldlinien im Bild in naher (grün), mittlerer (blau) und weiter (rot) Entfernung. Der Graph stellt die Abstände auf der Bodenebene zwischen benachbarten, projizierten Pixeln abhängig von der y-Koordinate im Bild dar. Die gestrichelte Linie visualisiert die Breite von Feldlinien nach den SPL-Regeln [Com19]. Sie bestimmt den maximalen Abstand zwischen Abtastpunkten auf der Bodenebene, bevor horizontale oder vertikale Feldlinien bei der Abbildung übersprungen werden können.

Im Folgenden wollen wir einen adaptiven Scanlinienalgorithmus zur Kantenerkennung betrachten, der das Ziel verfolgt, die Vorteile beider Methoden zu verknüpfen.

2.4.1 Adaptive Scanlinien

Im NaoTH-Framework [MSK+18] ist eine Kantenerkennung anhand von vertikalen Scanlinien implementiert. Um zusätzlich Rechenaufwand zu sparen, wird auf den Scanlinien eine Unterabtastung mit einem Faktor $u = 2$ angewendet.

Bei der Verwendung von ausschließlich vertikalen Scanlinien, kann der Roboter nur näherungsweise horizontal ausgerichtete Kanten erkennen. Außerdem beachten wir bei gleichmäßiger Aufteilung von Scanlinien mit einer statischen Abtastrate nicht die Abstände zwischen Abtastpunkten auf der Bodenebene.

Im Folgenden betrachten wir ein adaptives Unterabtastungsverfahren auf horizontalen und vertikalen Scanlinien unter Einbezug der perspektivischen Projektion, inspiriert von Härtl et. al. [HVR13]. Anders als im NaoTH-Framework wenden Härtl et. al. auf den berechneten Abtastpunkten ein Region-Growing-Verfahren an, um Farbregionen im Bild zu identifizieren und eine anschließende Objekterkennung zu ermöglichen.

Anstatt eines Region-Growing-Ansatzes wollen wir auf den adaptiven Scanlinien eine Kantenerkennung anwenden. Die Extraktion von Kanten kann unabhängig vom Farbwert der Pixel erfolgen und ist robuster bei unterschiedlichen Lichtverhältnissen.

Mit einer adaptiven Unterabtastung auf Scanlinien wollen wir Rechenleistung in unteren Teilen des Bildes sparen, um die Abtastrate in den oberen Bildbereichen zu erhöhen. Wichtig ist, dass Feldlinien möglichst von mindestens einem Abtastpunkt getroffen werden. Somit können wir in Abschnitt 2.4.2 Intervalle identifizieren, in denen eine Feldlinienkante enthalten ist. In diesem Intervall kann dann ein feinerer, pixelweiser Scan durchgeführt werden, um die genaue Position der Kante zu finden.

Bei der Verwendung von klassischen Scanlinien können wir die Menge der abzutastenden Pixel durch zwei Parameter beeinflussen: Die Anzahl von Scanlinien und die jeweilige Abtastrate.

Die Anzahl von vertikalen, bzw. horizontalen Scanlinien N_v und N_h ist abhängig von der Lücke δ zwischen Scanlinien im Bild. Je geringer der Abstand, desto mehr Edgels können auf einer Feldlinienkante gefunden und Linienkandidaten extrahiert werden. Und umso höher kann die Erkennungsrate einer anschließenden Linienerkennung sein.

Die Abtastrate r kann ähnlich wie bei der Unterabtastung entlang einer Scanlinie reduziert werden. Dies wirkt sich auf die Wahrscheinlichkeit aus, mit der eine Feldlinie registriert werden kann. Damit Feldlinien von mindestens einem Scanpunkt einer schneidenden Scanlinie getroffen werden, sollte der Abstand zwischen Abtastpunkten im Bezug zur Bodenebene die Breite einer Feldlinie (50 mm) nicht überschreiten.

Beide Auswirkungen der Parameter sind davon abhängig, wie weit eine Feldlinie vom Roboter entfernt ist. Weit entfernte Linien im Bild erfordern eine hohe Abtastrate, nahe Linien können schon mit wenigen Abtastpunkten erfasst werden. Wenn wir überflüssige Abtastungen vermeiden wollen, müssen wir bei unserer Betrachtung die Lage der Bodenebene mit einbeziehen.

Der Roboter bewegt sich während des Spiels fast ausschließlich aufrecht. Die Rotation der Kamera um die x-Achse in Roboterkoordinaten ist klein und kann vernachlässigt werden. Wir nehmen an, dass adaptive Abstände zwischen Abtastpunkten unabhängig von der x-Koordinate im Bild berechnet werden können [HVR13].

Ein adaptiver Abstand $\delta(y)$ zwischen vertikalen Scanlinien, abhängig von der y-Koordinate im Bild, kann basierend auf einem gleichmäßigem Abstandsparameter τ_δ auf der Bodenebene mithilfe des Strahlensatzes wie folgt berechnet werden:

$$\delta(y) = \frac{f \cdot \tau_\delta}{\sqrt{y^2 + C_x^2}}, \tag{2.4.1}$$

wobei f die Brennweite der Kamera und C_x die x-Koordinate der Kameraposition in Roboterkoordinaten repräsentiert.

Algorithmus 2.4 : Berechnung der adaptiven vertikalen Scanlinien.

Eingabe : Anzahl von vertikalen Scanlinien N_v, minimaler Abstand δ_{min} im Bild, Maximaler Abstand τ_δ auf der Bodenebene, y-Wert unterhalb der Horizontlinie y_H, Brennweite f, Kameramatrix $\boldsymbol{C}$, Bildhöhe h

Ausgabe : Liste von vertikalen Scanlinien $v \in V$ mit $v = (y\text{-Startwert } v_s,\ y\text{-Endwert } v_e,\ x\text{-Position } v_x)$

$\delta = \delta_{min}$;
Scanlinienfrequenz $\nu = 2$;
$i = 1$;
solange $i \leq N_v$ **tue**
    `# Berechne Startwert` y `von Bereich mit Scanlinienabstand` δ `im Bild (Umstellung von Gleichung 2.4.1 nach` y`)`
    $y = \sqrt{(f \cdot \frac{\tau_\delta}{\delta})^2 - C_x^2}$;
    $j = i - 1$;
    solange $j < N_v$ **tue**
        Erstelle Scanlinie v mit $\begin{pmatrix} v_s = \text{Kleinerer Wert von } y \text{ und } h-1 \\ v_e = y_H \\ v_x = j \cdot \delta_{min} \end{pmatrix}$;
        Füge v zur Liste von vertikalen Scanlinien V hinzu;
        $j = j + \nu$;
    verdopple i;
    verdopple Abstand δ;
    verdopple Scanlinienfrequenz ν;

Bei vertikalen Scanlinien können wir $\delta(y)$ über die Länge der Scanlinien steuern. Sei s_v der y-Wert des Startpunktes einer vertikalen Scanlinie v_x an der unteren Bildgrenze und e_v der y-Wert des Endpunktes an der oberen Bildgrenze, bzw. an der Horizontlinie l_H. Eine Verkürzung $s'_v = y'$ von v_x, mit $e_v < y' < s_v$, verdoppelt den Abstand zwischen den benachbarten vertikalen Scanlinien für alle $y > y'$ im Bild und auf der Bodenebene.

Die Anzahl erforderlicher vertikaler Scanlinien N_v und den minimalen Abstand zwischen Scanlinien δ_{min} für einen maximalen Abstandsparameter τ_δ auf der Bodenebene, lässt sich an der oberen Bildgrenze, bzw. an der Horizontlinie l_H berechnen, sodass

$$N_v = \frac{||p_2 - p_1||}{\tau_\delta} \quad \text{und} \quad \delta_{min} = \frac{w}{N_v}.$$

Die Punkte p_1 und p_2 sind die Projektionen der Bildpunkte $[0, y_H]^T$ und $[w-1, y_H]^T$ auf der Bodenebene, wobei w die Bildbreite und y_H den y-Wert unterhalb der Horizontlinie, bzw. am oberen Rand des Bildes bezeichnet, sodass

$$y_H = max\left(G(0), G(w-1)\right). \tag{2.4.2}$$

$G(x)$ ist die Funktion der oberen Begrenzung des Bildes durch die Horizontlinie, die wir in Gleichung 2.1.16 in Abschnitt 2.1.3 definiert haben. Wenn der Horizont unterhalb des Bildes liegt, muss keine Objekterkennung durchgeführt werden, weil der Roboter nicht auf das Feld schaut.

In Algorithmus 2.4 ist die Berechnung der vertikalen Scanlinien angegeben. In Zeile 5 werden y-Startwerte der Dichtebereiche für Verdopplungen des minimalen Abstandes δ_{min} berechnet.

Der oberste Dichtebereich mit einem Scanlinienabstand δ_{min} enthält alle Scanlinien N_v. Um die Abstände im nächsten Dichtebereich zu verdoppeln, müssen die Anzahl der darin enthaltenen Scanlinien halbiert werden. Dazu verkürzen wir jede zweite Scanlinie auf den Startwert des ersten Dichtebereiches. Somit sind die kürzesten Scanlinien nur im obersten Dichtebereich enthalten. Mit den folgenden Dichtebereichen gehen wir analog vor, sodass sich die Anzahl von vertikalen Scanlinien in tieferen Bereichen des Bildes immer wieder halbiert. Die Laufvariable j bestimmt die x-Position von Scanlinien gleicher Länge als j-faches von δ_{min} und i bestimmt den Anfangswert von j, sodass $j_0 = i - 1$.

Algorithmus 2.5 : Berechnung einer Menge von Abtastpunkten mit einer adaptiven Abtastrate $r_v(y)$

Eingabe : Abstandsparameter τ_r auf der Bodenebene in x_R-Richtung, Startpunkt auf der y-Achse in Bildkoordinaten y_0, y-Wert unterhalb der Horizontlinie y_H, Bildbreite w

Ausgabe : Liste von Abtastpunkten im Bild für eine vertikale Abtastrate $r(y)$

```
# Projektion auf die Bodenebene nach Abschnitt 2.1.2
```

1 $\boldsymbol{s}' \leftarrow$ Punkt $\begin{bmatrix} w/2 & y_0 \end{bmatrix}^T$ projiziert auf die Bodenebene in Weltkoordinaten;

2 $\boldsymbol{e}' \leftarrow$ Punkt $\begin{bmatrix} w/2 & y_H \end{bmatrix}^T$ projiziert auf die Bodenebene in Weltkoordinaten;

```
# Der x-Wert in Weltkoordinaten entspricht der Tiefe
```

3 **solange** $s'_x <= e'_x$ **tue**

4 $\quad \boldsymbol{s} \leftarrow \boldsymbol{s}'$ rückprojiziert auf die Bildebene;

5 $\quad$ füge $\boldsymbol{s}$ zur Liste von Abtastpunkten hinzu;

$\quad$ `# Berechne nächsten Punkt` $\boldsymbol{s}'$ `mit Abstand` τ_r

6 $\quad s'_x = s'_x + \tau_r$;

Eine Menge von Abtastpunkten auf den vertikalen Scanlinien mit einer adaptiven Abtastrate $r(y)$ im Bild erhalten wir durch Rückprojektion von Scanpunkten mit konstanten Abständen τ_r entlang der x-Achse in Weltkoordinaten auf der Bodenebene (Algorithmus 2.5).

Weil wir annehmen, dass die Abstände unabhängig von der x-Koordinate im Bild sind, ist die Abtastrate r_h entlang einer horizontalen Scanlinie h konstant. Wir können r_h

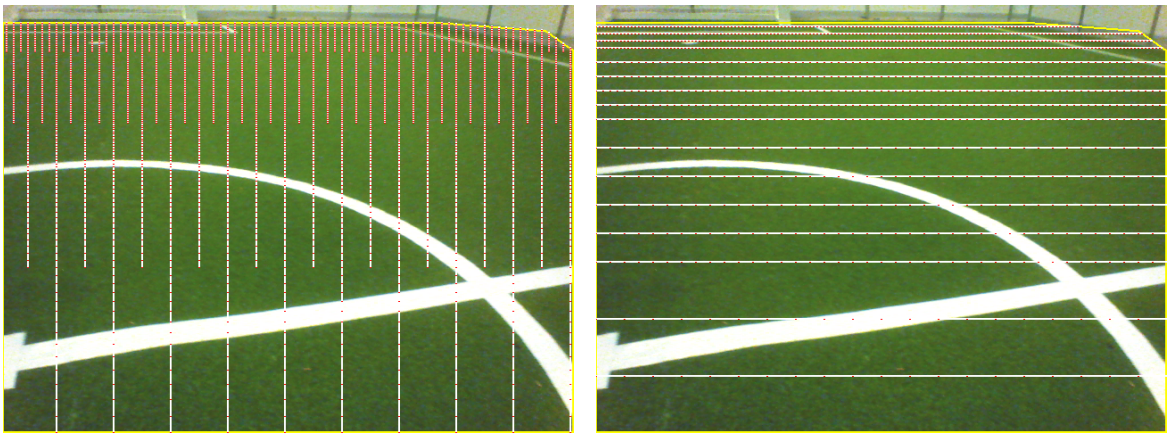
über die vertikale Abtastrate $r(h_y)$ an der y-Position von h ermitteln, sodass

$$r_h = r(h_y) \tag{2.4.3}$$

Dazu wählen wir die nächsten zwei Abtastpunkte p_1 und p_2 zu h_y aus der Liste vertikaler Abtastpunkte von Algorithmus 2.5. Der Abstand r_h zwischen Abtastpunkten auf h ergibt dann

$$r_h = ||p_2 - p_1||. \tag{2.4.4}$$

Der Abstand $\delta(y)$ zu der nächsten Scanlinie entspricht dem Abstand δ des Dichtebereiches, indem die horizontale Scanlinie enthalten ist. Die y-Startwerte der Dichtebereiche haben wir in Zeile 5 von Algorithmus 2.4 berechnet und der zugehörige Abstand ist der aktuelle Wert von δ.



