Technische Universität Hamburg

Vision Systems

Prof. Dr.-Ing. R.-R. Grigat

Detektion von Feldmerkmalen auf dem NAO-Robotiksystem in der RoboCup *Standard Platform League*

Projektarbeit

Pascal Loth

10. Oktober 2018

Eidesstattliche Erklärung

Ich, Pascal Loth, geboren am 14.09.1991 in Hamburg, versichere hiermit an Eides statt, dass ich diese von mir bei der Technischen Universität Hamburg (TUHH) vorgelegte Projektarbeit selbstständig verfasst habe. Ich habe ausschließlich die angegeben Quellen und Hilfsmittel benutzt.

Hamburg, Pascal Loth

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Kapitel 1

Einleitung

Diese Arbeit ist im Rahmen der Forschung des RoboCup Standard Platform League (SPL) Teams HULKs an der Technischen Universität Hamburg (TUHH) entstanden. Das folgende Kapitel geht kurz auf den Zusammenhang zum RoboCup und auf die verwendete Hardware Plattform das NAO-Robotiksystem ein. Darauffolgend wird in Abschnitt 1.3 auf den Hintergrund dieser Arbeit eingegangen und eine Abgrenzung zu verwandten Themen geschaffen, bevor es zur Zielsetzung dieser schriftlichen Darlegung übergeht.

1.1 RoboCup

Der RoboCup hat die Bestrebung die Forschung an künstlicher Intelligenz und Robotik zu fördern. Erzielt wird dies durch das Bereitstellen eines Standardproblems - zumeist das Fußballspiel. Es gilt, dieses unter Einsatz von Automaten, im Rahmen von Wettkämpfen, erfolgreich zu bewerkstelligen. Insbesondere Bereiche der Multi-Agenten-Forschung und des maschinellen Sehens kommen dabei zum tragen (OSAWA u. a.). Jährlich stattfindende Weltmeisterschaften, sowie vereinzelte lokale Wettbewerbe, prägen den Wettstreit und die damit verbundene Weiterentwicklung. Derweil formen unterschiedliche Ligen mit diversen Forschungsschwerpunkten den RoboCup. Das Spektrum umfasst die reine Simulation (ROBOCUP FEDERATION, 2018)(SKINNER & RAMCHURN, 2010), den Einsatz von heteronomen Bergungsmaschinen (SHEH u. a., 2016), bis hin zu autonomen Industrie- und Haushaltsrobotern (NIEMUELLER u. a., 2015)(WISSPEINTNER u. a., 2009).

1.2 Die SPL und das NAO-Robotiksystem

Das NAO-Robotiksystem - kurz NAO - kommt zum Einsatz in der RoboCup SPL. Die Liga zeichnet sich dadurch aus, dass die Teams auf gleicher Hardwaregrundlage gegeneinander antreten. Veränderungen an der Hardware sind nicht erlaubt. Demzufolge liegt der Forschungsschwerpunkt dieser Liga auf der Softwareentwicklung.

Abb. 1.1: *Umgangener Torwart der HULKs beim RoboCup 2018 in Montréal, Kanada. (*NAO-TEAM HTWK*, 2018)*

Der NAO ist ein 58 cm hoher humanoider Roboter und wird von der *Softbank Robotics Corp.* entwickelt (SOFTBANK ROBOTICS EUROPE, 2018b). Ausgestattet ist der Roboter mit einem Intel Atom Z530 Prozessor mit einer Taktrate von 1,6 GHz und 1 GB RAM. Im Kopf sind zwei nach vorne gerichtete Kameras verbaut, die allerdings kein Stereosehen erlauben, da sich die Sichtfelder, wie in Abbildung 1.2 zu erkennen ist, nur marginal überschneiden. Unterstützt werden Auflösungen bis 1280×960 Pixeln bei 29-30 Bildern pro Sekunde (SOFTBANK ROBOTICS EUROPE, 2017). Die softwareseitige Rahmenstruktur der HULKs verwendet eine Auflösung von 640×480 Pixeln und arbeitet mit dem YCbCr-Farbmodell. Details können in (LOTH, 2015) ermittelt werden. Zusätzlich verbaut sind vier gerichtete Mikrofone, Lautsprecher, sowie Inertial-, Ultraschall- und Drucksensorik. Seit kurzer Zeit ist eine potentere Version des NAO-

Robotiksystems verfügbar, welche aufgrund fehlender Schnittstellen in dieser Arbeit keine Verwendung findet (SOFTBANK ROBOTICS EUROPE, 2018a).

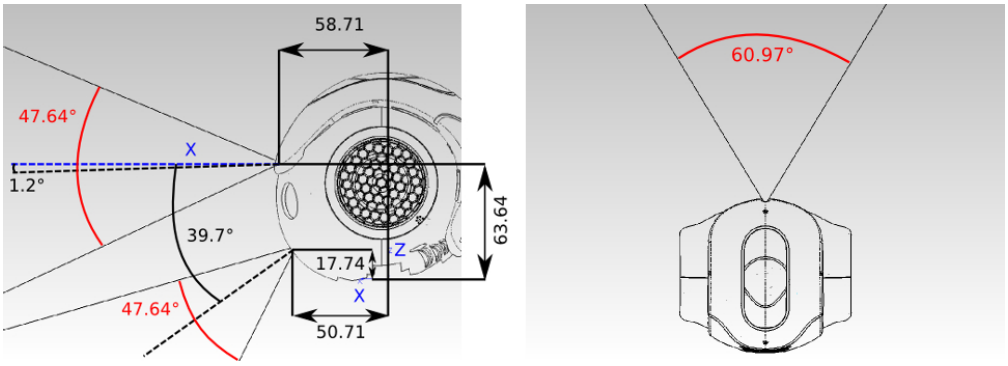


Abb. 1.2: *Position und Sichtfeld der im NAO verbauten Kameras. Längenangaben in Millimetern (*ALDEBARAN ROBOTICS*, 2018).*

1.3 Motivation

In allen Bereichen in denen autonome Systeme zum Einsatz kommen, hat das System mit der Umwelt in Interaktion zu treten. Die Verarbeitung von Sensordaten ist Hauptbestandteil bei dem Aufbau eines internen abstrakten Weltmodells. So ist auch die zuverlässige Bestimmung der Position und Orientierung bei dem autonomen Fußballspiel unabdingbar, um dieses erfolgreich zu bewältigen. Erst mit gelungener Lokalisierung kann der Agent vorausdenken und Strategien entwickeln. Es ermöglicht außerdem das Passspiel, das Verwalten eines gemeinsamen Weltmodells, progressive Pfadplanung, sowie die Kommunikation der Position von einem selbst, anderen Mit- und Gegenspielern und dem Ball. Erst mit diesen Informationen kann eine Rollenverteilung und sinnvolles Stellungsspiel ermöglicht werden.

Die Kameras sind die einzige Möglichkeit die Markierungen auf dem Spielfeld wahrzunehmen. Eine anschließende Lokalisierung der Feldlinien, Mittelkreis und Strafstoßmarken auf dem Bild ist notwendig.

Die bisherigen Erfolge der HULKs wären keineswegs denkbar gewesen, wäre nicht eine Form einer Feldmarkierungserkennung vorhanden, doch beschränkt sich diese allein auf die Feldlinien. Zudem ist die vorhandene Implementierung sehr anfällig für verrauschte Eingangsbilder und hat eine Vielzahl falsch positiv Detektionen in dem Mittelkreis, anderen Robotern und Toren. In Mittelkreisabschnitten lassen sich kurze Liniensegmente finden. Die Roboter sorgen, aufgrund ihrer Farbe und teils ihrer Form, für wie Feldlinien aussehende Elemente. Tore haben weiße Pfosten und können kaum von echten Linien

unterschieden werden. Auch das weiße Tornetz stellt eine besondere Herausforderung dar. Diese Umstände sprechen für die Qualität bereits implementierter Filter. Alle auf dem Bild erkannten Merkmale werden in einem Unscented Kalman-Filter (UKF) verarbeitet (PETERS, 2018) und zur Positions- sowieso Ausrichtungsbestimmung verwendet.

Der Fokus dieser Arbeit liegt auf der Detektion der Feldlinien und Strafstoßmarken.

1.4 Verwandte Arbeiten

Die Extraktion von Linien aus Bildern stellt kein neues Problem im Bereich des maschinellen Sehens dar. Vielmehr ist es elementarer Bestandteil und Grundlage für weiterführende komplexere Probleme. Das vorgegebene Problem in der SPL verlangt aber insbesondere Echtzeitfähigkeit auf vergleichsweise schwacher Hardware. Standardwerkzeuge wie die Hough-Transformation oder robuste Kantendetektionen können in ihrer Reinform aufgrund ihrer Komplexität, insbesondere bei hochdimensionalen Eingangsvektoren und damit einhergehender Laufzeit, nicht angewendet werden (LOTH, 2015)(RIEBESEL, 2018). Verwandte Probleme finden sich bei der Spurerkennung im Bereich des autonomen Fahrens. Neben Spline basierten Ansätzen (WANG u. a., 2000) kommen hierbei zumeist aktive Konturen (KASS u. a., 1988) – auch Snakes genannt – zum Einsatz (WANG u. a., 2004). Die Verwendung solcher Methoden ist für die vorliegende Art von Problem aber nicht sinnvoll einsetzbar.

In der SPL existieren verschiedene Ansätze zur Liniendetektion. Das Team B-Human segmentiert ein unterabgetastetes Bild entlang von Scanlines. Die Mittelpunkte, der darauf befindlichen weißen eindimensionalen Regionen, bilden die Kandidaten für den Mittelkreis und die Linien. Es sind zwei unterschiedliche Methoden vorhanden um den Mittelkreis, sofern im Bild vorhanden, zu erkennen. Übrige Punkte werden zu Liniensegmenten zusammengefasst (RÖFER u. a., 2017, S. 69). Das Nao-Team HTWK verwendet auf ähnliche Weise entstandene eindimensionale Regionen, gruppiert hingegen aber jeweils die Start- und Endpunkte untereinander. Zudem verwenden sie die Gradientenrichtungen und Verbindungslinien, um über ein Ähnlichkeitsmaß Punktpaare innerhalb ihrer Gruppe zu erzeugen. Die Punktpaare ergeben kurze Liniensegmente, welche anhand einer Distanzfunktion zusammengefasst werden (REINHARDT, 2011, S. 70). Über Krümmungsanalyse wird außerdem der Mittelkreis erkannt, worauf aber nicht weiter eingegangen wird (REINHARDT, 2011, S. 80).

Beide Teams können am Spiel teilhabende Roboter zuverlässig erkennen. Wie in Abschnitt 1.3 kurz erläutert, stellen diese ein nicht vernachlässigbares Problem dar. Das Team B-Human zeigte erste Fortschritte bei der Detektion anderer Roboter bereits im Jahre 2010 (RÖFER u. a., 2010, S. 51) und das Nao-Team HTWK erkennt andere Roboter seit 2014 (NAO-TEAM HTWK, 2014, S. 8).

Für den Strafstoßpunkt kombiniert B-Human geeignete benachbarte Regionen zu zweidimensionalen Bereichen. Die Bereiche werden mithilfe einer Kantendetektion auf das Vorhandensein der Kontur einer Strafstoßmarke überprüft (RÖFER u. a., 2017, S. 72).

1.5 Zielsetzung

In dem Framework der HULKs (ADIKARI u. a., 2017) existieren bisher keine Methoden zur visuellen Detektion anderer Roboter. Jüngste Fortschritte bei der Mittelkreisdetektion (RIEBESEL, 2018) scheitern unglücklicherweise an der Vielzahl und Qualität der Kandidaten. So müssen die in dieser Arbeit vorgestellten Algorithmen mit entsprechenden Umständen umgehen können.

Ziel dieser Arbeit ist die Entwicklung einer echtzeitfähigen Detektion der Feldlinien und Strafstoßmarke. Ein besonderes Augenmerk wird dabei auf die Extraktion und Klassifikation der durch Linien entstehenden Kreuzungspunkte gelegt. Diese höherwertigen Feldmerkmale dienen der eindeutigeren Positions- und Ausrichtungsbestimmung in nachfolgenden Filtermethoden. Zudem soll die zu entwickelnde Liniendetektion der bereits im Framework der HULKs existierenden Methode gegenübergestellt und hinsichtlich der Detektionsqualität und Ausführungszeit verglichen werden.

1.6 Gliederung der Arbeit

Die Arbeit wird zuerst in Kapitel 2 auf die in den Algorithmen verwendeten Grundlagen eingehen. Kapitel 3 stellt die entwickelten Verfahren zur Detektion der Feldlinien und der Strafstoßmarke vor und gibt einen Überblick über die von den HULKs verwendete Liniendetektion. Das Kapitel geht zusätzlich auf die Behandlung von Fehldetektionen in Mittelkreisen und Robotern ein. Anschließend werden die Algorithmen in Kapitel 4 auf Spielsequenzen der IranOpen2018, GermanOpen2018 und des RoboCups2018 getestet und ausgewertet. Außerdem wird ein Vergleich mit der bisherigen Implementierung der HULKs vorgenommen und die Algorithmen hinsichtlich ihrer Ausführungszeit untersucht. Kapitel 5 wird zusammenfassend tätig und gibt einen Ausblick auf die notwendigen zukünftigen wissenschaftlichen Abhandlungen.

Kapitel 2

Grundlagen

Das folgende Kapitel geht kurz auf einige Verfahren ein, die bei dem Verständnis der in dieser Arbeit entwickelten Algorithmen unterstützen.

2.1 Sobel Operator

Der Sobel-Operator ist ein diskreter Faltungskern, der über die Faltungsoperation, je nach Ausrichtung, die partiellen Ableitungen in x oder y-Richtung berechnet. Der Operator findet meist Anwendung in der Kantendetektion zur Featureextraktion auf Bildern (CANNY, 1986). Der euklidische Betrag der beiden partiellen Ableitungen nähert den Gradienten auf einer Intensitätsverteilung an.

Der Operator ist wie folgt definiert:

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.1}$$

$$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.2}$$

Die Besonderheit mancher Filterkerne liegt in der Möglichkeit diese in Zeilen- und Spaltenvektor zu separieren. Es wird damit eine deutliche Reduktion an Lesezugriffen, Multiplikationen und Additionen erreicht (LOTH, 2015, S. 21). Gleichung (2.3) verdeutlicht dies am Beispiel des S_x.

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \tag{2.3}$$

Die Faltungsoperation von Bild I mit einem 3x3 Kern H ist an der Position (x,y) über Gleichung (2.4) definiert.

$$I^{'}(x,y) = \sum_{j=-1}^{1} \sum_{i=-1}^{1} I(x+i, y+j) H(i,j) \tag{2.4}$$

Bildrandbereiche erfordern spezielle Behandlung. Näheres kann in (LOTH, 2015, S. 7) nachgelesen werden.

2.2 Lineare Regression

Die lineare Regression gehört zu den statistischen Analyseverfahren. Das Modell beschreibt die Beziehung zwischen einer abhängigen und einer oder mehreren unabhängigen Variablen. Die Regressionsanalyse unterscheidet grundlegend zwischen zwei Modellen der linearen Regression. Das erste Modell ist die einfache lineare Regression, die die Abhängigkeit von y über eine unabhängige Variable x modelliert (Gleichung (2.5)). Das zweite Modell ist die multiple lineare Regression die entsprechend der Anzahl der unabhängigen Variablen gleichermaßen Regressionsparameter β_i besitzt (Gleichung (2.6)). Dabei ist m die Anzahl der Gleichungen in dem Gleichungssytem und n die Anzahl der unabhängigen Variablen. Der Fehler ist über e_i notiert.

$$y_i = \beta_0 + \beta_1 x_i + e_i, \quad i = 1, \ldots, m \tag{2.5}$$

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_n x_{in} + e_i, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n \tag{2.6}$$

2.2.1 Methode der kleinsten Quadrate

Die Methode der kleinsten Quadrate dient der Bestimmung der Modellparameter β_i. Die Anwendbarkeit des Schätzers ist nicht an die Anzahl der unabhängigen Variablen gebunden. Sind mehr Gleichungen m als Unbekannte n gegeben, so spricht man von einem überbestimmten System ($m > n$). Gleichung (2.7) stellt das Gleichungssytem in Matrixschreibweise dar.

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} \\ X_{21} & X_{22} & \cdots & X_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m1} & X_{m2} & \cdots & X_{mn} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_m \end{bmatrix} \tag{2.7}$$

Ein solches System hat im Normalfall keine Lösung, daher werden die unbekannten Koeffizienten β_i gesucht, welche die Summe der quadratischen Fehler der Kostenfunktion Gleichung (2.8) minimiert.

$$V(\beta) = \sum_{i=1}^{m} \varepsilon_i^2 = \sum_{i=1}^{m} \mid y_i - \sum_{j=1}^{n} X_{ij}\beta_j \mid^2 = \|y - X\beta\|^2 \tag{2.8}$$

Es wird folgendes Minimierungsproblem formuliert:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} V(\beta) \tag{2.9}$$

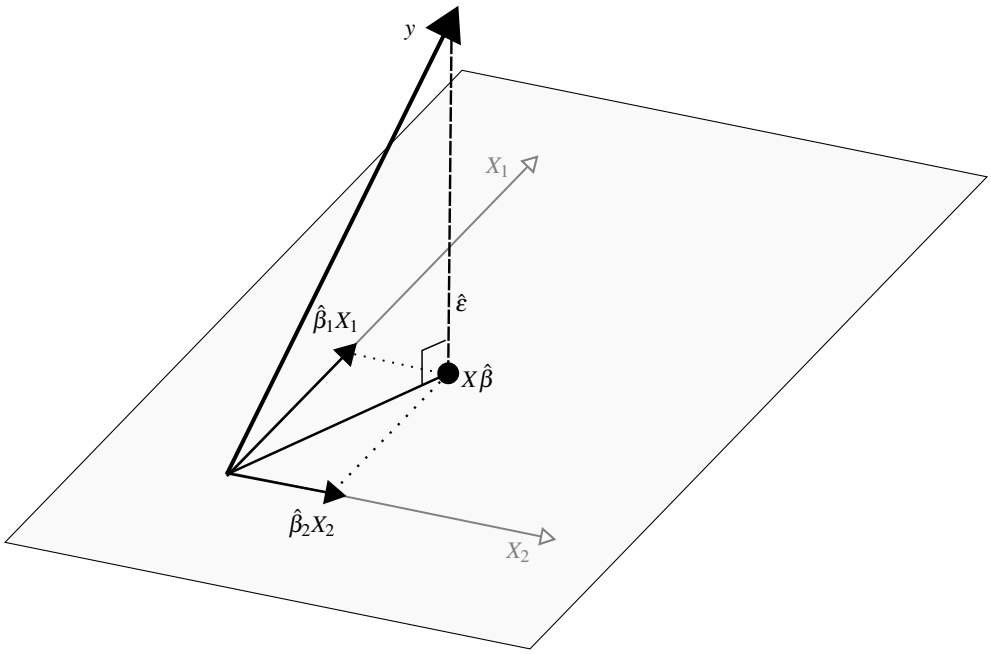


Abb. 2.1: *Visualisierung der Herleitung der Lösung des Minimierungsproblems.*

Verdeutlicht wird der Hintergrund in Abbildung 2.1. Die Spaltenvektoren X_n der Datenmatrix X stellen die Basis dar, aus der sich die Lösung als Linearkombination $X\hat{\beta}$

ergibt. Der Fehler bzw. in diesem Falle die Länge des Fehlervektors ist minimal, wenn der durch die Spalten von X aufgespannte Untervektorraum orthogonal auf $\hat{\varepsilon}$ steht. Dies ist in Gleichung (2.10) beschrieben.

