Institut für Parallele und Verteilte Systeme
Abteilung Bildverstehen
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2337

Echtzeit-Ballerkennung mit Hilfe von Tiefeninformationen im RoboCup-Szenario

Matthias Nösner

Studiengang:	Technische Kybernetik
Prüfer:	Prof. Dr. Paul Levi
Betreuer:	Dipl.-Inf. Daniel Di Marco
begonnen am:	20.05.2011
beendet am:	10.04.2012
CR-Klassifikation:	I.2.9, I.2.10, I.4.9

Zusammenfassung

Im Rahmen dieser Studienarbeit wurde eine Echtzeit-Ballerkennung entwickelt, die sich hauptsächlich auf die Verwendung von Tiefeninformation stützt. Dazu wurden mehrere Verfahren entwickelt, die aufeinander aufbauend eine Punktwolke bestehend aus Tiefeninformation so weit verkleinern, dass sich die Auswahl auf wenige Punkte beschränkt. Um einen Ball eindeutig verifizieren zu können, wurde die Farbinformation des Balls hinzugezogen. Die Vereinigung des Tiefen- und Farbbilds wurde durch eine speziell für diese Anwendung entwickelte manuelle Kalibrierung der beiden Kameras erreicht.

Echtzeit-Ballerkennung mit Hilfe von Tiefeninformationen (Kinect) im RoboCup-Szenario

Matthias Nösner

6. April 2012

Inhaltsverzeichnis

1 Einleitung

In der Robotik ist Objekterkennung durch Auswertung der Daten von bildgebender Hardware ein zentraler Bestandteil der aktuellen Forschung. Sich ständig verbessernde Technik und großer Forschungsaufwand in diesem Bereich haben die Verfahren leistungsfähiger und effizienter gemacht. Trotz dieser Fortschritte sind auch die aktuellen Verfahren noch weit entfernt von der menschlichen Perzeptionsfähigkeit. Das langfristige Ziel der Forschung ist es autonome Roboter zu entwickeln, die sowohl in der Lage sind sich in der menschlichen Welt frei zu bewegen als auch mit Menschen zu interagieren ohne sie zu gefährden [13]. Um dies erreichen zu können, ist eine robuste und schnelle Wahrnehmung der Umgebung durch Sensoren von essentieller Bedeutung.

Heutige autonome Systeme sind in ihren Ressourcen, wie beispielsweise dem Arbeitsspeicher, dem Permanentspeicher und der Rechenleistung, sehr begrenzt. Zwar können die verwendeten bildgebenden Sensoren die Umgebung heutzutage besser als das menschliche Auge darstellen, doch stellt die Kognition und Klassifizierung der Umwelt immer noch ein Gebiet dar auf dem viel Forschungsarbeit notwendig ist. Die Erstellung einer Objektdatenbank wie sie derzeit innerhalb des europäischen Forschungsprojekts “Roboearth“ [13] stattfindet, ist sicher ein Schritt in die richtige Richtung. Innerhalb dieses Projekts wird versucht ein Internet für Roboter aufzubauen, um redundante Entwicklungsarbeit zu verhindern.

Um entwickelte Algorithmen zu testen, nimmt die Universität Stuttgart jedes Jahr am RoboCup [10] in der Mid-Size-League teil. Im Rahmen dieses Tuniers und dessen Vorbereitung entstand auch die Fragestellung dieser Arbeit.

In der vorliegenden Arbeit wird ein speziell für die Ballerkennung evaluiertes Verfahren beschrieben und bewertet. Dabei fand vor allem die benötigte Ausführungszeit der Algorithmen Beachtung. Ziel war es, die Ballerkennung in die vorhandene RoboCup-Architektur des Instituts für Paralelle und Verteilte Sytsteme der Universität Stuttgart zu integrieren. Dafür musste gewährleistet werden, dass die Ausführungszeit der Ballerkennung ein 30ms-Fenster nicht überschreitet, besser noch so schnell wie möglich abläuft. Zusätzlich wurde auf die Robustheit der Ballerkennung besonderen Wert gelegt.

Da die Ballerkennung durch Farbkameras im Robocup weit verbreitet ist, wird in dieser Arbeit speziell auf die Verwendung der Tiefeninformation eingegangen. Es soll ein Überblick gegeben werden, wie Tiefeninformation aus einem verrauschten Tiefensensor zur Objekterkennung verwendet werden kann. Da die Arbeit vor allem im Zusammenhang mit der Teilnahme am sogenannten RoboCup entstanden ist, soll dieser einleitend kurz erläutert werden.

1.1 RoboCup

Jedes Jahr wird im Rahmen des RoboCups eine Weltmeisterschaft ausgetragen, bei der Teams aus allen Teilen der Welt in verschiedenen Klassen gegeneinander antreten. Neben der Klasse RoboCupSoccer, in der die Universität Stuttgart vertreten ist, gibt es noch zusätzlich die Klassen RoboCupRescue, RoboCup@Home und RoboCupJunior. Das selbst gesteckte Ziel des RoboCups ist es, im Jahre 2050 den aktuellen Fußball Weltmeister schlagen zu können.

1.2 MidSize League

In der MidSize League treten autonome Roboterplattformen gegeneinander an. Diese bestehen jeweils aus vier Feldspielern und einem Torwart. Das Spiel und seine Regeln lehnen sich im Allgemeinen an das menschliche Fußballspiel an. Ziel ist es, den Ball im Tor des Gegners zu versenken. Angetrieben werden die Plattformen durch omnidirektionale Antriebe und haben bei den Spitzenmannschaften inzwischen eine aktive Ballführung. Die Wahrnehmung der Umgebung findet hauptsächlich durch omnidirektionale Kameras statt. Durch Farbsegmentierung und vorherige Farbkalibrierung werden Informationen wie die Position der Gegner, die eigene Position auf dem Spielfeld und die Position des Balls erkannt.

2 Grundlagen und Stand der Technik

In diesem Kapitel wird auf schon vorhandene 3-D-Ballerkennungsalgorithmen eingegangen und die bekannten Lösungsansätze erläutert. Im Weiteren soll es einen Überblick über die theoretischen Grundlagen geben, die zum Teil im Laufe der Arbeit entstanden sind oder sich in diesem Zusammenhang als nützlich erwiesen haben.

2.1 3-D-Ballerkennung des Team Tech United beim Robocup 2011 in Istanbul

Die 3-D-Ballerkennung, mit Hilfe des Kinect Sensors, die das Team Tech United beim RoboCup 2011 in Istanbul einsetzte [7], benutzt eine manuell durchgeführte Farbkalibrierung des Balls. Hierdurch ist es möglich, den Ball von der Umgebung abzugrenzen. Um die Tiefeninformation eines Pixels aus dem Farbbild verwenden zu können, benutzt das Team eine exakte Kalibrierung des Kinect Sensors.

Der Großteil der Operationen wird auf dem Farbbild durchgeführt. Nur zur Verifizierung und Positionsbestimmung des Balls wird die Tiefeninformation verwendet.

Die Ballerkennung läuft in groben Schritten wie folgt ab. Im ersten Schritt werden Pixel herausgefiltert deren Tiefenwert nicht zwischen 0,31 und 6,0 Metern liegt. Darauf aufbauend setzt eine Farbsegmentierung an, die alle Pixel herausfiltert deren Farbwert nicht zu denen der kalibrierten Ballfarben passt. Um das Bild von unerwünschtem Rauschen zu befreien, wird “erosion” und “delation” [6] mit einer 3x3-Maske auf das Farbbild angewendet.

In einem nächsten Schritt findet ein mehrstufiges Clustering durch den “k-means” Algorithmus statt, den Hartigan und Wong in [8] beschreiben. Dabei werden die Pixel in mehrere Gruppen unterteilt, die sich durch unterschiedliche Farben voneinander abgrenzen. Die Entfernung zum Mittelpunkt des Clusters ist durch das Tiefenbild gegeben. Mit Hilfe dessen kann die maximale Pixelanzahl des Balls durch ein Ballmodell abgeschätzt werden. Durch Division der Anzahl der Pixel im Cluster mit der maximalen Pixelanzahl wird ein Vertauenswert erstellt, der beim Überschreiten für das Erkennen eines Balls steht.

2.2 Kantendetektion

In [9] beschreibt Bernd Jähne die Kantendetektion als Differentation durch Nachbarschaftsoperatoren. Bei diskreten Bildern wird durch eine diskrete Nachbarschaftsmaske die Differenz von z.B. den Grauwerten gebildet. Die Nachbarschaftsmaske entscheidet über die Nachbarschaftsbeziehungen der einzelnen Punkte zueinander. Das mit diesem Verfahren gebildete Merkmalsbild enthält je nach verwendeter Nachbarschaftsmaske

spezifische Kanten. Dabei ist zu berücksichtigen, dass nicht immer alle Kanten mit jeder Nachbarschaftsmaske dargstellt werden können. Aus dem Merkmalsbild kann in einem weiteren Schritt durch Segmentierung die gegebenen Kanten herausgefiltert werden. In der Mathematik wird die Anwendung einer Nachbarschaftsmaske auf ein diskretes Bild auch diskrete Faltung genannt. Im Folgenden soll kurz auf zwei spezifische Nachbarschaftsmasken, die auch als Filter bezeichnet werden, eingegangen werden.

2.2.1 Sobel-Operator

Der Sobel-Operator ist ein einfacher Kantenfilter, der sowohl für die horizontale als auch für die vertikale Kantenerkennung jeweils eine Faltung durchführt. Die zwei 3x3-Masken

$$S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \quad und \quad S_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \tag{1}$$

ergeben mit dem Bild A gefaltet

$$G_y = S_y * A \tag{2}$$

$$G_x = S_x * A. \tag{3}$$

Diese beiden Funktionen kombiniert ergeben den Gradienten der richtungsunabhängigen Information

$$G = \sqrt{G_y^2 * G_x^2}. \tag{4}$$

2.2.2 Laplacefilter

Der Laplacefilter vereinigt die horizontale und vertikale Kantenerkennung in einer Maske. Dabei wird die Summe der beiden zweiten Ableitungen approximiert. Mit

$$S = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{5}$$

genügt eine Faltung aus, um die Kanten im Bild A zu bestimmen.

$$G = S * A. \tag{6}$$

2.3 Objekterkennung

Damit ein autonomes System seine Umgebung deuten kann, ist die Perzeption der sich darin befindlichen Objekte von essentieller Bedeutung. Probleme dabei bereitet vor allem die Robustheit der Algorithmen, die sowohl unterschiedliche Lichtverhältnisse kompensieren müssen als auch Objekte deren Struktur sich ähnelt erkennen und unterscheiden müssen. Irving Biederman hat sich in [5] mit der menschlichen Objekterkennung befasst und ein System aus mehreren Stufen vorgestellt, das in abgewandelter Weise auf das maschinelle Sehen angewendet werden kann.

Die grundlegende Vorgehensweise der Objekterkennung ist ein modular aufgebautes Schema, bei dem die erste Stufe “Low-level preparation” das Bild für die weitere Verarbeitung vorbereiten soll. Bilder müssen dabei unter anderem von Rauschen befreit werden. Ein Beispiel ist das “Erosion”- und “Delation”-Verfahren in [6]. Finden die weiteren Schritte der Objekterkennung im Frequenzbereich statt, so kann hier z.B. eine diskrete Fouriertransformation [6] angewandt werden, um vom Bildbereich in den Frequenzbereich zu transformieren.

Ist das Bild in ein für die weitere Verarbeitung sinnvolles Format gebracht worden, so können in der zweiten Stufe “Feature extraction” durch Kantenerkennung die im Bild enthaltene Information weiter spezifiziert werden. Darauf aufbauend können die erkannten Kanten zu geometrischen Formen bzw. zu logisch zusammenhängenden Pixelgruppen zusammengefasst werden.

In der nächsten Stufe “Component Extraction” werden aus den gefunden Pixelwolken objektspezifische Merkmale extrahiert. Sind alle Merkmale gefunden, so können in der letzten Stufe “Object recognition” die gefundenen Merkmale mit einer Datenbank verglichen werden. Durch diesen Schritt kann durch verschiedene Entscheidungkriterien das Objekt klassifiziert werden.

Das hier beschriebene, grundelegende Verfahren soll einen groben Überblick über die Möglichkeiten der strukturell aufgebauten Objekterkennung geben. Es sollte beachtet werden, dass jede Stufe für sich ein komplexes Themengebiet abdeckt.

2.4 Stereokamera-Kalibrierung

Stereokameras besitzen zwei Objektive mit denen die Umgebung aus unterschiedlichen Blickwinkeln aufgenommen werden kann. Die verschiedenen Objektive unterscheiden sich in einigen Charakteristika wie der unterschiedlichen Position der Objektive zueinander sowie ihrer Brennweite und spezifischen Linseneigenschaften. Um die beiden Bilder in einem einheitlichen Koordinatensystem darstellen zu können, ist es nötig die Kameras zu kalibrieren. Tsai liefert in seiner Arbeit [12] eine sehr genaue Stereokamera-Kalibrierung. Die Grundlagen der Kalibrierung nach Tsai sind im Folgenden erläutert.

2.4.1 Tsai-Kalibrierung

Die Kalibrierung von Tsai transformiert das 2-D-Kamerabild in eine 3-D-Ebene, in der die beiden Kamerapositionen in Relation zu einem festen Punkt angegeben werden. Er geht dabei von elf Parametern aus, die in sechs extrinsische und fünf intrinsische Parameter unterteilt sind. Die extrinsische Kalibrierung beschreibt dabei die Verschiebung und Verdrehung der Kameras zueinander. Die intrinsischen Parameter hingegen beschreiben die unterschiedlichen Brennweiten sowie die radiale Verzerrung, die durch Produktionstoleranzen der Linsen entsteht. Die Parameter werden meist durch ein Kalibrierungspattern wie das Schachbrettmuster bestimmt. Dabei muss die Größe der einzelnen Felder des Schachbretts bekannt sein.

2.4.2 Tsai-Kalibrierung - Intrinsische Parameter

Die intrinsischen Parameter hängen im Gegensatz zu den extrinsischen Parametern nicht von der Kameraposition ab. Sie beschreiben die Transformation aus dem 2-D-Bildkoordinatensystem in das 3-D-Kamerakoordinatensystem. Sie kompensieren die spezifischen geometrischen Merkmale der Kamera wie Verzerrung, Brennweite und Skalierungsfaktor mit dem die Kamera die Umgebung abbildet. Es gibt zwei wichtige Arten von Verzerrungen. Die **radiale** und die **tangentiale Verzerrung**, wobei letztere hier nicht berücksichtigt wird, da bei Linsen radiale Verzerrungen weitaus stärker auftreten. In Abbildung 1 sind die beiden Verzerrungen dargestellt.

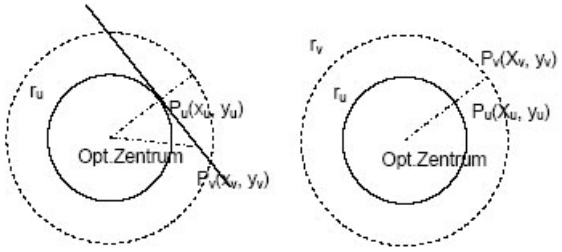

Abbildung 1: Tangetiale und radiale Verzerrung entnommen aus [**?**]

Je nach Art der radialen Verzerrung werden die Punkte in Richtung des Fokus oder in die entgegengesetzte Richtung skaliert. Beschrieben wird sie durch eine höher polynomiale Funktion.

$$X_u = D_x + X_v \tag{7}$$

$$Y_u = D_y + Y_v \tag{8}$$

$$D_x = Xu * (k_1 * r^2 + k_2 * r^4 + ...) \tag{9}$$

$$D_y = Y_u * (k_1 * r^2 + k_2 * r^4 + ...) \tag{10}$$

Dabei ist r der Abstand zum Fokus. Für die meisten Fälle ist es ausreichen nur k_1 zu bestimmen.