(a) (b)

Abbildung 2.4.2: Vertikale (a) und horizontale (b) Scanlinien mit einem Abstand τ_δ von 50 mm und einer Abtastrate τ_r von 25 mm. Die Längen aller Scanlinien wurden durch die Berechnung der Schnittpunkte an das detektierte Feldpolygon (gelb) aus Abschnitt 2.3 angepasst. Feldlinien, die von den Scanlinien geschnitten werden, können durch mindestens einem Scanpunkt (rot) erfasst werden.

Ein Beispiel für die berechneten Scanlinien eines Bildes der oberen Kamera ist in Abbildung 2.4.2 dargestellt.

2.4.2 Linienkantenerkennung auf adaptiven Scanlinien

Im Folgenden betrachten wir die Kantenerkennung am Beispiel der vertikalen Scanlinien v. Das hier vorgestellte Verfahren kann analog auf horizontale Scanlinien übertragen werden.

Die Kameras des Roboters geben Bilder im YUV-Format zurück. Das YUV 4:2:2 Format verfügt über einen Lumakanal Y, der die Helligkeit von Pixeln angibt und zwei Chromakanäle U und V, welche die Farbinformationen enthalten. Allerdings werden die Chromakanäle in horizontaler Richtung unterabgetastet. Jede Gruppe von 2×2 Pixeln enthält demnach 4 Y Kanäle und jeweils 2 U und V Kanäle [Vid19].

Die zwei Kanten einer Feldlinie zeichnen sich durch rapide Helligkeitsveränderungen im Bild aus und können durch den Y Kanal erfasst werden. Veränderungen in der Helligkeit können wir durch den Gradienten $f'_x(y)$ beschreiben [Rei11], sodass

$$f'_x(y) = \frac{1}{2} \cdot (f_x(y-1) - f_x(y+1)) \tag{2.4.5}$$

für eine regelmäßige vertikale Scanlinie f_x mit maximaler Abtastrate und einer Abtastung des Bildes von unten nach oben.

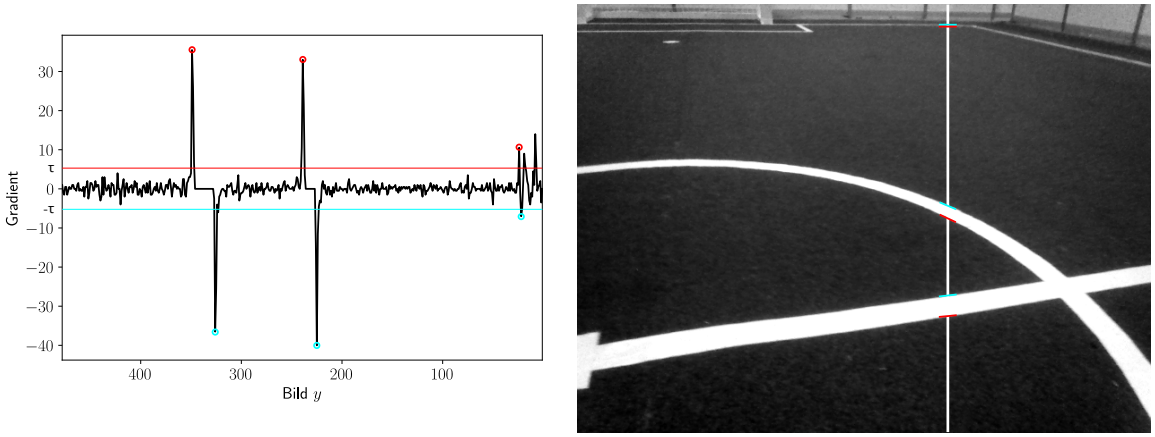
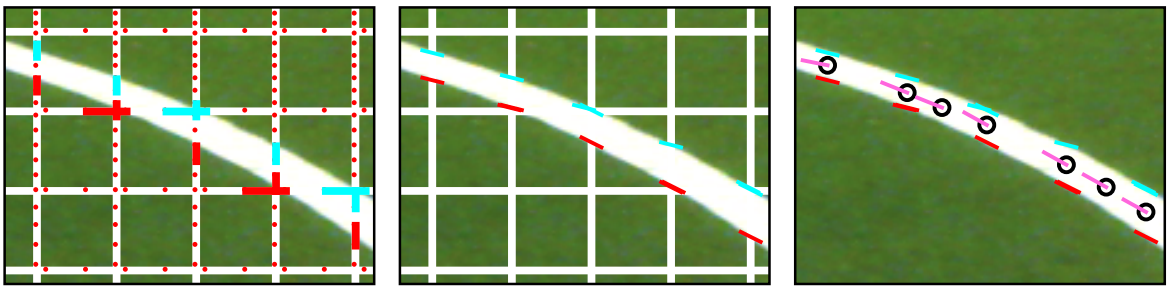


Abbildung 2.4.3: Ableitung des Y-Kanals von Pixeln entlang einer Scanlinie mit maximaler Abtastrate, die drei Feldlinien schneidet. Die Hochkanten (rot) und Tiefkanten (blau) von Feldlinien können als lokale Maxima oberhalb eines Gradiententhresholds τ, bzw. als lokale Minima unterhalb eines Thresholds $-\tau$ in der Ableitung identifiziert werden. Die Ausschläge des Gradienten oberhalb von τ nach der dritten Feldlinie können wir filtern, weil jeweils eine zugehörige Tiefkante fehlt. Außerdem kann der Scan durch die Felddetektion aus Abschnitt 2.3 frühzeitig abgebrochen werden.

Im Gradienten lässt sich die erste Kante einer Feldlinie als lokales Maximum bestimmen. Wir bezeichnen sie als Hochkante. Die darauffolgende zweite Feldlinienkante können wir als lokales Minimum im Gradienten identifizieren. Wir bezeichnen sie als Tiefkante. Bei eine Abtastung der Scanlinie von unten nach oben, treten Feldlinien als Hoch- und Tiefkantenpaare auf (Abbildung 2.4.3).

Weil die Abstände zwischen Scanpunkten auf adaptiven Scanlinien unterschiedlich sind und Pixel übersprungen werden, können wir die genaue Positionen der Kanten nicht bestimmen.



(a) (b) (c)

Abbildung 2.4.4: Die 3 Schritte der Kantenerkennung. (a) Identifizierung von Hochkantenintervallen (rot) und Tiefkantenintervallen (blau), in denen eine Kante enthalten ist, mithilfe der berechneten Abtastpunkte auf horizontalen und vertikalen Scanlinien aus Abschnitt 2.4.1. (b) Feiner pixelweiser Scan innerhalb der Intervalle um ein Kantenelement (Edgel) zu finden und anschließend die Edgelrichtungen zu bestimmen. (c) Kombination von Hoch- und Tiefkantenedgels zu Doppeledgels, die auf den Feldlinien im Bild liegen.

Sei $s(i)$ eine adaptive Scanlinie mit $i \in [1, \ldots, N_{px}]$ und N_{px} die Anzahl der Abtastpunkte von $s(i)$. Für einen ersten Scandurchlauf wollen wir uns mit der Erkennung von Intervallen $[s(i-1), s(i)]$ beschäftigen, in denen eine Kante enthalten ist. Anschließend können wir mit einem feinen, pixelweisen Scan innerhalb der Intervallgrenzen die genaue Position der Kante bestimmen (Abbildung 2.4.4).

Für die Erkennung von Hoch- und Tiefkanten von Feldlinien auf gleichmäßigen Scanlinien mit einer Abtastrate von 2 Pixeln stellt Reinhardt [Rei11] einen effizienten Algorithmus zur Suche von lokalen Extrema im Gradienten oberhalb eines Thresholds τ_{Kante} vor. Die Methode wird auch im NaoTH-Framework [MSK+18] für die Kantenerkennung verwendet.

In Algorithmus 2.6 ist eine Adaption für die Suche nach Hochkantenintervallen beschrieben. Bei der Umkehrung des Gradienten kann analog auch das Tiefkantenintervall gefunden werden. In Zeile 4 berechnen wir den Punkt der zwischen $s(i-1)$ und $s(i)$ liegt. Durch die Berechnung des Gradienten an diesem Punkt können wir entscheiden, ob ein Höhepunkt innerhalb des Intervalls vorliegt.

Um Kantenelemente, die sogenannten Edgels, innerhalb der Intervallgrenzen zu identifizieren, scannen wir alle Punkte in den Intervallen und berechnen den Gradienten nach Gleichung 2.4.5. In jedem Intervall brauchen wir nur nach einer Kante zu suchen. Wir erkennen eine Hochkante am Gradientenhöhepunkt oberhalb des schon vorher verwendeten Thresholds τ_{Kante} und eine Tiefkante am Gradiententiefpunkt unterhalb eines Thresholds $-\tau_{\text{Kante}}$. Es ist möglich, dass nicht alle gefundenen Intervalle einen Edgel enthalten, weil die Gradienten auf unterabgetasteten Scanlinien höher ausschlagen als die Gradienten des feinen Scans.

Für die Bestimmung von Richtungsvektoren der Edgels hat Reinhardt [Rei11] drei

Algorithmus 2.6 : Hochkantenintervallsuche auf einer vertikale Scanlinie s_x, adaptiert von Reinhardt [Rei11]

Eingabe : Threshold τ_{Kante}, vertikale adaptive Scanlinie mit Abtastpunkten geordnet von unten nach oben $s_x(i)$, Anzahl von Abtastpunkten N_{px}

Ausgabe : Liste l von Hochkantenintervallen $[s_x(i-1), s_x(i)]$

```
1  Höhepunkt ← τ_Kante;
2  Höhepunkt gefunden ← falsch;
3  für alle i ∈ [1, N_px] tue
4      y = (s_x(i−1)+s_x(i))/2 ;
       # Berechne Gradienten an Position y
5      g_y = (s_x(i)−s_x(i−1))/2;
6      wenn g_y ≥ Höhepunkt dann
7          Höhepunkt gefunden ← wahr;
8          Höhepunkt ← g_y ;
9          Höhepunktintervall ← [s_x(i−1), s_x(i)];
10     sonst
11         wenn Höhepunkt gefunden dann
12             Füge Höhepunktintervall zur Liste l hinzu;
13         Höhepunkt ← τ_Kante;
```

verschiedene Faltungsoperatoren zur Approximation der Gradientenrichtung auf den Kamerabildern des Roboters evaluiert. Dabei hat der Sobeloperator den geringsten Fehler produziert.

Für die Approximation des Gradienten g_x und g_y für den Pixel $p(x, y)$ wird eine Faltung mit der Umgebungsmatrix von p mit den Sobeloperatoren durchgeführt, sodass

$$g_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} p(x-1,y-1) & p(x,y-1) & p(x+1,y-1) \\ p(x-1,y) & p(x,y) & p(x+1,y) \\ p(x-1,y+1) & p(x,y+1) & p(x+1,y+1) \end{bmatrix} \tag{2.4.6}$$

$$g_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \begin{bmatrix} p(x-1,y-1) & p(x,y-1) & p(x+1,y-1) \\ p(x-1,y) & p(x,y) & p(x+1,y) \\ p(x-1,y+1) & p(x,y+1) & p(x+1,y+1) \end{bmatrix} \tag{2.4.7}$$

Aus den Gradienten g_x und g_y ergibt sich die Richtung des Edgels $\boldsymbol{d} = [g_x \; g_y]^T$.

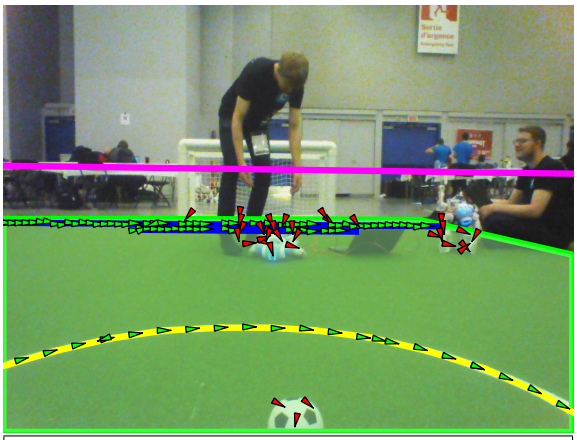
Mit den Kombinationen von detektierten Tiefkantenedgels $e_t = \{\boldsymbol{p_t}, \boldsymbol{d_t}\}$ und Hochkantenedgels $e_h = \{\boldsymbol{p_h}, \boldsymbol{d_h}\}$ können wir schließlich Doppeledgels $e = \{\boldsymbol{p}, \boldsymbol{d}\}$ berechnen, die

auf den Feldlinien im Bild liegen, sodass

$$\boldsymbol{p} = \frac{(\boldsymbol{p_h} + \boldsymbol{p_t})}{2} \quad \text{und} \quad \boldsymbol{d} = \boldsymbol{d_h} - \boldsymbol{d_t}. \tag{2.4.8}$$

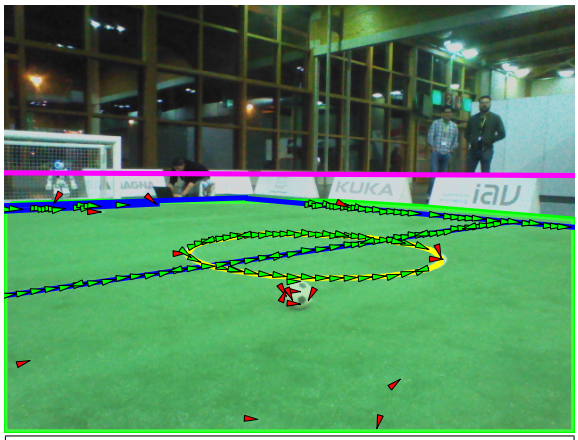
Durch die Kombination von $\boldsymbol{d_h}$ und $\boldsymbol{d_t}$ verringert sich der Fehler durch die Abschätzung der Gradientenrichtung.