$$X^T \hat{\varepsilon} = X^T (y - X\hat{\beta}) = 0 \tag{2.10}$$

Außerdem existiert eine eindeutige Lösung, wenn die n Spaltenvektoren von X linear unabhängig sind.

$$\begin{aligned} X^T y - X^T X\hat{\beta} &= 0 \\ X^T y &= X^T X\hat{\beta} \\ (X^T X)^{-1} X^T y &= \hat{\beta} \end{aligned}$$

2.2.2 Deming Regression

Die Deming Regression ist ein weiteres Regressionsmodell zur Bestimmung einer Ausgleichsgeraden. Minimiert werden dabei aber nicht die Abstände in x- bzw. y-Richtung, sondern die orthogonalen Abstände zwischen den Messpunkten und der Modellfunktion. In diesem Modell wird angenommen, dass sowohl x_i als auch y_i messfehlerbehaftet sind (ε_i und η_i). Außerdem geht man von bekannter Fehlervarianz aus. Das δ ist das Verhältnis der beiden Fehlervarianzen.

$$\begin{aligned} x_i &= x_i^* + \varepsilon_i \\ y_i &= y_i^* + \eta_i \end{aligned}$$

Die Kostenfunktion ist in Gleichung (2.11) dargestellt. Auch in diesem Fall, allerdings mit ihrer Varianz gewichtet, werden die Fehlerquadrate minimiert.

$$V(\beta) = \sum_{i=1}^{n} \left(\frac{\varepsilon_i^2}{\sigma_\varepsilon^2} + \frac{\eta_i^2}{\sigma_\eta^2} \right) = \frac{1}{\sigma_\varepsilon^2} \sum_{i=1}^{n} \left((y_i - \beta_0 - \beta_1 x_i^*)^2 + \delta (x_i - x_i^*)^2 \right) \tag{2.11}$$

Die Modellparameter können wie folgt gelöst werden (SAYLOR u. a., 2006):

$$\begin{aligned} \hat{\beta}_1 &= \frac{s_{yy} - \delta s_{xx} + \sqrt{(s_{yy} - \delta s_{xx}^2 + 4\delta s_{xy}^2}}{2 s_{xy}} \\ \hat{\beta}_0 &= \bar{y} - \hat{\beta}_1 \bar{x} \end{aligned}$$

wobei

$$\begin{aligned}
\bar{x} &= \frac{1}{n}\sum_{i=1}^{n} x_i \\
\bar{y} &= \frac{1}{n}\sum_{i=1}^{n} y_i \\
s_{xx} &= \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \bar{x})^2 \\
s_{yy} &= \frac{1}{n-1}\sum_{i=1}^{n} (y_i - \bar{y})^2 \\
s_{xy} &= \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})
\end{aligned}$$

Orthogonale Regression

Die Orthogonale Regression stellt einen Spezialfall der Deming Regression dar. Es wird von gleichen Fehlervarianzen ausgegangen, wodurch sich die Berechnung der Modellparameter leicht vereinfacht, da $\delta = 1$. Der Einsatz der orthogonalen Regression ist begründet, wenn sich die Messmethoden der Zufallsvariablen nicht unterscheiden, da in diesem Fall von etwa gleichen Fehlervarianzen ausgegangen werden kann.

2.3 Hough-Transformation

Die Hough-Transformation (HT) erlaubt die Detektion beliebig komplexer Strukturen auf Binärbildern (HOUGH, 1962)(BALLARD, 1981) und gehört damit zu den bekanntesten Methoden der Featureextraktion. Die grundlegende Funktionsweise basiert auf der Transformation eines jeden Punktes aus dem Binärbild in den Hough-Raum, welcher entsprechend der zu detektierenden Strukturen parametrisiert ist. Bei der Transformation wird jede Parametrisierung an dem Punkt angenommen und im Hough-Raum abgetragen.

Im Fall zu detektierender Linien, beläuft sich die Parametrisierung des Hough-Raumes auf den orthogonalen Abstand zu dem Bildursprung s, sowie den Winkel θ zu einer vorher definierten Bildachse. Jeder Punkt im Binärbild stellt eine Linienhypothese dar, somit werden alle möglichen Linien, die durch diesen Punkt verlaufen, in den Parameterraum transformiert und resultieren dort in einer Kurve. In der Anwendung sorgen

Punkte, die auf einer Linie liegen, für Häufungen in dem diskretisierten Hough-Raum. Darauf angewandte Schwellwerte entscheiden über im Bildbereich existierende Linien.

Geradengleichung in x, y:

$$Ax + By + C = 0 \tag{2.12}$$

Geradengleichung in s, θ:

$$s = x\cos\theta + y\sin\theta \tag{2.13}$$

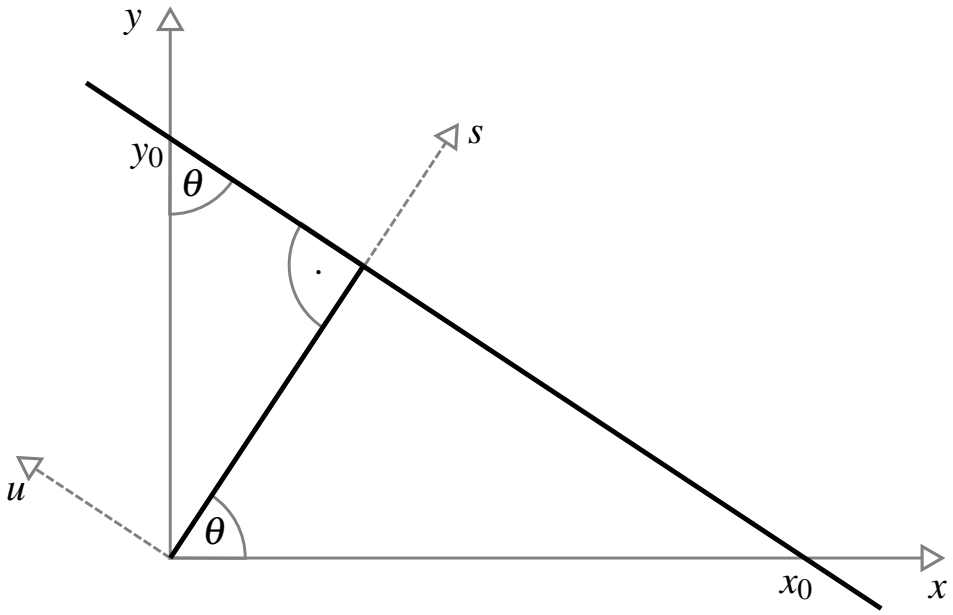


Abb. 2.2: *Visualisierung der Beziehung der Geradengleichungen zueinander.*

Die HT wandelt ein schwieriges Erkennungsproblem im Bildbereich in eine leichter lösbare lokale Höchstwertdetektion im Parameterraum um (ILLINGWORTH & KITTLER, 1988). Allerdings gilt die HT als sehr rechenintensiv (KALVIAINEN u. a., 1995), lässt sich aber aufgrund der Unabhängigkeit der einzelnen Transformationen gut parallelisieren (ILLINGWORTH & KITTLER, 1988).

2.3.1 Radon-Transformation

Die Radon-Transformation sei hier nur kurz erwähnt, da die HT als Spezialfall der Radon-Transformation in dieser Arbeit von größerem Interesse ist. In Gleichung (2.14) lässt sich der Zusammenhang erkennen (JAIN, 1989, S. 434). Die Radon-Transformation verwendet dieselbe Koordinatentransformation zur Integralbildung im rotierten Koordinatensystem. Die Radon-Transformation (Gleichung (2.18)) eines Punktes entspricht daher auch einer sinusförmigen Funktion.

$$s = \quad x\cos\theta + y\sin\theta \tag{2.14}$$
$$u = -x\sin\theta + y\cos\theta \tag{2.15}$$
$$x = \quad s\cos\theta - u\sin\theta \tag{2.16}$$
$$y = \quad s\sin\theta + u\cos\theta \tag{2.17}$$

$$g(s,\theta) = \int_{-\inf}^{\inf} f(s\cos\theta - u\sin\theta, s\sin\theta + u\cos\theta)du \tag{2.18}$$

2.3.2 Randomisierte Hough-Transformation

Auch die Randomisierte Hough-Transformation (RHT) dient der Detektion beliebiger Strukturen im Bildbereich. Der Algorithmus unterscheidet sich deutlich von der HT durch zufällige Abtastung des Bildbereichs, einer konvergierenden anstelle einer divergierenden Abbildung, sowie einem effizienteren Ansatz des zweidimensionalen Parameterraumes (XU & OJA, 2009).

Im Fall zu detektierender Linien werden zufällige Punktpaare aus dem Binärbild gewählt. Die Linie durch ein Punktpaar lässt sich, wie auch bei der HT, über den orthogonalen Abstand zu dem Bildursprung, sowie den Winkel zu einer vorher definierten Bildachse parametrisieren. Die konvergierende Abbildung äußert sich darin, dass das Parameterpaar nur einen Punkt im Parameterraum darstellt. Im Vergleich zu der HT, sorgt ein einzelner Punkt im Bildraum für eine Kurve im Parameterraum. Dies ermöglicht die Überführung des zweidimensionalen Akkumulators im Parameterraum in eine einfacher zu handhabende Listenstruktur. Ist ein Parametersatz bereits vorhanden, so werden die Parameter gemittelt und die Bewertung des Eintrages erhöht. Erreicht die Bewertung einen vorher festgelegten Schwellwert, werden dazugehörige Punkte aus dem Binärbild entfernt und es kann eine weitere Iteration durchgeführt werden (XU u. a., 1990).

Window Randomized Hough Transform

Eine Abwandlung der RHT ist die Window Randomized Hough Transform (WRHT). Die Kantenpunkte aus dem Binärbild bilden den Datensatz D. Die WRHT wählt einen zufälligen Punkt d_i aus D und berechnet die Regressionsgrade mit allen Punkten, in einem vorher definierten $m \times n$ großen Fenster, um d_i. Ist der Fehler der Ausgleichsgeraden innerhalb gewählter Schwellwerte, so wird die Gerade, entsprechend dem Verfahren der RHT, in den Akkumulator aufgenommen (KÄLVIÄINEN u. a., 1994, S. 4). Es folgen weitere Iterationen bis ein Akkumulatoreintrag die geforderte Bewertung aufweist.

Connective Randomized Hough Transform In einigen Anwendungsbereichen kann der Zusammenhang der Punkte d_i gefordert sein. Die Methode der WRHT kann dahingehend erweitert werden, indem von dem zufällig gewählten Punkt d_i eine gerichtete Suche ausgeht. Dabei werden nur die acht umliegenden Pixel in Betracht gezogen. Lassen sich ausreichend zusammenhängende Punkte finden, wird die Regressiongerade berechnet und wie in Abschnitt 2.3.2 weiter verfahren (KALVIAINEN u. a., 1995, S. 8).

2.4 Random Sample Consensus

Random sample consensus (RANSAC) ist ein Schätzer zur Bestimmung von Modellparametern. Der Algorithmus ist, entgegen Kleinster-Quadrate-Schätzern, äußerst robust gegenüber Ausreißern (FISCHLER & BOLLES, 1981). Das Verfahren besteht aus zwei Schritten. Der erste Schritt generiert eine Hypothese, welche im zweiten Schritt verifiziert wird (CHOI u. a., 1997).

Es werden zufällige Kanten aus dem Binärbild gewählt. Die Anzahl entspricht der minimal notwendigen Punkte, die das Modell beschreiben. Im Fall zu detektierender Linien werden somit zwei zufällige Punkte gewählt. Die Bewertung der Linie ergibt sich durch die Anzahl anderer Punkte, die eine vorher definierte maximale Distanz zu der Linie nicht überschreiten. Durch mehrfache Iterationen kann anschließend die beste Lösung gewählt werden (HARTLEY & ZISSERMAN, 2004, S. 117). Punkte, die zu weit von der besten Lösung entfernt sind, ergeben die Ausreißer. RANSAC sorgt damit für eine Klassifizierung, wodurch, im Kontrast zu der orthogonalen Regression (Abschnitt 2.2.2), nicht alle Messungen einen Einfluss auf die Modellparameter haben (RAGURAM u. a., 2008).

2.5 Polygone im Tangentialraum

Die Punkte $\{C_i\}_{i=0}^{n}$ eines Polygons C können wie folgt in einen Tangentialraum transformiert werden. Dabei sei l_i die Länge des Segments C_iC_{i+1} und $a_i = \angle(\overrightarrow{C_{i-1}C_i}, \overrightarrow{C_iC_{i+1}})$ (NGUYEN & DEBLED-RENNESSON, 2011, S. 5). Abhängig von der Position des Punktes C_{i+1} zu dem Segment $\overrightarrow{C_{i-1}C_i}$ ist der Winkel a_i mit unterschiedlichem Vorzeichen abzutragen. Abbildung 2.3 verdeutlicht das Vorgehen.

$$T_{02} = (0,0) \tag{2.19}$$

$$T_{i1} = (T_{(i-1)2}.x + l_{i-1}, T_{i-1)2}.y) \quad i = 1,2,\ldots,n \tag{2.20}$$

$$T_{i2} = (T_{i1}.x, T_{i1}.y + a_i) \quad i = 1,2,\ldots,n-1 \tag{2.21}$$

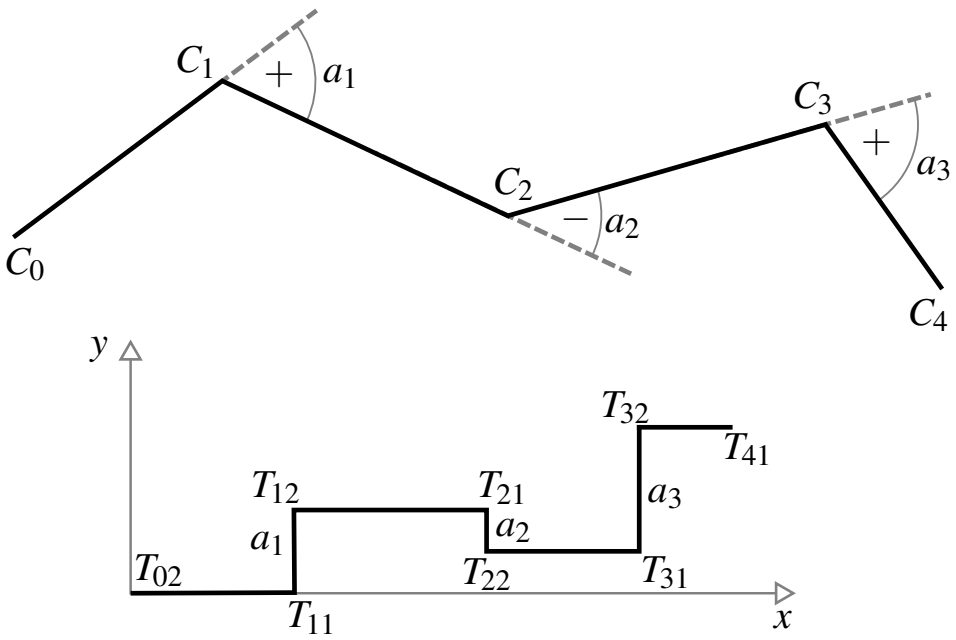


Abb. 2.3: *Visualisierung unterschiedlicher Polygonrepräsentationen. Die untere Grafik zeigt das obere Polygon im Tangentialraum.*

Kapitel 3

Algorithmen zur Detektion der Feldmerkmale

Das folgende Kapitel stellt die beiden entwickelten Algorithmen zur Detektion der Linien und der Strafstoßmarke vor. Die beiden Abschnitte bieten zu Beginn eine Übersicht in Form von Pseudocode, bevor auf einzelne Funktionsabschnitte genauer eingegangen wird. Darauf folgende Abschnitte behandeln aufgetretene Probleme und erläutern anhand dieser die Wahl verwendeter Methoden.

3.1 Linien und Kreuzungspunkte

Der entworfene Algorithmus zur Detektion von Linien und zur Klassifizierung der Kreuzungspunkte ist in Algorithmus 1 kurz zusammengefasst. Wie bereits in Abschnitt 1.5 erwähnt, werden im Framework der HULKs bisher keine Mittelkreise oder Roboter erkannt. Diese Umstände gestalten die Entwicklung einer Liniendetektion besonders schwierig, da falsch positiv Detektionen auf diesen Elementen für Schnittpunkte sorgen, die fälschlicherweise mit Ecken und anderen Arten von Kreuzungen der Spielfeldmarkierungen assoziiert werden können.

3.1.1 Bildsegmentierung

Um Rechenzeit für die Bearbeitung eines Kamerabildes zu reduzieren, werden im Framework der HULKs Regionen gleicher Helligkeit zusammengefasst. Es wird ein eindimensionaler symmetrischer Gradient berechnet, um Kanten entlang einer vertikalen Scanline zu detektieren (LOTH, 2015, S. 16)(FELBINGER, 2017, S. 15). Die Möglichkeit horizontale Kanten zu registrieren wird damit vollständig unterbunden. Vor einiger Zeit hat

Algorithmus 1 Übersicht Liniendetektion

```
1:  function LINEDETECTION(
            verticalScanlineSegments,
            horizontalScanlineSegments)
2:      verticalPoints ←FILTERSCANLINESEGMENTS(verticalScanlineSegments)
3:      horizontalPoints ←FILTERSCANLINESEGMENTS(horizontalScanlineSegments)
4:
5:      verticalSegments ←COMBINETOSEGMENTS(verticalPoints)
6:      horizontalSegments ←COMBINETOSEGMENTS(horizontalPoints)
7:
8:      segments ←{verticalSegments + horizontalSegments}
9:      (lineSegments, arcSegments) ←DIFFERENTIATE(segments)
10:
11:     houghLines ←HOUGHTRANSFORM(lineSegments)
12:     houghLines ←FILTERABSURDITY(houghLines)
13:
14:     projectedHoughLines ←PROJECTTOGROUNDCOORDINATES(
            houghLines)
15:     orthogonalLines ←ORTHOGONALCLUSTERING(projectedHoughLines)
16:
17:     intersections ←GETINTERSECTIONS(orthogonalLines)
18:     intersections ←REFINEINTERSECTIONS(intersections)
19:
20:     return lines, intersections, arcSegments
21: end function
```

man daher das Framework um zusätzliche horizontale Scanlines erweitert, welche in der Arbeit (FELBINGER, 2018, S. 21) Einsatz fanden und dort auch im Detail beschrieben sind.