Die fünf intrinsischen Parameter die Tsai für seine Kalibrierung benutzt sind

1. Die Brennweite f
2. Der Verzerrungskoeffizient k für radiale Verzerrung
3. Der Skalierungsfaktor S_x
4. Die Koordinaten des Zentrums der Verzerrung (Cx, Cy) bezüglich des verzerrten Kamerabildes.

2.4.3 Tsai-Kalibrierung - Extrinsische Parameter

Die extrinsichen Parameter der Kalibrierung dienen dazu die 3-D-Kamerakoordinatensysteme in einem globalen Koordinatensystem zu vereinigen. Sie werden durch eine **Rotationsmatrix R** und einen **Translationsvektor T** beschrieben. Wobei die Rotationsmatrix wie auch der Translationsvektor jeweils drei Freiheitsgrade besitzen.

2.4.4 Vorgehensweise

Die schematische Tsai-Kalibrierung ist in Abbildung 2 dargestellt.

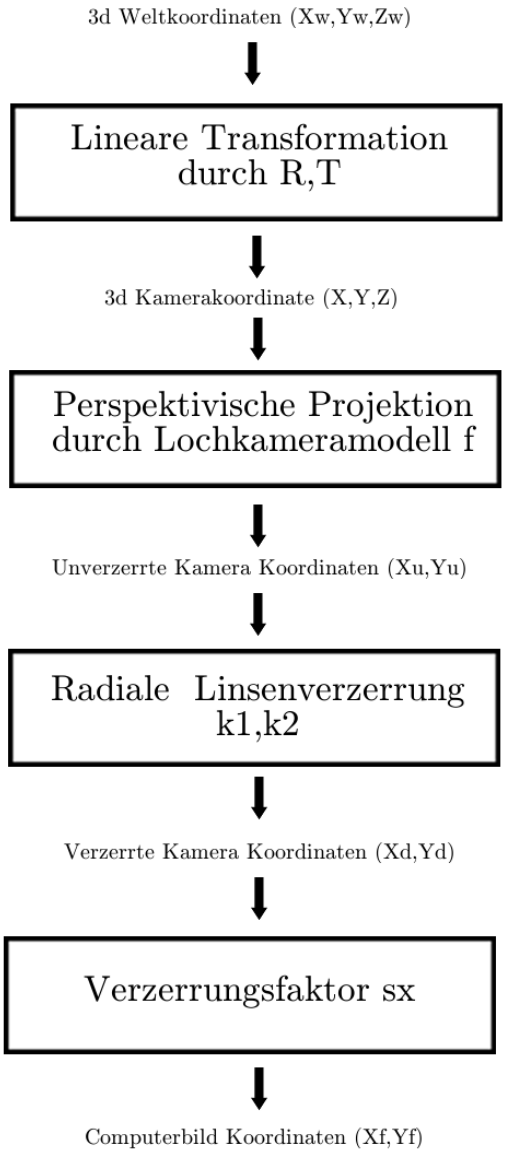

Abbildung 2: Vorgehensweise bei der Kamerakalibrierung nach Tsai entnommen aus [12]

2.5 Positionsbestimmung aus Tiefeninformation

Ein Pixel dessen Position im Bildkoordinatensystem $[x, y, z]$ gegeben ist, wobei x und y Pixelangaben sind und z der Tiefenwert ist, kann durch Beachtung der kameraspezifischen Parameter in das Ursprungskoordinatensystem des Kinect Sensors transformiert

werden. Dafür muss sowohl der horizontale Öffnungswinkel α des Kameraobjektivs bekannt sein als auch die horizontale Auflösung δ in Pixel. Im ersten Schritt wird der Winkel β berechnet, er beschreibt den Winkel um den das Pixel vom Ursprungkoordinatensystem verdreht ist.

$$\beta = \frac{x - \frac{\delta}{2}}{\frac{\delta}{2}} * \frac{\alpha}{2} \tag{11}$$

Daraus lassen sich x' und y' im Ursprungskoordinatensystem durch

$$x' = z * \cos(\beta * \frac{\pi}{2}) \tag{12}$$

$$y' = z * \sin(\beta * \frac{\pi}{2}) \tag{13}$$

berechnen.

2.6 Ballmodell

Um die Dimensionen des Balls im Computer abbilden zu können, ist ein a priori Wissen über den Ball sowie die kameraspezifischen Parameter notwendig. Der Ballumfang U, der Kameraöffnungswinkel δ und die Auflösung x bzw. y müssen bekannt sein. Abweichungen, die durch die unterschiedlichen Ballgrößen und durch Sensorungenauigkeiten entstehen, werden nicht berücksichtigt.
Der Ballradius r ergibt sich aus

$$r = \frac{U}{2 * \pi} \tag{14}$$

Für die Transformation in ein vom Computer benutztes Bildkoordinatensystem wird die Beziehung

$$\frac{x}{l} = \frac{\mu}{2 * r} \tag{15}$$

benötigt. Dabei ist die Länge des Bildausschnitts l in Abhängigkeit der Entfernung z

$$l = 2 * \tan(\frac{\delta}{2}) * z \tag{16}$$

gegeben. Für die Größe des Balls im Bildkoordinatensystem ergibt sich

$$\mu = \frac{x}{l} * 2 * r \tag{17}$$

bzw.

$$\mu = \frac{x * r}{\tan(\frac{\delta}{2})} * \frac{1}{z} \tag{18}$$

3 Übersicht über die Basiskomponenten

Im Laufe dieser Arbeit wurde verschiedene Hardware und Software benutzt, die im Folgenden kurz erläutert wird.

3.1 Der Audio-, Bild- und Tiefensensor Kinect

2010 brachte Microsoft den Kinect Sensor für die X-Box heraus. Aufgrund der Popularität der X-Box war es möglich diesen Sensor in großer Stückzahl fertigen zu lassen und somit gelang es zum ersten Mal einen 3-D-Sensor für einen erschwinglichen Preis anzubieten. Es dauerte nicht lange bis die Robotikforscher diesen Sensor für sich entdeckten.

Der Kinect Sensor besitzt eine RGB-Kamera, um die Farbinformationen aus der Umgebung zu verarbeiten, sowie vier Mikrofone zur Ortung einer Schallquelle in einem Raum. Der Klasse-1-Laser sowie der Infrarotsensor bilden zusammen einen Tiefensensor. In Abbildung 3 ist der Sensor dargestellt. Sowohl das Farb- als auch das Tiefenbild

Abbildung 3: Der Kinect Sensor

werden mit einer Bildwiederholungsrate von 30 Bilder pro Sekunde angegeben. Dabei ist aber nicht sichergestellt, dass beide Bilder synchron bereitgestellt werden. Aufgrund der ausreichend hohen Bildwiederholungsrate ist dies jedoch nicht von Bedeutung.

3.1.1 Tiefensensor

Wie oben bereits erwähnt besteht der Tiefensensor aus einem Klasse-1-Laser sowie einem Infrarotsensor. Dieser Laser projeziert ein Punktegitter mit verschiedenen Punktintensitäten in die Umgebung. Der Infrarotsensor ist in der Lage durch 2-D-Bilderkennungsverfahren einen Tiefenwert daraus zu berechnen. Zurückgegeben wird ein 640x480-Pixel-Bild mit einer Datentiefe von 11-Bit. Durch die geometrische Anordnung der beiden Einheiten des Tiefensensors entstehen Bereiche, in denen es nicht möglich ist Tiefeninformation zu ermitteln. Die Problematik wird in Abbildung 4 verdeutlicht.

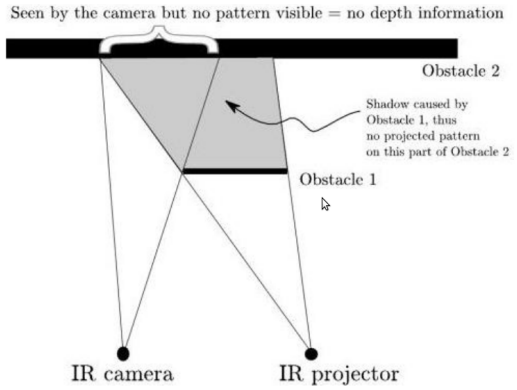

Abbildung 4: Bereich, in dem keine Tiefeninformation ermittelt werden kann, entnommen aus [11]

3.2 OpenNi

Um den Kinect Sensor mit einem Computer zu verbinden, ist ein Treiber nötig, der die von der Kinect generierten Daten in ein für die Weiterverarbeitung sinnvolles Format übersetzt. OpenNI [3] stellt einen solchen Treiber für den Kinect Sensor zur Verfügung, der als Schnittstelle zwischen Computer und Kinect Sensor fungiert.

3.3 OpenCv

OpenCv [1] ist eine Bibliothek zur Bildverarbeitung, die von Intel entwickelt wurde und zur Zeit von Willow Grage gepflegt wird. Darin sind viele bekannte Bildverarbeitungsalgortithmen integriert, die ohne exakte Kenntnisse des Verfahrens genutzt werden können. Verfahren zur Bildverarbeitung, Gesichtserkennung, Bildkonvertierung und Kamerakalibrierung sowie verschiedene Datenformate für Bilder werden bereitgestellt.

3.4 C++

Um den Code in das bestehende RoboCup-Projekt intergieren zu können, musste die Arbeit in C++ programmiert werden. Der Vorteil von C++ ist die manuelle Speicherverwaltung, die dem Programmierer überlassen ist. Dadurch lassen sich effiziente und rechenzeitsparende Algorithmen schreiben. Allerdings birgt diese Möglichkeit auch Gefahren durch unerlaubte Speicherzugriffe, die der Programmierer vermeiden muss. Hierdurch entsteht meist ein größerer Entwicklungsaufwand.

3.5 Qt

Da C++ keine integrierten Funktionen zur Gui-Programmierung besitzt, wurde das Qt Framework [4] von Nokia benutzt, um eine Benutzerschnittstelle herzustellen und die Daten zu visualisieren. Qt ist eine sehr gut dokumentierte Entwicklungsumgebung, die dem Anwender viele grundlegende Funktionen zur Verfügung stellt.

4 Entwurf

In diesem Kapitel werden alle im Laufe dieser Arbeit entwickelten und für die Ballerkennung relevanten Verfahren erwähnt und beschrieben. Das jeweils verwendete Verfahren wird meist durch ein Beispiel in Form von Pseudocode ergänzt. Hierdurch soll ein Überblick über die Implementierungsmöglichkeit gegeben werden. Einzelheiten, die die Algorithmen komplexer machen und nicht zum Verständnis beitragen, wurden nicht angegeben. Ein konkreter Fall hierfür ist die Einhaltung der Grenzen des Speicherbereichs, in dem das Tiefen- bzw. Farbbild gespeichert ist. Sobald ein Zugriff außerhalb des definierten Speicherbereichs stattfindet, stürzt das Programm aufgrund eines Speicherzugriffsfehler ab. Jedes Verfahren muss so angepasst werden, dass diese Grenzen nicht überschritten werden.

Der praktische Teil dieser Arbeit wurde in C++ programmiert. Für die Oberfläche wurde Qt [4] verwendet und um die Bilddaten zu verarbeiten wurde auf OpenCv [1] zurückgegriffen. Der OpenNI [3] Treiber stellt die Schnittstelle zwischen Kinect Sensor und C++ Umgebung her. Die hier verwendete Software wurde bereits in Kapitel 3 erwähnt und erläutert. Im Vorfeld sind einige Definitionen notwendig, um die beschriebenen Verfahren eindeutig verstehen zu können.

Koordinatensystem: Der Ursprung des Koordinatensystems befindet sich in der linken, oberen Bildecke. Die Koordinate x gibt den horizontalen Abstand zum Ursprung in Pixel an. Die Koordinate y gibt den vertikalen Abstand zum Ursprung ebenfalls in Pixel an. Zu beachten ist, dass y von der oberen Bildkante weg, zu der unteren Bildkante hin zeigt. Abbildung 5 zeigt das im Folgenden verwendete Koordinatensystem.

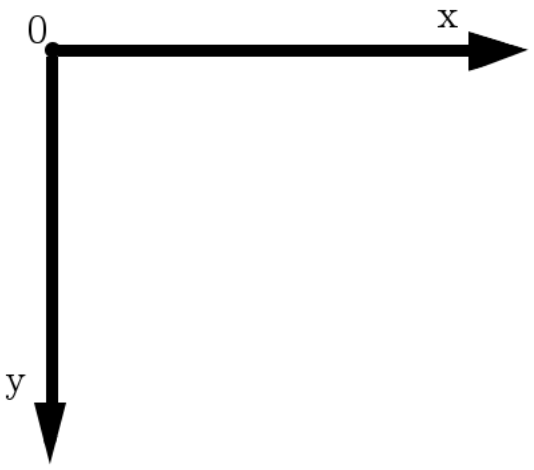

Abbildung 5: Das in der Bildverarbeitung typische Koordinatensystem

cpoint: Ist die Abkürzung für “center Point”. Damit ist ein Punkt gemeint, bei dem die Möglichkeit besteht, dass er sich in der Mitte des Balls befindet. Meist wird das nur angenommen, um den Punkt durch Ausschlussverfahren zu verifizieren.

point.x: x-Wert des Punkts “point”.

point.y: y-Wert des Punkts “point”.

point.z: Tiefenwert des Punkts “point”.

point.orientation = true: Punkt auf der linken Ballkante.

point.orientation = false: Punkt auf der rechten Ballkante.

ballmodel(distance): Berechnet den Balldurchmesser in Abhängigkeit der Größe “distance” aus dem Tiefenbild, siehe Kapitel 2.6.

edge_detected(x,y): Gibt true zurück, wenn sich eine Kante an dem Punkt (x,y) befindet. Das Verfahren wird in Kapitel 4.4 beschrieben.

4.1 Behandlung von Rauschen im Tiefenbild

Das Tiefenbild weist mit zunehmender Entfernung verstärkt verrauschte Punkte auf. Besonders an Kanten ist dieser Effekt zu beobachten. In Abbildung 15 und 20 ist zu erkennen, dass in dem Tiefenbild die Kanten verpixelt dargestellt sind. Um solche Effekte zu kompensieren, wurden den Verfahren Toleranzwerte hinzugefügt, die die Stärke des jeweiligen Ausschlusskriteriums herabsetzen. Je nach Verfahren müssen diese Toleranzwerte individuell eingestellt werden, damit der Ballerkennungsprozess eine ausreichende Robustheit erreicht.

Ein weiteres Problem besteht in der mit der Entfernung zunehmend ungenauen Darstellung der Tiefenwerte. Je weiter ein Punkt vom Kinect Sensor entfernt ist, desto ungenauer ist der berechnete Wert des Tiefensensors zum realen Wert. Würde man diese Werte direkt dem Ballmodell (Kapitel 2.6) übergeben, so würde die Größe des berechneten Balldurchmessers nicht stimmen. Deshalb wurde in mehreren Bereichen die Abweichung zwischen Tiefenwert und realem Wert ermittelt, sodass der Tiefenwert zur Laufzeit des Programms angepasst werden kann. Dadurch wird lediglich der Fehler auf ein für den Ballerkennungsprozess verträgliches Maß reduziert. Eine qualitativ hochwertige Korrektur der Tiefenwerte soll nicht Teil dieser Arbeit sein.

4.2 Intuitive Farb- und Tiefenbild-Kalibrierung

Da eine vollständige und damit hochwertige Kamerakalibrierung mit der Kalibrierungsmethode für Stereokameras wie in Kapitel 2.4 beschrieben im Rahmen dieser Arbeit zu aufwendig ist, wird hier eine vereinfachte Methode benutzt, die für die Ballerkennung

eine ausreichende Genauigkeit vorweist. Legt man die durch Kantenerkennung extrahierten Kanten des Tiefenbilds über das Farbbild, so ist der Positionsunterschied gut zu erkennen. Der Unterschied der beiden Bilder ergibt sich durch die geringfügig verschiedenen Öffnungswinkel der Kameras sowie durch deren Positionen. Nichtlineare Effekte, die duch Produktionstoleranzen der Linsen auftreten, spielen hier ebenfalls eine Rolle. Abbildung 6 zeigt die bestehende Problematik der nicht zueinander passenden Bilder bei der Überlagerung.