2.4.3 Auswertung der Linienkantenerkennung



ROI	Anzahl	OUT	TP	FP	PRC
Horizont	23946	11629	9097	14849	0.38
Feld	13071	1321	8943	4128	0.68
Truth	12317	0	9097	3220	0.74

(a) RoboCup 2018



ROI	Anzahl	OUT	TP	FP	PRC
Horizont	22961	7423	11976	10985	0.52
Feld	14347	168	11467	2880	0.80
Truth	15538	0	11976	3562	0.77

(b) German Open 2018

Abbildung 2.4.5: Kantenerkennung auf den Evaluierungsdatensätzen. Die erkannten Doppeledgels wurden als Pfeile für den Vergleich mit den annotierten Feldlinien (blau) und Kreislinienzug (gelb) eingezeichnet. Pfeile zeigen auf die Position des Doppeledgels und die Pfeilrichtung entspricht der Ausrichtung. True-Positives sind grün markiert, False-Positives hingegen rot. Außerdem wurde die Horizontlinie (Magenta) dargestellt. Die Tabellen zeigen die True-Positives (TP), False-Positives (FP) und die Precision (PRC) der Doppeledgelerkennung auf allen Bildern des jeweiligen Datensatzes auf drei verschiedenen ROIs. Für jede ROI ist die Anzahl detektierter Doppeledgels angegeben und die Anzahl von Doppeledgels außerhalb des annotierten Feldpolygons (OUT). Bei beiden Beispielbildern wurden Doppeledgels fehlerhaft im Ball erkannt. In Bild (a) sind falsch detektierte Doppeledgels in Robotern zu sehen. In Bild (b) gibt es vereinzelte Fehldetektionen aufgrund von Rauschen. Im oberen Bereich wurde das Feld nicht komplett erkannt, sodass Doppeledgels fehlen.

Im Folgenden wenden wir die vorgestellte Kantenerkennung mit adaptiven Scanlinien auf den Testbildern der Evaluierungsdatensätze an. Dazu betrachten wir drei verschiedene

Region of Interests (ROI) für die Kantenerkennung und vergleichen die Auswirkung auf die Performance:

1. Die ROI ist der Bildbereich unterhalb des Horizonts und spiegelt eine unzureichende Felderkennung wieder

2. Die ROI ist das detektierte Feldpolygon aus Abschnitt 2.3.

3. Die ROI ist das ideale annotierte Feldpolygon aus den Datensätzen.

Für die Auswertung vergleichen wir erkannte Doppeledgels mit den annotierten Feldlinien und Mittelkreislinienzügen. Linienzüge teilen wir in einzelne Strecken auf.

Weil annotierte Strecken nur näherungsweise auf der Mitte von Feldlinien und Mittelfeldkreis liegen, können wir die Genauigkeit der Kantenerkennung nicht auswerten. Der Fokus soll hier auf der Identifizierung von fehlerhaft detektierten Doppeledgels liegen.

Ein Doppeledgel ist ein True-Positive (TP), wenn seine Parameter mit der nächstliegenden annotierten Strecke ungefähr übereinstimmen. Bei unserer Überlegung müssen wir den Fehler bei der Annotation mit einbeziehen. Richtungen von Strecken auf dem Mittelfeldkreis spiegeln nur näherungsweise die Tangentenrichtung wieder. Wir legen fest, dass ein TP innerhalb des annotierten Feldpolygons liegen muss und nicht mehr als 5 Pixel von der nächstgelegenen Linie entfernt sein darf. Außerdem soll die Richtungsabweichung zu annotierten Strecken nicht mehr als 16° betragen. Andernfalls handelt es sich um einen False-Positive (FP).

Aus der Anzahl der TPs N_{TP} und FPs N_{FP} können wir die Precision (PRC) berechnen, sodass

$$PRC = \frac{N_{TP}}{N_{TP} + N_{FP}}. \tag{2.4.9}$$

Die Precision gibt den Anteil der validen Doppeledgels von allen erkannten Doppeledgels wieder.

Die Ergebnisse der Kantenerkennung auf den Datensätzen des RoboCup 2018 und der German Open 2018 sind in Abbildung 2.4.5 zusammengefasst.

Die Precision bei der Verwendung des Horizonts als ROI ist auf beiden Datensätzen mangelhaft. In Datensatz a wurde die Hälfte der Doppeledgels außerhalb des Feldes erkannt. Die Felderkennung verbessert die Precision deutlich. Dabei hat das erkannte Feld in Datensatz b die Precision stärker erhöht. Bei manueller Betrachtung von Einzelbildern der Datensätze ist auffällig, das die Grünerkennung für das Spielfeld bei der German Open 2018 deutlich bessere Ergebnisse liefert, sodass eine Felderkennung einfacher ist. Dies führt zu einer besseren Filterung von Doppeledgels außerhalb des Spielfeldes.

In Datensatz a wurde weiterhin eine hohe Anzahl von Doppeledgels außerhalb des wahren Feldpolygons detektiert und die generelle Anzahl der erkannten Doppeledgels ist höher. Dies lässt darauf schließen, dass das Feld häufiger zu groß detektiert wurde.

Bei Datensatz b dagegen, wurden weniger Doppeledgels detektiert als bei der Verwendung des wahren Feldpolygons als ROI. Es ist anzunehmen, dass das Feld eher zu klein detektiert wurde. In den Bildern wurden fehlerhafte Edgels häufig in den Teilen des Tores in der Nähe des Spielfeldrandes erkannt, sodass die zu kleine Felderkennung auf dem German Open Datensatz eine bessere Precision gegenüber dem wahren Feldpolygon als ROI erzielen kann.

Insgesamt hat die Kantenerkennung bessere Ergebnisse bei Datensatz b geliefert. Dies lässt sich mit einem höheren Auftreten von unbekannten Objekten in Datensatz a erklären. Doppeledgels werden fälschlicherweise in Menschenbeinen und Lan-Kabeln erkannt.

Auf den Datensätzen wurden falsche Doppeledgels außerdem in Robotern und Bällen detektiert. Roboter und Bälle sind weiß und die Helligkeitsveränderungen ähneln denen von Feldlinien.

2.5 Extraktion von Linienkandidaten

In diesem Abschnitt wollen wir uns mit der Gewinnung von Linienkandidaten aus den erkannten Edgels beschäftigen. Kandidaten sind gerichtete Linienpunkte, welche wir durch die Projektion von Edgels auf die Bodenebene berechnen. Dies hat den Vorteil, dass Linienkanteninformationen aus der oberen und unteren Kamera zusammengeführt werden können. Außerdem vereinfacht die Projektion das Matching mit Linien und Kreisen. Wenn wir eine hypothetische Feldlinie mit Linienkandidaten vergleichen sind Abstandsfehler im Bild abhängig von der y-Koordinate. Diese Abhängigkeit kann durch die Projektion von Positionen auf die Bodenebene aufgelöst werden.

Die Positionen der Edgels auf der Bodenebene können mit den Werkzeugen aus Abschnitt 2.1.2 berechnet werden. Eine Bestimmung der Richtung kann so jedoch nicht erfolgen.

Für die anschließende Linienerkennung ist es wichtig, dass der Richtungsvektor eines Linienpunktes auf der Bodenebene die Richtung einer zugehörigen Feldlinie widerspiegelt. Wir können die Position $\boldsymbol{p}_k$ und Richtung $\boldsymbol{d}_k$ eines Linienkandidaten k auf der Bodenebene aus zwei benachbarten Hoch- und Tiefkantenedgelpaaren (e_{t_1}, e_{h_1}) und (e_{t_2}, e_{h_2}) berechnen, sodass

$$\boldsymbol{p}_k = \frac{\boldsymbol{p}_1 + \boldsymbol{p}_2}{2} \quad \text{und} \quad \boldsymbol{d}_k = \frac{\boldsymbol{p}_2 - \boldsymbol{p_1}}{||\boldsymbol{p}_2 - \boldsymbol{p_1}||} \quad \text{mit} \tag{2.5.1}$$

$$\boldsymbol{p}_1 = \frac{e'_{t_1} + e'_{h_1}}{2} \quad \text{und} \quad \boldsymbol{p}_2 = \frac{e'_{t_2} + e'_{h_2}}{2}, \tag{2.5.2}$$

wobei $e'_{t_1}, e'_{h_1}, e'_{t_2}$ und e'_{h_2} die Positionen der jeweiligen Edgels auf Bodenebene sind.

Algorithmus 2.7 : Suche nach Doppeledgelnachbarn auf adaptiven Scanlinien

Eingabe : Doppeledgels $e_n = \{\boldsymbol{p}_n, \boldsymbol{d}_n\}$, vertikale Scanlinien v_x, minimale Ähnlichkeit τ_s
Ausgabe : Liste von Paaren u_m

```
1  e_n ← Sortiere Doppeledgels e_n aufsteigend nach ihren x-Koordinaten;
2  für alle i ∈ {1,...,n} tue
       # Suche nach Doppeledgelnachbar e_{j_max} mit maximaler Ähnlichkeit s_max
3      j_max = −1, s_max = 0;
       # x-Position der benachbarten Scanlinie
4      r_x = −1;
5      y_min = 0;
6      für alle j ∈ {i,...,n} tue
7          wenn p_{i,x} = p_{j,x} dann
               # Doppeledgels auf gleicher Scanlinie sind keine Nachbarn
8              Schleife fortfahren;
9          wenn p_{j,x} ≠ r_x dann
10             wenn r_x ≠ −1 dann
11                 wenn y_max von v_{r_x} < y_min dann
12                     y_min = y_max von v_{r_x};
13             r_x = p_{j,x};
14         wenn p_{j,y} ≤ y_min dann
               # Wir haben schon eine Scanlinie unterhalb von y_min untersucht
15             Schleife fortfahren;
16         s ← Berechne Ähnlichkeit zwischen e_i und e_j nach Gleichung 2.5.4 ;
17         wenn s > s_max dann
18             s_max ← s;
19             j_max ← j;
20     wenn j_max ≠ −1 und s_max > τ_s dann
21         Füge {e_i, e_{j_max}} zur Liste von Paaren u_m hinzu;
```

Die projizierten Positionen von den berechneten Doppeledgels aus Abschniit 2.4.2 sollten wir hingegen nicht verwenden, weil Mittelpunkte von Feldlinien im Bild und von Feldlinien auf der Bodenebene voneinander abweichen.

Die Suche nach benachbarten Edgels kann jedoch im Bild erfolgen, sodass wir von den kombinierten Gradientenrichtungen der Doppeledgels aus Abschnitt 2.4.2 Gebrauch machen können. Wenn Doppeledgels benachbart sind, dann sind auch die zugehörigen Edgelpaare Nachbarn.

Doppeledgels die auf der selben Linie liegen müssen eine ähnliche Richtung aufweisen. Außerdem nehmen wir an, dass benachbarte Doppeledgels auf benachbarten Scanlinien liegen.

In Algorithmus 2.7 stellen wir eine Adaption der Suche nach Nachbarn im NaoTH-Framework [MSK+18] orientiert an den vertikalen Scanlinien vor. Da die Längen vertikaler Scanlinien variabel sind, ist Bestimmung von benachbarten Scanlinien abhängig von der y Koordinate im Bild. Nachdem wir alle Doppeledgels einer Scanlinie mit den Doppeledgels einer angrenzenden Scanlinie abgeglichen haben, kommen als Nachbarn auf den nächsten Scanlinien nur noch Doppeledgels mit höheren y-Werten in Frage. In Zeile 12 wird der minimale y-Wert aus dem unteren Ende der vorherigen Scanlinie berechnet.

In Zeile 16 wird die Ähnlichkeit s zwischen Doppeledgels geprüft. Im NaoTH Framework [MSK+18] ist eine Formel zur Berechnung von s implementiert. Dabei wird der Winkel von beiden Doppeledgels $(\boldsymbol{p_1}, \boldsymbol{d_1})$ und $(\boldsymbol{p_2}, \boldsymbol{d_2})$ mit der Normalen $\boldsymbol{n}$ einer gedachten Linie durch beide Doppeledgels verglichen, sodass

$$\boldsymbol{n} = \begin{bmatrix} v_y & -v_x \end{bmatrix}^T \quad \text{mit} \quad \boldsymbol{v} = \frac{\boldsymbol{p_2} - \boldsymbol{p_1}}{||\boldsymbol{p_2} - \boldsymbol{p_1}||} \tag{2.5.3}$$

Eine Ähnlichkeit $s \in [0, 1]$ wird über den Kosinuswinkel zwischen Normale und Doppeledgelrichtung berechnet.

$$s = 1 - \frac{|\boldsymbol{d_1} \cdot \boldsymbol{n}| + |\boldsymbol{d_2} \cdot \boldsymbol{n}|}{2} \tag{2.5.4}$$

Benachbarte Doppeledgels e_1 und e_2 sind ähnlich ($s = 1$), wenn ihre Richtungen $\boldsymbol{d_1}$ und $\boldsymbol{d_2}$ orthogonal zu $\boldsymbol{n}$ sind und unähnlich ($s = 0$), wenn $\boldsymbol{n}$, $\boldsymbol{d_1}$ und $\boldsymbol{d_2}$ parallel sind.

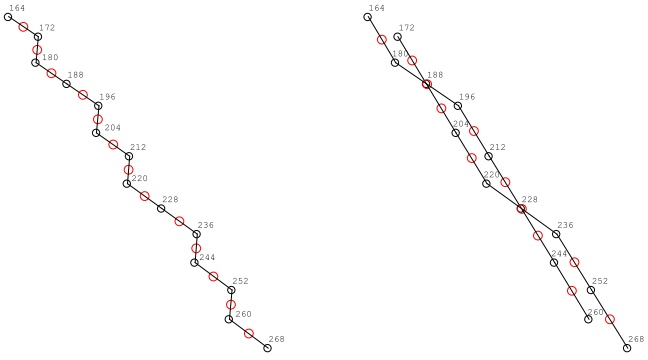
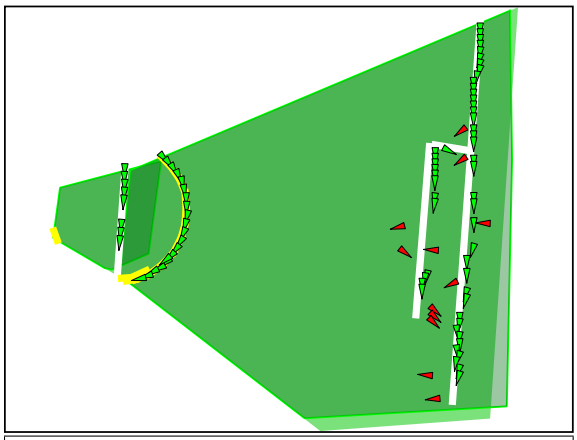


Abbildung 2.5.1: Dargestellt sind die mittleren Positionen von projizierten Edgelpaaren (schwarz). Die Zahlen bestimmen die x-Koordinate, welche mit der Position einer zugehörigen vertikalen Scanlinie übereinstimmt. Die extrahierten Linienkandidaten nach Gleichung 2.5.1 sind rot markiert. Schwarze Linien stellen deren Richtung dar. Wenn Scanlinien nah beieinander liegen und einzelne Pixel große Bereiche abdecken, kommt es zu Aliasing-Effekten. Die Richtung von Linienkandidaten wird durch die Projektion fehlerhaft bestimmt (links). Auf der rechten Seite ist festgelegt, dass Nachbarn im Bild mindestens 10 Pixel von einander entfernt sein müssen, sodass Fehler in der Richtungsberechnung abgeschwächt werden.