Ein Folgemodul filtert, sofern die Spielfeldfarbe detektiert werden konnte, alle damit übereinstimmenden Regionen. Außerdem werden alle Regionen, die auf dem Bild über dem berechneten Horizont liegen, verworfen. Das Modul stellt die übrigen Regionen als vertikale und horizontale Scanlinesegmente zur Verfügung.

3.1.2 Auswahl der Punktkandidaten

Ziel ist es eine Punktwolke zu erstellen, die den Eingangsvektor der Liniendetektion darstellt. Daher müssen in diesem Schritt die vertikalen und horizontalen Scanlinesegmente auf ihre Verwendbarkeit hin überprüft werden. Es gilt nach Möglichkeit alle Scanli-

nesegmente, die nicht auf einer Feldmarkierung liegen, auszuschließen. Der Startpixel eines Segments muss daher als eine steigende Kante klassifiziert sein, respektive der letzte Pixel des Segments als fallende Kante. In diesem Fall ist die Annahme, dass ein Scanlinesegment auf einer Feldmarkierung, heller als die Umgebung aufgenommen wird. Zusätzlich wird eine parametrisierbare Grundhelligkeit erwartet. Entspricht das Segment den bisherigen Anforderungen wird der Gradient über den Sobel Operator (Abschnitt 2.1) an dem ersten und letzten Pixel des Segments berechnet. Die Gradienten müssen aufeinander zu zeigen. Abbildung 3.1 verdeutlicht die geforderten Eigenschaften.

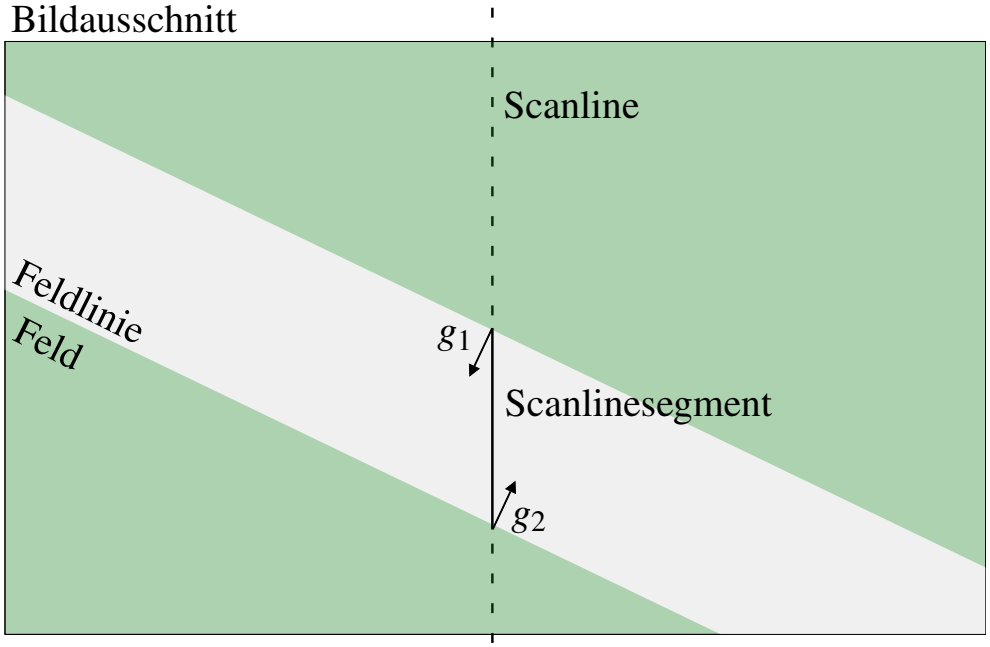


Abb. 3.1: *Visualisierung einer auf eine Feldlinie treffende Scanline, sowie das entstehende Scanlinesegment. An Start- und Endpunkt der Scanline werden die Gradienten g_1 und g_2 berechnet.*

Zu beachten sei hier, dass die Ergebnisse stark von denen aus einem Simulator abweichen können. Die HULKs verwenden SimRobot zur Simulation (LAUE u. a., 2005), den man bisher mit keinem Anti-Aliasing ausgestattet hat. Durch mehrfache Abtastung der Szene kommt es teilweise zu ungewünschten Effekten bei der Berechnung der Gradienten. Von dem Scanlinesegment wird der Mittelpunkt berechnet und der Punktwolke hinzugefügt. Dies entspricht implizit dem Mittel über die detektierte Position der Segmentkanten und wirkt sich rauschunterdrückend aus. Zusätzlich ist zu erwähnen, dass Anfangs, aufgrund der Berechnungskomplexität, die Bestrebung bestand ohne Kalku-

lation der Gradienten die Punktwolke in Linien- und Kreisbogensegmente zu zerlegen. Daher wurde in Algorithmus 2 zuerst darauf verzichtet. Allerdings zeigt Abschnitt 3.1.3, dass die Gradienten ein unvermeidbares Mittel zur Detektion von Kreisbögen sind. In den folgenden Kapiteln ist die exakte Ausrichten der Gradienten g_1 und g_2 zumeist nicht relevant. Von Interesse ist die Ausrichtung zwischen $[0,\pi]$, daher wird folglich nur noch auf den Gradienten referenziert, der wie folgt definiert wird:

$$g := \frac{g_1 - g_2}{2} \tag{3.1}$$

Algorithmus 2 Filterung der Segmente auf einer Scanline

```
1:  function FILTERSCANLINESEGMENTS(scanlineSegments)
2:      points ←{}
3:      for all scanlineSegments do
4:          if segment begin is not a rising edge or
              segment end is not a falling edge then
5:              Stop and evaluate next scanline segment.
6:          end if
7:          if segments center point is not bright enough then
8:              Stop and evaluate next scanline segment.
9:          end if
10:         if gradient vector at begin and end of the segment are not parallel or
              gradient vectors point in same direction then
11:             Stop and evaluate next scanline segment.
12:         end if
13:         Calculate the mean of the gradients from segment begin and end (0 to π).
14:         mid ←Segment center point
15:         points ←{points, mid}
16:     end for
17:     return points
18: end function
```

3.1.3 Zusammenfassung zu Segmenten

Die Idee dieser Liniendetektion basiert auf der Methodik der RHT. Neben der Transformationsvorschrift macht sich dieser Algorithmus auch den effizienten Parameterraum der RHT zunutze. Eine starke Einschränkung der HT und RHT ergibt sich aus der Tatsache, dass das Abstimmungsschema nicht berücksichtigt, dass Linien benachbarte aufeinanderfolgende Punkte sind. Linien haben nur eine endlich komplexe Struktur. Die

Nachbarschaftsbeziehung der Punkte im Binärbild bleibt bei beiden Algorithmen aber ungeachtet. Daher versucht diese Methode Punkte im vornherein zu Liniensegmenten zusammen zu fassen, welche im Folgeschritt, entsprechend der Hough-Transformation, im Parameterraum akkumuliert werden. Dadurch wird die Komplexität des Suchraumes stark reduziert. Dieser Ansatz weist Ähnlichkeiten zu der Connective Randomized Hough Transform (CRHT) auf (siehe Abschnitt 2.3.2). Allerdings sind in diesem Anwendungsfall, aufgrund der Abtastung durch die Scanlines, die Pixel nicht direkt benachbart. Folgt man benachbarten Punkten, ist darauf zu achten, dass nicht über Ecken hinweg Segmente zusammengefasst werden, da nur gerade Linienabschnitte von Relevanz sind.

Der Algorithmus, welcher in Algorithmus 3 aufgezeigt ist, iteriert durch alle Punkte und versucht diese zu bestehenden Segmenten hinzuzufügen. Existiert kein oder passendes Segment, so beginnt mit dem aktuell unter Betracht gezogenen Punkt ein neues Segment. Damit ein Punkt zu einem Segment hinzugefügt werden kann, sind mehrere Validierungsstufen erforderlich. Um den Berechnungsaufwand komplexerer Prüfverfahren zu vermeiden, wird zuerst nur der euklidische Abstand überprüft. Sobald ein Segment aus zwei Punkten besteht, kann ein Richtungsvektor v der Geraden berechnet werden. Weitere Punkte werden dem Segment hinzugefügt, sofern die projizierte Distanz d auf den Richtungsvektor v einen Schwellwert nicht überschreitet. Allerdings können die ersten beiden Punkte eines Segments beliebig schlecht positioniert sein. Abbildung 3.2 verdeutlicht die Problematik. Die Positionierung ist abhängig von Präzision der Bildsegmente.

Eine andere Möglichkeit ist eine bestimmte Anzahl an Punkten zuzulassen und darauf die Regressionsgerade zu berechnen (siehe Abschnitt 2.2). Anschließend lässt sich die Projektionsdistanz weiterer Punkte zu der Regressionsgerade bestimmen.

Außerdem wäre es möglich die Zugehörigkeit eines Punktes zu einem Segment von der Varianz der Segmentpunkte abhängig zu machen, bzw. nur eine maximale Varianz zu erlauben. Aufgrund der Berechnungskomplexität kann dies aber nicht angewendet werden, da nach dem Hinzufügen eines neuen Punktes die Varianz neu berechnet werden muss und dafür jeder Segmentpunkt erneut zu durchlaufen ist. Hingegen lässt sich der Mittelwert iterativ aktualisieren. Wird erfordert, dass der Mittelwert immer etwa auf der Regressionsgeraden zu liegen hat, lassen sich stark abweichende Punkte, beispielsweise an Ecken, nur langsam feststellen, da dieses Verfahren einen Tiefpasscharakter mit sich bringt.

Die Regressionsgerade nimmt an Qualität zu, wenn zu Anfang der Punktüberprüfung kleine euklidische Abstände gefordert werden. Sobald die Regressionsgerade berechnet wurde, können größere Distanzen akzeptiert werden. Dabei ist zu beachten, dass der Punkt an das erstbeste Segment angefügt wird, da die Evaluation aller Segmente, insbesondere auf stark rauschenden Eingangsvektoren, zu aufwendig ist.

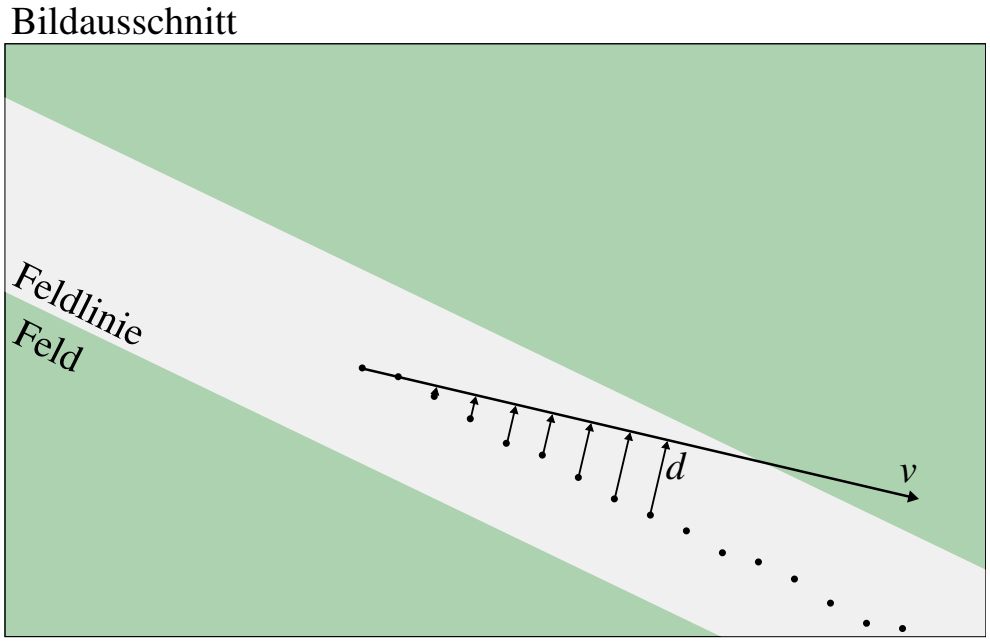


Abb. 3.2: *Ungenauigkeiten der auf einer Feldlinie liegender Segmentmittelpunkte. Die Berechnung der Projektionsdistanz d eignet sich nicht im Fall einer Definition des Richtungsvektor v durch die ersten beiden Segmentpunkte.*

Je nach Ausrichtung des Roboters auf dem Spielfeld können durchaus vertikal ausgerichtete Feldlinien auf dem Bild sichtbar sein. Die einfache lineare Regression liefert, sofern abhängige und unabhängige Variablen nicht getauscht werden, für diesen Fall keine verwendbaren Ergebnisse. Zudem ist nicht nur eine Koordinate messfehlerbehaftet. Daher ist für die Berechnung der Regressionsgerade nicht die einfache lineare Regression, sondern die orthogonale Regression zu verwenden (siehe Abschnitt 2.2.2). Schlägt die Berechnung der Ausgleichsgeraden fehl, so wird diese über den Start- und Endpunkt des Segments angenähert.

Abb. 3.3: *Zusammengefasste Liniensegmente. Start- und Endpunkte der schwarz markierten Segmente sind rot eingefärbt.*

(a) *Falsch positiv Detektion von Liniensegmenten in einem Roboter.*

(b) *Es werden kurze Liniensegmente in dem Mittelkreis erkannt.*

Abb. 3.4: *Visualisierung von zusammengefassten Segmenten nach dem Tracking. Segmente sind schwarz, Start- und Endpunkt eines Segments rot markiert.*

Wie Abbildung 3.3 darstellt, kann der Algorithmus mit dem bisher erwähnten Verfah-

ren nachvollziehbare Liniensegmente zusammenfassen. Allerdings zeigt Abbildung 3.4 falsch positiv Detektionen in dem Mittelkreis und anderen Robotern. Insbesondere in Abbildung 3.4b ist eine elementare Problematik zu erkennen. In Feldlinien lassen sich keine Kreisbogensegmente detektieren, allerdings finden sich immer kurze Liniensegmente in dem Mittelkreis. Ohne eine Detektion der Kreisbogensegmente, lässt sich dies nicht verhindern. Notwendig wird die Segmentierung benachbarter Punkte mit anschließender Klassifizierung von Linien und Kreisbögen. Insbesondere der Abbruch des Trackings, wenn die Projektionsdistanz des Punktes zu groß ist, stellt sich bei der Segmentierung des Mittelkreises als besonders hinderlich heraus. Wird dem Tracking erlaubt von einem linearen Kurs abzuweichen, werden Segmente über Ecken hinweg zusammengefasst. Damit dies nicht passiert, aber gleichzeitig von Kreisbögen differenziert werden können, wird ein Maß für die Krümmung erforderlich. Dies wird in diesem Fall, wie in Abschnitt 3.1.2 beschrieben, über die Berechnung der Gradienten realisiert. Zusätzlich lässt sich dem entgegenwirken, indem die Pixel aus den vertikalen und horizontalen Scanlines bei der Segmentierung getrennt voneinander betrachtet werden. Abbildung 3.5 zeigt, dass zwei aufeinander senkrecht stehende Linien von den vertikalen und horizontalen Scanlines unterschiedlich häufig wahrgenommen werden. Der Zusammenfassung zweier Linien zu einem Segment lässt sich daher entgegenwirken, wenn die Punktwolken getrennt voneinander behandelt werden.

Abb. 3.5: *Visualisierung der gemittelten Gradientenorientierungen der Linienkandidaten, Die rot markierten Gradienten stammen von vertikalen und die blauen von horizontalen Scanlinesegmenten.*

Demzufolge darf ein Punkt einem Segment nur hinzugefügt werden, wenn sein Gradient mit dem Gradienten des letzten Punktes aus dem Segment in einem gewissen Maße übereinstimmt. Bei Liniensegmenten wird angenommen, dass die Gradienten gleich ausgerichtet sind und sich nur minimal voneinander unterscheiden. Kreisbögen hingegen weisen mit jedem Punkt eine Zunahme der Winkeldifferenz zum vorherigen Punkt auf. Entsprechend Abbildung 2.3 lässt sich die Orthogonale des Gradienten als Segment eines Polygons betrachten. Werden die Winkeldifferenzen a_i aufsummiert, so lässt sich zu jedem Zeitpunkt eine Aussage treffen, ob es einem Linien- oder Kreisbogensegment entspricht. An dieser Stelle sei kurz erwähnt, dass der Winkel nicht zwangsweise immer zunimmt. Bilddaten können rauschbehaftet sein und die Gradientenberechnung stören. Es gilt daher das Vorzeichen, wie in Abschnitt 2.5 beschrieben, zu beachten. Mit dieser Zustandsaussage können Validierungen ausschließlich auf Liniensegmente angewandt werden. Das erlaubt die Überprüfung der Projektionsdistanz allein auf die Liniensegmente anzuwenden und im Fall eines Verstoßes das Tracking abzubrechen.

Die Gradienten geben ein besseres Maß dafür wie zwei aufeinander folgende Punkte zusammen passen. Insbesondere kann der Schwellwert für die euklidische Distanz der

ersten Überprüfung gelockert werden. Das sorgt gleichzeitig dafür, dass im Falle eines rauschenden und daher nicht übereinstimmenden Gradientens dieser übersprungen werden kann, ohne dass die Segmentierung abbricht. Daraus lässt sich ein grundlegendes Problem ableiten. Werden Abstandsschwellwerte zu klein gewählt, kann es passieren, dass das Tracking verfrüht abbricht, da keine weiteren Punkte in der Umgebung zu finden sind. Ein verfrüht abgebrochenes Tracking gibt einem Kreisbogen nicht die Möglichkeit sich als Kreisbogen zu identifizieren. Werden Abstandsschwellwerte zu groß gewählt, werden irrelevante Punkte in Betracht gezogen.

In (REINHARDT, 2011, S. 76) wird ein Ähnlichkeitsmaß zweier Punkte vorgestellt. Wird das Maß auf die hier vorliegende Problemstellung übertragen, wird die Verbindungslinie $v_{ij} := P_i - P_j$ zwischen dem letzten Punkt des Segments P_i und dem zu evaluierenden Punkt P_j in Betracht gezogen (siehe Abbildung 3.6). Die Berechnung des Maßes $s(i, j)$ erfolgt nach Gleichung (3.2).

$$s(i,j) := \begin{cases} 1 - \left(\frac{|v_{ij} \cdot g_i|}{|v_{ij}| * |g_i|} + \frac{|v_{ij} \cdot g_j|}{|v_{ij}| * |g_j|} \right) / 2 & \text{wenn } g_i \cdot g_j > 0 \\ 0 & \text{sonst} \end{cases} \tag{3.2}$$

In Abbildung 3.7 ist die Anwendung auf reale Punkte dargestellt. Dabei lässt sich erkennen, dass sich das Maß nicht auf die Scanlinesegmente, aus dem Framework der HULKs, anwenden lässt. Auf einem solchen Mittelkreisabschnitt entsteht ein Versatz in der Punktpositionierung, welcher mit der Präzision und Abtastrate der Kantendetektion zusammen hängt. Somit findet dieses Maß keine Anwendung in dem hier vorgestellten Algorithmus.