Abbildung 6: Farbbild mit überlagerten Kanten aus dem Tiefenbild

Es kann angenommen werden, dass jeder Punkt im Tiefenbild einem Punkt aus dem Farbbild zugeordnet werden kann. Durch die Annahme, dass das Bild der Farbkamera eine größere Fläche abdeckt als das Bild der Tiefenkamera und die Verdrehung der beiden Bilder zueinander sehr klein ist, gibt es je Tiefenebene einen Punkt, der nicht verschoben werden muss. Im Folgenden werden diese als Zentralpunkte χ bezeichnet. In Abbildung 7 ist die Projektion in der Nähe eines Zentralpunkts dargstellt.

Abbildung 7: Ballmittelpunkt aus dem Tiefenbild auf das Farbbild projiziert in der Nähe eines Zentralpunktes

Abbildung 8 zeigt die Problematik der falsch projizierten Ballmittelpunkte aus dem Tiefenbild.
Durch die Bestimmung von zwei Zentralpunkten in unterschiedlichen Entfernungen lassen sich die Zentralpunkte der dazwischenliegenden Tiefenebenen interpolieren. Die

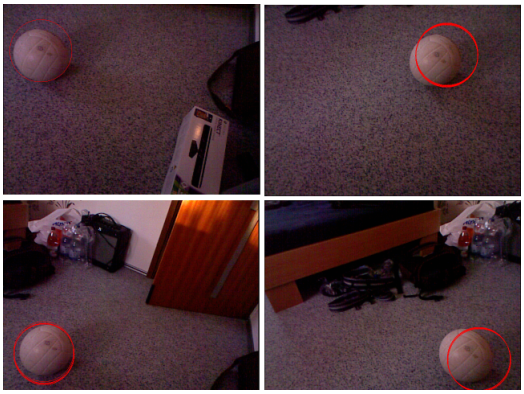

Abbildung 8: Projektion aus dem Tiefenbild in das Farbbild ohne Kalibrierung

Bestimmung der zwei Zentralpunkte ist durch die implementierte Ballerkennung manuell möglich. Soll nun zu einem Punkt im Tiefenbild

$$\mu = \begin{pmatrix} \mu_x \\ \mu_y \\ \mu_z \end{pmatrix} \tag{19}$$

der zugehörige Punkt im Farbbild bestimmt werden, so muss zuerst der zugehörige Zentralpunkt χ_3 aus der Tiefeninformation μ_z berrechnet werden. Durch die beiden im Vorraus bestimmten Zentralpunkte

$$\chi_1 = \begin{pmatrix} \chi_{x1} \\ \chi_{y1} \\ \chi_{z1} \end{pmatrix}, \chi_2 = \begin{pmatrix} \chi_{x2} \\ \chi_{y2} \\ \chi_{z2} \end{pmatrix} \tag{20}$$

wobei gilt,

$$\chi_{z1} < \chi_{z2} \tag{21}$$

wird der zu dem Tiefenwert μ_z passende Zentralpunkt χ_3 mit Hilfe der lineare Interpolation

$$\lambda = \frac{\mu_z - \chi_{z1}}{\chi_{z2} - \chi_{z1}} \tag{22}$$

$$\begin{pmatrix} \chi_{x3} \\ \chi_{y3} \\ \chi_{z3} \end{pmatrix} = \begin{pmatrix} \chi_{x1} + \frac{\chi_{x2} - \chi_{x1}}{\lambda} \\ \chi_{y1} + \frac{\chi_{y2} - \chi_{y1}}{\lambda} \\ \mu_z \end{pmatrix} \tag{23}$$

bestimmt. Durch

$$\begin{pmatrix} \triangle_x \\ \triangle_y \end{pmatrix} = \begin{pmatrix} |\chi_{x3} - \mu_x| \\ |\chi_{y3} - \mu_y| \end{pmatrix} \tag{24}$$

kann mit

$$\begin{vmatrix} \triangle_x \\ \triangle_y \end{vmatrix} \tag{25}$$

der Abstand zum Zentralpunkt χ_3 bestimmt werden. Die Abstandsbestimmung ist wichtig, da die Projektion aus dem Tiefenbild in das Farbbild nichtlinear ist. Durch die Entfernung zum Zentralpunkt kann die Entscheidung getroffen werden welcher linearisierte Arbeitsbereich benutzt werden soll. Die Arbeitsbereiche sind in Abbildung 9 erläutert. Es wurden nur drei Arbeitsbereiche ausgewählt, um den Kalibrierungsaufwand gering zu halten.

Für die Berechnung der Pixelposition in den Koordinaten des Farbbildes

$$\mu' = \begin{pmatrix} \mu'_x \\ \mu'_y \end{pmatrix}, \tag{26}$$

in Abhängikeit des im jeweiligen Arbeitsbereich gültigen Faktors Φ, gilt

$$\begin{pmatrix} \mu'_x \\ \mu'_y \end{pmatrix} = \begin{pmatrix} \chi_{x3} + (\mu_x - \chi_{x3}) * \Phi \\ \chi_{y3} + (\mu_y - \chi_{y3}) * \Phi \end{pmatrix} \tag{27}$$

wobei sich Φ zwischen 0 und 1 bewegt.

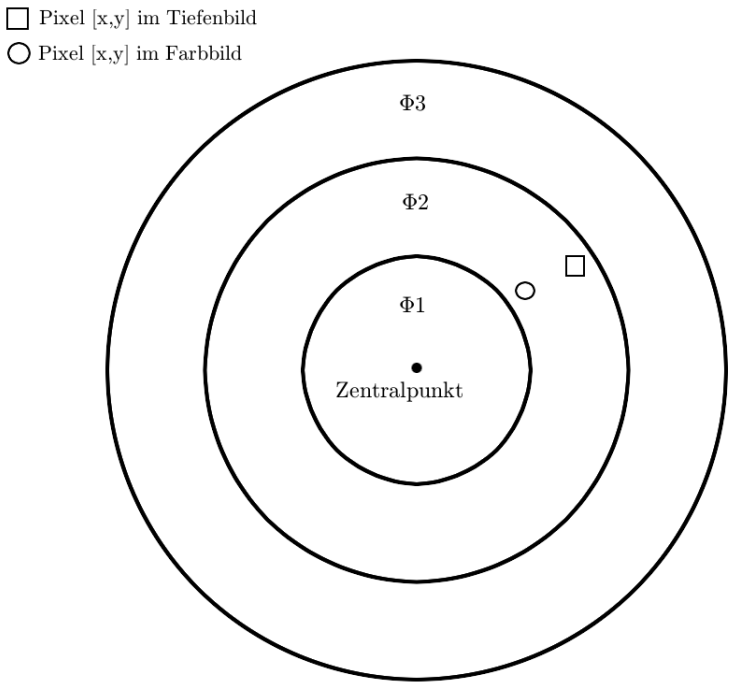

Abbildung 9: Die in den jeweiligen Arbeitsbereichen gültigen Faktoren $\Phi 1$,$\Phi 2$ und $\Phi 3$

Eine mögliche Implementierung der Transformation der Koordinaten des Tiefenbildes μ in die Koordinaten des Farbbildes μ' ist in folgendem Algorithmus 1 gegeben.

Algorithm 1 transformToColorCordinates(point)

$cpoint \leftarrow interpolatecenter(point.z)$
$deltax \leftarrow point.x - cpoint.x$
$deltay \leftarrow point.y - cpoint.y$
$distance \leftarrow \sqrt{deltax * deltax + deltay * deltay}$
if $distance \leq border1$ **then**
  $colorpoint.x \leftarrow cpoint.x + deltax * \Phi_1$
  $colorpoint.y \leftarrow cpoint.y + deltay * \Phi_1$
end if
if $border1 < distance \leq border2$ **then**
  $colorpoint.x \leftarrow cpoint.x + deltax * \Phi_2$
  $colorpoint.y \leftarrow cpoint.y + deltay * \Phi_2$
end if
if $border2 < distance \leq border3$ **then**
  $colorpoint.x \leftarrow cpoint.x + deltax * \Phi_3$
  $colorpoint.y \leftarrow cpoint.y + deltay * \Phi_3$
end if
return colorpoint

4.3 Stufen der Ballerkennung

Die Ballerkennung läuft in mehreren, aufeinander aufbauenden Stufen ab. Die Stufen können in zwei Kategorie unterteit werden, **Informationsextraktion** und **Reduzierung der Punktwolke durch Ausschlussverfahren**. Kantenerkennung und Merkmalsextraktion fallen in die erste Katergorie. Segmentierung und die Verifizierung der Punkte sind Verfahren aus der zweiten Kategorie. Die einzelnen Stufen sind in Abbildung 10 dargestellt.

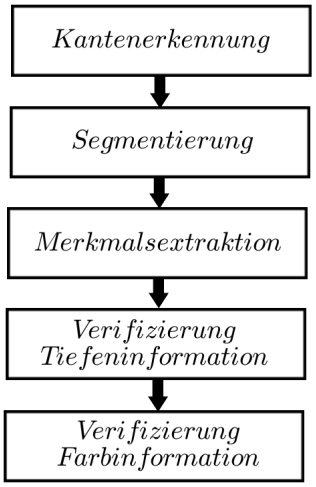

Abbildung 10: Die einzelnen Stufen der Ballerkennung in chronologischer Reihenfolge

Im Folgenden sollen nun die Verfahren genauer betrachtet und erläutert werden.

4.4 Kantenerkennung

Bei der hier verwendten Kantenerkennung wird die Tiefeninformation von jedem Pixel mit der des vorangegangenen Pixels subtrahiert. Die Maske ist in Abbildung 11 zu sehen.

-1	1

Abbildung 11: Nachbarschaftsmaske der Kantenerkennung

Durch die Bildung der Differenz ergibt sich ein Bild mit Intensitätswerten, die für die Stärke der Kante stehen. Die Maske wird für jeden Punkt des Tiefenbildes angewandt. Um keine unerlaubten Operationen außerhalb des Bilds durchzuführen, werden Pixel am linken Rand übersprungen. Das Nichterkennen horizontaler Kantenverläufe ist in diesem speziellen Anwendungsfall vernachlässigbar. Der Aufwand der Kantenerkennung hängt hier von den Dimensionen des zu verarbeiteten Bildes bzw. von dessen Auflösung ab.
Für ein Bild mit M Zeilen und N Spalten beträgt der Aufwand $O(M*(N\text{-}1))$.

4.5 Segmentierung

Auf die Kantenerkennung folgt ein Segmentierungsverfahren, das nur Kanten zulässt deren Intensität innerhalb vorgegebener Grenzen liegt. Die Grenzen werden im weiteren Verlauf der Arbeit mit *tresholdlow* für die minimale Intensität bzw. *tresholdhigh* für die maximale Intensität bezeichnet.

4.6 Orientierung auf dem Ball

Für die genauere Spezifikation der Kanteninformation ist es sinnvoll eine Richtungsinformation hinzuzufügen, die angibt, ob sich ein Punkt auf der linken oder rechten Kante des Balls befindet. Dadurch kann in nachfolgenden Verfahren Rechenzeit gespart werden. Die Zuordnung erfolgt durch Flankenerkennung, wobei die geometrischen Eigenschaften des Kinect Sensors (Kapitel 3.1) beachtet werden müssen. Das Verfahren gehört zur Stufe Merkmalsextraktion. Eine Möglichkeit das Orientierungsmerkmal zu extrahieren zeigt Algorithmus 2. Dabei ist davon auszugehen, dass das Bild zeilenweise von oben nach unten und innerhalb der Zeile von links nach rechts durchsucht wird. Der erste Punkt jeder Zeile wird übersprungen, da diese Punkte keine Vorgänger haben. $prepoint$ ist der Vorgänger von $point$. Ebenfalls ist sichergestellt, dass die Tiefenwerte der Punkte sich unterscheiden.

Algorithm 2 pointOrientation(prepoint,point)

$delta \leftarrow point.z - prepoint.z$
if $prepoint.z == 0$ **then**
  $delta \leftarrow -delta$
end if
if $point.z == 0$ **then**
  $delta \leftarrow \sqrt{delta * delta}$
end if
if $delta > 0$ **then**
  $point.orientation \leftarrow false$
else
  $point.orientation \leftarrow true$
end if
return $point$

4.7 Mittelpunktbestimmung

Für den Vergleich von Merkmalen, die aus der Geometrie des Balls hervorgehen, wird ein Fixpunkt benötigt. Der Ballmittelpunkt scheint hier die logische Wahl zu sein. Die Mittelpunktbestimmung ist für Punkte auf der Ballkante konzipiert. Da die Mittelpunktbestimmung im Ballerkennungsprozess vor den Verifizierungsverfahren steht, befinden sich noch viele Punkte in der Punktwolke, die auf keiner Ballkante liegen. Für diese Punkte wird das Verfahren genauso durchgeführt, da es nicht möglich ist sie in dieser Stufe zu unterscheiden.

Ausgangspunkt der Mittelpunktbestimmung ist eine Punktwolke der Kanten des Tiefenbilds, aus der zuerst durch Auswertung der horizontalen Kanten die vertikale Achse des Balls bestimmt wird. Das Aufliegen des Balls auf dem Boden erschwert die Kantendetektion an dieser Stelle, da durch den geringen Tiefenunterschied am Auflagepunkt kein Kontrast vorhanden ist.

4.7.1 Horizontal

Von einem Punkt aus der Punktwolke wird in Richtung der Ballvertikalen durch den Mittelpunkt nach der gegenüberliegenden Kante des Balls gesucht. Die Suchrichtung ergibt sich durch Auswertung der Information aus Kapitel 4.6. Durch die Daten des Ballmodells (Kapitel 2.6) kann der Suchbereich der gegenüberliegenden Ballkante eingeschränkt werden. Um Sensorungenauigkeiten sowie Ungenauigkeiten des Ballmodells auszugleichen, wird hier eine Variable benutzt die Toleranzen zulässt. Im Folgenden k_m genannt, spannt sie den horizontalen Suchbereich um den mit dem Ballmodell berechneten Punkt in negative und positive Richtung auf. Wird eine Kante im zugelassenen Bereich gefunden, so kann der Punkt aus der Punktwolke um die Hälfte der Strecke zu

der gefundenen Kante verschoben werden und es ergibt sich ein Punkt nahe der vertikalen Achse. Wird in dem Bereich kein passender Punkt gefunden, so kann der Ausgangspunkt aus der Punktwolke entfernt werden. Eine beispielhafte Implementierung ist in Algorithmus 3 zu sehen.

Algorithm 3 shiftToBallVertical(point)

```
ret ← false
edge_found ← false
ball_diameter ← ballmodel(point.z)
if point.orientation then
  i ← point.x + ball_diameter − k_m
  j ← point.x + ball_diameter + k_m
  for i ≤ j do
    i ← i + 1
    if edge_detected(i, point.y) then
      distance_to_edge ← i − point.x
      i ← j + 1
      edge_found ← true
    end if
  end for
  if edge_found then
    point.x ← point.x + distance_to_edge/2
    ret ← true
  end if
else
  //Code for points on the right side of the Ball
end if
return ret
```

Der Rückgabewert gibt an, ob der Punkt $point$ in Richtung der Ballvertikalen verschoben wurde ($true$) oder ob der Punkt aus der Punktwolke entfernt werden kann ($false$).