Wir suchen zu einem Doppeledgel e_1 einen Nachbarn e_2 mit maximaler Ähnlichkeit. Außerdem muss die Ähnlichkeit über einem Threshold τ_s liegen. Den Threshold sollten wir so einstellen, dass auch Nachbarn für Doppeledgels gefunden werden, die auf dem Mittelfeldkreis liegen.

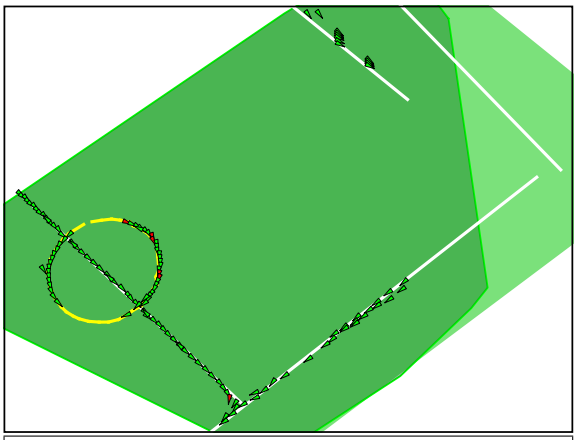
Weil Pixel in der Ferne große Bereiche abbilden (Abbildung 2.4.1), liegen projizierte Edgelpositionen mitunter nicht genau auf der Kante der Feldlinie. Dadurch kommt es zu Aliasing-Effekten und einer fehlerhaften Berechnung der Richtung von Linienkandidaten auf fernen Linien (Abbildung 2.5.1). Eine Möglichkeit, die Auswirkungen von Aliasing auf die Richtungsberechnung zu verringern, ist ein weiteres Kriterium in Zeile 7 von Algorithmus 2.7 einzuführen. Damit soll sichergestellt werden, dass für einen Doppeledgel e_i nur Nachbarn e_j von Scanlinien infrage kommen, die mindestens eine Anzahl von τ_{AA} Pixeln (zum Beispiel $\tau_{\text{AA}} = 10$) von der eigenen entfernt sind.

2.5.1 Evaluation der Linienkandidaten



AA	Anzahl	TP	FP	PRC
x	8587	6491	2096	0.76
30	8114	6871	1243	0.85

(a) RoboCup 2018



AA	Anzahl	TP	FP	PRC
x	9910	7595	2315	0.77
30	9357	8210	1147	0.88

(b) German Open 2018

Abbildung 2.5.2: Auswertung der Linienkandidaten auf den Evaluierungsdatensätzen. Die erkannten Linienkandidaten wurden als Pfeile für den Vergleich mit den projizierten annotierten Feldlinien (weiß) und Kreislinienzug (gelb) eingezeichnet. Pfeile zeigen auf die Position des Linienkandidaten und die Pfeilrichtung entspricht der Ausrichtung. True-Positives sind grün markiert, False-Positives hingegen rot. Außerdem ist das projizierte annotierte Feldpolygon dargestellt und mit dem projizierten detektierten Feldpolygon aus Abschnitt 2.3 überlagert. Die Tabellen zeigen die True-Positives (TP), False-Positives (FP) und die Precision (PCR) der erkannten Linienkandidaten von allen Bildern des jeweiligen Datensatzes.

Im Folgenden betrachten wir die Precision der Linienkandidaten mit den projizierten Annotationen der beiden Testdatensätze. Wir erwarten eine höhere Precision

bei Linienkandidaten, als bei den Doppeledgels, weil Edgels ohne Nachbarn gefiltert werden.

Für die Extraktion von Paaren haben wir jeweils einen Ähnlichkeitshreshold von $\tau_s = 0,8$ verwendet. Um die Auswirkungen von Aliasing auf die Precision zu betrachten, schalten wir für einen ersten Durchgang das Anti-Aliasing (AA) wie in Abbildung 2.5.1 aus. Anschließend setzen wir den Threshold für einen zweiten Durchgang auf $\tau_{\text{AA}} = 30$.

Ein Linienkandidat ist ein True-Positive (TP), wenn er nicht mehr als 200 mm von der nächstgelegenen annotierten Linie entfernt ist. Außerdem darf die Richtungsabweichung zwischen Linienkandidat und Linie nicht mehr als 20° betragen. Andernfalls handelt es sich um einen False-Positive (FP). Wir verwenden bewusst einen hohen Threshold, weil Fehler bei der Annotation durch die Projektion verstärkt werden.

In Abbildung 2.5.2 sind die Ergebnisse der Linienkandidatenevaluation zusammengefasst. Bei ausgeschaltetem AA hat sich die Precision bei Linienkandidaten gegenüber den Doppeledgels bei Datensatz a verbessert und bei Datensatz b leicht verschlechtert. Aliasing-Effekte treten eher in fernen Bereichen des Bildes auf. Eine Verschlechterung im German Open Datensatz ist damit zu erklären, dass hier öfter Bilder enthalten sind bei denen der Roboter vom Spielfeldrand auf das Feld schaut und abgebildete Linien weiter entfernt sind. Bei eingeschaltetem AA hat sich die Precision wie erwartet für beide Datensätze verbessert.

2.6 Linien- und Kreiserkennung

Wir können eine Lokalisierung bereits auf Basis der Menge von Linienkandidaten k_n implementieren. Dieser Ansatz hat jedoch einige Nachteile. Durch die große Menge an Linienkandidaten kann ein Update der Lokalisierung sehr rechenaufwendig sein. Die Robustheit leidet darunter, dass die Zuordnung von einzelnen Linienkandidaten zu den Feldlinien stark mehrdeutig ist. Des weiteren können Features mit hoher Aussagekraft, wie zum Beispiel der Kreis, nicht beachtet werden.

Um die Dimensionalität der Features weiter zu verringern, falsch detektierte Linienkandidaten zu filtern und signifikante Feldeigenschaften zu extrahieren, wollen wir im Folgenden eine Linien- und Kreiserkennung auf den Linienkandidaten realisieren.

Der von Fischler und Bolles [FB81] vorgestellte Random Sample Consensus (RANSAC) Algorithmus ist eine Standardmethode der Bildverarbeitung und ermöglicht die Suche eines Modells in verrauschten Daten. Für unsere Anwendung wollen wir Linien und Kreise modellieren. Ein wesentlicher Teil von RANSAC ist das Filtern der Linienkandidaten, die zu den Parametern des Modells passen (Inlier), von den Fehldetektionen (Outlier). Wir identifizieren Inlier, indem wir die Distanzen und Winkel von Linienkandidaten mit einem hypothetischen Modell abgleichen.

2.6.1 Linienmodell

Das Modell einer Linie $g(t)$ durch die Punkte $\boldsymbol{s}$ und $\boldsymbol{p}$ ist definiert durch

$$g(t) = t \cdot \boldsymbol{d} + \boldsymbol{s} \quad \text{mit} \quad \boldsymbol{d} = \frac{\boldsymbol{p} - \boldsymbol{s}}{||\boldsymbol{p} - \boldsymbol{s}||} \tag{2.6.1}$$

Ein Linienkandidat $k = (\boldsymbol{p}_k, \boldsymbol{d}_k)$ ist ein Inlier eines Linienmodells $g(t)$, wenn die folgenden zwei Kriterien erfüllt sind:

1. Sei $g(t_k)$ die orthogonale Projektion von $\boldsymbol{p}_k$ auf $g(t)$, sodass

   $$t_k = \boldsymbol{d} \cdot \boldsymbol{p}_k - \boldsymbol{d} \cdot \boldsymbol{s}. \tag{2.6.2}$$

   Für die Distanz zwischen $\boldsymbol{p}_k$ und $g(t_k)$ gilt

   $$||\boldsymbol{p}_k - g(t_k)|| + \epsilon_D = 0, \tag{2.6.3}$$

   wobei ϵ_D eine Zufallsvariable mit Mittelwert τ_D ist, die den Distanzfehler bei der Linienkandidatendetektion beschreibt. Wir führen τ_D als Threshold für den maximalen Distanzfehler ein. Für einen Linienkandidat k muss gelten, dass

   $$||\boldsymbol{p}_k - g(t_k)|| \leq \tau_D. \tag{2.6.4}$$

2. Für den Innenwinkel α zwischen $\boldsymbol{d}$ und $\boldsymbol{d}_k$ gilt:

   $$\alpha + \epsilon_\alpha = 0, \quad \text{mit} \quad \alpha = \arccos(|\boldsymbol{d}_k \cdot \boldsymbol{d}|), \tag{2.6.5}$$

   wobei ϵ_α eine Zufallsvariable mit Mittelwert τ_α ist, die den Winkelfehler repräsentiert. Wir führen τ_α wieder als Threshold für den maximalen Winkelfehler ein. Für einen Linienkandidat k muss gelten, dass

   $$\arccos(|\boldsymbol{d}_k \cdot \boldsymbol{d}|) \leq \tau_\alpha. \tag{2.6.6}$$

2.6.2 Kreismodell

Wir definieren das Modell eines Kreises durch seinen Mittelpunkt $\boldsymbol{m}$ und Radius r. In den Regeln [Com19] ist der Radius des Mittelfeldkreises auf 750 mm festgelegt.

Sei v die Wahrscheinlichkeit, dass ein Linienkandidat ein Outlier ist. Derpanis [Der10] zeigt, dass die Anzahl von benötigten Iterationen

$$N = \frac{\log(1-p)}{\log(1-(1-v)^m)} \tag{2.6.7}$$

betragen muss, sodass eine Menge von zufälligen m Proben mit einer Wahrscheinlichkeit von p keine Outlier enthält. Je mehr Proben für eine Hypothese benötigt werden, desto mehr Iterationen sind erforderlich.

Cai et. al. [CYW04] verwenden für die initiale Abschätzung eines Kreises 3 Punktproben um die Parameter m_x, m_y und r eines Kreises eindeutig zu bestimmen. Durch die Festlegung des Radius auf 750 mm und die Verwendung der Richtungsinformationen von Linienkandidaten, lässt sich ein Kreis schon mit 2 Kandidaten abschätzen. Seien k_1 und k_2 Linienkandidaten auf der Mittelfeldkreislinie, mit den Position $\boldsymbol{p}_1$ und $\boldsymbol{p}_2$ und orthogonalen Richtungen $\boldsymbol{n}_1$ und $\boldsymbol{n}_2$. Wir können den Mittelpunkt $\boldsymbol{m}$ als Schnittpunkt der Orthogonalen $g(t_1)$ aus $\boldsymbol{n}_1$ und $h(t_2)$ aus $\boldsymbol{n}_2$ berechnen, sodass

$$
\begin{aligned}
g(t_1) &= h(t_2) \\
\boldsymbol{p}_1 + t_1 \cdot \boldsymbol{n}_1 &= \boldsymbol{p}_2 + t_2 \cdot \boldsymbol{n}_2 \\
t_1 &= \frac{n_{2,x} \cdot (p_{2,y} - p_{1,y}) + n_{2,y} \cdot (p_{1,x} - p_{2,x})}{n_{2,x} \cdot n_{1,y} - n_{2,y} \cdot n_{1,x}} \\
\boldsymbol{m} &= g(t_1)
\end{aligned}
\tag{2.6.8}
$$

Der Nachteil ist, dass kleine Abweichungen in der Richtungsberechnung große Auswirkungen auf den Fehler von der Abschätzung des Mittelpunktes haben. Bei der Auswertung von Linienkandidaten in Abschnitt 2.5.1 haben wir festgestellt, dass die Richtungen durch Aliasing-Effekte fehlerhaft sein können.

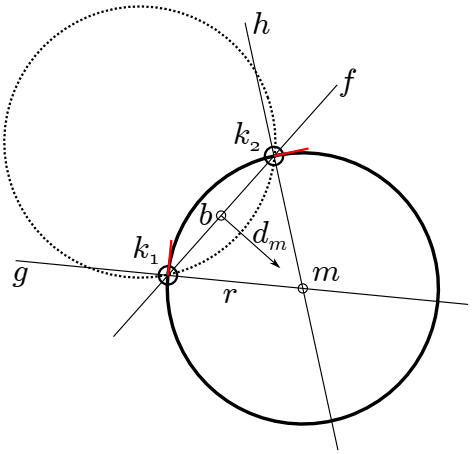


Abbildung 2.6.1: Abschätzung des Mittelpunktes $\boldsymbol{m}$ eines Kreises mit einem Radius r auf Basis von zwei Linienkandidaten k_1 und k_2. Der gestrichelte Kreis stellt die zweite mögliche Abschätzung ungeachtet der Richtungen von k_1 und k_2 dar. Der Kreismittelpunkt $\boldsymbol{m}$ hängt von dem Richtungsvektor $\boldsymbol{d_m}$ ab, sodass $\boldsymbol{m} = \boldsymbol{b} + l \cdot \boldsymbol{d_m}$, wobei l der Abstand zwischen $\boldsymbol{b}$ und $\boldsymbol{m}$ ist. Dabei gilt $l = \sqrt{r^2 - ||\boldsymbol{b} - \boldsymbol{k}_1||^2}$.