3.1.4 Differenzierung zwischen Linien- und Kreisbogensegmenten

Die Segmente sind nun in Algorithmus 4 nach Linien und Kreisbögen aufzuteilen, damit die relevanten Liniensegmente der Hough-Transformation unterzogen werden können. Zu kurze Segmente werden in diesem Schritt gefiltert. Von der Klassifizierung eines Kreisbogens anhand der Winkeldifferenz des ersten und letzten Gradienten ist abzusehen, da dies sehr rauschanfällig ist. Außerdem kann im Grenzfall nur eine maximale Differenz von $\frac{\pi}{2}$ festgestellt werden, da im Fall eines vertikal ausgerichteten Kreisbogenabschnittes Start und Ende des vertikalen Scanlinesegments tauschen. Der akkumulierte Winkel aus der Tangentialraumbetrachtung ist für die Klassifizierung aussagekräftiger. Außerdem sollte die Anzahl der Grenzwertüberschreitungen bei der Projektionsdistanz mit in Betracht gezogen werden. Alle nicht als Kreisbogen klassifizierten Segmente stellen den Eingang für die nächsten Berechnungen.

Algorithmus 3 Zusammenfassung von Punkten zu Segmenten

```
1: function COMBINETOSEGMENTS(points)
2:     segments ←{}
3:     for all points do
4:         pointAddedToSegment ←false
5:         for all segments do
6:             if point is not near segment end then
7:                 break
8:             end if
9:             if orthogonal regression line of the segment has been calculated and
                distance from point to regression line is too large then
10:                distanceViolation ←true
11:            end if
12:            if accumulated angle per segment point is small enough and
                distanceViolation then
13:                break
14:            end if
15:            angleDifference ←Calculate the angle between the current and previous gradient mean.
16:            if angleDifference is too big then
17:                break
18:            end if
19:            if segment consists of a predefined number of points then
20:                Calculate orthogonal regression line on segment points.
21:            end if
22:            if distanceViolation then
23:                Add one to segments distance violation counter.
24:            end if
25:            Accumulate the angleDifference. ▷ Depending on whether it follows the arc add to or substract from the accumulator.
26:            Add point to current segment.
27:            pointAddedToSegment ←true
28:            Stop and start over with next point.
29:        end for
30:        if not pointAddedToSegment then
31:            segment ←Convert current point into a segment with only one point.
32:            segments ←{segments, segment}
33:        end if
34:    end for
35:    return segments
36: end function
```

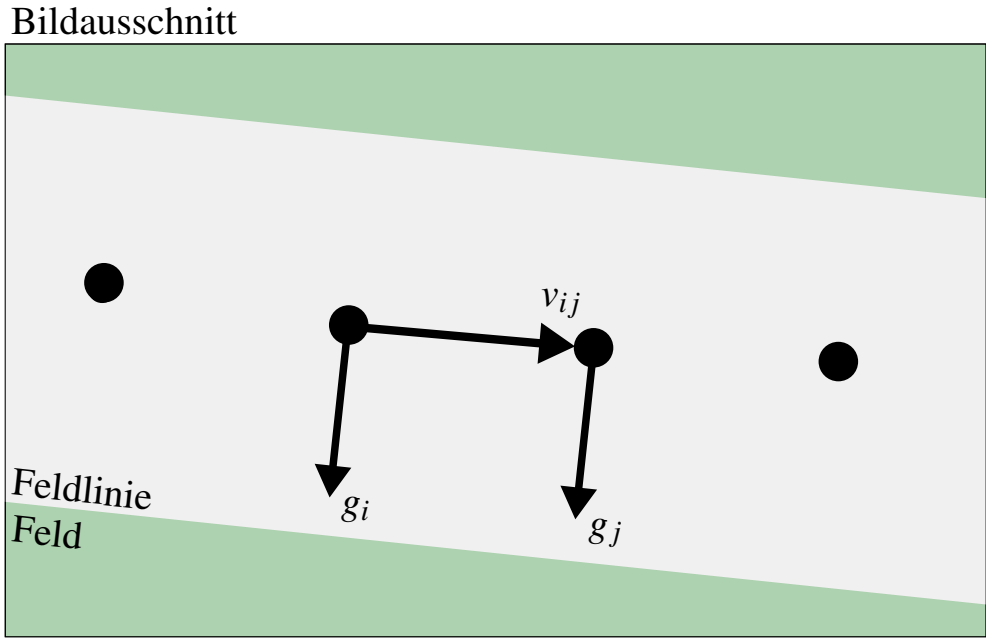


Abb. 3.6: *Visualisierung der für das Ähnlichkeitsmaß notwendigen Vektoren. Abgebildet sind die Gradienten g_1 und g_2 der Scanlinesegmente. Der Vektor v_{ij} ist Verbindungslinie zwischen zwei Kandidatenpunkten.*

3.1.5 Zusammenfassung von Liniensegmenten

Ziel in diesem Schritt ist die Zusammenfassung einzelner Liniensegmente, die auf derselben Feldmarkierung liegen. Dazu wird, wie bereits in Abschnitt 3.1.3 vorgegriffen, jedes Liniensegment in den Parameterraum transformiert und mit ähnlichen Liniensegmenten zusammengefasst. Ein Eintrag im Akkumulator identifiziert sich über die Distanz zum Bildursprung und über den Winkel relativ zu einer Bildachse. Das charakteristische Problem unendlicher Steigungen bei vertikalen Linien existiert daher in diesem Fall nicht. Zusätzlich werden dazugehörige Metadaten der Linien gespeichert, so kann nachträglich die gesamte Länge zusammengefasster Segmente im Bildraum festgestellt werden.

Algorithmus 5 iteriert durch alle Liniensegmente und versucht diese mit einem ähnlichen Eintrag im Akkumulator zu kombinieren. Findet sich kein Kandidat, so führt das Liniensegment zu einem neuen Eintrag im Akkumulator. Zuerst wird die Distanz zum Bildursprung in Betracht gezogen. Dabei ist es wichtig die prozentuale Abweichung zu bestimmen und nicht den Grenzwert an einer maximalen Abweichung der Distanz festzumachen. Abbildung 3.8 verdeutlicht das Problem fester Distanzschwellwerte.

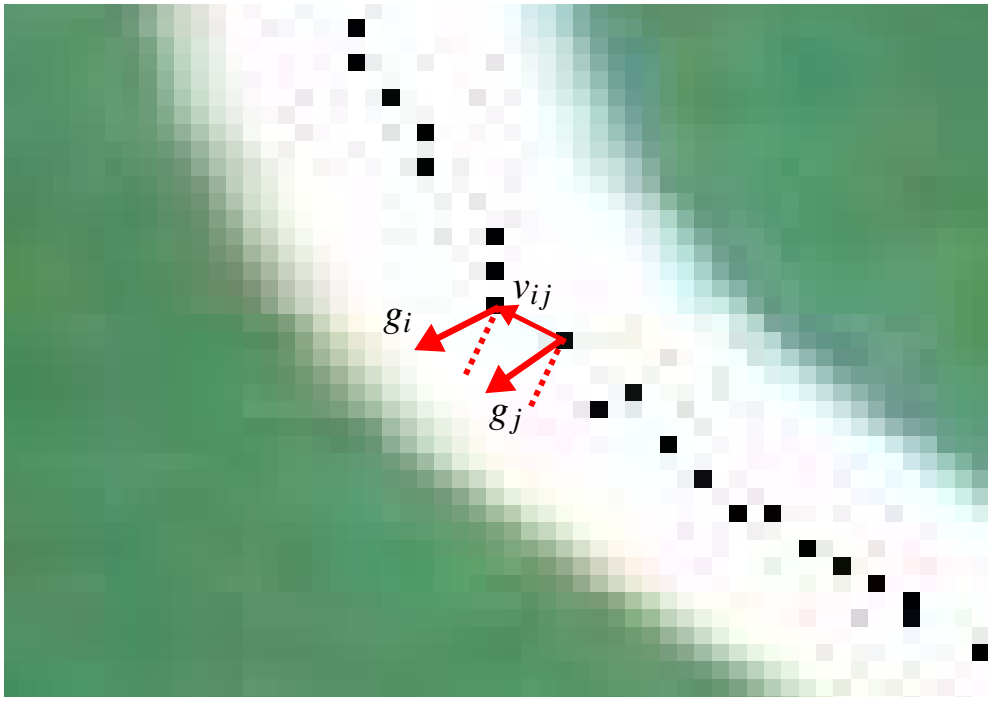


Abb. 3.7: *Berechnete Gradienten g_i und g_j zweier Linienkandidatenpunkte. Das Ähnlichkeitsmaß führt in einer solchen Situation zu einer schlechten Bewertung.*

Anschließend werden die Ausrichtungen der Linien miteinander verglichen. Zur Vermeidung der Gruppierung mit falsch positiv Detektionen, darf zusätzlich der zu überbrückende Abstand zwischen den Liniensegmenten nicht zu groß sein. Von der Überprüfung der Orientierung der Verbindungslinie zwischen den Segmenten ist in vielen Fällen abzusehen. Wird die gleiche Orientierung, wie die der zu gruppierenden Linien erfordert, kann dies bereits ein Ausschlusskriterium sein. Man beachte den Fall dicht aneinanderliegender Liniensegmente, die sich nur um einen leichten Versatz unterscheiden. Siehe dazu auch Abbildung 3.7. Zu beachten ist, dass das Liniensegment nicht mit allen innerhalb der Schwellwerte liegender Akkumulatoreinträge zusammengefasst werden darf, da es sonst zu der Aufwertung falsch positiver Einträge kommt. Ein Liniensegment sollte daher nur mit der am meisten übereinstimmenden Linie zusammengefasst werden. Bewertet wird dies anhand der Summe der Projektionsdistanzen des Start- und Endpunktes auf die Akkumulatorlinie. Für die Gruppierung zweier Linien sei hier noch angemerkt, dass in diesem Schritt zum ersten Mal Liniensegmente aus den Datenpunkten der vertikalen und horizontalen Scanlines zusammen kommen. Bisher lagen Datenpunkte entsprechend ihrer Herkunft sortiert vor. Dies erfordert nun bei der Zusammenführung die Neuberechnung der Linienendpunkte. Sofern die Distanz zum Bildursprung über den Normalenvektor der Geraden berechnet wird, kann sich die

Algorithmus 4 Differenzierung von Linien- und Kreisbogensegmenten

```
 1: function DIFFERENTIATE(segments)
 2:     lineSegments ←{}
 3:     arcSegments ←{}
 4:     for all segments do
 5:         if segment is too small then
 6:             Stop and go to next segment.
 7:         end if
 8:         if accumulated angle per segment point is big enough and
              enough distance violations then
 9:             arcSegments ←{arcSegments, segment}
10:         else
11:             lines ←{lines, segment}
12:         end if
13:     end for
14:     for all lineSegments do
15:         Update direction vector.
16:         Calculate perpendicular line.
17:         Calculate distance to image origin.
18:     end for
19:     return (lineSegments, arcSegments)
20: end function
```

Ausrichtung unterscheiden und es ist daher der Betrag zu verwenden.

3.1.6 Bereichsfilter

Dieser Schritt dient ausschließlich der Filterung weiterer falsch positiv Detektionen. Algorithmus 6 unterteilt dafür das Bild in acht gleich große Spalten und macht sich dabei die Anordnung der Feldmarkierungen zu Nutze. Auf den Körpern anderer Roboter werden zumeist Falschdetektionen wahrgenommen. Dies äußert sich in mehreren vertikalen Linien in den Beinen und Armen. Beginnt und endet eine Linie in der gleichen Spalte, erhält die Spalte einen Strafpunkt. Bei drei Strafpunkten werden alle ausschließlich in dieser Spalte befindlichen Linien für die weitere Verwendung ausgeschlossen. Dieser Schritt ist höchst anpassbar und auf weitere verschobene Raster erweiterbar. Auf dem Feld kann es ansonsten nicht vorkommen, dass drei Feldlinien auf gleicher Höhe starten und enden (ROBOCUP TECHNICAL COMMITTEE, 2018)

Algorithmus 5 Zusammenfassung von Liniensegmenten

```
1:  function HOUGHTRANSFORM(lineSegments)
2:      houghLines ←{}
3:      for all lineSegments do
4:          bestMatchingHoughLine ←{}
5:                                          ▷ Find similar line in houghLine accumulator
6:          for all houghLines do
7:                                          ▷ Compare each lineSegment with each houghLine.
8:              if distance to origin ratio is too big or
                  angle between them is too big or
                  connecting line between them is too long or
                  connecting line is longer than lineSegment then
9:                  break
10:             end if
11:             (d1,d2) ←Orthogonal distances from start and end point to houghLine.
12:             if d1 or d2 is too large then
13:                 break
14:             end if
15:             penalty ←d1 + d2
16:             if penalty smaller than bestMatchingHoughLine's penalty then
17:                 bestMatchingHoughLine ←houghLine
18:             end if
19:         end for
20:         if bestMatchingHoughLine then
21:             Combine line segment with bestMatchingHoughLine
22:             Update score.
23:             Recalculate line endings.
24:             Update direction vector.
25:             Recalculate perpendicular line.
26:             Recalculate distance to image origin.
27:         else
28:             houghLines ←{houghLines, lineSegment}
29:         end if
30:     end for
31:     for all houghLines do
32:         Calculate pixel length.
33:     end for
34: end function
```

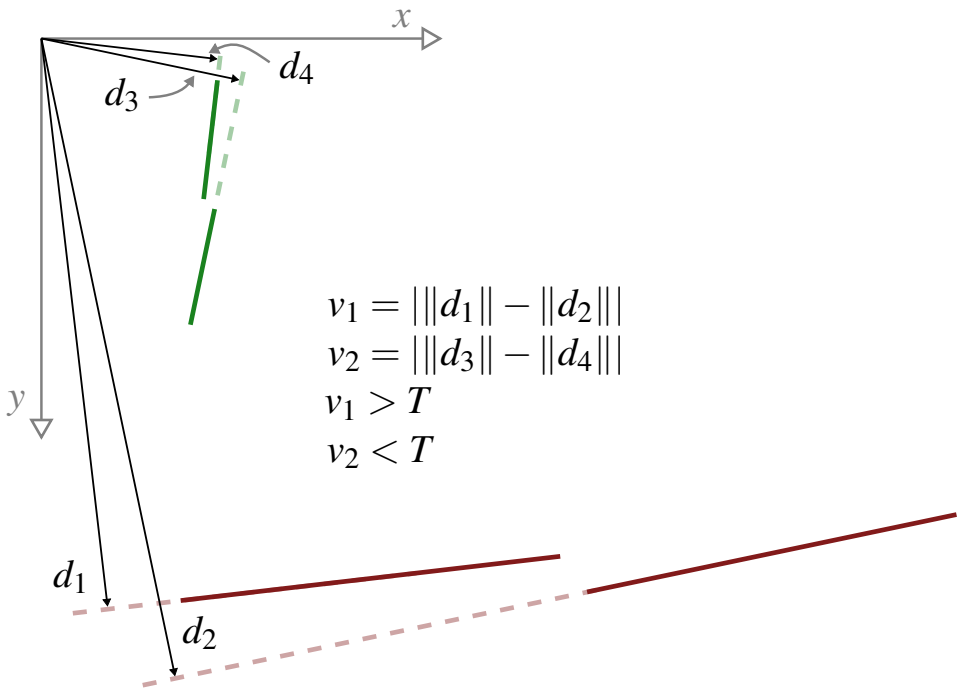


Abb. 3.8: *Visualisierung zweier Liniensegmentpaare, dessen Winkeldifferenz identisch ist. Das obere Linienpaar (grün) ist eine gedrehte verkleinerte Spiegelung des unteren Linienpaares (rot). Ersichtlich wird, dass kein fester Schwellwert der Differenzen zum Bildursprung herangezogen werden darf, um über die Ähnlichkeit zweier Liniensegmente zu entscheiden.*

Algorithmus 6 Bereichsfilter

```
1:  function FILTERABSURDITY(houghLines)
2:      Divide image in eight columns
3:      for all houghLines do
4:          if houghLine starts and ends in same column then
5:              Increase penalty of this column
6:          end if
7:      end for
8:      for all columns do
9:          if column penalty is equal or greater three then
10:             Delete all lines in this column
11:         end if
12:     end for
13:     return houghLines
14: end function
```

3.1.7 Gruppierung orthogonaler Linien

Auf die Projektion der Linienpunkte in das Roboterkoordinatensystem wurde bisher verzichtet, da die Transformation rechnerisch aufwendig ist (RIEBESEL, 2018, S. 6). Die Projektion großer Punktwolken ist daher zu vermeiden. Die Start- und Endpunkte der Linien werden in diesem Schritt in Roboterkoordinaten transformiert. Unter der Annahme korrekt vorliegender Transformationsmatrizen lassen sich so nach der Projektion rechte Winkel in den Ecken der Spielfeldmarkierungen messen. Algorithmus 7 soll in diesem Schritt die Linien gruppieren. In einer Gruppe befinden sich alle parallelen und dazu senkrecht stehenden Linien, die in zwei Untergruppen voneinander getrennt gehalten werden. Lässt sich eine Linie aufgrund der Anforderungen einer Gruppe nicht hinzufügen, so wird eine neue Gruppe erstellt. Die Bewertung einer Gruppe wird an der Anzahl der enthaltenen Linienpunkte gemessen. Nur die Linien aus der besten Gruppe werden für den nächsten Schritt verwendet. Eine vorherige Sortierung der Linien nach Länge sorgt dafür, dass sich die Gruppen vorzugsweise um die längsten und damit wahrscheinlich plausibelsten Linien bilden.

3.1.8 Extraktion der Linienkreuzungen

Die Linien der besten Gruppe aus der vorherigen Berechnung sind in zwei Untergruppen aufgeteilt, die um $\frac{\pi}{2}$ zueinander rotiert sind. In diesem Schritt werden in Algorithmus 8 die Linien aus der einen Untergruppe mit jeder Linie aus der anderen Untergruppe geschnitten. Die Berechnung findet auf den im Roboterkoordinatensystem befindlichen Datenpunkten statt. Nach der Berechnung des Schnittpunktes ist dieser zu klassifizieren. Es können X-, T- und L-Kreuzungen vorliegen. Für die Klassifizierung werden zuerst die Richtungsvektoren von den Linienendpunkten zu dem Schnittpunkt berechnet. Das Skalarprodukt der beiden zu einer Linie gehörenden Vektoren gibt Aufschluss über die Position des Schnittpunktes. Ist das Ergebnis des Skalarproduktes < 0, so zeigen die Richtungsvektoren in entgegengesetzte Richtungen und der Schnittpunkt liegt folglich auf der Linie. Sind beide Skalarprodukte < 0, so liegt eine X-Kreuzung vor. Ist nur eins der beiden Skalarprodukte < 0, so handelt es sich um eine T-Kreuzung. Bei der L-Kreuzung sind beide Skalarprodukte positiv.