4.7.2 Vertikal

Ausgehend von den Punkten, die durch den zuletzt beschriebenen Abschnitt auf die Ballvertikalen verschoben wurden, muss der Ballmittelpunkt approximiert werden. Da es auf der Vertikalen durch den Ballmittelpunkt nicht möglich ist eine Kante zwischen Boden und Ball zu erkennen, muss ein neuer Punkt ermittelt werden, der zwischen dem Punkt aus der Punktwolke und der vertikalen Achse durch den Mittelpunkt liegt. Die Auswahl dieses Punkts findet durch eine Verschiebung in Richtung einer Ballkante durch den Faktors ν statt. Von diesem neu ermittelten Punkt aus wird die Position der unteren und oberen Kante des Balls ermittelt und damit der Mittelpunkt approximiert. Dieses Verfahren ist nur begrenzt einsetzbar, da eine ausreichend gute Auflösung des Balls notwendig ist. Für die Fälle, in denen das Verfahren scheitert, ist es möglich den Ballmittelpunkt aufgrund der Entfernung der oberen Ballkante sowie des Ballmodells zu schätzen. Wichtig ist nicht, dass jeder einzelnen Punkt genau auf den Ballmittelpunkt verschoben wird, sondern dass nach der Anwendung des Verfahrens genügend Punkte in der unmittelbaren Umgebung des Ballmittelpunkts vorhanden sind. So passiert es statistisch gesehen nur sehr selten, dass in den folgenden Verifizierungsverfahren alle approximierten Mittelpunkte herausgefiltert werden.

4.8 Verifizierung durch Tiefeninformation

4.8.1 Lineare Überprüfung der Ballkantenumgebung

Durch einfaches Vergleichen der Tiefeninformation nahegelegener Pixel in horizontaler Richtung hin zur Ballvertikalen, lässt sich die Anzahl der Punkte in der Punktwolke reduzieren. Dabei werden die geometrischen Eigenschaften des Balls genutzt. Aus der Sicht eines Beobachters ist die Ballkante weiter entfernt als der Ballmittelpunkt. Auf einer Linie zwischen diesen Punkten muss der Tiefenwert von der Ballkante in Richtung des Ballmittelpunkts stetig fallen. Abbildung 12 zeigt das Verhalten der Tiefeninformation auf dem Ball.

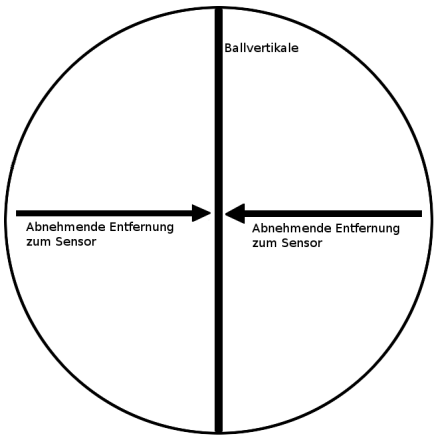

Abbildung 12: Tiefeninformation auf dem Ball

Diese Eigenschaft ergibt ein einfaches Ausschlusskriterium für Punkte, die nicht auf einer Ballkante liegen. Die Schrittweite τ sowie die Anzahl der Schritte η vom Ausgangspunkt auf einer horizontalen Linie gesehen bilden die Variablen des Verfahren. In jedem Schritt wird ein Punkt ermittelt dessen Tiefenwert kleiner als der des zuletzt ermittelte Tiefenwerts ist. Dabei muss gelten

$$\tau * \eta <= r_{pixel} \tag{28}$$

wobei r_{pixel} der Ballradius in Pixel ist. Sollte der Tiefenwert der durch den Algorithmus ermittelten Pixel nicht geringer sein als der Tiefenwert des Ausgangspunktes aus der Punktwolke, so wird der Ausgangspunkt aus der Punktwolke ausgeschlossen. Da durch das in Kapitel 4.6 beschriebene Verfahren bekannt ist, ob sich der Punkt auf der rechten oder linken Ballkante befindet, wird sichergestellt, dass die Suchrichtung in Richtung der Ballvertikalen zeigt. In Algorithmus 4 ist das Verfahren beispielhaft beschrieben.

Algorithm 4 linearEdgeVerification(point)

```
ret ← true
cache ← inf
i ← 0
if point.orientation then
  for i ≤ point.x + (τ * η) do
    i ← i + τ
    depth ← getDepth(point.x + i, point.y)
    if depth ≤ cache and depth ≠ 0 then
      cache ← depth
    else
      ret ← false
    end if
  end for
else
  for i ≤ cpoint.x + (τ * η) do
    i ← i + τ
    depth ← getDepth(point.x − i, point.y)
    if depth ≤ cache and depth ≠ 0 then
      cache ← depth
    else
      ret ← false
    end if
  end for
end if
return ret
```

4.8.2 Verifizierung durch das Ballmodell

Bei der Verifizierung durch das Ballmodell werden die Ballkanten links, rechts und oberhalb des Ballmittelpunkts überprüft. Durch den Vergleich mit dem Ballradius, der aus dem Ballmodel bestimmt wird, kann die Authentizität des Ballmittelpunktes überprüft werden. Aufgrund von Sensorungenauigkeiten und Ungenauigkeiten des Ballmodels macht es Sinn den Vertrauensbereich durch Toleranzwerte zu erweitern. Damit wird vermieden, dass gültige Ballmittelpunkte ausgeschlossen werden. Der Toleranzwert wird im Folgenden mit k_b bezeichnet. Algorithmus 5 zeigt eine beispielhafte Implementierungsmöglichkeit.

Algorithm 5 verifySizeFeature(cpoint,k_b) - Verifizierung durch das Ballmodell

$radius \leftarrow ballmodel(cpoint.z)/2$
$point \leftarrow cpoint$
$ret \leftarrow false$
$distanceToRE \leftarrow 0 \quad //Right\ Edge$
$distanceToLE \leftarrow 0 \quad //Left\ Edge$
$distanceToTop \leftarrow 0$
for $i \leq radius + k_b$ **do**
  $i \leftarrow i + 1$
  if $edge\_detected(point.x + i, point.y)\ and\ distanceToRE == 0$ **then**
    $distanceToRE \leftarrow i$
  end if
  if $edge\_detected(point.x - i, point.y)\ and\ distanceToLE == 0$ **then**
    $distanceToLE \leftarrow i$
  end if
  if $edge\_detected(point.x, point.y + i)\ and\ distanceToTop == 0$ **then**
    $distanceToTop \leftarrow i$
  end if
end for
$median \leftarrow (distanceToLE + distanceToRE + distanceToTop)/3$
if $radius - k_b \leq median \leq radius + k_b$ **then**
  $ret \leftarrow true$
end if
return ret

Ist der Rückgabewert dieses Algorithmus $true$, so wird der Punkt weiterhin in der Punktwolke gehalten. Bei $false$ wird er aus der Punktwolke der Ballmittelpunktskandidaten entfernt.

4.8.3 Radiale Überprüfung der Umgebung um einen Ballmittelpunkt

Hier wird ein Halbkreis bestimmt dessen Radius groß genug ist, um sich die Tiefeninformation zu Nutze zu machen und der kleiner ist als der Radius des Balls. Es ist es sinnvoll einen Halb- und keinen Vollkreis zu benutzen, denn am Rand des Tiefenbilds kann es vorkommen, dass die Krümmung des Balls eine optimale Tiefenwertermittlung verhindert. Die Ausrichtung des Halbkreises wird durch die Position des Ballmittelpunkts im Bild bestimmt. Dabei wird der Halbkreis immer zur vertikalen Bildachse hin berechnet. Auf diesem Halbkreis wird überprüft, ob der Tiefenwert der Punkte größer ist als der Tiefenwert des Ballmittelpunkts. Algorithmus 6 zeigt eine beispielhafte Implementierung.

Adjustradius(point.z): Berechnet den Radius so, dass der Kreisradius kleiner ist als der des Balls. Dabei ist es sinnvoll den Radius so groß wie möglich zu halten.

Algorithm 6 validateOnCircle(cpoint,deg) - Radiale Überprüfung der Umgebung

$radius \leftarrow ballmodel(cpoint.z)/2$
$radius \leftarrow adjustradius(radius, cpoint.z)$
$ret \leftarrow false$
$point.x \leftarrow cpoint.x - radius * \cos(deg * PI/180)$
$point.y \leftarrow cpoint.y - radius * \sin(deg * PI/180)$
$point.z \leftarrow depthimage(point.x, point.y)$
if $point.z > cpoint.z$ **then**
  $ret \leftarrow true$
end if
return ret

Auch hier gilt für den Rückgabewert $true$ kann der Punkt weiterhin in der Punktwolke verbleiben. Bei $false$ wird er aus der Punktwolke der Ballmittelpunktskandidaten entfernt.

4.8.4 Verifizierung durch Anzahl der Pixel auf dem Ball

Um die Anzahl der Punkte in der Punktwolke zu reduzieren, werden alle Pixel auf der oberen Ballhälfte gezählt und mit einem durch das Ballmodel geschätzen Wert verglichen. Liegt der Wert innerhalb eines Toleranzbereiches, wird der Punkt weiter in der Punktwolke behalten. Vom Ballmittelpunkt, der zuvor bestimmt wurde, aus kann zeilenweise in negative y-Richtung nach den Ballgrenzen gesucht werden. Hier kann wieder die in Kapitel 4.4 beschriebene Kantenerkennung benutzt werden. In jeder Zeile wird nach den horizontalen Kanten des Balls gesucht und dabei die Pixel gezählt. Der folgende Algorithmus 7 beschreibt das Vorgehen.

Algorithm 7 countPixelonBall(cpoint)

```
radius ← ballmodel(cpoint.z)/2
pixelamount ← 0
foundEdgeV ← false  //Vertical
foundEdgeH ← false  //Horizontal
i ← 0
repeat
  j ← 0
  i ← i + 1
  foundEdgeH ← false
  if edge_detected(cpoint.x, cpoint.y − i) then
    foundEdgeV ← true
  else
    repeat
      j ← j + 1
      pixelamount ← pixelamount + 1
      if edge_detected(cpoint.x + 1, cpoint.y) then
        foundEdgeH ← true
      end if
    until foundEdgeH ≠ true and j ≤ radius
  end if
until foundEdgeV ≠ true and i ≤ radius
foundEdgeV ← false
foundEdgeH ← false
i ← 0
repeat
  j ← 0
  i ← i + 1
  foundEdgeH ← false
  if edge_detected(cpoint.x, cpoint.y − i) then
    foundEdgeV ← true
  else
    repeat
      j ← j + 1
      pixelamount ← pixelamount + 1
      if edge_detected(cpoint.x − 1, cpoint.y) then
        foundEdgeH ← true
      end if
    until foundEdgeH ≠ true and j ≤ radius
    pixelamount ← pixelamount − 1
  end if
until foundEdgeV ≠ true and i ≤ radius
return pixelamount
```

Der Rückgabewert kann mit der durch das Ballmodell berechneten Punktzahl verglichen werden. Liegt der Wert innerhalb eines Toleranzbereichs, der durch den Toleranzwert k_c aufgespannt wird, so kann der Punkt $cpoint$ weiterhin in der Punktwolke der Ballmittelpunktkandidaten verbleiben.

4.9 Verifizierung durch Farbinformation

Allein durch Tiefeninformation ist es mit den beschriebenen Methoden nicht möglich einen Ball eindeutig zu bestimmen. Deshalb wird die Farbinformation des Balls dazu benutzt eine endgültige Verifizierung durchzuführen. Grundlage dafür ist die intuitive Farb- und Tiefenbild-Kalibrierung, die in Kapitel 4.2 beschrieben wurde. Die Farbinformation des Balls wird in einem 3-dimensionalen Array abgelegt. Die 3 Dimensionen stehen für die Farben Rot, Grün und Blau. Jede Dimension kann einen Intensitätswert von 0 - 255 annehmen. Ist eine bestimmte Farbkombination aus den drei Farben auf dem Ball enthalten, kann dies durch das Setzen eines Flags an der Farbposition im Array angegeben werden. Dadurch wird das Überprüfen der Farbe eines Punkt mit der kalibrierten Ballfarbe stark vereinfacht. Es muss nur noch im Array geschaut werden, ob das Flag an der entsprechenden Farbposition gesetzt ist oder nicht.

Um die Verifizierung der Farbinformation etwas aussagekräftiger zu machen, wird in einem quadratischen Bereich um den Ballmittelpunkt die Farbinformation jedes Pixel mit dem Array verglichen. Befindet sich dabei die Anzahl der positiv bestätigten Punkte über einem bestimmten Prozentsatz der überprüften Punkte aus dem quadratischen Bereich, so ist ein Ball erfolgreich erkannt worden.

Das Verifizierungsverfahren durch Farbinformationen ist der letzte Schritt im Ballerkennungprozess. Hier sollen die letzten, noch vorhandenen, falschen Ballmittelpunkte herausgefiltert werden.

4.10 Chronologischer Aufbau des Ballerkennungsprozesses

In diesem Abschnitt soll der chronologische Aufbau des Ballerkennungsprozess dargestellt und begründet werden. Verfahren, die während des Ballerkennungsprozess verwendet werden, wurde bereits in diesem Kapitel vorgestellt. Da die verschiedenen Verfahren zum einen aufeinander aufbauen und es zum anderen große Unterschiede in der benötigten Rechenzeit pro Ausführung je Verfahren gibt, wurde darauf geachtet sie in einer optimalen Reihenfolge ablaufen zu lassen.

Zu Beginn des Ballerkennungsprozess wird durch die Kantenerkennung eine Punktwolke mit allen vertikalen Kanten erstellt. Jede Kante wird durch einen Intensitätswert dargestellt, der die Differenz zwischen benachbarten Punkten wie in Kapitel 4.4 beschrieben angibt. Aus dieser Punktwolke werden durch Segmentierung Kanten deren Intensitätswert nicht innerhalb der definierten Grenzen (Kapitel 4.5) liegen herausgefiltert.

Das nächste Verfahren in der Kette ist ein Verfahren aus der Gruppe Merkmalsextraktion (Kapitel 2.3). “Orientierung auf dem Ball” (Kapitel 4.6) gibt für jeden verbleibenden Punkt in der Punktwolke an auf welcher Ballkante sich der Punkt befindet. Dabei wird zwischen linker und rechter Ballkante unterschieden. Ausgehend von dieser Information können die folgenden Schritte effizienter implementiert werden, da die Suche nach der gegenüberliegenden Ballkante nur in eine Richtung erfolgen muss. Zusätzlich wird durch sie das nächste Verfahren erst ermöglicht. “Lineare Überprüfung der Ballkanten Umgebung” (Kapitel 4.8.1) spezifiziert die Punkte in der Punktwolke und verkleinert die Punktwolke, um das darauf folgende Verfahren “Mittelpunktbestimmung” (Kapitel 4.7) zu beschleunigen. “Mittelpunktbestimmung” ist genau wie “Orientierung auf dem Ball” ein Verfahren der Gruppe Merkmalextraktion, das versucht jeden verbleibende Punkt in der Punktwolke in das vermutete Ballzentrum zu verschieben. Denn aufgrund der Tatsache, dass hier mit einem Kantenbild operiert wird, befindet sich kein Punkt im Ballzentrum. Der Mittelpunkt eignet sich in den nächsten Schritten als einheitlicher Fixpunkt, durch den die Implementierung der nächsten Verfahren ermöglicht wird.

Das abschließende Ziel im Ballerkennungsprozess ist eine möglichst optimale Bestimmung des Balls durch Ausschlussverfahren. Dazu werden Verfahren benutzt, die den tatsächlichen Ball aufgrund von Tiefeninformation eingrenzen. Das erste hier eingesetzte Verfahren “Verifizierung durch das Ballmodell” (Kapitel 4.8.2) überprüft die Authentizität des Ballmittelpunkts durch Suche nach den Ballkanten in plausiblem Abstand zum Ballmittelpunkt. Hier bleiben weiterhin Punkte in der Punktwolke, die keine Ballmittelpunkte sind. “Radiale Überprüfung der Umgebung um einen Ballmittelpunkt” (Kapitel 4.8.3) und im Anschluss “Anzahl der Pixel auf dem Ball” (Kapitel 4.8.4) versuchen die Punktwolke durch etwas aufwendigere Verfahren soweit wie möglich zu schrumpfen. In den meisten Fällen befinden sich auch nach diesen beiden Schritten weiterhin un-

erwünschte Punkte in der Punktwolke, weshalb in einem letzten Schritt die im Vorfeld kalibrierte Ballfarbe zur endgültigen Verifizierung des Balls herangezogen wird (Kapitel 4.9).