Die Abschätzung eines Kreises mit zwei Punkten kann genauer erfolgen, ist aber mehrdeutig (Abbildung 2.6.1). Wir können mithilfe der Richtungen entscheiden, welcher

der beiden Kreise berechnet werden soll. Die Modellierung des Kreises auf Basis von zwei Edgels mit einer Auflösung der Mehrdeutigkeit ist in Algorithmus 2.8 angegeben.

Algorithmus 2.8 : Abschätzung des Mittelkreises aus zwei Linienkandidaten

Eingabe : Linienkandidat $k_1 = \{\boldsymbol{p}_1, \boldsymbol{d}_1\}$, Linienkandidat $k_2 = \{\boldsymbol{p}_2, \boldsymbol{d}_2\}$, Kreisradius r
Ausgabe : Kreismittelpunkt $\boldsymbol{m}$

`# Berechne Linien orthogonal zu den Linienkandidatenrichtungen`

1 $g(t_1) = t_1 \cdot \begin{bmatrix} d_{1,y} & -d_{1,x} \end{bmatrix}^T + \boldsymbol{p_1}$;

2 $h(t_2) = t_2 \cdot \begin{bmatrix} d_{2,y} & -d_{2,x} \end{bmatrix}^T + \boldsymbol{p_2}$;

3 $\boldsymbol{m'} \leftarrow$ Schnittpunkt von $g(t_1)$ und $h(t_2)$ nach Gleichung 2.6.9;

`# Berechne Linie` $f(t_3)$ `durch` k_1 `und` k_2

4 $\boldsymbol{d_f} = \frac{\boldsymbol{p}_2 - \boldsymbol{p}_1}{||\boldsymbol{p}_2 - \boldsymbol{p}_1||}$;

5 $f(t_3) = t_3 \cdot \boldsymbol{d_f} + \boldsymbol{p}_1$;

`# Berechne orthogonale Projektion von` $\boldsymbol{m'}$ `auf` $f(t)$ `nach Gleichung 2.6.2`

6 $\boldsymbol{p}_{m'} = f(\boldsymbol{d_f} \cdot \boldsymbol{p}_2 - \boldsymbol{d_f} \cdot \boldsymbol{p}_1)$;

`# Berechne Richtungsvektor zum Kreismittelpunkt`

7 $\boldsymbol{d_m} = \frac{\boldsymbol{m'} - \boldsymbol{p}_{m'}}{||\boldsymbol{m'} - \boldsymbol{p}_{m'}||}$;

`# Punkt zwischen` $\boldsymbol{p}_1$ `und` $\boldsymbol{p}_2$

8 $\boldsymbol{b} = \frac{\boldsymbol{p}_1 + \boldsymbol{p}_2}{2}$;

`# Berechne den Abstand von` $\boldsymbol{b}$ `zum Kreismittelpunkt`

9 $l = \sqrt{r^2 - \left(\frac{||\boldsymbol{p}_2 - \boldsymbol{p}_1||}{2}\right)^2}$;

10 $\boldsymbol{m} = \boldsymbol{b} + l \cdot \boldsymbol{d_m}$;

In Zeile 7 wird die Richtung $\boldsymbol{d_m}$ zum Kreismittelpunkt durch die orthogonalen Projektion der Abschätzung $\boldsymbol{m'}$ nach Gleichung 2.6.9 bestimmt. Anschließend können wir die Länge l von $\boldsymbol{d_m}$, aus dem gegebenen Radius r und den halbierten Abstand zwischen k_1 und k_2 berechnen, damit wir in Zeile 10 den Kreismittelpunkt $\boldsymbol{m}$ erhalten. Die Modellierung des Kreises ist erfolgreich, wenn die Linienkandidaten k_1 und k_2 Inlier des Modells sind.

Ein Linienkandidat $k = (\boldsymbol{p}_k, \boldsymbol{d}_k)$ ist Inlier eines Kreismodells wenn die folgenden zwei Kriterien erfüllt sind:

1. Die Distanz zum Kreismittelpunkt ist gleich dem Radius, sodass

$$||\boldsymbol{m} - \boldsymbol{p}_k|| + \epsilon_r = r, \tag{2.6.9}$$

wobei ϵ_r eine Zufallsvariable mit Mittelwert τ_r ist, die den Fehler bei der Linienkandidatendetektion beschreibt. Wir führen τ_r als Threshold für die maximale Abweichung vom Radius ein, sodass

$$|r - ||\boldsymbol{m} - \boldsymbol{p}_k||| \leq \tau_r \tag{2.6.10}$$

2. Sei $\boldsymbol{d}_t$ die Richtung der Kreistangente an der Position des Linienkandidaten, sodass

$$\boldsymbol{d}_t = \begin{bmatrix} -n_y, n_x \end{bmatrix}^T \quad \text{mit} \quad \boldsymbol{n} = \frac{\boldsymbol{p}_k - \boldsymbol{m}}{||\boldsymbol{p}_k - \boldsymbol{m}||} \tag{2.6.11}$$

Für den Winkel β zwischen Linienkandidatenrichtung $\boldsymbol{d}_k$ und Kreistangentenrichtung $\boldsymbol{d}_t$ gilt

$$\beta + \epsilon_\beta = 0, \quad \text{mit} \quad \beta = \arccos(|\boldsymbol{d}_k \cdot \boldsymbol{d}_t|), \tag{2.6.12}$$

wobei die Zufallsvariable ϵ_β mit Mittelwert τ_β den Winkelfehler in der Richtungsabschätzung des Linienkandidaten beschreibt. Wieder führen wir τ_β als Threshold ein, sodass für einen Inlier gelten muss:

$$\arccos(|\boldsymbol{d}_k \cdot \boldsymbol{d}_t|) \leq \tau_\beta. \tag{2.6.13}$$

2.6.3 Modellfilterung

Zur Modellierung einer Linie und eines Kreises sind zwei Linienkandidaten notwendig. Beide Proben müssen Inlier des daraus erstellten Modells sein und damit die Kriterien aus Abschnitt 2.6.1 bzw. Abschnitt 2.6.2 erfüllen.

Als nächstes sammeln wir nach denselben Kriterien die Inlier des Modells und akkumulieren den Distanzfehler auf. Wenn die Anzahl von Inliern N nach einem Threshold τ_N ausreichend ist, betrachten wir das Modell als geeignete Lösung. Diese Schritte werden für eine festgelegte Anzahl von τ_I Iterationen wiederholt, sodass bessere Modelle gefunden werden können. Wir ersetzen eine Lösung, wenn sie über mehr Inlier verfügt oder bei gleicher Anzahl einen geringeren Distanzfehler produziert.

Bei den Feldlinien handelt es sich um Strecken. Sei $g(t)$ ein valides Linienmodell. Damit wir den Anfangspunkt $\boldsymbol{p_0}$ und den Endpunkt $\boldsymbol{p_1}$ von $g(t)$ bestimmen können, projizieren wir die Inlier nach Gleichung 2.6.2 auf die Linie und Berechnen t_{min} und t_{max}, sodass

$$\boldsymbol{p_0} = g(t_{min}) \quad \text{und} \quad \boldsymbol{p_1} = g(t_{max}). \tag{2.6.14}$$

Bei der Erkennung des Mittelfeldkreises kann es sein, dass Kreise in Linienkandidaten gefunden werden, die eigentlich zu den Feldlinien gehören (Abbildung 2.6.2). Eine Möglichkeit zur Validierung eines Kreises, ist die Abschätzung des Radius anhand der gefundenen Inlier. Wir nehmen an, dass die Abschätzung eines Radius r' von Inliern, die zu Feldlinien gehören, mit hoher Wahrscheinlichkeit zu lang, oder zu kurz ist. Die Ableitung von Methoden zur Abschätzung von Kreisen übersteigt den Umfang dieser Arbeit. Gander et. al. [GGS94] stellt Methoden für das Annähern von Kreisen an eine

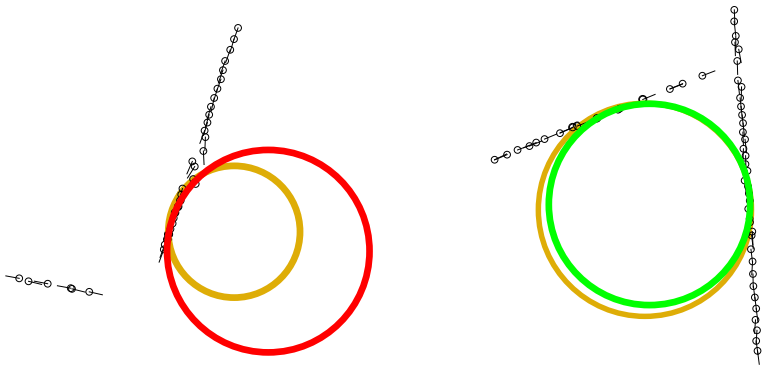


Abbildung 2.6.2: Mittelfeldkreise die fälschlicherweise in Feldlinien erkannt wurden. Die Linienkandidaten sind durch schwarze Punkte gekennzeichnet und Richtungen durch schwarze Linien dargestellt. Der gelbe Kreis repräsentiert die Abschätzung des Mittelpunktes und Radius durch die Kreisinlier. Bei dem linken Beispiel konnte die fehlerhafte Detektion (rot) durch die zu große Radiusabweichung erkannt werden. Bei dem rechten Beispiel wurde der Kreis in zwei sich kreuzenden Feldlinien detektiert (grün). Der Fehler konnte nicht erkannt werden.

Menge von Punkten vor. Hier verwenden wir einen Algorithmus, der bereits im NaoTH-Framework [MSK+18] implementiert ist. Dabei werden die Richtungsinformationen der Inlier vernachlässigt.

Sei r' der resultierende Radius bei einer Abschätzung eines Kreises anhand der Kreisinlier. Das Kreismodell ist valide, wenn für die Abweichung zwischen Mittelfeldkreisradius und r' gilt, dass

$$|750 - r'| \leq \tau_{\delta_r}, \tag{2.6.15}$$

wobei τ_{δ_r} ein Threshold für die maximale Abweichung ist.

Ein Großteil der Linienkandidaten liegen auf den Feldlinien. Bevor wir nach dem Kreis suchen, können wir die meisten Punkte durch die Erkennung der Feldlinien filtern. Nachdem wir eine Feldlinie identifiziert haben, wenden wir den RANSAC Algorithmus erneut auf den Outliern des validen Modells an, bis keine weiteren Feldlinien mehr gefunden werden. Auf den übrig gebliebenen Linienkandidaten suchen wir nach dem Kreismodell.

Durch diese Reihenfolge der Suche können wir die meisten Situationen wie in Abbildung 2.6.2 vermeiden, weil die Linienkandidaten schon durch die Linienerkennung gefiltert werden. Andererseits ist es möglich, dass Feldlinien in Linienkandidaten gefunden werden, die eigentlich zum Mittelfeldkreis gehören. Beobachtungen zeigen, dass es sich dabei aber eher um kurze Linien handelt.

Um zu kurze Linien zu filtern führen wie einen Threshold $\tau_{\delta_{min}}$ ein. Sei $\boldsymbol{s}$ Anfangspunkt

und $\boldsymbol{e}$ Endpunkt einer detektierten Feldlinie. Für die Länge muss gelten, dass

$$||\boldsymbol{e} - \boldsymbol{s}|| > \tau_{\delta_{min}}. \tag{2.6.16}$$

2.6.4 Auswertung der Linien- und Kreiserkennung

τ_I	100
τ_N	8
$\tau_{\delta_{min}}$	100 mm
τ_D	100 mm
τ_α	8°

(a) Linienerkennung

τ_I	50
τ_N	7
τ_{δ_r}	200 mm
τ_r	100 mm
τ_β	8°

(b) Kreiserkennung

Tabelle 2.6.1: Parameter der Linien und Kreiserkennung. τ_I gibt die Anzahl von Iterationen der RANSAC-Methode an, τ_N die minimale Anzahl von Inliern, $\tau_{\delta_{min}}$ die Mindestlänge von erkannten Feldlinien, τ_D den maximalen Abstand von Linieninliern, τ_α die maximale Winkelabweichung von Linieninliern, τ_{δ_r} die maximale Radiusabweichung von abgeschätzten Kreisen zum Mittelfeldkreis, τ_r die maximale Abweichung zwischen Inlierdistanz zum Mittelpunkt und Radius und τ_β die maximale Abweichung zwischen Inlierrichtung und Kreistangente.

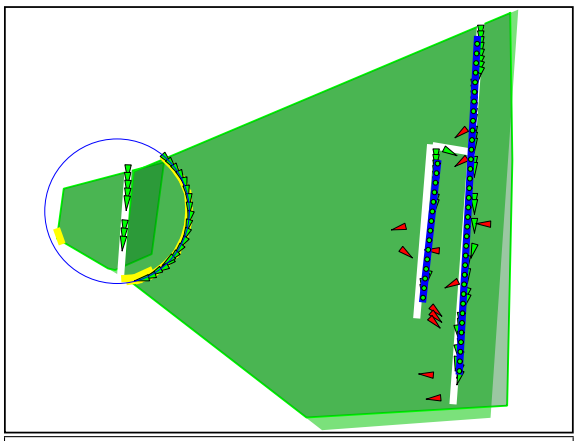
In diesem Abschnitt wenden wir die vorgestellte Mittelfeldkreis- und Linienerkennung auf den Evaluierungsdatensätzen an. Die verwendeten Parameter sind in Tabelle 2.6.1 zusammengefasst.

Wir haben die Anzahl an Iterationen τ_I für die Kreiserkennung reduziert, weil die meisten Outlier durch die vorangegangene Linienerkennung gefiltert werden. Dabei nehmen wir an, dass die Wahrscheinlichkeit bei der Kreiserkennung höher ist, dass eine Menge von Proben nur aus Inliern besteht. Die anderen Parameter wurden durch Beobachtungen festgestellt, sodass False-Positives möglichst unwahrscheinlich sind.

Um True-Postives und False-Positives in den erkannten Linien zu identifizieren, haben wir alle 100 mm auf den Linien eine gerichtete Linienprobe erzeugt. Die Richtung einer Linienprobe entspricht der Ausrichtung der zugehörigen Linie. Linienproben können wir genauso wie bei der Evaluierung der Linienkandidaten in Abschnitt 2.5.1 mit den projizierten Annotationen vergleichen, wenn wir nur die annotierten Feldlinien verwenden.