3.1.9 Klassifizierung der Linienkreuzungen

Die zuvor klassifizierten Kreuzungen können in Grenzbereichen falsche Klassen aufweisen. Überlappt bei einer X-Kreuzung eine Linie nur leicht die andere, so ist eine Reklassifizierung zu einer T-Kreuzung notwendig. Algorithmus 9 iteriert durch alle zuvor erkannten Kreuzungen und nimmt bei Notwendigkeit die Reklassifizierung vor. Zur

Algorithmus 7 Gruppierung orthogonaler Linien

```
1:  function PROJECTTOGROUNDCOORDINATES(houghLines)
2:      projectedHoughLines ←{}
3:      for all houghLines do
4:          Transform line start and end to robot coordinate system.
5:      end for
6:      return projectedHoughLines
7:  end function
8:
9:  function ORTHOGONALCLUSTERING(projectedHoughLines)
10:     lineCluster ←{}
11:             ▷ A lineCluster consists of all parallels and all their orthogonal lines.
12:     Sort projectedHoughLines by their pixel length in descending order.
13:     for all projectedHoughLines do
14:         partOfCluster ←false
15:         for all lineCluster do
16:                         ▷ Check each projectedHoughLine against all lineClusters.
17:             angle ←angle between projectedHoughLine and lineCluster.
18:             if angle is not roughly 0 or π/2 then
19:                 break
20:             end if
21:             Add projectedHoughLine to lineCluster.
22:             Maintain two sub clusters which are rotated by roughly π/2.
23:             Update lineClusters score based on projectedHoughLines score.
24:             partOfCluster ←true
25:         end for
26:         if not partOfCluster then
27:             newCluster ←Form a new cluster out of projectedHoughLine.
28:             lineClusters ←{lineClusters, newCluster}
29:         end if
30:     end for
31:     lines ←Get all lines from cluster with highest score.
32:     return lines
33: end function
```

Algorithmus 8 Extraktion der Linienkreuzungen

```
1:  function GETINTERSECTIONS(orthogonalLines)
2:      ▷ The orthogonal lines are separated in two clusters. The lines of one cluster
    are rotated by roughly π/2 to the other cluster.
3:      lines ←The lines from first cluster.
4:      rotatedLines ←The lines from second cluster.
5:      for all lines do
6:          for all rotatedlines do
7:              intersection ←Calculate intersection.
8:                                  ▷ Calculations are done in robot coordinate system.
9:              vl1 ←intersection − line.start
10:             vl2 ←intersection − line.end
11:             vr1 ←intersection − rotatedLine.start
12:             vr2 ←intersection − rotatedLine.end
13:             dpl ←vl1 ∗ vl2                                          ▷ Dot product
14:             dpr ←vr1 ∗ vr2
15:             if dpl < 0 then
16:                 Intersection is on line.
17:             end if
18:             if dpr < 0 then
19:                 Intersection is on rotated line.
20:             end if
21:             if intersection is on both lines then
22:                 Classify as X-intersection in first place.
23:             else if intersection is on one line then
24:                 Classify as T-intersection in first place.
25:             else if intersection is neither on one line nor on the other then
26:                 Classify as L-intersection in first place.
27:             end if
28:         end for
29:     end for
30:     return intersections
31: end function
```

Filterung falscher Kreuzungen werden die entfernt, die zu große Distanzen zu ihren Linien aufweisen.

Algorithmus 9 Klassifizierung der Linienkreuzungen

```
 1: function REFINEINTERSECTIONS(intersections)
 2:     for all intersections do
 3:         if is X-intersection and
              one line doesn't overlap enough across the intersection then
 4:             Degrade to T-intersection.
 5:         end if
 6:         if is T-intersection and
              one line doesn't overlap enough across the intersection then
 7:             Degrade to L-intersection.
 8:         end if
 9:         if intersection is too far away from one of the lines then
10:             Delete intersection.
11:         end if
12:     end for
13: end function
```

3.2 Liniendetektion der HULKs

Die bereits im Framework der HULKs bestehende Liniendetektion sei hier kurz beschrieben. Der Pseudocode ist in Algorithmus 10 aufgezeigt. Der wesentliche Unterschied besteht in der alleinigen Verwendung der vertikalen Scanlinesegmente und dem Einsatz von RANSAC. Der Algorithmus beginnt mit der Auswahl der Kandidatenpunkte anhand der gegebenen Scanlinesegmente. Entspricht der Start des Segments einer steigenden und das Ende einer fallenden Kante, so werden die Segmentkanten in das Roboterkoordinatensystem transformiert. Zwischen den transformierten Punkten wird der Abstand berechnet und grob auf die Dicke einer Feldlinie überprüft. Entsprechend Abbildung 3.1 werden die Gradienten an Segmentstart und -ende berechnet. Die Gradientenvektoren müssen etwa parallel und in entgegengesetzte Richtungen zeigen. Der Mittelpunkt des Segments wird als Kandidatenpunkt übernommen.

Anschließend wird mehrfach RANSAC auf die Punktwolke angewendet. Jede Iteration in Algorithmus 11 resultiert in einer Linie. Die an der Linie teilhabenden Punkte werden aus der Punktwolke entfernt und es beginnt eine neue Iteration. Das Verfahren ist in Abschnitt 2.4 beschrieben und kann auch in Algorithmus 11 nachvollzogen werden.

RANSAC klassifiziert alle Punkte nach der Zugehörigkeit zu einer Linie. Die äußersten Punkte, also die mit dem größten Abstand zueinander, definieren die aus RANSAC resultierende Linie. Dies führt dazu, dass zwischen Linienpunkten große Abstände herrschen können. Eine nachträgliche Korrektur, wie in Algorithmus 12 beschrieben, ist dadurch notwendig. Das Verfahren ist rekursiv und prüft die Linie auf größere Punktabstände. Ist die Linie vor einer auftretenden Lücke lang genug, wird sie beibehalten. Auf den zweiten Teil der Linie hinter der Lücke wird rekursiv die Korrektur erneut ausgeführt.

Algorithmus 10 Liniendetektion basierend auf RANSAC

```
1:  function LINEDETECTION(verticalScanlineSegments)
2:      lines ←{}
3:      points ←{}
4:      for all verticalScanlineSegments do
5:          if segment begin is not an rising edge or
              segment end is not an falling edge or
              segments projection too long then
6:              Stop and evaluate next vertical scanline segment.
7:          end if
8:          if gradient vectors at begin and end of the segment are not parallel or
              gradient vectors point in same direction then
9:              Stop and evaluate next vertical scanline segment.
10:         end if
11:         mid ←Segment center point
12:         points ←{points, mid}
13:     end for
14:
15:     relevantPoints ←points
16:     while enough relevant points do
17:         Calculate the number of iterations that RANSAC should run.
18:         (line, unusedPoints) ←RANSAC(relevantPoints, iterations)
19:         CORRECTLINE(line)
20:         relevantPoints ←unusedPoints
21:     end while
22:     return lines
23: end function
```

Algorithmus 11 RANSAC Algorithmus

```
1:  function RANSAC(relevantPoints, iterations)
2:      bestLineScore = 0
3:      bestLine ←{}
4:      for iterations do
5:          p1, p2 ←Pick two random points from relevantPoints.
6:          line ←Line through the points p1 and p2.
7:          lineScore = 0
8:          for all remaining points in relevantPoints do
9:              Calculate orthogonal distance from point to line.
10:             if orthogonal distance is smaller than a predefined threshold then
11:                 Increase lineScore by one.
12:             end if
13:         end for
14:         if score > bestLineScore then
15:             bestScore = score
16:             bestLine = line
17:         end if
18:     end for
19:     unusedPoints ←All points not part of bestLine.
20:     return bestLine, unusedPoints
21: end function
```

Algorithmus 12 Überprüfung einer Linie auf Unterbrechungen

```
1:  function CORRECTLINE(line)
2:      if line is too short then
3:          return
4:      end if
5:      if line has a large gap then
6:                                  ▷ A large gap is characterized by missing line points.
7:          Split line at first gap into two line parts (line1 and line2).
8:          if line1 is long enough then
9:              lines ←{lines, line1}
10:         end if
11:         CORRECTLINE(line2)
12:     else
13:         lines ←{lines, line}
14:     end if
15: end function
```

3.3 Strafstoßmarke

Der entworfene Algorithmus zur Detektion der Strafstoßmarke ist in Algorithmus 13 kurz zusammengefasst. Wie auch die Liniendetektion basiert dieser Algorithmus auf den vertikalen und horizontalen Scanlinesegmenten. In Abschnitt 3.1.1 ist die Generierung der Scanlinesegmente erläutert. Die Detektion der Strafstoßmarke wurde orthogonal zu der Liniendetektion entwickelt, um im ersten Zuge nicht auf diesen Ergebnissen zu basieren.

Algorithmus 13 Übersicht Strafstoßmarkendetektion

```
1: function PENALTYSPOTDETECTION(
        verticalScanlineSegments,
        horizontalScanlineSegments)
2:     horizontalCandidates ←FILTERSEGMENTS(horizontalScanlineSegments)
3:     pairs ←GETOVERLAPS(horizontalCandidates, verticalScanlineSegments)
4:     filteredPairs ←FILTER(pairs)
5:     penaltySpotCandidates ←CHECKELLIPTICALSURROUNDING(filteredPairs)
6:     penaltySpot ←EVALUATECANDIDATES(penaltySpotCandidates)
7: end function
```

3.3.1 Auswahl der Segmentkandidaten

Der Algorithmus funktioniert wie folgt. Horizontale Scanlinesegmente entstehen auf projizierten Scanlines mit gleichen Abständen in Weltkoordinaten (FELBINGER, 2018, S. 21). Dementsprechend liegen weniger horizontale als vertikale Segmente vor. Es lässt sich daher eine schnellere Übersicht schaffen, wenn die horizontalen Scanlinesegmente zuerst durchsucht werden. Der Unterschied in der Datenmenge lässt sich insbesondere auf der unteren der beiden Kameras des NAO-Robotiksystems feststellen. Ein Segment darf nicht zu klein und nicht zu weit von dem Roboter entfernt sein. Die Detektionsentfernung wird an dieser Stelle beschränkt, um falsch positiv Detektionen zu reduzieren. Außerdem wird berechnet wie breit eine Strafstoßmarke an der projizierten Position des Segments sein muss. Stimmt dies nicht überein, so wird das nächste Segment evaluiert. Andernfalls werden alle vertikalen Segmente an dieser Position in Betracht gezogen. Das vertikale Segment darf in Pixelkoordinaten nicht länger als das horizontale Segment sein. Sofern sich die beiden überlappenden Segmente nicht auf der detektierten Ballposition befinden, wird der theoretische Schnittpunkt berechnet. Diese Berechnung dient der Positionskorrektur. Der theoretische Schnittpunkt entspricht dem Punkt, in dem sich die beiden Segmente überschneiden würden, würden sie sich jeweils mittig überlappen.

3.3.2 Überprüfung der Ellipsenumgebung

Um den Mittelpunkt werden zwölf äquidistante Punkte auf einer Ellipsenbahn berechnet. Die Ellipse hat den anderthalbfachen Radius einer an der projizierten Position befindlichen Strafstoßmarke. Jeder dieser Punkte muss innerhalb des Bildes liegen und einen dunkleren Luminanzwert als der Mittelpunkt aufweisen. Zusätzlich hat man die Wahl zwischen zwei verschiedenen Überprüfungen. Entweder kann die Farbigkeit eines Pixels über die Summe der Chrominanzkanäle angenähert und eine höhere Farbigkeit erfordert werden oder andernfalls muss jeder Punkt der im vornherein detektierten Feldfarbe entsprechen. Für die Bewertung der Strafstoßmarkenkandidaten sind an dieser Stelle signifikante Differenzen in der Luminanz und Chrominanz zu zählen. Zusätzlich fließt die Abweichung der Segmentüberschneidung von dem theoretischen Mittelpunkt mit in die Bewertung einer Strafstoßmarke ein. Die Bewertungsfunktion ist in Gleichung (3.3) gegeben. Der vollständige Algorithmus zu der Detektion der Strafstoßmarke ist in Algorithmus 14 dargestellt.

$$S(\text{candidate}) = n_{luminance} + n_{chroma} - d_{deviation} \tag{3.3}$$

Algorithmus 14 Implementierung der Strafstoßmarkendetektion

```
1: function PENALTYSPOTDETECTION(
           verticalScanlineSegments,
           horizontalScanlineSegments)
2:     for all horizontalScanlineSegments do
3:         if too small or
            too far away or
            doesn't match in length with a penalty spot at that position then
4:             Stop and evaluate next horizontal scanline segment.
5:         end if
6:         for all verticalScanlineSegments do
7:             if doesn't overlap with the current horizontal segment or
              longer than the current horizontal segment or
              is on the ball then
8:                 Stop and evaluate next vertical scanline segment.
9:             end if
10:            Calculate mid point.  ▷ The mid point is the point where the segments
    would overlap if they intersect with their segment centers.
11:            Calculate 12 points on ellipse around mid point.
12:            for all points on ellipse do                    ▷ Compare to mid point
13:                if point is outside the image or
                  minimum difference in luma not met then
14:                    Stop and evaluate next vertical scanline segment.
15:                end if
16:                if chroma check is required and
                  minimum difference in chroma not met then
17:                    Stop and evaluate next vertical scanline segment.
18:                end if
19:                if field color check is required and
                  point is not field color then
20:                    Stop and evaluate next vertical scanline segment.
21:                end if
22:                Count significant differences in luma and chroma.
23:            end for
24:            Take mid point as penalty spot candidate.
25:            Calculate the score by adding the number of significant points and subs-
    tracting the distance between intersection and mid point.
26:        end for
27:    end for
28:    Pick penalty spot with highest score.
29: end function
```

Kapitel 4

Evaluation

Das folgende Kapitel behandelt die Auswertung der im Rahmen dieser Arbeit entwickelten Linien- und Strafstoßmarkendetektion. Das Framework der HULKs erlaubt die Aufnahme von Kamerabildern inklusive aller Metadaten, wie Kameramatrizen und Konfigurationswerte, während des Spiels. Aus Laufzeitgründen kann allerdings nicht jedes Bild gespeichert werden, so ergibt sich ein zeitlicher Sprung zwischen den Bildern von etwa drei Sekunden. Die folgenden Auswertungen gehen zuerst auf die verwendeten Metriken ein und werten die Algorithmen auf Spielsequenzen der IranOpen2018, GermanOpen2018 und des RoboCups2018 aus. Es wird außerdem auf vorhandene Probleme und Laufzeiten auf dem NAO-Robotiksystem eingegangen. Zudem wird die aktuelle Liniendetektion der HULKs dem Algorithmus dieser Arbeit gegenübergestellt.

Damit eine automatische Auswertung, auch zukünftiger Anpassungen, stattfinden kann, wurden auf 1535 Kamerabildern die Ground-Truth-Daten markiert[1]. Die Daten umfassen alle Roboter-, Ball-, Strafstoßmarken- und Linienpositionen auf dem Kamerabild.

Die zur Auswertung verwendeten Kamerabilder stammen von neun verschiedenen Spielen unterschiedlicher Wettkämpfe. Daher unterscheiden sich insbesondere die Lichtbedingungen, Spielfeld- und Trikotfarben. Eine Spielsequenz stammt immer von einem einzigen Roboter. Der repräsentative Roboter wurde zufällig gewählt.

4.1 Liniendetektion

Für die Auswertung wird eine Liniendicke von 10 px angenommen, damit übliche Metriken der Objektdetektion zum Einsatz kommen können. Mit den entstehenden ori-

[1] An dieser Stelle einen großen Dank an Felix Wege, Konrad Nölle, Pascal Gleske und René Kost für Unterstützung bei der Erstellung der Ground-Truth-Daten.

entierten Bounding-Boxen kann der Jaccard-Koeffizient bzw. die Intersection over Union (IoU) berechnet werden. Die IoU ist eine Maß für die Übereinstimmung zweier Mengen und ist definiert über die Fläche der Schnittmenge durch die Fläche der Vereinigung. Die IoU ist in Gleichung (4.1) definiert. Der Wertebereich beläuft sich auf $[0,1]$.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{4.1}$$

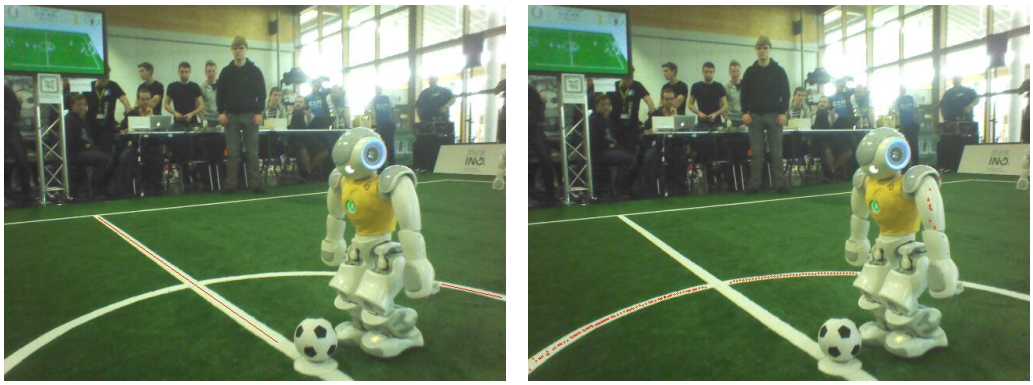
Für die Lokalisierung auf dem Spielfeld können kurze detektierte Abschnitte einer Linie ausreichend sein, sofern man sich dieser sicher ist. Dafür ist es notwendig, die falsch positiv Detektionen niedrig zu halten. In dem Folgenden gilt eine Linie als detektiert, wenn die IoU mindestens einen Wert von $0,2$ aufweist. Das entspricht einer Überlappung von 20%.

Bei der Liniendetektion handelt es sich nicht allein um ein Klassifizierungs-, sondern um ein Detektionsproblem. Der Unterschied liegt in der zusätzlich notwendigen Lokalisierung der Objekte auf dem Bild. Zudem lässt sich das Nichtvorhandensein einer Linie an einer Bildposition nicht quantisieren. So kann auch keine Aussage über die Gesamtheit der negativen Klassen oder die Anzahl negativer klassifizierter Ergebnisse gemacht werden. Bei mehrfacher Detektion der gleichen Linie wird nur eine richtig positiv Detektionen gezählt. Jede weitere Übereinstimmung sorgt für eine falsch positiv Detektion, da die Aufgabe des Detektors ist an dieser Stelle die Filterung zu übernehmen (EVERINGHAM u. a., 2010, S. 12).

4.1.1 Ergebnisse auf Spielsequenzen

Die Spielsequenzen enthalten zusammen 2028 Ground-Truth-Linien. Der Linienalgorithmus dieser Arbeit trifft auf den Sequenzen 2433 Linienvorhersagen. Es wird dabei bereits die hohe Anzahl an falsch positiv Detektionen deutlich. Der Algorithmus macht insgesamt 1345 falsche Vorhersagen, so kommt es in 55% zu Falschdetektionen. Dies entspricht einem positiv prädiktiven Wert von $0,44$, also einer Genauigkeit der Vorhersagen von 44%. Mit einer richtig positiv Rate von $0,54$ werden etwa die Hälfte der existierenden Linien gefunden, dies bedeutet aber natürlich gleichermaßen 46% verfehlte Detektionen. Für eine erfolgreiche Lokalisierung anhand der Feldlinien ist es, wie bereits erwähnt, nicht erforderlich alle Linien wahrzunehmen, wenn dies einer geringen Falscherkennungsrate gegenübersteht. Zudem ist es vernachlässigbar Linien in ihrer Vollständigkeit auf dem Kamerabild wahrzunehmen. Die hier erzielte IoU beläuft sich auf etwa 55%. Die aber in diesem Fall vorliegende hohe Falscherkennungsrate hat verschiedene Ursachen, die in dem Folgenden besprochen werden. Die Ergebnisse sind in Tabelle 4.1 zusammengefasst.