Die Robustheit und der Rechenzeitaufwand der Algorithmen werden durch die im folgenden Kapitel beschriebenen Tests überprüft.

5 Leistungsbewertung

Um die Leistungsfähigkeit der Algorithmen genauer deuten zu können, wurden mehrere Versuche mit bewegten sowie ruhenden Bällen durchgeführt. Im Mittelpunkt stand die Robustheit der Ballerkennung in zwei Entfernungsbereichen sowie die benötigte Ausführungszeit t.

Um den Rechenzeitaufwand zu deuten, den die einzelnen Algorithmen verursachen, wird zum Abschluss dieses Kapitels einzeln auf jeden Teil des Ballerkennungsalgorithmus eingegangen. Da die folgenden Werte aus mehreren Messungen gemittelt wurden, ist es möglich, dass rationale Zahlen anstatt den erwarteten ganzen Zahlen vorkommen.

Alle Messungen wurde auf einem Ubuntu Maverick System durchgeführt mit 4 GB Arbeitsspeicher und einem Intel i5 M460 mit 2,53 GHZ und 4 Kernen. Während der Tests wird die aus dem Tiefenbild des Kinect Sensors gewonnene Punktwolke "initiale Punktwolke" genannt. So ist beim Ballerkennungsalgorithmus jeder Punkt aus dem Tiefenbild auch in der initialen Punktwolke vorhanden. Für die Untersuchung der einzelnen Verfahren in Kapitel 5.3 wurde diese auf eine feste Anzahl an Punkten gesetzt.

5.1 Test der intuitiven Farb- und Tiefenbild-Kalibrierung

Hier wird das in Kapitel 4.2 beschriebene Verfahren mit der Standardkalibrierung durch den OpenNI [3] Treiber verglichen. Die intuitive Farb- und Tiefenbild-Kalibrierung des Kinect Sensors erfolgte mit den Werten aus Tabelle 1.

PARAMETER	X	Y	Z IN [MM]	Faktor
χ_1	256	316	2051	-
χ_2	280	310	3821	-
ϕ_1	-	-	-	0,9
ϕ_2	-	-	-	0,94
ϕ_3	-	-	-	0,95

Tabelle 1: Kalibrierungswerte der intuitiven Farb- und Tiefenbild-Kalibrierung

Der Abstand d in dem ϕ_1 gültig ist von einem zugehörigen Zentralpunkt aus gesehen ist $d \leq 50$. Für ϕ_2 gilt $50 < d \leq 150$ und für ϕ_3 gilt $d > 150$. Es wurden mehrere zufällig ausgewählte Ballpositionen bestimmt, um die Qualität der Kalibrierung zu testen.

In Abbildung 13 wurde die Position mehrerer Bälle durch das Tiefenbild berechnet und in das Farbbild transformiert. Die Kreise, die in Abbildung 13 keinem Ball zugeordnet werden können, sind falsch erkannte Bälle. Dies entsteht aufgrund der nicht verwendeten Farbinformation zur Verifizierung des Balls. Der Unterschied zwischen der nicht transformierten Position aus dem Tiefenbild (Rot) und der transformierten Position in

die Koordinaten des Farbbilds (Gelb) ist deutlich zu erkennen. Besonders bei Bällen, die in größerer Entfernung zum Kinect Sensor liegen, ist es wichtig eine funktionierende Kalibrierung zwischen der Position im Tiefenbild und der Position im Farbbild zu gewährleisten. Diese können sonst nicht durch die Farbinformation des Balls verifiziert werden. Dabei kann es ohne Kalibrierung passieren, dass der Ballmittelpunkt beim Überprüfen der Farbe im Farbbild nicht auf dem Ball liegt.

Obwohl Abbildung 13 nicht alle Bereiche des Farb- und Tiefenbilds abdeckt, kann erkannt werden, dass die Qualität der Kalibrierung für diesen Anwendungsfall ausreichend ist.

Abbildung 13: Transformation von vier zufällig ausgewählte Ballposition mit dem Verfahren aus Kapitel 4.2 vom Tiefenbild ins Farbbild

Die Qualität der Kalibrierung kann durch Hinzufügen zusätzlicher Faktoren ϕ und Anpassung deren Gültigkeitsbereiche gesteigert werden. Da diese Kalibrierung nur manuell durchgeführt werden kann, erhöht sich damit auch der Kalibrierungsaufwand. Erwähnenswert ist an dieser Stelle die Abhängigkeit der Kalibrierung von der Tiefeninformation. Die Genauigkeit der Tiefeninformation nimmt mit der Entfernung zwischen Ball und Sensor ab. Damit sinkt gleichzeitig die Güte der Kalibrierung. Zum Vergleich ist in Abbildung 14 die Ballerkennung mit der durch den OpenNI [3] Treiber bereitgestellten Kalibrierung des Kinect Sensors zu sehen.

Es ist zu erkennen, dass die Kalibrierung durch den OpenNi Treiber am Bildrand bessere Ergebnisse liefert als die im Laufe dieser Arbeit entwickelte Kalibrierung (Kapitel 4.2). Durch das Vergleichen von Abbildung 13 und 14, stellt man im Bild links oben fest, dass es auch Bereiche gibt, in denen die intuitive Farb- und Tiefenbild-Kalibrierung bessere Ergebnisse liefert. Der große Vorteil der OpenNI Kalibrierung ist der geringere Aufwand bei der Inbetriebnahme der Ballerkennung sowie die im Gesamten betrachtet höhere Güte der Kalibrierung.

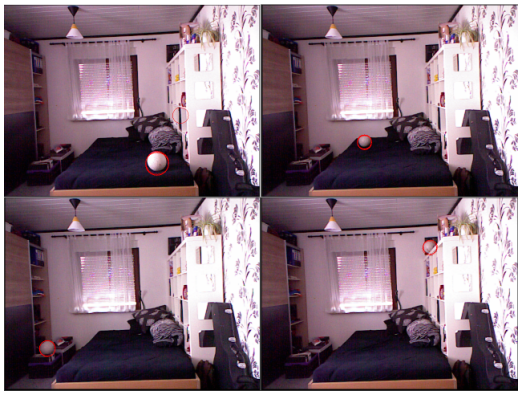

Abbildung 14: Transformation von vier Ballpositionen mit der voreingestellten Kalibrierung des Kinect Sensors

5.2 Bewegte Bälle

Während des Versuchs wurden folgende Parameter verwendet.

PARAMETER	WERT
$tresholdlow$	100
$tresholdhigh$	10000
τ	2
η	2
k_{m+}	10
k_{m-}	5
k_b	3
k_c	500

Tabelle 2: Verwendete Parameter während des Versuchs

Der Kinect Sensor wurde in einer Höhe von 42 cm aufgestellt, dies entspricht in etwa der Position, die der Sensor beim RoboCup 2011 auf dem Torwartroboter hatte. Um die Ballerkennung durch Farberkennung abzusichern, wurde der Kinect Sensor mit der intuitiven Farb- und Tiefenbild-Kalibrierung (Kapitel 4.2) kalibriert. Das Farb- und Tiefenbild des Versuchaufbaus ist in Abbildung 15 zu sehen.

Abbildung 15: Farb- und Tiefenbild des Versuchsaufbaus

Die zwei sich überlappenden Entfernungsbereiche sind 50 cm - 300 cm sowie 50 cm - 600 cm. Im Folgenden werden diese “Nahbereich” und “Fernbereich” genannt.

Um die Anzahl der Punkte in der initialen Punktwolke zu verringern, wurden nur Teile der von dem Kinect Sensor stammenden Punktwolke betrachtet. Es wurde je Zeile nur jeder $x - te$ Punkt bzw. jede $y - te$ Zeile beachtet. Damit wurde der Bildausschnitt nicht verändert, sondern nur die Auflösung künstlich reduziert. Dies wirkt sich ausschließlich auf die Kantenerkennung aus. Die darauf aufbauenden Verfahren benutzen das Tiefenbild mit der maximalen Auflösung von 640x480 Pixel.

Es wurden pro Messpunkt zehn Bälle auf den Sensor zugerollt oder geworfen, um repräsentative Werte für die Ballerkennungsrate bzw. die benötigte Ausführungszeit in Abhängigkeit der Auflösung zu bekommen. Vom Eintreten in den Bereich des Sensors bis zum Austreten wurde die Anzahl der Bilder und die Anzahl der gefundenen Bälle gezählt. Aus den gesammelten Daten wurde jeweils der Mittelwert gebildet.

5.2.1 Rollende Bälle

Rollend bedeutet, dass der Ball ständig Bodenkontakt hat. Dabei ist im Tiefenbild keine Kante zwischen Boden und Ball erkennbar. Die Werte für die Ballerkennungsrate bzw. den Rechenzeitaufwand in Abhängigkeit der Auflösung sind in Abbildung 16 und 17 zu sehen.

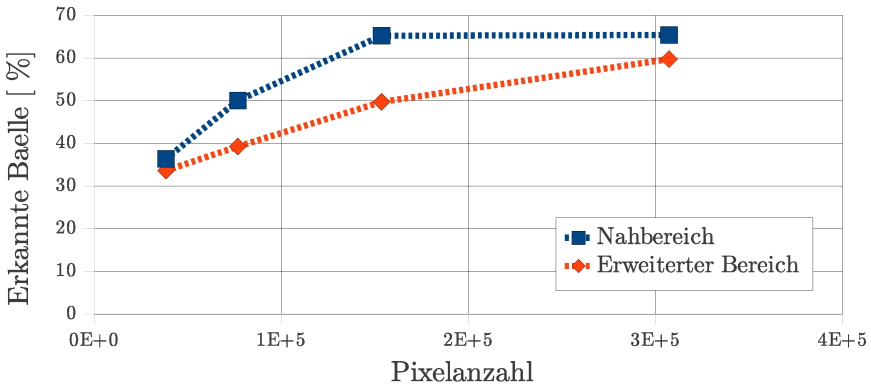

Abbildung 16: Erkannte, rollende Bälle in Prozent in Abhängigkeit der Auflösung für die initiale Kantenerkennung

Erwartungsgemäß nimmt die Ballerkenungsrate durch die größere Entfernung zwischen Ball und Sensor ab. Durch eine hohe Auflösung der initialen Punktwolke lässt sich ein ähnliches Niveau wie bei der Ballerkennung im Nahbereich erreichen.

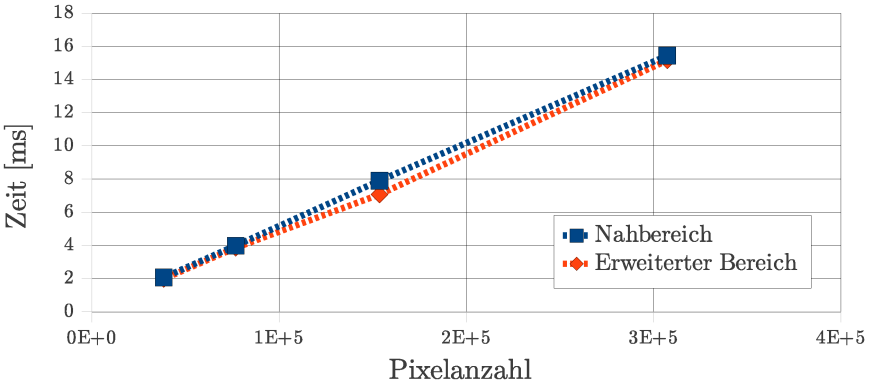

Abbildung 17: Benötigte Ausführungszeit des Ballerkennungsalgorithmus für rollende Bälle in Abhängigkeit der Auflösung für die initiale Kantenerkennung

Die geringfügig kürzere Ausführungszeit des Algorithmus im erweiterten Bereich lässt sich auf die geringere Informationsdichte in größerer Entfernung zurückführen. Es sind weniger Punkte auf dem Ball vorhanden, die überprüft werden müssen.

5.2.2 Fliegende Bälle

Fliegende Bälle befinden sich die meiste Zeit über in der Luft. Nur wenn sie “aufprallen” haben sie Bodenkontakt. Im RoboCup ist es wichtig diese Bälle zu erkennen, da die gegnerischen Roboter in der Lage sind Bälle im großen Bogen sehr präzise ins Tor zu spielen. Abbildung 18 und 19 veranschaulichen die Ballerkennungsrate bzw. Rechenzeit in Abhängigkeit der Auflösung.

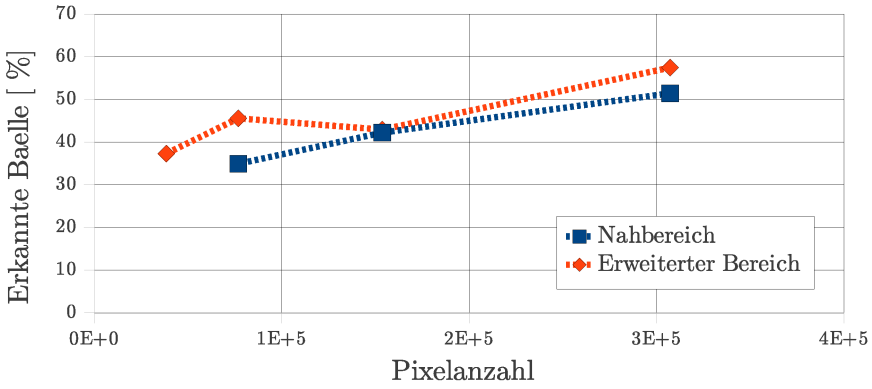

Abbildung 18: Erkannte, fliegende Bälle in Prozent in Abhängigkeit der Auflösung für die initiale Kantenerkennung

Auffallend ist hier, dass die Ballerkennungsrate im Nahbereich tendenziell schlechter ist als im erweiterten Bereich. Eine Ursache hierfür liegt vermutlich an dem erhöhten Geschwindigkeitsvektor des Balls, welcher aufgrund der Vertikalbewegung verstärkt wird. Ein weiterer Faktor ist die Nähe des Balls zum Sensor. Betrachtet man zwei Bälle deren Geschwindigkeit konstant ist und lässt sie in unterschiedlichen Entfernungen von oben nach unten durch das Bild fallen, so braucht der Ball der näher am Sensor liegt weniger Zeit, um den Bildausschnitt zu durchqueren. Grund hierfür ist der kleinere Bildausschnitt, der abgebildet wird. Die benötigte Ausführungszeit verhält sich sehr ähnlich wie bei rollenden Bällen.

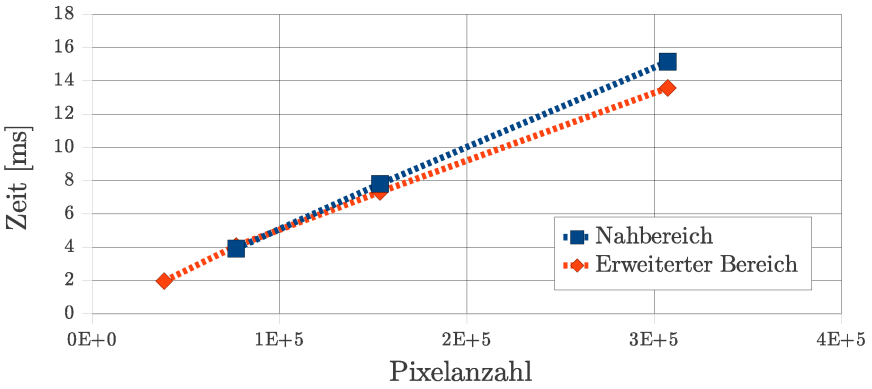

Abbildung 19: Benötigte Ausführungszeit des Ballerkennungsalgorithmus für fliegende Bälle in Abhängigkeit der Auflösung für die initiale Kantenerkennung

5.2.3 Fazit

Es ist gut zu erkennen, dass bei rollenden Bällen in dem erweiterten Bereich die Ballerkennungsrate abnimmt. Das liegt sowohl an den Ungenauigkeiten des Kinect Sensors, der einen mittleren Fehlerwert in 4 m Entfernung von 10 cm hat [2], als auch an der geringeren Datendichte, die durch die größere Entfernung zwischen Ball und Kinect Sensor zu Stande kommt. In weiterer Entfernung verhält sich die Fehlerrate des Kinect

Sensors stark nichtlinear, was eine Modellierung erschwert.