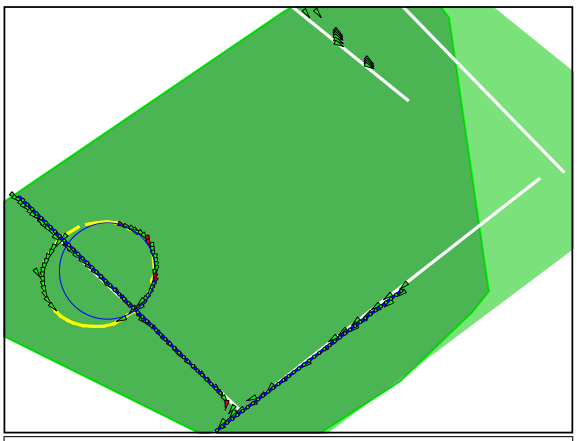
Eine erkannte Feldlinie ist ein True-Postive, wenn mindestens die Hälfte der zugehörigen Linienproben True-Postives sind, ansonsten handelt es sich um einen False-Positive.

Für die Evaluation des Kreises haben wir die projizierten Punkte aus den annotierten Kreislinienzügen verwendet. Die Punkte wurden durch einen erkannten Kreis bestätigt, wenn der Unterschied zwischen Abstand zum Mittelpunkt und Kreisradius weniger als 100 mm beträgt.



	Anzahl	TP	FP	PRC	Recall
Linien-proben	4407.90 ± 52.99	4108.80 ± 43.47	299.10 ± 24.57	0.93 ± 0.01	
Linien	201.00 ± 2.19	188.70 ± 1.73	12.30 ± 0.90	0.94 ± 0.00	0.44 ± 0.00
Kreise	91.20 ± 0.87	91.20 ± 0.87	0.00 ± 0.00	1.00 ± 0.00	0.77 ± 0.01

(a) RoboCup 2018



	Anzahl	TP	FP	PRC	Recall
Linien-proben	5983.60 ± 96.38	5767.40 ± 57.43	216.20 ± 56.61	0.96 ± 0.01	
Linien	210.00 ± 2.68	205.00 ± 2.05	5.00 ± 1.00	0.98 ± 0.00	0.46 ± 0.00
Kreise	93.10 ± 1.87	89.50 ± 1.69	3.60 ± 0.49	0.96 ± 0.00	0.73 ± 0.01

(b) German Open 2018

Abbildung 2.6.3: Linien- und Kreiserkennung auf den beiden Datensätzen. Die Bilder zeigen die Erkennung auf je einem Beispielframe aus dem jeweiligen Datensatz. Zusätzlich zu den projizierten annotierten Linien, Feldern und Linienkandidaten wie in Abbildung 2.5.2, sind die erkannten Feldlinien und Mittelfeldkreise in blau eingezeichnet. Die Punkte auf den erkannten Linien stellen die Linienproben dar. True-Positives (TP) sind grün markiert, False-Positives (FP) hingegen rot. Die Linienerkennung wurde jeweils 10 mal auf den gesamten Datensätzen ausgeführt. In der Tabelle sind jeweils die durchschnittlichen Resultate und darunter die Standardabweichung eingezeichnet.

Durch Fehler in der perspektivischen Projektion kann es sein, das der Kreis auf der Bodenebene verzerrt ist und nicht alle annotierten Punkte auf dem unverzerrten erkannten Kreis liegen. Wir legen fest, dass ein detektierter Mittelfeldkreis ein True-Positive ist, wenn wenigstens ein Drittel der annotierten Punkte bestätigt wird.

Durch die Verschiebung von Linienkandidaten bei Aliasing-Effekten kann es sein, dass Feldlinien doppelt erkannt werden. Um die Anzahl von True-Positives zu evaluieren berechnen wir für Linien und Kreise den Recall, sodass

$$Recall = \frac{N_{TP}}{N_{TP} + N_{FN}}, \tag{2.6.17}$$

wobei N_{FN} die Anzahl der nicht erkannten Features (False-Negatives) repräsentiert.

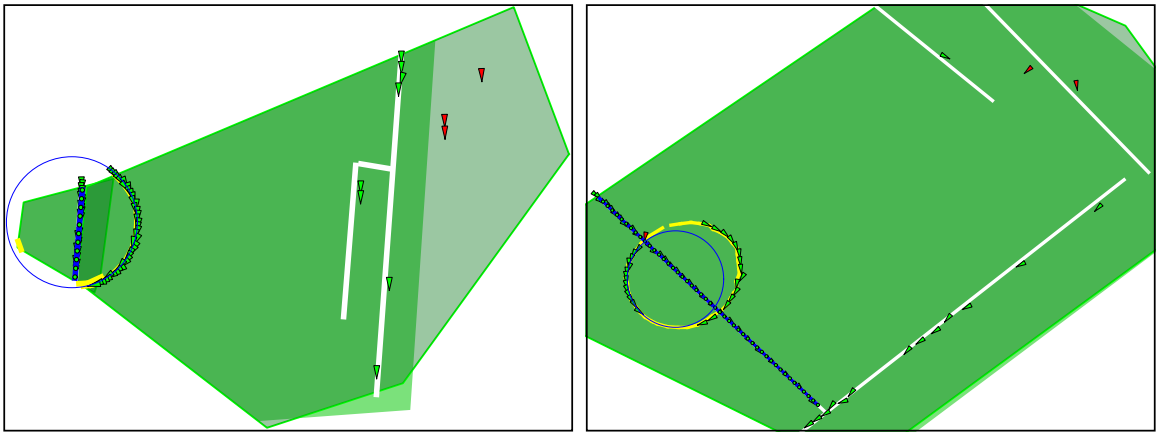
Für die Berechnung des Recalls von Feldlinien definieren wir N_{TP} als die Anzahl der bestätigten annotierten Feldlinien und N_{FN} als die Anzahl an annotierten Feldlinien für die keine Linie gefunden wurde.

Weil die RANSAC-Methode eine Zufallskomponente hat, haben wir die Erkennung auf den Datensätzen jeweils 10 mal ausgeführt und die Standardabweichung der Resultate berechnet. In Abbildung 2.6.3 sind die Erkennung auf Beispielbildern der Datensätze dargestellt und die Ergebnisse von allen Bildern in jeweils einer Tabelle zusammengefasst.

Wenn wir die Precision von Linienproben mit der von Linienkandidaten in Abschnitt 2.5.1 vergleichen, können wir eine deutliche Verbesserung feststellen. Auf beiden Datensätzen gab es nur wenige Fehlerkennungen bei Linien und Kreisen. In Datensatz 2.6.3a wurden alle gefundenen Kreise richtig erkannt.

Der Recall bei Linien ist dagegen eher gering. Beobachtungen der Einzelbilder zeigen, dass vor allem weiter entfernte Linien weniger häufig erkannt werden, weil der Fehler bei den berechneten Linienkandidaten größer ist.

Bei der Erkennung des Mittelfeldkreises konnte ein deutlich höherer Recall erzielt werden. Dabei ist der Recall beim German Open Datensatz im Vergleich zum RoboCup Datensatz etwas geringer. Bei Betrachtung der Einzelbilder stellen wir fest, dass Kreise hier eher in weiterer Entfernung auftreten, sodass der Linienkandidatenfehler größer und die Erkennung schwieriger ist.



(a) RoboCup 2018 (b) German Open 2018

Abbildung 2.6.4: Linien- und Kreisdetektion auf Linienkandidaten des NaoTH-Frameworks. Die Bilder zeigen die Erkennung auf je einem Beispielframe aus dem jeweiligen Datensatz. Zusätzlich zu den projizierten annotierten Annotationen, wurde das im NaoTH-Framework detektierte Feldpolygon dargestellt. Außerdem sind die erkannten Feldlinien und Mittelfeldkreise in blau eingezeichnet. In beiden Bildern wurden auf Feldlinien in fernen Bereichen durch die uniforme Abtastung im Bild nur wenige Linienkandidaten gefunden, sodass keine Feldlinien detektiert werden konnten.

2.7 Vergleich der Linienkandidatenerkennung dieser Arbeit mit dem NaoTH-Framework

	Anzahl	TP	FP	PRC
Doppel-edgels	13071	8943	4128	0.68
Kandidaten	8114	6871	1243	0.85

	Anzahl	TP	FP	PRC	Recall
Linien	201.00 ± 2.19	188.70 ± 1.73	12.30 ± 0.90	0.94 ± 0.00	0.44 ± 0.00
Kreise	91.20 ± 0.87	91.20 ± 0.87	0.00 ± 0.00	1.00 ± 0.00	0.77 ± 0.01

(a) RC18, adaptive Scanlinien

	Anzahl	TP	FP	PRC
Doppel-edgels	13633	7931	5702	0.58
Kandidaten	6942	6447	495	0.93

	Anzahl	TP	FP	PRC	Recall
Linien	166.40 ± 2.42	113.90 ± 1.51	52.50 ± 1.75	0.68 ± 0.01	0.27 ± 0.00
Kreise	96.70 ± 1.19	96.70 ± 1.19	0.00 ± 0.00	1.00 ± 0.00	0.82 ± 0.01

(b) RC18, NaoTH Scanlinien

	Anzahl	TP	FP	PRC
Doppel-edgels	14347	11467	2880	0.80
Kandidaten	9357	8210	1147	0.88

	Anzahl	TP	FP	PRC	Recall
Linien	210.00 ± 2.68	205.00 ± 2.05	5.00 ± 1.00	0.98 ± 0.00	0.46 ± 0.00
Kreise	93.10 ± 1.87	89.50 ± 1.69	3.60 ± 0.49	0.96 ± 0.00	0.73 ± 0.01

(c) GO18, adaptive Scanlinien

	Anzahl	TP	FP	PRC
Doppel-edgels	15101	9384	5717	0.62
Kandidaten	7799	7170	629	0.92

	Anzahl	TP	FP	PRC	Recall
Linien	184.60 ± 4.63	134.80 ± 2.23	49.80 ± 3.03	0.73 ± 0.01	0.30 ± 0.00
Kreise	84.20 ± 2.52	84.20 ± 2.52	0.00 ± 0.00	1.00 ± 0.00	0.67 ± 0.02

(d) GO18, NaoTH Scanlinien

Tabelle 2.7.1: Resultate der Linienkandidatenerkennung mit adaptiven Scanlinien im Vergleich mit uniformen Scanlinien des NaoTH-Frameworks auf den RoboCup 2018 (RC18) und German Open 2018 (GO18) Datensätzen. Darunter sind jeweils in Tabellen die Ergebnisse der Mittelfeldkreis- und Linienerkennung dieser Arbeit auf den resultierenden Linienkandidaten aufgelistet.

In diesem Abschnitt wollen wir die Linienkandidatendetektion dieser Arbeit, mit der bestehenden Erkennung im NaoTH-Framework [MSK+18] vergleichen und darauf jeweils die vorgestellte Mittelfeldkreis- und Feldlinienerkennung testen. In Tabelle 2.7.1 sind die jeweiligen Resultate zusammengefasst.

Für die Berechnung der True-Positives (TP) und False-Positives (FP) wurden jeweils die gleichen Parameter wie in den Abschnitten 2.4.3, 2.5.1 und 2.6.4 verwendet.

Bei der Erkennung von Doppeledgels im Bild konnte auf den adaptiven Scanlinien eine höhere Precision (PRC) erzielt werden. Die Abtastrate bei der Intervallsuche auf adaptiven Scanlinien ist vor allem in nahen Bereichen deutlich reduziert. Es ist wahrscheinlich, dass Ausreißer im Gradienten übersprungen werden.

Die Linienkandidaten im NaoTH-Framework weisen zwar eine höhere Precision auf, werden aber weniger in fernen Bereichen gefunden, wo die Auswirkungen von Aliasing-Effekten stärker sind. Stattdessen ist die Anzahl von Linienkandidaten besonders im Nahbereich groß. Das hat zur Folge, dass die Wahrscheinlichkeit in nahen Kreisen Linien zu erkennen größer wird. Dadurch leidet die Precision bei der Linienerkennung auf dem NaoTH-Framework.

In fernen Bereichen ist die Dichte von Linienkandidaten bei adaptiven Scanlinien wesentlich höher als im NaoTH-Framework (Abbildung 2.6.4). Dadurch können insgesamt eine größere Anzahl von Linien erkannt und der Recall gesteigert werden.

Bei den Mittelfeldkreisen im RoboCup 2018 Datensatz wurde ein größerer Recall bei der Kreiserkennung auf dem NaoTH-Framework erzielt. Wie wir bereits erwähnt haben, bilden hier die meisten Kamerabilder nahe Kreise ab. Durch das höhere Auftreten von Linienkandidaten in nahen Bereichen sind diese einfacher zu detektieren. Im German Open 2018 Datensatz ist der Recall bei Kreisen dagegen geringer. Hier sind mehr ferne Mittelfeldkreise in den Bildern enthalten, sodass die Kreiserkennung auf den Kandidaten der adaptiven Scanlinien vorteilhafter ist.

3 Lokalisierung im RoboCup

Die Selbstlokalisierung umfasst die Abschätzung der Position und Richtung des Roboters relativ zu seiner Umgebung und kann in zwei Kategorien eingeteilt werden: Globale Positionsbestimmung und Positionsverfolgung.

Bei der Positionsverfolgung gehen wir davon aus, den ungefähren Standort des Roboters zu kennen. Während sich der Roboter bewegt und neue Landmarken wahrnimmt, muss er seine Positionsabschätzung korrigieren und verfeinern. Die globale Positionsbestimmung befasst sich mit der initialen Abschätzung einer Position, ohne vorherige Anhaltspunkte.

Weil die Kameras einen limitierten Öffnungswinkel haben, kann nicht das ganze Feld zu einem Zeitpunkt überblickt werden. Wenige Feldeigenschaften sind eindeutig und die Wahrnehmung ist von Rauschen beeinflusst. Wenn wir Unsicherheiten nicht in Betracht ziehen, kann eine fehlerhafte Messung dazu führen, dass der Roboter seine Orientierung verliert.