Der Algorithmus ist darauf angewiesen, dass er selbständig Kreisbogensegmente detektiert und von weiterer Betrachtung ausschließt. Jede auf einem Kreisbogensegment detektierte Linie resultiert in einer falsch positiv Detektion. Abbildung 4.1a zeigt exemplarisch die Fehldetektion auf einem Mittelkreis. Der NAO Roboter verdeckt teilweise den Mittelkreis, wodurch aufgrund der Kürze keine Krümmung auf dem Segment rechts von dem NAO festgestellt werden kann. In Abbildung 4.1b lässt sich allerdings erkennen, dass die Mittelkreiskandidaten auch überwiegend auf dem Mittelkreis detektiert werden können. Es sind einige wenige falsch positiv Detektionen der Kreisbogenkandidaten auf dem Arm des NAO-Roboters zu erkennen.



(a) *Liniendetektion* **(b)** *Kandidaten Kreisbogendetektion*

Abb. 4.1: *Liniendetektion und Detektion der Kreisbogenkandidaten während eines Spiels auf den GermanOpen2018 gegen die Nao Devils (Roboter mit gelben Trikot). Ein kurzes Kreisbogensegment wird fälschlicherweise als Linie detektiert, da die Klassifizierung als Kreisbogensegment fehlgeschlagen ist.*

Aber nicht nur die Verdeckung durch Objekte erschwert die Detektion von Kreisbögen. Natürlich vorkommende Begrenzungen eines Kamerabildes sorgen für Informationsdefizite, die zu Fehleinschätzungen von Kreisbögen führen. In Abbildung 4.2a sorgt eine zu große Abweichung zum vorherigen Gradienten für die Unterbrechung des Trackings. Aufgrund zu weit entfernter Alternativpunkte kann das Tracking auf dem Mittelkreis nicht wieder aufgenommen werden. Diese Tatsache spiegelt die bereits in Abschnitt 3.1.3 erwähnte Problematik mit fehlerhaften Gradienten wider. Abbildung 4.2b zeigt die berechneten Gradientenorientierungen an den Kandidatenpunkten der Liniendetektion. An der Stelle der Unterbrechung kann der Sprung und der fehlerhafte Gradient wahrgenommen werden.

(a) *Fehlerhafte Liniendetektion auf dem Mittelkreis.*

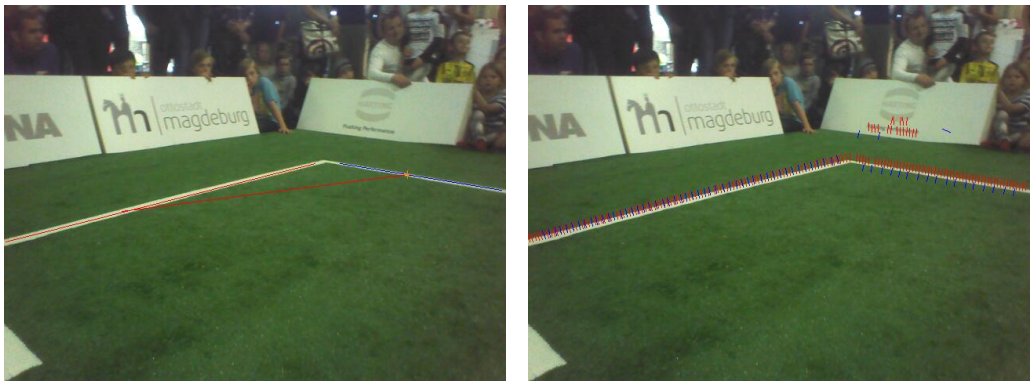
(b) *Berechnete Gradientenorientierungen auf dem Mittelkreis.*

Abb. 4.2: *Fehlerhafte Liniendetektion auf dem Mittelkreis und die dazugehörigen berechneten Gradientenorientierungen während eines Spiels gegen die Nao Devils auf den GermanOpen2018.*

Aus Gründen der Laufzeit wird bei dem Tracking (Algorithmus 3) ein Kandidatenpunkt an das erste passende Segment angefügt, anstatt das am besten übereinstimmende Segment zu suchen. Ein solches Vorgehen sorgt dafür, dass es nicht zu der erwarteten Zusammenfassung zu Segmenten entlang einer Linie kommen muss. Die Datenpunkte werden entsprechend der Reihenfolge der im Speicher liegenden Scanlinesegmente bearbeitet. So kommt es im Beispiel einer angehenden horizontalen Linie, bei vertikalen Scanlinesegmenten, zuerst zu der Überprüfung von Punkten kleinerer x-Koordinate. Können die Tests bestehen, führt dies zu der in Abbildung 4.3 dargestellten Fehldetektion.

Abb. 4.3: *Fehlende Priorisierung bei dem Tracking führt zu falscher Assoziierung der Linienkandidaten.*

Akkumuliert ein Segment, entsprechend Abschnitt 2.5, einen identifizierbaren Winkel und verstößt mehrfach gegen die Projektionsdistanz, so kann ein Segment als Kreisbogen identifiziert werden. Liniensegmente akkumulieren vergleichsweise sehr kleine Winkel. Der Fall eines kleinen akkumulierten Winkels und mehrfachen Projektionsdistanzverstößen wird allerdings nicht behandelt (siehe Algorithmus 4). Dies kann vorkommen, wenn der Übergang zwei aufeinander zulaufender Feldlinien, aufgrund ähnlicher Gradientenorientierungen, nicht detektiert wird. Ein solcher Umstand wird in Abbildung 4.4a aufgezeigt. Gleichzeitig ist zu erkennen, dass in diesem Fall, trotz der Fehldetektion, auch die korrekte Linie detektiert werden konnte. Möglich macht dies die separate Behandlung vertikaler und horizontaler Scanlinesegmente im Tracking.



(a) *Fehlerhafte Liniendetektion bei einer L-Kreuzung.*

(b) *Berechnete Gradientenorientierungen in einer L-Kreuzung.*

Abb. 4.4: *Fehlerhafte Liniendetektion bei einer L-Kreuzung und die dazugehörigen berechneten Gradientenorientierungen während eines Spiels gegen B-Human auf den GermanOpen2018. Die Gradientenorientierungen der Kandidatenpunkte sind, entsprechend der Herkunft von vertikalen oder horizontalen Scanlinesegmenten, farblich kodiert.*

Seltener treten aber auch falsch positiv Detektionen auf Robotern auf. Stimmt zuletzt auch noch die auf den Boden projizierte Ausrichtung eines auf dem Roboter liegenden Liniensegments, so kann es zu einer falsch positiven Linie kommen. Abbildung 4.5a zeigt eine solche Fehldetektion. Mehrfachfindungen einer einzigen Linie zählen auch als falsch positive Detektion, da in diesem Fall spätestens die Hough-Transformation die Liniensegmente hätte zusammenfassen müssen. In dem auf Abbildung 4.5b gezeigten Fall kommt es zu der besagten mehrfachen Detektionen der gleichen Linie. Die Linien werden nicht zusammengefasst, da der Abstand der beiden Liniensegmente eine maximale Distanz überschreitet. Die maximale Distanz berechnet sich relativ zu der Pixellänge der hinzuzufügenden Linie.

(a) *Falsch positiv Detektion auf einem Roboter (hier blaue Linie auf dem Arm).*

(b) *Mehrfachfindung der Mittellinie. Die Sicht wird durch andere Roboter unterbrochen.*

Abb. 4.5: *Unterschiedliche Falschdetektionen der Linien im Spiel gegen B-Human bei den GermanOpen2018.*

Die Auswertung beinhaltet außerdem die durchschnittlichen Längen der falsch positiv Detektionen, zum einen inklusive der Falschdetektionen auf Robotern $l_{FP,r}$ und zum anderen ausgenommen der Falschdetektionen auf Robotern l_{FP}. Die Übersicht dient der anschließenden Filterung und Gewichtung nachfolgender Schritte. Außerdem lässt sich eine Aussage über den Einfluss auf die Lokalisierung der Falschdetektion treffen, sofern längeren Linien ein größeres Vertrauen zugesprochen wird. Dies ist der Fall in dem Framework der HULKs.

Tab. 4.1: *Zusammenfassung der Ergebnisse über alle Spielsequenzen der im Rahmen dieser Arbeit entwickelten Liniendetektion.*

Kenngröße	Wert
Ground-Truth	2028
Vorhersagen	2433
Richtig Positive (TP)	1088
Falsch Negative (FN)	940
Falsch Positive (FP)	1345
davon Mehrfachvorhersagen	60
davon auf Robotern	191
Trefferquote (TPR)	0,54
Genauigkeit (PPV)	0,45
$\overline{\text{IoU}}$	0,55

Kenngröße	Wert
$\overline{l_{FP}}$ [px]	87,73
$\overline{l_{FP,r}}$ [px]	97,08
$\overline{l_{FP}}$ [m]	1,19
$\overline{l_{FP,r}}$ [m]	1,77

Vergleich zu bestehender Implementierung der HULKs

Im Vergleich zu der bereits vorhandenen Implementierung der HULKs, trifft die Liniendetektion der HULKs 74% mehr Vorhersagen auf den gleichen Spielsequenzen. Allerdings werden dabei nur 24% mehr Ground-Truth-Linien gefunden. Demzufolge ist die Anzahl der falsch Negativen auch nur geringfügig niedriger. Die richtig positiv Rate liegt bei $0,66$. So werden 34% der vorhandenen Linien übersehen. Hervorzuheben sind 114% mehr Falschdetektionen im Vergleich zu der im Rahmen dieser Arbeit entwickelten Liniendetektion. Aufgrund der hohen Anzahl getroffener Vorhersagen kann die Liniendetektion der HULKs nur eine Genauigkeit der Vorhersagen von $0,32$ aufweisen. Die IoU liegt bei 57% und unterscheidet sich demnach nur kaum von der IoU des Algorithmus dieser Arbeit.

Bedeutend sind die durchschnittlichen Längen der falsch positiv Detektionen. In der Pixellänge sind nur kleine Unterschiede festzustellen, doch unterscheiden sich die projizierten Längen gravierend. Eine falsch positiv Linie des Algorithmus dieser Arbeit ist im Schnitt doppelt so lang. Das hat den Hintergrund, dass die Liniendetektion der HULKs allein vertikale Scanlinesegmente verwendet. Vertikal verlaufende Linien können darüber nicht detektiert werden (LOTH, 2015). Demzufolge sind Kandidatenpunkte horizontal verlaufender Kanten überrepräsentiert. Dies hat zur Folge, dass auch die falsch positiv Detektionen vorzugsweise horizontal ausgerichtet sind. Aufgrund der Projektionseigenschaften resultieren jedoch vertikale Linien im Bildbereich in längere Linien in Roboterkoordinaten. Die im Rahmen dieser Arbeit entwickelten Liniendetektion kann Linien jeglicher Ausrichtung detektieren, so ist auch die durchschnittliche projizierte Länge eines falsch Positiven höher.

Die Ergebnisse der Spielsequenzen sind in Tabelle 4.2 zusammengefasst.

Tab. 4.2: *Zusammenfassung der Ergebnisse über alle Spielsequenzen der Liniendetektion der HULKs.*

Kenngröße	Wert
Ground-Truth	2028
Vorhersagen	4227

Kenngröße	Wert
Richtig Positive (TP)	1346
Falsch Negative (FN)	682
Falsch Positive (FP)	2881
davon Mehrfachvorhersagen	97
davon auf Robotern	698
Trefferquote (TPR)	0,66
Genauigkeit (PPV)	0,32
$\overline{\text{IoU}}$	0,58
$\overline{l_{FP}}$ [px]	79,55
$\overline{l_{FP,r}}$ [px]	99,12
$\overline{l_{FP}}$ [m]	0,55
$\overline{l_{FP,r}}$ [m]	0,82

4.1.2 Laufzeiten auf dem NAO-Robotiksystem

Das Fußballspiel erfordert die Echtzeitfähigkeit der verwendeten Algorithmen. Die HULKs arbeiten mit 30 Bildern pro Sekunde. Jedes Bild stellt einen Zyklus dar. Die Bilder der oberen und unteren Kamera werden näherungsweise abwechselnd bearbeitet. In jedem Zyklus werden zuerst die Module der visuellen Wahrnehmung ausgeführt. Die Ergebnisse werden anschließend gefiltert und weiter verarbeitet. Anhand der berechneten Ergebnisse werden Entscheidungen getroffen und entsprechende Befehle an die Motoren gegeben. Entsprechend der eintreffenden Bilder darf somit ein Zyklus nicht länger als ungefähr 33 ms dauern. Dies beschränkt den Einsatz komplexer und aufwendiger Algorithmen.

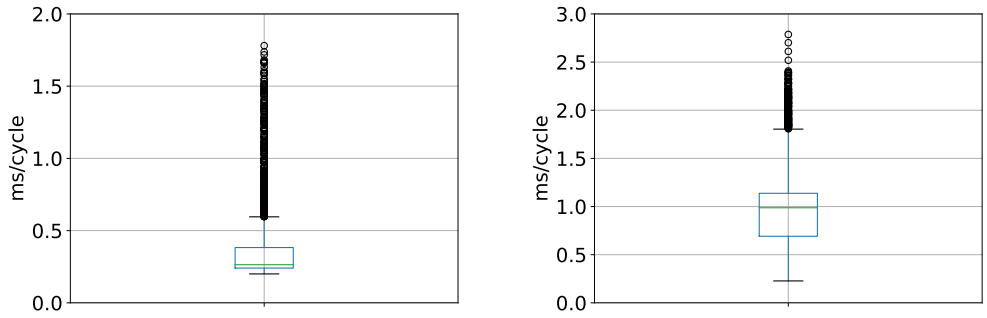
Für die Laufzeitanalyse wird der Intel VTune Amplifier 2018 verwendet. Der Einsatz im Framework der HULKs ist in (RIEBESEL, 2018, S. 20) erläutert. Die Software wurde außerdem dazu verwendet kritische Pfade des Algorithmus zu detektieren, um gezielt Optimierungen vorzunehmen.

Es wurde eine Analyse über knapp 10000 Zyklen durchgeführt. Die durchschnittlichen Laufzeiten des Algorithmus unterscheiden sich je nach verwendeter Kamera, da sich die Feinheit der Bildsegmentierung unterscheidet. Die Segmente stellen den Eingangsvektor für die Liniendetektion dar. Die durchschnittliche Laufzeit beträgt auf der oberen Kamera $0,43$ ms und auf der unteren Kamera etwa 1 ms (Tabelle 4.3).

Tab. 4.3: *Durchschnittliche Laufzeiten und Anzahl analysierter Zyklen der im Rahmen dieser Arbeit entwickelten Liniendetektion.*

Kamera	Zyklen	Durchschnittslaufzeit
Oben	4978	0,425371 ms
Unten	4978	1,053018 ms

Abbildung 4.6 zeigt die zu der Analyse gehörenden Box-Whisker-Plots über die Laufzeit eines jeden Zyklus in ms. Die Box entspricht den mittleren 50% der Messwerte, welche sich bei der oberen Kamera deutlich unter einer halben Millisekunde befinden. Insbesondere handelt es sich um eine sehr schiefe Verteilung. Der Median, markiert durch den grünen Strich innerhalb der Box, befindet sich nah an dem unteren Quartil. Die unteren 50% der Datenpunkte befinden sich somit gehäuft bei etwa $0,25$ ms. Die Whisker sind über den 1,5-Fachen Interquartilsabstand begrenzt. Ausreißer sind vorhanden und reichen im Ausnahmefall bis etwa $1,8$ ms. Die Laufzeit auf der unteren Kamera beträgt etwa das Vierfache, aufgrund der Vielzahl der zu evaluierenden Segmente. Das obere Quartil befindet sich unter 2 ms und Ausreißer reichen bis $2,8$ ms. Der Algorithmus ist damit in einer echtzeitfähigen Umgebung einsetzbar.



(a) *Obere Kamera* **(b)** *Untere Kamera*

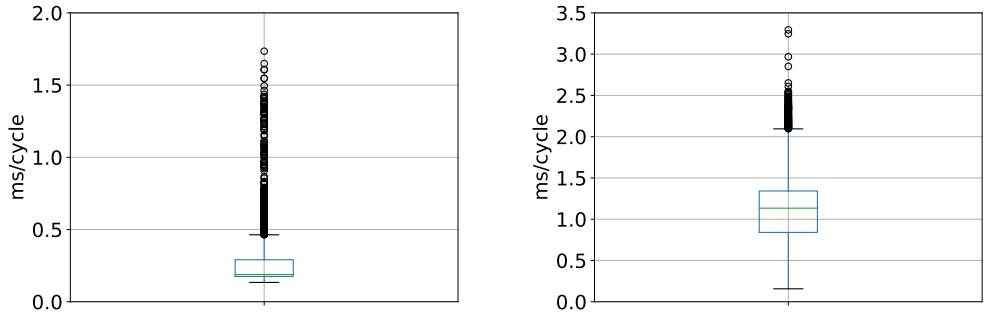
Abb. 4.6: *Box-Whisker-Plots über die Laufzeit in ms pro Zyklus der im Rahmen dieser Arbeit entwickelten Liniendetektion. Die Länge der Whisker wurde auf den 1,5-Fachen Interquartilsabstand begrenzt. Es wird zwischen der oberen und unteren Kamera unterschieden.*

Vergleich zu bestehender Implementierung der HULKs

Die Laufzeit unterscheidet sich zu der Liniendetektion der HULKs nur minimal. Auf den ersten Blick scheint die Durchschnittslaufzeit der oberen Kamera geringer (Tabelle 4.4), doch zeigen die Box-Plots in Abbildung 4.7, dass sich die mittleren 50% der Messdaten auch bei $0,25$ ms befinden. Die Ausführungszeit auf der oberen Kamera ist etwas langsamer und die Ausreißer reichen bis $3,3$ ms. Beide Algorithmen befinden sich bezüglich ihrer Laufzeiten auf dem gleichen Niveau.

Tab. 4.4: *Durchschnittliche Laufzeiten und Anzahl analysierter Zyklen der Liniendetektion der HULKs.*

Kamera	Zyklen	Durchschnittslaufzeit
Oben	4997	0,281864 ms
Unten	4997	1,179578 ms



(a) *Obere Kamera* **(b)** *Untere Kamera*

Abb. 4.7: *Box-Whisker-Plots über die Laufzeit in ms pro Zyklus der Liniendetektion der HULKs. Die Länge der Whisker wurde auf den 1,5-Fachen Interquartilsabstand begrenzt. Es wird zwischen der oberen und unteren Kamera unterschieden.*

4.2 Strafstoßmarkendetektion

Die Auswertung der Strafstoßmarkendetektion folgt Abschnitt 4.1. Der Datensatz enthält die Ground-Truth-Label der Strafstoßmarken in Form von Bounding-Boxen. Über die im Algorithmus detektierte Breite und Höhe der Strafstoßmarke lässt sich auch eine entsprechende Bounding-Box berechnen. Für die Auswertung wird auch in diesem Fall

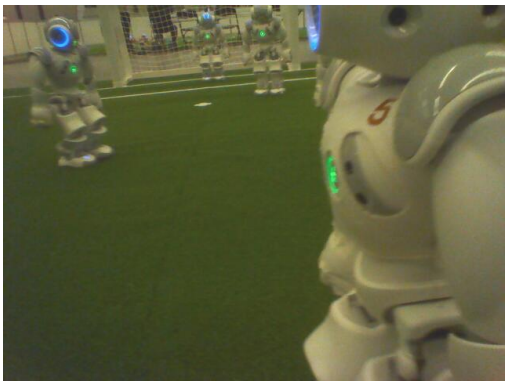
die IoU verwendet. Eine Strafstoßmarke gilt als detektiert, wenn die IoU einen Wert von 20% aufweist.