Die schlechtere Ballerkennungsrate der fliegenden Bälle entsteht durch mehrere Fehlerquellen. Der Ball wurde bei den Messungen so geworfen, dass er mehrere Male aufgeprallt ist. Beim Aufprallen verändert der Ball seine Form und wird platter bzw. er wird schmaler beim Wegspringen vom Boden. Da die Maße des Balls in diesen Phasen von denen des Ballmodels abweichen, erhöht sich die Wahrscheinlichkeit, dass ein Ball nicht erkannt wird. Auffällig ist das starke Abweichen der Ballerkennungsrate im Nahbereich zwischen fliegenden und rollenden Bällen. Dies könnte ebenfalls an dem soeben beschriebenen Sachverhalt liegen und durch den höheren Geschwindigkeitsvektor des Balls infolge der Vertikalbewegung der fliegenden Bälle noch verstärkt werden.

Um die Robustheit der Ballerkennung bewerten zu können, muss auf die Bedeutung der prozentualen Ballerkennungsrate eingegangen werden. Der Kinect Sensor liefert 30 Bilder pro Sekunde. Das bedeutet, dass ca. alle 33 ms ein neues Bild ausgewertet werden kann. Die maximal benötigte Ausführungszeit des Ballerkennungsalgorithmus beträgt laut Tabelle 6 15,44 ms. Das heißt, es ist genug Zeit vorhanden jedes der 30 Bilder auszuwerten. Aus Tabelle 2 ergibt sich für eine Auflösung von 640x240 eine Ballerkennungsrate von knapp 50 %. Hiermit kann die Annahme getroffen werden, dass im Durchschnitt alle zwei Bilder der Ball erkannt wird. Daraus ergibt sich eine mittlere Dauer für die Ballerkennung von ca. 66 ms. Da die Beschleunigungsfähigkeit des Torwartroboters um ein Vielfaches langsamer ist, kann die Ballerkennung je nach Konfiguration als ausreichend robust bezeichnet werden.

Dennoch macht es Sinn bei der Wahl der Auflösung die benötigte Rechenzeit zu betrachten. Auffallend ist, dass dabei die Geschwindigkeit der Ballerkennung sich annähernd proportional zur Abnahme der Pixelanzahl der initialen Punktwolke verhält. Das bedeutet, dass in der angegebenen Konfiguration des Ballerkennugnsalgorithmus die benötigte Ausführungszeit fast ausschließlich von der Auflösung bzw. Größe der initialen Punktwolke abhängig ist. Die effektivste Möglichkeit Rechenzeit zu sparen ist somit das Verringern der Auflösung der initialen Punktwolke.

5.3 Einzelne Betrachtung der Verfahren

Die Ballerkennung setzt sich aus mehreren Teilalgorithmen der Arbeit zusammen, welche sich durch verschiedene Parameter beeinflussen lassen (Kapitel 4). In diesem Teil wird auf die Ausführungszeit und Güte der einzelnen Algorithmen in Abhängigkeit der einstellbaren Parameter bzw. Toleranzwerte eingegangen. Dazu wurde ein Ball in 149 cm Entfernung vom Kinect Sensor platziert. Die Größe der Punktwolke, die jedem Verfahren übergeben wurde, betrug 80.000 Punkte. Um die Güte zu bestimmen, wurde ein Rechteck um den Ball herum bestimmt bzw. um den Ballmittelpunkt, je nachdem was bei den spezifischen Verfahren sinnvoller war. Alle Versuche wurden mit derselben

Entfernung zwischen Sensor und Ball durchgeführt. Zu Vergleichszwecken wurde die Anzahl der verbleibenden Punkte nach Anwendung des jeweiligen Algorithmus innerhalb und außerhalb des rechteckigen Bereichs bestimmt.

Die Einführung des Quotienten q, der aus den Punkten im Ballzentrum p_z und Punkten im Bild p_b berechnet wird, stellt ein dimensionsloses Verhältnismaß dar.

$$q = \frac{p_z}{p_b}, \tag{29}$$

Er erleichtert die Analyse der Verfahren. q steht für die Effektivität der Reduzierung der Punkte in der Punktwolke. Ist der Quotient 1, so bedeutet das, dass nur gewünschte Punkte bestätigt wurden. 0 hingegen bedeutet, dass keine gewünschten Punkte bestätigt wurden, wobei zu beachten ist, dass $p_b > 0$ sein muss.

Durch den Quotienten aus Rechenzeit t und Punkte im Ballzentrum p_z ergibt sich ein Maß mit der Einheit $[\frac{Zeit}{Punkt}]$.

$$v = \frac{t}{p_z} \tag{30}$$

Hiermit lässt sich vergleichen wie viel Zeit benötigt wird, um einen Punkt innerhalb eines sinnvoll gewählten Bereichs zu berechnen.

Nachdem der jeweilige Verifizierungsalgorithmus angewendet wurde, müssen weniger Punkte in der Punktwolke vorhanden sein als davor. Durch die Anzahl dieser noch verbliebenen Punkte lassen sich Aussagen über die Qualität des Algorithmus treffen. In Abbildung 20 ist der Versuchsaufbau zu sehen.

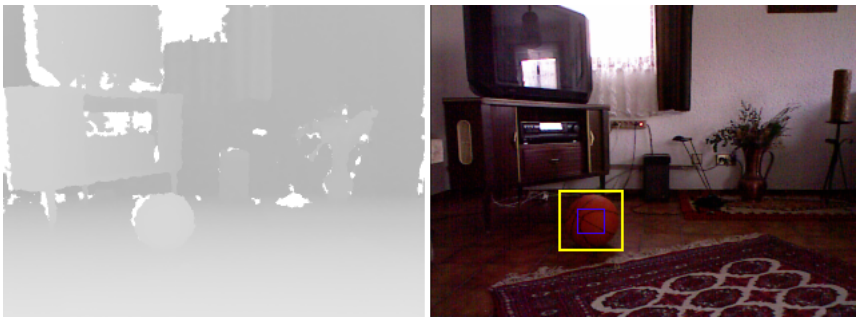

Abbildung 20: Versuchsaufbau

5.3.1 Initiale Kantenerkennung, Segmentierung und Orientierung auf dem Ball

Als erster Schritt nach der initialen Kantenerkennung schrumpft das Segmentierungsverfahren die Punktwolke um ca. 96,4 % zusammen (Tabelle 3).

Auflösung	Prozentuale Abnahme der Punktwolke
640 x 480	-96,45%
640 x 240	-96,44%
320 x 240	-96,44%
320 x 120	-96,29%

Tabelle 3: Verkleinerung der Punktwolke in Prozent

Bei diesem Schritt wird zusätzlich jedem Punkt seine Orientierung auf dem Ball (Kapitel 4.6) zugeordnet, was in der Praxis zusätzliche Rechenzeit benötigt. Durch die Verkleinerung der Punktwolke können in den darauf folgenden Schritten kompliziertere und mehr Rechenzeit pro Aufruf benötigende Algorithmen zum Einsatz kommen.

Die Anzahl der Funktionsaufrufe hängt direkt von der Auflösung ab und ergibt sich durch die in Kapitel 4.4 beschriebene Formel $O(M*(N\text{-}1))$. Was bei einer Auflösung von 640x480 306720 Aufrufe ergibt. Wie schon in Kapitel 5.2.3 gesehen, macht die initiale Kantenerkennung den Großteil der verbrauchten Ausführungszeit des Ballerkennungsalgorithmus aus. Aus diesem Grund ist es wichtig den Algorithmus so kompakt und effizient wie möglich zu halten.

Ein wesentlicher Faktor ist hier die Wahl des Kantenfilters. Ein 3x3-Filter, der vertikale und horizontale Kanten erkennt, ist aufgrund der zeitkritischen Anwendung nicht angebracht. Da in dieser Arbeit schon ein sehr rechenzeitschonender Kantenfilter implementiert wurde (Kapitel 4.4), ist die Reduzierung der Punkte der initialen Punktwolke am erfolgsversprechensten.

Probleme könne Bälle bereiten, die von einem Roboter geführt werden, da hier der untere Schwellwert sehr klein gewählt werden muss, um auch Kanten mit geringerem Kontrast zu erkennen. Dies führt dazu, dass weniger Punkte ausgeschlossen werden können und sich die Ausführungszeit des Gesamtalgorithmus erhöht.

Da diese Bälle von der omnidirektionalen Kamera des Torwartroboters durch Farbsegmentierung erkannt werden können, wurde kein besonderer Wert auf das Erkennen geführter Bälle gelegt. In Abbildung 21 ist noch einmal die Verkleinerung der Punktwolke durch das Segmentierungsverfahren grafisch dargestellt.

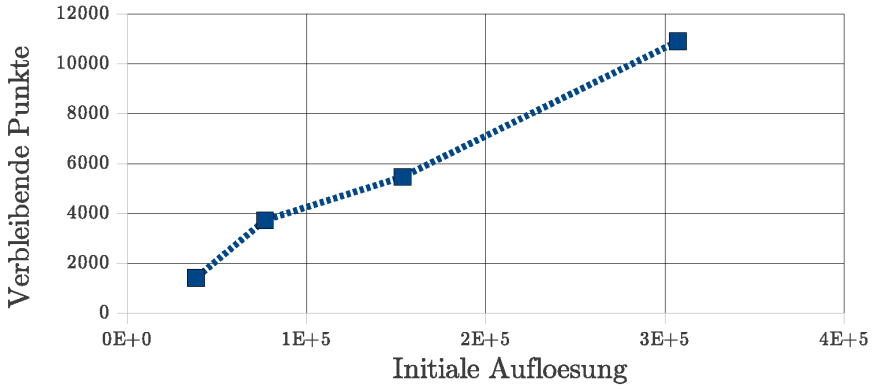

Abbildung 21: Das Segmentierungsverfahren berechnet mit den Werten $tresholdlow = 100$ und $tresholdhigh = 10000$

5.3.2 Test der linearen Verifizierung der Ballkante

Das gelbe Quadrat in Abbildung 20 zeigt den hier verwendeten Bereich der Punkte in unmittelbarer Ballumgebung. Bei diesem Verfahren muss darauf geachtet werden, dass Gleichung 23 erfüllt ist. In dem hier vorliegenden Fall bedeutet das

$$\tau * \eta <= 30, \tag{31}$$

denn für $z = 149\ cm$, $\delta = 58°$, $x = 640$ und $r = 12,5\ cm$ ergibt sich durch Gleichung 18 $\mu = 60,52$. Was für den Radius des Balls in Pixel $r_{pixel} = 30$ bedeutet. In Abbildung 22 ist zu sehen, dass die Anzahl der verbleibenden Punkte in unmittelbarer Ballumgebung nach Anwendung des Verfahrens mit der Schrittweite vier deutlich geringer wird. Das hängt mit dem Nichterfüllen von Gleichung 23 zusammen. Je nachdem in welcher Entfernung Bälle erkannt werden sollen, müssen τ und η angepasst werden.

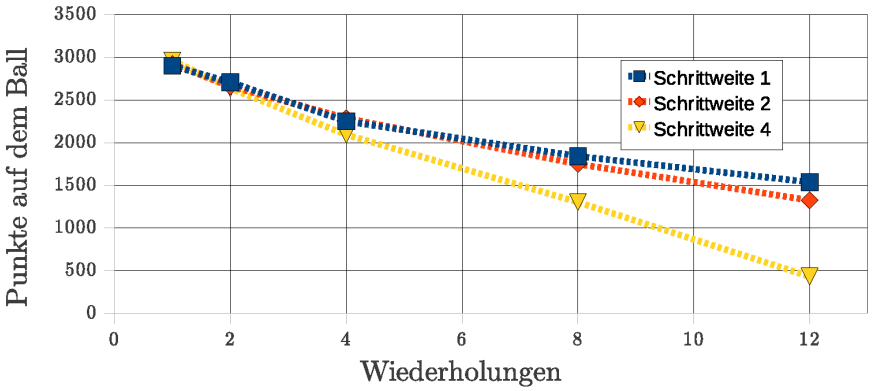

Abbildung 22: Verbleibende Punkte in der Ballumgebung nach Anwendung des Verfahrens

Das Ziel des Verfahrens ist es, die Punktwolke so weit wie möglich zu verkleinern ohne dabei Punkte aus der Ballumgebung auszuschließen. Die besten Werte für die Verklei-

nerung der Punktwolke werden erzielt, wenn $\tau * \eta$ maximal ist, dies ist in Abbildung 23 zu sehen. Trotzdem darf die Grenze r_{pixel} nicht überschritten werden, was hier aber bei der Schrittweite vier passiert.

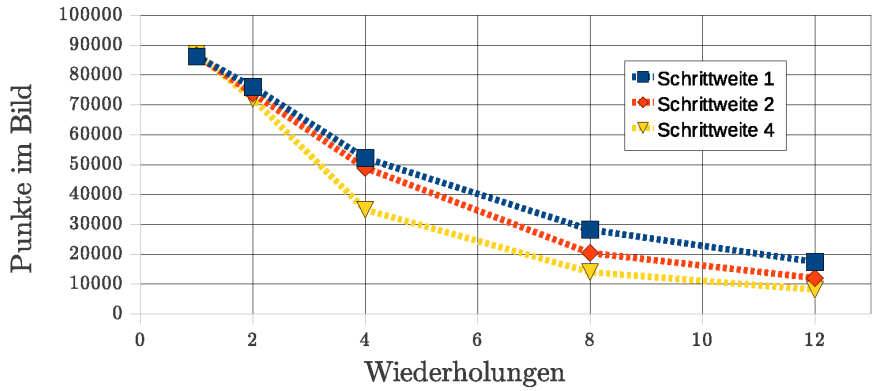

Abbildung 23: Verbleibende Punkte in der Punktwolke nach Anwendung des Verfahrens

Die benötigte Rechenzeit in Abhängigkeit von $\tau * \eta$ ist in Abbildung 24 dargestellt. Durch die geringe Komplexität des Verfahrens steigt sie moderat an und verhält sich für alle Schrittweiten ähnlich.

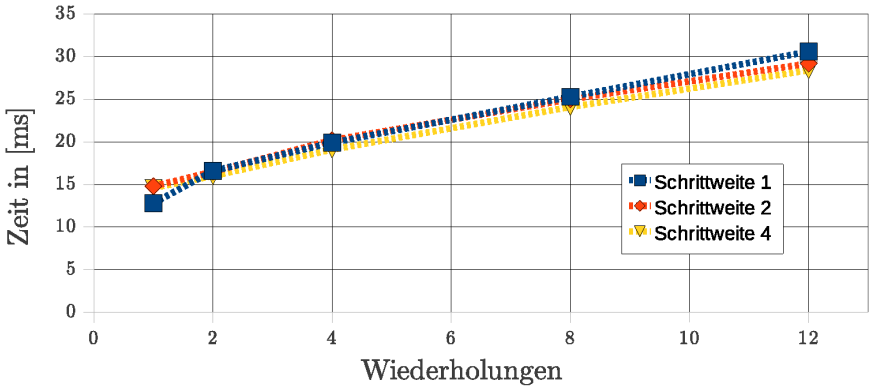

Abbildung 24: Benötigte Zeit des Verfahrens in Abhängigkeit der Schrittweite bzw. der Wiederholungen

Durch das Betrachten des Quotienten q, lässt sich der optimale Wert von $\tau * \eta$ ermitteln. Für $\tau = 2$ und $\eta = 12$ ergibt sich $q_{max} = 0,11$. Durch einen Vergleich des q_{max}-Werts mit den anderen Verfahren (Tabelle 10) ergibt sich das zweitbeste Ergebnis im Ballerkennungsalgorithmus.