Bei der Verwendung von direktionalen Wahrnehmungssystemen hat sich im RoboCup die probabilistische Lokalisierung durchgesetzt [BLR10]. Dabei wird die Unsicherheit über die Position auf dem Spielfeld durch eine Wahrscheinlichkeitsdichtefunktion beschrieben.

Zwei oft verwendete Methoden der probabilistischen Lokalisierung sind der Unscented Kalmanfilter, der zum Beispiel von den Teams Nao Devils [HSU+16], B-Human [RLB+17] und HULKS [AFG+18] eingesetzt wird und die Monte Carlo Lokalisierungsmethode auf Basis des Partikelfilters, die im NaoTH-Framework [MSK+18] implementiert ist.

Beide Methoden unterscheiden sich in der Repräsentation des Zustandsraums. Bei dem Kalmanfilter wird der Zustand über eine Gauß-Verteilung mit dem Mittelpunkt $\boldsymbol{m}$ als wahrscheinlichste Position des Roboters und der Kovarianzmatrix $\boldsymbol{\Sigma}$ als Unsicherheit repräsentiert. Um die Erholung von einer fehlerhaften Lokalisierung durch ein sogenanntes Sensor Resetting zu gewährleisten, wird oft auf einen Multi-Hypothesen Ansatz zurückgegriffen. Dabei können durch eine Menge von Gaußverteilungen mehrere Positionshypothesen verfolgt und neue Hypothesen bei der Detektion eines eindeutigen Features, wie zum Beispiel das eigene Tor, generiert werden.

Anstatt durch eine kontinuierliche Funktion, wird bei dem Partikelfilter der Zustand durch eine begrenzte Menge von Samples repräsentiert. Dadurch eignet sich der Partikelfilter, um jegliche Wahrscheinlichkeitsverteilungen zu approximieren. Die Methode kann von Natur aus mit Sensorresetting umgehen, sodass kein aufwendiges Pruning von Hypothesen notwendig ist. Allerdings kann ein Update von einer großen Menge an Samples sehr rechenaufwendig sein, sodass die Anzahl der verarbeitbaren Landmarken innerhalb eines Zeitschrittes begrenzt ist.

Der generelle Kalmanfilter wird für die Verfolgung und Verfeinerung einer bestehenden Zustandsabschätzung eingesetzt und ist daher für eine initiale Positionsbestimmung nicht geeignet. Prinzipiell kann der Partikelfilter durch eine anfänglich uniforme Verteilung der Samples das initiale Lokalisierungsproblem lösen. Dies erfordert aber eine großzügige Abdeckung des Zustandsraums, welche mit der vom Roboter verfügbaren Rechenleistung nicht realisiert werden kann. Die oben genannten Teams lösen das Problem durch detailliertes Vorwissen. So werden zum Beispiel bei dem Multi-Hypothesen Ansatz des Kalmanfilters übliche Startpositionen des Roboters abgedeckt.

Im folgenden Abschnitt betrachten wir Grundlagen der probabilistischen Lokalisierung und geben einen Überblick über die Funktionsweise des Partikelfilters. Anschließend beschäftigen wir uns mit der Modellierung der Unsicherheit bei der Erkennung von Feldlinien- und Mittelfeldkreisfeatures, um damit ein Update des im NaoTH Framework [MSK+18] implementierten Partikelfilters zu realisieren. Dabei sind Teile der vorgestellten Methodik auch für den Kalmanfilter anwendbar.

3.1 Grundlagen

Zustand: Der Zustand ist eine Ansammlung aller Eigenschaften des Roboters und seiner Umwelt, welche eine Auswirkung auf die Zukunft haben. Dazu gehören sowohl die Position und Orientierung des Roboters und seine eigene Bewegung im Raum, als auch Landmarken, die vom Roboter wahrgenommen werden. In der folgenden Betrachtung bezeichnen wir den Zustand zu einem Zeitpunkt t mit der Variable ω_t.

Belief: Die Überzeugung, oder der Belief, ist eine interne Repräsentation des Roboters, welche sein Wissen über den Zustand widerspiegelt. Sie kann durch eine Wahrscheinlichkeitsverteilung über den Zustandsraum repräsentiert werden, sodass

$$bel(x_t) = p(\omega_t | u_{1:t}, z_{1:t}) \tag{3.1.1}$$

[TBF05]. u_t beschreibt die Veränderung des Zustandes, typischerweise durch die Bewegung des Roboters, während z_t die Sensormessungen an Zeitpunkt t umfasst.

3.1.1 Monte-Carlo Lokalisierung

Die Anwendung des Partikelfilters im Kontext der Selbstlokalisierung wird Monte-Carlo Lokalisierung genannt. Weil der Partikelfilter effizient mit nichtlinearen Zustandsveränderungen und vielseitigen Wahrscheinlichkeitsverteilungen umgehen kann, findet er eine Anwendung in vielen Bereichen des RoboCup Szenarios [BLR10].

Die Aufgabe des Partikelfilters umfasst die rekursive Abschätzung eines diskreten Beliefs $p(\omega_t | u_{1:t}, z_{1:t})$ über dem Zustand ω_t.

Markow-Eigenschaft: Wir nehmen an, wenn wir einen Zustand ω_t zum Zeitpunkt t kennen, sind zukünftige und vorherige Daten unabhängig voneinander [TBF05], sodass

$$p(\omega_t|\omega_{0:t-1}, z_{1:t}, u_{1:t}) = p(\omega_t|\omega_{t-1}, z_t, u_t) \tag{3.1.2}$$

In diesem Fall nennen wir den Zustand ω_{t-1} komplett, weil er alle Informationen über vergangene Messungen $u_{1:t-1}$ und Bewegungen $z_{1:t-1}$ des Roboters enthält.

Bei einem Partikelfilter wird der Belief $bel(\omega_t)$ durch eine Menge von Partikeln Ψ_t approximiert. Jeder Partikel $\psi \in \Psi_t$ stellt eine Hypothese über den Zustand ω_t dar. Im Kontext der Lokalisierung ist ψ definiert durch die Position $\boldsymbol{p}$, einer Ausrichtung θ und einem Gewicht w.

Der Ablauf des Partikelfilters lässt sich durch die folgenden Schritte zusammenfassen:

1. Beim Bewegungsupdate wird jeder Partikel anhand der Veränderung des Zustands u_t verschoben.

   Wie sich ein Roboter zwischen zwei Zeitpunkten bewegt hat, kann über die Odometrie beschrieben werden. Sie gibt die Winkel- und Positionsveränderung über die Zeit zu einer gedachten Anfangsposition an. Im NaoTH Framework [MSK+18] wird die Odometrie durch die Laufengine aus den ausgeführten Bewegungen der Gelenke und den Lagesensoren des Roboters berechnet.

2. Im Sensorupdate wird allen Partikeln anhand eines Sensormodells ein Gewicht zugeteilt. Es gibt an, wie plausibel der Zustand ausgehend von der Stichprobe verglichen mit der Messung z_t erscheint. Sensormodelle für Mittelfeldkreis und Feldlinien werden wir im folgenden Abschnitt vorstellen.

3. Die Verteilung der Gewichte ergeben eine diskrete Wahrscheinlichkeitsverteilung über den Zustand des Systems. Im sogenannten Resampling-Schritt werden Partikel anhand dieser Verteilung neu geworfen, sodass sie mit höherer Wahrscheinlichkeit auf Bereiche fallen die ein hohes Gewicht haben. Orte mit größerer Gewichtung repräsentieren mit höherer Wahrscheinlichkeit den wahren Zustand der Umgebung. Nach einigen Iterationen erwarten wir, dass sich die Partikel um den wahren Zustand der Umgebung sammeln.

3.2 Sensormodelle

Mit einem Sensormodell wollen wir die Unsicherheit von Linien und Kreismessungen modellieren. Der Roboter verfügt über eine Karte des Spielfeldes, in der alle Feldlinien und der Mittelfeldkreis mit den zugehörigen Positionen in Weltkoordinaten eingezeichnet sind. Ausgehend von einer hypothetischen Roboterposition $(\boldsymbol{p}, \theta)$, die durch einen Partikel gegeben ist, wollen wir den Fehler zu erkannten Feldlinien und Mittelfeldkreis berechnen.

Bei der Erkennung von Feldlinien und Mittelfeldkreis im Bild aus Abschnitt 2.6 werden die Resultate in Roboterkoordinaten zurückgegeben. Durch die hypothetische Position können wir Punktlandmarken $\boldsymbol{l}$ transformieren und die zugehörige globale Position $\boldsymbol{l}'$ bestimmen, sodass

$$\boldsymbol{l}' = \boldsymbol{R_\theta}^T \cdot (\boldsymbol{l} - \boldsymbol{p}) \quad \text{mit} \quad \boldsymbol{R_\theta} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \tag{3.2.1}$$

wobei $\boldsymbol{R_\theta}$ die Rotationsmatrix für den Winkel θ bezeichnet.

Um den Fehler einer Messung berechnen zu können, müssen wir sie einer Landmarke in der Karte zuordnen können. Das Problem ist, dass Linien mehrdeutig sind. In diesem Fall schlagen Thrun et. al. [TBF05] vor, die wahrscheinlichste zugehörige Landmarke ausgehend von der hypothetischen Roboterposition zu berechnen.

Linien auf dem Spielfeld können in zwei Ausrichtungen anhand ihres Richtungsvektors $\boldsymbol{d}$ eingeteilt werden. Wenn $d_x > d_y$, dann verlaufen sie entlang der x-Achse, ansonsten entlang der y-Achse. Wir nehmen an, dass die Referenzlinie in der Karte die nächste Linie mit gleicher Achsenausrichtung ist. Abstände bestimmen wir zwischen dem Mittelpunkt $\boldsymbol{m}$ der gemessenen Linie und der orthogonalen Projektion $\boldsymbol{m}'$ von $\boldsymbol{m}$ auf die Referenzlinie.

Sei $g(t)$ die Gleichung der Referenzlinie mit Startpunkt $\boldsymbol{s}$ und Endpunkt $\boldsymbol{e}$, sodass

$$g(t) = t \cdot \boldsymbol{d} + \boldsymbol{s} \quad \text{mit} \quad \boldsymbol{d} = \frac{\boldsymbol{e} - \boldsymbol{s}}{||\boldsymbol{e} - \boldsymbol{s}||} \tag{3.2.2}$$

Weil es sich bei Feldlinien um Strecken handelt definieren wir $t \in [0, ||\boldsymbol{e} - \boldsymbol{s}||]$. Für die orthogonale Projektion von $\boldsymbol{m}$ auf $g(t)$ gilt

$$\boldsymbol{m}' = g(t_m) \quad \text{mit} \quad t_m = \boldsymbol{d} \cdot \boldsymbol{m} - \boldsymbol{d} \cdot \boldsymbol{s}. \tag{3.2.3}$$

Wieder limitieren wir $t_m \in [0, ||\boldsymbol{e} - \boldsymbol{s}||]$, sodass überstehende Punkte auf den Anfangspunkt bzw. Endpunkt projiziert werden.

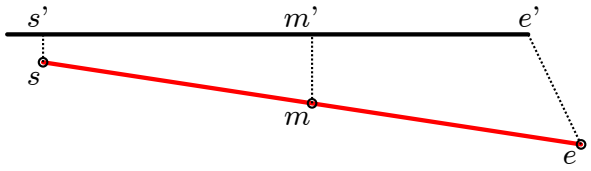


Abbildung 3.2.1: Projektion des Anfangs-, Mittel- und Endpunkt einer Linienmessung (rot) auf die Referenzlinie (schwarz).

Bei dem Sensormodell für Linien wollen wir den Abstands-, die Richtungsfehler beachten, aber keinen Fehler modellieren, wenn die Linie zu kurz ist. Durch den limitierten Öffnungswinkel der Kamera können wir oft nur Teile einer Linie erkennen. Dafür

konvertieren wir den Start-, Mittel- und Endpunkt der gemessenen Linie in 3 Punktlandmarken dessen Referenzpunkte die jeweiligen Projektionen auf die Referenzlinie ergeben (Abbildung 3.2.1).

Im NaoTH-Framework ist ein Algorithmus zur Modellierung der Unsicherheit einer Punktlandmarke implementiert. Dabei vergleichen wir die erwarte Position einer Punktlandmarke mit der gemessenen Position aus der Sicht des Roboters.

Referenzpunkte in Roboterkoordinaten $\boldsymbol{p}_R$ können mit der Umkehrung von Gleichung 3.2.1 aus der zugehörigen Position in Weltkoordinaten $\boldsymbol{p}_W$ berechnet werden, sodass

$$\boldsymbol{p}_R = \boldsymbol{R}_\theta \cdot \boldsymbol{p}_W + \boldsymbol{p}. \tag{3.2.4}$$

Algorithmus 3.1 : Punktlandmarkenmodell im NaoTH-Framework [MSK+18]

Eingabe : Relative Punktlandmarke (δ, γ), Relative Referenzlandmarke (δ', γ'), Standardabweichung σ_δ, Standardabweichung σ_γ, Kamerahöhe C_z

Ausgabe : Wahrscheinlichkeit p der Landmarkenmessung

`# Winkel zwischen` z`-Achse in Kamerakoordinaten und Punktlandmarke`

1 $\alpha = \arctan2(\delta, C_z)$;

`# Winkel zwischen` z`-Achse in Kamerakoordinaten und Referenzlandmarke`

2 $\alpha' = \arctan2(\delta', C_z)$;

3 $p = \exp\left\{-\frac{1}{2}\frac{(\alpha'-\alpha)^2}{\sigma_\delta^2}\right\} \cdot \exp\left\{-\frac{1}{2}\frac{(\gamma'-\gamma)^2}{\sigma_\gamma^2}\right\}$;

Relative Positionen auf der Bodenebene aus der Sicht des Roboters können eindeutig durch den Abstand δ und Winkel γ zum Roboter bestimmt werden, sodass

$$\delta = ||\boldsymbol{q}|| \quad \text{und} \quad \gamma = \arctan2(q_y, q_x), \tag{3.2.5}$$

wobei $\boldsymbol{q}$ die Position der Punktlandmarke in Roboterkoordinaten ist.