4.2.1 Ergebnisse auf Spielsequenzen

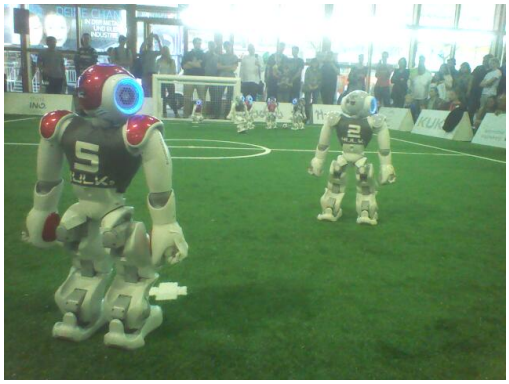
Die Spielsequenzen enthalten 334 Ground-Truth-Strafstoßmarken. Der im Rahmen dieser Arbeit entwickelte Algorithmus trifft 140 Vorhersagen. Darunter befinden sich 140 richtige Vorhersagen. Damit liegt die Genauigkeit bei 1. Die durchschnittliche IoU beläuft sich auf knapp 70%. Die Detektion ist allerdings sehr restriktiv eingestellt, daher kommt der Algorithmus nur auf eine Trefferquote von $0,41$. Damit werden 59% der vorhanden Strafstoßmarken übersehen.

Hintergrund der hohen Fehlquote sind jegliche Strafstoßmarken, die nicht vollständig zu sehen sind. Liegt die Markierung am Bildrand, steht oder liegt ein Roboter zum Teil auf der Markierung oder kommt es zu einer anderen Art von Verdeckung, kann die Strafstoßmarke nicht mehr detektiert werden. Die Detektion ist, zur Vorbeugung von falsch positiv Detektionen, künstlich auf eine maximale Detektionsdistanz von 3 m beschränkt. In Abbildung 4.8 sind unterschiedliche Arten nicht detektierbarer Strafstoßmarken dargestellt. Abbildung 4.8a zeigt eine teilweise verdeckte Strafstoßmarke. Die Umgebung der Marke kann nicht ausreichend überprüft werden. Dies sorgt für den sofortigen Abbruch der Überprüfung des Kandidaten. In Abbildung 4.8b ist der Strafstoßpunkt zu weit entfernt. In Abbildung 4.8c ist die Marke zwar vollständig sichtbar, doch kann auch in diesem Fall die Umgebung nicht ausreichend überprüft werden. Strafstoßmarken, wie in Abbildung 4.8d, können nicht detektiert werden, da bereits zu Anfang entsprechende Scanlinesegmente ausgeschlossen werden und auch weitere Überprüfungen in so einem Fall fehlschlagen.

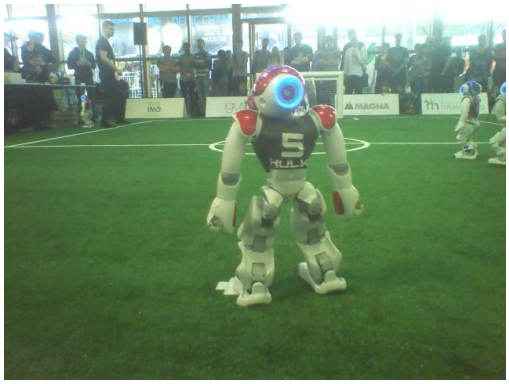
(a) *Teilweise Verdeckung durch den Bildrand*



(b) *Zu große Entfernung*



(c) *Teilweise Verdeckung durch Roboter*



(d) *Verdeckung durch den Fuß eines Roboters*

Abb. 4.8: *Falsch Negativbeispiele bei der Strafstoßmarkendetektion. Die Kamerabilder stammen aus den Spielsequenzen, welche für die Auswertung verwendet wurden.*

Außerdem stellt sich heraus, dass unter manchen Lichtbedingungen die Annährung der Farbigkeit, wie in Abschnitt 3.3.2 beschrieben, zu keinen gewünschten Ergebnissen führt. Dies ist vor Spielbeginn zu untersuchen und gegebenenfalls durch die Überprüfung der Feldfarbe zu ersetzen (siehe Algorithmus 14).

Die Ergebnisse der Spielsequenzen sind in Tabelle 4.5 zusammengefasst.

Tab. 4.5: *Zusammenfassung der Ergebnisse über alle Spielsequenzen der im Rahmen dieser Arbeit entwickelten Strafstoßmarkendetektion.*

Kenngröße	Wert
Ground-Truth	334

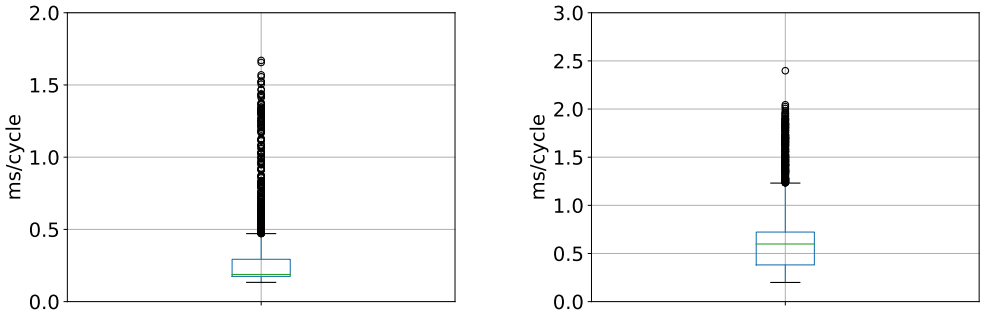
Kenngröße	Wert
Vorhersagen	140
Richtig Positive (TP)	140
Falsch Negative (FN)	195
Falsch Positive (FP)	0
Trefferquote (TPR)	0,42
Genauigkeit (PPV)	1
$\overline{\text{IoU}}$	0,68

4.2.2 Laufzeiten auf dem NAO-Robotiksystem

Die durchschnittliche Laufzeit der Strafstoßmarkendetektion liegt für die obere Kamera bei $0,28$ ms (Tabelle 4.6) bei einem Median von unter $0,25$ ms. Auch das obere Quartil der Messungen übersteigt nicht die $0,5$ ms. Einzelne Ausreißer treten bis zu $1,74$ ms auf. Die Ausführungszeit ist auf der unteren Kamera aus bereits genannten Gründen langsamer, aber bewegt sich mit den mittleren 50% der Messungen bei weit unter 1 ms (Abbildung 4.9). Der Algorithmus ist in einer Echtzeitumgebung einsetzbar.

Tab. 4.6: *Durchschnittliche Laufzeiten und Anzahl analysierter Zyklen der im Rahmen dieser Arbeit entwickelten Strafstoßmarkendetektion.*

Kamera	Zyklen	Durchschnittslaufzeit
Oben	4978	0,282643 ms
Unten	4978	0,636189 ms



(a) *Obere Kamera* **(b)** *Untere Kamera*

Abb. 4.9: *Box-Whisker-Plots über die Laufzeit in ms pro Zyklus der im Rahmen dieser Arbeit entwickelten Strafstoßmarkendetektion. Die Länge der Whisker wurde auf den 1,5-Fachen Interquartilsabstand begrenzt. Es wird zwischen der oberen und unteren Kamera unterschieden.*

Kapitel 5

Zusammenfassung und Ausblick

Es wurden Algorithmen zur Detektion von Feldlinien und Strafstoßmarken für das NAO-Robotiksystem entwickelt. Der Feldlinienalgorithmus umfasst zusätzlich noch die Detektion und Klassifizierung der auf dem Spielfeld vorkommender Kreuzungspunkte und eine Kandidatengenerierung für eine Mittelkreisdetektion. Es wurde auf Probleme bei der Detektion eingegangen und der Linienalgorithmus mit dem der HULKs verglichen. Die behandelten Algorithmen wurden auf Spielsequenzen der letzten RoboCups evaluiert und hinsichtlich ihrer Echtzeitfähigkeit überprüft.

Die im Rahmen dieser Arbeit entwickelten Liniendetektion stellt sich unter Betrachtung der richtig positiv Rate und der Genauigkeit im Vergleich zu dem Algorithmus der HULKs als überlegen heraus. Es werden deutlich weniger falsche Vorhersagen getroffen. Insbesondere konnten die falsch positiv Detektionen auf Robotern um über 70% reduziert werden. Trotzdem ist die Trefferquote und Genauigkeit weiter zu erhöhen. In der Laufzeitanalyse stellte sich heraus, dass der Algorithmus in einer Echtzeitumgebung eingesetzt werden kann. Aus Laufzeitgründen wurde in dem Tracking bisher auf die Berechnung der Varianz der Segmentkoordinaten verzichtet, da dies eine ständige Neuberechnung erfordert. Eine Annäherung an die Varianz kann sich als ein wertvolles Maß im Tracking herausstellen. Die in (KNUTH, 1997, S. 232) vorgestellte Annäherung lässt sich insbesondere iterativ aktualisieren und gilt zu evaluieren. Außerdem können Fehler im Tracking behoben werden, wenn ein Punkt an das am besten passende Segment angefügt wird und nicht an das erstbeste. Damit die Suche nach dem passenden Segment minimiert werden kann, sollte beispielsweise die Verwendung von Binärbäumen angestrebt werden. Eine Baumstruktur bietet sich auch für die Suche im Parameterraum an (XU u. a., 1990, S. 2). Kann die Mittelkreisdetektion von (RIEBESEL, 2018) auf den Kreisbogenkandidaten dieser Arbeit erfolgreiche Ergebnisse erzielen, lassen sich falsch positiv Detektionen auf dem Mittelkreis vermeiden. Vielversprechend ist auch die Verkettung mehrfacher Iterationen der RHT (KÄLVIÄINEN u. a., 1994, S. 3). Entgegen der

Methode in zweiter Iteration mit einem leeren und höher aufgelösten Akkumulator zu starten, kann der Akkumulator mit den Parametern konfidenter Linien initialisiert und begrenzt werden. Andere Liniensegmente können somit nur mit Linien höherer Sicherheit kombiniert werden. Mit dieser Methode können weitere FP und Unterbrechungen der Linien vermieden werden.

Die Strafstoßmarkendetektion weist in der Evaluation keine einzige falsch positiv Detektion auf. Der Algorithmus ist dafür aber sehr restriktiv und kommt somit nur auf eine TPR von knapp über $0,4$. Mit der vorgestellten Methode ist es nicht möglich teils verdeckte Strafstoßpunkte zu detektieren. Konturbasierte Ansätze könnten eine Klassifizierung anhand einer teilweise sichtbaren Marke treffen (RÖFER u. a., 2017, S. 72). Mit Laufzeiten unter einer Millisekunde auf dem NAO-Roboter kann der Algorithmus echtzeitfähig eingesetzt werden.

Das implementierte Auswertungsframework erlaubt in Zukunft die Verwendung eines Optimierungsverfahrens zur Findung optimaler Konfigurationsparameter für die Linien- und Strafstoßmarkendetektion.

Abkürzungsverzeichnis

RoboCup Robot World Cup Initiative v

HULKs Hamburg Ultra Legendary Kickers v

SPL Standard Platform League 1

UKF Unscented Kalman-Filter 4

HT Hough-Transformation 10

RHT Randomisierte Hough-Transformation 12

CRHT Connective Randomized Hough Transform 19

WRHT Window Randomized Hough Transform 13

RANSAC Random sample consensus 13

IoU Intersection over Union 41

Literatur

ADIKARI, DARSHANA ; FELBINGER, GEORG ; HASSELBRING, ARNE ; KONDA, YURIA ; KOST, RENE ; LOTH, PASCAL ; PETERS, LASSE ; RIEBESEL, NICOLAS ; U. A.: HULKs Team Research Report 2017 (2017)

ALDEBARAN ROBOTICS: *2D Camera*. URL `http://doc.aldebaran.com/2-1/family/robots/video_robot.html`

BALLARD, DANA H: Generalizing the Hough transform to detect arbitrary shapes. In: *Pattern recognition* Bd. 13, Elsevier (1981), Nr. 2, S. 111–122

CANNY, JOHN: A computational approach to edge detection. In: *IEEE Transactions on pattern analysis and machine intelligence*, Ieee (1986), Nr. 6, S. 679–698

CHOI, SUNGLOK ; KIM, TAEMIN ; YU, WONPIL: Performance evaluation of RANSAC family. In: *Journal of Computer Vision* Bd. 24 (1997), Nr. 3, S. 271–300

EVERINGHAM, MARK ; VAN GOOL, LUC ; WILLIAMS, CHRISTOPHER KI ; WINN, JOHN ; ZISSERMAN, ANDREW: The pascal visual object classes (voc) challenge. In: *International journal of computer vision* Bd. 88, Springer (2010), Nr. 2, S. 303–338

FELBINGER, GEORG CHRISTIAN: *A genetic approach to design convolutional neural networks for the purpose of a ball detection on the NAO robotic system*, Projektarbeit, 2017

FELBINGER, GEORG CHRISTIAN: *Optimal CNN Hyperparameters for Object Detection on NAO Robots*, Masterarbeit, 2018

FISCHLER, MARTIN A ; BOLLES, ROBERT C: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In: *Communications of the ACM* Bd. 24, ACM (1981), Nr. 6, S. 381–395

HARTLEY, R. I. ; ZISSERMAN, A.: *Multiple View Geometry in Computer Vision*. Second. Aufl. : Cambridge University Press, ISBN: 0521540518, 2004 – MaSSstab

HOUGH, PAUL VC: Method and means for recognizing complex patterns.

ILLINGWORTH, JOHN ; KITTLER, JOSEF: A survey of the Hough transform. In: *Computer vision, graphics, and image processing* Bd. 44, Elsevier (1988), Nr. 1, S. 87–116

JAIN, ANIL K.: *Fundamentals of Digital Image Processing*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1989 – MaSSstab — ISBN 0-13-336165-9

KALVIAINEN, HEIKKI ; HIRVONEN, PETRI ; XU, LEI ; OJA, ERKKI: Probabilistic and non-probabilistic Hough transforms: overview and comparisons. In: *Image and vision computing* Bd. 13, Guildford, Surrey: Butterworths, c1983- (1995), Nr. 4, S. 239–252

KÄLVIÄINEN, HEIKKI ; HIRVONEN, PETRI ; XU, LEI ; OJA, ERKKI: Comparisons of probabilistic and non-probabilistic Hough transforms. In: *European Conference on Computer Vision* : Springer, 1994, S. 350–360

KASS, MICHAEL ; WITKIN, ANDREW ; TERZOPOULOS, DEMETRI: Snakes: Active contour models. In: *International journal of computer vision* Bd. 1, Springer (1988), Nr. 4, S. 321–331

KNUTH, D: The Art of Computer Programming. Volume 2 (Seminumerical Algorithms). 3. Ausg.

LAUE, TIM ; SPIESS, KAI ; RÖFER, THOMAS: SimRobot–a general physical robot simulator and its application in robocup. In: *Robot Soccer World Cup* : Springer, 2005, S. 173–183

LOTH, PASCAL: *Implementierung und Evaluation einer robusten Echtzeitkantendetektion auf dem humanoiden NAO-Robotiksystem*, Bachelorarbeit, 2015

NAO-TEAM HTWK: Team Research Report 2014 (2014)

NAO-TEAM HTWK: *HTWK 3:1 HULKs*. URL `https://naoteamhtwk.blogspot.com/2017/07/htwk-31-hulks.html`

NGUYEN, THANH PHUONG ; DEBLED-RENNESSON, ISABELLE: Decomposition of a curve into arcs and line segments based on dominant point detection. In: *Scandinavian Conference on Image Analysis* : Springer, 2011, S. 794–805

NIEMUELLER, TIM ; LAKEMEYER, GERHARD ; FERREIN, ALEXANDER: The RoboCup logistics league as a benchmark for planning in robotics. In: *WS on planning and robotics (PlanRob) at Int. Conf. on Aut. planning and scheduling (ICAPS)*, 2015

OSAWA, EIICHI ; KITANO, HIROAKI ; ASADA, MINORU ; KUNIYOSHI, YASUO ; NODA, ITSUKI: RoboCup: the robot world cup initiative. In:,

PETERS, LASSE: *Adaption und Vergleich von nichtlinearen Filtermethoden zur Selbstlokalisierung auf einem Feld mit dem humanoiden NAO-Robotiksystem*, Bachelorarbeit, 2018

RAGURAM, RAHUL ; FRAHM, JAN-MICHAEL ; POLLEFEYS, MARC: A comparative analysis of RANSAC techniques leading to adaptive real-time random sample consensus. In: *European Conference on Computer Vision* : Springer, 2008, S. 500–513

REINHARDT, THOMAS: *Kalibrierungsfreie Bildverarbeitungsalgorithmen zur echtzeitfähigen Objekterkennung im Roboterfußball*, Masterarbeit, 2011

RIEBESEL, NICOLAS: *Center circle detection on the NAO robotic platform for the RoboCup Standard Platform League*, Projektarbeit, 2018

ROBOCUP FEDERATION: *RoboCupSoccer - Simulation 2D*. URL `http://robocup.org/leagues/24`

ROBOCUP TECHNICAL COMMITTEE: RoboCup Standard Platform League (NAO) Rule Book: 2018 rules, as of May 14, 2018 (2018)

RÖFER, THOMAS ; LAUE, TIM ; BÜLTER, YANNICK ; KRAUSE, DANIEL ; KUBALL, JONAS ; MÜHLENBROCK, ANDRE ; POPPINGA, BERND ; PRINZLER, MARKUS ; U. A.: B-Human Team Report and Code Release 2017.

RÖFER, THOMAS ; LAUE, TIM ; MÜLLER, JUDITH ; BURCHARDT, ARMIN ; DAMROSE, ERIK ; FABISCH, ALEXANDER ; FELDPAUSCH, FYNN ; GILLMANN, KATHARINA ; U. A.: B-Human Team Report and Code Release 2010.