In der Praxis ist es nötig Bälle in einer Entfernung von bis zu sechs Meter zu erkennen. Wodurch sich auch q_{max} verändert. Durch das Einsetzen der oben genannten Werte in Gleichung 23 und der Wahl von $z = 600\ cm$, ergibt sich für $\mu = 15,03$ und dadurch für

$r_{pixel} = 7$. Das heißt

$$\tau * \eta <= 7 \tag{32}$$

Damit stellt sich das Optimum des Verfahren durch das neue $q_{max} = 0,04$ bei $\tau = 1$ und $\eta = 12$ ein.

In der Praxis würden bei einer optimal gewählten Konfiguration nur Punkte erlaubt, die auf der Horizontalen durch den Ballmittelpunkt liegen. Da es für die folgenden Schritte von Vorteil ist, wenn sich Punkte auf der gesamten Ballkante befinden, ist es sinnvoll den Wert $\tau * \eta$ kleiner zu halten als es das Optimum vorgibt.

5.3.3 Test der Mittelpunktbestimmung

Da dies ein Verfahren aus der Gruppe der Merkmalsextraktion ist und keine Punkte ausschließt, werden nur Punkte innerhalb eines Bereichs um den Ballmittelpunkt betrachtet. Der Bereich ist als blaues Quadrat in Abbildung 20 zu sehen. Der Toleranzwert k_m des Verfahrens erweitert den Bereich in positive sowie in negative Richtung von einem durch das Ballmodell berechneten Punkt aus gesehen. In Abbildung 25 ist gut zu erkennen, dass erst ab einem Toleranzwert von 10 die Anzahl der Punkte in der Nähe des Ballzentrums ansteigt.

Bei diesem Versuch stand nur ein Basketball zur Verfügung dessen Radius größer ist als der eines Fußballs. Das Ballmodell hingegen war auf die Größe eines Fußballs eingestellt. Der Radius des Basketballs betrug 12,5 cm. Ein Fußball hat im RoboCup üblicherweise einen Radius von 10 cm. Werden diese Werte mit Hilfe von Gleichung 18 in Pixel umgerechnet, so ergibt sich für den Radius des Basketballs in Pixel $r_{bb} = 30$ sowie für den Radius des Fußballs in Pixel $r_{fb} = 24$. Wird Abbildung 25 mit diesem Hintergrund betrachtet, so ist anzunehmen, dass der Anstieg der Punkte ab einem Toleranzwert 10 auch von dem zu großen Durchmesser des Balls kommt. Da $r_{bb} - r_{fb} = 6$ macht es Sinn den Anstieg der Punkte im Ballzentrum bei korrekt eingestelltem Ballmodell ab $k_m = 4$ anzunehmen.

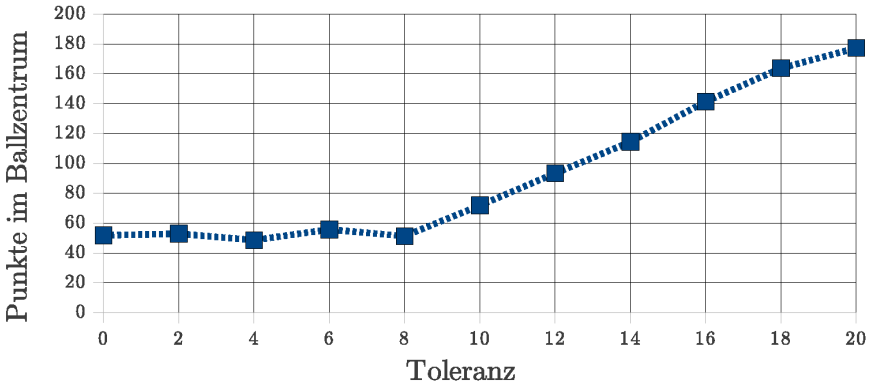

Abbildung 25: Punkte im Ballzentrum in Abhängigkeit der Toleranz k_m

Die benötigte Rechenzeit in Abbildung 26 nimmt mit höheren Toleranzwerten zu. Für die maximal zu überprüfenden Punkte n_{max} in Abhängigkeit des Toleranzwerts k_m gilt

$$n_{max} = 2 * k_m + 1 \tag{33}$$

wobei $k_m \in \mathbb{N} \cup \{0\}$. Dabei ist die Anzahl der zu überprüfenden Punkte in der Regel kleiner, da das Verfahren bei einer gefundenen Kante abbricht.

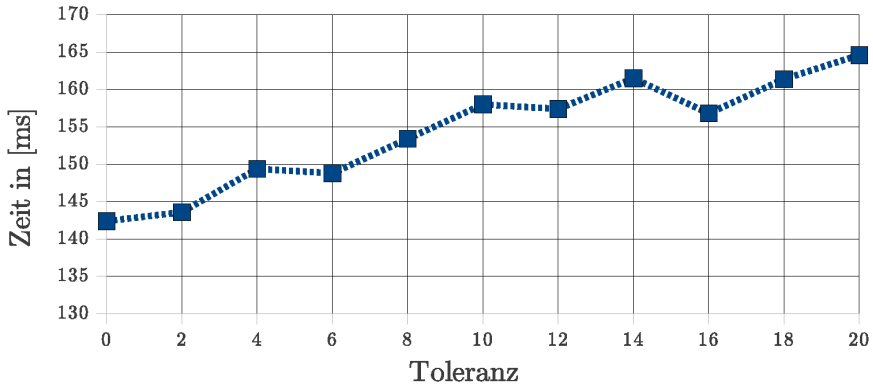

Abbildung 26: Ausführungszeit der Mittelpunktbestimmung in Abhängigkeit von k_m

Es lässt sich hier schon erkennen, dass es sinnvoll ist dem Verfahren eine relativ kleine initiale Punktwolke zu übergeben, da die Dauer des Verfahrens deutlich höher ist als die von anderen Verfahren im Ballerkennungsalgorithmus.

5.3.4 Test der Verifizierung durch das Ballmodell

Bei diesem Verfahren wirkt sich der Toleranzwert k_b nicht auf die Rechenzeit t aus, weil keine zusätzlichen Punkte überprüft werden müssen. Deshalb kann diese konstant mit $t = 6,5$ ms angegeben werden. t ist nur abhängig von der initialen Punktwolke, die dem Verfahren übergeben wird.

Das Verfahren liefert sehr gute Ergebnisse für die Punkte in der Nähe des Ballzentrums. Aus Abbildung 27, für $k_b > 10$, ergeben sich keine zusätzlichen Verbesserungen für die Punkte im Ballzentrum. Allerdings erhöht sich die Anzahl der verbleibenden Punkte in der Punktwolke wie erwartet stetig mit steigender Toleranz.

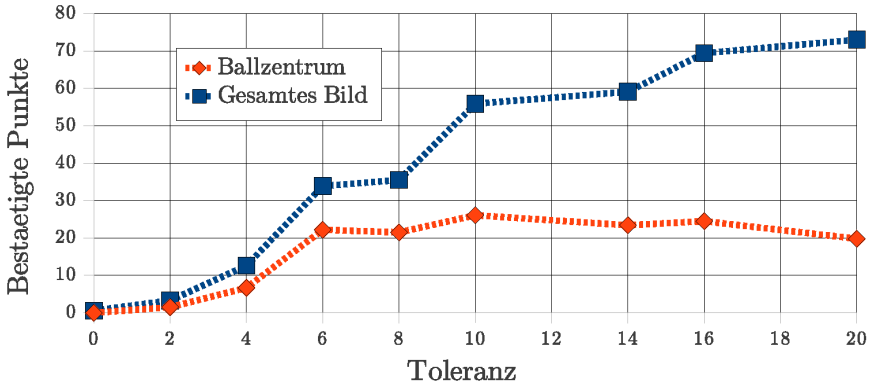

Abbildung 27: Punkte im Ballzentrum in Abhängigkeit der Toleranz k_b

Wird der Quotient q betrachtet und die Bedingung $p_z \geq 20$ aufgestellt, so ergibt sich für $q_{max} = 0,655$ bei $k_b = 6$, was aufgrund der konstanten Rechenzeit t als optimal bezeichnet werden kann.

5.3.5 Test der radialen Überprüfung der Umgebung um einen Ballmittelpunkt

Dieses Verfahren versucht möglichst viele Punkte im Ballzentrum zu verifizieren. Die maximale Anzahl der zu überprüfenden Punkte, pro Punkt aus der initialen Punktwolke, p_n ist in diesem Verfahren 180, was einem halben Kreis entspricht. Das Verdoppeln der Schrittweite s bewirkt eine Halbierung von p_n, was eine quadratische Abnahme der Ausführungszeit des Verfahrens in Abhängigkeit der Schrittweite annehmen lässt. In Abbildung 28 und 32 (siehe Anhang) sind die Diagramme für die verbleibenden Punkte in der Punktwolke sowie die Punkte um das Ballzentrum abgebildet.

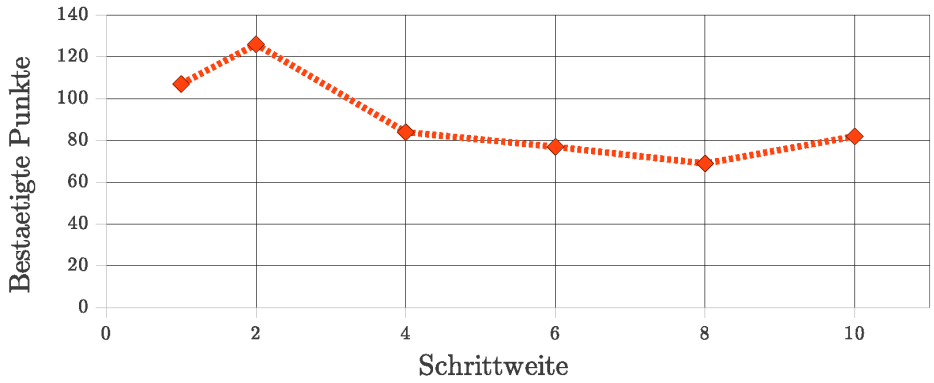

Abbildung 28: Punkte im Ballzentrum in Abhängigkeit der Schrittweite s

Für den Quotienten q_{max} ergibt sich $q_{max} = 0,0071$ bei $s = 2$. In Tabelle 10 lässt sich gut erkennen, dass dieses Verfahren nicht dazu geeignet ist die Punktwolke effizient zu verkleinern. Die Rechenzeit t ist in Abbildung 29 zu sehen. Sie fällt annähernd quadratisch mit der Zunahme von s ab, was aufgrund der quadratischen Abnahme von p_n erwartet wurde.

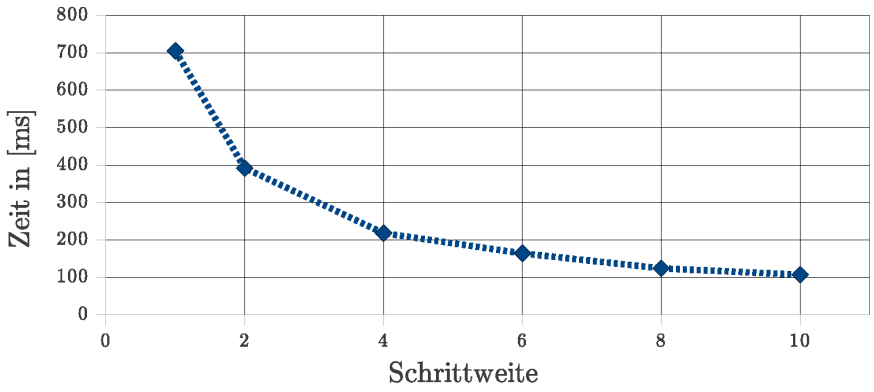

Abbildung 29: Ausführungszeit der radialen Überprüfung der Umgebung in Abhängigkeit der Schrittweite s

In diesem Verfahren muss abgewägt werden, ob es sinnvoll ist den Toleranzwert aufgrund von q_{max} als optimal zu bezeichnen. Durch die annähernd quadratische Abnahme der Rechenzeit t ist es sinnvoll diese in die Betrachtung mit einzubeziehen. Da das Verfahren im Ballerkennungsprozess zur Verifizierung der Ballmittelpunkte verwendet wird und im Vorfeld die Punktwolke schon stark verkleinert worden ist, ist es sinnvoll s groß zu wählen. Die folgende Tabelle 4 gibt den Quotienten q in Abhängigkeit der Schrittweite s an.

SCHRITTWEITE s	QUOTIENT q
1	0,0066
2	0,0071
4	0,0047
6	0,0041
8	0,0037
10	0,0042

Tabelle 4: q-Werte des Verfahrens

Da q für $s = 4, 6, 8$ sich nur wenig verändert und die absolute Zahl der Punkte in der Nähe des Ballmittelpunkts auch nur wenig schwankt, ist aufgrund der optimalen Nutzung der Ressourcen $s = 8$ die optimale Wahl. Die Erhöhung der Punkte sowohl in der Nähe des Ballmittelpunkts als auch im gesamten Bild spricht gegen eine Verwendung von $s = 10$, da hier die Vermutung nahe liegt, dass das Kriterium des Verfahrens zu schwach gewählt ist.

5.3.6 Test der Verifizierung durch Anzahl der Punkte auf dem Ball

Das Zählen der Hälfte der Punkte auf dem Ball ist ein Verfahren, das sehr viel Rechenzeit verbraucht. Mit $t = 3053,5\ ms$ ist es das langsamste hier eingesetzte Verfahren. Auch hier hat die Toleranz k_c keinen Einfluss auf die benötigte Rechenzeit t, weshalb

sie konstant ist. Von einem Zahlenwert aus gesehen, der für die Anzahl der Punkte auf der oberen Ballhälfte steht, erweitert k_c den zulässigen Toleranzbereich in positive sowie negative Richtung. In Abbildung 30 und 33 (siehe Anhang) sind die bestätigten Punkte in der Nähe des Ballzentrums bzw. die Anzahl der Punkte im Bild abgebildet. In Abbildung 30 ist ein deutlicher Anstieg im Bereich zwischen $k_c = 30$ bis 50 zu sehen. Für den Quotient q ergibt sich ein Maximum von $q_{max} = 0,032$ bei $k = 50$, was hier das Optimum darstellt. Für die Reduzierung der Punktwolke ist das Verfahren aufgrund der großen Ausführungszeit t nicht geeignet. Als zusätzliches Verifizierungsverfahren ist es das letzte in der chronologischen Reihenfolge der Verifizierungsverfahren durch Tiefeninformation.

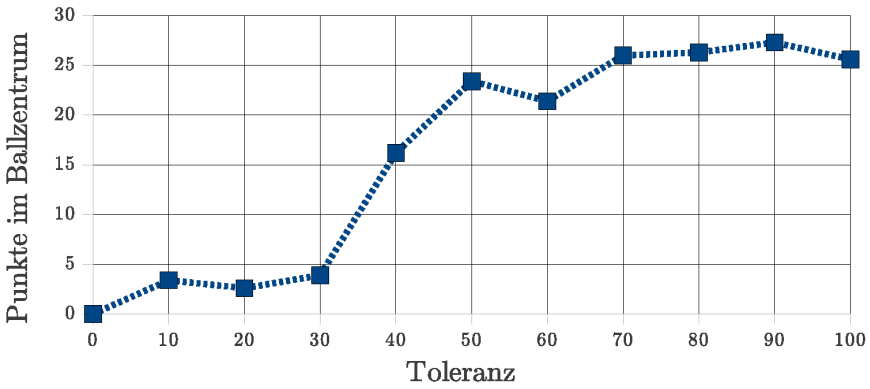

Abbildung 30: Anzahl der Punkte im Ballzentrum in Abhängigkeit der Toleranz k_c

5.3.7 Vergleich der Verifizierungsverfahren

Durch den in Kapitel 5.3 beschrieben Quotient v können die Verifizierungsverfahren in Bezug auf die benötigten Ressourcen bewertet werden. Als Grundlage dafür werden die in den vorangegangenen Abschnitten beschriebenen Werte, die sich für jedes Verfahren unterscheiden, betrachtet.