In Algorithmus 3.1 ist die Modellierung der Unsicherheit einer Punktlandmarke durch die Distanz- und Winkelabweichung zwischen Messung und Referenz wie im NaoTH-Framework [MSK+18] angegeben. Dabei modellieren wir den Fehler zwischen gemessenen Wert a und erwarteten Wert a' durch eine Gaußverteilung mit dem Mittelpunkt 0 und der Standardabweichung σ, die den erwartenden durchschnittlichen Fehler repräsentiert, sodass

$$p = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2}\frac{(a'-a)^2}{\sigma^2}\right\}. \tag{3.2.6}$$

Der Faktor vor der Exponentialfunktion ist für alle Messungen konstant, sodass wir ihn für eine Gewichtung von Partikeln vernachlässigen können.

In Zeile 1 und 2 werden die Winkel zwischen z-Achse der Kamera und Punktlandmarke, sowie zwischen z-Achse und Referenzlandmarke berechnet. Wenn wir die relativen Abstände über diese Winkel vergleichen, werden Fehler von weiter entfernten Landmarken weniger stark gewichtet.

Aus den Wahrscheinlichkeiten der Punktlandmarken p_s, p_m und p_e einer Feldlinienmessung l, lässt sich die Unsicherheit p_l berechnen als

$$p_l = p_s \cdot p_m \cdot p_e. \tag{3.2.7}$$

Für die Modellierung der Kreismessung k können wir den Kreismittelpunkt in eine Punktlandmarke konvertieren und mit der Position des Mittelfeldkreises in der Karte vergleichen. So können wir mit Algorithmus 3.1 die Wahrscheinlichkeit p_k der Kreismessung berechnen.

Die Gewichtung eines Partikels w anhand einer Anzahl N von Linienmessungen l_i und Kreismessung k ergibt dann schließlich

$$w = p_k \cdot \prod_{1}^{N} p_{l_i}. \tag{3.2.8}$$

4 Zusammenfassung

In dieser Arbeit haben wir Methoden zur Erkennung von Mittelfeldkreis und Feldlinien auf dem RoboCup Spielfeld vorgestellt, damit sie als Landmarken für die Lokalisierung eingesetzt werden können.

Dabei ist eine robuste Methode zur Erkennung des Spielfeldes implementiert worden, mit der eine Region of Interest für die Kantenerkennung berechnet werden kann. In den Datensätzen konnten wir insgesamt eine gute Erkennung feststellen und den Algorithmus erfolgreich im RoboCup 2019 auf den Robotern anwenden. Eine genauere Auswertung von Spezialfällen muss in einer zukünftigen Betrachtung noch erfolgen.

Im Anschluss wurde eine Kantenerkennung implementiert, die eine gleichmäßigere Abtastung des Bildes in Bezug zur Bodenebene ermöglicht. Damit können wir besonders in entfernteren Bereichen des Robotersichtfeldes eine größere Anzahl von Linienkandidaten auf der Bodenebene erkennen.

Um Linienkandidaten für die nachfolgende Feldlinien- und Mittelfeldkreiserkennung zu erzeugen, haben wir die Paarsuche für Edgels im NaoTH-Framework an die neue Kantenerkennung angepasst. Außerdem wurde ein Ansatz zur Reduzierung der Auswirkungen von auftretenden Aliasing-Effekten vorgestellt.

Anschließend haben wir die RANSAC-Methode implementiert, um Feldlinien und Mittelfeldkreis in den Linienkandidaten zu identifizieren. Dabei wurde das Auftreten von möglichen Fehldetektionen betrachtet und Lösungsansätze vorgestellt.

Für jeden Objekterkennungsschritt haben wir die Resultate anhand der Bilddatensätze evaluiert, die für diese Arbeit erstellt wurden. Darüber hinaus wurde die Linienkandidatendetektion dieser Arbeit, mit der bestehenden Erkennung im NaoTH-Framework verglichen und darauf jeweils die vorgestellte Mittelfeldkreis- und Feldlinienerkennung getestet.

Bei der Auswertung haben die vorgestellten Methoden vielversprechende Ergebnisse geliefert. Durch die Erhöhung der Abstände zwischen Scanpunkten auf der Bodenebene unter Beachtung der Feldlinienbreite, konnten wir die Wahrscheinlichkeit der Fehlerkennungen bei Kanten aufgrund von Ausreißern im Gradienten verringern.

Obwohl sich die Precision bei der Erzeugung von Linienkandidaten verringert hat, ist durch eine gleichmäßigere Verteilung auf der Bodenebene die Erkennung von Feldlinien auch in fernen Bereichen möglich geworden. Gleichzeitig hat sich die Precision der Linienerkennung verbessert. Feldlinien werden seltener in nahen Mittelfeldkreisen erkannt, weil die Anzahl von möglichen Linienkandidaten in unteren Bildbereichen verringert wurde.

Schließlich haben wir eine Modellierung des Messfehlers vorgestellt, wodurch wir Kreis- und Linienfeatures als Landmarken für die bestehende Lokalisierung im Partikelfilter des NaoTH-Frameworks verwenden können.

Die erkannten Linien und Kreise auf der Linienkandidatendetektion des NaoTH-Frameworks wurden erfolgreich als Landmarken in vergangenen RoboCup Veranstaltungen eingesetzt. Dabei hat die Erweiterung der Landmarkenerkennung die Robustheit der Positionsbestimmung des Roboters deutlich verbessert. In der Zukunft müssen wir die Beobachtungen in einer weiterführenden Betrachtung experimentell bestätigen.

4.1 Ausblick

Bei der Evaluation der Objekterkennung haben wir uns vor allem mit der Identifikation von Fehlerkennungen beschäftigt. Dabei nehmen wir an, dass der Partikelfilter durch die Modellierung des Messfehlers robust gegenüber kleineren Abweichungen ist.

Durch die begrenzte Anzahl von Partikeln aufgrund einer mangelnden Rechenleistung des Roboters, werden mitunter nicht alle möglichen Positionen durch eine Hypothese erfasst und die Markow-Eigenschaft verletzt (Abschnitt 3.1.1). Deswegen können auch kleine Abweichungen einen Einfluss auf die Robustheit der Lokalisierung haben.

Vor allem die initiale Lokalisierung erfordert eine großzügige Abdeckung des Zustandsraums. Im NaoTH-Framework umgehen wir dieses Problem durch detailliertes Vorwissen über die üblichen Startpositionen des Roboters.

Während der Roboter steht, muss kein Bewegungsupdate im Partikelfilter ausgeführt werden. Eine Möglichkeit zur weiteren Reduktion von lokalen Maxima bei der Wahrscheinlichkeit von Positionen im Zustandsraum, kann ein Sensorupdate auf gesammelten Messungen sein. Dabei können wir Informationen über das Feld aus mehreren Blickwinkeln einfließen lassen.

Bei einem sogenannten Sensor-Resetting, können wir während des Spiels mögliche Positionen aus eindeutigen Konstellationen von Feldmerkmalen berechnen. Das Team B-Human [RLB+17] implementiert zum Beispiel eine Methode, bei der aus Kombinationen von Feldfeatures Positionshypothesen generieren werden.

Der Mittelfeldkreis kann mit der in dieser Arbeit vorgestellten Objekterkennungsmethode robust detektiert werden. Im RoboCup 2019 in Sydney haben wir im NaoTH-Framework eine Sensor-Resetting Methode implementiert, bei der durch die Kombination von erkannter Mittelfeldlinie und Mittelfeldkreis eine eindeutige Position bestimmt werden kann. Dies setzt voraus, dass dem Roboter bekannt ist, in welcher Spielfeldhälfte er sich befindet. Dadurch konnte die Position des Roboters während des Spiels korrigiert werden, wenn er versetzt oder gedreht war.

Eine weitere Möglichkeit zum Sensor-Resetting kann die Kombination von Feldlinienkreuzungen sein, welche über die erkannten Feldlinien detektiert werden können. Durch die Implementierung solcher Methoden kann die Robustheit der Lokalisierung im NaoTH-Framework in der Zukunft noch wesentlich gesteigert werden.

Literatur

[AFG+18] Darshana Adikari, Georg Felbinger, Pascal Gleske, Chris Kahlefendt, Yuria Konda, René Kost, Pascal Loth, Konrad Nölle, Lasse Peters, Nicolas Riebesel, Thomas Schattschneider, Maximilian Schmidt, Erik Schröder, Felix Warmuth, and Felix Wege. Hulks team research report 2018, 2018. `https://hulks.de/_files/TRR_2018.pdf`.

[And79] Alex M Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(5):216–219, 1979.

[BLR10] Armin Burchardt, Tim Laue, and Thomas Röfer. Optimizing particle filter parameters for self-localization. In *Robot Soccer World Cup*, pages 145–156. Springer, 2010.

[Can87] John Canny. A computational approach to edge detection. In *Readings in computer vision*, pages 184–203. Elsevier, 1987.

[Com19] RoboCup Technical Committee. Robocup standard platform league (nao) rule book. *https://spl.robocup.org/wp-content/uploads/downloads/Rules2019.pdf*, 2019. Visited on 05/19/2019.

[CYW04] Wenchao Cai, Qian Yu, and Hong Wang. A fast contour-based approach to circle and ellipse detection. In *Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No. 04EX788)*, volume 5, pages 4686–4690. IEEE, 2004.

[Der10] Konstantinos G Derpanis. Overview of the ransac algorithm. *Image Rochester NY*, 4(1):2–3, 2010.

[FB81] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[FP17] Miha Finžgar and Primož Podržaj. Machine-vision-based human-oriented mobile robots: A review. *Strojniski Vestnik/Journal of Mechanical Engineering*, 63(5), 2017.

[GGS94] Walter Gander, Gene H Golub, and Rolf Strebel. Least-squares fitting of circles and ellipses. *BIT Numerical Mathematics*, 34(4):558–578, 1994.

[HSU+16] M Hofmann, I Schwarz, O Urbann, F Rensen, A Larisch, A Moos, and J Hemmers. Nao devils dortmund team report 2016. *https://naodevils.de/*, 2016.

[HTW18] Nao-Team HTWK. Team research report, 2018. `http://robocup.imn.htwk-leipzig.de/documents/TRR_2018.pdf?lang=en`.

[HVR13] Alexander Härtl, Ubbo Visser, and Thomas Röfer. Robust and efficient object recognition for a humanoid soccer robot. In *Robot Soccer World Cup*, pages 396–407. Springer, 2013.

[JHL03] Matthias Jüngel, Jan Hoffmann, and Martin Lötzsch. A real-time auto-adjusting vision system for robotic soccer. In *Robot Soccer World Cup*, pages 214–225. Springer, 2003.

[KAK+95] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. 1995.

[LDHB+09] Tim Laue, Thijs Jeffry De Haas, Armin Burchardt, Colin Graf, Thomas Röfer, Alexander Härtl, and Andrik Rieskamp. Efficient and reliable sensor models for humanoid soccer robot self-localization. In *Proceedings of the Fourth Workshop on Humanoid Soccer Robots in conjunction with the*, pages 22–29, 2009.

[MGFCGV+14] Jesus Martinez-Gomez, Antonio Fernandez-Caballero, Ismael Garcia-Varea, Luis Rodriguez, and Cristina Romero-Gonzalez. A taxonomy of vision systems for ground mobile robots. *International Journal of Advanced Robotic Systems*, 11(7):111, 2014.

[MSK+18] Heinrich Mellmann, Benjamin Schlotter, Steffen Kaden, Philipp Strobel, Thomas Krause, Etienne Couque-Castelnovo, Claas-Norman Ritter, Tobias Hübner, and Schahin Tofangchi. Berlin united - nao team humboldt team report 2018. techreport, Adaptive Systeme, Institut für Informatik, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany, 2018.

[ORB04] Umit Ozguner, Keith A Redmill, and Alberto Broggi. Team terramax and the darpa grand challenge: a general overview. In *Intelligent Vehicles Symposium, Parma, Italy*, 2004.

[Pap13] Panagiotis Papadakis. Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Engineering Applications of Artificial Intelligence*, 26(4):1373–1385, 2013.

[Ras08] Christopher Rasmussen. Roadcompass: following rural roads with vision+ ladar using vanishing point tracking. *Autonomous Robots*, 25(3):205–229, 2008.

[Rei11] Thomas Reinhardt. Kalibrierungsfreie bildverarbeitungsalgorithmen zur echtzeitfähigen objekterkennung im roboterfußball. Master's thesis, 2011.

[RLB+17] Thomas Röfer, Tim Laue, Yannick Bülter, Daniel Krause, Jonas Kuball, Andre Mühlenbrock, Bernd Poppinga, Markus Prinzler, Lukas Post, Enno Roehrig, René Schröder, and Felix Thielke. B-Human team report and code release 2017, 2017. Only available

online: http://www.b-human.de/downloads/publications/2017/coderelease2017.pdf.

[RLH+18] Thomas Röfer, Tim Laue, Arne Hasselbring, Jannik Heyen, Bernd Poppinga, Philip Reichenberg, Enno Roehrig, and Felix Thielke. B-Human team report and code release 2018, 2018. Only available online: http://www.b-human.de/downloads/publications/2018/CodeRelease2018.pdf.

[SAD+14] Davide Scaramuzza, Michael C Achtelik, Lefteris Doitsidis, Fraundorfer Friedrich, Elias Kosmatopoulos, Agostino Martinelli, Markus W Achtelik, Margarita Chli, Savvas Chatzichristofis, Laurent Kneip, et al. Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in gps-denied environments. *IEEE Robotics & Automation Magazine*, 21(3):26–40, 2014.

[Sch05] Oliver Schreer. *Stereoanalyse und Bildsynthese*. Springer-Verlag, 2005.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[Vid19] VideoLAN. Yuv, 2019. https://wiki.videolan.org/YUV/#YUV_4:2:2_.28I422.2FJ422.29, zuletzt besucht am 26.07.2019.

[VJ+01] Paul Viola, Michael Jones, et al. Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, 1:511–518, 2001.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 26. August 2019 .....