SAYLOR, RICK D ; EDGERTON, ERIC S ; HARTSELL, BENJAMIN E: Linear regression techniques for use in the EC tracer method of secondary organic aerosol estimation. In: *Atmospheric Environment* Bd. 40, Elsevier (2006), Nr. 39, S. 7546–7556

SHEH, RAYMOND ; SCHWERTFEGER, SÖREN ; VISSER, ARNOUD: 16 Years of RoboCup Rescue. In: *KI - Künstliche Intelligenz* Bd. 30 (2016), Nr. 3, S. 267–277

SKINNER, CAMERON ; RAMCHURN, SARVAPALI: The robocup rescue simulation platform. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1* : International Foundation for Autonomous Agents; Multiagent Systems, 2010, S. 1647–1648

SOFTBANK ROBOTICS EUROPE: *NAO Humanoid Robot Platform Datasheet*. URL `https://www.ald.softbankrobotics.com/sites/aldebaran/files/datasheet_nao_next_gen_en.pdf`

SOFTBANK ROBOTICS EUROPE: *NAO v6 - Overview*. URL `http://doc.aldebaran.com/2-8/`

SOFTBANK ROBOTICS EUROPE: *Who is NAO?* URL `https://www.ald.softbankrobotics.com/en/robots/nao`

WANG, YUE ; SHEN, DINGGANG ; TEOH, EAM KHWANG: Lane detection using spline model. In: *Pattern Recognition Letters* Bd. 21, Elsevier (2000), Nr. 8, S. 677–689

WANG, YUE ; TEOH, EAM KHWANG ; SHEN, DINGGANG: Lane detection and tracking using B-Snake. In: *Image and Vision computing* Bd. 22, Elsevier (2004), Nr. 4, S. 269–280

WISSPEINTNER, THOMAS ; VAN DER ZANT, TIJN ; IOCCHI, LUCA ; SCHIFFER, STEFAN: RoboCup@Home Scientific competition and benchmarking for domestic service robots. In: *Interaction Studies* Bd. 10, John Benjamins Publishing Company (2009), Nr. 3, S. 392–426

XU, LEI ; OJA, ERKKI: Randomized hough transform. In: *Encyclopedia of Artificial Intelligence* : IGI Global, 2009, S. 1343–1350

XU, LEI ; OJA, ERKKI ; KULTANEN, PEKKA: A new curve detection method: randomized Hough transform (RHT). In: *Pattern recognition letters* Bd. 11, Elsevier (1990), Nr. 5, S. 331–338

Anhang A

Detektionsbeispiele

A.1 Liniendetektion

(a) *Aufgrund der Detektion der Kreisbogenkandidaten finden sich keine falsch positiv Detektionen von Linien auf dem Mittelkreis.*

(b) *Liniendetektion im Bereich des Strafraumes.*

(c) *Liniendetektion auf einen Strafraumausschnitt. Fehldetektion durch nicht detektierte Kante in der L-Kreuzung.*

(d) *Liniendetektion im Bereich des Strafraumes.*

Abb. A.1: *Liniendetektion und Klassifizierung der Eckpunkte im Labor der HULKs. L-Kreuzungen sind gelb, T-Kreuzungen orange markiert.*

(a) *Liniendetektion in der Angriffssituation des Roboters.*

(b) *Erfolgreiche Liniendetektion trotz herumliegender Roboter und sichtbaren Mittelkreis.*

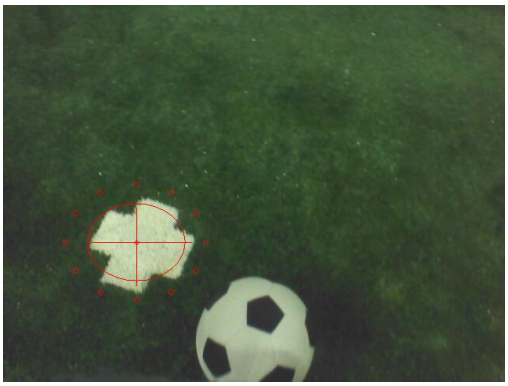
Abb. A.2: *Liniendetektion und Klassifizierung der Eckpunkte im Labor der HULKs. L-Kreuzungen sind gelb, T-Kreuzungen orange markiert. Es sind keine falsch positiv Detektionen auf Robotern vorhanden.*

A.2 Strafstoßmarkendetektion

(a) *Strafstoßmarkendetektion eines Verteidigers.*

(b) *Strafstoßmarkendetektion kurz vor dem Schuss auf das Tor der B-Human.*

(c) *Strafstoßmarkendetektion in der gegnerischen Hälfte unseres Stürmers.*



(d) *Strafstoßmarkendetektion auf der unteren Kamera während des Dribbelns.*

Abb. A.3: *Strafstoßmarkendetektion bei den GermanOpen2018 in verschiedenen Spielen gegen B-Human und den Nao Devils.*

(a) *Erfolgreiche Strafstoßmarkendetektion auf der unteren Kamera trotz starker Lichteinstrahlung.*

(b) *Erfolgreiche Strafstoßmarkendetektion während andere Roboter diesen für einen Ball halten.*

Abb. A.4: *Strafstoßmarkendetektion bei dem RoboCup2018 auf dem Outdoor-Feld unter starken Schlaglichtbedingungen.*

Anhang B

Detaillierte Ergebnisse der Spielsequenzen

B.1 Liniendetektion

IranOpen2018 - HULKs : Nao-Team HTWK

Tab. B.1: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Liniendetektion und die der HULKs auf der Spielsequenz aus dem Spiel HULKs : Nao-Team HTWK bei den IranOpen2018.*

Kenngröße	Liniendetektion	$\text{Liniendetektion}_{\text{HULKs}}$
Ground-Truth	13	13
Vorhersagen	20	81
Richtig Positive (TP)	3	4
Falsch Negative (FN)	10	9
Falsch Positive (FP)	17	77
davon Mehrfachvorhersagen	1	0
davon auf Robotern	0	26
Trefferquote (TPR)	0,230769	0,307692
Genauigkeit (PPV)	0,15	0,049383
$\overline{\text{IoU}}$	0,480056	0,460227
$\overline{l_{FP}}$ [px]	106,118	67,4026
$\overline{l_{FP,r}}$ [px]	106,118	88,1818

Kenngröße	Liniendetektion	Liniendetektion$_{\text{HULKs}}$
$\overline{l_{FP}}$ [m]	0,332347	0,313429
$\overline{l_{FP,r}}$ [m]	0,332347	0,47896

IranOpen2018 - Dutch Nao Team : HULKs

Tab. B.2: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Liniendetektion und die der HULKs auf der Spielsequenz aus dem Spiel Dutch Nao Team : HULKs bei den IranOpen2018.*

Kenngröße	Liniendetektion	Liniendetektion$_{\text{HULKs}}$
Ground-Truth	147	147
Vorhersagen	331	574
Richtig Positive (TP)	82	126
Falsch Negative (FN)	65	21
Falsch Positive (FP)	249	448
davon Mehrfachvorhersagen	2	35
davon auf Robotern	2	12
Trefferquote (TPR)	0,557823	0,857143
Genauigkeit (PPV)	0,247734	0,219512
$\overline{\text{IoU}}$	0,439888	0,553918
$\overline{l_{FP}}$ [px]	119,378	98,0647
$\overline{l_{FP,r}}$ [px]	119,892	99,846
$\overline{l_{FP}}$ [m]	0,126677	0,203012
$\overline{l_{FP,r}}$ [m]	0,127154	0,208009

GermanOpen2018 - HULKs : B-Human

Tab. B.3: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Liniendetektion und die der HULKs auf der Spielsequenz aus dem Spiel HULKs : B-Human bei den GermanOpen2018.*

Kenngröße	Liniendetektion	Liniendetektion$_{\text{HULKs}}$
Ground-Truth	52	52
Vorhersagen	68	117

Kenngröße	Liniendetektion	Liniendetektion$_{\text{HULKs}}$
Richtig Positive (TP)	27	39
Falsch Negative (FN)	25	13
Falsch Positive (FP)	41	78
davon Mehrfachvorhersagen	0	0
davon auf Robotern	6	28
Trefferquote (TPR)	0,519231	0,75
Genauigkeit (PPV)	0,397059	0,333333
$\overline{\text{IoU}}$	0,530375	0,595519
$\overline{l_{FP}}$ [px]	84,6098	70,5
$\overline{l_{FP,r}}$ [px]	93,2439	97,2436
$\overline{l_{FP}}$ [m]	1,40697	0,436864
$\overline{l_{FP,r}}$ [m]	2,59279	0,528263

GermanOpen2018 - HULKs : Nao Devils

Tab. B.4: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Liniendetektion und die der HULKs auf der Spielsequenz aus dem Spiel HULKs : Nao Devils bei den GermanOpen2018.*

Kenngröße	Liniendetektion	Liniendetektion$_{\text{HULKs}}$
Ground-Truth	281	281
Vorhersagen	375	549
Richtig Positive (TP)	191	193
Falsch Negative (FN)	90	88
Falsch Positive (FP)	184	356
davon Mehrfachvorhersagen	13	20
davon auf Robotern	29	77
Trefferquote (TPR)	0,679715	0,686833
Genauigkeit (PPV)	0,509333	0,351548
$\overline{\text{IoU}}$	0,622032	0,633861
$\overline{l_{FP}}$ [px]	81,5489	93,0393
$\overline{l_{FP,r}}$ [px]	93,4293	109,784
$\overline{l_{FP}}$ [m]	1,91155	0,907899

Kenngröße	Liniendetektion	$\text{Liniendetektion}_{\text{HULKs}}$
$\overline{l_{FP,r}}$ [m]	2,05744	1,02074

GermanOpen2018 - NomadZ : HULKs

Tab. B.5: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Liniendetektion und die der HULKs auf der Spielsequenz aus dem Spiel NomadZ : HULKs bei den GermanOpen2018.*

Kenngröße	Liniendetektion	$\text{Liniendetektion}_{\text{HULKs}}$
Ground-Truth	428	428
Vorhersagen	382	800
Richtig Positive (TP)	117	248
Falsch Negative (FN)	311	180
Falsch Positive (FP)	265	552
davon Mehrfachvorhersagen	10	21
davon auf Robotern	48	87
Trefferquote (TPR)	0,273364	0,579439
Genauigkeit (PPV)	0,306283	0,31
$\overline{\text{IoU}}$	0,51094	0,533826
$\overline{l_{FP}}$ [px]	81,2792	97,5707
$\overline{l_{FP,r}}$ [px]	92,4377	110,627
$\overline{l_{FP}}$ [m]	0,319117	0,755441
$\overline{l_{FP,r}}$ [m]	0,45229	1,98172

GermanOpen2018 - B-Human : HULKs

Tab. B.6: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Liniendetektion und die der HULKs auf der Spielsequenz aus dem Spiel B-Human : HULKs bei den GermanOpen2018.*

Kenngröße	Liniendetektion	$\text{Liniendetektion}_{\text{HULKs}}$
Ground-Truth	307	307
Vorhersagen	314	407
Richtig Positive (TP)	154	165

Kenngröße	Liniendetektion	Liniendetektion$_{\text{HULKs}}$
Falsch Negative (FN)	153	142
Falsch Positive (FP)	160	242
davon Mehrfachvorhersagen	4	4
davon auf Robotern	31	50
Trefferquote (TPR)	0,501629	0,537459
Genauigkeit (PPV)	0,490446	0,405405
$\overline{\text{IoU}}$	0,630623	0,593845
$\overline{l_{FP}}$ [px]	87,1937	98,5
$\overline{l_{FP,r}}$ [px]	98,4	113,409
$\overline{l_{FP}}$ [m]	5,25156	1,00434
$\overline{l_{FP,r}}$ [m]	6,86065	1,34723

RoboCup2018 - HULKs : UT Austin Villa

Tab. B.7: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Liniendetektion und die der HULKs auf der Spielsequenz aus dem Spiel HULKs : UT Austin Villa bei dem RoboCup2018.*

Kenngröße	Liniendetektion	Liniendetektion$_{\text{HULKs}}$
Ground-Truth	54	54
Vorhersagen	62	144
Richtig Positive (TP)	31	40
Falsch Negative (FN)	23	14
Falsch Positive (FP)	31	104
davon Mehrfachvorhersagen	1	3
davon auf Robotern	6	48
Trefferquote (TPR)	0,574074	0,740741
Genauigkeit (PPV)	0,5	0,277778
$\overline{\text{IoU}}$	0,62315	0,626492
$\overline{l_{FP}}$ [px]	84,6452	53,5192
$\overline{l_{FP,r}}$ [px]	102,258	87,6635
$\overline{l_{FP}}$ [m]	0,251686	0,254726
$\overline{l_{FP,r}}$ [m]	0,278171	0,456431

RoboCup2018 - HULKs : B-Human

Tab. B.8: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Liniendetektion und die der HULKs auf der Spielsequenz aus dem Spiel HULKs : B-Human bei dem RoboCup2018.*

Kenngröße	Liniendetektion	$\text{Liniendetektion}_{\text{HULKs}}$
Ground-Truth	414	414
Vorhersagen	519	1042
Richtig Positive (TP)	279	285
Falsch Negative (FN)	135	129
Falsch Positive (FP)	240	757
davon Mehrfachvorhersagen	23	8
davon auf Robotern	42	280
Trefferquote (TPR)	0,673913	0,688406
Genauigkeit (PPV)	0,537572	0,273512
$\overline{\text{IoU}}$	0,588475	0,591737
$\overline{l_{FP}}$ [px]	72,5125	67,8045
$\overline{l_{FP,r}}$ [px]	85,275	93,5773
$\overline{l_{FP}}$ [m]	0,360962	0,324042
$\overline{l_{FP,r}}$ [m]	0,75528	0,446889

RoboCup2018 - rUNSWift : HULKs

Tab. B.9: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Liniendetektion und die der HULKs auf der Spielsequenz aus dem Spiel rUNSWift : HULKs bei dem RoboCup2018.*

Kenngröße	Liniendetektion	$\text{Liniendetektion}_{\text{HULKs}}$
Ground-Truth	332	332
Vorhersagen	362	513
Richtig Positive (TP)	204	246
Falsch Negative (FN)	128	86
Falsch Positive (FP)	158	267
davon Mehrfachvorhersagen	6	6
davon auf Robotern	27	90

Kenngröße	Liniendetektion	$\text{Liniendetektion}_{\text{HULKs}}$
Trefferquote (TPR)	0,614458	0,740964
Genauigkeit (PPV)	0,563536	0,479532
$\overline{\text{IoU}}$	0,55434	0,593343
$\overline{l_{FP}}$ [px]	72,3228	69,5431
$\overline{l_{FP,r}}$ [px]	82,7089	91,7603
$\overline{l_{FP}}$ [m]	0,724445	0,714995
$\overline{l_{FP,r}}$ [m]	2,49831	0,919859

B.2 Strafstoßmarkendetektion

IranOpen2018 - HULKs : Nao-Team HTWK

Tab. B.10: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Strafstoßmarkendetektion auf der Spielsequenz aus dem Spiel HULKs : Nao-Team HTWK bei den IranOpen2018.*

Kenngröße	Wert
Ground-Truth	1
Vorhersagen	0
Richtig Positive (TP)	0
Falsch Negative (FN)	1
Falsch Positive (FP)	0
Trefferquote (TPR)	0
Genauigkeit (PPV)	N/A
$\overline{\text{IoU}}$	N/A

IranOpen2018 - Dutch Nao Team : HULKs

Tab. B.11: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Strafstoßmarkendetektion auf der Spielsequenz aus dem Spiel Dutch Nao Team : HULKs bei den IranOpen2018.*

Kenngröße	Wert
Ground-Truth	2
Vorhersagen	0
Richtig Positive (TP)	0
Falsch Negative (FN)	2
Falsch Positive (FP)	0
Trefferquote (TPR)	0
Genauigkeit (PPV)	N/A
$\overline{\text{IoU}}$	N/A

GermanOpen2018 - HULKs : B-Human

Tab. B.12: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Strafstoßmarkendetektion auf der Spielsequenz aus dem Spiel HULKs : B-Human bei den GermanOpen2018.*

Kenngröße	Wert
Ground-Truth	10
Vorhersagen	5
Richtig Positive (TP)	5
Falsch Negative (FN)	6
Falsch Positive (FP)	0
Trefferquote (TPR)	0,5
Genauigkeit (PPV)	1
$\overline{\text{IoU}}$	0,664502

GermanOpen2018 - HULKs : Nao Devils

Tab. B.13: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Strafstoßmarkendetektion auf der Spielsequenz aus dem Spiel HULKs : Nao Devils bei den GermanOpen2018.*

Kenngröße	Wert
Ground-Truth	38
Vorhersagen	13
Richtig Positive (TP)	13
Falsch Negative (FN)	25
Falsch Positive (FP)	0
Trefferquote (TPR)	0,342105
Genauigkeit (PPV)	1
$\overline{\text{IoU}}$	0,645102

GermanOpen2018 - NomadZ : HULKs

Tab. B.14: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Strafstoßmarkendetektion auf der Spielsequenz aus dem Spiel NomadZ : HULKs bei den GermanOpen2018.*

Kenngröße	Wert
Ground-Truth	59
Vorhersagen	39
Richtig Positive (TP)	39
Falsch Negative (FN)	20
Falsch Positive (FP)	0
Trefferquote (TPR)	0,661017
Genauigkeit (PPV)	1
$\overline{\text{IoU}}$	0,708996

GermanOpen2018 - B-Human : HULKs

Tab. B.15: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Strafstoßmarkendetektion auf der Spielsequenz aus dem Spiel B-Human: HULKs bei den GermanOpen2018.*

Kenngröße	Wert
Ground-Truth	21
Vorhersagen	10
Richtig Positive (TP)	10
Falsch Negative (FN)	11
Falsch Positive (FP)	0
Trefferquote (TPR)	0,47619
Genauigkeit (PPV)	1
$\overline{\text{IoU}}$	0,727868

RoboCup2018 - HULKs : UT Austin Villa

Tab. B.16: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Strafstoßmarkendetektion auf der Spielsequenz aus dem Spiel HULKs : UT Austin Villa bei dem RoboCup2018.*

Kenngröße	Wert
Ground-Truth	9
Vorhersagen	2
Richtig Positive (TP)	2
Falsch Negative (FN)	7
Falsch Positive (FP)	0
Trefferquote (TPR)	0,222222
Genauigkeit (PPV)	1
$\overline{\text{IoU}}$	0,6865

RoboCup2018 - HULKs : B-Human

Tab. B.17: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Strafstoßmarkendetektion auf der Spielsequenz aus dem Spiel HULKs : B-Human bei dem RoboCup2018.*

Kenngröße	Wert
Ground-Truth	82
Vorhersagen	29
Richtig Positive (TP)	29
Falsch Negative (FN)	53
Falsch Positive (FP)	0
Trefferquote (TPR)	0,353659
Genauigkeit (PPV)	1
$\overline{\text{IoU}}$	0,611315

RoboCup2018 - rUNSWift : HULKs

Tab. B.18: *Ergebnisse der im Rahmen dieser Arbeit entwickelten Strafstoßmarkendetektion auf der Spielsequenz aus dem Spiel rUNSWift : HULKs bei dem RoboCup2018.*

Kenngröße	Wert
Ground-Truth	112
Vorhersagen	42
Richtig Positive (TP)	42
Falsch Negative (FN)	70
Falsch Positive (FP)	0
Trefferquote (TPR)	0,375
Genauigkeit (PPV)	1
$\overline{\text{IoU}}$	0,721571