LVB steht für lineare Verifizierung der Ballkante (Kapitel 4.8.1), mit $\tau = 2$, $\eta = 2$
VB steht für Verifizierung durch das Ballmodell (Kapitel 4.8.2), mit $k_b = 6$
RUUB steht für radiale Überprüfung der Umgebung um einen Ballmittelpunkt (Kapitel 4.8.3), mit $s = 8$
VAPB steht für Verifizierung durch Anzahl der Punkte auf dem Ball (Kapitel 4.8.4), mit $k_c = 50$

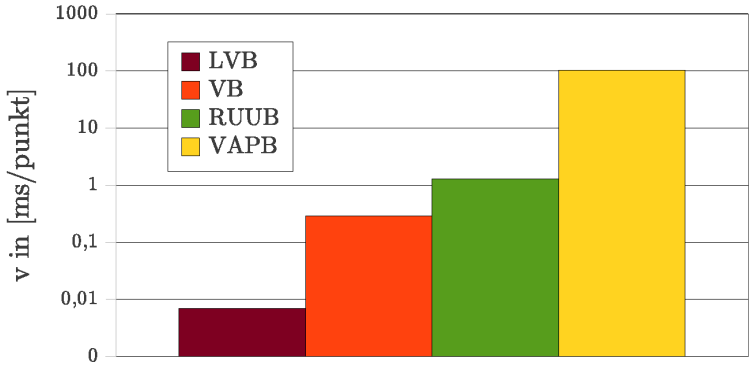

Abbildung 31: Quotient v der Verfahren auf einer logarithmischen Skala aufgetragen

Abbildung 31 veranschaulicht welches Verfahren am geeignetsten ist, um eine Vorauswahl der Punkte aus großen Punktwolken zu treffen. Je kleiner v ist desto früher sollte es in der chronologischen Reihenfolge des Ballerkennungsprozesses stehen. Ein Vergleich mit Kapitel 4.10 bestätigt, dass die Algorithmen unter Berücksichtigung des Aspekts der Ausführungszeit in der richtigen Reihenfolge verwendet werden.

5.4 Genauigkeit der Ballposition

Um die Güte der Ballerkennung bewerten zu können, werden durch den Kinect Sensor ermittelte Ballpositionen mit gemessenen Ballpositionen verglichen. Der Kinect Sensor wurde dafür in einer Höhe h angebracht, wobei $h = 42\ cm$ beträgt. In jedem Entfernungsabschnitt wurden jeweils fünf Bälle platziert. Für jeden Ball wurde die Abweichung vom gemessenen Wert in Prozent berechnet. Um ein repräsentatives Ergebnis zu erzielen, wurde der Durchschnitt aus den fünf gemessenen Bällen gebildet. Gemessen wurde der Abstand d' zwischen Ball und Standsockel des Sensors. Der Winkel γ, der die Abweichung von der Orthogonalen in Bezug auf den Sensor angibt, wurde ebenfalls gemessen. Die Entfernung zwischen Sensor und Ball d kann durch

$$d = \sqrt{h^2 + d'^2} \tag{34}$$

bestimmt werden. Die Entfernungsbestimmung durch den Sensor beruht auf dem in Kapitel 2.5 beschriebenen Verfahren. In Tabelle 5 sind die Abweichungen in den verschiedenen Entfernungsbereichen dargestellt. Auffallend sind die starken Abweichungen in dem Bereich 500 - 2000 mm. Sie können durch Ungenauigkeit während des Messens entstanden sein und sich aufgrund der kleineren Absolutwerte stärker ausgewirkt haben.

Entfernungsbereich in mm	Abweichung in %
500 - 1000	6,32
1000 - 2000	1,96
2000 - 3000	0,79
3000 - 4000	0,88
4000 - 5000	2,7
5000 - 6000	2,8

Tabelle 5: Abweichungen zwischen Ballpositionen aus dem Kinect Sensor und gemessenen Ballpositionen

Der Unterschied zwischen den Bereichen 2000 - 4000 mm und 4000 - 6000 mm verhält sich dagegen wie erwartet. Die Zunahme der Ungenauigkeit ist auf den Kinect Sensor zurückzuführen, der in größere Entfernung stärkere Abweichungen von der gemessenen Entfernung aufweist.

6 Schlußfolgerungen und Ausblick

Das Ergebnis der Arbeit ist eine Echtzeit-Ballerkennung, die hauptsächlich auf Tiefeninformation aufbaut. Die Farbinformation dient ausschließlich als Unterstützung, um die wenigen Punkte, die sich nach der Bearbeitung der initialen Punktwolke durch die auf Tiefeninformation beruhenden Verfahren noch in der Punktwolke befinden, eindeutig einem Ball zuordnen zu können bzw. auszuschliessen.

Die Ballerkennung wurde speziell für die Verwendung im RoboCup Szenario entwickelt und an die dort herrschende Gegebenheiten angepasst. Mit einer maximalen Rechenzeit pro Ausführung des Ballerkennungsalgorithmus von 15,44 ms wurde die zeitliche Vorgabe zur Integration in die Architektur des RoboCup Teams der Universität Stuttgart eingehalten. Mit Einbußen in der Robustheit des Algorithmus von 15-20 % lassen sich Zeiten von ca. 4 ms pro Iteration erreichen. Dies kann ausschlaggebend sein, um den Prozessor innerhalb des 33,3 ms Zyklus für andere Funktionen frei zu halten.

Die Robustheit des Algorithmus mit maximal 65,37 % (Tabelle 6) bei rollenden Bällen scheint auf den ersten Blick nicht sonderlich hoch zu sein. Sie ist aber eine Steigerung zu dem vorherig verwendeten Verfahren auf dem Torwartroboter. Der Vorteil begründet sich vor allem darin, dass hier davon ausgegangen werden kann, dass es sich mit sehr hoher Wahrscheinlichkeit um einen Ball handelt. Desweiteren bin ich der Meinung, dass sich die Robustheit durch die Erkentnisse aus Abschnitt 5.3 und durch eine bessere Anpassung des Ballmodels an den Kinect Sensor noch weiter steigern lässt. Auch wenn in dieser Arbeit nur Bälle bis sechs Meter Entfernung eine Rolle spielen, war es in der Praxis möglich Bälle bis zu acht Meter Entfernung zu erkennen.

Aufgrund der zu Beginn der Arbeit nicht verfügbaren Kalibrierung des Kinect Sensors, wurde eine eigene Kalibrierung entwickelt, die eine ausreichende Genauigkeit für die Ballerkennung erreicht. Diese hier entwickelte Kalibrierung sticht durch ihr ressourcenschonendes Verfahren hervor, erreicht aber nicht die Qualität der voreingestellten Kalibrierung. Interessant zu sehen wäre, ob sich die Prozessorauslastung durch die Verwendung der Kinect eigenen Kalibrierung erhöht. Leider war es nicht mehr möglich diesen Punkt hier zu testen. In Systemen mit begrenzten Kapazitäten des Prozessors wäre dann eine Anwendung dieser hier entwickelten Kalibrierung denkbar, wenn die Genauigkeit der Kalibrierung nicht von essenzieller Bedeutung ist.

Ein noch zu lösendes Problem besteht beim Erkennen von Bällen, die von einem Roboter geführt werden. Durch eine genaue Anpassung der Grenzwerte des Segmentierungsverfahren sollte es möglich sein diese Bälle in einem Bereich bis ca. 4 m vom Sensor entfernt erkennen zu können. In weiterer Entfernung ist es aufgrund des Rauschens des Kinect Sensors nicht möglich diese Bälle zu erkennen. Diese Funktion macht sich in einer höheren Prozessorauslastung bemerkbar. Da die im RoboCup verwendeten

omnidirektionalen Kameras diese Bälle gut erkennen können, wurde kein Wert auf die Implementierung gelegt.

Durch statistische Verfahren sollte es möglich sein, aufbauend auf dieser Arbeit, die Ballerkennung im RoboCup Szenario unabhängig von einer vorab Farbkalibrierung durchzuführen. So könnte ein Roboter z.B beim Anstoß den Ball durch Tiefeninformation erkennen und selbstständig die Farbe des Balls kalibrieren. Die Punkte die falsch erkannt wurden, können durch ein a priori Wissen der Position des Balls relativ zum Roboter ausgeschlossen werden. Dazu müssten die Farbräume der omnidirektionalen Kamera und des Kinect Sensors aufeinander abgestimmt werden.

Im Gegensatz zu dem Ballerkennungsalgorithmus des Team Tech United [7], wird hier viel Wert auf die Verwendung der Tiefeninformation gelegt. Tech United gibt die Ballerkennungsrate ihres Verfahrens mit ca. 80-90% an sowie die benötigte Rechenzeit auf einem Dual-Core 2Ghz Prozessor mit 10-20 ms. Hier war es leider nicht möglich genauere Angaben zu bekommen.

Mit dem Einsatz der Echtzeit-Ballerkennung beim RoboCup 2011 in Istanbul wurde die Funktion der Ballerkennung auch unter Wettkampfbedingungen gezeigt.

A Anhang

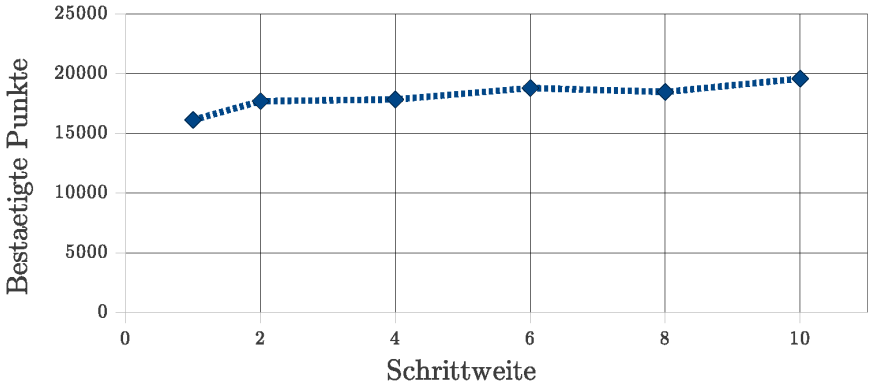

Abbildung 32: Verbleibende Punkte in der Punktwolke des Verfahrens “Radiale Überprüfung der Umgebung um einen Ballmittelpunkt”

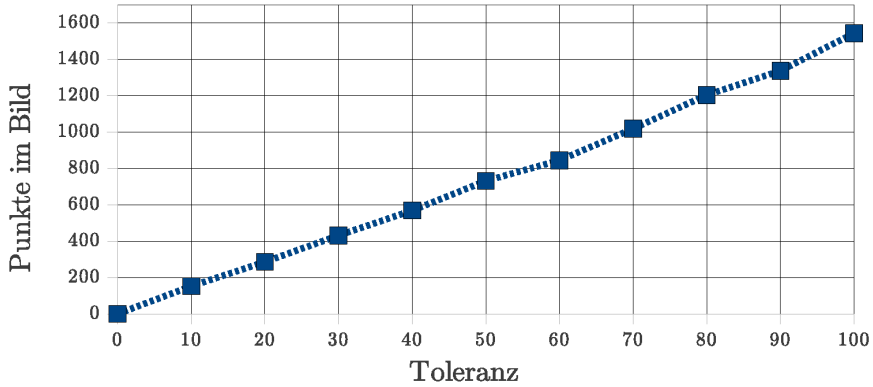

Abbildung 33: Verbleibende Punkte in der Punktwolke des Verfahrens “Verifizierung durch Anzahl der Punkte auf dem Ball”

In Tabelle 6 und 7 sind die Werte aus Abbildung 16 noch einmal dargestellt.

Auflösung	Erkannte Bälle in Prozent	Durchschnittliche Dauer in ms
640 x 480	65,37	15,44
640 x 240	65,10	7,9
320 x 240	50,0	3,99
320 x 120	36,37	2,07

Tabelle 6: Rollende Bälle - Nahbereich

Auflösung	Erkannte Bälle in Prozent	Durchschnittliche Dauer in ms
640 x 480	59,73	15,16
640 x 240	49,70	7,07
320 x 240	39,26	3,86
320 x 120	33,63	1,99

Tabelle 7: Rollende Bälle - Erweiterter Bereich

In Tabelle 8 und 9 sind die Werte aus Abbildung 17 noch einmal dargestellt

Auflösung	Erkannte Bälle in Prozent	Durchschnittliche Dauer in ms
640 x 480	51,42	15,15
640 x 240	42,19	7,8
320 x 240	34,91	3,91
320 x 120	NaN	NaN

Tabelle 8: Fliegende Bälle - Nahbereich

Auflösung	Erkannte Bälle in Prozent	Durchschnittliche Dauer in ms
640 x 480	57,44	13,57
640 x 240	42,92	7,32
320 x 240	45,55	4,07
320 x 120	37,28	1,96

Tabelle 9: Fliegende Bälle - Erweiterter Bereich

In Tabelle 10 sind die q_{max} Werte der Verfahren dargestellt.

Verfahren	q_{max}
LVB	0,11/0,043
VB	0,655
RUUB	0,0071
VAPB	0,032

Tabelle 10: Verkleinerung der Punktkwolke in Prozent

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Hauptseminar Augmented Reality / Kamera Kalibrierung. Technische Universität München Dimitrova, I. , Sielhorst, T. (2005).

[2] Opencv. http://opencv.willowgarage.com/wiki/. Accessed February 09, 2012.

[3] Openkinect - image information. http://openkinect.org/wiki/Imaging_Information. Accessed June, 2011.

[4] Openni. http://75.98.78.94/default.aspx. Accessed February 09, 2012.

[5] Qt - framework, nokia. http://qt.nokia.com/products/. Accessed February 09, 2012.

[6] BIEDERMAN, I. Recognition-by-components: A theory of human image understanding. *Psychological Review 94*, 2 (1987), 115–147.

[7] GONZALES, R. C., AND WOODS, R. E. *Digital Image Processing*, vol. Third Edition. Pearson Education, (2008).

[8] GROENEN, J., AND DOUVEN, Y. 3d ball recognition with kinect. http://www.techunited.nl/wiki/index.php?title=3D_Ball_Recognition_with_Kinect Accessed July 2011.

[9] HARTIGAN, J., AND WONG, M. A k-means clustering algorithm. *Journal of the Royal Statistical Society 28*, 1 (1979).

[10] JÄHNE, B. *Digitale Bildverarbeitung*, 6 ed. Springer, (2005).

[11] KITANO, H., MINORU, A., KUNIYOSHI, Y., NODA, I., OSAWA, E., AND MATSUBARA, H. Robocup. *AI Magazine 18*, 1 (1997), 74–85.

[12] KOFLER, M. Inbetriebnahme und untersuchung des kinect sensors. Master's thesis, Fh Oberösterreich, (2011).

[13] TSAI, R. A versatile camera calibration technique for high-accurancy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics & Automation Magazine 3*, 4 (1987), 323–344.

[14] WAIBEL, M., BEETZ, M., CIVERA, J., D'ANDREA, R., ELFRING, J., GALVEZ-LOPEZ, D., HAUSSERMANN, K., JANSSEN, R., MONTIEL, J., PERZYLO, A., SCHIESSLE, B., TENORTH, M., ZWEIGLE, O., AND VAN DE MOLENGRAFT, R. Roboearth. *IEEE Robotics & Automation Magazine 18*, 2 (2011), 69–82.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und dabei keine andere als die angegebene Literatur verwendet habe.

Stuttgart, den 10. April 2012

Matthias Nösner